GLS UNIVERSITY

# Unit 3 Modules, Name Spaces and Exception Handling

- Prof. Poonam Dang
- Prof. Neha Samsir

## Modules - Introduction

- Modules are used to categorize code into smaller part.
- A module is simply a file, where classes, functions and variables are defined.
- Modules provide reusability of code.
- User can define our most used functions in a module and import it, instead of copying their definitions into different programs.
- The name of the module is the name of the file name with .py extension.
- Modules are not loaded unless we execute in Python interpreter or call within a program.

- **Advantages:**
  - **Reusability:** Module can be used in some other python code. Hence it provides the facility of code reusability.
  - **Categorization:** Similar type of attributes can be placed in one module.

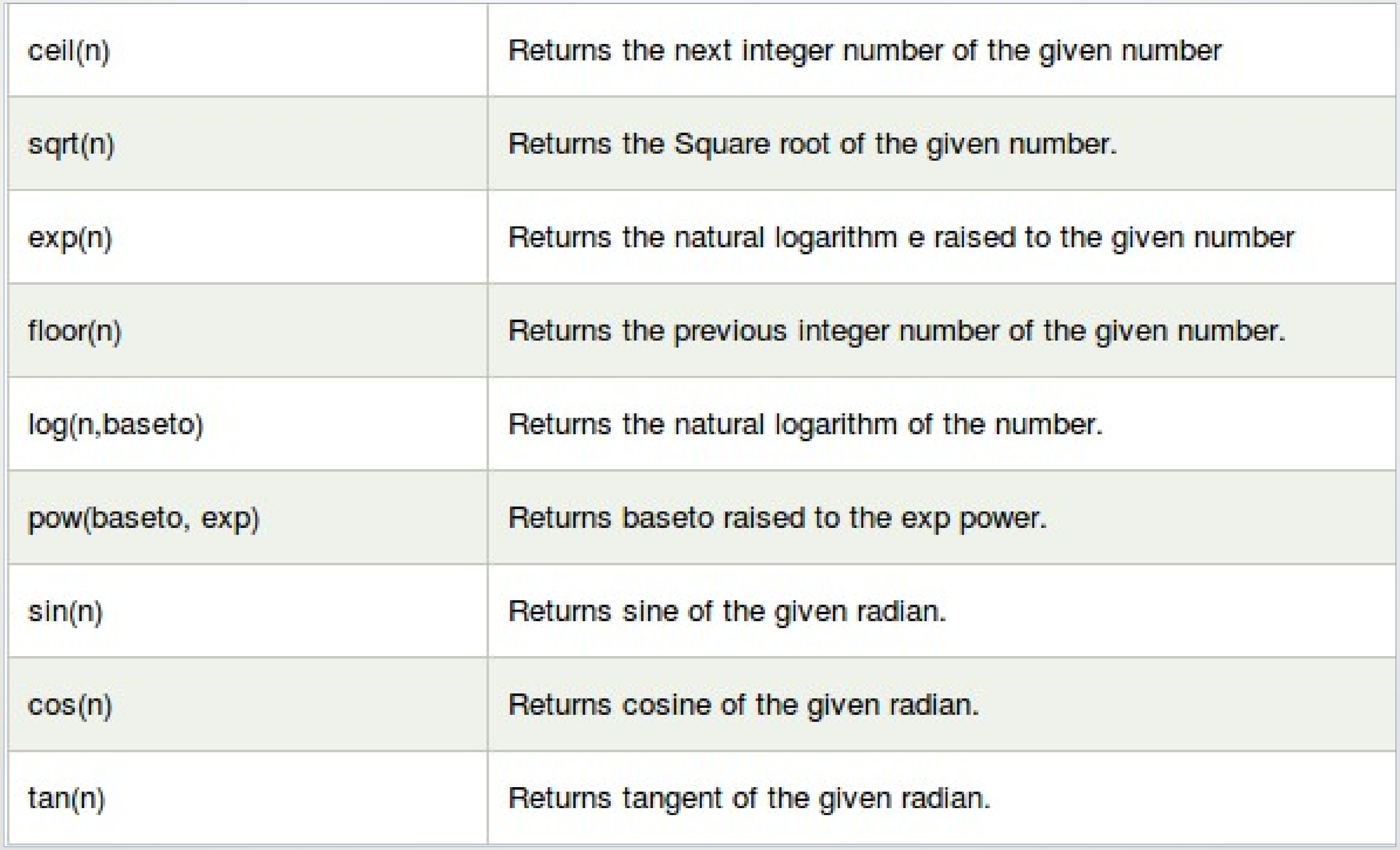
## Built – In Modules

- **Built in Modules**

- There are many built in modules in Python.
- Some of them are as  follows:
  - math, random , threading , collections , os , mailbox , string , time , tkinter etc..
- Each module has a number of built in functions which can be used to perform various functions.

