**PYTHON MANUAL**

**TABLE OF CONTENTS**

1. General Python Overview

2. Installing Python

3. Variables & Data Types

4. Strings & Formatting

5. Lists

6. Tuples and Set

7. Dictionaries

8. Arithmetic, Comparison & Logical Operators

9. Functions

10. Conditionals

11. Loops

12. Modules

13. Classes

14. Files

**1. GENERAL PYTHON OVERVIEW**

binding Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python is a general purpose programming language developed by a dutch programmer Guido Van Rossum in 1990. Some of the biggest web brands like Youtube, Google,Instagram etc are powered by python.

The official website for the python language is [www.python.org](http://www.python.org). we have different aspect of kinds of python (CPython[default],Pypy,IronPython, Jython[for old java guys])

**PYTHON USE CASES**

* web apps development: django, flask
* Gui/desktop apps development: tkinter,kivy, PyQt, PyGobject
* Data science/ data analysis/ AI: pandas, jupyter notebook,sci-kit learn, tensorflow etc.
* Software testing / automation: selenium

**2. INSTALLING PYTHON**

In order to become Python developer, the first step is to learn how to install or update Python on a local machine or computer. In this tutorial, we will discuss the installation of Python on various operating systems.

Check if python is installed on your workstation: python --version or python -V

## Installation on Windows

* Visit the link <https://www.python.org/downloads/> to download the latest release of [Python](https://www.javatpoint.com/python-tutorial). In this process, we will install latest version on our [Windows operating system](https://www.javatpoint.com/windows). When we click on the above link, it will bring us the following page.
* Double-click the executable file, which is downloaded; the following window will open. Select Install now and proceed. Click on the Add Path check box, it will set the Python path automatically.

## Installation on Mac

To install python3 on MacOS, visit the link <https://www.javatpoint.com/how-to-install-python-on-mac> and follow the instructions , just click on continue, agree, click on customize to check if the feature are all checked and then click install and put in your password.

**3. VARIABLES & DATA TYPES**

**Comments**

1. Single Line Comment: For this we use the **#** hash for a single line comment
2. Multiline Comment: We use quote or docstring ‘’’ ……’’’ his can either be a single or double quote; these can be used to describe what a function do.

**Variable Rules**

1. They are case sensitive (name and NAME are different variables)
2. Must start with a letter or an underscore ( a or \_a)
3. Can have numbers but cannot start with one (boy1 not 1boy)

***Variable Declaration or assignment***

*#this is how we declare a variable*

x = 1

y = 2.5

name = 'Anchorsoft'

is\_best = True

**Data Types**

1. **Int :** The variable X has a data type of ***integer value***, and can be negative or positive (-2,-1,1,2,3…)
2. **Float:** The variable Y has a data type of ***float value,*** these are just decimal number values( 1.5, 3.9, 3.142 …)
3. **String:** The variable name has a data type of ***string value,*** these are just characters or letters and must be in a quote either double “Anchorsoft” or single ‘Academy’
4. **Bool:** The variable is\_best has a data type of ***bool value,*** this means True or False, and the letter T and F must start with uppercase

x = 1                 *# int*

y = 2.5               *# float*

name = 'Anchorsoft'   *# string*

is\_best = True        *# bool*

**Multiple Assignments**

This is similar to the code above but in a single line

*#multiple assignment*

x,y,name, is\_best = (1, 2.5, 'Anchorsoft', True)

**How to Output**

In other version of python i.e python2 bellow we use the **print variable name** i.e print x.

But in python3 we have to put the variable name in a parenthesis print(x)

***Single Output***

print(x)

***Multiple Output***

print(x, y, name, is\_best)

**How to accept input**

Name = input(“Enter your name: ”)

***Integer Input***

num= int(input(“Enter number one: ”))

***Checking The Data type***

We use the TYPE to check the kind of data the variable has.

print(type(x))

***Casting***

This is changing a variable data type to another.

