Built-in Exceptions - https://docs.python.org/3/library/exceptions.html

Python - Error Types

The most common reason of an error in a Python program is when a certain statement is not in accordance with the prescribed usage. Such an error is called a syntax error. The Python interpreter immediately reports it, usually along with the reason.

Example: Error — Copy

```
>>> print "hello"
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello")?
```

In Python 3.x, print is a built-in function and requires parentheses. The statement above violates this usage and hence syntax error is displayed.

Many times though, a program results in an error after it is run even if it doesn't have any syntax error. Such an error is a runtime error, called an exception. A number of built-in exceptions are defined in the Python library. Let's see some common error types.

The following table lists important built-in exceptions in Python.

ImportError

The `ImportError` is thrown when a specified function can not be found.

Example: ImportError — Copy

```
>>> from math import cube
Traceback (most recent call last):
File "<pyshell#16>", line 1, in <module>

from math import cube
ImportError: cannot import name 'cube'
```

Exception	Description
AssertionError	Raised when the assert statement fails.
AttributeError	Raised on the attribute assignment or reference fails.
EOFError	Raised when the input() function hits the end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raised when a generator's close() method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when the index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits the interrupt key (Ctrl+c or delete).
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in the local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when a system operation causes a system-related error.
OverflowError	Raised when the result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by the next() function to indicate that there is no further item to be returned by the iterator.
SyntaxError	Raised by the parser when a syntax error is encountered.
IndentationError	Raised when there is an incorrect indentation.
TabError	Raised when the indentation consists of inconsistent tabs and spaces.
SystemError	Raised when the interpreter detects internal error.
SystemExit	Raised by the sys.exit() function.
TypeError	Raised when a function or operation is applied to an object of an incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translation.
ValueError	Raised when a function gets an argument of correct type but improper value.
ZeroDivisionError	Raised when the second operand of a division or module operation is zero.

IndexError

The `IndexError` is thrown when trying to access an item at an invalid index.

Example: IndexError Copy

```
>>> L1=[1,2,3]
>>> L1[3]
Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>

L1[3]
IndexError: list index out of range
```

ModuleNotFoundError

The `ModuleNotFoundError` is thrown when a module could not be found.

Example: ModuleNotFoundError Copy

```
>>> import notamodule
Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>

import notamodule
ModuleNotFoundError: No module named 'notamodule'
```

KeyError

The `KeyError` is thrown when a key is not found.

Example: KeyError Copy

```
>>> D1={'1':"aa", '2':"bb", '3':"cc"}
>>> D1['4']
Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>


D1['4']
KeyError: '4'
```

StopIteration

The `StopIteration` is thrown when the `next()` function goes beyond the iterator items.

Example: StopIteration

Copy

```
>>> it=iter([1,2,3])
>>> next(it)
1
>>> next(it)
2
>>> next(it)
3
>>> next(it)
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>

next(it)
StopIteration
```

TypeError

The `TypeError` is thrown when an operation or function is applied to an object of an inappropriate type.

Example: TypeError

Copy

```
>>> '2'+2
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>

'2'+2
TypeError: must be str, not int
```

ValueError

The `ValueError` is thrown when a function's argument is of an inappropriate type.

Example: ValueError

Copy

```
>>> int('xyz')
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module>

int('xyz')
ValueError: invalid literal for int() with base 10: 'xyz'
```

NameError

The `NameError` is thrown when an object could not be found.

Example: NameError

Copy

```
>>> age
Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>

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

ZeroDivisionError

The `ZeroDivisionError` is thrown when the second operator in the division is zero.

Example: ZeroDivisionError

Copy

```
>>> x=100/0
Traceback (most recent call last):
File "<pyshell#8>", line 1, in <module>

x=100/0
ZeroDivisionError: division by zero
```

KeyboardInterrupt

The `KeyboardInterrupt` is thrown when the user hits the interrupt key (normally Control-C) during the execution of the program.

Example: KeyboardInterrupt

Copy

```
>>> name=input('enter your name')
enter your name^c
Traceback (most recent call last):
File "<pyshell#9>", line 1, in <module>

name=input('enter your name')
KeyboardInterrupt
```

Python uses `try` and `except` keywords to handle exceptions. Both keywords are followed by indented blocks.

Syntax:

```
try :
    #statements in try block
except :
    #executed when error in try block
```

Example: Catch Specific Error Type

Copy

```
try:
    a=5
    b='0'
    print (a+b)
except TypeError:
    print('Unsupported operation')
print ("Out of try except blocks")
```

As mentioned above, a single try block may have multiple except blocks. The following example uses two except blocks to process two different exception types:

Example: Multiple except Blocks

Copy

```
try:
    a=5
    b=0
    print (a/b)
except TypeError:
    print('Unsupported operation')
except ZeroDivisionError:
    print ('Division by zero not allowed')
print ('Out of try except blocks')
```

In Python, keywords `else` and `finally` can also be used along with the try and except clauses. While the except block is executed if the exception occurs inside the try block, the else block gets processed if the try block is found to be exception free.

Syntax:

```
try:
    #statements in try block
except:
    #executed when error in try block
else:
    #executed if try block is error-free
finally:
    #executed irrespective of exception occured or not
```

Python also provides the `raise` keyword to be used in the context of exception handling. It causes an exception to be generated explicitly. Built-in errors are raised implicitly. However, a built-in or custom exception can be forced during execution.

The following code accepts a number from the user. The try block raises a ValueError exception if the number is outside the allowed range.

Example: Raise an Exception

Copy

```
try:
    x=int(input('Enter a number upto 100: '))
    if x > 100:
        raise ValueError(x)
except ValueError:
    print(x, "is out of allowed range")
else:
    print(x, "is within the allowed range")
```