EXCEPTION HANDLING : ¶

Exception is an abnormal condition which affects the normal flow of program, when occurs.

An exception, in programming, is an unplanned event, such as invalid input or a loss of connectivity, that occurs while a program is executing and disrupts the flow of its instructions. Exception is a short way of saying exceptional event.

Exceptions in Python :

In Python, exceptions can be handled using a try statement.

A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.

It is up to us, what operations we perform once we have caught the exception.

In [1]:

```
# import module sys to get the type of exception
import sys

randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        print("Next entry.")
        print()
print("The reciprocal of",entry,"is",r)
```

```
The entry is a
Oops! <class 'ValueError'> occured.
Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError'> occured.
Next entry.

The entry is 2
The reciprocal of 2 is 0.5
```

Cathing specific exceptions in Python :

In the above example, we did not mention any exception in the except clause.

This is not a good programming practice as it will catch all exceptions and handle every case in the same way. We can specify which exceptions an except clause will catch.

A try clause can have any number of except clause to handle them differently but only one will be executed in case an exception occurs.

We can use a tuple of values to specify multiple exceptions in an except clause.

In [2]:

```
try:
   # do something
   pass

except ValueError:
   # handle ValueError exception
   pass

except (TypeError, ZeroDivisionError):
   # handle multiple exceptions
   # TypeError and ZeroDivisionError
   pass

except:
   # handle all other exceptions
   pass
```

Raising Exceptions :

In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword **raise**.

We can also optionally pass in value to the exception to clarify why that exception was raised.

In [3]:

```
raise KeyboardInterrupt
```

```
----------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
 last)
<ipython-input-3-7d145351408f> in <module>()
----> 1 raise KeyboardInterrupt

KeyboardInterrupt:
```

In [4]:

```
raise MemoryError("This is an argument")
```

```
---------------------------------------------------------------------------
-----
MemoryError                               Traceback (most recent call
 last)
<ipython-input-4-3bee564e94ad> in <module>()
----> 1 raise MemoryError("This is an argument")

MemoryError: This is an argument
```

In [7]:

```
try:
    a = int(input("Enter a positive integer: "))
    if a <= 0:
        raise ValueError("That is not a positive number!")
except ValueError as ve:
    print(ve)
```

```
Enter a positive integer: -10
That is not a positive number!
```

try...finally :

The **try** statement in Python can have an optional **finally** clause. This clause is executed no matter what, and is generally used to release external resources.

In [8]:

```
try:
   f = open("test.txt",encoding = 'utf-8')
   # perform file operations
finally:
   f.close()
```

```
----------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
 last)
<ipython-input-8-5a8f24f64426> in <module>()
      1 try:
----> 2    f = open("test.txt",encoding = 'utf-8')
      3    # perform file operations

FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'

During handling of the above exception, another exception occurred:

NameError                                 Traceback (most recent call
 last)
<ipython-input-8-5a8f24f64426> in <module>()
      3    # perform file operations
      4 finally:
----> 5    f.close()

NameError: name 'f' is not defined
```

This type of construct makes sure the file is closed even if an exception occurs.

In [ ]: