Q1 of 7outlined\_flag

What is the output of the following code snippet?

class Parent:  
    def \_\_init\_\_(self,num):  
      self.\_\_num=num  
    def get\_num(self):  
      return self.\_\_num  
class Child(Parent):  
    def \_\_init\_\_(self,num,val):  
      super().\_\_init\_\_(num)  
      self.\_\_val=val  
    def get\_val(self):  
      return self.\_\_val  
son=Child(100,200)  
print(son.get\_num())  
print(son.get\_val())

100 200

200 100

None 200

Error: arguments cannot be passed through super() to the parent class constructor

**Answer - 100 200**

//-----------------------------------------------------------------------

Q2 of 7outlined\_flag

What is the output of the following code snippet?

class Parent:  
    def \_\_init\_\_(self):  
        self.num=100  
  
class Child(Parent):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.var=200  
    def show(self):  
        print(self.num)  
        print(self.var)  
  
son=Child()  
son.show()

None 200

Error: A parent class instance variable cannot be accessed in a child class method

100 200

Error: super() can invoke only parameterized constructor of a parent class

**Answer - 100 200**

//-----------------------------------------------------------------------

Q3 of 7outlined\_flag

Consider the following python function for representing the customers of a retail store.5 min  
Objective of the code is to record the details of the customers.

def customer\_record(customer\_type, name, discount, points\_earned, membership\_card\_type):  
    if(customer\_type=="Regular"):  
        record="Record Regular Customer:"+name+" "+(str)(discount)  
    elif(customer\_type=="Privileged"):  
        record="Record Privileged Customer:"+name+" "+(str)(points\_earned)  
    elif(customer\_type=="Elite"):  
        record="Record Elite Customer:"+name+" "+membership\_card\_type  
    print(record)

What will be the optimal class structure if this has to be re-written in Object oriented programming?

3 independent classes: Regular, Privileged, Elite

1 class: Customer

4 classes with inheritance: Base class: Customer; Child classes: Regular, Privileged; Grand Child of Privileged: Elite

4 classes with inheritance: Base class: Customer; Child classes: Regular, Privileged, Elite

4 classes with inheritance: Base class: Customer; Child classes: Regular, Privileged; Grand Child of Regular: Elite

**Answer - C(4 classes with inheritance: Base class: Customer; Child classes: Regular, Privileged, Elite )**

//-----------------------------------------------------------------------

Q4 of 7outlined\_flag

What is the output of the following code snippet?

class Parent:  
    def \_\_init\_\_(self):  
        self.\_\_num=100  
  
    def show(self):  
        print("Parent:",self.\_\_num)  
  
class Child(Parent):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.\_\_var=10  
  
    def show(self):  
        print("Child:",self.\_\_var)  
  
dad=Parent()  
dad.show()  
son=Child()  
son.show()

a) Child: 10  
    Child: 10

b) Parent: 100  
    Parent: 100

c) Error: Methods in parent and child classes cannot be same

d) Parent: 100  
    Child: 10

a

b

c

d

**Answer - D**

//-----------------------------------------------------------------------

Q5 of 7outlined\_flag

What should be written in line 14 to get the output as mentioned below?4 min  
Parent: 100  
Child: 10

class Parent:  
    def \_\_init\_\_(self):  
        self.\_\_num=100  
  
    def show(self):  
        print("Parent:",self.\_\_num)  
  
class Child(Parent):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.\_\_var=10  
  
    def show(self):  
        \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
        print("Child:",self.\_\_var)  
  
obj=Child()  
obj.show()

show()

super().show()

self.show()

Parent.show()

**Answer - B(super().show())**

//-----------------------------------------------------------------------

Q6 of 7outlined\_flag

What is the output of the below code?

class Customer:  
    def \_\_init\_\_(self):  
        self.\_\_cust\_id = None  
        self.\_\_cust\_name = None  
  
    def get\_cust\_id(self):  
        print("Normal customer")  
  
    def get\_cust\_name(self):  
        return self.\_\_cust\_name  
  
    def set\_cust\_id(self, value):  
        self.\_\_cust\_id = value  
  
    def set\_cust\_name(self, value):  
        self.\_\_cust\_name = value  
  
class PrivilegedCustomer(Customer):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.\_\_overdraft\_limit = None  
  
    def get\_overdraft\_limit(self):  
        return self.\_\_overdraft\_limit  
  
    def set\_overdraft\_limit(self, value):  
        self.\_\_overdraft\_limit = value  
  
    def get\_cust\_id(self):  
        print("Privileged Customer")  
  
class RegularCustomer(Customer):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.\_\_min\_balance = None  
  
    def get\_min\_balance(self):  
        return self.\_\_min\_balance  
  
    def set\_min\_balance(self, value):  
        self.\_\_min\_balance = value  
  
r=RegularCustomer()  
p=PrivilegedCustomer()  
r.get\_cust\_id()  
p.get\_cust\_id()

a) Normal Customer   
    Privileged Customer

b) Privileged Customer   
    Normal Customer

c) Error

d) None of the above

a

b

c

d

**Answer - A**

//-----------------------------------------------------------------------

Q7 of 7outlined\_flag

For the given Customer class, what should be the subclasses so that the output of the code is:  
        Regular Customer  
        Normal Customer  
        Privileged Customer

