Introduction

This book is intended to be a concise reference to the Python programming language. Although an experienced programmer will probably be able to learn Python from this book, it’s not intended to be an extended tutorial or a treatise on how to program. Rather, the goal is to present the core Python language, and the most essential parts of the Python library in a manner that’s accurate and concise. This book assumes that the reader has prior programming experience with Python or another language such as C or Java. In addition, a general familiarity with systems programming topics (for example, basic operating system concepts and network programming) may be useful in understanding certain parts of the library reference.

Python is freely available for download at http://www.python.org. Versions are available for almost every operating system, including UNIX, Windows, and Macintosh. In addition, the Python website includes links to documentation, how-to guides, and a wide assortment of third-party software.

This edition of *Python Essential Reference* comes at a pivotal time in Python’s evolution. Python 2.6 and Python 3.0 are being released almost simultaneously. Yet, Python 3 is a release that breaks backwards compatibility with prior Python versions. As an author and programmer, I’m faced with a dilemma: do I simply jump forward to Python 3.0 or do I build upon the Python 2.x releases that are more familiar to most programmers?

Years ago, as a C programmer I used to treat certain books as the ultimate authority on what programming language features should be used. For example, if you were using something that wasn’t documented in the K&R book, it probably wasn’t going to be portable and should be approached with caution. This approach served me very well as a programmer and it’s the approach I have decided to take in this edition of the Essential Reference. Namely, I have chosen to omit features of Python 2 that have been removed from Python 3. Likewise, I don’t focus on features of Python 3 that have not been back-ported (although such features are still covered in an appendix). As a result, I hope this book can be a useful companion for Python programmers, regardless of what Python version is being used.

The fourth edition of *Python Essential Reference* also includes some of the most exciting changes since its initial publication nearly ten years ago. Much of Python’s development throughout the last few years has focused on new programming language features—especially related to functional and meta programming. As a result, the chapters on functions and object-oriented programming have been greatly expanded to cover topics such as generators, iterators, coroutines, decorators, and metaclasses. The library chapters have been updated to focus on more modern modules. Examples and code fragments have also been updated throughout the book. I think most programmers will be quite pleased with the expanded coverage.

Finally, it should be noted that Python already includes thousands of pages of useful documentation. The contents of this book are largely based on that documentation, but with a number of key differences. First, this reference presents information in a much more compact form, with different examples and alternative descriptions of many topics. Second, a significant number of topics in the library reference have been expanded...
to include outside reference material. This is especially true for low-level system and networking modules in which effective use of a module normally relies on a myriad of options listed in manuals and outside references. In addition, in order to produce a more concise reference, a number of deprecated and relatively obscure library modules have been omitted.

In writing this book, it has been my goal to produce a reference containing virtually everything I have needed to use Python and its large collection of modules. Although this is by no means a gentle introduction to the Python language, I hope that you find the contents of this book to be a useful addition to your programming reference library for many years to come. I welcome your comments.

David Beazley
Chicago, Illinois
June, 2009
All the data stored in a Python program is built around the concept of an object. Objects include fundamental data types such as numbers, strings, lists, and dictionaries. However, it’s also possible to create user-defined objects in the form of classes. In addition, most objects related to program structure and the internal operation of the interpreter are also exposed. This chapter describes the inner workings of the Python object model and provides an overview of the built-in data types. Chapter 4, “Operators and Expressions,” further describes operators and expressions. Chapter 7, “Classes and Object-Oriented Programming,” describes how to create user-defined objects.

Terminology

Every piece of data stored in a program is an object. Each object has an identity, a type (which is also known as its class), and a value. For example, when you write `a = 42`, an integer object is created with the value of 42. You can view the identity of an object as a pointer to its location in memory. `a` is a name that refers to this specific location.

The type of an object, also known as the object’s class, describes the internal representation of the object as well as the methods and operations that it supports. When an object of a particular type is created, that object is sometimes called an instance of that type. After an instance is created, its identity and type cannot be changed. If an object’s value can be modified, the object is said to be mutable. If the value cannot be modified, the object is said to be immutable. An object that contains references to other objects is said to be a container or collection.

Most objects are characterized by a number of data attributes and methods. An attribute is a value associated with an object. A method is a function that performs some sort of operation on an object when the method is invoked as a function. Attributes and methods are accessed using the dot (.) operator, as shown in the following example:

```python
a = 3 + 4j          # Create a complex number
r = a.real          # Get the real part (an attribute)

b = [1, 2, 3]       # Create a list
b.append(7)         # Add a new element using the append method
```

Object Identity and Type

The built-in function `id()` returns the identity of an object as an integer. This integer usually corresponds to the object’s location in memory, although this is specific to the Python implementation and no such interpretation of the identity should be made. The
is operator compares the identity of two objects. The built-in function type() returns the type of an object. Here’s an example of different ways you might compare two objects:

```python
# Compare two objects
def compare(a,b):
    if a is b:
        # a and b are the same object
        statements
    if a == b:
        # a and b have the same value
        statements
    if type(a) is type(b):
        # a and b have the same type
        statements
```

The type of an object is itself an object known as the object’s class. This object is uniquely defined and is always the same for all instances of a given type. Therefore, the type can be compared using the is operator. All type objects are assigned names that can be used to perform type checking. Most of these names are built-ins, such as list, dict, and file. Here’s an example:

```python
if type(s) is list:
s.append(item)
```

```python
if type(d) is dict:
d.update(t)
```

Because types can be specialized by defining classes, a better way to check types is to use the built-in isinstance(object, type) function. Here’s an example:

```python
if isinstance(s, list):
s.append(item)
```

```python
if isinstance(d, dict):
d.update(t)
```

Because the isinstance() function is aware of inheritance, it is the preferred way to check the type of any Python object.

Although type checks can be added to a program, type checking is often not as useful as you might imagine. For one, excessive checking severely affects performance. Second, programs don’t always define objects that neatly fit into an inheritance hierarchy. For instance, if the purpose of the preceding isinstance(s, list) statement is to test whether s is “list-like,” it wouldn’t work with objects that had the same programming interface as a list but didn’t directly inherit from the built-in list type. Another option for adding type-checking to a program is to define abstract base classes. This is described in Chapter 7.

Reference Counting and Garbage Collection

All objects are reference-counted. An object’s reference count is increased whenever it’s assigned to a new name or placed in a container such as a list, tuple, or dictionary, as shown here:

```python
a = 37       # Creates an object with value 37
b = a        # Increases reference count on 37
c = []
c.append(b) # Increases reference count on 37
```
This example creates a single object containing the value 37. a is merely a name that refers to the newly created object. When b is assigned a, b becomes a new name for the same object and the object’s reference count increases. Likewise, when you place b into a list, the object’s reference count increases again. Throughout the example, only one object contains 37. All other operations are simply creating new references to the object.

An object’s reference count is decreased by the del statement or whenever a reference goes out of scope (or is reassigned). Here’s an example:

```python
del a      # Decrease reference count of 37
b = 42     # Decrease reference count of 37
c[0] = 2.0 # Decrease reference count of 37
```

The current reference count of an object can be obtained using the `sys.getrefcount()` function. For example:

```python
>>> a = 37
>>> import sys
>>> sys.getrefcount(a)
7
```

In many cases, the reference count is much higher than you might guess. For immutable data such as numbers and strings, the interpreter aggressively shares objects between different parts of the program in order to conserve memory.

When an object’s reference count reaches zero, it is garbage-collected. However, in some cases a circular dependency may exist among a collection of objects that are no longer in use. Here’s an example:

```python
a = { }
b = { }
a['b'] = b      # a contains reference to b
b['a'] = a      # b contains reference to a
del a
del b
```

In this example, the del statements decrease the reference count of a and b and destroy the names used to refer to the underlying objects. However, because each object contains a reference to the other, the reference count doesn’t drop to zero and the objects remain allocated (resulting in a memory leak). To address this problem, the interpreter periodically executes a cycle detector that searches for cycles of inaccessible objects and deletes them. The cycle-detection algorithm runs periodically as the interpreter allocates more and more memory during execution. The exact behavior can be fine-tuned and controlled using functions in the `gc` module (see Chapter 13, “Python Runtime Services”).

### References and Copies

When a program makes an assignment such as `a = b`, a new reference to b is created. For immutable objects such as numbers and strings, this assignment effectively creates a copy of b. However, the behavior is quite different for mutable objects such as lists and dictionaries. Here’s an example:

```python
>>> a = [1,2,3,4]
>>> b = a       # b is a reference to a
>>> b is a
True
```
Because `a` and `b` refer to the same object in this example, a change made to one of the variables is reflected in the other. To avoid this, you have to create a copy of an object rather than a new reference.

Two types of copy operations are applied to container objects such as lists and dictionaries: a shallow copy and a deep copy. A shallow copy creates a new object but populates it with references to the items contained in the original object. Here’s an example:

```python
>>> a = [1, 2, [3, 4]]
>>> b = list(a)  # Create a shallow copy of a.
>>> b is a False
>>> b.append(100)  # Append element to b.
>>> b
[1, 2, [3, 4], 100]
>>> a
[1, 2, [3, 4]]
>>> b[2][0] = -100  # Modify an element inside b
>>> b
[1, 2, [-100, 4], 100]
>>> a
[1, 2, [3, 4]]
```

In this case, `a` and `b` are separate list objects, but the elements they contain are shared. Therefore, a modification to one of the elements of `a` also modifies an element of `b`, as shown.

A deep copy creates a new object and recursively copies all the objects it contains. There is no built-in operation to create deep copies of objects. However, the `copy.deepcopy()` function in the standard library can be used, as shown in the following example:

```python
>>> import copy
>>> a = [1, 2, [3, 4]]
>>> b = copy.deepcopy(a)
>>> b[2][0] = -100
>>> b
[1, 2, [-100, 4]]
>>> a
[1, 2, [3, 4]]
```

### First-Class Objects

All objects in Python are said to be “first class.” This means that all objects that can be named by an identifier have equal status. It also means that all objects that can be named can be treated as data. For example, here is a simple dictionary containing two values:

```python
items = {
    'number' : 42,
    'text' : "Hello World"
}
```
The first-class nature of objects can be seen by adding some more unusual items to this dictionary. Here are some examples:

```python
items["func"] = abs  # Add the abs() function
import math
items["mod"] = math    # Add a module
items["error"] = ValueError  # Add an exception type
nums = [1,2,3,4]
items["append"] = nums.append  # Add a method of another object
```

In this example, the `items` dictionary contains a function, a module, an exception, and a method of another object. If you want, you can use dictionary lookups on `items` in place of the original names and the code will still work. For example:

```python
>>> items["func"](-45)   # Executes abs(-45)
45
>>> items["mod"].sqrt(4)  # Executes math.sqrt(4)
2.0
>>> try:
...     x = int("a lot")
... except items["error"] as e:  # Same as except ValueError as e
...     print("Couldn't convert")
... Couldn't convert
>>> items["append"](100)  # Executes nums.append(100)
>>> nums
[1, 2, 3, 4, 100]
```

The fact that everything in Python is first-class is often not fully appreciated by new programmers. However, it can be used to write very compact and flexible code. For example, suppose you had a line of text such as "GOOG,100,490.10" and you wanted to convert it into a list of fields with appropriate type-conversion. Here’s a clever way that you might do it by creating a list of types (which are first-class objects) and executing a few simple list processing operations:

```python
>>> line = "GOOG,100,490.10"
>>> field_types = [str, int, float]
>>> raw_fields = line.split(',
')
>>> fields = [ty(val) for ty,val in zip(field_types,raw_fields)]
>>> fields
['GOOG', 100, 490.10000000000002]
```

**Built-in Types for Representing Data**

There are approximately a dozen built-in data types that are used to represent most of the data used in programs. These are grouped into a few major categories as shown in Table 3.1. The Type Name column in the table lists the name or expression that you can use to check for that type using `isinstance()` and other type-related functions. Certain types are only available in Python 2 and have been indicated as such (in Python 3, they have been deprecated or merged into one of the other types).
The None Type

The None type denotes a null object (an object with no value). Python provides exactly one null object, which is written as None in a program. This object is returned by functions that don’t explicitly return a value. None is frequently used as the default value of optional arguments, so that the function can detect whether the caller has actually passed a value for that argument. None has no attributes and evaluates to False in Boolean expressions.

Numeric Types

Python uses five numeric types: Booleans, integers, long integers, floating-point numbers, and complex numbers. Except for Booleans, all numeric objects are signed. All numeric types are immutable.

Booleans are represented by two values: True and False. The names True and False are respectively mapped to the numerical values of 1 and 0.

Integers represent whole numbers in the range of $-2147483648$ to $2147483647$ (the range may be larger on some machines). Long integers represent whole numbers of unlimited range (limited only by available memory). Although there are two integer types, Python tries to make the distinction seamless (in fact, in Python 3, the two types have been unified into a single integer type). Thus, although you will sometimes see references to long integers in existing Python code, this is mostly an implementation detail that can be ignored—just use the integer type for all integer operations. The one exception is in code that performs explicit type checking for integer values. In Python 2, the expression isinstance(x, int) will return False if x is an integer that has been promoted to a long.

Floating-point numbers are represented using the native double-precision (64-bit) representation of floating-point numbers on the machine. Normally this is IEEE 754, which provides approximately 17 digits of precision and an exponent in the range of

---

### Table 3.1 Built-In Types for Data Representation

<table>
<thead>
<tr>
<th>Type Category</th>
<th>Type Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>type(None)</td>
<td>The null object None</td>
</tr>
<tr>
<td>Numbers</td>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td>Arbitrary-precision integer (Python 2 only)</td>
</tr>
<tr>
<td></td>
<td>float</td>
<td>Floating point</td>
</tr>
<tr>
<td></td>
<td>complex</td>
<td>Complex number</td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td>Boolean (True or False)</td>
</tr>
<tr>
<td>Sequences</td>
<td>str</td>
<td>Character string</td>
</tr>
<tr>
<td></td>
<td>unicode</td>
<td>Unicode character string (Python 2 only)</td>
</tr>
<tr>
<td></td>
<td>list</td>
<td>List</td>
</tr>
<tr>
<td></td>
<td>tuple</td>
<td>Tuple</td>
</tr>
<tr>
<td></td>
<td>xrange</td>
<td>A range of integers created by xrange() (In Python 3, it is called range.)</td>
</tr>
<tr>
<td>Mapping</td>
<td>dict</td>
<td>Dictionary</td>
</tr>
<tr>
<td>Sets</td>
<td>set</td>
<td>Mutable set</td>
</tr>
<tr>
<td></td>
<td>frozenset</td>
<td>Immutable set</td>
</tr>
</tbody>
</table>
–308 to 308. This is the same as the double type in C. Python doesn’t support 32-bit single-precision floating-point numbers. If precise control over the space and precision of numbers is an issue in your program, consider using the numpy extension (which can be found at http://numpy.sourceforge.net).

Complex numbers are represented as a pair of floating-point numbers. The real and imaginary parts of a complex number \( z \) are available in \( z.real \) and \( z.imag \). The method \( z.conjugate() \) calculates the complex conjugate of \( z \) (the conjugate of \( a+bj \) is \( a-bj \)).

Numeric types have a number of properties and methods that are meant to simplify operations involving mixed arithmetic. For simplified compatibility with rational numbers (found in the fractions module), integers have the properties \( x.numerator \) and \( x.denominator \). An integer or floating-point number \( y \) has the properties \( y.real \) and \( y.imag \) as well as the method \( y.conjugate() \) for compatibility with complex numbers. A floating-point number \( y \) can be converted into a pair of integers representing a fraction using \( y.as_integer_ratio() \). The method \( y.is_integer() \) tests if a floating-point number \( y \) represents an integer value. Methods \( y.hex() \) and \( y.fromhex() \) can be used to work with floating-point numbers using their low-level binary representation.

Several additional numeric types are defined in library modules. The decimal module provides support for generalized base-10 decimal arithmetic. The fractions module adds a rational number type. These modules are covered in Chapter 14, “Mathematics.”

Sequence Types

Sequences represent ordered sets of objects indexed by non-negative integers and include strings, lists, and tuples. Strings are sequences of characters, and lists and tuples are sequences of arbitrary Python objects. Strings and tuples are immutable; lists allow insertion, deletion, and substitution of elements. All sequences support iteration.

Operations Common to All Sequences

Table 3.2 shows the operators and methods that you can apply to all sequence types. Element \( i \) of sequence \( s \) is selected using the indexing operator \( s[i] \), and subsequences are selected using the slicing operator \( s[i:j] \) or extended slicing operator \( s[i:j:stride] \) (these operations are described in Chapter 4). The length of any sequence is returned using the built-in \( len(s) \) function. You can find the minimum and maximum values of a sequence by using the built-in \( min(s) \) and \( max(s) \) functions. However, these functions only work for sequences in which the elements can be ordered (typically numbers and strings). \( sum(s) \) sums items in \( s \) but only works for numeric data.

Table 3.3 shows the additional operators that can be applied to mutable sequences such as lists.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s[i] )</td>
<td>Returns element ( i ) of a sequence</td>
</tr>
<tr>
<td>( s[i:j] )</td>
<td>Returns a slice</td>
</tr>
<tr>
<td>( s[i:j:stride] )</td>
<td>Returns an extended slice</td>
</tr>
</tbody>
</table>
Chapter 3 Types and Objects

Table 3.2  Continued

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len(s)</code></td>
<td>Number of elements in <code>s</code></td>
</tr>
<tr>
<td><code>min(s)</code></td>
<td>Minimum value in <code>s</code></td>
</tr>
<tr>
<td><code>max(s)</code></td>
<td>Maximum value in <code>s</code></td>
</tr>
<tr>
<td><code>sum(s [,initial])</code></td>
<td>Sum of items in <code>s</code></td>
</tr>
<tr>
<td><code>all(s)</code></td>
<td>Checks whether all items in <code>s</code> are True.</td>
</tr>
<tr>
<td><code>any(s)</code></td>
<td>Checks whether any item in <code>s</code> is True.</td>
</tr>
</tbody>
</table>

Table 3.3  Operations Applicable to Mutable Sequences

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s[i] = v</code></td>
<td>Item assignment</td>
</tr>
<tr>
<td><code>s[i:j] = t</code></td>
<td>Slice assignment</td>
</tr>
<tr>
<td><code>s[i:j:stride] = t</code></td>
<td>Extended slice assignment</td>
</tr>
<tr>
<td><code>del s[i]</code></td>
<td>Item deletion</td>
</tr>
<tr>
<td><code>del s[i:j]</code></td>
<td>Slice deletion</td>
</tr>
<tr>
<td><code>del s[i:j:stride]</code></td>
<td>Extended slice deletion</td>
</tr>
</tbody>
</table>

Lists

Lists support the methods shown in Table 3.4. The built-in function `list(s)` converts any iterable type to a list. If `s` is already a list, this function constructs a new list that’s a shallow copy of `s`. The `s.append(x)` method appends a new element, `x`, to the end of the list. The `s.index(x)` method searches the list for the first occurrence of `x`. If no such element is found, a `ValueError` exception is raised. Similarly, the `s.remove(x)` method removes the first occurrence of `x` from the list or raises `ValueError` if no such item exists. The `s.extend(t)` method extends the list `s` by appending the elements in sequence `t`.

The `s.sort()` method sorts the elements of a list and optionally accepts a key function and reverse flag, both of which must be specified as keyword arguments. The key function is a function that is applied to each element prior to comparison during sorting. If given, this function should take a single item as input and return the value that will be used to perform the comparison while sorting. Specifying a key function is useful if you want to perform special kinds of sorting operations such as sorting a list of strings, but with case insensitivity. The `s.reverse()` method reverses the order of the items in the list. Both the `sort()` and `reverse()` methods operate on the list elements in place and return `None`.

Table 3.4  List Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>list(s)</code></td>
<td>Converts <code>s</code> to a list.</td>
</tr>
<tr>
<td><code>s.append(x)</code></td>
<td>Appends a new element, <code>x</code>, to the end of <code>s</code>.</td>
</tr>
<tr>
<td><code>s.extend(t)</code></td>
<td>Appends a new list, <code>t</code>, to the end of <code>s</code>.</td>
</tr>
<tr>
<td><code>s.count(x)</code></td>
<td>Counts occurrences of <code>x</code> in <code>s</code>.</td>
</tr>
</tbody>
</table>
Table 3.4  Continued

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.index(x [,start [,stop]])</code></td>
<td>Returns the smallest $i$ where $s[i]==x$. start and stop optionally specify the starting and ending index for the search.</td>
</tr>
<tr>
<td><code>s.insert(i,x)</code></td>
<td>Inserts $x$ at index $i$.</td>
</tr>
<tr>
<td><code>s.pop([i])</code></td>
<td>Returns the element $i$ and removes it from the list. If $i$ is omitted, the last element is returned.</td>
</tr>
<tr>
<td><code>s.remove(x)</code></td>
<td>Searches for $x$ and removes it from $s$.</td>
</tr>
<tr>
<td><code>s.reverse()</code></td>
<td>Reverses items of $s$ in place.</td>
</tr>
<tr>
<td><code>s.sort([key [, reverse]])</code></td>
<td>Sorts items of $s$ in place. key is a key function. reverse is a flag that sorts the list in reverse order. key and reverse should always be specified as keyword arguments.</td>
</tr>
</tbody>
</table>

Strings

Python 2 provides two string object types. Byte strings are sequences of bytes containing 8-bit data. They may contain binary data and embedded NULL bytes. Unicode strings are sequences of unencoded Unicode characters, which are internally represented by 16-bit integers. This allows for 65,536 unique character values. Although the Unicode standard supports up to 1 million unique character values, these extra characters are not supported by Python by default. Instead, they are encoded as a special two-character (4-byte) sequence known as a surrogate pair—the interpretation of which is up to the application. As an optional feature, Python may be built to store Unicode characters using 32-bit integers. When enabled, this allows Python to represent the entire range of Unicode values from U+000000 to U+110000. All Unicode-related functions are adjusted accordingly.

Strings support the methods shown in Table 3.5. Although these methods operate on string instances, none of these methods actually modifies the underlying string data. Thus, methods such as `s.capitalize()`, `s.center()`, and `s.expandtabs()` always return a new string as opposed to modifying the string $s$. Character tests such as `s.isalnum()` and `s.isupper()` return True or False if all the characters in the string $s$ satisfy the test. Furthermore, these tests always return False if the length of the string is zero.

The `s.find()`, `s.index()`, `s.rfind()`, and `s.rindex()` methods are used to search $s$ for a substring. All these functions return an integer index to the substring in $s$. In addition, the `find()` method returns -1 if the substring isn’t found, whereas the `index()` method raises a ValueError exception. The `s.replace()` method is used to replace a substring with replacement text. It is important to emphasize that all of these methods only work with simple substrings. Regular expression pattern matching and searching is handled by functions in the `re` library module.

The `s.split()` and `s.rsplit()` methods split a string into a list of fields separated by a delimiter. The `s.partition()` and `s.rpartition()` methods search for a separator substring and partition $s$ into three parts corresponding to text before the separator, the separator itself, and text after the separator.

Many of the string methods accept optional `start` and `end` parameters, which are integer values specifying the starting and ending indices in $s$. In most cases, these values
may be given negative values, in which case the index is taken from the end of the string.

The \texttt{s.translate()} method is used to perform advanced character substitutions such as quickly stripping all control characters out of a string. As an argument, it accepts a translation table containing a one-to-one mapping of characters in the original string to characters in the result. For 8-bit strings, the translation table is a 256-character string. For Unicode, the translation table can be any sequence object \texttt{s} where \texttt{s[n]} returns an integer character code or Unicode character corresponding to the Unicode character with integer value \texttt{n}.

The \texttt{s.encode()} and \texttt{s.decode()} methods are used to transform string data to and from a specified character encoding. As input, these accept an encoding name such as \texttt{'ascii'}, \texttt{'utf-8'}, or \texttt{'utf-16'}. These methods are most commonly used to convert Unicode strings into a data encoding suitable for I/O operations and are described further in Chapter 9, “Input and Output.” Be aware that in Python 3, the \texttt{encode()} method is only available on strings, and the \texttt{decode()} method is only available on the bytes datatype.

The \texttt{s.format()} method is used to perform string formatting. As arguments, it accepts any combination of positional and keyword arguments. Placeholders in \texttt{s} denoted by \texttt{\{item\}} are replaced by the appropriate argument. Positional arguments can be referenced using placeholders such as \texttt{\{0\}} and \texttt{\{1\}}. Keyword arguments are referenced using a placeholder with a name such as \texttt{\{name\}}. Here is an example:

```python
>>> a = "Your name is \{0\} and your age is \{age\}"
>>> a.format("Mike", age=40)
'Your name is Mike and your age is 40'
```

Within the special format strings, the \texttt{\{item\}} placeholders can also include simple index and attribute lookup. A placeholder of \texttt{\{item[n]\}} where \texttt{n} is a number performs a sequence lookup on \texttt{item}. A placeholder of \texttt{\{item[key]\}} where \texttt{key} is a non-numeric string performs a dictionary lookup of \texttt{item["key"]}. A placeholder of \texttt{\{item.attr\}} refers to attribute \texttt{attr} of \texttt{item}. Further details on the \texttt{format()} method can be found in the “String Formatting” section of Chapter 4.

### Table 3.5 String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{s.capitalize()}</td>
<td>Capitalizes the first character.</td>
</tr>
<tr>
<td>\texttt{s.center(width [, pad])}</td>
<td>Centers the string in a field of length \texttt{width}. \texttt{pad} is a padding character.</td>
</tr>
<tr>
<td>\texttt{s.count(sub [, start [,end]])}</td>
<td>Counts occurrences of the specified substring \texttt{sub}.</td>
</tr>
<tr>
<td>\texttt{s.decode([encoding [,errors]])}</td>
<td>Decodes a string and returns a Unicode string (byte strings only).</td>
</tr>
<tr>
<td>\texttt{s.encode([encoding [,errors]])}</td>
<td>Returns an encoded version of the string (unicode strings only).</td>
</tr>
<tr>
<td>\texttt{s.endswith(suffix [,start [,end]])}</td>
<td>Checks the end of the string for a suffix.</td>
</tr>
<tr>
<td>\texttt{s.expandtabs([tabsize])}</td>
<td>Replaces tabs with spaces.</td>
</tr>
<tr>
<td>\texttt{s.find(sub [, start [,end]])}</td>
<td>Finds the first occurrence of the specified substring \texttt{sub} or returns -1.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>s.format(*args, **kwargs)</code></td>
<td>Formats s.</td>
</tr>
<tr>
<td><code>s.index(sub [, start [,end]])</code></td>
<td>Finds the first occurrence of the specified substring sub or raises an error.</td>
</tr>
<tr>
<td><code>s.isalnum()</code></td>
<td>Checks whether all characters are alphanumeric.</td>
</tr>
<tr>
<td><code>s.isalpha()</code></td>
<td>Checks whether all characters are alphabetic.</td>
</tr>
<tr>
<td><code>s.isdigit()</code></td>
<td>Checks whether all characters are digits.</td>
</tr>
<tr>
<td><code>s.islower()</code></td>
<td>Checks whether all characters are lowercase.</td>
</tr>
<tr>
<td><code>s.isspace()</code></td>
<td>Checks whether all characters are whitespace.</td>
</tr>
<tr>
<td><code>s.istitle()</code></td>
<td>Checks whether the string is a title-cased string (first letter of each word capitalized).</td>
</tr>
<tr>
<td><code>s.isupper()</code></td>
<td>Checks whether all characters are uppercase.</td>
</tr>
<tr>
<td><code>s.join(t)</code></td>
<td>Joins the strings in sequence t with s as a separator.</td>
</tr>
<tr>
<td><code>s.ljust(width [, fill])</code></td>
<td>Left-aligns s in a string of size width.</td>
</tr>
<tr>
<td><code>s.lower()</code></td>
<td>Converts to lowercase.</td>
</tr>
<tr>
<td><code>s.lstrip([chrs])</code></td>
<td>Removes leading whitespace or characters supplied in chrs.</td>
</tr>
<tr>
<td><code>s.partition(sep)</code></td>
<td>Partitions a string based on a separator string sep. Returns a tuple (head,sep,tail) or (s, &quot;&quot;,&quot;&quot;) if sep isn’t found.</td>
</tr>
<tr>
<td><code>s.replace(old, new [,maxreplace])</code></td>
<td>Replaces a substring.</td>
</tr>
<tr>
<td><code>s.rfind(sub [,start [,end]])</code></td>
<td>Finds the last occurrence of a substring.</td>
</tr>
<tr>
<td><code>s.rindex(sub [,start [,end]])</code></td>
<td>Finds the last occurrence or raises an error.</td>
</tr>
<tr>
<td><code>s.rjust(width [, fill])</code></td>
<td>Right-aligns s in a string of length width.</td>
</tr>
<tr>
<td><code>s.rpartition(sep)</code></td>
<td>Partitions s based on a separator sep, but searches from the end of the string.</td>
</tr>
<tr>
<td><code>s.rsplit([sep [,maxsplit]])</code></td>
<td>Splits a string from the end of the string using sep as a delimiter. maxsplit is the maximum number of splits to perform. If maxsplit is omitted, the result is identical to the split() method.</td>
</tr>
<tr>
<td><code>s.rstrip([chrs])</code></td>
<td>Removes trailing whitespace or characters supplied in chrs.</td>
</tr>
<tr>
<td><code>s.split([sep [,maxsplit]])</code></td>
<td>Splits a string using sep as a delimiter. maxsplit is the maximum number of splits to perform.</td>
</tr>
</tbody>
</table>
Table 3.5  Continued

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.splitlines([keepends])</code></td>
<td>Splits a string into a list of lines. If <code>keepends</code> is 1, trailing newlines are preserved.</td>
</tr>
<tr>
<td><code>s.startswith(prefix [,start [,end]])</code></td>
<td>Checks whether a string starts with <code>prefix</code>.</td>
</tr>
<tr>
<td><code>s.strip([chrs])</code></td>
<td>Removes leading and trailing whitespace or characters supplied in <code>chrs</code>.</td>
</tr>
<tr>
<td><code>s.swapcase()</code></td>
<td>Converts uppercase to lowercase, and vice versa.</td>
</tr>
<tr>
<td><code>s.title()</code></td>
<td>Returns a title-cased version of the string.</td>
</tr>
<tr>
<td><code>s.translate(table [,deletechars])</code></td>
<td>Translates a string using a character translation table <code>table</code>, removing characters in <code>deletechars</code>.</td>
</tr>
<tr>
<td><code>s.upper()</code></td>
<td>Converts a string to uppercase.</td>
</tr>
<tr>
<td><code>s.zfill(width)</code></td>
<td>Pads a string with zeros on the left up to the specified <code>width</code>.</td>
</tr>
</tbody>
</table>

**xrange() Objects**

The built-in function `xrange([i,,j [,stride]])` creates an object that represents a range of integers $k$ such that $i \leq k < j$. The first index, $i$, and the `stride` are optional and have default values of 0 and 1, respectively. An `xrange` object calculates its values whenever it’s accessed and although an `xrange` object looks like a sequence, it is actually somewhat limited. For example, none of the standard slicing operations are supported. This limits the utility of `xrange` to only a few applications such as iterating in simple loops.

