Python Exception Handling

Two stages of error

- During compilation -> Syntax Error
- During execution -> Exceptions

Syntax error

- Something in the program is not written according to the program grammar.
- Error is raised by the interpreter/compiler
- You can solve it by rectifying the program

Examples of syntax error

- Leaving symbols like colon,brackets
- Misspelling a keyword
- Incorrect indentation
- empty if/else/loops/class/functions

In [ ]:
```
print "hello world"
# it will give error since there is no bracket
```

In [ ]:
```
a=5
if a==3
    print("hii")
# it will give error since there is no colon after if
```

Types of Syntax error

Index Error

- when we try to access a item from index which does not exist

In [27]:
```
L=[1,2,3]
L[100]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[27], line 2
      1 L=[1,2,3]
----> 2 L[100]

IndexError: list index out of range
```

ModuleNotFoundError

- When we import a module which does not exist

In [37]:
```
import mathi
print(math.sqrt(5))
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[37], line 1
----> 1 import mathi
      2 print(math.sqrt(5))

ModuleNotFoundError: No module named 'mathi'
```

KeyError

- When a key is not found in dictionary

In [42]:
```
d={'a':"Apple",'b':"banana"}
d['c']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[42], line 2
      1 d={'a':"Apple",'b':"banana"}
----> 2 d['c']

KeyError: 'c'
```

TypeError

- When a operation or function is applied on a object of inappropiate type

In [45]:
```
"hii"+5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[45], line 1
----> 1 "hii"+5

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

ValueError

- When a function argument of an appropiate type

In [58]:
```
a,b=[1,2,3]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[58], line 1
----> 1 a,b=[1,2,3]

ValueError: too many values to unpack (expected 2)
```

NameError

- when a object not found

```
In [65]: k
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[65], line 1
----> 1 k

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

AttributeError

- If a method or object does not have such attribute

```
In [70]: L=[1,2,3]
         L.add(5)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[70], line 2
      1 L=[1,2,3]
----> 2 L.add(5)

AttributeError: 'list' object has no attribute 'add'
```

Exception

If things go wrong during the execution of the program(runtime). It generally happens when something unforeseen has happened.

- Exceptions are raised by python runtime
- You have to takle is on the fly

Examples

- Memory overflow
- Divide by 0 -> logical error
- Database error

Stack Trace :

The content we see on getting error. It tells on which line we are getting error , what kind of error it is and some basic details about the type of error.

Why it is important to handle exception ?

It helps in making a good user experience on handling exception.On doing exception handling , it helps to hide crucial info provided by normal interpreter in stack trace.

`How to handle exception`

Try and Except Block :

- Try : contain code that can give error in some speicific scenario
- Except : contains code which handle that scenrio if raised from try

In [80]:
```python
with open('sample.txt','w') as f :
    f.write("Hello world")
```

In [94]:
```python
try:
    with open('sampl.txt','r') as f:
        print(f.read())
except:
    print("file not found")
```

```
file not found
```

Catching multiple exception

In [103…
```python
try:
    m=5
    f=open('sample.txt','r')
    print(f.read())
    print(m)
    print(5/0)
    L=[1,2,3]
    L[100]
except FileNotFoundError:
    print("file not found")
except NameError:
    print("please declare the variable first before use")
except ZeroDivisionError:
    print("cannot divided by zero")
except Exception as e:  # this block should be at last
    print(e)
```

```
Hello world
5
cannot divided by zero
```

Else : contain content which execute when try get succesfully executed

In [108…

```
try:
  f = open('sample.txt','r')
except FileNotFoundError:
  print('file nai mili')
except Exception:
  print('kuch to lafda hai')
else:
  print(f.read())
```

```
Hello world
```

Finally : It is the block which runs anyhow

- Finally can be used to close connection like db connection or bluetooth connection

In [111…

```
try:
  f = open('sample1.txt','r')
except FileNotFoundError:
  print('file nai mili')
except Exception:
  print('kuch to lafda hai')
```

```
else:
  print(f.read())
finally:
  print('ye to print hoga hi')
```

```
file nai mili
ye to print hoga hi
```

Raise Exception

- In Python programming, exceptions are raised when errors occur at runtime.
- We can also manually raise exceptions using the raise keyword.
- It helps to raise exception with a msg
- We can optionally pass values to the exception to clarify why that exception was raised

In [121…

```
raise ZeroDivisionError("this is a exception raised using 'raise' keyword")
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[121], line 1
----> 1 raise ZeroDivisionError("this is a exception raised using 'raise' keyword")

ZeroDivisionError: this is a exception raised using 'raise' keyword
```

In [131…

```
class Bank:
    def __init__(self,bal):
        self.balance=bal
    def withdrawn(self,amt):
        if amt<0:
            raise Exception("your amount is in negative")
        if self.balance<amt:
            raise Exception("amount is less than balance")
        self.balance-=amt

obj=Bank(19000)
try:
    obj.withdrawn(-35)
except Exception as e:
    print(e)
else:
    print(obj.balance)
```

```
your amount is in negative
```

Creating Custom Exception

Why we are creating own custom exception class ?

- To get full control and able to do multiple task if get such user defined exception

In [144…

```
class MyException(Exception):
  def __init__(self,message):
    print(message)

class Bank:

  def __init__(self,balance):
    self.balance = balance

  def withdraw(self,amount):
    if amount < 0:
      raise MyException('amount cannot be -ve')
    if self.balance < amount:
      raise MyException('paise nai hai tere paas')
    self.balance = self.balance - amount

obj = Bank(10000)
try:
  obj.withdraw(50000)
except MyException as e:
  # print(e)
    pass
else:
  print(obj.balance)
```

```
paise nai hai tere paas
```

In [147…

```
class SecurityError(Exception):

  def __init__(self,message):
    print(message)

  def logout(self):
    print('logout')

class Google:

  def __init__(self,name,email,password,device):
    self.name = name
    self.email = email
    self.password = password
    self.device = device

  def login(self,email,password,device):
    if device != self.device:
      raise SecurityError('bhai teri to lag gayi')
    if email == self.email and password == self.password:
      print('welcome')
    else:
      print('login error')



obj = Google('nitish','nitish@gmail.com','1234','android')

try:
  obj.login('nitish@gmail.com','1234','windows')
except SecurityError as e:
  e.logout()
else:
  print(obj.name)
finally:
  print('database connection closed')
```

```
bhai teri to lag gayi
logout
database connection closed
```

END