Learn Python

A Beginner's Guide to Python, Numpy, Pandas and Scipy

M R KISHORE KUMAR

see more please visit: https://homeofpdf.com

Learn Python

A Beginner’s Guide to Python, NumPy, Pandas and SciPy.

M R KISHORE KUMAR

For you.

One way or another this book ended up in your hands. I’m excited to
see what you do with it, and I hope the knowledge within this book
makes as large an impact on your life as it has on my own.

see more please visit: https://homeofpdf.com

About the Tutorial

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Audience

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

Prerequisites

Before proceeding with this tutorial you should have a basic working knowledge of Windows or Linux operating system, additionally, you must be familiar with:

- Experience with any text editor like sublime, Visual Studio Code, or pycharm, etc.
- How to create directories and files on your computer.
- How to navigate through different directories.
- How to type content in a file and save them on a computer.

Copyright © 2021 by M R KISHORE KUMAR

All rights reserved. This book or any portion thereof May not be reproduced or used in any manner whatsoever Without the express written permission of the publisher Except for the use of brief quotations in a book review.

Printed in India.

First Printing, 2021

see more please visit: https://homeofpdf.com

Table of Content

1.Python Introduction…………………………………………...6

2.Python Getting Started……………………………………...9

3.Python Syntax…………………………………………...11

4.Python Comments…………………………………………...12

5.Python Variables…………………………………………...14

6.Python Operators…………………………………………...19

7.Python Data Types…………………………………………...23

8.Python Numbers…………………………………………...26

9.Python Strings…………………………………………...29

10.Python Casting…………………………………………...42

11.Python Booleans…………………………………………...43

12.Python Lists…………………………………………...47

13.Python Tuples…………………………………………...68

14.Python range() Function……………………………………80

15.Python Sets…………………………………………...82

16.Python frozenset() Function……………………………………92

17.Python Dictionaries……………………………………………93

18.Python math Module……………………………………………106

19.Python User Input…………………………………………...110

see more please visit: https://homeofpdf.com

20.Python eval() Function……………………………………………111
21.Python If ...
Else…………………………………………………...112
22.Python While
Loops……………………………………………….117
23.Python For
Loops……………………………………………...119
24.Python Arrays……………………………………………...124
25.Python Functions………………………………………….127
26.Python Lambda…………………………………………...134
27.Python Classes and Objects………………………………..136
28.Python
Inheritance……………………………………………140
29.Python Iterators…………………………………………...144
30.Python Scope……………………………………………...148
31.Python Modules…………………………………………...154
32.Python
Datetime………………………………………………...158
33.Python Math……………………………………………...162
34.Python JSON……………………………………………...164
35.Python RegEx……………………………………………...169
36.Python PIP………………………………………………...177
37.Python Try
Except…………………………………………...180
38.Python String Formatting……………………………………183
39.Python File Handling……………………………………………

see more please visit: https://homeofpdf.com

185
40.NumPy Tutorial…………………………………………………...191
41.Pandas Tutorial…………………………………………………...283
42.SciPy Tutorial…………………………………………………...312

01.Python Introduction

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

see more please visit: https://homeofpdf.com

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has a syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way, or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial, Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans, or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed for readability and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions, and classes. Other programming languages often use curly brackets for this purpose.

Example

```
print("Hello, World!")
```

Definition and Usage

The print() function prints the specified message to the screen or another

see more please visit: https://homeofpdf.com

standard output device.

The message can be a string or any other object, the object will be converted into a string before written to the screen.

Syntax

print(*object(s)*, sep=*separator*, end=*end*, file=*file*, flush=*flush*)

Parameter Values

Parameter	Description
object(s)	Any object, and as many as you like. Will be converted to a string before printed
sep='*separator*'	Optional. Specify how to separate the objects, if there is more than one. Default is ' '
end='*end*'	Optional. Specify what to print at the end. Default is '\n' (line feed)
file	Optional. An object with a write method. Default is sys.stdout
flush	Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

see more please visit: https://homeofpdf.com

02.Python Getting Started

Python Install

Many PCs and Macs will have python already installed.

To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

C:\Users*Your Name*>python --version

To check if you have python installed on a Linux or Mac, then on Linux open the command line or on Mac open the Terminal, and type:

python --version

If you find that you do not have python installed on your computer, then you can download it for free from the following website: https://www.python.org/

Python Quickstart

Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

The way to run a python file is like this on the command line:

C:\Users*Your Name*>python helloworld.py

Where "helloworld.py" is the name of your python file.

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

```
print("Hello, World!")
```

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:

C:\Users*Your Name*>python helloworld.py

The output should read: **Hello, World!**

Congratulations, you have written and executed your first Python

see more please visit: https://homeofpdf.com

program!

The Python Command Line

To test a short amount of code in python sometimes it is the quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command-line itself.

Type the following on the Windows, Mac, or Linux command line:

C:\Users*Your Name*>python

Or, if the "python" command did not work, you can try "py":

C:\Users*Your Name*>py

From there you can write any python, including our hello world example from earlier in the tutorial:

```
C:\Users\Your Name>python

Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> print("Hello, World!")
```

Which will write "Hello, World!" in the command line:

```
C:\Users\Your Name>python

Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> print("Hello, World!")

Hello, World!
```

Whenever you are done in the python command line, you can simply type the following to quit the python command-line interface: `exit()`

see more please visit: https://homeofpdf.com

03.Python Syntax

Execute Python Syntax

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")
Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
  print("Five is greater than two!")
```

Python will give you an error if you skip the indentation!

04.Python Comments

see more please visit: https://homeofpdf.com

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

Creating a Comment

Comments start with a #, and Python will ignore them:

```
#This is a comment

print("Hello, World!")
```

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

```
print("Hello, World!") #This is a comment
```

A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

```
#print("Hello, World!")

print("Cheers, Mate!")
```

Multi-Line Comments

Python does not really have a syntax for multi-line comments.

To add a multiline comment you could insert a # for each line.

```
#This is a comment

#written in

#more than just one line

print("Hello, World!")
```

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

```
"""

This is a comment
```

see more please visit: https://homeofpdf.com

```
written in
more than just one line
"""
print("Hello, World!")
```

05.Python Variables

Variables

Variables are containers for storing data values.

Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

```
x = 5
```

see more please visit: https://homeofpdf.com

```
y = "John"
print(x)
print(y)
```

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Casting

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3) # x will be '3'
y = int(3) # y will be 3
z = float(3)  # z will be 3.0
```

Get the Type

You can get the data type of a variable with the type() function.

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Single or Double Quotes?

String variables can be declared either by using single or double quotes:

```
x = "John"
```

see more please visit: https://homeofpdf.com

 # is the same as

x = 'John'

Case-Sensitive

Variable names are case-sensitive.

a = 4

A = "Sally"

#A will not overwrite a

Python - Variable Names

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age, and AGE are three different variables)

Legal variable names:

myvar = "John"

my_var = "John"

_my_var = "John"

myVar = "John"

MYVAR = "John"

myvar2 = "John"

Remember that variable names are case-sensitive

see more please visit: https://homeofpdf.com

Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

Camel Case

Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "John"
```

Snake Case

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

Python Variables - Assign Multiple Values

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Note: Make sure the number of variables matches the number of values, or

see more please visit: https://homeofpdf.com

else you will get an error.

One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Unpack a Collection

If you have a collection of values in a list, tuple, etc. Python allows you extract the values into variables. This is called *unpacking*.

Learn more about unpacking in Tuples Chapter.

Python - Output Variables

Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

Example

```
x = "awesome"
print("Python is " + x)
```

You can also use the + character to add a variable to another variable**:**

Example

```
x = "Python is "
y = "awesome"
```

see more please visit: https://homeofpdf.com

```
z =  x + y
print(z)
```

For numbers, the + character works as a mathematical operator:

Example

```
x = 5
y = 10
print(x + y)
```

06.Python Operators

Python Operators

Operators are used to performing operations on variables and values.

In the example below, we use the + operator to add together two values:

Example

```
print(10 + 5)
```

Python divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

see more please visit: https://homeofpdf.com

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	**Name**	**Example**
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

Assignment operators are used to assigning values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

see more please visit: https://homeofpdf.com

<table>
<tr><td>&=</td><td>x &= 3</td><td>x = x & 3</td></tr>
<tr><td>|=</td><td>x |= 3</td><td>x = x | 3</td></tr>
<tr><td>^=</td><td>x ^= 3</td><td>x = x ^ 3</td></tr>
<tr><td>>>=</td><td>x >>= 3</td><td>x = x >> 3</td></tr>
<tr><td><<=</td><td>x <<= 3</td><td>x = x << 3</td></tr>
</table>

Python Identity Operators

Identity operators are used to comparing the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
Is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to testing if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Comparison Operators

Comparison operators are used to comparing two values:

see more please visit: https://homeofpdf.com

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	or equal to x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combining conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Bitwise Operators

Bitwise operators are used to comparing (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

see more please visit: https://homeofpdf.com

07.Python Operators

Built-in Data Types

In programming, the data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

Getting the Data Type

You can get the data type of any object by using the type() function:

Example

Print the data type of the variable x:

```
x = 5

print(type(x))
```

see more please visit: https://homeofpdf.com

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	**Data Type**
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

see more please visit: https://homeofpdf.com

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
x = str("Hello World")	str
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list(("apple", "banana", "cherry"))	list
x = tuple(("apple", "banana", "cherry"))	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set
x = frozenset(("apple", "banana", "cherry"))	frozenset
x = bool(5)	bool
x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

see more please visit: https://homeofpdf.com

08.Python Numbers

Python Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them:

Example

```
x = 1         # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the type() function:

Example

```
print(type(x))
print(type(y))
print(type(z))
```

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

see more please visit: https://homeofpdf.com

Float

Float, or "floating-point number" is a number, positive or negative, containing one or more decimals.

Example

Floats:

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example

Floats:

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

Complex

Complex numbers are written with a "j" as the imaginary part:

Example

Complex:

```
x = 3+5j
y = 5j
z = -5j
```

see more please visit: https://homeofpdf.com

```
print(type(x))
print(type(y))
print(type(z))
```

Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods:

Example

Convert from one type to another:

```
x = 1        # int

y = 2.8  # float

z = 1j   # complex

#convert from int to float:

a = float(x)

#convert from float to int:

b = int(y)

#convert from int to complex:

c = complex(x)

print(a)

print(b)

print(c)

print(type(a))

print(type(b))

print(type(c))
```

Note: You cannot convert complex numbers into another number type.

see more please visit: https://homeofpdf.com

09.Python Strings

Strings

Strings in python are surrounded by either single quotation marks or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example

```
print("Hello")
print('Hello')
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"
print(a)
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Or three single quotes:

see more please visit: https://homeofpdf.com

Example

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

Note: in the result, the line breaks are inserted at the same position as in the code.

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a for a loop.

Example

Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

Learn more about For Loops in our Python For Loops chapter.

see more please visit: https://homeofpdf.com

String Length

To get the length of a string, use the len() function.

Example

The len() function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

Example

Check if "free" is present in the following text:

```
txt = "The best things in life are free!"
print("free" in txt)
```

Use it in an if statement:

Example

Print only if "free" is present:

```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")
```

Learn more about If statements in our Python If...Else chapter.

Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

Example

Check if "expensive" is NOT present in the following text:

see more please visit: https://homeofpdf.com

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

Use it in an if statement:

Example

print only if "expensive" is NOT present:

```
txt = "The best things in life are free!"
if "expensive" not in txt:
  print("Yes, 'expensive' is NOT present.")
```

Python - Slicing Strings

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

Note: The first character has an index of 0.

Slice From the Start

By leaving out the start index, the range will start at the first character:

Example

Get the characters from the start to position 5 (not included):

```
b = "Hello, World!"
print(b[:5])
```

see more please visit: https://homeofpdf.com

Slice To the End

By leaving out the *end* index, the range will go to the end:

Example

Get the characters from position 2, and all the way to the end:

```
b = "Hello, World!"
print(b[2:])
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example

Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

```
b = "Hello, World!"
print(b[-5:-2])
```

Python - Modify Strings

Python has a set of built-in methods that you can use on strings.

Upper Case

Example

The upper() method returns the string in the upper case:

```
a = "Hello, World!"
print(a.upper())
```

Lower Case

see more please visit: https://homeofpdf.com

Example

The lower() method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

The strip() method removes any whitespace from the beginning else the end:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

Replace String

Example

The replace() method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Split String

The split() method returns a list where the text between the specified separator becomes the list items.

Example

The split() method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

Learn more about Lists in our Python Lists chapter.

see more please visit: https://homeofpdf.com

Python - String Concatenation

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable a with variable b into variable c:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

Example

To add a space between them, add a " ":

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

Python - Format - Strings

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example

```
age = 36
txt = "My name is John, I am " + age
```

see more please visit: https://homeofpdf.com

```
print(txt)
```

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Example

Use the format() method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Python - Escape Characters

see more please visit: https://homeofpdf.com

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

txt = "We are the so-called "Vikings" from the north."

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

txt = "We are the so-called \"Vikings\" from the north."

Escape Characters

Other escape characters used in Python:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return

see more please visit: https://homeofpdf.com

\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

Python - String Methods

String Methods

Python has a set of built-in methods that you can use on strings.

Note: All string methods return new values. They do not change the original string.

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string

see more please visit: https://homeofpdf.com

format_map() Formats specified values in a string

index() Searches the string for a specified value and returns the position of where it was found

isalnum() Returns True if all characters in the string are alphanumeric

isalpha() Returns True if all characters in the string are in the alphabet

isdecimal() Returns True if all characters in the string are decimals

isdigit() Returns True if all characters in the string are digits

isidentifier() Returns True if the string is an identifier

islower() Returns True if all characters in the string are lower case

isnumeric() Returns True if all characters in the string are numeric

isprintable() Returns True if all characters in the string are printable

isspace() Returns True if all characters in the string are whitespaces

istitle() Returns True if the string follows the rules of a title

isupper() Returns True if all characters in the string are upper case

join() Joins the elements of an iterable to the end of the string

ljust() Returns a left justified version of the string

lower() Converts a string into lower case

lstrip() Returns a left trim version of the string

maketrans() Returns a translation table to be used in translations

partition() Returns a tuple where the string is parted into three parts

replace() Returns a string where a specified value is replaced with a specified value

see more please visit: https://homeofpdf.com

rfind()	Searches the string for a specified value and returns the last position of where it was found
zfill()	Fills the string with a specified number of 0 values at the beginning
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case

see more please visit: https://homeofpdf.com

10.Python Casting

Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example

Integers:

```
x = int(1)   # x will be 1

y = int(2.8) # y will be 2

z = int("3") # z will be 3
```

Floats:

```
x = float(1)     # x will be 1.0

y = float(2.8)   # y will be 2.8

z = float("3")   # z will be 3.0

w = float("4.2") # w will be 4.2
```

Strings:

```
x = str("s1") # x will be 's1'      y = str(2)     # y will be '2'

z = str(3.0)  # z will be '3.0'
```

see more please visit: https://homeofpdf.com

11.Python Booleans

Booleans represent one of two values: True or False.

Boolean Values

In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

Example

```
print(10 > 9)

print(10 == 9)

print(10 < 9)
```

When you run a condition in an if statement, Python returns True or False:

Example

Print a message based on whether the condition is True or False:

```
a = 200

b = 33

if b > a:

  print("b is greater than a")

else:

  print("b is not greater than a")
```

Evaluate Values and Variables

see more please visit: https://homeofpdf.com

The bool() function allows you to evaluate any value, and give you True or False in return,

Example

Evaluate a string and a number:

```
print(bool("Hello"))

print(bool(15))
```

Example

Evaluate two variables:

```
x = "Hello"

y = 15

print(bool(x))

print(bool(y))
```

Most Values are True

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

Example

The following will return True:

```
bool("abc")

bool(123)

bool(["apple", "cherry", "banana"])
```

Some Values are False

see more please visit: https://homeofpdf.com

In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

Example

The following will return False:

```
bool(False)

bool(None)

bool(0)

bool("")

bool(())

bool([])

bool({})
```

One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a __len__ function that returns 0 or False:

Example

```
class myclass():
  def __len__(self):
    return 0

myobj = myclass()

print(bool(myobj))
```

Functions can Return a Boolean

You can create functions that returns a Boolean Value:

Example

see more please visit: https://homeofpdf.com

Print the answer of a function:

```
def myFunction() :
  return True
print(myFunction())
```

You can execute code based on the Boolean answer of a function:

Example

Print "YES!" if the function returns True, otherwise print "NO!":

```
def myFunction() :
  return True
if myFunction():
  print("YES!")
else:
  print("NO!")
```

Python also has many built-in functions that return a boolean value, like the isinstance() function, which can be used to determine if an object is of a certain data type:

Example

Check if an object is an integer or not:

```
x = 200
print(isinstance(x, int))
```

12.Python Lists

```
mylist = ["apple", "banana", "cherry"]
```

List

see more please visit: https://homeofpdf.com

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]

print(thislist)
```

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

Note: There are some list methods that will change the order, but in general: the order of the items will not change.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

see more please visit: https://homeofpdf.com

Example

Lists allow duplicate values:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

List Length

To determine how many items a list has, use the len() function:

Example

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

List Items - Data Types

List items can be of any data type:

Example

String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

A list can contain different data types:

Example

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

type()

From Python's perspective, lists are defined as objects with the data type 'list':

see more please visit: https://homeofpdf.com

```
<class 'list'>
```

Example

What is the data type of a list?

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

The list() Constructor

It is also possible to use the list() constructor when creating a new list.

Example

Using the list() constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered* and changeable. No duplicate members.

*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and it could mean an increase in efficiency or security.

see more please visit: https://homeofpdf.com

Python - Access List Items

Access Items

List items are indexed and you can access them by referring to the index number:

Example

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Note: The first item has index 0.

Negative Indexing

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

Example

Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon",
"mango"]
```

see more please visit: https://homeofpdf.com

```
print(thislist[2:5])
```

Note: The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to, but NOT including, "kiwi":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" to the end:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

Check if Item Exists

see more please visit: https://homeofpdf.com

To determine if a specified item is present in a list use the in keyword:

Example

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Python - Change List Items

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Change a Range of Item Values

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Example

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

see more please visit: https://homeofpdf.com

If you insert *more* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second value by replacing it with *two* new values:

```
thislist = ["apple", "banana", "cherry"]

thislist[1:2] = ["blackcurrant", "watermelon"]

print(thislist)
```

Note: The length of the list will change when the number of items inserted does not match the number of items replaced.

If you insert *less* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second and third value by replacing it with *one* value:

```
thislist = ["apple", "banana", "cherry"]

thislist[1:3] = ["watermelon"]

print(thislist)
```

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the insert() method.

The insert() method inserts an item at the specified index:

Example

Insert "watermelon" as the third item:

```
thislist = ["apple", "banana", "cherry"]

thislist.insert(2, "watermelon")

print(thislist)
```

Note: As a result of the example above, the list will now contain 4 items.

see more please visit: https://homeofpdf.com

Python - Add List Items

Append Items

To add an item to the end of the list, use the append() method:

Example

Using the append() method to append an item:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Insert Items

To insert a list item at a specified index, use the insert() method.

The insert() method inserts an item at the specified index:

Example

Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Note: As a result of the examples above, the lists will now contain 4 items.

Extend List

To append elements from *another list* to the current list, use the extend() method.

Example

Add the elements of tropical to thislist:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
```

see more please visit: https://homeofpdf.com

```
thislist.extend(tropical)
print(thislist)
```

The elements will be added to the *end* of the list.

Add Any Iterable

The extend() method does not have to append *lists*, you can add any iterable object (tuples, sets, dictionaries etc.).

Example

Add elements of a tuple to a list:

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Python - Remove List Items

Remove Specific Item

The remove() method removes the specified item.

Example

Remove "banana":

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Remove Specific Index

The pop() method removes the specified index.

Example

see more please visit: https://homeofpdf.com

Remove the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

If you do not specify the index, the pop() method removes the last item.

Example

Remove the last item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

The del keyword also removes the specified index:

Example

Remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

The del keyword can also delete the list completely.

Example

Delete the entire list:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

Example

see more please visit: https://homeofpdf.com

Clear the list content:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Python - Loop Lists

Loop Through a List

You can loop through the list items by using a for loop:

Example

Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Learn more about for loops in our Python For Loops Chapter.

Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

Example

Print all items by referring to their index number:

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

The iterable created in the example above is [0, 1, 2].

Using a While Loop

see more please visit: https://homeofpdf.com

You can loop through the list items by using a while loop.

Use the len() function to determine the length of the list, then start at 0 and loop your way through the list items by referring to their indexes.

Remember to increase the index by 1 after each iteration.

Example

Print all items, using a while loop to go through all the index numbers

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

Learn more about while loops in our Python While Loops Chapter.

Looping Using List Comprehension

List Comprehension offers the shortest syntax for looping through lists:

Example

A shorthand for loop that will print all items in a list:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

Learn more about list comprehension in the next chapter: List Comprehension.

Python - List Comprehension

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list

see more please visit: https://homeofpdf.com

based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside:

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
  if "a" in x:
    newlist.append(x)
print(newlist)
```

With list comprehension you can do all that with only one line of code:

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist)
```

The Syntax

newlist = [*expression* for *item* in *iterable* if *condition* == True]

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that only accepts the items that valuate to True.

see more please visit: https://homeofpdf.com

Example

Only accept items that are not "apple":

newlist = [x for x in fruits if x != "apple"]

The condition **if x != "apple"** will return True for all elements other than "apple", making the new list contain all fruits except "apple".

The *condition* is optional and can be omitted:

Example

With no if statement:

newlist = [x for x in fruits]

Iterable

The *iterable* can be any iterable object, like a list, tuple, set etc.

Example

You can use the range() function to create an iterable:

newlist = [x for x in range(10)]

Same example, but with a condition:

Example

Accept only numbers lower than 5:

newlist = [x for x in range(10) if x < 5]

Expression

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

Example

Set the values in the new list to upper case:

newlist = [x.upper() for x in fruits]

You can set the outcome to whatever you like:

see more please visit: https://homeofpdf.com

Example

Set all values in the new list to 'hello':

```
newlist = ['hello' for x in fruits]
```

The *expression* can also contain conditions, not like a filter, but as a way to manipulate the outcome:

Example

Return "orange" instead of "banana":

```
newlist = [x if x != "banana" else "orange" for x in fruits]
```

The *expression* in the example above says:

"Return the item if it is not banana, if it is banana return orange".

Python - Sort Lists

Sort List Alphanumerically

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

Example

Sort the list alphabetically:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

Example

Sort the list numerically:

```
thislist = [100, 50, 65, 82, 23]
```

see more please visit: https://homeofpdf.com

```
thislist.sort()
print(thislist)
```

Sort Descending

To sort descending, use the keyword argument reverse = True:

Example

Sort the list descending:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Example

Sort the list descending:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
```

Customize Sort Function

You can also customize your own function by using the keyword argument key = *function*.

The function will return a number that will be used to sort the list (the lowest number first):

Example

Sort the list based on how close the number is to 50:

```
def myfunc(n):
  return abs(n - 50)
```

see more please visit: https://homeofpdf.com

```
thislist = [100, 50, 65, 82, 23]
thislist.sort(key = myfunc)
print(thislist)
```

Case Insensitive Sort

By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters:

Example

Case sensitive sorting can give an unexpected result:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
```

Luckily we can use built-in functions as key functions when sorting a list.

So if you want a case-insensitive sort function, use str.lower as a key function:

Example

Perform a case-insensitive sort of the list:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)
print(thislist)
```

Reverse Order

What if you want to reverse the order of a list, regardless of the alphabet?

The reverse() method reverses the current sorting order of the elements.

Example

Reverse the order of the list items:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Python - Copy Lists

Copy a List

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a *reference* to list1, and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in List method copy().

Example

Make a copy of a list with the copy() method:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Another way to make a copy is to use the built-in method list().

Example

Make a copy of a list with the list() method:

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Python - Join Lists

see more please visit: https://homeofpdf.com

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways is by using the + operator.

