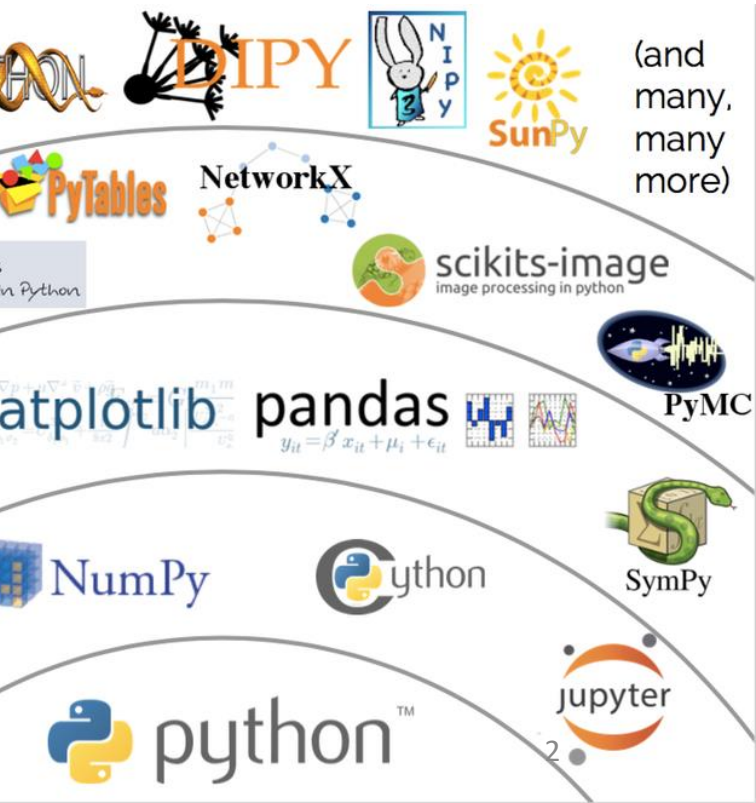
Python Programming

Exception Handling

Dr. Sarwan Singh
NIELIT Chandigarh

Agenda

- Type of errors
- Introduction –Exception, assertion
- Handling exceptions
- User defined exception
- Exception block
- Argument of an exception

