classmate

PULSE

6 Subject

INSERT DOODLED SELFIE HERE

NAME ANEES UR REHMAN KHAN

ADDRESS BANGALORE

MOBILE 9986786453 / 8088131784

EMAIL anees@arkprocoder.tech

EMERGENCY CONTACT 9986786453 [whatsapp]

Instagram : ark_dev

By:- ARK

73

PYTHON PROGRAMING

Introduction to python and Fundamentals.

1] What is python?
It is a programming Language Easy to understand as simple as reading English.

Compiled vs Interpreted languages

Compiled languages are converted directly into the machine code that processor can execute as a result it is faster and more efficient

* Whenever we do the changes in program we need to rebuild the program
Ex:- C / C++ / Go

Interpreter run through a program line by line Execution. It is slower than compiled language. In Interpreters source code is not directly translated by the target machine. Instead different program abc the interpreter reads and Executes the code. Ex:- php / Ruby / python / JS

Example:- Greek book translation.

Advantages (C)	Disadvantages. (C)
* faster than Interpreted code. because it run time adds to overhead.	* Additional step time needed to complete Entire compilation step before testing * platform dependent Binary code.

(Python)	Disadvantage is
* Dynamic typing small code code itself is a platform independent	this is slower than compiled.

Why phython is hybrid ?

The python interpreter first reads the human code and optimizes it to some intermediate code before interpreting it into machine code. thats why you always need another program to run to python.

Python Versions (https://python.org/doc/versions/)

Python 1.4 25 October 1996

python 3.9.1 8 december 2020

Python 3.9.2 19 feb 2021

python programming building blocks like Variables.

1. Input & processing 3.
2. Output.

Input → data receives when the program is running

processing → Mathematic calculation

Output → final output

Displaying Output.

Print Function. → It is a pre-written code that performs an Operation.

print function output on the screen.

DATA TYPES

1. Boolean → True / False
2. Integer (int) → For storing integer value.
3. floating (float) → Storing decimal values
4. Storing (str) → store storing in memory.
5. list → are also referred to as arrays that can contain mixed datatypes.
6. Object → User defined types.
7. None. → None Keyword is an object.

Download python IDE & VS code or Pycharm.

Link:- https://www.python.org/download/release/python-392/

VS cod → https://code.visualstudio.com/download.

pycharm → https://getbrains.com/pycharm/download/.

Rules For Defining a variables.

* A variables can contains alpha, num, digits,
* A variables can start with alpha underscore but not with numbers
* No white spaces should be used.

```
a = 10
print (a)
print (type(a))
```

o/p:- 10
class <INT>

```
a = 10.98
print (a)
print (type(a))
```

o/p:- 10.98
class <Float>.

```
a = "ARKprocoder"
print (a)
print (type(a))
```

o/p:- ARKprocoder
class <str>.

Strings

(sequence of characters)

The text which is enclosed in double or single or triple quotes we called that as strings.

print (mystr.isalnum())

o/p:- False because it has space so it not alphanumeric (boolean value return)

print (mystr.isalpha())
o/p:- False.

print (mystr.Ends with("boy"))

o/p True.

print (mystr.count("b"))
o/p 1

print(mystr.capitalize())
o/p : Anees is a good boy.

print(mystr.find("is"))
o/p : 6 // It tells the index position.

print.(mystr.lower())
o/p :- anees is a good boy

print(mystr.upper())
o/p: ANEES IS A GOOD BOY.

print(mystr. startswith ("Anee")

o/p:- True.

String replace (oldword, newword)

Escape Sequence Characters

Sequence characters after backslash '\'

Escape Sequence Characters comprises of more than one characters but represents one character when used within the string.

Example: \n \t \' \\ etc.

newline. Tab single quote. backslash

Lists and Tuples.

Python lists are containers to store a set of values of any data types.

```
mylist = ["Apple", "Banana", 7, False]
```

0 1 2 3.

We can store any values of any datatype.

