Python

> Python is an interpreted, object-oriented, high level programming language with dynamic semantics.

💡 **interpreted** = code doesn't stop until found error, if found error on last lines the above code gets executed.
object-oriented = works on the concept of 'object' which can contain data and code
high level = made in that way that everyone can understand (simple syntax)
dynamic semantics = will know what kind of data structure(int, char) is passed in a variable when starts running.

▼ Data types

💡 type() use check the datatype of a variable

💡 mutable object can change its state or contents and immutable objects cannot
-- **Mutable datatypes -- list , dictionary , byte array , set**
-- Immutable datatypes -- int , float , complex , string , tuple.

💡 changing the type of a data type is known as "Type Casting"-- int(), float(), eval()[eval can handle int, float and binary too].

1. **Numeric** --
   integers (57) ,
   float (57.7) ,
   complex numbers (5 + 7j)
2. **Sequence** --
   string (`'hi', "hello", ''' hey '''`) (putting addition sign between strings makes it CONCAT // `'10' + '20' = '1020'`),
   list (`l = [1,57.7,'sw']`)(`l[1] = 57.7`) ,
   tuple (`1,57.7,'sw',1+5j`)(same as list but immutable)
3. **Dictionary**
   (`d = { key: 'value' , 'name': 'python' , 3:3 }`)(key has to be unique) (d['key'] = 'value')
4. **Set**
   (`s = {1,2,3,4,5,6}`)(every value is unique in curly brackets)

▼ Operators in python

1. **Arithmetic operators**
   (`+, -, * , /, %` *(modulus)(remainder),*
   ** *(exponents)(5 ** 3* = 55*5 = 125),
   // (floor division)(10 // 3 = 3.3333 = 3))
2. **Assignment operators**
   (`=, +=` ('x += 3' = 'x = x + 3) ,
   `-=` ('x -= 3' = 'x = x - 3))
3. **Comparison**
   (`== , != , > , < , >= , <=`)
4. **Logical**
   (`and` (return true if both statement are true)(x<5 and x<10),
   `or` (return true if one is true),
   `not` (return false if both are true)('not(x<5 and x<10)'))
5. **Membership**
   (`in` (return true if present in object) ,
   `not in` (return true if not present in object))
6. **Identity**
   (`is` (return true if both variable are same), not is (opposite))
7. **Bitwise**
   (`&` (and) , `|` (or) , `^` (xor)) [bin() to change int to binary]

▼ String functions

> anything in between quotes is string

`s = "Welcome", x = "Welcome123", y = "5757`
`lower()` (makes a string in lower case)
`upper()` (upper case)
`title()` (makes every word's first letter capital)
`capitalize()` (makes first letter upper and every other lower case)
`find()` (`s.find('e')` => 1) (gives the index of first found) (`s.find('e',2)` => 6) (gives the second 'e' after giving start para.) (if not found will return '-1')
`index()` (`s.index('e')` => 1) (if not found will return 'Error') (we can give 2nd para.)
`isalpha()` (`s.isalpha()` => true) (`x.isalpha()` => false -- because it contains numeric digit)
`isdigit()` (`x.isdigit()` => false -- because it contains alphabets) (`y.isdigit()` => true)
`isalnum()` (checks if the string contains special characters) (if contains then false else true)
`chr()` (convert int to ASCII character) (`chr(65)` => 'A' => `type('str')`)
`ord()` (opposite of chr()) (`ord('A')` => 65 => `type('int')`)

▼ `format()` -- string formating

```
named indexes
txt1 = "my name is {fname} {lname}".format(fname='tabish',lname='shaikh')

#numbered indexes
txt2 = "my name is {0} {1}".format('tabish','shaikh')

#empty indexes
txt3 = "my name is {} {}".format('tabish','shaikh')
```

```
#Output
#All output will be => "my name is tabish shaikh"
```

💡 if we put it like this {a:10} then the output will take 10 char. space from left
if we put it like this {a:^10} then the output will take 10 char. equal space from both sides
if we put it like this {a:>10} then the output will take 10 char. space from right

▼ List

> - if list contains another list then its known as "Nested list"

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

▼ slicing of list

```
l = [2,3,'Hello',[3,4,5]]
l[0:2] = [2,3]
l[0::2] = [2,'Hello']
l[-1] =  [3,4,5]
l[-1::-2] = [[3,4,5],3]
l[-1::-1] = [[3,4,5],'Hello',3,2]
```

▼ List Iteration

```
l = [10,20,30,50,60]
t = len(l)
for i in range(t):
  print(l[i])
```

```
l = [10,20,30,50]
t = len(l)
for i in range(t-1,-1,-1):
  print(l[i])
```