Example

Join two list:

```
list1 = ["a", "b", "c"]

list2 = [1, 2, 3]

list3 = list1 + list2

print(list3)
```

Another way to join two lists is by appending all the items from list2 into list1, one by one:

Example

Append list2 into list1:

```
list1 = ["a", "b" , "c"]

list2 = [1, 2, 3]

for x in list2:

  list1.append(x)

print(list1)
```

Or you can use the extend() method, which purpose is to add elements from one list to another list:

Example

Use the extend() method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]

list2 = [1, 2, 3]

list1.extend(list2)
```

see more please visit: https://homeofpdf.com

```
print(list1)
```

Python - List Methods

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

see more please visit: https://homeofpdf.com

13.Python Tuples

```
mytuple = ("apple", "banana", "cherry")
```

Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

Example

Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates

Since tuples are indexed, they can have items with the same value:

see more please visit: https://homeofpdf.com

Example

Tuples allow duplicate values:

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

Tuple Length

To determine how many items a tuple has, use the len() function:

Example

Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

Example

One item tuple, remember the comma:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Tuple Items - Data Types

Tuple items can be of any data type:

Example

String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
```

see more please visit: https://homeofpdf.com

```
tuple3 = (True, False, False)
```

A tuple can contain different data types:

Example

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")
```

type()

From Python's perspective, tuples are defined as objects with the data type 'tuple':

```
<class 'tuple'>
```

Example

What is the data type of a tuple?

```
mytuple = ("apple", "banana", "cherry")
print(type(mytuple))
```

The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

Example

Using the tuple() method to make a tuple:

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double
round-brackets
print(thistuple)
```

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.

see more please visit: https://homeofpdf.com

- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered* and changeable. No duplicate members.

*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and it could mean an increase in efficiency or security.

Python - Access Tuple Items

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Note: The first item has index 0.

Negative Indexing

Negative indexing means starting from the end.

-1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

Range of Indexes

see more please visit: https://homeofpdf.com

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

Example

Return the third, fourth, and fifth item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

Note: The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to, but NOT included, "kiwi":

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" and to the end:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

see more please visit: https://homeofpdf.com

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon",
"mango")
print(thistuple[-4:-1])
```

Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

Example

Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Python - Update Tuples

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.

But there are some workarounds.

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
```

see more please visit: https://homeofpdf.com

```
y[1] = "kiwi"
x = tuple(y)

print(x)
```

Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

1. **Convert into a list**: Just like the workaround for *changing* a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

Example

Convert the tuple into a list, add "orange", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

2. **Add tuple to a tuple**. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

Example

Create a new tuple with the value "orange", and add that tuple:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

Note: When creating a tuple with only one item, remember to include a comma after the item, otherwise it will not be identified as a tuple.

Remove Items

Note: You cannot remove items in a tuple.

see more please visit: https://homeofpdf.com

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

Example

Convert the tuple into a list, remove "apple", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

Or you can delete the tuple completely:

Example

The del keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

Python - Unpack Tuples

Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

Example

Packing a tuple:

```
fruits = ("apple", "banana", "cherry")
```

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

Example

Unpacking a tuple:

see more please visit: https://homeofpdf.com

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

Note: The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

Using Asterisk*

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

Example

Assign the rest of the values as a list called "red":

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

Example

Add a list of values the "tropic" variable:

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)
```

see more please visit: https://homeofpdf.com

Python - Loop Tuples

Loop Through a Tuple

You can loop through the tuple items by using a for loop.

Example

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Learn more about for loops in our Python For Loops Chapter.

Loop Through the Index Numbers

You can also loop through the tuple items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

Example

Print all items by referring to their index number:

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

Using a While Loop

You can loop through the list items by using a while loop.

Use the len() function to determine the length of the tuple, then start at 0 and loop your way through the tuple items by refering to their indexes.

Remember to increase the index by 1 after each iteration.

Example

Print all items, using a while loop to go through all the index numbers:

see more please visit: https://homeofpdf.com

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Learn more about while loops in our Python While Loops Chapter.

Python - Join Tuples

Join Two Tuples

To join two or more tuples you can use the + operator:

Example

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Multiply Tuples

If you want to multiply the content of a tuple a given number of times, you can use the * operator:

Example

Multiply the fruits tuple by 2:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

Python - Tuple Methods

see more please visit: https://homeofpdf.com

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

14.Python range() Function

Example

Create a sequence of numbers from 0 to 5, and print each item in the sequence:

```
x = range(6)
for n in x:
  print(n)
```

see more please visit: https://homeofpdf.com

Definition and Usage

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Syntax

range(*start*, *stop*, *step*)

Parameter Values

Parameter	Description
start	Optional. An integer number specifying at which position to start. Default is 0
stop	Required. An integer number specifying at which position to stop (not included).
step	Optional. An integer number specifying the incrementation. Default is 1

More Examples

Create a sequence of numbers from 3 to 5, and print each item in the sequence:

```
x = range(3, 6)
for n in x:
  print(n)
```

Create a sequence of numbers from 3 to 19, but increment by 2 instead of 1:

```
x = range(3, 20, 2)
for n in x:
  print(n)
```

see more please visit: https://homeofpdf.com

15.Python Sets

```
myset = {"apple", "banana", "cherry"}
```

Set

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is both *unordered* and *unindexed*.

Sets are written with curly brackets.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Note: Sets are unordered, so you cannot be sure in which order the items will appear.

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Sets are unchangeable, meaning that we cannot change the items after the set

see more please visit: https://homeofpdf.com

has been created.

Once a set is created, you cannot change its items, but you can add new items.

Duplicates Not Allowed

Sets cannot have two items with the same value.

Example

Duplicate values will be ignored:

```
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
```

Get the Length of a Set

To determine how many items a set has, use the len() method.

Example

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}
print(len(thisset))
```

Set Items - Data Types

Set items can be of any data type:

Example

String, int and boolean data types:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

A set can contain different data types:

see more please visit: https://homeofpdf.com

Example

A set with strings, integers and boolean values:

```
set1 = {"abc", 34, True, 40, "male"}
```

type()

From Python's perspective, sets are defined as objects with the data type 'set':

<class 'set'>

Example

What is the data type of a set?

```
myset = {"apple", "banana", "cherry"}
print(type(myset))
```

The set() Constructor

It is also possible to use the set() constructor to make a set.

Example

Using the set() constructor to make a set:

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate

see more please visit: https://homeofpdf.com

members.
- **Dictionary** is a collection which is ordered* and changeable. No duplicate members.

*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Python - Access Set Items

Access Items

You cannot access items in a set by referring to an index or a key.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Example

Check if "banana" is present in the set:

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Change Items

Once a set is created, you cannot change its items, but you can add new items.

see more please visit: https://homeofpdf.com

Python - Add Set Items

Add Items

Once a set is created, you cannot change its items, but you can add new items.

To add one item to a set use the add() method.

Example

Add an item to a set, using the add() method:

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

Add Sets

To add items from another set into the current set, use the update() method.

Example

Add elements from tropical into thisset:

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

Add Any Iterable

The object in the update() method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

Example

see more please visit: https://homeofpdf.com

Add elements of a list to at set:

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]
thisset.update(mylist)
print(thisset)
```

Python - Remove Set Items

Remove Item

To remove an item in a set, use the remove(), or the discard() method.

Example

Remove "banana" by using the remove() method:

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

Note: If the item to remove does not exist, remove() will raise an error.

Example

Remove "banana" by using the discard() method:

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

Note: If the item to remove does not exist, discard() will **NOT** raise an error.

You can also use the pop() method to remove an item, but this method will remove the *last* item. Remember that sets are unordered, so you will not know what item that gets removed.

The return value of the pop() method is the removed item.

see more please visit: https://homeofpdf.com

Example

Remove the last item by using the pop() method:

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)
print(thisset)
```

Note: Sets are *unordered*, so when using the pop() method, you do not know which item that gets removed.

Example

The clear() method empties the set:

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

Example

The del keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

Python - Loop Sets

Loop Items

You can loop through the set items by using a for loop:

Example

Loop through the set, and print the values:

see more please visit: https://homeofpdf.com

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

Python - Join Sets

Join Two Sets

There are several ways to join two or more sets in Python.

You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

Example

The union() method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

Example

The update() method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

Note: Both union() and update() will exclude any duplicate items.

Keep ONLY the Duplicates

see more please visit: https://homeofpdf.com

The intersection_update() method will keep only the items that are present in both sets.

Example

Keep the items that exist in both set x, and set y:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.intersection_update(y)
print(x)
```

The intersection() method will return a *new* set that only contains the items that are present in both sets.

Example

Return a set that contains the items that exist in both set x, and set y:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.intersection(y)
print(z)
```

Keep All, But NOT the Duplicates

The symmetric_difference_update() method will keep only the elements that are NOT present in both sets.

Example

Keep the items that are not present in both sets:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.symmetric_difference_update(y)
print(x)
```

see more please visit: https://homeofpdf.com

The symmetric_difference() method will return a new set, that contains only the elements that are NOT present in both sets.

Example

Return a set that contains all items from both sets, except items that are present in both:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.symmetric_difference(y)
print(z)
```

Python - Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)

see more please visit: https://homeofpdf.com

isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

16.Python frozenset() Function

Example

Freeze the list, and make it unchangeable:

```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
```

Definition and Usage

The frozenset() function returns an unchangeable frozenset object (which is like a set object, only unchangeable).

see more please visit: https://homeofpdf.com

Syntax

frozenset(*iterable*)

Parameter Values

Parameter	Description
iterable	An iterable object, like list, set, tuple etc.

More Examples

Example

Try to change the value of a frozenset item.

This will cause an error:

```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
x[1] = "strawberry"
```

17.Python Dictionaries

```
thisdict = { "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Dictionary

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered*, changeable and does not allow duplicates.

see more please visit: https://homeofpdf.com

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

Dictionaries are written with curly brackets, and have keys and values:

Example

Create and print a dictionary:

```
thisdict ={
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Dictionary Items

Dictionary items are ordered, changeable, and do not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

Example

Print the "brand" value of the dictionary:

```
thisdict ={ "brand": "Ford","model": "Mustang", "year": 1964}
print(thisdict["brand"])
```

Ordered or Unordered?

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

see more please visit: https://homeofpdf.com

Unordered means that the item does not have a defined order, you cannot refer to an item by using an index.

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Duplicates Not Allowed

Dictionaries cannot have two items with the same key:

Example

Duplicate values will overwrite existing values:

```
thisdict ={"brand": "Ford", "model": "Mustang","year": 1964,"year": 2020}
print(thisdict)
```

Dictionary Length

To determine how many items a dictionary has, use the len() function:

Example

Print the number of items in the dictionary:

```
print(len(thisdict))
```

Dictionary Items - Data Types

The values in dictionary items can be of any data type:

Example

String, int, boolean, and list data types:

```
thisdict ={ "brand": "Ford","electric": False,"year": 1964,"colors": ["red", "white", "blue"]}
```

see more please visit: https://homeofpdf.com

type()

From Python's perspective, dictionaries are defined as objects with the data type 'dict':

<class 'dict'>

Example

Print the data type of a dictionary:

```
thisdict ={"brand": "Ford", "model": "Mustang","year": 1964}
print(type(thisdict))
```

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered and changeable. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Python - Access Dictionary Items

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

see more please visit: https://homeofpdf.com

Example

Get the value of the "model" key:

```
thisdict ={"brand": "Ford","model": "Mustang", "year": 1964}
x = thisdict["model"]
```

There is also a method called get() that will give you the same result:

Example

Get the value of the "model" key:

```
x = thisdict.get("model")
```

Get Keys

The keys() method will return a list of all the keys in the dictionary.

Example

Get a list of the keys:

```
x = thisdict.keys()
```

The list of the keys is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.

Example

Add a new item to the original dictionary, and see that the keys list gets updated as well:

```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.keys()
print(x) #before the change
car["color"] = "white"
print(x) #after the change
```

Get Values

see more please visit: https://homeofpdf.com

The values() method will return a list of all the values in the dictionary.

Example

Get a list of the values:

```
x = thisdict.values()
```

The list of the values is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.

Example

Make a change in the original dictionary, and see that the values list gets updated as well:

```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.values()
print(x) #before the change
car["year"] = 2020
print(x) #after the change
```

Example

Add a new item to the original dictionary, and see that the values list gets updated as well:

```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.values()
print(x) #before the change
car["color"] = "red"
print(x) #after the change
```

Get Items

The items() method will return each item in a dictionary, as tuples in a list.

Example

see more please visit: https://homeofpdf.com

Get a list of the key:value pairs

x = thisdict.items()

The returned list is a *view* of the items of the dictionary, meaning that any changes done to the dictionary will be reflected in the items list.

Example

Make a change in the original dictionary, and see that the items list gets updated as well:

```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.items()
print(x) #before the change
car["year"] = 2020
print(x) #after the change
```

Example

Add a new item to the original dictionary, and see that the items list gets updated as well:

```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.items()
print(x) #before the change
car["color"] = "red"
print(x) #after the change
```

Check if Key Exists

To determine if a specified key is present in a dictionary use the in keyword:

Example

Check if "model" is present in the dictionary:

```
thisdict ={"brand": "Ford","model": "Mustang","year": 1964}
```

see more please visit: https://homeofpdf.com

```
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Python - Change Dictionary Items

Change Values

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

```
thisdict ={"brand": "Ford","model": "Mustang","year": 1964}
thisdict["year"] = 2018
```

Update Dictionary

The update() method will update the dictionary with the items from the given argument.

The argument must be a dictionary, or an iterable object with key:value pairs.

Example

Update the "year" of the car by using the update() method:

```
thisdict ={"brand": "Ford","model": "Mustang", "year": 1964}
thisdict.update({"year": 2020})
```

Python - Add Dictionary Items

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

see more please visit: https://homeofpdf.com

Example

```
thisdict ={"brand": "Ford", "model": "Mustang","year": 1964}
thisdict["color"] = "red"
print(thisdict)
```

Update Dictionary

The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

The argument must be a dictionary, or an iterable object with key:value pairs.

Example

Add a color item to the dictionary by using the update() method:

```
thisdict ={ "brand": "Ford", "model": "Mustang","year": 1964}
thisdict.update({"color": "red"})
```

Python - Remove Dictionary Items

There are several methods to remove items from a dictionary:

Example

The pop() method removes the item with the specified key name:

```
thisdict ={ "brand": "Ford","model": "Mustang","year": 1964}
thisdict.pop("model")
print(thisdict)
```

Example

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
thisdict ={"brand": "Ford","model": "Mustang","year": 1964}
thisdict.popitem()
```

see more please visit: https://homeofpdf.com

```
print(thisdict)
```

Example

The del keyword removes the item with the specified key name:

```
thisdict ={"brand": "Ford","model": "Mustang","year": 1964}
del thisdict["model"]
print(thisdict)
```

Example

The del keyword can also delete the dictionary completely:

```
thisdict ={"brand": "Ford","model": "Mustang", "year": 1964}
del thisdict
print(thisdict) #this will cause an error because "thisdict" no longer exists.
```

Example

The clear() method empties the dictionary:

```
thisdict ={"brand": "Ford", "model": "Mustang","year": 1964}
thisdict.clear()
print(thisdict)
```

Python - Loop Dictionaries

Loop Through a Dictionary

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

Example

see more please visit: https://homeofpdf.com

Print all key names in the dictionary, one by one:

```
for x in thisdict:
  print(x)
```

Example

Print all *values* in the dictionary, one by one:

```
for x in thisdict:
  print(thisdict[x])
```

Example

You can also use the values() method to return values of a dictionary:

```
for x in thisdict.values():
  print(x)
```

Example

You can use the keys() method to return the keys of a dictionary:

```
for x in thisdict.keys():
  print(x)
```

Example

Loop through both *keys* and *values*, by using the items() method:

```
for x, y in thisdict.items():
  print(x, y)
```

Python - Copy Dictionaries

Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2

see more please visit: https://homeofpdf.com

will only be a *reference* to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

Example

Make a copy of a dictionary with the copy() method:

```
thisdict ={ "brand": "Ford","model": "Mustang","year": 1964}
mydict = thisdict.copy()
print(mydict)
```

Another way to make a copy is to use the built-in function dict().

Example

Make a copy of a dictionary with the dict() function:

```
thisdict ={"brand": "Ford","model": "Mustang","year": 1964}
mydict = dict(thisdict)
print(mydict)
```

Python - Nested Dictionaries

Nested Dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries.

Example

Create a dictionary that contain three dictionaries:

```
myfamily = {
```

see more please visit: https://homeofpdf.com

```
  "child1" : {"name" : "Emil","year" : 2004 },
  "child2" : {"name" : "Tobias","year" : 2007},
  "child3" : {"name" : "Linus","year" : 2011}
}
```

Or, if you want to add three dictionaries into a new dictionary:

Example

Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

```
child1 = {"name" : "Emil","year" : 2004}
child2 = {"name" : "Tobias","year" : 2007}
child3 = {"name" : "Linus", "year" : 2011}

myfamily = { "child1" : child1, "child2" : child2, "child3" : child3}
```

Python Dictionary Methods

see more please visit: https://homeofpdf.com

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary
clear()	Removes all the elements from the dictionary

18.Python math Module

see more please visit: https://homeofpdf.com

Python math Module

Python has a built-in module that you can use for mathematical tasks.

The math module has a set of methods and constants.

Math Constants

Constant	Description
math.e	Returns Euler's number (2.7182...)
math.inf	Returns a floating-point positive infinity
math.nan	Returns a floating-point NaN (Not a Number) value
math.pi	Returns PI (3.1415...)
math.tau	Returns tau (6.2831...)

Math Methods

math.acos() Returns the arc cosine of a number

math.acosh() Returns the inverse hyperbolic cosine of a number

math.asin() Returns the arc sine of a number

math.asinh() Returns the inverse hyperbolic sine of a number

math.atan() Returns the arc tangent of a number in radians

math.atan2() Returns the arc tangent of y/x in radians

math.atanh() Returns the inverse hyperbolic tangent of a number

math.ceil() Rounds a number up to the nearest integer

math.comb() Returns the number of ways to choose k items from n items without repetition and order

see more please visit: https://homeofpdf.com

math.copysign() Returns a float consisting of the value of the first parameter and the sign of the second paramete**r**

math.cos() Returns the cosine of a number

math.cosh() Returns the hyperbolic cosine of a number

math.degrees() Converts an angle from radians to degrees

math.dist() Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point

math.erf() Returns the error function of a number

math.erfc() Returns the complementary error function of a number

math.exp() Returns E raised to the power of x

math.expm1() Returns Ex - 1

math.fabs() Returns the absolute value of a number

math.factorial() Returns the factorial of a number

math.floor() Rounds a number down to the nearest integer

math.fmod() Returns the remainder of x/y

math.frexp() Returns the mantissa and the exponent, of a specified number

math.fsum() Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)

math.gamma() Returns the gamma function at x

math.gcd() Returns the greatest common divisor of two integers

math.hypot() Returns the Euclidean norm

math.isclose() Checks whether two values are close to each other, or not

see more please visit: https://homeofpdf.com

math.isfinite() Checks whether a number is finite or not

math.isinf() Checks whether a number is infinite or not

math.isnan() Checks whether a value is NaN (not a number) or not

math.isqrt() Rounds a square root number downwards to the nearest integer

math.ldexp() Returns the inverse of math.frexp() which is x * (2**i) of the given numbers x and i

math.lgamma() Returns the log gamma value of x

math.log() Returns the natural logarithm of a number, or the logarithm of number to base

math.log10() Returns the base-10 logarithm of x

math.log1p() Returns the natural logarithm of 1+x

math.log2() Returns the base-2 logarithm of x

math.perm() Returns the number of ways to choose k items from n items with order and without repetition

math.pow() Returns the value of x to the power of y

math.prod() Returns the product of all the elements in an iterable

math.radians() Converts a degree value into radians

math.remainder() Returns the closest value that can make numerator completely divisible by the denominator

math.sin() Returns the sine of a number

math.sinh() Returns the hyperbolic sine of a number

see more please visit: https://homeofpdf.com

math.sqrt() Returns the square root of a number

math.tan() Returns the tangent of a number

math.tanh() Returns the hyperbolic tangent of a number

math.trunc() Returns the truncated integer parts of a number

19.Python User Input

User Input

Python allows for user input.

That means we are able to ask the user for input.

The method is a bit different in Python 3.6 than Python 2.7.

Python 3.6 uses the input() method.

see more please visit: https://homeofpdf.com

Python 2.7 uses the raw_input() method.

The following example asks for the username, and when you entered the username, it gets printed on the screen:

Python 3.6

```
username = input("Enter username:")
print("Username is: " + username)
```

Python 2.7

```
username = raw_input("Enter username:")
print("Username is: " + username)
```

Python stops executing when it comes to the input() function, and continues when the user has given some input.

20.Python eval() Function

Example

Evaluate the expression 'print(55)':

```
x = 'print(55)'
eval(x)
```

Definition and Usage

The eval() function evaluates the specified expression, if the expression is a legal Python statement, it will be executed.

see more please visit: https://homeofpdf.com

Syntax

eval(*expression*, *globals*, *locals*)

Parameter Values

Parameter	Description
expression	A String, that will be evaluated as Python code
globals	Optional. A dictionary containing global parameters
locals	Optional. A dictionary containing local parameters

21.Python If ... Else

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if

see more please visit: https://homeofpdf.com

statements" and loops.

An "if statement" is written by using the if keyword.

Example

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

Elif

see more please visit: https://homeofpdf.com

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

Else

The else keyword catches anything which isn't caught by the preceding conditions.

Example

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

You can also have an else without the elif:

see more please visit: https://homeofpdf.com

Example

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

```
if a > b: print("a is greater than b")
```

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

This technique is known as **Ternary Operators**, or **Conditional Expressions**.

You can also have multiple else statements on the same line:

Example

see more please visit: https://homeofpdf.com

One line if else statement, with 3 conditions:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

And

The and keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

Or

The or keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c:

```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```

see more please visit: https://homeofpdf.com

Nested If

You can have if statements inside if statements, this is called *nested* if statements.

Example

```
x = 41

if x > 10:
        print("Above ten,")
  if x > 20:
     print("and also above 20!")
  else:
     print("but not above 20.")
```

The pass Statement

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

Example

```
a = 33
b = 200

if b > a:
  pass
```

22.Python While Loops

see more please visit: https://homeofpdf.com

Python Loops

Python has two primitive loop commands:

- while loops
- for loops

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Note: remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

The break Statement

With the break statement we can stop the loop even if the while condition is true:

Example

Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
```

see more please visit: https://homeofpdf.com

```
  if i == 3:
    break
  i += 1
```

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

Example

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

see more please visit: https://homeofpdf.com

23.Python For Loops

Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
        print(x)
```

The for loop does not require an indexing variable to set beforehand.

Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

Example

Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

see more please visit: https://homeofpdf.com

Example

Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

Example

Exit the loop when x is "banana", but this time the break comes before the print:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

Example

Do not print banana:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

see more please visit: https://homeofpdf.com

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example

Using the range() function:

```
for x in range(6):
    print(x)
```

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

Example

Using the start parameter:

```
for x in range(2, 6):
    print(x)
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):

Example

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
    print(x)
```

Else in For Loop

see more please visit: https://homeofpdf.com

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

Note: The else block will NOT be executed if the loop is stopped by a break statement.

Example

Break the loop when x is 3, and see what happens with the else block:

```
for x in range(6):
  if x == 3: break
    print(x)
else:
    print("Finally finished!")
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

Print each adjective for every fruit:

see more please visit: https://homeofpdf.com

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
  for y in fruits:
    print(x, y)
```

The pass Statement

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

Example

```
for x in [0, 1, 2]:
  pass
```

see more please visit: https://homeofpdf.com

24.Python Arrays

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

Arrays

Note: This page shows you how to use LISTS as ARRAYS, however, to work with arrays in Python you will have to import a library, like the NumPy library.

Arrays are used to store multiple values in one single variable:

Example

Create an array containing car names:

```
cars = ["Ford", "Volvo", "BMW"]
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford"

car2 = "Volvo"

car3 = "BMW"
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Access the Elements of an Array

see more please visit: https://homeofpdf.com

You refer to an array element by referring to the *index number*.

Example

Get the value of the first array item:

```
x = cars[0]
```

Example

Modify the value of the first array item:

```
cars[0] = "Toyota"
```

The Length of an Array

Use the len() method to return the length of an array (the number of elements in an array).

Example

Return the number of elements in the cars array:

```
x = len(cars)
```

Note: The length of an array is always one more than the highest array index.

Looping Array Elements

You can use the for in loop to loop through all the elements of an array.

Example

Print each item in the cars array:

```
for x in cars:
  print(x)
```

Adding Array Elements

You can use the append() method to add an element to an array.

see more please visit: https://homeofpdf.com

Example

Add one more element to the cars array:

cars.append("Honda")

Removing Array Elements

You can use the pop() method to remove an element from the array.

Examples

Delete the second element of the cars array:

cars.pop(1)

You can also use the remove() method to remove an element from the array.
Delete the element that has the value "Volvo":

cars.remove("Volvo")

Note: The list's remove() method only removes the first occurrence of the specified value.

Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value

see more please visit: https://homeofpdf.com

insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

25.Python Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

Example

```
def my_function():
  print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
def my_function():
  print("Hello from a function")


my_function()
```

see more please visit: https://homeofpdf.com

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

```
def my_function(fname):
  print(fname + " Refsnes")


my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Arguments are often shortened to *args* in Python documentations.

Parameters or Arguments?

The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called.

Number of Arguments

By default, a function must be called with the correct number of arguments.

see more please visit: https://homeofpdf.com

Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Example

This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

If you try to call the function with 1 or 3 arguments, you will get an error:

Example

This function expects 2 arguments, but gets only 1:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a *tuple* of arguments, and can access the items accordingly:

Example

If the number of arguments is unknown, add a * before the parameter name:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])
```

see more please visit: https://homeofpdf.com

```
my_function("Emil", "Tobias", "Linus")
```

Arbitrary Arguments are often shortened to **args* in Python documentations.

Keyword Arguments

You can also send arguments with the *key* = *value* syntax.

This way the order of the arguments does not matter.

Example

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

The phrase *Keyword Arguments* are often shortened to *kwargs* in Python documentations.

Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a *dictionary* of arguments, and can access the items accordingly:

Example

If the number of keyword arguments is unknown, add a double ** before the parameter name:

```
def my_function(**kid):
```

see more please visit: https://homeofpdf.com

```
  print("His last name is " + kid["lname"])


my_function(fname = "Tobias", lname = "Refsnes")
```

Arbitrary Kword Arguments are often shortened to ***kwargs* in Python documentations.

Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

Example

```
def my_function(country = "Norway"):
  print("I am from " + country)


my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

Example

see more please visit: https://homeofpdf.com

```
def my_function(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

Return Values

To let a function return a value, use the return statement:

Example

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

The pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

Example

```
def myfunction():
  pass
```

see more please visit: https://homeofpdf.com

Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

In this example, tri_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

Example

Recursion Example

```
def tri_recursion(k):
    if(k > 0):
        result = k + tri_recursion(k - 1)
        print(result)
    else:
        result = 0
    return result
print("\n\nRecursion Example Results")
```

see more please visit: https://homeofpdf.com

```
tri_recursion(6)
```

26.Python Lambda

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax

lambda *arguments* : *expression*

The expression is executed and the result is returned:

Example

Add 10 to argument a, and return the result:

```
x = lambda a : a + 10
print(x(5))
```

Lambda functions can take any number of arguments:

Example

see more please visit: https://homeofpdf.com

Multiply argument a with argument b and return the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Example

Summarize argument a, b, and c and return the result:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Why Use Lambda Functions?

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

```
def myfunc(n):
  return lambda a : a * n
```

Use that function definition to make a function that always doubles the number you send in:

Example

```
def myfunc(n):
  return lambda a : a * n
mydoubler = myfunc(2)
print(mydoubler(11))
```

Or, use the same function definition to make a function that always *triples* the number you send in:

Example

```
def myfunc(n):
  return lambda a : a * n
```

see more please visit: https://homeofpdf.com

```
mytripler = myfunc(3)

print(mytripler(11))
```

Or, use the same function definition to make both functions, in the same program:

Example

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

mytripler = myfunc(3)

print(mydoubler(11))

print(mytripler(11))
```

Use lambda functions when an anonymous function is required for a short period of time.

27.Python Classes and Objects

Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword class:

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

see more please visit: https://homeofpdf.com

Create Object

Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Note: The __init__() function is called automatically every time the class is being used to create a new object.

see more please visit: https://homeofpdf.com

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
```

see more please visit: https://homeofpdf.com

```
        print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Modify Object Properties

You can modify properties on objects like this:

Example

Set the age of p1 to 40:

```
p1.age = 40
```

Delete Object Properties

You can delete properties on objects by using the del keyword:

Example

Delete the age property from the p1 object:

```
del p1.age
```

Delete Objects

You can delete objects by using the del keyword:

Example

Delete the p1 object:

```
del p1
```

The pass Statement

class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

Example

```
class Person:
  pass
```

see more please visit: https://homeofpdf.com

28.Python Inheritance

Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Create a Parent Class

Any class can be a parent class, so the syntax is the same as creating any other class:

Example

Create a class named Person, with firstname and lastname properties, and a printname method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

see more please visit: https://homeofpdf.com

Create a Child Class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

Example

Create a class named Student, which will inherit the properties and methods from the Person class:

```
class Student(Person):
  pass
```

Note: Use the pass keyword when you do not want to add any other properties or methods to the class.

Now the Student class has the same properties and methods as the Person class.

Example

Use the Student class to create an object, and then execute the printname method:

```
x = Student("Mike", "Olsen")
x.printname()
```

Add the __init__() Function

So far we have created a child class that inherits the properties and methods from its parent. We want to add the __init__() function to the child class (instead of the pass keyword).

Note: The __init__() function is called automatically every time the class is being used to create a new object.

Example

Add the __init__() function to the Student class:

```
class Student(Person):
```

see more please visit: https://homeofpdf.com

```
  def __init__(self, fname, lname):
    #add properties etc.
```

When you add the __init__() function, the child class will no longer inherit the parent's __init__() function.

Note: The child's __init__() function **overrides** the inheritance of the parent's __init__() function. To keep the inheritance of the parent's __init__() function, add a call to the parent's __init__() function:

Example

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

Now we have successfully added the __init__() function, and kept the inheritance of the parent class, and we are ready to add functionality in the __init__() function.

Use the super() Function

Python also has a super() function that will make the child class inherit all the methods and properties from its parent:

Example

```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
```

By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

Add Properties

Example

see more please visit: https://homeofpdf.com

Add a property called graduationyear to the Student class:

```
class Student(Person):
  def __init__(self, fname, lname):
       super().__init__(fname, lname)
      self.graduationyear = 2019
```

In the example below, the year 2019 should be a variable, and passed into the Student class when creating student objects. To do so, add another parameter in the __init__() function:

Example

Add a year parameter, and pass the correct year when creating objects:

```
class Student(Person):
  def __init__(self, fname, lname, year):
       super().__init__(fname, lname)
      self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
```

Add Methods

Example

Add a method called welcome to the Student class:

```
class Student(Person):
  def __init__(self, fname, lname, year):
       super().__init__(fname, lname)
      self.graduationyear = year

  def welcome(self):
      print("Welcome", self.firstname, self.lastname, "to the class of",
```

see more please visit: https://homeofpdf.com

self.graduationyear)

If you add a method in the child class with the same name as a function in the parent class, the inheritance of the parent method will be overridden.

29.Python Iterators

Python Iterators

An iterator is an object that contains a countable number of values.

An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.

Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods __iter__() and __next__().

Iterator vs Iterable

Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable *containers* which you can get an iterator from.

All these objects have a iter() method which is used to get an iterator:

Example

Return an iterator from a tuple, and print each value:

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

Even strings are iterable objects, and can return an iterator:

see more please visit: https://homeofpdf.com

Example

Strings are also iterable objects, containing a sequence of characters:

```
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

Looping Through an Iterator

We can also use a for loop to iterate through an iterable object:

Example

Iterate the values of a tuple:

```
mytuple = ("apple", "banana", "cherry")

for x in mytuple:
  print(x)
```

Example

Iterate the characters of a string:

```
mystr = "banana"

for x in mystr:
  print(x)
```

The for loop actually creates an iterator object and executes the next() method for each loop.

Create an Iterator

see more please visit: https://homeofpdf.com

To create an object/class as an iterator you have to implement the methods __iter__() and __next__() to your object.

As you have learned in the Python Classes/Objects chapter, all classes have a function called __init__(), which allows you to do some initializing when the object is being created.

The __iter__() method acts similar, you can do operations (initializing etc.), but must always return the iterator object itself.

The __next__() method also allows you to do operations, and must return the next item in the sequence.

Example

Create an iterator that returns numbers, starting with 1, and each sequence will increase by one (returning 1,2,3,4,5 etc.):

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

StopIteration

see more please visit: https://homeofpdf.com

The example above would continue forever if you had enough next() statements, or if it was used in a for loop.

To prevent the iteration to go on forever, we can use the StopIteration statement.

In the __next__() method, we can add a terminating condition to raise an error if the iteration is done a specified number of times:

Example

Stop after 20 iterations:

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```

30.Python Scope

see more please visit: https://homeofpdf.com

A variable is only available from inside the region it is created. This is called **scope**.

Local Scope

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

Example

A variable created inside a function is available inside that function:

```
def myfunc():
  x = 300
  print(x)
myfunc()
```

Function Inside Function

As explained in the example above, the variable x is not available outside the function, but it is available for any function inside the function:

Example

The local variable can be accessed from a function within the function:

```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()
myfunc()
```

Global Scope

A variable created in the main body of the Python code is a global variable and belongs to the global scope.

see more please visit: https://homeofpdf.com

Global variables are available from within any scope, global and local.

Example

A variable created outside of a function is global and can be used by anyone:

```
x = 300
def myfunc():
  print(x)
myfunc()
print(x)
```

Naming Variables

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):

Example

The function will print the local x, and then the code will print the global x:

```
x = 300
def myfunc():
  x = 200
  print(x)
myfunc()
print(x)
```

Global Keyword

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.

The global keyword makes the variable global.

see more please visit: https://homeofpdf.com

Example

If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = 300
myfunc()
print(x)
```

Also, use the global keyword if you want to make a change to a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = 300
def myfunc():
  global x
  x = 200
myfunc()
print(x)
```

see more please visit: https://homeofpdf.com

30.Python Scope

A variable is only available from inside the region it is created. This is called **scope**.

Local Scope

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

Example

A variable created inside a function is available inside that function:

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

Function Inside Function

As explained in the example above, the variable x is not available outside the function, but it is available for any function inside the function:

Example

The local variable can be accessed from a function within the function:

```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```

Global Scope

A variable created in the main body of the Python code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

see more please visit: https://homeofpdf.com

Example

A variable created outside of a function is global and can be used by anyone:

```
x = 300

def myfunc():
  print(x)

myfunc()

print(x)
```

Naming Variables

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):

Example

The function will print the local x, and then the code will print the global x:

```
x = 300

def myfunc():
  x = 200
  print(x)

myfunc()

print(x)
```

Global Keyword

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.

The global keyword makes the variable global.

Example

If you use the global keyword, the variable belongs to the global scope:

see more please visit: https://homeofpdf.com

```
def myfunc():
  global x
  x = 300

myfunc()

print(x)
```

Also, use the global keyword if you want to make a change to a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = 300

def myfunc():
  global x
  x = 200
myfunc()
print(x)
```

31.Python Modules

What is a Module?

Consider a module to be the same as a code library.

see more please visit: https://homeofpdf.com

A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

Example

Save this code in a file named mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```

Note: When using a function from a module, use the syntax: *module_name.function_name*.

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Example

Save this code in the file mymodule.py

```
person1 = {
```

see more please visit: https://homeofpdf.com

```
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Naming a Module

You can name the module file whatever you like, but it must have the file extension .py

Re-naming a Module

You can create an alias when you import a module, by using the as keyword:

Example

Create an alias for mymodule called mx:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

see more please visit: https://homeofpdf.com

Example

Import and use the platform module:

```
import platform

x = platform.system()
print(x)
```

Using the dir() Function

There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

Example

List all the defined names belonging to the platform module:

```
import platform

x = dir(platform)
print(x)
```

Note: The dir() function can be used on *all* modules, also the ones you create yourself.

Import From Module

You can choose to import only parts from a module, by using the from keyword.

Example

The module named mymodule has one function and one dictionary:

see more please visit: https://homeofpdf.com

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import only the person1 dictionary from the module:

```
from mymodule import person1
print (person1["age"])
```

Note: When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1["age"], **not** ~~mymodule.person1["age"]~~

32.Python Datetime

Python Dates

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

Example

Import the datetime module and display the current date:

```
import datetime

x = datetime.datetime.now()
print(x)
```

Date Output

When we execute the code from the example above the result will be:

2021-07-04 15:19:53.819856

The date contains year, month, day, hour, minute, second, and microsecond.

The datetime module has many methods to return information about the date object.

Here are a few examples, you will learn more about them later in this chapter:

Example

Return the year and name of weekday:

```
import datetime
x = datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
```

Creating Date Objects

To create a date, we can use the datetime() class (constructor) of the datetime module.

The datetime() class requires three parameters to create a date: year, month,

see more please visit: https://homeofpdf.com

day.

Example

Create a date object:

```
import datetime
x = datetime.datetime(2020, 5, 17)
print(x)
```

The datetime() class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of 0, (None for timezone).

The strftime() Method

The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

Example

Display the name of the month:

```
import datetime
x = datetime.datetime(2018, 6, 1)
print(x.strftime("%B"))
```

A reference of all the legal format codes:

Directive	Description	Example

see more please visit: https://homeofpdf.com

%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day	52

see more please visit: https://homeofpdf.com

	of week, 00-53	
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

33.Python Math

Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.

see more please visit: https://homeofpdf.com

Built-in Math Functions

The min() and max() functions can be used to find the lowest or highest value in an iterable:

Example

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

The abs() function returns the absolute (positive) value of the specified number:

Example

```
x = abs(-7.25)

print(x)
```

The pow(*x*, *y*) function returns the value of x to the power of y (x^y).

Example

Return the value of 4 to the power of 3 (same as 4 * 4 * 4):

```
x = pow(4, 3)

print(x)
```

The Math Module

Python has also a built-in module called math, which extends the list of mathematical functions.

To use it, you must import the math module:

```
import math
```

When you have imported the math module, you can start using methods and constants of the module.

The math.sqrt() method for example, returns the square root of a number:

see more please visit: https://homeofpdf.com

Example

```
import math

x = math.sqrt(64)

print(x)
```

The math.ceil() method rounds a number upwards to its nearest integer, and the math.floor() method rounds a number downwards to its nearest integer, and returns the result:

Example

```
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x) # returns 2
print(y) # returns 1
```

The math.pi constant, returns the value of PI (3.14...):

Example

```
import math

x = math.pi

print(x)
```

Complete Math Module Reference

In our Math Module Reference you will find a complete reference of all methods and constants that belongs to the Math module.

34.Python JSON

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

see more please visit: https://homeofpdf.com

JSON in Python

Python has a built-in package called json, which can be used to work with JSON data.

Example

Import the json module:

```
import json
```

Parse JSON - Convert from JSON to Python

If you have a JSON string, you can parse it by using the json.loads() method.

The result will be a Python dictionary.

Example

Convert from JSON to Python:

```
import json

# some JSON:
x =  '{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
print(y["age"])
```

Convert from Python to JSON

If you have a Python object, you can convert it into a JSON string by using the json.dumps() method.

Example

Convert from Python to JSON:

```
import json
```

see more please visit: https://homeofpdf.com

```
# a Python object (dict):
x = {
  "name": "John",
  "age": 30,
  "city": "New York"
}

# convert into JSON:
y = json.dumps(x)

# the result is a JSON string:
print(y)
```

You can convert Python objects of the following types, into JSON strings:

- dict
- list
- tuple
- string
- int
- float
- True
- False
- None

Example

Convert Python objects into JSON strings, and print the values:

```
import json

print(json.dumps({"name": "John", "age": 30}))
print(json.dumps(["apple", "bananas"]))
print(json.dumps(("apple", "bananas")))
print(json.dumps("hello"))
print(json.dumps(42))
print(json.dumps(31.76))
print(json.dumps(True))
print(json.dumps(False))
```

see more please visit: https://homeofpdf.com

```python
print(json.dumps(None))
```

When you convert from Python to JSON, Python objects are converted into the JSON (JavaScript) equivalent:

Python	JSON
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	true
False	false
None	null

Example

Convert a Python object containing all the legal data types:

```python
import json

x = {
  "name": "John",
```

see more please visit: https://homeofpdf.com

```
  "age": 30,
  "married": True,
  "divorced": False,
  "children": ("Ann","Billy"),
  "pets": None,
  "cars": [
    {"model": "BMW 230", "mpg": 27.5},
    {"model": "Ford Edge", "mpg": 24.1}
  ]
}
print(json.dumps(x))
```

Format the Result

The example above prints a JSON string, but it is not very easy to read, with no indentations and line breaks.

The json.dumps() method has parameters to make it easier to read the result:

Example

Use the indent parameter to define the numbers of indents:

```
json.dumps(x, indent=4)
```

You can also define the separators, default value is (", ", ": "), which means using a comma and a space to separate each object, and a colon and a space to separate keys from values:

Example

Use the separators parameter to change the default separator:

```
json.dumps(x, indent=4, separators=(". ", " = "))
```

Order the Result

see more please visit: https://homeofpdf.com

The json.dumps() method has parameters to order the keys in the result:

Example

Use the sort_keys parameter to specify if the result should be sorted or not:

```
json.dumps(x, indent=4, sort_keys=True)
```

35.Python RegEx

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

RegEx Module

Python has a built-in package called re, which can be used to work with Regular Expressions.

Import the re module:

```
import re
```

see more please visit: https://homeofpdf.com

RegEx in Python

When you have imported the re module, you can start using regular expressions:

Example

Search the string to see if it starts with "The" and ends with "Spain":

```
import re
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```

RegEx Functions

The re module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

Metacharacters

Metacharacters are characters with a special meaning:

Character	Description	Example
[ ]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"

see more please visit: https://homeofpdf.com

^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{ }	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
( )	Capture and group	

Sets

A set is a set of characters inside a pair of square brackets [] with a special meaning:

Set	Description
[arn]	Returns a match where one of the specified characters (a, r, or n) are present
[a-n]	Returns a match for any lower case character, alphabetically between a and n
[^arn]	Returns a match for any character EXCEPT a, r, and n
[0123]	Returns a match where any of the specified digits (0, 1, 2, or 3) are present
[0-9]	Returns a match for any digit between 0 and 9
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string.

see more please visit: https://homeofpdf.com

Special Sequences

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string"	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT	"\W"

see more please visit: https://homeofpdf.com

	contain any word characters	
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

The findall() Function

The findall() function returns a list containing all matches.

Example

Print a list of all matches:

```
import re
txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

The list contains the matches in the order they are found.

If no matches are found, an empty list is returned:

Example

Return an empty list if no match was found:

```
import re
txt = "The rain in Spain"
x = re.findall("Portugal", txt)
print(x)
```

The search() Function

The search() function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned:

see more please visit: https://homeofpdf.com

Example

Search for the first white-space character in the string:

```
import re
txt = "The rain in Spain"
x = re.search("\s", txt)
print("The first white-space character is located in position:", x.start())
```

If no matches are found, the value None is returned:

Example

Make a search that returns no match:

```
import re
txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

The split() Function

The split() function returns a list where the string has been split at each match:

Example

Split at each white-space character:

```
import re
txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

You can control the number of occurrences by specifying the maxsplit parameter:

see more please visit: https://homeofpdf.com

Example

Split the string only at the first occurrence:

```
import re
txt = "The rain in Spain"
x = re.split("\s", txt, 1)
print(x)
```

The sub() Function

The sub() function replaces the matches with the text of your choice:

Example

Replace every white-space character with the number 9:

```
import re
txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

You can control the number of replacements by specifying the count parameter:

Example

Replace the first 2 occurrences:

```
import re
txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)
print(x)
```

Match Object

see more please visit: https://homeofpdf.com

A Match Object is an object containing information about the search and the result.

Note: If there is no match, the value None will be returned, instead of the Match Object.

Example

Do a search that will return a Match Object:

```
import re

txt = "The rain in Spain"

x = re.search("ai", txt)

print(x) #this will print an object
```

The Match object has properties and methods used to retrieve information about the search, and the result:

.span() returns a tuple containing the start-, and end positions of the match.

.string returns the string passed into the function

.group() returns the part of the string where there was a match

Example

Print the position (start- and end-position) of the first match occurrence.

The regular expression looks for any words that starts with an upper case "S":

```
import re

txt = "The rain in Spain"

x = re.search(r"\bS\w+", txt)

print(x.span())
```

Example

Print the string passed into the function:

see more please visit: https://homeofpdf.com

```
import re
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
```

Example

Print the part of the string where there was a match.

The regular expression looks for any words that starts with an upper case "S":

```
import re
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```

Note: If there is no match, the value None will be returned, instead of the Match Object.

see more please visit: https://homeofpdf.com

36.Python PIP

What is PIP?

PIP is a package manager for Python packages, or modules if you like.

Note: If you have Python version 3.4 or later, PIP is included by default.

What is a Package?

A package contains all the files you need for a module.

Modules are Python code libraries you can include in your project.

Check if PIP is Installed

Navigate your command line to the location of Python's script directory, and type the following:

Example

Check PIP version:

C:\Users*Your Name*\AppData\Local\Programs\Python\Python36-32\Scripts>pip --version

Install PIP

If you do not have PIP installed, you can download and install it from this page: https://pypi.org/project/pip/

Download a Package

Downloading a package is very easy.

Open the command line interface and tell PIP to download the package you want.

Navigate your command line to the location of Python's script directory, and type the following:

Example

see more please visit: https://homeofpdf.com

Download a package named "camelcase":

C:\Users*Your Name*\AppData\Local\Programs\Python\Python36-32\Scripts>pip install camelcase

Now you have downloaded and installed your first package!

Using a Package

Once the package is installed, it is ready to use.

Import the "camelcase" package into your project.

Example

Import and use "camelcase":

```
import camelcase

c = camelcase.CamelCase()

txt = "hello world"

print(c.hump(txt))
```

Find Packages

Find more packages at https://pypi.org/.

Remove a Package

Use the uninstall command to remove a package:

Example

Uninstall the package named "camelcase":

C:\Users*Your Name*\AppData\Local\Programs\Python\Python36-32\Scripts>pip uninstall camelcase

The PIP Package Manager will ask you to confirm that you want to remove the camelcase package:

Uninstalling camelcase-02.1:

see more please visit: https://homeofpdf.com

```
   Would remove:
         c:\users\Your Name\appdata\local\programs\python\python36-32\lib\site-packages\camecase-0.2-py3.6.egg-info
         c:\users\Your Name\appdata\local\programs\python\python36-32\lib\site-packages\camecase\*
Proceed (y/n)?
```

Press y and the package will be removed.

List Packages

Use the list command to list all the packages installed on your system:

Example

List installed packages:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip list
```

Result:

```
Package              Version
-----------------------
camelcase            0.2
mysql-connector 2.1.6
pip              18.1
pymongo              3.6.1
setuptools           39.0.1
```

37.Python Try Except

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

Examples

The try block will generate an exception, because x is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Since the try block raises an error, the except block will be executed.

Without the try block, the program will crash and raise an error:

This statement will raise an error, because x is not defined:

```
print(x)
```

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

Print one message if the try block raises a NameError and another for other errors:

```
try:
  print(x)
```

see more please visit: https://homeofpdf.com

```
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Else

You can use the else keyword to define a block of code to be executed if no errors were raised:

In this example, the try block does not generate any error:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

Finally

The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

This can be useful to close objects and clean up resources:

see more please visit: https://homeofpdf.com

Example

Try to open and write to a file that is not writable:

```
try:
  f = open("demofile.txt")
  f.write("Lorum Ipsum")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()
```

The program can continue, without leaving the file object open.

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

Example

Raise an error and stop the program if x is lower than 0:

```
x = -1
if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Raise a TypeError if x is not an integer:

```
x = "hello"
if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

see more please visit: https://homeofpdf.com

38.Python String Formatting

To make sure a string will display as expected, we can format the result with the format() method.

String format()

The format() method allows you to format selected parts of a string.

Sometimes there are parts of a text that you do not control, maybe they come from a database, or user input?

To control such values, add placeholders (curly brackets {}) in the text, and run the values through the format() method:

Example

Add a placeholder where you want to display the price:

```
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```

You can add parameters inside the curly brackets to specify how to convert the value:

Example

Format the price to be displayed as a number with two decimals:

```
txt = "The price is {:.2f} dollars"
```

Check out all formatting types in our String format() Reference.

Multiple Values

If you want to use more values, just add more values to the format() method:

```
print(txt.format(price, itemno, count))
```

And add more placeholders:

Example

```
quantity = 3
```

see more please visit: https://homeofpdf.com

```
itemno = 567
price = 49
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

Index Numbers

You can use index numbers (a number inside the curly brackets {0}) to be sure the values are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

Also, if you want to refer to the same value more than once, use the index number:

Example

```
age = 36
name = "John"
txt = "His name is {1}. {1} is {0} years old."
print(txt.format(age, name))
```

Named Indexes

You can also use named indexes by entering a name inside the curly brackets {carname}, but then you must use names when you pass the parameter values txt.format(carname = "Ford"):

Example

```
myorder = "I have a {carname}, it is a {model}."
print(myorder.format(carname = "Ford", model = "Mustang"))
```

see more please visit: https://homeofpdf.com

39.Python File Handling

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

File Handling

The key function for working with files in Python is the open() function.

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need

see more please visit: https://homeofpdf.com

to specify them.

Note: Make sure the file exists, or else you will get an error.

Python File Open

Open a File on the Server

Assume we have the following file, located in the same folder as Python:

demofile.txt

Hello! Welcome to demofile.txt

This file is for testing purposes.

Good Luck!

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

Example

```
f = open("demofile.txt", "r")

print(f.read())
```

If the file is located in a different location, you will have to specify the file path, like this:

Example

Open a file on a different location:

```
f = open("D:\\myfiles\welcome.txt", "r")

print(f.read())
```

Read Only Parts of the File

see more please visit: https://homeofpdf.com

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

Example

Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")

print(f.read(5))
```

Read Lines

You can return one line by using the readline() method:

Example

Read one line of the file:

```
f = open("demofile.txt", "r")

print(f.readline())
```

By calling readline() two times, you can read the two first lines:

Example

Read two lines of the file:

```
f = open("demofile.txt", "r")

print(f.readline())

print(f.readline())
```

By looping through the lines of the file, you can read the whole file, line by line:

Example

Loop through the file line by line:

```
f = open("demofile.txt", "r")

for x in f:
```

see more please visit: https://homeofpdf.com

```
print(x)
```

Close Files

It is a good practice to always close the file when you are done with it.

Example

Close the file when you are finish with it:

```
f = open("demofile.txt", "r")

print(f.readline())

f.close()
```

Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

Python File Write

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Example

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
```

see more please visit: https://homeofpdf.com

```
print(f.read())
```

Example

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

Note: the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Example

Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

see more please visit: https://homeofpdf.com

Python Delete File

Delete a File

To delete a file, you must import the OS module, and run its os.remove() function:

Example

Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example

Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the os.rmdir() method:

Example

Remove the folder "myfolder":

see more please visit: https://homeofpdf.com

```
import os
os.rmdir("myfolder")
```

Note: You can only remove *empty* folders.

40.NumPy Tutorial

NumPy is a Python library.

NumPy is used for working with arrays.

NumPy is short for "Numerical Python".

Learning by Reading

We have created 43 tutorial pages for you to learn more about NumPy.

Starting with a basic introduction and ends up with creating and plotting random data sets, and working with NumPy functions:

Basic

- Introduction
- Getting Started
- Creating Arrays
- Array Indexing
- Array Slicing
- Data Types
- Copy vs View
- Array Shape
- Array Reshape
- Array Iterating
- Array Join
- Array Split
- Array Search
- Array Sort

see more please visit: https://homeofpdf.com

- Array Filter

Random

- Random Intro
- Data Distribution
- Random Permutation
- Seaborn Module
- Normal Dist.
- Binomial Dist.
- Poisson Dist.
- Uniform Dist.
- Logistic Dist.
- Multinomial Dist.
- Exponential Dis.
- Chi Square Dist.
- Rayleigh Dist.
- Pareto Dist.
- Zipf Dist.

Ufunc

- ufunc Intro
- Create Function
- Simple Arithmetic
- Rounding Decimals
- Logs
- Summations
- Products
- Differences
- Finding LCM
- Finding GCD
- Trigonometric
- Hyperbolic
- Set Operations

see more please visit: https://homeofpdf.com

NumPy Introduction

What is NumPy?

NumPy is a Python library used for working with arrays.

It also has functions for working in the domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with the latest CPU architectures.

Which Language is NumPy written in?

see more please visit: https://homeofpdf.com

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Where is the NumPy Codebase?

The source code for NumPy is located at this github repository https://github.com/numpy/numpy

github: enables many people to work on the same codebase.

NumPy Getting Started

Installation of NumPy

If you have Python and PIP already installed on a system, then installation of NumPy is very easy.

Install it using this command:

C:\Users*Your Name*>pip install numpy

If this command fails, then use a python distribution that already has NumPy installed like Anaconda, Spyder etc.

Import NumPy

Once NumPy is installed, import it in your applications by adding the import keyword:

```
import numpy
```

Now NumPy is imported and ready to use.

Example

```
import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```

see more please visit: https://homeofpdf.com

NumPy as np

NumPy is usually imported under the np alias.

alias: In Python aliases are an alternate name for referring to the same thing.

Create an alias with the as keyword while importing:

```
import numpy as np
```

Now the NumPy package can be referred to as np instead of numpy.

Example

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

Checking NumPy Version

The version string is stored under __version__ attribute.

Example

```
import numpy as np

print(np.__version__)
```

NumPy Creating Arrays

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

Example

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
```

see more please visit: https://homeofpdf.com

```
print(arr)

print(type(arr))
```

type(): This built-in Python function tells us the type of the object passed to it. Like in above code it shows that arr is numpy.ndarray type.

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

Example

Use a tuple to create a NumPy array:

```
import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)
```

Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

nested array: are arrays that have arrays as their elements.

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example

Create a 0-D array with value 42

```
import numpy as np

arr = np.array(42)

print(arr)
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

see more please visit: https://homeofpdf.com

These are the most common and basic arrays.

Example

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

Example

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np
```

see more please visit: https://homeofpdf.com

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

Check Number of Dimensions?

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array has.

Example

Check how many dimensions the arrays have:

```
import numpy as np

a = np.array(42)

b = np.array([1, 2, 3, 4, 5])

c = np.array([[1, 2, 3], [4, 5, 6]])

d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)

print(b.ndim)

print(c.ndim)

print(d.ndim)
```

Higher Dimensional Arrays

An array can have any number of dimensions.

When the array is created, you can define the number of dimensions by using the ndmin argument.

Example

Create an array with 5 dimensions and verify that it has 5 dimensions:

```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
```

see more please visit: https://homeofpdf.com

```
print('number of dimensions :', arr.ndim)
```

In this array the innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dim has 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

NumPy Array Indexing

Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example

Get the first element from the following array:

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])
```

Example

Get the second element from the following array.

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[1])
```

Example

see more please visit: https://homeofpdf.com

Get third and fourth elements from the following array and add them.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])
```

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Example

Access the 2nd element on 1st dim:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st dim: ', arr[0, 1])
```

Example

Access the 5th element on 2nd dim:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd dim: ', arr[1, 4])
```

Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Example

Access the third element of the second array of the first array:

see more please visit: https://homeofpdf.com

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

Example Explained

arr[0, 1, 2] prints the value 6.

And this is why:

The first number represents the first dimension, which contains two arrays:

[[1, 2, 3], [4, 5, 6]]

and:

[[7, 8, 9], [10, 11, 12]]

Since we selected 0, we are left with the first array:

[[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays:

[1, 2, 3]

and:

[4, 5, 6]

Since we selected 1, we are left with the second array:

[4, 5, 6]

The third number represents the third dimension, which contains three values:

4

5

6

Since we selected 2, we end up with the third value:

6

see more please visit: https://homeofpdf.com

Negative Indexing

Use negative indexing to access an array from the end.

Example

Print the last element from the 2nd dim:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

NumPy Array Slicing

Slicing arrays

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [*start*:*end*].

We can also define the step, like this: [*start*:*end*:*step*].

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

Example

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```

Note: The result *includes* the start index, but *excludes* the end index.

Example

Slice elements from index 4 to the end of the array:

see more please visit: https://homeofpdf.com

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

Example

Slice elements from the beginning to index 4 (not included):

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[:4])
```

Negative Slicing

Use the minus operator to refer to an index from the end:

Example

Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

STEP

Use the step value to determine the step of the slicing:

Example

Return every other element from index 1 to index 5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

see more please visit: https://homeofpdf.com

Example

Return every other element from the entire array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[::2])
```

Slicing 2-D Arrays

Example

From the second element, slice elements from index 1 to index 4 (not included):

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

Note: Remember that the second *element* has index 1.

Example

From both elements, return index 2:

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 2])
```

Example

From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 1:4])
```

see more please visit: https://homeofpdf.com

NumPy Data Types

Data Types in Python

By default Python have these data types:

- strings - used to represent text data, the text is given under quote marks. e.g. "ABCD"
- integer - used to represent integer numbers. e.g. -1, -2, -3
- float - used to represent real numbers. e.g. 1.2, 42.42
- boolean - used to represent True or False.
- complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

Data Types in NumPy

NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

Checking the Data Type of an Array

The NumPy array object has a property called dtype that returns the data type of the array:

see more please visit: https://homeofpdf.com

Example

Get the data type of an array object:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

Example

Get the data type of an array containing strings:

```
import numpy as np
arr = np.array(['apple', 'banana', 'cherry'])
print(arr.dtype)
```

Creating Arrays With a Defined Data Type

We use the array() function to create arrays, this function can take an optional argument: dtype that allows us to define the expected data type of the array elements:

Example

Create an array with data type string:

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
```

For i, u, f, S and U we can define size as well.

Example

Create an array with data type 4 bytes integer:

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='i4')
```

see more please visit: https://homeofpdf.com

```
print(arr)

print(arr.dtype)
```

What if a Value Can Not Be Converted?

If a type is given in which elements can't be casted then NumPy will raise a ValueError.

ValueError: In Python ValueError is raised when the type of passed argument to a function is unexpected/incorrect.

Example

A non integer string like 'a' can not be converted to integer (will raise an error):

```
import numpy as np

arr = np.array(['a', '2', '3'], dtype='i')
```

Converting Data Type on Existing Arrays

The best way to change the data type of an existing array, is to make a copy of the array with the astype() method.

The astype() function creates a copy of the array, and allows you to specify the data type as a parameter.

The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

Example

Change data type from float to integer by using 'i' as parameter value:

```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')
```

see more please visit: https://homeofpdf.com

```
print(newarr)
print(newarr.dtype)
```

Example

Change data type from float to integer by using int as parameter value:

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype(int)
print(newarr)
print(newarr.dtype)
```

Example

Change data type from integer to boolean:

```
import numpy as np
arr = np.array([1, 0, 3])
newarr = arr.astype(bool)
print(newarr)
print(newarr.dtype)
```

NumPy Array Copy vs View

The Difference Between Copy and View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The copy *owns* the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The view *does not own* the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the

see more please visit: https://homeofpdf.com

view.

COPY:

Example

Make a copy, change the original array, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

x = arr.copy()

arr[0] = 42

print(arr)

print(x)
```

The copy SHOULD NOT be affected by the changes made to the original array.

VIEW:

Example

Make a view, change the original array, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

x = arr.view()

arr[0] = 42

print(arr)

print(x)
```

The view SHOULD be affected by the changes made to the original array.

Make Changes in the VIEW:

Example

see more please visit: https://homeofpdf.com

Make a view, change the view, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

x = arr.view()

x[0] = 31

print(arr)

print(x)
```

The original array SHOULD be affected by the changes made to the view.

Check if Array Owns it's Data

As mentioned above, copies *owns* the data, and views *does not own* the data, but how can we check this?

Every NumPy array has the attribute base that returns None if the array owns the data.

Otherwise, the base attribute refers to the original object.

Example

Print the value of the base attribute to check if an array owns it's data or not:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

x = arr.copy()

y = arr.view()

print(x.base)

print(y.base)
```

The copy returns None.

The view returns the original array.

see more please visit: https://homeofpdf.com

NumPy Array Shape

Shape of an Array

The shape of an array is the number of elements in each dimension.

Get the Shape of an Array

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

Example

Print the shape of a 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```

The example above returns (2, 4), which means that the array has 2 dimensions, and each dimension has 4 elements.

Example

Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4:

```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)

print('shape of array :', arr.shape)
```

What does the shape tuple represent?

Integers at every index tells about the number of elements the corresponding dimension has.

In the example above at index-4 we have value 4, so we can say that 5th (4 + 1 th) dimension has 4 elements.

see more please visit: https://homeofpdf.com

NumPy Array Reshaping

Reshaping arrays

Reshaping means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

Reshape From 1-D to 2-D

Example

Convert the following 1-D array with 12 elements into a 2-D array.

The outermost dimension will have 4 arrays, each with 3 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

Reshape From 1-D to 3-D

Example

Convert the following 1-D array with 12 elements into a 3-D array.

The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(2, 3, 2)

print(newarr)
```

see more please visit: https://homeofpdf.com

Can We Reshape Into Any Shape?

Yes, as long as the elements required for reshaping are equal in both shapes.

We can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3 elements 3 rows 2D array as that would require 3x3 = 9 elements.

Example

Try converting 1D array with 8 elements to a 2D array with 3 elements in each dimension (will raise an error):

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(3, 3)

print(newarr)
```

Return Copy or View?

Example

Check if the returned array is a copy or a view:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

print(arr.reshape(2, 4).base)
```

The example above returns the original array, so it is a view.

Unknown Dimension

You are allowed to have one "unknown" dimension.

Meaning that you do not have to specify an exact number for one of the dimensions in the reshape method.

Pass -1 as the value, and NumPy will calculate this number for you.

Example

see more please visit: https://homeofpdf.com

Convert 1D array with 8 elements to 3D array with 2x2 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(2, 2, -1)

print(newarr)
```

Note: We can not pass -1 to more than one dimension.

Flattening the arrays

Flattening array means converting a multidimensional array into a 1D array.

We can use reshape(-1) to do this.

Example

Convert the array into a 1D array:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

newarr = arr.reshape(-1)


print(newarr)
```

Note: There are a lot of functions for changing the shapes of arrays in numpy, flatten, ravel and also for rearranging the elements rot90, flip, fliplr, flipud etc. These fall under the Intermediate to Advanced section of numpy.

NumPy Array Iterating

Iterating Arrays

Iterating means going through elements one by one.

As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.

see more please visit: https://homeofpdf.com

If we iterate on a 1-D array it will go through each element one by one.

Example

Iterate on the elements of the following 1-D array:

```
import numpy as np

arr = np.array([1, 2, 3])

for x in arr:
  print(x)
```

Iterating 2-D Arrays

In a 2-D array it will go through all the rows.

Example

Iterate on the elements of the following 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  print(x)
```

If we iterate on a *n*-D array it will go through n-1th dimension one by one.

To return the actual values, the scalars, we have to iterate the arrays in each dimension.

Example

Iterate on each scalar element of the 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  for y in x:
    print(y)
```

see more please visit: https://homeofpdf.com

Iterating 3-D Arrays

In a 3-D array it will go through all the 2-D arrays.

Example

Iterate on the elements of the following 3-D array:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

for x in arr:

  print(x)
```

To return the actual values, the scalars, we have to iterate the arrays in each dimension.

Example

Iterate down to the scalars:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

for x in arr:

  for y in x:

    for z in y:

      print(z)
```

Iterating Arrays Using nditer()

The function nditer() is a helping function that can be used from very basic to very advanced iterations. It solves some basic issues which we face in iteration, let's go through it with examples.

Iterating on Each Scalar Element

In basic for loops, iterating through each scalar of an array we need to use *n* for loops which can be difficult to write for arrays with very high dimensionality.

see more please visit: https://homeofpdf.com

Example

Iterate through the following 3-D array:

```
import numpy as np

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):
  print(x)
```

Iterating Array With Different Data Types

We can use the op_dtypes argument and pass it the expected datatype to change the datatype of elements while iterating.

NumPy does not change the data type of the element in-place (where the element is in array) so it needs some other space to perform this action, that extra space is called buffer, and in order to enable it in nditer() we pass flags=['buffered'].

Example

Iterate through the array as a string:

```
import numpy as np

arr = np.array([1, 2, 3])

for x in np.nditer(arr, flags=['buffered'], op_dtypes=['S']):
  print(x)
```

Iterating With Different Step Size

We can use filtering and followed by iteration.

Example

Iterate through every scalar element of the 2D array skipping 1 element:

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
```

see more please visit: https://homeofpdf.com

```
for x in np.nditer(arr[:, ::2]):
  print(x)
```

Enumerated Iteration Using ndenumerate()

Enumeration means mentioning the sequence number of somethings one by one.

Sometimes we require a corresponding index of the element while iterating, the ndenumerate() method can be used for those use cases.

Example

Enumerate on following 1D arrays elements:

```
import numpy as np

arr = np.array([1, 2, 3])

for idx, x in np.ndenumerate(arr):
  print(idx, x)
```

Example

Enumerate on following 2D array's elements:

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for idx, x in np.ndenumerate(arr):
  print(idx, x)
```

NumPy Joining Array

Joining NumPy Arrays

Joining means putting contents of two or more arrays in a single array.

In SQL we join tables based on a key, whereas in NumPy we join arrays by axes.

see more please visit: https://homeofpdf.com

We pass a sequence of arrays that we want to join to the concatenate() function, along with the axis. If the axis is not explicitly passed, it is taken as 0.

Example

Join two arrays

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

Example

Join two 2-D arrays along rows (axis=1):

```
import numpy as np

arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

arr = np.concatenate((arr1, arr2), axis=1)

print(arr)
```

Joining Arrays Using Stack Functions

Stacking is the same as concatenation, the only difference is that stacking is done along a new axis.

We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, ie. stacking.

We pass a sequence of arrays that we want to join to the stack() method along with the axis. If the axis is not explicitly passed it is taken as 0.

Example

```
import numpy as np
```

see more please visit: https://homeofpdf.com

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.stack((arr1, arr2), axis=1)

print(arr)
```

Stacking Along Rows

NumPy provides a helper function: hstack() to stack along rows.

Example

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.hstack((arr1, arr2))

print(arr)
```

Stacking Along Columns

NumPy provides a helper function: vstack() to stack along columns.

Example

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.vstack((arr1, arr2))

print(arr)
```

Stacking Along Height (depth)

NumPy provides a helper function: dstack() to stack along height, which is the same as depth.

Example

see more please visit: https://homeofpdf.com

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.dstack((arr1, arr2))
print(arr)
```

NumPy Splitting Array

Splitting NumPy Arrays

Splitting is the reverse operation of Joining.

Joining merges multiple arrays into one and Splitting breaks one array into multiple.

We use array_split() for splitting arrays, we pass it the array we want to split and the number of splits.

Example

Split the array in 3 parts:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
```

Note: The return value is an array containing three arrays.

If the array has less elements than required, it will adjust from the end accordingly.

Example

Split the array in 4 parts:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
```

see more please visit: https://homeofpdf.com

```
newarr = np.array_split(arr, 4)

print(newarr)
```

Note: We also have the method split() available but it will not adjust the elements when elements are less in source array for splitting like in example above, array_split() worked properly but split() would fail.

Split Into Arrays

The return value of the array_split() method is an array containing each of the splits as an array.

If you split an array into 3 arrays, you can access them from the result just like any array element:

Example

Access the splitted arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr[0])

print(newarr[1])

print(newarr[2])
```

Splitting 2-D Arrays

Use the same syntax when splitting 2-D arrays.

Use the array_split() method, pass in the array you want to split and the number of splits you want to do.

Example

Split the 2-D array into three 2-D arrays.

```
import numpy as np

arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
```

see more please visit: https://homeofpdf.com

```
newarr = np.array_split(arr, 3)

print(newarr)
```

The example above returns three 2-D arrays.

Let's look at another example, this time each element in the 2-D arrays contains 3 elements.

Example

Split the 2-D array into three 2-D arrays.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16,
17, 18]])

newarr = np.array_split(arr, 3)

print(newarr)
```

The example above returns three 2-D arrays.

In addition, you can specify which axis you want to do the split around.

The example below also returns three 2-D arrays, but they are split along the row (axis=1).

Example

Split the 2-D array into three 2-D arrays along rows.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16,
17, 18]])

newarr = np.array_split(arr, 3, axis=1)

print(newarr)
```

An alternate solution is using hsplit() opposite of hstack()

Example

Use the hsplit() method to split the 2-D array into three 2-D arrays along rows.

see more please visit: https://homeofpdf.com

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

newarr = np.hsplit(arr, 3)

print(newarr)
```

Note: Similar alternatives to vstack() and dstack() are available as vsplit() and dsplit().

NumPy Searching Arrays

Searching Arrays

You can search an array for a certain value, and return the indexes that get a match.

To search an array, use the where() method.

Example

Find the indexes where the value is 4:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr == 4)

print(x)
```

The example above will return a tuple: (array([3, 5, 6],)

Which means that the value 4 is present at index 3, 5, and 6.

Example

Find the indexes where the values are even:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.where(arr%2 == 0)
```

see more please visit: https://homeofpdf.com

```python
print(x)
```

Example

Find the indexes where the values are odd:

```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.where(arr%2 == 1)

print(x)
```

Search Sorted

There is a method called searchsorted() which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

The searchSorted() method is assumed to be used on sorted arrays.

Example

Find the indexes where the value 7 should be inserted:

```python
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7)

print(x)
```

Example explained: The number 7 should be inserted on index 1 to remain the sort order.

The method starts the search from the left and returns the first index where the number 7 is no longer larger than the next value.

Search From the Right Side

By default the leftmost index is returned, but we can give side='right' to return the rightmost index instead.

Example

see more please visit: https://homeofpdf.com

Find the indexes where the value 7 should be inserted, starting from the right:

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7, side='right')

print(x)
```

Example explained: The number 7 should be inserted on index 2 to remain the sort order.

The method starts the search from the right and returns the first index where the number 7 is no longer less than the next value.

Multiple Values

To search for more than one value, use an array with the specified values.

Example

Find the indexes where the values 2, 4, and 6 should be inserted:

```
import numpy as np

arr = np.array([1, 3, 5, 7])

x = np.searchsorted(arr, [2, 4, 6])

print(x)
```

The return value is an array: [1 2 3] containing the three indexes where 2, 4, 6 would be inserted in the original array to maintain the order.

NumPy Sorting Arrays

Sorting Arrays

Sorting means putting elements in an *ordered sequence*.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a

see more please visit: https://homeofpdf.com

specified array.

Example

Sort the array:

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

Note: This method returns a copy of the array, leaving the original array unchanged.

You can also sort arrays of strings, or any other data type:

Example

Sort the array alphabetically:

```
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

Example

Sort a boolean array:

```
import numpy as np

arr = np.array([True, False, True])

print(np.sort(arr))
```

Sorting a 2-D Array

If you use the sort() method on a 2-D array, both arrays will be sorted:

Example

Sort a 2-D array:

```
import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])
```

see more please visit: https://homeofpdf.com

```
print(np.sort(arr))
```

NumPy Filter Array

Filtering Arrays

Getting some elements out of an existing array and creating a new array out of them is called *filtering*.

In NumPy, you filter an array using a *boolean index list*.

A *boolean index list* is a list of booleans corresponding to indexes in the array.

If the value at an index is True that element is contained in the filtered array, if the value at that index is False that element is excluded from the filtered array.

Example

Create an array from the elements on index 0 and 2:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```

The example above will return [41, 43], why?

Because the new filter contains only the values where the filter array had the value True, in this case, index 0 and 2.

Creating the Filter Array

In the example above we hard-coded the True and False values, but the common use is to create a filter array based on conditions.

Example

see more please visit: https://homeofpdf.com

Create a filter array that will return only values higher than 42:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

# Create an empty list

filter_arr = []

# go through each element in arr

for element in arr:

  # if the element is higher than 42, set the value to True, otherwise
False:

  if element > 42:

       filter_arr.append(True)

  else:

       filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)

print(newarr)
```

Example

Create a filter array that will return only even elements from the original array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

# Create an empty list

filter_arr = []

# go through each element in arr

for element in arr:

  # if the element is completely divisble by 2, set the value to True,
```

see more please visit: https://homeofpdf.com

```
otherwise False
  if element % 2 == 0:
        filter_arr.append(True)
  else:
        filter_arr.append(False)
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

Creating Filter Directly From Array

The above example is quite a common task in NumPy and NumPy provides a nice way to tackle it.

We can directly substitute the array instead of the iterable variable in our condition and it will work just as we expect it to.

Example

Create a filter array that will return only values higher than 42:

```
import numpy as np
arr = np.array([41, 42, 43, 44])
filter_arr = arr > 42
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

Example

Create a filter array that will return only even elements from the original array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
```

see more please visit: https://homeofpdf.com

```
filter_arr = arr % 2 == 0
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

Random Numbers in NumPy

What is a Random Number?

Random number does NOT mean a different number every time. Random means something that can not be predicted logically.

Pseudo Random and True Random.

Computers work on programs, and programs are definitive set of instructions. So it means there must be some algorithm to generate a random number as well.

If there is a program to generate random number it can be predicted, thus it is not truly random.

Random numbers generated through a generation algorithm are called *pseudo random*.

Can we make truly random numbers?

Yes. In order to generate a truly random number on our computers we need to get the random data from some outside source. This outside source is generally our keystrokes, mouse movements, data on network etc.

We do not need truly random numbers, unless its related to security (e.g. encryption keys) or the basis of application is the randomness (e.g. Digital roulette wheels).

In this tutorial we will be using pseudo random numbers.

Generate Random Number

see more please visit: https://homeofpdf.com

NumPy offers the random module to work with random numbers.

Example

Generate a random integer from 0 to 100:

```
from numpy import random

x = random.randint(100)

print(x)
```

Generate Random Float

The random module's rand() method returns a random float between 0 and 1.

Example

Generate a random float from 0 to 1:

```
from numpy import random

x = random.rand()

print(x)
```

Generate Random Array

In NumPy we work with arrays, and you can use the two methods from the above examples to make random arrays.

Integers

The randint() method takes a size parameter where you can specify the shape of an array.

Example

Generate a 1-D array containing 5 random integers from 0 to 100:

```
from numpy import random

x=random.randint(100, size=(5))

print(x)
```

Example

see more please visit: https://homeofpdf.com

Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:

```
from numpy import random

x = random.randint(100, size=(3, 5))

print(x)
```

Floats

The rand() method also allows you to specify the shape of the array.

Example

Generate a 1-D array containing 5 random floats:

```
from numpy import random

x = random.rand(5)

print(x)
```

Example

Generate a 2-D array with 3 rows, each row containing 5 random numbers:

```
from numpy import random

x = random.rand(3, 5)

print(x)
```

Generate Random Number From Array

The choice() method allows you to generate a random value based on an array of values.

The choice() method takes an array as a parameter and randomly returns one of the values.

Example

Return one of the values in an array:

```
from numpy import random
```

see more please visit: https://homeofpdf.com

```
x = random.choice([3, 5, 7, 9])

print(x)
```

The choice() method also allows you to return an *array* of values.

Add a size parameter to specify the shape of the array.

Example

Generate a 2-D array that consists of the values in the array parameter (3, 5, 7, and 9):

```
from numpy import random

x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

Random Data Distribution

What is Data Distribution?

Data Distribution is a list of all possible values, and how often each value occurs.

Such lists are important when working with statistics and data science.

The random module offers methods that return randomly generated data distributions.

Random Distribution

A random distribution is a set of random numbers that follow a certain *probability density function*.

Probability Density Function: A function that describes a continuous probability. i.e. probability of all values in an array.

We can generate random numbers based on defined probabilities using the choice() method of the random module.

The choice() method allows us to specify the probability for each value.

see more please visit: https://homeofpdf.com

The probability is set by a number between 0 and 1, where 0 means that the value will never occur and 1 means that the value will always occur.

Example

Generate a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9.

The probability for the value to be 3 is set to be 0.1

The probability for the value to be 5 is set to be 0.3

The probability for the value to be 7 is set to be 0.6

The probability for the value to be 9 is set to be 0

```
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(100))

print(x)
```

The sum of all probability numbers should be 1.

Even if you run the example above 100 times, the value 9 will never occur.

You can return arrays of any shape and size by specifying the shape in the size parameter.

Example

Same example as above, but return a 2-D array with 3 rows, each containing 5 values.

```
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(3, 5))

print(x)
```

Random Permutations

Random Permutations of Elements

A permutation refers to an arrangement of elements. e.g. [3, 2, 1] is a

see more please visit: https://homeofpdf.com

permutation of [1, 2, 3] and vice-versa.

The NumPy Random module provides two methods for this: shuffle() and permutation().

Shuffling Arrays

Shuffle means changing the arrangement of elements in-place. i.e. in the array itself.

Example

Randomly shuffle elements of following array:

```
from numpy import random

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

random.shuffle(arr)

print(arr)
```

The shuffle() method makes changes to the original array.

Generating Permutation of Arrays

Example

Generate a random permutation of elements of following array:

```
from numpy import random

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(random.permutation(arr))
```

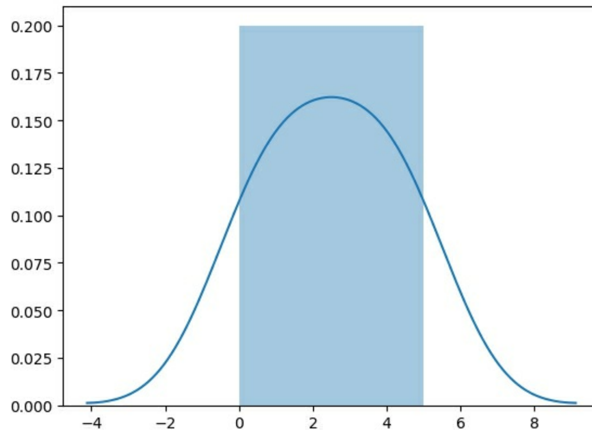
The permutation() method *returns* a re-arranged array (and leaves the original array un-changed).

Visualize Distributions With Seaborn

Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be

see more please visit: https://homeofpdf.com

used to visualize random distributions.



Install Seaborn.

If you have Python and PIP already installed on a system, install it using this command:

C:\Users\Your Name>pip install seaborn

If you use Jupyter, install Seaborn using this command:

C:\Users\Your Name>!pip install seaborn

Distplots

Distplot stands for distribution plot, it takes as input an array and plots a curve corresponding to the distribution of points in the array.

Import Matplotlib

Import the pyplot object of the Matplotlib module in your code using the following statement:

import matplotlib.pyplot as plt

Import Seaborn

Import the Seaborn module in your code using the following statement:

see more please visit: https://homeofpdf.com

```
import seaborn as sns
```

Plotting a Displot

Example

```
import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5])

plt.show()
```

Plotting a Distplot Without the Histogram

Example

```
import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5], hist=False)

plt.show()
```

Note: We will be using: sns.distplot(arr, hist=False) to visualize random distributions in this tutorial.

Normal (Gaussian) Distribution

Normal Distribution

The Normal Distribution is one of the most important distributions.

It is also called the Gaussian Distribution after the German mathematician Carl Friedrich Gauss.

It fits the probability distribution of many events, eg. IQ Scores, Heartbeat etc.

Use the random.normal() method to get a Normal Data Distribution.

It has three parameters:

see more please visit: https://homeofpdf.com

loc - (Mean) where the peak of the bell exists.

scale - (Standard Deviation) how flat the graph distribution should be.

size - The shape of the returned array.

Example

Generate a random normal distribution of size 2x3:

```
from numpy import random
x = random.normal(size=(2, 3))
print(x)
```

Generate a random normal distribution of size 2x3 with mean at 1 and standard deviation of 2:

```
from numpy import random
x = random.normal(loc=1, scale=2, size=(2, 3))
print(x)
```

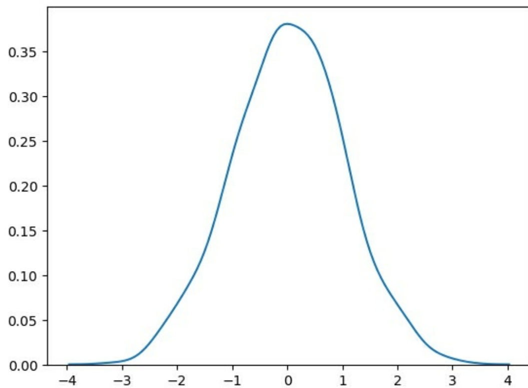
Visualization of Normal Distribution

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.normal(size=1000), hist=False)
plt.show()
```

Result

see more please visit: https://homeofpdf.com



Note: The curve of a Normal Distribution is also known as the Bell Curve because of the bell-shaped curve.

Binomial Distribution

Binomial Distribution

Binomial Distribution is a *Discrete Distribution*.

It describes the outcome of binary scenarios, e.g. toss of a coin, it will either be head or tails.

It has three parameters:

n - number of trials.

p - probability of occurence of each trial (e.g. for toss of a coin 0.5 each).

size - The shape of the returned array.

Discrete Distribution: The distribution is defined at separate set of events, e.g. a coin toss's result is discrete as it can be only head or tails whereas height of people is continuous as it can be 170, 170.1, 170.11 and so on.

Example

see more please visit: https://homeofpdf.com

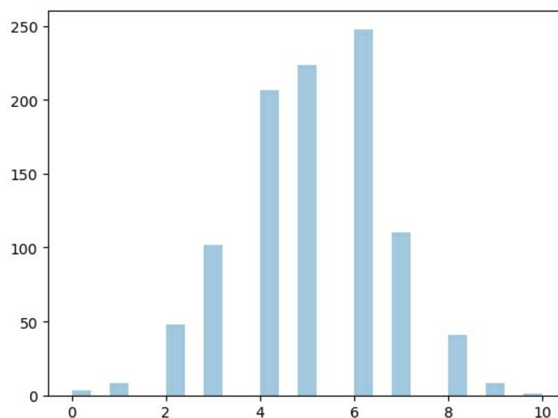
Given 10 trials for coin toss generate 10 data points:

```
from numpy import random
x = random.binomial(n=10, p=0.5, size=10)
print(x)
```

Visualization of Binomial Distribution

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.binomial(n=10, p=0.5, size=1000), hist=True, kde=False)
plt.show()
```



Result

Difference Between Normal and Binomial Distribution

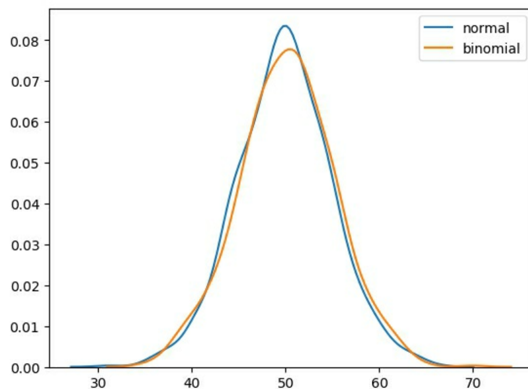
The main difference is that normal distribution is continuous whereas binomial is discrete, but if there are enough data points it will be quite similar to normal distribution with certain loc and scale.

see more please visit: https://homeofpdf.com

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False,
label='normal')

sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False,
label='binomial')

plt.show()
```

Result



Poisson Distribution

Poisson Distribution

see more please visit: https://homeofpdf.com

Poisson Distribution is a *Discrete Distribution*.

It estimates how many times an event can happen in a specified time. e.g. If someone eats twice a day what is the probability he will eat thrice?

It has two parameters:

lam - rate or known number of occurrences e.g. 2 for above problem.

size - The shape of the returned array.

Example

Generate a random 1x10 distribution for occurence 2:

```
from numpy import random

x = random.poisson(lam=2, size=10)

print(x)
```

Visualization of Poisson Distribution

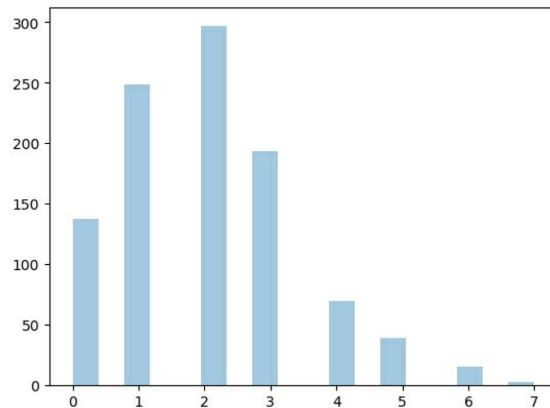
Example

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.poisson(lam=2, size=1000), kde=False)

plt.show()
```

Result

Difference Between Normal and Poisson

see more please visit: https://homeofpdf.com



Distribution

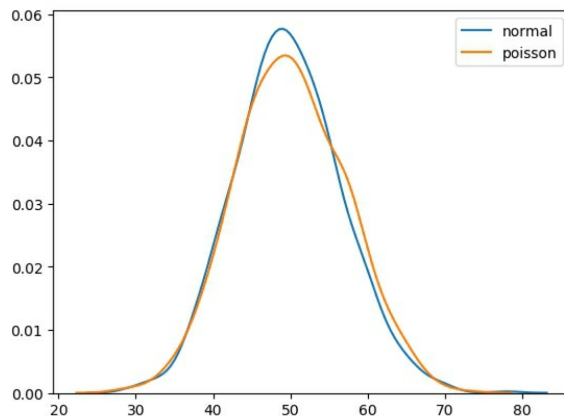
Normal distribution is continuous whereas poisson is discrete.

But we can see that similar to binomial for a large enough poisson distribution it will become similar to normal distribution with certain std dev and mean.

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(loc=50, scale=7, size=1000), hist=False, label='normal')
sns.distplot(random.poisson(lam=50, size=1000), hist=False, label='poisson')

plt.show()
```

see more please visit: https://homeofpdf.com



Result

Difference Between Poisson and Binomial Distribution

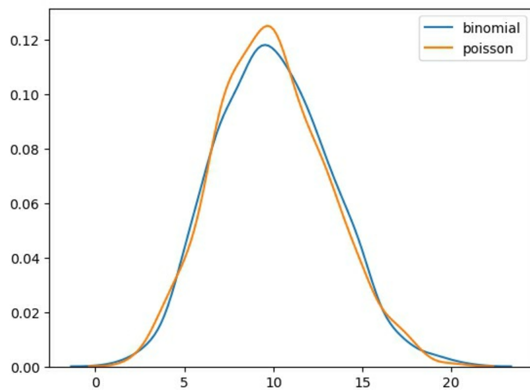
The difference is very subtle: binomial distribution is for discrete trials, whereas poisson distribution is for continuous trials.

But for very large n and near-zero p binomial distribution is near identical to poisson distribution such that n * p is nearly equal to lam.

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.binomial(n=1000, p=0.01, size=1000), hist=False, label='binomial')
sns.distplot(random.poisson(lam=10, size=1000), hist=False, label='poisson')

plt.show()
```

Result



Uniform Distribution

Uniform Distribution

Used to describe probability where every event has equal chances of occuring.

E.g. Generation of random numbers.

It has three parameters:

a - lower bound - default 0 .0.

b - upper bound - default 1.0.

size - The shape of the returned array.

Example

Create a 2x3 uniform distribution sample:

```
from numpy import random

x = random.uniform(size=(2, 3))

print(x)
```

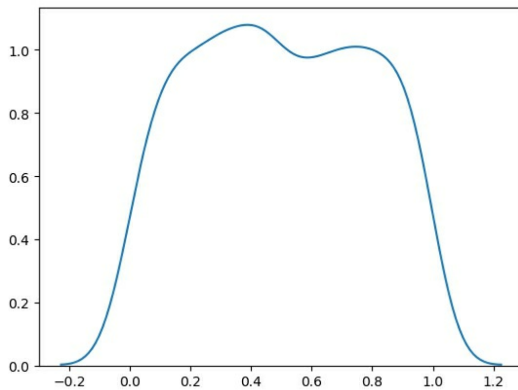
Visualization of Uniform Distribution

Example

see more please visit: https://homeofpdf.com

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.uniform(size=1000), hist=False)

plt.show()
```

Result



Logistic Distribution

Logistic Distribution

Logistic Distribution is used to describe growth.

Used extensively in machine learning in logistic regression, neural networks etc.

It has three parameters:

loc - mean, where the peak is. Default 0.

scale - standard deviation, the flatness of distribution. Default 1.

size - The shape of the returned array.

see more please visit: https://homeofpdf.com

Example

Draw 2x3 samples from a logistic distribution with mean at 1 and stddev 2.0:

```
from numpy import random
x = random.logistic(loc=1, scale=2, size=(2, 3))
print(x)
```

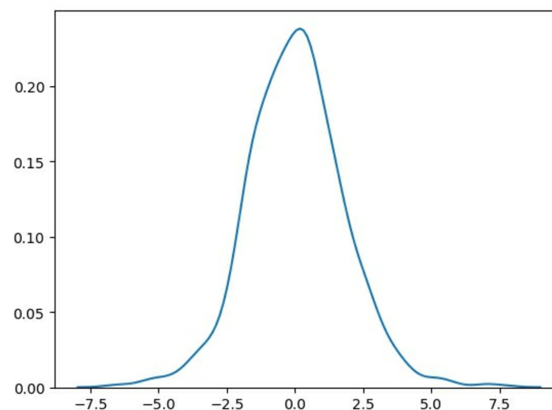
Visualization of Logistic Distribution

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.logistic(size=1000), hist=False)
plt.show()
```

Result

Difference Between Logistic and Normal



Distribution

Both distributions are nearly identical, but logistic distribution has more area under the tails. ie. It represented more possibility of occurrence of an event further away from mean.

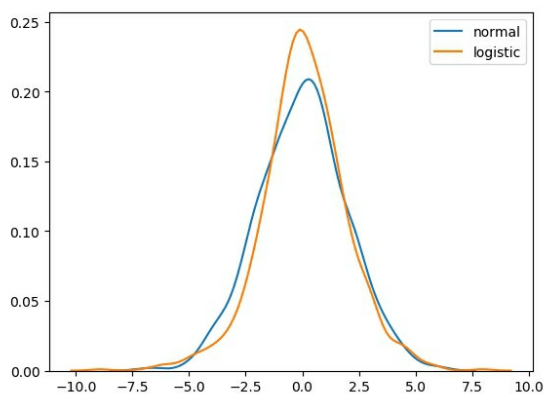
see more please visit: https://homeofpdf.com

For higher values of scale (standard deviation) the normal and logistic distributions are nearly identical apart from the peak.

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(scale=2, size=1000), hist=False, label='normal')
sns.distplot(random.logistic(size=1000), hist=False, label='logistic')

plt.show()
```

Result



Multinomial Distribution

Multinomial Distribution

see more please visit: https://homeofpdf.com

Multinomial distribution is a generalization of binomial distribution.

It describes outcomes of multinomial scenarios unlike binomial where scenarios must be only one of two. e.g. Blood type of a population, dice roll outcome.

It has three parameters:

n - number of possible outcomes (e.g. 6 for dice roll).

pvals - list of probabilities of outcomes (e.g. [1/6, 1/6, 1/6, 1/6, 1/6, 1/6] for dice roll).

size - The shape of the returned array.

Example

Draw out a sample for dice roll:

```
from numpy import random

x = random.multinomial(n=6, pvals=[1/6, 1/6, 1/6, 1/6, 1/6, 1/6])

print(x)
```

Note: Multinomial samples will NOT produce a single value! They will produce one value for each pval.

Note: As they are generalizations of binomial distribution their visual representation and similarity of normal distribution is same as that of multiple binomial distributions.

Exponential Distribution

Exponential Distribution

Exponential distribution is used for describing time till the next event e.g. failure/success etc.

It has two parameters:

scale - inverse of rate (see lam in poisson distribution) defaults to 1.0.

size - The shape of the returned array.

see more please visit: https://homeofpdf.com

Example

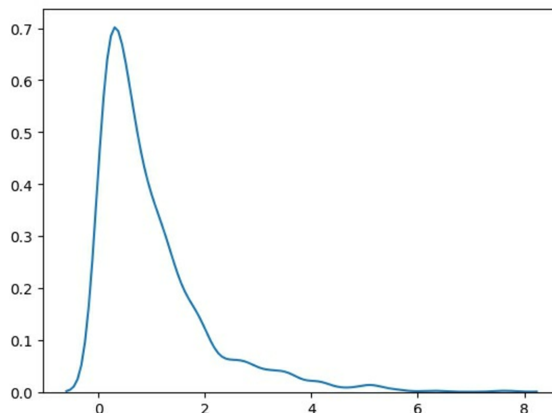
Draw out a sample for exponential distribution with 2.0 scale with 2x3 size:

```
from numpy import random

x = random.exponential(scale=2, size=(2, 3))
print(x)
```

Visualization of Exponential Distribution

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.exponential(size=1000), hist=False)
plt.show()
```



Result

Relation Between Poisson and Exponential Distribution

Poisson distribution deals with the number of occurrences of an event in a time period whereas exponential distribution deals with the time between

see more please visit: https://homeofpdf.com

these events.

Chi Square Distribution

Chi Square Distribution

Chi Square distribution is used as a basis to verify the hypothesis.

It has two parameters:

df - (degree of freedom).

size - The shape of the returned array.

Example

Draw out a sample for chi squared distribution with degree of freedom 2 with size 2x3:

```
from numpy import random

x = random.chisquare(df=2, size=(2, 3))

print(x)
```

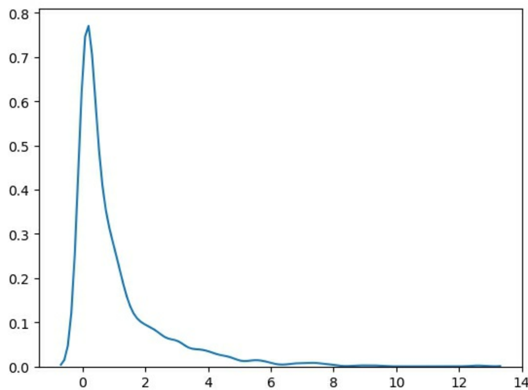
Visualization of Chi Square Distribution

Example

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.chisquare(df=1, size=1000), hist=False)

plt.show()
```

Result

see more please visit: https://homeofpdf.com



Rayleigh Distribution

Rayleigh Distribution

Rayleigh distribution is used in signal processing.

It has two parameters:

scale - (standard deviation) decides how flat the distribution will be default 1.0).

size - The shape of the returned array.

Example

Draw out a sample for rayleigh distribution with scale of 2 with size 2x3:

```
from numpy import random

x = random.rayleigh(scale=2, size=(2, 3))

print(x)
```

Visualization of Rayleigh Distribution

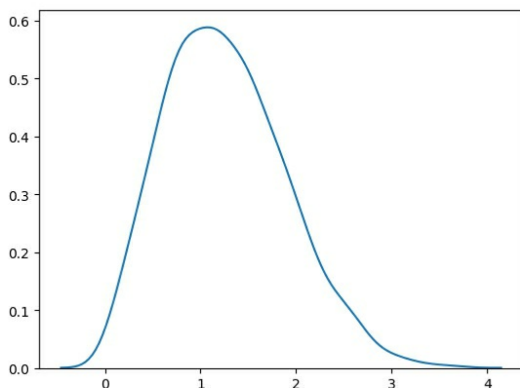
Example

```
from numpy import random

import matplotlib.pyplot as plt
```

see more please visit: https://homeofpdf.com

```
import seaborn as sns

sns.distplot(random.rayleigh(size=1000), hist=False)

plt.show()
```

Result



Similarity Between Rayleigh and Chi Square Distribution

At unit stddev the and 2 degrees of freedom rayleigh and chi square represent the same distributions.

Pareto Distribution

Pareto Distribution

A distribution following Pareto's law i.e. 80-20 distribution (20% factors cause 80% outcome).

It has two parameter:

see more please visit: https://homeofpdf.com

a - shape parameter.

size - The shape of the returned array.

Example

Draw out a sample for pareto distribution with shape of 2 with size 2x3:

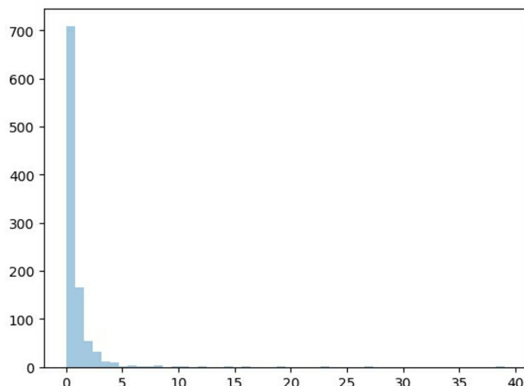
```
from numpy import random

x = random.pareto(a=2, size=(2, 3))

print(x)
```

Visualization of Pareto Distribution

Example

```
from numpy import random

import matplotlib.pyplot as plt

import seaborn as sns


sns.distplot(random.pareto(a=2, size=1000), kde=False)


plt.show()
```

Result



see more please visit: https://homeofpdf.com

Zipf Distribution

Zipf distributions are used to sample data based on zipf's law.

Zipf's Law: In a collection the nth common term is 1/n times of the most common term. E.g. the 5th common word in english has occurs nearly 1/5th times as of the most used word.

It has two parameters:

a - distribution parameter.

size - The shape of the returned array.

Example

Draw out a sample for zipf distribution with distribution parameter 2 with size 2x3:

```
from numpy import random

x = random.zipf(a=2, size=(2, 3))

print(x)
```

Visualization of Zipf Distribution

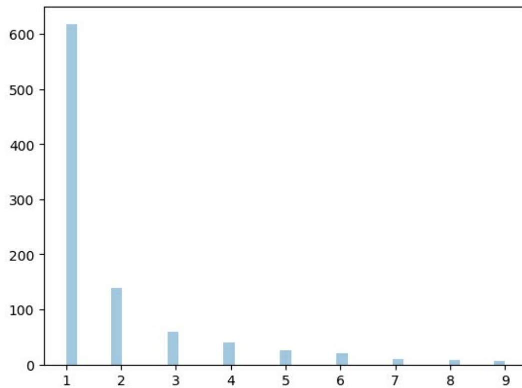
Sample 1000 points but plotting only ones with value < 10 for more meaningful charts.

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

x = random.zipf(a=2, size=1000)
sns.distplot(x[x<10], kde=False)

plt.show()
```

Result

see more please visit: https://homeofpdf.com



NumPy ufuncs

What are ufuncs?

ufuncs stands for "Universal Functions" and they are NumPy functions that operate on the ndarray object.

Why use ufuncs?

ufuncs are used to implement *vectorization* in NumPy which is way faster than iterating over elements.

They also provide broadcasting and additional methods like reduce, accumulate etc. that are very helpful for computation.

ufuncs also take additional arguments, like:

where boolean array or condition defining where the operations should take place.

dtype defining the return type of elements.

out output array where the return value should be copied.

What is Vectorization?

Converting iterative statements into a vector based operation is called

see more please visit: https://homeofpdf.com

vectorization.

It is faster as modern CPUs are optimized for such operations.

Add the Elements of Two Lists

list 1: [1, 2, 3, 4]

list 2: [4, 5, 6, 7]

One way of doing it is to iterate over both of the lists and then sum each element.

Example

Without ufunc, we can use Python's built-in zip() method:

```
x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = []

for i, j in zip(x, y):
  z.append(i + j)
print(z)
```

NumPy has a ufunc for this, called add(x, y) that will produce the same result.

Example

With ufunc, we can use the add() function:

```
import numpy as np

x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = np.add(x, y)

print(z)
```

Create Your Own ufunc

see more please visit: https://homeofpdf.com

How To Create Your Own ufunc

To create your own ufunc, you have to define a function, like you do with normal functions in Python, then you add it to your NumPy ufunc library with the frompyfunc() method.

The frompyfunc() method takes the following arguments:

1. *function* - the name of the function.
2. *inputs* - the number of input arguments (arrays).
3. *outputs* - the number of output arrays.

Example

Create your own ufunc for addition:

```
import numpy as np

def myadd(x, y):
  return x+y

myadd = np.frompyfunc(myadd, 2, 1)

print(myadd([1, 2, 3, 4], [5, 6, 7, 8]))
```

Check if a Function is a ufunc

Check the *type* of a function to check if it is a ufunc or not.

A ufunc should return <class 'numpy.ufunc'>.

Example

Check if a function is a ufunc:

```
import numpy as np

print(type(np.add))
```

If it is not a ufunc, it will return another type, like this built-in NumPy function for joining two or more arrays:

Example

see more please visit: https://homeofpdf.com

Check the type of another function: concatenate():

```
import numpy as np

print(type(np.concatenate))
```

If the function is not recognized at all, it will return an error:

Example

Check the type of something that does not exist. This will produce an error:

```
import numpy as np

print(type(np.blahblah))
```

To test if the function is a ufunc in an if statement, use the numpy.ufunc value (or np.ufunc if you use np as an alias for numpy):

Example

Use an if statement to check if the function is a ufunc or not:

```
import numpy as np

if type(np.add) == np.ufunc:
  print('add is ufunc')
else:
  print('add is not ufunc')
```

Simple Arithmetic

You could use arithmetic operators + - * / directly between NumPy arrays, but this section discusses an extension of the same where we have functions that can take any array-like objects e.g. lists, tuples etc. and perform arithmetic *conditionally*.

Arithmetic Conditionally: means that we can define conditions where the arithmetic operation should happen.

All of the discussed arithmetic functions take a where parameter in which we can specify that condition.

see more please visit: https://homeofpdf.com

Addition

The add() function sums the content of two arrays, and return the results in a new array.

Example

Add the values in arr1 to the values in arr2:

```
import numpy as np

arr1 = np.array([10, 11, 12, 13, 14, 15])
arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = np.add(arr1, arr2)

print(newarr)
```

The example above will return [30 32 34 36 38 40] which is the sums of 10+20, 11+21, 12+22 etc.

Subtraction

The subtract() function subtracts the values from one array with the values from another array, and returns the results in a new array.

Example

Subtract the values in arr2 from the values in arr1:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = np.subtract(arr1, arr2)

print(newarr)
```

The example above will return [-10 -1 8 17 26 35] which is the result of 10-20, 20-21, 30-22 etc.

Multiplication

see more please visit: https://homeofpdf.com

The multiply() function multiplies the values from one array with the values from another array, and returns the results in a new array.

Example

Multiply the values in arr1 with the values in arr2:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = np.multiply(arr1, arr2)

print(newarr)
```

The example above will return [200 420 660 920 1200 1500] which is the result of 10*20, 20*21, 30*22 etc.

Division

The divide() function divides the values from one array with the values from another array, and returns the results in a new array.

Example

Divide the values in arr1 with the values in arr2:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 5, 10, 8, 2, 33])

newarr = np.divide(arr1, arr2)

print(newarr)
```

The example above will return [3.33333333 4. 3. 5. 25. 1.81818182] which is the result of 10/3, 20/5, 30/10 etc.

Power

The power() function raises the values from the first array to the power of the

see more please visit: https://homeofpdf.com

values of the second array, and returns the results in a new array.

Example

Raise the values in arr1 to the power of values in arr2:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 5, 6, 8, 2, 33])

newarr = np.power(arr1, arr2)

print(newarr)
```

The example above will return [1000 3200000 729000000 6553600000000 2500 0] which is the result of 10*10*10, 20*20*20*20*20, 30*30*30*30*30*30 etc.

Remainder

Both the mod() and the remainder() functions return the remainder of the values in the first array corresponding to the values in the second array, and return the results in a new array.

Example

Return the remainders:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 7, 9, 8, 2, 33])

newarr = np.mod(arr1, arr2)

print(newarr)
```

The example above will return [1 6 3 0 0 27] which is the remainder when you divide 10 with 3 (10%3), 20 with 7 (20%7) 30 with 9 (30%9) etc.

You get the same result when using the remainder() function:

Example

see more please visit: https://homeofpdf.com

Return the remainders:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 7, 9, 8, 2, 33])

newarr = np.remainder(arr1, arr2)

print(newarr)
```

Quotient and Mod

The divmod() function returns both the quotient and the mod. The return value is two arrays, the first array contains the quotient and the second array contains the mod.

Example

Return the quotient and mod:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 7, 9, 8, 2, 33])

newarr = np.divmod(arr1, arr2)

print(newarr)
```

The example above will return:

(array([3, 2, 3, 5, 25, 1]), array([1, 6, 3, 0, 0, 27]))

The first array represents the quotients, (the integer value when you divide 10 with 3, 20 with 7, 30 with 9 etc.

The second array represents the remainder of the same divisions.

Absolute Values

Both the absolute() and the abs() functions functions do the same absolute operation element-wise but we should use absolute() to avoid confusion with python's inbuilt math.abs()

see more please visit: https://homeofpdf.com

Example

Return the quotient and mod:

```
import numpy as np

arr = np.array([-1, -2, 1, 2, 3, -4])

newarr = np.absolute(arr)

print(newarr)
```

The example above will return [1 2 1 2 3 4].

Rounding Decimals

There are primarily five ways of rounding off decimals in NumPy:

- truncation
- fix
- rounding
- floor
- ceil

Truncation

Remove the decimals, and return the float number closest to zero. Use the trunc() and fix() functions.

Example

Truncate elements of following array:

```
import numpy as np

arr = np.trunc([-3.1666, 3.6667])

print(arr)
```

Example

Same example, using fix():

```
import numpy as np
```

see more please visit: https://homeofpdf.com

```
arr = np.fix([-3.1666, 3.6667])

print(arr)
```

Rounding

The around() function increments preceding digit or decimal by 1 if >=5 else do nothing.

E.g. round off to 1 decimal point, 3.16666 is 3.2

Example

Round off 3.1666 to 2 decimal places:

```
import numpy as np

arr = np.around(3.1666, 2)

print(arr)
```

Floor

The floor() function rounds off decimal to the nearest lower integer.

E.g. floor of 3.166 is 3.

Example

Floor the elements of following array:

```
import numpy as np

arr = np.floor([-3.1666, 3.6667])

print(arr)
```

Note: The floor() function returns floats, unlike the trunc() function who returns integers.

Ceil

The ceil() function rounds off decimal to the nearest upper integer.

E.g. The level of 3.166 is 4.

see more please visit: https://homeofpdf.com

Example

Ceil the elements of following array:

```
import numpy as np

arr = np.ceil([-3.1666, 3.6667])

print(arr)
```

NumPy Logs

Logs

NumPy provides functions to perform log at the base 2, e and 10.

We will also explore how we can take log for any base by creating a custom ufunc.

All of the log functions will place -inf or inf in the elements if the log can not be computed.

Log at Base 2

Use the log2() function to perform log at the base 2.

Example

Find log at base 2 of all elements of following array:

```
import numpy as np

arr = np.arange(1, 10)

print(np.log2(arr))
```

Note: The range(1, 10) function returns an array with integers starting from 1 (included) to 10 (not included).

Log at Base 10

Use the log10() function to perform log at the base 10.

see more please visit: https://homeofpdf.com

Example

Find log at base 10 of all elements of following array:

```
import numpy as np

arr = np.arange(1, 10)

print(np.log10(arr))
```

Natural Log, or Log at Base e

Use the log() function to perform log at the base e.

Example

Find log at base e of all elements of following array:

```
import numpy as np

arr = np.arange(1, 10)

print(np.log(arr))
```

Log at Any Base

NumPy does not provide any function to take log at any base, so we can use the frompyfunc() function along with inbuilt function math.log() with two input parameters and one output parameter:

Example

```
from math import log
import numpy as np

nplog = np.frompyfunc(log, 2, 1)

print(nplog(100, 15))
```

NumPy Summations

see more please visit: https://homeofpdf.com

Summations

What is the difference between summation and addition?

Addition is done between two arguments whereas summation happens over n elements.

Example

Add the values in arr1 to the values in arr2:

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])

newarr = np.add(arr1, arr2)

print(newarr)
```

Returns: [2 4 6]

Example

Sum the values in arr1 and the values in arr2:

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])

newarr = np.sum([arr1, arr2])

print(newarr)
```

Returns: 12

Summation Over an Axis

If you specify axis=1, NumPy will sum the numbers in each array.

Example

Perform summation in the following array over 1st axis:

see more please visit: https://homeofpdf.com

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])

newarr = np.sum([arr1, arr2], axis=1)

print(newarr)
```

Returns: [6 6]

Cumulative Sum

Cumulative sum means partially adding the elements in an array.

E.g. The partial sum of [1, 2, 3, 4] would be [1, 1+2, 1+2+3, 1+2+3+4] = [1, 3, 6, 10].

Perform partial sum with the cumsum() function.

Example

Perform cumulative summation in the following array:

```
import numpy as np

arr = np.array([1, 2, 3])

newarr = np.cumsum(arr)

print(newarr)
```

Returns: [1 3 6]

NumPy Products

Products

To find the product of the elements in an array, use the prod() function.

Example

Find the product of the elements of this array:

see more please visit: https://homeofpdf.com

```
import numpy as np
arr = np.array([1, 2, 3, 4])
x = np.prod(arr)
print(x)
```

Returns: 24 because 1*2*3*4 = 24

Example

Find the product of the elements of two arrays:

```
import numpy as np
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])
x = np.prod([arr1, arr2])
print(x)
```

Returns: 40320 because 1*2*3*4*5*6*7*8 = 40320

Product Over an Axis

If you specify axis=1, NumPy will return the product of each array.

Example

Perform summation in the following array over 1st axis:

```
import numpy as np
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])
newarr = np.prod([arr1, arr2], axis=1)
print(newarr)
```

Returns: [24 1680]

Cumulative Product

see more please visit: https://homeofpdf.com

Cumulative product means taking the product partially.

E.g. The partial product of [1, 2, 3, 4] is [1, 1*2, 1*2*3, 1*2*3*4] = [1, 2, 6, 24]

Perform partial sum with the cumprod() function.

Example

Take cumulative product of all elements for following array:

```
import numpy as np

arr = np.array([5, 6, 7, 8])

newarr = np.cumprod(arr)

print(newarr)
```

Returns: [5 30 210 1680]

NumPy Differences

Differences

A discrete difference means subtracting two successive elements.

E.g. for [1, 2, 3, 4], the discrete difference would be [2-1, 3-2, 4-3] = [1, 1, 1]

To find the discrete difference, use the diff() function.

Example

Compute discrete difference of the following array:

```
import numpy as np

arr = np.array([10, 15, 25, 5])

newarr = np.diff(arr)

print(newarr)
```

Returns: [5 10 -20] because 15-10=5, 25-15=10, and 5-25=-20

We can perform this operation repeatedly by giving parameter n.

see more please visit: https://homeofpdf.com

E.g. for [1, 2, 3, 4], the discrete difference with n = 2 would be [2-1, 3-2, 4-3] = [1, 1, 1] , then, since n=2, we will do it once more, with the new result: [1-1, 1-1] = [0, 0]

Example

Compute discrete difference of the following array twice:

```
import numpy as np

arr = np.array([10, 15, 25, 5])

newarr = np.diff(arr, n=2)

print(newarr)
```

Returns: [5 -30] because: 15-10=5, 25-15=10, and 5-25=-20 AND 10-5=5 and -20-10=-30

NumPy LCM Lowest Common Multiple

Finding LCM (Lowest Common Multiple)

The Lowest Common Multiple is the least common multiple of both of the numbers.

Example

Find the LCM of the following two numbers:

```
import numpy as np

num1 = 4

num2 = 6

x = np.lcm(num1, num2)

print(x)
```

Returns: 12 because that is the lowest common multiple of both numbers (4*3=12 and 6*2=12).

see more please visit: https://homeofpdf.com

Finding LCM in Arrays

To find the Lowest Common Multiple of all values in an array, you can use the reduce() method.

The reduce() method will use the ufunc, in this case the lcm() function, on each element, and reduce the array by one dimension.

Example

Find the LCM of the values of the following array:

```
import numpy as np

arr = np.array([3, 6, 9])

x = np.lcm.reduce(arr)

print(x)
```

Returns: 18 because that is the lowest common multiple of all three numbers (3*6=18, 6*3=18 and 9*2=18).

Example

Find the LCM of all of an array where the array contains all integers from 1 to 10:

```
import numpy as np

arr = np.arange(1, 11)

x = np.lcm.reduce(arr)

print(x)
```

NumPy GCD Greatest Common Denominator

Finding GCD (Greatest Common Denominator)

The GCD (Greatest Common Denominator), also known as HCF (Highest Common Factor) is the biggest number that is a common factor of both of the

see more please visit: https://homeofpdf.com

numbers.

Example

Find the HCF of the following two numbers:

```
import numpy as np

num1 = 6

num2 = 9

x = np.gcd(num1, num2)

print(x)
```

Returns: 3 because that is the highest number both numbers can be divided by (6/3=2 and 9/3=3).

Finding GCD in Arrays

To find the Highest Common Factor of all values in an array, you can use the reduce() method.

The reduce() method will use the ufunc, in this case the gcd() function, on each element, and reduce the array by one dimension.

Example

Find the GCD for all of the numbers in following array:

```
import numpy as np

arr = np.array([20, 8, 32, 36, 16])

x = np.gcd.reduce(arr)

print(x)
```

Returns: 4 because that is the highest number all values can be divided by.

NumPy Trigonometric Functions

Trigonometric Functions

see more please visit: https://homeofpdf.com

NumPy provides the ufuncs sin(), cos() and tan() that take values in radians and produce the corresponding sin, cos and tan values.

Example

Find sine value of PI/2:

```
import numpy as np

x = np.sin(np.pi/2)

print(x)
```

Example

Find sine values for all of the values in arr:

```
import numpy as np

arr = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])

x = np.sin(arr)

print(x)
```

Convert Degrees Into Radians

By default all of the trigonometric functions take radians as parameters but we can convert radians to degrees and vice versa as well in NumP.

Note: radians values are pi/180 * degree_values.

Example

Convert all of the values in following array arr to radians:

```
import numpy as np

arr = np.array([90, 180, 270, 360])

x = np.deg2rad(arr)

print(x)
```

Radians to Degrees

Example

see more please visit: https://homeofpdf.com

Convert all of the values in following array arr to degrees:

```
import numpy as np

arr = np.array([np.pi/2, np.pi, 1.5*np.pi, 2*np.pi])

x = np.rad2deg(arr)

print(x)
```

Finding Angles

Finding angles from values of sine, cos, tan. E.g. sin, cos and tan inverse (arcsin, arccos, arctan).

NumPy provides ufuncs arcsin(), arccos() and arctan() that produce radian values for corresponding sin, cos and tan values given.

Example

Find the angle of 1.0:

```
import numpy as np

x = np.arcsin(1.0)

print(x)
```

Angles of Each Value in Arrays

Example

Find the angle for all of the sine values in the array

```
import numpy as np

arr = np.array([1, -1, 0.1])

x = np.arcsin(arr)

print(x)
```

Hypotenuse

Finding hypotenuse using pythagoras theorem in NumPy.

see more please visit: https://homeofpdf.com

NumPy provides the hypot() function that takes the base and perpendicular values and produces a hypotenuse based on pythagoras theorem.

Example

Find the hypotenuse for 4 base and 3 perpendicular:

```
import numpy as np

base = 3

perp = 4

x = np.hypot(base, perp)

print(x)
```

NumPy Hyperbolic Functions

NumPy provides the ufuncs sinh(), cosh() and tanh() that take values in radians and produce the corresponding sinh, cosh and tanh values..

Example

Find sinh value of PI/2:

```
import numpy as np

x = np.sinh(np.pi/2)

print(x)
```

Example

Find cosh values for all of the values in arr:

```
import numpy as np

arr = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])

x = np.cosh(arr)

print(x)
```

Finding Angles

see more please visit: https://homeofpdf.com

Finding angles from values of hyperbolic sine, cos, tan. E.g. sinh, cosh and tanh inverse (arcsinh, arccosh, arctanh).

Numpy provides ufuncs arcsinh(), arccosh() and arctanh() that produce radian values for corresponding sinh, cosh and tanh values given.

Example

Find the angle of 1.0:

```
import numpy as np

x = np.arcsinh(1.0)

print(x)
```

Angles of Each Value in Arrays

Example

Find the angle for all of the tanh values in array:

```
import numpy as np

arr = np.array([0.1, 0.2, 0.5])

x = np.arctanh(arr)

print(x)
```

NumPy Set Operations

What is a Set

A set in mathematics is a collection of unique elements.

Sets are used for operations involving frequent intersection, union and difference operations.

Create Sets in NumPy

We can use NumPy's unique() method to find unique elements from any array. E.g. create a set array, but remember that the set arrays should only be

see more please visit: https://homeofpdf.com

1-D arrays.

Example

Convert following array with repeated elements to a set:

```
import numpy as np
arr = np.array([1, 1, 1, 2, 3, 4, 5, 5, 6, 7])
x = np.unique(arr)
print(x)
```

Finding Union

To find the unique values of two arrays, use the union1d() method.

Example

Find union of the following two set arrays:

```
import numpy as np
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])
newarr = np.union1d(arr1, arr2)
print(newarr)
```

Finding Intersection

To find only the values that are present in both arrays, use the intersect1d() method.

Example

Find intersection of the following two set arrays:

```
import numpy as np
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])
```

see more please visit: https://homeofpdf.com

```
newarr = np.intersect1d(arr1, arr2, assume_unique=True)

print(newarr)
```

Note: the intersect1d() method takes an optional argument assume_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.

Finding Difference

To find only the values in the first set that is NOT present in the seconds set, use the setdiff1d() method.

Example

Find the difference of the set1 from set2:

```
import numpy as np

set1 = np.array([1, 2, 3, 4])

set2 = np.array([3, 4, 5, 6])

newarr = np.setdiff1d(set1, set2, assume_unique=True)

print(newarr)
```

Note: the setdiff1d() method takes an optional argument assume_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.

Finding Symmetric Difference

To find only the values that are NOT present in BOTH sets, use the setxor1d() method.

Example

Find the symmetric difference of the set1 and set2:

```
import numpy as np

set1 = np.array([1, 2, 3, 4])

set2 = np.array([3, 4, 5, 6])
```

see more please visit: https://homeofpdf.com

```
newarr = np.setxor1d(set1, set2, assume_unique=True)
print(newarr)
```

Note: the setxor1d() method takes an optional argument assume_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.

41.Pandas Tutorial

Pandas is a Python library.

Pandas is used to analyze data.

Learning by Reading

We have created 14 tutorial pages for you to learn more about Pandas.

Starting with a basic introduction and ends up with cleaning and plotting data:

Basic

- Introduction
- Getting Started
- Pandas Series
- DataFrames
- Read CSV
- Read JSON
- Analyze Data

Cleaning Data

see more please visit: https://homeofpdf.com

- Clean Data
- Clean Empty Cells
- Clean Wrong Format
- Clean Wrong Data
- Remove Duplicates

Advanced

- Correlations
- Plotting

Pandas Introduction

What is Pandas?

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is the average value?
- Max value?

see more please visit: https://homeofpdf.com

- Min value?

Pandas are also able to delete rows that are not relevant, or contain wrong values, like empty or NULL values. This is called *cleaning* the data.

Where is the Pandas Codebase?

The source code for Pandas is located at this github repository https://github.com/pandas-dev/pandas

github: enables many people to work on the same codebase.

Pandas Getting Started

Installation of Pandas

If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

Install it using this command:

C:\Users\Your Name>pip install pandas

If this command fails, then use a python distribution that already has Pandas installed like Anaconda, Spyder etc.

Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

```
import pandas
```

Now Pandas are imported and ready to use.

Example

```
import pandas

mydataset = {
```

see more please visit: https://homeofpdf.com

```
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}
myvar = pandas.DataFrame(mydataset)
print(myvar)
```

Pandas as pd

Pandas is usually imported under the pd alias.

alias: In Python aliases are an alternate name for referring to the same thing.

Create an alias with the as keyword while importing:

```
import pandas as pd
```

Now the Pandas package can be referred to as pd instead of pandas.

Example

```
import pandas as pd
mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}
myvar = pd.DataFrame(mydataset)
print(myvar)
```

Checking Pandas Version

The version string is stored under __version__ attribute.

Example

```
import pandas as pd

print(pd.__version__)
```

Pandas Series

What is a Series?

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

Example

Create a simple Pandas Series from a list:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value.

Example

Return the first value of the Series:

```
print(myvar[0])
```

Create Labels

With the index argument, you can name your own labels.

Example

see more please visit: https://homeofpdf.com

Create your own labels:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

When you have created labels, you can access an item by referring to the label.

Example

Return the value of "y":

```
print(myvar["y"])
```

Key/Value Objects as Series

You can also use a key/value object, like a dictionary, when creating a Series.

Example

Create a simple Pandas Series from a dictionary:

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

Note: The keys of the dictionary become the labels.

To select only some of the items in the dictionary, use the index argument and specify only the items you want to include in the Series.

Example

Create a Series using only data from "day1" and "day2":

see more please visit: https://homeofpdf.com

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```

DataFrames

Datasets in Pandas are usually multi-dimensional tables, called DataFrames.

Series is like a column, a DataFrame is the whole table.

Example

Create a DataFrame from two Series:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

myvar = pd.DataFrame(data)

print(myvar)
```

Pandas DataFrames

What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

see more please visit: https://homeofpdf.com

Example

Create a simple Pandas DataFrame:

```
import pandas as pd

data = {

  "calories": [420, 380, 390],

  "duration": [50, 40, 45]

}

#load data into a DataFrame object:

df = pd.DataFrame(data)

print(df)
```

Result

calories   420
duration 50
Name: 0, dtype: int64

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

Example

Return row 0:

```
#refer to the row index:

print(df.loc[0])
```

Result

see more please visit: https://homeofpdf.com

	calories	duration
0	420	50
1	380	40

Note: This example returns a Pandas **Series**.

Example

Return row 0 and 1:

```
#use a list of indexes:

print(df.loc[[0, 1]])
```

Result

	calories	duration
day1	420	50
day2	380	40
day3	390	45

Note: When using [], the result is a Pandas **DataFrame**.

Locate Named Indexes

Use the named index in the loc attribute to return the specified row(s).

Example

Return "day2":

```
#refer to the named index:

print(df.loc["day2"])
```

Result

calories   380

see more please visit: https://homeofpdf.com

duration 40

Name: 0, dtype: int64

Load Files Into a DataFrame

If your data sets are stored in a file, Pandas can load them into a DataFrame.

Example

Load a comma separated file (CSV file) into a DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df)
```

Pandas Read CSV

Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contain plain text and is a well known format that can be read by everyone including Pandas.

In our examples we will be using a CSV file called 'data.csv'.

Download data.csv. or **Open data.csv**

Example

Load the CSV into a DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')
print(df.to_string())
```

Tip: use to_string() to print the entire DataFrame.

see more please visit: https://homeofpdf.com

By default, when you print a DataFrame, you will only get the first 5 rows, and the last 5 rows:

Example

Print a reduced sample:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```

Pandas Read JSON

Read JSON

Big data sets are often stored, or extracted as JSON.

JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.

In our examples we will be using a JSON file called 'data.json'.

Open data.json.

Example

Load the JSON file into a DataFrame:

```
import pandas as pd

df = pd.read_json('data.json')

print(df.to_string())
```

Tip: use to_string() to print the entire DataFrame.

Dictionary as JSON

JSON = Python Dictionary

JSON objects have the same format as Python dictionaries.

If your JSON code is not in a file, but in a Python Dictionary, you can load it

see more please visit: https://homeofpdf.com

into a DataFrame directly:

Example

Load a Python Dictionary into a DataFrame:

```
import pandas as pd

data = {  "Duration":{"0":60,"1":60,"2":60, "3":45,"4":45,"5":60 },
  "Pulse":{"0":110,"1":117,"2":103,"3":109,"4":117,"5":102 },
  "Maxpulse":{"0":130,"1":145,"2":135,"3":175,"4":148,"5":127 },
  "Calories":{"0":409,"1":479,"2":340,"3":282,"4":406,"5":300}
}

df = pd.DataFrame(data)

print(df)
```

Pandas - Analyzing DataFrames

Viewing the Data

One of the most used methods for getting a quick overview of the DataFrame, is the head() method.

The head() method returns the headers and a specified number of rows, starting from the top.

Example

Get a quick overview by printing the first 10 rows of the DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head(10))
```

In our examples we will be using a CSV file called 'data.csv'.

Download data.csv, or open data.csv in your browser.

see more please visit: https://homeofpdf.com

Note: if the number of rows is not specified, the head() method will return the top 5 rows.

Example

Print the first 5 rows of the DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head())
```

There is also a tail() method for viewing the *last* rows of the DataFrame.

The tail() method returns the headers and a specified number of rows, starting from the bottom.

Example

Print the last 5 rows of the DataFrame:

```
print(df.tail())
```

Info About the Data

The DataFrames object has a method called info(), that gives you more information about the data set.

Example

Print information about the data:

```
print(df.info())
```

Result

```
<class 'pandas.core.frame.DataFrame'>
  RangeIndex: 169 entries, 0 to 168
  Data columns (total 4 columns):
   #   Column     Non-Null Count  Dtype
  ---  ------     --------------  -----
   0   Duration  169 non-nullint64
```

see more please visit: https://homeofpdf.com

1 Pulse 169 non-null int64
2 Maxpulse 169 non-null int64
3 Calories 164 non-null float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None

Result Explained

The result tells us there are 169 rows and 4 columns:

RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):

And the name of each column, with the data type:

```
 #   Column     Non-Null Count   Dtype
---  ------    --------------  -----
 0   Duration   169 non-null        int64
 1   Pulse      169 non-null     int64
 2   Maxpulse  169 non-null         int64
 3   Calories   164 non-null       float64
```

Null Values

The info() method also tells us how many Non-Null values there are present in each column, and in our data set it seems like there are 164 of 169 Non-Null values in the "Calories" column.

Which means that there are 5 rows with no value at all, in the "Calories" column, for whatever reason.

Empty values, or Null values, can be bad when analyzing data, and you should consider removing rows with empty values. This is a step towards what is called *cleaning data.*

Pandas - Cleaning Data

see more please visit: https://homeofpdf.com

Data cleaning means fixing bad data in your data set.

Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

In this tutorial you will learn how to deal with all of them.

Our Data Set

In the next chapters we will use this data set:

	Duration	Date	Pulse	Max Pulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	2020/12/26	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	

The data set contains some empty cells ("Date" in row 22, and "Calories" in row 18 and 28).

The data set contains the wrong format ("Date" in row 26).

see more please visit: https://homeofpdf.com

The data set contains wrong data ("Duration" in row 7).

The data set contains duplicates (row 11 and 12).

Pandas - Cleaning Empty Cells

Empty Cells

Empty cells can potentially give you a wrong result when you analyze data.

Remove Rows

One way to deal with empty cells is to remove rows that contain empty cells.

This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

Example

Return a new Dataframe with no empty cells:

```
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

In our cleaning examples we will be using a CSV file called 'dirtydata.csv'.

Download dirtydata.csv. or Open dirtydata.csv

Note: By default, the dropna() method returns a *new* DataFrame, and will not change the original.

If you want to change the original DataFrame, use the inplace = True argument:

Example

Remove all rows with NULL values:

```
import pandas as pd
```

see more please visit: https://homeofpdf.com

```
df = pd.read_csv('data.csv')

df.dropna(inplace = True)

print(df.to_string())
```

Note: Now, the dropna(inplace = True) will NOT return a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.

Replace Empty Values

Another way of dealing with empty cells is to insert a *new* value instead.

This way you do not have to delete entire rows just because of some empty cells.

The fillna() method allows us to replace empty cells with a value:

Example

Replace NULL values with the number 130:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(130, inplace = True)
```

Replace Only For a Specified Columns

The example above replaces all empty cells in the whole Data Frame.

To only replace empty values for one column, specify the *column name* for the DataFrame:

Example

Replace NULL values in the "Calories" columns with the number 130:

```
import pandas as pd

df = pd.read_csv('data.csv')

df["Calories"].fillna(130, inplace = True)
```

see more please visit: https://homeofpdf.com

Replace Using Mean, Median, or Mode

A common way to replace empty cells is to calculate the mean, median or mode value of the column.

Pandas uses the mean() median() and mode() methods to calculate the respective values for a specified column:

Example

Calculate the MEAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mean()

df["Calories"].fillna(x, inplace = True)
```

Mean = the average value (the sum of all values divided by number of values).

Example

Calculate the MEDIAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].median()

df["Calories"].fillna(x, inplace = True)
```

Median = the value in the middle, after you have sorted all values ascending.

Example

Calculate the MODE, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mode()[0]
```

see more please visit: https://homeofpdf.com

df["Calories"].fillna(x, inplace = True)

Mode = the value that appears most frequently.

Pandas - Cleaning Data of Wrong Format

Data of Wrong Format

Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

Convert Into a Correct Format

In our Data Frame, we have two cells with the wrong format. Check out row 22 and 26, the 'Date' column should be a string that represents a date:

	Duration	Date	Pulse	Max Pulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	2020/12/26	100	120	250.0
27	60	'2020/12/27'	92	118	241.0

see more please visit: https://homeofpdf.com

28	*60*	*'2020/12/28'*	*103*	*132*	*NaN*
29	*60*	*'2020/12/29'*	*100*	*132*	*280.0*
30	*60*	*'2020/12/30'*	*102*	*129*	*380.3*
31	*60*	*'2020/12/31'*	*92*	*115*	

Let's try to convert all cells in the 'Date' column into dates.

Pandas has a to_datetime() method for this:

Example

Convert to date:

```
import pandas as pd
df = pd.read_csv('data.csv')
df['Date'] = pd.to_datetime(df['Date'])
print(df.to_string())
```

Result:

	Duration	Date	Pulse	Max Pulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0

see more please visit: https://homeofpdf.com

20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaT	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	'2020/12/26'	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	

As you can see from the result, the date in row 26 was fixed, but the empty date in row 22 got a NaT (Not a Time) value, in other words an empty value. One way to deal with empty values is simply removing the entire row.

Removing Rows

The result from the conversion in the example above gave us a NaT value, which can be handled as a NULL value, and we can remove the row by using the dropna() method.

Example

Remove rows with a NULL value in the "Date" column:

```
df.dropna(subset=['Date'], inplace = True)
```

Pandas - Fixing Wrong Data

Wrong Data

"Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.

see more please visit: https://homeofpdf.com

If you take a look at our data set, you can see that in row 7, the duration is 450, but for all the other rows the duration is between 30 and 60.

It doesn't have to be wrong, but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact that this person did not work out in 450 minutes.

	Duration	Date	Pulse	Max Pulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	2020/12/26	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	

How can we fix wrong values, like the one for "Duration" in row 7?

Replacing Values

One way to fix wrong values is to replace them with something else.

In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7:

Example

see more please visit: https://homeofpdf.com

Set "Duration" = 45 in row 7:

```
df.loc[7, 'Duration'] = 45
```

For small data sets you might be able to replace the wrong data one by one, but not for big data sets.

To replace wrong data for larger data sets you can create some rules, e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

Example

Loop through all values in the "Duration" column.

If the value is higher than 120, set it to 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 120
```

Removing Rows

Another way of handling wrong data is to remove the rows that contains wrong data.

This way you do not have to find out what to replace them with, and there is a good chance you do not need them to do your analyses.

Example

Delete rows where "Duration" is higher than 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.drop(x, inplace = True)
```

Pandas - Removing Duplicates

see more please visit: https://homeofpdf.com

Discovering Duplicates

Duplicate rows are rows that have been registered more than one time.

	Duration	Date	Pulse	Max Pulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	2020/12/26	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	

By taking a look at our test data set, we can assume that row 11 and 12 are duplicates.

To discover duplicates, we can use the duplicated() method.

The duplicated() method returns a Boolean values for each row:

Example

Returns True for every row that is a duplicate, otherwise False:

```
print(df.duplicated())
```

Removing Duplicates

see more please visit: https://homeofpdf.com

To remove duplicates, use the drop_duplicates() method.

Example

Remove all duplicates:

df.drop_duplicates(inplace = True)

Remember: The (inplace = True) will make sure that the method does NOT return a *new* DataFrame, but it will remove all duplicates from the *original* DataFrame.

Pandas - Data Correlations

Finding Relationships

A great aspect of the Pandas module is the corr() method.

The corr() method calculates the relationship between each column in your data set.

The examples in this page uses a CSV file called: 'data.csv'.

Download data.csv. or Open data.csv

Example

Show the relationship between the columns:

df.corr()

Result

	Duration	Pulse	Max Pulse	Calories
Duration	1.000000	-0.155408	0.009403	0.922721
Pulse	-0.155408	1.000000	0.786535	0.025120
Maxpulse	0.009403	0.786535	1.000000	0.203814
Calories	0.922721	0.025120	0.203814	1.000000

Note: The corr() method ignores "not numeric" columns.

see more please visit: https://homeofpdf.com

Result Explained

The Result of the corr() method is a table with a lot of numbers that represents how well the relationship is between two columns.

The number varies from -1 to 1.

1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.

0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.

-0.9 would be just as good a relationship as 0.9, but if you increase one value, the other will probably go down.

0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.

What is a good correlation? It depends on the use, but I think it is safe to say you have to have at least 0.6 (or -0.6) to call it a good correlation.

Perfect Correlation:

We can see that "Duration" and "Duration" got the number 1.000000, which makes sense, each column always has a perfect relationship with itself.

Good Correlation:

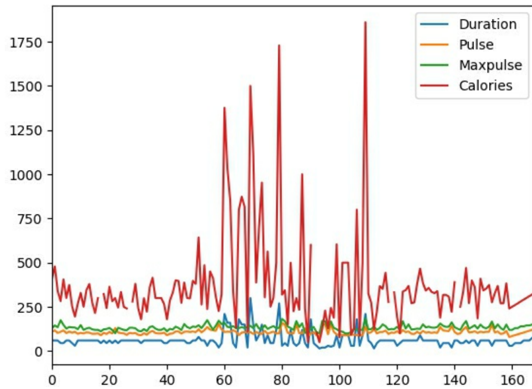
"Duration" and "Calories" have a 0.922721 correlation, which is a very good correlation, and we can predict that the longer you work out, the more calories you burn, and the other way around: if you burned a lot of calories, you probably had a long workout.

Bad Correlation:

"Duration" and "Maxpulse" have a 0.009403 correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the workout, and vice versa.

see more please visit: https://homeofpdf.com

Pandas - Plotting



Plotting

Pandas uses the plot() method to create diagrams.

We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.

Read more about Matplotlib in our Matplotlib Tutorial.

Example

Import pyplot from Matplotlib and visualize our DataFrame:

```
import pandas as pd

import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot()

plt.show()
```

The examples in this page uses a CSV file called: 'data.csv'.

Download data.csv or Open data.csv

Scatter Plot

see more please visit: https://homeofpdf.com

Specify that you want a scatter plot with the kind argument:

kind = 'scatter'

A scatter plot needs an x- and a y-axis.

In the example below we will use "Duration" for the x-axis and "Calories" for the y-axis.

Include the x and y arguments like this:

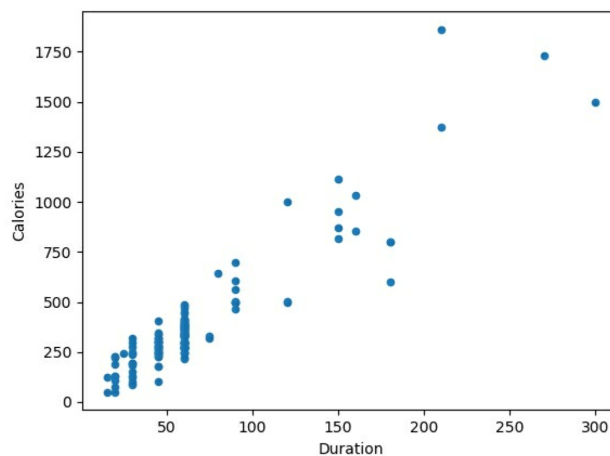
x = 'Duration', y = 'Calories'

Example

```
import pandas as pd

import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()
```

Result

Remember: In the previous example, we learned that the correlation between "Duration" and "Calories" was 0.922721, and we concluded that higher duration means more calories burned.



see more please visit: https://homeofpdf.com

By looking at the scatterplot, I will agree.

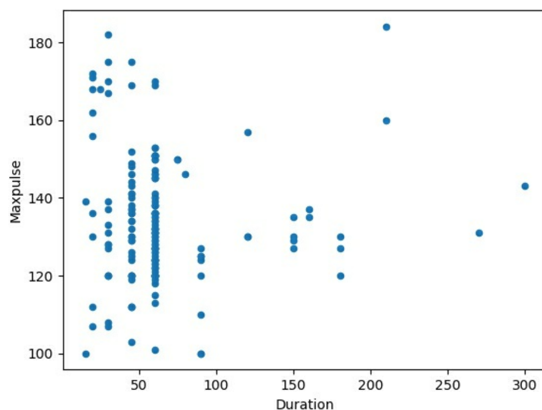
Let's create another scatterplot, where there is a bad relationship between the columns, like "Duration" and "Maxpulse", with the correlation 0.009403:

Example

A scatterplot where there are no relationship between the columns:

```
import pandas as pd

import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot(kind = 'scatter', x = 'Duration', y = 'Maxpulse')

plt.show()
```

Result



Histogram

Use the kind argument to specify that you want a histogram:

kind = 'hist'

A histogram needs only one column.

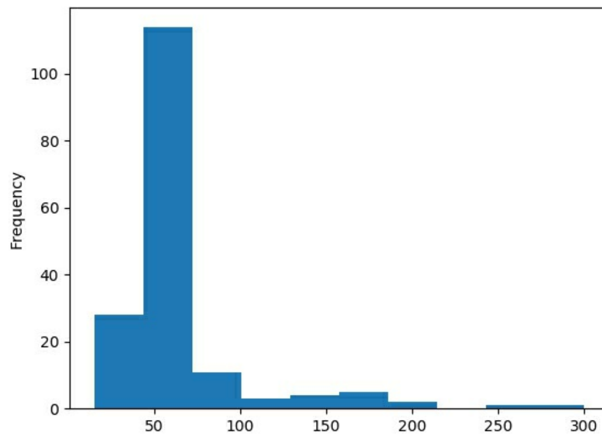
A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?

see more please visit: https://homeofpdf.com

In the example below we will use the "Duration" column to create the histogram:

Example

```
df["Duration"].plot(kind = 'hist')
```



Result

Note: The histogram tells us that there were over 100 workouts that lasted between 50 and 60 minutes.

see more please visit: https://homeofpdf.com

42.SciPy Tutorial

SciPy is a scientific computation library that uses NumPy underneath.

SciPy stands for Scientific Python.

Learning by Reading

We have created 10 tutorial for you to learn the fundamentals of SciPy:

- Getting Started
- Constants
- Optimizers
- Sparse Data
- Graphs
- Spatial Data
- Matlab Arrays
- Interpolation
- Significance Tests

SciPy Introduction

What is SciPy?

SciPy is a scientific computation library that uses NumPy underneath.

SciPy stands for Scientific Python.

It provides more utility functions for optimization, stats and signal processing.

Like NumPy, SciPy is open source so we can use it freely.

SciPy was created by NumPy's creator Travis Olliphant.

Why Use SciPy?

If SciPy uses NumPy underneath, why can we not just use NumPy?

see more please visit: https://homeofpdf.com

SciPy has optimized and added functions that are frequently used in NumPy and Data Science.

Which Language is SciPy Written in?

SciPy is predominantly written in Python, but a few segments are written in C.

Where is the SciPy Codebase?

The source code for SciPy is located at this github repository https://github.com/scipy/scipy

github: enables many people to work on the same codebase.

SciPy Getting Started

Installation of SciPy

If you have Python and PIP already installed on a system, then installation of SciPy is very easy.

Install it using this command:

C:\Users*Your Name*>pip install scipy

If this command fails, then use a Python distribution that already has SciPy installed like Anaconda, Spyder etc.

Import SciPy

Once SciPy is installed, import the SciPy module(s) you want to use in your applications by adding the from scipy import *module* statement:

```
from scipy import constants
```

Now we have imported the *constants* module from SciPy, and the application is ready to use it:

see more please visit: https://homeofpdf.com

Example

How many cubic meters are in one liter:

```
from scipy import constants

print(constants.liter)
```

constants: SciPy offers a set of mathematical constants, one of them is liter which returns 1 liter as cubic meters.

You will learn more about constants in the next chapter.

Checking SciPy Version

The version string is stored under the __version__ attribute.

Example

```
import scipy

print(scipy.__version__)
```

Note: two underscore characters are used in __version__.

SciPy Constants

Constants in SciPy

As SciPy is more focused on scientific implementations, it provides many built-in scientific constants.

These constants can be helpful when you are working with Data Science.

PI is an example of a scientific constant.

Example

Print the constant value of PI:

```
from scipy import constants
```

see more please visit: https://homeofpdf.com

```
print(constants.pi)
```

Constant Units

A list of all units under the constants module can be seen using the dir() function.

Example

List all constants:

```
from scipy import constants

print(dir(constants))
```

Unit Categories

The units are placed under these categories:

- Metric
- Binary
- Mass
- Angle
- Time
- Length
- Pressure
- Volume
- Speed
- Temperature
- Energy
- Power
- Force

Metric (SI) Prefixes:

Return the specified unit in **meter** (e.g. centi returns 0.01)

Example

see more please visit: https://homeofpdf.com

```
from scipy import constants

print(constants.yotta)    #1e+24
print(constants.zetta)    #1e+21
print(constants.exa)      #1e+18
print(constants.peta)     #1000000000000000.0
print(constants.tera)     #1000000000000.0
print(constants.giga)     #1000000000.0
print(constants.mega)     #1000000.0
print(constants.kilo)     #1000.0
print(constants.hecto)    #100.0
print(constants.deka)     #10.0
print(constants.deci)     #0.1
print(constants.centi)    #0.01
print(constants.milli)    #0.001
print(constants.micro)    #1e-06
print(constants.nano)     #1e-09
print(constants.pico)     #1e-12
print(constants.femto)    #1e-15
print(constants.atto)     #1e-18
print(constants.zepto)    #1e-21
```

Binary Prefixes:

Return the specified unit in **bytes** (e.g. kibi returns 1024)

see more please visit: https://homeofpdf.com

Example

```
from scipy import constants


print(constants.kibi)      #1024
print(constants.mebi)      #1048576
print(constants.gibi)    #1073741824
print(constants.tebi)       #1099511627776
print(constants.pebi)      #1125899906842624
print(constants.exbi)      #1152921504606846976
print(constants.zebi)       #1180591620717411303424
print(constants.yobi)      #1208925819614629174706176
```

Mass:

Return the specified unit in **kg** (e.g. gram returns 0.001)

Example

```
from scipy import constants


print(constants.gram)       #0.001
print(constants.metric_ton)  #1000.0
print(constants.grain)    #6.479891e-05
print(constants.lb)         #0.45359236999999997
print(constants.pound)      #0.45359236999999997
print(constants.oz)         #0.028349523124999998
print(constants.ounce)           #0.028349523124999998
```

see more please visit: https://homeofpdf.com

```
print(constants.stone)    #6.3502931799999995
print(constants.long_ton)       #1016.0469088
print(constants.short_ton)   #907.1847399999999
print(constants.troy_ounce)  #0.031103476799999998
print(constants.troy_pound)  #0.37324172159999996
print(constants.carat)     #0.0002
print(constants.atomic_mass) #1.66053904e-27
print(constants.m_u)      #1.66053904e-27
print(constants.u)         #1.66053904e-27
```

Angle:

Return the specified unit in **radians** (e.g. degree returns 0.017453292519943295)

Example

```
from scipy import constants


print(constants.degree)     #0.017453292519943295
print(constants.arcmin) #0.0002908882086657216
print(constants.arcminute)  #0.0002908882086657216
print(constants.arcsec)  #4.84813681109536e-06
print(constants.arcsecond)  #4.84813681109536e-06
```

Time:

Return the specified unit in **seconds** (e.g. hour returns 3600.0)

Example

see more please visit: https://homeofpdf.com

```
from scipy import constants


print(constants.minute)        #60.0
print(constants.hour)      #3600.0
print(constants.day)       #86400.0
print(constants.week)      #604800.0
print(constants.year)     #31536000.0
print(constants.Julian_year) #31557600.0
```

Length:

Return the specified unit in **meters** (e.g. nautical_mile returns 1852.0)

Example

```
from scipy import constants


print(constants.inch)         #0.0254
print(constants.foot)               #0.30479999999999996
print(constants.yard)             #0.9143999999999999
print(constants.mile)           #1609.3439999999998
print(constants.mil)             #2.5399999999999997e-05
print(constants.pt)                #0.00035277777777777776
print(constants.point)            #0.00035277777777777776
print(constants.survey_foot)    #0.3048006096012192
print(constants.survey_mile)          #1609.3472186944373
print(constants.nautical_mile)    #1852.0
```

see more please visit: https://homeofpdf.com

```
print(constants.fermi)            #1e-15
print(constants.angstrom)     #1e-10
print(constants.micron)          #1e-06
print(constants.au)          #149597870691.0
print(constants.astronomical_unit) #149597870691.0
print(constants.light_year)      #9460730472580800.0
print(constants.parsec)        #3.0856775813057292e+16
```

Pressure:

Return the specified unit in **pascals** (e.g. psi returns 6894.757293168361)

Example

```
from scipy import constants


print(constants.atm)              #101325.0
print(constants.atmosphere)  #101325.0
print(constants.bar)       #100000.0
print(constants.torr)       #133.32236842105263
print(constants.mmHg)        #133.32236842105263
print(constants.psi)       #6894.757293168361
```

Area:

Return the specified unit in **square meters**(e.g. hectare returns 10000.0)

Example

```
from scipy import constants
```

see more please visit: https://homeofpdf.com

```
print(constants.hectare) #10000.0
print(constants.acre)      #4046.8564223999992
```

Volume:

Return the specified unit in **cubic meters** (e.g. liter returns 0.001)

Example

```
from scipy import constants


print(constants.liter)                #0.001
print(constants.litre)                 #0.001
print(constants.gallon)          #0.0037854117839999997
print(constants.gallon_US)       #0.0037854117839999997
print(constants.gallon_imp)          #0.00454609
print(constants.fluid_ounce)         #2.9573529562499998e-05
print(constants.fluid_ounce_US)   #2.9573529562499998e-05
print(constants.fluid_ounce_imp)  #2.84130625e-05
print(constants.barrel)           #0.15898729492799998
print(constants.bbl)         #0.15898729492799998
```

Speed:

Return the specified unit in **meters per second** (e.g. speed_of_sound returns 340.5)

Example

```
from scipy import constants
```

see more please visit: https://homeofpdf.com

```
print(constants.kmh)              #0.2777777777777778
print(constants.mph)              #0.44703999999999994
print(constants.mach)             #340.5
print(constants.speed_of_sound) #340.5
print(constants.knot)             #0.5144444444444445
```

Temperature:

Return the specified unit in **Kelvin** (e.g. zero_Celsius returns 273.15)

Example