List indexing starts from 0

List Methods.

```
numbers = [2, 3, 4, 5, 9, 8, 6, 7]
numbers.sort()
```

numbers.reverse()

Slicing

```
print(numbers[:])
print(numbers[:8])
print(numbers[3:8])
print(numbers[::1])
print(numbers[::-1])
print(numbers[1:8:2])
print(len(numbers))
```

numbers.append(10)

numbers.Insert(2, 22)
↓
Index

numbers.remove(22)
numbers.pop()

numbers.clear()

Tuple

mutable: Can change Ex:- list
Immutable: Cannot change Ex:- Tuple

tp = (1, 2, 3)

tp[1]=8. //

Error Cannot change tuple

Swaping

a=1 b=8 temp=a b=temp

print(a, b)

print(b, a)

a = (1, 7, 2) || a(1,) Tuple with one ele need ,

a.count(1) : a count [1] will return number of times 1 occurs in a

a.index(1) : a.index(1) will return the index of first occurance of 1 in a.

Taking Input from User

Input() || default acts at string

a= int (input(" Enter number"))

b= float (input (" Enter float number"))

c= String (input (" Enter name ") .

Dictionary

Dict is a collection of Key-Value pairs

Syntax:-

```
a = {   "Key" : " Values",
        "Anees" : " Programmer",
        "marks" : "100",
        "list" : [1, 2, 3] }
```

a["Key"] => prints "Value"
a["list"] => print [1, 2, 3]

Properties of Python Dict

1] It is unordered
2] It is Mutable
3] It is Indexed
4] Cannot contain duplicate keys.

1. a.items() : return a list of (Key, Value) tuple
2. a.keys() : return a list containing dict keys
3. a.update({"age": 23}) : update the dict with supplied key-value pair
4. a.get("name") : returns the value of the specified keys and value is returned

Sets

Set is a collection of non repetitative elements.

```
s = set()
s.add(1)
s.add(2)
```

OR set = {1, 2}.

// No repetation allowed.

__/__/__

properties of set

1] Sets are unordered // Order doesn't matter
2] Sets are unindexed //cannot access by index
3] There is no way to change items in set
4] sets cannot contain duplicate values.

s={1, 8, 2, 3, 4, 5}.

len(s)
s.remove(2)
s.pop()
s.clear
s.union({-})
s.intersection({})

s= set()
s= {1, 2, 3}
s1= s.union({2, 3, 4, 5})

print(s1)
o/p:- {1, 2, 3, 4, 5}.

s2= s1. s.intersection({3, 4})
print(s2)
o/p {3, 4}

print(s.issdisjoint(s1))

return T/F.

Operators in Python

① Assignment

```
x = 5
print(x)
x += 7
print(x)
```

x -= 7	x = x - 7
x /= 7	x = x / 7
x %= 7	x = x % 7

② Comparision.

<, >, <=, >=, ==, !=

```
i = 8
print(i == 5)
```

o/p :- False

```
print(i != 5)
```

o/p :- True.

Logical

AND OR NOT

```
i = 5
print (i and 5)
```

o/p = True.

a=True
b=False

print (a and a)
T
print (a or b)
T
print (a and b)
F

Identity operator

is and is not
print (a is not b)
True

print (5 is 7)
False.

Membership Operator

list1= [3, 4, 5]
print(3 in list1)
o/p =True

print (2 not in list1)
o/p: False.

Bitwise operator

print(0 & 1)
o/p:- 0

print(0 | 1)
o/p = 1

Conditional Expression.

Decisions taken when condition being met.

If Else and elif in python.
If Else and Elif statements are a multiway decision taken by our program due to certain conditions in our code.

syntax:

```
if (condition):
        print("Yes")
elif (condition 2):
        print("No")
Else
        print("Maybe").
```

There can be any number of elif statement last else is executed if all elif statement fails.

Loops in Python:

Sometimes we want to repeat a set of statements in our program For instance print 1 to 1000

loops make it easy for a programmer to tell the computer which set of instruction to repeat and how.

__/__/__

Type of loop in python

1] While Loop
2] For Loop.

While loop

syntax :-

```
While (condition):
    # Body of the loop.
```

In while loop the condition is checked first If it Evaluates to true the body of the loop is excuted otherwise not.

If the loop is Entered the process of condition checks & Excution is continued until the condition becomes false.

For loop

A for loop is used to iterate through a Sequence like list, tuple or string iterables

```
l = [1,2,7,3,8]

for item in l:
    print(item)
```

Range function in python

Used to generate a Sequence of numbers.

Range (Start, Stop, Step-size)

Break Statement:

'Break' is used to come out of the loop when Encountered it interrupts the program to Exit the loop now.