class Customer:  
    def \_\_init\_\_(self):  
        self.\_\_cust\_id = None  
        self.\_\_cust\_name = None  
  
    def get\_cust\_id(self):  
        print("Normal customer")  
  
    def get\_cust\_name(self):  
        return self.\_\_cust\_name  
  
    def set\_cust\_id(self, value):  
        self.\_\_cust\_id = value  
  
    def set\_cust\_name(self, value):  
        self.\_\_cust\_name = value

**Option A**

class PrivilegedCustomer(Customer):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.\_\_overdraft\_limit = None  
  
    def get\_overdraft\_limit(self):  
        return self.\_\_overdraft\_limit  
  
    def set\_overdraft\_limit(self, value):  
        self.\_\_overdraft\_limit = value  
  
    def get\_cust\_id(self):  
        print("Privileged Customer")  
  
class RegularCustomer(PrivilegedCustomer):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.\_\_min\_balance = None  
  
    def get\_min\_balance(self):  
        return self.\_\_min\_balance  
  
    def set\_min\_balance(self, value):  
        self.\_\_min\_balance = value  
  
    def get\_cust\_id(self):  
        print("Regular Customer")  
  
r=RegularCustomer()  
p=PrivilegedCustomer()  
n=Customer()  
r.get\_cust\_id()  
p.get\_cust\_id()  
n.get\_cust\_id()

**Option B**

class PrivilegedCustomer(Customer):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.\_\_overdraft\_limit = None  
  
    def get\_overdraft\_limit(self):  
        return self.\_\_overdraft\_limit  
  
    def set\_overdraft\_limit(self, value):  
        self.\_\_overdraft\_limit = value  
  
    def get\_cust\_id(self):  
        print("Privileged Customer")  
  
class RegularCustomer(Customer):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.\_\_min\_balance = None  
  
    def get\_min\_balance(self):  
        return self.\_\_min\_balance  
  
    def set\_min\_balance(self, value):  
        self.\_\_min\_balance = value  
  
    def get\_cust\_id(self):  
        print("Regular Customer")  
  
r=RegularCustomer()  
p=PrivilegedCustomer()  
n=Customer()  
r.get\_cust\_id()  
n.get\_cust\_id()  
p.get\_cust\_id()

A

B

**Answer - B**

//-----------------------------------------------------------------------

//-----------------------------------------------------------------

**Defined& Undefined Methods -**

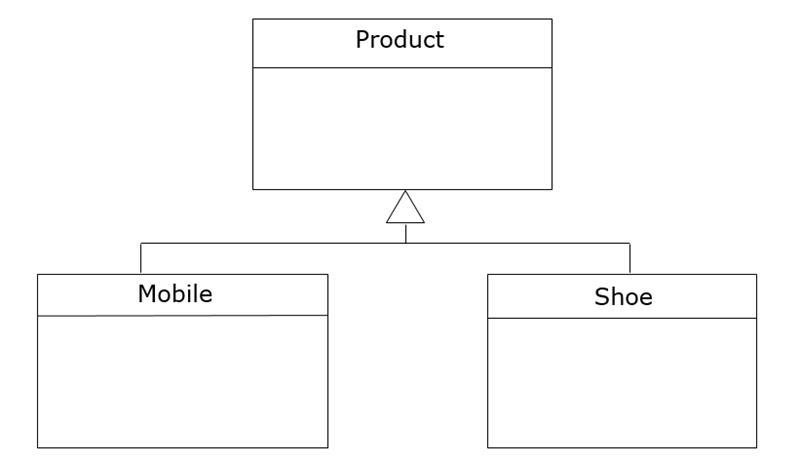
**Abstract Class -**

**Abstract Method-**

**//-----------------------------**

Abstract Class - Introduction

Let us assume that we have a Product class , all items being sold in our online app and extend this Product class.



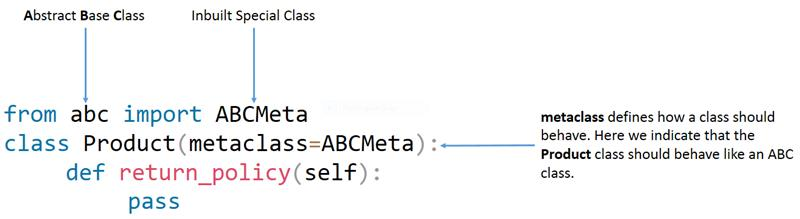
In an online shopping app, we only have specific types of products. We don’t have something called a Product itself. In that sense, an object of Product class would not represent a real world object, because a Product is just an abstract concept. While shopping, we purchase types of products, not a product itself. Thus the best practice is not to create object of the parent class

Preventing instantiation

Since we are the creator of the Product class, we know it is abstract in nature and we won’t create an instance for it. But how can another programmer know about it? How can we ensure that some other developer does not end up creating an object of such an abstract class?

We can programmatically declare a class as an abstract class. An abstract class should not be instantiated.

