Lab 8

September 27, 2023

[ ]:
```
#Python code after removing the syntax error
string = "Python Exceptions"

for s in string:
    if (s != o:
        print( s )
```

[ ]:
```
#Python code after removing the syntax error
string = "Python Exceptions"

for s in string:
    if (s != o):
        print( s )
```

[ ]:
```
# Python code to catch an exception and handle it using try and except code␣
 ↪blocks

a = ["Python", "Exceptions", "try and except"]
try:
    #looping through the elements of the array a, choosing a range that goes␣
 ↪beyond the length of the array
     for i in range( 4 ):
        print( "The index and element from the array is", i, a[i] )
#if an error occurs in the try block, then except block will be executed by the␣
 ↪Python interpreter
except:
    print ("Index out of range")
```

[ ]:
```
class car:
    def __init__(self,modelname, year):
        self.modelname = modelname
        self.year = year
    def display(self):
        print(self.modelname,self.year)

c1 = car("Toyota", 2016)
c1.display()
```

[ ]:
```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def greet(self):
        print("Hello, my name is " + self.name)

# Create a new instance of the Person class and assign it to the variable␣
 ↪person1
person1 = Person("Ayan", 25)
person1.greet()
```

[ ]:
```python
class Person:
    count = 0   # This is a class variable

    def __init__(self, name, age):
        self.name = name    # This is an instance variable
        self.age = age
        Person.count += 1   # Accessing the class variable using the name of␣
 ↪the class
person1 = Person("Ayan", 25)
person2 = Person("Bobby", 30)
print(Person.count)
```

[ ]:
```python
class Person:
    def __init__(self, name, age):
        self.name = name    # This is an instance variable
        self.age = age
person1 = Person("Ayan", 25)
person2 = Person("Bobby", 30)
print(person1.name)
print(person2.age)
```

[ ]:
```python
class Employee:
    def __init__(self, name, id):
        self.id = id
        self.name = name

    def display(self):
        print("ID: %d \nName: %s" % (self.id, self.name))


emp1 = Employee("John", 101)
emp2 = Employee("David", 102)

# accessing display() method to print employee 1 information
```

```
emp1.display()

# accessing display() method to print employee 2 information
emp2.display()
```

[ ]:
```
class Student:
    count = 0
    def __init__(self):
        Student.count = Student.count + 1
s1=Student()
s2=Student()
s3=Student()
print("The number of students:",Student.count)
```

[ ]:
```
class Student:
    # Constructor - non parameterized
    def __init__(self):
        print("This is non parametrized constructor")
    def show(self,name):
        print("Hello",name)
student = Student()
student.show("John")
```

[ ]:
```
class Student:
    # Constructor - parameterized
    def __init__(self, name):
        print("This is parametrized constructor")
        self.name = name
    def show(self):
        print("Hello",self.name)
student = Student("John")
student.show()
```

[ ]:
```
class Student:
    roll_num = 101
    name = "Joseph"

    def display(self):
        print(self.roll_num,self.name)

st = Student()
st.display()
```

[ ]:
```
class Student:
    def __init__(self):
        print("The First Constructor")
    def __init__(self):
```

```
        print("The second contructor")

st = Student()
```

[ ]:
```
class Student:
    def __init__(self, name, id, age):
        self.name = name
        self.id = id
        self.age = age

    # creates the object of the class Student
s = Student("John", 101, 22)

# prints the attribute name of the object s
print(getattr(s, 'name'))

# reset the value of attribute age to 23
setattr(s, "age", 23)

# prints the modified value of age
print(getattr(s, 'age'))

# prints true if the student contains the attribute with name id

print(hasattr(s, 'id'))
# deletes the attribute age
delattr(s, 'age')

# this will give an error since the attribute age has been deleted
print(s.age)
```

[ ]:
```
class Student:
    def __init__(self,name,id,age):
        self.name = name;
        self.id = id;
        self.age = age
    def display_details(self):
        print("Name:%s, ID:%d, age:%d"%(self.name,self.id))
s = Student("John",101,22)
print(s.__doc__)
print(s.__dict__)
print(s.__module__)
```

[ ]:
```
class Animal:
    def speak(self):
        print("Animal Speaking")
#child class Dog inherits the base class Animal
```

```
class Dog(Animal):
    def bark(self):
        print("dog barking")
d = Dog()
d.bark()
d.speak()
```

[ ]:
```
class Animal:
    def speak(self):
        print("Animal Speaking")
#The child class Dog inherits the base class Animal
class Dog(Animal):
    def bark(self):
        print("dog barking")
#The child class Dogchild inherits another child class Dog
class DogChild(Dog):
    def eat(self):
        print("Eating bread...")
d = DogChild()
d.bark()
d.speak()
d.eat()
```

[ ]:
```
class Calculation1:
    def Summation(self,a,b):
        return a+b;
class Calculation2:
    def Multiplication(self,a,b):
        return a*b;
class Derived(Calculation1,Calculation2):
    def Divide(self,a,b):
        return a/b;
d = Derived()
print(d.Summation(10,20))
print(d.Multiplication(10,20))
print(d.Divide(10,20))
```

[ ]:
```
class Calculation1:
    def Summation(self,a,b):
        return a+b;
class Calculation2:
    def Multiplication(self,a,b):
        return a*b;
class Derived(Calculation1,Calculation2):
    def Divide(self,a,b):
        return a/b;
d = Derived()
```

```
print(issubclass(Derived,Calculation2))
print(issubclass(Calculation1,Calculation2))
```

[ ]:
```
class Calculation1:
    def Summation(self,a,b):
        return a+b;
class Calculation2:
    def Multiplication(self,a,b):
        return a*b;
class Derived(Calculation1,Calculation2):
    def Divide(self,a,b):
        return a/b;
d = Derived()
print(isinstance(d,Derived))
```

[ ]:
```
class Animal:
    def speak(self):
        print("speaking")
class Dog(Animal):
    def speak(self):
        print("Barking")
d = Dog()
d.speak()
```

[ ]:
```
class Bank:
    def getroi(self):
        return 10;
class SBI(Bank):
    def getroi(self):
        return 7;

class ICICI(Bank):
    def getroi(self):
        return 8;
b1 = Bank()
b2 = SBI()
b3 = ICICI()
print("Bank Rate of interest:",b1.getroi());
print("SBI Rate of interest:",b2.getroi());
print("ICICI Rate of interest:",b3.getroi());
```

[ ]:
```
class Employee:
    __count = 0;
    def __init__(self):
        Employee.__count = Employee.__count+1
    def display(self):
        print("The number of employees",Employee.__count)
```

```
emp = Employee()
emp2 = Employee()
try:
    print(emp.__count)
finally:
    emp.display()
```

[ ]:
```
# Python program demonstrate
# abstract base class work
from abc import ABC, abstractmethod
class Car(ABC):
    def mileage(self):
        pass

class Tesla(Car):
    def mileage(self):
        print("The mileage is 30kmph")
class Suzuki(Car):
    def mileage(self):
        print("The mileage is 25kmph ")
class Duster(Car):
     def mileage(self):
          print("The mileage is 24kmph ")

class Renault(Car):
    def mileage(self):
            print("The mileage is 27kmph ")

# Driver code
t= Tesla ()
t.mileage()

r = Renault()
r.mileage()

s = Suzuki()
s.mileage()
d = Duster()
d.mileage()
```

[ ]:
```
# Python program to define
# abstract class

from abc import ABC

class Polygon(ABC):
```

```
    # abstract method
    def sides(self):
        pass

class Triangle(Polygon):


    def sides(self):
        print("Triangle has 3 sides")

class Pentagon(Polygon):


    def sides(self):
        print("Pentagon has 5 sides")

class Hexagon(Polygon):

    def sides(self):
        print("Hexagon has 6 sides")

class square(Polygon):

    def sides(self):
        print("I have 4 sides")

# Driver code
t = Triangle()
t.sides()

s = square()
s.sides()

p = Pentagon()
p.sides()

k = Hexagon()
K.sides()
```

[ ]: