8. Exceptions Handling

1. Introductions

Errors

i) Syntax Errors -

Lack of language Knowledge [Syntax of rule language]

Error due invalid statements or because of grammatical Error of Language

If you have an syntax error your code will not run on any condition

In [4]:
```
if (a>5){
    print(hello);
}
```

```
  File "C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_4308/4244572009.py", line 1
    if (a>5){
            ^
SyntaxError: invalid syntax
```

ii) Exceptions

Run Time Error (execute), Unwanted Error, Logical Errors

Error which does not occur in general test cases but due some logic failure they can trigger / occur in special test cases

whenever Exceptions Occures program terminates immedieatly

Eg:
- ZeroDivisionError
- TypeError
- ValueError
- FileNotFoundError
- EOFError
- SleepingError
- TyrePuncturedError

Division by Zero Exceptions

In [6]:
```
def div_(x, y):
    print(x / y)
```

In [8]:
```
div_(4, 5)
div_(10, 2)
```

```
0.8
5.0
```

In [9]:
```
div_(123, 0) # infy
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_4308/2229025419.py in <module>
----> 1 div_(123, 0) # infy

C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_4308/1080098325.py in div_(x, y)
      1 def div_(x, y):
----> 2     print(x / y)

ZeroDivisionError: division by zero
```

Name Error

In [10]:
```
print(data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_4308/927109222.py in <module>
----> 1 print(data)

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

File Not Found

In [11]:
```
fp = open('xyz')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_4308/1401770933.py in <module>
----> 1 fp = open('xyz')

FileNotFoundError: [Errno 2] No such file or directory: 'xyz'
```

Type Error

In [12]:
```
'5' + 7
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_4308/2933671510.py in <module>
----> 1 '5' + 7

TypeError: can only concatenate str (not "int") to str
```

Value Error

In [42]:
```
a = int(input())
print(a)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_4308/976948877.py in <module>
----> 1 a = int(input())
      2 print(a)

ValueError: invalid literal for int() with base 10: 'jhe;'
```

Exceptions Type

Built-in Exceptions

they automatically trigger whenever some logical problem occures defined by language

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
```

```
|    +-- UnicodeError
|         +-- UnicodeDecodeError
|         +-- UnicodeEncodeError
|         +-- UnicodeTranslateError
+-- Warning
     +-- DeprecationWarning
     +-- PendingDeprecationWarning
     +-- RuntimeWarning
     +-- SyntaxWarning
     +-- UserWarning
     +-- FutureWarning
     +-- ImportWarning
     +-- UnicodeWarning
     +-- BytesWarning
     +-- ResourceWarning
```

Custom Exceptions

We need to Raise them in orders to trigger exception

In [55]:
```
pos_num = int(input())

if pos_num < 0:
    raise ArithmeticError("We need an positive Integer Number")
```

```
---------------------------------------------------------------------------
ArithmeticError                           Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_4308/434463745.py in <module>
      2
      3 if pos_num < 0:
----> 4     raise ArithmeticError("We need an positive Integer Number")

ArithmeticError: We need an positive Integer Number
```

2. Exceptions Handling

It is highly recommended to handle exceptions. The main objective of exception handling is Graceful Termination of the program(i.e we should not block our resources and we should not miss anything)

Every exception in Python is an object. For every exception type the corresponding classes are available. Whevever an exception occurs PVM will create the corresponding exception object and will check for handling code. If handling code is not available then Python interpreter terminates the program abnormally and prints corresponding exception information to the console.The rest of the program won't be executed

In [78]:
```
print("Hello")
print(10/0)
print("Hi")
```

```
Hello
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_4308/1665254487.py in <module>
      1 print("Hello")
```

```
----> 2 print(10/0)
      3 print("Hi")

ZeroDivisionError: division by zero
```

1. We Use try except block for handling exception

```
try:
    stmt-1
    stmt-2
    stmt-3
except XXX:
    stmt-4
stmt-5
```

case-1:

If there is no exception 1,2,3,5 and Normal Termination

case-2:

If an exception raised at stmt-2 and corresponding except block matched 1,4,5 Normal Termination

case-3:

If an exception raised at stmt-2 and corresponding except block not matched 1, Abnormal Termination

case-4:

If an exception raised at stmt-4 or at stmt-5 then it is always abnormal termination.

2. finallly Block

we required some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether exception handled or not handled. Such type of best place is nothing but finally block.

```
try:
    Risky Code
except:
    Handling Code
finally:
    Cleanup code
```

3. else block with try-except-finally:

else block will be executed if and only if there are no exceptions inside try block.

```
try:
    Risky Code
except:
```

```
        will be executed if exception inside try
    else:
        will be executed if there is no exception inside try
    finally:
        will be executed whether exception raised or not raised and handled or not
handled
```

Note:

1. Whenever we are writing try block, compulsory we should write except or finally block.i.e without except or finally block we cannot write try block.
2. Wheneever we are writing except block, compulsory we should write try block. i.e except without try is always invalid.
3. Whenever we are writing finally block, compulsory we should write try block. i.e finally without try is always invalid.
4. We can write multiple except blocks for the same try,but we cannot write multiple finally blocks for the same try
5. Whenever we are writing else block compulsory except block should be there. i.e without except we cannot write else block.
6. In try-except-else-finally order is important.
7. We can define try-except-else-finally inside try,except,else and finally blocks. i.e nesting of try-except-else-finally is always possible.

Example: Division By Zero Error

In [14]:
```
a = int(input("a: "))
b = int(input("b: "))

try:
    r = a / b #ZeroDivisionError
except:
    r = 'infinity'
    print("!!!Error Occured!!!")

print('result', r)
```

```
!!!Error Occured!!!
result infinity
```

In [58]:
```
a = int(input("a: "))
b = int(input("b: "))

try:
    r = a / b #ZeroDivisionError
except ZeroDivisionError as msg:
    r = "Infinity plzz don't give denomenator as zero"
    print(f"!!!Error {msg}")
else:
    print("If no exception occures than else part execute")
finally:
    print("I will always exceute")
    # for clean up actions

print('result', r)
```

```
If no exception occures than else part execute
I will always exceute
result 2.0
```

Example: Value Error

In [39]:
```
try:
    a = int(input())
except:
    a = "a should be an integer"
    print("!!!Error Occured!!!")

print(a)
```

```
!!!Error Occured!!!
a should be an integer
```

4. try with multiple except block

```
try
    -------
    -------
    -------
except ValueError:
    perform alternative
except ZeroDivisionError:
    perform alternative
    arithmetic operations
except FileNotFoundError:
    use local file instead of remote file
```

Note:

```
If try with multiple except blocks available then the order of these except blocks
is important .Python interpreter will always consider from top to bottom until
matched except block identified
```

We can write a single except block that can handle multiple different types of exceptions.

```
    except (Exception1,Exception2,exception3,..): or
    except (Exception1,Exception2,exception3,..) as msg :

Parenthesis are mandatory and this group of exceptions internally considered as
tuple.
```

Default except block:

```
We can use default except block to handle any type of exceptions.
In default except block generally we can print normal error messages.

Syntax:-
        except:
            statements
```

Note:

```
***Note: If try with multiple except blocks available then default except block
should be last,otherwise we will get SyntaxError.
```

Example: Value Error and Division By Zero Error

Case 1:

In [81]:
```
def func():
    try:
        a = int(input("a: "))    # break bcz ValueError
        b = int(input("b: "))
        r = a / b    # ZeroDivisionError
        f = 1
    except ValueError as msg:
        r = "Give input integer numbers"
        print("!!!Error!!! Please Mind Your input a, b need to be integer")
    except ZeroDivisionError as msg:
        r = "Infinity plzz don't give denomenator as zero"
        print(f"!!!Error {msg}")
    finally:
        print('result ',r)
```

In [82]:
```
func()
```

```
!!!Error!!! Please Mind Your input a, b need to be integer
result  Give input integer numbers
```

In [83]:
```
func()
```

```
!!!Error division by zero
result  Infinity plzz don't give denomenator as zero
```

In [84]:
```
func()
```

```
result  2.5
```

Case 2:

