# Assignment No. 10

Submitted By: Sachin Dodake

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

**Ans-1**

1. In Python, errors and exceptions can occur at runtime due to a variety of reasons, such as invalid input, network errors, file system errors, and so on. When such errors occur, they can cause the program to crash or behave unpredictably, which is not desirable. This is where Try Except comes in.
2. Try-except is a construct in Python that allows you to catch and handle exceptions that occur during the execution of your program. By wrapping potentially error-prone code in a try block, you can catch and handle any exceptions that occur in a graceful and controlled manner, instead of allowing them to propagate and cause your program to crash.

    i) Error handling: catch and handle exceptions in a controlled way, preventing your program from crashing and providing a more user-friendly experience for your users
    ii) Robustness: make your programs more resilient to errors by handling exceptions gracefully
    iii) Debugging: provide more detailed and helpful error messages that aid in debugging and troubleshooting
    iv) Testing: Intentionally trigger exceptions in your code to test and ensure that it behaves as expected under different error conditions

**`Syntax of Try-Except Block:`**

```
try:
        # There can be errors in this block
except <error type>:
        # Do this to handle exception;
        # executed if the try block throws an error
else:
        # Do this if try block executes successfully without errors
finally:
        # This block is always executed
```

- The **`try block`** is the block of statements you'd like to try executing. However, there may be runtime errors due to an exception, and this block may fail to work as intended.
- The **`except block`** is triggered when the **`try block`** fails due to an exception. It contains a set of statements that often give you some context on what went wrong inside the try block.
- You should always mention the type of error that you intend to catch as exception inside the **`except block`**, denoted by the placeholder **`error type`** in the above snippet.
- You might as well use **`except`** without specifying the **`error type`**. But, this is not a recommended practice as you're not accounting for the different types of errors that can occur.

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

**Ans-2**

- The basic syntax of Try-except block is as below,

```
try:
        # There can be errors in this block
except <error type>:
        # Do this to handle exception;
        # executed if the try block throws an error
else:
        # Do this if try block executes successfully without errors
finally:
        # This block is always executed
```

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

**Ans-3**

- If an **`exception occurs`** during **`execution of the try clause`**, the rest of the clause is skipped. Then, if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try/except block.
- If an **`exception occurs`** which **`does not match the exception`** named in the except clause, it is passed on to outer try statements; if no handler is found, it is an unhandled exception and execution stops with a message as shown above

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

**Ans-4**

**`Bare Except Block:`**

- A bare except clause will catch SystemExit and KeyboardInterrupt exceptions, making it harder to interrupt a program with Control-C, and can disguise other problems.
- If you want to catch all exceptions that signal program errors, use except Exception (bare except is equivalent to except BaseException).
- The use of bare except catches all exceptions indiscriminately – even the ones that you didn't expect and can crash your program (e.g. SystemExit or KeyboardInterrupt.
- **Example:** The following code has a bare except: clause

In [ ]:
```
try:
    user = User.objects.get(pk=user_id)
    user.send_mail('Hello World')
except:
    logger.error('An error occurred!')
```

**`Specific Exception Block:`**

- Catching every exception could cause our application to fail without us really knowing why. This is a horrible idea as it makes our code harder to maintain as bugs can be hidden behind an unspecific exception handler.
- Instead of catching all exceptions with bare except, catch only the specific exceptions you expect to occur. Using our example above, we can introduce FileTypeError and UserPermissionError just to name a few.
- This way, you can handle the problems you know about, and let all the other exceptions bubble up to a higher-level error-handling mechanism.
- **Example:**

In [ ]:
```
try:
    user = User.objects.get(pk=user_id)
    user.send_mail('Hello World')
except User.DoesNotExist:
    logger.error('The user does not exist with that ID')
```

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

**Ans-5**

- In python nested try except finally blocks are allowed. We can take try-except-finally blocks inside try block. We can take try-except-finally blocks inside except block. We can take try-except-finally blocks inside finally block.
- In this case, if an exception is raised in the nested try block, the nested except block is used to handle it. In case the nested except is not able to handle it, the outer except blocks are used to handle the exception.
- **Example:**