It should be noted that in Python 3, `xrange()` has been renamed to `range()`. However, it operates in exactly the same manner as described here.

**Mapping Types**

A mapping object represents an arbitrary collection of objects that are indexed by another collection of nearly arbitrary key values. Unlike a sequence, a mapping object is unordered and can be indexed by numbers, strings, and other objects. Mappings are mutable.

Dictionaries are the only built-in mapping type and are Python’s version of a hash table or associative array. You can use any immutable object as a dictionary key value (strings, numbers, tuples, and so on). Lists, dictionaries, and tuples containing mutable objects cannot be used as keys (the dictionary type requires key values to remain constant).

To select an item in a mapping object, use the key index operator `m[k]`, where `k` is a key value. If the key is not found, a `KeyError` exception is raised. The `len(m)` function returns the number of items contained in a mapping object. Table 3.6 lists the methods and operations.
Table 3.6 Methods and Operations for Dictionaries

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len(m)</code></td>
<td>Returns the number of items in <code>m</code>.</td>
</tr>
<tr>
<td><code>m[k]</code></td>
<td>Returns the item of <code>m</code> with key <code>k</code>.</td>
</tr>
<tr>
<td><code>m[k]=x</code></td>
<td>Sets <code>m[k]</code> to <code>x</code>.</td>
</tr>
<tr>
<td><code>del m[k]</code></td>
<td>Removes <code>m[k]</code> from <code>m</code>.</td>
</tr>
<tr>
<td><code>k in m</code></td>
<td>Returns <code>True</code> if <code>k</code> is a key in <code>m</code>.</td>
</tr>
<tr>
<td><code>m.clear()</code></td>
<td>Removes all items from <code>m</code>.</td>
</tr>
<tr>
<td><code>m.copy()</code></td>
<td>Makes a copy of <code>m</code>.</td>
</tr>
<tr>
<td><code>m.fromkeys(s [,value])</code></td>
<td>Create a new dictionary with keys from sequence <code>s</code> and values all set to <code>value</code>.</td>
</tr>
<tr>
<td><code>m.get(k [,v])</code></td>
<td>Returns <code>m[k]</code> if found; otherwise, returns <code>v</code>.</td>
</tr>
<tr>
<td><code>m.has_key(k)</code></td>
<td>Returns <code>True</code> if <code>m</code> has key <code>k</code>; otherwise, returns <code>False</code>. (Deprecated, use the <code>in</code> operator instead. Python 2 only)</td>
</tr>
<tr>
<td><code>m.items()</code></td>
<td>Returns a sequence of <code>(key,value)</code> pairs.</td>
</tr>
<tr>
<td><code>m.keys()</code></td>
<td>Returns a sequence of key values.</td>
</tr>
<tr>
<td><code>m.pop(k [,default])</code></td>
<td>Returns <code>m[k]</code> if found and removes it from <code>m</code>; otherwise, returns <code>default</code> if supplied or raises <code>KeyError</code> if not.</td>
</tr>
<tr>
<td><code>m.popitem()</code></td>
<td>Removes a random <code>(key,value)</code> pair from <code>m</code> and returns it as a tuple.</td>
</tr>
<tr>
<td><code>m.setdefault(k [,v])</code></td>
<td>Returns <code>m[k]</code> if found; otherwise, returns <code>v</code> and sets <code>m[k] = v</code>.</td>
</tr>
<tr>
<td><code>m.update(b)</code></td>
<td>Adds all objects from <code>b</code> to <code>m</code>.</td>
</tr>
<tr>
<td><code>m.values()</code></td>
<td>Returns a sequence of all values in <code>m</code>.</td>
</tr>
</tbody>
</table>

Most of the methods in Table 3.6 are used to manipulate or retrieve the contents of a dictionary. The `m.clear()` method removes all items. The `m.update(b)` method updates the current mapping object by inserting all the `(key,value)` pairs found in the mapping object `b`. The `m.get(k [,v])` method retrieves an object but allows for an optional default value, `v`, that’s returned if no such key exists. The `m.setdefault(k [,v])` method is similar to `m.get()`, except that in addition to returning `v` if no object exists, it sets `m[k] = v`. If `v` is omitted, it defaults to `None`. The `m.pop()` method returns an item from a dictionary and removes it at the same time. The `m.popitem()` method is used to iteratively destroy the contents of a dictionary.

The `m.copy()` method makes a shallow copy of the items contained in a mapping object and places them in a new mapping object. The `m.fromkeys(s [,value])` method creates a new mapping with keys all taken from a sequence `s`. The type of the resulting mapping will be the same as `m`. The value associated with all of these keys is set to `None` unless an alternative value is given with the optional `value` parameter. The `fromkeys()` method is defined as a class method, so an alternative way to invoke it would be to use the class name such as `dict.fromkeys()`. The `m.items()` method returns a sequence containing `(key,value)` pairs. The `m.keys()` method returns a sequence with all the key values, and the `m.values()` method returns a sequence with all the values. For these methods, you should assume that the only safe operation that can be performed on the result is iteration. In Python 2 the result is a list, but in Python 3 the result is an iterator that iterates over the current contents of the mapping. If you write code that simply assumes it is an iterator, it will...
be generally compatible with both versions of Python. If you need to store the result of these methods as data, make a copy by storing it in a list. For example, `items = list(m.items())`. If you simply want a list of all keys, use `keys = list(m.keys())`.

## Set Types

A set is an unordered collection of unique items. Unlike sequences, sets provide no indexing or slicing operations. They are also unlike dictionaries in that there are no key values associated with the objects. The items placed into a set must be immutable. Two different set types are available: `set` is a mutable set, and `frozenset` is an immutable set. Both kinds of sets are created using a pair of built-in functions:

```python
s = set([1, 5, 10, 15])
f = frozenset(['a', 37, 'hello'])
```

Both `set()` and `frozenset()` populate the set by iterating over the supplied argument. Both kinds of sets provide the methods outlined in Table 3.7.

### Table 3.7 Methods and Operations for Set Types

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len(s)</code></td>
<td>Returns the number of items in <code>s</code>.</td>
</tr>
<tr>
<td><code>s.copy()</code></td>
<td>Makes a copy of <code>s</code>.</td>
</tr>
<tr>
<td><code>s.difference(t)</code></td>
<td>Set difference. Returns all the items in <code>s</code>, but not in <code>t</code>.</td>
</tr>
<tr>
<td><code>s.intersection(t)</code></td>
<td>Intersection. Returns all the items that are both in <code>s</code> and in <code>t</code>.</td>
</tr>
<tr>
<td><code>s.isdisjoint(t)</code></td>
<td>Returns <code>True</code> if <code>s</code> and <code>t</code> have no items in common.</td>
</tr>
<tr>
<td><code>s.issubset(t)</code></td>
<td>Returns <code>True</code> if <code>s</code> is a subset of <code>t</code>.</td>
</tr>
<tr>
<td><code>s.issuperset(t)</code></td>
<td>Returns <code>True</code> if <code>s</code> is a superset of <code>t</code>.</td>
</tr>
<tr>
<td><code>s.symmetric_difference(t)</code></td>
<td>Symmetric difference. Returns all the items that are in <code>s</code> or <code>t</code>, but not in both sets.</td>
</tr>
<tr>
<td><code>s.union(t)</code></td>
<td>Union. Returns all items in <code>s</code> or <code>t</code>.</td>
</tr>
</tbody>
</table>

The `s.difference(t)`, `s.intersection(t)`, `s.symmetric_difference(t)`, and `s.union(t)` methods provide the standard mathematical operations on sets. The returned value has the same type as `s` (set or `frozenset`). The parameter `t` can be any Python object that supports iteration. This includes sets, lists, tuples, and strings. These set operations are also available as mathematical operators, as described further in Chapter 4.

Mutable sets (`set`) additionally provide the methods outlined in Table 3.8.

### Table 3.8 Methods for Mutable Set Types

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.add(item)</code></td>
<td>Adds <code>item</code> to <code>s</code>. Has no effect if <code>item</code> is already in <code>s</code>.</td>
</tr>
<tr>
<td><code>s.clear()</code></td>
<td>Removes all items from <code>s</code>.</td>
</tr>
<tr>
<td><code>s.difference_update(t)</code></td>
<td>Removes all the items from <code>s</code> that are also in <code>t</code>.</td>
</tr>
</tbody>
</table>
Table 3.8  Continued

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.discard(item)</code></td>
<td>Removes <code>item</code> from <code>s</code>. If <code>item</code> is not a member of <code>s</code>, nothing happens.</td>
</tr>
<tr>
<td><code>s.intersection_update(t)</code></td>
<td>Computes the intersection of <code>s</code> and <code>t</code> and leaves the result in <code>s</code>.</td>
</tr>
<tr>
<td><code>s.pop()</code></td>
<td>Returns an arbitrary set element and removes it from <code>s</code>.</td>
</tr>
<tr>
<td><code>s.remove(item)</code></td>
<td>Removes <code>item</code> from <code>s</code>. If <code>item</code> is not a member, <code>KeyError</code> is raised.</td>
</tr>
<tr>
<td><code>s.symmetric_difference_update(t)</code></td>
<td>Computes the symmetric difference of <code>s</code> and <code>t</code> and leaves the result in <code>s</code>.</td>
</tr>
<tr>
<td><code>s.update(t)</code></td>
<td>Adds all the items in <code>t</code> to <code>s</code>. <code>t</code> may be another set, a sequence, or any object that supports iteration.</td>
</tr>
</tbody>
</table>

All these operations modify the set `s` in place. The parameter `t` can be any object that supports iteration.

**Built-in Types for Representing Program Structure**

In Python, functions, classes, and modules are all objects that can be manipulated as data. Table 3.9 shows types that are used to represent various elements of a program itself.

Table 3.9  Built-in Python Types for Program Structure

<table>
<thead>
<tr>
<th>Type Category</th>
<th>Type Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callable</td>
<td>types.BuiltinFunctionType</td>
<td>Built-in function or method</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>Type of built-in types and classes</td>
</tr>
<tr>
<td></td>
<td>object</td>
<td>Ancestor of all types and classes</td>
</tr>
<tr>
<td></td>
<td>types.FunctionType</td>
<td>User-defined function</td>
</tr>
<tr>
<td></td>
<td>types.MethodType</td>
<td>Class method</td>
</tr>
<tr>
<td>Modules</td>
<td>types.ModuleType</td>
<td>Module</td>
</tr>
<tr>
<td>Classes</td>
<td>object</td>
<td>Ancestor of all types and classes</td>
</tr>
<tr>
<td>Types</td>
<td>type</td>
<td>Type of built-in types and classes</td>
</tr>
</tbody>
</table>

Note that `object` and `type` appear twice in Table 3.9 because classes and types are both callable as a function.

**Callable Types**

Callable types represent objects that support the function call operation. There are several flavors of objects with this property, including user-defined functions, built-in functions, instance methods, and classes.
User-Defined Functions

User-defined functions are callable objects created at the module level by using the `def` statement or with the `lambda` operator. Here’s an example:

```python
def foo(x, y):
    return x + y

bar = lambda x, y: x + y
```

A user-defined function `f` has the following attributes:

<table>
<thead>
<tr>
<th>Attribute(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f.__doc__</code></td>
<td>Documentation string</td>
</tr>
<tr>
<td><code>f.__name__</code></td>
<td>Function name</td>
</tr>
<tr>
<td><code>f.__dict__</code></td>
<td>Dictionary containing function attributes</td>
</tr>
<tr>
<td><code>f.__code__</code></td>
<td>Byte-compiled code</td>
</tr>
<tr>
<td><code>f.__defaults__</code></td>
<td>Tuple containing the default arguments</td>
</tr>
<tr>
<td><code>f.__globals__</code></td>
<td>Dictionary defining the global namespace</td>
</tr>
<tr>
<td><code>f.__closure__</code></td>
<td>Tuple containing data related to nested scopes</td>
</tr>
</tbody>
</table>

In older versions of Python 2, many of the preceding attributes had names such as `func_code`, `func_defaults`, and so on. The attribute names listed are compatible with Python 2.6 and Python 3.

Methods

Methods are functions that are defined inside a class definition. There are three common types of methods—instance methods, class methods, and static methods:

```python
class Foo(object):
    def instance_method(self, arg):
        statements
    @classmethod
    def class_method(cls, arg):
        statements
    @staticmethod
    def static_method(arg):
        statements
```

An instance method is a method that operates on an instance belonging to a given class. The instance is passed to the method as the first argument, which is called `self` by convention. A class method operates on the class itself as an object. The class object is passed to a class method in the first argument, `cls`. A static method is just a function that happens to be packaged inside a class. It does not receive an instance or a class object as a first argument.

Both instance and class methods are represented by a special object of type `types.MethodType`. However, understanding this special type requires a careful understanding of how object attribute lookup (`.`) works. The process of looking something up on an object (`.`) is always a separate operation from that of making a function call. When you invoke a method, both operations occur, but as distinct steps. This example illustrates the process of invoking `f.instance_method(arg)` on an instance of `Foo` in the preceding listing:

```python
f = Foo()                 # Create an instance
meth = f.instance_method # Lookup the method and notice the lack of ()
meth(37)                  # Now call the method
```
In this example, `meth` is known as a *bound method*. A bound method is a callable object that wraps both a function (the method) and an associated instance. When you call a bound method, the instance is passed to the method as the first parameter (`self`). Thus, `meth` in the example can be viewed as a method call that is primed and ready to go but which has not been invoked using the function call operator (`()`).

Method lookup can also occur on the class itself. For example:

```python
umeth = Foo.instance_method  # Lookup instance_method on Foo
umeth(f, 37)                 # Call it, but explicitly supply self
```

In this example, `umeth` is known as an *unbound method*. An unbound method is a callable object that wraps the method function, but which expects an instance of the proper type to be passed as the first argument. In the example, we have passed `f`, an instance of `Foo`, as the first argument. If you pass the wrong kind of object, you get a `TypeError`. For example:

```python
>>> umeth("hello", 5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: descriptor 'instance_method' requires a 'Foo' object but received a 'str'
```

For user-defined classes, bound and unbound methods are both represented as an object of type `types.MethodType`, which is nothing more than a thin wrapper around an ordinary function object. The following attributes are defined for method objects:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m.__doc__</code></td>
<td>Documentation string</td>
</tr>
<tr>
<td><code>m.__name__</code></td>
<td>Method name</td>
</tr>
<tr>
<td><code>m.__class__</code></td>
<td>Class in which this method was defined</td>
</tr>
<tr>
<td><code>m.__func__</code></td>
<td>Function object implementing the method</td>
</tr>
<tr>
<td><code>m.__self__</code></td>
<td>Instance associated with the method (None if unbound)</td>
</tr>
</tbody>
</table>

One subtle feature of Python 3 is that unbound methods are no longer wrapped by a `types.MethodType` object. If you access `Foo.instance_method` as shown in earlier examples, you simply obtain the raw function object that implements the method. Moreover, you’ll find that there is no longer any type checking on the `self` parameter.

### Built-in Functions and Methods

The object `types.BuiltinFunctionType` is used to represent functions and methods implemented in C and C++. The following attributes are available for built-in methods:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>b.__doc__</code></td>
<td>Documentation string</td>
</tr>
<tr>
<td><code>b.__name__</code></td>
<td>Function/method name</td>
</tr>
<tr>
<td><code>b.__self__</code></td>
<td>Instance associated with the method (if bound)</td>
</tr>
</tbody>
</table>

For built-in functions such as `len()`, `__self__` is set to `None`, indicating that the function isn’t bound to any specific object. For built-in methods such as `x.append`, where `x` is a list object, `__self__` is set to `x`. 
Classes and Instances as Callables

Class objects and instances also operate as callable objects. A class object is created by the `class` statement and is called as a function in order to create new instances. In this case, the arguments to the function are passed to the `__init__()` method of the class in order to initialize the newly created instance. An instance can emulate a function if it defines a special method, `__call__()`.

If this method is defined for an instance, `x`, then `x(args)` invokes the method `x.__call__(args)`.

Classes, Types, and Instances

When you define a class, the class definition normally produces an object of type `type`. Here’s an example:

```python
>>> class Foo(object):
...     pass
... >>> type(Foo)
<type 'type'>
```

The following table shows commonly used attributes of a type object `t`:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t.__doc__</code></td>
<td>Documentation string</td>
</tr>
<tr>
<td><code>t.__name__</code></td>
<td>Class name</td>
</tr>
<tr>
<td><code>t.__bases__</code></td>
<td>Tuple of base classes</td>
</tr>
<tr>
<td><code>t.__dict__</code></td>
<td>Dictionary holding class methods and variables</td>
</tr>
<tr>
<td><code>t.__module__</code></td>
<td>Module name in which the class is defined</td>
</tr>
<tr>
<td><code>t.__abstractmethods__</code></td>
<td>Set of abstract method names (may be undefined if there aren’t any)</td>
</tr>
</tbody>
</table>

When an object instance is created, the type of the instance is the class that defined it. Here’s an example:

```python
>>> f = Foo()
>>> type(f)
<class '__main__.Foo'>
```

The following table shows special attributes of an instance `i`:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>i.__class__</code></td>
<td>Class to which the instance belongs</td>
</tr>
<tr>
<td><code>i.__dict__</code></td>
<td>Dictionary holding instance data</td>
</tr>
</tbody>
</table>

The `__dict__` attribute is normally where all of the data associated with an instance is stored. When you make assignments such as `i.attr = value`, the value is stored here. However, if a user-defined class uses `__slots__`, a more efficient internal representation is used and instances will not have a `__dict__` attribute. More details on objects and the organization of the Python object system can be found in Chapter 7.

Modules

The `module` type is a container that holds objects loaded with the `import` statement. When the statement `import foo` appears in a program, for example, the name `foo` is
assigned to the corresponding module object. Modules define a namespace that’s imple-
mented using a dictionary accessible in the attribute __dict__. Whenever an attribute
of a module is referenced (using the dot operator), it’s translated into a dictionary
lookup. For example, \texttt{m.x} is equivalent to \texttt{m.__dict__["x"]}. Likewise, assignment to
an attribute such as \texttt{m.x = y} is equivalent to \texttt{m.__dict__["x"] = y}. The following
attributes are available:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m.<strong>dict</strong></td>
<td>Dictionary associated with the module</td>
</tr>
<tr>
<td>m.<strong>doc</strong></td>
<td>Module documentation string</td>
</tr>
<tr>
<td>m.<strong>name</strong></td>
<td>Name of the module</td>
</tr>
<tr>
<td>m.<strong>file</strong></td>
<td>File from which the module was loaded</td>
</tr>
<tr>
<td>m.<strong>path</strong></td>
<td>Fully qualified package name, only defined when the module object refers to a package</td>
</tr>
</tbody>
</table>

### Built-in Types for Interpreter Internals

A number of objects used by the internals of the interpreter are exposed to the user. These include traceback objects, code objects, frame objects, generator objects, slice objects, and the \texttt{Ellipsis} as shown in Table 3.10. It is relatively rare for programs to manipulate these objects directly, but they may be of practical use to tool-builders and framework designers.

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>types.CodeType</td>
<td>Byte-compiled code</td>
</tr>
<tr>
<td>types.FrameType</td>
<td>Execution frame</td>
</tr>
<tr>
<td>types_GENERATOR</td>
<td>Generator object</td>
</tr>
<tr>
<td>types.TracebackType</td>
<td>Stack traceback of an exception</td>
</tr>
<tr>
<td>slice</td>
<td>Generated by extended slices</td>
</tr>
<tr>
<td>Ellipsis</td>
<td>Used in extended slices</td>
</tr>
</tbody>
</table>

### Code Objects

Code objects represent raw byte-compiled executable code, or \texttt{bytecode}, and are typically returned by the built-in \texttt{compile()} function. Code objects are similar to functions except that they don’t contain any context related to the namespace in which the code was defined, nor do code objects store information about default argument values. A code object, \texttt{c}, has the following read-only attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c.co_name</td>
<td>Function name.</td>
</tr>
<tr>
<td>c.co_argcount</td>
<td>Number of positional arguments (including default values).</td>
</tr>
<tr>
<td>c.co_nlocals</td>
<td>Number of local variables used by the function.</td>
</tr>
<tr>
<td>c.co_varnames</td>
<td>Tuple containing names of local variables.</td>
</tr>
</tbody>
</table>
Chapter 3  Types and Objects

Attribute Description

c.co_cellvars Tuple containing names of variables referenced by nested functions.
c.co_freevars Tuple containing names of free variables used by nested functions.
c.co_code String representing raw bytecode.
c.co_consts Tuple containing the literals used by the bytecode.
c.co_names Tuple containing names used by the bytecode.
c.co_filename Name of the file in which the code was compiled.
c.co_firstlineno First line number of the function.
c.co_lnotab String encoding bytecode offsets to line numbers.
c.co_stacksize Required stack size (including local variables).
c.co_flags Integer containing interpreter flags. Bit 2 is set if the function uses a variable number of positional arguments using "*args". Bit 3 is set if the function allows arbitrary keyword arguments using "**kwargs". All other bits are reserved.

Frame Objects

Frame objects are used to represent execution frames and most frequently occur in traceback objects (described next). A frame object, \( f \), has the following read-only attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f.f_back</td>
<td>Previous stack frame (toward the caller).</td>
</tr>
<tr>
<td>f.f_code</td>
<td>Code object being executed.</td>
</tr>
<tr>
<td>f.f_globals</td>
<td>Dictionary used for global variables.</td>
</tr>
<tr>
<td>f.f_lineno</td>
<td>Line number.</td>
</tr>
<tr>
<td>f.f_locals</td>
<td>Dictionary used for local variables.</td>
</tr>
<tr>
<td>f.f_builtins</td>
<td>Dictionary used for built-in names.</td>
</tr>
<tr>
<td>f.f_lineno</td>
<td>Line number.</td>
</tr>
<tr>
<td>f.f_lasti</td>
<td>Current instruction. This is an index into the bytecode string of f_code.</td>
</tr>
</tbody>
</table>

The following attributes can be modified (and are used by debuggers and other tools):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f.f_trace</td>
<td>Function called at the start of each source code line</td>
</tr>
<tr>
<td>f.f_exc_type</td>
<td>Most recent exception type (Python 2 only)</td>
</tr>
<tr>
<td>f.f_exc_value</td>
<td>Most recent exception value (Python 2 only)</td>
</tr>
<tr>
<td>f.f_exc_traceback</td>
<td>Most recent exception traceback (Python 2 only)</td>
</tr>
</tbody>
</table>

Traceback Objects

Traceback objects are created when an exception occurs and contain stack trace information. When an exception handler is entered, the stack trace can be retrieved using the
The `sys.exc_info()` function. The following read-only attributes are available in traceback objects:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t.tb_next</code></td>
<td>Next level in the stack trace (toward the execution frame where the exception occurred)</td>
</tr>
<tr>
<td><code>t.tb_frame</code></td>
<td>Execution frame object of the current level</td>
</tr>
<tr>
<td><code>t.tb_lineno</code></td>
<td>Line number where the exception occurred</td>
</tr>
<tr>
<td><code>t.tb_lasti</code></td>
<td>Instruction being executed in the current level</td>
</tr>
</tbody>
</table>

## Generator Objects

Generator objects are created when a generator function is invoked (see Chapter 6, “Functions and Functional Programming”). A generator function is defined whenever a function makes use of the special `yield` keyword. The generator object serves as both an iterator and a container for information about the generator function itself. The following attributes and methods are available:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>g.gi_code</code></td>
<td>Code object for the generator function.</td>
</tr>
<tr>
<td><code>g.gi_frame</code></td>
<td>Execution frame of the generator function.</td>
</tr>
<tr>
<td><code>g.gi_running</code></td>
<td>Integer indicating whether or not the generator function is currently running.</td>
</tr>
<tr>
<td><code>g.next()</code></td>
<td>Execute the function until the next yield statement and return the value (this method is called <code>__next__</code> in Python 3).</td>
</tr>
<tr>
<td><code>g.send(value)</code></td>
<td>Sends a value to a generator. The passed value is returned by the <code>yield</code> expression in the generator that executes until the next <code>yield</code> expression is encountered. <code>send()</code> returns the value passed to <code>yield</code> in this expression.</td>
</tr>
<tr>
<td><code>g.close()</code></td>
<td>Closes a generator by raising a <code>GeneratorExit</code> exception in the generator function. This method executes automatically when a generator object is garbage-collected.</td>
</tr>
<tr>
<td><code>g.throw(exc [,exc_value [,exc_tb ]])</code></td>
<td>Raises an exception in a generator at the point of the current <code>yield</code> statement. <code>exc</code> is the exception type, <code>exc_value</code> is the exception value, and <code>exc_tb</code> is an optional traceback. If the resulting exception is caught and handled, returns the value passed to the next <code>yield</code> statement.</td>
</tr>
</tbody>
</table>

## Slice Objects

Slice objects are used to represent slices given in extended slice syntax, such as `a[i:j:stride], a[i:j, n:m],` or `a[... , i:j]`. Slice objects are also created using the built-in `slice([i,] j [,stride])` function. The following read-only attributes are available:
### Chapters and Objects

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.start</td>
<td>Lower bound of the slice; None if omitted</td>
</tr>
<tr>
<td>s.stop</td>
<td>Upper bound of the slice; None if omitted</td>
</tr>
<tr>
<td>s.step</td>
<td>Stride of the slice; None if omitted</td>
</tr>
</tbody>
</table>

Slice objects also provide a single method, `s.indices(length)`. This function takes a length and returns a tuple `(start, stop, stride)` that indicates how the slice would be applied to a sequence of that length. Here’s an example:

```python
s = slice(10, 20)    # Slice object represents [10:20]
s.indices(100)        # Returns (10, 20, 1) -> [10:20]
s.indices(15)         # Returns (10, 15, 1) -> [10:15]
```

### Ellipsis Object

The `Ellipsis` object is used to indicate the presence of an ellipsis (…) in an index lookup `[]`. There is a single object of this type, accessed through the built-in name `Ellipsis`. It has no attributes and evaluates as `True`. None of Python’s built-in types make use of `Ellipsis`, but it may be useful if you are trying to build advanced functionality into the indexing operator `[]` on your own objects. The following code shows how an `Ellipsis` gets created and passed into the indexing operator:

```python
class Example(object):
    def __getitem__(self, index):
        print(index)

e = Example()
e[3, ..., 4]          # Calls e.__getitem__((3, Ellipsis, 4))
```

### Object Behavior and Special Methods

Objects in Python are generally classified according to their behaviors and the features that they implement. For example, all of the sequence types such as strings, lists, and tuples are grouped together merely because they all happen to support a common set of sequence operations such as `s[n]`, `len(s)`, etc. All basic interpreter operations are implemented through special object methods. The names of special methods are always preceded and followed by double underscores (`__`). These methods are automatically triggered by the interpreter as a program executes. For example, the operation `x + y` is mapped to an internal method, `x.__add__(y)`, and an indexing operation, `x[k]`, is mapped to `x.__getitem__(k)`. The behavior of each data type depends entirely on the set of special methods that it implements.

User-defined classes can define new objects that behave like the built-in types simply by supplying an appropriate subset of the special methods described in this section. In addition, built-in types such as lists and dictionaries can be specialized (via inheritance) by redefining some of the special methods.

The next few sections describe the special methods associated with different categories of interpreter features.

### Object Creation and Destruction

The methods in Table 3.11 create, initialize, and destroy instances. `__new__()` is a class method that is called to create an instance. The `__init__()` method initializes the
attributes of an object and is called immediately after an object has been newly created. The \_\_del\_\_() method is invoked when an object is about to be destroyed. This method is invoked only when an object is no longer in use. It's important to note that the statement `del x` only decrements an object’s reference count and doesn’t necessarily result in a call to this function. Further details about these methods can be found in Chapter 7.

Table 3.11  **Special Methods for Object Creation and Destruction**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__new__()</td>
<td>A class method called to create a new instance</td>
</tr>
<tr>
<td>__init__()</td>
<td>Called to initialize a new instance</td>
</tr>
<tr>
<td>__del__()</td>
<td>Called when an instance is being destroyed</td>
</tr>
</tbody>
</table>

The \_\_new\_\_() and \_\_init\_\_() methods are used together to create and initialize new instances. When an object is created by calling `A(args)`, it is translated into the following steps:

\[
x = A._\_new\_\_\(A, args) \\
is isinstance(x,A): x._\_init\_\_\(args\)
\]

In user-defined objects, it is rare to define \_\_new\_\_() or \_\_del\_\_(). \_\_new\_\_() is usually only defined in metaclasses or in user-defined objects that happen to inherit from one of the immutable types (integers, strings, tuples, and so on). \_\_del\_\_() is only defined in situations in which there is some kind of critical resource management issue, such as releasing a lock or shutting down a connection.

**Object String Representation**

The methods in Table 3.12 are used to create various string representations of an object.

Table 3.12  **Special Methods for Object Representation**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__format__()</td>
<td>Creates a formatted representation</td>
</tr>
<tr>
<td>__repr__()</td>
<td>Creates a string representation of an object</td>
</tr>
<tr>
<td>__str__()</td>
<td>Creates a simple string representation</td>
</tr>
</tbody>
</table>

The \_\_repr\_\_() and \_\_str\_\_() methods create simple string representations of an object. \_\_repr\_\_() method normally returns an expression string that can be evaluated to re-create the object. This is also the method responsible for creating the output of values you see when inspecting variables in the interactive interpreter. This method is invoked by the built-in `repr()` function. Here’s an example of using `repr()` and `eval()` together:

```python
a = [2, 3, 4, 5]  # Create a list
s = repr(a)      # s = '[2, 3, 4, 5]'
b = eval(s)      # Turns s back into a list
```
If a string expression cannot be created, the convention is for `__repr__()` to return a string of the form `<...message...>`, as shown here:

```python
f = open("foo")
a = repr(f)        # a = "<open file 'foo', mode 'r' at dc030>"
```

The `__str__()` method is called by the built-in `str()` function and by functions related to printing. It differs from `__repr__()` in that the string it returns can be more concise and informative to the user. If this method is undefined, the `__repr__()` method is invoked.

The `__format__()` method is called by the `format()` function or the `format()` method of strings. The `format_spec` argument is a string containing the format specification. This string is the same as the `format_spec` argument to `format()`. For example:

```python
format(x, "spec")            # Calls x.__format__("spec")
"x is {0:spec}".format(x)   # Calls x.__format__("spec")
```

The syntax of the format specification is arbitrary and can be customized on an object-by-object basis. However, a standard syntax is described in Chapter 4.

**Object Comparison and Ordering**

Table 3.13 shows methods that can be used to perform simple tests on an object. The `__bool__()` method is used for truth-value testing and should return `True` or `False`. If undefined, the `__len__()` method is a fallback that is invoked to determine truth. The `__hash__()` method is defined on objects that want to work as keys in a dictionary. The value returned is an integer that should be identical for two objects that compare as equal. Furthermore, mutable objects should not define this method; any changes to an object will alter the hash value and make it impossible to locate an object on subsequent dictionary lookups.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__bool__()</code></td>
<td>Returns <code>False</code> or <code>True</code> for truth-value testing</td>
</tr>
<tr>
<td><code>__hash__()</code></td>
<td>Computes an integer hash index</td>
</tr>
</tbody>
</table>

Objects can implement one or more of the relational operators (<, >, <=, >=, ==, !=). Each of these methods takes two arguments and is allowed to return any kind of object, including a Boolean value, a list, or any other Python type. For instance, a numerical package might use this to perform an element-wise comparison of two matrices, returning a matrix with the results. If a comparison can’t be made, these functions may also raise an exception. Table 3.14 shows the special methods for comparison operators.

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__lt__()</code></td>
<td><code>self &lt; other</code></td>
</tr>
<tr>
<td><code>__le__()</code></td>
<td><code>self &lt;= other</code></td>
</tr>
<tr>
<td><code>__gt__()</code></td>
<td><code>self &gt; other</code></td>
</tr>
<tr>
<td><code>__ge__()</code></td>
<td><code>self &gt;= other</code></td>
</tr>
</tbody>
</table>
Table 3.14  **Continued**

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__eq__</code>(self,other)</td>
<td><code>self == other</code></td>
</tr>
<tr>
<td><code>__ne__</code>(self,other)</td>
<td><code>self != other</code></td>
</tr>
</tbody>
</table>

It is not necessary for an object to implement all of the operations in Table 3.14. However, if you want to be able to compare objects using `==` or use an object as a dictionary key, the `__eq__()` method should be defined. If you want to be able to sort objects or use functions such as `min()` or `max()`, then `__lt__()` must be minimally defined.

**Type Checking**

The methods in Table 3.15 can be used to redefine the behavior of the type checking functions `isinstance()` and `issubclass()`. The most common application of these methods is in defining abstract base classes and interfaces, as described in Chapter 7.

Table 3.15  **Methods for Type Checking**

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__instancecheck__</code>(cls,object)</td>
<td><code>isinstance(object, cls)</code></td>
</tr>
<tr>
<td><code>__subclasscheck__</code>(cls, sub)</td>
<td><code>issubclass(sub, cls)</code></td>
</tr>
</tbody>
</table>

**Attribute Access**

The methods in Table 3.16 read, write, and delete the attributes of an object using the dot (.) operator and the `del` operator, respectively.

Table 3.16  **Special Methods for Attribute Access**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__getattr__</code>(self,name)</td>
<td>Returns the attribute <code>self.name</code>.</td>
</tr>
<tr>
<td><code>__setattr__</code>(self,name,value)</td>
<td>Sets the attribute <code>self.name</code> = <code>value</code>. Overrides the default mechanism.</td>
</tr>
<tr>
<td><code>__delattr__</code>(self,name)</td>
<td>Deletes the attribute <code>self.name</code>.</td>
</tr>
</tbody>
</table>

Whenever an attribute is accessed, the `__getattr__`() method is always invoked. If the attribute is located, it is returned. Otherwise, the `__getattr__`() method is invoked. The default behavior of `__getattr__`() is to raise an `AttributeError` exception. The `__setattr__`() method is always invoked when setting an attribute, and the `__delattr__`() method is always invoked when deleting an attribute.
Attribute Wrapping and Descriptors

A subtle aspect of attribute manipulation is that sometimes the attributes of an object are wrapped with an extra layer of logic that interact with the get, set, and delete operations described in the previous section. This kind of wrapping is accomplished by creating a descriptor object that implements one or more of the methods in Table 3.17. Keep in mind that descriptions are optional and rarely need to be defined.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>get</strong>(self, instance, cls)</td>
<td>Returns an attribute value or raises AttributeError</td>
</tr>
<tr>
<td><strong>set</strong>(self, instance, value)</td>
<td>Sets the attribute to value</td>
</tr>
<tr>
<td><strong>delete</strong>(self, instance)</td>
<td>Deletes the attribute</td>
</tr>
</tbody>
</table>

The __get__(), __set__(), and __delete__() methods of a descriptor are meant to interact with the default implementation of __getattr__(), __setattr__(), and __delattr__() methods on classes and types. This interaction occurs if you place an instance of a descriptor object in the body of a user-defined class. In this case, all access to the descriptor attribute will implicitly invoke the appropriate method on the descriptor object itself. Typically, descriptors are used to implement the low-level functionality of the object system including bound and unbound methods, class methods, static methods, and properties. Further examples appear in Chapter 7.

Sequence and Mapping Methods

The methods in Table 3.18 are used by objects that want to emulate sequence and mapping objects.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>len</strong>(self)</td>
<td>Returns the length of self</td>
</tr>
<tr>
<td><strong>getitem</strong>(self, key)</td>
<td>Returns self[key]</td>
</tr>
<tr>
<td><strong>setitem</strong>(self, key, value)</td>
<td>Sets self[key] = value</td>
</tr>
<tr>
<td><strong>delitem</strong>(self, key)</td>
<td>Deletes self[key]</td>
</tr>
<tr>
<td><strong>contains</strong>(self, obj)</td>
<td>Returns True if obj is in self; otherwise, returns False</td>
</tr>
</tbody>
</table>

Here’s an example:

```
a = [1,2,3,4,5,6]
len(a) # a.__len__()
x = a[2] # x = a.__getitem__(2)
a[1] = 7 # a.__setitem__(1,7)
del a[2] # a.__delitem__(2)
5 in a # a.__contains__(5)
```

The __len__ method is called by the built-in len() function to return a nonnegative length. This function also determines truth values unless the __bool__() method has also been defined.
For manipulating individual items, the \_getitem\_() method can return an item by key value. The key can be any Python object but is typically an integer for sequences. The \_setitem\_() method assigns a value to an element. The \_delitem\_() method is invoked whenever the del operation is applied to a single element. The \_contains\_() method is used to implement the in operator.

The slicing operations such as \$x = s[i:j]\$ are also implemented using \_getitem\_(), \_setitem\_(), and \_delitem\_(). However, for slices, a special slice object is passed as the key. This object has attributes that describe the range of the slice being requested. For example:

\[
a = [1,2,3,4,5,6]
\]
\[
x = a[1:5]  \quad \# \quad x = a._getitem_((slice(1,5,\text{None}))
\]
\[
a[1:3] = [10,11,12]  \quad \# \quad a._setitem_((slice(1,3,\text{None})), [10,11,12])
\]
\[
del a[1:4]  \quad \# \quad a._delitem_((slice(1,4,\text{None}))
\]

The slicing features of Python are actually more powerful than many programmers realize. For example, the following variations of extended slicing are all supported and might be useful for working with multidimensional data structures such as matrices and arrays:

\[
a = m[0:100:10]  \quad \# \quad \text{Strided slice (stride=10)}
b = m[1:10, 3:20]  \quad \# \quad \text{Multidimensional slice}
c = m[0:100:10, 50:75:5]  \quad \# \quad \text{Multiple dimensions with strides}
m[0:5, 5:10] = n  \quad \# \quad \text{extended slice assignment}
del m[10:20]  \quad \# \quad \text{extended slice deletion}
\]

The general format for each dimension of an extended slice is \$i:j[:\text{stride}]\$, where \text{stride} is optional. As with ordinary slices, you can omit the starting or ending values for each part of a slice. In addition, the ellipsis (written as ...) is available to denote any number of trailing or leading dimensions in an extended slice:

\[
a = m[...; 10:20]  \quad \# \quad \text{extended slice access with Ellipsis}
m[10:20; ...] = n
\]

When using extended slices, the \_getitem\_(), \_setitem\_(), and \_delitem\_() methods implement access, modification, and deletion, respectively. However, instead of an integer, the value passed to these methods is a tuple containing a combination of slice or Ellipsis objects. For example,

\[
a = m[0:10, 0:100:5, ...]
\]

invokes \_getitem\_() as follows:

\[
a = m._getitem_((slice(0,10,\text{None})), slice(0,100,5), \text{Ellipsis})
\]

Python strings, tuples, and lists currently provide some support for extended slices, which is described in Chapter 4. Special-purpose extensions to Python, especially those with a scientific flavor, may provide new types and objects with advanced support for extended slicing operations.

### Iteration

If an object, \text{obj}, supports iteration, it must provide a method, \text{obj.__iter__()}, that returns an iterator object. The iterator object \text{iter}, in turn, must implement a single method, \text{iter.next()} (or \text{iter._next_()} in Python 3), that returns the next object or raises \text{StopIteration} to signal the end of iteration. Both of these methods are used by the implementation of the for statement as well as other operations that
implicitly perform iteration. For example, the statement `for x in s` is carried out by performing steps equivalent to the following:

```python
_iter = s.__iter__()
while 1:
    try:
        x = _iter.next()  # _iter.__next__() in Python 3
    except StopIteration:
        break
# Do statements in body of for loop
...
```

**Mathematical Operations**

Table 3.19 lists special methods that objects must implement to emulate numbers. Mathematical operations are always evaluated from left to right according the precedence rules described in Chapter 4; when an expression such as `x + y` appears, the interpreter tries to invoke the method `x.__add__(y)`. The special methods beginning with `r` support operations with reversed operands. These are invoked only if the left operand doesn’t implement the specified operation. For example, if `x` in `x + y` doesn’t support the `__add__()` method, the interpreter tries to invoke the method `y.__radd__(x)`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add</strong>(self,other)</td>
<td><code>self + other</code></td>
</tr>
<tr>
<td><strong>sub</strong>(self,other)</td>
<td><code>self - other</code></td>
</tr>
<tr>
<td><strong>mul</strong>(self,other)</td>
<td><code>self * other</code></td>
</tr>
<tr>
<td><strong>div</strong>(self,other)</td>
<td><code>self / other</code> (Python 2 only)</td>
</tr>
<tr>
<td><strong>truediv</strong>(self,other)</td>
<td><code>self / other</code> (Python 3)</td>
</tr>
<tr>
<td><strong>floordiv</strong>(self,other)</td>
<td><code>self // other</code></td>
</tr>
<tr>
<td><strong>mod</strong>(self,other)</td>
<td><code>self % other</code></td>
</tr>
<tr>
<td><strong>divmod</strong>(self,other)</td>
<td><code>divmod(self,other)</code></td>
</tr>
<tr>
<td><strong>pow</strong>(self,other</td>
<td>[modulo])</td>
</tr>
<tr>
<td><strong>lshift</strong>(self,other)</td>
<td><code>self &lt;&lt; other</code></td>
</tr>
<tr>
<td><strong>rshift</strong>(self,other)</td>
<td><code>self &gt;&gt; other</code></td>
</tr>
<tr>
<td><strong>or</strong>(self,other)</td>
<td>`self</td>
</tr>
<tr>
<td><strong>xor</strong>(self,other)</td>
<td><code>self ^ other</code></td>
</tr>
<tr>
<td><strong>and</strong>(self,other)</td>
<td><code>self &amp; other</code></td>
</tr>
<tr>
<td><strong>radd</strong>(self,other)</td>
<td><code>other + self</code> (Python 2 only)</td>
</tr>
<tr>
<td><strong>rsub</strong>(self,other)</td>
<td><code>other - self</code></td>
</tr>
<tr>
<td><strong>rmul</strong>(self,other)</td>
<td><code>other * self</code></td>
</tr>
<tr>
<td><strong>rdiv</strong>(self,other)</td>
<td><code>other / self</code> (Python 2 only)</td>
</tr>
<tr>
<td><strong>rtruediv</strong>(self,other)</td>
<td><code>other / self</code> (Python 3)</td>
</tr>
<tr>
<td><strong>rfloordiv</strong>(self,other)</td>
<td><code>other // self</code></td>
</tr>
<tr>
<td><strong>rmod</strong>(self,other)</td>
<td><code>other % self</code></td>
</tr>
<tr>
<td><strong>rdivmod</strong>(self,other)</td>
<td><code>divmod(other,self)</code></td>
</tr>
</tbody>
</table>
The methods \_\_iadd\_/(), \_\_isub\_/(), and so forth are used to support in-place arithmetic operators such as a+=b and a-=b (also known as augmented assignment). A distinction is made between these operators and the standard arithmetic methods because the implementation of the in-place operators might be able to provide certain customizations such as performance optimizations. For instance, if the \texttt{self} parameter is not shared, the value of an object could be modified in place without having to allocate a newly created object for the result.

The three flavors of division operators—\_\_div\_/(), \_\_truediv\_/(), and \_\_floordiv\_/()—are used to implement true division (/) and truncating division (\//) operations. The reasons why there are three operations deal with a change in the semantics of integer division that started in Python 2.2 but became the default behavior in Python 3. In Python 2, the default behavior of Python is to map the / operator to \_\_div\_/(). For integers, this operation truncates the result to an integer. In Python 3, division is mapped to \_\_truediv\_/() and for integers, a float is returned. This latter
behavior can be enabled in Python 2 as an optional feature by including the statement
from __future__ import division in a program.

The conversion methods __int__(), __long__(), __float__(), and
__complex__() convert an object into one of the four built-in numerical types. These
methods are invoked by explicit type conversions such as int() and float().
However, these methods are not used to implicitly coerce types in mathematical opera-
tions. For example, the expression 3 + x produces a TypeError even if x is a user-
defined object that defines __int__() for integer conversion.

Callable Interface

An object can emulate a function by providing the __call__(self, *args, **kwargs) method. If an object, x, provides this method, it can be invoked like a
function. That is, x(arg1, arg2, ...) invokes x.__call__(self, arg1, arg2, ...). Objects that emulate functions can be useful for creating functors or proxies.

Here is a simple example:

class DistanceFrom(object):
    def __init__(self, origin):
        self.origin = origin
    def __call__(self, x):
        return abs(x - self.origin)

nums = [1, 37, 42, 101, 13, 9, -20]
nums.sort(key=DistanceFrom(10))  # Sort by distance from 10

In this example, the DistanceFrom class creates instances that emulate a single-
argument function. These can be used in place of a normal function—for instance, in
the call to sort() in the example.

Context Management Protocol

The with statement allows a sequence of statements to execute under the control of
another object known as a context manager. The general syntax is as follows:

with context [ as var]:
    statements

The context object shown here is expected to implement the methods shown in Table
3.20. The __enter__() method is invoked when the with statement executes. The
value returned by this method is placed into the variable specified with the optional as
specifier. The __exit__() method is called as soon as control-flow leaves from the
block of statements associated with the with statement. As arguments, __exit__() receives the current exception type, value, and traceback if an exception has been raised.
If no errors are being handled, all three values are set to None.

Table 3.20 Special Methods for Context Managers

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| __enter__(self) | Called when entering a new context. The return value is placed in the variable listed with the as specifier to the with state-
|                 | ment.       |
### Table 3.20  
**Continued**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__exit__(self, type, value, tb)</code></td>
<td>Called when leaving a context. If an exception occurred, <code>type</code>, <code>value</code>, and <code>tb</code> have the exception type, value, and traceback information. The primary use of the context management interface is to allow for simplified resource control on objects involving system state such as open files, network connections, and locks. By implementing this interface, an object can safely clean up resources when execution leaves a context in which an object is being used. Further details are found in Chapter 5, “Program Structure and Control Flow.”</td>
</tr>
</tbody>
</table>

### Object Inspection and `dir()`

The `dir()` function is commonly used to inspect objects. An object can supply the list of names returned by `dir()` by implementing `__dir__(self)`. Defining this makes it easier to hide the internal details of objects that you don’t want a user to directly access. However, keep in mind that a user can still inspect the underlying `__dict__` attribute of instances and classes to see everything that is defined.
Symbols & Numbers

! debugger command, pdb module, 187
!= not equal to operator, 66
' single quotes, 11, 27
" double quotes, 11, 27
''' triple quotes, 11, 27
""" triple quotes, 11, 27
# comment, 6, 26
#! in Unix shell scripts, 6, 176
  rewriting on package installation, 153
% modulo operator, 65
% string formatting operator, 8, 70, 162
%= operator, 75
& bitwise-and operator, 65
& set intersection operator, 15, 75
&= operator, 75
() function call operator, 76
() tuple, 14, 29
* keyword only arguments, Python 3, 625
* multiplication operator, 65
* passing sequences as function arguments, 94
* sequence replication operator, 67
* variable arguments in function definition, 94
* wildcard
  from module import, 24, 145
  iterable unpacking in Python 3, 623
** passing dictionaries as keyword arguments, 95
** power operator, 65
** variable keyword arguments in function definition, 95
**= operator, 75
*= operator, 75
+= operator, 75
= addition operator, 65
+ list concatenation operator, 12
+ sequence concatenation operator, 67
+ string concatenation operator, 11
+= unary plus operator, 65
+= operator, 75
- hyphen character, used as filename, 174
- set difference operator, 15, 75
- subtraction operator, 65
- unary minus operator, 65
-* coding: comment, in source code, 31
-= operator, 75
  attribute binding operator, 33, 48, 76, 118
  and modules, 51
    special methods for, 57
  directory reference in relative import statements, 150
... Ellipsis, 30, 54, 59
... interpreter prompt, 175
/ division operator, 65
// truncating division operator, 65
//= operator, 75
/= operator, 75
: colon in string formatting specifiers, 72
; semicolon, 7, 26
< left alignment in string format specifiers, 73
< less than operator, 66
<< left shift operator, 65
<<= operator, 75
<= less than or equal to operator, 66
== equal to operator, 66, 78
> greater than operator, 66
>= right alignment in string format specifiers, 73
\[ \geq \] greater than or equal to operator, 66
\[ \gg \] file redirection modifier to print, 10, 163
\[ \gg \] right shift operator, 65
\[ \gg \] operator, 75
\[ \gg \] interpreter prompt, 5, 175
@ decorator, 30, 101
\[ [] \] extended slicing operator, 39-40, 67-68
\[ [:] \] slicing operator, 39-40, 67-68
\[ [] \] indexing operator, 39-40, 67
    and special methods, 58
    on mappings, 45
    on sequences, 68
\[ [] \] list, 12, 29
\line continuation character, 9, 25, 29
\string escape codes, 27
\^ bitwise-xor operator, 65
\^ centered alignment in string format specifiers, 73
\^ set symmetric difference operator, 15, 75
\^ operator, 75
_ variable, interactive mode, 6, 176
\{ \} dict, 16, 29
\{ \} placeholder in format strings, 72
\{ \} set literal, Python 3, 622
| bitwise-or operator, 65
| set union operator, 15, 75
|= operator, 75
~ bitwise-negation operator, 65
~ expanding user home directory in filenames, 397
$variables in strings, 72
0b binary integer literal, 27
0o octal integer literal, 27
0x hexadecimal integer literal, 27
2's complement and integers, 66
2to3 tool, 635-637
    limitations of, 636
-3 command line option, 173, 635

A

‘a’ mode, to open() function, 159
a(args) debugger command, pdb module, 187
a2b_base64() function, binascii module, 547
a2b_hex() function, binascii module, 548
a2b_hqx() function, binascii module, 548
a2b_uu() function, binascii module, 547
abc module, 136, 257
ABCMeta metaclass, 136, 257
abort() function, os module, 390
abort() method, of FTP objects, 497
abs() function, 66, 201
    operator module, 273
__abs__() method, 61
absolute imports, 151
    Python 3, 634
    compared to relative imports, 151
absolute value, 66
absopath() function, os.path module, 396
abstract base class, 34, 136, 257
    calling methods in subclasses, 137
    checking performed, 137
    container objects, 265
    error if instantiated, 137
    example, 258
    files and I/O, 354
    numeric types, 253
    registering pre-existing classes, 137
    special methods for, 57
@abstractmethod decorator, 136-137, 257-258
__abstractmethods__ attribute, of types, 50
@abstractproperty decorator, 136-137, 257-258
accept() method
    of Listener objects, 433
    of dispatcher objects, 456
    of socket objects, 478
accept2dyear variable, time module, 405
access control specifiers, lack of, 127
access() function, os module, 386
acos() function, math module, 251
acosh() function, math module, 251
acquire() method
  of Condition objects, 441
  of Lock objects, 438
  of RLock objects, 438
  of Semaphore objects, 439
activate() method, of SocketServer class, 493
active_children() function, multiprocessing module, 434
active_count() function, threading module, 443
ActivePython, 5
add() function, operator module, 273
add() method
  of TarFile objects, 320
  of sets, 15, 46
__add__() method, 60
add_data() method, of Request objects, 516
add_header() method
  of Message objects, 555
  of Request objects, 516
add_option() method, of OptionParser objects, 158, 375
add_password() method, of AuthHandler objects, 519
add_section() method, of ConfigParser objects, 332
add_type() function, mimetypes module, 567
add_unredirected_header() method, of Request objects, 517
addfile() method, of TarFile objects, 320
addFilter() method
  of Handler objects, 364
  of Logger objects, 359
addHandler() method, of Logger objects, 361
addition operator +, 65
addLevelName() function, logging module, 366
address attribute
  of BaseManager objects, 431
  of Listener objects, 433
address families, of sockets, 470
address_family attribute, of SocketServer class, 492
addresses, network, 471
addressof() function, ctypes module, 617
adjacent string literals, concatenation of, 27
adler32() function, zlib module, 328
advanced string formatting, 8, 42, 72
AF_* constants, socket module, 470
aifc module, 588
aio_* family of system calls, lack of, 469
AJAX, example of, 531
alarm() function, signal module, 399
alarms, 399
alias debugger command, pdb module, 187
alignment() function, ctypes module, 617
__all__ variable
  and import statements, 145
  in packages, 150
all() function, 40, 67, 201
allow_reuse_address attribute, of SocketServer class, 492
altsep variable, os module, 386
altzone variable, time module, 405
and operator, boolean expressions, 9, 77
and() function, operator module, 274
__and__() method, 60
__annotations__ attribute, of functions, 624
anonymous functions, 112
any() function, 40, 67, 201
anydbm module, 310
api_version variable, sys module, 229
%APPDATA% environment variable, Windows, 177

How can we make this index more useful? Email us at indexes@samspublishing.com
append() method
  of Element objects, 576
  of array objects, 259
  of deque objects, 262
  of lists, 12, 40
appendChild() method, of DOM Node objects, 571
appendleft() method, of deque objects, 262
application logging, 355
applications, WSGI, 540
applicative order evaluation, 76
apply() method, of Pool objects, 424
apply_async() method, of Pool objects, 424
args attribute
  of Exception objects, 213
  of exceptions, 88
  of partial objects, 268
argtypes attribute, of ctypes function objects, 613
argv variable, sys module, 13, 157, 174, 229
ArithmeticError exception, 87, 212
array module, 259
Array() function, multiprocessing module, 426
Array() method, of Manager objects, 428
array() function, array module, 259
arrays, creating from uniform type, 259
arraysize attribute, of Cursor objects, 299
as qualifier
  of except statement, 22, 85
  of from-import statement, 145
  of import statement, 24, 144
  of with statement, 62, 90
as_integer_ratio() method, of floating point, 39
as_string() method, of Message objects, 555
ascii encoding, description of, 169
ascii() function, 201
  and Python 3, 201
  future_builtins module, 217
ASCII, and compatibility with UTF-8, 170
ascii_letters variable, string module, 287
ascii_lowercase variable, string module, 287
ascii_uppercase variable, string module, 287
asctime() function, time module, 405
asin() function, math module, 251
asinh() function, math module, 251
assert statement, 91
assert_() method, of TestCase objects, 185
assertAlmostEqual() method, of TestCase objects, 185
assertEqual() method, of TestCase objects, 185
assertRaises() method, of TestCase objects, 185
AssertionError exception, 87, 91, 213
assertions, 91
  stripping with -O option, 148
assertNotAlmostEqual() method, of TestCase objects, 185
assertNotEqual() method, of TestCase objects, 185
assertRaises() method, of TestCase objects, 185
assignment
  and reference counting, 34
  augmented, 61
  in-place operators, 61
  of instance attributes, 131
  of variables, 7
  to variables in nested functions, 97
associative array, 16, 44
associativity of operators, 78
astimezone() method, of datetime objects, 340
async class, async module, 452
async module, 452
  use of, 467
asynchronous I/O, 415
asynchronous networking
  and blocking operations, 469
  when to consider, 467
asyncore module, 415, 455
  use of, 467
AsyncResult objects, multiprocessing module, 425
atan() function, math module, 251
atan2() function, math module, 251
atanh() function, math module, 251
atexit module, 179, 219
atomic operations, disassembly, 193
attach() method, of Message objects, 556
attrgetter() function, operator module, 275
attrib attribute, of Element objects, 576
attribute assignment, on instances, 131
attribute binding operator ., 33, 76
  optimization of, 196
attribute binding
  and inheritance, 119
  and methods, 48
  instances and classes, 118
  of user-defined objects, 131
  process of, 57
  redefining in classes, 132
  special methods for, 57
attribute deletion, on instances, 131
attribute lookup in string formatting, 72
AttributeError exception, 87, 213
  and attribute binding, 132
attributes attribute, of DOM Node objects, 570
attributes
  computed as properties, 117, 124
  creation in __init__() method, 118
  descriptors, 58, 126
  encapsulation of, 127
  lookup in composite string formatting, 42
  of objects, 33
  private, 127
  restricting names with __slots__, 132
  user defined on functions, 114
audioop module, 588
augmented assignment operators, 61, 75
authentication, fetching URLs, 519
authkey attribute, of Process objects, 416
awk UNIX command, similarity to list comprehensions, 111

How can we make this index more useful? Email us at indexes@samspublishing.com
BaseRequestHandler class, SocketServer module, 490
__bases__ attribute
    of classes, 131
    of types, 50
basestring variable, 202
basicConfig() function, logging module, 355
BasicContext variable, decimal module, 248
.bat files, Windows, 176
bdb module, 585
Beautiful Soup package, 563
betavariate() function, random module, 255
bidirectional() function, unicodedata module, 293
big endian format, 167
big endian, packing and unpacking, 292
bin() function, 77, 202
binary data structures, packing and unpacking, 290
binary distribution, creating with distutils, 153
binary file mode, 159
binary files, 350
    buffered I/O, 351
    caution on using line-oriented functions, 351-352
binary integer literals, 27
Binary() function
database API, 301
    xmlrpc.client module, 526
binascii module, 547
bind() method
    of SocketServer class, 493
    of dispatcher objects, 456
    of socket objects, 478
binhex module, 587
bisect module, 261
bisect() function, bisect module, 261
bisect_left() function, bisect module, 261
bisect_right() function, bisect module, 261
bitwise operations and native integers, 66
    bitwise-and operator &, 65
    bitwise-negation operator ~, 65
    bitwise-or operator |, 65
    bitwise-xor operator ^, 65
blank lines, 26
block_size attribute, of digest objects, 559
blocking operations, and asynchronous networking, 469
Bluetooth protocol, 470
    address format, 472
BOM (byte order marker), 280
    and Unicode, 168
BOM_* constants, codecs module, 280
bool type, 38
bool() function, 202
    __bool__() method, 56, 58
boolean expressions, 9, 77
    evaluation rules, 78
boolean operators, 66
boolean values, 27, 38
boolean() function, xmlrpc.client module, 525
bound method, 49, 125
BoundedSemaphore object
    multiprocessing module, 427
    threading module, 439
BoundedSemaphore() method, of Manager objects, 428
break statement, 83-84
    and generators, 103
breaking long statements on multiple lines, 9
breakpoint
    setting in debugger, 187
    setting manually, 186
browser, launching from Python, 544
BSD, kqueue interface, 460
BTPROTO_* constants, socket module, 477
buffer, circular, 262
buffer_info() method, of array objects, 259
buffered binary I/O, 351
BufferedReader class, io module, 352
BufferedRandom class, io module, 351
BufferedReader class, io module, 352
built_in exceptions, 23, 87
built-in functions and types, 201
built-in functions, using Python 3 functions in Python 2, 217
built-in types, 37
__builtin__ module, 201
builtin_module_names variable, sys module, 229
BuiltinFunctionType, 49
BuiltinFunctionType type, 47, 237
builtins module, Python 3, 201
byref() function, ctypes module, 615
byte literals, 29
byte strings, 41
and WSGI, 541
and files, 160
and system interfaces in Python 3, 630
as in-memory binary files, 352
decoding as Unicode, 165
different behavior in Python 3, 629
lack of formatting in Python 3, 629
mixing with Unicode strings, 70, 167
mutable byte arrays, 202
use in system interfaces, 633
bytearray() function, 202
byteorder variable, sys module, 229
bytes datatype, Python 3, 29
bytes() function, 202-203
bytes, escape code in strings, 28
BytesIO class, io module, 352
byteswap() method, of array objects, 259
bz2 module, 313
BZ2Compressor() function, bz2 module, 313
BZ2Decompressor() function, bz2 module, 314
BZ2File() function, bz2 module, 313

C
C extensions, 591
and .egg files, 147
and module reloading, 149
compiling with distutils, 596
creating with SWIG, 619
example with ctypes, 618
releasing global interpreter lock, 444
-c command line option, 173-174
C#, 620
c(ont(inue)) debugger command, pdb module, 188
C++, difference in class system, 119
C
Python variables compared to, 7
implementation of functions, 49
C/C++ code, in third-party packages, 154
C3 linearization algorithm, and inheritance, 121
c_* datatypes, ctypes module, 614
CacheFTPHandler class, urllib.request module, 518
caching results of a function, 242
calcsize() function, struct module, 291
calendar module, 588
call() function, subprocess module, 403
__call__() method, 50, 62
Callable abstract base class, 265
callable objects
and __call__() method, 62
classes, 50
instances, 50
types of, 47
callback functions, 98
and lambda, 112
calling Python functions from C, 610
calling a function, 18, 93
__callmethod() method, of BaseProxy objects, 433
callproc() method, of Cursor objects, 298
cancel() method, of Timer objects, 438
cancel_join_thread() method, of Queue objects, 418
CannotSendHeader exception, http.client module, 504
CannotSendRequest exception, http.client module, 504
capitalize() method, of strings, 41-42
capitals attribute, of Context objects, 247
capwords() function, string module, 290
case conversion, of strings, 43-44
case sensitivity, of identifiers, 26
case statement, lack of, 9
cast() function, ctypes module, 617
catching all exceptions, 85
catching multiple exceptions, 85
category() function, unicodedata module, 170, 293
__cause__ attribute, of Exception objects, 213, 627
cautions with range() function, 17
cDLL() function, ctypes module, 612
ceil() function, math module, 251
center() method, of strings, 41-42
cert_time_to_seconds() function, ssl module, 488
cGI script, 533
    advice for writing, 537
environment variables, 533
    executing XML-RPC server within, 529
    running WSGI application, 542
    use of databases, 538
    web frameworks, 538
cgi module, 533
CGIHandler() function, wsgiref.handlers module, 542
CGIHTTPRequestHandler class, http.server module, 507
CGIHTTPServer module, see http.server, 506
cgitb module, 539
cGIXMLRPCRequestHandler class, xmlrpc.server module, 528
chain() function, itertools module, 270
chained comparisons, 66
chained exceptions, Python 3, 627
changing display of results, interactive mode, 176
changing module name on import, 144
changing the working directory, 379
changing user-agent header in HTTP requests, 517
character substitution, 42
characters() method, of ContentHandler objects, 581
characters
    escape codes, 28
    specifying Unicode, 28
chdir() function, os module, 379
check_call() function, subprocess module, 403
check_unused_args() method, of Formatter objects, 289
checking if running as main program, 146
checking multiple cases with a conditional, 9
chflags() function, os module, 386
chicken, multithreaded, 414
childNodes attribute, of DOM Node objects, 570
chmod() function, os module, 387
choice() function, random module, 254
chown() function, os module, 387
chr() function, 77, 203
chroot() function, os module, 379
chunk module, 588
cipher() method, of ssl objects, 487
circular buffer or queue with deque objects, 262
cl(ear) debugger command, pdb module, 188
class decorators, 102, 141
class method, 48, 123
    attribute binding of, 124
    practical use of, 124
class statement, 21, 117
    and inheritance, 21, 119
    execution of class body, 138
class variables, 117
sharing by all instances, 118
__class__ attribute
of instances, 50, 131
of methods, 49
classes, 21
__del__() method and garbage collection, 221-222
__init__() method, 118
__init__() method and inheritance, 120
__slots__ attribute, 132
abstract base class, 136, 257
access control specifiers, lack of, 127
accessing in modules, 144
and metaclasses, 138
as callable, 50
as namespaces, 117
attribute binding rules, 118
class method, 203
creation of instances, 22, 55, 118
customizing attribute access, 57-58
decorators applied to, 102, 141
defining methods, 21
descriptor attributes, 58, 126
difference from C++ or Java, 119
inheritance, 21, 119
inheriting from built-in types, 22
memory management, 128
mixin, 122
multiple inheritance, 120-121
object base class, 119
old-style, 139
operating overloading, 54
optimization of, 195
optimization of inheritance, 233
performance of __slots__, 196
pickling of, 228
private members, 26
redefining attribute binding, 132
scoping rules within, 118
self parameter of methods, 119
special methods, 54
static methods, 22
super() function in methods, 120
supporting pickle module, 228
type of, 47
uniform access principle, 125
versus dicts for storing data, 195
@classmethod decorator, 48, 123, 125, 203
ClassType type, old-style classes, 139
cleandoc() function, inspect module, 222
clear() method
of Element objects, 576
of Event objects, 440
of deque objects, 262
of dicts, 45
of sets, 46
clear_flags() method, of Context objects, 247
clear_memo() method, of Pickler objects, 228
__clear_type_cache() function, sys module, 233
clearing a dictionary, 45
clearing last exception, 233
Client class, multiprocessing module, 433
client program, 449
   TCP example, 451
   UDP example, 486
client_address attribute
   of BaseHTTPRequestHandler objects, 509
   of BaseRequestHandler objects, 490
clock() function, time module, 191, 405
cloneNode() method, of DOM Node objects, 571
close() function, os module, 381
close() method
   of Connection objects, 297, 421
   of Cursor objects, 298
   of FTP objects, 498
   of HTMLParser objects, 561
   of HTTPConnection objects, 503
   of Handler objects, 364
of IOBase objects, 349
of Listener objects, 434
of Pool objects, 424
of Queue objects, 418
of TarFile objects, 320
of TreeBuilder objects, 577
of ZipFile objects, 325
of dbm-style database objects, 310
of dispatcher objects, 456
of files, 159
of generators, 20, 53, 103, 105
of generators and synchronization, 104
of mmap objects, 371
of shelf objects, 311
of socket objects, 478
of urlopen objects, 515

close_when_done() method, of asyncchat objects, 452
close attribute
of IOBase objects, 349
of files, 161
closefd attribute, of FileIO objects, 350
closefd parameter, to open() function, 159
CloseKey() function, winreg module, 408
closerange() function, os module, 382
closing() function, contextlib module, 268
__closure__ attribute, of functions, 48, 100
closures, 98-99
and decorators, 101
and nested functions, 99
and wrappers, 100
speedup over classes, 100
cmath module, 251
cmd module, 588
cmp() function, 203
filecmp module, 314
cmpfiles() function, filecmp module, 314
co_ attributes, of code objects, 51-52
code execution, in modules, 143-144
code migration
Python 2 to 3, 634
practical strategy for, 637
code module, 585
code objects, 51
attributes of, 51
creating with compile() function, 115
code point, Unicode, 28
__code__ attribute, of functions, 48
__code__ attribute, of classes, 48
code, executing strings, 115
CodecInfo class, codecs module, 277
codes module, 167, 277
removal of compression codecs, 280
use of byte strings, 280
coded_value attribute, of Morsel objects, 512
codeop module, 585
CodeType type, 51, 237
__coerce__() method, deprecation of, 134
coercion of numeric types, 66-67
collect function, gc module, 179
collect() function, gc module, 220
collect_incoming_data() method, of asyncchat objects, 452
collection, definition of, 33
collections module, 138, 262
colorsys module, 588
combinations() function, itertools module, 271
combine() method, of datetime class, 339
combining() function, unicodedata module, 294
command attribute, of
BaseHTTPRequestHandler objects, 509
command line options, 13, 157
Python 3, 633
detecting settings in a program, 230
for interpreter, 173
parsing with optparse, 374
commands debugger command, pdb module, 188
commands module, 331
comment attribute, of ZipInfo objects, 327
Comment() function, xml.etree.ElementTree module, 575
comments, 6, 26
commit() method, of Connection objects, 298
common attribute, of dircmp objects, 315
common_dirs attribute, of dircmp objects, 315
common_files attribute, of dircmp objects, 315
common_funny attribute, of dircmp objects, 315
commonprefix() function, os.path module, 396
communicate() method, of Popen objects, 403
comparison operators, 56
comparison, 66
  Python 3, 633
  chained, 66
  of incompatible objects, 78
  of objects, 34
  of sequences, 70
  of weak references, 242
compilation into bytecode, 148
compile() function, 115, 203
  re module, 283–284
compileall module, 585
compiler, lack of, 181
complete_statement() function, sqlite3 module, 305
Complex abstract base class, 253
complex numbers, 27, 39
  cmath library module, 251
  comparison of, 66
complex type, 38
complex() function, 76, 203
  __complex__() method, 61–62
    and type coercion, 134
composing email messages, 555
composite string formatting, 8, 42, 72
  and __format__(), 56
  and lookups, 42
compress() function
  bz2 module, 314
  zlib module, 328
compress() method
  of BZ2Compressor objects, 313
  of compressobj objects, 328
compress_size attribute, of ZipInfo objects, 327
compress_type attribute, of ZipInfo objects, 327
compression
  of files, 313, 317
  zlib compression, 328
CompressionError exception, tarfile module, 322
compressobj() function, zlib module, 328
computed attributes and properties, 124
concat() function, operator module, 274
concatenation
  of adjacent string literals, 27
  of lists, 12
  of strings, 11
concurrency, 413
  advice on multiprocessing, 435
  and Python programs, 414
  and side effects, 96
  coroutines, 446
  global interpreter lock, 414
  limitations on multicore, 414
  message passing, 413–415
  multitasking with generators, 447
  scaling issues, 415
  synchronization problems, 414
concurrent programming, 413
Condition object
  multiprocessing module, 427
  threading module, 441
condition debugger command, pdb module, 188
condition variable, 441
Condition() method, of Manager objects, 429
conditional expressions, 79
conditionals, 9, 81
ConfigParser class, configparser module, 332
configparser module, 332
configuration files, 332
difference from Python script, 334-335
for logging module, 368
variable substitution, 335
confstr() function, os module, 395
conjugate() method
of complex numbers, 39
of floating point, 39
connect() function
database API, 297
sqlite3 module, 304
connect() method
of BaseManager objects, 431
of FTP objects, 498
of HTTPConnection objects, 503
of SMTP objects, 514
of dispatcher objects, 456
of socket objects, 478
connect_ex() method, of socket objects, 478
connecting processes, multiprocessing module, 433
Connection class
database API, 297
sqlite3 module, 305
ConnectRegistry() function, winreg module, 408
console window, Windows, 176
Container abstract base class, 265
container objects, 29
and reference counting, 34
definition of, 33
containment test, in operator, 9
contains() function, operator module, 274
__contains__() method, 58
ContentHandler class, xml.sax module, 581
ContentTooShort exception, urllib.error module, 523
Context class, decimal module, 244
context management protocol, 62
context managers, 62, 89
decimal module, 248
defining with generator, 267
locking, 442
nested, 267
__context__ attribute, of Exception objects, 213, 627
contextlib module, 90, 267
@contextmanager decorator, 90
continue statement, 83-84
control characters, stripping from a string, 42
conversion of strings to numbers, 11
conversion operations, 76
convert_field() method, of Formatter objects, 289
converting Python types to C, 611
converting dictionaries to a list, 16
converting sequences to a list, 77
converting sequences to a set, 15
converting sequences to a tuple, 77
converting types from C to Python, 602
converting types from Python to C, 597
Cookie module, see http.cookies, 511
CookieError exception, http.cookies module, 513
CookieJar class, http.cookiejar module, 513
cookielib module, see http.cookiejar, 513
cookies
HTTP, 511
fetching URLs with cookie support, 519
copy module, 36, 67, 219
limitations of, 220
copy() function
copy module, 219
shutil module, 318
copy() method
of Context objects, 247
of dicts, 45
of digest objects, 559
of hmac objects, 560
of sets, 46
copy2() function, shutil module, 318
__copy__() method, 220
copy_reg module, 585
copyfile() function, shutil module, 318
copyfileobj() function, shutil module, 318
copying directories, 318
copying files, 318
copying
and reference counting, 35
depth copy, 36
dictionary, 45
of mutable objects, 35
shallow copy, 36
copymode() function, shutil module, 318
copyright variable, sys module, 230
copysign() function, math module, 251
copystat() function, shutil module, 318
copytree() function, shutil module, 318
@coroutine decorator example, 105
coroutines, 20, 104
advanced example, 460
asynchronous I/O handling, 460
building a call stack of, 463
concurrency, 108
concurrent programming, 446
example of, 20
execution behavior, 105
message passing, 108, 415
multitasking example, 447
practical use of, 107
recursion, 112
sending and returning values, 106
task scheduler with select(), 460
transferring control to another coroutine, 463
use of next() method, 104
use with network programming, 467
cos() function, math module, 251
cosh() function, math module, 251
count() function, itertools module, 271
count() method
of array objects, 260
of lists, 40
of strings, 42
counting, in loops, 83
countOf() function, operator module, 274
cp1252 encoding, description of, 169
cp437 encoding, description of, 169
cPickle module, 229
cProfile module, 190
CPU time, obtaining, 191, 405
CPU, obtaining number on system, 434
CPU-bound tasks and threads, 444
cpu_count() function, multiprocessing module, 434
CRC attribute, of ZipInfo objects, 327
crc32() function
binascii module, 548
zlib module, 328
crc_hqx() function, binascii module, 548
cREATE AGGREGATE() method, of Connection objects, 305
cREATE_COLLATION() method, of Connection objects, 306
cREATE_CONNECTION() function, socket module, 473
cREATE_DECIMAL() method, of Context objects, 247
cREATE_FUNCTION() method, of Connection objects, 305
CREATE_SOCKET() method, of dispatcher objects, 456
CREATE_STRING_BUFFER() function, ctypes module, 617
CREATE_SYSTEM attribute, of ZipInfo objects, 327
CREATE_UNICODE_BUFFER() function, ctypes module, 617
CREATE_VERSION attribute, of ZipInfo objects, 327
created attribute, of Record objects, 359
CreateKey() function, winreg module, 408
creating a Windows installer, 153
creating a binary distribution, 153
creating a source distribution, 153
creating custom string formatters, 288

How can we make this index more useful? Email us at indexes@samspublishing.com
database
    and CGI script, 538
    persistent dictionary, 171
DatabaseError exception, database API, 302
databases, DBM-style, 310
DataError exception, database API, 302
DatagramHandler class, logging module, 362
DatagramRequestHandler class,
    SocketServer module, 491
datagrams, 470
date and time manipulation, 336
date class, datetime module, 336
date parsing, 343, 407
Date() function, database API, 301
date() method, of datetime objects, 340
date_time attribute, of ZipInfo objects, 327
DateFromTicks() function, database API,
    301
datetime class, datetime module, 339
datetime module, 336
DateTime() function, xmlrpc.client module,
    526
day attribute, of date objects, 337
daylight variable, time module, 405
dbhash module, 310
dbm module, 310
DBM-style databases, 310
deadlock, source with locking, 442
debug attribute
    of TarFile objects, 320
    of ZipFile objects, 325
    of sys.flags, 230
__debug__ variable, 91, 369
debug() method, of Logger objects, 357
debugging
    CGI scripts, 539
    after an uncaught exception, 186
    checking for memory leaks, 221
    configuring the debugger, 190
    entire programs from command shell, 189
    manually setting a breakpoint, 186
    pdb module, 186
    running a function, 186
    specifying a breakpoint, 187
    use of logging module, 355
Decimal class, decimal module, 243
Decimal object, converting to a fraction, 250
decimal module, 39, 243
    and sum() function, 69
    and threads, 249
    rounding behavior, 245
decimal() function, unicodedata module,
    295
declarative programming, 110
decode() function
    base64 module, 547
    quopri module, 568
decode() method
    in Python 3, 629
    of CodecInfo objects, 277
    of IncrementalDecoder objects, 279
    of JSONDecoder objects, 565
    of strings, 29, 42, 165–166
    proper use of, 166
decodestring() function
    base64 module, 547
    quopri module, 568
decomposition() function, unicodedata
    module, 295
decompress() function
    bz2 module, 314
    zlib module, 328
decompress() method
    of BZ2Decompressor objects, 314
    of decompressobj objects, 328
decompressobj() function, zlib module, 328
decorators, 22, 101
    applied to class definitions, 102, 141
    copying function attributes, 269

How can we make this index more useful? Email us at indexes@samspublishing.com
documentation strings, 102, 113
example of, 101
multiple, 31
performance benefits, 197
placement of, 30, 101
recursive functions, 102, 113
user-defined function attributes, 102, 114
with arguments, 102
deep copy, 36
deeppcopy() function, copy module, 219
__deeeepcopy__() method, 220
def statement, 18, 48, 93
default Unicode encoding, 166, 177
default Unicode error handling policy, 167
default arguments, 18, 93
and mutable values, 94
binding of values, 93
default() method, of JSONEncoder objects, 566
default_factory attribute, of defaultdict objects, 263
DefaultContext variable, decimal module, 248
defaultdict() function, collections module, 263
__defaults__ attribute, of functions, 48
defaults() method, of ConfigParser objects, 333
defects attribute, of Message objects, 554
defining functions, 18
defining multiple instance creation methods, 123
defining new exceptions, 88
degrees() function, math module, 251
del operator, on dictionaries, 16, 74
del statement, 35, 69
and __del__() method, 129
and slices, 40
deleting mapping items, 45
__del__() method, 55, 129
and program termination, 179
danger of defining, 129
garbage collection, 129, 221-222
uses of, 55
del_param() method, of Message objects, 556
deattr() function, 203
and private attributes, 128
__deattr__() method, 57-58, 131
delayed evaluation, 99
delayed execution, using threads, 437
delete() method, of FTP objects, 498
__delete__() method, of descriptors, 58, 126
DeleteKey() function, winreg module, 409
@deleter decorator of properties, 126
DeleteValue() function, winreg module, 409
deleting items from a dictionary, 16
deleting sequence items, 40
deleting slices, 40, 69
deletion of instance attributes, 131
delimiters, 30
delitem() function, operator module, 274
__delitem__() method, 58-59
and slices, 59
delslice() function, operator module, 274
demo_app() function, wsgiref.simple_server module, 542
denominator attribute
of Fraction objects, 250
of integers, 39
DeprecationWarning warning, 216, 238
deque object
collections module, 194
versus list, 194
deque() function, collections module, 262
DER_cert_to_PEM_cert() function, ssl module, 488
dereference attribute, of TarFile objects, 320
derived class, 119
description attribute, of Cursor objects, 299
descriptors, 58, 126
and metaclasses, 140
detecting end of file (EOF), 160
detecting the settings of interpreter command line options, 230
devnull variable, os module, 386
Dialect class, csv module, 551
dict type, 38
__dict__ attribute
  of classes, 131
  of functions, 48, 114
  of instances, 50, 131
  of modules, 51, 144
  of types, 50
  of user-defined objects, 63
dict() function, 16, 77, 204
  performance properties, 195
dict() method, of Manager objects, 429
dictionary comprehension, Python 3, 623
dictionary, 16, 44
  acceptable key types, 16
  accessing items, 16
  and Python 3 caution, 45
  and __hash__() method, 56
  and string formatting, 70, 72
  automatic creation of initial values, 263
  clearing, 45
  compared to defaultdict objects, 263
  converting to a list, 16
  copying, 45
  creating from database results, 303
  creation with dict() function, 204
  defining empty, 16
  deleting items, 16, 45
  equality of, 78
  indexing operator, 74
  inserting items, 16
  item assignment, 74
  item deletion, 74
  iterating over keys, 17
  key values, 44, 74
  list of items, 45
  lookup in composite string formatting, 42, 72
  lookup with default value, 16
  obtaining keys, 45-46
  obtaining values, 45
  performance of, 16
  performance of in operator, 197
  persistent with shelve module, 171
  removing items, 16
  shared by multiple processes, 429
  tuples as keys, 74
  updating, 45
  use as a data structure, 16
  use as a lookup table, 16
  using functions as values, 37
  using to pass keyword function arguments, 95
  view objects in Python 3, 632
DictReader() function, csv module, 550
dicts, versus classes for storing data, 195
DictWriter() function, csv module, 550
diff_files attribute, of dircmp objects, 315
difference operator -, of sets, 15
difference() method, of sets, 46
difference_update() method, of sets, 46
difflib module, 586
dig attribute, of sys.float_info, 231
digest() method
  of digest objects, 559
  of hmac objects, 560
digest_size attribute, of digest objects, 559
digit() function, unicodedata module, 295
digits variable, string module, 287
dir() function, 21, 24, 204
  hiding attribute names in classes, 128, 204
dir() method, of FTP objects, 498
__dir__() method, 63, 128, 204
dircmp() function, filecmp module, 314
directories
  comparing, 314
  copying, 318
  reading files with shell wildcards, 317
  recursive traversal, 390

How can we make this index more useful? Email us at indexes@samspublishing.com
system functions for accessing, 386
temporary, 323
testing filenames for, 397
dirname() function, os.path module, 396
dis module, 585
dis(), dis module, 193
disable debugger command, pdb module, 188
disable() function
gc module, 220
logging module, 366
disable_interspersed_args() method, of
OptionParser objects, 376
disabling garbage collection, 220
disabling newline translation, 159
disassembly, 193
discard() method, of sets, 47
discard_buffers() method, of asynchat
objects, 452
dispatcher class, asyncore module, 455
__displayhook__ variable, sys module, 230
displayhook() function, sys module, 176, 233
disposition attribute, of FieldStorage
objects, 535
disposition_options attribute, of
FieldStorage objects, 535
distributed computing, and multiprocessing
module, 435
distributing programs, 152
distutils module, 152-153, 585, 596
and extension modules, 596
creating binary distributions, 153
creating extensions with SWIG, 620
div() function, operator module, 273
__div__() method, 60
division of integers, Python 3, 633
division operator /, 65
division operator, Python 2 versus Python
3, 61
division, truncation of integers, 61-62, 65
division_new attribute, of sys.flags, 230
division_warning attribute, of sys.flags, 230
divmod() function, 66, 204
__divmod__() method, 60
dllhandle variable, sys module, 230
DLLs
creating with distutils, 596
extension modules, 148
loading with ctypes, 612
do_handshake() method, of ssl objects, 487
__doc__ attribute
of built-in functions, 49
of functions, 24, 48, 113
of methods, 49
of modules, 51
of objects, 30
of types, 50
DocCGIXMLRPCRequestHandler class, xmlrpc.
server module, 528
doctest module, 181-182
verbose option, 182
Document class, xml.dom.minidom mod-
ule, 571
documentation string, copying to decorator, 269
documentation strings, 24, 30, 48, 113
and XML-RPC, 527
and decorators, 102, 113
and indentation, 30
doctest module, 181
in extension modules, 595
stripping with -OO option, 148
testing, 181
documentElement attribute, of DOM
Document objects, 571
DocXMLRPCServer class, xmlrpc.server
module, 527
DocXMLRPCServer module, 527
dollar-variable substitution, 163
DOM interface
XML parsing, 568
example of, 573
dont_write_bytecode attribute, of sys.flags, 230
dont_write_bytecode variable, sys module, 230
double precision floating point, 38
double-clicking on .py files, 6
double-underscores, use in identifiers, 26
dropwhile() function, itertools module, 271
dst() method
    of time objects, 338
    of tzinfo objects, 342
duck typing, 122
dumbdbm module, 310
dump() function
    json module, 564
    marshal module, 226
    pickle module, 171, 227
    xml.etree.ElementTree module, 578
dump() method, of Pickler objects, 228
dumps() function
    json module, 564
    marshal module, 226
    pickle module, 227
    xmlrpc.client module, 526
dup() function, os module, 382
dup2() function, os module, 382
dynamic binding, of object attributes, 122
dynamic loading, of modules, 144
dynamic scope, lack of, 97
dynamic typing of variables, 7

E

-E command line option, 173
e variable, math module, 252
EAI_* constants, socket module, 485
east_asian_width() function, unicodedata module, 296
easy_install command, setuptools package, 155
.egg files, 154
    and modules, 147
    and site configuration, 177
    structure of, 147
Element class, xml.dom.minidom module, 572

Element() function, xml.etree.ElementTree module, 575
ElementTree class, xml.etree.ElementTree module, 573
ElementTree interface, XML parsing, 569
ElementTree, examples of, 578
elif statement, 9, 81
Ellipsis, 30, 51, 54
    expression in Python 3, 626
    type of, 51
    use in extended slicing, 59
    use in indexing methods, 54
else clause
    of try statement, 86
    of while and for loops, 84
else statement, 9, 81
email messages
    composing, 555
    example of composing and sending, 558
    example of sending, 514
    parsing, 552
e-mail package, 552
Emax attribute, of Context objects, 247
embedding Unicode characters in web pages, 167
embedding the interpreter in C programs, 591, 608
embedding
    calling functions from C, 610
    converting Python types to C, 611
Emin attribute, of Context objects, 247
Empty exception, Queue module, 418, 445
empty dictionary, 16
empty list, 12
empty() method, of Queue objects, 418, 445
enable debugger command, pdb module, 188
enable() function
    cgitb module, 539
    gc module, 220

How can we make this index more useful? Email us at indexes@samspublishing.com
enable_callback_tracebacks() function, sqlite3 module, 305
enable_interspersed_args() method, of OptionParser objects, 376
enabling the print() function in Python 2.6, 163
encapsulation, 127
encode() function
  base64 module, 547
  quopri module, 568
encode() method
  in Python 3, 629
  of CodecInfo objects, 277
  of IncrementalEncoder objects, 278
  of JSONEncoder objects, 566
  of strings, 42, 165-166
  proper use of, 166
EncodedFile class, codecs module, 279
EncodedFile object, codecs module, 167
encodestring() function
  base64 module, 547
  quopri module, 568
encoding argument to open() function, 159
encoding attribute
  of TextIOWrapper objects, 353
  of files, 161
encoding issues with network programming, 452
encoding, of source code, 31
end attribute, of slices, 54
end keyword argument, to print() function, 163
end() method
  of MatchObject objects, 286
  of TreeBuilder objects, 577
end_headers() method, of BaseHTTPRequestHandler objects, 509
endDocument() method, of ContentHandler objects, 581
endElement() method, of ContentHandler objects, 581
endElementNS() method, of ContentHandler objects, 581
endheaders() method, of HTTPConnection objects, 503
endpos attribute, of MatchObject objects, 286
endPrefixMapping() method, of ContentHandler objects, 581
endswith() method, of strings, 42
__enter__() method, of context managers, 62, 89
enumerate() function, 83, 204
  threading module, 443
EnumKey() function, winreg module, 409
EnumValue() function, winreg module, 409
environ variable, os module, 158
environ variable, os module, 379
environment variables, 158, 379
  Python 3, 633
  WSGI, 540
  expanding in filenames, 397
  in CGI script, 533
  unsetting, 381
  used by interpreter, 174
EnvironmentError exception, 87, 212
EOF character, interactive mode, 7
EOF indication, file I/O, 160
EOFError exception, 87, 213
epilogue attribute, of Message objects, 554
epoll interface, Linux, 460
epsilon attribute, of sys.float_info, 231
eq() function, operator module, 274
__eq__() method, 57
equal to operator ==, 66, 78
equality comparison of objects, 34
errcheck attribute, of ctypes function objects, 613
errno module, 343
error codes, list of system errors, 344
error message format attribute, of BaseHTTPRequestHandler class, 508
errorcode variable, errno module, 344
errorlevel attribute, of TarFile objects, 320
errors attribute, of TextIOWrapper objects, 353
errors parameter
    of encoding functions, 166
to open() function, 159
escape codes
    disabling in a string literal, 29
    in string literals, 27
    non-printing characters, 28
escape() function
    cgi module, 536
    re module, 283
    xml.sax.saxutils module, 583
escaping characters for use in HTML, 536
eval() function, 55, 77, 115, 204, 206
    and repr(), 55
evaluation
    of function arguments, 76
    operator precedence and associativity, 78
    order of, 78
Event object
    multiprocessing module, 427
    threading module, 440
event loop
    and asyncore module, 455
    coroutines, 108
Event() method, of Manager objects, 429
event-driven I/O, 415
    polling for signals, 399
    when to consider, 467
EX_* exit code constants, 391
exc_clear() function, sys module, 233
exc_info attribute, of Record objects, 359
exc_info() function, sys module, 53, 89, 233
except statement, 22, 84-85
    change of syntax, 85
    __excepthook__ variable, sys module, 230
    excepthook() function, sys module, 85, 233
Exception class, 212
Exception exception, 87
exception handling, in extension modules, 605
exception() method, of Logger objects, 358
exceptions, 22, 84-85
    and locks, 23
    attributes of, 212
    catching all, 85
    catching multiple types, 85
    caution with catching all exceptions, 86
    chained in Python 3, 626
    clearing last exception, 233
    defining new, 88
    difference from warnings, 216
    error codes for system errors, 344
    finally statement, 86
    handling of, 22-23
    hierarchy of, 88
    ignoring, 85
    list of built-in, 87
    matching rules, 85
    optimization strategies, 196
    performance of, 197
    propagation of, 85
    reraising the last exception, 84
    value of, 23, 85
.exe file, creating with distutils, 153
exec statement, caution with legacy code, 115
exec() function, 115, 204
    Python 3, 631
exec_prefix variable, sys module, 177, 230
execl() function, os module, 390
execlp() function, os module, 390
executable variable, sys module, 230
execute() method
    of Connection objects, 306
    of Cursor objects, 298
executemany() method
    of Connection objects, 306
    of Cursor objects, 298

How can we make this index more useful? Email us at indexes@samspublishing.com
executescript() method, of Connection objects, 306
executing programs, 6
executing strings as code, 115
executing system commands, 331
  popen() function, 392
  subprocess module, 402
  system() function, 393
execution model, 81
execution of __init__.py files, 150
execution of class bodies, 117, 138
execv() function, os module, 390
execve() function, os module, 391
execvp() function, os module, 391
execvpe() function, os module, 391
exists() function, os.path module, 396
  _exit() function, os.path module, 396
  exit() function, sys module, 179, 391
  __exit__() method, 63
    of context managers, 62, 89
exitcode attribute, of Process objects, 417
exp() function, math module, 251
exp() method, of Decimal objects, 243
expand() method, of MatchObject objects, 285
ExpandEnvironmentStrings() function, winreg module, 409
expandtabs() method, of strings, 41-42
expanduser() function, os.path module, 397
expandvars() function, os.path module, 397
exponents, range on floating point, 38
expovariate() function, random module, 255
expressions, 7
extend() method
  of array objects, 260
  of deque objects, 262
  of lists, 40
extended slices, 39, 59
  assignment to, 40, 69
  deletion of, 40, 69
  on sequences, 68
extended slicing operator [:], 67
extended unpacking of iterables, Python 3, 623
ExtendedContext, decimal module, 248
extending with C, 591
extendleft() method, of deque objects, 262
extensible code, with modules, 144
extension modules, 591
  compilation with distutils, 596
  converting types from C to Python, 602
  converting types from Python to C, 597
crtypes module, 612
difference in Python 3, 595
documentation strings, 595
documentation of, 593
exception handling, 605
global interpreter lock, 607
hand-written, 593
manual compilation, 597
  naming of, 595
  reference counting, 607
  threads, 607
  wrapper functions, 594
Extension() function, distutils module, 596
extensions_map attribute, of
  HTTPRequestHandler class, 508
external_attr attribute, of ZipInfo objects, 327
extra attribute, of ZipInfo objects, 327
extract() method
  of TarFile objects, 320
  of ZipFile objects, 325
extract_stack() function, traceback module, 236
extract_tb() function, traceback module, 236
extract_version attribute, of ZipInfo objects, 327
extractall() method, of ZipInfo objects, 325
ExtractError exception, tarfile module, 322
extractfile() method, of TarFile objects, 320
extsep variable, os module, 386
F

F_* constants, fcntl() function, 347
f_* attributes
  of frame objects, 52
  of statvfs objects, 389
fabs() function, math module, 251
factorial() function, math module, 251
fail() method, of TestCase objects, 185
failIf() method, of TestCase objects, 185
failIfAlmostEqual() method, of TestCase objects, 185
failIfEqual() method, of TestCase objects, 185
failUnless() method, of TestCase objects, 185
failUnlessAlmostEqual() method, of TestCase objects, 185
failUnlessEqual() method, of TestCase objects, 185
failUnlessRaises() method, of TestCase objects, 185
failureException attribute, of TestCase objects, 185
False value, 9, 27, 38
family attribute, of socket objects, 484
Fault exception, xmlrpc.client module, 527
fchdir() function, os module, 379
fchmod() function, os module, 382
fchown() function, os module, 382
fcntl module, 347
fcntl() function, fcntl module, 347
fdatasync() function, os module, 382
fdopen() function, os module, 382
feed() method, of HTMLParser objects, 561
fetchall() method, of Cursor objects, 299
fetching URLs
  example of, 514
  example with authentication, 519
  example with cookies, 519
fetchmany() method, of Cursor objects, 299
fetchone() method, of Cursor objects, 298
FieldStorage() function, cgi module, 534
file I/O, 10
file attribute, of FieldStorage objects, 535
file descriptors, 347
  functions for manipulation, 382
file keyword argument, to print() function, 10, 163
file locking, 348
  Windows, 373
    by sqlite3 module, 303
file modes, use with open() function, 159
file upload, in CGI scripts, 536
__file__ attribute, of modules, 51
file-like objects, 122
file_offset attribute, of ZipInfo objects, 327
file_size attribute, of ZipInfo objects, 327
filecmp module, 314
fileConfig() function, logging module, 367
FileCookieJar class, http.cookiejar module, 513
FileHandler class
  logging module, 362
    urllib.request module, 518
FileIO class, io module, 350
filename attribute
  of FieldStorage objects, 535
  of Record objects, 359
    of ZipInfo objects, 327
filenames
  Windows drive letters, 399
    absolute path of, 396
    in Python 3, 633
    matching with shell wildcards, 316
    portable manipulation of, 396
    splitting into directory and base name, 398
    testing for existence, 396
    testing if directory, 397
    testing if link, 397

How can we make this index more useful? Email us at indexes@samspublishing.com
fileno() method
   of Connection objects, 421
   of IOBase objects, 349
   of SocketServer objects, 491
   of files, 160-161
   of files and sockets, 459
   of socket objects, 478
   of urlopen objects, 515
files, 10
   absolute path of, 396
   and Python 3, 160
   attributes of, 161
   buffer size, 159
   buffered binary I/O, 351
   bz2 compression, 313
   comparing, 314
   copying, 318
   creation time, 397
   decoding as Unicode, 167
   description of file modes, 159
   detecting end of file (EOF), 160
   file pointer, 161, 352
   finding on the filesystem, 390
   functions for manipulating metadata, 386
   gzip compression, 317
   iterating over lines, 17-18
   last modification time, 397
   locking on Windows, 373
   low-level control, 347
   low-level system calls, 382
   memory mapped, 370
   methods on, 159
   opening, 10, 158-159
   opening with Unicode decoding, 167
   parsing CSV, 548
   problems with io library module, 354
   raw binary I/O, 350
   reading line by line, 10
   seeking, 161
   size of, 397
   softspace attribute and print statement, 162
   temporary, 323
   testing for existence, 396
   types of, 159
   writing to, 159
fill characters in string format specifiers, 73
filter class, logging module, 359
filter() function, 205
   and Python 3, 205
   and optimization, 197
   fnmatch module, 316
   future_bultins module, 217
filterwarnings() function, warnings module, 239
finally statement, 86
   and locks, 442
find() method
   of Element objects, 576
   of ElementTree objects, 574
   of mmap objects, 371
   of strings, 41-42
find_library() function, ctypes module, 612
findall() function, re module, 283
findall() method
   of Element objects, 576
   of ElementTree objects, 574
   of Regex objects, 285
findCaller() method, of Logger objects, 358
finding all loaded modules, 144
finding files, 390
finditer() function, re module, 284
finditer() method, of Regex objects, 285
findtext() method
   of Element objects, 576
   of ElementTree objects, 574
   of Regex objects, 285
finish() method, of BaseRequestHandler objects, 490
first-class objects, 36
   use of, 37
firstChild attribute, of DOM Node objects, 570
flag_bits attribute, of ZipInfo objects, 327
flags attribute
   of Context objects, 247
   of Regex objects, 284
flags variable, sys module, 230
flaming death, in extension modules, 595
float type, 38
float() function, 13, 62, 76, 205
__float__() method, 61-62
and type coercion, 134
float_info variable, sys module, 231
floating point, 27
as dictionary key, 16
binary representation, 39
compared to decimal numbers, 243
converting to a fraction, 39, 250
defining NaN and Inf, 213, 252
inexact representation, 12, 243
low-level properties of, 231
mixing with complex numbers, 39
precision of, 38
random number distributions, 255
representation of, 38
FloatingPointError exception, 87, 213
flock() function, fcntl module, 348
floor division, 65
floor() function, math module, 251
floordiv() function, operator module, 273
__floordiv__() method, 60
flush() method
of BZ2Compressor objects, 314
of BufferWriter objects, 352
of Handler objects, 364
of IOBase objects, 349
of compressobj objects, 328
de of decompressobj objects, 329
of files, 160
of mmap objects, 371
FlushKey() function, winreg module, 409
fma() method, of Decimal objects, 243
fmod() function, math module, 251
fnmatch module, 316
fnmatch() function, fnmatch module, 316
fnmatchcase() function, fnmatch module, 316
foot, how to shoot, 36, 86, 442, 617
for statement, 10, 17, 59, 69, 82
and files, 10, 160
and generators, 19
and tuple unpacking, 15
forcing garbage collection, 220
foreign function interface, ctypes module, 612
fork() function, os module, 391
ForkingMixIn class, SocketServer module, 493
ForkingTCPserver class, SocketServer module, 494
ForkingUDPserver class, SocketServer module, 494
forkpty() function, os module, 392
format attribute, of Struct objects, 291
format codes
for dates and times, 406
for string formatting operator %, 70-71
format specifiers
alignment characters, 73
customized, 74
fill character, 73
format() method of strings, 72-74
nesting of fields, 74
format() function, 8, 11-12, 56, 77, 205
format() method
format specifier codes, 72
of Formatter objects, 288
of strings, 8, 42-43, 56, 72-73, 162
of strings and variable interpolation, 164
__format__() method, 55-56, 74
format_exc() function, traceback module, 236
format_exception() function, traceback module, 236
format_exception_only() function, traceback module, 236
format_list() function, traceback module, 236
format_stack() function, traceback module, 236
How can we make this index more useful? Email us at indexes@samspublishing.com
format_tb() function, traceback module, 236
format_value() method, of Formatter objects, 289
formatargspec() function, inspect module, 222
formatargvalues() function, inspect module, 222
formatted printing, 8, 70-71, 162
formatted strings, 42, 71-72
Formatter class
logging module, 365
string module, 288
formatter module, 587
formatting, of log messages, 358, 365
formatwarning() function, warnings module, 239
Fortran common blocks, lack of, 146
found_terminator() method, of asynchat objects, 453
fpathconf() function, os module, 382
fpectl module, 585
fpformat module, 586
Fraction class, fractions module, 250
fractions module, 39, 250
fragment attribute
of urlparse objects, 520
of urlsplit objects, 521
frame objects, 51-52
attributes of, 52
FrameType type, 51, 237
free variables, in functions, 98
freeze_support() function, multiprocessing module, 434
frexp() function, math module, 251
from __future__ import, 178
from module import *, 24, 145
global variables, 146
identifiers with underscores, 26
__all__ variable, 145
from statement
and import statement, 24
module imports, 145
from_address() method, of ctypes type objects, 616
from_buffer() method, of ctypes type objects, 616
from_buffer_copy() method, of ctypes type objects, 616
from_decimal() method, of Fraction class, 250
from_float() method, of Fraction class, 250
from_iterable() method, of objects, 270
from_param() method, of ctypes type objects, 616
fromfd() function, socket module, 473
fromfile() method, of array objects, 260
fromhex() method, of floating point, 39
fromkeys() method, of dicts, 45
fromlist() method, of array objects, 260
fromordinal() method
of date class, 336
of datetime class, 339
fromstring() function,
xml.etree.ElementTree module, 575
fromstring() method, of array objects, 260
fromtimestamp() method
of date class, 336
of datetime class, 339
fromutc() method, of tzinfo objects, 342
frozenset type, 38, 46, 75
frozenset() function, 77, 205
fstat() function, os module, 383
fstatvfs() function, os module, 383
fsum() function, math module, 251
fsync() function, os module, 383
FTP server, uploading files to, 500
FTP() function, ftplib module, 497
FTPHandler class, urllib.request module, 518
ftplib module, 497
ftruncate() function, os module, 383
Full exception, Queue module, 418, 445
full() method, of Queue objects, 418, 445
func attribute, of partial objects, 268
__func__ attribute, of methods, 49
funcName attribute, of Record objects, 359
function call operator (), 47, 76
functions, 18
  __doc__ attribute, 24
  and coroutines, 20
  and generators, 19
  annotations in Python 3, 624
  anonymous, 112
  as closures, 98
  as dictionary values, 37
  as objects, 98
  attributes and decorators, 102
  attributes of, 48
  binding of default values, 93
  built-in, 201
  callback, 98
  calling, 18, 93
  change in func_* attribute names, 48
  changing recursion limit, 112, 235
  copying attributes to decorator, 269
  creating wrappers for, 100
  decorators, 101
  decorators and attributes, 114
  default arguments, 18, 93
  defining, 93
  delayed execution with threads, 437
  documentation strings, 48, 113
  evaluation of arguments, 76
  example of taking any number of arguments, 95
  free variables, 98
  keyword arguments, 18, 94
  lambda operator, 112
  modifying global variables, 18
  mutable parameters, 95
  nested, 97, 99
  optional arguments and None, 38
  parameter passing, 95
  partial evaluation, 76, 268
  pickling of, 228
  recursion, 112
  returning multiple values from, 18, 96
  running in the debugger, 186
  scoping rules, 18, 96
  side effects, 95
  termination functions, 219
  type of built-in, 49
  type of user-defined, 47
  user-defined, 48
  user-defined attributes, 114
  variable arguments, 94
  variable number of keyword arguments, 95

FunctionType type, 47, 237
functools module, 76, 114, 268
functor, 62
funny_files attribute, of dircmp objects, 315
future features, enabling, 178
__future__ module, 178
  and division, 62
  list of features, 178
future_builtins module, 217
FutureWarning warning, 216, 238
fuzzy date and time parsing, 343

G

gaierror exception, socket module, 485
gammavariate() function, random module, 255
garbage collection, 34-35, 220
  and __del__() method, 129
  and cycles, 35
  and program termination, 179
  description of process, 221
  observer pattern example, 130
  problem with __del__() method, 221
garbage variable, gc module, 220
gauss() function, random module, 255
gc module, 35, 179, 220
gcd() function, fractions module, 251
gdbm module, 310
ge() function, operator module, 274
  __ge__() method, 56
generator expressions, 109-110
conditional expressions, 79
converting into a list, 110
difference from list comprehension, 110
generator function, and context managers, 90
generator objects, 51, 53
attributes of, 53
GeneratorExit exception, 87, 104, 213
generators, 19, 102-103
and break statement in iteration, 103
and memory efficiency, 107
and processing pipelines, 19
closing, 53
concurrent programming, 446
execution model, 19
handling of GeneratorExit exception, 213
multitasking example, 447
practical use of, 106
recursion, 112
throwing exception in, 53
use with I/O, 164-165
use with WSGI, 165
GeneratorType type, 51, 237
get() function, webbrowser module, 544
get() method
of AsyncResult objects, 425
of ConfigParser objects, 333
of Element objects, 576
of Message objects, 552
of Queue objects, 418, 445
of dicts, 16, 45
__get__() method, of descriptors, 58, 126
get_all() method, of Message objects, 552
get_boundary() method, of Message objects, 552
get_charset() method, of Message objects, 553
get_charsets() method, of Message objects, 553
get_content_charset() method, of Message objects, 553
get_content_subtype() method, of Message objects, 553
get_content_type() method, of Message objects, 553
get_count() function, gc module, 221
get_data() method, of Request objects, 517
get_debug() function, gc module, 221
get_default_type() method, of Message objects, 553
get_dialect() function, csv module, 551
get_errno() function, ctypes module, 617
get_field() method, of Formatter objects, 288
get_filename() method, of Message objects, 553
get_full_url() method, of Request objects, 517
get_host() method, of Request objects, 517
get_last_error() function, ctypes module, 617
get_logger() function, multiprocessing module, 435
get_method() method, of Request objects, 517
get_nowait() method, of Queue objects, 418, 445
get_objects() function, gc module, 221
get_origin_req_host() method, of Request objects, 517
get_osfhandle() function, msvcrt module, 372
get_param() method, of Message objects, 553
get_params() method, of Message objects, 553
get_payload() method, of Message objects, 554
get_referents() function, gc module, 221
get_referrers() function, gc module, 221
get_selector() method, of Request objects, 517
get_server_certificate() function, ssl module, 488
get_starttag_text() method, of HTMLParser objects, 561
get_terminator() method, of asynchat objects, 453
get_threshold() function, gc module, 221
get_type() method, of Request objects, 517
get_unixfrom() method, of Message objects, 554
get_value() method, of Formatter objects, 289
getaddrinfo() function, socket module, 473
getargvalues() function, inspect module, 222
getatime() function, os.path module, 397
getattribute() function, 205
and private attributes, 128
__getattribute__() method, 57
and __slots__, 133
getAttribute() method, of DOM Element objects, 572
__getattribute__() method, 57-58, 132
and __slots__, 133
getAttributeNS() method, of DOM Element objects, 572
getboolean() method, of ConfigParser objects, 333
getch() function, msvcrt module, 372
getche() function, msvcrt module, 372
getcheckinterval() function, sys module, 234
getchildren() method, of Element objects, 577
getclasstree() function, inspect module, 222
getcode() method, of urlopen objects, 515
getcomments() function, inspect module, 223
getcontext() function, decimal module, 247
getctime() function, os.path module, 397
getcwd() function, os module, 379
getcwd() function, os module, 379
getdefaultencoding() function, sys module, 166, 234
getdefaulftimeout() function, socket module, 474
getdlopenflags() function, sys module, 234
getdoc() function, inspect module, 223
getEffectiveLevel() method, of Logger objects, 360
getegid() function, os module, 379
getElementsByTagName() method of DOM Document objects, 571
of DOM Element objects, 572
getElementsByTagNameNS() method of DOM Document objects, 571
of DOM Element objects, 572
geteuid() function, os module, 380
getfile() function, inspect module, 223
getfilesystemencoding() function, sys module, 234
getfirst() method, of FieldStorage objects, 535
getfloat() method, of ConfigParser objects, 333
getfqdn() function, socket module, 474
_getframe() function, sys module, 234
getframeinfo() function, inspect module, 223
getgid() function, os module, 380
getgroups() function, os module, 380
getheader() method, of HTTPResponse objects, 504
getheaders() method, of HTTPResponse objects, 504
gethostbyaddr() function, socket module, 474
gethostname() function, socket module, 474
gethostbyname() function, socket module, 474
gethostbyname_ex() function, socket module, 474
gethostbyname() function, socket module, 474
gethostbyname_ex() function, socket module, 474
gethostname() function, socket module, 474
getinfo() method, of ZipFile objects, 326
getinnerframes() function, inspect module, 223

How can we make this index more useful? Email us at indexes@samspublishing.com
getint() method, of ConfigParser objects, 333
getitem() function, operator module, 274
__getitem__() method, 58-59
    and slices, 59
getiterator() method
    of Element objects, 577
    of ElementTree objects, 574
getitimer() function, signal module, 399
getLength() method, of SAX attributes objects, 582
getLevelName() function, logging module, 366
getlist() method, of FieldStorage objects, 535
getloadavg() function, os module, 395
getLogger() function, logging module, 356
getlogin() function, os module, 380
getmember() method, of TarFile objects, 320
getmembers() function, inspect module, 223
getmembers() method, of TarFile objects, 321
getmodule() function, inspect module, 224
getmoduleinfo() function, inspect module, 223
getmodulename() function, inspect module, 224
getmro() function, inspect module, 224
getmtime() function, os.path module, 397
getname() method, of Thread objects, 436
getNameByQName() method, of SAX attributes objects, 582
getnameinfo() function, socket module, 474
getName() method, of SAX attributes objects, 582
getNames() method, of SAX attributes objects, 582
getrandbits() function, random module, 254
getrecursionlimit() function, sys module, 112, 234
getrefcount() function, sys module, 35, 234
getresponse() method, of HTTPConnection objects, 503
getroot() method, of ElementTree objects, 574
getservbyname() function, socket module, 475
getservbyport() function, socket module, 475
GetSetDescriptorType type, 237
getsock() function, os module, 380
getservbyname() function, signal module, 399
getsize() function, os.path module, 397
getsizeof() function, sys module, 192, 234
getslice() function, operator module, 274
getsockname() method, of socket objects, 478
getsockopt() method, of socket objects, 478
getstate() function, commands module, 331
getpeername() method, of socket objects, 478
getpeername() method, of socket objects, 478
getpgid() function, os module, 380
getpgrp() function, os module, 380
getpos() method, of HTMLParser objects, 561
getppid() function, os module, 380
getprofile() function, sys module, 234
getprotobyname() function, socket module, 475
getQNameByQName() method, of SAX attributes objects, 582
getQNames() method, of SAX attributes objects, 582
getrandbits() function, random module, 254
getrecursionlimit() function, sys module, 112, 234
getrefcount() function, sys module, 35, 234
getresponse() method, of HTTPConnection objects, 503
getroot() method, of ElementTree objects, 574
getservbyname() function, socket module, 475
getservbyport() function, socket module, 475
GetSetDescriptorType type, 237
getsock() function, os module, 380
getservbyname() function, signal module, 399
getsize() function, os.path module, 397
getsizeof() function, sys module, 192, 234
getslice() function, operator module, 274
getsockname() method, of socket objects, 478
getsockopt() method, of socket objects, 478
getstate() function, commands module, 331
__getstate__() method, 228
   and copying, 220
   and pickle module, 172
getstatusoutput() function, commands module, 332
gettarinfo() method, of TarFile objects, 321
gettempdir() function, tempfile module, 323
gettempprefix() function, tempfile module, 323
ggettext module, 587
gettimeout() method, of socket objects, 482
going help, help() function, 24
getting the current working directory, 379
gettrace() function, sys module, 234
getType() method, of SAX attributes objects, 582
getuid() function, os module, 380
geturl() method, of urlopen objects, 515
_getvalue() method, of BaseProxy objects, 433
getValue() method, of SAX attributes objects, 582
groups() method, of MatchObject objects, 285
groupby() function, itertools module, 271
groupdict() method, of MatchObject objects, 286
groupindex attribute, of Regex objects, 284
groups() method, of MatchObject objects, 285
group() method, of MatchObject objects, 285
gmtime() function, time module, 406
gname attribute, of TarInfo objects, 321
goto statement, lack of, 84
gt() function, operator module, 274
__gt__() method, 56
guess_all_extensions() function, mimetypes module, 567
guess_extension() function, mimetypes module, 567
guess_type() function, mimetypes module, 567
GUI programming, use of partial function evaluation, 268
GUIs, and network programming, 669
__globals__ attribute, of functions, 48, 99
__globals__ attribute, of functions, 48, 99
GUIs, and network programming, 669

How can we make this index more useful? Email us at indexes@samspublishing.com
gzip module, 317
GzipFile() function, gzip module, 317

-h command line option, 173
h(elp) debugger command, pdb module, 188
handle() function, cgitb module, 539
handle() method, of BaseRequestHandler objects, 490
handle_accept() method, of dispatcher objects, 455
handle_charref() method, of HTMLParser objects, 561
handle_close() method, of dispatcher objects, 455
handle_comment() method, of HTMLParser objects, 561
handle_connect() method, of dispatcher objects, 455
handle_data() method, of HTMLParser objects, 561
handle_decl() method, of HTMLParser objects, 561
handle_endtag() method, of HTMLParser objects, 562
handle_entityref() method, of HTMLParser objects, 562
handle_error() method
  of SocketServer class, 493
  of dispatcher objects, 455
handle_expt() method, of dispatcher objects, 456
handle_pi() method, of HTMLParser objects, 562
handle_read() method, of dispatcher objects, 456
handle_startendtag() method, of HTMLParser objects, 562
handle_starttag() method, of HTMLParser objects, 562
handle_timeout() method, of SocketServer class, 493
handle_write() method, of dispatcher objects, 456
has_data() method, of Request objects, 517
has_header() method
  of Request objects, 517
  of Sniffer objects, 550
has_ipv6 variable, socket module, 475
has_key() method, of dicts, 45
has_option() method, of ConfigParser objects, 333
has_section() method, of ConfigParser objects, 333
hasattr() function, 205
  and private attributes, 128
hasAttribute() method, of DOM Element objects, 572
hasAttributeNS() method, of DOM Element objects, 572
hasAttributes() method, of DOM Node objects, 571
hasChildNodes() method, of DOM Node objects, 571
hash table, 16, 44
hash table based databases, 310
hash() function, 205
__hash__() method, 56
Hashable abstract base class, 265
hashlib module, 559
  example of, 425
header_offset attribute, of ZipInfo objects, 327
headers attribute
  of BaseHTTPRequestHandler objects, 509
  of FieldStorage objects, 535
heap, 269
heapify() function, heapq module, 269
heapmin() function, msvcrt module, 372
heappop() function, heapq module, 269
heappush() function, heapq module, 269
heappushpop() function, heapq module, 270
heapq module, 269
heapreplace() function, heapq module, 270
hello world program, 5
help() function, 24, 206
    bizarre output with decorators, 113
herror exception, socket module, 485
hex() function, 77, 206
    future_builtins module, 217
hex() method, of floating point, 39
hexadecimal
    creating strings from integers, 77
    integer literals, 27
hexdigest() method
    of digest objects, 559
    of hmac objects, 560
hexdigits variable, string module, 287
hexversion variable, sys module, 231
hiding attribute names from dir() function, 128
hierarchical locking, 442
hierarchical logging, logging module, 360
hierarchy of exceptions, 87-88
hierarchy of objects, 137
HIGHEST_PROTOCOL constant, pickle module, 172
HKEY_* constants, winreg module, 408
HMAC authentication, 559
hmac module, 559
hostname attribute
    of urlparse objects, 520
    of urlsplit objects, 521
hostname, obtaining for host machine, 474
hour attribute, of time objects, 338
HTML forms
    example of, 531
    uploading with urllib package, 515
HTML parsing, 561
html.parser module, 561
HTMLParser class, html.parser module, 561
HTMLParser module, see html.parser, 561
HTMLParserError exception, html.parser module, 562
htonl() function, socket module, 475
htons() function, socket module, 475
HTTP cookies, 511
HTTP protocol
    description of, 500
    request methods, 501
    response codes, 501
HTTP server
    custom handling of requests, 510
    example with asynchat module, 453
    example with asyncio module, 457
    example with coroutines, 466
    example with firewall, 507
    standalone example, 508
    uploading files in POST request, 505
http package, 500
http.client module, 502
http.cookiejar module, 513
http.cookies module, 511
http.server module, 506
HTTPBasicAuthHandler class, urllib.request module, 518-519
HTTPConnection() function, http.client module, 502
HTTPCookieProcessor class, urllib.request module, 518-519
HTTPDefaultErrorHandler class, urllib.request module, 518
HTTPDigestAuthHandler class, urllib.request module, 518-519
HTTPError exception, urllib.error module, 523
HTTPException exception, http.client module, 504
HTTPHandler class
    logging module, 362
    urllib.request module, 518
httplib module, see http.client, 502
HTTPRedirectHandler class, urllib.request module, 518
HTTPResponse objects, http.client module, 504

How can we make this index more useful? Email us at indexes@samspublishing.com
HTTPConnection() function, http.client module, 502
HTTPSConnection() function, http.client module, 502
HTTPServer class, http.server module, 506
HTTPSHandler class, urllib.request module, 518
hypot() function, math module, 251

-i command line option, 173-174
I/O buffering, and generators, 165
I/O multiplexing, 459
__iadd__() method, 61
__iand__() method, 61
IBM General Decimal Arithmetic Standard, 243
id() function, 33, 206
ident attribute, of Thread objects, 436
identifiers, 26
    and first-class data, 36
    case sensitivity, 26
    reserved words, 26
    usage of underscores, 26
    use of Unicode in Python 3, 622
identity comparison of objects, 34
identity of objects, 33
identity operator is, 78
__idiv__() method, 61
__idivmod__() method, 61
IDLE, 5-6
    and standard I/O streams, 162
IEEE 754, 243
if statement, 9, 81
    and __debug__ variable, 91
ifilter() function, itertools module, 271
ifilterfalse() function, itertools module, 271
__ifloordiv__() method, 61
iglob() function, glob module, 317
ignoreableWhitespace() method, of ContentHandler objects, 581
ignore debugger command, pdb module, 188
'ignore' error handling, Unicode encoding, 166
ignore_environment attribute, of sys.flags, 230
ignore_pattern() function, shutil module, 318
ignore_zeros attribute, of TarFile objects, 321
ignored NameError exception in __del__, 179
ignoring exceptions, 85
__ilshift__() method, 61
imag attribute
    of complex numbers, 39
    of floating point, 39
imap() function, itertools module, 272
imap() method, of Pool objects, 424
imap_unordered() method, of Pool objects, 424
imaplib module, 587
imghdr module, 588
immutability, of tuples, 14
immutable types, inheriting from, 55
immutable, definition of, 33
__imod__() method, 61
imp module, 224, 585
implicit type conversion, lack of, 62
import statement, 13, 23-24, 50, 143-144
    Python 3, 151
        absolute imports in packages, 151
        and main program, 146
        and sys.modules, 144
        and sys.path variable, 147
        as qualifier, 144
        case sensitivity, 148
        compilation of .pyc files, 148
        execution of modules, 143
        module search path, 147
        multiple modules, 144
        one-time execution of modules, 144
        packages, 150
        placement within a program, 144
        relative package imports, 150-151
        scoping rules of loaded code, 145
        types of modules, 148
ImportError exception, 87, 148, 214
importing selected symbols from a module, 145
ImproperConnectionState exception, http.client module, 504
__imul__() method, 61
in operator, 9
    and __contains__ method, 58
    and checking for substrings, 69
    on dicts, 16, 45, 74
    on sequences, 67, 69
in-place assignment operators, 75
in-place file updates, 159
in-place mathematical operators, 61
in-place modification
    of lists, 40
    of sets, 47
in_dll() method, of ctypes type objects, 616
INADDR_* constants, socket module, 478
IncompleteRead exception, http.client module, 504
IncrementalDecoder class, codecs module, 279
incrementaldecoder() method, of CodecInfo objects, 279
IncrementalEncoder class, codecs module, 278
incrementalencoder() method, of CodecInfo objects, 278
indentation, 8, 25
    and documentation strings, 30
    and line continuation character \, 9
    and tabs, 26
    preferred style, 8
    putting statements on the same line, 25
IndentationError exception, 87, 214
index() method
    of array objects, 260
    of lists, 40-41
    of strings, 41, 43
IndexError exception, 69, 87, 214
indexing operator [], 39, 67
    on lists, 12
    on sequences, 68

indexing, 0-based, 11
indexOf() function, operator module, 274
indices() method, of slices, 54
inet_pton() function, socket module, 475
inf variable, decimal module, 248
Inf
    decimal module, 244
    infinity, 213
info() method
    of Logger objects, 357
    of urlopen objects, 515
infolist() method, of ZipFile objects, 326
inheritance, 21, 119
    __mro__ attribute of classes, 121
    abstract base classes, 137
    attribute binding, 119
    calling methods in superclasses, 120
    from built-in types, 22
    from immutable types, 55
    initialization of superclasses, 120
    interaction with __slots__, 133
    internal optimization of, 233
    issubclass() function, 34
    metaclasses, 139
    method resolution order, 121
    multiple inheritance, 121
    preventing redefinition of methods, 128
    private methods, 128
    use with exceptions, 88
ini files
    configuring logging with, 368
    reading from Python, 332
init() function, mimetypes module, 567

How can we make this index more useful? Email us at indexes@samspublishing.com
__init__() method, 50, 54-55
  and exceptions, 88
  and inheritance, 120
  and instance creation, 129
  and metaclasses, 139
  and pickle, 228
defining multiple instance creation methods, 123
of classes, 22, 118
__init__.py files in packages, 149
input() function, 162, 206
  Python 3, 11
insert() method
  of Element objects, 577
  of array objects, 260
  of lists, 12, 41
insertBefore() method, of DOM Node objects, 571
inserting items into a dictionary, 16
inserting items into a list, 41
insert() function, bisect module, 261
insert-left() function, bisect module, 261
insert-right() function, bisect module, 261
inspect attribute, of sys.flags, 230
inspect module, 222
inspecting objects, with dir(), 63
install command, of setup.py files, 154
install_opener() function, urllib.request module, 518
installation of third-party packages, 154
  in user directory, 154
installing a package, 153
instance methods, 48, 118
__instancecheck__() method, 57, 136
instances, 117
  as callable, 50
  attribute assignment, 131
  attribute deletion, 131
  creation of, 55, 118, 129
  definition of, 33
  pickling of, 228
  type of, 50
instantiation of abstract base class, 137
int() function, 11, 62, 76, 206
__int__() method, 61-62
  and type coercion, 134
integer division, Python 3, 633
integers, 27
  2's complement representation, 66
  as dictionary key, 16
  automatic promotion to long, 27
  conversion to longs, 38
  creating hexadecimal strings, 77
  creating random, 254
  overflow behavior, 66
  range of, 38
  specifying as hex, octal, or binary, 27
Integral abstract base class, 253
IntegrityError exception, database API, 302
interactive attribute, of sys.flags, 230
interactive mode, 6, 175
  and blank lines, 26
  display of results, 55, 176
interactive terminal, 174
InterfaceError exception, database API, 302
internal_attr attribute, of ZipInfo objects, 327
InternalError exception, database API, 302
international characters
  and string comparison, 70
  in source code, 31
interpolation, of values in strings, 72
interpreter, 5
interpreter command line options, 173
interpreter environment variables, 174
interpreter prompts, 175
interpreter, -t and -tt options, 26
interprocess communication (IPC), 413
interrupt() method, of Connection objects, 306
intersection operator &, of sets, 15
intersection() method, of sets, 46
intersection_update() method, of sets, 47
interval timer, 399
introspection of objects, 222
inv() function, operator module, 273
InvalidURL exception, http.client module, 504
invert() function, operator module, 274
__invert__() method, 61
io module, 349
Python 3, 631
problems associated with, 354
IOBase abstract base class, 354
IOBase class, io module, 349
ioctl() function, fcntl module, 348
ioctl() method, of socket objects, 482
IError exception, 87, 214
__ior__() method, 61
IP_* socket options, socket module, 480
__ipow__() method, 61
IPPROTO_* constants, socket module, 476
IPv4 protocol, 470
address format, 471
IPv6 protocol, 470
address format, 471
IPv6_* socket options, socket module, 480-481
IronPython, 5
example of, 620
__irshift__() method, 61
is operator, object identity, 34, 78
is() function, operator module, 274
is_alive() method
of Process objects, 416
of Thread objects, 436
is_multipart() method, of Message objects, 554
is_not() function, operator module, 274
is_set() method, of Event objects, 440
isfile() function, tarfile module, 319
is_unverifiable() method, of Request objects, 517
is_zipfile() function, zipfile module, 325
isabs() function, os.path module, 397
isabstract() function, inspect module, 224
isAlive() method, of Thread objects, 436
isalnum() method, of strings, 41, 43
isalpha() method, of strings, 43
isatty() function, os module, 383
isatty() method
of IOBase objects, 349
of files, 160
isblk() method, of TarInfo objects, 321
isbuiltin() function, inspect module, 224
ischr() method, of TarInfo objects, 321
isclass() function, inspect module, 224
iscode() function, inspect module, 225
isDaemon() method, of Thread objects, 437
isdatadescr() function, inspect module, 225
isdev() method, of TarInfo objects, 321
isdigit() method, of strings, 43
isdir() function, os.path module, 397
isdir() method, of TarInfo objects, 321
isdisjoint() method, of sets, 46
iselement() function, xml.etree.ElementTree module, 578
isenabled() function, gc module, 221
isEnabledFor() method, of Logger objects, 359
isfifo() method, of TarInfo objects, 321
isfile() function, os.path module, 397
isfile() method, of TarInfo objects, 321
isframe() function, inspect module, 225
isfunction() function, inspect module, 225
isgenerator() function, inspect module, 225
isgeneratorfunction() function, inspect module, 225
isin() function, math module, 252
isinstance() function, 34, 37, 135, 206-207
and inheritance, 134
and proxy objects, 135
redefining behavior of, 136
islcase() function, itertools module, 272
islink() function, os.path module, 397
islnk() method, of TarInfo objects, 321
islower() method, of TarInfo objects, 321
ismethod() function, inspect module, 225
ismethoddescriptor() function, inspect module, 225
ismodule() function, inspect module, 225
ismount() function, os.path module, 397
isnan() function, math module, 252
iso-8859-1 encoding, description of, 169
isocalendar() method, of date objects, 337
isofmt() method
   of date objects, 337
   of time objects, 338
isoweekday() method, of date objects, 337
isroutine() function, inspect module, 225
isSameNode() method, of DOM Node objects, 571
isspace() method, of strings, 43
issubclass() function, 135, 206
   redefining behavior of, 136
issubset() method, of sets, 46
issuperset() method, of sets, 46
issym() method, of TarInfo objects, 322
istitle() method, of strings, 43
istraceback() function, inspect module, 225
   __isub__() method, 61
isupper() method, of strings, 41, 43
itemgetter() function, operator module, 275
items() method
   of ConfigParser objects, 333
   of Element objects, 577
   of Message objects, 552
   of dicts, 45
   of dicts in Python 3, 632
itemsize attribute, of array objects, 259
ItemsView abstract base class, 266
iter() function, 206
   __iter__() method, 59, 82
Iterable abstract base class, 265
iteration, 10, 17, 59, 82
   breaking out of a loop, 83
   iteration variable, 82
   over a sequence, 39, 69
   over dictionary keys, 17
   over dictionary values, 45
   over multiple sequences, 83
   portable function for next() operation, 207
protocol change in Python 3, 633
protocol of, 60, 82
scope of iteration variable, 82
   supported objects, 17
   unpacking of tuples, 82

Iterator abstract base class, 265
iterators, use in Python 3, 632

iterdecode() function, codecs module, 279
iterdump() method, of Connection objects, 306
iterencode() function, codecs module, 279
iterencode() method, of JSONEncoder objects, 566
iterkeyrefs() method, of WeakKeyDictionary objects, 241
iterparse() function, xml.etree.ElementTree module, 578
itertools module, 83, 270
itervaluerefs() method, of WeakValueDictionary objects, 241
   __itruediv__() method, 61
   __ixor__() method, 61
izip() function, itertools module, 83, 212, 272
izip_longest() function, itertools module, 272

J

J character, on complex number literals, 27
j(ump) debugger command, pdb module, 188

Java, 620
   difference in class system, 119

Javascript, pop-up window example, 531
join() function, os.path module, 397
join() method
   of JoinableQueue objects, 419
   of Pool objects, 424
of Process objects, 416
of Queue objects, 445
of Thread objects, 436
of strings, 43
join_thread() method, of Queue objects, 418
JoinableQueue() function, multiprocessing module, 419
js_output() method
of Morsel objects, 512
of SimpleCookie objects, 512
JSON (JavaScript Object Notation), 563
json module, 563
difference from pickle and marshal, 565
JSONDecoder class, json module, 565
JSONEncoder class, json module, 566
jumpahead() function, random module, 254
Jython, 5
example of, 620

keys
acceptable types for dictionaries, 16
of dicts, 44
KeysView abstract base class, 266
keyword arguments, 18, 94
mixing with positional arguments, 94
keyword module, 585
keyword-only arguments, Python 3, 625
keywords attribute, of partial objects, 268
kill() function, os module, 392
kill() method, of Popen objects, 403
killpg() function, os module, 392
kqueue, BSD, 460

L
L character, on long integers, 27
l(ist) debugger command, pdb module, 188
lambda operator, 48, 112
alternatives to, 274–275
LambdaType type, 237
last_accepted attribute, of Listener objects, 434
last_traceback variable, sys module, 231
last_type variable, sys module, 231
last_value variable, sys module, 231
lastChild attribute, of DOM Node objects, 570
lastgroup attribute, of MatchObject objects, 286
lastindex attribute, of MatchObject objects, 286
latin-1 encoding, description of, 169
launching a web browser, 544
launching python applications, 176
launching subprocesses, 402
examples, 404
lazy evaluation, 99
lchflags() function, os module, 387
lchmod() function, os module, 387
lchown() function, os module, 387

How can we make this index more useful? Email us at indexes@samspublishing.com
ldexp() function, math module, 252
le() function, operator module, 274
__le__() method, 56
leading 0b on integers, binary, 27
leading 0o on integers, octal, 27
leading 0x on integers, hexadecimal, 27
leading b character on string literals, byte strings, 29
leading r character on strings, raw strings, 29
leading u character on string literals, Unicode strings, 28
left shift operator <
left_list attribute, of dircmp objects, 315
left_only attribute, of dircmp objects, 315
legacy code, and exec statement, 115
len() function, 58, 206
  on dicts, 74
  on mappings, 44-45
  on sequences, 39-40, 67, 69
  on sets, 46, 75
__len__() method, 56, 58
  and truth testing, 56
length attribute, of HTTPResponse objects, 504
less than operator
less than or equal to operator <=, 66
letters variable, string module, 287
levelname attribute, of Record objects, 359
levelno attribute, of Record objects, 359
lexical scoping, 97
lexicographical ordering
  of UTF-8, 170
  of strings, 70
lexists() function, os.path module, 397
LifoQueue() function, queue module, 444
limit_denominator() method, of Fraction objects, 250
limiting the output of error tracebacks, 232
line continuation character \, 9, 25, 29
line continuation, and parentheses, braces, or brackets, 25
line separator character for files, 379
line structure of programs, 25
line_buffering attribute, of TextIOWrapper objects, 353
linecache module, 585
linenon attribute, of Record objects, 359
linesep variable, os module, 379
link() function, os module, 387
linkname attribute, of TarInfo objects, 322
Linux, 331
Linux link-level packet protocol, 470
  address format, 472
Linux, epoll interface, 460
list comprehensions, 13
  and declarative programming, 110
  conditional expressions, 79
  creation of tuples within, 109
  difference from generator expression, 110
  general syntax of, 108
  scope of iteration variable, 109
  similarity to SQL queries, 111
  similarity to awk command, 111
list of Unix signal names, 400
list type, 38
list() function, 12, 40, 77, 207
  applied to dictionaries, 16
list() method
  of Manager objects, 429
  of TarFile objects, 321
list_dialects() function, csv module, 551
listdir() function
  Python 3, 630, 633
  os module, 387
listen() method
  of dispatcher objects, 456
  of socket objects, 483
Listener class, multiprocessing module, 433
lists, 12, 40
  appending to, 12, 40
  as sequence, 39
  compared to array objects, 260
  comparison of, 70
  concatenation, 12
counting items, 40
deletion of items, 69
empty, 12
equality of, 78
indexing operator, 12
inefficiency of insert(), 194
inserting items, 12, 40, 69
item assignment, 12, 69
keeping in sorted order, 261
list comprehension, 108
making shallow copy of, 40
nested, 13
random shuffling, 254
reassigning a slice, 12
removing items, 40
reversing, 40
searching, 40
shared by multiple processes, 429
slice assignment, 69
slice deletion, 69
slices, 12
sorting, 40
versus deque, 194, 262
versus tuples, 14
little endian format, 167
little endian, packing and unpacking, 292
ljust() method, of strings, 43
ln() method, of Decimal objects, 243
load() function
json module, 565
marshal module, 226
pickle module, 171, 227
load() method
of SimpleCookie objects, 512
of Unpickler objects, 228
loads() function
json module, 565
marshal module, 226
pickle module, 227
xmlrpc.client module, 526
local storage for threads, 443
local variables, 96
and eval(), 115
storage of in stack frames, 52
use before defined, 98
local() function, threading module, 443
localcontext() function, decimal module, 248
locale module, 587
locale setting, and string comparison, 70
localName attribute, of DOM Node objects, 570
locals() function, 207
localtime() function, time module, 406
Lock object
multiprocessing module, 427
threading module, 438
Lock() method, of Manager objects, 429
LOCK_* constants, flock() function, 348
lockf() function, fcntl module, 348
locking() function, msvcrt module, 373
locking
avoiding deadlock, 442
files on Windows, 373
multiprocessing module, 427
of critical sections, 414
of files, 348
threading module, 439
locks
and context managers, 89
and exceptions, 23
proper management of, 442
log files, real-time monitoring example, 19
log() function, math module, 252
log() method, of Logger objects, 358
log10() function, math module, 252
log10() method, of Decimal objects, 243
log1p() function, math module, 252
log_error() method, of
BaseHTTPRequestHandler objects, 510
log_message() method, of
BaseHTTPRequestHandler objects, 510

How can we make this index more useful? Email us at indexes@samspublishing.com
log_request() method, of BaseHTTPRequestHandler objects, 510
LogAdapter() function, logging module, 366
logging module, 355
  adding extra fields to log messages, 365
  and multiprocessing module, 435
  basic configuration, 355
  configuring with .ini files, 368
  filtering messages, 359
  formatting of messages, 365
  handler objects, 362
  how to configure, 367
  including exceptions in log messages, 358
  issuing log messages, 356
  logger hierarchy, 360
  message handling, 361
  message propagation, 360
  picking logger names, 356
  using a null logger, 369
login() method
  of FTP objects, 498
  of SMTP objects, 514
lognormvariate() function, random module, 255
long integers, 27
  and integers, 38
  automatic promotion from integers, 27
long type, 38
long() function, 207
__long__() method, 61-62
lookup table, and dictionaries, 16
lookup() function
  codecs module, 277
  unicodedata module, 296
LookupError exception, 87, 212
loop() function, asyncore module, 457
looping, 17, 82
  breaking out prematurely, 83
  keeping a loop counter, 83
  while statement, 8
loose-coupling of objects, 122
low-level file manipulation, 382
lower() method, of strings, 43
lowercase variable, string module, 287
lseek() function, os module, 383
lshift() function, operator module, 274
__lshift__() method, 60
lstat() function, os module, 387
lstrip() method, of strings, 43
lt() function, operator module, 274
__lt__() method, 56
LWPCookieJar class, http.cookiejar module, 513

M

-m command line option, 173-174
-m pdb option to interpreter, 189
mailbox module, 587
mailcap module, 587
main program execution, 146
main program, and pickle module, 228
main thread, 413
__main__ module, 146, 174
main() function, unittest module, 184
__main__, check needed for multiprocessing module, 417
major() function, os module, 387
make_server() function,
  wsgiref.simple_server module, 542
makedev() function, os module, 387
makedirs() function, os module, 387
makefile() method, of socket objects, 483
maketrans() function, string module, 290
making timing measurements, 191
managed objects, multiprocessing module, 428
Manager() function, multiprocessing module, 428
mantDig attribute, of sys.float_info, 231
map() function, 207
  and Python 3, 207
  and optimization, 197
  future_builtins module, 217
map() method, of Pool objects, 424
map-reduce, multiprocessing module, 424
map_async() method, of Pool objects, 425
Mapping abstract base class, 266
mappings, 44
deletion of items, 45
key index operator, 44
special methods of, 58
MappingView abstract base class, 266
marshal module, 226
match() function, re module, 284
match() method, of Regex objects, 285
MatchObject objects, re module, 285
math module, 251
mathematical operators
in-place, 61
mixed types, 66-67
mathematical special methods, 60
max attribute
of date class, 337
of datetime class, 340
of sys.float_info, 231
of time class, 338
of timedelta class, 341
max() function, 13, 39-40, 67, 69, 207
on sets, 75
required methods for user-defined objects, 57
max_10_exp attribute, of sys.float_info, 231
max_exp attribute, of sys.float_info, 231
maxint variable, sys module, 231
maxsize variable, sys module, 231
maxunicode variable, sys module, 231
md5() function, hashlib module, 559
MemberDescriptorType type, 237
membership test
of dicts, 16, 74
of sequences, 67
memmove() function, ctypes module, 617
memoization of results, 242
memory efficiency
and __slots__, 133
of generator expressions, 110
of generators, 107
memory management, 128
checking for leaks, 221
creation of instances, 129
garbage collection, 35, 220
reference counting, 129
memory mapped files, 370
and IPC, 413
memory use
array objects, 260
measuring, 192
obtaining size of objects, 234
tuples versus lists, 14
memory, location of objects, 33
MemoryError exception, 87, 214
MemoryHandler class, logging module, 362
memset() function, ctypes module, 617
merge() function, heapq module, 270
Mersenne Twister, 254
Message class, email package, 552, 555
message attribute, of Exception objects, 213
message digests, 559
message passing, 414-415
and coroutines, 415
and synchronization, 415
coroutines, 108
definition of, 413
sending byte buffers, 421
sending objects between processes, 421
message propagation, of log messages, 360
message queues, 415
coroutines, 108
multiprocessing module, 418
message_from_file() function, email package, 552
message_from_string() function, email package, 552

How can we make this index more useful? Email us at indexes@samspublishing.com
metaclass keyword argument, of class definitions, 139
__metaclass__ attribute, of classes, 139
__metaclass__ global variable, 139
metaclasses, 138
    __prepare__(cls) method, 627–628
    and descriptors, 140
    and inheritance, 139
    caution on use, 141
    example of, 140
    how to define, 139
    performance benefits, 197
    use of __new__(cls) method, 55, 129
    use of a custom dictionary object, 628
method resolution order, and TypeError exception, 122
method resolution
    __mro__ attribute, 121
    multiple inheritance, 121
    single inheritance, 120
methodcaller() function, operator module, 275
methodHelp() method, of ServerProxy objects, 525
methods, 48, 117
    bound, 49, 125
    calling process, 48
    class, 125
    @classmethod decorator, 48
    defining in classes, 21
    definition of, 118
    handling as properties, 125
    preventing redefinition in subclasses, 128
    static, 125
    @staticmethod decorator, 48
    type of, 47
    type of built-in, 49
    unbound, 49
    use of super() function, 120
methodSignatures() method, of ServerProxy objects, 525
MethodType type, 47–48, 237
microsecond attribute, of time objects, 338
microthreading, 446
migrating code
    Python 2 to 3, 634
    practical strategy, 637
MIMEApplication class, email package, 557
MIMEAudio class, email package, 557
MIMEImage class, email package, 557
MIMEMessage class, email package, 557
MIMEMultipart class, email package, 557
MIMEText class, email package, 558
mimetypes module, 566
min attribute
    of date class, 337
    of datetime class, 340
    of sys.float_info, 231
    of time class, 338
    of timedelta class, 341
min() function, 13, 39–40, 67, 69, 207
    on sets, 75
    required methods for user-defined objects, 57
min_10_exp attribute, of sys.float_info, 231
min_exp attribute, of sys.float_info, 231
minimum requirements for supporting equality, 57
minor() function, os module, 387
minute attribute, of time objects, 338
mirrored() function, unicodedata module, 296
missing parentheses, and tuples, 14
mixed-type mathematical operations, 66–67
mixin classes, 122
mixing byte strings and Unicode, 167
mkd() method, of FTP objects, 498
mkdir() function, os module, 388
mkdtemp() function, tempfile module, 323
mkfifo() function, os module, 388
mknod() function, os module, 388
mkstemp() function, tempfile module, 323
mktemp() function, tempfile module, 323
mktime() function, time module, 406
mmap module, 369
mmap() function, mmap module, 370
mod() function, operator module, 273
__mod__() method, 60
mode attribute
  of FileIO objects, 350
  of TarInfo objects, 322
  of files, 161
modf() function, math module, 252
modifying global variables from a function, 18
modifying the module search path, 147
module attribute, of Record objects, 359
module loading, 147
module reloading, 149
module search path
  and site module, 177
  and zip files, 147
  modifying, 147
  setting with environment variable, 174
module unloading, 149
__module__ attribute, of classes, 121
modulefinder module, 585
modules variable, sys module, 149, 231
modules, 23, 143
  accessing classes, 144
  and .pyc files, 148
  as objects, 144
  attribute access, 51
  attributes of, 51
  dynamic loading, 144
  global namespace for functions, 96
  importing multiple, 144
  one-time execution, 144
  search path of, 147
  self-testing with doctest, 182
  type of, 50
  type of module object, 47
  types of recognized files, 148
  using to write extensible programs, 144
ModuleType type, 47, 237
modulo operator %, 65
month attribute, of date objects, 337
Morsel class, http.cookies module, 512
move() function, shutil module, 319
move() method, of mmap objects, 371
moving the file pointer, 161
MozillaCookieJar class, http.cookiejar module, 513
__mro__ attribute, of classes, 121
MSG_* constants, socket module, 483
msvcrt module, 372
mtime attribute, of TarInfo objects, 322
mul() function, operator module, 273
__mul__() method, 60
multi-dimensional lists, 13
MultiCall() function, xmlrpc.client module, 526
multicore, and program execution, 414
multiple inheritance, 120-121
multiple statements on the same line, 26
multiplexing, of I/O, 459
multiplication operator *, 65
multiprocessing module, 415
  and global interpreter lock, 444
  and pickle, 435
  connecting separate processes, 433
  distributed computing, 435
  logging, 435
  managed objects, 428
  passing a list through shared memory, 427
  pipes, 421
  process pools, 424
  queues, 418
  shared memory, 426
  synchronization primitives, 427
  use of __main__ check, 417
multithreaded chicken, 414
mutability
  default function arguments, 94
  dictionary keys, 44
  function parameters, 95
  in-place assignment operators, 75
  reference counting, 35
mutable, definition of, 33
MutableMapping abstract base class, 266
MutableSequence abstract base class, 266
MutableSet abstract base class, 266
mutual exclusion lock, 438
MySQL, accessing from Python, 297

\N escape code, in strings, 28
n(ext) debugger command, pdb module, 189
name attribute
  of FieldStorage objects, 535
  of FileIO objects, 350
  of Process objects, 417
  of Record objects, 359
  of TarInfo objects, 322
  of Thread objects, 436
  of files, 161
name mangling, of private attributes, 127
name variable, os module, 379
__name__ attribute
  of built-in functions, 49
  of functions, 48
  of methods, 49
  of modules, 51
  of types, 50
__name__ variable, of modules, 146
name() function, unicodedata module, 296
named tuples
  use as tuples, 264
  use by standard library, 265
NamedTemporaryFile() function, tempfile module, 324
namedtuple() function, collections module, 264
NameError exception, 87, 214
NameError exception in __del__ ignored, 179
NameError exception, and variable lookup, 96
namelist() method, of ZipFile objects, 326
Namespace() method, of Manager objects, 429
namespace
  and classes, 117
  and import statement, 24, 143
  local variables of function, 96
namespaceURI attribute, of DOM Node objects, 570
NaN variable, decimal module, 248
NaN
  not a number, 213
  not a number, decimal module, 244
ne() function, operator module, 274
__ne__() method, 57
neg() function, operator module, 273
__neg__() method, 61
negative indices, 68-69
negInf variable, decimal module, 248
nested classes, problem with pickle, 228
nested functions, 97
  and closures, 99
nested lists, 13
nested() function, contextlib module, 267
Netlink protocol, 470
  address format, 472
netloc attribute
  of urlparse objects, 520
  of urlsplit objects, 521
netrc module, 587
network programming modules, Python 3 reorganization, 497
network programming
  Unicode encoding, 452
  asynchronous, 467
  event-driven programming, 455
  getting hostname, 474
  introduction, 449
  performance of polling, 468
new() function
  hashlib module, 559
  hmac module, 559
__new__() method, 54-55
  and instance creation, 129
  and metaclasses, 139
caution when reading code, 129
use by immutable types, 129
uses of, 55
newline character, difference on
Unix/Windows, 159
newline escape code in strings, 28
newline parameter, to open() function, 159
newline suppression, print statement, 162
newline termination of statements, 7
newlines attribute
of TextIOWrapper objects, 353
of files, 161
next() function, 207
next() method, 59
of TarFile objects, 321
of files, 160
of generators, 19, 53, 103
of iterators, 82
use with coroutines, 104
__next__() method, 59
Python 3, 633
of generators, 19, 103
of iterators, 82
nextset() method, of Cursor objects, 299
nextSibling attribute, of DOM Node objects,
570
NL_* constants, socket module, 475
nice() function, os module, 392
nis module, 586
nlargest() function, heapq module, 270
ntplib module, 587
no_site attribute, of sys.flags, 230
nodeName attribute, of DOM Node objects,
570
nodeType attribute, of DOM Node objects,
570
nodeValue attribute, of DOM Node objects,
570
non-printing characters, specifying in string
literals, 28

None, 38
and default arguments, 94
return statement in functions, 96
nonlocal statement, Python 3, 97, 624
normalization of Unicode strings, 171
normalize() function, unicodedata module,
171, 296
normalize() method, of DOM Node objects,
571
normalvariate() function, random module,
255
normcase() function, os.path module, 398
normpath() function, os.path module, 398
not equal to operator !=, 66
not operator, boolean expressions, 9, 77
not_() function, operator module, 274
NotConnected exception, http.client mod-
ule, 504
notify() method, of Condition objects, 441
notify_all() method, of Condition objects,
441
NotImplementedError exception, 87, 214
NotSupportedError exception, database API,
302
now() method, of datetime class, 339
nsmallest() function, heapq module, 270
NTEventLogHandler class, logging module,
362
ntohl() function, socket module, 476
ntohs() function, socket module, 476
ntransfercmd() method, of FTP objects, 498
null object, 369
null values, 38
NULL-terminated strings, and UTF-8, 170
Number abstract base class, 253
number of CPUs on system, 434
numbers module, 138, 252
numbers, example of defining new type,
133
numerator attribute
of Fraction objects, 250
of integers, 39
numeric data, and strings, 11
numeric literals, 26-27
numeric type coercion, 66-67
numeric type hierarchy, 137, 253
numeric types, 38
numeric() function, unicodedata module, 296
numpy extension, 39, 261

-O command line option, 91, 148, 173-174, 369
object, 47
object base class, 21, 119
object() function, 208
objects, 21
attributes of, 33
class of, 34
comparison, 34
comparison in Python 3, 633
container or collection, 33
defining a null object, 369
definition of, 33
first-class status, 36
getting a list of referrers, 221
getting the size of, 192
hierarchy of, 137
how to copy, 36
identity of, 33
inspecting with dir(), 63
instance of, 33
introspection of, 222
methods for comparison, 56
name of, 35
obtaining size of, 234
persistence, 171
proxies in multiprocessing module, 431
reference counting of, 34
representation of, 131
requirements for ordering, 57
sending between processes with pipes, 421
sending between processes with queues, 418
serializing with marshal, 226
serializing with pickle, 227
sharing in the interpreter, 35
supporting iteration, 82
type of, 33
weak references to, 240
observer pattern, 130, 240
oct() function, 77, 208
future_bultins module, 217
octal integer literals, 27
octdigits variable, string module, 287
old-style classes, 139
-00 command line option, 148, 173-174
open() function, 10, 158, 208
Python 3, 159
codecs module, 167, 279
codecs module and Python 3, 279
dbm module, 310
description of file modes, 159
difference between Python 2 and 3, 208
gzip module, 317
io module, 354
os module, 384
shelve module, 171, 311
tarfile module, 319
webbrowser module, 544
open() method
of ZipFile objects, 326
of controller objects, 544
open_new() function, webbrowser module, 544
open_new() method, of controller objects, 544
open_new_tab() function, webbrowser module, 544
open_osfhandle() function, msvcrt module, 373
OpenKey() function, winreg module, 410
OpenKeyEx() function, winreg module, 410
openpty() function, os module, 385
OpenSSL, 486
example of creating certificates, 489
operating system, scheduling by, 414
OperationalError exception, database API, 302
operator module, 273
  alternative to lambda, 274
  use in optimization, 274
operator overloading, 54
  example of, 133
  order of operands, 134
  reversed operands, 60
  type coercion, 134
operators, 30, 65
  mathematical, 60
  precedence of, 78
optimization
  __slots__ attribute of classes, 132, 196
  array objects, 260
  attribute binding, 195-196
  built-in types, 194
  creation of instances, 195
  decorators and metaclasses, 197
  definition of speedup, 192
  deque objects, 263
  dict() function, 195
  dict lookups, 197
  dicts versus classes, 195
  disassembly, 193
  effect of adding layers, 195
  exceptions, 196-197
  formatting of log messages, 358
  functional programming, 197
  impact of I/O polling, 469
  internal type cache, 233
  lists versus array objects, 260
  logging module, 369
  making timing measurements, 191
  map() and filter() functions, 197
  marshal versus pickle, 226
  measuring memory use, 192
  repeated timing measurements, 192
  select() function, 467
  sorting callback functions, 275
  speedup, 194
tuning strategies, 194
use of io module, 354
use of multiprocessing pools, 426
use of operator module, 274
user defined classes, 195
optimize attribute, of sys.flags, 230
optimized mode, enabling with an environment variable, 174
optional function arguments, 18, 93
  and None, 38
OptionParser() function, optparse module, 374
options() method, of ConfigParser objects, 333
optionxform() method, of ConfigParser objects, 333
optparse module, 374
  example, 157
or operator, boolean expressions, 9, 77
or_() function, operator module, 274
__or__() method, 60
ord() function, 77, 208
order of evaluation, 78
  attempts to modify, 79
order of operands, operator overloading, 134
organizing code for distribution, 152
OS X, 331
os module, 158, 378
os.environ variable, 158
os.path module, 396
OSError exception, 87, 214
ossaudiodev module, 588
output() method
  of Morsel objects, 512
  of SimpleCookie objects, 512
OutputString() method, of Morsel objects, 513
overflow, lack of with integers, 66
OverflowError exception, 214

How can we make this index more useful? Email us at indexes@samspublishing.com
p debugger command, pdb module, 189
P_* constants, spawnv() function, 392
pack() function, struct module, 290
pack() method, of Struct objects, 291
pack_into() function, struct module, 290
pack_into() method, of Struct objects, 291
packages, 149
   relative import, 150-151
PACKET_* constants, socket module, 472
packing
   binary data structures, 290
   of tuples, 14
pairs, creating a list of from dictionary, 45
parallel iteration over sequences, 83
parameter passing to functions, 95
params attribute, of urlparse objects, 520
paramstyle variable, database API, 300
pardir variable, os module, 386
parent class, 119
parentNode attribute, of DOM Node objects, 570
paretovariate() function, random module, 256
parse() function
   xml.dom.minidom module, 570
   xml.etree.ElementTree module, 578
   xml.sax module, 580
parse() method
   of ElementTree objects, 574
   of Formattter objects, 288
parse_args() method, of OptionParser objects, 158, 376
parse_header() function, cgi module, 536
parse_multipart() function, cgi module, 536
parse_qs() function, urllib.parse module, 521
parse_qsl() function, urllib.parse module, 522
parser module, 586
parseString() function
   xml.dom.minidom module, 570
   xml.sax module, 580

parsing
   CSV files, 548
   HTML, 561
   URLs, 520
   XML, 568
      command line options, 157, 374
      email messages, 552
      form fields in CGI scripts, 534
      large XML documents with
         ElementTree, 579
         robots.txt file, 523
partial() function
   functools module, 76, 268
      use with network handlers, 510
partition() method, of strings, 41, 43
pass statement, 9, 25, 82
password attribute
   of urlparse objects, 520
   of urlsplit objects, 521
path attribute
   of BaseHTTPRequestHandler objects, 509
   of urlparse objects, 520
   of urlsplit objects, 521
path variable
   os module, 379
   sys module, 147, 177, 232
   __path__ attribute, of modules, 51
   __path__ variable, in packages, 151
pathconf() function, os module, 388
pathname attribute, of Record objects, 359
pathsep variable, os module, 386
pattern attribute, of Regex objects, 284
pattern syntax, regular expressions, 281
pause() function, signal module, 399
pdb module, 186
   debugging programs from command
      shell, 189
      .pdbrc configuration file, 190
      .pdbrcrc configuration file, 190
peek() method, of BufferedReader objects, 351
PEM_cert_to_DER_cert() function, ssl module, 488
PEP 249, Python Database API
Specification, 297
PEP 333 (WSGI), 540
per-user site directory, 154, 177
installing packages in, 178
performance
of binary file I/O, 351-352
of generator expressions, 110
of logging module, 369
of type checking, 34
Perl
and dynamic scope, 97
interpretation of numeric strings vs. Python, 11
permutations() function, itertools module, 272
persistent dictionary, shelve module, 171
PHP, interpretation of numeric strings vs. Python, 11
pi variable, math module, 252
pickle module, 171, 226
__main__ module, 228
and multiprocessing module, 435
cPickle, 229
incompatible objects, 171
interaction with copy module, 220
protocol selection, 171-172
security concerns, 172
used by shelve, 311
pickle protocol, selecting in shelve module, 172
Pickler class, pickle module, 228
pickletools module, 586
pid attribute
of Popen objects, 404
of Process objects, 417
Pipe() function, multiprocessing module, 421
pipe() function, os module, 385
pipelines and generators, 19
pipelines, and generators, 106-107
pipes module, 586
pipes, creating with subprocess module, 403
pkgutil module, 586
placement of decorators, 101
platform module, 586
platform variable, sys module, 232
plistlib module, 587
plock() function, os module, 392
pm() function, pdb module, 186
POINTER() function, ctypes module, 614
pointer() function, ctypes module, 615
poll() function, select module, 459
poll() method
of Connection objects, 421
of Poll objects, 460
of Popen objects, 403
POLL* constants, select module, 459
polling, 459
performance of, 468-469
polymorphism, 122
Pool() function, multiprocessing module, 424
pop() method
of array objects, 260
of deque objects, 262
of dicts, 45, 95
of lists, 41
of sets, 47
Popen() function, subprocess module, 402
popen() function, os module, 392
popitem() method, of dicts, 45
popleft() method, of deque objects, 262
popenlib module, 587
port attribute
of urlparse objects, 520
of urlsplit objects, 521
port number
in network programs, 449
list of well known, 450
portability, of marshal module, 226
portable manipulation of filenames, 396

How can we make this index more useful? Email us at indexes@samspublishing.com
pos attribute, of MatchObject objects, 286
pos() function, operator module, 273
__pos__() method, 61
POSIX interface, 331
posix attribute, of TarFile objects, 321
post_mortem() function, pdb module, 186
pow() function, 66, 208
    math module, 252
__pow__() method, 60
power operator **, 65
pp debugger command, pdb module, 189
pprint module, 586
preamble attribute, of Message objects, 554
prec attribute, of Context objects, 247
precision, of floating point, 38
predicate() function, itertools module, 271
prefix attribute, of DOM Node objects, 570
—prefix option to setup.py, 154
prefix variable, sys module, 177, 232
__prepare__() method, Python 3 metaclasses, 627-628
preventing the creation of .pyc files, 230
previousSibling attribute, of DOM Node objects, 570
print statement, 6, 162
    and __str__(), 56
    and sys.stdout, 161
file redirection, 10, 163
formatted output, 8, 162
newline suppression, 162
softspace attribute of files, 162
syntax error with Python 3, 6
trailing comma, 10
print() function, 163, 209
    Python 3, 631
    enabling in Python 2.6, 163
    file redirection, 163
    newline suppression, 163
    separator character, 163
print_directory() function, cgi module, 537
print_environ() function, cgi module, 537
print_exc() function, traceback module, 236
print_exception() function, traceback module, 236
print_form() function, cgi module, 537
print_last() function, traceback module, 236
print_stack() function, traceback module, 236
print_tb() function, traceback module, 235-236
printable variable, string module, 287
printf() function equivalent, 8
printing
    creating custom formatters, 288
    dates and times, 406
    formatted, 8
priority queue, 269
PriorityQueue() function, queue module, 445
private attributes, 127
    and properties, 128
    name mangling of, 127
private class members, 26
private methods, and inheritance, 128
private specifier, lack of, 127
probability, random number distributions, 255
process attribute, of Record objects, 359
process id, getting, 380
Process() function, multiprocessing module, 416
processes
    connecting with pipes, 421
daemonic, 415
definition of, 413
joining, 415
scheduling of, 414
sending signals to, 392
terminating, 392, 403-404, 415
worker pools, 424
ProcessingInstruction() function, xml.etree.ElementTree module, 575
processingInstruction() method, of ContentHandler objects, 581
producer-consumer
  with coroutines, 20
  with pipes, 422
  with queues, 419
  with threads and condition variables, 441
  with threads and semaphores, 439
product() function, itertools module, 272
profile module, 190
profiling, 190
  interpreting output, 191
program execution model, 81
program execution, main program, 146
program structure, 81
program termination, 7, 179, 233
  and garbage collection, 179
  brutal, 179
  brute force, 391
  ignored NameError exception, 179
  registering cleanup functions, 219
programming errors, lack of compiler checking, 181
ProgrammingError exception, database API, 302
prompts
  changing, 176
  interactive mode, 175
propagate attribute, of Logger objects, 360
properties, 117
  and __setattr__() method, 131
  and private attributes, 128
  definition of, 124
  set and delete functions, 126
  uniform access principle, 125
  use by methods, 125
@property decorator, 124
property() function, 126, 209
protected specifier, lack of, 127
proto attribute, of socket objects, 484
protocol parameter, to pickle functions, 171
protocol_version attribute, of BaseHTTPRequestHandler class, 509
ProtocolError exception, xmlrpc.client module, 527
proxies, and attribute binding methods, 132
proxy, 62
proxy functions, 95
proxy objects
  and multiprocessing module, 428, 431
  problem with type checking, 135
proxy() function, weakref module, 241
ProxyBasicAuthHandler class, urllib.request module, 518-519
ProxyDigestAuthHandler class, urllib.request module, 518-519
ProxyHandler class, urllib.request module, 518-519
ProxyTypes class, weakref module, 241
ps1 variable, sys module, 232
ps2 variable, sys module, 232
.pth files, site configuration, 177
pty module, 586
punctuation variable, string module, 287
push() method, of asynchat objects, 453
push_with_producer() method, of asynchat objects, 453
put() method, of Queue objects, 418, 445
put_nowait() method, of Queue objects, 419, 445
putwch() function, msvcrt module, 373
putenv() function, os module, 380
putheader() method, of HTTPConnection objects, 503
putrequest() method, of HTTPConnection objects, 503
putwch() function, msvcrt module, 373
pwd module, 586
pwd() method, of FTP objects, 498

How can we make this index more useful? Email us at indexes@samspublishing.com
.py files, 6, 147
and library modules, 23
py2app package, 154
py2exe package, 154
py3k_warning attribute, of sys.flags, 230
py3kwarning variable, sys module, 232
Py_BEGIN_ALLOW_THREADS macro, 607
Py_BuildValue() function, 602
py_compile module, 586
Py_DECREF() macro, 607
Py_END_ALLOW_THREADS macro, 607
Py_Finalize() function, 609
Py_GetExecPrefix() function, 609
Py_GetPath() function, 609
Py_GetPrefix() function, 609
Py_GetProgramFullPath() function, 609
Py_INCREF() macro, 607
Py_Initialize() function, 609
Py_IsInitialized() function, 609
Py_SetProgramName() function, 609
Py_XDECREF() macro, 607
Py_XINCREF() macro, 607
PyArg_ParseTuple() function, 597
PyArg_ParseTupleAndKeywords() function, 597
PyBytes_AsString() function, 611
.pyc files, 147
  compilation on import, 148
  preventing the creation of, 230
when created, 148
pyclbr module, 586
.pyd files, compiled extensions, 148
pydev, 5
pydoc command, 24
PyErr_Clear() function, 606
PyErr_ExceptionMatches() function, 606
PyErr_NoMemory() function, 605
PyErr_Occurred() function, 606
PyErr_SetFromErrno() function, 605
PyErr_SetFromErrnoWithFilename() function, 605
PyErr_SetObject() function, 605
PyErr_SetString() function, 605
PyEval_CallObject() function, 610
PyEval_CallObjectWithKeywords() function, 610
PyExc_* exceptions, in extension modules, 605
PyFloat_AsDouble() function, 611
PyImport_ImportModule() function, 610
PyInt_AsLong() function, 611
PyLong_AsLong() function, 611
PyModule_AddIntConstant() function, 604
PyModule_AddIntMacro() function, 605
PyModule_AddObject() function, 604
PyModule_AddStringConstant() function, 604
PyModule_AddStringMacro() function, 605
PyObject_GetAttrString() function, 611
PyObject_SetAttrString() function, 610
pypi (Python Package Index), 154
pyprocessing library, 435
PyRun_AnyFile() function, 608
PyRun_InteractiveLoop() function, 609
PyRun_InteractiveOne() function, 609
PyRun_SimpleFile() function, 609
PyRun_SimpleString() function, 609
PyString_AsString() function, 611
PySys_GetArgv() function, 610
Python 3
  2to3 tool, 635–637
  Ellipsis as an expression, 626
  I/O system, 349, 631
  Unicode characters in identifiers, 622
  __next__() method, 633
  absolute imports, 634
  abstract base class, 137
  adoption of, 621
  and WSGI, 541
  byte strings and system interfaces, 630
  chained exceptions, 626
  command line options, 633
  commands module, 332
comparison, 633
dictionary comprehension, 623
dictionary operations, 45
difference in extension modules, 595
different behavior of byte strings, 629
division operator, 65
encode() and encode() methods, 629
environment variables, 633
exception attributes, 213
exec() function, 631
extended iterable unpacking, 623
filenames, 633
files, 160
filter() function, 205
function annotations, 624
generator changes, 103
import statement, 151
incompatibility with Python 2, 621
type division, 633
interactive mode encoding issues, 175
iterator protocol, 633
keyword-only arguments, 625
map() function, 207
metaclasses, 139, 627–628
migration pitfalls, 629
network programming, 452
next() method of generators, 53
nonlocal statement, 624
open() function, 159, 208, 279
practical porting strategy, 637
print() function, 209, 631
raw_input() function, 209
reorganization of network modules, 497
round() function, 209
set comprehension, 623
set literals, 622
socketserver module, 489
standard library reorganization, 634
super() function, 120, 210, 627
supporting both Python 2 and 3, 638
syntax error with print, 6
third party libraries, 621
types module, 237
unbound methods, 49
unicode() function removal, 211
using new built-in functions in Python 2, 217
view objects on dicts, 632
viewing objects in ASCII, 201
who should use, 621
xrange() and range() functions, 17
xrange() function removal, 44, 211
zip() function, 83, 211
python interpreter, 6
PYTHON* environment variables, 174
Python.h header file, in extensions, 594
.pyw files, 147, 176
PyZipFile() function, zipfile module, 325
Q
-Q command line option, 173
q(uit) debugger command, pdb module, 189
qsize() method, of Queue objects, 419, 445
queries, how to safely form for databases, 300
query attribute
of urlparse objects, 520
of urlsplit objects, 521
QueryInfoKey() function, winreg module, 410
QueryValue() function, winreg module, 410
QueryValueEx() function, winreg module, 410
queue module, 444
Queue() function
multiprocessing module, 418
queue module, 444
Queue() method, of Manager objects, 429
queue, circular, 262
queues
coroutines, 108
d example with threads, 446

How can we make this index more useful? Email us at indexes@samspublishing.com
message passing, 415
multiple consumers and producers, 420
priority, 269
shared by multiple processes, 429
thread programming, 444
quit() method
of FTP objects, 498
of SMTP objects, 514
quitting the interactive interpreter, 7
quopri module, 567
quote() function, urllib.parse module, 522
quote_from_bytes() function, urllib.parse module, 522
quote_plus() function, urllib.parse module, 522
quoteattr() function, xml.sax.saxutils module, 583
quotes, difference between styles, 27
quoting, characters in URLs, 522

R

`r` specifier in string formatting, 74
r character, before a string literal, 29
`r` mode, to open() function, 159
r(eturn) debugger command, pdb module, 189
race condition, 193, 414
__radd__() method, 60
when invoked over __add__(), 134
radians() function, math module, 251
radix attribute, of sys.float_info, 231
raise statement, 23, 84-85, 88
__rand__() method, 61
RAND_add() function, ssl module, 488
RAND_egd() function, ssl module, 488
RAND_status() function, ssl module, 488
randint() function, random module, 254
random module, 254
random numbers, and threads, 256
random() function, random module, 255
randrange() function, random module, 254
range() function, 17, 209
range of integer values, 38
Rational abstract base class, 253
rational numbers, 250
raw I/O on files, 350
raw socket, 470
raw strings, 29
Unicode, 29
backslash rules, 29
use in regular expressions, 281
raw-unicode-escape encoding, description of, 170
raw_decode() method, of JSONDecoder objects, 566
raw_input() function, 10, 162, 209
Python 3, 11, 209
RawArray() function, multiprocessing module, 427
RawConfigParser class, configparser module, 336
RawIOBase abstract base class, 354
RawValue() function, multiprocessing module, 426
RCVALL_* constants, socket module, 482
__rdiv__() method, 60
__rdivmod__() method, 60
re attribute, of MatchObject objects, 286
re module, 41, 69, 281
read() function, os module, 385
read() method
of BufferedReader objects, 351
of ConfigParser objects, 333
of FileIO objects, 350
of HTTPResponse objects, 504
of StreamReader objects, 278
of TextIOWrapper objects, 353
of ZipFile objects, 326
of files, 159-160
of mmap objects, 371
of ssl objects, 488
of urlopen objects, 515
read-eval loop, 5
read1() method, of BufferedReader objects, 351
read_byte() method, of mmap objects, 371
read_mime_types() function, mimetypes module, 567
readable() method
  of IOBase objects, 349
  of dispatcher objects, 456
readall() method, of FileIO objects, 350
reader() function, csv module, 549
ReadError exception, tarfile module, 322
readfp() method, of ConfigParser objects, 334
reading CSV data, example of, 14
reading configuration files, 332
reading lines, files, 10
reading user input, 10, 162
readinto() method, of BufferedReader objects, 351
readline library, 176
readline module, 586
readline() method
  of IOBase objects, 349
  of StreamReader objects, 278
  of TextIOWrapper objects, 353
  of files, 10, 159-160
  of mmap objects, 371
  of urlopen objects, 515
readlines() method
  of IOBase objects, 349
  of StreamReader objects, 278
  of files, 13, 159-160
  of urlopen objects, 515
readlink() function, os module, 388
ready() method, ofAsyncResult objects, 425
Real abstract base class, 253
real attribute
  of complex numbers, 39
  of floating point, 39
realpath() function, os.path module, 398
reason attribute, of HTTPResponse objects, 504
reassigning part of a list, 12
Record objects, logging module, 359
recursion limit, changing, 112, 235
recursion, 112
  and decorators, 102, 113
  and generator functions, 112
recursive traversal of directory trees, 390
recv() method
  of Connection objects, 421
  of dispatcher objects, 456
  of socket objects, 483
recv_bytes() method, of Connection objects, 421
recv_bytes_into() method, of Connection objects, 422
recv_into() method, of socket objects, 483
recvfrom() method, of socket objects, 483
recvfrom_info() method, of socket objects, 484
recvmsg() system call, lack of support, 486
reduce() function, functools module, 268
  __reduce__() method, 229
  __reduce_ex__() method, 229
reentrant mutex lock, 438
ref() function, weakref module, 240
reference counting, 34, 129
  and copying, 35
  and del statement, 35
  and memory use, 192
  and mutable objects, 36
  in extension modules, 607
  obtaining, 35
reference cycles
  and garbage collection, 221
  avoiding with weak references, 130, 240
ReferenceError exception, 87, 214
REG_* constants, winreg module, 409
Regex objects, re module, 284
register command of setup.py file, 155

How can we make this index more useful? Email us at indexes@samspublishing.com
register() function
- atexit module, 179, 219
- webbrowser module, 544

register() method
- of BaseManager class, 430
- of Poll objects, 459
- of abstract base classes, 137

register_adapter() function, sqlite3 module, 305

register_converter() function, sqlite3 module, 304

register_dialect() function, csv module, 551

register_function() method, of XMLRPCServer objects, 527

register_instance() method, of XMLRPCServer objects, 527

register_introspection_functions() method, of XMLRPCServer objects, 528

register_multicall_functions() method, of XMLRPCServer objects, 528

RegLoadKey() function, winreg module, 409

regular expressions
- pattern syntax, 281
- re module, 281
- use of raw strings, 281

relational databases, accessing from Python, 297

relational operators, 9, 56

relative package imports, 150-151

release() method
- of Condition objects, 441
- of Lock objects, 438
- of RLock objects, 439
- of Semaphore objects, 439

reliable datagrams, 470

reload() function, 149

reloading modules, 149

relpath() function, os.path module, 398

remote procedure call
- XML-RPC, 524
- multiprocessing module, 423-424

remove() function, os module, 388

remove() method
- of Element objects, 577
- of array objects, 260

of deque objects, 262
of lists, 40-41
of sets, 15, 47

remove_option() method, of ConfigParser objects, 334

remove_section() method, of ConfigParser objects, 334

removeChild() method, of DOM Node objects, 571

removedirs() function, os module, 388

removeFilter() method
- of Handler objects, 364
- of Logger objects, 359

removeHandler() method, of Logger objects, 361

removing directories, 318

removing files, 388

removing sequence items, 40

removing slices, 40

rename() function, os module, 388

rename() method, of FTP objects, 498

renames() function, os module, 388

repeat() function
- cProfile module, 190
- itertools module, 272
- operator module, 274
- timeit module, 192

'replace' error handling, Unicode encoding, 166

replace() method
- of date objects, 337
- of datetime objects, 340
- of strings, 41, 43
- of time objects, 338

replace_header() method, of Message objects, 556

replaceChild() method, of DOM Node objects, 571

replacing substrings, 41

replication, of sequences and shallow copies, 67

report() method, of dircmp objects, 315

report_full_closure() method, of dircmp objects, 315
report_partial_closure() method, of dircmp objects, 315
repr (replib) module, 586
repr() function, 11, 55, 77, 176, 209
    and eval(), 55
    difference from str(), 12
__repr__() method, 55-56
representing dates and times, 336
request attribute, of BaseRequestHandler objects, 490
Request() function, urllib.request module, 516
request() method, of HTTPConnection objects, 503
request_queue_size attribute, of SocketServer class, 492
request_version attribute, of BaseHTTPRequestHandler objects, 509
RequestHandlerClass attribute, of SocketServer objects, 492
rerasing the last exception, 84
reserved attribute, of ZipInfo objects, 327
reserved words, 26
reset() method
    of HTMLParser objects, 562
    of IncrementalDecoder objects, 279
    of IncrementalEncoder objects, 278
    of StreamReder objects, 278
    of StreamWriter objects, 278
resetwarnings() function, warnings module, 239
resize() function, ctypes module, 617
resize() method, of mmap objects, 371
resolution attribute
    of date class, 337
    of datetime class, 340
    of time class, 338
    of timedelta class, 341
resource module, 587
response time, asynchronous networking, 467
ResponseNotReady exception, http.client module, 504
responses attribute, of BaseHTTPRequestHandler class, 509
restricting attribute names, 132
restype attribute, of ctypes function objects, 613
result of last operation in interactive mode, 6, 176
retrbinary() method, of FTP objects, 499
retrlines() method, of FTP objects, 499
return statement, 96
reverse keyword argument, to sort(), 40
reverse() method
    of array objects, 260
    of lists, 40-41
reversed operand methods, when invoked, 134
reversed operands, operator overloading, 60
reversed() function, 209
reversing a list, 40
rfile attribute
    of BaseHTTPRequestHandler objects, 509
    of StreamRequestHandler objects, 491
rfind() method, of strings, 41, 43
__rfloordiv__() method, 60
right shift operator >>, 65
right_list attribute, of dircmp objects, 315
right_only attribute, of dircmp objects, 315
rindex() method, of strings, 41, 43
rjust() method, of strings, 43
rlcompleter module, 586
rledecode_hqx() function, binascii module, 548
rledecode_hqx() function, binascii module, 548
scoping rules
and module imports, 145
and self parameter in methods, 118
lexical scoping of functions, 97
of classes, 118
of function variables, 18, 96
of iteration variable in list comprehension, 109
of iteration variables, 82
script name, 157
search path, for modules, 147
search() function, re module, 284
search() method, of Regex objects, 285
searching, strings with an offset, 41
second attribute, of time objects, 338
sections() method, of ConfigParser objects, 334
secure sockets layer (SSL), 486
security
XML-RPC servers, 530
database queries, 300
marshal module, 226
pickle module, 172, 229
seed() function, random module, 254
seek() method
of IOBase objects, 350
of files, 160-161, 352
of mmap objects, 371
seekable() method, of IOBase objects, 350
select module, 415, 459
signal handling, 399
select() function
and asyncore module, 455
performance problems, 467
select module, 459
self parameter of methods, 22, 118
why required, 119
_self_ attribute
of built-in functions, 49
of methods, 49
Semaphore object
multiprocessing module, 427
threading module, 439
Semaphore objects, use for signaling, 439
Semaphore() method, of Manager objects, 429
semicolon ;, 26
send() method
of Connection objects, 422
of HTTPConnection objects, 503
of dispatcher objects, 456
of generators, 20, 53, 104
of socket objects, 484
send_bytes() method, of Connection objects, 422
send_error() method, of
BaseHTTPRequestHandler objects, 509
send_header() method, of
BaseHTTPRequestHandler objects, 509
send_response() method, of
BaseHTTPRequestHandler objects, 509
send_signal() method, of Popen objects, 403
sendall() method, of socket objects, 484
sendcmd() method, of FTP objects, 499
sending email, example of, 514, 558
sendmail() method, of SMTP objects, 514
sendmsg() system call, lack of support, 486
sendto() method, of socket objects, 484
sentinel, use with queuing, 420, 446
sep keyword argument, to print() function, 163
sep variable, os module, 386
separator character, print() function, 163
Sequence abstract base class, 266
sequences, 39
comparison of, 70
concatenation, 67
extended slicing of, 68
in operator, 67
indexing in string formatting, 72
item assignment, 40
iteration over, 39, 69
lookup in composite string formatting, 42
negative indices, 68
operators, 67
picking random elements, 254
random sampling, 254
replication, 67
shallow copies in replication, 67
slice assignment, 40
slicing operator, 68
special methods of, 58
unpacking, 67-68
serve_forever() method
of BaseManager objects, 431
of SocketServer objects, 491
server attribute, of BaseRequestHandler objects, 490
server program, 449
TCP example, 451
UDP example, 485
example of restricting access with HTTP, 507
example with SocketServer module, 490
example with coroutines, 464
server_address attribute, of SocketServer objects, 492
server_version attribute
of BaseHTTPRequestHandler class, 508
of HTTPRequestHandler class, 507
ServerProxy() function, xmlrpc.client module, 524
Set abstract base class, 266
set comprehension, Python 3, 623
set difference operator -, 75
set intersection operator &, 75
set literals, Python 3, 622
set symmetric difference operator ^, 75
set theory, similarity to list comprehensions, 110
set type, 38, 46, 75
set union operator |, 75
set() method
of ConfigParser objects, 334
of Element objects, 577
of Event objects, 440
of Morsel objects, 512
__set__() method, of descriptors, 58, 126
set_authorizer() method, of Connection objects, 306
set_boundary() method, of Message objects, 556
set_charset() method, of Message objects, 556
set_conversion_mode() function, ctypes module, 617
set_debug() function, gc module, 221
set_default_type() method, of Message objects, 556
set_defaults() method, of OptionParser objects, 158, 377
set_errno() function, ctypes module, 618
set_executable() function, multiprocessing module, 435
set_last_error() function, ctypes module, 618
set_param() method, of Message objects, 556
set_proxy() method, of Request objects, 517
set_server_documentation() method, of XMLRPCServer objects, 528
set_server_name() method, of XMLRPCServer objects, 528
set_server_title() method, of XMLRPCServer objects, 528
set_terminator() method, of asyncchat objects, 453
set_threshold() function, gc module, 221
set_trace() function, pdb module, 186
set_type() method, of Message objects, 557
set_unixfrom() method, of Message objects, 557
set_usage() method, of OptionParser objects, 377
set_wakeup_fd() function, signal module, 399
setattr() function, 210
   and private attributes, 128
   __setattr__() method, 57-58, 131
   and __slots__, 133
setblocking() method, of socket objects, 484
setcheckinterval() function, sys module, 235
setcontext() function, decimal module, 248
setDaemon() method, of Thread objects, 437
setdefault() method
   of dicts, 45
   of dicts and defaultdict objects, 263
setdefaultencoding() function, sys module, 235
setdefaultencoding() method, sys module, 177
setdefaulttimeout() function, socket module, 476
setdlopenflags() function, sys module, 235
setDocumentLocator() method, of ContentHandler objects, 581
setegid() function, os module, 380
seteuid() function, os module, 380
setName() method, of Thread objects, 436
setinputsize() method, of Cursor objects, 299
setitem() function, operator module, 274
   __setitem__() method, 58-59
   and slices, 59
setitem() function, signal module, 399
setLevel() method
   of Handler objects, 364
   of Logger objects, 359
setmode() function, msvcrt module, 373
setName() method, of Thread objects, 436
setoutputsize() method, of Cursor objects, 299
setpassword() method, of ZipFile objects, 326
setpgid() function, os module, 381
setpgrp() function, os module, 381
setprofile() function
   sys module, 235
   threading module, 444
setrecursionlimit() function, sys module, 235
setsid() function, os module, 381
setregid() function, os module, 381
setreuid() function, os module, 381
_setroot() method, of ElementTree objects, 574
sets, 15
   adding items, 15
   creating from iterable objects, 46
   difference operator, 15
   equality of, 78
   in-place modification of, 47
   intersection operator, 15
   length of, 75
   removing items, 15
   symmetric difference operator, 15
   union operator, 15
   updating, 15
setsid() function, os module, 381
setslice() function, operator module, 274
setsockopt() method, of socket objects, 484
setState() function, random module, 254
   __setState__() method, 228
   and copying, 220
   and pickle module, 172
@setter decorator of properties, 126
settimeout() method, of socket objects, 484
setting default encoding of standard I/O, 175

settrace() function
   sys module, 235
   threading module, 444

setuid() function, os module, 381

setUp() method
   TestCase objects, 184
   of TestCase objects, 184

setup() function, distutils module, 152, 596

setup() method, of BaseRequestHandler objects, 490

setup.py file
   C extensions, 596
   SWIG extensions, 620
   and setuptools, 154
   creating, 152-153
   install command, 153-154
   installing in per-user site directory, 178

setuptools library, 147, 154

SetValue() function, winreg module, 411

SetValueEx() function, winreg module, 411

sha1() function, hashlib module, 559

sha224() function, hashlib module, 559

sha256() function, hashlib module, 559

sha384() function, hashlib module, 559

sha512() function, hashlib module, 559

shallow copy, 36
   of dicts, 45
   of lists, 40
   sequence replication, 67

shared arrays, multiprocessing module, 426

shared libraries
   extension modules, 148
   loading with ctypes, 612

shared memory
   example of passing a list, 427
   multiprocessing module, 426

sharing of objects, 35

Shelf class, shelve module, 311

shell commands
   collecting output from, 331
   emulating in Python, 318

shell pipes, similarity to generators, 106

shelve module, 171, 311
   dbhash module, 311
   selecting the pickle protocol, 172

shlex module, 588

short-circuit evaluations, of boolean expressions, 78

showwarning() function, warnings module, 239

shuffle() function, random module, 255

shutdown() function, logging module, 366

shutdown() method
   of BaseManager objects, 431
   of SocketServer objects, 492
   of socket objects, 484

shutil module, 318

side effects
   in functions, 95
   reasons to avoid, 96

SIG* signal names, 400

SIGHUP signal, 179

siginterrupt() function, signal module, 400

signal handling, 399

signal module, 399

signal() function, signal module, 400

signaling, with semaphores, 439

signals
   close() method of generators, 104
   list of, 400
   mixing with threads, 402
   throw() method of generators, 105

SIGTERM signal, 179

simple_producer() function, asynchat module, 453

SimpleCookie() function, http.cookies module, 512

SimpleHandler() function, wsgiref.handlers module, 543

SimpleHTTPRequestHandler class, http.server module, 507

SimpleHTTPServer module, see http.server, 506

SimpleXMLRPCServer class, xmlrpc.server module, 527
SimpleXMLRPCServer module, 527
sin() function, math module, 252
single precision floating point, 39
singleton tuple, 14
sinh() function, math module, 252
site configuration files, 177
site module, 166, 174, 177
site-packages directory, 175
sitecustomize module, 177
size attribute
  of Struct objects, 291
  of TarInfo objects, 322
size() method
  of FTP objects, 499
  of mmap objects, 371
Sized abstract base class, 265
sizeof() function, ctypes module, 618
skippedEntity() method, of ContentHandler objects, 581
sleep() function, time module, 406
sleeping, 406
  until signal received, 399
slice assignment, lists, 12
slice objects, 51, 53
  and indexing methods, 59
  attributes of, 54
slice type, 51
slice() function, 53, 210
slices, 39
  and special methods, 59
  and xrange objects, 44
  assignment to, 40, 69
  deletion of, 40, 69
  multidimensional, 59
slicing operator [:], 67-68
  on lists, 12
  on strings, 11
__slots__ attribute
  and __dict__ attribute of instances, 50
  compatibility with other code, 133
  inheritance, 133
  of class definitions, 132
  optimization, 196
Smalltalk, differences in object system, 124
SMTP protocol, example of sending a message, 514
SMTP() function, smtplib module, 514
smtpd module, 587
SMTPHandler class, logging module, 363
smtplib module, 513
sndhdr module, 588
sniff() method, of Sniffer objects, 550
Sniffer() function, csv module, 550
SO_* socket options, socket module, 479
SOCK_* constants, socket module, 470
socket attribute, of SocketServer objects, 492
socket module, 469
socket() function, socket module, 476
socket, definition of, 449
socket_type attribute, of SocketServer class, 493
SocketHandler class, logging module, 363
socketpair() function, socket module, 477
sockets
  address families, 470
  methods on, 478
  polling with select(), 459
  specifying network addresses, 471
  types of, 470
SocketServer module, 489
  and Python 3, 489
  changing server parameters, 492
softspace attribute, of files, 161
sort() method, of lists, 40-41
sorted() function, 210
sorting
  changing behavior of, 40
  in-place on lists, 40
  requirements for objects, 57
  reverse order, 40
  use of operator module, 275

How can we make this index more useful? Email us at indexes@samspublishing.com
source code encoding, 31
Python 3, 622
span() method, of MatchObject objects, 286
spawnl() function, os module, 393
spawnle() function, os module, 393
spawnlp() function, os module, 393
spawnlpe() function, os module, 393
spawnv() function, os module, 392
spawnve() function, os module, 393
spawnvpe() function, os module, 393
special methods, 21, 54
special symbols, 30
speedup, definition of, 192
split() function
   os.path module, 398
   re module, 284
split() method
   of Regex objects, 285
   of strings, 14, 41, 43
splitdrive() function, os.path module, 398
splttext() function, os.path module, 398
splitlines() method, of strings, 44
splitting, strings, 14, 43
splitunc() function, os.path module, 398
SpooledTemporaryFile() function, tempfile module, 324
sprintf() function equivalent, 70
spwd module, 587
SQL queries
   SQL injection attack, 300
      examples of, 309
      executing on database, 297
      how to form, 300
      similarity to list comprehensions, 111
SQLite database, 303
sqlite3 module, 303
sqrt() function, math module, 252
sqrt() method, of Decimal objects, 243
ssl module, 486
SSL, example of creating certificates, 489
st_* attributes, of stat objects, 389
stack frames, 52
   in tracebacks, 53
stack size, for threads, 444
stack() function, inspect module, 225
stack_size() function, threading module, 444
Stackless Python, 467
standard I/O streams, 161
   and integrated development environments, 162
   setting the default encoding, 175
standard error, 157
standard input and output, 10
standard library reorganization, Python 3, 634
standard_b64decode() function, base64 module, 546
standard_b64encode() function, base64 module, 546
StandardError exception, 87
starmap() function, itertools module, 272
start attribute, of slices, 54
start() method
   of BaseManager objects, 431
   of MatchObject objects, 286
   of Process objects, 416
   of Thread objects, 436
   of Timer objects, 438
   of TreeBuilder objects, 577
startDocument() method, of ContentHandler objects, 582
startElement() method, of ContentHandler objects, 582
startElementNS() method, of ContentHandler objects, 582
startfile() function, os module, 393
startPrefixMapping() method, of ContentHandler objects, 582
startswith() method, of strings, 44
startup script, in interactive mode, 174
stat module, 387, 587
stat() function
   os module, 388
   os.path module, 398
| **stat_float_times() function, os module, 389** |
| **statement termination, 7** |
| and semicolons, 7 |
| **statements** |
| breaking across multiple lines, 9 |
| putting on the same line, 25-26 |
| running in the debugger, 186 |
| **static method, 22, 48, 123, 125** |
| practical use of, 123 |
| **@staticmethod decorator, 22, 48, 123, 125, 210** |
| **statistics, random number distributions, 255** |
| **status attribute, of HTTPResponse objects, 504** |
| **statvfs() function, os module, 389** |
| **stderr attribute, of Popen objects, 404** |
| stderr variable, sys module, 161, 232 |
| __stderr__ variable, sys module, 162, 232 |
| **stdin attribute, of Popen objects, 404** |
| stdin variable, sys module, 10, 161, 232 |
| __stdin__ variable, sys module, 162, 232 |
| **stdout attribute, of Popen objects, 404** |
| stdout variable, sys module, 10, 161, 232 |
| __stdout__ variable, sys module, 162, 232 |
| **step attribute, of slices, 54** |
| **StopIteration exception, 59, 87, 215** |
| and generators, 103 |
| **storbinary() method, of FTP objects, 499** |
| **storlines() method, of FTP objects, 499** |
| **str type, 38** |
| **str() function, 11, 56, 76, 210** |
| and print, 162 |
| difference from repr(), 12 |
| __str__() method, 55-56 |
| **StreamError exception, tarfile module, 322** |
| **StreamHandler class, logging module, 363** |
| **StreamReader class, codecs module, 278** |
| **streamreader() method, of CodecInfo objects, 278** |
| **StreamRequestHandler class, SocketServer module, 491** |
| **streams, 470** |
| **StreamWriter class, codecs module, 278** |
| **strftime() method, of CodecInfo objects, 278** |
| **strftime() function, os module, 381** |
| **strftime() function, time module, 406** |
| **strftime() method** |
| of date objects, 337 |
| of time objects, 338 |
| ‘strict’ error handling, Unicode encoding, 166 |
| **string attribute, of MatchObject objects, 286** |
| **string formatting, 70** |
| ‘l’ specifier, 74 |
| ‘s’ specifier, 74 |
| alignment, 73 |
| attribute lookup, 72 |
| codes for % operator, 70-71 |
| customizing format() method, 74 |
| dictionaries, 72 |
| dictionary lookup, 72 |
| fill characters, 73 |
| format specifiers, 72 |
| formatting operator %, 70 |
| **string interpolation, 72, 163** |
| **string literals, 27** |
| Unicode characters, 28 |
| and Unicode encodings, 29 |
| and documentation strings, 30 |
| byte strings, 29 |
| unicode characters in source code, 31 |
| **string module, 287** |
| Template strings, 164 |
| **string_at() function, ctypes module, 618** |
| **StringIO class, io module, 353** |
| **stringprep module, 586** |
| **strings, 11** |
| Unicode, 41, 165 |
| and numeric calculations, 11 |

*How can we make this index more useful? Email us at indexes@samspublishing.com*
as a dictionary key, 16
as in-memory text files, 353
as sequences, 39
basestring object for type checking, 202
byte literals, 29
byte strings, 41, 202
case conversion, 43–44
case conversion, 43
character substitution, 42
comparison of, 70
concatenation, 11
concatenation of adjacent literals, 27
creating custom formatters, 288
disabling escape codes in literals, 29
encoding for URLs, 522
escape codes in literals, 27
escaping characters for use in HTML, 536
escaping characters for use in XML, 583
executing Python code contained within, 115
format() method, 8, 72
formatting, 8, 42
formatting in log messages, 358
immutability of, 41, 69
indexing, 11
internal representation, 28
iterating over characters, 17
joining, 43
line structure, 11
mixing byte strings and Unicode, 70
mutable byte arrays, 202
partitioning, 41
regular expressions, 281
replacing substrings, 41
searching for substrings, 41
slices, 11
sorting and internationalization, 70
specifiers for format() method, 72–73
splitting, 43
splitting into fields, 14, 41
stripping, 43
unescaping XML character references, 583
strip() method, of strings, 44

stripping
control characters from a string, 42
strings, 44
strptime() function, time module, 343, 407
strptime() method, of datetime class, 339
Struct class, struct module, 291
struct module, 290
Structure class, ctypes module, 614
structures, and tuples, 14
sub() function
operator module, 273
re module, 284
sub() method, of Regex objects, 285
__sub__() method, 60
subclass, 119
__subclasscheck__() method, 57, 136
subdirs attribute, of dircmp objects, 315
SubElement() function,
xmletree.ElementTree module, 575
subn() function, re module, 284
subn() method, of Regex objects, 285
subprocess module, 402
subprocess, definition of, 413
substitute() method
of Template objects, 289
of Template strings, 164
substrings
checking for existence using in, 69
searching for, 41
subtraction operator -, 65
successful() method, ofAsyncResult objects, 425
sum() function, 39–40, 67, 210
accuracy of, 252
and decimal module, 69
restriction to numeric data, 39
versus math.fsum() function, 252
sunau module, 588
super() function, 120, 210
Python 3, 210, 627
superclass, 119
calling methods in, 120
super() function, 120
supporting both Python 2 and 3, 638
supports_unicode_filenames variable, 
os.path module, 398
surrogate pair, 28, 41
suspension, of threads, 443
swapcase() method, of strings, 44
SWIG, 591
   example of, 619
   interface file, 619
switch statement, lack of, 9
symbol module, 586
symbolic links, testing a filename for, 397
symlink() function, os module, 389
symmetric pair, 28, 41
symmetric_difference operator ^, of sets, 15
symmetric_difference() method, of sets, 46
symmetric_difference_update() method, of 
   sets, 47
sync() method
   of dbm-style database objects, 310
   of shelve objects, 311
synchronization primitives
   multiprocessing module, 427
   threading module, 438
synchronization
   of close() method of generators, 104
   of concurrent programs, 414
   of throw() method of generators, 105
SyntaxError exception, 87, 215
   Python 3 print statement, 6
   and default arguments, 93
   and except statements, 85
SyntaxWarning warning, 216, 238
sys module, 13, 229
sys.argv variable, 13, 157, 174
sys.displayhook variable, 176
sys.exec_prefix variable, 177
sys.exit() function, 179
sys.modules variable, 144, 149
sys.path variable, 147
   and site module, 177
   third-party modules, 154
sys.prefix variable, 177
sys.ps1 variable, 176
sys.ps2 variable, 176
sys.stderr variable, 157, 161
sys.stdin variable, 161
sys.stdout variable, 161
sys_version attribute, of 
   BaseHTTPRequestHandler class, 508
sysconf() function, os module, 395
syslog module, 587
SysLogHandler class, logging module, 363
system calls, os module, 378
system error codes, 344
system() function, os module, 393
system.listMethods() method, of 
   ServerProxy objects, 525
SystemError exception, 87, 215
SystemExit exception, 7, 87-88, 157, 179,
   215
-t command line option, 26, 173-174
tab escape code in strings, 28
tabcheck attribute, of sys.flags, 230
TabError exception, 26, 87, 215
tabnanny module, 586
tabs, and indentation, 26
tag attribute, of Element objects, 576
tagName attribute, of DOM Element 
   objects, 572
tail attribute, of Element objects, 576
tail command, example with generators, 19
tail-recursion optimization, lack of, 112	
takewhile() function, itertools module, 272
tan() function, math module, 252
tanh() function, math module, 252
TarError exception, tarfile module, 322
TarFile objects, tarfile module, 320
tarfile module, 319
TarInfo objects, tarfile module, 321

How can we make this index more useful? Email us at indexes@samspublishing.com
task scheduler, example with coroutines and select(), 460

**task_done() method**
- of JoinableQueue objects, 419
- of Queue objects, 445

**tasklets, 446**
- asynchronous I/O, 460

**tasks, and coroutines, 20**

**tb_* attributes**, of traceback objects, 53

**tb_lineno() function**, traceback module, 236

**tbreak debugger command**, pdb module, 189

**tcgetpgrp() function**, os module, 385

**TCP connection**, diagram of, 450

**TCP protocol, 449**
- example code, 451

**TCP_* socket options**, socket module, 482

**TCPServer class**, SocketServer module, 491

**tcsetpgrp() function**, os module, 385

**tearDown() method**
- of TestCase objects, 184
- of TestCase objects, 184

**tee() function**, itertools module, 273

**tell() method**
- of IOBase objects, 350
- of files, 160-161
- of mmap objects, 371

**telnetlib module, 587**

**tempdir variable**, tempfile module, 324

**tempfile module, 323**

**Template class**, string module, 289

**Template strings**
- string module, 164
- use in CGI script, 537

**template attribute**, of Template objects, 290

**template variable**, tempfile module, 324

**temporary files**, 323

**TemporaryFile() function**, tempfile module, 323

**terminate() method**
- of Pool objects, 425
- of Popen objects, 403-404
- of Process objects, 416

**terminating statements with semicolons, 7**

**termination**
- immediate without garbage collection, 391
- of programs, 179
- of threads, 443
- registering cleanup functions, 219
- sys.exit() function, 233
- without garbage collection, 179

**termios module, 587**

**test module, 586**

**test() function**, cgi module, 537

**TestCase class**, unittest module, 184

**testing**
- doctest module, 181
- documentation strings, 181
- limitations of doctest, 183
- unit testing, 183

**testmod() function**
- doctest() module, 182
- doctest module, 182

**testzip() method**, of ZipFile objects, 326

**Text class**, xml.dom.minidom module, 572

**text I/O, 353**

**text attribute**, of Element objects, 576

**text file mode, 159**

**text replacement, replace() method of strings, 41**

**text versus bytes in Python 3, 629**

**text_factory attribute**, of Connection objects, 308

**TextIOBase abstract base class, 354**

**TextIOWrapper class**, io module, 353

**textwrap module, 586**

**third-party libraries, and Python 3, 621**

**third-party packages**
- and C/C++ code, 154
- and sys.path variable, 154
threads

CPU-bound tasks, 444
adding to network servers, 494
atomic operations and disassembly, 193
check interval, 234
close() method of generators, 104
compared to coroutines, 467
directory variables, 441
decimal module, 249
definition of, 413
events, 440
extension modules, 607
global interpreter lock, 414, 444
local storage, 443
main thread, 413
mutex lock, 438
obtaining number of active, 443
random number generation, 256
reentrant mutex lock, 438
scaling properties, 415
scheduling of, 414
semaphores, 439
setting stack size, 444
setting the name, 436

signal handling, 402
signaling with semaphores, 439
suspending, 443
synchronization of, 414
termination of, 443
throw() method of generators, 105
use of queues, 444
worker thread example, 446

threadsafety variable, database API, 302
throw() method, of generators, 105

time and date parsing, 407
time class, datetime module, 338
time manipulation, 336
time module, 191, 405
accuracy of time functions, 408
current time, 405
time parsing, 343

Time() function, database API, 301
time() function, time module, 191, 407
time() method, of datetime objects, 340
timedelta class, datetime module, 340
TimedRotatingFileHandler class, logging module, 363
TimeFromFile() function, database API, 301
timeit module, 191
timeit() function
cProfile module, 190
timeit module, 191
timeout attribute, of SocketServer class, 493
timeout exception, socket module, 485
timeout, example with alarm signals, 401
Timer() function, threading module, 437
times() function, os module, 394
Timestamp() function, database API, 301
TimestampFromFile() function, database API, 301
timetuple() method, of date objects, 337
timetz() method, of datetime objects, 340
timezone variable, time module, 405

How can we make this index more useful? Email us at indexes@samspublishing.com
timing measurements, 191
TIPC protocol, 470
    address format, 472
TIPC_* constants, socket module, 473
    title() method, of strings, 44
Tkinter module, 588
today() method, of date class, 336
tofile() method, of array objects, 260
token module, 586
tokenize module, 586
tolist() method, of array objects, 260
totdordinal() method, of date objects, 337
toprettyxml() method, of DOM Node objects, 572
tostring() function, xml.etree.ElementTree module, 578
tostring() method, of array objects, 260
total_changes attribute, of Connection objects, 308
townload() method, of array objects, 260
toxml() method, of DOM Node objects, 573	race() function, inspect module, 225
traceback messages, 22
traceback module, 235
traceback objects, 51-52
    attributes of, 53
    stack frames, 53
__traceback__ attribute, of Exception objects, 213
tracebacklimit variable, sys module, 232
tracebacks
    creating with traceback module, 236
    limiting the amount of output, 232
TracebackType type, 51, 237
trailing J on complex number literals, 27
trailing L on long integers, 27
trailing comma
    and tuples, 14
    print statement, 162
transfercmd() method, of FTP objects, 500
translate() method, of strings, 42, 44
traps attribute, of Context objects, 247

TreeBuilder() function,
    xml.etree.ElementTree module, 577
triangular() function, random module, 256
triple-quoted strings, 11
    and variable interpolation, 163
True value, 9, 27, 38
truediv() function, operator module, 273
    __truediv__() method, 60
trunc() function, math module, 252
truncate() method
    of IOBase objects, 350
    of files, 160
truncating division operator //, 65
truncation, of integer division, 61, 65
truth value testing, 78
truth values, 9
try() function, operator module, 274
try statement, 22, 84-85
-1t command line option, 26, 173-174
tty module, 587
tyname() function, os module, 385
tuning strategies, 194
tuple type, 38
tuple unpacking, and for-loops, 15
tuple() function, 77, 211
tuples, 14
    and string formatting, 70
    as dictionary key, 16
    as dictionary keys, 74
    as sequence, 39
    comparison on, 70
    concatenation, 14
    creating a list of from dictionary, 45
    creating with named attributes, 264
    immutability, 14
    immutability of, 69
    indexing, 14
    memory savings of, 14
    omission of parantheses, 14
    problems with using as data structure, 264
    representing records, 14
    singleton, 14
slicing, 14
unpacking in Python 3, 623
unpacking in iteration, 82
use by standard library, 265
use in list comprehensions, 109
versus lists, 14
Twisted library, 415, 467
two-dimensional lists, 13, 15
type, 47
type attribute
of FieldStorage objects, 535
of TarInfo objects, 322
of socket objects, 484
type checking
example with metaclass, 140
of objects, 34
performance impact of, 34
problem with proxy objects, 135
type coercion, and operator overloading, 134
type comparison of objects, 34
type conversion, 76
lack of implicit conversion, 62
of columns in a datafile, 37
special methods for, 62
type hierarchies, 138
type objects, 50
type of objects, 33
type() function, 34, 211
and exceptions, 89
type() metaclass, 138
type_options attribute, of FieldStorage objects, 535
typecode attribute, of array objects, 259
TypeError exception, 87, 215
and function calls, 94
and type coercion, 62
method resolution order, 122
types module, 47, 237
and Python 3, 237
types
boolean, 38
built-in, 37, 201
callable, 47
dictionary, 44
floating point, 38
defrozenset, 46
integers, 38
of sockets, 470
set, 46
type of, 47
tzinfo attribute, of time objects, 338
tzname variable, time module, 405
tzname() method
of time objects, 339
of tzinfo objects, 342
tzset() function, time module, 407
\U escape code, in strings, 28
\u escape code, in strings, 28
-U command line option, 28, 173
‘U’ mode, to open() function, 159
u character, before a string literal, 28
-u command line option, 173
u(p) debugger command, pdb module, 189
UDP client example, 486
UDP communication, diagram of, 451
UDP protocol, 449
UDP server example, 485
UDPServer class, SocketServer module, 491
uid attribute, of TarInfo objects, 322
umask() function, os module, 381
unalias debugger command, pdb module, 189
uname attribute, of TarInfo objects, 322
uname() function, os module, 381
unary minus operator -, 65
unary plus operator +, 65

How can we make this index more useful? Email us at indexes@samspublishing.com
unbound method, 49
   and Python 3, 49
UnboundLocalError exception, 87, 98, 215
unbuffered file I/O, 159
unconsumed_tail attribute, of decompression objects, 329
underscores, usage in identifiers, 26
unescape() function, xml.sax.saxutils module, 583
ungetch() function, msvcrt module, 373
ungetwch() function, msvcrt module, 373
unhexlify() function, binascii module, 548
unichr() function, 77, 211
Unicode character database, 293
Unicode characters, representation of, 41
Unicode string literals, 28
Unicode strings, 41
   and WSGI, 541
      common encodings, 166
      decomposing, 295
      encoding and decoding, 165
      encoding in network programs, 452
      error handling options, 166
      handling of, 165
      mixing with byte strings, 167
      normalizing, 296
      regular expressions, 281
unicode attribute, of sys.flags, 230
unicode() function, 211
   and Python 3, 211
Unicode
   Python 2 vs. Python 3, 28
      and BOM characters, 168
      and XML, 168
      byte order marker, 280
      character encoding and decoding, 42
      character properties database, 170
      code points, 28
      common encodings, 168–169
      encoding in string literals, 29
      encoding of source code, 31
      file I/O, 167
      mixing with byte strings, 70
   normalization of strings, 171
   specifying characters in string literals, 28
   surrogate pairs, 28, 41
   using 32-bit character code points, 41
unicode-escape encoding, description of, 170
unicodedata module, 170, 293
UnicodeDecodeError exception, 87, 215
UnicodeEncodeError exception, 87, 215
   Python 3 interactive mode, 175
UnicodeError exception, 87, 166, 215
UnicodeTranslateError exception, 87, 215
unidata_version variable, unicodedata module, 296
unification of integer and long types, 38
uniform access principle, 125
uniform type arrays, 259
uniform() function, random module, 255
UnimplementedFileMode exception, http.client module, 504
Union class, ctypes module, 614
union operator |, of sets, 15
union() method, of sets, 46
unit testing
   Python 3 migration, 635
   example, 184
   unittest module, 183
unittest module, 183
   example, 184
universal newline mode, 159
UNIX domain protocol, 470
   address format, 472
Unix systems log, issuing message to, 363
Unix
   #! execution of programs, 6
   per-user site directory, 177
   time epoch definition, 405
UnixDatagramServer class, SocketServer module, 491
UnixStreamServer class, SocketServer module, 491
UnknownHandler class, urllib.request module, 518
UnknownProtocol exception, http.client module, 504
UnknownTransferEncoding exception, http.client module, 504
unlink() function, os module, 390
unloading modules, 149
unpack() function, struct module, 290
unpack() method, of Struct objects, 291
unpack_from() function, struct module, 291
unpack_from() method, of Struct objects, 291
unpacking
  binary data structures, 290
  of sequences, 67-68
  of tuples, 14
Unpickler class, pickle module, 228
unquote() function, urllib.parse module, 522
unquote_plus() function, urllib.parse module, 522
unquote_to_bytes() function, urllib.parse module, 522
unregister() method, of Poll objects, 460
unregister_dislect() function, csv module, 551
unsetenv() function, os module, 381
until debugger command, pdb module, 189
unused_data attribute, of decompressobj objects, 329
unwrap() method, of ssl objects, 488
update() method
  of dicts, 45
  of digest objects, 559
  of hmac objects, 560
  of sets, 15, 47
update_wrapper() function, functools module, 269
updating a dictionary, 45
uploading
  files in CGI scripts, 536
  files to a HTTP server with POST, 505
  files to an FTP server, 500
  packages to pypi, 155
upper() method, of strings, 44
uppercase variable, string module, 287
urandom() function, os module, 396
uridefrag() function, urllib.parse module, 521
urllibencode() function, urllib.parse module, 522
URLError exception, 516
  urllib.error module, 523
urljoin() function, urllib.parse module, 521
urlib module, 522
  see urllib.request, 515
urllib package, 514
urllib.error module, 523
urllib.parse module, 520
urllib.request module, 515
urllib.response module, 520
urllib.robotparser module, 523
urllib2 module, see urllib.request, 515
urlopen() function, urllib.request module, 515
urlparse module, 520
urlparse() function, urllib.parse module, 520
urisafe_b64decode() function, base64 module, 546
urisafe_b64encode() function, base64 module, 546
urisplit() function, urllib.parse module, 521
urlunparse() function, urllib.parse module, 521
urlunsplit() function, urllib.parse module, 521
user directory, installation of packages, 154
user module, 586
  —user option to setup.py, 154
user-agent header in HTTP requests, changing, 517
username attribute
  of urlparse objects, 520
  of urllib.parse objects, 521
UserWarning warning, 216, 238

How can we make this index more useful? Email us at indexes@samspublishing.com
using Python as a calculator, 6
utcfromtimestamp() method, of datetime class, 339
utctime() method, of datetime class, 339
utc() method
of time objects, 339
of tzinfo objects, 342
utctimetuple() method, of datetime objects, 340
UTF-16 encoding, description of, 170
UTF-8 compatibility with ASCII, 170
description of, 169-170
encoding and decoding, 42
including in string literals, 29
lexicographic ordering, 170
utime() function, os module, 390
uu module, 587

\-V command line option, 173
\-v command line option, 173-174
validator() function, wsgiref.handlers module, 543
value attribute
of FieldStorage objects, 535
of Morsel objects, 512
Value() function, multiprocessing module, 426
Value() method, of Manager objects, 429
ValueError exception, 87, 215
and lists, 40
and strings, 41
valuerefs() method, of WeakValueDictionary objects, 241
values() method
of Message objects, 552
of dicts, 45
of dicts in Python 3, 632
ValuesView abstract base class, 266
variable interpolation in strings, 163
variable keyword arguments in function definition, 95

variable number of arguments in function definition, 94
variables, 7
as names of objects, 35
binding and module imports, 145
binding of globals in functions, 98
class, 117-118
in nested functions, 97
iteration, 82
naming rules, 26
scope of, 96, 98
vars() function, 72, 211
verbose attribute, of sys.flags, 230
verify_request() method, of SocketServer class, 493
version attribute, of HTTPResponse objects, 504
version information, of interpreter, 231
version variable, sys module, 232
version_info variable, sys module, 232
vformat() method, of Formatter objects, 288
view objects, Python 3, 632
volume attribute, of ZipInfo objects, 327
vonmisesvariate() function, random module, 256

-W command line option, 216, 239-240
‘w’ mode, to open() function, 159
w(her) debugger command, pdb module, 189
wait() function, os module, 394
wait() method
ofAsyncResult objects, 425
of Condition objects, 441
of Event objects, 440
of Popen objects, 404
wait3() function, os module, 394
wait4() function, os module, 394
waitpid() function, os module, 394
walk() function, os module, 390
walk() method, of Message objects, 554
wall-clock time, obtaining, 191
warn() function, warnings module, 216, 239
warn_explicit() function, warnings module, 239
Warning warning, 216, 238
warning() method, of Logger objects, 357
warnings module, 238
warnings
  converting into exceptions, 239
difference from exceptions, 216
  suppression of, 238
warnoptions variable, sys module, 232
WatchedFileHandler class, logging module, 363
wave module, 588
WCOREDUMP() function, os module, 394
weak reference, 130
weak references, 240
WeakKeyDictionary class, weakref module, 241
weakref module, 130, 240
WeakValueDictionary class, weakref module, 241
web frameworks, 538
  and template strings, 164
web programming, 531
web server
  custom handling of requests, 510
  running standalone in Python, 508
webbrowser module, 544
weekday() method, of date objects, 338
weibullvariate() function, random module, 256
well known port numbers, 450
WEXITSTATUS() function, os module, 394
wfile attribute
  of BaseHTTPRequestHandler objects, 509
  of StreamRequestHandler objects, 491
whichdb module, 310
whichdb() function, dbm module, 310
while statement, 8, 82
whitespace variable, string module, 287
WIFCONTINUED() function, os module, 395
WIFEXITED() function, os module, 394
WIFSIGNALED() function, os module, 395
WIFSTOPPED() function, os module, 395
WinDLL() function, ctypes module, 612
Windows, 331
  accessing registry, 408
  accessing registry, 175
  creating a binary distribution with distutils, 153
  double-clicking on Python programs, 176
  drive letters in filenames, 399
  file locking, 373
  issuing message to event log, 362
  list of error codes, 346
  main program with multiprocessing module, 417
  per-user site directory, 177
  process fork with multiprocessing, 435
  running programs, 6
WindowsError exception, 215
Wing IDE, 5
winreg module, 408
winsound module, 588
winver variable, sys module, 232
with statement, 62, 89
  and exceptions, 23
  and locking, 89
decimal module, 248
  locking primitives, 442
work pools, of processes, 424
wrap_socket() function, ssl module, 486
wrapper functions, 95
  and closures, 100
  example of, 101
  in extension modules, 594
wrappers, and attribute binding methods, 132

How can we make this index more useful? Email us at indexes@samspublishing.com
wraps decorator, functools module, 114, 269

writable() method
  of IOBase objects, 350
  of dispatcher objects, 456
write() function, os module, 385
write() method
  of BufferWriter objects, 352
  of ConfigParser objects, 334
  of ElementTree objects, 575
  of FileIO objects, 350
  of StreamWriter objects, 278
  of TextIOWrapper objects, 353
  of ZipFile objects, 326
  of files, 10, 159-160
  of mmap objects, 372
  of ssl objects, 488
write_byte() method, of mmap objects, 372
writelines() method
  of IOBase objects, 350
  of StreamWriter objects, 278
  of files, 159-160
writepy() method, of ZipFile objects, 326
writer() function, csv module, 549
writerow() method
  of csv DictWriter objects, 550
  of csv writer objects, 549
writerows() method
  of csv DictWriter objects, 550
  of csv writer objects, 549
writestr() method, of ZipFile objects, 327
 writexml() method, of DOM Node objects, 573
writing to a file, 159
WSGI (Web Server Gateway Interface), 540

WSGI
  application specification, 540
  example of, 541
  integration with web frameworks, 543
  processing form fields, 541
  running a stand-alone server, 542
  running in CGI scripts, 542
  use of generators for I/O, 165
  validation of applications, 543

\x escape code, in strings, 28
-x command line option, 173-174
xdrlib module, 587
xml package, 568
XML() function, xml.etree.ElementTree module, 575

XML
  escaping and unescaping characters, 583
  example document, 569
  incremental parsing of large files, 579
  namespaces in ElementTree module, 578
  parsing, 568

XML-RPC, 524

XML-RPC server, multithreaded example, 494

XML-RPC
  example of, 529
  server customization, 530
xml.dom.minidom module, 570
xml.etree.ElementTree module, 573
xml.sax module, 580
xml.sax.saxutils module, 583

‘xmlcharrefreplace’ error handling, 166-167
XMLGenerator() function, xml.sax.saxutils module, 584

XMLID() function, xml.etree.ElementTree module, 575
xmlrpc package, 524
xmlrpc.client module, 524
xmlrpc.server module, 527
xmlrpclib module, 524
XMLRPCServer class, xmlrpc.server module, 527
xor() function, operator module, 274
__xor__() method, 60
XPATH, 569
xrange type, 38
xrange() compared to a list, 44
xrange() function, 17, 44, 211
and Python 3, 17, 44, 211
XSLT, 569

Y

Y2K handling, 407
year attribute, of date objects, 337
yield expressions, 20, 104
yield statement, 19, 53, 102
and context managers, 90
use with I/O, 164–165

Z

ZeroDivisionError exception, 87, 215
zfill() method, of strings, 44
.zip files
and modules, 147
decoding and encoding, 324
use as a code archive, 147
zip() function, 83, 211
and Python 3, 211
future_builtins module, 217
type-conversion example, 37
zipfile module, 324
ZipFile() function, zipfile module, 325
zipimport module, 586
ZipInfo() function, zipfile module, 325