## Built – In Modules

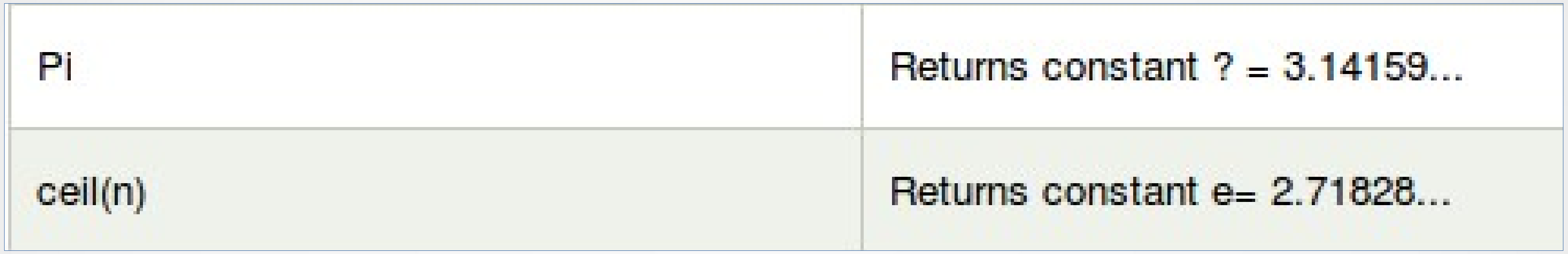
**Math Built in Modules:** Using math module , you can use different built in mathematical functions.



| | |
|---|---|
| ceil(n) | Returns the next integer number of the given number |
| sqrt(n) | Returns the Square root of the given number. |
| exp(n) | Returns the natural logarithm e raised to the given number |
| floor(n) | Returns the previous integer number of the given number. |
| log(n,baseto) | Returns the natural logarithm of the number. |
| pow(baseto, exp) | Returns baseto raised to the exp power. |
| sin(n) | Returns sine of the given radian. |
| cos(n) | Returns cosine of the given radian. |
| tan(n) | Returns tangent of the given radian. |

## Built – In Modules

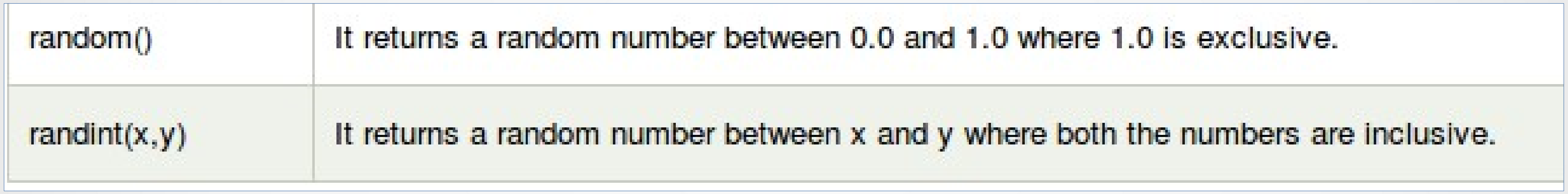
**Math Built in Modules:** The math module provides two constants for mathematical Operations



| | |
|---|---|
| Pi | Returns constant ? = 3.14159... |
| ceil(n) | Returns constant e= 2.71828... |

## Built – In Modules

**Random Built in Modules:** The random module is used to generate the random numbers. It provides the following two built in functions:



| | |
|---|---|
| random() | It returns a random number between 0.0 and 1.0 where 1.0 is exclusive. |
| randint(x,y) | It returns a random number between x and y where both the numbers are inclusive. |

**Other modules will be covered in their respective topics.**

## Built – In Modules

```
import math
a=4.6
print math.ceil(a)
print math.floor(a)
b=9
print math.sqrt(b)
print math.exp(3.0)
print math.log(2.0)
print math.pow(2.0,3.0)
print math.sin(0)
print math.cos(0)
print math.tan(45)

print math.pi
print math.e
```

## Modules - Import

**Importing a Module:** There are different ways by which you we can import a module.

**1. Using import statement:** "import" statement can be used to import a module.

- **Syntax:**
- **import <file_name1, file_name2,...file_name(n)="">**
- 
- **NOTE:** You can access any function which is inside a module by module name and function name separated by dot. It is also known as period. Whole notation is known as dot notation.

## Modules - Import

**Importing a Module:** There are different ways by which you we can import a module.

**2. Using from.. import statement:** from..import statement is used to import particular attribute from a module.
In case you do not want whole of the module to be imported then you can use from...import statement.

- **Syntax:**

**From <module_name> import <attribute1, attribute2, attribute3,... attributen>**
**</attribute1,attribute2,attribute3,...attributen></module_name>**

## Modules - Import

**Importing a Module:** There are different ways by which you we can import a module.

**3. To import whole module:** You can import whole of the module using "from...import *".

- **Syntax:**
  **from <module_name> import ***
  **</module_name>**

## Modules - Import

```
def circle(r):
        print(3.14*r*r)
        return
def square(l):
    print(l*l)
    return
def rectangle(l,b):
    print(l*b)
    return
def triangle(b,h):
    print(0.5*b*h)
    return
```

Save the file as area.py

## Modules - Import

```
from area import square,rectangle
        square(10)
        rectangle(2,5)
```

```
from area import *
square(10)
rectangle(2,5)
circle(5)
triangle(10,20)
```

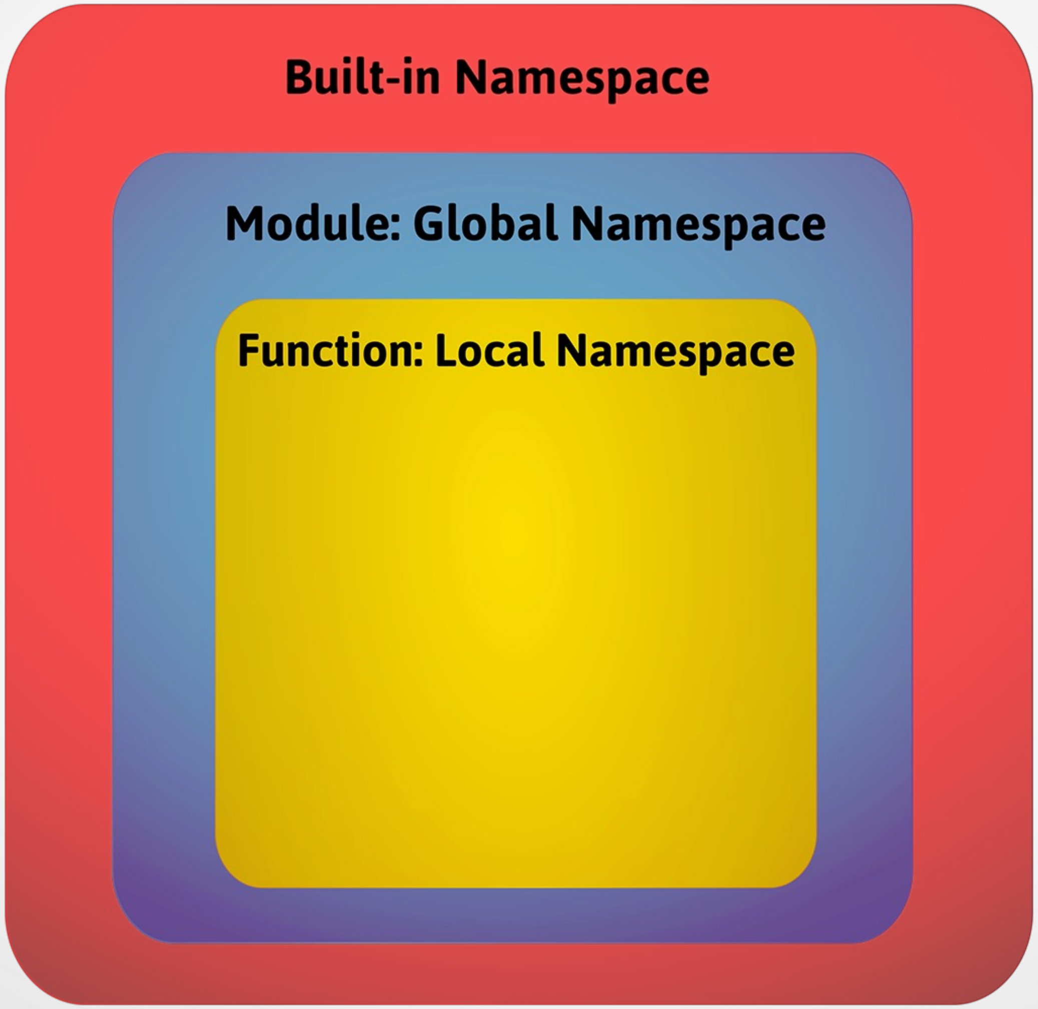
## Namespaces

Variables are names (identifiers) that map to objects.
A namespace in python is a collection of names.
So, a namespace is essentially a mapping of names to corresponding objects.
At any instant, different python namespaces can coexist completely isolated- the isolation ensures that there are no name collisions.
Two namespaces in python can have the same name without facing any problem.
A namespace is implemented as a **Python dictionary.**

Different examples of Namespace:

- Local Namespace
- Global Namespace
- Built-in Namespace

## Namespaces


Built-in Namespace
Module: Global Namespace
Function: Local Namespace

## Namespaces

**Local Namespace**
This namespace covers the local names inside a function. Python creates this namespace for every function called in a program. It remains active until the function returns.

**Global Namespace**
This namespace covers the names from various imported modules used in a project. Python creates this namespace for every module included in your program. It'll last until the program ends.

**Built-in Namespace**
This namespace covers the built-in functions and built-in exception names. Python creates it as the interpreter starts and keeps it until you exit.

## Namespaces

**dir( ) Function:**

The dir() built-in function returns a sorted list of strings containing the names defined by a module.
The list contains the names of all the modules, variables and functions that are defined in a module.
dir() tries to return a valid list of attributes of the object it is called upon.
Also, dir() function behaves rather differently with different type of objects, as it aims to produce the most relevant one, rather than the complete information.
If no parameters are passed it returns a list of names in the current local scope

- **Syntax:**
- dir(object)

## Namespaces

Example:

```
number = [1, 2, 3]
print(dir(number))

print('\nReturn Value from empty dir()')
print(dir())
```

## Scope of Variables

Scope is the portion of the program from where a namespace can be accessed directly without any prefix.
At any given moment, there are at least three nested scopes.

- Scope of the current function which has **local** names
- Scope of the module which has **global** names
- Outermost scope which has built-in names

When a reference is made inside a function, the name is searched in the local namespace, then in the global namespace and finally in the built-in namespace.

## Namespaces

**Global Keyword:**

In Python, global keyword allows you to modify the variable outside of the current scope. It is used to create a global variable and make changes to the variable in a local context.

**Rules of global Keyword**

When we create a variable inside a function, it’s local by default.

When we define a variable outside of a function, it’s global by default. You don’t have to use global keyword.

We use global keyword to read and write a global variable inside a function.

Use of global keyword outside a function has no effect

## Packages

**Example:**

- First of all, we need a directory.
- The name of this directory will be the name of the package, which we want to create.
- We will call our package "pack1".
- This directory needs to contain a file with the name "__init__.py".
- This file can be empty, or it can contain valid Python code.
- This code will be executed when a package will be imported, so it can be used to initialize a package, e.g. to make sure that some other modules are imported or some values set.
- Now we can put into this directory all the Python files which will be the submodules of our module.

## Packages

We create two simple files a.py and b.py just for the sake of filling the package with modules.

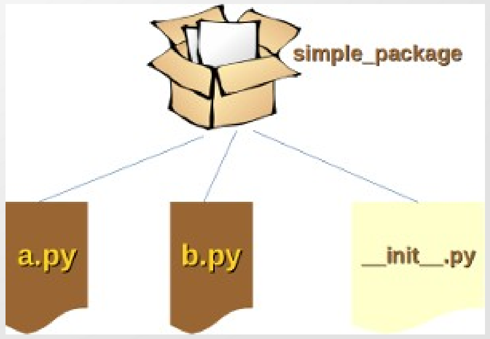
The content of a.py:

```
def a1():
    print("Hello, function 'a1' from module 'a' calling")
```

The content of b.py:

```
def b1():
    print("Hello, function 'b1' from module 'b' calling")
```

**Note:** __init__.py is simply a file that is used to consider the directories on the disk as the package of the Python. It is basically used to initialize the python packages.

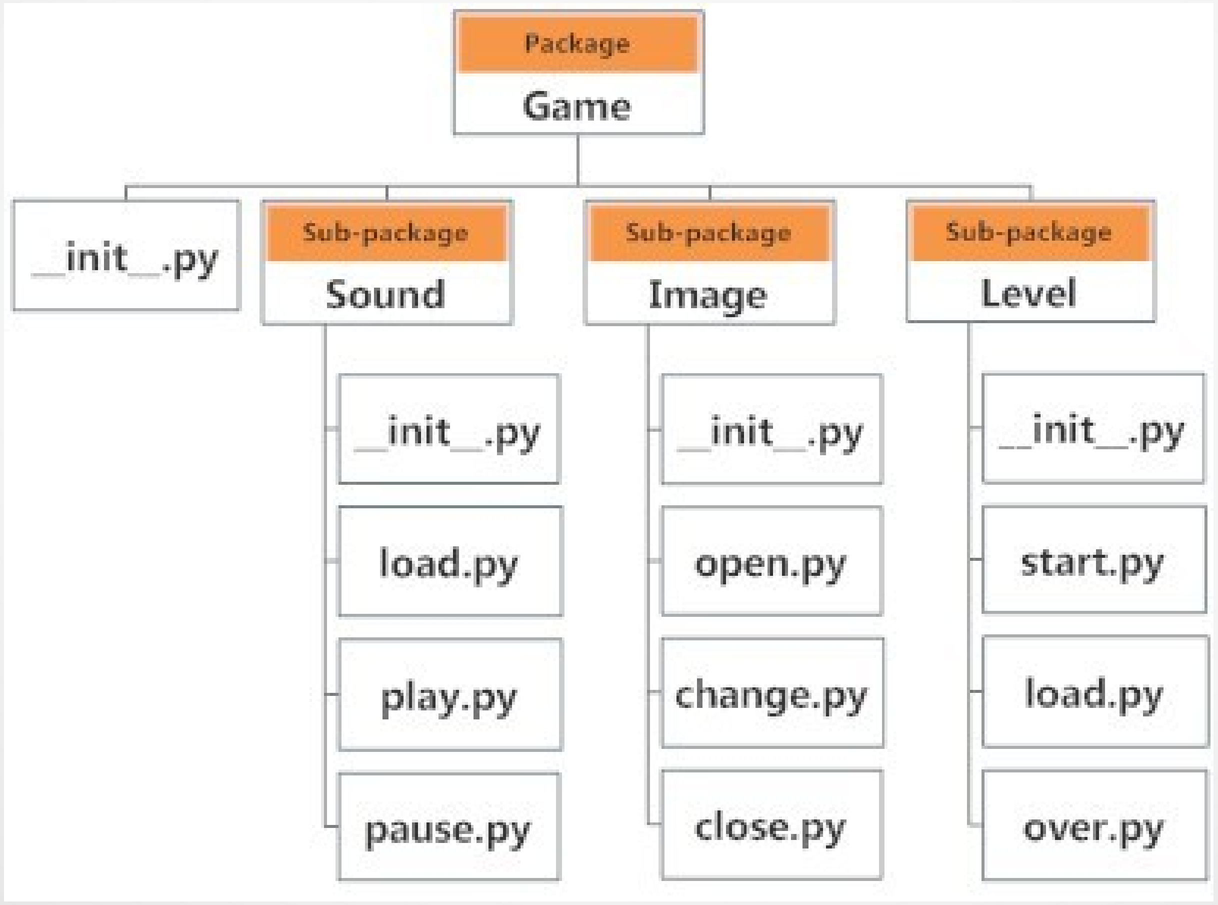


## Packages

- A Package is simply a collection of similar modules, sub-packages etc.
- A package is basically a directory with Python files and a file with the name __init__.py.
- This means that every directory inside of the Python path, which contains a file named __init__.py, will be treated as a package by Python.
- It's possible to put several modules into a Package.

- Packages are a way of structuring Python’s module namespace by using "dotted module names".
- A.B stands for a submodule named B in a package named A.
- Two different packages like P1 and P2 can both have modules with the same name, let's say A, for example.
- The submodule A of the package P1 and the submodule A of the package P2 can be totally different.
- A package is imported like a "normal" module.

