EXCEPTION HANDLING Concept:

========== ==========:-

Exception definition :

- An exception is an event, which occurs during the excution of a program,that disrupts the normal flow of the program instructions
- We can handle the exceptions at runtime. But we cannot handle errors.
- Exceptions are related to application, where as errors are related to environment in which the application is running.

NOTE: Errors detected during the excution are called exceptions .

Error:

- An error is a term which is used to describe any issue that arises unexpectedly that cause a computer do not functioning properly.

Debugging:

- The process of finding and eliminating the errors is called Debugging.

In Python We have 2 types of Errors:

1) SyntaxError

2) RuntimeError

1 Syntax error :

---->> The errors which occurs because of "invalid syntax" are known as "Syntax errors"

------>> Interpreter will check for syntax errors when ever we run the python program

------->> If syntax error is found , Then no byte code is generating.

------->> With out byte code, program excution is not possible.

------>> Developers only responsible to solve these errors.

Example1:

```
def m1():
    print('Hi')    #  It is Ok
```

Example1:

```
def m1():
print('Hi')    #  It is not  Ok
```

Output : **SyntaxError** : Expected an indented block

2) RUNTIME ERRORS :

- The errors which occurs at the time of excution of a program are known as runtime errors.
- We get Runtime Errors because of programing logic, invalid input,memory related issues etc...
- For every RuntimeError , Python is providing corresponding exception class is available.

Example1: **KeyError , IndexError, ValueError, NameError** etc ....

- At the time of execution of a program if any Runtime Error is occur then internally corresponding Runtime Error representation classes object will be created
- NOTE:
- If python program doesn't contain the code to handle that exception object then our program execution will be terminated abnormally.

What is "Abnormal Termination ?

- The concept of terminating the program in the middle of its execution with out executing last step of the program is known as a "Abnormal Termination".
- It is undesirable situation in any programming language.

Example:

```
print('Hi')
i=10
j=0
k = i / j
print(k)
print('bye')
```

Output : **ZeroDivisionError** : division by zero

EXCEPTION HANDLING :

- The concept of identifying the Runtime Error representation class object , which is created at the time of execution of the program, receiving that object and assigning that object to the reference variable of corresponding Runtime Error representation class is known as a "Exception handling".

- Exception handling is used to stop the abnormal termination when ever Runtime Error is occur.
- By using "try --- except" block, we can implement exceptions.
- Python is providing some keywords for handling exceptions.
  For example: **try , except , else , finally , as , raise**

try : block

- The statement's which causes to Runtime Errors or risky code lines we will write in side "try" block.
- At execution time of try - block, if any Runtime Error representation class object is created, then it is identified by try - block, received by try - block and forwrd it to except block, with out executing remaining statements.
- If No errors then except-block not executed.

except: block

- "except" block should be followed by try - block.
- It receives Runtime Error class object which is given by try -block .
- Assign this object to the reference variable of RuntimeError representation class.
- In except block, we can display user friendly messages and error messages related to raised exception.

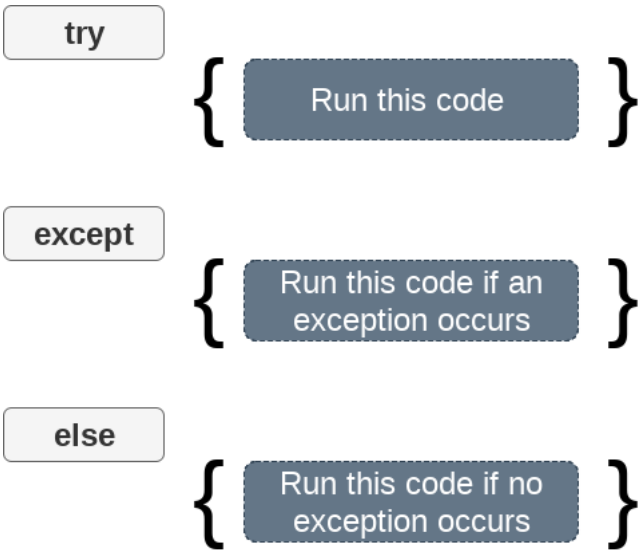
Syntax:

```
try:
    =====
except:
    ======
```

For example:

```
try:
   print(10/0)
except:
     print('Any number can't division by zero')
```

Output: Any number can't division by zero



Single "try" block with multiple named "except" blocks :

- If try block returns Runtime Error , control goes to first except block.
- If first except block not handled raised exception then control goes to second except block.
- If all except blocks not handled raised exceptions then returns "abnormally" terimination.