**Note:** In python, you will not get an error if you try to instantiate it. However, in languages like Java, C++, C# you will get an error if you try to instantiate an abstract class



What is the use of abstract?

If an abstract class should never be instantiated, then what is the use of such a class? The only way we can use an abstract class is to make other classes inherit from the abstract class. An abstract class is meant to be sub classed.

Abstract Method - A Scenario

Let us extend the concept of abstract further. All products must have a return\_policy() method which will display the number of days within which the products have to be returned

1. class Product:
2. def return\_policy(self):
3. pass
4. class Mobile(Product):
5. pass
6. class Shoe(Product):
7. pass

Problem Statement

Now each type of product will have its own return\_policy(), so we will override the return\_policy() in each of the child classes.

class Mobile(Product):

   def return\_policy(self):

       print("All mobiles must be returned within 10 days of purchase")

class Shoe(Product):

   def return\_policy(self):

       print("All shoes must be returned within 7 days of purchase")

from abc import ABCMeta,abstractmethod

class Shape(metaclass=ABCMeta):

    @abstractmethod

    def aera(self):

        pass

class Rectangle(Shape):

    def \_\_init\_\_(self,length,breadth):

        self.length=length

        self.breadth=breadth

    def aera(self):

        return self.length\*self.breadth

class Square(Shape):

    def \_\_init\_\_(self,side):

        self.side=side

    def aera(self):

        return self.side\*self.side

s1=Rectangle(10,20)

res=s1.aera();

print('Rectangle Aera= ',res)

s2=Square(10)

res=s2.aera()

print('Sqaure Aera= ',res)

**//---------------------------------**

Abstract method:  
====================  
|-A method without body is called as abstract method

|-To mark our undefined method as abstract method we have needs to annotate our undefined method by @abstractmethod annottation

@abstractmethod  
def payBiil(self,amount):  
     pass

Abstract class  
===============  
|-The class which contains atleast one abstract method that class is called as a   Abstract class.

|-To mark our class as abstrac class we needs to mark ABCMeta

from abc import ABCMeta  
class Product(metaclass=ABCMeta):  
       @abstractmethod  
  def payBiil(self,amount):  
         pass

Summary:  
=======  
-Abstract classes should not be instantiated.  
-An abstract class may contain 0 or many abstract methods.  
-Usually the parent class is an abstract class.  
-Abstract classes are meant to be inherited.  
-If a class has an abstract method, then the class cannot be instantiated.  
-The child class must implement/override all the abstract methods of the parent   class. Else the child class cannot be instantiated.

Q ) **Abstract Class Example -**

from abc import ABCMeta, abstractmethod

class Shape(metaclass=ABCMeta):

@abstractmethod

def area(self):

pass

class Rectangle(Shape):

def \_\_init\_\_(self, length, breadth):

self.length = length

self.breadth = breadth

def area(self):

return self.length \* self.breadth

class Square(Shape):

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

self.side = side

def area(self):

return self.side \* self.side

s1 = Rectangle(10, 20)

res = s1.area()

print('Rectangle Area= ', res)

s2 = Square(10)

res = s2.area()

print('Square Area= ', res)

**Output - Rectangle Area= 200**

**Square Area= 100**

**//-----------------------------------------------------------------------------------------**

Q1 of 6outlined\_flag

What is the output of the following code snippet?

from abc import ABCMeta, abstractmethod  
class Parent(metaclass=ABCMeta):  
    def \_\_init\_\_(self):  
        self.num=100  
  
    @abstractmethod  
    def show(self):  
        pass  
  
class Child(Parent):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.\_\_var=10  
  
    def show(self):  
        print(self.num)  
        print(self.\_\_var)  
  
obj=Parent()  
obj.show()

a) 100  
    10

b) 10  
    100

c) Error: abstract method should always have a valid statement other than pass

d) Error: abstract class cannot be instantiated

a

b

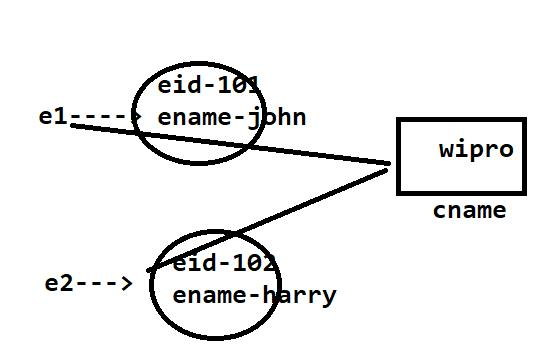
c

d

**Answer - D**

**//------------------------------------------------------------------------------  
//--------------------------------------------------------------------------------**

**Static Variables -**



class Emp:

    cname='wipro'

    def \_\_init\_\_(self,eid,ename):

        self.eid=eid

        self.ename=ename

e1=Emp(101,'john')

e2=Emp(102,'harry')

print(e1.eid)

print(e2.eid)

print(Emp.cname)

 //----------------------------------------------------------------

**Example static variable -**

ABC DTH (Direct to Home) firm wants to calculate monthly rent for its consumers.   
A consumer can register for one Base Package. Write a python program to implement the below given class diagram.



