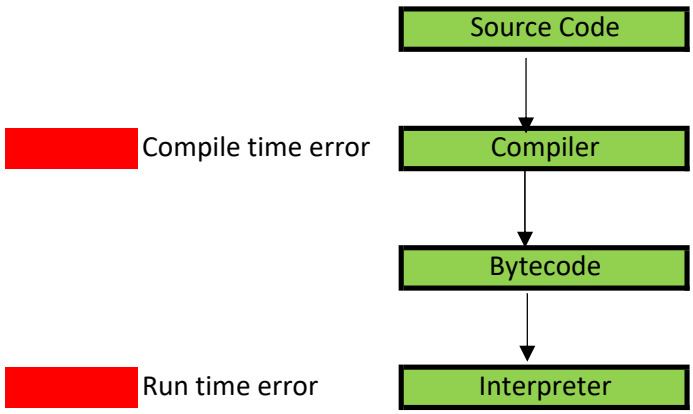
EXCEPTION HANDLING

An **exception** is an error that occurs during the execution of code. This error causes the code to raise an exception and if not prepared to handle it will halt the execution of the code.

A **try except** will allow you to execute code that might raise an exception and in the case of any exception or a specific one we can handle or catch the exception and execute specific code. This will allow us to continue the execution of our program even if there is an exception

There are 2 types of error that a code or a program faces

1. Compile time error — Basically syntax error
2. Run time error — Error at interpreter level.



Eg. Code to a two variables

```
a = 10
b = "s"
c = a + b
print(c')
```

Type error : Cannot add int and str type
This is run time error

Once run the code will give exception saying cant add int and str
Finding the isuue is called exception handling.

Compiler will give error in case of missing words/characters eg print(c
or case

Exception ==== Run time error

Handling the error

Exception Handling

Why do we do exception handlng ?

Because of two reasons

1. When there is issue in one part of code, it will to help run code excluding that part of code.

```
Eg a = int(input("enter first number"))
   b = int(input("enter second number"))
   c = a/b
   return c
   d = a*b
   print(d)
   enter first number10          It is run time error
   enter second number0          It will give error for division but it does not give
                                 output for multiplication even when it is correct.
```

2. It error which pops out, its interface is not user friendly.

Catching an Exception

Python tries to execute the code in the try block. In this case if there is any exception raised by the code in the try block, it will be caught and the code block in the except block will be executed. After that, the code that comes after the try except will be executed.

Try Except Specific

Try Except Else and Finally

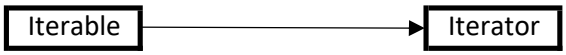
Iterators and Generators

Iterators:

• An **iterator** is an object that manages an iteration through a series of values. If variable, i, identifies an iterator object, then each call to the built-in function, next(i), produces a subsequent element from the underlying series, with a StopIteration exception raised to indicate that there are no further elements.

• An **iterable** is an object, obj, that produces an iterator via the syntax iter(obj).

list, tuple, and set, qualify as iterable types



data = [1, 2, 4, 8]

iter(data)	Will convert iterable to iterator
next(data)	extract subsequent data
StopIteration exception	When iterations are complete

Generators

A **generator** is implemented with a syntax that is very similar to a function, but instead of returning values, a yield statement is executed to indicate each element of the series

Keywords in exception handling
While true
try
except

else
raise
finally

Comprehension Syntax

Comprehension syntax is to produce one series of values based upon the processing of another series.

[k k for k in range(1, n+1)] list comprehension
{ k k for k in range(1, n+1) } set comprehension
(k k for k in range(1, n+1)) generator comprehension
{ k : k k for k in range(1, n+1) } dictionary comprehension

Packing and Unpacking of Sequences

data = 2, 4, 6, 8

results in identifier, data, being assigned to the tuple (2, 4, 6, 8). This behavior is called **automatic packing** of a tuple.

return x, y

it will be formally returning a single object that is the tuple (x, y).
As a dual to the packing behavior, Python can automatically **unpack a sequence,**

Simultaneous Assignments

The combination of automatic packing and unpacking forms a technique known as **simultaneous assignment**, whereby we explicitly assign a series of values to a series of identifiers,

x, y, z = 6, 2, 5

In effect, the right-hand side of this assignment is automatically packed into a tuple, and then automatically unpacked with its elements assigned to the three identifiers on the left-hand side.

all of the expressions are evaluated on the right-hand side before any of the assignments are made to the left-hand variables. This is significant, as it provides a convenient means for swapping the values associated with two variables:

j, k = k, j

j will be assigned to the old value of k, and k will be assigned to the old value of j.