Example:-

```
for i in range(0, 50):
    print(i)
    if i == 3:
        break
```

//0, 1, 2 & 3

continue statement

The continue is used to stop the current iteration of the loop and continue with the next one it instruct the program to "skip this iteration"

Example:-

```
for i in range(4):
    print(i)
    if i == 2:
        continue
    print(i)
```

if 2 is a iterations skipped.

Pass statement.

Pass is a null Statement in python. it mean not to do anything.

```
a = [1, 2]
for i in a:
    pass
```

// pass without error

(Look on to the Practice Programes. set 1)

Files handling in python

Random Access memory is volatile and all its contents are lost once the program terminate. In order to persist the data forever we use files.

A file is a data stored in a storage device A python program can talk to the file by reading content from it and writing content to it.

RAM => Volatile HDD => Non Volatile.

Types of files

1] Text Files (.txt)
2] Binary Files. (.JPG, .data, .bin etc)

Modes to files

"r" => Open file for reading (read mode)
"w" => write mode
"x" => Create a file if not exist If already exist. it wont create.

"a" => Append mode (Add more content to the file)

"t" ⇒ Text mode (deals with .txt files)

"b" ⇒ Binary Mode

"+" ⇒ read + write mode

"rt" ⇒ read + text mode (default)

Example

Create a file ank.txt

Anees is a good boy Anees is the King of universe Anees is very smart.

//Now we want to read this file content

```
f = open("ank.txt")
```

↓
file pointer

```
content = f.read()
print(content)
f.close().
```

//output:- Anees is a good boy
Anees is the King of universe
Anees is very smart.

__/__/__

```
f= open ("ark.txt", "rt")
content = f.read (3)
print(content)
f.close()
```

o/p:- Ane

If you want to print character by character

```
f= open ("ark.txt", "rt")
content = f.read()
for line in content:
        print(line)
f.close()
```

o/p:- A
N
E
E
A
:

If I need to print character in Horizontal fashion

```
for line in content:
        print( line, end= " ")
```

Readline() and Readlines()

```
f= Open("ark.txt", "rt")
print (f.readline())
```

o/p:- Anu is a good boy.

```
print (f.readline())
```

o/p:- Anu is King of Universe.

print (f.readline())
o/p: Anu is smart

f.close()

Basically readline() function read a single line from the file, If we print two line of text from a file we need to write 2 times readline() function.

Readlines()

This function is use to read all the lines from the file and store it in a List format so we can Extract any line by giving array Index.

```
f = open("ank.txt", "rt")
print (f.readlines())
```

O/p:- ['Anus is a good boy \n', 'Anu is King of universe\n', 'Anus is very smart'\n',]

Writing files

```
f = open("ank.txt", "w")
f.write("ANK is coder")
f.close.
```

It lose previous data in ark.txt and add new text data which is passed. ("Ark is code").

ark.txt

Ark is Code.

If I want to add a new line without losing the previous content I will be using Append mode.

```
f= open("ark.txt", "a")
f.write("\n Ark is Programmer")
f.close().
```

ark.txt

Ark is code Ark is Programmer.

← replaced and not lost old content.

Handle read and write Both

```
f= Open("ark.txt" "r+")
print (f.read())
```

o/p :- Ark is code
Ark is Programmer.

f.write ("\n Thank You")
f.close()

ark.txt

Ank is Coder Ank is Programmer Thank You.

f.tell() :- It tell the position of character

a) f.seek(20) :- It start from 20th character. and print the line.

b) f.readline()

How to read files using with function block

```
with open ("ark.txt") as f:
    a= f.read(5)
    print(a)
```

o/p:- Ank.

F strings :- It used to read string & variables

```
Mc="Ameer"
a= "Hello"
print(f "{a} this is {mc}")
```

o/p:- Hello this is Ameer.

Functions

A function is a group of statements performing a specific task

When a program gets bigger in size and its complexity grows it gets difficult for a programmer to keep track on which piece of code is doing what.

A function can be reused by the programmer in a given program any number of.

Syntax:-

```
def func1():
    print ("Hello")
```

This above func1 can called any number of time anywhere in the program.

function Call

whenever we want to call a function we put the name of the function by parenthesis.

func1() → This is called function call.

function definition

The part containing the exact set of instruction which are executed during the function call.

Types of functions

1] Built in function
[Already built in function in python like sort(), reverse, upper() so on..]

2] User Defined function.
[we will define newly].

Functions with arguements

A function can accepts some value it can work with we can put these value in the parenthesis.

A function can return value. as shown.

```
def greet (name):
    greet= "Hello" + name -
    return greet

a= greet("ARK")
```

Default Parameter Value

We can have a value as default argument in a function.

```
def greeting(name = "Anees"):
    #function body
```

greeting() => name is Anees default.
greeting("ARK") -> name will be ARK when function body is passed.

Recursion

Recursion is a function which calls itself. It directly used mathematical formula.

factorial(n) = n x factorial(n-1)

```
def factorial(n):
    if n==0 or n==1:
        return 1

    else
        return n * factorial(n-1)
        #function calling itself.
```

Work flow

factorial (4)
4 x factorial (3)
4 x [3 x factorial (2)]
4 x [3 x 2 [x factorial (1)]]
4 x 3 x 2 x [1] [function returned]

① a=9 b=8
d= sum (a, b) // built in functions.
print (d)

sum= 17//

def function1():
print ("Hello you are in function1")

print (function1())

o/p:- Hello you are in function1.
None

If we remove print(in print(function1()))
None wont come.

② def func2 (a, b):
print ("Hey ", a+b)

func2 (6, 5)

o/p:- Hey 11

③ def fun3 (a, b):
av = (a+b)/2
print (av)

fun3 (3, 3)
o/p:- 3//

(A)
```
def fun4(a,b):
    av=(a+b)/2
    print(av)
    return av.

v= fun4(5,5).
print(v)
```

O/p:- 5.
5 so #print(average)

DOC String

```
def function2(a,b):
    """ this function calculate average of
    two numbers"""

print(function2.__doc__)
```

O/p:- This function calculate average of two numbers.

Try Exception handling.

```
num1 = input()
num2 = input()
```
} take input string.

1/
ans
Error we get so we can handle try and except.

```
print("Enter number 1")
num1 = input()

print("Enter number 2")
num2 = input()

try:
    print("the sum of 2 numbers",
    int(num1) + int(num2)).

except Exception as e:
    print(e).

print("this is amazing")
```

O/p:- Enter number 1
2
Enter number 2
www

Invalid literal for int() with base 10: "www"

This is amazing.

Global Keywords and Global Variables.

① def function1(n):
print (n, "I have printed")

function1 ("this is me")

o/p:- This is me I have printed.

② l=10 # Global variable

def function1(m):
l= 5 # local variable.
m= 8
print (l, m)

function1()

o/p:- 5, 8.

always local variable is first preference inside the functions.

③ l= 10 #Global
def function1(n):
m= 8
l= l+45
print (l, m)

// wont give permission l=10

function1 ()

Op:- Error cant add global one.

Lamda functions or Anonymous function

```
def add (a, b):
    return a + b

minus = lambda x, y: x - y    // can write in single line

print (minus (9, 4))
```

O/P: 5 //.

f strings

```
me = "Aneeq"
age = 21
print(f" my name is {me} and I am {age}")
```

O/P: my name is Aneeq and I am 21

f strings are enclosed in ' ' or " " single quotes or double quotes. We can pass both string and variables in it.

*Args and **Kwargs

Args is a short form used for arguments it is used to unpack arguements in case of *args the argument could be list or tuple.

Note: the name args can be used anyname like argsances or ances or anything just * makes the difference *args or *ances. (*) Asterisk.

**Kwargs

Full form is Keyword arguements. It passes the data in the form of dictionary the **Kwargs sends the arguements in the Key Value pairs.

Note: Same as args. We can use any name for Kwargs matter is only **

Example for *args

```
def funargs(normal, *args):
    print(type(args))
    print(normal)
    for i in ~~range~~ args:
        print(i)

normal = "This is normal value"

funargs(normal, "ARK", "Python", "JAVA")
```

O/P: tuple
This is normal
ARK
Python
JAVA

note :- (normal, *args) ✓
(*args, normal) X (wrong syntax)

Example for. **Kwargs

```
def func_arg_kwarg(normal, *args, **kwargs):
    print(normal)
    print("\n now args will print")

    for n in args:
        print(f"Name is {n}")

    print("\n Now kwargs will print")
    for name, des in kwargs.items():
        print(f"{name} is {des}")


names = ["ARIE", "MEHHS", "HARSH"]

title_ark = "Besties"

kw = {
    "ARIE": "Developer",
    "MEHH": "Pastry Lover",
    "HARSH": "Naughty".
}.

func_arg_kwarg(title_ark, *names, **kw)
```

o/p

Begins
now args will print

Name is ARK
Name is MEGHS
Name is HARSH

Now kwargs will print.

ARK is Developer
MEGH is pastry Lover
HARSH is Naughty.

Time module

import time.

In python time can be tracked through its built in libraries. the time module consists of all time related functions.

time.time() => It returns us seconds.

time.asctime() => It returns full format.

```
import time
seor = time.time()
print(seor)
print(time.asctime())
```

time.sleep(5)

It will sleep program for 5 seconds

time.localtime()

The time.localtime() is used to convert the number of seconds to local time

This function takes seconds as a parameter and returns the data and time in time.struct_time format.

```
import time
print("time is localtime() returns :%s",
      %time.localtime())
```

Virtual Environment

In python we create projects and some projects will accept the only modules/library with suitable versions stay so if we use a normal python interpreter and later if we install module and library it may affect to our project created so we use virtualEnvironment for projects later if we run the project after n years our project should run without errors.

Installation

1. pip install virtualenv

2. //create a folder Jarvis

C://Jarvis.

//open command prompt

3. virtualenv . // virtualenv dot

//folder will create a scripts files and so on

4. //Activate the folder by entering below command
./Scripts/activate

(Jarvis) ://

how we can install any modules libraries
pip install django
pip install pandas so on . . .

Note:- If we install modules or libraries inside the virtualenvironment it will effects on only that particular folder its like not related to any other system folders or libraries.

pip freeze > requirement.txt

// It create requirement.txt file which will contains all the modules libraries which needed to run the python project.

Enumerate functions

An Enumerate is a built-in function that provides an index to data structure elements making iterations through them easier and simpler.

Syntax:- enumerate(iterable, start=0)

When we calling a simple Enumeration function we have to provide two parameters.
* The data structure that we want to iterate
* The index from where we want to start our iteration.

Example : let Enumerate using list first

```
mylist = ["ARK", "MEGH", "Aadi", "Harshit",
          "Amrutha"]
for index, name in mylist:
        print(index, name)
```

o/p:-

```
0 ARK
1 MEGH
2 Aadi
3 Harshit
4 Amrutha
```

Now lets use Enumerate function.

```
newlist = enumerate(mylist, 5)
print(list(newlist))
```

__/__/__

O/P:- [(5, 'ARK'), (6, 'MEGH'), (7, 'Aadi'), (8, 'Harshit'), (9, 'Amrutha')]

We can also convert the Enumerator object into a list(), tuple(), set() and many more

Using List

```
l1 = ["MEGH", "ARK", "HARSH", "AADI"]

i = 1
for item in l1:
    if i % 2 != 0:
        print(f"My Besties {item}")
    i = i + 1
```

Using Enumerate

```
for index, item in enumerate(l1):
    if index % 2 == 0:
        print(f"my besties {item}")
```

O/P:- ~~AP~~ My Besties MEGH
My Besties HARSH

Import functions

```
import sys
print(sys.path)
```

It prints the system path of file location

file1.py

```
def greet(str, gender):
    print(f"Hello {str} gender is {gender}")
```

file2.py

```
import file1
print("This is file 2")
file1.greet("ARK", "Male")
```

(or)

file2.py

```
from file1 import greet
greet("ARK", "Male")
```

O/P:- This is file2
Hello ARK gender is Male.

// Basically we can import files and run the functions of the files in calling in different python files.

If __name__ == __main__

* Using the main in our file we can restrict some data from exporting to other files when imported

* We can restrict the unnecessary data thus making the output cleaner and more readable

* We can choose what others may import

__/__/__

On what they may not while using our module.

Modules in python has a special attribute called __name__. The value of the __name__ attribute is set to __main__ when the module is run as the main program. Otherwise the value of __name__ is set to the name of the module. The if __name__ == "__main__" block prevents the certain code from being run when the module is imported.

Let us Understand with the Example

1. main1.py

```
print("__name__ in main1.py is set to "
      + __name__)

def add(num1, num2):
    return num1+num2

print("The name is", __name__)

if __name__ == '__main__':
    res = add(14, 6)
    print(f"In main output {res}")
```

main2.py

```
import main1

print("__name__ in main2.py is set to "
      + __name__)
```

o/p:-

// Running main2.py

output:- __name__ in main1.py is set to main1
The name is main1
__name__ in main2.py is set to __main__

// Run main1.py

output:- __name__ in main1.py is set to __main__
The name is __main__
In main output 20.

Note:- i by using __main__ & name concept we can avoid running the whole program from another python file.

Join, Map, filter and Reduce functions

Join → Join is a function, the iterable can be list, dict, tuple, or string. that returns a string by joining the elements of an iterable using a string or character of our choice.

Example

```
mylist = ['ARK', 'MEGH', 'HARSH', 'AADI']
Separator = " , "
print (Separator.join (mylist))
```

o/p:- 'ARK', 'MEGH', 'HARSH', 'AADI'

```
Mydict = {"name": "ARK", "age": 21 }
Separator = "_Seprate_"
print (Separator.join (Mydict))
```

o/p:- name_Separator_age.

Note:- Join() only works if some type of data is in list or dict or string variable

```
a = [1, "ark", [1, 2]]
Sep = '_'
print (Sep.join (a))
```

o/p:- Error Expected int

map()

A map function Executes certain instructions or functionality provided to it on Every item of an iterable.

Syntax:- map(function, iterable)

1.
```
items = [1, 2, 3]
a = list(map(lambda x: x**3), items))
print(a)
```
o/p:- [1, 8, 27]

2.
```
def square(a):
    return a*a

def cube(a):
    return a*a*a

functions = [square, cube]

mynum = [2, 3, 5, 6]

for i in mynum:

    val = list(map(lambda x: x(i), functions))
    print(val)
```

o/p:- [4, 8]
[9, 27]
[25, 125]
[36, 216]

Filter()

A filter function in python test a specific user-defined condition for a function and returns an iterable for the elements and values that satisfy the condition or in other word return true"

Syntax:

filter (function, iterable)

Example

1. a = [1, 2, 3, 4, 5, 6]

b = [2, 5, 0, 7, 3]

c = list (filter (lambda x: x in a, b))

print (c)

o/p:- [2, 5, 3]

2. def greater_(num):
    return num > 5

res = list (filter (greater_, a))
print (res)

o/p:- 6

reduce()

Reduce functions apply a function to every item of an iterable and gives back a single value as a resultant.

We import reduce function is functools module.

Syntax:- reduce(function, iterable)

```
1. from functools import reduce
   a= reduce((lambda x, y : x * y),
                [1, 2, 3,4])
   print(a)
```

o/p:- 24

Python Decorators

Decorator as can be noticed by the name is like a designer that helps to modify a function. The decorator can be said as a modification to the external layer of function, as it does not make any change in its structure. What a decorator does is, it takes a function and insert some new functionality in it without changing the function itself.

Two ways to write python decorator

* We can pass our function to the decorator as an argument, thus define a function and pass it our decorator

* We can simply use @ symbol before the function we would like to decorate.

Example

```
def ark(func):

        def megh():
                print("MEGH FUNCTION IS
                                EXECUTING")
                func()
                print(" AFTER FUNCTION IS
                                EXECUTING")
        return megh

@ark
def harsh():
        print ("HARSH FUNCTION IS EXECUTING")

harsh()
```

o/p:- MEGH FUNCTION IS EXECUTING
HARSH FUNCTION IS EXECUTING
AFTER FUNCTION IS EXECUTING

```
def ark():
    def magh():
        # body
    return magh.

def harsh():
    # body

harsh = ark(harsh)
harsh().
```

// One more way to define the function.

//

OBJECT ORIENTED PROGRAMMING.

It is nothing but DRY [Do not Repeat Yourself]

In OOPs we deal with the classes and Objects

A class is a collection of objects and object is defined as a instance of class possessing attribute

Along with these we have

* Instances
* Constructor
* Methods
* Abstraction
* Inheritance

__/__/__

Defining a class in python.

class Begins with a Keyword class

```
class Myclass:
    pass
```

Classes are not like function, so we do not have to use the Keyword define to create a class Instead we use the Keyword Class along with the name of the class Also we do not call a class as a whole Instead we use a object to access its different attributes.

Creating a Object

```
obj1 = Myclass()
obj2 = Myclass()
```

An object consist of:-

* The State which is represented by attributes of an Object which reflects the properties of an object

* The Methods of an object which represent the behavior of the object and the response of an object with other objects

* Identity which gives a unique name to an object so that one object can interact with other objects.

Example

```
class Friends:
    pass

meghs = Friends()
harsh = Friends()

meghs.name = "MEGHANA"
harsh.name = "HARSHIT"
meghs.hobby = ["cooking", "Reading"]
harsh.hobby = ["playing Cricket", "carrom"]

print(meghs.name, harsh.name)
print(harsh)
```

O/P: MEGHANA, Harshit
<--main--.Friends object at 0XYZ>

Instance and class Variables.

Instance Variables are the Variable for which the value of the Variables is different for Every Instance

Instance Variable	class Variable
* when a object is created with the use of the new Keyword, instance Variable are Created They destroy when object destroy	* When a pgm Starts Static Variables are created and destroys when pgm stop
* ObjectReference.Variable Name	* can access by ClassName.VariableName

Example

```
class Friends:
    stars = 5

meghs = Friends()
harsh = Friends()

meghs.name = "Meghana"
harsh.name = "Harshit"
meghs.age = 20
harsh.age = 21

print("\n)
print(harsh.name, harsh.age)
print(harsh.age, meghs.age)

print(f"This is harshit stars {harsh.stars}")
print(f"This is Meghana stars {meghs.stars}")
print(f"This is class Friends stars {Friends.stars}")

print(Friends.__dict__)
print(harsh.__dict__)

meghs.stars = "infinity"
print(meghs.__dict__)

print("this is Meghana stars after updating
    is ; {meghs.stars}")

Friends.stars = 8
```

print(f" This is Harshith stars {harsh.stars}")
print(f" This is Class Friends stars after updating : {Friends.stars}")

Output

Harshit 21
21 20
This is Harshit stars 5
— " — Meghana — " — 5
— " — Friend — " — 5

{'__module__': '__main__', 'stars': 5, '__dict__'
Friend ... etc }.

{'name': 'Harshit', 'age' 21}
{'name': 'Meghana', 'age', 20, 'stars': 'infinity'}.

This is Meghana stars after updating is : infinity
This is harshit stars 8
This is Class Friends stars after updating is : 8

Method, Self & __init__() Constructors

Method is just like a function the purpose of method is used to show all the details related to the object

Self

The self keyword is used in the method to refer to the instance of the current class we are using the self keyword is passed as a parameter explicitly every time we define a method.

```
def read(self):
    print(self.name)
```

__init__

It is a also called as constructor in oops

Constructors in python is used to assign values to the variables or data members of a class when an object is created.

```
def __init__(self):
    #body of constructor.
```

* The def keyword function
* init method arguements are optional we can define and pass n arguements
* The first arguement refers to the current object which binds the instance to the init() method.

Example

```
class Bestfrs:
    stars = 5

    def __init__(self, bname, bage, blook, bheight,
                 bhobby):

        self.name = bname
        self.age = bage
        self.look = blook
        self.height = bheight
        self.hobby = bhobby.

    def besticDetails(self):
        return f"My bestie name is {self.name}\n
Age is {self.age}\nLook is {self.look}\n Height is
{self.height}\n Hobbies are {self.hobby}\n No. of
stars{self.stars}"

aadi = Besties("Aadithya", 21, "cool", "Medium shaped",
               "Watching Movies")
Harsh = Besties("Harshith", 21, "Handsome", "Medium
               shout", "Talking")

print(aadi.name)
print(aadi.besticDetails())
print( Harsh.name)
print( Harsh.bestie Details())
```

Output

Aadithya
My butic name is Aadithya. Age is 21
Look is cool
Height is Medium short
Hobbies are Watching Movies.
No, of stars 5.

// Same point for harshit too //

Class Methods In Python.

Syntax:-

```
class myclass

    @classmethod
    def myfunc(cls, arg1... ):
        #body
```

myfunc is a function that need to be converted into a class method

returns: @classmethod returns a class method for function.

