



The Short Version

This is a minimal introduction to Python, based on my popular web tutorial, “Instant Python” (<http://hetland.org/writing/instant-python.html>). It targets programmers who already know a language or two, but who want to get up to speed with Python. For information on downloading and executing the Python interpreter, see Chapter 1.

The Basics

To get a basic feel for the Python language, think of it as pseudocode, because that’s pretty close to the truth. Variables don’t have types, so you don’t need to declare them. They appear when you assign to them, and disappear when you don’t use them anymore. Assignment is done with the = operator, like this:

```
x = 42
```

Note that equality is tested by the == operator.

You can assign several variables at once, like this:

```
x,y,z = 1,2,3
first, second = second, first
a = b = 123
```

Blocks are indicated through indentation, and *only* through indentation. (No begin/end or braces.) The following are some common control structures:

```
if x < 5 or (x > 10 and x < 20):
    print "The value is OK."
```

```
if x < 5 or 10 < x < 20:
    print "The value is OK."
```

```
for i in [1,2,3,4,5]:
    print "This is iteration number", i
```

```
x = 10
while x >= 0:
    print "x is still not negative."
    x = x-1
```

The first two examples are equivalent.

The index variable given in the for loop iterates through the elements of a list¹ (written with brackets, as in the example). To make an “ordinary” for loop (that is, a counting loop), use the built-in function `range`:

```
# Print out the values from 0 to 99, inclusive
for value in range(100):
    print value
```

The line beginning with # is a comment and is ignored by the interpreter.

Now you know enough (in theory) to implement any algorithm in Python. Let’s add some *basic* user interaction. To get input from the user (from a text prompt), use the built-in function `input`:

```
x = input("Please enter a number: ")
print "The square of that number is", x*x
```

The `input` function displays the (optional) prompt given and lets the user enter any valid Python value. In this case, we were expecting a number. If something else (such as a string) is entered, the program would halt with an error message. To avoid that, you would need to add some error checking. I won’t go into that here; suffice it to say that if you want the user input returned *verbatim* as a string (so that *anything* can be entered), use the function `raw_input` instead. If you wanted to convert an input string `s` to an integer, you could then use `int(s)`.

Note If you want to input a string with `input`, the user must write the quotes explicitly. In Python, strings can be enclosed in either single or double quotes. In Python 3.0, the original `input` disappears, and `raw_input` is renamed `input`. See Appendix D for more on Python 3.0.

So, you have control structures, input, and output covered—now you need some snazzy data structures. The most important ones are *lists* and *dictionaries*. Lists are written with brackets, and can (naturally) be nested:

```
name = ["Cleese", "John"]
x = [[1,2,3],[y,z],[[]]]
```

One of the nice things about lists is that you can access their elements separately or in groups, through *indexing* and *slicing*. Indexing is done (as in many other languages) by writing the index in brackets after the list. (Note that the first element has index 0.)

```
print name[1], name[0] # Prints "John Cleese"
name[0] = "Smith"
```

1. Or any other iterable object, actually.

Slicing is almost like indexing, except that you indicate both the start and stop index of the result, with a colon (:) separating them:

```
x = ["SPAM", "SPAM", "SPAM", "SPAM", "SPAM", "eggs", "and", "SPAM"]
print x[5:7] # Prints the list ["eggs", "and"]
```

Notice that the end is noninclusive. If one of the indices is dropped, it is assumed that you want everything in that direction. In other words, the slice `x[:3]` means “every element from the beginning of `x` up to element 3, noninclusive” (well, element 3 is actually the fourth element, because the counting starts at 0). The slice `x[3:]` would, on the other hand, mean “every element in `x`, starting at element 3 (inclusive) up to, and including, the last one.” For really interesting results, you can use negative numbers, too: `x[-3]` is the third element from the end of the list.

Now then, what about dictionaries? To put it simply, they are like lists, except that their contents aren’t ordered. How do you index them then? Well, every element has a *key*, or a *name*, which is used to look up the element, just as in a real dictionary. The following example demonstrates the syntax used to create dictionaries:

```
phone = { "Alice" : 23452532, "Boris" : 252336,
          "Clarice" : 2352525, "Doris" : 23624643 }

person = { 'first name': "Robin", 'last name': "Hood",
           'occupation': "Scoundrel" }
```

Now, to get person’s occupation, you use the expression `person["occupation"]`. If you wanted to change the person’s last name, you could write this:

```
person['last name'] = "of Locksley"
```

Simple, isn’t it? Like lists, dictionaries can hold other dictionaries, or lists, for that matter. And naturally, lists can hold dictionaries, too. That way, you can easily make some quite advanced data structures.

