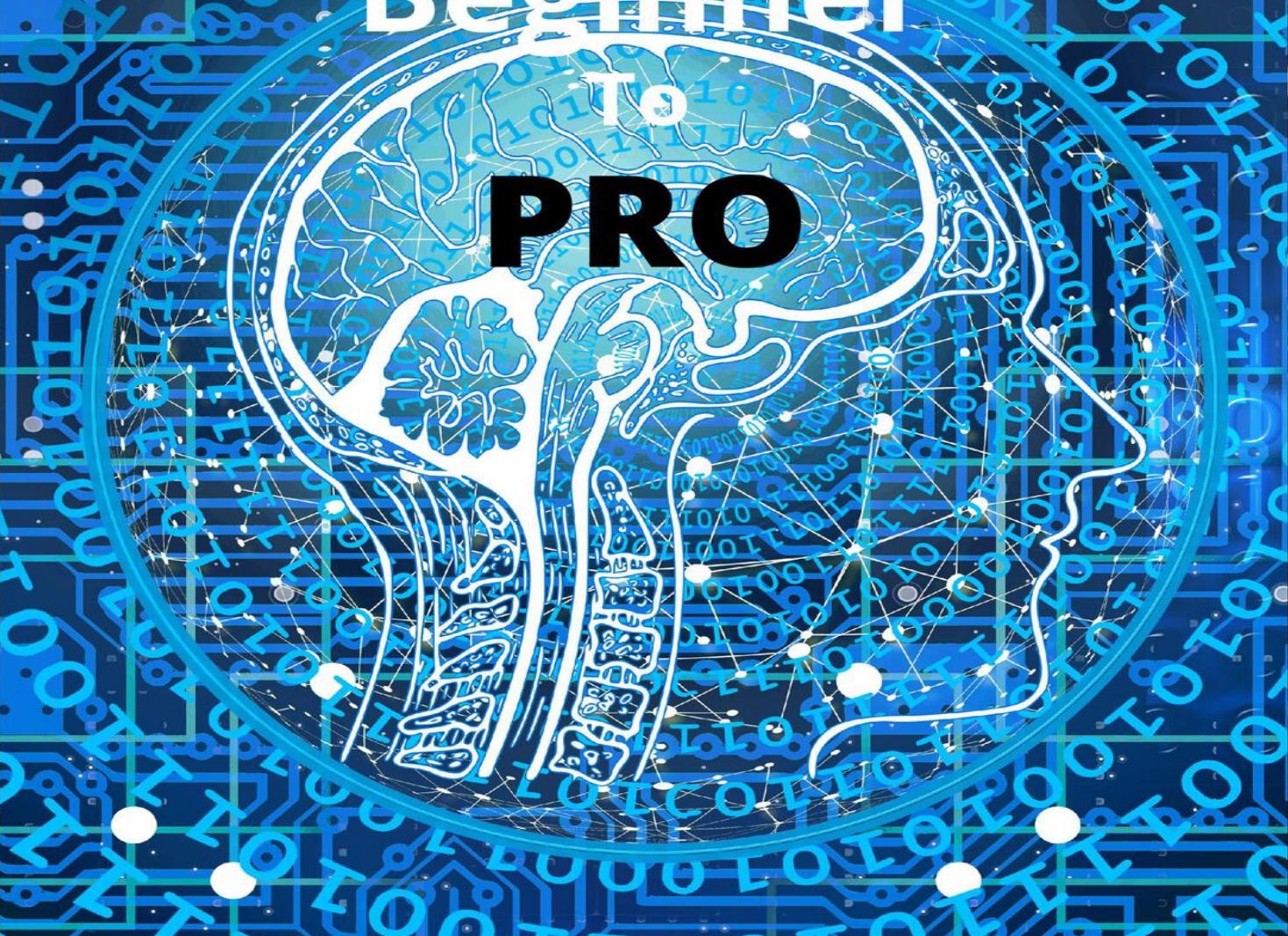


PYTHON

Beginner

To
PRO

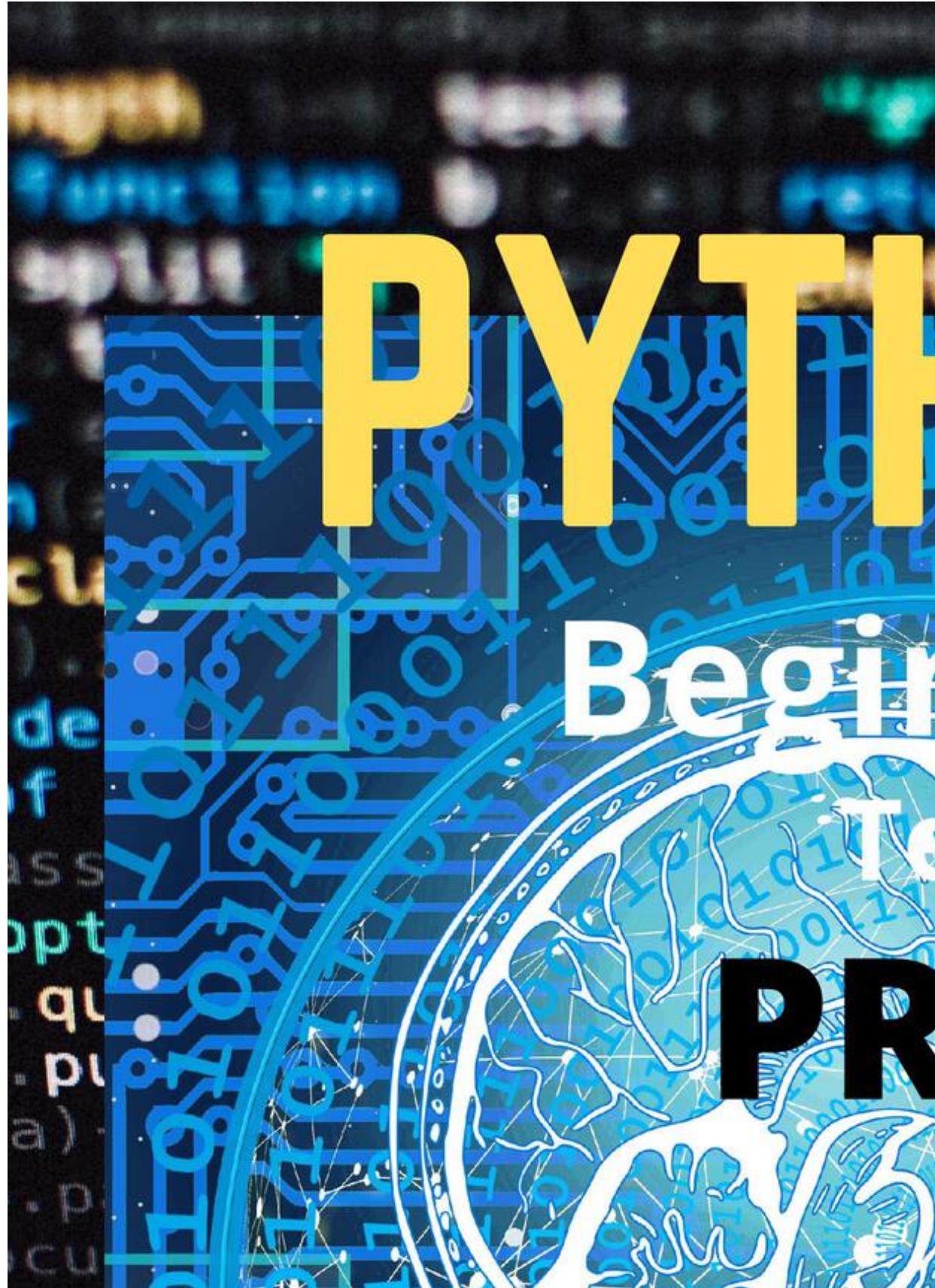


**Python Tutorial, File Handling, Python NumPy,
Python Matplotlib, Python SciPy, Machine
Learning, Python MySQL, Python MySQL, Python
Reference, Module Reference, Python Examples**

PYTHON

Beginner

PR



Python

Beginner To Pro

Every topics with unique Examples

In this book you will learn from A to Z in Python such as Python Tutorial, File Handling, Python NumPy, Python Matplotlib, Python SciPy, Machine Learning, Python MySQL, Python MySQL, Python Reference, Module Reference, Python Examples And more.

N KRISHNA KUMAR

Table Of Content

[What can Python do?](#)

[Why Python?](#)

[Good to know](#)

[Python Syntax compared to other programming languages](#)

[Python Quickstart](#)

[The Python Command Line](#)

[Execute Python Syntax](#)

[Python Indentation](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Python Variables](#)

[Example](#)

[Comments](#)

[Example](#)

[Creating a Comment](#)

[Example](#)

[Example](#)

[Example](#)

[Multi Line Comments](#)

[Example](#)

[Example](#)

[Variables](#)

[Creating Variables](#)

[Example](#)

[Example](#)

[Casting](#)

[Example](#)

[Get the Type](#)

[Example](#)

[Single or Double Quotes?](#)

[Example](#)

[Case-Sensitive](#)

[Example](#)

[Python - Variable Names](#)

[Variable Names](#)

[Example](#)

[Example](#)

[Multi Words Variable Names](#)

[Camel Case](#)

[Pascal Case](#)

[Snake Case](#)

[Python Variables - Assign Multiple Values](#)

[Many Values to Multiple Variables](#)

[Example](#)

[One Value to Multiple Variables](#)

[Example](#)

[Unpack a Collection](#)

[Example](#)

[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:](#)

`print("Python is " + x)`

[You can also use the + character to add a variable to another variable:](#)

`x = "Python is "`

`y = "awesome"`

`z = x + y`

`print(z)`

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

Example

x = 5

y = 10

print(x + y)

Example

x = 5

y = "John"

print(x + y)

Python - Global Variables

Global Variables

Example

Example

The global Keyword

Example

Example

Built-in Data Types

Getting the Data Type

Example

Setting the Data Type

Setting the Specific Data Type

Python Numbers

Example

Example

Int

Example

Float

Example

Example

Complex

Example

[Type Conversion](#)

[Example](#)

[Random Number](#)

[Example](#)

[Specify a Variable Type](#)

[Example](#)

[Example](#)

[Example](#)

[Strings](#)

[Example](#)

[Assign String to a Variable](#)

[Example](#)

[Multiline Strings](#)

[Example](#)

[Example](#)

[Strings are Arrays](#)

[Example](#)

[Looping Through a String](#)

[Example](#)

[String Length](#)

[Example](#)

[Check String](#)

[Example](#)

[Example](#)

[Check if NOT](#)

[Example](#)

[Example](#)

[Python - Slicing Strings](#)

[Slicing](#)

[Example](#)

[Slice From the Start](#)

[Example](#)

[Slice To the End](#)

[Example](#)

[Negative Indexing](#)

[Example](#)

[Python - Modify Strings](#)

[Upper Case](#)

[Example](#)

[Lower Case](#)

[Example](#)

[Remove Whitespace](#)

[Example](#)

[Replace String](#)

[Example](#)

[Split String](#)

[Example](#)

[Python - String Concatenation](#)

[String Concatenation](#)

[Example](#)

[Example](#)

[String Format](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Escape Character](#)

[Example](#)

[Example](#)

[Escape Characters](#)

[Python - String Methods](#)

[String Methods](#)

[Boolean Values](#)

[Example](#)

[Example](#)

[Evaluate Values and Variables](#)

[Example](#)

[Example](#)

[Most Values are True](#)

[Example](#)

[Some Values are False](#)

[Example](#)

[Example](#)

[Functions can Return a Boolean](#)

[Example](#)

[Example](#)

[Example](#)

[Python Operators](#)

[Example](#)

[Python Arithmetic Operators](#)

[Python Assignment Operators](#)

[Python Comparison Operators](#)

[Python Logical Operators](#)

[Python Identity Operators](#)

[Python Membership Operators](#)

[Python Bitwise Operators](#)

[List](#)

[Example](#)

[List Items](#)

[Ordered](#)

[Changeable](#)

[Allow Duplicates](#)

[Example](#)

[List Length](#)

[Example](#)

[List Items - Data Types](#)

[Example](#)

[Example](#)

[type\(\)](#)

[Example](#)

[The list\(\) Constructor](#)

[Example](#)

[Python Collections \(Arrays\)](#)

[Python - Access List Items](#)

[Access Items](#)

[Example](#)

[Negative Indexing](#)

[Example](#)

[Range of Indexes](#)

[Example](#)

[Example](#)

[Example](#)

[Range of Negative Indexes](#)

[Example](#)

[Check if Item Exists](#)

[Example](#)

[Python - Change List Items](#)

[Change Item Value](#)

[Example](#)

[Change a Range of Item Values](#)

[Example](#)

[Example](#)

[Example](#)

[Insert Items](#)

[Example](#)

[Python - Add List Items](#)

[Append Items](#)

[Example](#)

[Insert Items](#)

[Example](#)

[Extend List](#)

[Example](#)

[Add Any Iterable](#)

[Example](#)

[Python - Remove List Items](#)

[Remove Specified Item](#)

[Example](#)

[Remove Specified Index](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Clear the List](#)

[Example](#)

[Python - Loop Lists](#)

[Loop Through a List](#)

[Example](#)

[Loop Through the Index Numbers](#)

[Example](#)

[Using a While Loop](#)

[Example](#)

[Looping Using List Comprehensive](#)

[Example](#)

[Python - List Comprehension](#)

[List Comprehension](#)

[Example](#)

[Example](#)

[The Syntax](#)

[Condition](#)

[Example](#)

[Example](#)

[Iterable](#)
[Example](#)
[Example](#)
[Expression](#)
[Example](#)
[Example](#)
[Example](#)

[Python - Sort Lists](#)

[Sort List Alphanumerically](#)

[Example](#)
[Example](#)

[Sort Descending](#)

[Example](#)
[Example](#)

[Customize Sort Function](#)

[Example](#)

[Case Insensitive Sort](#)

[Example](#)
[Example](#)

[Reverse Order](#)

[Example](#)

[Python - Copy Lists](#)

[Copy a List](#)

[Example](#)

[Example](#)

[Python - Join Lists](#)

[Join Two Lists](#)

[Example](#)
[Example](#)
[Example](#)

[Python - List Methods](#)

[List Methods](#)

Python List Exercises

[Test Yourself With Exercises](#)

[Exercise:](#)

Python Tuples

[Tuple](#)

[Example](#)

[Tuple Items](#)

[Ordered](#)

[Unchangeable](#)

[Allow Duplicates](#)

[Example](#)

[Tuple Length](#)

[Example](#)

[Create Tuple With One Item](#)

[Example](#)

[Tuple Items - Data Types](#)

[Example](#)

[Example](#)

[type\(\)](#)

[Example](#)

[The tuple\(\) Constructor](#)

[Example](#)

[Python Collections \(Arrays\)](#)

Python - Access Tuple Items

[Access Tuple Items](#)

[Example](#)

[Negative Indexing](#)

[Example](#)

[Range of Indexes](#)

[Example](#)

[Example](#)

[Example](#)

[Range of Negative Indexes](#)

[Example](#)

[Check if Item Exists](#)

[Example](#)

[Python - Update Tuples](#)

[Change Tuple Values](#)

[Example](#)

[Example](#)

[Example](#)

[Remove Items](#)

[Example](#)

[Example](#)

[Python - Unpack Tuples](#)

[Unpacking a Tuple](#)

[Example](#)

[Example](#)

[Using Asterix*](#)

[Example](#)

[Example](#)

[Python - Loop Tuples](#)

[Loop Through a Tuple](#)

[Example](#)

[Loop Through the Index Numbers](#)

[Example](#)

[Using a While Loop](#)

[Example](#)

[Python - Join Tuples](#)

[Join Two Tuples](#)

[Example](#)

[Multiply Tuples](#)

[Example](#)

[Python - Tuple Methods](#)

[Tuple Methods](#)

[Python Sets](#)

[Set](#)

[Example](#)

[Set Items](#)

[Unordered](#)

[Unchangeable](#)

[Duplicates Not Allowed](#)

[Example](#)

[Get the Length of a Set](#)

[Example](#)

[Set Items - Data Types](#)

[Example](#)

[Example](#)

[type\(\)](#)

[Example](#)

[The set\(\) Constructor](#)

[Example](#)

[Python Collections \(Arrays\)](#)

[Python - Add Set Items](#)

[Add Items](#)

[Example](#)

[Add Sets](#)

[Example](#)

[Add Any Iterable](#)

[Example](#)

[Python - Remove Set Items](#)

[Remove Item](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Python - Loop Sets](#)

[Loop Items](#)

[Example](#)

[Python - Join Sets](#)

[Join Two Sets](#)

[Example](#)

[Example](#)

[Keep ONLY the Duplicates](#)

[Example](#)

[Example](#)

[Keep All, But NOT the Duplicates](#)

[Example](#)

[Example](#)

[Python - Set Methods](#)

[Python Dictionaries](#)

[Dictionary](#)

[Example](#)

[Dictionary Items](#)

[Example](#)

[Unordered](#)

[Changeable](#)

[Duplicates Not Allowed](#)

[Example](#)

[Dictionary Length](#)

[Example](#)

[Dictionary Items - Data Types](#)

[Example](#)

[type\(\)](#)

[Example](#)

[Python Collections \(Arrays\)](#)

[Python - Access Dictionary Items](#)

[Accessing Items](#)

[Example](#)

[Example](#)

[Get Keys](#)

[Example](#)

[Example](#)

[Get Values](#)

[Example](#)

[Example](#)

[Get Items](#)

[Example](#)

[Example](#)

[Check if Key Exists](#)

[Example](#)

[Python - Change Dictionary Items](#)

[Change Values](#)

[Example](#)

[Update Dictionary](#)

[Example](#)

[Python - Add Dictionary Items](#)

[Adding Items](#)

[Example](#)

[Update Dictionary](#)

[Example](#)

[Python - Remove Dictionary Items](#)

[Removing Items](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Python - Loop Dictionaries](#)

[Loop Through a Dictionary](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Python - Copy Dictionaries](#)

[Copy a Dictionary](#)

[Example](#)

[Example](#)

[Python - Nested Dictionaries](#)

[Nested Dictionaries](#)

[Example](#)

[Example](#)

[Python Dictionary Methods](#)

[Dictionary Methods](#)

[Python If ... Else](#)

[Python Conditions and If statements](#)

[Example](#)

[Indentation](#)

[Example](#)

[Elif](#)

[Example](#)

[Else](#)

[Example](#)

[Example](#)

[Short Hand If](#)

[Example](#)

[Short Hand If ... Else](#)

[Example](#)

[Example](#)

[And](#)

[Example](#)

[Or](#)

[Example](#)

[Nested If](#)

[Example](#)

[The pass Statement](#)

[Example](#)

[Python While Loops](#)

[Python Loops](#)

[The while Loop](#)

[Example](#)

[The break Statement](#)

[Example](#)

[The continue Statement](#)

[Example](#)

[The else Statement](#)

[Example](#)

[Python For Loops](#)

[Example](#)

[Looping Through a String](#)

[Example](#)

[The break Statement](#)

[Example](#)

[Example](#)

[The continue Statement](#)

[Example](#)

[The range\(\) Function](#)

[Example](#)

[Example](#)

[Example](#)

[Else in For Loop](#)

[Example](#)

[Nested Loops](#)

[Example](#)

[The pass Statement](#)

[Example](#)

[Python Functions](#)

[Creating a Function](#)

[Example](#)

[Calling a Function](#)

[Example](#)

[Arguments](#)

[Example](#)

[Parameters or Arguments?](#)

[Number of Arguments](#)

[Example](#)

[Example](#)

[Arbitrary Arguments, *args](#)

[Example](#)

[Keyword Arguments](#)

[Example](#)

[Arbitrary Keyword Arguments, **kwargs](#)

[Example](#)

[Default Parameter Value](#)

[Example](#)

[Passing a List as an Argument](#)

[Example](#)

[Return Values](#)

[Example](#)

[The pass Statement](#)

[Example](#)

[Recursion](#)

[Example](#)

[Syntax](#)

[Example](#)

[Example](#)

[Example](#)

[Why Use Lambda Functions?](#)

[Example](#)

[Example](#)

[Example](#)

[Arrays](#)

[Example](#)

[What is an Array?](#)

[Access the Elements of an Array](#)

[Example](#)

[Example](#)

[The Length of an Array](#)

[Example](#)

[Looping Array Elements](#)

[Example](#)

[Adding Array Elements](#)

[Example](#)

[Removing Array Elements](#)

[Example](#)

[Example](#)

[Array Methods](#)

[Python Classes/Objects](#)

[Create a Class](#)

[Example](#)

[Create Object](#)

[Example](#)

[The __init__ Function](#)

[Example](#)

[Object Methods](#)

[Example](#)

[The self Parameter](#)

[Example](#)

[Modify Object Properties](#)

[Example](#)

[Delete Object Properties](#)

[Example](#)

[Delete Objects](#)

[Example](#)

[The pass Statement](#)

[Example](#)

[Create a Parent Class](#)

[Example](#)

[Create a Child Class](#)

[Example](#)

[Example](#)

[Add the __init__\(\) Function](#)

[Example](#)

[Example](#)

[Use the super\(\) Function](#)

[Example](#)

[Add Properties](#)

[Example](#)

[Example](#)

[Add Methods](#)

[Example](#)

[Iterator vs Iterable](#)

[Example](#)

[Example](#)

[Looping Through an Iterator](#)

[Example](#)

[Example](#)

[Create an Iterator](#)

[Example](#)

[StopIteration](#)

[Example](#)

[Local Scope](#)

[Example](#)

[Function Inside Function](#)

[Example](#)

[Global Scope](#)

[Example](#)

[Naming Variables](#)

[Example](#)

[Global Keyword](#)

[Example](#)

[Example](#)

[What is a Module?](#)

[Create a Module](#)

[Example](#)

[Use a Module](#)

[Example](#)

[Variables in Module](#)

[Example](#)

[Example](#)

[Naming a Module](#)

[Re-naming a Module](#)

[Example](#)

[Built-in Modules](#)

[Example](#)

[Using the dir\(\) Function](#)

[Example](#)

[Import From Module](#)

[Example](#)

[Example](#)

[Python Dates](#)

[Example](#)

[Date Output](#)

[Example](#)

[Creating Date Objects](#)

[Example](#)

[The strftime\(\) Method](#)

[Example](#)

[Built-in Math Functions](#)

[Example](#)

[Example](#)

[Example](#)

[The Math Module](#)

[Example](#)

[Example](#)

[Example](#)

[Complete Math Module Reference](#)

[JSON in Python](#)

[Example](#)

[Parse JSON - Convert from JSON to Python](#)

[Example](#)

[Convert from Python to JSON](#)

[Example](#)

[Example](#)

[Example](#)

[Format the Result](#)

[Example](#)

[Example](#)

[Order the Result](#)

[Example](#)

[RegEx Module](#)

[RegEx in Python](#)

[Example](#)

[RegEx Functions](#)

[Metacharacters](#)

[Special Sequences](#)

[Sets](#)

[The findall\(\) Function](#)

[Example](#)

[Example](#)

[The search\(\) Function](#)

[Example](#)

[Example](#)

[The split\(\) Function](#)

[Example](#)

[Example](#)

[The sub\(\) Function](#)

[Example](#)

[Example](#)

[Match Object](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[What is PIP?](#)

[What is a Package?](#)

[Check if PIP is Installed](#)

[Example](#)

[Download a Package](#)

[Example](#)

[Using a Package](#)

[Example](#)

[Remove a Package](#)

[Example](#)

[List Packages](#)

[Example](#)

[Exception Handling](#)

[Example](#)

[Example](#)
[Many Exceptions](#)
 [Example](#)
 [Else](#)
 [Example](#)
 [Finally](#)
 [Example](#)
 [Example](#)
[Raise an exception](#)
 [Example](#)
 [Example](#)
[User Input](#)
 [Python 3.6](#)
 [Python 2.7](#)
[String format\(\)](#)
 [Example](#)
 [Example](#)
[Multiple Values](#)
 [Example](#)
[Index Numbers](#)
 [Example](#)
 [Example](#)
[Named Indexes](#)
 [Example](#)
[Python File Open](#)
 [File Handling](#)
 [Syntax](#)
[Open a File on the Server](#)
 [Example](#)
 [Example](#)
[Read Only Parts of the File](#)
 [Example](#)

[Read Lines](#)

[Example](#)

[Example](#)

[Example](#)

[Close Files](#)

[Example](#)

[Write to an Existing File](#)

[Example](#)

[Example](#)

[Create a New File](#)

[Example](#)

[Example](#)

[Delete a File](#)

[Example](#)

[Check if File exist:](#)

[Example](#)

[Delete Folder](#)

[Example](#)

[What is NumPy?](#)

[Why Use NumPy?](#)

[Why is NumPy Faster Than Lists?](#)

[Which Language is NumPy written in?](#)

[Installation of NumPy](#)

[Import NumPy](#)

[Example](#)

[NumPy as np](#)

[Example](#)

[Checking NumPy Version](#)

[Example](#)

[Create a NumPy ndarray Object](#)

[Example](#)

[Example](#)

Dimensions in Arrays

0-D Arrays

Example

1-D Arrays

Example

2-D Arrays

Example

3-D arrays

Example

Check Number of Dimensions?

