Python

Hello Word

```
print ("hello world")
```

Input

to take input we use input function

Calculator

```
val = int(input("enter value 1 : "))
val_2 = int(input("enter value 2 : "))
print(val+val_2)
print(val-val_2)
print(val*val_2)
```

slicing

name = "https://www.online-python.com/"

firstn= slice(8,-5)

print(name[firstn])

if else condition

```
print ("heelloo")
age =int (input ("what is your age"))
if age  == 18:
    print ("welcome")
elif age >=18:
    print("hey")
```

use "**:**" in condition

While loop

```
i = 1
while i <= 6:
    print(i)
    i += 1
```

For Loop

```
for i in range (50, 100+1) :
    print(i)
```

```
for i in "Ammar Saqib Arain" :
    print(i)
```

```
import time

for second in range(3, 0, -1):
    print(second)
    time.sleep(2)

print("PILLAY")
```

Nested Loop

```
rows = int(input("How many rows? "))
columns = int(input("How many columns? "))
symbol = input("Enter symbol: ")

for i in range(rows):
    for j in range(columns):
        print(symbol, end="")
    print()
```

Match Case

It same like break case statement in c++

```
x=int(input("enter value"))
match x:
    case 0: print("hello")
    case 20: print("bye")
    case _: print("ho")
```

List

It is same like Array

```
list = [0,2,23,4,43,5]
print(list)
print(type(list))
print(list[0])
```

append will add in last of list

pop will remove element from last

reverse will reverse the list

sort will sort the list

```
list = [0,2,23,4,43,5]
print(list)
print(type(list))
print(list[0])
list.append(20)
print(list)
list.sort()
print(list)
list.pop()
print(list)
list.reverse
print(list)
```

2d array

```
 drinks = ["coffe", "tea"]
dinner = ["fisj", "burger"]
desert = ["icecream", "sweets"]
fodd=[drinks,dinner,desert]
print(fodd[2][1])
```

Tuple

Collection which is unchangeable and ordered use to group together related data

```
student= ["bro", 21, "male"]
print(student)
print(student.index("bro"))
if "bro" in student:
    print("ok")
```

Set

Set is collection which is unordered and unindexed. No duplicate value

```
utensils = {"fork", "spone", "kmife"}
for x in utensils:
    print(utensils)
```

```
utensils.add("ballon")
utensils.remove("fork")
```

Dictionaries

It is changeable, unordered collection of unique key values pairs. They are fast because they use hashing and they allow us to access a value quickly. It is very similar to set.

```
capital = {'usa' : 'washington'
           , 'china' : 'bejing'
           , 'russia' : 'mosscow'}
print(capital['russia'])
```

here we store value in string and use string(key) to print it's value

```
capital = {'usa' : 'washington'
           , 'china' : 'bejing'
           , 'russia' : 'mosscow'}
print(capital.get('pakistan'))
```

get function check pakistan then it give answer none

to change in list we us **"update"** key

```
 capital = {'usa' : 'washington'
           , 'china' : 'bejing'
           , 'russia' : 'mosscow'}
print(capital.get('pakistan'))
capital.update({'germany' : 'berlin' })
capital.update({'china' : 'china'})
print(capital)
```

Indexing

it give acess to sequence element we use ' **[]'** for indexing

```
name = "ammar"
if (name[0].islower()):
    name = name.capitalize()
    print(name)
```

for specific size of string we use **"upper"**

```
name = "ammar"
if (name[2].islower()):
    name = name.capitalize()
    first_name = name[:3].upper()
    last_name = name[3:5].upper()

    print(first_name)
    print(last_name)
```

negative indexing also use in negative index we use [-n{n can be any value}]

```
name = "ammar"
if (name[2].islower()):
    name = name.capitalize()
    first_name = name[:3].upper()
    last_name = name[3:5].upper()
    last_character=name[-1]
    print(first_name)
    print(last_name)
    print(last_character)
```

Function

Block of code which execute only when its called

It syntax start with def and name and just called it

```
def hello ():
    print("Hello")
hello()
```

the number of time you call it, then number of time will print

Function argument

```
def ammar (name):
    print("Hello" +name )
ammar("world")
ammar("10")
```

Function with 2 argument

```
def ammar (name,gg):
    print("Hello" +name +gg )
ammar("world","2")
ammar("bro","22")
```

Return statement

Functions send value/object back at caller. The value is called as function's return value

```
def mul (num_1, num_2):
    return num_1 * num_2
x = mul(5,8)
print(x)
```