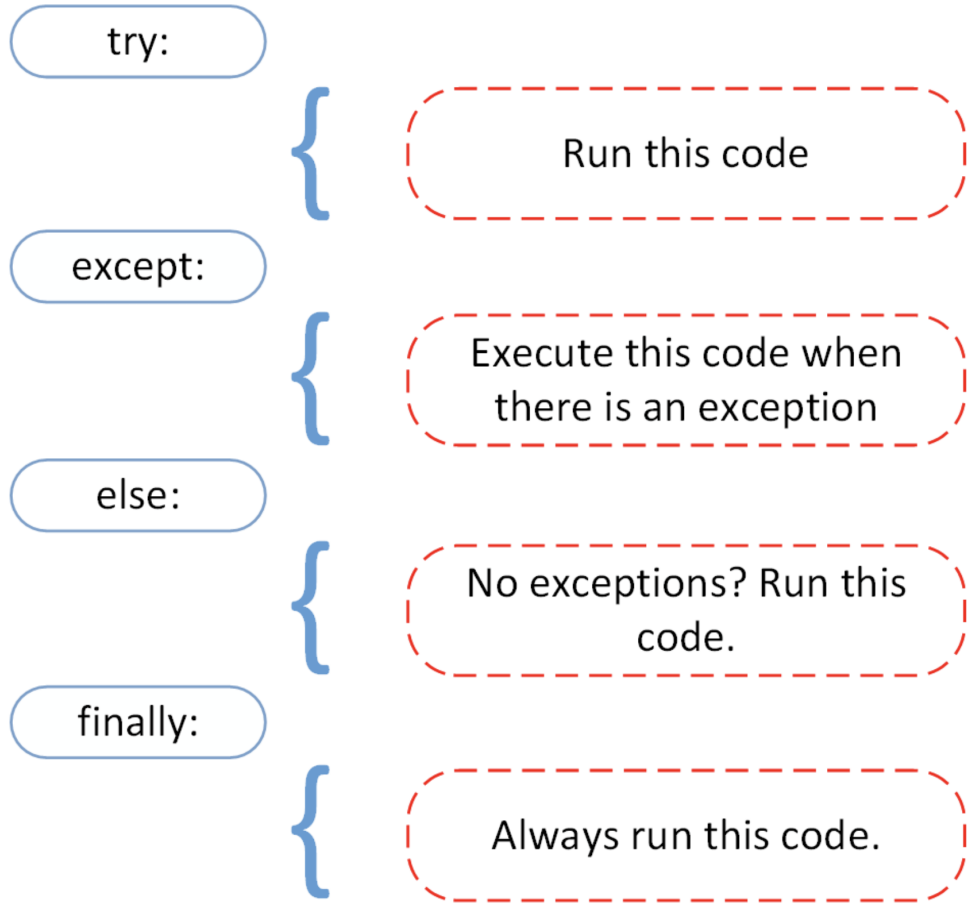


One guiding principle of Python code is that

"explicit is better than implicit"

References

- Docs.python.org
- Tutorialpoint.org
- learnbyexample.org
- Datacamp.org



Errors and Exceptions

- There are (at least) two distinguishable kinds of errors:
  - *syntax errors* - Syntax errors, also known as parsing errors, are perhaps the most common kind of complaint every programmer has
  - *exceptions* - Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it.

    Errors detected during execution are called *exceptions* and are not unconditionally fatal.

Errors and Exceptions

Other categories can be :

- *Out of Memory Error –* Memory errors are mostly dependent on your systems RAM and are related to **Heap**.
  - large objects (or) referenced objects in memory, then – OutofMemoryError.
  - Loading a very large data file, running Deep Learning model
- *Recursion Error -* It is related to **stack.**
  - error transpires when too many methods, one inside another is executed (one with an infinite recursion), which is limited by the size of the stack.

Exception handling and debugging capabilities

- **Exception Handling**
  - An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.
  - When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.
- **Assertions**
  - An assertion is a sanity-check that you can turn on or turn off when you are done with your testing of the program.
  - The easiest way to think of an assertion is to liken it to a **raise-if** statement (or to be more accurate, a raise-if-not statement). An expression is tested, and if the result comes up false, an exception is raised.
  - Assertions are carried out by the assert statement