▼ List function

`l = [10,20,30,50]`
`del` -- del needs list name and index to delete it (`del l[1]`)
`pop()` -- same as del but also return the delete character (`l.pop(2)`)
`remove()` -- it needs value to delete instead of index (`l.remove(50)`)
`clear()` -- makes blank(delete) the whole list (`l.clear()`)

`insert()` -- inserts the value to given index (`l.insert(0,5)`) `insert(index,value)` and pushes other value to +1 index and to work on if indexes is 4 and passes 5
`append()` -- added the value in last of list (`l.append(60)`) if passes another list in append then makes it nested list
`extend()` -- same as append but when passes list it doesn't make nested list picks one by one value and added it

`count()` -- counts the value in list (`l.count(10)`)
`max()` -- gives the maximum value in list (`max(l)`) this also works for alphabets
`min()` -- gives the minimum value in list (`min(l)`) this also works for alphabets
`sort()` -- update and sorts the list in ascending order (l`.sort()`)
`reverse()` -- update and reverse the given list (`l.reverse()`)
`index()` -- gives the index of passed value (`l.index(20)`)

▼ List Comprehension

`n = [i for i in range(1,101)]` => n = [1,2,3,.....,99,100]
`n = [i for i in range(1,101) if h%2 == 0]` => n = [2,4,6,8,10,....,98,100]

`s = 'Tabish'`
`n = [g for g in s]` => n = [T,a,b,i,s,h]

▼ Zip function

> iterate two same list at a same time

```
l =[10,20,30,40]
ll =[98,54,33,53,43]
t = len(l)
for a,b in zip(l,ll):
  print(a,b)

for h in range(t):
  print(l[h],ll[h])
```

💡 #output
10 98
20 54
30 33
40 53

▼ string to list

`split()` -- makes a string to list gives comma on every space.
`n = 'md tabish shaikh'`
`l = n.split()` => ['md', 'tabish', 'shaikh']

```
l = []
for a in range(1,4):
  n = input('Enter the value '+str(a)+' :-')
  l.append(n)
print(l)
```

▼ Stack

> Stack is a linear data structure just like books on top of books

> stores item in Last-in/First-out (LIFO) or First-in/Last-out (FILO) manner

push -- inserting an element -- `append()`
pop() -- deletion of last element -- `pop()`
peek -- display the last element -- `l[-1]`
display -- display list -- `print(l)`
exit -- to exit -- `break`;

▼ Queue

> Queue is a linear data structure just like a queue for ticket in railway station

> stores item in First-in./First-out (FIFO) manner

Enqueue -- Adds an item to queue -- `append()`
Dequeue -- Removes an item from queue -- `del l[0]`

Front -- Get the front(1st) item from queue -- `l[0]`
Rear -- Get the last(last) item from queue -- `l[-1]`

▼ Dictionary

> Dictionary is a Unordered datatype, It is Mutable, Index doesn't work in dictionary

> key : value -- key is unique

> it is defined in “{}”

syntax--

```
d = {'name' : 'tabish', 'age': 21,'profession' : 'student'}
for i in d:
  print(i+" -- "+str(d[i]))
```

💡 => output
name -- tabish
age -- 21
profession -- student

▼ Dictionary Function

```
d = {'name' : 'tabish', 'age': 21,'profession' : 'student'}
```

`get()` -- gives the value when passes key (`d.get('name')`) => (`d['name']`) => tabish
`keys()` -- gives keys (`d.keys()`) (`for i in d.keys(): print(i)`)
`values()` -- gives values (`d.values()`) (same for loop as keys)
`items()` -- gives keys and values (`d.items()`) (`for i,j in d.items():print(i,j)`)
`del` -- deletes key and value when passes only key (`del d['name']`)
`pop()` -- deletes(key,value) and return(value) the key and value (`d.pop('name')`)
`dict()` -- creates a dictionary (`dict(name='tabish',age=21)`) (`d['age'] = 22`)
`update()` -- updates the value (`d.update({'age':22})`)

`d['qualification']` = 'b.tech' — inserting new key and value in dictionary
`clear()` -- clear the whole dictionary (`d.clear()`)

▼ Nested Dictionary

> collection of dictionaries in one single dictionary

```
course = {
'php' : {'duration': '2 months' , 'fees' : 14000},
'python' : {'duration': '2 months' , 'fees' : 15500},
'java' : {'duration': '2 months' , 'fees' : 17000}
}
print(course['php'])
print(course['php']['fees'])
```