In [85]:
```
def func():
    try:
        a = int(input("a: "))    # break bcz ValueError
        b = int(input("b: "))
        r = a / b    # ZeroDivisionError
        f = 1
    except Exception as msg:
        r = 'do you understand ?'
        print("Hello I am Super Error!")
    except ValueError as msg:
        r = "Give input integer numbers"
        print("!!!Error!!! Plz Mind your input, need to be integers")
    except ZerDivisionError as msg:
        r = "infinity"
        print(f"!!!Error!!! {msg}")
    finally:
        print('result', r)
```

In [86]:
```
func()
```

```
Hello I am Super Error!
```

```
result do you understand ?
```

In [87]:
```
func()
```

```
Hello I am Super Error!
result do you understand ?
```

In [88]:
```
func()
```

```
result 2.0
```

Case 3:

In [89]:
```
def func():
    try:
        a = int(input("a: "))
        b = int(input("b: "))
        r = a/b
        f = 1
    except ValueError as msg:
        r = "Give input Integers NUmbers"
        print("!!!Error!!! ", msg)
    except ZeroDivisionError as msg:
        r = 'Infinity'
        print("!!!Error!!! ", msg)
    except Exception as msg:
        r = 'Do You Understand ?'
        print("I'm Super Error !", msg)
    finally:
        print('result ',r)
```

In [90]:
```
func()
```

```
!!!Error!!!  invalid literal for int() with base 10: 'hggh;'
result  Give input Integers NUmbers
```

In [91]:
```
func()
```

```
!!!Error!!!  division by zero
result  Infinity
```

In [92]:
```
func()
```

```
result  1.6666666666666667
```

Case 4:

In [96]:
```
def func():
    try:
        a = int(input("a: "))
        b = int(input("b: "))
        r = a/b
        f = 1
    except (ZeroDivisionError,ValueError) as msg:
        r = msg
```

```
        print("Plz Provide valid numbers only and problem is: ",msg)
    print(r)
```

In [97]:
```
func()
```

```
Plz Provide valid numbers only and problem is:  invalid literal for int() with base 10: 'r
fg;'
invalid literal for int() with base 10: 'rfg;'
```

In [98]:
```
func()
```

```
Plz Provide valid numbers only and problem is:  division by zero
division by zero
```

In [99]:
```
func()
```

```
2.5
```

3. Custom Exception class

In [51]:
```
class A(Exception):
    def __init__(self, arg1, arg2):
        super().__init__()
        self.args = (arg1, arg2)
    def __str__(self):
        return str(self.args)
    def do_this(self):
        print("if error come do this")
    def do_that(self):
        print("also do that")
```

In [52]:
```
try:
    raise A('first', 'second argument')
except A as e:
    print(e)
    e.do_this()
    e.do_that()
except ValueError as e:
    print(e.args)
```

```
('first', 'second argument')
if error come do this
also do that
```

Debugging

In [59]:
```
try:
    pos_num = int(input())
    if pos_num < 0:
        raise ArithmeticError("We need an positive Integer number")
        # we are raise an Error
except Exception as e:
    print(f"!!Error!! {e}")
else:
```

```
    print("If no exception occures than else part execute")
    print("Given value is ", pos_num)
finally:
    print("it will run always exceptions comes or not")
    # for clean up actions
```

```
!!Error!! We need an positive Integer number
it will run always exceptions comes or not
```

In [64]:
```
def func():
    try:
        pos_num = int(input())
        if pos_num < 0:
            raise ArithmeticError("We need an positive Integer number")
            # we are raise an Error
    except ValueError as e:
        print('!!Error!! Plese input only numbers containin 0-9 characters')
    except ArithmeticError as e:
        print(f"!!Arithmetic Error!!{e}")
    except Exception as e:
        print("Unwanted Occured!! Please Contct to Adminstration with following message")
        print(f"!!Error!! {e}")
    else:
        print("If no exception occures than else part executie")
        print("Given value is ", pos_num)
    finally:
        print("it will run always exceptions comes or not")
        # for clean up actions
```

