Q1. What is the relationship between classes and modules?

A module can have zero or one or multiple classes. A class can be implemented in one or more . py files (modules).

Q2. How do you make instances and classes?

Class megha:

pass

class Parrot:

# class attribute

species = "bird"

# instance attribute

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Q3. Where and how should be class attributes created?

class Parrot:

# class attribute

species = "bird"

Q4. Where and how are instance attributes created?

class Parrot:

# class attribute

species = "bird"

# instance attribute

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Q5. What does the term "self" in a Python class mean?

self represents the instance of the class. By using the “self” keyword we can access the attributes and methods of the class in python. It binds the attributes with the given arguments.

The reason you need to use self. is because Python does not use the @ syntax to refer to instance attributes. Python decided to do methods in a way that makes the instance to which the method belongs be passed automatically, but not received automatically: the first parameter of methods is the instance the method is called on.

Q6. How does a Python class handle operator overloading?

perator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because ‘+’ operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.

To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator, the magic method \_\_add\_\_ is automatically invoked in which the operation for + operator is defined.

# Python Program illustrate how

# to overload an binary + operator

class A:

def \_\_init\_\_(self, a):

self.a = a

# adding two objects

def \_\_add\_\_(self, o):

return self.a + o.a

ob1 = A(1)

ob2 = A(2)

ob3 = A("megha”)

ob4 = A("For")

print(ob1 + ob2)

print(ob3 + ob4)

3

meghaFor

+ \_\_add\_\_(self, other)

– \_\_sub\_\_(self, other)

\* \_\_mul\_\_(self, other)

/ \_\_truediv\_\_(self, other)

// \_\_floordiv\_\_(self, other)

% \_\_mod\_\_(self, other)

\*\* \_\_pow\_\_(self, other)

>> \_\_rshift\_\_(self, other)

<< \_\_lshift\_\_(self, other)

& \_\_and\_\_(self, other)

| \_\_or\_\_(self, other)

^ \_\_xor\_\_(self, other)

Binary operators

Q7. When do you consider allowing operator overloading of your classes?

Consider that we have two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary ‘+’ operator it throws an error, because compiler don’t know how to add two objects. So we define a method for an operator and that process is called operator overloading. We can overload all existing operators but we can’t create a new operator. To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator, the magic method \_\_add\_\_ is automatically invoked in which the operation for + operator is defined.

Q8. What is the most popular form of operator overloading?

Python operators work for built-in classes. But the same operator behaves differently with different types. For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.

This feature in Python that allows the same operator to have different meaning according to the context is called operator overloading.

Q9. What are the two most important concepts to grasp in order to comprehend Python OOP code?

Abstraction

Encpsulation