Keyword argument

Argument preceded by an identifier when we pass them to function. The order of argument does not matter, unlike potential argument. Python knows the arguments name which receive function

```
def hello (fast, middle, last,):
    print(fast, last, middle)
    print(fast, middle, last)
hello("sami","rauf","jhon")
```

```
def info(name,age,cast):
    print ("my name is " +name)
    print ("my cast is" +cast)
    print ("I'm " +str(age)+ " old")
info (age=20, name="Ammar" ,cast ="Arain")
```

Random function

It is an built in function which perform random things

```python
import random
x= random.randint(1,6)
y=random.random()
card=["1","2","3","4","5","6","K","Q"]
random.shuffle(card)
print(x)
print(y)
print(card)
```

Exception

We use exception so our program run with out any interception

```python
try:
    num1 = int(input("enter value : "))
    num2= int(input("enter value : "))
    result =num1/num2
    print(result)
except Exception:
    print("something went wrong")
except ZeroDivisionError:
    print("give non zero value")
```

Read file with help of Python

For this program you need a txt file, you also need to create a file

```python
with open('test.txt') as file:
    print(file.read())
```

Module

A file contain python code in other file and we import it in our main file. For this need to create another file.

```python
#2nd file
def hello():
    print("hello")
def bye ():
    print("see you next time")
```

```python
#main file
import work
work.hello()
work.bye()
```

Or also can use

```
from work import *
hello()
bye()
```

if you want to know all built in module then

```
help ("modules")
```

str format

we can use str format for numbers to convert

```
number = 3989
print("The number is {:.3f}".format(number)) #will print 3 number after point
print("the value in binary is {:b}".format(number)) #for binary number
print("the value in octal is {:o}".format(number)) #for octal number
print("the value in octal is {:x}".format(number)) # for hexadecimal number
print("the value in octal is {:e}".format(number)) # for scientific number
```

Game paper, scissor and rock

```
import random

choose = ["rock", "paper", "scissor"]

computer = random.choice(choose)
player = None
while player not in choose:
    player = input("chosse any of these\n  rock paper or scissor : ")
if player == computer:
    print("player", player)
    print("computer" , computer)
    print("tied")
elif player =="rock":
    if computer=="paper":
        print("player", player)
        print("computer", computer)
        print("you lose")
    if computer=="scissor":
        print("player", player)
        print("computer", computer)
        print("you win")
elif player =="scissor":
    if computer=="rock":
        print("player", player)
        print("computer", computer)
        print("you lose")
    if computer=="paper":
        print("player", player)
        print("computer", computer)
        print("you win")
```

```python
elif player =="paper":
    if computer=="scissor":
        print("player", player)
        print("computer", computer)
        print("you lose")
    if computer=="rock":
        print("player", player)
        print("computer", computer)
        print("you win")
```

OBJECT ORIENTED PROGRAMMING

Can crate in same module or can be seperate

```python
class person:
   name = "ammar"
   occupatin = "student"
p = person
print(p.name)
#VS CODE
class Person:
    name = "ammar"
    occupation = "student"

    def info(self):
        print(f"{self.name} is a {self.occupation}")

a = Person()
a.info()
```

Constructor

Init is a built in function

```python
class Person:
    def __init__(self):
        print("hello")
a=Person()
```

```python
class Person:
    def __init__(self):
        self.name = "ammar"
        self.occupation = "student"

    def info(self):
        print(f"{self.name} is a {self.occupation}")


a = Person()
a.info()
```

```
class Person:
    def __init__(self,n,o):
        print("hello")
        self.name = n
        self.occ= o
        print(f"{self.name} is a {self.occ}")


a=Person("ammar","student")
```

```
class Car:
    def __init__(self, make, model, year, color):
        self.make = make
        self.model = model
        self.year = year
        self.color = color

     def drive(self):
        print("The car is driving")

    def stop(self):
        print("The car has stopped")

car1 = Car("Toyota", "Camry", 2022, "Blue")
print(car1.make)
print(car1.model)
print(car1.year)
print(car1.color)

car1.drive()
car1.stop()
```

Inheritence

```
class Dog:
    alive = True
    def eat(self):
        print("hello")

class Cat(Dog):
    pass

class Hawk(Cat):
    pass

dog = Dog()
cat = Cat()
```

```
hawk = Hawk()
hawk.eat()
```

Method Overriding:

```
class Person:
    def eat(self):
        print(f"your name is age")
class Player:
    def eat (self):
        print(f"your name is razzaq and age is 22")
player = Player()
player.eat()
```

Method Chaining

In method chaining function call by chain one by one

```
class Test:
    def start (self):
        print("the test is start")
    def mid (self):
        print("half of time is spent")
    def end (self):
         print("the test is end ")
t1= Test()
t1.start()
t1.mid()
t1.end()
```

Abstract

```
#Abstract class is a class which  contain abstract method
#Abstract Method: it have declaration but not implementation
from abc import ABC, abstractmethod
class Vehicle (ABC):
    @abstractmethod
    def go(self):
        pass
class Bike(Vehicle):
    def go(self):
        print("this is bike")
class Car(Vehicle):
    def go(self):
        print("this is car")
bike= Bike()
car = Car()
#vechile = Vehicle()
#print(vehicle.go() )if you remove comment too, it didn't work because it's
ghost class
bike.go()
car.go ()
```

Super Function

```python
class IT:
    def __init__(self, name, age, height):
        self.name = name
        self.age = age
        self.height = height

class Math(IT):
    def __init__(self, name, age, height):
        super().__init__(name, age, height)

    def display(self):
        return self.age, self.name

class Chemistry(IT):
    def __init__(self, name, age, height):
        super().__init__(name, age, height)

    def show(self):
        return self.age, self.name, self.height

math = Math("Yasir", 19, 170)
chemistry = Chemistry("Ali", 22, 160)


print(math.display())
print(chemistry.show())
```

Object As Argument

```python
class IT:
    def __init__(self, name, age, height):
        self.name = name
        self.age = age
        self.height = height

class Math(IT):
    def __init__(self, name, age, height):
        super().__init__(name, age, height)

    def display(self):
        return self.age, self.name

class Chemistry(IT):
    def __init__(self, name, age, height):
        super().__init__(name, age, height)

    def show(self):
        return self.age, self.name, self.height
```

```
math = Math("Yasir", 19, 170)
chemistry = Chemistry("Ali", 22, 160)


print(math.display())
print(chemistry.show())
```

Walrus Operator

```
# walrus operator in python 3.8, it assign to variable as part f longer
exprssion
foods = list()
while food := input("whta's your favourte food ") !="quit":
    foods.append(food)
```

Function to Variable

Here we give print function to variable name say, and after it we print our message through say which is print function

```
say = print
say ("no wayyyy, it print on console screen :)")
```

Lambda

Useful when time is short

```
double = lambda x:x *2
multiply = lambda x,y : x*y
add = lambda x,y,z: x+y+z
print(double(5))
print(multiply(5,6))
print(add(8,9,10))
```

Sort function and method

Sort () method = use in list

Sort () function = use in iterable

Sort Method

```
student= ["Hassan", "Umer", "Yasir", "Ammar", "Markaram"]
student.sort()
for i in student:
    print(i)

student.sort(reverse=True)
for i in student:
    print(i)
```

Sorted Function

```
student= ("Hassan", "Umer", "Yasir", "Ammar", "Markaram")
sorted_student = sorted(student)
for i in sorted_student:
    print(i)
```

```
student = [("Hassan","A", 78),
          ("Umer", "B", 65),
          ("Yasir", "A1", 80),
           ("Jhon", "F",45),
            ("Markaram", "C", 55)]
grade = lambda grades:grades[1]
student.sort(key=grade)

for i in student:
    print(i)
```

Maps

Applies a function to each item in iterable (list, tuples, etc)

```
store = [("pant", 400),
         ("shiry", 250),
         ("sock", 50)]
to_dollor = lambda data : (data[0], data[1]*286)
store_dollar = list(map(to_dollor,store))
for i in store_dollar:
    print(i)
```

Filter Function

Create a collection of elements from an iterable for which a function is return

```
friend = [("Rehan", 20),
          ("Aadil" ,22),
          ("Yasir", 19),
          ("Abdullah",22)]
age = lambda data:data[1]>=20
buddy = list(filter(age,friend))
for i in buddy:
    print(i)
```

Reduce

Apply a function to an interable and reduce to it a single value

```
import functools
letter = ["H", "E", "L", "L", "O"]
word =  functools.reduce(lambda x,y:x+y , letter)
print(word)
factorial = [5,4,3,2,1]
```

```
result = functools.reduce(lambda x,y:x*y , factorial)
print(result)
```

List Comprehension

Way to create a new list less syntax can mimic ,certain lambda functions,_easier to read

```
student = [99,96,89,81,75,68,56,50,44,40]
#passed_student = list(filter(lambda x: x>=60, student))
#print(passed_student )
passed_student = [i if i>= 60 else "Failed" for i in student]
print(passed_student)
```

Time Module

```
student = [99,96,89,81,75,68,56,50,44,40]
#passed_student = list(filter(lambda x: x>=60, student))
#print(passed_student )
passed_student = [i if i>= 60 else "Failed" for i in student]
print(passed_student)
```