Exception Handling:

- Mostly Error will occur at the time of compilation just because of regular syntax mistakes
- We can the make the program to compile success by making the necessary correction in the syntax mistakes
- In this case client is not getting any problem just because as it in production
- Exception is a typically an "error" it will occur at the time of execution at any point of time.
- If any exceptions are occur then normal flow of the program will get disturbed and program will be terminated abnormally or unsuccessfully

- **Some possible reasons for the Exceptions are**
  - Invalid Number of Inputs
  - Invalid Input types
  - Any logical Error Occur

- **Benefits Of Handling Exceptions are :**
  - Making the program to terminate Successfully
  - Grouping and Supporting the Error Types
  - Separating Error Logic with Normal Logic

- **Default Exception handler is the PVM**
  - During Executing the Python Programs if at any where anything goes wrong , then PVM will recognize the corresponding "ExceptionClassName" Based on the context.
  - It will print Exception Class Name , Description of the Exception
  - TraceBack of the Exception [program name and line number] where the exception is got raised and finally making the program to terminate abnormally or unsuccessfully

Example:

```
print("Hello 1")
print("Hello 2")
print(5/0)
print("Hello 3")

'''
output:
Hello 1
Hello 2
Traceback (most recent call last):
  File "E:/Python_Online9/ADV_PYTHON/EXCEPTION/ExceptionDemo.py", line 3, in <module>
    print(5/0)    #ZeroDivisionError
ZeroDivisionError: division by zero   '''
```

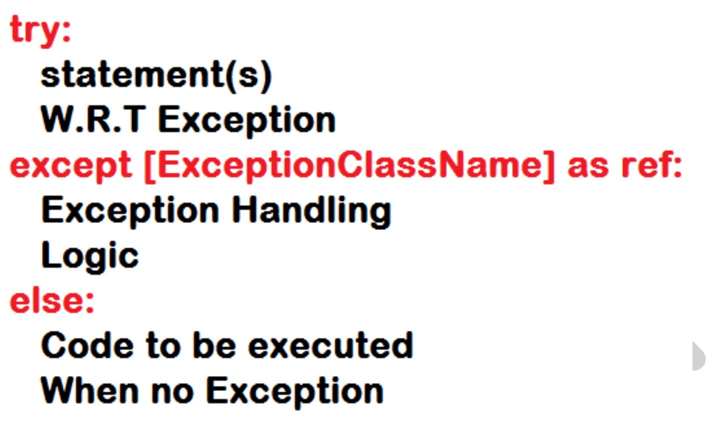
- **Keywords Related to Handle the Exception:**
  - try | except | raise | else | finally

List of possible Exception

BaseException

- **Exception**
  - **ArithmeticError**
    - **FloatingPointError**
    - **OverflowError**
    - **ZeroDivisionError**
  - **AssertionError**
  - **AttributeError**
  - **BufferError**
  - **EOFError**
  - **ImportError**
    - **ModuleNotFoundError**
  - **LookupError**
    - **IndexError**
    - **KeyError**
  - **MemoryError**
  - **NameError**

Functionality of try and except Block:



```
try:
  statement(s)
  W.R.T Exception
except [ExceptionClassName] as ref:
  Exception Handling
  Logic
else:
  Code to be executed
  When no Exception
```

- in the try block is always recommended to write the lines which are possible for only exceptions
- except is the most immediate block to try.
- In the except block is always recommend to write the logic related to handle the Exception | Solutions of the Exception
- During executing the statements of try block if at all any thing goes wrong , then PVM will recognize the corresponding Exception class name based on the context.
- It will create an object of the corresponding "Exception Class" and it will throw to the Exception handler
- The except block the exception handler, it will receive the Exception class object. Handle the Exception and finally making the program to terminate successfully.

Note: The argument of catch block should be same as the Object which thrown by the catch block.