Functions

Our next step is abstraction. You want to give a name to a piece of code and call it with a couple of parameters. In other words, you want to define a *function* (also called a *procedure*). That’s easy. Use the keyword `def`, as follows:

```
def square(x):
    return x*x

print square(2) # Prints out 4
```

The return statement is used to return a value from the function.

When you pass a parameter to a function, you bind the parameter to the value, thus creating a new reference. This means that you can modify the original value directly inside the function, but if you make the parameter name refer to something else (rebind it), that change won't affect the original. This works just like in Java, for example. Let's take a look at an example:

```
def change(x):
    x[1] = 4

y = [1,2,3]
change(y)
print y # Prints out [1,4,3]
```

As you can see, the original list is passed in, and if the function modifies it, these modifications carry over to the place where the function was called. Note the behavior in the following example, however, where the function body *rebinds* the parameter:

```
def nochange(x):
    x = 0

y = 1
nochange(y)
print y # Prints out 1
```

Why doesn't *y* change now? Because you *don't change the value!* The value that is passed in is the number 1, and you can't change a number in the same way that you change a list. The number 1 is (and will always be) the number 1. What the example *does* change is what the parameter *x* *refers to*, and this does *not* carry over to the calling environment.

Python has all kinds of nifty things such as *named arguments* and *default arguments*, and can handle a variable number of arguments to a single function. For more information about this, see Chapter 6.

If you know how to use functions in general, what I've told you so far is basically what you need to know about them in Python.

It might be useful to know, however, that functions are *values* in Python. So if you have a function such as `square`, you could do something like the following:

```
queebble = square
print queebble(2) # Prints out 4
```

To call a function without arguments, you must remember to write `doit()` and not `doit`. The latter, as shown, only returns the function itself, as a value. This goes for methods in objects, too. Methods are described in the next section.

Objects and Stuff . . .

I assume you know how object-oriented programming works. Otherwise, this section might not make much sense. No problem—start playing without the objects, or check out Chapter 7.

In Python, you define classes with the (surprise!) `class` keyword, as follows:

```
class Basket:

    # Always remember the *self* argument
    def __init__(self, contents=None):
        self.contents = contents or []

    def add(self, element):
        self.contents.append(element)

    def print_me(self):
        result = ""
        for element in self.contents:
            result = result + " " + repr(element)
        print "Contains:" + result
```

Several things are worth noting in this example:

- Methods are called like this: `object.method(arg1, arg2)`.
- Some arguments can be *optional* and given a default value (as mentioned in the previous section on functions). This is done by writing the definition like this:

```
def spam(age=32): ...
```

- Here, `spam` can be called with one or zero parameters. If it's called without any parameters, `age` will have the default value of 32.
- `repr` converts an object to its string representation. (So if `element` contains the number 1, then `repr(element)` is the same as `"1"`, whereas `'element'` is a literal string.)

No methods or member variables (attributes) are protected (or private or the like) in Python. Encapsulation is pretty much a matter of programming style. (If you *really* need it, there are naming conventions that will allow some privacy, such as prefixing a name with a single or double underscore.)

Now, about that short-circuit logic . . .

All values in Python can be used as logic values. Some of the more empty ones (such as `False`, `[]`, `0`, `""`, and `None`) represent logical falsity; most other values (such as `True`, `[0]`, `1`, and `"Hello, world"`) represent logical truth.

Logical expressions such as `a and b` are evaluated like this:

- Check if `a` is true.
- If it is *not*, then simply return it.
- If it *is*, then simply return `b` (which will represent the truth value of the expression).

The corresponding logic for `a or b` is this:

- If `a` is true, then return it.
- If it isn't, then return `b`.

This short-circuit mechanism enables you to use `and` and `or` like the Boolean operators they are supposed to implement, but it also enables you to write short and sweet little conditional expressions. For example, this statement:

```
if a:
    print a
else:
    print b
```

could instead be written like this:

```
print a or b
```

Actually, this is somewhat of a Python idiom, so you might as well get used to it.

Note In Python 2.5, actual conditional expressions were introduced, so you could, in fact, write this:

```
print a if a else b
```

The `Basket` constructor (`Basket.__init__`) in the previous example uses this strategy in handling default parameters. The argument `contents` has a default value of `None` (which is, among other things, `false`); therefore, to check if it had a value, you could write this:

```
if contents:
    self.contents = contents
else:
    self.contents = []
```

Instead, the constructor uses this simple statement:

```
self.contents = contents or []
```

Why don't you give it the default value of `[]` in the first place? Because of the way Python works, this would give all the `Basket` instances the same empty list as default `contents`. As soon as one of them started to fill up, they all would contain the same elements, and the default would not be empty anymore. To learn more about this, see the discussion about the difference between *identity* and *equality* in Chapter 5.

