Date: 01/09/2025
MON TUE WED THU FRI SAT SUN

Module - 2

Object - oriented Programming with python

Q1 Define class and Object in python with suitable examples. Explain how objects are created and used.

⇒

① Class:

→ A class is a blue print or tempelate for creating objects.

→ It defines variables (called attributes) and functions (called methods) that describe the behavior of the object.

→ Example: A "car" class can define properties like color, model, speed, and methods like start() or stop().

code:

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

    def show(self):
        print(f"car model: {self.model}
              , Color: {self.color}")
```

Date : __/__/____
MON TUE WED THU FRI SAT SUN

(2) Object :

→ An object is an instance of a class.

→ When we create an object, memory is allocated and we can access the attributes and methods.

code :

```
# Creating objects

car1 = Car ("Tesla", "Red")
car 2 = Car ("BMW", "Black")

# using objects.

car 1. show ()
car 2. show ()
```

Output :

```
Car Model : Tesla , Color : Red.
Car Model : BMW , color : Black.
```

In short : Class = definition, Object = real thing created from that definition.

Date : __/__/____
MON TUE WED THU FRI SAT SUN

Q2. Explain the purpose of the __init__ method in python. Write a program to create a student class that stores name and marks using __init__.

⇒

① Purpose of __init__ :

- It is a constructor in python.
- It is automatically called when an object is created.
- It initializes the attributes (like name, age, marks) of the objects.

② Program Example :

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def display(self):
        print(f"Name: {self.name}
              Marks: {self.marks}")

# Creating Objects
```

e __init__
a program
class that
using

python.

when an

utes (like
he objects.

, marks):

.name,
arks)")

Date : __/__/____
MON TUE WED THU FRI SAT SUN

```
S1 = Student("Pranay", 95)
S2 = Student("Sayan", 88)

S1.display()
S2.display()
```

③ Output :

```
Name : Pranay , Marks : 95
Name : Sayan , Marks : 88
```

The __init__ method ensures that each object starts with a proper initial values.

Q3. Differentiate between a normal method and the __init__ method with an example.

⇒

① Normal Method :
- Defined inside a class.
- Called explicitly using object name.
- performs specific tasks (like displaying info, doing calculation).

② __init__ Method :
- Special built-in method (constructor).
- Called automatically when object is created.
- used to initialize attributes of the object.

Date : __/__/____
MON TUE WED THU FRI SAT SUN

③ Example :

```
class Student:
    def __init__(self, name):
        self.name = name      #auto called

    def greet(self):          # normal method
        print(f"Hello, {self.name}")

s1 = Student("Pranay")  # __init__ runs
                        #automatically

s1.greet()  # normal method must be
            #called
```

④ Output :

```
Hello, Pranay
```

Difference: __init__ is automatic, normal methods are manual.

Q4 What is the __call__ method in python? Explain with an example where an object can be used like a function.

1. The __call__ method allows an object to be called like a function.

Date : __/__/____
MON TUE WED THU FRI SAT SUN

auto called

rmal method
me?")

-- runs
tically

must be

tic, normal
el.

ed in
example
e used

an object
function.

2. Normally, we call functions with (). But with __call__, we can do the same with objects.

3. Example:

```
class Adder:
    def __call__(self, x, y):
        return x + y

add = Adder()
print(add(10,20))  # object behaves like
                   # a function
```

4. Output : 30

__call__ gives flexibility to use objects like functions.

Q5. Write a program (Python) using __call__ method to greet users by their names. Show the output for two different names.

⇒ Code:

```
class Greeter:
    def __call__(self, name):
        return f"Hello, {name}! Welcome!"
greet = Greeter()
print(greet("Pranay"))
print(greet("Sayam"))
```

Date : __/__/____
MON TUE WED THU FRI SAT SUN

output :-

Hello, Pranay! Welcome!
Hello, Rayan! Welcome!

Here, the object greet is used like a function.

06) Define the __iter__ method in python. why is it important for loops? Write a program to create an iterator for even numbers upto 10.

→

① Definition →
→ __iter__ makes a class iterable.
→ It allows objects to be used in loops (for, while).
→ works with __next__ to return next value.

② Importance →
- without __iter__, objects cannot be directly used in for loops.
- Iterators simplify sequence traversal.

Date : __/__/____
TUE WED THU FRI SAT SUN

Date : __/__/____
MON TUE WED THU FRI SAT SUN

ed like

d in
ant for
to create
mbers upto

able
used in

eturn next

ot be directly

traversal.

(3) Program :-

```
class EvenNumbers :
    def __init__ (self, limit) :
        self.limit = limit
        self.num = 0

    def __iter__ (self) :
        return self

    def __next__ (self) :
        self.num += 2
        if self.num <= self.limit :
            return self.num
        else :
            raise StopIteration

evens = EvenNumbers (10)
for n in evens :
    print (n)
```

Output :
2
4
6
8
10

→ __iter__ helps create custom loops.

Date : __/__/____
MON TUE WED THU FRI SAT SUN

Q7. What is the purpose of the __getitem__ method? Write a program where you can fetch a student's marks by index using __getitem__

⇒ (1) Purpose:

- __getitem__ allows objects to be accessed using indexing (obj[index]).
- like lists and tuples

(2) Program:

```
class Student:
    def __init__(self, marks):
        self.marks = marks

    def __getitem__(self, index):
        return self.marks[index]

s = Student([95, 88, 76])

print(s[0])  # first mark
print(s[2])  # Third mark
```

(3) output : 95
76

__getitem__ makes objects behave like lists.

Date : __/__/____
MON TUE WED THU FRI SAT SUN

the _getitem_
m where you
e marks by

to be
(obj[index]).

ks) :
ks

index):
s[index]

k
ak

behave like

Date : __/__/____
MON TUE WED THU FRI SAT SUN

Q8. Explain the use of the _len_ method in python. Write a program where a classroom object returns the number of students when len() is used.

① Use of _len_ :

- It allows the object to work on built-in len() function.
- Returns the size or number of elements.

② Program :-

```
class classroom:
    def __init__(self, students):
        self.students = students

    def __len__(self):
        return len(self.students)

c = classroom(["Pranay", "Sayan", "Vinay"])
print(len(c))
```

③ output : 3

→ _len_ makes custom objects work with len().

Date : __/__/____
MON TUE WED THU FRI SAT SUN

Q9 Compare and contrast the working of __len__ and __iter__. Give an eg. where both are implemented in the same class.

⇒ ① Comparision:

- __len__ : Returns the total count of elements.
- __iter__ : Allows looping through each element one by one.

② Contrast: • __len__ ⇒ quantity.
• __iter__ ⇒ iteration

③ Example:

```
class Group:
    def __init__(self, members):
        self.members = members

    def __len__(self):
        return len(self.members)

    def __iter__(self):
        return iter(self.members)

g = Group(["Pranay", "Sayan", "Vinay"])
print(len(g))          # calls __len__
for m in g:
    print(m)
```

Date : __/__/____
MON TUE WED THU FRI SAT SUN

Output :- 3
Pranay
Sayan
Vinay.

→ Both methods make objects work more naturally with python.

Q10 Explain the four main principles of OOPs in python (Encapsulation, Abstraction, inheritance, polymorphism) with small examples.

⇒

① Encapsulation :

- Binding data and methods into a single unit (class).
- Protects data from direct modification
- Example :

```
class Bank:
    def __init__(self, balance):
        self.__balance = balance
                # private variable

    def get_balance(self):
        return self.__balance
```

Date : __/__/____
MON TUE WED THU FRI SAT SUN

(2) Abstraction:

- Hiding internal details, showing only necessary informations.
- Example :

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
```

(3) Inheritance:

- A class (child) can use properties and methods of another class (parent
- Example :

```
class Animal:
    def speak(self):
        print("Animal Speaks :")

class dog(Animal):
    def bark(self):
        print("Dog barks")
```

Date : __/__/____
MON TUE WED THU FRI SAT SUN

④ Polymorphism :

- Same function name but different behaviours.

- Example :

```
class cat :
    def sound (self) :
        return "Meow"

Class dog :
    def sound (self) :
        return "Bark"

for animal in (cat(), Dog()) :
    print (animal.sound())
```

output : Meow
Bark

These four principles make OOP powerful, reusable, and organised.