Example :

```
x=int( input("enter a number ") ) #360
y=int( input("enter a number ") )  #0

try:
   z=x/y
except ZeroDivisionError:
   print("Sorry V R N D By Zero....")
else:
   print("Result is : ",z)
```

Example:

```
stu={"sno":101,"sname":"ramesh","scity":"hyd"}
print("Keys : ",stu.keys())

key=input("Enter Key : ")  #sname
try:
   value=stu[key]
except KeyError:
   print("Sorry Key Is Not Existed ....!!!")
else:
   print("Value of the Given key : ",value)
```

Example 3:

```
#      0    1   2   3    4   index values
lst=[10,20,30,40,50]
print("List : ",lst)

pos=int( input("Enter index position ") )  #2
try:
   item=lst[pos]
except IndexError:
   print("sorry Invalid Index ")
else:
   print("Item @ given pos : ",item)
```

If Required we can also write more than one except for a single try block. If any possibilities for multiple exception then it all ways good programming practice to handle every possible exception Separately

Try with multiple Except Blocks:

Example:

```
import sys

try:
   x=int(sys.argv[1])
   y=int(sys.argv[2])
   z=x/y
except IndexError:
   print("Plz Enter min 2 value ")
except ValueError:
   print("Plz Enter an int value ")
except ZeroDivisionError:
   print("Sorry V R N D By Zero...!!!")
else:
   print("Result is : ",z)
```

Note: Based on the application requirements we can also handle more than one exception using single except block, but in this case all the Exception classes should be written in the form of tuple collection.

Example:

```
import sys

try:
   x=int(sys.argv[1])
   y=int(sys.argv[2])
   z=x/y

except (IndexError,ValueError,ZeroDivisionError):
   print("Sorry Unable to continue ....")

else:
   print("Result is : ",z)
```

- **If you want see the reason of the exception or description of the exception then we have to provide ref for the exception classes by using "as" keyword**

Example:

```
import sys

try:
   x=int(sys.argv[1])
   y=int(sys.argv[2])
   z=x/y

except (IndexError,ValueError,ZeroDivisionError) as r:
   print("Sorry Unable to continue ....")
   print("Reason is : ",r)

else:
   print("Result is : ",z)
```

Handling any Exception By a except block:

It is possible by passing "Exception" or by defining only except without Exception ClassNames. as an argument to the except block. This process is also called generic exception handling mechanism or default Exception handling mechanism

Excample:

```
import sys

try:
   x=int(sys.argv[1])
   y=int(sys.argv[2])
   z=x/y
except:
   print("From Except ")
   print("Sorry Unable to process...")
else:
   print("Result is : ",z)
```

Example 2:

```
import sys

try:
   x=int(sys.argv[1])
   y=int(sys.argv[2])
   z=x/y
except Exception as ref:
   print("From Except ")
   print("Sorry Unable to process...")
   print("Reason : ",ref)
else:
   print("Result is : ",z)
```

“raise” keyword

- “raise “ is keyword
- Exceptions are classified into 2 types
  - Predefined Exceptions [Predefined Exception classes]
  - User defined Exceptions [Use defined Exception Classes]
- All the Exceptions are the classes only
- Pre-defined exceptions are the classes which are provided by Python Software. All the predefined Exceptions are well known the PVM thus PVM will take responsibility in [raising the Exception] Based on the context implicitly

- User defined Exception are the classes which are defined by us based on our application requirements , User defined exceptions are unknown to the PVM thus PVM doesn’t cares about invoking the user defined Exception, thus it is our responsibility to raise that exception based the application required | context

- In order raise the user defined exception then we have use the keyword “raise”.

- If required we can raise either predefined Exception or user defined Exception by using “raise” keyword

- **Syn: raise <ExceptionClsName>([list of args])**

Example:
```
age=int(input("enter age : "))
if age>=18:
   print("Eligible For Vote ")
else:
   try:
      raise ZeroDivisionError()  #calling constructor is creating an Object
   except ZeroDivisionError:
      print("from Except ")
      print("Sorry Not Eligible For Vote")
```

User defined Exception:

- The Exception classes which defined by us for our application requirements
- User defined Exception is nothing but defining user defined class
- Steps-1 for defining user defined Exception
    - Define A class that should be extended by either "Exception"
- Step -2:
    - Define a parameterized constructor in order to display the description of user defined Exception
- Step-3:
    - Invoke the user defined Exception based on the context by using keyword "raise"

Example:

```
class MyLoginException(Exception):
   def __init__(self,msg):
      self.msg=msg

uname=input("Enter username : ")
pword=input(" Enter Password : ")

if uname=='sssit' and pword=='kphb':
   print("Valid User ")
else:
   try:
      raise MyLoginException("Invalid Username or Password .... :( ")
   except MyLoginException as m:
      print("Sorry Unable to continue ...")
      print("Reason : ",m)
```

Example 2:

```
import re

class MobileException(Exception):
   def __init__(self,msg):
      self.msg=msg

data=input("Enter mobile number ")
match=re.fullmatch("[6-9]\d{9}",data)
if match is not None:
   print("Valid Mobile number ")
else:
   try:
      raise MobileException("Invalid Mobile Number")
   except MobileException as a:
      print("Sorry Unable to continue ....")
      print("Reason  is : ",a)
```

Finally:

- Finally is an optional block
- Finally block is mostly recommended whenever you want perform some operation compulsorily before terminating the program execution
- In the finally block it is recommended to write the logic related to resource de-allocations. Such closing the files, closing Databases , disconnection networks …
- Finally block will be executed all the cases when an Exception got raised or not .

```
try:
   statement(s)
   W.R.T Exception
except [ExceptionClassName] as ref:
   Exception Handling
   Logic
else:
   Code to be executed
   When no Exception
finally:
   Resource DE-Allocation
   Logic
```

Example On finally:

```
import time

try:
   time.sleep(1)
   print("try-1")
   print("try-2")
   raise IndexError()
except ZeroDivisionError:
   time.sleep(1)
   print("except-4")
   print("except-5")
else:
   time.sleep(1)
   print("else-6")
   print("else-7")
finally:
   time.sleep(1)
   print("finally-8")
   print("finally-9")

time.sleep(1)
print("outside-10")
print("outside-11")
```

IQs:

```
# 1.try: pass  //invalid
# 2.except : pass  //invalid
# 3.else:pass  //invalid
# 4.finally:pass //invalid
'''
5
try: pass
except: pass '''

'''
6
try:pass
finally:pass  '''

'''
7.
try:  pass
else:  pass  invalid '''

'''8
try: pass
except: pass
finally:pass    valid   '''

'''9.
try:pass
except:pass
else:pass     valid '''

'''10
try:pass
except:pass
else:pass
finally:pass '''
```

```
'''11.
try:
   try: pass
   except:pass
except: pass   //valid '''

'''12.
try:
   pass
except:
   try: pass
   except : pass   //valid '''

'''13.
try:
   pass
except:
   pass
else:
   try:pass
   except: pass   //valid '''

'''14.
try:
   pass
except:
   pass
else:
   pass
finally:
   try:
      pass
   except:
      pass  '''

'''15.
try:
   try:
      pass
   except:
      pass   //invalid '''

'''16
try:
   try: pass
   except: pass
finally:
   pass  valid '''

'''17.
try:
   pass
except ZeroDivisionError:
   pass
except IndexError:
   pass    valid '''

'''18.
try:
   try:
      pass
   except:
      pass
except Exception:
   pass    valid '''

'''19.
try:
   pass
except:
   pass
except:
   pass   //invalid default except should be the last '''
```

```
'''20
try:
   pass
except Exception:
   pass
except ZeroDivisionError:
   pass
except IndexError:
   pass     //valid  '''
```