*# Casting*

x = str(x)

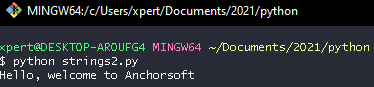
**4. STRINGS & FORMATTING**

**Concatenate**

*# concatenation*

name = 'Anchorsoft'

print('Hello, welcome to ' + name)

***Result***

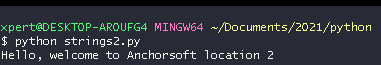
**Concatenation with number**

*# concatenation*

name = 'Anchorsoft'

number = 2

print('Hello, welcome to ' + name + ' location '+ str(number))

***Result***

They are easier and neater way to print concatenated strings; the string formatting and string methods.

**String Formatting**

1. **Argument by position**

*# 1. Argument by position*

print('{}, {}, {}'. format('Anchor', 'Soft', 'Academy'))

we can also have this by order/position, starting from 0 - ~, let print out the code above and ‘Soft’ should come first, ‘Anchor’ take last position, then we have the code like this;

print('{1}, {2}, {0}'. format('Anchor', 'Soft', 'Academy'))



1. **Argument by name**

*# 2. Argument by name*

print('My name is {name} and i am number {num}'.format(name= 'Anchorsoft', num='3'))



1. **F-Strings**

This only works for python version 3.6 and above, we add the F then the string.

*# 3. F-Strings (only in 3.6 +)*

name = 'Anchorsoft'

number = 2

print(f'My name is {name} and i am number {number}')



The F-string is the most recommended one to use.

**String Methods**

1. **Capitalize first letter**

*# Capitalize first letter*

name = 'anchorsoft academy'

print(name.capitalize())

1. **Uppercase**

*# Uppercase*

print(name.upper())

1. **Lowercase**

*# Lowercase*

print(name.lower())

1. **Swap Case:** This occurs if the sentence has uppercase and lowercase

*# Swap Case*

print(name.swapcase())

1. **Variable Length**

*# Get Lenth*

print(len(name))

1. **Replace:** This takes in two parameters the word to be replaced and the word you want

*# Replace*

print(name.replace('Academy', 'Institution'))

1. **Count:** This counts the word unlike the Length (), it checks the number or how many times an alphabet occurs.

*# Count*

a = "a"

print(name.count(a))

They are numerous kinds of string formats and you can easily get them just by adding (.) dot to the variable name i.e **name.**

**Classwork**

1. Create a variable name, age, time
2. Do a casting of string to integer, float to string
3. Display your full name, the length, make them uppercase, count a particular alphabet, replace your name to any of your colleague’s name

**5. LISTS**

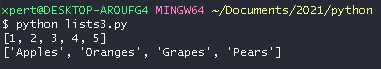
A list is a collection which is ordered and changeable. It allows duplicate members. This is basically an array in javascript, php etc. List is always in a square bracket [ ]

numbers = [1,2,3,4,5]

fruits = ['Apples', 'Oranges', 'Grapes', 'Pears']

print(numbers)

print(fruits)



**Get Value**

*#get value*

print(fruits[2])

The above code will print Grapes as the value

**Get Length**

*#get length*

print(len(fruits))

The above code result will be 4

**Append to list**

*#Append to list*

fruits.append('Oranges')

print(len(fruits))

This will add oranges to the list of fruits

**Remove from list**

*# Remove from list*

fruits.remove('Pears')

print(fruits)

This will remove pears from the fruit list

**Inserting into position:** This takes in two parameters, the position and item you want to add

*# Insert into a position*

fruits.insert(2,'Banana')

print(fruits)

**Remove from a position:**

*# Remove from a postion*

fruits.pop(3)

print(fruits)

**Reverse list**

*# Reverse list*

fruits.reverse()

print(fruits)

**Sort list**

*# sort list*

fruits.sort()

print(fruits)

There are more methods you can do on a list just like the string method, but these are the most commonly used.