In [8]:
```
# Exaplanation on try-except block

x = 10
y = 0

try:
    print("OUTER TRY BLOCK")
    try:
        print("INNER TRY BLOCK")
        print(x / y)
    except TypeError as te:
        print("INNER EXCEPT BLOCK")
        print(te)
except ZeroDivisionError as ze:
    print("OUTER EXCEPT BLOCK")
    print(ze)
```

```
OUTER TRY BLOCK
INNER TRY BLOCK
OUTER EXCEPT BLOCK
division by zero
```

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

**Ans-6**

- Yes, we can catch multiple exceptions in separate except blocks. The try block can have multiple except blocks, each handling a different type of exception.
- The interpreter will execute the first except block whose exception matches the exception that was raised. If no matching except block is found, the interpreter will continue searching for an exception handler in the calling function or script.
- **Example:**

In [10]:
```
# Python catch multiple exceptions
try:
    # defining variables
    a = 10
    b= 0
    c = "abc"
    # adding the variables
    d =a+c

except ZeroDivisionError:                   # ZeroDivision ZeroDivisionError
    print("Zero Division Error occurs")     # printing error

except IndexError:                          # Index IndexError
    print("Index error occurs")             # printing error

except TypeError:                           # type error
    print("Type error occurs")              # printing TypeError
```

```
Type error occurs
```

## 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-7**

**`a) EOFError:`**

- EOFError is raised when the input() function hits the condition of end-of-file.

**`b) FloatingPointError:`**

- FloatingPointError is raised when a floating-point operation fails.

**`c) IndexError:`**

- As the name suggests, the IndexError is raised when the wrong index of a list is used.

**`d) MemoryError:`**

- MemoryError is raised when a program runs out of memory.

**`e) OverflowError:`**

- OverflowError is raised when the result of an arithmetic operation is too large.

**`f) TabError:`**

- TabError is raised when interpreter detects internal error.

**`g) ValueError:`**

- ValueError is raised when a function is given the correct type of argument but with an improper value.

## 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-8**

**`a. Program to divide two numbers`**

In [15]:
```
# Program to divide two numbers

try:
    num1 = int(input("Enter First Number: "))
    num2 = int(input("Enter Second Number: "))

    result = num1 / num2

    print(result)
except ValueError as e:
    print("Please enter th valid integer number.")
except ZeroDivisionError as msg:
    print(f"\tError Message : {msg}")
```

```
Enter First Number: 35
Enter Second Number: 0
        Error Message : division by zero
```

**`b. Program to convert a string to an integer`**

In [19]:
```
string_num = "12"

try:
    int_num = int(string_num)
    print(int_num)
except ValueError:
    print("Catch Error")
```

```
12
```

**`c. Program to access an element in a list`**

In [23]:
```
a = [1111, 2222, 3333, 4444, 5555]
try:
    #looping through the elements of the array a, choosing a range that goes beyond the length of the array
    for i in range( 5 ):
        print( f"The '{i}' th index  element of a given list is :  {a[i]}" )
#if an error occurs in the try block, then except block will be executed by the Python interpreter
except :
    print ("Index out of range")
```

```
The '0' th index  element of a given list is :  1111
The '1' th index  element of a given list is :  2222
The '2' th index  element of a given list is :  3333
The '3' th index  element of a given list is :  4444
The '4' th index  element of a given list is :  5555
```

**`d. Program to handle a specific exception`**

In [28]:
```
# Handling specific exception

try:
    age = int(input('Enter your age: '))
except ValueError:
    print('Please enter the valid age.')
else:
    if age >= 21 &  age <= 99:
        print('Congratulations! Your account has been successfully created in SHAADI.COM.')
    else:
        print('Sorry ! You are not eligible to register in SHAADI.COM.')
```

```
Enter your age: 45
Congratulations! Your account has been successfully created in SHAADI.COM.
```

> **e. Program to handle any exception**

In [32]:
```
try:
    even_numbers = [1,2,4,6,8,10]
    print(even_numbers[5])

except ZeroDivisionError:
    print("Denominator cannot be 0.")

except IndexError:
    print("Index Out of Bound.")
```

```
10
```