Exception Handler

P-1 (USE OF ASSERT)

```
def count(n):
   print(" Your Name is",n)
   assert len(n) == 0, "PLz Insert name Value"

name = input("Enter Your Name :")
count(name)
```

P-2 (USE OF FINALLY)

```
try:
   num1 = int(input("Enter First numbers : "))
   num2 = int(input("Enter Two numbers : "))
   result = num1 / num2
   print("Result is", result)
except ZeroDivisionError:
    print("Division by zero is error !!")
except SyntaxError:
    print("Comma is missing. Enter numbers separated by comma like this 1, 2")
except:
    print("Wrong input")
else:
    print("No exceptions")
finally:
    print("This will execute no matter what")
```

P-3 (USE OF RAISE)

```
x = int(input("Enter Number :"))
if x !=10 :
   raise Exception(" X Value is Not Equal to 10")
```

P-4

```
try:
        a = input("Enter Number")
        b = int(a)
        print("You Value is ",a)
        print(" Variable is ",b)
except ValueError:
        print("Plz Input only number")
except NameError:
        print(" Variable name Not Found")
```

P-5 (USE OF ELSE)

```
try:
        a = int(input("Enter number "))
        b = int(input("Enter Number "))
        c = a/b
except SyntaxError:
    print("Syntax Error")
except ValueError:
    print("Plz Input Number")
except ZeroDivisionError:
    print("Zero is not Division Error")
else:
    print (" Division is %d" %c)
```

P-1 (BUILT METHOD)

```
class Student:
  'Common base class for all Student'
  Count= 0

  def __init__(self, name,age):
      self.name = name
      self.age = age
      Student.Count += 1

  def displayCount(self):
      print ("Total Student %d" %Student.Count)

  def displayStudent(self):
      print("Name : ", self.name,  ", Age: ", self.age)

print("Student.__doc__:",Student.__doc__)
print ("Student.__name__:", Student.__name__)
print ("Student.__module__:", Student.__module__)
print ("Student.__bases__:", Student.__bases__)
print ("Student.__dict__:", Student.__dict__)
```

P-2

```
# WRITE A PYTHON SCRIPT FOR ACCEPT NAME,AGE FROM USER AND DISPLAY IT
class person:
        age=0
        name=""
        def getdata(self):
                self.name = input("Enter Name :-")
                self.age = int(input("Enter Age:-"))

        def display(self):
                print("\nYour name ",self.name,"\nYour Age ",self.age)


p=person()
p.getdata()
p.display()
```

P-3 # constrator

```
class Student:
  count=0
  def __init__(self, name, age):            # constrator
    self.name = name
    self.age = age
    Student.count+=1
```

```
  def displayCount(self):
    print ("Total Student %d" % Student.count)

  def displayStudent(self):
     print ("Name : ", self.name,  ", age: ", self.age)


st1 = Student("ABC",20)
st1.displayStudent()
st1.displayCount()
st2 = Student("DEF",30 )
st2.displayStudent()
print ("Total Student %d" %Student.count)
```

P-4

```
# Constractor
class Student:
  'Common base class for all Student'
  Count= 0
  def __init__(self, name,age):
     self.name = name
     self.age = age
     Student.Count += 1

  def displayCount(self):
     print ("Total Student %d" %Student.Count)

  def displayStudent(self):
     print("Name : ", self.name,  ", Age: ", self.age)

print("\n1st Student\n")
name=input("Enter Name :")
age = int(input("Entet Age :"))
st1 = Student(name,age)
st1.displayStudent();
st1.displayCount();

print("\n2nd Student\n")
name=input("Enter Name :")
age = int(input("Entet Age :"))
st2 = Student(name,age)
st2.displayStudent();
st2.displayCount();
```