Example

Higher Dimensional Arrays

Example

Access Array Elements

Example

Example

Example

Access 2-D Arrays

Example

Example

Access 3-D Arrays

Example

Example Explained

Negative Indexing

Example

Slicing arrays

Example

Example

Example

Negative Slicing

Example

STEP

[Example](#)

[Example](#)

[Slicing 2-D Arrays](#)

[Example](#)

[Example](#)

[Example](#)

[Data Types in Python](#)

[Data Types in NumPy](#)

[Checking the Data Type of an Array](#)

[Example](#)

[Example](#)

[Creating Arrays With a Defined Data Type](#)

[Example](#)

[Example](#)

[What if a Value Can Not Be Converted?](#)

[Example](#)

[Converting Data Type on Existing Arrays](#)

[Example](#)

[Example](#)

[Example](#)

[The Difference Between Copy and View](#)

[COPY:](#)

[Example](#)

[VIEW:](#)

[Example](#)

[Make Changes in the VIEW:](#)

[Example](#)

[Check if Array Owns it's Data](#)

[Example](#)

[Shape of an Array](#)

[Get the Shape of an Array](#)

[Example](#)

[Example](#)

[What does the shape tuple represent?](#)

[Reshaping arrays](#)

[Reshape From 1-D to 2-D](#)

[Example](#)

[Reshape From 1-D to 3-D](#)

[Example](#)

[Can We Reshape Into any Shape?](#)

[Example](#)

[Returns Copy or View?](#)

[Example](#)

[Unknown Dimension](#)

[Example](#)

[Flattening the arrays](#)

[Example](#)

[Iterating Arrays](#)

[Example](#)

[Iterating 2-D Arrays](#)

[Example](#)

[Example](#)

[Iterating 3-D Arrays](#)

[Example](#)

[Example](#)

[Iterating Arrays Using nditer\(\)](#)

[Iterating on Each Scalar Element](#)

[Example](#)

[Iterating Array With Different Data Types](#)

[Example](#)

[Iterating With Different Step Size](#)

[Example](#)

[Enumerated Iteration Using ndenumerate\(\)](#)

[Example](#)

[Example](#)

[Joining NumPy Arrays](#)

[Example](#)

[Example](#)

[Joining Arrays Using Stack Functions](#)

[Example](#)

[Stacking Along Rows](#)

[Example](#)

[Stacking Along Columns](#)

[Example](#)

[Stacking Along Height \(depth\)](#)

[Example](#)

[Splitting NumPy Arrays](#)

[Example](#)

[Example](#)

[Split Into Arrays](#)

[Example](#)

[Splitting 2-D Arrays](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Searching Arrays](#)

[Example](#)

[Example](#)

[Example](#)

[Search Sorted](#)

[Example](#)

[Search From the Right Side](#)

[Example](#)

[Multiple Values](#)

[Example](#)

[Sorting Arrays](#)

[Example](#)

[Example](#)

[Example](#)

[Sorting a 2-D Array.](#)

[Example](#)

[Filtering Arrays](#)

[Example](#)

[Creating the Filter Array](#)

[Example](#)

[Example](#)

[Creating Filter Directly From Array](#)

[Example](#)

[Example](#)

[What is a Random Number?](#)

[Pseudo Random and True Random.](#)

[Generate Random Number](#)

[Example](#)

[Generate Random Float](#)

[Example](#)

[Generate Random Array](#)

[Integers](#)

[Example](#)

[Example](#)

[FLOATS](#)

[Example](#)

[Example](#)

[Generate Random Number From Array](#)

[Example](#)

[Example](#)

[What is Data Distribution?](#)

[Random Distribution](#)

[Example](#)

[Example](#)

[What are ufuncs?](#)

[Why use ufuncs?](#)

What is Vectorization?

[Add the Elements of Two Lists](#)

[Example](#)

[Example](#)

How To Create Your Own ufunc

[Example](#)

Check if a Function is a ufunc

[Example](#)

[Example](#)

[Example](#)

[Example](#)

What is Matplotlib?

Matplotlib Getting Started

[Installation of Matplotlib](#)

[Import Matplotlib](#)

[Checking Matplotlib Version](#)

[Example](#)

Matplotlib Pyplot

[Pyplot](#)

[Example](#)

[Plotting x and y points](#)

[Example](#)

[Plotting Without Line](#)

[Example](#)

[Multiple Points](#)

[Example](#)

[Default X-Points](#)

[Example](#)

Matplotlib Markers

[Markers](#)

[Example](#)

[Example](#)

[Marker Reference](#)

[Format Strings fmt](#)

[Example](#)

[Line Reference](#)

[Color Reference](#)

[Marker Size](#)

[Example](#)

[Marker Color](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Line](#)

[Example](#)

[Example](#)

[Result:](#)

[Shorter Syntax](#)

[Example](#)

[Line Styles](#)

[Line Color](#)

[Example](#)

[Example](#)

[Example](#)

[Line Width](#)

[Example](#)

[Multiple Lines](#)

[Example](#)

[Example](#)

[Display Multiple Plots](#)

[Example](#)

[The subplots\(\) Function](#)

[Example](#)

[Example](#)

[Title](#)

[Example](#)

[Super Title](#)

[Example](#)

[Compare Plots](#)

[Example](#)

[Colors](#)

[Example](#)

[Color Each Dot](#)

[Example](#)

[ColorMap](#)

[How to Use the ColorMap](#)

[Example](#)

[Example](#)

[Available ColorMaps](#)

[Size](#)

[Example](#)

[Alpha](#)

[Example](#)

[Combine Color Size and Alpha](#)

[Example](#)

[Matplotlib Bars](#)

[Creating Bars](#)

[Example](#)

[Example](#)

[Horizontal Bars](#)

[Example](#)

[Bar Color](#)

[Example](#)

[Color Names](#)

[Example](#)

[Color Hex](#)

[Example](#)

[Bar Width](#)

[Example](#)

[Bar Height](#)

[Example](#)

[Matplotlib Histograms](#)

[Histogram](#)

[Create Histogram](#)

[Example](#)

[Example](#)

[Creating Pie Charts](#)

[Example](#)

[Labels](#)

[Example](#)

[Start Angle](#)

[Example](#)

[Explode](#)

[Example](#)

[Shadow](#)

[Example](#)

[Colors](#)

[Example](#)

[Legend](#)

[Example](#)

[Legend With Header](#)

[Example](#)

[What is SciPy?](#)

[Why Use SciPy?](#)

[Which Language is SciPy Written in?](#)

[SciPy Getting Started](#)

[Installation of SciPy](#)

[Import SciPy](#)

[Example](#)

[Checking SciPy Version](#)

[Example](#)

[Constants in SciPy](#)

[Example](#)

[Constant Units](#)

[Example](#)

[Unit Categories](#)

[Metric \(SI\) Prefixes:](#)

[Example](#)

[Binary Prefixes:](#)

[Example](#)

[Mass:](#)

[Example](#)

[Angle:](#)

[Example](#)

[Time:](#)

[Example](#)

[Length:](#)

[Example](#)

[Pressure:](#)

[Example](#)

[Area:](#)

[Example](#)

[Volume:](#)

[Example](#)

[Speed:](#)

[Example](#)

[Temperature:](#)

[Example](#)

[Energy:](#)

[Example](#)

[Power:](#)

[Example](#)

[Force:](#)

[Example](#)

[Optimizers in SciPy](#)

[Optimizing Functions](#)

[Roots of an Equation](#)

[Example](#)

[Example](#)

[Minimizing a Function](#)

[Finding Minima](#)

[Example](#)

[What is Sparse Data](#)

[How to Work With Sparse Data](#)

[CSR Matrix](#)

[Example](#)

[Sparse Matrix Methods](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[Example](#)

[SciPy Graphs](#)

[Working with Graphs](#)

[Adjacency Matrix](#)

[Connected Components](#)

[Example](#)

[Dijkstra](#)

[Example](#)

[Floyd Warshall](#)

[Example](#)

[Bellman Ford](#)

[Example](#)

[Depth First Order](#)

[Example](#)

[Breadth First Order](#)

[Example](#)

[SciPy Spatial Data](#)

[Working with Spatial Data](#)

[Triangulation](#)

[Example](#)

[Convex Hull](#)

[Example](#)

[KDTrees](#)

[Example](#)

[Distance Matrix](#)

[Euclidean Distance](#)

[Example](#)

[Cityblock Distance \(Manhattan Distance\)](#)

[Example](#)

[Cosine Distance](#)

[Example](#)

[Hamming Distance](#)

[Example](#)

[SciPy Matlab Arrays](#)

[Working With Matlab Arrays](#)

[Exporting Data in Matlab Format](#)

[Example](#)

[Import Data from Matlab Format](#)

[Example](#)

[Example](#)

[Example](#)

[SciPy Interpolation](#)

[What is Interpolation?](#)

[How to Implement it in SciPy?](#)

[1D Interpolation](#)

[Example](#)

[Result:](#)

[Spline Interpolation](#)

[Example](#)

[Result:](#)

[Interpolation with Radial Basis Function](#)

[Example](#)

[Result:](#)

[SciPy Statistical Significance Tests](#)

[What is Statistical Significance Test?](#)

[Hypothesis in Statistics](#)

[Null Hypothesis](#)

[Alternate Hypothesis](#)

[One tailed test](#)

[Two tailed test](#)

[Alpha value](#)

[P value](#)

[T-Test](#)

[Example](#)

[Result:](#)

[Example](#)

[Result:](#)

[KS-Test](#)

[Example](#)

[Result:](#)

[Statistical Description of Data](#)

[Example](#)

[Result:](#)

[Normality Tests \(Skewness and Kurtosis\)](#)

[Skewness:](#)

[Kurtosis:](#)

[Example](#)

[Result:](#)

[Example](#)

[Result:](#)

[Where To Start?](#)

[Data Set](#)

[Data Types](#)

[Mean, Median, and Mode](#)

[Mean](#)

[Example](#)

[Median](#)

[Example](#)

[Example](#)

[Mode](#)

[Example](#)

[Chapter Summary](#)

[Machine Learning - Standard Deviation](#)

[What is Standard Deviation?](#)

[Example](#)

[Example](#)

[Variance](#)

[Example](#)

[Standard Deviation](#)

[Example](#)

[Symbols](#)

[Chapter Summary](#)

[Machine Learning - Percentiles](#)

[What are Percentiles?](#)

[Example](#)

[Example](#)

[Machine Learning - Data Distribution](#)

[Data Distribution](#)

[How Can we Get Big Data Sets?](#)

[Example](#)

[Histogram](#)

[Example](#)

[Histogram Explained](#)

[Big Data Distributions](#)

[Example](#)

[Machine Learning - Normal Data Distribution](#)

[Normal Data Distribution](#)

[Example](#)

[Histogram Explained](#)

[Machine Learning - Scatter Plot](#)

[Scatter Plot](#)

[Example](#)

[Random Data Distributions](#)

[Example](#)

[Scatter Plot Explained](#)

[Machine Learning - Linear Regression](#)

[Regression](#)

[Linear Regression](#)

[How Does it Work?](#)

[Example](#)

[Example](#)

[Example Explained](#)

[R for Relationship](#)

[Example](#)

[Predict Future Values](#)

[Example](#)

[Bad Fit?](#)

[Example](#)

[Example](#)

[Machine Learning - Polynomial Regression](#)

[Polynomial Regression](#)

[How Does it Work?](#)

[Example](#)

[Example Explained](#)

[R-Squared](#)

[Example](#)

[Predict Future Values](#)

[Example](#)

[Bad Fit?](#)

[Example](#)

[Example](#)

[Machine Learning - Multiple Regression](#)

[Multiple Regression](#)

[How Does it Work?](#)

[Example](#)

[Result:](#)

[Coefficient](#)

[Example](#)

[Result:](#)

[Result Explained](#)

[Example](#)

[Result:](#)

[Machine Learning - Scale](#)

[Scale Features](#)

[Example](#)

[Result:](#)

[Predict CO2 Values](#)

[Example](#)

[Result:](#)

[Evaluate Your Model](#)

[What is Train/Test](#)

[Start With a Data Set](#)

Example

Result:

Split Into Train/Test

Display the Training Set

Example

Result:

Display the Testing Set

Example

Result:

Fit the Data Set

Example

R2

Example

Bring in the Testing Set

Example

Predict Values

Example

Decision Tree

How Does it Work?

Example

Example

Example

Example

Result Explained

Rank

Gini

True - 5 Comedians End Here:

False - 8 Comedians Continue:

Nationality

True - 4 Comedians Continue:

Age

False - 4 Comedians End Here:

True - 2 Comedians End Here:

[False - 2 Comedians Continue:](#)

[Experience](#)

[True - 1 Comedian Ends Here:](#)

[False - 1 Comedian Ends Here:](#)

[Predict Values](#)

[Example](#)

[Example](#)

[Different Results](#)

[Install MySQL Driver](#)

[Test MySQL Connector](#)

[Create Connection](#)

[Python MySQL Create Database](#)

[Creating a Database](#)

[Example](#)

[Check if Database Exists](#)

[Example](#)

[Example](#)

[Python MySQL Create Table](#)

[Creating a Table](#)

[Example](#)

[Check if Table Exists](#)

[Example](#)

[Primary Key](#)

[Example](#)

[Example](#)

[Python MySQL Insert Into Table](#)

[Insert Into Table](#)

[Example](#)

[Insert Multiple Rows](#)

[Example](#)

[Get Inserted ID](#)

[Example](#)

[Python MySQL Select From](#)

[Select From a Table](#)

[Example](#)

[Selecting Columns](#)

[Example](#)

[Using the fetchone\(\) Method](#)

[Example](#)

[Python MySQL Where](#)

[Select With a Filter](#)

[Example](#)

[Wildcard Characters](#)

[Example](#)

[Prevent SQL Injection](#)

[Example](#)

[Python MySQL Delete From By](#)

[Delete Record](#)

[Example](#)

[Prevent SQL Injection](#)

[Example](#)

[Python MySQL Drop Table](#)

[Delete a Table](#)

[Example](#)

[Drop Only if Exist](#)

[Example](#)

[Python MySQL Update Table](#)

[Update Table](#)

[Example](#)

[Prevent SQL Injection](#)

[Example](#)

[Python MySQL Limit](#)

[Limit the Result](#)

[Example](#)

[Start From Another Position](#)

[Example](#)

[Python MySQL Join](#)

[Join Two or More Tables](#)

[users](#)

[products](#)

[Example](#)

[LEFT JOIN](#)

[Example](#)

[RIGHT JOIN](#)

[Example](#)

[MongoDB](#)

[PyMongo](#)

[Test PyMongo](#)

[Creating a Database](#)

[Example](#)

[Check if Database Exists](#)

[Example](#)

[Example](#)

[Creating a Collection](#)

[Example](#)

[Check if Collection Exists](#)

[Example](#)

[Example](#)

[Insert Into Collection](#)

[Example](#)

[Return the _id Field](#)

[Example](#)

[Insert Multiple Documents](#)

[Example](#)

[Insert Multiple Documents, with Specified IDs](#)

[Example](#)

[Find One](#)

[Example](#)

[Find All](#)

[Example](#)

[Return Only Some Fields](#)

[Example](#)

[Example](#)

[Example](#)

[Filter the Result](#)

[Example](#)

[Advanced Query](#)

[Example](#)

[Filter With Regular Expressions](#)

[Example](#)

[Sort the Result](#)

[Example](#)

[Sort Descending](#)

[Example](#)

[Delete Document](#)

[Example](#)

[Delete Many Documents](#)

[Example](#)

[Delete All Documents in a Collection](#)

[Example](#)

[Delete Collection](#)

[Example](#)

[Delete Collection](#)

[Example](#)

[Limit the Result](#)

[Customers](#)

[Example](#)

[Python Built in Functions](#)

[Python String Methods](#)

[Python List/Array Methods](#)

[Python Dictionary Methods](#)

[Python Set Methods](#)

[Python File Methods](#)

[Python Keywords](#)

[Python Built-in Exceptions](#)

[Built-in Exceptions](#)

[Module Reference](#)

[Python Requests Module](#)

[Example](#)

[Definition and Usage](#)

[Download and Install the Requests Module](#)

[Syntax](#)

[Methods](#)

[Python statistics Module](#)

[Python statistics Module](#)

[Python math Module](#)

[Math Methods](#)

[Python cmath Module](#)

[Example](#)

[Example Explained](#)

[A List with Duplicates](#)

[Create a Dictionary](#)

[Convert Into a List](#)

[Print the List](#)

[Create a Function](#)

[Example](#)

[Example Explained](#)

[Create a Function](#)

[Create a Dictionary](#)

[Convert Into a List](#)

[Return List](#)

[Call the Function](#)

[Print the Result](#)

[How to Reverse a String in Python](#)

[Example](#)

[Example Explained](#)

[The String to Reverse](#)

[Slice the String](#)

[Print the List](#)

[Create a Function](#)

[Example](#)

[Example Explained](#)

[Create a Function](#)

[Slice the String](#)

[Return the String](#)

[Call the Function](#)

[Print the Result](#)

[How to Add Two Numbers in Python](#)

[Example](#)

[Add Two Numbers with User Input](#)

[Example](#)

Python Introduction

Python is a popular programming language. Which 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?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has 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.

Python Getting Started

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
```

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.

Example:helloworld.py

```
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 program.

The Python Command Line

To test a short amount of code in python sometimes it is 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:d48ebeb, 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()
```

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:

Example

Syntax Error:

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

The number of spaces is up to you as a programmer, but it has to be at least one.

Example

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

```
print ("Five is greater than two!" )
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Example

Syntax Error:

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

Python Variables

In Python, variables are created when you assign a value to it:

Example

Variables in Python:

```
x = 5
```

```
y = "Hello, World!"
```

Python has no command for declaring a variable.

You will learn more about variables in the Python chapter.

Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

Example

Comments in Python:

```
#This is a comment.
```

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

Python Comments

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 starts with a `#` , and Python will ignore them:

Example

```
#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:

Example

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

Comments does not have to be text to explain the code, it can also be used to prevent Python from executing code:

Example

```
#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:

Example

```
#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:

Example

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

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

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.

Example

```
x = 5
```

```
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.

Example

```
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.

Example

```
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.

Example

```
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:

Example

```
x = "John"
```

is the same as

```
x = 'John'
```

Case-Sensitive

Variable names are case-sensitive.

Example

This will create two variables:

```
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)

Example

Legal variable names:

myvar = "John"

my_var = "John"

_my_var = "John"

myVar = "John"

MYVAR = "John"

myvar2 = "John"

Example

Illegal variable names:

2myvar = "John"

my-var = "John"

my var = "John"

Remember that variable names are case-sensitive

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 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 to extract the values into variables. This is called *unpacking* .

Example

Unpack a list:

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

```
x, y, z = fruits
```

```
print (x)
```

```
print (y)
```

```
print (z)
```

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"
```

```
z = x + y
```

```
print (z)
```

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

Example

```
x = 5
```

```
y = 10
```

```
print (x + y)
```

If you try to combine a string and a number, Python will give you an error :

Example

```
x = 5
```

```
y = "John"
```

```
print (x + y)
```

Python - Global Variables Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Example

Create a variable outside of a function, and use it inside the function

```
x = "awesome"

def myfunc():
    print ("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

Create a variable inside a function, with the same name as the global variable

```
x = "awesome"

def myfunc():
    x = "fantastic"
    print ("Python is " + x)

myfunc()

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

The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the `global` keyword.

Example

If you use the `global` keyword, the variable belongs to the global scope:

```
def myfunc():
```

```
    global x
```

```
    x = "fantastic"
```

```
myfunc()
```

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

Also, use the `global` keyword if you want to change 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 = "awesome"

def myfunc():
    global x

    x = "fantastic"

myfunc()

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

Python Data Types

Built-in Data Types

In programming, 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))
```

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

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

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))
```

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

```
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.

