Assignment -4

Q1.

```
 from abc import ABC, abstractmethod
class Animal(ABC):
        @abstractmethod
        def eat1(self):
           pass
        @abstractmethod
        def eat2(self):
          pass
class Tiger(Animal):
        def eat1(self):
                print("Tiger implementation ...")
class lion(Tiger):
        def eat2(self):
                print("lion implementation ...")
t = lion()
t.eat1()
t.eat2()
```

Output:

```
Tiger implementation ...
lion implementation ...
```

Q2.

```
from abc import ABC, abstractmethod
class AbstractClassExample(ABC):
```

```
    def __init__(self, value):
        self.value = value
        super().__init__()

    @abstractmethod
    def do_something(self):
        pass

class DoAdd(AbstractClassExample):
    def do_something(self):
        return self.value + 42

class DoMul(AbstractClassExample):

    def do_something(self):
        return self.value * 42

x = DoAdd(10)
y = DoMul(10)
print(x.do_something())
print(y.do_something())
```

Output:

52

420

Q3.

```
def status(age):
        if age < 0:
                raise ValueError("Only positive integers are allowed")
        if age>22:
                print("eligible for mrg")
        else:
                print("not eligible for mrg try after some time")
try:
        num = int(input("Enter your age: "))
        status(num)
except ValueError:
        print("Only positive integers are allowed you ......")
finally:
        print("finally block....")
```

Output:

```
Enter your age: 3
not eligible for mrg try after some time
finally block....
```

Q4.

```
class NegativeAgeException(RuntimeError):
        def __init__(self, age):
                super().__init__()
                self.age = age
def status(age):
                if age < 0:
```

```
                raise NegativeAgeException("Only positive integers are
allowed")
        if age>22:
                print("Eligible for mrg")
        else:
                print("not Eligible for mrg....")
try:
    num = int(input("Enter your age: "))
    status(num)
except NegativeAgeException:
    print("Only positive integers are allowed")
except:
    print("something is wrong”)
```

Output:

```
Enter your age: 34
Eligible for mrg
```

Q5.

```
class TooYoungException(Exception):
 def __init__(self,age):
                self.age=age
class TooOldException(Exception):
 def __init__(self,age):
            self.age=age
try:
 age=int(input("Enter Age:"))
 if age<18:
```

```
        raise YoungException("Plz wait some time ")
 elif age>65:
        raise TooOldException("Your age too old")
 else:
        print("we will find one girl soon”)
except YoungException as e:
        print("Plz wait some time ")
except OldException as e:
        print("Your age too old ")
```

Output:

```
Enter Age:34
we will find one girl soon
```

Q6.

```
try:
 print("outer try block")
 n = int(input("enter a number"))
 print(10/n)
 try:
  print("inner try")
  print("anu"+"preet")
 except TypeError:
  print("Hello")
 else:
  print("inner no exception")
except ArithmeticError:
```

```
 print(10/5)
else:
 print("outer no excepiton")
finally:
 print("finally block")
```

Output:

```
outer try block
enter a number3
3.3333333333333335
inner try
anupreet
inner no exception
outer no excepiton
finally block
```

Q7.

```
class Person(object):
        def __init__(self, first, last):
                self.firstname = first
                self.lastname = last
        def Name(self):
                return self.firstname + " " + self.lastname
class Employee(Person):
        def __init__(self, first, last, staffnum):
                super().__init__(first,last)
                #Person.__init__(self,first, last)
```

```
        self.staffnumber = staffnum
    def GetEmployee(self):
        return self.Name() + ", " + self.staffnumber
x = Person("komal", "addanki")
y = Employee("komal", "addanki", "111")
print(x.Name())
print(y.GetEmployee())
```

Output:

```
komal addanki
komal addanki, 111
```

Q8.

```
class Person:
    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last
    def __str__(self):
        return self.firstname + " " + self.lastname
class Employee(Person):
 def __init__(self, first, last, id):
        super().__init__(first, last)
        self.id = id
 def __str__(self):
        return super().__str__()+" "+self.id
x = Person("kamalpreet", "gurpreet")
y = Employee("amalpreet", "gurpreet", "111")
```

```
print(x)
print(y)
```

Output:

kamalpreet gurpreet

amalpreet gurpreet 111

Q9.

```
class Vehicle:
    def __del__(self):
        print("Vehicle object destroyed")
        print(10/0)
v = Vehicle()
del v
```

Output:

Vehicle object destroyed

ZeroDivisionError: division by zero

Q10.

```
class Customer:
    def __init__(self,name,bal=0.0):
        self.name=name
        self.bal=bal

    def deposit(self,amount):
        self.bal=self.bal+amount
    def withdraw(self,amount):
```

```
            if amount>self.bal:
                raise RuntimeError("withdraw amount is more than
balance")
            else:
                self.bal=self.bal-amount

    def remaining(self):
            return self.bal;

c = Customer("Komal",10000)
damt = int(input("enter amount to deposit"))
c.deposit(damt)
amt = int(input("enter amount to withdraw"))
c.withdraw(amt)
print(c.remaining())
```

Output:

```
enter amount to deposit5000
enter amount to withdraw2000
13000
```