ATTRIBUTES AND METHODS

1. What are data members ?

Data members are attributes declared within a class. They are properties that further define a class.

There are two types of attributes: class attributes and instance attributes.

2. What is a class attribute ?

An attribute that is common across all instances of a class is called a class attribute.

Class attributes are accessed by using class name as the prefix.

Syntax:

```
class className:
  classAttribute = value
```

Example:

```
class Employee:
  # This attribute is common across all instances of this class
  numberOfEmployees = 0

      employeeOne = Employee()
      employeeTwo = Employee()

Employee.numberOfEmployees += 1
print(employeeOne.numberOfEmployees)
print(employeeTwo.numberOfEmployees)

# Output => 1, 1
# Since numberOfEmployees was created as a class attribute, we incremented it to 1 by making use of its class name. This value has been reflected across objects employeeOne and employeeTwo.
```

3. What is an instance attribute ?

An attribute that is specific to each instance of a class is called an instance attribute.

Instance attributes are accessed within an instance method by making use of the self object.

Syntax:

```
class className:
  def methodName(self):
    self.instanceAttribute = value
```

Example:

```
class Employee:
  def employeeDetails(self, name):
    # name is the instance attribute
    self.name = name
```

4. What is the self parameter ?

Every instance method accepts has a default parameter that is being accepted. By convention, this parameter is named self.

The self parameter is used to refer to the attributes of that instance of the class.

Example:

```
class Employee:
  def setName(self, name):
    self.name = name

  employee = Employee()
employee.setName('John')
# The above statement is inferred as Employee.setName(employee, 'John') where the object employee is taken as the self argument.
# In the init method when we say self.name, Python actually takes it as employee.name, which is being stored with the value John.
```

5. What are methods ?

A function within a class that performs a specific action is called a method belonging to that class.

Methods are of two types: static method and instance method

6. What is an instance method ?

A method which can access the instance attributes of a class by making use of self object is called an instance method

Syntax:

```
def methodName(self):
  # method body
```

Example:

```
class Employee:
# employeeDetails() is the instance method
  def employeeDetails(self, name):
    self.name = name
```

7. What is a static method ?

A method that does not have access to any of the instance attributes of a class is called a static method.

Static method uses a decorator @staticmethod to indicate this method will not be taking the default self parameter.

Syntax:

```
@staticmethod
# Observe that self is not being declared since this is a static method
def methodName():
  # method body
```

Example:

```
class Employee:
  numberOfEmployees = 0
  @StaticMethod
```

```
    def updateNumberOfEmployees():
      Employee.numberOfEmployees += 1

  employeeOne = Employee()
  employeeTwo = Employee()
  employeeOne.updateNumberOfEmployees()
  employeeTwo.updateNumberOfEmployees()
  print(Employee.numberOfEmployees)

  # We have used a static method that updates the class
attribute of the class Employee.
```

8. What are the ways to access the attributes and methods of a class ?

Attributes and methods of a class can be accessed by creating an object and accessing the attributes and objects using class name or object name depending upon the type of attribute followed by the dot operator (.) and the attribute name or method name.

Example:

```
class Employee:
  numberOfEmployees = 0
  def printNumberOfEmployees(self):
    print(Employee.numberOfEmployees)

employee = Employee()
# Modify class attribute using dot operator
Employee.numberOfEmployees += 1
# Call the instance method using dot operator
employee.printNumberOfEmployees()
```

9. What is an init method ?

An init method is the constructor of a class that can be used to initialize data members of that class.

It is the first method that is being called on creation of an

object.

Syntax:

```
def __init__(self):
  # Initialize the data members of the class
```

Example:

```
class Employee:
  def __init__(self):
    print("Welcome!")

employee = Employee()
# This would print Welcome! since __init__ method was called on object instantiation.
```