Note When using `None` as a placeholder as done in the `Basket.__init__` method, using `contents is None` as the condition is safer than simply checking the argument's Boolean value. This will allow you to pass in a false value such as an empty list of your own (to which you could keep a reference outside the object).

If you would like to use an empty list as the default value, you can avoid the problem of sharing this among instances by doing the following:

```
def __init__(self, contents=[]):
    self.contents = contents[:]
```

Can you guess how this works? Instead of using the same empty list everywhere, you use the expression `contents[:]` to make a copy. (You simply slice the entire thing.)

So, to actually make a `Basket` and to use it (to call some methods on it), you would do something like this:

```
b = Basket(['apple', 'orange'])
b.add("lemon")
b.print_me()
```

This would print out the contents of the `Basket`: an apple, an orange, and a lemon.

There are magic methods other than `__init__`. One such method is `__str__`, which defines how the object wants to look if it is treated like a string. You could use this in the basket instead of `print_me`:

```
def __str__(self):
    result = ""
    for element in self.contents:
        result = result + " " + repr(element)
    return "Contains:" + result
```

Now, if you wanted to print the basket `b`, you could just use this:

```
print b
```

Cool, huh?

Subclassing works like this:

```
class SpamBasket(Basket):
    # ...
```

Python allows multiple inheritance, so you can have several superclasses in the parentheses, separated by commas. Classes are instantiated like this: `x = Basket()`. Constructors are, as I said, made by defining the special member function `__init__`. Let's say that `SpamBasket` had a constructor `__init__(self, type)`. Then you could make a spam basket like this: `y = SpamBasket("apples")`.

If in the constructor of `SpamBasket`, you needed to call the constructor of one or more superclasses, you could call it like this: `Basket.__init__(self)`. Note that in addition to supplying the ordinary parameters, you must explicitly supply `self`, because the superclass `__init__` doesn't know which instance it is dealing with.

For more about the wonders of object-oriented programming in Python, see Chapter 7.

Some Loose Ends

Here, I'll quickly review a few other useful things before ending this appendix. Most useful functions and classes are put in *modules*, which are really text files with the file name extension `.py` that contain Python code. You can import these and use them in your own programs. For example, to use the function `sqrt` from the standard module `math`, you can do either this:

```
import math
x = math.sqrt(y)
```

or this:

```
from math import sqrt
x = sqrt(y)
```

For more information on the standard library modules, see Chapter 10.

All the code in the module/script is run when it is imported. If you want your program to be both an importable module and a runnable program, you might want to add something like this at the end of it:

```
if __name__ == "__main__": main()
```

This is a magic way of saying that if this module is run as an executable script (that is, it is not being imported into another script), then the function `main` should be called. Of course, you could do anything after the colon there.

And for those of you who want to make an executable script in UNIX, use the following first line to make it run by itself:

```
#!/usr/bin/env python
```

Finally, a brief mention of an important concept: *exceptions*. Some operations (such as dividing something by zero or reading from a nonexistent file) produce an error condition or *exception*. You can even make your own exceptions and raise them at the appropriate times.

If nothing is done about the exception, your program ends and prints out an error message. You can avoid this with a `try/except` statement, as in this example:

```
def safe_division(a, b):
    try:
        return a/b
    except ZeroDivisionError: pass
```

`ZeroDivisionError` is a standard exception. In this case, you *could* have checked if `b` was zero, but in many cases, that strategy is not feasible. And besides, if you removed the `try/except` statement in `safe_division`, thereby making it a risky function to call (called something like `unsafe_division`), you could still do the following:

```
try:
    unsafe_division(a, b)
except ZeroDivisionError:
    print "Something was divided by zero in unsafe_division"
```


In cases in which you *typically* would not have a specific problem, but it *might* occur, using exceptions enables you to avoid costly testing and so forth.

Well, that's it. Hope you learned something. Now go and play. And remember the Python motto of learning: use the source (which basically means read all the code you can get your hands on).



Python Reference

This is not a full Python reference by far—you can find that in the standard Python documentation (<http://python.org/doc/ref>). Rather, this is a handy “cheat sheet” that can be useful for refreshing your memory as you start out programming in Python. See Appendix D for changes in the language that are introduced in version 3.0.

Expressions

This section summarizes Python expressions. Table B-1 lists the most important basic (literal) values in Python; Table B-2 lists the Python operators, along with their precedence (those with high precedence are evaluated before those with low precedence); Table B-3 describes some of the most important built-in functions; Tables B-4 through B-6 describe the list methods, dictionary methods, and string methods, respectively.