💡 => {'duration': '2 months' , 'fees' : 14000}

💡 => 14000

▼ Tuple

```
t = (20,30,40,50)
t[2] => 40
```

`count()` -- counts the value in list (`t.count(50)`)
`max()` -- gives the maximum value in list (`max(t)`) this also works for alphabets
`min()` -- gives the minimum value in list (`min(t)`) this also works for alphabets
`index()` -- gives the index of passed value (`t.index(20)`)
`sum()` -- sum all elements in tuple (only works on int and float)(`sum(t)` => 140) (`sum(t,10)` => 150)

▼ Set

> Index doesn't work on Set, Every value in set is unique, it is defined in '{}'

> set can be randomly printed

```
s = {10,20,30}
for i in s:
  print(i)
```

```
=>
10
20
30
```

`set()` -- converts a list to set (`set([10,20,30])`)
`add()` -- adds new value to set(`s.add(40)`)
`pop()` -- delete and return random value(`s.pop()`)
`remove()` -- takes value and delete it (`s.remove(20)`)
`discard()` -- same as remove (`s.dicard(20)`)
`clear()` -- deletes everything (`s.clear()`)
`update()` -- adds list to set (`s.update(l)` => {10,20,30,40,50})

▼ Conditional Statements

1. **if statement**
2. **if else statement**
3. **if elif else statement**

```
if a==10:
  print('a is ten')
elif a == 20:
  print('a is twenty')
else :
  print ('a is not ten,twenty')
```

▼ Range

▼ `range(5)`

start = 0 -- default
conditon < 5
increment = 1 -- default

▼ `range(5,7)`

start = 5
conditon < 7

increment = 1 -- default

▼ `range(1,15,3)`

start = 1
conditon < 44
increment = 3

▼ `range(10,0,-1)`

this is a reverse function
start = 10
condition > 0
decrement = -1

▼ Loops

▼ for loop

```
for i in range(5);
  print(i, 'hi')
```

▼ while loop

```
i = 1
while i <= 10:
print(i)
i += 1
```

💡 after while loop the value of `i` will be 11

▼ Functions

> A function is a block of statements which can be used repetitively in a program. It saves the time of a developer. In python concept of function is same as in other language. you can pass data, known as parameters or arguments into a function.

💡 creating a function is defined using the "def" keyword

Input
`input('Enter the value:- ')`

(put input function in a variable to safe the input value in the same variable)

---

1. **Simple function**
2. **Function with arguments**
3. **Return type**

▼ simple function

```
def func():
  print('this is a simple function ')

func()
```

▼ function with argument

```
def sumdata(i,j=5):
  print(i+j)

sumdata(10)
sumdata(10,20)
```

💡 => 15

▼ return type function

```
def subtractData(a,b):
  return a-b

s = subtractData(40,25)
print(s)
```

▼ Modules

> you can import modules

```
import module1 as m //giving alias as m
from module1 import sum // importing only a function from a module
from module1 import * //importing everything from a module
```

▼ Math Module

```
import math as m
```

`ceil()` = changes float value to int by adding in it (10.5 => 11)
`fabs()` = changes negative value to positive
`factorial()` = gives the factorial of the given value, value shouldn't be negative or non-integer (3 => 1*2*3 => 6)
`floor()` = changes float value to int by subtarcting in it (10.5 => 10)
`fsum()` = can get list & tuple as a parameter to add its element and return it
`sqrt()` = returns the square root

▼ Random Module

```
import random as r
```

`randint()` => takes 2 arguments and gives the random value from between them and both parameter are included
`randrange()` => same as `randint()` but second argument is not included
`choice()` => return a random element from a list
`random()` => retrun random float between 0 to 1
`shuffle()` => takes a sequence and return the it in random order
`unifrom()` => returns a random float between 2 parameters

▼ Date and time module

```
import datetime as dt
now = dt.datetime.now()
```

`print(now)` => gives current date and time
`datetime(2022,8,16)` => gives parameter in date and time format

```
print(now.strftime()) => shows everything of now but takes arguments as
        %b = Dec
        %B = December
        %m = 12 (month)
        %y = 21 (year)
        %Y = 2021 (year)
        %H = 17 (hour in 24)
        %I = 7 (hour in 12)
        %p = PM (AM/PM)
        %M = 54 (minutes)
```

▼ Pickle module

> This module implements a fundamental, but powerful algorithm for serializing and DE serializing a python object structure

```
import pickle as p
```

`dump()` -- to serialize an object hierarchy
`load()` -- to de-serialize a data stream
`wb` -- write binary
`rb` -- read binary

