Assignment 10

1. What is the role of try and exception block?

Explanation:

- **In Python, try and exception blocks are used to handle the errors at the time of execution of the program.**
- **It will prevent the program from being abnormally terminated.**
- **It will catch the specific exceptions, handle it properly and execute the program without crashing.**

Snippet:

```
def divide(a,b):
        try:
            print(f'The value is: {a/b}')
            print('--Try block executed--')
        except:
            print("Divide by zero")
            print('--Except block executed--')
    a = int(input('Enter the numerator: '))
    b = int(input('Enter the denominator: '))
    divide(a,b)
```

Output:

```
Enter the numerator  : 10
Enter the denominator: 20
The value is: 0.5
--Try block executed--

Enter the numerator  : 10
Enter the denominator: 0
Divide by zero
--Except block executed--
```

2.What is the syntax for a basic try-except block?

Syntax:

```
try:
        Code which raises an exception
except:
        Code that will handle the exception
```

3. What happens if an exception occurs inside a try block and there is no matching except block?

Explanation:

- **If an exception occurs inside a try block and there is no matching except block it will propagate up the call stack until it reaches the top level of the program.**
- **If there is no matching except block it will throwback error.**

Snippet:

```
def divide(a,b):
        try:
            print(f'The value is: {a/b}')
            print('--Try block executed--')
```

```
    except ValueError:
        print("Divide by zero")
        print('--Except block executed--')
a = int(input('Enter the numerator  : '))
b = int(input('Enter the denominator: '))
divide(a,b)
```

Output:

```
Enter the numerator  : 1
Enter the denominator: 0
ERROR!
Traceback (most recent call last):
  File "<string>", line 10, in <module>
  File "<string>", line 3, in divide
ZeroDivisionError: division by zero
```

4. What is the difference between using a bare except block and specifying a specific exception type?
Explanation:
Bare except block:
It catches all types of exceptions indiscriminately.
This means that any exception, regardless of its type, will be caught and handled by the bare except block.
This can be convenient for catching and handling unexpected or unknown exceptions

Snippet:

```
try:
    a = int(input('Enter the number: '))
    b = int(input('Enter the number: '))
    result = a / b
    print("The result is: ",result)
except:
    print("An exception occurred")
```

Output:

```
Enter the number: 10
Enter the number: 20
The result is:  0.5

Enter the number: 10
Enter the number: 0
An exception occurred
```

Specific Exception:
-It only catches exceptions of that type or its derived types.
-This allowsus to handle different exceptions in different ways, providing more control over the exception handling process.
By catching specific exception types, we can tailor our error handling logic to handle differ ent exceptions differently,
such as providing specific error messages or taking appropriate recovery actions.

Snippet:

```
def add():
   try:
      a = int(input('Enter the value 1:'))
      b = int(input('Enter the value 2:'))
      print(f'The sum of numbers are {a+b}')
   except ValueError:
      print('Enter the integer ')
add()
```

Output:

```
Enter the value 1: 10
Enter the value 2: 20
the sum of numbers is: 30
```

5. Can you have nested try-except blocks in Python? If yes, then give an example.

Explanation:

Yes, python can have nested try-except blocks in python.

Snippet:

```
def div():
   try:
      numerator =   int(input('Enter the numerator  : '))
      denominator = int(input('Enter the denominator: '))
      try:
         division = numerator/denominator
         print("The division is: ",division)
      except ZeroDivisionError:
         print('Denominator should not be zero.. \nenter the values above 0')
   except ValueError:
         print('You should not a enter the string or any other values apart from
integer')
div()
```

Output:

```
Enter the numerator  : 10
Enter the denominator: 20
The division is:  0.5

Enter the numerator  : 10
Enter the denominator: 0
Denominator should not be zero..
enter the values above 0

Enter the numerator  : 10
Enter the denominator: a
You should not a enter the string or any other values apart from  integer
```

6. Can we use multiple exception blocks, if yes then give an example.

Explanation:

Yes, in Python we can use exception blocks.

Snippet:

```
        def div():
            try:
              numerator =   int(input('Enter the numerator  : '))
              denominator = int(input('Enter the denominator: '))
              division = numerator/denominator
              print("The division is: ",division)
            except ZeroDivisionError:
              print('Denominator should not be zero.. \nenter the values above 0')
            except ValueError:
              print('You should not a enter the string or any other values apart from
        integer')
div()
```

Output:

```
        Enter the numerator  : 10
        Enter the denominator: 20
        The division is:  0.5

        Enter the numerator  : 10
        Enter the denominator: 0
        Denominator should not be zero..
        enter the values above 0

        Enter the numerator  : 10
        Enter the denominator: a
        You should not a enter the string or any other values apart from integer
```

7. Write the reason due to which following errors are raised:

a. EOFError
b. FloatingPointError
c. IndexError
d. MemoryError
e. OverflowError
f. TabError
g. ValueError

a. **EOFError: It's an end of file error. It will occur after the function reaches the end of the file.**
   Snippet:

```
        def handling_exception():
          try:
            user_input=input('Enter the string:')
          except EOFError:
             print('End Of File reached...')
        handling_exception()
```

b. FloatingPointError:
   This error raise when there is an error in floating-point calculations, such as dividing a number by zero or attempting to perform

an invalid mathematical operation.
Snippet:

```
try:
    user_input = input("Enter a value: ")
except FloatingPointError:
    print("End of input reached.")
```

c. Index Error:

IndexError is raised when we try to access a list, tuple, or other sequence with an invalid index or an index that is out of range

Snippet:

```
def access():
   try:
      my_list=list(input('Enter the elements to be in the list: '))
      access = int(input('Enter the index to be accessed in the list: '))
      access1 = my_list[access]
      print(f'The element {access} in the list are ',access1)
   except:
      print('Index out of range...')
access()
```

Output:

```
Enter the elements to be in the list: 123
Enter the index to be accessed in the list: 2
The element 2 in the list are  3

Enter the elements to be in the list: 123
Enter the index to be accessed in the list: 5
Index out of range...
```

d. MemoryError

- **This error is raised when the program runs out of available memory.**
- **This can happen when a program tries to allocate more memory than the system has a vailable.**

Snippet:

```
try:
   big_list = [0] * (10**9)
except MemoryError:
   print("Memory allocation error occurred.")
```

e. OverflowError:

OverflowError occurs in situations where the result of a calculation is too large to be stored within the specified data type.

Snippet:

```
try:
   result = 10**1000  # Attempt to calculate a large exponentiation
except OverflowError:
   print("Arithmetic overflow occurred.")
```

e. TabError:

This error is raised when there are inconsistencies in the indentation of code u sing tabs and spaces.

Snippet:

```
try:
   if True:
      print("Indented with spaces")
   \tprint("Indented with a tab")
except TabError:
   print("Indentation error occurred.")
```

f. ValueError:

ValueError occurs when a function receives an argument of the correct type but an invalid value.

Snippet:

```
try:
   string = input('Enter the string: ')
   res = int(string)
   print('The value is',res)
except ValueError:
   print("Cannot be converted from string to integer")
```

Output:

```
Enter the string:surya
The value is 123

Enter the string:surya
Cannot be converted from string to integer
```

8. Write

a. Program to divide two numbers
b. Program to convert a string to an integer
c. Program to access an element in a list
d. Program to handle a specific exception
e. Program to handle any exception
code for the following given scenario and add try-exception block to it.

a. Program to divide two numbers

Snippet:

```
def div():
      try:
         numerator =   int(input('Enter the numerator  : '))
         denominator = int(input('Enter the denominator: '))
         division = numerator/denominator
         print("The division is: ",division)
      except:
         print('Denominator should not be zero.. \nenter the values above 0')
div()
```

Output:

```
Enter the numerator  : 10
Enter the denominator: 20
The division is:  0.5

Enter the numerator  : 10
Enter the denominator: 0
Denominator should not be zero..
enter the values above 0
```

b. Program to convert a string to an integer

Snippet:

```
def convert(word):
   try:
      word = int(word)
      print(f'Converted string is {word}')
   except:
      print('we cannot convert string to integer')
word = input('Enter the String: ')
convert(word)
```

Output:

```
Enter the String: 123
Converted from String to Integer is 123

Enter the String: surya
Cannot convert string to integer
```

c. Program to access an element in a list

Snippet:

```
def accessElements():
   try:
      my_list = list(input('Enter the elements that need to be added in the list:'))
      print(my_list)
      access = int(input('Enter the element to be accessed from the list:'))
      accessElement = my_list[access]
      print('The element in the list is ',accessElement)
   except:
      print('Index Out of Range..')
accessElements()
```

Output:

```
Enter the elements that need to be added in the list:12345
['1', '2', '3', '4', '5']
Enter the element to be accessed from the list:1
The element in the list is  2

Enter the elements that need to be added in the list:123
['1', '2', '3']
Enter the element to be accessed from the list:4
```

```
Index Out of Range..
```

d. Program to handle a specific exception

Snippet:
```
def div():
      try:
         numerator =   int(input('Enter the numerator  : '))
         denominator = int(input('Enter the denominator: '))
         division = numerator/denominator
         print("The division is: ",division)
      except ZeroDivisionError:
         print('Denominator should not be zero.. \nenter the values above
0')
div()
```

Output:
```
Enter the numerator  : 10
Enter the denominator: 20
The division is:  0.5

Enter the numerator  : 10
Enter the denominator: 0
Denominator should not be zero..
enter the values above 0
```

e. Program to handle any exception

Snippet:
```
def handling_exception():
   try:
      a = int(input('Enter the value1: '))
      b = int(input('Enter the value2'))
      c = a/b
      print('Result = 'c)
      print('No errors..')
   except Exception as e:
      print('Error Occured',type(e).__name__)
handling_exception()
```

Output:
```
Enter the value1: 10
Enter the value220
Result =  0.5
No errors..

Enter the value1: 10
Enter the value2:0
Error Occured ZeroDivisionError
```