**Class Description:**  
**DirectToHomeService class:**

1. Initialize static variable counter to 101
2. Inside constructor, auto-generate consumer\_number starting from 101

**BasePackage class:**

1. **validate\_base\_pack\_name():**
   1. Validate base pack name. Valid values are "Silver", "Gold" and "Platinum".
   2. If invalid, set attribute, base\_pack\_name as "Silver" and display "Base package name is incorrect, set to Silver"
2. **calculate\_monthly\_rent():**
   1. Check if subscription period is between 1 and 24 (both inclusive). If so,
      1. Validate base pack name
      2. Identify monthly rent based on base pack. Refer table given.
      3. Consumers are eligible for discount of one month's rent, if subscription period is more than 12 months
      4. Calculate final monthly rent as per the formula given below:
      5. final monthly rent = ((monthly rent \* subscription period) – discount amount)/subscription period
      6. Return the calculated final monthly rent
   2. If not, return -1

|  |  |
| --- | --- |
| **Base Pack Name** | **Monthly Rent** |
| Silver | 350.00 |
| Gold | 440.00 |
| Platinum | 560.00 |

**Note:**Perform case sensitive string comparison

**For testing:**

* Create objects of BasePackage class
* Invoke **calculate\_monthly\_rent()** on BasePackage object
* Display the details

**Code -**

from abc import ABCMeta, abstractmethod

class DirectToHomeService:

counter = 101

def \_\_init\_\_(self, consumer\_name):

self.consumer\_name = consumer\_name

DirectToHomeService.counter += 1

self.consumer\_number = DirectToHomeService.counter

def get\_consumer\_number(self):

return self.consumer\_number

def get\_consumer\_name(self):

return self.consumer\_name

@abstractmethod

def calculate\_monthly\_rent(self):

pass

class BasePackage(DirectToHomeService):

def \_\_init\_\_(self, consumer\_name, base\_pack\_name, subscription\_period):

super().\_\_init\_\_(consumer\_name)

self.base\_pack\_name = base\_pack\_name

self.subscription\_period = subscription\_period

def get\_base\_pack\_name(self):

return self.base\_pack\_name

def get\_subscription\_period(self):

return self.subscription\_period

def validate\_base\_pack\_name(self):

valid\_packs = ["Silver", "Gold", "Platinum"]

if self.base\_pack\_name not in valid\_packs:

self.base\_pack\_name = "Silver"

print("Base package name is incorrect, set to Silver")

def calculate\_monthly\_rent(self):

if 1 <= self.subscription\_period <= 24:

self.validate\_base\_pack\_name()

monthly\_rent = {"Silver": 350.00, "Gold": 440.00, "Platinum": 560.00}

discount\_amount = 0

if self.subscription\_period > 12:

discount\_amount = monthly\_rent[self.base\_pack\_name]

final\_rent = ((monthly\_rent[self.base\_pack\_name] \* self.subscription\_period) - discount\_amount) / self.subscription\_period

return final\_rent

else:

return -1

# Testing the implementation

consumer1 = BasePackage("John", "Silver", 6)

print("Consumer Number:", consumer1.get\_consumer\_number())

print("Consumer Name:", consumer1.get\_consumer\_name())

print("Base Package:", consumer1.get\_base\_pack\_name())

print("Subscription Period:", consumer1.get\_subscription\_period())

print("Monthly Rent:", consumer1.calculate\_monthly\_rent())

consumer2 = BasePackage("Alice", "Gold", 18)

print("\nConsumer Number:", consumer2.get\_consumer\_number())

print("Consumer Name:", consumer2.get\_consumer\_name())

print("Base Package:", consumer2.get\_base\_pack\_name())

print("Subscription Period:", consumer2.get\_subscription\_period())

print("Monthly Rent:", consumer2.calculate\_monthly\_rent())

//-------------------------------------------------------------------------------------

//------------------------------------------------------------------------------------------

**Exception Handling -**

In python uses **“try” and “except”** keyword to handle the exception.

Link Notes - [6. Built-in Exceptions — Python 2.7.18 documentation](https://docs.python.org/2/library/exceptions.html" \l "exceptions.Exception" \o "https://docs.python.org/2/library/exceptions.html#exceptions.exception" \t "_blank)

//------------------------------------------------------

Exception Handling - The Scenario

Problem Statement

Go through the below code, execute and observe the result, which calculates the airport expenditure.

#calculating airport expenditure

def calculate\_expenditure(list\_of\_expenditure):

  total=0

  for expenditure in list\_of\_expenditure:

       total+=expenditure

  print(total)

list\_of\_values=[100,200,300,"400",500]

calculate\_expenditure(list\_of\_values)

Problem Statement

* We got an error in the below code, one way to take care of such error situation is to use selection constructs.
* The error was due to addition of a string (“400”) to an integer. If we add a condition to check whether the expenditure is of type int, that would solve this error. But that can cause further issues.

**#calculating airport expenditure**

def calculate\_expenditure(list\_of\_expenditure):

  total=0

  for expenditure in list\_of\_expenditure:

       total+=expenditure

  print(total)

list\_of\_values=[100,200,300,"400",500]

calculate\_expenditure(list\_of\_values)

**Try out - Exception Handling using 'try' & 'except'**

Problem Statement

* In python we can create a try and except block of code.
* If any error occurs in the try block of code, it will jump to except block of code.
* Once the except block is executed, the code continues to execute other statements outside except block.

Code in Python 3

#calculating airport expenditure

def calculate\_expenditure(list\_of\_expenditure):

   total=0

   try:

       for expenditure in list\_of\_expenditure:

          total+=expenditure

       print(total)

   except:

      print("Some error occured")

  print("Returning back from function.")

list\_of\_values=[100,200,300,"400",500]

calculate\_expenditure(list\_of\_values)

**Exception Handling - Different Types**

Python has many kinds of errors predefined as part of the language. Here are some of the common types.

|  |  |  |
| --- | --- | --- |
| **Built-in Exception** | **When it will be raised** | **Example** |
| ZeroDivisionError | When a value is divided by zero | num\_list=[] total=10 avg=total/len(num\_list) |
| TypeError | When we try to do an operation with incompatible data types | total=10 total+="20" |
| NameError | When we try to access a variable which is not defined | avg=total/10 where total is not defined |
| IndexError | When we try to access an index value which is out of range | num\_list=[1,2,3,4] value=num\_list[4] |
| ValueError | When we use a valid data type for an argument of a built-in function but passes an invalid value for it | #string is a valid data type for int() but the value “A” is invalid, as "A" can't be converted into int. value="A" num=int(value) |

Python also allows us to handle different errors that can occur separately. That means you can have a different action or message for every unique error that occurs.

Here is the same expenditure calculation code with additional average expenditure calculation

def calculate\_expenditure(list\_of\_expenditure):

   total=0

   try:

       for expenditure in list\_of\_expenditure:

           total+=expenditure

       print("Total:",total)

       avg=total/num\_values

       print("Average:",avg)

   except ZeroDivisionError:

       print("Divide by Zero error")

   except TypeError:

       print("Wrong data type")

   except:

       print("Some error occured")

list\_of\_values=[100,200,300,"400",500]

num\_values=0

calculate\_expenditure(list\_of\_values)

**Try out - Customized Error Messages**

Problem Statement

Fix the below code to get customized error messages. You can use various except blocks.

Code in Python 3

def calculate\_expenditure(list\_of\_expenditure):

  total=0

  try:

       for expenditure in list\_of\_expenditure:

           total+=expenditure

       print("Total:",total)

       avg=total/num\_values

       print("Average:",avg)

  except:

       print("Some error occured")

  except ZeroDivisionError:

       print("Divide by Zero error")

  except TypeError:

       print("Wrong data type")

list\_of\_values=[100,200,300,"400",500]

num\_values=0

calculate\_expenditure(list\_of\_values)

**Try out - Multiple Try Blocks**

Problem Statement

* If an error occurs inside a function and if the error is not caught inside it, then the error is transferred to the function call where we have another opportunity to catch it.

Execute the below code and observe the results.

Code in Python 3

def calculate\_sum(list\_of\_expenditure):

  total=0

  try:

       for expenditure in list\_of\_expenditure:

           total+=expenditure

       print("Total:",total)

       avg=total/no\_values

       print("Average:",avg)

  except ZeroDivisionError:

       print("Divide by Zero error")

  except TypeError:

       print("Wrong data type")

try:

  list\_of\_values=[100,200,300,400,500]

  num\_values=len(list\_of\_values)

  calculate\_sum(list\_of\_values)

except NameError:

  print("Name error occured")

except:

  print("Some error occured")

**Try out - Finally Block**

Problem Statement

* Sometimes in programming we need to execute some code irrespective of whether the primary program logic itself succeeds or fails to do its job. In Python we can achieve this using a finally block.
* A finally block of statement is an optional part of the try-except statements. A code written inside the finally block will ALWAYS be executed.

Execute the below code and observe the results.

Code in Python 3

balance=1000

amount="300Rs"

def take\_card():

  print("Take the card out of ATM")

try:

  if balance>=int(amount):

       print("Withdraw")

  else:

       print("Invalid amount")

except TypeError:

  print("Type Error Occurred")

except ValueError:

  print("Value Error Occurred")

except:

  print("Some error Occurred")

finally:

  take\_card()

**Quiz - Exception Handling**

Q2 of

Go through the below code and predict the output:

num1=100

num2=0

try:

   result=num1/num2

   print(result)

except ZeroDivisionError:

   print("Zero Division Error Occurred")

Q3 of 4outlined\_flag

What will be the output of the code given below?

def division(value\_1,value\_2):

   try:

       return int(value\_1)/value\_2

   except TypeError:

       print("Type error")

   except ValueError:

       print("Value error")

   finally:

       print("Finally")

   print("Done")

division('A',10)

a. Value error  
    Finally  
    Done

b. Type error  
    Finally  
    Done

c. Type error  
    Finally

d. Value error  
    Finally

Q4 of 4outlined\_flag

What will be the output of the below code?

def find\_sum(value\_1,value\_2):

   try:

       print(value\_1+value\_3)

   except NameError:

       print("Function name error")

   finally:

       print("Sum finally")

try:

   find\_sum(12,13)

except NameError:

   print("Invocation name error")

finally:

   print("Invocation finally")

a. Function name error  
    Sum finally

b. Function name error  
    Sum finally  
    Invocation finally

c. Function name error  
    Sum finally  
    Invocation name error  
    Invocation finally

d. Sum finally  
    Invocation name error  
    Invocation finally

**Exercise on Exception Handling - Level 1**

Problem Statement

The python function, **find\_average()** given below, is written to accept a list of marks and return the average marks. On execution, the program is resulting in an error.  
  