Default except:

- It handled any type of exception(RuntimeError) here we write common message for exception.
- Default except block should be last except block When using single try - block with multiple excepts block.

Syntax:

```
try :
    =====  =====
except NameError :
   =====  =====
except ValueError :
   =====  =====
except :
   =====  =====
```

Example 1:

```
print('Hi')
try :
   i = eval(input("Enter 1st number :"))
   j = eval(input("Enter 2nd number :"))
   k = i / j
   print(k)
except ZeroDivisionError:
    print('Sorry , you cont do division by zero')
except TypeError:
    print(' Sorry , Both data types must be integers')
except NameError:
    print('Sorry, your eneterd name is not defined')
except Exception:
    print('sorry, some thing wrong')
print('bye')
```

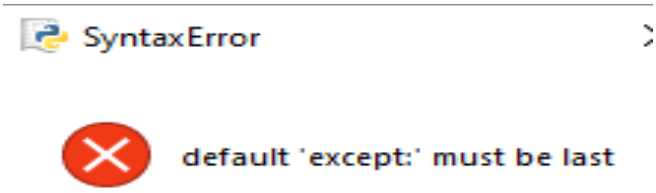
Output:

```
Hi
Enter 1st number :10
Enter 2nd number :a
Sorry, your eneterd name is not defined
bye
```

Eample 2:

```
print('Hi')
try:
   i = 10
   j = 0
   k =  i /  j
   print( k )
except:
   print( 'yes' )
except NameError:
   print('no')
print('bye')
```

Output:



finally: block

- The set of statements which are **compulsary** to execute if exception is comes or not then those statements we can create inside **finally** block.
- Even though exception is occured whether it is handled or not handled , if we want to execute some statements then those statements are recommended to represent in 'finally block' .
- Resource releasing statements [Eg : File closing statements , data base connection objects closing] are recomended to represent in "finally - block".
- For example, **File closing statements** , data base **connection objects closing** statements.
- finally block is followed by either **try** or **except** block.

Syntax 1:

```
try:
   =======
except:
    =======
finaly:
    ======
```

Syntax 2:

```
try:
   ====
finally:
   =====
```

Note: Identify the which blocks combination working

Note: try - except block working

```
try:
      pass
```

```
except:
        pass
```

Note: between try and except block if any other statements then not working

```
try:
     pass
print('hii')
except:
    pass
```

Note: only try block not working

```
try:
     pass
print('hi')
```

Note : try-finally blocks working

```
try:
     pass
finally:
     pass
```

Note: finally block followed by except block is working

```
try:
    pass
except:
     pass
finally:
    pass
```

Note: except block followed by finally block not working

```
try:
      pass
finally:
      pass
```

```
except:
    pass
```
Note: else block followed by except block working
```
try:
    pass
except:
     pass
else:
     pass
```

Note: else block followed by finally block not working
```
try:
    pass
finally:
    pass
else:
    pass
```

Note: try-except-else-finally blocks working
```
try:
   pass
except:
   pass
else:
   pass
finally:
   pass
```

Note: try-except-finally-else block not working
```
try:
     pass
except:
      pass
```

```
finally:
       pass
else:
      pass
```

Note: try - multiple named excepts - default except - else - finally working

```
try:
   pass
except TypeError:
   pass
except ValueError:
   pass
except KeyError:
   pass
except:
   pass
else:
   pass
finally:
   pass
```

Nested try - block :

```
try :
    =====
     try :
          =======
     except :
           ========
except  :
      ====
```