Table B-1. *Basic (Literal) Values*

Type	Description	Syntax Samples
Integer	Numbers without a fractional part	42
Long integer	Large integer numbers	42L
Float	Numbers with a fractional part	42.5, 42.5e-2
Complex	Sum of a real (integer or float) and imaginary number	38 + 4j, 42j
String	An immutable sequence of characters	'foo', "bar", ""baz"", r'\n'
Unicode	An immutable sequence of Unicode characters	u'foo', u"bar", u""baz""

Table B-2. *Operators*

Operator	Description	Precedence
lambda	Lambda expression	1
or	Logical or	2
and	Logical and	3

Continued

Table B-2. *Continued*

Operator	Description	Precedence
not	Logical negation	4
in	Membership test	5
not in	Negative membership test	5
is	Identity test	6
is not	Negative identity test	6
<	Less than	7
>	Greater than	7
<=	Less than or equal to	7
>=	Greater than or equal to	7
==	Equal to	7
!=	Not equal to	7
	Bitwise or	8
^	Bitwise exclusive or	9
&	Bitwise and	10
<<	Left shift	11
>>	Right shift	11
+	Addition	12
-	Subtraction	12
*	Multiplication	13
/	Division	13
%	Remainder	13
+	Unary identity	14
-	Unary negation	14
~	Bitwise complement	15
**	Exponentiation	16
x.attribute	Attribute reference	17
x[index]	Item access	18
x[index1:index2[:index3]]	Slicing	19
f(args...)	Function call	20
(...)	Parenthesized expression or tuple display	21

Operator	Description	Precedence
[...]	List display	22
{key:value, ...}	Dictionary display	23
`expressions...`	String conversion	24

Table B-3. *Some Important Built-in Functions*

Function	Description
<code>abs(number)</code>	Returns the absolute value of a number.
<code>apply(function[, args[, kwds]])</code>	Calls a given function, optionally with parameters.
<code>all(iterable)</code>	Returns True if all the elements of iterable are true; otherwise, it returns False.
<code>any(iterable)</code>	Returns True if any of the elements of iterable are true; otherwise, it returns False.
<code>basestring()</code>	An abstract superclass for <code>str</code> and <code>unicode</code> , usable for type checking.
<code>bool(object)</code>	Returns True or False, depending on the Boolean value of object.
<code>callable(object)</code>	Checks whether an object is callable.
<code>chr(number)</code>	Returns a character whose ASCII code is the given number.
<code>classmethod(func)</code>	Creates a class method from an instance method (see Chapter 7).
<code>cmp(x, y)</code>	Compares <code>x</code> and <code>y</code> . If <code>x < y</code> , it returns a negative number; if <code>x > y</code> , it returns a positive number; and if <code>x == y</code> , it returns zero.
<code>complex(real[, imag])</code>	Returns a complex number with the given real (and, optionally, imaginary) component.
<code>delattr(object, name)</code>	Deletes the given attribute from the given object.
<code>dict([mapping-or-sequence])</code>	Constructs a dictionary, optionally from another mapping or a list of (key, value) pairs. May also be called with keyword arguments.
<code>dir([object])</code>	Lists (most of) the names in the currently visible scopes, or optionally (most of) the attributes of the given object.
<code>divmod(a, b)</code>	Returns <code>(a//b, a%b)</code> (with some special rules for floats).

Continued

Table B-3. *Continued*

Function	Description
<code>enumerate(iterable)</code>	Iterates over (index, item) pairs, for all items in iterable.
<code>eval(string[, globals[, locals]])</code>	Evaluates a string containing an expression, optionally in a given global and local scope.
<code>execfile(file[, globals[, locals]])</code>	Executes a Python file, optionally in a given global and local scope.
<code>file(filename[, mode[, bufsize]])</code>	Creates a file object with a given file name, optionally with a given mode and buffer size.
<code>filter(function, sequence)</code>	Returns a list of the elements from the given sequence for which function returns true.
<code>float(object)</code>	Converts a string or number to a float.
<code>frozenset([iterable])</code>	Creates a set that is immutable, which means it can be added to other sets.
<code>getattr(object, name[, default])</code>	Returns the value of the named attribute of the given object, optionally with a given default value.
<code>globals()</code>	Returns a dictionary representing the current global scope.
<code>hasattr(object, name)</code>	Checks whether the given object has the named attribute.
<code>help([object])</code>	Invokes the built-in help system, or prints a help message about the given object.
<code>hex(number)</code>	Converts a number to a hexadecimal string.
<code>id(object)</code>	Returns the unique ID for the given object.
<code>input([prompt])</code>	Equivalent to <code>eval(raw_input(prompt))</code> .
<code>int(object[, radix])</code>	Converts a string or number (optionally with a given radix) or number to an integer.
<code>isinstance(object, classinfo)</code>	Checks whether the given object is an instance of the given classinfo value, which may be a class object, a type object, or a tuple of class and type objects.
<code>issubclass(class1, class2)</code>	Checks whether class1 is a subclass of class2 (every class is a subclass of itself).
<code>iter(object[, sentinel])</code>	Returns an iterator object, which is object. <code>__iter__()</code> , an iterator constructed for iterating a sequence (if object supports <code>__getitem__</code>), or, if sentinel is supplied, an iterator that keeps calling object in each iteration until sentinel is returned.
<code>len(object)</code>	Returns the length (number of items) of the given object.