1: Add code to handle the exception occurring in the code.  
  
2: Make the necessary correction in the program to remove the error.  
  
3: Make the following changes in the code, execute, observe the results. Add code to handle the errors occurring in each case.  
  
Case – 1: Initialize m\_list as ["1",2,3,4]  
  
Case – 2: Initialize m\_list as given below

mark1="A"

mark1=int("A")

m\_list=[mark1,2,3,4]

Case – 3: Initialize m\_list as []  
  
Case – 4: Make the following change in the for loop statement

for i in range(0, len(mark\_list)+1):

//----------------------------------------------------------------------------------

**Exception Handling Examples -**

Q1 of 8outlined\_flag

What is the output of the below program?

class InvalidAccountException(Exception):  
    pass  
class Account:  
    account\_list=[1001,1002,1003,1004]  
    def validate\_account(self,account\_id):  
        status=0  
        for acct\_id in self.account\_list:  
            if(account\_id==acct\_id):  
                status=1  
                break  
        if(status!=0):  
            return True  
        else:  
            raise InvalidAccountException  
try:  
    account1=Account()  
    account1.validate\_account(1006)  
    print("Valid account number")  
except InvalidAccountException:  
    print("Invalid account number")

Valid account number

Invalid account number

Error: Element not found in list

**Answer - (B) Invalid account number**

//-------------------------------------------

Q2 of 8outlined\_flag

What will be the output of the code given below?

class NameLengthException(Exception):  
    pass  
class EmployeeIdException(Exception):  
    pass  
class Employee:  
    \_\_trials=0  
    def \_\_init\_\_(self,emp\_id,emp\_name):  
        self.\_\_emp\_name=emp\_name  
        self.\_\_emp\_id=emp\_id  
    def validate\_name(self):  
        try:  
            if(len(self.\_\_emp\_name) < 4):  
                Employee.\_\_trials=Employee.\_\_trials+1  
                raise NameLengthException  
            if( not(self.\_\_emp\_id.startswith('E'))):  
                raise EmployeeIdException  
        except NameLengthException:  
            Employee.\_\_trials=Employee.\_\_trials+1  
            print(Employee.\_\_trials)  
        except EmployeeIdException:  
            Employee.\_\_trials=Employee.\_\_trials+1  
            print(Employee.\_\_trials)  
emp1=Employee('E1001','Tom')  
emp1.validate\_name()  
emp2=Employee('1001','Tomy')  
emp2.validate\_name()

a) 2  
    3

b) 2  
    1

c) 3

d) 2

a

b

c

D

**Answer - (A) 2 3**

//---------------------

Q3 of 8outlined\_flag

What will be the output of the code given below?

class Project:  
    def \_\_init\_\_(self,employee\_list):  
        self.\_\_employee\_list=employee\_list  
  
    def validate\_employee(self,employee\_id):  
        try:  
            if employee\_id not in self.\_\_employee\_list:  
                raise Exception  
                print("1")  
        except Exception:  
            print("2")  
  
project1=Project([1001,1002,1003])  
project1.validate\_employee(1005)  
print("3")

a) 2

b) 2  
    3

c) 2  
    1  
    3

d) 3

a

b

c

D

**Answer - (B) 2 3**

//----------------------

Q4 of 8outlined\_flag

What will be the output of the code given below?

class NotEligibleException(Exception):  
    pass  
class Employee:  
    def \_\_init\_\_(self,salary):  
        self.\_\_salary=salary  
  
    def check\_salary(self):  
        if(self.\_\_salary < 2000):  
            raise NotEligibleException  
            return False  
        else:  
            return True  
  
emp1=Employee(5000)  
emp2=Employee(1000)  
try:  
    status=emp1.check\_salary()  
    print(status)  
    status=emp2.check\_salary()  
    print(status)  
except NotEligibleException:  
    print("Not Eligible")

a) True  
    Not Eligible

b) True  
    False

c) True  
    Not Eligible  
    False

a

b

C

**Answer - (A) True Not Eligible**

//------------------------------------------------

Q5 of 8outlined\_flag

What will be the output of the code given below?

class NotEligibleException(Exception):  
    pass  
class Employee:  
    def \_\_init\_\_(self,salary):  
        self.\_\_salary=salary  
  
    def check\_salary(self):  
        try:  
            if(self.\_\_salary < 2000):  
                raise NotEligibleException  
            else:  
                return True  
        except NotEligibleException:  
            print("1")  
            raise NotEligibleException  
  
emp1=Employee(1000)  
try:  
    status=emp1.check\_salary()  
    print("2")  
except NotEligibleException:  
    print("3")

a) 2

b) Error: An exception cannot be raised from except block of another exception

c) Error: Two exceptions (inside a method and calling block) cannot have the same name

d) 1  
    3

a

b

c

D

**Answer - (d) 1 3**

//-------------------------------------------

Q6 of 8outlined\_flag

What will be the output of the code given below?

class InvalidEmployeeException(Exception):  
    pass  
class Project:  
    def \_\_init\_\_(self,employee\_list):  
        self.\_\_employee\_list=employee\_list  
  
    def validate\_employee(self,employee\_id):  
        flag=False  
        for key in self.\_\_employee\_list:  
            if(key==employee\_id):  
                flag=True  
        if(flag==False):  
            raise InvalidEmployeeException  
            print("1")  
        return True  
  
project1=Project([1001,1002,1003])  
try:  
    print(project1.validate\_employee(1005))  
except Exception:  
    print("2")  
except InvalidEmployeeException:  
    print("3")

a) 2

b) 2  
    3

c) 3

d) Error: Except should be the last block

a

b

c

D

**Answer - (A) 2**

//----------------------------------------------

Q7 of 8outlined\_flag

What will be the output of the code given below?

class CustomerBusiness:  
    def get\_customer(self,cust\_id):  
        if cust\_id == "":  
            raise InvalidCustomerException()  
        return cust\_id  
  
class AccountUI:  
    def deposit\_money\_ui(self):  
        try:  
            cust\_id = CustomerBusiness().get\_customer("")  
        except Exception:  
            print("Exception raised")  
        except InvalidCustomerException:  
            print("Invalid Customer Exception raised")  
  
class InvalidCustomerException(Exception):  
      pass  
  
a=AccountUI()  
a.deposit\_money\_ui()

Exception raised

Invalid Customer Exception raised

Compile Time Error

No Exception

**Answer - (A) Exception raised**

//------------------------------------------------

Q8 of 8outlined\_flag

What should be done in order to raise InvalidCustomerException?

class CustomerBusiness:  
    def get\_customer(self,cust\_id):  
        if cust\_id == "":  
            raise InvalidCustomerException()  
        return cust\_id  
  
class AccountUI:  
    def deposit\_money\_ui(self):  
        try:  
            cust\_id = CustomerBusiness().get\_customer("")  
        except Exception:  
            print("Exception raised")  
        except InvalidCustomerException:  
            print("Invalid Customer Exception raised.")  
  
class InvalidCustomerException():  
    pass  
  
a=AccountUI()  
a.deposit\_money\_ui()

**Option A**

class CustomerBusiness:  
    def get\_customer(self,cust\_id):  
        if cust\_id == "":  
            raise InvalidCustomerException()  
        return cust\_id  
  
class AccountUI:  
    def deposit\_money\_ui(self):  
        try:  
            cust\_id = CustomerBusiness().get\_customer("")  
        except Exception:  
            print("Exception raised")  
        except InvalidCustomerException:  
            print("Invalid Customer Exception raised.")  
  
class InvalidCustomerException(Exception):  
    pass  
  
a=AccountUI()  
a.deposit\_money\_ui()

**Option B**

class CustomerBusiness:  
    def get\_customer(self,cust\_id):  
        if cust\_id == "":  
            raise InvalidCustomerException()  
        return cust\_id  
  
class AccountUI:  
    def deposit\_money\_ui(self):  
        try:  
            cust\_id = CustomerBusiness().get\_customer("")  
    except InvalidCustomerException:  
            print("Invalid Customer Exception raised.")  
        except Exception:  
            print("Exception raised")  
  
class InvalidCustomerException(Exception):  
    pass  
  
a=AccountUI()  
a.deposit\_money\_ui()

**Option C**

class CustomerBusiness:  
    def get\_customer(self,cust\_id):  
        if cust\_id == None:  
            raise InvalidCustomerException()  
        return cust\_id  
  
class AccountUI:  
    def deposit\_money\_ui(self):  
        try:  
            cust\_id = CustomerBusiness().get\_customer("")  
        except Exception:  
            print("Exception raised")  
        except InvalidCustomerException:  
            print("Invalid Customer Exception raised.")  
  
class InvalidCustomerException(Exception):  
    pass  
  
a=AccountUI()  
a.deposit\_money\_ui()

Option A

Option B

Option C

**Answer - (B)**

**//----------------------------------------------------------------------------**

**File Handling -**

Python allows to create files, read from files, write content to file and append content to existing content through inbuilt functions!

|  |  |
| --- | --- |
| **Method** | **Description** |
| open(file\_path,operation) | This method is used to open the file for the specified operation. The operation can either be r,w,a for read, write and append. |
| close() | This method is used to close a file which is already open. |
| write() | This method is used to write a string to a file, if file is present. If not, it creates the file and writes the string into it. |
| read() | This method is used to read all the contents of a file into a string. |

The number of files that can be simultaneously opened by a program is limited. So it is very important to close all the files, once the operations are completed.

flight\_file=open("flight.txt","w")flight\_file.write("Hello")flight\_file.close()