Random Number

Python does not have a `random()` function to make a random number, but Python has a built-in module called `random` that can be used to make random numbers:

Example

Import the random module, and display a random number between 1 and 9:

```
import random
```

```
print (random.randrange(1 , 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 rounding down to the previous whole number), 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
```

Example

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
```

Example

Strings:

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

```
y = str (2) # y will be '2'
```

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

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:

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** loop.

Example

Loop through the letters in the word "banana":

```
for x in "banana" :
```

```
    print (x)
```

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:

```
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 index 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 ])
```

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 upper case:

```
a = "Hello, World!"
```

```
print (a.upper())
```

Lower Case

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 or 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!']
```

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
```

```
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

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

`zfill()` Fills the string with a specified number

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

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

In fact, there are not many values that evaluates 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 **Fals e** , and that is if you have an object that is made from a class with a **__len__** function that returns **0** or **Fals e** :

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

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 returns 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 ))
```

Python Operators

Python Operators

Operators are used to perform 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 in the following groups:

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

Python Arithmetic Operators

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

Operator	Name	Example
<code>+</code>	Addition	<code>x + y</code>
<code>-</code>	Subtraction	<code>x - y</code>
<code>*</code>	Multiplication	<code>x * y</code>

/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Python Assignment Operators

Assignment operators are used to assign values to variables:

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

<code>&=</code>	<code>x &= 3</code>	<code>x = x & 3</code>
<code> =</code>	<code>x = 3</code>	<code>x = x 3</code>
<code>^=</code>	<code>x ^= 3</code>	<code>x = x ^ 3</code>
<code>>>=</code>	<code>x >>= 3</code>	<code>x = x >> 3</code>
<code><<=</code>	<code>x <<= 3</code>	<code>x = x << 3</code>

Python Comparison Operators

Comparison operators are used to compare two values:

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

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	$x < 5 \text{ and } x < 10$
or	Returns True if one of the statements is true	$x < 5 \text{ or } x < 4$
not	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

Python Identity Operators

Identity operators are used to compare 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	<code>x is y</code>
is not	Returns True if both variables are not the same object	<code>x is not y</code>

Python Membership Operators

Membership operators are used to test 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	<code>x in y</code>

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

Python Bitwise Operators

Bitwise operators are used to compare (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

Python Lists

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

List

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:

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':

```
<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 indexed. No duplicate members.
- Dictionary is a collection which is unordered 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 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" ]
```

```
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 included, "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" and 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 included.
"mango" (-1):

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

```
print (thislist[-4 :-1 ])
```

Check if Item Exists

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)
```

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)
```

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" ]
```

```
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 Specified Item

The `remove()` method removes the specified item.

Example

Remove "banana":

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

Remove Specified Index

The `pop()` method removes the specified index.

Example

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

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

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 Comprehensive

List Comprehensive offers the shortest syntax for looping through lists:

Example

A short hand `for` loop that will print all items in a list:

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

```
[print(x) for x in thislist]
```

Learn more about list comprehensive in the next chapter: List Comprehensive.

Python - List Comprehension

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list 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 accepts only the items that evaluate to **True**.

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:

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 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 ]  
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 )
```

```
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 after 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 = list 1` , because: `list 2` will only be a *reference* to `list 1` , and changes made in `list 1` will automatically also be made in `list 2` .

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

Join Two Lists

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

One of the easiest ways are 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 are 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)  
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

Python List Exercises

Test Yourself With Exercises

Now you have learned a lot about lists, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Exercise:

Print the second item in the **fruits** list.

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

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, tuples can have items with the same value:

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 )
```

```
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.
- Set is a collection which is unordered and indexed. No duplicate members.
- Dictionary is a collection which is unordered 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 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 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 tuple:

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

```
print (thistuple[-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 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:

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 **i n** 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)
```

```
y[1 ] = "kiwi"
```

```
x = tuple (y)
```

```
print (x)
```

Add Items

Once a tuple is created, you cannot add items to it.

Example

You cannot add items to a tuple:

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

```
thistuple.append("orange" ) # This will raise an error
```

```
print (thistuple)
```

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)
```

Remove Items

Note: You cannot remove items in a tuple.

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:

```
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 asterix to collect the remaining values as a list.

Using Asterix *

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 asterix 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 to the "tropic" variable:

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

```
(green, *tropic, red) = fruits
```

```
print (green)
```

```
print (tropic)
```

```
print (red)
```

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 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:

```
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:

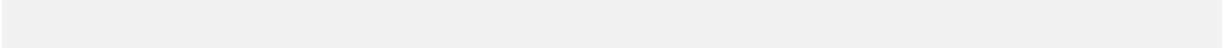
Example

Multiply the fruits tuple by 2:

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

```
mytuple = fruits * 2
```

```
print (mytuple)
```



Python - Tuple Methods

Tuple Methods

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

Met hod	Description
cou nt()	Returns the number of times a specified value occurs in a tuple
inde x()	Searches the tuple for a specified value and returns the position of where it was found

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 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:

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 members.
- Dictionary is a collection which is unordered 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 - 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` and `thisset` into `newset` :

```
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

Add elements of a list to a 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.

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:

```
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

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)
```

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

Method	Description
<code>add()</code>	Adds an element to the set
<code>clear()</code>	Removes all the elements from the set
<code>copy()</code>	Returns a copy of the set
<code>difference()</code>	Returns a set containing the difference between two or more sets
<code>difference_update()</code>	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)
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

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 unordered, changeable and does not allow duplicates.

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 unordered, changeable, and does 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"])
```

Unordered

When we say that dictionaries are unordered, it means that the items 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" ]  
}
```

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 indexed. No duplicate members.
- Dictionary is a collection which is unordered 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:

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

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

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.values()
```

```
print (x) #before the change
```

```
car["year"] = 2020
```

```
print (x) #after the change
```

Get Items

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

Example

```
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

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["year"] = 2020
```

```
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  
}  
  
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:

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

Removing 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()  
  
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

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 = dict 1** , because: **dict 2** will only be a *reference* to **dict 1** , and changes made in **dict 1** will automatically also be made in **dict 2** .

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 = {  
    "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

Dictionary Methods

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

Method	Description
clear()	Removes all the elements from the dictionary
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
)

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 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

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 **eli f** condition is true, so we print to screen that "a and b are equal".

Else

The **els e** 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 **eli f** condition is not true, so we go to the **els e** condition and print to screen that "a is greater than b".

You can also have an `else` without the `elif` :

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

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" )
```

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

Python While Loops

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)  
    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" )
```

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:

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 **continu e** 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)
```

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

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! " )
```

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:

```
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
```

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()
```

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. 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 ])
```

```
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):  
    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

```
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
```

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" )
```

```
tri_recursion(6 )
```

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

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
```

```
mytrippler = myfunc(3 )
```

```
print (mytrippler(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 )
```

```
mytrippler = myfunc(3 )
```

```
print (mydoubler(11 ))
```

```
print (mytrippler(11 ))
```

Use lambda functions when an anonymous function is required for a short period of time.

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

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 `car s` 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 i n` loop to loop through all the elements of an array.

Example

Print each item in the `car s` array:

```
for x in cars:
```

```
    print(x)
```

|

Adding Array Elements

You can use the `append()` method to add an element to an array.

Example

Add one more element to the `car s` array:

```
cars.append("Honda")
```

|

Removing Array Elements

You can use the `pop()` method to remove an element from the array.

Example

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.

Example

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

nd()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
coun t()	Returns the number of elements with the specified value
exten d()	Add the elements of a list (or any iterable), to the end of the current list
inde x()	Returns the index of the first element with the specified value
inser t()	Adds an element at the specified position
pop()	Removes the element at the specified position
remo ve()	Removes the first item with the specified value
rever se()	Reverses the order of the list
sort()	Sorts the list

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

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
```

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.

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):
```

```
    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

`clas s` definitions cannot be empty, but if you for some reason have a `clas s` definition with no content, put in the `pas s` statement to avoid getting an error.

Example

```
class Person:
```

```
    pass
```

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()
```

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):
```

```
    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

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" ,  
          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.

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:

Example

Strings are also iterable objects, containing a sequence of characters:

```
mystr = "banana"
```

```
myit = iter (mystr)
```

```
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

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

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)
```

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.

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:

```
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)
```

Python Modules

What is a Module?

Consider a module to be the same as a code library.

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 = {  
    "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.

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:

```
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"]`

Python Datetime

Python Dates

A date in Python is not a data type of its own, but we can import a module named **datetim e** 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:

2020-12-15 00:09:56.598812

The date contains year, month, day, hour, minute, second, and microsecond.

The **datetim e** 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, 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
-----------	-------------

%a	Wed
----	-----

	Weekday, short version	
%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 of week, 00-53	52
%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	%

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.

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:

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.

Python JSON

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

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
```

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
```

```
# 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)
```

```
[REDACTED]
```

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 ))  
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

Example

Convert a Python object containing all the legal data types:

```
import json
```

```
x = {  
    "name": "John",  
    "age": 30,  
    "married": True,  
    "divorced": False,  
    "children": ("Ann", "Billy"),  
    "pets": None,  
    "cars": [  
        {"model": "BMW 230", "r": 1},  
        {"model": "Ford Edge", "r": 2}  
    ]  
}
```

```
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

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 )
```

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
```

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:

Funct ion	Description
<code>findal l</code>	Returns a list containing all matches
<code>searc h</code>	Returns a Match object if there is a match anywhere in the string
<code>split</code>	Returns a list where the string has been split at each match
<code>sub</code>	Replaces one or many matches with a string

Metacharacters

Metacharacters are characters with a special meaning:

Chara cter	Description	Exampl e
<code>[]</code>	A set of characters	"[a-m]"
<code>\</code>	Signals a special sequence (can also be used to escape special characters)	"\d"
<code>.</code>	Any character (except newline character)	"he..o"

^	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 sta ys"
()	Capture and group	

Special Sequences

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Chara cter	Description	Exam ple
\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	r"\Bain "

	(the "r" in the beginning is making sure that the string is being treated as a "raw string")	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 contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Sets

A set is a set of characters inside a pair of square brackets [] with a special meaning:

Se	Description
[ar]	Returns a match where one of the specified characters (a , r , or n

n]) are present

[a-n] Returns a match for any lower case character, alphabetically between a and n

[^ar-n] Returns a match for any character EXCEPT a , r , and n

[0-123] 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

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:

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:

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 **coun t** 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

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:

```
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.

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
```

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

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))
```

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-0.2.1:
```

```
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
------------	--------

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:

Example

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:

Example

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)
```

```
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:

Example

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:

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.

Example

Raise a `TypeError` if `x` is not an integer:

```
x = "hello"
```

```
if not type(x) is int :
```

```
    raise TypeError("Only integers are allowed")
```

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.

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 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
```

```
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" ))
```

Python File Handling

Python File Open

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 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

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:  
    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" )  
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" )
```

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":

```
import os  
  
os.rmdir("myfolder" )
```

Note: You can only remove *empty* folders.

NumPy Introduction

What is NumPy?

NumPy is a Python library used for working with arrays.

It also has functions for working in 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 **ndarra y** , it provides a lot of supporting functions that make working with **ndarra y** 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 latest CPU architectures.

Which Language is NumPy written in?

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++.

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)
```

NumPy as np

NumPy is usually imported under the `np` alias.

alias: In Python alias 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 ])
```

```
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.

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
```

```
arr = np.array([[1 , 2 , 3 ], [4 , 5 , 6 ]], [[1 , 2 , 3 ], [4 , 5 , 6 ]]])
```

```
print (arr)
```

```
[[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]]
```

Check Number of Dimensions?

NumPy Arrays provides the `ndim` attribute that returns an integer that tells us how many dimensions the array have.

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)
```

```
[1]
```

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)
```

```
print ('number of dimensions :', arr.ndim)
```

```
[1]
```

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

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:

```
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:

5

6

Since we selected 2 , we end up with the third value:

6

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:

```
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 ])
```

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 *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 ])
```

NumPy Data Types

Data Types in Python

By default Python have these data types:

- **string s** - used to represent text data, the text is given under quote marks. eg. "ABCD"
 - **integer r** - used to represent integer numbers. eg. -1, -2, -3
 - **float t** - used to represent real numbers. eg. 1.2, 42.42
 - **boolean n** - used to represent True or False.
 - **complex x** - used to represent a number in complex plain. eg. 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:

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: `dtyp e` 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' )
```

```
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')
```

```
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 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

```
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 its 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.

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=5` 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.

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)
```

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 $3 \times 3 = 9$ 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)
```

Returns 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

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`, `fliph`, `filpd` etc. These fall under 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.

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 $n-1$ th 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)
```

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, lets 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.

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 `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 ]])
```

```
for x in np.nditer(arr[:, ::2]):  
    print (x)
```

Enumerated Iteration Using ndenumerate()

Enumeration means mentioning sequence number of somethings one by one.

Sometimes we require corresponding index of the element while iterating, the `ndenumerate()` method can be used for those usecases.

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.

We pass a sequence of arrays that we want to join to the `concatenate()` function, along with the axis. If 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 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 axis is not explicitly passed it is taken as 0.

Example

```
import numpy as np
```

```
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

```
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 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 ])
```

```
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 split 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 ]])  
  
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.

```
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 alternates 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 )
```

```
print (x)
```

Example

Find the indexes where the values are odd:

```
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:

```
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 left most index is returned, but we can give `side='right'` to return the right most index instead.

Example

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 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 ]])
```

```
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

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 divisible by 2, set the value to True,
    # 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 ])
```

```
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

NumPy offers the `rando m` 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

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
```

```
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 offer methods that returns 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 `rando m` module.

The `choice()` method allows us to specify the probability for each value.

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)

NumPy ufuncs

What are ufuncs?

ufuncs stands for "Universal Functions" and they are NumPy functions that operates on the `ndarra y` 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 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 elements.

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:

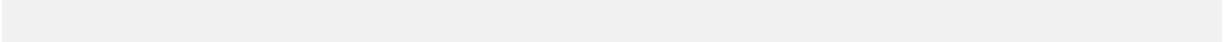
```
import numpy as np
```

```
x = [1 , 2 , 3 , 4 ]
```

```
y = [4 , 5 , 6 , 7 ]
```

```
z = np.add(x, y)
```

```
print (z)
```



Create Your Own ufunc

How To Create Your Own ufunc

To create you 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

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.ufun c` value (or `np.ufun c` 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')
```

Python Matplotlib

What is Matplotlib?

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Matplotlib Getting Started

Installation of Matplotlib

If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.

Install it using this command:

```
C:\Users\Your Name >pip install matplotlib
```

If this command fails, then use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.

Import Matplotlib

Once Matplotlib is installed, import it in your applications by adding the `import module` statement:

```
import matplotlib
```

Now Matplotlib is imported and ready to use:

Checking Matplotlib Version

The version string is stored under `__version__` attribute.

Example

```
import matplotlib
```

```
print (matplotlib.__version__)
```



Note: two underscore characters are used in `__version__`.

Matplotlib Pyplot

Pyplot

Most of the Matplotlib utilities lies under the `pyplo t` submodule, and are usually imported under the `pl t` alias:

```
import matplotlib.pyplot as plt
```

Now the Pyplot package can be referred to as `pl t`.

Example

Draw a line in a diagram from position (0,0) to position (6,250):

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
xpoints = np.array([0 , 6 ])
```

```
ypoints = np.array([0 , 250 ])
```

```
plt.plot(xpoints, ypoints)
```

```
plt.show()
```

Matplotlib Plotting

Plotting x and y points

The `plot()` function is used to draw points (markers) in a diagram.

By default, the `plot()` function draws a line from point to point.

The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the x-axis.

Parameter 2 is an array containing the points on the y-axis.

If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

Example

Draw a line in a diagram from position (1, 3) to position (8, 10):

```
import matplotlib.pyplot as plt  
import numpy as np
```

```
xpoints = np.array([1 , 8 ])
```

```
ypoints = np.array([3 , 10 ])
```

```
plt.plot(xpoints, ypoints)
```

```
plt.show()
```

Plotting Without Line

To plot only the markers, you can use *shortcut string notation* parameter 'o', which means 'rings'.

Example

Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
xpoints = np.array([1 , 8 ])
```

```
ypoints = np.array([3 , 10 ])
```

```
plt.plot(xpoints, ypoints, 'o' )
```

```
plt.show()
```

Multiple Points

You can plot as many points as you like, just make sure you have the same number of points in both axis.

Example

Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

```
import matplotlib.pyplot as plt  
import numpy as np  
  
xpoints = np.array([1 , 2 , 6 , 8 ])  
ypoints = np.array([3 , 8 , 1 , 10 ])  
  
plt.plot(xpoints, ypoints)  
plt.show()
```

Default X-Points

If we do not specify the points in the x-axis, they will get the default values 0, 1, 2, 3, (etc. depending on the length of the y-points).

So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

Example

Plotting without x-points:

```
import matplotlib.pyplot as plt  
import numpy as np
```

```
ypoints = np.array([3 , 8 , 1 , 10 , 5 , 7 ])
```

```
plt.plot(ypoints)
```

```
plt.show()
```

Matplotlib Markers

Markers

You can use the keyword argument `marker` to emphasize each point with a specified marker:

Example

Mark each point with a circle:

```
import matplotlib.pyplot as plt  
import numpy as np
```

```
y = np.array([3, 8, 1, 10])
```

```
plt.plot(y, marker = 'o')  
plt.show()
```

Example

Mark each point with a star:

```
...  
plt.plot(y, marker = '*')  
...
```

Marker Reference

You can choose any of these markers:

Marker	Description
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond
'd'	Diamond (thin)
'p'	Pentagon

'H'	Hexagon
'h'	Hexagon
'v'	Triangle Down
'^'	Triangle Up
Triangle Left	
Triangle Right	
'1'	Tri Down
'2'	Tri Up
'3'	Tri Left
'4'	Tri Right
' '	Vline
'_'	Hline

Format Strings **fmt**

You can also use the *shortcut string notation* parameter to specify the marker.

This parameter is also called **fm t** , and is written with this syntax:

marker | line | color

Example

Mark each point with a circle:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
y = np.array([3, 8, 1, 10])
```

```
plt.plot(y, 'o:r')
```

```
plt.show()
```

The marker value can be anything from the Marker Reference above.

The line value can be one of the following:

Line Reference

Line Syntax	Description
'-	Solid line
:	Dotted line
--	Dashed line

'-'	Dashed/dotted line
-----	--------------------

Note: If you leave out the *line* value in the `fmt` parameter, no line will be plotted.

The short color value can be one of the following:

Color Reference

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Marker Size

You can use the keyword argument `markersize` or the shorter version, `m` to set the size of the markers:

Example

Set the size of the markers to 20:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
ypoints = np.array([3 , 8 , 1 , 10 ])  
  
plt.plot(ypoints, marker = 'o' , ms = 20 )  
plt.show()
```

Marker Color

You can use the keyword argument `markeredgecolor` or the shorter `me c` to set the color of the *edge* of the markers:

Example

Set the EDGE color to red:

```
import matplotlib.pyplot as plt  
import numpy as np
```

```
ypoints = np.array([3 , 8 , 1 , 10 ])  
  
plt.plot(ypoints, marker = 'o' , ms = 20 , mec = 'r' )  
plt.show()
```

You can use the keyword argument `markerfacecolor` or the shorter `mfc` to set the color inside the edge of the markers:

Example

Set the FACE color to red:

```
import matplotlib.pyplot as plt  
  
import numpy as np
```

```
ypoints = np.array([3 , 8 , 1 , 10 ])
```

```
plt.plot(ypoints, marker = 'o' , ms = 20 , mfc = 'r' )  
plt.show()
```

Use *both* the `me c` and `mf c` arguments to color of the entire marker:

Example

Set the color of both the *edge* and the *face* to red:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
ypoints = np.array([3 , 8 , 1 , 10 ])
```

```
plt.plot(ypoints, marker = 'o' , ms = 20 , mec = 'r' , mfc = 'r' )
```

```
plt.show()
```

Or any of the 140 supported color names.

Example

Mark each point with the color named "hotpink":

...

```
plt.plot(ypoints, marker = 'o' , ms = 20 , mec = 'hotpink' , mfc = 'hotpink' )
```

...

Line

You can use the keyword argument `linestyle` , or shorter `ls` , to change the style of the plotted line:

Example

Use a dotted line:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y whole points = np.array([3 , 8 , 1 , 10 ])  
  
plt.plot(y whole points, linestyle = 'dotted' )  
plt.show()
```

Example

Use a dashed line:

```
plt.plot(y whole points, linestyle = 'dashed' )
```

Result:

Shorter Syntax

The line style can be written in a shorter syntax:

linestyl e can be written as `ls` .

dotte d can be written as `:` .

dashe d can be written as `--` .

Example

Shorter syntax:

```
plt.plot(ypoints, ls = ':')
```

Line Styles

You can choose any of these styles:

Style	Or
'solid' (default)	'-'
'dotted'	::
'dashed'	'--'
'dashdot'	'-.'

'None'

" or ''

Line Color

You can use the keyword argument `color` or the shorter `c` to set the color of the line:

Example

Set the line color to red:

```
import matplotlib.pyplot as plt  
import numpy as np
```

```
ypoints = np.array([3, 8, 1, 10])
```

```
plt.plot(ypoints, color = 'r' )
```

```
plt.show()
```

You can also use Hexadecimal color values:

Example

Plot with a beautiful green line:

...

```
plt.plot(ypoints, c = '#4CAF50' )
```

...

Or any of the 140 supported color names.

Example

Plot with the color named "hotpink":

...

```
plt.plot(ypoints, c = 'hotpink' )
```

...

Line Width

You can use the keyword argument `linewidt h` or the shorter `l w` to change the width of the line.

The value is a floating number, in points:

Example

Plot with a 20.5pt wide line:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
ypoints = np.array([3 , 8 , 1 , 10 ])
```

```
plt.plot(ypoints, linewidth = '20.5' )
```

```
plt.show()
```

Multiple Lines

You can plot as many lines as you like by simply adding more `plt.plot()` functions:

Example

Draw two lines by specifying a `plt.plot()` function for each line:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
y1 = np.array([3 , 8 , 1 , 10 ])
```

```
y2 = np.array([6 , 2 , 7 , 11 ])
```

```
plt.plot(y1)
```

```
plt.plot(y2)
```

```
plt.show()
```

You can also plot many lines by adding the points for the x- and y-axis for each line in the same `plt.plot()` function.

(In the examples above we only specified the points on the y-axis, meaning that the points on the x-axis got the the default values (0, 1, 2, 3).)

The x- and y- values come in pairs:

Example

Draw two lines by specifiyng the x- and y-point values for both lines:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x1 = np.array([0 , 1 , 2 , 3 ])
```

```
y1 = np.array([3 , 8 , 1 , 10 ])
```

```
x2 = np.array([0 , 1 , 2 , 3 ])
```

```
y2 = np.array([6 , 2 , 7 , 11 ])
```

```
plt.plot(x1, y1, x2, y2)
```

```
plt.show()
```

Matplotlib Subplots

Display Multiple Plots

With the `subplots()` function you can draw multiple plots in one figure:

Example

Draw 2 plots:

```
import matplotlib.pyplot as plt  
import numpy as np
```

#plot 1:

```
x = np.array([0 , 1 , 2 , 3 ])  
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.subplot(1 , 2 , 1 )  
plt.plot(x,y)
```

#plot 2:

```
x = np.array([0 , 1 , 2 , 3 ])  
y = np.array([10 , 20 , 30 , 40 ])
```

```
plt.subplot(1 , 2 , 2 )
```

```
plt.plot(x,y)
```

```
plt.show()
```

The subplots() Function

The `subplots()` function takes three arguments that describes the layout of the figure.

The layout is organized in rows and columns, which are represented by the *first* and *second* argument.

The third argument represents the index of the current plot.

```
plt.subplot(1 , 2 , 1 )
```

#the figure has 1 row, 2 columns, and this plot is the *first* plot.

```
plt.subplot(1 , 2 , 2 )
```

#the figure has 1 row, 2 columns, and this plot is the *second* plot.

So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

Example

Draw 2 plots on top of each other:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

#plot 1:

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.subplot(2 , 1 , 1 )
```

```
plt.plot(x,y)
```

#plot 2:

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([10 , 20 , 30 , 40 ])
```

```
plt.subplot(2 , 1 , 2 )
```

```
plt.plot(x,y)
```

```
plt.show()
```

You can draw as many plots you like on one figure, just describe the number of rows, columns, and the index of the plot.

Example

Draw 6 plots:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.subplot(2 , 3 , 1 )
```

```
plt.plot(x,y)
```

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([10 , 20 , 30 , 40 ])
```

```
plt.subplot(2 , 3 , 2 )
```

```
plt.plot(x,y)
```

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.subplot(2 , 3 , 3 )
```

```
plt.plot(x,y)
```

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([10 , 20 , 30 , 40 ])
```

```
plt.subplot(2 , 3 , 4 )
```

```
plt.plot(x,y)
```

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.subplot(2 , 3 , 5 )
```

```
plt.plot(x,y)
```

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([10 , 20 , 30 , 40 ])
```

```
plt.subplot(2 , 3 , 6 )
```

```
plt.plot(x,y)
```

```
plt.show()
```

Title

You can add a title to each plot with the `title()` function:

Example

2 plots, with titles:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

#plot 1:

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.subplot(1 , 2 , 1 )
```

```
plt.plot(x,y)
```

```
plt.title("SALES" )
```

#plot 2:

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([10 , 20 , 30 , 40 ])
```

```
plt.subplot(1 , 2 , 2 )
```

```
plt.plot(x,y)
```

```
plt.title("INCOME" )
```

```
plt.show()
```

Super Title

You can add a title to the entire figure with the `suptitle()` function:

Example

Add a title for the entire figure:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

#plot 1:

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.subplot(1 , 2 , 1 )
```

```
plt.plot(x,y)
```

```
plt.title("SALES" )
```

#plot 2:

```
x = np.array([0 , 1 , 2 , 3 ])
```

```
y = np.array([10 , 20 , 30 , 40 ])
```

```
plt.subplot(1 , 2 , 2 )
```

```
plt.plot(x,y)
```

```
plt.title("INCOME" )
```

```
plt.suptitle("MY SHOP" )
```

```
plt.show()
```

The observation in the example above is the result of 13 cars passing by.

The X-axis shows how old the car is.

The Y-axis shows the speed of the car when it passes.

Are there any relationships between the observations?

It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

Compare Plots

In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

Example

Draw two plots on the same figure:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

#day one, the age and speed of 13 cars:

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
```

```
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
```

```
plt.scatter(x, y)
```

#day two, the age and speed of 15 cars:

```
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
```

```
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
```

```
plt.scatter(x, y)
```

```
plt.show()
```

Note: The two plots are plotted with two different colors, by default blue and orange, you will learn how to change colors later in this chapter.

By comparing the two plots, I think it is safe to say that they both gives us the same conclusion: the newer the car, the faster it drives.

Colors

You can set your own color for each scatter plot with the `color` or the `c` argument:

Example

Set your own color of the markers:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
```

```
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
```

```
plt.scatter(x, y, color = 'hotpink' )
```

```
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
```

```
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
```

```
plt.scatter(x, y, color = '#88c999' )
```

```
plt.show()
```

Color Each Dot

You can even set a specific color for each dot by using an array of colors as value for the `c` argument:

Note: You *cannot* use the `color` argument for this, only the `c` argument.

Example

Set your own color of the markers:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
```

```
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
```

```
colors = np.array(["red","green","blue","yellow","pink","black",
,"orange","purple","beige","brown","gray","cyan","magenta"])
```

```
plt.scatter(x, y, c=colors)
```

```
plt.show()
```

ColorMap

The Matplotlib module has a number of available colormaps.

A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.

This colormap is called 'viridis' and as you can see it ranges from 0, which is a purple color, and up to 100, which is a yellow color.

How to Use the ColorMap

You can specify the colormap with the keyword argument `cmap` with the value of the colormap, in this case '`'viridis'`' which is one of the built-in colormaps available in Matplotlib.

In addition you have to create an array with values (from 0 to 100), one value for each of the point in the scatter plot:

Example

Create a color array, and specify a colormap in the scatter plot:

```
import matplotlib.pyplot as plt  
  
import numpy as np  
  
  
  
x = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6])  
  
y = np.array([99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86])  
  
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])  
  
  
  
plt.scatter(x, y, c=colors, cmap='viridis')  
  
plt.show()
```

You can include the colormap in the drawing by including the `plt.colorbar()` statement:

Example

Include the actual colormap:

```
import matplotlib.pyplot as plt  
  
import numpy as np  
  
  
  
x = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6])  
  
y = np.array([99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86])  
  
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])  
  
  
  
plt.scatter(x, y, c=colors, cmap='viridis')  
  
  
  
plt.colorbar()  
  
  
plt.show()
```

Available ColorMaps

You can choose any of the built-in colormaps:

Name	Reverse
Accent	Accent_r
Blues	Blues_r
BrBG	BrBG_r
BuGn	BuGn_r
BuPu	BuPu_r
CMRmap	CMRmap_r
Dark2	Dark2_r
GnBu	GnBu_r
Greens	Greens_r
Greys	Greys_r
OrRd	OrRd_r

Oranges	Oranges_r
PRGn	PRGn_r
Paired	Paired_r
Pastel1	Pastel1_r
Pastel2	Pastel2_r
PiYG	PiYG_r
PuBu	PuBu_r
PuBuGn	PuBuGn_r
PuOr	PuOr_r
PuRd	PuRd_r
Purples	Purples_r
RdBu	RdBu_r

RdGy	RdGy_r
RdPu	RdPu_r
RdYlBu	RdYlBu_r
RdYlGn	RdYlGn_r
Reds	Reds_r
Set1	Set1_r
Set2	Set2_r
Set3	Set3_r
Spectral	Spectral_r
Wistia	Wistia_r
YlGn	YlGn_r
YlGnBu	YlGnBu_r

YlOrBr	YlOrBr_r
YlOrRd	YlOrRd_r
afmhot	afmhot_r
autumn	autumn_r
binary	binary_r
bone	bone_r
brg	brg_r
bwr	bwr_r
cividis	cividis_r
cool	cool_r
coolwarm	coolwarm_r
copper	copper_r

cubehelix	cubehelix_r
flag	flag_r
gist_earth	gist_earth_r
gist_gray	gist_gray_r
gist_heat	gist_heat_r
gist_ncar	gist_ncar_r
gist_rainbow	gist_rainbow_r
gist_stern	gist_stern_r
gist_yarg	gist_yarg_r
gnuplot	gnuplot_r
gnuplot2	gnuplot2_r
gray	gray_r

hot	hot_r
hsv	hsv_r
inferno	inferno_r
jet	jet_r
magma	magma_r
nipy_spectral	nipy_spectral_r
ocean	ocean_r
pink	pink_r
plasma	plasma_r
prism	prism_r
rainbow	rainbow_r
seismic	seismic_r

spring	spring_r
summer	summer_r
tab10	tab10_r
tab20	tab20_r
tab20b	tab20b_r
tab20c	tab20c_r
terrain	terrain_r
twilight	twilight_r
twilight_shifted	twilight_shifted_r
viridis	viridis_r
winter	winter_r

Size

You can change the size of the dots with the `s` argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

Example

Set your own size for the markers:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6])
```

```
y = np.array([99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86])
```

```
sizes = np.array([20, 50, 100, 200, 500, 1000, 60, 90, 10, 300, 600, 800, 75])
```

```
plt.scatter(x, y, s=sizes)
```

```
plt.show()
```

Alpha

You can adjust the transparency of the dots with the `alph a` argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

Example

Set your own size for the markers:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])  
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])  
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])
```

```
plt.scatter(x, y, s=sizes, alpha=0.5 )
```

`plt.show()`

Combine Color Size and Alpha

You can combine a colormap with different sizes on the dots. This is best visualized if the dots are transparent:

Example

Create random arrays with 100 values for x-points, y-points, colors and sizes:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
  
  
x = np.random.randint(100 , size=(100 ))  
y = np.random.randint(100 , size=(100 ))  
colors = np.random.randint(100 , size=(100 ))  
sizes = 10 * np.random.randint(100 , size=(100 ))  
  
  
  
plt.scatter(x, y, c=colors, s=sizes, alpha=0.5 , cmap='nipy_spectral' )  
  
  
  
plt.colorbar()  
  
  
plt.show()
```

Matplotlib Bars

Creating Bars

With Pyplot, you can use the `bar()` function to draw bar graphs:

Example

Draw 4 bars:

```
import matplotlib.pyplot as plt  
  
import numpy as np  
  
  
  
x = np.array(["A" , "B" , "C" , "D" ])  
  
y = np.array([3 , 8 , 1 , 10 ])  
  
  
  
plt.bar(x,y)  
plt.show()
```

The `bar()` function takes arguments that describes the layout of the bars.

The categories and their values represented by the *first* and *second* argument as arrays.

Example

```
x = ["APPLES" , "BANANAS" ]  
  
y = [400 , 350 ]  
  
plt.bar(x, y)
```

Horizontal Bars

If you want the bars to be displayed horizontally instead of vertically, use the `barh()` function:

Example

Draw 4 horizontal bars:

```
import matplotlib.pyplot as plt  
  
import numpy as np  
  
  
  
x = np.array(["A" , "B" , "C" , "D" ])  
y = np.array([3 , 8 , 1 , 10 ])  
  
  
  
plt.barh(x, y)  
plt.show()
```

Bar Color

The `bar()` and `barh()` takes the keyword argument `color` to set the color of the bars:

Example

Draw 4 red bars:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.array(["A" , "B" , "C" , "D" ])  
y = np.array([3 , 8 , 1 , 10 ])  
  
plt.bar(x, y, color = "red" )  
plt.show()
```

Color Names

You can use any of the 140 supported color names.

Example

Draw 4 "hot pink" bars:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.array(["A" , "B" , "C" , "D" ])  
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.bar(x, y, color = "hotpink" )
```

```
plt.show()
```

Color Hex

Or you can use Hexadecimal color values:

Example

Draw 4 bars with a beautiful green color:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x = np.array(["A" , "B" , "C" , "D" ])
```

```
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.bar(x, y, color = "#4CAF50" )
```

```
plt.show()
```

Bar Width

The `bar()` takes the keyword argument `widt h` to set the width of the bars:

Example

Draw 4 very thin bars:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
  
  
x = np.array(["A" , "B" , "C" , "D" ])  
y = np.array([3 , 8 , 1 , 10 ])  
  
  
  
plt.bar(x, y, width = 0.1 )  
plt.show()
```

The default width value is 0.8

Note: For horizontal bars, use `height` instead of `width`.

Bar Height

The `barh()` takes the keyword argument `height` to set the height of the bars:

Example

Draw 4 very thin bars:

```
import matplotlib.pyplot as plt  
import numpy as np
```

```
x = np.array(["A" , "B" , "C" , "D" ])
```

```
y = np.array([3 , 8 , 1 , 10 ])
```

```
plt.barh(x, y, height = 0.1 )
```

```
plt.show()
```

Matplotlib Histograms

Histogram

A histogram is a graph showing *frequency* distributions.

It is a graph showing the number of observations within each given interval.

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:

You can read from the histogram that there are approximately:

2 people from 140 to 145cm

5 people from 145 to 150cm

15 people from 151 to 156cm

31 people from 157 to 162cm

46 people from 163 to 168cm

53 people from 168 to 173cm

45 people from 173 to 178cm

28 people from 179 to 184cm

21 people from 185 to 190cm

4 people from 190 to 195cm

Create Histogram

In Matplotlib, we use the `hist()` function to create histograms.

The `hist()` function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10. Learn more about [Normal Data Distribution](#) in our [Machine Learning Tutorial](#).

Example

A Normal Data Distribution by NumPy:

```
import numpy as np
```

```
x = np.random.normal(170 , 10 , 250 )
```

```
print (x)
```

The `hist()` function will read the array and produce a histogram:

Example

A simple pie chart:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.random.normal(170 , 10 , 250 )
```

```
plt.hist(x)
```

```
plt.show()
```

Matplotlib Pie Charts

Creating Pie Charts

With Pyplot, you can use the `pie()` function to draw pie charts:

Example

A simple pie chart:

```
import matplotlib.pyplot as plt  
  
import numpy as np  
  
y = np.array([35 , 25 , 25 , 15 ])  
  
  
plt.pie(y)  
plt.show()
```

Labels

Add labels to the pie chart with the `label` parameter.

The `label` parameter must be an array with one label for each wedge:

Example

A simple pie chart:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35 , 25 , 25 , 15 ])  
mylabels = ["Apples" , "Bananas" , "Cherries" , "Dates" ]  
  
plt.pie(y, lables = mylabels)  
plt.show()
```

Start Angle

As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a `startangle` parameter.

The `startangle` parameter is defined with an angle in degrees, default angle is 0:

Example

Start the first wedge at 90 degrees:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35 , 25 , 25 , 15 ])  
mylabels = ["Apples" , "Bananas" , "Cherries" , "Dates" ]  
  
plt.pie(y, lables = mylabels, startangle = 90 )  
plt.show()
```

Explode

Maybe you want one of the wedges to stand out? The `explode` parameter allows you to do that.

The `explode` parameter, if specified, and not `None`, must be an array with one value for each wedge.

Each value represents how far from the center each wedge is displayed:

Example

Pull the "Apples" wedge 0.2 from the center of the pie:

```
import matplotlib.pyplot as plt  
  
import numpy as np  
  
  
  
y = np.array([35, 25, 25, 15])  
  
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]  
  
myexplode = [0.2, 0, 0, 0]  
  
  
  
plt.pie(y, labels = mylabels, explode = myexplode)  
plt.show()
```

Shadow

Add a shadow to the pie chart by setting the `shadow` parameter to `True`:

Example

Add a shadow:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35 , 25 , 25 , 15 ])  
mylabels = ["Apples" , "Bananas" , "Cherries" , "Dates" ]  
myexplode = [0.2 , 0 , 0 , 0 ]  
  
plt.pie(y, labels = mylabels, explode = myexplode, shadow = True )  
plt.show()
```

Colors

You can set the color of each wedge with the `color s` parameter.

The `color s` parameter, if specified, must be an array with one value for each wedge:

Example

Specify a new color for each wedge:

```
import matplotlib.pyplot as plt  
import numpy as np
```

```
y = np.array([35 , 25 , 25 , 15 ])  
mylabels = ["Apples" , "Bananas" , "Cherries" , "Dates" ]  
mycolors = ["black" , "hotpink" , "b" , "#4CAF50" ]
```

```
plt.pie(y, labels = mylabels, colors = mycolors)  
plt.show()
```

You can use [Hexadecimal color values](#) , any of the [140 supported color names](#) , or one of these shortcuts:

'r' - Red

'g' - Green

'b' - Blue

'c' - Cyan

'm' - Magenta

'y' - Yellow

'k' - Black

'w' - White

Legend

To add a list of explanation for each wedge, use the `legend()` function:

Example

Add a legend:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35 , 25 , 25 , 15 ])  
mylabels = ["Apples" , "Bananas" , "Cherries" , "Dates" ]  
  
plt.pie(y, labels = mylabels)  
plt.legend()  
plt.show()
```

Legend With Header

To add a header to the legend, add the `title` parameter to the `legend` function.

Example

Add a legend with a header:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35 , 25 , 25 , 15 ])  
mylabels = ["Apples" , "Bananas" , "Cherries" , "Dates" ]
```

```
plt.pie(y, labels = mylabels)  
plt.legend(title = "Four Fruits:" )  
plt.show()
```

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?

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.

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:

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 `lite r` 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__)
```

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
```

```
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

```
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. **kib i** returns **1024**)

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. `gra m` 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) #0.4535923699999997  
print (constants.pound) #0.4535923699999997  
print (constants.oz) #0.02834952312499998  
print (constants.ounce) #0.02834952312499998  
print (constants.stone) #6.350293179999995  
print (constants.long_ton) #1016.0469088  
print (constants.short_ton) #907.184739999999
```

```
print (constants.troy_ounce) #0.03110347679999998  
print (constants.troy_pound) #0.3732417215999996  
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

```
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)           #2.539999999999997e-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
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. `ps i` returns `6894.75729316836 1`)

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

print (constants.hectare) #10000.0
print (constants.acre)   #4046.856422399992
```

Volume:

Return the specified unit in cubic meters (e.g. `lite_r` returns `0.001`)

Example

```
from scipy import constants

print (constants.liter)          #0.001
print (constants.litre)          #0.001
print (constants.gallon)         #0.003785411783999997
print (constants.gallon_US)       #0.003785411783999997
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
```

```
print (constants.kmh)          #0.2777777777777778  
print (constants.mph)          #0.4470399999999994  
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  
print (constants.Btu)         #1055.05585262  
print (constants.Btu_IT)      #1055.05585262  
print (constants.Btu_th)       #1054.350264488888  
print (constants.ton_TNT)     #4184000000.0
```

Power:

Return the specified unit in watts (e.g. `horsepowe r` returns
`745.699871582270 1`)

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
```

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 *non* linear equations, like this one:

$x + \cos(x)$

For that you can use SciPy's `optimze.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)
```

```
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 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'

'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)
```

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:

```
[1, 0, 2, 0, 0, 3, 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 CSR matrix by passing an array into 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))
```

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  
  
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)
```

SciPy Graphs

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 $n \times n$ matrix where n is the number of elements in a graph.

And the values represents 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 Adjency 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  
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 ]
```

```
])
```

```
newarr = csr_matrix(arr)
```

```
print (dijkstra(newarr, return_predecessors=True , indices=0 ))
```

Floyd Warshall

Use the `floyd_marshall()` method to find 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_marshall  
  
from scipy.sparse import csr_matrix  
  
  
  
arr = np.array([  
    [0 , 1 , 2 ],  
    [1 , 0 , 0 ],  
    [2 , 0 , 0 ]  
])  
  
  
  
newarr = csr_matrix(arr)  
  
  
  
print (floyd_marshall(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

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 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 ]  
])
```

```
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 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

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 ]
```

```
])
```

```
simplices = Delaunay(points).simplices
```

```
plt.triplot(points[:, 0 ], points[:, 1 ], simplices)
```

```
plt.scatter(points[:, 0 ], points[:, 1 ], color='r' )
```

```
plt.show()
```

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([
```

```
[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()
```

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)
```

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 metrics. E.g. "K Nearest Neighbors", or "K Means" etc.

Let us look at some of the Distance Metrics:

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)
```

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)
```

Cosine Distance

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)
```

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)
```

SciPy Matlab Arrays

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:

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)
```

Use the variable name "vec" to display only the array from the matlab data:

Example

...

```
print (mydata['vec' ])
```

Note: We can see that the array originally was 1D, but on extraction it has increased one dimension.

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' ])
```

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 usage, 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 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 same range as of the old xs, meaning that we cant 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 `x s` and `y s` and produce a callable funciton that can be called with new `x s` .

Piecewise function: A function that has different definition 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  
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 `x s` and `y s` as arguments and produces a callable function that can be called with new `x s`.

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:

```
[6.25748981 6.62190817 7.00310702 7.40121814 7.8161443 8.247734  
02]
```

```
8.69590519 9.16070828 9.64233874]
```

SciPy Statistical Significance Tests

What is 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 called `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.

Its alternate 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 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 testing for both side 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 the statistical significance.

If p value \leq alpha we reject the null hypothesis and say that the data is statistically significant. otherwise we accept the null hypothesis.

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 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:

```
Ttest_indResult(statistic=0.40833510339674095,  
pvalue=0.68346891833752133)
```

If you want to return only the p-value, use the `pvalue` 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.

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 tailed. We can pass parameter alternative 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

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:

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)
```

Machine Learning

Machine Learning is making the computer learn from studying data and statistics.

Machine Learning is a step into the direction of artificial intelligence (AI).

Machine Learning is a program that analyses data and learns to predict the outcome.

Where To Start?

In this tutorial we will go back to mathematics and study statistics, and how to calculate important numbers based on data sets.

We will also learn how to use various Python modules to get the answers we need.

And we will learn how to make functions that are able to predict the outcome based on what we have learned.

Data Set

In the mind of a computer, a data set is any collection of data. It can be anything from an array to a complete database.

Example of an array:

[99 ,86 ,87 ,88 ,111 ,86 ,103 ,87 ,94 ,78 ,77 ,85 ,86]

Example of a database:

Carname	Color	Age	Speed	AutoPass
BMW	red	5	99	Y
Volvo	black	7	86	Y
VW	gray	8	87	N
VW	white	7	88	Y
Ford	white	2	111	Y

VW	white	17	86	Y
Tesla	red	2	103	Y
BMW	black	9	87	Y
Volvo	gray	4	94	N
Ford	white	11	78	N
Toyota	gray	12	77	N
VW	white	9	85	N
Toyota	blue	6	86	Y

By looking at the array, we can guess that the average value is probably around 80 or 90, and we are also able to determine the highest value and the lowest value, but what else can we do?

And by looking at the database we can see that the most popular color is white, and the oldest car is 17 years, but what if we could predict if a car had an AutoPass, just by looking at the other values?

That is what Machine Learning is for! Analyzing data and predicting the outcome!

In Machine Learning it is common to work with very large data sets. In this tutorial we will try to make it as easy as possible to understand the different concepts of machine learning, and we will work with small easy-to-understand data sets.

Data Types

To analyze data, it is important to know what type of data we are dealing with.

We can split the data types into three main categories:

- Numerical
- Categorical
- Ordinal

Numerical data are numbers, and can be split into two numerical categories:

- Discrete Data
 - numbers that are limited to integers. Example: The number of cars passing by.
- Continuous Data
 - numbers that are of infinite value. Example: The price of an item, or the size of an item

Categorical data are values that cannot be measured up against each other. Example: a color value, or any yes/no values.

Ordinal data are like categorical data, but can be measured up against each other. Example: school grades where A is better than B and so on.

By knowing the data type of your data source, you will be able to know what technique to use when analyzing them.

You will learn more about statistics and analyzing data in the next chapters.

Mean, Median, and Mode

What can we learn from looking at a group of numbers?

In Machine Learning (and in mathematics) there are often three values that interests us:

- Mean - The average value
- Median - The mid point value
- Mode - The most common value

Example: We have registered the speed of 13 cars:

speed = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]

What is the average, the middle, or the most common speed value?

Mean

The mean value is the average value.

To calculate the mean, find the sum of all values, and divide the sum by the number of values:

(99 + 86 + 87 + 88 + 111 + 86 + 103 + 87 + 94 + 78 + 77 + 85 + 86) / 13 = 89.77

The NumPy module has a method for this. Learn about the NumPy module in our NumPy Tutorial.

Example

Use the NumPy `mean()` method to find the average speed:

```
import numpy
```

```
speed = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
```

```
x = numpy.mean(speed)
```

```
print(x)
```

Median

The median value is the value in the middle, after you have sorted all the values:

```
77, 78, 85, 86, 86, 86, 86, 87, 87, 88, 94, 99, 103, 111
```

It is important that the numbers are sorted before you can find the median.

The NumPy module has a method for this:

Example

Use the NumPy `median()` method to find the middle value:

```
import numpy
```

```
speed = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
```

```
x = numpy.median(speed)
```

```
print (x)
```

If there are two numbers in the middle, divide the sum of those numbers by two.

```
77 , 78 , 85 , 86 , 86 , 86, 87 , 87 , 94 , 98 , 99 , 103
```

$$(86 + 87) / 2 = 86.5$$

Example

Using the NumPy module:

```
import numpy
```

```
speed = [99 ,86 ,87 ,88 ,86 ,103 ,87 ,94 ,78 ,77 ,85 ,86 ]
```

```
x = numpy.median(speed)
```

```
print (x)
```

Mode

The Mode value is the value that appears the most number of times:

```
99 , 86 , 87 , 88 , 111 , 86 , 103 , 87 , 94 , 78 , 77 , 85 , 86 = 86
```

The SciPy module has a method for this. Learn about the SciPy module in our SciPy Tutorial.

Example

Use the SciPy `mode()` method to find the number that appears the most:

```
from scipy import stats
```

```
speed = [99 ,86 ,87 ,88 ,111 ,86 ,103 ,87 ,94 ,78 ,77 ,85 ,86 ]
```

```
x = stats.mode(speed)
```

```
print (x)
```

Chapter Summary

The Mean, Median, and Mode are techniques that are often used in Machine Learning, so it is important to understand the concept behind them.

Machine Learning - Standard Deviation

What is Standard Deviation?

Standard deviation is a number that describes how spread out the values are.

A low standard deviation means that most of the numbers are close to the mean (average) value.

A high standard deviation means that the values are spread out over a wider range.

Example: This time we have registered the speed of 7 cars:

```
speed = [86, 87, 88, 86, 87, 85, 86]
```

The standard deviation is:

```
0.9
```

Meaning that most of the values are within the range of 0.9 from the mean value, which is 86.4.

Let us do the same with a selection of numbers with a wider range:

```
speed = [32, 111, 138, 28, 59, 77, 97]
```

The standard deviation is:

```
37.85
```

Meaning that most of the values are within the range of 37.85 from the mean value, which is 77.4.

As you can see, a higher standard deviation indicates that the values are spread out over a wider range.

The NumPy module has a method to calculate the standard deviation:

Example

Use the NumPy `std()` method to find the standard deviation:

```
import numpy
```

```
speed = [86, 87, 88, 86, 87, 85, 86]
```

```
x = numpy.std(speed)
```

```
print (x)
```

Example

```
import numpy
```

```
speed = [32 ,111 ,138 ,28 ,59 ,77 ,97 ]
```

```
x = numpy.std(speed)
```

```
print (x)
```

Variance

Variance is another number that indicates how spread out the values are.

In fact, if you take the square root of the variance, you get the standard deviation!

Or the other way around, if you multiply the standard deviation by itself, you get the variance!

To calculate the variance you have to do as follows:

1. Find the mean:

$$(32 + 111 + 138 + 28 + 59 + 77 + 97) / 7 = 77.4$$

2. For each value: find the difference from the mean:

$$32 - 77.4 = -45.4$$

$$111 - 77.4 = 33.6$$

$$138 - 77.4 = 60.6$$

$$28 - 77.4 = -49.4$$

$$59 - 77.4 = -18.4$$

$$77 - 77.4 = -0.4$$

$$97 - 77.4 = 19.6$$

3. For each difference: find the square value:

$$(-49.4)^2 = 2061.16$$

$$(33.6)^2 = 1128.96$$

$$(60.6)^2 = 3672.36$$

$$(-0.4)^2 = 0.16$$

$$(-18.4)^2 = 338.56$$

$$(19.6)^2 = 384.16$$

4. The variance is the average number of these squared differences:

$$(2061.16 + 1128.96 + 3672.36 + 2440.36 + 338.56 + 0.16 + 384.16) / 7 = 1432.2$$

Luckily, NumPy has a method to calculate the variance:

Example

Use the NumPy `var()` method to find the variance:

```
import numpy
```

```
speed = [32 ,111 ,138 ,28 ,59 ,77 ,97 ]
```

```
x = numpy.var(speed)
```

```
print (x)
```

Standard Deviation

As we have learned, the formula to find the standard deviation is the square root of the variance:

$$\sqrt{1432.25} = 37.85$$

Or, as in the example from before, use the NumPy to calculate the standard deviation:

Example

Use the NumPy `std()` method to find the standard deviation:

```
import numpy
```

```
speed = [32 ,111 ,138 ,28 ,59 ,77 ,97 ]
```

```
x = numpy.std(speed)
```

```
print (x)
```

Symbols

Standard Deviation is often represented by the symbol Sigma: σ

Variance is often represented by the symbol Sigma Square: σ^2

Chapter Summary

The Standard Deviation and Variance are terms that are often used in Machine Learning, so it is important to understand how to get them, and the concept behind them.

Machine Learning - Percentiles

What are Percentiles?

Percentiles are used in statistics to give you a number that describes the value that a given percent of the values are lower than.

Example: Let's say we have an array of the ages of all the people that lives in a street.

```
ages = [5, 31, 43, 48, 50, 41, 7, 11, 15, 39, 80, 82, 32, 2, 8, 6, 25, 36, 27, 61, 31]
```

What is the 75. percentile? The answer is 43, meaning that 75% of the people are 43 or younger.

The NumPy module has a method for finding the specified percentile:

Example

```
Use the NumPy percentile( ) method to find the percentiles:
```

```
import numpy
```

```
ages = [5, 31, 43, 48, 50, 41, 7, 11, 15, 39, 80, 82, 32, 2, 8, 6, 25, 36, 27, 61, 31]
```

```
x = numpy.percentile(ages, 75 )
```

```
print (x)
```

Example

What is the age that 90% of the people are younger than?

```
import numpy
```

```
ages = [5 ,31 ,43 ,48 ,50 ,41 ,7 ,11 ,15 ,39 ,80 ,82 ,32 ,2 ,8 ,6 ,25 ,36 ,27 ,61 ,31 ]
```

```
x = numpy.percentile(ages, 90 )
```

```
print (x)
```

Machine Learning - Data Distribution

Data Distribution

Earlier in this tutorial we have worked with very small amounts of data in our examples, just to understand the different concepts.

In the real world, the data sets are much bigger, but it can be difficult to gather real world data, at least at an early stage of a project.

How Can we Get Big Data Sets?

To create big data sets for testing, we use the Python module NumPy, which comes with a number of methods to create random data sets, of any size.

Example

Create an array containing 250 random floats between 0 and 5:

```
import numpy
```

```
x = numpy.random.uniform(0.0 , 5.0 , 250 )
```

```
print (x)
```

Histogram

To visualize the data set we can draw a histogram with the data we collected.

We will use the Python module Matplotlib to draw a histogram.

Learn about the Matplotlib module in our Matplotlib Tutorial.

Example

Draw a histogram:

```
import numpy
```

```
import matplotlib.pyplot as plt
```

```
x = numpy.random.uniform(0.0 , 5.0 , 250 )
```

```
plt.hist(x, 5 )
```

```
plt.show()
```

Histogram Explained

We use the array from the example above to draw a histogram with 5 bars.

The first bar represents how many values in the array are between 0 and 1.

The second bar represents how many values are between 1 and 2.

Etc.

Which gives us this result:

- 52 values are between 0 and 1
- 48 values are between 1 and 2
- 49 values are between 2 and 3
- 51 values are between 3 and 4
- 50 values are between 4 and 5

Note: The array values are random numbers and will not show the exact same result on your computer.

Big Data Distributions

An array containing 250 values is not considered very big, but now you know how to create a random set of values, and by changing the parameters, you can create the data set as big as you want.

Example

Create an array with 100000 random numbers, and display them using a histogram with 100 bars:

```
import numpy  
import matplotlib.pyplot as plt  
  
x = numpy.random.uniform(0.0 , 5.0 , 100000 )
```

```
plt.hist(x, 100 )
```

```
plt.show()
```

Machine Learning - Normal Data Distribution

Normal Data Distribution

In the previous chapter we learned how to create a completely random array, of a given size, and between two given values.

In this chapter we will learn how to create an array where the values are concentrated around a given value.

In probability theory this kind of data distribution is known as the *normal data distribution*, or the *Gaussian data distribution*, after the mathematician Carl Friedrich Gauss who came up with the formula of this data distribution.

Example

A typical normal data distribution:

```
import numpy  
import matplotlib.pyplot as plt  
  
x = numpy.random.normal(5.0 , 1.0 , 100000 )  
  
plt.hist(x, 100 )  
plt.show()
```

Note: A normal distribution graph is also known as the *bell curve* because of its characteristic shape of a bell.

Histogram Explained

We use the array from the `numpy.random.normal()` method, with 100000 values, to draw a histogram with 100 bars.

We specify that the mean value is 5.0, and the standard deviation is 1.0.

Meaning that the values should be concentrated around 5.0, and rarely further away than 1.0 from the mean.

And as you can see from the histogram, most values are between 4.0 and 6.0, with a top at approximately 5.0.

Machine Learning - Scatter Plot

Scatter Plot

A scatter plot is a diagram where each value in the data set is represented by a dot.

The Matplotlib module has a method for drawing scatter plots, it needs two arrays of the same length, one for the values of the x-axis, and one for the values of the y-axis:

```
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
```

```
y = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
```

The `x` array represents the age of each car.

The `y` array represents the speed of each car.

Example

Use the `scatter()` method to draw a scatter plot diagram:

```
import matplotlib.pyplot as plt
```

```
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
```

```
y = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
```

```
plt.scatter(x, y)
```

```
plt.show()
```

Scatter Plot Explained

The x-axis represents ages, and the y-axis represents speeds.

What we can read from the diagram is that the two fastest cars were both 2 years old, and the slowest car was 12 years old.

Note: It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

Random Data Distributions

In Machine Learning the data sets can contain thousands-, or even millions, of values.

You might not have real world data when you are testing an algorithm, you might have to use randomly generated values.

As we have learned in the previous chapter, the NumPy module can help us with that!

Let us create two arrays that are both filled with 1000 random numbers from a normal data distribution.

The first array will have the mean set to 5.0 with a standard deviation of 1.0.

The second array will have the mean set to 10.0 with a standard deviation of 2.0:

Example

A scatter plot with 1000 dots:

```
import numpy  
import matplotlib.pyplot as plt  
  
  
x = numpy.random.normal(5.0 , 1.0 , 1000 )  
y = numpy.random.normal(10.0 , 2.0 , 1000 )  
  
  
plt.scatter(x, y)  
plt.show()
```

Scatter Plot Explained

We can see that the dots are concentrated around the value 5 on the x-axis, and 10 on the y-axis.

We can also see that the spread is wider on the y-axis than on the x-axis.

Machine Learning - Linear Regression

Regression

The term regression is used when you try to find the relationship between variables.

In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events.

Linear Regression

Linear regression uses the relationship between the data-points to draw a straight line through all them.

This line can be used to predict future values.

In Machine Learning, predicting the future is very important.

How Does it Work?

Python has methods for finding a relationship between data-points and to draw a line of linear regression. We will show you how to use these methods instead of going through the mathematic formula.

In the example below, the x-axis represents age, and the y-axis represents speed. We have registered the age and speed of 13 cars as they were passing a tollbooth. Let us see if the data we collected could be used in a linear regression:

Example

Start by drawing a scatter plot:

```
import matplotlib.pyplot as plt

x = [5 ,7 ,8 ,7 ,2 ,17 ,2 ,9 ,4 ,11 ,12 ,9 ,6 ]
y = [99 ,86 ,87 ,88 ,111 ,86 ,103 ,87 ,94 ,78 ,77 ,85 ,86 ]
```

```
plt.scatter(x, y)
```

```
plt.show()
```

Example

Import `scipy` and draw the line of Linear Regression:

```
import matplotlib.pyplot as plt
from scipy import stats

x = [5 ,7 ,8 ,7 ,2 ,17 ,2 ,9 ,4 ,11 ,12 ,9 ,6 ]
y = [99 ,86 ,87 ,88 ,111 ,86 ,103 ,87 ,94 ,78 ,77 ,85 ,86 ]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
    return slope * x + intercept

mymodel = list (map (myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Example Explained

Import the modules you need.

You can learn about the Matplotlib module in our [Matplotlib Tutorial](#).

You can learn about the SciPy module in our [SciPy Tutorial](#).

```
import matplotlib.pyplot as plt
```

```
from scipy import stats
```

Create the arrays that represent the values of the x and y axis:

```
x = [5 ,7 ,8 ,7 ,2 ,17 ,2 ,9 ,4 ,11 ,12 ,9 ,6 ]
```

```
y = [99 ,86 ,87 ,88 ,111 ,86 ,103 ,87 ,94 ,78 ,77 ,85 ,86 ]
```

Execute a method that returns some important key values of Linear Regression:

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

Create a function that uses the `slope` and `intercept` values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed:

```
def myfunc(x):
```

```
    return slope * x + intercept
```

Run each value of the x array through the function. This will result in a new array with new values for the y-axis:

```
mymodel = list (map (myfunc, x))
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of linear regression:

```
plt.plot(x, mymodel)
```

Display the diagram:

```
plt.show()
```

R for Relationship

It is important to know how the relationship between the values of the x-axis and the values of the y-axis is, if there are no relationship the linear regression can not be used to predict anything.

This relationship - the coefficient of correlation - is called **r**.

The **r** value ranges from 0 to 1, where 0 means no relationship, and 1 means 100% related.

Python and the Scipy module will compute this value for you, all you have to do is feed it with the x and y values.

Example

How well does my data fit in a linear regression?

```
from scipy import stats
```

```
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
```

```
y = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
```

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

```
print(r)
```

Note: The result -0.76 shows that there is a relationship, not perfect, but it indicates that we could use linear regression in future predictions.

Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a 10 years old car.

To do so, we need the same **myfunc()** function from the example above:

```
def myfunc(x):  
    return slope * x + intercept
```

Example

Predict the speed of a 10 years old car:

```
from scipy import stats
```

```
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]  
y = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
```

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

```
def myfunc(x):  
    return slope * x + intercept  
  
speed = myfunc(10)
```

```
print(speed)
```

Bad Fit?

Let us create an example where linear regression would not be the best method to predict future values.

Example

These values for the x- and y-axis should result in a very bad fit for linear regression:

```
import matplotlib.pyplot as plt  
from scipy import stats  
  
x = [89, 43, 36, 36, 95, 10, 66, 34, 38, 20, 26, 29, 48, 64, 6, 5, 36, 66, 72, 40]  
y = [21, 46, 3, 35, 67, 95, 53, 72, 58, 10, 26, 34, 90, 33, 38, 20, 56, 2, 47, 15]
```

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

```
def myfunc(x):  
    return slope * x + intercept
```

```
mymodel = list (map (myfunc, x))
```

```
plt.scatter(x, y)
```

```
plt.plot(x, mymodel)
```

```
plt.show()
```

And the **r** for relationship?

Example

You should get a very low **r** value.

```
import numpy  
from scipy import stats
```

```
x = [89 ,43 ,36 ,36 ,95 ,10 ,66 ,34 ,38 ,20 ,26 ,29 ,48 ,64 ,6 ,5 ,36 ,66 ,72 ,40 ]
```

```
y = [21 ,46 ,3 ,35 ,67 ,95 ,53 ,72 ,58 ,10 ,26 ,34 ,90 ,33 ,38 ,20 ,56 ,2 ,47 ,15 ]
```

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

```
print (r)
```

The result: 0.013 indicates a very bad relationship, and tells us that this data set is not suitable for linear regression.

Machine Learning - Polynomial Regression

Polynomial Regression

If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for polynomial regression.

Polynomial regression, like linear regression, uses the relationship between the variables x and y to find the best way to draw a line through the data points.

How Does it Work?

Python has methods for finding a relationship between data-points and to draw a line of polynomial regression. We will show you how to use these methods instead of going through the mathematic formula.

In the example below, we have registered 18 cars as they were passing a certain tollbooth.

We have registered the car's speed, and the time of day (hour) the passing occurred.

The x-axis represents the hours of the day and the y-axis represents the speed:

Example

Start by drawing a scatter plot:

```
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

plt.scatter(x, y)
plt.show()
```

Example Explained

Import the modules you need.

You can learn about the NumPy module in our NumPy Tutorial.

You can learn about the SciPy module in our SciPy Tutorial.

```
import numpy
```

```
import matplotlib.pyplot as plt
```

Create the arrays that represent the values of the x and y axis:

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
```

```
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

NumPy has a method that lets us make a polynomial model:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Then specify how the line will display, we start at position 1, and end at position 22:

```
myline = numpy.linspace(1, 22, 100)
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of polynomial regression:

```
plt.plot(myline, mymodel(myline))
```

Display the diagram:

```
plt.show()
```

R-Squared

It is important to know how well the relationship between the values of the x- and y-axis is, if there are no relationship the polynomial regression can not be used to predict anything.

The relationship is measured with a value called the r-squared.

The r-squared value ranges from 0 to 1, where 0 means no relationship, and 1 means 100% related.

Python and the Sklearn module will compute this value for you, all you have to do is feed it with the x and y arrays:

Example

How well does my data fit in a polynomial regression?

```
import numpy  
from sklearn.metrics import r2_score
```

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
```

```
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

```
print (r2_score(y, mymodel(x)))
```

Note: The result 0.94 shows that there is a very good relationship, and we can use polynomial regression in future predictions.

Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a car that passes the tollbooth at around 17 P.M:

To do so, we need the same `mymode l` array from the example above:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3 ))
```

Example

Predict the speed of a car passing at 17 P.M:

```
import numpy  
from sklearn.metrics import r2_score
```

```
x = [1 ,2 ,3 ,5 ,6 ,7 ,8 ,9 ,10 ,12 ,13 ,14 ,15 ,16 ,18 ,19 ,21 ,22 ]
```

```
y = [100 ,90 ,80 ,60 ,60 ,55 ,60 ,65 ,70 ,70 ,75 ,76 ,78 ,79 ,90 ,99 ,99 ,100 ]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3 ))
```

```
speed = mymodel(17 )
```

```
print (speed)
```

Bad Fit?

Let us create an example where polynomial regression would not be the best method to predict future values.

Example

These values for the x- and y-axis should result in a very bad fit for polynomial regression:

```
import numpy  
import matplotlib.pyplot as plt  
  
x = [89, 43, 36, 36, 95, 10, 66, 34, 38, 20, 26, 29, 48, 64, 6, 5, 36, 66, 72, 40]  
y = [21, 46, 3, 35, 67, 95, 53, 72, 58, 10, 26, 34, 90, 33, 38, 20, 56, 2, 47, 15]  
  
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))  
  
myline = numpy.linspace(2, 95, 100)  
  
plt.scatter(x, y)  
plt.plot(myline, mymodel(myline))  
plt.show()
```

And the r-squared value?

Example

You should get a very low r-squared value.

```

import numpy

from sklearn.metrics import r2_score


x = [89 ,43 ,36 ,36 ,95 ,10 ,66 ,34 ,38 ,20 ,26 ,29 ,48 ,64 ,6 ,5 ,36 ,66 ,72 ,40 ]
y = [21 ,46 ,3 ,35 ,67 ,95 ,53 ,72 ,58 ,10 ,26 ,34 ,90 ,33 ,38 ,20 ,56 ,2 ,47 ,15 ]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3 ))


print (r2_score(y, mymodel(x)))

```

Machine Learning - Multiple Regression

Multiple Regression

Multiple regression is like linear regression, but with more than one independent value, meaning that we try to predict a value based on two or more variables.

Take a look at the data set below, it contains some information about cars.

Car	Model	Volume	Weight	CO2
Toyota	Aygo		1000	790
Mitsubishi	Space Star		1200	1160
Skoda	Citigo		1000	929
Fiat	500		900	865
Mini	Cooper		1500	1140
VW	Up!		1000	929
Skoda	Fabia		1400	1109

Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114

Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

We can predict the CO2 emission of a car based on the size of the engine, but with multiple regression we can throw in more variables, like the weight of the car, to make the prediction more accurate.

How Does it Work?

In Python we have modules that will do the work for us. Start by importing the Pandas module.

```
import pandas
```

The Pandas module allows us to read csv files and return a DataFrame object.

```
df = pandas.read_csv("cars.csv")
```

Then make a list of the independent values and call this variable **X** .

Put the dependent values in a variable called **y** .

```
X = df[['Weight' , 'Volume']]
```

```
y = df['CO2']
```

Tip: It is common to name the list of independent values with a upper case X, and the list of dependent values with a lower case y.

We will use some methods from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the `LinearRegression()` method to create a linear regression object.

This object has a method called `fit()` that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
regr = linear_model.LinearRegression()
```

```
regr.fit(X, y)
```

Now we have a regression object that are ready to predict CO2 values based on a car's weight and volume:

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
```

```
predictedCO2 = regr.predict([[2300 , 1300 ]])
```

Example

See the whole example in action:

```
import pandas
```

```
from sklearn import linear_model
```

```
df = pandas.read_csv("cars.csv" )
```

```
X = df[['Weight' , 'Volume' ]]
```

```
y = df['CO2' ]
```

```
regr = linear_model.LinearRegression()
```

```
regr.fit(X, y)
```

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm  
3 :
```

```
predictedCO2 = regr.predict([[2300 , 1300 ]])
```

```
print (predictedCO2)
```

Result:

```
[107.2087328]
```

We have predicted that a car with 1.3 liter engine, and a weight of 2300 kg, will release approximately 107 grams of CO2 for every kilometer it drives.

Coefficient

The coefficient is a factor that describes the relationship with an unknown variable.

Example: if x is a variable, then $2x$ is x two times. x is the unknown variable, and the number 2 is the coefficient.

In this case, we can ask for the coefficient value of weight against CO2, and for volume against CO2. The answer(s) we get tells us what would happen if we increase, or decrease, one of the independent values.

Example

Print the coefficient values of the regression object:

```
import pandas
```

```
from sklearn import linear_model
```

```
df = pandas.read_csv("cars.csv")  
  
X = df[['Weight' , 'Volume' ]]  
y = df['CO2']  
  
regr = linear_model.LinearRegression()  
regr.fit(X, y)  
  
print (regr.coef_)
```

Result:

```
[0.00755095 0.00780526]
```

Result Explained

The result array represents the coefficient values of weight and volume.

Weight: 0.00755095

Volume: 0.00780526

These values tell us that if the weight increase by 1kg, the CO2 emission increases by 0.00755095g.

And if the engine size (Volume) increases by 1 cm³, the CO2 emission increases by 0.00780526 g.

I think that is a fair guess, but let test it!

We have already predicted that if a car with a 1300cm³ engine weighs 2300kg, the CO2 emission will be approximately 107g.

What if we increase the weight with 1000kg?

Example

Copy the example from before, but change the weight from 2300 to 3300:

```
import pandas  
from sklearn import linear_model  
  
  
df = pandas.read_csv("cars.csv")  
  
  
X = df[['Weight', 'Volume']]  
y = df['CO2']  
  
  
regr = linear_model.LinearRegression()  
regr.fit(X, y)  
  
  
predictedCO2 = regr.predict([[3300, 1300]])  
  
  
print(predictedCO2)
```

Result:

```
[114.75968007]
```

We have predicted that a car with 1.3 liter engine, and a weight of 3300 kg, will release approximately 115 grams of CO2 for every kilometer it drives.

Which shows that the coefficient of 0.00755095 is correct:

$$107.2087328 + (1000 * 0.00755095) = 114.75968$$

Machine Learning - Scale

Scale Features

When your data has different values, and even different measurement units, it can be difficult to compare them. What is kilograms compared to meters? Or altitude compared to time?

The answer to this problem is scaling. We can scale data into new values that are easier to compare.

Take a look at the table below, it is the same data set that we used in the multiple regression chapter, but this time the volume column contains values in *liters* instead of *cm³* (1.0 instead of 1000).

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1.0	790	99
Mitsubishi	Space Star	1.2	1160	95
Skoda	Citigo	1.0	929	95
Fiat	500	0.9	865	90
Mini	Cooper	1.5	1140	105
VW	Up!	1.0	929	105
Skoda	Fabia	1.4	1109	90
Mercedes	A-Class	1.5	1365	92
Ford	Fiesta	1.5	1112	98
Audi	A1	1.6	1150	99
Hyundai	I20	1.1	980	99

Suzuki	Swift	1.3	990	101
Ford	Fiesta	1.0	1112	99
Honda	Civic	1.6	1252	94
Hundai	I30	1.6	1326	97
Opel	Astra	1.6	1330	97
BMW	1	1.6	1365	99
Mazda	3	2.2	1280	104
Skoda	Rapid	1.6	1119	104
Ford	Focus	2.0	1328	105
Ford	Mondeo	1.6	1584	94
Opel	Insignia	2.0	1428	99
Mercedes	C-Class	2.1	1365	99
Skoda	Octavia	1.6	1415	99
Volvo	S60	2.0	1415	99
Mercedes	CLA	1.5	1465	102
Audi	A4	2.0	1490	104
Audi	A6	2.0	1725	114
Volvo	V70	1.6	1523	109
BMW	5	2.0	1705	114
Mercedes	E-Class	2.1	1605	115
Volvo	XC70	2.0	1746	117
Ford	B-Max	1.6	1235	104
BMW	2	1.6	1390	108

Opel	Zafira	1.6	1405	109
Mercedes	SLK	2.5	1395	120

It can be difficult to compare the volume 1.0 with the weight 790, but if we scale them both into comparable values, we can easily see how much one value is compared to the other.

There are different methods for scaling data, in this tutorial we will use a method called standardization.

The standardization method uses this formula:

$$z = (x - u) / s$$

Where z is the new value, x is the original value, u is the mean and s is the standard deviation.

If you take the weight column from the data set above, the first value is 790, and the scaled value will be:

$$(790 - 1292.23) / 238.74 = -2.1$$

If you take the volume column from the data set above, the first value is 1.0, and the scaled value will be:

$$(1.0 - 1.61) / 0.38 = -1.59$$

Now you can compare -2.1 with -1.59 instead of comparing 790 with 1.0.

You do not have to do this manually, the Python sklearn module has a method called [StandardScaler\(\)](#) which returns a Scaler object with methods for transforming data sets.

Example

Scale all values in the Weight and Volume columns:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
```

```
df = pandas.read_csv("cars2.csv")
```

```
X = df[['Weight' , 'Volume']]
```

```
scaledX = scale.fit_transform(X)
```

```
print (scaledX)
```

Result:

Note that the first two values are -2.1 and -1.59, which corresponds to our calculations:

```
[[ -2.10389253 -1.59336644]
 [ -0.55407235 -1.07190106]
 [ -1.52166278 -1.59336644]
 [ -1.78973979 -1.85409913]
 [ -0.63784641 -0.28970299]
 [ -1.52166278 -1.59336644]
 [ -0.76769621 -0.55043568]
 [ 0.3046118 -0.28970299]
 [ -0.7551301 -0.28970299]
 [ -0.59595938 -0.0289703 ]
 [ -1.30803892 -1.33263375]
 [ -1.26615189 -0.81116837]
 [ -0.7551301 -1.59336644]
 [ -0.16871166 -0.0289703 ]
 [ 0.14125238 -0.0289703 ]
 [ 0.15800719 -0.0289703 ]
 [ 0.3046118 -0.0289703 ]
 [ -0.05142797 1.53542584]
 [ -0.72580918 -0.0289703 ]
 [ 0.14962979 1.01396046]
 [ 1.2219378 -0.0289703 ]
 [ 0.5685001 1.01396046]
 [ 0.3046118 1.27469315]
 [ 0.51404696 -0.0289703 ]
```

```
[ 0.51404696  1.01396046]  
[ 0.72348212 -0.28970299]  
[ 0.8281997  1.01396046]  
[ 1.81254495  1.01396046]  
[ 0.96642691 -0.0289703 ]  
[ 1.72877089  1.01396046]  
[ 1.30990057  1.27469315]  
[ 1.90050772  1.01396046]  
[-0.23991961 -0.0289703 ]  
[ 0.40932938 -0.0289703 ]  
[ 0.47215993 -0.0289703 ]  
  
[ 0.4302729  2.31762392]]
```

Predict CO2 Values

The task in the Multiple Regression chapter was to predict the CO2 emission from a car when you only knew its weight and volume.

When the data set is scaled, you will have to use the scale when you predict values:

Example

Predict the CO2 emission from a 1.3 liter car that weighs 2300 kilograms:

```
import pandas  
  
from sklearn import linear_model  
  
from sklearn.preprocessing import StandardScaler  
  
scale = StandardScaler()
```

```
df = pandas.read_csv("cars2.csv")
```

```
X = df[['Weight', 'Volume']]
```

```
y = df['CO2']
```

```
scaledX = scale.fit_transform(X)
```

```
regr = linear_model.LinearRegression()
```

```
regr.fit(scaledX, y)
```

```
scaled = scale.transform([[2300 , 1.3 ]])
```

```
predictedCO2 = regr.predict([scaled[0 ]])
```

```
print (predictedCO2)
```

Result:

```
[107.2087328]
```

Machine Learning - Train/Test

Evaluate Your Model

In Machine Learning we create models to predict the outcome of certain events, like in the previous chapter where we predicted the CO₂ emission of a car when we knew the weight and engine size.

To measure if the model is good enough, we can use a method called Train/Test.

What is Train/Test

Train/Test is a method to measure the accuracy of your model.

It is called Train/Test because you split the data set into two sets: a training set and a testing set.

80% for training, and 20% for testing.

You *train* the model using the training set.

You *test* the model using the testing set.

Train the model means *create* the model.

Test the model means test the accuracy of the model.

Start With a Data Set

Start with a data set you want to test.

Our data set illustrates 100 customers in a shop, and their shopping habits.

Example

```
import numpy  
import matplotlib.pyplot as plt  
  
numpy.random.seed(2)  
  
x = numpy.random.normal(3, 1, 100)  
y = numpy.random.normal(150, 40, 100) / x  
  
plt.scatter(x, y)  
plt.show()
```

Result:

The x axis represents the number of minutes before making a purchase.

The y axis represents the amount of money spent on the purchase.

Split Into Train/Test

The *training* set should be a random selection of 80% of the original data.

The *testing* set should be the remaining 20%.

```
train_x = x[:80 ]
```

```
train_y = y[:80 ]
```

```
test_x = x[80 :]
```

```
test_y = y[80 :]
```

Display the Training Set

Display the same scatter plot with the training set:

Example

```
plt.scatter(train_x, train_y)
```

```
plt.show()
```

Result:

It looks like the original data set, so it seems to be a fair selection:

Display the Testing Set

To make sure the testing set is not completely different, we will take a look at the testing set as well.

Example

```
plt.scatter(test_x, test_y)  
plt.show()
```

Result:

The testing set also looks like the original data set:



Fit the Data Set

What does the data set look like? In my opinion I think the best fit would be a polynomial regression, so let us draw a line of polynomial regression.

To draw a line through the data points, we use the `plot()` method of the `matplotlib` module:

Example

Draw a polynomial regression line through the data points:

```
import numpy  
import matplotlib.pyplot as plt  
  
numpy.random.seed(2)  
  
x = numpy.random.normal(3, 1, 100)
```

```
y = numpy.random.normal(150 , 40 , 100 ) / x
```

```
train_x = x[:80 ]
```

```
train_y = y[:80 ]
```

```
test_x = x[80 :]
```

```
test_y = y[80 :]
```

```
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4 ))
```

```
myline = numpy.linspace(0 , 6 , 100 )
```

```
plt.scatter(train_x, train_y)
```

```
plt.plot(myline, mymodel(myline))
```

```
plt.show()
```

The result can back my suggestion of the data set fitting a polynomial regression, even though it would give us some weird results if we try to predict values outside of the data set. Example: the line indicates that a customer spending 6 minutes in the shop would make a purchase worth 200. That is probably a sign of overfitting.

But what about the R-squared score? The R-squared score is a good indicator of how well my data set is fitting the model.

R2

Remember R2, also known as R-squared?

It measures the relationship between the x axis and the y axis, and the value ranges from 0 to 1, where 0 means no relationship, and 1 means totally related.

The sklearn module has a method called `r2_score()` that will help us find this relationship.

In this case we would like to measure the relationship between the minutes a customer stays in the shop and how much money they spend.

Example

How well does my training data fit in a polynomial regression?

```
import numpy  
from sklearn.metrics import r2_score  
numpy.random.seed(2)  
  
x = numpy.random.normal(3, 1, 100)  
y = numpy.random.normal(150, 40, 100) / x  
  
train_x = x[:80]  
train_y = y[:80]
```

```
test_x = x[80 :]
```

```
test_y = y[80 :]
```

```
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4 ))
```

```
r2 = r2_score(train_y, mymodel(train_x))
```

```
print (r2)
```

Note: The result 0.799 shows that there is a OK relationship.

Bring in the Testing Set

Now we have made a model that is OK, at least when it comes to training data.

Now we want to test the model with the testing data as well, to see if gives us the same result.

Example

Let us find the R2 score when using testing data:

```
import numpy
```

```
from sklearn.metrics import r2_score
```

```
numpy.random.seed(2 )
```

```
x = numpy.random.normal(3 , 1 , 100 )  
y = numpy.random.normal(150 , 40 , 100 ) / x
```

```
train_x = x[:80 ]
```

```
train_y = y[:80 ]
```

```
test_x = x[80 :]
```

```
test_y = y[80 :]
```

```
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4 ))
```

```
r2 = r2_score(test_y, mymodel(test_x))
```

```
print (r2)
```

Note: The result 0.809 shows that the model fits the testing set as well, and we are confident that we can use the model to predict future values.

Predict Values

Now that we have established that our model is OK, we can start predicting new values.

Example

How much money will a buying customer spend, if she or he stays in the shop for 5 minutes?

```
print (mymodel(5 ))
```

Decision Tree

In this chapter we will show you how to make a "Decision Tree". A Decision Tree is a Flow Chart, and can help you make decisions based on previous experience.

In the example, a person will try to decide if he/she should go to a comedy show or not.

Luckily our example person has registered every time there was a comedy show in town, and registered some information about the comedian, and also registered if he/she went or not.

Age	Experience	Rank	Nationality	Go
36	10	9	UK	NO
42	12	4	USA	NO
23	4	6	N	NO
52	4	4	USA	NO
43	21	8	USA	YES
44	14	5	UK	NO
66	3	7	N	YES

35	14	9	UK	YES
52	13	7	N	YES
35	5	9	N	YES
24	3	5	USA	NO
18	3	7	UK	YES
45	9	9	UK	YES

Now, based on this data set, Python can create a decision tree that can be used to decide if any new shows are worth attending to.

How Does it Work?

First, import the modules you need, and read the dataset with pandas:

Example

Read and print the data set:

```
import pandas  
  
from sklearn import tree  
  
import pydotplus  
  
from sklearn.tree import DecisionTreeClassifier  
  
import matplotlib.pyplot as plt  
  
import matplotlib.image as pltimg
```

```
df = pandas.read_csv("shows.csv" )
```

```
print (df)
```

To make a decision tree, all data has to be numerical.

We have to convert the non numerical columns 'Nationality' and 'Go' into numerical values.

Pandas has a `map()` method that takes a dictionary with information on how to convert the values.

```
{'UK': 0, 'USA': 1, 'N': 2}
```

Means convert the values 'UK' to 0, 'USA' to 1, and 'N' to 2.

Example

Change string values into numerical values:

```
d = {'UK' : 0 , 'USA' : 1 , 'N' : 2 }
```

```
df['Nationality' ] = df['Nationality' ].map (d)
```

```
d = {'YES' : 1 , 'NO' : 0 }
```

```
df['Go' ] = df['Go' ].map (d)
```

```
print (df)
```

Then we have to separate the *feature* columns from the *target* column.

The feature columns are the columns that we try to predict *from* , and the target column is the column with the values we try to predict.

Example

X is the feature columns, y is the target column:

```
features = ['Age', 'Experience', 'Rank', 'Nationality']
```

```
X = df[features]
```

```
y = df['Go']
```

```
print(X)
```

```
print(y)
```

Now we can create the actual decision tree, fit it with our details, and save a .png file on the computer:

Example

Create a Decision Tree, save it as an image, and show the image:

```
dtree = DecisionTreeClassifier()
```

```
dtree = dtree.fit(X, y)
```

```
data = tree.export_graphviz(dtree, out_file=None, feature_names=features)
```

```
graph = pydotplus.graph_from_dot_data(data)
```

```
graph.write_png('mydecisiontree.png')
```

```
img=pltimg.imread('mydecisiontree.png')  
imgplot = plt.imshow(img)  
plt.show()
```

Result Explained

The decision tree uses your earlier decisions to calculate the odds for you to wanting to go see a comedian or not.

Let us read the different aspects of the decision tree:

Rank

Rank <= 6.5 means that every comedian with a rank of 6.5 or lower will follow the **True** arrow (to the left), and the rest will follow the **False** arrow (to the right).

gini = 0.497 refers to the quality of the split, and is always a number between 0.0 and 0.5, where 0.0 would mean all of the samples got the same result, and 0.5 would mean that the split is done exactly in the middle.

samples = 13 means that there are 13 comedians left at this point in the decision, which is all of them since this is the first step.

value = [6, 7] means that of these 13 comedians, 6 will get a "NO", and 7 will get a "GO".

Gini

There are many ways to split the samples, we use the GINI method in this tutorial.

The Gini method uses this formula:

$$\text{Gini} = 1 - \left(\frac{x}{n}\right)^2 - \left(\frac{y}{n}\right)^2$$

Where x is the number of positive answers ("GO"), n is the number of samples, and y is the number of negative answers ("NO"), which gives us this calculation:

$$1 - \left(\frac{7}{13}\right)^2 - \left(\frac{6}{13}\right)^2 = 0.497$$

The next step contains two boxes, one box for the comedians with a 'Rank' of 6.5 or lower, and one box with the rest.

True - 5 Comedians End Here:

gini = 0.0 means all of the samples got the same result.

samples = 5 means that there are 5 comedians left in this branch (5 comedian with a Rank of 6.5 or lower).

value = [5, 0] means that 5 will get a "NO" and 0 will get a "GO".

False - 8 Comedians Continue:

Nationality

Nationality <= 0.5 means that the comedians with a nationality value of less than 0.5 will follow the arrow to the left (which means everyone from the UK,), and the rest will follow the arrow to the right.

gini = 0.219 means that about 22% of the samples would go in one direction.

samples = 8 means that there are 8 comedians left in this branch (8 comedian with a Rank higher than 6.5).

value = [1, 7] means that of these 8 comedians, 1 will get a "NO" and 7 will get a "GO".

True - 4 Comedians Continue:

Age

Age <= 35.5 means that comedians at the age of 35.5 or younger will follow the arrow to the left, and the rest will follow the arrow to the right.

gini = 0.375 means that about 37,5% of the samples would go in one direction.

samples = 4 means that there are 4 comedians left in this branch (4 comedians from the UK).

value = [1, 3] means that of these 4 comedians, 1 will get a "NO" and 3 will get a "GO".

False - 4 Comedians End Here:

gini = 0.0 means all of the samples got the same result.

samples = 4 means that there are 4 comedians left in this branch (4 comedians not from the UK).

value = [0, 4] means that of these 4 comedians, 0 will get a "NO" and 4 will get a "GO".

True - 2 Comedians End Here:

gini = 0.0 means all of the samples got the same result.

samples = 2 means that there are 2 comedians left in this branch (2 comedians at the age 35.5 or younger).

value = [0, 2] means that of these 2 comedians, 0 will get a "NO" and 2 will get a "GO".

False - 2 Comedians Continue:

Experience

Experience <= 9.5 means that comedians with 9.5 years of experience, or less, will follow the arrow to the left, and the rest will follow the arrow to the right.

gini = 0.5 means that 50% of the samples would go in one direction.

samples = 2 means that there are 2 comedians left in this branch (2 comedians older than 35.5).

value = [1, 1] means that of these 2 comedians, 1 will get a "NO" and 1 will get a "GO".

True - 1 Comedian Ends Here:

gini = 0.0 means all of the samples got the same result.

samples = 1 means that there is 1 comedian left in this branch (1 comedian with 9.5 years of experience or less).

value = [0, 1] means that 0 will get a "NO" and 1 will get a "GO".

False - 1 Comedian Ends Here:

gini = 0.0 means all of the samples got the same result.

samples = 1 means that there is 1 comedians left in this branch (1 comedian with more than 9.5 years of experience).

value = [1, 0] means that 1 will get a "NO" and 0 will get a "GO".

Predict Values

We can use the Decision Tree to predict new values.

Example: Should I go see a show starring a 40 years old American comedian, with 10 years of experience, and a comedy ranking of 7?

Example

Use predict() method to predict new values:

```
print (dtree.predict([[40 , 10 , 7 , 1 ]]))
```

Example

What would the answer be if the comedy rank was 6?

```
print (dtree.predict([[40 , 10 , 6 , 1 ]]))
```

Different Results

You will see that the Decision Tree gives you different results if you run it enough times, even if you feed it with the same data.

That is because the Decision Tree does not give us a 100% certain answer. It is based on the probability of an outcome, and the answer will vary.

Python MySQL

Python can be used in database applications.

One of the most popular databases is MySQL.

Install MySQL Driver

Python needs a MySQL driver to access the MySQL database.

In this tutorial we will use the driver "MySQL Connector".

We recommend that you use PIP to install "MySQL Connector".

PIP is most likely already installed in your Python environment.

Navigate your command line to the location of PIP, and type the following:

Download and install "MySQL Connector":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-  
32\Scripts>python -m pip install mysql-connector-python
```

Now you have downloaded and installed a MySQL driver.

Test MySQL Connector

To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:

```
demo_mysql_test.py:
```

```
import mysql.connector
```

If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database:

```
demo_mysql_connection.py:
```

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword"
```

```
)
```

```
print(mydb)
```

```
Now you can start querying the database using SQL statements.
```

Python MySQL Create Database

Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

Example

create a database named "mydatabase":

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("CREATE DATABASE mydatabase" )
```

If the above code was executed with no errors, you have successfully created a database.

Check if Database Exists

You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:

Example

Return a list of your system's databases:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("SHOW DATABASES")
```

```
for x in mycursor:
```

```
    print (x)
```

Or you can try to access the database when making the connection:

Example

Try connecting to the database "mydatabase":

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword" ,
```

```
    database="mydatabase"
```

```
)
```

If the database does not exist, you will get an error.

Python MySQL Create Table

Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection

Example

Create a table named "customers":

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername " ,
```

```
    password="yourpassword " ,
```

```
    database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("CREATE TABLE customers (name VARCHAR(255),  
address VARCHAR(255))" )
```

If the above code was executed with no errors, you have now successfully created a table.

Check if Table Exists

You can check if a table exist by listing all tables in your database with the "SHOW TABLES" statement:

Example

Return a list of your system's databases:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase")
    
mycursor = mydb.cursor()

mycursor.execute("SHOW TABLES" )

for x in mycursor:
```

```
print(x)
```

Primary Key

When creating a table, you should also create a column with a unique key for each record.

This can be done by defining a PRIMARY KEY.

We use the statement "INT AUTO_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

Example

Create primary key when creating the table:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)

mycursor = mydb.cursor()
```

```
mycursor.execute("CREATE TABLE customers (id INT  
AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), address  
VARCHAR(255))" )
```

If the table already exists, use the ALTER TABLE keyword:

Example

Create primary key on an existing table:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
host="localhost" ,
```

```
user="yourusername" ,
```

```
password="yourpassword" ,
```

```
database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("ALTER TABLE customers ADD COLUMN id INT  
AUTO_INCREMENT PRIMARY KEY" )
```

Python MySQL Insert Into Table

Insert Into Table

To fill a table in MySQL, use the "INSERT INTO" statement.

Example

Insert a record in the "customers" table:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword" ,
```

```
    database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
```

```
val = ("John" , "Highway 21" )
```

```
mycursor.execute(sql, val)
```

```
mydb.commit()
```

```
print(mycursor.rowcount, "record inserted." )
```

Important!: Notice the statement: `mydb.commit()` . It is required to make the changes, otherwise no changes are made to the table.

Insert Multiple Rows

To insert multiple rows into a table, use the `executemany()` method.

The second parameter of the `executemany()` method is a list of tuples, containing the data you want to insert:

Example

Fill the "customers" table with data:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
host="localhost",
```

```
user="yourusername ",
```

```
password="yourpassword" ,  
database="mydatabase"  
)
```

```
mycursor = mydb.cursor()
```

```
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
```

```
val = [  
    ('Peter' , 'Lowstreet 4' ),  
    ('Amy' , 'Apple st 652' ),  
    ('Hannah' , 'Mountain 21' ),  
    ('Michael' , 'Valley 345' ),  
    ('Sandy' , 'Ocean blvd 2' ),  
    ('Betty' , 'Green Grass 1' ),  
    ('Richard' , 'Sky st 331' ),  
    ('Susan' , 'One way 98' ),  
    ('Vicky' , 'Yellow Garden 2' ),  
    ('Ben' , 'Park Lane 38' ),  
    ('William' , 'Central st 954' ),  
    ('Chuck' , 'Main Road 989' ),
```

```
('Viola' , 'Sideway 1633' )
```

```
]
```

```
mycursor.executemany(sql, val)
```

```
mydb.commit()
```

```
print (mycursor.rowcount, "was inserted." )
```

Get Inserted ID

You can get the id of the row you just inserted by asking the cursor object.

Note: If you insert more than one row, the id of the last inserted row is returned.

Example

Insert one row, and return the ID:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
host="localhost" ,  
user="yourusername" ,  
password="yourpassword" ,  
database="mydatabase"  
  
)  
  
mycursor = mydb.cursor()  
  
  
  
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"  
val = ("Michelle" , "Blue Village" )  
mycursor.execute(sql, val)  
  
  
  
mydb.commit()  
  
  
  
print ("1 record inserted, ID:" , mycursor.lastrowid)
```

Python MySQL Select From Select From a Table

To select from a table in MySQL, use the "SELECT" statement:

Example

Select all records from the "customers" table, and display the result:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword" ,
```

```
    database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("SELECT * FROM customers" )
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:
```

```
    print(x)
```

Note: We use the `fetchall()` method, which fetches all rows from the last executed statement.

Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

Example

Select only the name and address columns:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword" ,
```

```
    database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("SELECT name, address FROM customers" )
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:
```

```
    print (x)
```

Using the fetchone() Method

If you are only interested in one row, you can use the **fetchone()** method.

The **fetchone()** method will return the first row of the result:

Example

Fetch only one row:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword" ,
```

```
    database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()  
  
mycursor.execute("SELECT * FROM customers" )  
  
myresult = mycursor.fetchone()  
  
print (myresult)
```

Python MySQL Where

Select With a Filter

When selecting records from a table, you can filter the selection by using the "WHERE" statement:

Example

Select record(s) where the address is "Park Lane 38": result:

```
import mysql.connector  
  
mydb = mysql.connector.connect(  
    host="localhost" ,  
    user="yourusername" ,  
    password="yourpassword" ,
```

```
database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address ='Park Lane 38'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

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

Wildcard Characters

You can also select the records that starts, includes, or ends with a given letter or phrase.

Use the `%` to represent wildcard characters:

Example

Select records where the address contains the word "way":

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword" ,
```

```
    database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
sql = "SELECT * FROM customers WHERE address LIKE '%way%'"
```

```
mycursor.execute(sql)
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:
```

```
    print (x)
```

Prevent SQL Injection

When query values are provided by the user, you should escape the values.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module has methods to escape query values:

Example

Escape query values by using the placeholder `% s` method:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
host="localhost" ,
```

```
user="yourusername" ,
```

```
password="yourpassword" ,
```

```
database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
sql = "SELECT * FROM customers WHERE address = %s"
```

```
adr = ("Yellow Garden 2" , )
```

```
mycursor.execute(sql, adr)
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:
```

```
    print (x)
```

Python MySQL Delete From By Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

Example

Delete any record where the address is "Mountain 21":

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
host="localhost" ,  
user="yourusername" ,  
password="yourpassword" ,  
database="mydatabase"  
  
)  
  
mycursor = mydb.cursor()  
  
  
  
sql = "DELETE FROM customers WHERE address = 'Mountain 21'"  
  
  
  
mycursor.execute(sql)  
  
  
  
mydb.commit()  
  
  
  
print(mycursor.rowcount, "record(s) deleted" )
```

Important!: Notice the statement: **mydb.commit()** . It is required to make the changes, otherwise no changes are made to the table.

Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record(s) that should be deleted. If you omit the WHERE clause, all records will be deleted!

Prevent SQL Injection

It is considered a good practice to escape the values of any query, also in delete statements.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module uses the placeholder `% s` to escape values in the delete statement:

Example

Escape values by using the placeholder `% s` method:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)
```

```
mycursor = mydb.cursor()
```

```
sql = "DELETE FROM customers WHERE address = %s"  
adr = ("Yellow Garden 2" , )  
  
mycursor.execute(sql, adr)  
  
mydb.commit()  
  
print (mycursor.rowcount, "record(s) deleted" )
```

Python MySQL Drop Table Delete a Table

You can delete an existing table by using the "DROP TABLE" statement:

Example

Delete the table "customers":

```
import mysql.connector
```

```
mydb = mysql.connector.connect(  
    host="localhost" ,  
    user="yourusername" ,
```

```
password="yourpassword" ,  
database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "DROP TABLE customers"  
  
mycursor.execute(sql)
```

Drop Only if Exist

If the the table you want to delete is already deleted, or for any other reason does not exist, you can use the IF EXISTS keyword to avoid getting an error.

Example

Delete the table "customers" if it exists:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
host="localhost" ,  
user="yourusername" ,  
password="yourpassword" ,  
database="mydatabase"  
)
```

```
mycursor = mydb.cursor()
```

```
sql = "DROP TABLE IF EXISTS customers"
```

```
mycursor.execute(sql)
```

Python MySQL Update Table Update Table

You can update existing records in a table by using the "UPDATE" statement:

Example

Overwrite the address column from "Valley 345" to "Canyoun 123":

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername" ,
```

```
    password="yourpassword" ,
```

```
    database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
sql = "UPDATE customers SET address = 'Canyon 123' WHERE address =  
'Valley 345'"
```

```
mycursor.execute(sql)
```

```
mydb.commit()
```

```
print(mycursor.rowcount, "record(s) affected" )
```

Important!: Notice the statement: `mydb.commit()` . It is required to make the changes, otherwise no changes are made to the table.

Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Prevent SQL Injection

It is considered a good practice to escape the values of any query, also in update statements.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module uses the placeholder `% s` to escape values in the delete statement:

Example

Escape values by using the placeholder `% s` method:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
host="localhost" ,
```

```
user="yourusername" ,
```

```
password="yourpassword" ,
```

```
database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
  
  
sql = "UPDATE customers SET address = %s WHERE address = %s"  
val = ("Valley 345" , "Canyon 123" )  
  
  
  
mycursor.execute(sql, val)  
  
  
  
mydb.commit()  
  
  
  
print (mycursor.rowcount, "record(s) affected" )
```

Python MySQL Limit Limit the Result

You can limit the number of records returned from the query, by using the "LIMIT" statement:

Example

Select the 5 first records in the "customers" table:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers LIMIT 5" )

myresult = mycursor.fetchall()

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

Start From Another Position

If you want to return five records, starting from the third record, you can use the "OFFSET" keyword:

Example

Start from position 3, and return 5 records:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername " ,
```

```
    password="yourpassword " ,
```

```
    database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("SELECT * FROM customers LIMIT 5 OFFSET 2" )
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:
```

```
    print(x)
```

Python MySQL Join

Join Two or More Tables

You can combine rows from two or more tables, based on a related column between them, by using a JOIN statement.

Consider you have a "users" table and a "products" table:

users

```
{ id: 1, name: 'John', fav: 154},  
{ id: 2, name: 'Peter', fav: 154},  
{ id: 3, name: 'Amy', fav: 155},  
{ id: 4, name: 'Hannah', fav: },  
{ id: 5, name: 'Michael', fav: }
```

products

```
{ id: 154, name: 'Chocolate Heaven' },  
{ id: 155, name: 'Tasty Lemons' },  
{ id: 156, name: 'Vanilla Dreams' }
```

These two tables can be combined by using users' **fa v** field and products' **i d** field.

Example

Join users and products to see the name of the users favorite product:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
```

```
    host="localhost" ,
```

```
    user="yourusername " ,
```

```
    password="yourpassword " ,
```

```
    database="mydatabase"
```

```
)
```

```
mycursor = mydb.cursor()
```

```
sql = "SELECT \
```

```
users.name AS user, \
```

```
products.name AS favorite \
```

```
FROM users \
```

```
INNER JOIN products ON users.fav = products.id"
```

```
mycursor.execute(sql)
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:
```

```
    print(x)
```

Note: You can use JOIN instead of INNER JOIN. They will both give you the same result.

LEFT JOIN

In the example above, Hannah, and Michael were excluded from the result, that is because INNER JOIN only shows the records where there is a match.

If you want to show all users, even if they do not have a favorite product, use the LEFT JOIN statement:

Example

Select all users and their favorite product:

```
sql = "SELECT \
users.name AS user, \
products.name AS favorite \
```

```
FROM users \
LEFT JOIN products ON users.fav = products.id"
```

RIGHT JOIN

If you want to return all products, and the users who have them as their favorite, even if no user have them as their favorite, use the RIGHT JOIN statement:

Example

Select all products, and the user(s) who have them as their favorite:

```
sql = "SELECT \
users.name AS user, \
products.name AS favorite \
FROM users \
RIGHT JOIN products ON users.fav = products.id"
```

Python MongoDB

Python can be used in database applications.

One of the most popular NoSQL database is MongoDB.

MongoDB

MongoDB stores data in JSON-like documents, which makes the database very flexible and scalable.

To be able to experiment with the code examples in this tutorial, you will need access to a MongoDB database.

PyMongo

Python needs a MongoDB driver to access the MongoDB database.

In this tutorial we will use the MongoDB driver "PyMongo".

We recommend that you use PIP to install "PyMongo".

PIP is most likely already installed in your Python environment.

Navigate your command line to the location of PIP, and type the following:

Download and install "PyMongo":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-  
32\Scripts>python -m pip install pymongo
```

Now you have downloaded and installed a mongoDB driver.

Test PyMongo

To test if the installation was successful, or if you already have "pymongo" installed, create a Python page with the following content:

```
demo_mongodb_test.py:
```

```
import pymongo
```

If the above code was executed with no errors, "pymongo" is installed and ready to be used.

Creating a Database

To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

MongoDB will create the database if it does not exist, and make a connection to it.

Example

```
Create a database called "mydatabase":
```

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

Important: In MongoDB, a database is not created until it gets content!

MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

Check if Database Exists

Remember: In MongoDB, a database is not created until it gets content, so if this is your first time creating a database, you should complete the next two chapters (create collection and create document) before you check if the database exists!

You can check if a database exist by listing all databases in you system:

Example

Return a list of your system's databases:

```
print (myclient.list_database_names())
```

Or you can check a specific database by name:

Example

Check if "mydatabase" exists:

```
dblist = myclient.list_database_names()
```

```
if "mydatabase" in dblist:
```

```
print ("The database exists." )
```

A collection in MongoDB is the same as a table in SQL databases.

Creating a Collection

To create a collection in MongoDB, use database object and specify the name of the collection you want to create.

MongoDB will create the collection if it does not exist.

Example

Create a collection called "customers":

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

Important: In MongoDB, a collection is not created until it gets content!

MongoDB waits until you have inserted a document before it actually creates the collection.

Check if Collection Exists

Remember: In MongoDB, a collection is not created until it gets content, so if this is your first time creating a collection, you should complete the next chapter (create document) before you check if the collection exists!

You can check if a collection exist in a database by listing all collections:

Example

Return a list of all collections in your database:

```
print (mydb.list_collection_names())
```

Or you can check a specific collection by name:

Example

Check if the "customers" collection exists:

```
collist = mydb.list_collection_names()
```

```
if "customers" in collist:
```

```
    print ("The collection exists. ")
```

Insert Into Collection

To insert a record, or *document* as it is called in MongoDB, into a collection, we use the `insert_one()` method.

The first parameter of the `insert_one()` method is a dictionary containing the name(s) and value(s) of each field in the document you want to insert.

Example

Insert a record in the "customers" collection:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
mydb = myclient["mydatabase"]
```

```
mycol = mydb["customers"]
```

```
mydict = { "name": "John", "address": "Highway 37" }
```

```
x = mycol.insert_one(mydict)
```

Return the `_id` Field

The `insert_one()` method returns a `InsertOneResult` object, which has a property, `inserted_id`, that holds the id of the inserted document.

Example

Insert another record in the "customers" collection, and return the value of the `_id` field:

```
mydict = { "name" : "Peter" , "address" : "Lowstreet 27" }
```

```
x = mycol.insert_one(mydict)
```

```
print (x.inserted_id)
```

If you do not specify an `_id` field, then MongoDB will add one for you and assign a unique id for each document.

In the example above no `_id` field was specified, so MongoDB assigned a unique `_id` for the record (document).

Insert Multiple Documents

To insert multiple documents into a collection in MongoDB, we use the `insert_many()` method.

The first parameter of the `insert_many()` method is a list containing dictionaries with the data you want to insert:

Example

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
mylist = [  
  
    { "name" : "Amy" , "address" : "Apple st 652" },  
  
    { "name" : "Hannah" , "address" : "Mountain 21" },  
  
    { "name" : "Michael" , "address" : "Valley 345" },  
  
    { "name" : "Sandy" , "address" : "Ocean blvd 2" },  
  
    { "name" : "Betty" , "address" : "Green Grass 1" },  
  
    { "name" : "Richard" , "address" : "Sky st 331" },  
  
    { "name" : "Susan" , "address" : "One way 98" },  
  
    { "name" : "Vicky" , "address" : "Yellow Garden 2" },  
  
    { "name" : "Ben" , "address" : "Park Lane 38" },  
  
    { "name" : "William" , "address" : "Central st 954" },  
  
    { "name" : "Chuck" , "address" : "Main Road 989" },  
  
    { "name" : "Viola" , "address" : "Sideway 1633" }  
  
]  
  
x = mycol.insert_many(mylist)
```

```
#print list of the _id values of the inserted documents:
```

```
print (x.inserted_ids)
```

The `insert_many()` method returns a `InsertManyResult` object, which has a property, `inserted_ids`, that holds the ids of the inserted documents.

Insert Multiple Documents, with Specified IDs

If you do not want MongoDB to assign unique ids for your document, you can specify the `_id` field when you insert the document(s).

Remember that the values have to be unique. Two documents cannot have the same `_id`.

Example

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
mydb = myclient["mydatabase"]
```

```
mycol = mydb["customers"]
```

```
mylist = [
```

```
    { "_id" : 1 , "name" : "John" , "address" : "Highway 37" },
```

```
{ "_id" : 2 , "name" : "Peter" , "address" : "Lowstreet 27" },  
 { "_id" : 3 , "name" : "Amy" , "address" : "Apple st 652" },  
 { "_id" : 4 , "name" : "Hannah" , "address" : "Mountain 21" },  
 { "_id" : 5 , "name" : "Michael" , "address" : "Valley 345" },  
 { "_id" : 6 , "name" : "Sandy" , "address" : "Ocean blvd 2" },  
 { "_id" : 7 , "name" : "Betty" , "address" : "Green Grass 1" },  
 { "_id" : 8 , "name" : "Richard" , "address" : "Sky st 331" },  
 { "_id" : 9 , "name" : "Susan" , "address" : "One way 98" },  
 { "_id" : 10 , "name" : "Vicky" , "address" : "Yellow Garden 2" },  
 { "_id" : 11 , "name" : "Ben" , "address" : "Park Lane 38" },  
 { "_id" : 12 , "name" : "William" , "address" : "Central st 954" },  
 { "_id" : 13 , "name" : "Chuck" , "address" : "Main Road 989" },  
 { "_id" : 14 , "name" : "Viola" , "address" : "Sideway 1633" }  
 ]
```

```
x = mycol.insert_many(mylist)
```

```
#print list of the _id values of the inserted documents:
```

```
print (x.inserted_ids)
```

In MongoDB we use the find and findOne methods to find data in a collection.

Just like the SELECT statement is used to find data in a table in a MySQL database.

Find One

To select data from a collection in MongoDB, we can use the `find_one()` method.

The `find_one()` method returns the first occurrence in the selection.

Example

Find the first document in the customers collection:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
x = mycol.find_one()
```

```
print (x)
```

Find All

To select data from a table in MongoDB, we can also use the **find()** method.

The **find()** method returns all occurrences in the selection.

The first parameter of the **find()** method is a query object. In this example we use an empty query object, which selects all documents in the collection.

No parameters in the **find()** method gives you the same result as **SELECT *** in MySQL.

Example

Return all documents in the "customers" collection, and print each document:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
for x in mycol.find():
```

```
    print (x)
```

Return Only Some Fields

The second parameter of the `find()` method is an object describing which fields to include in the result.

This parameter is optional, and if omitted, all fields will be included in the result.

Example

Return only the names and addresses, not the `_ids`:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
for x in mycol.find({}, { "_id" : 0 , "name" : 1 , "address" : 1 }):
```

```
    print (x)
```

You are not allowed to specify both 0 and 1 values in the same object (except if one of the fields is the `_id` field). If you specify a field with the value 0, all other fields get the value 1, and vice versa:

Example

This example will exclude "address" from the result:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )  
mydb = myclient["mydatabase" ]  
mycol = mydb["customers" ]
```

```
for x in mycol.find({},{ "address" : 0 }):  
    print (x)
```

Example

You get an error if you specify both 0 and 1 values in the same object (except if one of the fields is the _id field):

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )  
mydb = myclient["mydatabase" ]  
mycol = mydb["customers" ]
```

```
for x in mycol.find({},{ "name": 1 , "address": 0 }):
    print (x)
```

Filter the Result

When finding documents in a collection, you can filter the result by using a query object.

The first argument of the `find()` method is a query object, and is used to limit the search.

Example

Find document(s) with the address "Park Lane 38":

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
myquery = { "address": "Park Lane 38" }
```

```
mydoc = mycol.find(myquery)
```

```
for x in mydoc:
```

```
    print (x)
```

Advanced Query

To make advanced queries you can use modifiers as values in the query object.

E.g. to find the documents where the "address" field starts with the letter "S" or higher (alphabetically), use the greater than modifier: `{"$gt": "S"}` :

Example

Find documents where the address starts with the letter "S" or higher:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
mydb = myclient["mydatabase"]
```

```
mycol = mydb["customers"]
```

```
myquery = { "address" : { "$gt" : "S" } }
```

```
mydoc = mycol.find(myquery)
```

```
for x in mydoc:
```

```
    print (x)
```

Filter With Regular Expressions

You can also use regular expressions as a modifier.

Regular expressions can only be used to query *strings*.

To find only the documents where the "address" field starts with the letter "S", use the regular expression `{"$regex": "^S" }` :

Example

Find documents where the address starts with the letter "S":

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
myquery = { "address" : { "$regex" : "^S" } }
```

```
mydoc = mycol.find(myquery)
```

```
for x in mydoc:
```

```
    print (x)
```

Sort the Result

Use the `sort()` method to sort the result in ascending or descending order.

The `sort()` method takes one parameter for "fieldname" and one parameter for "direction" (ascending is the default direction).

Example

Sort the result alphabetically by name:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
mydoc = mycol.find().sort("name" )
```

```
for x in mydoc:
```

```
print (x)
```

Sort Descending

Use the value -1 as the second parameter to sort descending.

```
sort("name", 1) #ascending
```

```
sort("name", -1) #descending
```

Example

Sort the result reverse alphabetically by name:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
mydoc = mycol.find().sort("name" , -1 )
```

```
for x in mydoc:
```

```
print (x)
```

Delete Document

To delete one document, we use the `delete_one()` method.

The first parameter of the `delete_one()` method is a query object defining which document to delete.

Note: If the query finds more than one document, only the first occurrence is deleted.

Example

Delete the document with the address "Mountain 21":

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
myquery = { "address" : "Mountain 21" }
```

```
mycol.delete_one(myquery)
```

Delete Many Documents

To delete more than one document, use the `delete_many()` method.

The first parameter of the `delete_many()` method is a query object defining which documents to delete.

Example

Delete all documents where the address starts with the letter S:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
mydb = myclient["mydatabase"]
```

```
mycol = mydb["customers"]
```

```
myquery = { "address" : { "$regex" : "^S" } }
```

```
x = mycol.delete_many(myquery)
```

```
print(x.deleted_count, " documents deleted.")
```

Delete All Documents in a Collection

To delete all documents in a collection, pass an empty query object to the `delete_many()` method:

Example

Delete all documents in the "customers" collection:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
x = mycol.delete_many({})
```

```
print (x.deleted_count, " documents deleted." )
```

Delete Collection

You can delete a table, or collection as it is called in MongoDB, by using the `drop()` method.