**Classwork**

1. create a list of students
2. check the length, remove from a list and with position
3. add a student to a list and from a position
4. print out a particular student

**6. TUPLES AND SET**

**Tuples**

This is a collection which is ordered and unchangeable. It allows duplicate members. And it always in a bracket ( )

*# Tuples*

fruit\_turple =('Apple', 'Orange', 'Mango')

print(fruit\_turple)



**Get Value**

*#Get Value*

print(fruit\_turple[2])

Tuples with one value should have trailing comma, else the result will be a string instead of tuple.

*# Trailing comma*

fruit\_turple = ('Apple',)

print(fruit\_turple)

**Get Length**

*#Get Length*

print(len(fruit\_turple))

**Delete**

*# Delete a tuple*

del fruit\_turple

**SET**

This is a collection which is unordered and unindexed. And no duplicate members. And it always in a curly braces { }

fruit\_set = {'Apple', 'Orange', 'Mango', 'Grape'}

print(fruit\_set)



**Check if in set:** This brings out a True or False result

*# Check if in set*

print('Apple' in fruit\_set)

**Add to set**

*# Add to set*

fruit\_set.add('Banana')

**Remove to set**

*# Remove from set*

fruit\_set.remove('Orange')

**Delete to set**

*# Delete set*

del fruit\_set

**7. DICTIONARIES**

A dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

students = {

  'first\_name': 'Anchorsoft',

  'last\_name': 'Academy',

  'number': 23,

}

print(students)



**Access value**

*# Access value*

print(students['first\_name'])

**Add key/value**

*# Add key/value*

students['phone'] = '08029501995'

print(students)

**Get keys**

*# Get key*

print(students.keys())

**Get Items**

*# Get items*

print(students.items())

**Making Copy**

*# Make Copy*

student1 = students.copy()

student1['address'] = 'Yinusa Adeniji'

print(student1)

**Get Length**

*# Get lenght*

print(len(students))

**List of dictionary**

*# List of dict*

student = [

  {

    'name': 'Anchorsoft',

    'number': 12,

  },

  {

    'name': 'Academy',

    'number': 1,

  },

  ]

**Classwork**

1. create a dictionary of a person
2. print the age of the person and add hobbies to the dictionary
3. make a copy of the person to another person2 adding height
4. create a list of 4 dictionary and access the information in the 3rd dictionary

**9. ARITHEMETIC, COMPARISON AND LOGICAL OPERATORS**

**Arithmetic Operators**

*# Arithemetic operators*

names            signs

----------------------

addition          +

subtraction       -

division          /

multiplication    \*

module %

**Comparison Operators**

*# Comparison Operators*

names                      signs

---------------------------------

equal to                      ==

not eqaul to                  !=

greater than                  >

less than                     <

greater than and equal to     >=

less than and equal to        <=

**Logical Operators**

*# Logical Operators*

AND: True if both the operands are true

OR: True if either of the operands is true

NOT: True if operand is false

**9. FUNCTIONS**

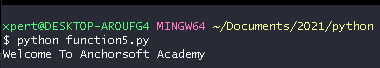
Function is a block of code which only runs when it is called. In python, we do not use parentheses and curly brackets; we use indentation with tabs or spaces.

*# create function*

def sayHello():

  print('Welcome To Anchorsoft Academy')

sayHello()  *#we have to call the function*



**Function with arguments**

*# Parameters or arguments*

def sayHello(name):

  print('Welcome ' + name)

sayHello('Anchorsoft')

**Default parameter/ arguments**

*# Default parameters or arguments*

def sayHello(name = 'Anchorsoft'):

  print('Welcome ' + name)

sayHello()

However, passing a default parameter or argument and also passing a parameter to the call function at the same time is possible, but the call function will override the default parameter.

def sayHello(name = 'Anchorsoft'):

  print('Welcome ' + name)

sayHello('Academy')

The code above, the Academy will override the Anchorsoft.

**Return Value**

Usually we don’t print out from a function we return.

*# Return Value*

def sumNum(num1, num2):

  total = num1 + num2

  return total

print(sumNum(3,5))

Note variable define in a function cannot be accessible outside the function.

**Lambda Function**

This is a small anonymous function and can take any number of arguments, but can only have one expression.

*# Lambda*

getSum = lambda num1, num2 : num1 + num2

print(getSum(3, 5))

**Classwork**

1. Create a simple calculator to add, subtract, divide, multiply
2. Adding two unknown number i.e user must input any number

**Assignment**

1. Adding two unknown number but one should have a default parameter
2. A simple calculator to add three numbers, add two of the three numbers and subtract one.

**10. CONDITIONALS**

This is use to decide an action base on certain condition; if (condition) then do this.

**If Statement**

*# if statement*

x = 10

y = 5

if x == y:

  print(f'{x} is equal to {y}')

**If/else Statement**

*# if/else statement*

x = 10

y = 5

if x == y:

  print(f'{x} is equal to {y}')

else:

  print(f'{x} is not equal to {y}')

**elif Statement**

*# elif statement*

x = 10

y = 5

if x > y:

  print(f'{x} is greater than {y}')

elif x < y:

  print(f'{x} is less than {y}')

else:

  print(f'{x} is not greater or less than {y}')

**Nested if Statement**

*# Nested if*

x = 6

if x > 2:

  if x <= 10:

    print(f'{x} is greater than 2 and less than 10')

**Logical Operator**

The code above using nested if should be written using logical operator

*# Logical Operator*

x = 6

if x > 2 and x <= 10:

  print(f'{x} is greater than 2 and less than 10')

#or

x = 6

if x < 2 or x <= 10:

  print(f'{x} is greater than 2 and less than 10')

*#not*

x =3

y =4

if not(x==y):

  print(f'{x} is not equal to {y}')

**Membership Operators**

These are used to test if a sequence is presented in an object and they are (not, not in) and returns True or False as the output.

*#Membership Operators*

number = [1,2,3,4,5,6]

x= 6

if x in number:

  print(x in number)

*# not in*

number = [1,2,3,4,5,6]

x= 6

if x not in number:

  print(x in number)

**Identity Operators**

These are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location. And they are (is, is not) and returns True or False as the output.

*# Identity Operators*

*# is*

x = 10

y = 10

if x is y:

  print(x is y)

*#is not*

x = 10

y = 5

if x is not y:

  print(x is not y)

**Classwork**

1. Create a simple program to accept number from user and display if the number is odd or even number.
2. Create a simple program that accept alphabet from user and display if it a vowel or a consonant.

**Assignment**

1. Write a program to compare three numbers entered by user and display the greatest among the three.
2. Write a script containing 4 different functions (addition, subtraction, multiplication and division), when you run the code, a message should display asking you to select a number to continue(1. Addition, 2. Subtraction, 3. Division, 4. Multiplication), when you select any number the program should ask you to input two numbers that you want to execute base on any selection you made.

**11. LOOPS**

A Loop is used for iterating over a sequence (that is either a list, turple, dictionary, set or a string). And we have several loops methods for, while, range

**For Loop**

*## Loop*

*# for loop*

student = ['John','Mary','Nath','Badmus']

for person in student:

  print(person, ' is a student')

**Range**

This is use to loop through something in a specific number of times

for i in range(0, 10):

  print(i)

This will print a number from 0 to 9, range(0, 10) means start from 0 and end at 10 but it won’t print 10, if you want it to print to 10 it will be range(0, 11).

Let loop through a range of people

people = ['John','Mary','Tayo','Nath','Adams','Folorunsho']

for i in range(len(people)):

  print(people[i])

**While Loop**

This execute a set of statement as long as a condition is true

*# While*

*# ....printing a number from 0 to 10*

num = 0

while num <= 10:

  print(num)

  num += 1

**Break and Continue**

**Break:** This is use to stop the loop is a certain condition is met

*# Break*

number = [1,2,3,4,5,6,7,8]

for i in number:

  if i == 4:

    break

  print(i)

The block of code above will loop through the list (number) and once it get to 4 it stop looping and will not print 4, only if the print statement is above the if condition.