A swap typically requires more delicate use of a temporary variable, such as

temp = j
j = k
k = temp

Scopes and Namespaces

Top-level assignments are typically made in what is known as **global scope**.
Assignments made within the body of a function typically have scope that is **local**

Each distinct scope in Python is represented using an abstraction known as a **namespace**.
A namespace manages all identifiers that are currently defined in a given scope.

dir( ) reports the names of the identifiers in a given namespace (i.e., the keys of the dictionary) It gives list of names comprising (some of) the attributes of the given object, and of attributes reachable from it.
vars( ) returns the full dictionary of identifier, value pairs

First-class objects are instances of a type that can be assigned to an identifier, passed as a parameter, or returned by a function.
Eg int and list, are clearly first-class types in Python.

scream = print() ***Python allows one function to be passed as a parameter to another.***
scream(Hello)

Modules and the Import Statement

from math **import** pi, sqrt	Allows direct use of the identifier, pi, or a call of the function, sqrt(2)
import math	Accessed using a fully-qualified name, such as math.pi or math.sqrt(2).

If there is file named utility.py, we could import that function from that module using the syntax:

from utility **import** count

The condition **if __name__ == '__main__'** is used in a Python program to execute the code inside the if statement only when the program is executed directly by the Python interpreter. When the code in the file is imported as a module the code inside the if statement is not executed

Existing Modules:

array	Provides compact array storage for primitive types.
collections	Defines additional data structures and abstract base classes involving collections of objects.
copy	Defines general functions for making copies of objects.
heapq	Provides heap-based priority queue functions
math	Defines common mathematical constants and functions.
os	Provides support for interactions with the operating system.
random	Provides random number generation.
re	Provides support for processing regular expressions.
sys	Provides additional level of interaction with the Python interpreter.
time	Provides support for measuring time, or delaying a program.

Pseudo-Random Number Generation

next = (a*current + b) % n; Generate random no each time you pass the argument

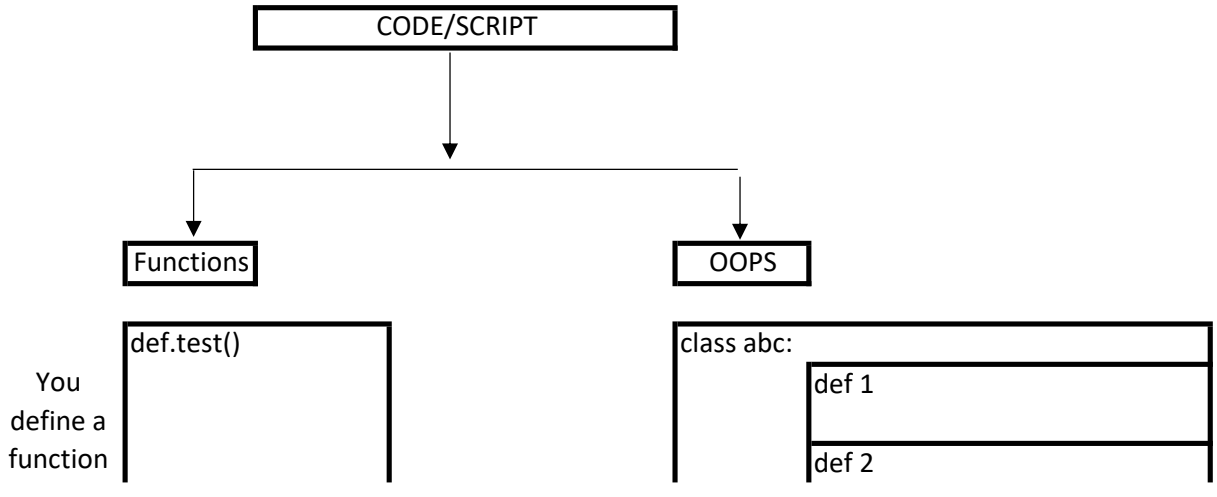
Syntax	Description
seed(hashable)	Initializes the pseudo-random number generator based upon the hash value of the parameter
random()	Returns a pseudo-random floating-point value in the interval [0.0,1.0).
randint(a,b)	Returns a pseudo-random integer in the closed interval [a,b].
randrange(start, stop, step)	Returns a pseudo-random integer in the standard Python range indicated by the parameters.
choice(seq)	Returns an element of the given sequence chosen pseudo-randomly.
shuffle(seq)	Reorders the elements of the given sequence pseudo-randomly.

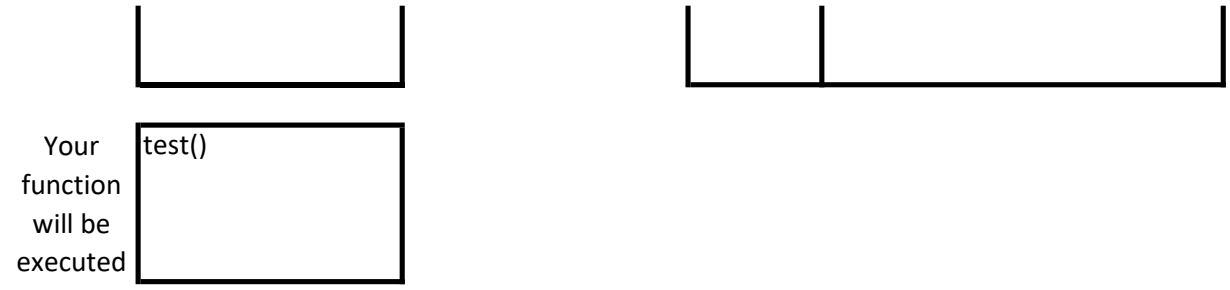
Classes and Objects

Creating a Class:
The first step in creating a class is giving it a name.