Arithmetic Error

- Zero Division Error
- OverFlow Error
- Floating Point Error

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
```

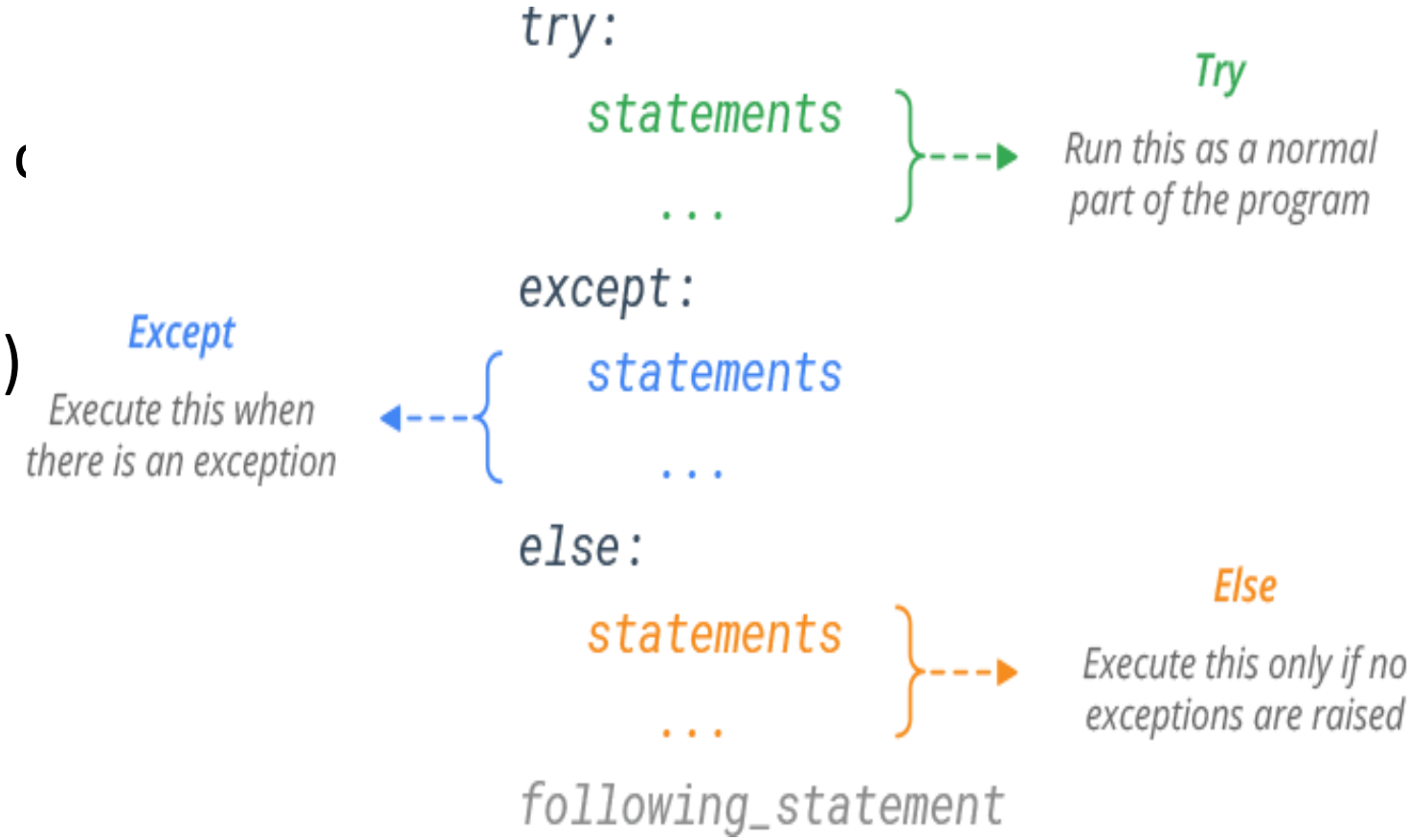
Lookup Error

- Lookup Error acts as a base class for the exceptions that occur when a **key** or **index** used on a mapping or sequence of a list/dictionary is invalid or does not exists.
- The two types of exceptions raised are:
  - **IndexError -** When you are trying to access an index (sequence) of a list that does not exist in that list or is out of range of that list
  - **KeyError -** If a key you are trying to access is not found in the dictionary

Try-except block

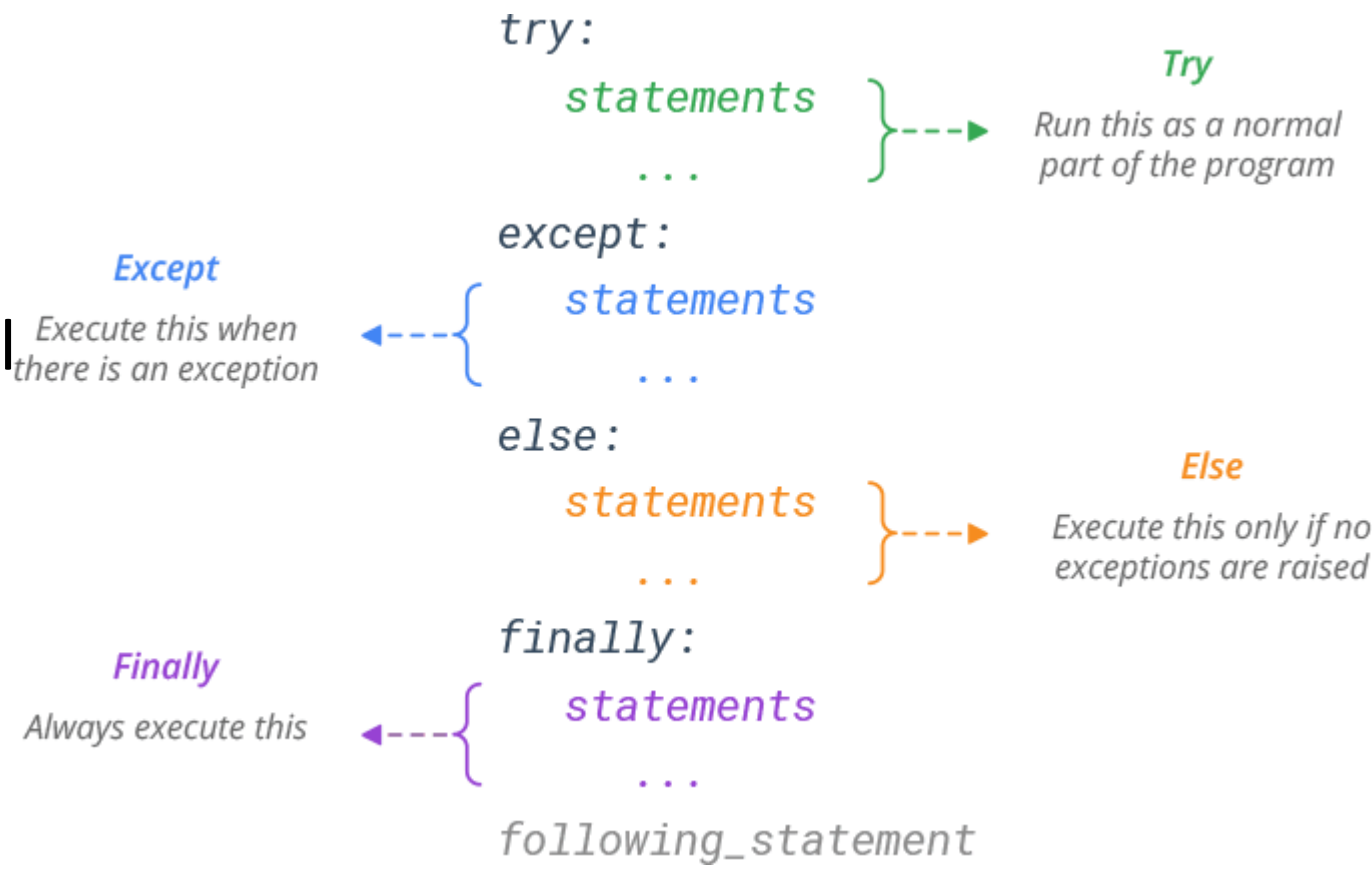
```
try:
        x = 1/0
        #statement or function call
except :
        print('divide by 0 error')
```

```
try:
        x = 1/0
        #statement or function c
except :
        print('divide by 0 error')
else:
        print('no error')
```



Finally clause

```
try:
        x = 1/0
        #statement or function call
except :
        print('divide by 0 error')
else:
        print('no error')
finally:
        print('this will always execute')
```



```
try:
      You do your operations here;
      ......................
except ExceptionI:
      If there is ExceptionI, then
      execute this block
except ExceptionII:
      If there is ExceptionII, then
      execute this block.
      ......................
else:
      If there is no exception then
      execute this block.
finally:
      this block will always execute
```

Using finally for clean-up actions

```
# Exception handling during file manipulation
f = open('myfile.txt')
try:
        print(f.read())
except:
        print("Something went wrong")
finally:
        f. close()
```

Raising an Exception

- User can raise an exception when a certain condition occurs, using raise keyword

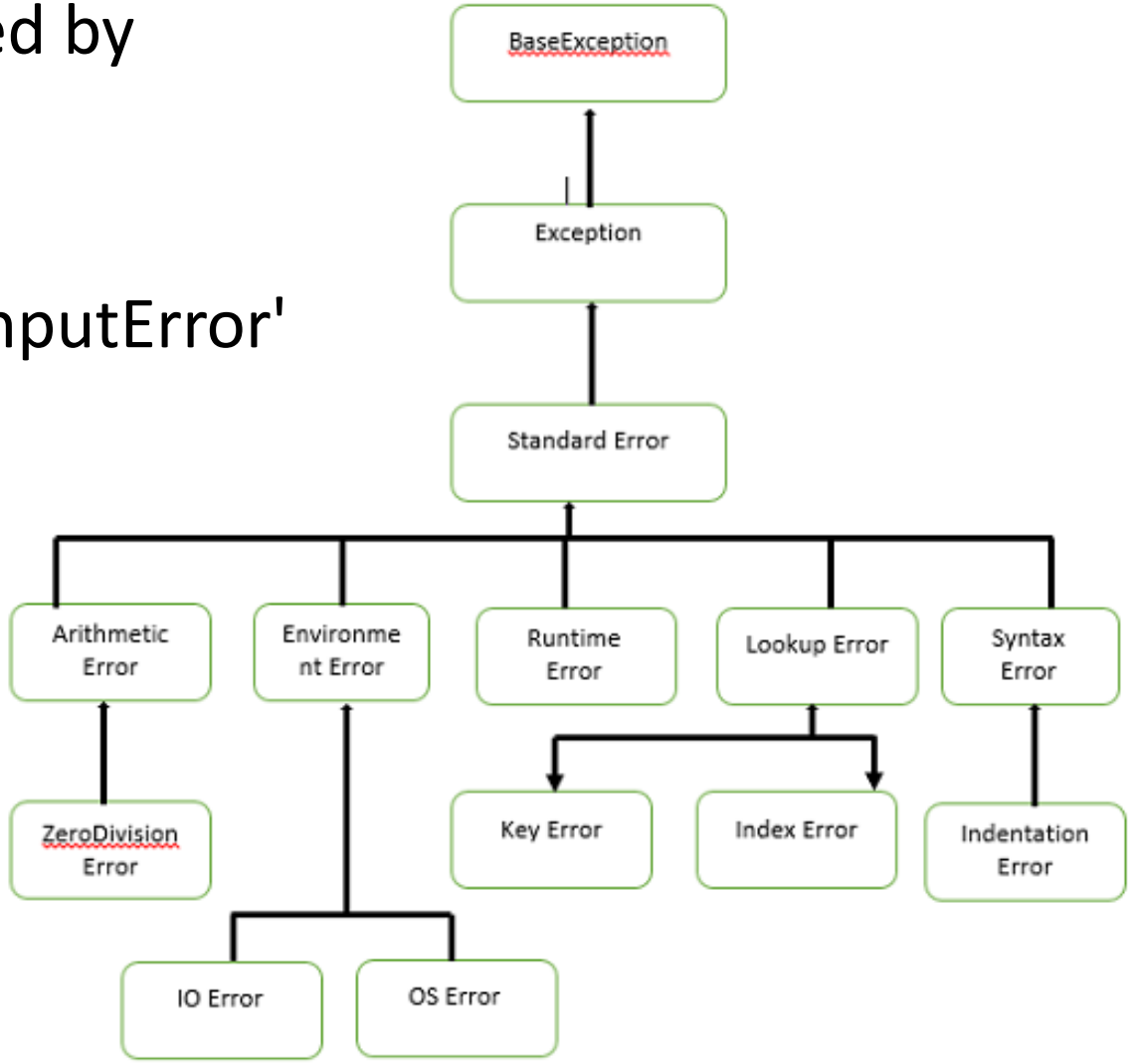
```
# Raise built-in exception 'NameError'
raise NameError('An exception occured!')
```

User-defined Exceptions

- User defined exceptions can be created by inheriting new exception class

```
# Create and raise Custom exception 'InputError'
class InputError(Exception):
    pass

raise InputError('Custom exception')
```



Argument of an Exception

- An exception can have an *argument*, which is a value that gives additional information about the problem.
- The contents of the argument vary by exception.

```
try:
      You do your operations here;
      ......................
except ExceptionType as Argument:
      You can print value of Argument here...
```

Argument in user-defined Exceptions

```
# create user-defined exception
# derived from super class Exception
class MyError(Exception):
 # Constructor or Initializer
  def __init__(self, value):
    self.value = value

  # __str__ is to print() the value
  def __str__(self):
    return(repr(self.value))
try:
  raise(MyError("Some Error Data"))
# Value of Exception is stored in error
except MyError as Argument:
  print('This is the Argument\n', Argument)
```

Sr.No.	Exception Name & Description
1	**Exception** Base class for all exceptions
2	**StopIteration** Raised when the next() method of an iterator does not point to any object.
3	**SystemExit** Raised by the sys.exit() function.
4	**StandardError** Base class for all built-in exceptions except StopIteration and SystemExit.
5	**ArithmeticError** Base class for all errors that occur for numeric calculation.
6	**OverflowError** Raised when a calculation exceeds maximum limit for a numeric type.
7	**FloatingPointError** Raised when a floating point calculation fails.
8	**ZeroDivisionError** Raised when division or modulo by zero takes place for all numeric types.
9	**AssertionError** Raised in case of failure of the Assert statement.
10	**AttributeError** Raised in case of failure of attribute reference or assignment.

assert Expression[, Arguments]

- When it encounters an assert statement, Python evaluates the accompanying expression, which is hopefully true. If the expression is false, Python raises an *AssertionError* exception.
- AssertionError exceptions can be caught and handled like any other exception using the try-except statement, but if not handled, they will terminate the program and produce a traceback.