Assignment 06 Solutions

Q1. Describe three applications for exception processing.

Ans: Exception Processing is important to find exceptions that causes the runtime error as runtime errors halt the program execution when exception occurs.

Exception Processing is used in Various Applications. Few examples are:

1. Checking Appropriate use of input in an application
2. Checking for Arithmetic exceptions in mathematical executions
3. Checking File I/O exceptions during File handling

In [2]:

```
#we can alwaws put our code in try expect blocks to handle exceptions
try:
    a = int(input("Enter a:"))
    b = int(input("Enter b:"))
    c = a/b
    print("a/b = %d",c)
    # Using exception object with the except statement
except Exception as e:  #general statement to handle different exceptions
    print("can't divide by zero")
    print(e)
else:
    print("Hi I am else block")
finally:
    print("Hi I am finally block")
```

```
Enter a:6
Enter b:0
can't divide by zero
division by zero
Hi I am finally block
```

Q2. What happens if you don't do something extra to treat an exception?

Ans: When an exception occurred, if you don't handle it, the program terminates abruptly and the code past the line that caused the exception will not get executed. Because of not handling exceptions while deployment of end products users can face issues at their end while using the product.

Q3. What are your options for recovering from an exception in your script?

Ans: Exception handling and logging the exceptions are basic ideas that a programmer should follow. Also writing the code in modular fashion helps in easily fidong the errors in end products and logging make it easier for deelopers to resolve the issue at their end. Python provides **`try`** and **`except`** statements for recovering from an exception in our script.

In [3]:

```
try:
  print(x)
except:
  print("An exception occurred")
```

```
An exception occurred
```

Q4. Describe two methods for triggering exceptions in your script ?

Ans: **`raise`** and **`assert`** are two methods that can be used to trigger manual exceptions in our script.

- **`raise`** method triggers an exception if condition provided to it turns out to be True.
- **`assert`** will let the program to continue execution if condition provided to it turns out to be True else exception will be raised

In [4]:

```
# Example of raise
x = 101
raise Exception(f'X Value Should not exceed 5 The Provided Value of X is {x}')

```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15284/3193836954.py in <module>
      1 # Example of raise
      2 x = 101
----> 3 raise Exception(f'X Value Should not exceed 5 The Provided Value of X is {x}')

Exception: X Value Should not exceed 5 The Provided Value of X is 101
```

In [5]:

```
# Example of assert
assert(2==10), "2 is not equal to 4"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15284/2640526208.py in <module>
      1 # Example of assert
----> 2 assert(2==10), "2 is not equal to 4"

AssertionError: 2 is not equal to 4
```

Q5. Identify two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists.

Ans: Python Provides **`else`** and **`finally`** blocks for specifying actions to be executed at termination time, regardless of whether an exceptions exists or not.