Note: You can execute all the codes related to file handling in Eclipse Plug-in and observe the output

Sometimes, we may encounter exceptions when dealing with files.  
In this example, we are trying to write to a file which is opened in read-only mode. This will result in an exception.  
In such a case, let’s see how we can use try except block to catch this exception.

try:    flight\_file=open("flight.txt","r")

text=flight\_file.read()

   print(text)

flight\_file.write(",Good Morning")

 flight\_file.close()

except:

 print("Error occurred")

if flight\_file.closed:

print("File is closed")

else:

     print("File is open")

//-------------------------------------------------------------

try:  
    flight\_file=open("flight.txt","r")  
    text=flight\_file.read()  
    print(text)  
    flight\_file.write(",Good Morning")  
except:  
    print("Error occurred")  
finally:  
    print("File is being closed")  
    flight\_file.close()  
    if flight\_file.closed:  
        print("File is closed")  
    else:  
        print("File is open")

//------------------------------------------------------------------

When we want to write a code that will run ... by Ashutosh (Unverified)Ashutosh (Unverified)2:29 PM

When we want to write a code that will run in all situations, we can put it in a finally block.  
Since closing a file is necessary we can do it in the finally block instead of in the try block.

**Write text in file -**

try:

hello\_file = open("flight.txt", "w")

text = "Hello everyone! Welcome"

hello\_file.write(text)

except:

print("Error occurred, not able to write to file")

finally:

hello\_file.close()

try:

hello\_file = open("flight.txt", "r")

text\_from\_file = hello\_file.read()

print(text\_from\_file)

except:

print("Error Occurred, not able to read from file")

finally:

hello\_file.close()

//-----------------------------------------------------------

1. try:
2. hello\_file=open("flight.txt","w")
3. text="Hello everyone! Welcome"
4. hello\_file.write(text)
5. except:
6. print("Error occurred, not able to write to file")
7. finally:
8. hello\_file.close()
9. try:
10. hello\_file=open("flight.txt","r")
11. text\_from\_file=hello\_file.read()
12. print(text\_from\_file)
13. except:
14. print("Error Occurred, not able to read from file")
15. finally:
16. hello\_file.close()

//------------------------------------------------------------------------------

f=open('abc.txt','w')

f.write('GET')

f.write('SET')

f.write('TECH')

print('Data Written Successfully')

f.close()

f=open('abc.txt','w')

f.write('GET\n')

f.write('SET\n')

f.write('TECH\n')

print('Data Written Successfully')

f.close()

f=open('abc.txt','a')

f.write('GET\n')

f.write('SET\n')

f.write('TECH\n')

print('Data Written Successfully')

f.close()

f=open('abc.txt','w')

list=['Sachin','Virat','Rohit','Rahul','Dhawan']

f.writelines(list)

print('Data Written Successfully')

f.close()

f=open('abc.txt','r')

data=f.read(20)

print(data)

f.close()

f=open('abc.txt','r')

line1=f.readline()

print(line1)

line2=f.readline()

print(line2)

line3=f.readline()

print(line3)

f.close()

f=open('abc.txt','r')

line1=f.readline()

print(line1,end='')

line2=f.readline()

print(line2,end='')

line3=f.readline()

print(line3,end='')

f.close()

f=open('abc.txt','r')

lines=f.readlines()

for line in lines:

  print(line)

f.close()

f=open('abc.txt','r')

lines=f.readlines()

for line in lines:

  print(line,end='')

f.close()

def read\_file(fname):

 f=open(fname+'.txt','r')

  data=f.read()

  print(data)

star 1

[3:06 PM] Ashutosh (Unverified)

1. try:
2. hello\_file=open("flight.txt","w")
3. text="Hello everyone! Welcome"
4. hello\_file.write(text)
5. except:
6. print("Error occurred, not able to write to file")
7. finally:
8. hello\_file.close()
9. try:
10. hello\_file=open("flights.txt","r")
11. text\_from\_file=hello\_file.read()
12. print(text\_from\_file)
13. except:
14. print("Error Occurred, not able to read from file")
15. finally:
16. hello\_file.close()

star 1

[3:06 PM] Ashutosh (Unverified)

try:  
    hello\_file=open("flight.txt","w")  
    text="Hello everyone! Welcome"  
    hello\_file.write(text)  
except:  
    print("Error occurred, not able to write to file")  
finally:  
    hello\_file.close()  
try:  
    hello\_file=open("flights.txt","r")  
    text\_from\_file=hello\_file.read()  
    print(text\_from\_file)  
except:  
    print("Error Occurred, not able to read from file")  
finally:  
    hello\_file.close()

star 1

[3:07 PM] Ashutosh (Unverified)

try:  
    hello\_file=open("flight.txt","w")  
    text="Hello everyone! Welcome"  
    hello\_file.write(text)  
except:  
    print("Error occurred, not able to write to file")  
finally:  
    hello\_file.close()  
try:  
    hello\_file=open("flight.txt","r")  
    text\_from\_file=hello\_file.read()  
    print(text\_from\_file)  
except:  
    print("Error Occurred, not able to read from file")  
finally:  
    hello\_file.close()

//-------------------------------------------------------