**17 th June ’23 Assignment ‘.**

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

Ans-

A try block identifies a block of code that may cause an error and the except block catches the error and provides a solution or alternative code to run.

In coded form =

example of try and except blocks in Python:

try:

# code that may cause an error

except:

# code to run if there's an error

You can also specify the type of error you want to catch:

try:

# code that may cause an error

except ValueError:

# code to run if there's a ValueError

except TypeError:

# code to run if there's a TypeError

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

Ans-

The try block is used to identify a block of code that may cause an error. The code inside the try block is executed first. If there is no error, then the except block is skipped. However, if an error occurs, the code inside the try block is stopped and the code inside the except block is executed instead. The except block provides a solution or alternative code to run if there is an error.

You can also specify the type of error you want to catch by specifying the error type after the except keyword. If the error type matches the error that occurred in the try block, then the code inside that except block is executed. If the error type does not match, then the except block is skipped and the program continues to the next except block or the rest of the program.

In code =

basic syntax for a try-except block in Python:

try:

# code that may cause an error

except:

# code to run if there's an error

You can also specify the type of error you want to catch:

try:

# code that may cause an error

except ValueError:

# code to run if there's a ValueError

except TypeError:

# code to run if there's a TypeError

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

**except block?**

Ans-

If an exception occurs inside a try block and there is no matching except block, then the program will terminate and an error message will be displayed. The error message will show the type of error that occurred and the line number where the error occurred. This is because the program doesn't know how to handle the error and there is no code to run to provide a solution or alternative code.

To prevent the program from terminating, it's important to include at least one except block to catch any errors that may occur.

In code=

example of what happens when an exception occurs inside a try block and there is no matching except block:

try:

# code that may cause an error

except ValueError:

# code to run if there's a ValueError

If an exception occurs in the try block and it's not a ValueError, the program will terminate and display an error message. For example:

try:

x = 10 / 0

except ValueError:

print("This code won't be executed")

This code will raise a ZeroDivisionError because you can't divide by zero. Since there's no matching except block for a ZeroDivisionError, the program will terminate and display an error message.

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

Ans-

The difference between using a bare except block and specifying a specific exception type is that a bare except block catches all types of exceptions, whereas specifying a specific exception type only catches that specific type of exception.

Using a bare except block can be dangerous because it catches all types of exceptions, including unexpected ones that you may not have anticipated. This can make it difficult to debug your code because you don't know what caused the error. It's better to specify the type of exception you want to catch so that you can provide a specific solution or alternative code.

On the other hand, specifying a specific exception type can make your code more robust because you know exactly what type of exception you're handling. This can make it easier to debug your code because you can identify the source of the error more easily. However, it's important to make sure you're catching all the types of exceptions that may occur so that you can provide a complete solution or alternative code

In code =

example of using a bare except block:

try:

# code that may cause an error

except:

# code to run if there's an error

This code will catch any type of exception that occurs in the try block. However, it's not recommended to use a bare except block because it can make it difficult to debug your code.

Here's an example of specifying a specific exception type:

try:

# code that may cause a ValueError

except ValueError:

# code to run if there's a ValueError

This code will only catch a ValueError that occurs in the try block. If another type of exception occurs, the program will terminate and display an error message. This makes it easier to debug your code because you know exactly what type of exception you're handling.

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

Ans-

Yes, you can have nested try-except blocks in Python. Here's an example:

try:

# code that may cause an error

try:

# code that may cause another error

except AnotherError:

# code to run if there's another error

except:

# code to run if there's an error

In this example, there are two try-except blocks. The inner try block may cause another error, which is caught by the inner except block. If there's no error in the inner try block, the code continues to run until it reaches the outer except block. If there's an error in the outer try block, the code skips the inner try block and goes straight to the outer except block.

It's important to note that you should only use nested try-except blocks when necessary. If possible, it's better to use a single try-except block to avoid making your code too complex.

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

Ans-

Yes, you can use multiple except blocks in Python. Here's an example:

try:

# code that may cause an error

except ValueError:

# code to run if there's a ValueError

except TypeError:

# code to run if there's a TypeError

except:

# code to run if there's any other type of error

In this example, there are three except blocks. The first except block is responsible for handling a ValueError, the second except block is responsible for handling a TypeError, and the third except block is responsible for handling any other type of error that may occur in the code.

When an error occurs in the code, Python checks each except block in turn to see if it matches the type of error that occurred. If it finds a match, it executes the code inside that except block. If it doesn't find a match, it executes the code inside the last except block.

It's important to note that you should only use multiple except blocks when necessary. If possible, it's better to use a single except block to avoid making your code too complex.

**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**

Ans-

a. EOFError: This error is raised when the input() function reaches the end of a file or input stream.

b. FloatingPointError: This error is raised when a floating point operation fails. This can happen if you try to divide by zero or if the result of a calculation is too large or too small to be represented as a float.

c. IndexError: This error is raised when you try to access an index that is out of range for a sequence (e.g. a list or a tuple).

d. MemoryError: This error is raised when a program runs out of memory. This can happen if you try to allocate too much memory or if your program has a memory leak.

e. OverflowError: This error is raised when a calculation produces a number that is too large to be represented. This can happen with integer calculations or with floating point calculations.

f. TabError: This error is raised when you mix tabs and spaces in your code. Python uses whitespace to delimit blocks of code, and mixing tabs and spaces can cause indentation errors.

g. ValueError: This error is raised when a function receives an argument that has the correct type but an inappropriate value. For example, if you try to convert the string "hello" to an integer using the int() function, you will get a ValueError.

**8. Write code for the following given scenario and add try-exception block to it.**

**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**

Ans-

a. Program to divide two numbers:

try:

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

result = num1 / num2

print("The result is:", result)

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

b. Program to convert a string to an integer:

try:

num\_str = input("Enter a number: ")

num = int(num\_str)

print("The number is:", num)

except ValueError:

print("Error: Invalid input. Please enter a number.")

c. Program to access an element in a list:

try:

my\_list = [1, 2, 3, 4, 5]

index = int(input("Enter an index: "))

print("The value at index", index, "is:", my\_list[index])

except IndexError:

print("Error: Index out of range.")

d. Program to handle a specific exception:

try:

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

result = num1 / num2

print("The result is:", result)

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

except ValueError:

print("Error: Invalid input. Please enter a number.")

e. Program to handle any exception:

try:

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

result = num1 / num2

print("The result is:", result)

except Exception as e:

print("Error:", e)

In this example, the except block catches any exception that occurs and prints the error message. The "as e" part of the except statement assigns the exception object to the variable e, which can be used to get more information about the error.