@classmethod Decorator

A @classmethod Decorator is a built-in function in python. It can be applied to any method of the class. We can change value of it too.

Class method	Static Method
* Taking a class or in short form of cls as an argument is must for class method.	There is no such restriction.
* With the help of class method we can change and alter the variable of class	with this we can't change the class state
* We can only use inside the class.	not restricted

Example [Main Example] → (1)

```
class Besties:
    stars = 5

    def __init__(self, bname, bage, blook, bheight,
                 bhobby):

        self.name = bname
        self.age = bage
        self.look = blook
        self.height = bheight
        self.hobby = bhobby.

    def bestieDetails(self):
        return f"My bestie name is {self.name}
\nAge is {self.age} \n {self.look} \n {self.height}
\n {self.hobby} \n {self.stars}."
```

__/__/__

```
@classmethod
def change_stars(cls, newstars):
    cls.stars = newstars

Megh = Besties("Meghana", 21, "Pretty", "Medium Short",
    "Eating Pastries", Racing")

Harsh = Besties("Harshith", 21, "Handsome",
    "Medium Short", "Talking")

print(Megh.name)
print(Megh.stars)

Megh.change_stars(10)

print(f"After updating new stars {Megh.stars}")

print(Megh.bestieDetails())
print("++++++++++++++")

Harsh.change_stars(15)
print(Harsh.bestieDetails())

print("*****")
print(Besties.stars)
```

O/P

5
Meghana
5
After updating new stars 10

My bestie name is Meghana
Age is 21
Look is Pretty
Height is Medium Short
Hobbies are Eating Pastries and Racing
No of Stars 10

++++++++

My bestie name is Harshit
Age is 21
Look is Handsome
Height is medium Short
Hobbies are Talking
No of Stars 15

****
15

Split()

```
text= "ARK Pro Coder"
p = text.split()
print(P)
```

• ['ARK', 'Pro', 'Coder']

For previous example classes method in python I can use split() function

Example

__/__/____

```
class Bustin:                    [ Refer previous Example ]
    stars = 5
    def __init__(self, ....):
        #body.

    def BustieDetails(self):
        #body

    @class method
    def change_stars(cls, newstar):
        #body.

    @classmethod
    def message(cls, greet):
        # context = greet.split("/")
        # print(context)
        # return cls(context[0], context[1],
          context[2], context[3], context[4],
        return cls(*greet.split("/"))

Aadi = Bustin.message("Aadi/21/Cool/Tall/Movielover")
print(Aadi.name)
print(Aadi.hobby)
```

O/P:- Aadi
Movielover.

Static Method

To define static method we use @static

Example [Refer Main Example 1]

```
class Beauties:

    #body
    def __init__ :
        #body
    def beauticDetails:
        #body
    @classmethod:
        #body
    @classmethod..
        #body.

    @staticmethod
    def greet(str):
        print(f"{str} is the beauty")


Megh= Beauties("Meghana", 21, "Pretty", "Medium Short",
        "Eating")

Beauties.greet("Harshith")
Megh.greet("Meghana")

O/p: Harshith is the beauty
     Meghana is the beauty.
```

__/__/__

Abstraction and Encapsulation.

Abstraction:- Refers to hiding the unnecessary details so that you could be on the whole product instead of parts of the product separately. It is a mechanism that represents the important features without including implementation details

Example:- Take example of bike., It has tyre, Engine etc all these combine to make bike which is Abstraction.

Encapsulation

Means: hiding the under layers. When working with classes and handling sensitive data global access to all the variables and in the program is not secure. In Encapsulation the internal representation of an object is generally hidden from outside to secure data.

Example:- capsule in which the medicine is Encapsulated.

Note:-
Abstraction and Encapsulation is fundamental concepts of OOPs Encapsulation take away all worry just give the product that he required. Abstraction focus on the working of the object and provide the working model.

Single Level Inheritance

Inheritance is the ability to define a new class (child class) that is a modified version of an Existing Class (parent class)

Syntax:-

```
class parent:
class child(parent):
  # child code
```

* Parent class is the one that is giving access to its methods or properties to the child class or derived class
* child class is the one that is inheriting methods and properties from the parent class.

Advantages of Inheritance

* Promotes code reusability
* Makes program most Efficient
* We can add more features to our already built class without modifying it or changing its functionality.
* It provide a representation of real world relationships.