Instances of a Class: Objects and Attributes
An instance of an object is the realisation of a class,

Methods
Methods give you a way to change or interact with the object;
They are functions that interact with objects.

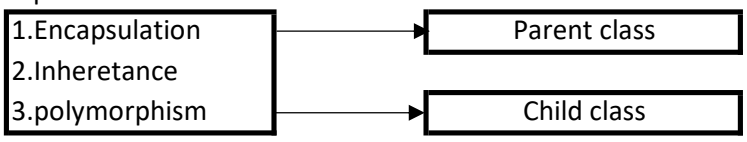


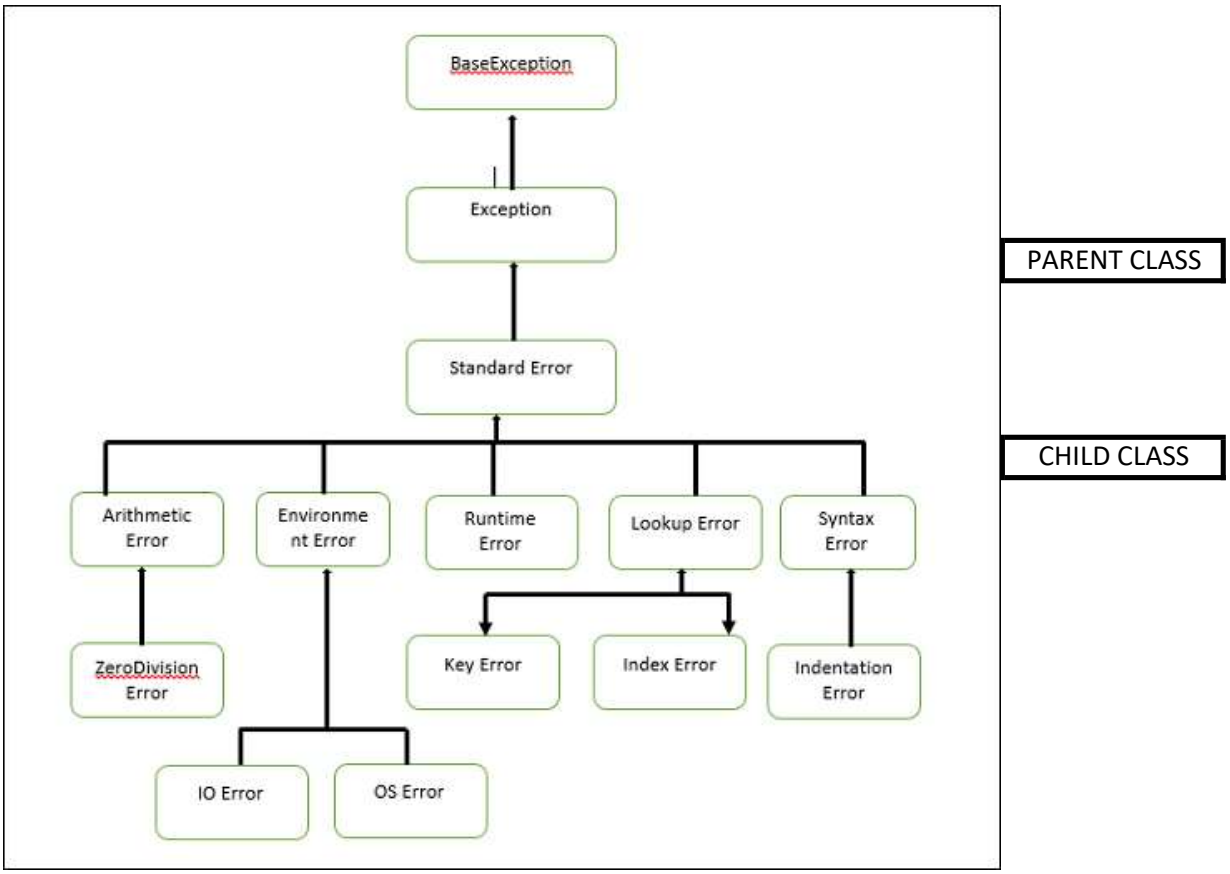


Your function will be executed

obj,abc()

3 pillars





Exception handling cannot rectify syntax error as it is checked by the compiler and handling can be done only at interpreter level.

Note: While mentioning class always make sure that you define child class before parent class

In case if it is mentioned vice versa then child class exception will not be executed and only parent class exception will run.

Note: You can also use traceback.format_exc() in place of sys.exc_info() and both the commands are used for debugging the entire project