Session 5

Assignment 9

Classes and Objects.

Q. no - 1. The code below shows the definition of a Laptop Class.

```
class Laptop:
    def __init__(self, capacity):
        self.capacity = capacity
        self.color = 'green'
```

Question:

[MCQ] Which of the following options create an instance of Laptop correctly?

1) laptop = Laptop(2000, 'green')

2) laptop = Laptop()

3) laptop = Laptop(2000)

4) laptop = Laptop('green')

5) None of the above.

Ans - 3

Q. no - 2. The code below shows the definition of a Laptop Class.

```
class Laptop:
        def __init__(self, capacity):
            self.capacity = capacity
            self.color = 'green'
        def change_color(self, new_color):
            self.color = new_color
```

Question:

[MCQ] Which of the following options create an instance of Laptop and change the color of the laptop to white correctly?

1) laptop = Laptop()
   laptop.change_color('white')
2) laptop = Laptop(2000, 'green')
   laptop.change_color(self, 'white')
3) laptop = Laptop(2000)

laptop.change_color(self, 'white')

4) laptop = Laptop(2000)
laptop.change_color('white')

5) None of the above.

Ans - 4

Q. no - 3. To create an object, it is important to look at the 'constructor' method: __init__() to find out the arguments that are required. The example below shows you how to create a Laptop object. Create a Person object based on the class definition given in the textbox below.

Examples

```
>>> class Laptop:
        def __init__(self, weight, color):
            self.weight = weight
            self.color = color
>>> laptop = Laptop(1.2, "Silver") #Create a Laptop object
>>> laptop = Laptop()

Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
    laptop = Laptop()
TypeError: __init__() takes exactly 3 arguments (1 given)
```

Ans - class Person:
def __init__(self, weight, height):
self.weight = weight
self.height = height

# Create a Person object with weight = 60, height = 1.7
p = Person(60,1.7)

Q. no - 4. An instance method is a function which operates on an instance of a class.

Examples

Question:

[MCQ] Which of the following is the correct definition of an instance method, where 'bar' is an argument of the method?

1) def foo($self, bar):
2) def foo(bar):
3) def foo(self, bar):
4) def __foo__(bar):
5) None of the above.

Ans - 3

Q. no - 5. Construct a class named 'Point', which takes in the x and y coordinates as parameters. It should include the documentation string and the methods '__init__' and '__str__'.

Examples

```
>>> P1  = Point(2,4)
 >>> P1.__doc__
'A class implementation of a 2-dimensional point.'
>>> str(P1)
'(2, 4)'
>>> print P1
(2, 4)
```

Ans - class Point:

```
    '''A class implementation of a 2-dimensional point.'''
    def __init__(self, x,y):
            self.x=x
            self.y=y


    def __str__(self):
            return "(%s, %s)"%(self.x, self.y)
```

Q. no - 6. Class variables are accessible by all objects (instances) of a class, while object variables are only applicable to the object (instance) itself.

Examples

Question:[MCQ] Which of the following statement is not true?

```
class Point:
    x = 0
    y = 0
    def __init__(self, x, y):
            Point.x = x
            Point.y = y
            self.x = x
            self.y = y

p1 = Point(1,2)
p2 = Point(3,4)
p2.x = 2
```

1) Point.x equals to 3 and Point.y equals to 4.
2) Point.x equals to 1 and Point.y equals to 2.
3) p1.x equals to 1 and p1.y equals to 2.
4) p2.x equals to 2 and p2.y equals to 4.
5) None of the above.

Ans - 2

Q. no - 7. It is possible to change the behaviour of built-in opertators with special methtods. For example, the '+' operator can be implemented with the __add__ method. Define a Point class that supports operator overloading for the '+' and '-' operators.

Examples

```
>>> a = Point(1,3)
 >>> b = Point(7,2)
 >>> print a+b
 '(8, 5)'
 >>> print a-b
 '(-6, 1)'
```

Ans - class Point:

```
    "A class implementation of 2-Dimensional point."
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return '(%d, %d)' % (self.x, self.y)

    def __add__(self, other):
        return self.x+other.x, self.y+other.y

    def __sub__(self, other):
        return self.x-other.x, self.y-other.y
```

Q. no - 8. Inheritance allows the reuse of code, by implementing a parent-child relationship between classes. Create 2 derived classes Student and WorkingAdult from the base class Person.

Examples

```
>>> s = Student('Peter', 9, 'ABC Primary School')
 >>> s.introduce()
 'My name is Peter. I am 9 years old. I am studying at ABC Primary School.'
 >>> a = WorkingAdult('John', 23, 'Waiter')
```

```
>>> a.introduce()
'My name is John. I am 23 years old. I am a waiter.'
```

Ans -

```
class Person:
    """A base class"""
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Student(Person):
    """A derived class for Student"""
    def __init__(self, name, age, school):
        Person.__init__(self, name, age)
        self.school=school

    def introduce(self):
        return "My name is %s. I am %s years old. I am studying at %s." % (self.name,
self.age, self.school)


class WorkingAdult(Person):
    """A derived class for WorkingAdult"""
    def __init__(self, name, age, job):
        Person.__init__(self,name,age)
        self.job=job


    def introduce(self):
        return "My name is %s. I am %s years old. I am a %s." % (self.name, self.age,
self.job.lower())
```