▼ OOPs (Object Oriented Programming)

```
class DemoClass :
  a =10
  def sumValue(self) :
    print(20+30)

demo = DemoClass()
demo1 = DemoClass()

print(demo.a)
print(demo1.a)
demo.sumValue()
```

💡 **Method** -- if function is define in class it is known as method and you have to call it
when making a method you have to pass any one argument (self) and if you are using class variable then you have to call self to use it
Constructor -- same as method but you don't call it

```
class Dummy:
  a=10
  def init(self):
    print('This is a Constructor')

  def showValue(self):
    print(self.a)
    print(self.a*self.a)

  def show(self,a,b):
    print('this is tabish shaikh ',a,b)

obj = Dummy()
obj.showValue()
obj.show(5,7)
```

▼ Inheritance

- **Single level**

```
class A:
  def displayA(self):
    print('Tabish A')

class B(A):                           \\passing class as a argument to another class
  def displayB(self):
    print('Tabish B')

obj=B()                               \\ object b can use class A's methods
obj.displayB()
obj.displayA()
```

- **Multi level**

```
class A:
  def displayA(self):
    print('Tabish A')

class B(A):                     \\passing class as a argument to another class
  def displayB(self):
    print('Tabish B')

class C(B):                     \\passing class as a argument to another class
  def displayC(self):
    print('Tabish C')

obj=C()                         \\ object of C can use class B's as well as A's methods bcoz its passed from c to b to a
obj.displayC()
obj.displayB()
obj.displayA()
```

- **Multiple level**

```
class A:
  def displayA(self):
    print('Tabish A')

class B:
  def displayB(self):
    print('Tabish B')

class C(A,B):                \\passing class as a argument to another class
  def displayC(self):
    print('Tabish C')

obj=C()                      \\ obj of C can use class B's and A's methods as they were passed in C class but object B can't us
obj.displayC()
obj.displayB()
obj.displayA()
```

▼ Encapsulation

> An objects variable should not always be directly accessible
> the method can check the correct values are set, if not then return an error

```
class Student:
  def init (self):
    self.__name=""
  def getname(self):
    return self.__name
  def setname(self,name):
    self.__name = name

obj = Student()
obj.setname("Tabish")
name = obj.getname()
print(name)
```

```
class Student:
  __name = "shaikh" #the double underscore means private or encapsulation. this can't be use in object until constructor ask
  def init(self):
```

```
    print(self.__name)
    self.__displayInfo()
  def __displayInfo(self):
    print("this is private method")

obj=Student()
```

▼ Polymorphism

> it means same function name (but different signatures) being uses for diff. types => `len([12,32,2])` => `len('shaikh)`

▼ Overloading

> same method different type of parameter

```
class sh:
  def dInfo(self,name=''):
    print('welcome to terminal',name)

obj = sh()
obj.dInfo()
obj.dInfo('tabish')
```

```
# another example
class Area:
  def find_area(self,x=None,y=None):
    if x != None and y != None:
      print('The Area of Rectangle is ', xy)
    elif x != None:
      print('The Area of Square is ', xx)
    else:
      print('Nothing to find')

obj=Area()
obj.find_area()
obj.find_area(10)
obj.find_area(10,20)
```

▼ Overriding

> to use the same name method from parent class

```
class sh:
  def dInfo(self):
    print('welcome to SH')

class jh(sh):
  def dInfo(self):
    super().dInfo() #to use the same name method from parent class use super
    print('welcome to JH')

obj = jh()
obj.dInfo()
```

▼ Error and Built-in Exception

1. **Syntax Error**
2. **Logical error (Exception)**
   a. Zero Division Error (`a=0` -- `print(a/0)`)
   b. Name Error (`print(b)`)
   c. Type Error (`print('10' + 10)`)

d. Value Error (`a=int(input(enter))` -- `a='hello'`)

e. Index Error (`l=[1,2,3,4]` -- `print(l[6])`)

f. Key Error (`a={'name':'error','fee':7000}` -- `print(a['fees'])`)

g. Module Not Found Error (`cal.py` -- `import cals`)

h. Import Error (`sum()` -- `from cal import sum1`)

```
num1 = input('enter num1')
num2 = input('enter num2')

try:
  print('num1 + num2 => ',num1+num2)
except Exception as e:
  print(e)

print('done')
```

▼ SQLite

> extension => .db

```
import sqlite3      #importing sqlite module
con = sqlite3.connect("sqlite1.db")         #connecting OR connecting and creating database file in same path as py file
con.execute('create table tabish')      # creating table and passing column names and data structure that they will carry
con.close()     #closing file
```

```
this file is created by Md Tabish Shaikh for learning purpose.
```