P-5

```
# DeConstractor
class Student:
```

```
   'Common base class for all Student'
   Count= 0
   def __init__(self, name,age):
      self.name = name
      self.age = age
      Student.Count += 1
   def displayCount(self):
      print ("Total Student %d" %Student.Count)
   def displayStudent(self):
      print("Name : ", self.name,  ", Age: ", self.age)
   def __del__(self):
       class_name=self.__class__.__name__
       print("Destroyed Student Class")

print("\n1st Student\n")
name=input("Enter Name :")
age = int(input("Entet Age :"))
st1 = Student(name,age)
st1.displayStudent();
st1.displayCount();

del st1

P-6
# inheritance
class College:
    code=23
    def __init__(self):
        print("Parent Class")

    def College_Method(self):
        print("College_Method-1")

    def setAttr(self,a):
        College.code=a

    def getAttr(self):
        print(" Parent Value",College.code)

class Student(College) :
    def __init__(self):
        print(" Child Class")

    def Student_Method(self):
        print(" Child Method")


c = Student()
c.Student_Method()
```

```
c.College_Method()
c.setAttr(23028)
c.getAttr()
```

P-7

```
#inheritance
class class1:
    a=0
    b=0
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def  getValue_2(a,b):
        self.a=a
        self.b=b
    def Display_a(self):
        print("Value of a is {}".format(self.a))
    def Display_b(self):
        print("Value of b is {}".format(self.b))


class class2(class1):
    c=0
    def __init__(self,c):
        self.c=c
    def getValue_1(self,c):
        self.c=c
    def Display_c(self):
        print("Value of c is {}".format(self.c))


d = class2(5)
# SUB CLASS
print("SUBCLASS")
d.Display_c();
d.getValue_1(10)
d.Display_c();

# main class
print("Main CLASS")
e = class1(20,30)
e.Display_a();
e.Display_b();
```

P-8

```
#inheritance
class c1:
    a=0
```

```
    b=0
    def  get_ab(self,a,b):
        self.a=a
        self.b=b
    def show(self):
        print("Main Class")
        print("Value of a is {}".format(self.a))
        print("Value of b is {}".format(self.b))

class c2(c1):
   c=0
   def get_c(self,c):
        self.c=c

   def mul(self):
        self.c=self.b*self.a;
   def display(self):
        print("CHILD CLASS")
        print(" Value of a is {}".format(self.a))
        print(" Value of b is {}".format(self.b))
        print(" Value of c is {}".format(self.c))


x = c2()
x.get_ab(10,20)
x.mul()
x.show()
x.display()

x.a=35
x.b=25
x.mul()
x.display()
```

P-9

```
# method OverRidding
class class1(object):
    a=0
    b=0
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def Display(self):
        print("Value of a is {}".format(self.a))
        print("Value of b is {}".format(self.b))


class class2(class1):
    c=0
```

```
    def __init__(self,a,b,c):
        super().__init__(a,b)  # main class constrator call
        self.c=c
    def Display(self):
        print("Value of a is {}".format(self.a))
        print("Value of b is {}".format(self.b))
        print("Value of c is {}".format(self.c))

x = class2(10,20,30)
x.Display()
```

P-10

```
# method Overload
class class1:
    a=0
    b=0
    c=0
    def __init__(self,a,b,c):
        self.a=a
        self.b=b
        self.c=c
    def  Getdata(self,a):
        self.a=a
    def  Getdata(self,a,b):
        self.a=a
        self.b=b
    def  Getdata(self,a,b,c):  #  USE ONLY LAST METHOD CREATED BY USER
        self.a=a
        self.b=b
        self.c=c
    def Display(self):
        print(" Value of a {}".format(self.a))
        print(" Value of b {}".format(self.b))
        print(" Value of c {}".format(self.c))


x = class1(1,2,3)
x.Display()


x.Getdata("JJKCC","MGCT","RAJKOT")
x.Display()
```

P-11

```
# MULTI LEVEL INHERITANCE
class student:
    roll_no=0
    def getnumber(self,a):
```

```
        self.roll_no=a
    def putnumber(self):
        return self.roll_no

class test(student): # First Level
    sub1=0
    sub2=0
    sub3=0
    def getmark(self,a,b,c):
        self.sub1=a;
        self.sub2=b;
        self.sub3=c;
    def putmark(self):
        return self.sub1,self.sub2,self.sub3

class result(test): # Second Level
    total=0
    def Display(self):
        self.total=self.sub1+self.sub2+self.sub3
        print("Roll Number {}".format(self.roll_no))
        print("Subject marks 1 {}".format(self.sub1))
        print("Subject marks 2 {}".format(self.sub2))
        print("Subject marks 3 {}".format(self.sub3))
        print("Subject Total {}".format(self.total))


st1 = result()
st1.getnumber(10)
st1.getmark(45,55,65)
st1.Display()
# put method display Value
print("PUT METHOD Value Dispaly")
print("PUT METNOD",st1.putmark())
print("PUT METNOD",st1.putnumber())
```

P-12

```
# multiple Inheritance

class class1():
    v1=0
    def getv1(self,a):
        self.v1=a

class class2():
    v2=0
    def getv2(self,b):
        self.v2=b

class class3(class1, class2):
```

```
    def Display(self) :
        print(" Value of First Class Variable is {}".format(self.v1))
        print(" Value of Second Class Variable is {}".format(self.v2))


x = class3()
x.getv1(10)
x.getv2(20)
x.Display()
```

P-13

```
# HYBRID INHERITANCE
class student(): # Main -1
    rollno=0
    def getrollno(self,a):
        self.rollno=a
    def putrollno(self):
            return self.rollno

class test(student): # sub of main -1
    sub1=0
    sub2=0
    sub3=0
    def getsub(self,a,b,c):
        self.sub1=a
        self.sub2=b
        self.sub3=c
    def putsub(self):
        return self.sub1, self.sub2, self.sub3

class sport(): # main -2
    score=0
    def getscore(self,a):
        self.score=a
    def putscore(self):
        return self.score

class result(test,sport):
    total=0
    def Display(self):
        self.total = self.sub1+self.sub2+ self.sub3
        print("Roll Number :",student.putrollno(self))
        print(" Marks :",test.putsub(self))
        print("Score : ",sport.putscore(self))

x = result()
x.getrollno(101)
x.getsub(45,45,45)
x.getscore(1)
```

```
x.Display()
```

P-14

```
class media:
   title=""
   price=0.0
   def __init__(self,a,b):
      self.title=a
      self.price=b

class book(media):
   page=0
   def __init__(self,a,b,c):
      super().__init__(a,b)
      self.page=c
   def display(self):
      print(" Book - Title ",self.title)
      print(" Book-Price " ,self.price)
      print(" Book-Page",self.page)

class disk(media):
    time=0.0
    def __init__(self,a,b,c):
       super().__init__(a,b)
       self.time=c
    def display(self):
       print(" DISK - Title ",self.title)
       print(" DISK-Price " ,self.price)
       print("DISK-Time",self.time)


try :
   x = input("Enter Book Title : ")
   y= int(input("Enter Book Price: "))
   z= int(input(" Enter Pages:  "))
except ValueError:
   print(" PLz Input Valid Values in Book")

b1 = book(x,y,z)
b1.display()


try :
   x = input("Enter Disk Title : ")
   y= int(input("Enter Disk Price: "))
   z= input(" Enter Time:  ")
except ValueError:
   print(" PLz Input Valid Values Disk")
```

```
d1 = disk(x,y,z)
d1.display()

P-15
class Data:
   __var1=0
   def addition(self,value):
      self.__var1+=value
      print(self.__var1)


a = Data()
a.addition(3)
a.addition(4)
print(a.__var1)

# hide data Display
print(" Encapsulation",a._ata__var1)


P-16
# Arithmetic Operator Overload
"""
Operator                Expression              Internally
Addition                    p1 + p2             __add__
Subtraction               p1 - p2               __sub__
Multiplication             p1 * p2              __mul__
Power                   p1 ** p2       __pow__
Division                    p1 / p2             __truediv__
Floor Division             p1 // p2          __floordiv__
Remainder (modulo)          p1 % p2                 __mod__
Bitwise Left Shift          p1 << p2            __lshift__
Bitwise Right Shift         p1 >> p2            __rshift__
Bitwise AND               p1 & p2             __and__
Bitwise OR                p1 | p2             __or__
Bitwise XOR               p1 ^ p2             __xor__
Bitwise NOT               ~p1                   __invert__

"""

class Example:
  """ Program to overload Arithmetic Operator """
  def __init__ (self, a , b):
     self.a = a
     self.b = b
  def __add__(self,other):
     a=self.a + other.a
     b=self.b+other.b
```

```
    return Example(a,b)
    #return Example(self.a + other.a, self.b + other.b)
 def __sub__(self,other):
     return Example(self.a - other.a, self.b - other.b)
 def __mul__(self,other):
     return Example(self.a * other.a, self.b * other.b)
 def __str__(self):
     return "({0},{1})".format(self.a,self.b)
 def  __pow__(self,other):
     return Example(self.a ** other.a, self.b ** other.b)


v1 = Example(6,8)
v2 = Example(3,4)
v3 = Example(1.2,2.2)


print("VALUE OF FIRST OBJECT: -",v1)
print("VALUE OF SECOND OBJECT: -",v2)
print("VALUE OF THIRD OBJECT: -",v3)
print ("ADDTION 1st and 2nd Object: -", v1 + v2)
print ("SUBTRACTION 1st and 3rd Obejct: -" ,v1 - v3)
print ("MULTIPLIATION 1at and 2nd Object :-",v1 * v2)
print("POWER OF 1st and 2nd  Object ",v1**v2)
```

