Date
-09-23

Date / /
Page No.
Shivalal

INTRODUCTION

⊙ History of python:

⊙ Inventor of python : Guido van Rossum
Date of Birth of python : 20 Feb 1991

1989 ⟶ 20 Feb 1991

⟶ Guido van Rossum when working at NRI (national research institution) in Netherlands.

⊙ Python :- Python is a general purpose high level programming language.

⊙ Why name python :- when Guido Van Rossm began implementing python, he was also reading the published scripts from "Monty Python's Flying circus" BBC comedy series from the 1970s. He thought that he needed a name that was short, unique and slightly mysterious. So, he decided to call the language python.

⊙ Python Interpreter :-

⊙ Python being a scripting language. It contains a software package called Interpreter.

OR

→ A python interpreter is a computer program that converts each high level program statement into

machine code:
· An interpreter translate the command the you write out into code that the computer can understand.

⊙ Features of python:-

1. Free and open source.
2. Easy to code.
3. Easy to read.
4. object-oriented language.
5. High-level language.
6. Large community support
7. Easy to Debug.

⊙ Coding standard:-

→ Creating variables

python has no command for declaring a variable.
A variable is created the moment you first assign a value to it.

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

output =

5
John

⊙ Identifiers:

Date
09-09-23

Date / /
Page No.
Shivalal

WORKING WITH PYTHON

⊙ Basic syntax :-

(i) print statement :-

⊙ python syntax can be executed by writing directly in the command line;

Ex- print ("Hello world").

⊙ Python indentation :-

⊙ Indentation refers to the spaces at the beginning of a code line.
Where in other programming languages the indentation in code is for readability only, the indentation in python is very important.

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

→ python will give you an error if you skip the indentation:

Ex- if 5 > 2:
print ("Five is greater than two").

⊙ Variable :-

⊙ Variable are containers for storing data values.

Date / /
Page No.
Shivalal

String and string methods in python.

⊙ String :-

Collection of characters.

new_string = "my name is Aditya".
print (new_string)

Method

1. len().
to find length.

Ex- data = "Aditya"
x = len(data).
print(x)

2. .lower().

to make small.
name = "Aditya"
x = name.lower()
print(x).

3. .Upper()

to make capital.

4. .replace()

```
name = " Aditya pratap Singh"
x= name.replace("Aditya","Adi")
print(x).
```

5. .Capitalize().

"to make first letter of word capital.

6. .Center(?)

```
data = " Name".
x= data.Centre(100)
print(x)
```

7. .endswith(?)

check.

```
data = " Aditya Singh"
x = data.endswith("gh").
print(x)
```

8. .find(?)

find anything in string you want

```
data = " Aditya Singh"
x= data.find("i")
print(x)
```

9. .index(?)

find the position of character.

```
name = "Aditya"
X = name.index("d")
print(X).
```

10. .isalpha()

No space

check string is made of alphabet or not.

```
Ex =  data = "string"
      X = data.isalpha()
```

11. .isdigit()

No space.

check string is made up of digit or not

12. .isspace()

check whether the string is empty or not.

13. .istitle()

check whether all the word in string has capital (first character) or not.

14. .count()

to count any thing in string.

15. .Join(str_name)

List and List method

⊙ List :-

⊙ List is a collection which is ordered and changeable. It is create with using square bracket.

Ex- names = ["adityo", "Nitin", "prashant", "sumit"]
print (names)

Output :- ["adityo", "Nitin", "prashant", "sumit"]

Note :- List are mutable.

Ex- names = ["Adityo", "prashant"]
names[0] = "Adii".
print (names)

output :- ['Adii', 'prashant']

Methods

1. .append("?")

to add something at end and you add duplicate items also.

names = ["Adii", "sumit"]
x = names.append("prashant")
print (names).

2. .insert (index, —)

2. . remove ("?")

to remove something from list.

3. del

to delete list.

Ex -

```
data = ["Adi", "Sumit"]
del data
print(data)
```

output: Error. [bcz list is deleted]

4. . clear()

to clear data from list.

5. . reverse()

to reverse the list.

6.

6. . sort()

to make the list in assending order.

Tuples and tuples method

⊙ **Tuples :-**

⊙ Tuple is also a collection which is ordered and unchangeable.
It is created with the help of round bracked.
( ).

Note :- We cannot change tuples or they are immutable.

Methods

1. len(Tuplename).

   To find length.

2. to add.

   x = tup-1 + tup-2
   print(x).

3. . count (" ? ")

   to count something.

4. . index (" ? ")

   to find the position of something.

Sets and sets function (Methods)

Sets :-

A set is a collection which is unordered and unindexed. can't add duplicate item.
we create set with curly braked { }

Ex- items = {"rice", "sugar", "salt"}.
print (items).

Note:- They are unordered.

Methods

1. .add ("?")
to add something in sets. (only one item)

2. .update ("_", "_")
used to add multiples items in sets.

Ex-
names = {"Aditya", "XZ", "ABC"}
print (names).
names.update (["PQR", "MAC", "LPU"])
print (names).

Dictionary and Dictionary methods

⊙ Dictionary :-

⊙ A dictionary is a collection which is unordered, changeable & indexed.
We can create dictionary with curly bracket & then they have keys and values.

Ex-
```
student = {
    "name" : "Rohan",
    "class" : 4,
    "Rollno" : 23,
    "age" : 9
    }
print(student)
```

method

1. To print any specific value.

```
student = {
        " same as upper.
    }
print( student["name"]
```

Output:- Rohan.

2. To change the ~~key~~ value of any ~~value~~ key.

```
student["name"] = "xyz"
print(student)
```

3. len(dict name)

   to find length.

4. To add new items.

```
student{
    ""
    }
student["marks"] = "95%"
print(student)
```

5. To remove something from dictionary.

(i) .pop("___")

(ii) del student("___")

} same

6. To delete full dictionary.

(i) del student

7. .clear()

   to delete key-value pair.

8. .copy()

to make copy of dictionary.

Ex- student = {
    ||
}

phone = student.copy()
print(phone)

Date / /
Page No.
Shivalal

LOOPS

⊙ **while loop:-**

```
i = 1
while i<=10:
    print(i)
    i+=1.
    If i==4:
        Break
```

⊙ **Break:-**

```
i = 0
while i<=10:
    i+=1
    If i==4:
        Break
    print(i)
```

✗

```
i=0
while i<=10:
    print(i)
    If i==4:
        Break
    i+=1.
```

✓

⊙ **Continue:-**

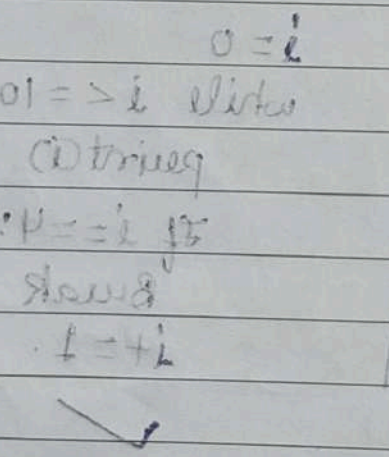
```
i=0
while i<=10:
    i+=1
    If i==4:
        continue.
    print(i)
```

⊙ For :-

Ex- for x in range(0,10):
print(x)
x+=1

To make table.

num= int(input("enter the no."))
for x in range(1,11):
b= num* x
print(num,"*",x,"=",b)



FUNCTIONS

⊙ Function:-

⊙ A function is a block of code that performs a specific task.

⊙ Types of function:-

1. Built in function.
2. User difined function.

⊙ Built in function:-

1. max((,))

check the maximum between or from two.

Ex
```
a = 10
b = 20
c = max((a,b))
print(c)
```

2. sum(,)

add.

Ex =
```
a = 10
b = 10
c = sum(a,b)
print(c)
```

Date / /
Page No.
Shivalal

User define function :-

- def my-function()

we make function by using def.

Ex-
```
a = int(input("enter the number"))
def my-square():
    b = a**2
    print(b)
my-square()
```

Date / /
Page No.
Shivalal

⊙ Lamda function:-

Ex- normal function

Ex-(i)
```
def squ(a):
    print(a**2)
squ(a)
```
normal

Ex- lamda function

Ex (ii)
```
squ = lamda a: a**2
print(squ(2))
```

Ex (iii)
```
tot = lamda a,b: a+b
print(tot(2,3))
```

FILE HANDELING

⊙ File handeling :-

⊙ Types of file in python :-

1. Text files (Ex.. txt)
2. Binary files (.jpg, .dat)

⊙ Opening a file :-

```
open ("file_name.txt", "r")
```

⊙ Reading a file in python :-

```
f = open ("Test.txt", "r")
myfile = f.read()
print(myfile)
f.close()
```

first make file named "test.txt". and then do this.

```
f = open ("Test.txt", "r")
    for x in f:
        print (x)
```