## Packages


Package
Game
__init__.py
Sub-package
Sound
Sub-package
Image
Sub-package
Level
__init__.py
load.py
play.py
pause.py
__init__.py
open.py
change.py
close.py
__init__.py
start.py
load.py
over.py

- <u>Importing module from a package</u>
- We can import modules from packages using the dot (.) operator.
- For example, if want to import the start module in the above example, it is done as follows.
- import Game.Level.start

- Now if this module contains a function named select_difficulty(), we must use the full name to reference it.
- 
- Game.Level.start.select_difficulty(2)
- 
- If this construct seems lengthy, we can import the module without the package prefix as follows.
- 
- from Game.Level import start

**try and except in Python**

try() is used in Error and Exception Handling

There are two kinds of errors :

Syntax Error : Also known as Parsing Errors, most basic. Arise when the Python parser is unable to understand a line of code.

Exception : Errors which are detected during execution. eg – ZeroDivisionError.

How try() works?
First try clause is executed i.e. the code between try and except clause.
If there is no exception, then only try clause will run, except clause is finished.
If any exception occured, try clause will be skipped and except clause will run.
If any exception occurs, but the except clause within the code doesn’t handle it, it is passed on to the outer try statements. If the exception left unhandled, then the execution stops.
A try statement can have more than one except clause

## Modules, Namespaces & Exception Handling

**Exception Handling:**

- Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.
- Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.
- When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash.
- If never handled, an error message is spit out and our program come to a sudden, unexpected halt.
- **Hierarchy Of Exception:**
  - ZeroDivisionError: Occurs when a number is divided by zero.
  - NameError: It occurs when a name is not found. It may be local or global.
  - IndentationError: If incorrect indentation is given.
  - IOError: It occurs when Input Output operation fails.
  - EOFError: It occurs when end of file is reached and yet operations are being performed, etc..

## Modules, Namespaces & Exception Handling

**Exception Handling:**

```
except IOError:
    print('An error occurred trying to read the file.')

except ValueError:
    print('Non-numeric data found in the file.')

except ImportError:
    print "NO module found"

except EOFError:
    print('Why did you do an EOF on me?')

except KeyboardInterrupt:
    print('You cancelled the operation.')

except:
    print('An error occurred.')
```

## Modules, Namespaces & Exception Handling

**Catching Exceptions in Python:**

In Python, exceptions can be handled using a try statement.

**Syntax:**

try:

-     malicious code
- except Exception1:
-     execute code
- except Exception2:
-     execute code
- ....
- ....
- except ExceptionN:
-     execute code
- else:
-     In case of no exception, execute the else block code.

# Modules, Namespaces & Exception Handling

**Catching Exceptions in Python:**

- The malicious code (code having exception) is enclosed in the try block.

Try block is followed by except statement. There can be multiple except statement with a single try block.

- Except statement specifies the exception which occurred. In case that exception is occurred, the corresponding statement will be executed.
- At the last you can provide else statement. It is executed when no exception is occurred.

## Modules, Namespaces & Exception Handling

**The except Clause with No Exceptions:**

Except statement can also be used without specifying Exception.

**Syntax:**

```
try:
    code
except:
    code to be executed in case exception occurs.
else:
    code to be executed in case exception does not occur.
```

## Modules, Namespaces & Exception Handling

**Declaring Multiple Exception:**

User can also use the same except statement to handle multiple exceptions.

**Syntax:**

```
try:
    code
except Exception1,Exception2,Exception3,..,ExceptionN
    execute this code in case any Exception of these occur.
else:
    execute code in case no exception occurred.
```

## Modules, Namespaces & Exception Handling

**Finally Block:**

In case if there is any code which the user want to be executed, whether exception occurs or not then that code can be placed inside the finally block. Finally block will always be executed irrespective of the exception.

This clause is executed no matter what, and is generally used to release external resources.

It is intended to define clean-up actions that must be executed under all circumstances

**Syntax:**

```
try:
    Code
finally:
    code which is must to be executed.
```

# Modules, Namespaces & Exception Handling

**Raise an Exception:**

In Python, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.

raise will cause an exception to occur and thus execution control will stop in case it is not handled.

**Syntax:**

```
    raise Exception_class,<value>
```