Assignment No-10

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

Solution: The role of a try and exception block is to handle potential errors or exceptions that can occur within a specific section of code.

The try block is used to enclose the code that may throw an exception. When an exception is thrown, the execution immediately jumps out of the try block and into the corresponding catch block.

The catch block defines the specific exception(s) that can be caught and the code to be executed when that exception occurs. It allows the program to gracefully handle the exception by providing an alternative flow of execution or error-handling mechanism.

By using try and exception blocks, developers can anticipate and handle errors in their code, improving program stability and preventing crashes or unexpected behaviors.

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

Solution: The syntax for a basic try-except block in Python is as follows:

“
```
try:
   # Code that might raise an exception
   # ...
except ExceptionType:
   # Code to handle the exception
   # ...
```
“

For example, consider the following code that attempts to divide two numbers:

“
```
try:
   result = 10 / 0
except ZeroDivisionError:
   print("Error: division by zero occurred")
```
“

In this example, the code within the try block attempts to divide 10 by 0, which raises a ZeroDivisionError. The except block catches this exception and prints an error message.

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

Solution: If an exception occurs inside a try block and there is no matching except block to handle that exception, the program will terminate, and the default exception handler will be activated. This default handler typically prints a traceback to the console, which includes information about the error and the line where it occurred.

4. What is the difference between using a bare except block and specifying a specific exception type?

Solution: Using a bare except block means that any exception that occurs will be caught and handled by that block. This includes both expected exceptions that you're explicitly trying to handle and unexpected exceptions that you may not have anticipated.
On the other hand, specifying a specific exception type means that only exceptions of that type will be caught and handled by that block. This allows you to selectively catch and handle specific exceptions while letting others propagate up the call stack.
Using a bare except block is generally discouraged because it can hide errors and complicate debugging and fixing issues in your code. It's usually better to explicitly handle the exceptions you expect to occur while letting unexpected exceptions propagate up to a higher level where they can be dealt with appropriately.

5. Can you have nested try-except blocks in Python? If yes, then give an example.
Solution: Yes, nested try-except blocks can be used in Python to handle different types of exceptions in different levels of code.
Here's an example:
“
```
try:
    # Outer try block
    x = 10
    y = 0
    try:
        # Inner try block
        result = x / y
        print('Result:', result)
    except ZeroDivisionError:
        print('Error: Division by zero')
except:
    print(‘Error: Something went wrong’)
```
“
In this example, we have an outer try-except block that handles any potential exceptions that may occur within it. Inside this outer block is an inner try-except block that handles the ‘ZeroDivisionError’ that can be raised when dividing ‘x’ by ‘y’. If this exception occurs, the inner except block executes, printing an error message. If any other exception occurs within the outer block, the outer except block executes and prints a generic error message.

6. Can we use multiple exception blocks, if yes, then give an example.
Solution: We can use multiple exception blocks in a try-catch statement. Each exception block can handle a specific type of exception.

Here's an example:

“
```
try:
    # Your code that may raise exceptions
    num1 = int(input("Enter a number: "))
    num2 = int(input("Enter another number: "))

    result = num1 / num2

    print("The result is:", result)

except ValueError:
    print("Invalid input. Please enter a valid number.")

except ZeroDivisionError:
    print("Division by zero is not allowed.")
```
“

In the above code, we have two exception blocks. The first block, ‘except ValueError’, will handle the case when the user inputs a non-integer value. The second block, ‘except ZeroDivisionError’, will handle the case when the user inputs 0 as the second number, resulting in a division by zero error.

If either of these exceptions is raised in the try block, the corresponding exception block will be executed.

7. Write the reason due to which the following errors are raised:
a. EOF Error
b. Floating Point Error
c. Index Error
d. Memory Error
e. Overflow Error
f. Tab Error
g. Value Error

Solution: 1) EOF Error: This error is raised when the input() function reaches the end of the file or when there is no input available to read.

2) Floating Point Error: This error is raised when a floating-point calculation results in an error or undefined result, such as division by zero or a value too large to represent.

3) Index Error: This error is raised when trying to access a non-existent index or element in a sequence like a list or string.

4) Memory Error: This error is raised when the program runs out of available memory to allocate for the execution.

5) Overflow Error: This error is raised when a calculation exceeds the maximum limit of a numeric type, such as when an integer becomes too large to be represented.

6) Tab Error: This error is raised when the indentation in Python code is inconsistent, such as mixing tabs and spaces or using an incorrect number of spaces for indentation.

7) Value Error: This error is raised when a function receives an argument of an inappropriate type, or an invalid value is passed to a function.

8. Write code for the following scenario and add a try-exception block.
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

Solution: a) Program to divide two numbers:

“
```
    a = float(input("Enter the first number: "))
    b = float(input("Enter the second number: "))
    result = a / b
    print("The result of division is:", result)

except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")

except Exception as e:
    print("An error occurred:", e)
```
“

b) Program to convert a string to an integer:

“
```
    string_num = input("Enter a number: ")
    num = int(string_num)
    print("The converted number is:", num)
except ValueError:
    print("Error: Invalid input. Please enter a valid integer.")
except Exception as e:
    print("An error occurred:", e)
```
“

c) Program to access an element in a list:

“
```
    my_list = [1, 2, 3, 4, 5]
    index = int(input("Enter the index of the element you want to access: "))
    element = my_list[index]
    print("The element at index", index, "is:", element)

except IndexError:
    print("Error: Index out of range. Please enter a valid index.")
except Exception as e:
```
“

d) Program to handle a specific exception:

```
“
    a = 10
    b = "2"
    result = a / int(b)
    print("The result of division is:", result)
except ValueError:
    print("Error: Invalid input. Please enter a valid integer.")
   except Exception as e:
    print("An error occurred:", e)
“
```

e) Program to handle any exception:

```
“
try:
    a = 10
    b = 0
        result = a / b
    print("The result of division is:", result)
except Exception as e:
    print("An error occurred:", e)
“
```