In [67]:
```
func()
```

```
!!Error!! Plese input only numbers containin 0-9 characters
it will run always exceptions comes or not
```

In [65]:
```
func()
```

```
!!Arithmetic Error!!We need an positive Integer number
it will run always exceptions comes or not
```

In [66]:
```
func()
```

```
If no exception occures than else part executie
Given value is  10
it will run always exceptions comes or not
```

In [68]:
```
def func(num):
    try:
        if num == 0:
            raise ZeroDivisionError('number can not be zero')
        elif num < 0:
            raise ArithmeticError('number should be positive')
        else:
            num = num**2 # num = 25
        return num
    except ArithmeticError:
        print("you better answer it correctly")
    except Exception as e:
        print(f"!!Error!! General Error with message {e}")
```

```
    except ZeroDivisionError as e:
        print("wow genius")
        print(f"!!Error!!{e}")
    else:
        return num
    finally:
        return num // 2 # 25 // 2 -> 12
```

In [69]:
```
print(func(5))
```

```
12
```

In [70]:
```
print(func(0))
```

```
you better answer it correctly
0
```

In [71]:
```
print(func(-10))
```

```
you better answer it correctly
-5
```

Assertion Error

Unit Test

In [73]:
```
def func(x,y):
    """
    if x > y then return x**2 - y
    elif x < y then return y - x**2
    else x == y then return x**2 + y**2
    """
    if x > y:
        return x**2 - y
    elif y > x:
        return y - x**2
    else:
        return x**2 + y**2

def test_func():
    assert func(10, 5) == 95, "Test Case 1 Failed"
    assert func(5, 6) == -19, "Test Case 2 Failed"
    assert func(10, 10) == 200, "Test case 3 Failed"

try:
    test_func()
except AssertionError as e:
    print("!!Test Case Failed!!")
    print(e)
else:
    print("your code is working fine")
```

```
your code is working fine
```

In [74]:
```
def func(x,y):
    """
    if x > y then return x**2 - y
    elif x < y then return y - x**2
    else x == y then return x**2 + y**2
```

```
    """
    if x > y:
        return x**2 - y
    elif y > x:
        return y - x**2
    else:
        return x*2 + y**2

def test_func():
    assert func(10, 5) == 95, "Test Case 1 Failed"
    assert func(5, 6) == -19, "Test Case 2 Failed"
    assert func(10, 10) == 200, "Test case 3 Failed"

try:
    test_func()
except AssertionError as e:
    print("!!Test Case Failed!!")
    print(e)
else:
    print("your code is working fine")
```

```
!!Test Case Failed!!
Test case 3 Failed
```

Integrated Testing

```
30 func -> application
```

Problem

1.

In [75]:
```
class NegativeNumber(ArithmeticError):
    def __init__(self, arg):
        self.arg = arg
    def __str__(self):
        return f"{self.arg}"
    def do_this(self):
        print("do this task if Negative Number Error Comes")
    def do_that(self):
        print("do that task if Negative number Error Comes")
```

In [76]:
```
try:
    num = int(input("Enter number: "))
    if num < 0:
        raise NegativeNumber("please provide a positive number")
except ValueError as e:
    print("Invalid Input Try Again and only integer input allowed")
except NegativeNumber as e:
    print("!!Error!!", e)
    e.do_this()
    e.do_that()
```

```
!!Error!! please provide a positive number
do this task if Negative Number Error Comes
do that task if Negative number Error Comes
```

2.

In [77]:

```
class InvalidLengthException(Exception):
    pass
class Mobile:
    def __init__(self, mob_no):
        self.__mob_no = mob_no
    def validate_mobile_number(self):
        try:
            if (len(self.__mob_no) != 10):
                raise InvalidLengthException
            else:
                print("Valid Mobile Number")
        except InvalidLengthException:
            print('Invalid Length - inside class')
        print("Inside the class")

mob = Mobile("987665")

try:
    mob.validate_mobile_number()
    print("Outside the class")
except InvalidLengthException:
    print("Invalid Length - outside Class")
```

```
Invalid Length - inside class
Inside the class
Outside the class
```

In [ ]: