The Python Exception Class Hierarchy

The Python exception class hierarchy consists of a few dozen different exceptions spread across a handful of important base class types. As with most programming languages, errors occur within a Python application when something unexpected goes wrong. Anything from improper arithmetic and running out of memory to invalid file references and unicode formatting errors may be raised by Python under certain circumstances.

Most of the errors we'll explore in this series are considered , which indicate that these are errors. While a error will halt execution of the current application, all non-fatal exceptions allow execution to continue. This allows our code to explicitly catch or the raised exception and programmatically react to it in an appropriate manner.

Let's start by looking at the full Python exception class hierarchy, as seen below:

- BaseException
  - Exception
    - ArithmeticError
      - FloatingPointError
      - OverflowError
      - ZeroDivisionError
    - AssertionError
    - AttributeError
    - BufferError
    - EOFError
    - ImportError
      - ModuleNotFoundError
    - LookupError
      - IndexError
      - KeyError
    - MemoryError
    - NameError
      - UnboundLocalError
    - OSError
      - BlockingIOError
      - ChildProcessError
      - ConnectionError
        - BrokenPipeError
        - ConnectionAbortedError
        - ConnectionRefusedError
        - ConnectionResetError
      - FileExistsError
      - FileNotFoundError
      - InterruptedError
      - IsADirectoryError
      - NotADirectoryError
      - PermissionError

                - ProcessLookupError
                - TimeoutError
            - ReferenceError
            - RuntimeError
                - NotImplementedError
                - RecursionError
            - StopIteration
            - StopAsyncIteration
            - SyntaxError
                - IndentationError
                    - TabError
            - SystemError
            - TypeError
            - ValueError
                - UnicodeError
                    - UnicodeDecodeError
                    - UnicodeEncodeError
                    - UnicodeTranslateError
            - Warning
                - BytesWarning
                - DeprecationWarning
                - FutureWarning
                - ImportWarning
                - PendingDeprecationWarning
                - ResourceWarning
                - RuntimeWarning
                - SyntaxWarning
                - UnicodeWarning
                - UserWarning
        - GeneratorExit
        - KeyboardInterrupt
        - SystemExit

As we publish future exception-specific articles in this series we'll update the full list above to relevant tutorial and article links for each exception, so this post can act as a go-to resource for Python exception handling tips.

Major Exception Types Overview

Next, let's briefly discuss each important top-level exception type. These top-level exceptions will serve as a basis for digging into specific exceptions in future articles. Before we do that, however, it's worth pointing out what might appear as a slight discrepancy when looking over the list of exception classes provided in Python. To illustrate, look closely at this small snippet of the Python exception class hierarchy and see if anything slightly strange pops out to you:

- BaseException

- Exception
    - ArithmeticError
        - FloatingPointError
        - OverflowError
        - ZeroDivisionError
    - AssertionError

For developers that have worked with other programming languages in the past, what you might take note of is the distinction between using the word in the and parent classes, and the use of in most subclasses therein. Most other languages, such as .NET or Java, explicitly differentiate between and by separating them into distinct categories. In such languages, typically denote errors (those that crash the application), whereas are catchable/rescuable errors.

Yet, as we see in the hierarchy above, Python merely inherits from with a series of classes. The reason for this naming convention comes from the Python style guide, which makes an explicit mention that “you should use the suffix ‘Error’ on your exception names (if the exception is actually an error).” I’ve added the extra emphasis to that quote, because the latter point is critical here — most Python exceptions with in the name are, in fact, errors.

BaseException

The class is, as the name suggests, the base class for all built-in exceptions in Python. Typically, this exception is never raised on its own, and should instead be inherited by other, lesser exception classes that can be raised.

The class (and, thus, all subclass exceptions as well) allows a of arguments to be passed when creating a new instance of the class. In most cases, a single argument will be passed to an exception, which is a string value indicating the specific error message.

This class also includes a method, which explicitly sets the new traceback information to the argument that was passed to it.

Exception

is the most commonly-inherited exception type (outside of the true base class of). In addition, all exception classes that are considered errors are subclasses of the class. In general, any custom exception class you create in your own code should inherit from .

The class contains many direct child subclasses that handle most Python errors, so we’ll briefly go over each below:

- – The base class for the variety of arithmetic errors, such as when attempting to divide by zero, or when an arithmetic result would be too large for Python to accurately represent.
- – This error is raised when a call to the [] statement fails.
- – Python’s syntax includes something called , which is just the Python way of describing what you might know of as . In essence, any in Python (like an , , and so forth) can be written and followed by a period (), which is then followed by an . That

syntax (i.e.) is called an , and anytime the executing script encounters an error in such syntax an is raised.

- – Python allows applications to access low level memory streams in the form of a . For example, the class can be used to directly work with bytes of information via a memory buffer. When something goes wrong within such a buffer operation a is raised.
- – Similar to the Java EOFException article we did a few days ago, Python's is raised when using the function and reaching the end of a file without any data.
- – Modules are usually loaded in memory for Python scripts to use via the statement (e.g.). However, if an attempt fails an will often be raised.
- – Like , the is generally considered a base class from which other subclasses should inherit. All subclasses deal with improper calls to a collection are made by using invalid or values.
- – In the event that your Python application is about to run out of memory a will be raised. Since Python is smart enough to detect this potential issue slightly before all memory is used up, a can be and allow you to recover from the situation by performing garbage collection of some kind.
- – Raised when trying to use an with an invalid or unknown name.
- – This error is raised when a system-level problem occurs, such as failing to find a local file on disk or running out of disk space entirely. is a parent class to many subclasses explicitly used for certain issues related to operating system failure, so we'll explore those in future publications.
- – Python includes the weakref module, which allows Python code to create a specific type of reference known as a . A is a reference that is not "strong" enough to keep the referenced object alive. This means that the next cycle of garbage collection will identify the weakly referenced object as no longer strongly referenced by another object, causing the weakly referenced object to be destroyed to free up resources. If a proxy created via the function is used after the object that is referenced has already been destroyed via garbage collection, a will be raised.
- – A is typically used as a catchall for when an error occurs that doesn't really fit into any other specific error classification.
- – If no value is passed to the function when iterating over a collection, and that collection has no more iterated value to retrieve, a exception is raised. Note that this is not classified as an , since it doesn't mean that an error has occurred.
- – As of version 3.5, Python now includes coroutines for asynchronous transactions using the and syntax. As part of this feature, collections can be asynchronously iterated using the method. The method requires that a instance be raised in order to halt async iteration.
- – Just like most programming languages, a in Python indicates that there is some improper syntax somewhere in your script file. A can be raised directly from an executing script, or produced via functions like and .
- – A generic error that is raised when something goes wrong with the Python interpreter (not to be confused with the , which handles operating system issues).
- – This error is raised when attempting to perform an operation on an incorrect object type.

- – Should be raised when a function or method receives an argument of the correct type, but with an actual value that is invalid for some reason.
- – Another parent class to many subclasses, the class is used to alert the user in non-dire situations. There are a number of subclass warnings that we'll explore in future articles.

GeneratorExit

A is a specific type of iterator in Python, which simplifies the process of creating iterators with constantly changing values. By using the statement within a generator code block, Python will return or "generate" a new value for each call to . When the explicit method is called a instance is raised.

KeyboardInterrupt

This simple exception is raised when the user presses a key combination that causes an to the executing script. For example, many terminals accept as an interrupt keystroke.

SystemExit

Finally, the exception is raised when calling the method, which explicitly closes down the executing script and exits Python. Since this is an exception, it can be and programmatically responded to immediately before the script actually shuts down.