Date / /
Page No.
Shivalal

⊙ write a file in python:-

```
f = open("text1.txt","x")
```
— make file in python.

```
f = open("text1.txt","w")
f.write("_____")
f.close()
```

⊙ append a file in python (update):-

```
f = open("text1.txt","w")
f = open("text1.txt","a")
f.write("_____")
f.close()
```
} overwrite

```
f = open("text1.txt","a")
f.write("_____")
f.close()
```
} add.

with open

```
with open("text1.txt","r") as f:
    myfile = f.read()
    print(myfile)
```

⊙ Rename and delete a file:-

import file :

```
import os
old_file_name = ("_____")
new_file_name = ("_____")
f = open("old_file_name", "__")
    :
    :
f.close()
```

Date / /
Page No.
Shivlal

OBJECT ORIENTED PROGRAMMING

- python is an object oriented programming language.
- **Class :-**
- A class is a blueprint or template for creating objects
- **Object :-**
- An object is a real world entity.

```
class fourm:
    def printdetails(self):
        print("your name is:", self.name)
        print("your rollno is:", self.Rollno)

object = fourm()
object.name = str(input("enter you name"))
object.rollno = int(input("enter your rollno"))
object.printdetails
```

oops in python:-

- To map with real world scenarios, we started using objects in code.
  This is called object oriented programming.

class:-

- class is a blueprint for creating object.

```
class student:
    name = "Karan"

# object
s1 = student()
print(s1.name)
```

__init__ function:-

Constructor:-

All class have a function called __init__(), which is always executed when the class is being initiated.

creating class

```
class student:
    def __init__(self, fullname):
        self.fullname = name
```

⊙ The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class

⊙ **class & Instance attributes:-**

→ class attribute ≤ object attribute.

```
class student:
    school = "GLA"   # class attribute
    def __init__(self, school):
        self.school = school   # object attribute.
```

⊙ **Methods:-**

⊙ Methods are function that belongs to object

⊙ **static method:-**

⊙ Methods that doesn't use the self parameter (work at class level)

Ex:-
```
class student:
    def college():          # simple method
        print("ABC college")
```

Ex:-
```
class student:
    @staticmethod           # static method.
    def hello():
        print("hello")
```

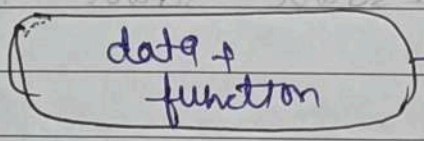
⊙ Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it.

⊙ Abstraction:-

⊙ Hiding the implementation details of a class and only showing the essential features to the user.

⊙ To hide unnecessary thing.

⊙ Encapsulation:-



→ wrapping data and function into a single unit (object).

⊙ del keyword:-

⊙ used to delete object properties or object itself.

```
# del S1.name
# del S1
```

⊙ private (like) attributes & methods:-

conceptual implementation in python:-

⊙ Private attributes & method are meant to be used only within the class and are not accessible for outside the class.

Ex
```
    self.name = name
    def hello(self):
```

Inheritance :-

- when one class (child / derived) derives the properties & method of another class (parent / base).

Ex-
```
class Car:
    ----
    ----

class ToyotaCar(Car):
    ----
    ----
```

Types of Inheritance :-

1. Single Inheritence.
2. Multi-level Inheritence.
3. Multiple Inheritence.

Super method :-

- Super() method is used to access methods of the parent class.

class method :-

- A class method is baind to the class & receives the class as an implicit first argument.

@ getter
@ setter

Date / /
Page No.
Shivalal

Note:- Static method can't access or modify class state & generally for utility.

```
class student:
    @classmethod    # decorator
    def college(cls)
        pass
```

→ Static method
→ class method (cls)
→ Instance method (self)

⊙ property :-

⊙ We use @ property decorator on any method in the class to use the method as a property.

⊙ Polymorphism:-

⊙ operator overloading

⊙ When the same operator is allowed to have different meaning according to the context.

⊙ operator & dunder function

• a+b	a.__add__(b)
• a-b	a.__sub__(b)
• a*b	a.__mul__(b)
• a/b	a.__truediv__(b)
• a%b	a.__mod__(b)