**Continue:** This is the opposite of break; it will skip the condition if it’s true and continue the loop.

*# Continue*

number = [1,2,3,4,5,6,7,8]

for i in number:

  if i == 4:

    continue

  print(i)

This will print 1,2,3 and skip 4 then continue with 5,6,7,8.

**Classwork**

1. Write a simple program to loop through a list of fruit
2. Write a simple program to print your name 6 times
3. Write a program to print factorial of any number

**Assignment**

1. Write a multiplication program that allows user to specify the number to be multiplied.
2. Write a program to display multiplication table from 1 to 5
3. Write a program to check if the password entered from a user is correct and say welcome, and also give 2 chances for incorrect password

**12. MODULES**

These are basically a file containing a set of functions to include in your application. There are core python modules, and also modules you can install using PIP package manager (including Django) as well as custom modules (the one you create yourself).

When working with modules we use the word import, from to call modules.

*# Core*

import datetime

todayDate = datetime.date.today()

print(todayDate)

**datetime** is a core python module that we import in other to use it in our code

**todayDate** a variable name we use so as to display our result

**date and today** they are both function of datetime module that can be accessed, and you can use the **(.)** to see more function of the date time module.

However we can also pull the method of the module(date) out, if we are not importing the entire module of the **datetime**

from datetime import date

todayDate = date.today()

print(todayDate)

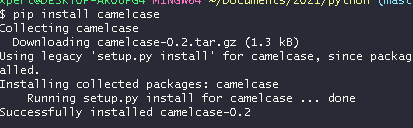
Most of the time you see developers use **from (module name) import (module method)**

**Pip Installation**

Let take a look at pip cause we will be using it quite a bit and more in Django development. We use **pip freeze** to check our entire package that is installed globally and virtually (we discuss more of this in django development).

Let install a module and use it in our code

**pip3 install camelcase or pip install camelcase**

calmelcase is a model that is used to make every first text of a sentence a uppercase.

import camelcase

tcamel = camelcase.CamelCase()

text = 'hello, welcome to anchorsoft'

print(tcamel.hump(text))



**Custom Modules**

This is when we create our own function/method file and call it in another folder/file .

Take for instance we have **folderA** containing a file called greetings.py and write a function/method

def igbos():

  return 'Nno'

And another **folderB** with a file that can access the file in **folderA** as a module.

from folderA.greetings import igbos

print(igbos())

This is how we create our function and use it in other files, it helps to reduce number of lines of code and reusable for other purpose.

However you can access a file in the same folder with just from .filename import function name

**Classwork**

1. Write a program to print your name in every 3 seconds using a module
2. Create two different files, one with a function that first request for your name and returns ‘Hello {with your name}’ and print the output in another file.

**Assignment**

1. Create two different files, one with all arithmetic operation as function asking for user input and other file to call the result as a module
2. A program to generate calendar of the year.

**13. CLASSES**

This is like a blueprint for creating objects. An object has **properties and methods** (functions) associated with it. Almost everything in python is an object.

They are two kinds of classes

1. Instance of a classes
2. Class of a classes

The two class have components attributes/properties and method

* **Attributes/Properties:** This is the data you want to hold in a class, like the name, color, speed, height etc of a car
* **Method:** This is the function you want the car to do, move forward, reverse, horn etc

**Instance of a Class**

Creating properties and method of an Instance class

***Properties***

We use the dunda init : \_\_init\_\_(self)

class Car:

*# Instance of a class*

*# properties/ attributes*

  def \_\_init\_\_(self):

***Method***

*# Method*

  def toyota(self):

    return

Now that you have fully understand how to create a constructor of an instance let do some examples

**Working with properties of an instance of a class**

class Students:

  def \_\_init\_\_(self, name, email,age):

    self.name = name

    self.email = email

    self.age = age

*# initialise a student object*

john = Students('Anchorsoft', 'john@gmail.com', 20)

mary = Students('Academy', 'mary@gmail.com', 18)

