

GeeksforGeeks

A computer science portal for geeks

[Practice](#)[GATE CS](#)[Placements](#)[Videos](#)[Contribute](#)[Login/Register](#)

Quick Links for Python

[Recent Articles](#)[MCQ / Quizzes](#)[Practice Problems](#)

Basics

[Introduction](#)[New Generation Language](#)[Keywords, Set 1 Set 2](#)[Explore More...](#)

Variables

[Variables, Expressions & Functions](#)[Global and Local Variables](#)[Type Conversion](#)[Explore More...](#)

Operators

[Increment and Decrement Operator](#)[Teranry Operator & Divison Operator](#)[Logical and Bitwise Not Operators on Boolean](#)[Any & ALL](#)

Operator Functions Set 1 & Set 2
Data Types
Introduction
Arrays Set 1, Set 2
String Methods Set 1, Set 2, Set 3
String Template Class & String Formatting using %
List Methods Set 1, Set 2, Set 3
Tuples & Sets
Dictionary Methods Set 1, Set 2
ChainMap
Explore More...
Control Flow
Loops and Control Statements
Counters & Accessing Counters
Iterators & Iterator Functions Set 1, Set 2
Generators
Explore More...
Functions
Function Decorators
Returning Multiple Values
Yield instead of Return
Python Closures & Coroutine
Explore More...
Modules
Introduction



Numeric Functions & Logarithmic and Power functions
Calender Functions Set 1, Set 2
Complex Numbers Introduction & Important functions
Explore More...
Object Oriented Concepts
Class, Object and Members
Data Hiding and Object Printing
Inheritance, Subclass and super
Class method vs static method & Class or Static Variables
Explore More...
Exception Handling
Exception Handling
User-Defined Exceptions
Built-in Exceptions
Libraries and Functions
Timeit
Numpy Set 1, Set 2
Get and Post
import module & reload module
Collection Modules Deque, Namedtuple & Heap
Explore More...
Machine Learning with Python
Classifying data using Support Vector Machines(SVMs) in Python



K means Clustering
How to get synonyms/antonyms from NLTK WordNet in Python?
Explore More...
Misc
Sql using Python & MongoDB and Python
Json formatting & Python Virtual environment
Metaprogramming with Metaclasses in Python
Python Input Methods for Competitive Programming
Explore More...
Applications and Projects
Creating a proxy webserver Set 1, Set 2
Send Message to FB friend
Twitter Sentiment Analysis & Whatsapp using Python
Desktop Notifier & Junk File Organizer
Explore More...

Built-in Exceptions in Python

All instances in Python must be instances of a class that derives from **BaseException**. Two exception classes that are not related via subclassing are never equivalent, even if they have the same name. The built-in exceptions can be generated by the interpreter or built-in functions.

There are several built-in exceptions in Python that are raised when errors occur. These built-in exceptions can be viewed using the **local()** built-in functions as follows :

```
>>> locals()['__builtins__']
```



This returns a dictionary of built-in exceptions, functions and attributes.

Base Classes

The following exceptions are used mostly as base classes for other exceptions.

1. **exception BaseException**

This is the base class for all built-in exceptions. It is not meant to be directly inherited by user-defined classes. For, user-defined classes, Exception is used. This class is responsible for creating a string representation of the exception using str() using the arguments passed. An empty string is returned if there are no arguments.

- **args** : The args are the tuple of arguments given to the exception constructor.
- **with_traceback(tb)** : This method is usually used in exception handling. This method sets tb as the new traceback for the exception and returns the exception object.

Code :

```
try:
    ...
except SomeException:
    tb = sys.exc_info()[2]
    raise OtherException(...).with_traceback(tb)
```

2. **exception Exception**

This is the base class for all built-in non-system-exiting exceptions. All user-defined exceptions should also be derived from this class.

3. **exception ArithmeticError**

This class is the base class for those built-in exceptions that are raised for various arithmetic errors such as :

- OverflowError
- ZeroDivisionError
- FloatingPointError

Example :

```
try:
    a = 10/0
    print a
except ArithmeticError:
    print "This statement is raising an arithmetic exception."
else:
    print "Success."
```

Run on IDE

Output :

This statement is raising an arithmetic exception.



4. **exception BufferError**

This exception is raised when buffer related operations cannot be performed.

5. **exception LookupError**

This is the base class for those exceptions that are raised when a key or index used on a mapping or sequence is invalid or not found. The exceptions raised are :

- **KeyError**
- **IndexError**

Example :

```
try:
    a = [1, 2, 3]
    print a[3]
except LookupError:
    print "Index out of bound error."
else:
    print "Success"
```

[Run on IDE](#)

Output :

```
Index out of bound error.
```

Concrete exceptions

The following exceptions are the exceptions that are usually raised.

1. **exception AssertionError**

An AssertionError is raised when an assert statement fails.

Example :

```
assert False, 'The assertion failed'
```

Output :

```
Traceback (most recent call last):
  File "exceptions_AssertionError.py", line 12, in
    assert False, 'The assertion failed'
AssertionError: The assertion failed
```

2. **exception AttributeError**

An AttributeError is raised when an attribute reference or assignment fails such as when a non-existent attribute is referenced.

Example :

```
class Attributes(object):
    pass

object = Attributes()
```



```
print object.attribute
```

[Run on IDE](#)**Output :**

```
Traceback (most recent call last):
  File "d912bae549a2b42953bc62da114ae7a7.py", line 5, in
    print object.attribute
AttributeError: 'Attributes' object has no attribute 'attribute'
```

3. exception EOFError

An EOFError is raised when built-in functions like input() hits an end-of-file condition (EOF) without reading any data. The file methods like readline() return an empty string when they hit EOF.

Example :

```
while True:
    data = raw_input('Enter name : ')
    print 'Hello ', data
```

[Run on IDE](#)**Output :**

```
Enter Name :Hello Aditi
Enter Name :Traceback (most recent call last):
  File "exceptions_EOFError.py", line 13, in
    data = raw_input('Enter name :')
EOFError: EOF when reading a line
```

4. exception FloatingPointError

A FloatingPointError is raised when a floating point operation fails. This exception is always defined, but can only be raised when Python is configured with the-with-fpectl option, or the WANT_SIGFPE_HANDLER symbol is defined in the pyconfig.h file.

Example :

```
import math
print math.exp(1000)
```

[Run on IDE](#)**Output :**

```
Traceback (most recent call last):
  File "", line 1, in
FloatingPointError: in math_1
```



5. exception GeneratorExit

This exception directly inherits from BaseException instead of Exception since it is technically not an error. A GeneratorExit exception is raised when a generator or coroutine is closed.

Example :

```
def my_generator():
    try:
        for i in range(5):
            print 'Yielding', i
            yield i
    except GeneratorExit:
        print 'Exiting early'

g = my_generator()
print g.next()
g.close()
```

[Run on IDE](#)

Output :

```
Yielding 0
0
Exiting early
```

6. exception ImportError

An ImportError is raised when the import statement is unable to load a module or when the “from list” in from ... import has a name that cannot be found.

Example :

```
import module_does_not_exist
```

[Run on IDE](#)

Output :

```
Traceback (most recent call last):
  File "exceptions_ImportError_nomodule.py", line 12, in
    import module_does_not_exist
ImportError: No module named module_does_not_exist
```

Example :

```
from exceptions import Userexception
```

[Run on IDE](#)

Output :


```
Traceback (most recent call last):
  File "exceptions_ImportError_missingname.py", line 12, in
    from exceptions import Userexception
ImportError: cannot import name Userexception
```

7. exception ModuleNotFoundError

This is the subclass of ImportError which is raised by import when a module could not be found. It is also raised when None is found in sys.modules.

8. exception IndexError

An IndexError is raised when a sequence is referenced which is out of range.

Example :

```
array = [ 0, 1, 2 ]
print array[3]
```

[Run on IDE](#)

Output :

```
Traceback (most recent call last):
  File "exceptions_IndexError.py", line 13, in
    print array[3]
IndexError: list index out of range
```

9. exception KeyError

A KeyError is raised when a mapping key is not found in the set of existing keys.

Example :

```
array = { 'a':1, 'b':2 }
print array['c']
```

[Run on IDE](#)

Output :

```
Traceback (most recent call last):
  File "exceptions_KeyError.py", line 13, in
    print array['c']
KeyError: 'c'
```

10. exception KeyboardInterrupt

This error is raised when the user hits the interrupt key such as Control-C or Delete.

Example :

```
try:
    print 'Press Return or Ctrl-C:',
```



```
        ignored = raw_input()
except Exception, err:
    print 'Caught exception:', err
except KeyboardInterrupt, err:
    print 'Caught KeyboardInterrupt'
else:
    print 'No exception'
```

[Run on IDE](#)**Output :**

```
Press Return or Ctrl-C: ^CCaught KeyboardInterrupt
```

11. exception MemoryError

This error is raised when an operation runs out of memory.

Example :

```
def fact(a):
    factors = []
    for i in range(1, a+1):
        if a%i == 0:
            factors.append(i)
    return factors
```

```
num = 600851475143
print fact(num)
```

[Run on IDE](#)**Output :**

```
Traceback (most recent call last):
  File "4af5c316c749aff128df20714536b8f3.py", line 9, in
    print fact(num)
  File "4af5c316c749aff128df20714536b8f3.py", line 3, in fact
    for i in range(1, a+1):
MemoryError
```

12. exception NameError

This error is raised when a local or global name is not found. For example, an unqualified variable name.

Example :

```
def func():
    print ans
```

```
func()
```

[Run on IDE](#)

Output :

```
Traceback (most recent call last):
  File "cfba0a5196b05397e0a23b1b5b8c7e19.py", line 4, in
    func()
  File "cfba0a5196b05397e0a23b1b5b8c7e19.py", line 2, in func
    print ans
NameError: global name 'ans' is not defined
```

13. exception NotImplementedError

This exception is derived from RuntimeError. Abstract methods in user defined classed should raise this exception when the derived classes override the method.

Example :

```
class BaseClass(object):
    """Defines the interface"""
    def __init__(self):
        super(BaseClass, self).__init__()
    def do_something(self):
        """The interface, not implemented"""
        raise NotImplementedError(self.__class__.__name__ + '.do_something')

class SubClass(BaseClass):
    """Implementes the interface"""
    def do_something(self):
        """really does something"""
        print self.__class__.__name__ + ' doing something!'
```

```
SubClass().do_something()
BaseClass().do_something()
```

[Run on IDE](#)**Output :**

```
Traceback (most recent call last):
  File "b32fc445850cbc23cd2f081ba1c1d60b.py", line 16, in
    BaseClass().do_something()
  File "b32fc445850cbc23cd2f081ba1c1d60b.py", line 7, in do_something
    raise NotImplementedError(self.__class__.__name__ + '.do_something')
NotImplementedError: BaseClass.do_something
```

14. exception OSError([arg])

The OSError exception is raised when a system function returns a system-related error, including I/O failures such as "file not found" or "disk full" errors.

Example :

```
def func():
    print ans

func()
```



[Run on IDE](#)

```
Traceback (most recent call last):
  File "442eccd7535a2704adbe372cb731fc0f.py", line 4, in
    print i, os.ttyname(i)
OSError: [Errno 25] Inappropriate ioctl for device
```

15. exception OverflowError

The OverflowError is raised when the result of an arithmetic operation is out of range. Integers raise MemoryError instead of OverflowError. OverflowError is sometimes raised for integers that are outside a required range. Floating point operations are not checked because of the lack of standardization of floating point exception handling in C.

Example :

```
import sys

print 'Regular integer: (maxint=%s)' % sys.maxint
try:
    i = sys.maxint * 3
    print 'No overflow for ', type(i), 'i =', i
except OverflowError, err:
    print 'Overflowed at ', i, err

print
print 'Long integer:'
for i in range(0, 100, 10):
    print '%2d' % i, 2L ** i

print
print 'Floating point values:'
try:
    f = 2.0**i
    for i in range(100):
        print i, f
        f = f ** 2
except OverflowError, err:
    print 'Overflowed after ', f, err
```

[Run on IDE](#)

Output :

```
Regular integer: (maxint=9223372036854775807)
No overflow for   i = 27670116110564327421

Long integer:
0 1
10 1024
20 1048576
30 1073741824
40 1099511627776
50 1125899906842624
60 1152921504606846976
```



```

70 1180591620717411303424
80 1208925819614629174706176
90 1237940039285380274899124224

Floating point values:
0 1.23794003929e+27
1 1.53249554087e+54
2 2.34854258277e+108
3 5.5156522631e+216
Overflowed after 5.5156522631e+216 (34, 'Numerical result out of range')

```

16. exception RecursionError

The RecursionError is derived from the RuntimeError. This exception is raised when the interpreter detects that the maximum recursion depth is exceeded.

17. exception ReferenceError

The ReferenceError is raised when a weak reference proxy is used to access an attribute of the referent after the garbage collection.

Example :

```

import gc
import weakref

class Foo(object):

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

    def __del__(self):
        print '(Deleting %s)' % self

obj = Foo('obj')
p = weakref.proxy(obj)

print 'BEFORE:', p.name
obj = None
print 'AFTER:', p.name

```

[Run on IDE](#)
Output :

```

BEFORE: obj
(Deleting )
AFTER:

Traceback (most recent call last):
  File "49d0c29d8fe607b862c02f4e1cb6c756.py", line 17, in
    print 'AFTER:', p.name
ReferenceError: weakly-referenced object no longer exists

```

18. exception RuntimeError

The RuntimeError is raised when no other exception applies. It returns a string indicating what precisely

went wrong.

19. exception StopIteration

The StopIteration error is raised by built-in function next() and an iterator's __next__() method to signal that all items are produced by the iterator.

Example :

```
Arr = [3, 1, 2]
i=iter(Arr)

print i
print i.next()
print i.next()
print i.next()
print i.next()
```

[Run on IDE](#)

Output :

```
3
1
2
```

```
Traceback (most recent call last):
  File "2136fa9a620e14f8436bb60d5395cc5b.py", line 8, in
    print i.next()
StopIteration
```

20. exception SyntaxError

The SyntaxError is raised when the parser encounters a syntax error. A syntax error may occur in an import statement or while calling the built-in functions exec() or eval(), or when reading the initial script or standard input.

Example :

```
try:
    print eval('geeks for geeks')
except SyntaxError, err:
    print 'Syntax error %s (%s-%s): %s' % \
        (err.filename, err.lineno, err.offset, err.text)
    print err
```

[Run on IDE](#)

Output :

```
Syntax error (1-9): geeks for geeks
invalid syntax (, line 1)
```



21. exception SystemError

The SystemError is raised when the interpreter finds an internal error. The associated value is a string indicating what went wrong.

22. exception SystemExit

The SystemExit is raised when sys.exit() function is called. A call to sys.exit() is translated into an exception to execute clean-up handlers (finally clauses of try statements) and to debug a script without running the risk of losing control.

23. exception TypeError

TypeError is raised when an operation or function is applied to an object of inappropriate type. This exception returns a string giving details about the type mismatch.

Example :

```
arr = ('tuple', ) + 'string'
print arr
```

[Run on IDE](#)**Output :**

```
Traceback (most recent call last):
  File "30238c120c0868eba7e13a06c0b1b1e4.py", line 1, in
    arr = ('tuple', ) + 'string'
TypeError: can only concatenate tuple (not "str") to tuple
```

24. exception UnboundLocalError

UnboundLocalError is a subclass of NameError which is raised when a reference is made to a local variable in a function or method, but no value has been assigned to that variable.

Example :

```
def global_name_error():
    print unknown_global_name

def unbound_local():
    local_val = local_val + 1
    print local_val

try:
    global_name_error()
except NameError, err:
    print 'Global name error:', err

try:
    unbound_local()
except UnboundLocalError, err:
    print 'Local name error:', err
```

[Run on IDE](#)**Output :**

```
Global name error: global name 'unknown_global_name' is not defined
Local name error: local variable 'local_val' referenced before assignment
```

25. exception UnicodeError

This exception is a subclass of ValueError. UnicodeError is raised when a Unicode-related encoding or decoding error occurs.

26. exception ValueError

A ValueError is raised when a built-in operation or function receives an argument that has the right type but an invalid value.

Example :

```
print int('a')
```

[Run on IDE](#)

Output :

```
Traceback (most recent call last):
  File "44f00efda935715a3c5468d899080381.py", line 1, in
    print int('a')
ValueError: invalid literal for int() with base 10: 'a'
```

27. exception ZeroDivisionError

A ZeroDivisionError is raised when the second argument of a division or modulo operation is zero. This exception returns a string indicating the type of the operands and the operation.

Example :

```
print 1/0
```

[Run on IDE](#)

Output :

```
Traceback (most recent call last):
  File "c31d9626b41e53d170a78eac7d98cb85.py", line 1, in
    print 1/0
ZeroDivisionError: integer division or modulo by zero
```

This article is contributed by **Aditi Gupta**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Python](#)[Python-Library](#)

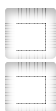
Recommended Posts:

[User-defined Exceptions in Python with Examples](#)[Python | Set 5 \(Exception Handling\)](#)[loops in python](#)[Object Oriented Programming in Python | Set 1 \(Class, Object and Members\)](#)[Object Oriented Programming in Python | Set 2 \(Data Hiding and Object Printing\)](#)

([Login](#) to Rate and Mark)

3

Average Difficulty : **3/5.0**
Based on **1** vote(s)



Add to TODO List



Mark as DONE


Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Share this post!



0 Comments

GeeksforGeeks

 Login ▾ Recommend Share

Sort by Newest ▾



LOG IN WITH

OR SIGN UP WITH DISQUS 

Be the first to comment.

@geeksforgeeks, Some rights reserved

[Contact Us!](#)[About Us!](#)[Careers!](#)[Privacy Policy](#)