Exceptional Handling

- Used to handle any unexpected error in Python programs
- Few Standard Exceptions:

Exception Name	Description
Exception	Base class for all exceptions
Arithmetic Error	Base class for all errors that occur for arithmetic calculations
Floating Point Error	Raised when a floating point calculation fails.
Zero Division Error	Raised when division or modulo by zero takes place for all numeric types.
IO Error	Raised when an input / output operation fails, such as print() or open() functions when trying to open a file that does not exist.
Syntax error	Raised when there is a error on Python syntax
Indentation error	Raised when indentation is not specified properly
Value Error	Raised when built-in-function for a data type has a valid type of arguments, but the arguments have invalid values specified
Runtime Error	Raised when a generated error does not fall into any category

Handling an Exception

- Exception is an event, which occurs during the execution of program and disrupts the normal flow of program's instructions.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.
- If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block.
- After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.
- Different ways of Exception Handling in Python are:
  - try....except...else
  - try...except
  - try...finally

Handling an Exception…

- try…except…else
  - A single try statement can have multiple except statements
  - Useful when we have a try block that may throw different types of exceptions
  - Code in else-block executes if the code in the try: block does not raise an exception

Syntax:

```
try:
   You do your operations here;
   ......................
except ExceptionA:
   If there is ExceptionA, then execute this
block.
except ExceptionB:
   If there is ExceptionB, then execute this
block.
   ......................
else:
   If there is no exception then execute
this block
```

Example:

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file")
except IOError:
   print ("Error: can\'t find file or read data")
else:
   print ("Written content to file successfully")
   fh.close()
```

Try to open the same file when you do not have write permission, it raises an exception

Output

Written content to file successfully

- try…except..
  - Catches all exceptions that occur
  - It is not considered as good programming practice though it catches all exceptions as it does help the programmer in identifying the root cause of the problem that may occur.

Syntax:

```
try:
You do your operations here;
   ......................
except ExceptionA:
   If there is ExceptionA, then execute this block.
except ExceptionB:
   If there is ExceptionB, then execute this block.
   ......................
```

Example:

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print ("Error: can\'t find file or write data")
```

Try to write to the file when you do not have write permission, it raises an exception

Output

```
Error: can't find file or write data
```

- try…finally..
  - finally block is a place to put any code that must execute irrespective of try-block raised an exception or not.
  - else block can be used with finally block

Syntax:

```
try:
   You do your operations here;
   ......................
   Due to any exception, this may be
skipped.
finally:
   This would always be executed.
   ......................
```

Try to write to the file when you do not have write permission

Example:

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception
handling!!")
finally:
   print("Error: can\'t find file or write data")
```

Output

```
Error: can't find file or write data
```

Multiple except blocks

```
try:
    statement(s)
except Exception1:
    statement
except Exception2:
    statement
except Exception3:
    statement
```

Multiple Exceptions in a Single except block

```
try:
    n=int(input('Enter the number'))
    print(n**2)
except (KeyboardInterrupt, ValueError,TypeError):
    print("Please Check before you enter")
print("Bye")
```

1. ZeroDivisionError

```
a=int(input('Enter the number'))
b=int(input('Enter the number'))
try:
    c=a/b
     print(c)
except ZeroDivisionError as e:
    print("Please Check Denominator ",e)
print("Bye")
```

2. ValueError

```
try:
    n=int(input('Enter the number'))
    print(n**2)
except ValueError as e:
     print("Please Check before you enter ",e)
print("Bye")
```

3.KeyboardInterrupt

```
try:
    n=int(input('Enter the number'))
    print(n**2)
except KeyboardInterrupt as e:
     print("Do not press ctrl+c ",e)
print("Bye")
```

4. TypeError

```
try:
    n=int(input('Enter the number'))
    print(n**2+'Hello')
except TypeError as e:
    print("Please Check the datatype in expression",e)
print("Bye")
```

5. FileNotFoundError

```
fname=input("Enter the filename")
try:
    fp=open(fname,'r')
    print(fp.read())
except FileNotFoundError as e:
     print("Please Check before you enter ",e)
print("Bye")
```

6. PermissionError

```
fname=input("Enter the filename")
try:
    fp=open(fname,'w')
    fp.write('piyush')
    fp.close()
except PermissionError as e:
     print("Please Check the Permission of file ",e)
print("Bye")
```

7. IndexError

```
L=[1,2,3,4]
try:
    print(L[9])
except IndexError as e:
    print("Please Check the index ",e)
print("Bye")
```

8.NameError

```
try:
    print(a)
except NameError as e:
     print(“Please Check the variable a ”,e)
print(“Bye”)
```

9. KeyError

```
D={1:1,2:8,3:27,4:64}
try:
    print(D[5])
except KeyError as e:
    print("Please Check the key exists or not ",e)
print("Bye")
```

Raising Exceptions

- We can deliberately raise an exception using the raise keyword
- Syntax

raise [Exception [args]]

Example:

```
try:
    n=10
  print(n)
    raise ValueError
except:
    print("Exception Occurred")
```

Re-raise an Exception

```
try:
    raise NameError
except:
    print("Re-raising the exception")
    raise
```

Handling Exceptions in Invoked Functions

```
def fun(a,b):
    try:
        c=a/b
    except ZeroDivisionError:
        print("You can not divide a no. by zero")

a=int(input("Enter the Number"))
b=int(input("Enter the Number"))
fun(a,b)
```

```
def fun(a, b):
    return a/b

a=int(input("Enter the Number"))
b=int(input("Enter the Number"))
try:
    fun(a,b)
except ZeroDivisionError:
    print("You can not divide a no bye Zero")
```

Programs

1. WAP that prompts the user to enter a number. If the number is positive or zero, print it, otherwise raise an exception ValueError with some message.
2. WAP which infinitely print natural numbers. Raise the StopIteration exception after displaying first 20 numbers to exit from the program.
3. WAP that prompts the user to enter his name. The program then greets the person with his name. But if the person's name is "Rahul", an exception is thrown explicitly and he is asked to quit the program.

Tricky Questions (Code 1)

1. What will be the output?

```
def fun():
    try:
        print("Hello")
        return 1
    finally:
        print("DONE")
fun()
print("Hi")
```

2. What will be the output?

```
def fun():
    for i in range(5):
        try:
            if i==0:
                print("Hello")
                break
        finally:
            print("DONE")
fun()
print("Hi")
```

3. What will be the output?

```
def fun():
    for i in range(5):
        try:
            if i==0:
                print("Hello")
                continue
        finally:
            print("DONE")

fun()
print("Hi")
```