*# edit a property*

john.age = 25

print(john.name)

print(mary.age)

**Working with properties and method of an instance of a class**

class Students:

*# properties of an instance*

  def \_\_init\_\_(self, name, email, age):

    self.name = name

    self.email = email

    self.age = age

*# method of an instance*

  def greetings(self):

    return f'My name is {self.name} i am {self.age}, you can mail me {self.email}'

*# initialise a student object*

john = Students('Anchorsoft', 'john@gmail.com', 20)

mary = Students('Academy', 'mary@gmail.com', 18)

*# edit a property*

john.age = 25

*# accessing the method*

print(john.greetings())

You can define more than one method for the properties/attributes

**Inheritance**

This is when another class/constructor is accessing/ inheriting information from another class.

We have Person class, and it has attributes (name, age etc) and method, we need another class called User, we can just inherit the properties of a person for the user to use.

*# Inheritance*

class Person:

  def \_\_init\_\_(self, name, email,age):

    self.name = name

    self.email = email

    self.age = age

  def greetings(self):

    return f'My name is {self.name} i am {self.age}, you can mail me {self.email}'

class User(Person):

  def \_\_init\_\_(self, name, email, age):

    self.name = name

    self.email = email

    self.age = age

    self.sex = sex

  def gender(self, sex):

    self.sex = sex

*# initialise a student object*

john = User('Anchorsoft', 'john@gmail.com', 20)

john.gender('Male')

print(john.sex)

**Class Work**

1. Create a class user and give it a method to greet
2. Create two class the second should have extra parameter and also inherit the first class parameters
3. Calculate the area of a cylinder and return the size (small, medium, big) of the cylinder (formular = pi \* radius^2 \* height )

**Assignment**

1. Write a student attendance program to print the average score of the student and times he attended the class

**Class of a Class**

***Properties***

This is just written without any dunda init

class Circle:

  pi = 3.142

***Method***

We use the **@classmethod** above the function and pass **cls** in the function name instead of **self**

class Circle:

  pi = 3.142

  @classmethod

  def objectCounter(cls):

     print(f'this is a class method')

**Instance members;** are ONLY accessible via an object of the class. While **Class member;** are accessible via both the object and the class itself.

**Example**

class Circle:

  pi = 3.142

  def \_\_init\_\_(self, r):

    self.radius = r

area = Circle('')

print(area.pi)

print(Circle.pi)

pi is now both accessible via an instance or class. Circle. pi and obj.pi will return 3.142 but the radius is only accessible via instance/object.

Another example to explain the Instance and Class member

class Planet:

*#class properties*

  shape = 'round'

*#instance properties*

  def \_\_init\_\_ (self, name, radius):

    self.name = name

    self.radius = radius

*#instance method*

  def orbit(self):

    return f'{self.name} is a good planet'

*# class method*

  @classmethod

  def common(cls):

    return f' all the planet are {cls.shape} due to gravity'

venus = Planet('Venus', 3)

print(Planet.common())

print(venus.common())

**14. FILES**

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

**File Modes**

r - read

a - append

w - write

1. **Open or Creating a file**

*# opening a file*

myFile = open('myfile.txt', 'w')

1. **Getting Info**

*# Get some info*

print('Name: ', myFile.name)

print('Is Closed: ', myFile.closed)

print('Opening Mode: ', myFile.mode)

1. **Write to file**

*# Write to file*

myFile.write('I love Python')

myFile.write(' and JavaScript')

myFile.close()

1. **Append to a file**

*# Append to file*

myFile = open('myfiles.txt', 'a')

myFile.write(' I also like PHP')

myFile.close()

The ‘a’ means append, nothing will change it will add the text, but if we replace the ‘a’ with ‘w’, then we will not get the old text but this new one alone

1. **Read from a file**

*# Read from file*

myFile = open('myfiles.txt', 'r+')

text = myFile.read(8)

print(text)

This will print 8 character including the space

**Class Work**

1. Create a file, write to the file, append to the file and print out characters out of the file.