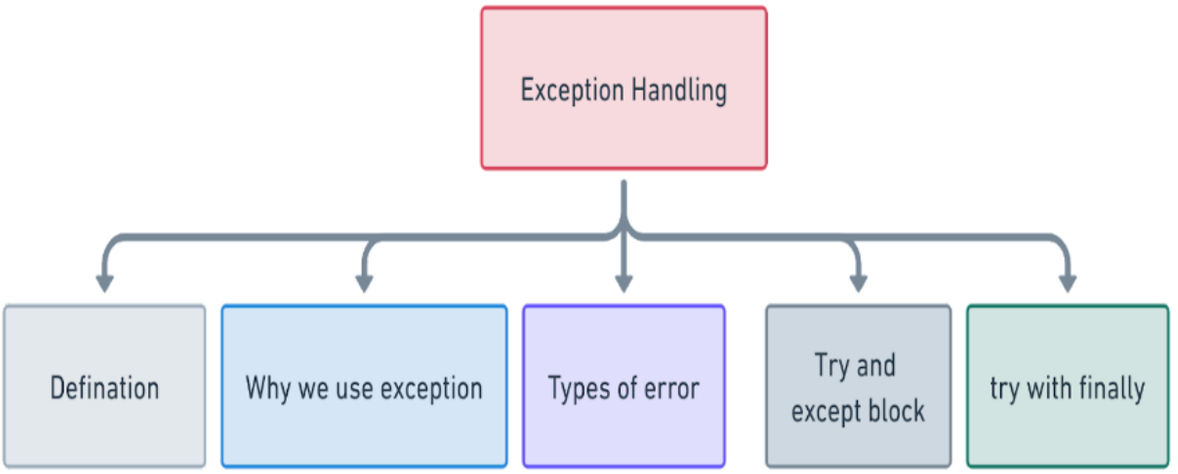
PW SKILLS

Lesson Plan

Exception Handling Using Try Catch Block

Topics to be covered :

- Exception Handling Using Try Catch Block



Exception :

An exception is an event that occurs during programme execution that disrupts the regular flow of execution (for example, when a key is not found in a dictionary). A Python object that reflects an error is known as an exception.

In Python, an exception is an object derived from the BaseException class that includes information about a method error occurrence.

Below are some standard exceptions.

- FileNotFoundException
- ImportError
- RuntimeError
- TypeError

Why we use Exception.

- Standard error handling
- Cleaner code
- Robust application
- Exception propagation
- Differerent error types

Why we use Exception.

An error, on the other hand, is defined as an incorrect or inaccurate activity. Consider a syntax error. As a result, the programme fails to execute.

Types of errors:

- Syntax error
- Logical error

Try and except block to handling exceptions.

When an exception occurs, Python stops the program and generates an exception message. Handling exceptions is strongly advised. Risky code is questionable code that may throw an exception.

To handle exceptions, we must utilise the try and except blocks. Define dangerous code that can throw an exception inside the try block and related handling code inside the except block.

Syntax:

```
try:
 statements in try block
except:
 executed when exception occurred in try block
```

Ex:

```
[2] try:
      a = 10
      b = 0
      c = a/b
      print('The answer of a divide by b: ',c)
    except:
      print("Can't divide with zero.")

    Can't divide with zero.
```

Catching Specific Exceptions.

We can also catch a particular exception. In the preceding example, no specific exception was mentioned in the unless block. It is not good programming practise to catch and handle all exceptions.

It is best to provide a specific exception that the except clause should catch. To catch an exception that occurs when a user submits a non-numerical value instead of a number, for example, we can only catch the built-in ValueError exception, which will handle such an occurrence properly.

Ex:

```
[1] try:
        a = int(input("Enter value of a:"))
        b = int(input("Enter value of b:"))
        c = a/b
        print("The answer of a divide by b:", c)
    except ValueError:
        print("Entered value is wrong")
    except ZeroDivisionError:
        print("Can't divide by zero")

    Enter value of a:ten
    Entered value is wrong
```

Handle multiple exceptions with a single exception block.

A single except clause can handle many exceptions. In an except clause, we can use a tuple of values to define several exceptions.

```
[3] try:
        a = int(input("Enter value of a:"))
        b = int(input("Enter value of b:"))
        c = a / b
        print("The answer of a divide by b:", c)
    except(ValueError, ZeroDivisionError):
        print("Please enter a valid value")
```

```
Enter value of a:2
Enter value of b:0
Please enter a valid value
```

Using try with finally :

The finally block in Python is used in conjunction with the try block statement. The finally block is used to write code that must be executed regardless of whether the try block generates an error.

Syntax :

try :
#Block of code
finally:
#this will always be executed

Use try with else:

Use the else statement in conjunction with If you want to run a specific code only if an exception is not produced, use the try block to check if the try block was executed without an exception.

Syntax:

try:
#Code
else:

#This code executes when exceptions not occurred

Real-Time Analogy

Example-1

```
# Real-time analogy of try and exception
# Division by Zero
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print("Error:", e)
Output:
Error: division by zero
```

Example-2

```
# File Handling - Opening a non-existent file
try:
    file = open('nonexistent.txt', 'r')
    content = file.read()
    file.close()
except FileNotFoundError as e:
    print("Error:", e)
Output:
Error: [Errno 2] No such file or directory: 'nonexistent.txt'
```

Example-3

```
# Input Validation - Handling invalid input type
try:
    num = int(input("Enter a number: "))
except ValueError as e:

 print("Error:", e)
# Input Validation - Handling invalid input type
try:
    num = int(input("Enter a number: "))
except ValueError as e:
    print("Error:", e)
Output:
Enter a number:
Error: invalid literal for int() with base 10: ''
```

Example-4

```
# Dictionary Access - Handling key error
my_dict = {'key1': 100, 'key2': 200}
try:
    value = my_dict['key3']
except KeyError as e:
    print("Error:", e)
Output:
Error: 'key3'
```

Example-5

```
# API Request - Handling connection errors
import requests

try:
    response = requests.get('https://nonexistent-api.com')
    data = response.json()
except requests.ConnectionError as e:
    print("Error:", e)

Output:
Error: HTTPSConnectionPool(host='nonexistent-api.com', port=443): Max retries
exceeded with url: / (Caused by
NameResolutionError("<urllib3.connection.HTTPSConnection object at
0x7a1b4f7323e0>: Failed to resolve 'nonexistent-api.com' ([Errno -2] Name or
service not known)"))
```