Example

Delete the "customers" collection:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
mycol.drop()
```

The `drop()` method returns true if the collection was dropped successfully, and false if the collection does not exist.

Delete Collection

You can delete a table, or collection as it is called in MongoDB, by using the `drop()` method.

Example

Delete the "customers" collection:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase"]
```

```
mycol = mydb["customers"]
```

```
mycol.drop()
```

The `drop()` method returns true if the collection was dropped successfully, and false if the collection does not exist.

Limit the Result

To limit the result in MongoDB, we use the `limit()` method.

The `limit()` method takes one parameter, a number defining how many documents to return.

Consider you have a "customers" collection:

Customers

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
```

```
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
```

```
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
```

```
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
```

```
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
```

```
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
```

```
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
```

```
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
```

```
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
```

```
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
```

```
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
```

```
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
```

```
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
```

```
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
```

Example

Limit the result to only return 5 documents:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/" )
```

```
mydb = myclient["mydatabase" ]
```

```
mycol = mydb["customers" ]
```

```
myresult = mycol.find().limit(5 )
```

```
#print the result:
```

```
for x in myresult:
```

```
print (x)
```

Python Reference

Python Built in Functions

Python has a set of built-in functions.

Function	Description
abs()	Returns the absolute value of a number
all()	Returns True if all items in an iterable object are true
any()	Returns True if any item in an iterable object is true
ascii()	Returns a readable version of an object. Replaces none-ascii characters with escape character
bin()	Returns the binary version of a number
bool()	Returns the boolean value of the specified object

bytearray()	Returns an array of bytes
bytes()	Returns a bytes object
callable()	Returns True if the specified object is callable, otherwise False
chr()	Returns a character from the specified Unicode code.
classmethod()	Converts a method into a class method
compile()	Returns the specified source as an object, ready to be executed
complex()	Returns a complex number
delattr()	Deletes the specified attribute (property or

method) from the specified object

`dict()`

Returns a dictionary (Array)

`dir()`

Returns a list of the specified object's properties and methods

`divmod()`

Returns the quotient and the remainder when argument1 is divided by argument2

`enumerate()`

Takes a collection (e.g. a tuple) and returns it as an enumerate object

`eval()`

Evaluates and executes an expression

`exec()`

Executes the specified code

(or object)

filter()	Use a filter function to exclude items in an iterable object
float()	Returns a floating point number
format()	Formats a specified value
frozenset()	Returns a frozenset object
getattr()	Returns the value of the specified attribute (property or method)
globals()	Returns the current global symbol table as a dictionary
hasattr()	Returns True if the specified object has

the specified attribute
(property/method)

`hash()` Returns the hash value of a specified object

`help()` Executes the built-in help system

`hex()` Converts a number into a hexadecimal value

`id()` Returns the id of an object

`input()` Allowing user input

`int()` Returns an integer number

`isinstance()` Returns True if a specified object is an instance of a specified object

issubclass()	Returns True if a specified class is a subclass of a specified object
iter()	Returns an iterator object
len()	Returns the length of an object
list()	Returns a list
locals()	Returns an updated dictionary of the current local symbol table
map()	Returns the specified iterator with the specified function applied to each item
max()	Returns the largest item in an iterable

memoryview()	Returns a memory view object
min()	Returns the smallest item in an iterable
next()	Returns the next item in an iterable
object()	Returns a new object
oct()	Converts a number into an octal
open()	Opens a file and returns a file object
ord()	Convert an integer representing the Unicode of the specified character
pow()	Returns the value of x to

the power of y

`print()`

Prints to the standard output device

`property()`

Gets, sets, deletes a property

`range()`

Returns a sequence of numbers, starting from 0 and increments by 1 (by default)

`repr()`

Returns a readable version of an object

`reversed()`

Returns a reversed iterator

`round()`

Rounds a numbers

`set()`

Returns a new set object

setattr()	Sets an attribute (property/method) of an object
slice()	Returns a slice object
sorted()	Returns a sorted list
@staticmethod()	Converts a method into a static method
str()	Returns a string object
sum()	Sums the items of an iterator
super()	Returns an object that represents the parent class
tuple()	Returns a tuple
type()	Returns the type of an object

`vars()` Returns the `__dict__` property of an object

`zip()` Returns an iterator, from two or more iterators

Python String Methods

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

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

Method	Description
<code>capitalize()</code>	Converts the first character to upper case
<code>casefold()</code>	Converts string into lower case
<code>center()</code>	Returns a centered string
<code>count()</code>	Returns the number of times a specified value occurs in a string
<code>encode()</code>	Returns an encoded version of the string
<code>endswith()</code>	Returns true if the string ends with the specified value
<code>expandtabs()</code>	Sets the tab size of the string
<code>find()</code>	Searches the string for a specified value and returns the position of where it was found
<code>format()</code>	Formats specified values in a string
<code>format_map()</code>	Formats specified values in a string
<code>index()</code>	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
rfind()	Searches the string for a specified value and returns the last position of where it was found
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
zfill()	Fills the string with a specified number of 0 values at the beginning

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

Learn more about strings in our [Python Strings Tutorial](#).

Python List/Array Methods

Python List/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
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

Python Dictionary Methods

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

Method	Description
clear()	Removes all the elements from the dictionary
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

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

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)

`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

<code>union()</code>	Return a set containing the union of sets
<code>update()</code>	Update the set with the union of this set and others

Python File Methods

Python has a set of methods available for the file object.

Method	Description
<code>close()</code>	Closes the file
<code>detach()</code>	Returns the separated raw stream from the buffer
<code>fileno()</code>	Returns a number that represents the stream, from the operating system's perspective
<code>flush()</code>	Flushes the internal buffer
<code>isatty()</code>	Returns whether the file stream is interactive or not
<code>read()</code>	Returns the file content
<code>readable()</code>	Returns whether the file stream can be read or not
<code>readline()</code>	Returns one line from the file
<code>readlines()</code>	Returns a list of lines from the file
<code>seek()</code>	Change the file position
<code>seekable()</code>	Returns whether the file allows us to change the file position
<code>tell()</code>	Returns the current file position
<code>truncate()</code>	Resizes the file to a specified size
<code>writable()</code>	Returns whether the file can be written to or not
<code>write()</code>	Writes the specified string to the file
<code>writelines()</code>	Writes a list of strings to the file

Python Keywords

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers:

Keyword	Description
<code>and</code>	A logical operator
<code>as</code>	To create an alias
<code>assert</code>	For debugging
<code>break</code>	To break out of a loop
<code>class</code>	To define a class
<code>continue</code>	To continue to the next iteration of a loop

def To define a function
del To delete an object
elif Used in conditional statements, same as else if
else Used in conditional statements
except Used with exceptions, what to do when an exception occurs
False Boolean value, result of comparison operations
finally Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for To create a for loop
from To import specific parts of a module
global To declare a global variable
if To make a conditional statement
import To import a module
in To check if a value is present in a list, tuple, etc.
is To test if two variables are equal
lambda To create an anonymous function
None Represents a null value
nonlocal To declare a non-local variable
not A logical operator
or A logical operator
pass A null statement, a statement that will do nothing
raise To raise an exception
return To exit a function and return a value
True Boolean value, result of comparison operations
try To make a try...except statement
while To create a while loop
with Used to simplify exception handling
yield To end a function, returns a generator

Python Built-in Exceptions

Built-in Exceptions

The table below shows built-in exceptions that are usually raised in Python:

Exception	Description
ArithmetError	Raised when an error occurs in numeric calculations
AssertionError	Raised when an assert statement fails
AttributeError	Raised when attribute reference or assignment fails
Exception	Base class for all exceptions
EOFError	Raised when the input() method hits an "end of file" condition (EOF)
FloatingPointError	Raised when a floating point calculation fails
GeneratorExit	Raised when a generator is closed (with the close() method)
ImportError	Raised when an imported module does not exist
IndentationError	Raised when indentation is not correct
IndexError	Raised when an index of a sequence does not exist
KeyError	Raised when a key does not exist in a dictionary
KeyboardInterrupt	Raised when the user presses Ctrl+c, Ctrl+z or Delete
LookupError	Raised when errors raised cant be found
MemoryError	Raised when a program runs out of memory
NameError	Raised when a variable does not exist
NotImplementedE rror	Raised when an abstract method requires an inherited class to override the method
OSError	Raised when a system related operation causes an

	error
OverflowError	Raised when the result of a numeric calculation is too large
ReferenceError	Raised when a weak reference object does not exist
RuntimeError	Raised when an error occurs that do not belong to any specific exceptions
StopIteration	Raised when the next() method of an iterator has no further values
SyntaxError	Raised when a syntax error occurs
TabError	Raised when indentation consists of tabs or spaces
SystemError	Raised when a system error occurs
SystemExit	Raised when the sys.exit() function is called
TypeError	Raised when two different types are combined
UnboundLocalError	Raised when a local variable is referenced before assignment
UnicodeError	Raised when a unicode problem occurs
UnicodeEncodeError	Raised when a unicode encoding problem occurs
UnicodeDecodeError	Raised when a unicode decoding problem occurs
UnicodeTranslateError	Raised when a unicode translation problem occurs
ValueError	Raised when there is a wrong value in a specified data type

`ZeroDivisionError` Raised when the second operator in a division is zero

Module Reference

Python has a built-in module that you can use to make random numbers.

The `random` module has a set of methods:

Method	Description
<code>seed()</code>	Initialize the random number generator
<code>Getstate()</code>	Returns the current internal state of the random number generator
<code>setstate()</code>	Restores the internal state of the random number generator
<code>getrandbits()</code>	Returns a number representing the random bits
<code>randrange()</code>	Returns a random number between the given range
<code>randint()</code>	Returns a random number between the given range
<code>choice()</code>	Returns a random element from the given sequence
<code>choices()</code>	Returns a list with a random selection from the given sequence
<code>shuffle()</code>	Takes a sequence and returns the sequence in a random order
<code>sample()</code>	Returns a given sample of a sequence
<code>random()</code>	Returns a random float number between 0 and 1

uniform()	Returns a random float number between two given parameters
triangular()	Returns a random float number between two given parameters, you can also set a mode parameter to specify the midpoint between the two other parameters
betavariate()	Returns a random float number between 0 and 1 based on the Beta distribution (used in statistics)
expovariate()	Returns a random float number based on the Exponential distribution (used in statistics)
gammavariate() ()	Returns a random float number based on the Gamma distribution (used in statistics)
gauss()	Returns a random float number based on the Gaussian distribution (used in probability theories)
lognormvariat e()	Returns a random float number based on a log-normal distribution (used in probability theories)
normalvariate ()	Returns a random float number based on the normal distribution (used in probability theories)
vonmisesvari ate()	Returns a random float number based on the von Mises distribution (used in directional statistics)
paretovariate()	Returns a random float number based on the Pareto distribution (used in probability theories)
weibullvariat e()	Returns a random float number based on the Weibull distribution (used in statistics)

Python Requests Module

Example

Make a request to a web page, and print the response text:

```
import requests
```

```
x = requests.get('https://w3schools.com/python/demopage.htm' )
```

```
print (x.text)
```

Definition and Usage

The `request s` module allows you to send HTTP requests using Python.

The HTTP request returns a Response Object with all the response data (content, encoding, status, etc).

Download and Install the Requests Module

Navigate your command line to the location of PIP, and type the following:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-  
32\Scripts>pip install requests
```

Syntax

`requests.methodname(params)`

Methods

Method	Description
<code>delete(url , args)</code>	Sends a DELETE request to the specified url
<code>get(url , params, args)</code>	Sends a GET request to the specified url
<code>head(url , args)</code>	Sends a HEAD request to the specified url
<code>patch(url , data, args)</code>	Sends a PATCH request to the specified url
<code>post(url , data, json, args)</code>	Sends a POST request to the specified url
<code>put(url , data, args)</code>	Sends a PUT request to the specified url
<code>request(method , url , args)</code>	Sends a request of the specified method to the specified url

Python statistics Module

Python statistics Module

Python has a built-in module that you can use to calculate mathematical statistics of numeric data.

The `statistics` module was new in Python 3.4.

Statistics Methods

Method	Description
<code>statistics.harmonic_mean()</code>	Calculates the harmonic mean (central location) of the given data
<code>statistics.mean()</code>	Calculates the mean (average) of the given data
<code>statistics.median()</code>	Calculates the median (middle value) of the given data
<code>statistics.median_grouped()</code>	Calculates the median of grouped continuous data
<code>statistics.median_high()</code>	Calculates the high median of the given data
<code>statistics.median_low()</code>	Calculates the low median of the given data
<code>statistics.mode()</code>	Calculates the mode (central tendency) of the given numeric or nominal data
<code>statistics.pstdev()</code>	Calculates the standard deviation from an entire population
<code>statistics.stdev()</code>	Calculates the standard deviation from a sample of data
<code>statistics.pvariance()</code>	Calculates the variance of an entire population
<code>statistics.variance()</code>	Calculates the variance from a sample of data

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 Methods

Method	Description
<code>math.acos()</code>	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
`math.copysign()` Returns a float consisting of the value of the first parameter and the sign of the second parameter
`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
`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

<code>math.isqrt()</code>	Rounds a square root number downwards to the nearest integer
<code>math.ldexp()</code>	Returns the inverse of <code>math.frexp()</code> which is $x * (2**i)$ of the given numbers x and i
<code>math.lgamma()</code>	Returns the log gamma value of x
<code>math.log()</code>	Returns the natural logarithm of a number, or the logarithm of number to base
<code>math.log10()</code>	Returns the base-10 logarithm of x
<code>math.log1p()</code>	Returns the natural logarithm of $1+x$
<code>math.log2()</code>	Returns the base-2 logarithm of x
<code>math.perm()</code>	Returns the number of ways to choose k items from n items with order and without repetition
<code>math.pow()</code>	Returns the value of x to the power of y
<code>math.prod()</code>	Returns the product of all the elements in an iterable
<code>math.radians()</code>	Converts a degree value into radians
<code>math.remainder()</code>	Returns the closest value that can make numerator completely divisible by the denominator
<code>math.sin()</code>	Returns the sine of a number
<code>math.sinh()</code>	Returns the hyperbolic sine of a number
<code>math.sqrt()</code>	Returns the square root of a number
<code>math.tan()</code>	Returns the tangent of a number
<code>math.tanh()</code>	Returns the hyperbolic tangent of a number
<code>math.trunc()</code>	Returns the truncated integer parts of a number

Math Constants

Constant	Description
<code>math.e</code>	Returns Euler's number (2.7182...)
<code>math.inf</code>	Returns a floating-point positive infinity
<code>math.nan</code>	Returns a floating-point NaN (Not a Number) value
<code>math.pi</code>	Returns PI (3.1415...)
<code>math.tau</code>	Returns tau (6.2831...)

Python cmath Module

Python has a built-in module that you can use for mathematical tasks for complex numbers.

The methods in this module accept int, float, and complex numbers. It even accepts Python objects that have a `__complex__()` or `__float__()` method.

The methods in this module almost always return a complex number. If the return value can be expressed as a real number, the return value has an imaginary part of 0.

The cmath module has a set of methods and constants.

cMath Methods

Method Description

<code>cmath.acos(x)</code>	Returns the arc cosine value of x
<code>cmath.acosh(x)</code>	Returns the hyperbolic arc cosine of x
<code>cmath.asin(x)</code>	Returns the arc sine of x
<code>cmath.asinh(x)</code>	Returns the hyperbolic arc sine of x
<code>cmath.atan(x)</code>	Returns the arc tangent value of x
<code>cmath.atanh(x)</code>	Returns the hyperbolic arctangent value of x
<code>cmath.cos(x)</code>	Returns the cosine of x
<code>cmath.cosh(x)</code>	Returns the hyperbolic cosine of x
<code>cmath.exp(x)</code>	Returns the value of Ex, where E is Euler's number (approximately 2.718281...), and x is the number passed to it
<code>cmath.isclose()</code>	Checks whether two values are close, or not
<code>cmath.isfinite(x)</code>	Checks whether x is a finite number
<code>cmath.isinf(x)</code>	Check whether x is a positive or negative infinity
<code>cmath.isnan(x)</code>	Checks whether x is NaN (not a number)
<code>cmath.log(x[, base])</code>	Returns the logarithm of x to the base
<code>cmath.log10(x)</code>	Returns the base-10 logarithm of x
<code>cmath.phase()</code>	Return the phase of a complex number
<code>cmath.polar()</code>	Convert a complex number to polar coordinates
<code>cmath.rect()</code>	Convert polar coordinates to rectangular form
<code>cmath.sin(x)</code>	Returns the sine of x
<code>cmath.sinh(x)</code>	Returns the hyperbolic sine of x
<code>cmath.sqrt(x)</code>	Returns the square root of x
<code>cmath.tan(x)</code>	Returns the tangent of x
<code>cmath.tanh(x)</code>	Returns the hyperbolic tangent of x

cMath Constants

Constant	Description
cmath.e	Returns Euler's number (2.7182...)
cmath.inf	Returns a floating-point positive infinity value
cmath.infj	Returns a complex infinity value
cmath.nan	Returns floating-point NaN (Not a Number) value
cmath.nanj	Returns coplext NaN (Not a Number) value
cmath.pi	Returns PI (3.1415...)
cmath.tau	Returns tau (6.2831...)

How to Remove Duplicates From a Python List

Learn how to remove duplicates from a List in Python.

Example

Remove any duplicates from a List:

```
mylist = ["a" , "b" , "a" , "c" , "c" ]  
mylist = list(dict.fromkeys(mylist))  
print (mylist)
```

Example Explained

First we have a List that contains duplicates:

A List with Duplicates

```
mylist = ["a" , "b" , "a" , "c" , "c" ]  
mylist = list(dict.fromkeys(mylist))  
print (mylist)
```

Create a dictionary, using the List items as keys. This will automatically remove any duplicates because dictionaries cannot have duplicate keys.

Create a Dictionary

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

```
mylist = list( dict .fromkeys(mylist) )
```

```
print (mylist)
```

Then, convert the dictionary back into a list:

Convert Into a List

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

```
mylist = list( dict .fromkeys(mylist) )
```

```
print (mylist)
```

Now we have a List without any duplicates, and it has the same order as the original List.

Print the List to demonstrate the result

Print the List

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

```
mylist = list(dict .fromkeys(mylist))
```

```
print (mylist)
```

Create a Function

If you like to have a function where you can send your lists, and get them back without duplicates, you can create a function and insert the code from the example above.

Example

```
def my_function(x):  
    return list (dict .fromkeys(x))  
  
mylist = my_function(["a" , "b" , "a" , "c" , "c" ])  
  
print (mylist)
```

Example Explained

Create a function that takes a List as an argument.

Create a Function

```
def my_function(x):  
    return list (dict .fromkeys(x))  
  
mylist = my_function(["a" , "b" , "a" , "c" , "c" ])
```

```
print (mylist)
```

Create a dictionary, using this List items as keys.

Create a Dictionary

```
def my_function(x):  
    return list ( dict .fromkeys(x) )
```

```
mylist = my_function(["a" , "b" , "a" , "c" , "c" ])
```

```
print (mylist)
```

Convert the dictionary into a list.

Convert Into a List

```
def my_function(x):  
    return list ( dict .fromkeys(x) )
```

```
mylist = my_function(["a" , "b" , "a" , "c" , "c" ])
```

```
print (mylist)
```

Return the list

Return List

```
def my_function(x):  
    return list (dict .fromkeys(x))
```

```
mylist = my_function(["a" , "b" , "a" , "c" , "c" ])
```

```
print (mylist)
```

Call the function, with a list as a parameter:

Call the Function

```
def my_function(x):  
    return list (dict .fromkeys(x))
```

```
mylist = my_function(["a" , "b" , "a" , "c" , "c" ])
```

```
print (mylist)
```

Print the result:

Print the Result

```
def my_function(x):
```

```
return list (dict .fromkeys(x))
```

```
mylist = my_function(["a" , "b" , "a" , "c" , "c" ])
```

```
print (mylist)
```

How to Reverse a String in Python

Learn how to reverse a String in Python.

There is no built-in function to reverse a String in Python.

The fastest (and easiest?) way is to use a slice that steps backwards, `-1`.

Example

Reverse the string "Hello World":

```
txt = "Hello World" [::-1]  
print (txt)
```

Example Explained

We have a string, "Hello World", which we want to reverse:

The String to Reverse

```
txt = "Hello World" [::-1]  
print (txt)
```

Create a slice that starts at the end of the string, and moves backwards.

In this particular example, the slice statement `[::-1]` means start at the end of the string and end at position 0, move with the step `-1`, *negative* one, which means one step backwards.

Slice the String

```
txt = "Hello World" [::-1 ]
```

```
print (txt)
```

Now we have a string `tx t` that reads "Hello World" backwards.

Print the String to demonstrate the result

Print the List

```
txt = "Hello World" [::-1 ]
```

```
print (txt)
```

Create a Function

If you like to have a function where you can send your strings, and return them backwards, you can create a function and insert the code from the example above.

Example

```
def my_function(x):  
    return x[::-1 ]
```

```
mytxt = my_function("I wonder how this text looks like backwards" )
```

```
print (mytxt)
```

Example Explained

Create a function that takes a String as an argument.

Create a Function

```
def my_function(x):  
    return x[::-1]
```

```
mytxt = my_function("I wonder how this text looks like backwards" )
```

```
print (mytxt)
```

Slice the string starting at the end of the string and move backwards.

Slice the String

```
def my_function(x):  
    return x[::-1]
```

```
mytxt = my_function("I wonder how this text looks like backwards" )
```

```
print (mytxt)
```

Return the backward String

Return the String

```
def my_function(x):
```

```
    return x[::-1]
```

```
mytxt = my_function("I wonder how this text looks like backwards" )
```

```
print (mytxt )
```

Call the function, with a string as a parameter:

Call the Function

```
def my_function(x):
```

```
    return x[::-1]
```

```
mytxt = my_function("I wonder how this text looks like backwards" )
```

```
print (mytxt)
```

Print the result:

Print the Result

```
def my_function(x):
```

```
    return x[::-1]
```

```
mytxt = my_function("I wonder how this text looks like backwards" )
```

```
print (mytxt)
```

How to Add Two Numbers in Python

Learn how to add two numbers in Python.

Use the `+` operator to add two numbers:

Example

```
x = 5
```

```
y = 10
```

```
print (x + y)
```

Add Two Numbers with User Input

In this example, the user must input two numbers. Then we print the sum by calculating (adding) the two numbers:

Example

```
x = input ("Type a number: " )
```

```
y = input ("Type another number: " )
```

```
sum = int (x) + int (y)
```

```
print ("The sum is: " , sum )
```

The End