Q1.Write a program that accepts two integers and displays the result of dividing the first number by the second number. It should raise ZeroDivisionError if the denominator is zero.

In [17]:
```
def func(a,b):
    print("You entered the function...")
    result = a/b
    print('End of the function')
    return result
try:
    first = int(input("Enter the first number : "))
    second = int(input("Enter the second number : "))
    res = func(first,second)
    print("The result is : ",res)

except ZeroDivisionError as z:
    print(z, error has raised')

else:
    print("No Exception occured in this program")
finally:
    print("End of the program")
```

```
Enter the first number : 4
Enter the second number : 0
You entered the function...
division by zero error has raised
End of the program
```

In [ ]:

Q2.Implement the above program with the help of Assertion

In [4]:
```
first = int(input("Enter the first number : "))
second = int(input("Enter the second number : "))
assert second!=0,'Denominator is zero'
res = func(first,second)
print("The result is : ",res)
print('After assertion')
func(first,second)
```

```
Enter the first number : 4
Enter the second number : 0
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [4], in <cell line: 3>()
      1 first = int(input("Enter the first number : "))
      2 second = int(input("Enter the second number : "))
----> 3 assert second!=0,'Denominator is zero'
      4 res = func(first,second)
      5 print("The result is : ",res)

AssertionError: Denominator is zero
```

In [ ]:

Q3.Write a program that accepts n positive integers as input and prints their square. If a negative number is entered, then raise ValueError exception and display a relevant error message and make an exit.

In [5]:
```
class Squares(object):
    def __init__(self,l):
        self._limit =l
        self._val = 1
    def __iter__(self):
        return self
    def __next__(self):
        if self._val > self._limit:
            raise StopIteration

        else:
            result = self._val * self._val
            self._val += 1
            return result



num = int(input("Enter the n positive interger : "))


try:
    if num < 0:
        raise ValueError()

except ValueError as v:
    print('Enter the positive integer of n')

for i in Squares(num):
    print(i,end = ', ')


print('\nEnd of the program')
```

```
Enter the n positive interger : 3
1, 4, 9,
End of the program
```

In [ ]:

Q4.Write a program that receives an integer as input and finds its factorial. If a non-integer input is entered, then report an error and accept the input again. Continue this process until correct input is entered.

In [ ]:
```
def factorial(n):
    f = 1
    for i in range(n):
        f *= (i+1)
    return f

while True:
    try:
        N = int(input("Enter the number/string : "))
        if(isinstance(N,int) == True):
            print(factorial(N))

        else:
            continue

    except Exception as e:
        print(e)
```

In [ ]:

Q5.Create an iterator that returns numbers, starting with 1, and each sequence will increase by one (returning 1,2,3,4,5 etc.):

In [10]:
```
class Seq(object):
    def __init__(self,l):
        self._limit =l
        self._val = 1

    def __iter__(self):
        return self

    def __next__(self):
        if self._val > self._limit:
            raise StopIteration

        else:
            result = self._val
            self._val += 1
            return result



n = int(input("Enter the integer : "))
print('The sequence of 1 to ',n,' is : ')
for i in Seq(n):
    print(i,end='\n')
```

```
Enter the integer : 5
The sequence of 1 to  5  is :
1
2
3
4
5
```

In [ ]:

Q6.Write a program that implements iterator class called Prime and which prints prime numbers up to n.

In [12]:
```
def Prime(n):
    if(n==1 or n==0):
        return False
    for i in range(2,n):
        if(n%i==0):
            return False

    return True
print("Starting...")

try:
    n = int(input("Enter an integer :"))
    for i in range(2,n):
        if(Prime(i)):
            print(i,end="\n")

except Exception:
    print("Exception Occured at runtime")

finally:
    print("End of the program")
```

```
Starting...
Enter an integer :10
2
3
5
7
End of the program
```

In [ ]:

Q7.Define a class, Circle, which takes two arguments when defined — a sequence and a number. The idea is that the object will then return elements the defined number of times. If the number is greater than the number of elements, then the sequence repeats as necessary.

In [13]:
```
class Circle():
    def __init__(self,data,max_times):
        self.data = data
        self.max_times = max_times
        self.index = 0


    def __iter__(self):
        return self


    def __next__(self):
        if self.index >= self.max_times:
            raise StopIteration
        value = self.data[self.index % len(self.data)]
        self.index += 1
        return value

c = Circle('abcd', 7)
print(list(c))
```

```
['a', 'b', 'c', 'd', 'a', 'b', 'c']
```

In [ ]:

Q8.Write a Python program to demonstrate working of iterators using an example type that iterates from 10 to given value

In [14]:
```
#Q8:
class Test:
    def __init__(self, limit):
        self.limit = limit

    def __iter__(self):
        self.x = 10
        return self
    def __next__(self):
        x = self.x
        if x > self.limit:
            raise StopIteration
        self.x = x + 1;
        return x


n=int(input('Enter a value :'))
for i in Test(n):
    print(i)
```

```
Enter a value :34
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
```

In [ ]:

Q9.Write a program to illustrate the concept of multiple decorators

In [15]:
```
def ash(func):
    def inner(*args,**kwargs):
        print('#' * 50)
        func(*args,**kwargs)
        print('#'*50)
    return inner

def dollar(func):
    def inner(*args,**kwargs):
        print('$' * 50)
        func(*args,**kwargs)
        print('$'*50)
    return inner



@ash
@dollar
def message(msg):
    print(msg)
message('Multiple Decorators')
```

```
##################################################
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Multiple Decorators
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
##################################################
```

In [ ]:

Q10.Write a decorator called timer that calculates the time required to execute a function. Its wrapper function called calculates should take all the parameters that are passed to the decorated function

In [16]:
```
import time

def timer(func):
    def calculates(*args,**kwargs):
        start_time = time.perf_counter()
        value = func(*args,**kwargs)
        end_time = time.perf_counter()
        runtime = end_time - start_time
        print('Program Execution timing is :::{0:.20.10f}secs'.format(runtime))
        return value
    return calculates

@timer
def factorial(n):
    f = 1
    for i in range(n):
        f *= (i+1)
    return f
n = int(input("Enter the number : "))
print('Factorial of _n_ is: ',factorial(n))
```

```
Enter the number : 5
Program Execution timing is :::          0.0000097000secs
Factorial of  5  is:  120
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: