**Course Data**

1. **Python**

**#What is it?:** Python is an easy to learn, powerful programming language. It is used to create web applications. It was created by Guido Van Rossum and released in 1991.

**#Why to learn python (It’s Features):** It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

It is used for: Web development, software development, mathematics, server scripting and a lot more....

**#About this tutorial:** This tutorial is designed to help you learn python easily and introduce you to it’s basic concepts.

Please keep in mind that this is not an extensive tutorial, rather an overview to learning python

The tutorial spans 3 levels based on complexity of topics.

Have a great time learning!

**#Table of contents:**

Level1:

Installing Python

Syntax

Variables and Data types

Operators

Data structures

Conditional Statements and Loops

Level 2:

Functions

Classes and Objects

Inheritance

Exception Handling

Level 3:

Python modules

File Handling

**Level 1:**

1. **Installing Python**

Many PCs and Macs will have python already installed.

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

python –version

If you do not have it installed, just go to the below website and get it for free!

<https://www.python.org/>

The installation does not require any hassle.

Python is an interpreted language, and one should save the files with .py extension. Just write “python filename.py” to execute the file on your cmd

If one wants to test a small amount of code without creating python files, just type: “python” in the cmd, this starts the python command line, where you can continue to code.

Just type exit() to exit out of it.

1. **Syntax**

Python syntax can be executed by writing directly in the Command Line

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

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.

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

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

x = 5

Python has no command for declaring a variable.

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:

1. **Variables and Data types**

Python has no command for declaring a variable.

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

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

“print” is used to print a line of text. It is an output variable.

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

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

x = str(3) # x will be '3'  
y = int(3) # y will be 3

Python has several in built data types as:

|  |  |
| --- | --- |
| 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 |

Use type() function to get the data type of variable

1. **Operators**

Python divides the operators in the following groups:

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

#Please insert the operators list as per your convenience

1. **Collection Data Types**

Python has 4 collection data types called list, set, tuple and dictionaries.

## 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](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

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

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.

## 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](https://www.w3schools.com/python/python_lists.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

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

Tuples are written with round brackets.

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

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.

## 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](https://www.w3schools.com/python/python_lists.asp), [Tuple](https://www.w3schools.com/python/python_tuples.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A set is a collection which is *unordered*, *unchangeable\**, and *unindexed*.

**\* Note:** Set *items* are unchangeable, but you can remove items and add new items.

Sets are written with curly brackets.

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

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

## Dictionary

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

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

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

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

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

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

1. **Conditional Statements and Loops**

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")

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

The else keyword catches anything which isn't caught by the preceding conditions.

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

## 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)

**Level 2:**

1. **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()

1. **Classes and 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)

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()

1. **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()

1. **Exception Handling**

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

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")

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")

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")

**Level 3:**

1. **Python modules**

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)

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")

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)

1. **File Handling**

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

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)

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")

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())

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())

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()

## 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())

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")