P-17

```
"""
Operator                    Expression Internally
Less than                   p1 < p2          __lt__
Less than or equal to      p1 <= p2        __le__
Equal to                 p1 == p2              __eq__
Not equal to              p1 != p2             __ne__
Greater than              p1 > p2              __gt__
Greater than or equal to     p1 >= p2              __ge__

"""

class distance:
   def __init__(self, x=5,y=5):
      self.ft=x
      self.inch=y

   def __eq__(self, other):
      if self.ft==other.ft and self.inch==other.inch:
         return "both objects are equal"
      else:
         return "both objects are not equal"
```

```
    def __ne__(self, other):
        if self.ft!=other.ft and self.inch !=other.inch:
            return "both objects are not equal"
        else:
            return "both objects are equal"

    def __ge__(self, other):
        in1=self.ft+self.inch
        in2=other.ft+other.inch
        if in1>=in2:
            return "first object greater than or equal to other"
        else:
            return "first object smaller than other"

    def __lt__(self, other):
        in3=self.ft+self.inch
        in4=other.ft+other.inch
        if in3<=in4:
            return "first object less than or equal to other"
        else:
            return "first object smaller than other"



d1=distance(5,5)
d2=distance()
print (d1==d2)

d3=distance()
d4=distance(6,10)
print (d3!=d4)

d5=distance(3,11)
d6=distance()
print(d5>=d6)

d7=distance()
d8=distance(6,11)
print(d7<d8)

P-18
"""
        Operators       Expression              Internally
    -=                  p2 -= p1                __isub__
                +=                  p2 += p1                __iadd__
                *=                  p2 *= p1                __imul__
                /=                  p2 /= p1                __idiv__
                //=                 p2 //= p1               __ifloordiv__
```

```
                %=              p2 %= p1            __imod__
                **=             p2 **= p1           __ipow__
                >>=             p2 >>= p1           __irshift__
                <<=             p2 <<= p1           __ilshift__
                &=              p2 &= p1            __iand__
                !=              p2 != p1            __ior__
                ^=              p2 ^= p1            __ixor__
"""

class Example:
   def __init__(self,a,b):
      self.a = a
      self.b = b

   def __str__(self):
      return "({0},{1})".format(self.a,self.b)

   def __iadd__(self,other):
      self.a += other.a
      self.b += other.b
      return Example(self.a,self.b)

   def __isub__(self,other):
      self.a += other.a
      self.b += other.b
      return Example(self.a,self.b)

v1 = Example(1,2)
v2 = Example(2,3)
v2 += v1
print ("Assignment with Plus Operater",v2)

v3 = Example(10,20)
v4 = Example(2,3)
v3 -= v4
print ("Assignment with Minus Operater",v3)
```