```
from scipy import constants

print(constants.zero_Celsius)         #273.15
print(constants.degree_Fahrenheit) #0.5555555555555556
```

Energy:

Return the specified unit in **joules** (e.g. calorie returns 4.184)

Example

```
from scipy import constants

print(constants.eV)                   #1.6021766208e-19
print(constants.electron_volt) #1.6021766208e-19
print(constants.calorie)       #4.184
print(constants.calorie_th)     #4.184
print(constants.calorie_IT)     #4.1868
print(constants.erg)         #1e-07
```

see more please visit: https://homeofpdf.com

```
print(constants.Btu)             #1055.05585262
print(constants.Btu_IT)          #1055.05585262
print(constants.Btu_th)          #1054.3502644888888
print(constants.ton_TNT)         #4184000000.0
```

Power:

Return the specified unit in **watts** (e.g. horsepower returns 745.6998715822701)

Example

```
from scipy import constants


print(constants.hp)                #745.6998715822701
print(constants.horsepower) #745.6998715822701
```

Force:

Return the specified unit in **newton** (e.g. kilogram_force returns 9.80665)

Example

```
from scipy import constants


print(constants.dyn)          #1e-05
print(constants.dyne)         #1e-05
print(constants.lbf)          #4.4482216152605
print(constants.pound_force)  #4.4482216152605
print(constants.kgf)               #9.80665
print(constants.kilogram_force)  #9.80665
```

see more please visit: https://homeofpdf.com

SciPy Optimizers

Optimizers in SciPy

Optimizers are a set of procedures defined in SciPy that either find the minimum value of a function, or the root of an equation.

Optimizing Functions

Essentially, all of the algorithms in Machine Learning are nothing more than a complex equation that needs to be minimized with the help of given data.

Roots of an Equation

NumPy is capable of finding roots for polynomials and linear equations, but it can not find roots for *nonlinear* equations, like this one:

x + cos(x)

For that you can use SciPy's optimize.root function.

This function takes two required arguments:

fun - a function representing an equation.

x0 - an initial guess for the root.

The function returns an object with information regarding the solution.

The actual solution is given under attribute x of the returned object:

Example

Find root of the equation x + cos(x):

```
from scipy.optimize import root

from math import cos

def eqn(x):

  return x + cos(x)
```

see more please visit: https://homeofpdf.com

```
myroot = root(eqn, 0)

print(myroot.x)
```

Note: The returned object has much more information about the solution.

Example

Print all information about the solution (not just x which is the root)

```
print(myroot)
```

Minimizing a Function

A function, in this context, represents a curve, curves have *high points* and *low points*.

High points are called *maxima*.

Low points are called *minima*.

The highest point in the whole curve is called *global maxima*, whereas the rest of them are called *local maxima*.

The lowest point in the whole curve is called *global minima*, whereas the rest of them are called *local minima*.

Finding Minima

We can use scipy.optimize.minimize() function to minimize the function.

The minimize() function takes the following arguments:

fun - a function representing an equation.

x0 - an initial guess for the root.

method - name of the method to use. Legal values:

'CG'

'BFGS'

'Newton-CG'

see more please visit: https://homeofpdf.com

'L-BFGS-B'

'TNC'

'COBYLA'

'SLSQP'

callback - function called after each iteration of optimization.

options - a dictionary defining extra params:

```
{
    "disp": boolean - print detailed description
    "gtol": number - the tolerance of the error
 }
```

Example

Minimize the function x^2 + x + 2 with BFGS:

```
from scipy.optimize import minimize
def eqn(x):
  return x**2 + x + 2
mymin = minimize(eqn, 0, method='BFGS')
print(mymin)
```

SciPy Sparse Data

What is Sparse Data

Sparse data is data that has mostly unused elements (elements that don't carry any information).

It can be an array like this one:

see more please visit: https://homeofpdf.com

[1, 0, 2, 0, 0, 3, 0, 0, 0, 0, 0, 0]

Sparse Data: is a data set where most of the item values are zero.

Dense Array: is the opposite of a sparse array: most of the values are *not* zero.

In scientific computing, when we are dealing with partial derivatives in linear algebra we will come across sparse data.

How to Work With Sparse Data

SciPy has a module, scipy.sparse that provides functions to deal with sparse data.

There are primarily two types of sparse matrices that we use:

CSC - Compressed Sparse Column. For efficient arithmetic, fast column slicing.

CSR - Compressed Sparse Row. For fast row slicing, faster matrix vector products

We will use the **CSR** matrix in this tutorial.

CSR Matrix

We can create a CSR matrix by passing an array into the function scipy.sparse.csr_matrix().

Example

Create a CSR matrix from an array:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])

print(csr_matrix(arr))
```

see more please visit: https://homeofpdf.com

The example above returns:

(0, 5)   1

(0, 6)   1

(0, 8)   2

From the result we can see that there are 3 items with value.

The 1. item is in row 0 position 5 and has the value 1.

The 2. item is in row 0 position 6 and has the value 1.

The 3. item is in row 0 position 8 and has the value 2.

Sparse Matrix Methods

Viewing stored data (not the zero items) with the data property:

Example

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
print(csr_matrix(arr).data)
```

Counting nonzeros with the count_nonzero() method:

Example

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
print(csr_matrix(arr).count_nonzero())
```

Removing zero-entries from the matrix with the eliminate_zeros() method:

Example

```
import numpy as np
from scipy.sparse import csr_matrix
```

see more please visit: https://homeofpdf.com

```
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
mat = csr_matrix(arr)
mat.eliminate_zeros()
print(mat)
```

Eliminating duplicate entries with the sum_duplicates() method:

Example

Eliminating duplicates by adding them:

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
mat = csr_matrix(arr)
mat.sum_duplicates()
print(mat)
```

Converting from csr to csc with the tocsc() method:

Example

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
newarr = csr_matrix(arr).tocsc()
print(newarr)
```

Note: Apart from the mentioned sparse specific operations, sparse matrices support all of the operations that normal matrices support e.g. reshaping, summing, arithmetic, broadcasting etc.

SciPy Graphs

see more please visit: https://homeofpdf.com

Working with Graphs

Graphs are an essential data structure.

SciPy provides us with the module scipy.sparse.csgraph for working with such data structures.

Adjacency Matrix

Adjacency matrix is a nxn matrix where n is the number of elements in a graph.

And the values represent the connection between the elements.

For a graph like this, with elements A, B and C, the connections are:

A & B are connected with weight 1.

A & C are connected with weight 2.

C & B is not connected.

The Adjacency Matrix would look like this:

A B C

A:[0 1 2]

B:[1 0 0]

C:[2 0 0]

Below follows some of the most used methods for working with adjacency matrices.

Connected Components

Find all of the connected components with the connected_components() method.

Example

```
import numpy as np

from scipy.sparse.csgraph import connected_components
```

see more please visit: https://homeofpdf.com

```
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
print(connected_components(newarr))
```

Dijkstra

Use the dijkstra method to find the shortest path in a graph from one element to another.

It takes following arguments:

1. **return_predecessors:** boolean (True to return whole path of traversal otherwise False).
2. **indices:** index of the element to return all paths from that element only.
3. **limit:** max weight of path.

Example

Find the shortest path from element 1 to 2:

```
import numpy as np
from scipy.sparse.csgraph import dijkstra
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
```

see more please visit: https://homeofpdf.com

```
])
newarr = csr_matrix(arr)
print(dijkstra(newarr, return_predecessors=True, indices=0))
```

Floyd Warshall

Use the floyd_warshall() method to find the shortest path between all pairs of elements.

Example

Find the shortest path between all pairs of elements:

```
import numpy as np
from scipy.sparse.csgraph import floyd_warshall
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
print(floyd_warshall(newarr, return_predecessors=True))
```

Bellman Ford

The bellman_ford() method can also find the shortest path between all pairs of elements, but this method can handle negative weights as well.

Example

Find shortest path from element 1 to 2 with given graph with a negative weight:

```
import numpy as np
```

see more please visit: https://homeofpdf.com

```
from scipy.sparse.csgraph import bellman_ford
from scipy.sparse import csr_matrix
arr = np.array([
  [0, -1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
print(bellman_ford(newarr, return_predecessors=True, indices=0))
```

Depth First Order

The depth_first_order() method returns a depth first traversal from a node.

This function takes following arguments:

1. the graph.
2. the starting element to traverse the graph from.

Example

Traverse the graph depth first for given adjacency matrix:

```
import numpy as np
from scipy.sparse.csgraph import depth_first_order
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 0, 1],
  [1, 1, 1, 1],
  [2, 1, 1, 0],
  [0, 1, 0, 1]
])
```

see more please visit: https://homeofpdf.com

```
newarr = csr_matrix(arr)
print(depth_first_order(newarr, 1))
```

Breadth First Order

The breadth_first_order() method returns a breadth first traversal from a node.

This function takes following arguments:

1. the graph.
2. the starting element to traverse the graph from.

Example

Traverse the graph breadth first for given adjacency matrix:

```
import numpy as np
from scipy.sparse.csgraph import breadth_first_order
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 0, 1],
  [1, 1, 1, 1],
  [2, 1, 1, 0],
  [0, 1, 0, 1]
])
newarr = csr_matrix(arr)
print(breadth_first_order(newarr, 1))
```

SciPy Spatial Data

Working with Spatial Data

see more please visit: https://homeofpdf.com

Spatial data refers to data that is represented in a geometric space.

E.g. points on a coordinate system.

We deal with spatial data problems on many tasks.

E.g. finding if a point is inside a boundary or not.

SciPy provides us with the module scipy.spatial, which has functions for working with spatial data.

Triangulation

A Triangulation of a polygon is to divide the polygon into multiple triangles with which we can compute an area of the polygon.

A Triangulation *with points* means creating surface composed triangles in which all of the given points are on at least one vertex of any triangle in the surface.

One method to generate these triangulations through points is the Delaunay() Triangulation.

Example

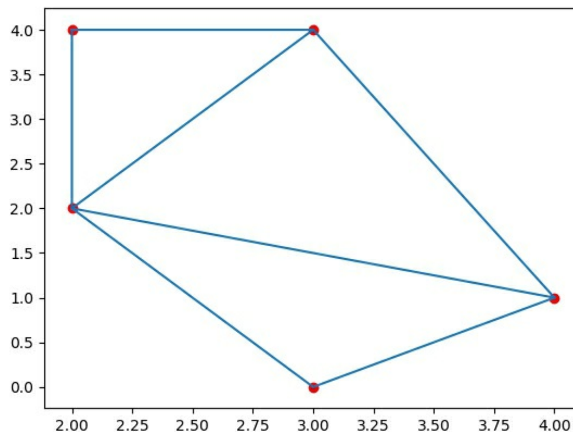
Create a triangulation from following points:

```
import numpy as np
from scipy.spatial import Delaunay
import matplotlib.pyplot as plt

points = np.array([
  [2, 4],
  [3, 4],
  [3, 0],
  [2, 2],
  [4, 1]
```

see more please visit: https://homeofpdf.com

```
])
simplices = Delaunay(points).simplices

plt.triplot(points[:, 0], points[:, 1], simplices)
plt.scatter(points[:, 0], points[:, 1], color='r')
plt.show()
```

Result:



Note: The simplices property creates a generalization of the triangle notation.

Convex Hull

A convex hull is the smallest polygon that covers all of the given points.

Use the ConvexHull() method to create a Convex Hull.

Example

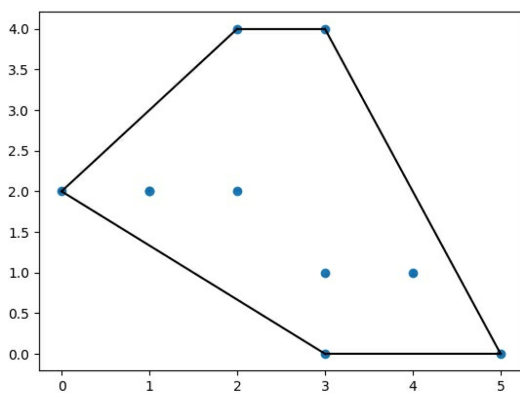
Create a convex hull for following points:

```
import numpy as np
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt
points = np.array([
```

see more please visit: https://homeofpdf.com

```
  [2, 4],
  [3, 4],
  [3, 0],
  [2, 2],
  [4, 1],
  [1, 2],
  [5, 0],
  [3, 1],
  [1, 2],
  [0, 2]
])

hull = ConvexHull(points)
hull_points = hull.simplices
plt.scatter(points[:,0], points[:,1])
for simplex in hull_points:
  plt.plot(points[simplex,0], points[simplex,1], 'k-')
plt.show()
```

Result:



see more please visit: https://homeofpdf.com

KDTrees

KDTrees are a datastructure optimized for nearest neighbor queries.

E.g. in a set of points using KDTrees we can efficiently ask which points are nearest to a certain given point.

The KDTree() method returns a KDTree object.

The query() method returns the distance to the nearest neighbor *and* the location of the neighbors.

Example

Find the nearest neighbor to point (1,1):

```
from scipy.spatial import KDTree

points = [(1, -1), (2, 3), (-2, 3), (2, -3)]
kdtree = KDTree(points)
res = kdtree.query((1, 1))
print(res)
```

Result:
(2.0, 0)

Distance Matrix

There are many Distance Metrics used to find various types of distances between two points in data science, Euclidean distance, cosine distance etc.

The distance between two vectors may not only be the length of straight line between them, it can also be the angle between them from origin, or number of unit steps required etc.

Many of the Machine Learning algorithm's performance depends greatly on distance matrices. E.g. "K Nearest Neighbors", or "K Means" etc.

Let us look at some of the Distance Metrics:

see more please visit: https://homeofpdf.com

Euclidean Distance

Find the euclidean distance between given points.

Example

```
from scipy.spatial.distance import euclidean

p1 = (1, 0)
p2 = (10, 2)
res = euclidean(p1, p2)
print(res)
```

Result:

9.21954445729

Cityblock Distance (Manhattan Distance)

Is the distance computed using 4 degrees of movement.

E.g. we can only move: up, down, right, or left, not diagonally.

Example

Find the cityblock distance between given points:

```
from scipy.spatial.distance import cityblock
p1 = (1, 0)
p2 = (10, 2)
res = cityblock(p1, p2)
print(res)
```

Result:

11

Cosine Distance

see more please visit: https://homeofpdf.com

Is the value of cosine angle between the two points A and B.

Example

Find the cosine distance between given points:

```
from scipy.spatial.distance import cosine

p1 = (1, 0)

p2 = (10, 2)

res = cosine(p1, p2)

print(res)
```

Result:
0.019419324309079777

Hamming Distance

Is the proportion of bits where two bits are different.

It's a way to measure distance for binary sequences.

Example

Find the hamming distance between given points:

```
from scipy.spatial.distance import hamming

p1 = (True, False, True)

p2 = (False, True, True)

res = hamming(p1, p2)

print(res)
```

Result:
0.666666666667

SciPy Matlab Arrays

see more please visit: https://homeofpdf.com

Working With Matlab Arrays

We know that NumPy provides us with methods to persist the data in readable formats for Python. But SciPy provides us with interoperability with Matlab as well.

SciPy provides us with the module scipy.io, which has functions for working with Matlab arrays.

Exporting Data in Matlab Format

The savemat() function allows us to export data in Matlab format.

The method takes the following parameters:

1. **filename** - the file name for saving data.
2. **mdict** - a dictionary containing the data.
3. **do_compression** - a boolean value that specifies whether to compress the result or not. Default False.

Example

Export the following array as variable name "vec" to a mat file:

```
from scipy import io

import numpy as np

arr = np.arange(10)

io.savemat('arr.mat', {"vec": arr})
```

Note: The example above saves a file name "arr.mat" on your computer.

To open the file, check out the "Import Data from Matlab Format" example below:

Import Data from Matlab Format

The loadmat() function allows us to import data from a Matlab file.

The function takes one required parameter:

see more please visit: https://homeofpdf.com

filename - the file name of the saved data.

It will return a structured array whose keys are the variable names, and the corresponding values are the variable values.

Example

Import the array from following mat file.:

```
from scipy import io
import numpy as np
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,])
# Export:
io.savemat('arr.mat', {"vec": arr})
# Import:
mydata = io.loadmat('arr.mat')
print(mydata)
```

Result:

```
{
  '__header__': b'MATLAB 5.0 MAT-file Platform: nt, Created on: Tue Sep 22 13:12:32 2020',
  '__version__': '1.0',
  '__globals__': [],
  'vec': array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
}
```

Use the variable name "vec" to display only the array from the matlab data:

Example

```
...
print(mydata['vec'])
```

Result:

```
[[0 1 2 3 4 5 6 7 8 9]]
```

Note: We can see that the array originally was 1D, but on extraction it has increased one dimension.

see more please visit: https://homeofpdf.com

In order to resolve this we can pass an additional argument squeeze_me=True:

Example

```
# Import:
mydata = io.loadmat('arr.mat', squeeze_me=True)
print(mydata['vec'])
```

Result:

[0 1 2 3 4 5 6 7 8 9]

SciPy Interpolation

What is Interpolation?

Interpolation is a method for generating points between given points.

For example: for points 1 and 2, we may interpolate and find points 1.33 and 1.66.

Interpolation has many uses, in Machine Learning we often deal with missing data in a dataset, interpolation is often used to substitute those values.

This method of filling values is called *imputation*.

Apart from imputation, interpolation is often used where we need to smooth the discrete points in a dataset.

How to Implement it in SciPy?

SciPy provides us with a module called scipy.interpolate which has many functions to deal with interpolation:

1D Interpolation

The function interp1d() is used to interpolate a distribution with 1 variable.

It takes x and y points and returns a callable function that can be called with

see more please visit: https://homeofpdf.com

new x and returns corresponding y.

Example

For given xs and ys interpolate values from 2.1, 2.2... to 2.9:

```
from scipy.interpolate import interp1d
import numpy as np
xs = np.arange(10)
ys = 2*xs + 1
interp_func = interp1d(xs, ys)
newarr = interp_func(np.arange(2.1, 3, 0.1))
print(newarr)
```

Result:

```
[5.2  5.4  5.6  5.8  6.   6.2  6.4  6.6  6.8]
```

Note: that new xs should be in the same range as of the old xs, meaning that we can't call interp_func() with values higher than 10, or less than 0.

Spline Interpolation

In 1D interpolation the points are fitted for a *single curve* whereas in Spline interpolation the points are fitted against a *piecewise* function defined with polynomials called splines.

The UnivariateSpline() function takes xs and ys and produces a callback function that can be called with new xs.

Piecewise function: A function that has different definitions for different ranges.

Example

Find univariate spline interpolation for 2.1, 2.2... 2.9 for the following non linear points:

```
from scipy.interpolate import UnivariateSpline
```

see more please visit: https://homeofpdf.com

```
import numpy as np
xs = np.arange(10)
ys = xs**2 + np.sin(xs) + 1
interp_func = UnivariateSpline(xs, ys)
newarr = interp_func(np.arange(2.1, 3, 0.1))
print(newarr)
```

Result:

*[5.62826474 6.03987348 6.47131994 6.92265019 7.3939103 7.88514634
 8.39640439 8.92773053 9.47917082]*

Interpolation with Radial Basis Function

Radial basis function is a function that is defined corresponding to a fixed reference point.

The Rbf() function also takes xs and ys as arguments and produces a callable function that can be called with new xs.

Example

Interpolate following xs and ys using rbf and find values for 2.1, 2.2 ... 2.9:

```
from scipy.interpolate import Rbf

import numpy as np

xs = np.arange(10)

ys = xs**2 + np.sin(xs) + 1

interp_func = Rbf(xs, ys)

newarr = interp_func(np.arange(2.1, 3, 0.1))

print(newarr)
```

Result:

see more please visit: https://homeofpdf.com

*[6.25748981 6.62190817 7.00310702 7.40121814 7.8161443   8.24773402
  8.69590519 9.16070828 9.64233874]*

SciPy Statistical Significance Tests

What is a Statistical Significance Test?

In statistics, statistical significance means that the result that was produced has a reason behind it, it was not produced randomly, or by chance.

SciPy provides us with a module named scipy.stats, which has functions for performing statistical significance tests.

Here are some techniques and keywords that are important when performing such tests.

Hypothesis in Statistics

Hypothesis is an assumption about a parameter in population.

Null Hypothesis

It assumes that the observation is not statistically significant.

Alternate Hypothesis

It assumes that the observations are due to some reason.

It's an alternative to Null Hypothesis.

Example:

For an assessment of a student we would take:

"student is worse than average" - as a null hypothesis, and:

"student is better than average" - as an alternate hypothesis.

One tailed test

When our hypothesis is testing for one side of the value only, it is called "one

see more please visit: https://homeofpdf.com

tailed test".

Example:

For the null hypothesis:

"the mean is equal to k", we can have alternate hypothesis:

"the mean is less than k", or:

"the mean is greater than k"

Two tailed test

When our hypothesis is tested for both sides of the values.

Example:

For the null hypothesis:

"the mean is equal to k", we can have alternate hypothesis:

"the mean is not equal to k"

In this case the mean is less than, or greater than k, and both sides are to be checked.

Alpha value

Alpha value is the level of significance.

Example:

How close to extremes the data must be for null hypothesis to be rejected.

It is usually taken as 0.01, 0.05, or 0.1.

P value

P value tells how close to extreme the data actually is.

P value and alpha values are compared to establish statistical significance.

If p value <= alpha we reject the null hypothesis and say that the data is statistically significant. otherwise we accept the null hypothesis.

see more please visit: https://homeofpdf.com

T-Test

T-tests are used to determine if there is significant difference between means of two variables. and lets us know if they belong to the same distribution.

It is a two tailed test.

The function ttest_ind() takes two samples of the same size and produces a tuple of t-statistic and p-value.

Example

Find if the given values v1 and v2 are from same distribution:

```
import numpy as np
from scipy.stats import ttest_ind
v1 = np.random.normal(size=100)
v2 = np.random.normal(size=100)
res = ttest_ind(v1, v2)
print(res)
```

Result:

0.68346891833752133

If you want to return only the p-value, use the p value property:

Example

```
...
res = ttest_ind(v1, v2).pvalue
print(res)
```

Result:

0.68346891833752133

KS-Test

KS test is used to check if given values follow a distribution.

see more please visit: https://homeofpdf.com

The function takes the value to be tested, and the CDF as two parameters.

A **CDF** can be either a string or a callable function that returns the probability.

It can be used as a one tailed or two tailed test.

By default it is two tails. We can pass parameter alternatives as a string of one of two-sided, less, or greater.

Example

Find if the given value follows the normal distribution:

```
import numpy as np
from scipy.stats import kstest
v = np.random.normal(size=100)
res = kstest(v, 'norm')
print(res)
```

Result:

KstestResult(statistic=0.047798701221956841, pvalue=0.97630967161777515)

Statistical Description of Data

In order to see a summary of values in an array, we can use the describe() function.

It returns the following description:

1. number of observations (nobs)
2. minimum and maximum values = minmax
3. mean
4. variance
5. skewness
6. kurtosis

see more please visit: https://homeofpdf.com

Example

Show statistical description of the values in an array:

```
import numpy as np
from scipy.stats import describe

v = np.random.normal(size=100)
res = describe(v)
print(res)
```

Result:

DescribeResult(

nobs=100,

minmax=(-2.0991855456740121, 2.1304142707414964),

mean=0.11503747689121079,

variance=0.99418092655064605,

skewness=0.013953400984243667,

kurtosis=-0.671060517912661

)

Normality Tests (Skewness and Kurtosis)

Normality tests are based on the skewness and kurtosis.

The normaltest() function returns p value for the null hypothesis:

"x comes from a normal distribution".

Skewness:

A measure of symmetry in data.

For normal distributions it is 0.

If it is negative, it means the data is skewed left.

If it is positive it means the data is skewed right.

Kurtosis:

see more please visit: https://homeofpdf.com

A measure of whether the data is heavy or lightly tailed to a normal distribution.

Positive kurtosis means heavy tailed.

Negative kurtosis means lightly tailed.

Example

Find skewness and kurtosis of values in an array:

```
import numpy as np
from scipy.stats import skew, kurtosis
v = np.random.normal(size=100)
print(skew(v))
print(kurtosis(v))
```

Result:

0.11168446328610283

-0.1879320563260931

Example

Find if the data comes from a normal distribution:

```
import numpy as np
from scipy.stats import normaltest
v = np.random.normal(size=100)
print(normaltest(v))
```

Result:

NormaltestResult(statistic=4.4783745697002848, pvalue=0.10654505998635538)

see more please visit: https://homeofpdf.com