Function	Description
<code>list([sequence])</code>	Constructs a list, optionally with the same items as the supplied sequence.
<code>locals()</code>	Returns a dictionary representing the current local scope (do not modify this dictionary).
<code>long(object[, radix])</code>	Converts a string (optionally with a given radix) or number to a long integer.
<code>map(function, sequence, ...)</code>	Creates a list consisting of the values returned by the given function when applying it to the items of the supplied sequence(s).
<code>max(object1, [object2, ...])</code>	If <code>object1</code> is a nonempty sequence, the largest element is returned; otherwise, the largest of the supplied arguments (<code>object1, object2, ...</code>) is returned.
<code>min(object1, [object2, ...])</code>	If <code>object1</code> is a nonempty sequence, the smallest element is returned; otherwise, the smallest of the supplied arguments (<code>object1, object2, ...</code>) is returned.
<code>object()</code>	Returns an instance of <code>object</code> , the base class for all new style classes.
<code>oct(number)</code>	Converts an integer number to an octal string.
<code>open(filename[, mode[, bufsize]])</code>	An alias for <code>file</code> (use <code>open</code> , not <code>file</code> , when opening files).
<code>ord(char)</code>	Returns the ASCII value of a single character (a string or Unicode string of length 1).
<code>pow(x, y[, z])</code>	Returns <code>x</code> to the power of <code>y</code> , optionally modulo <code>z</code> .
<code>property([fget[, fset[, fdel[, doc]]]])</code>	Creates a property from a set of accessors (see Chapter 9).
<code>range([start,]stop[, step])</code>	Returns a numeric range (as a list) with the given <code>start</code> (inclusive, default 0), <code>stop</code> (exclusive), and <code>step</code> (default 1).
<code>raw_input([prompt])</code>	Returns data input by the user as a string, optionally using a given prompt.
<code>reduce(function, sequence[, initializer])</code>	Applies the given function cumulatively to the items of the sequence, using the cumulative result as the first argument and the items as the second argument, optionally with a start value (<code>initializer</code>).
<code>reload(module)</code>	Reloads an already loaded module and returns it.
<code>repr(object)</code>	Returns a string representation of the object, often usable as an argument to <code>eval</code> .
<code>reversed(sequence)</code>	Returns a reverse iterator over the sequence.

Continued

Table B-3. *Continued*

Function	Description
<code>round(float[, n])</code>	Rounds off the given float to <i>n</i> digits after the decimal point (default zero).
<code>set([iterable])</code>	Returns a set whose elements are taken from <i>iterable</i> (if given).
<code>setattr(object, name, value)</code>	Sets the named attribute of the given object to the given value.
<code>sorted(iterable[, cmp][, key][, reverse])</code>	Returns a new sorted list from the items in <i>iterable</i> . Optional parameters are the same as for the list method <code>sort</code> .
<code>staticmethod(func)</code>	Creates a static (class) method from an instance method (see Chapter 7).
<code>str(object)</code>	Returns a nicely formatted string representation of the given object.
<code>sum(seq[, start])</code>	Returns the sum of a sequence of numbers, added to the optional parameter <i>start</i> (default 0).
<code>super(type[, obj/type])</code>	Returns the superclass of the given type (optionally instantiated).
<code>tuple([sequence])</code>	Constructs a tuple, optionally with the same items as the supplied sequence.
<code>type(object)</code>	Returns the type of the given object.
<code>type(name, bases, dict)</code>	Returns a new type object with the given name, bases, and scope.
<code>unichr(number)</code>	The Unicode version of <code>chr</code> .
<code>unicode(object[, encoding[, errors]])</code>	Returns a Unicode encoding of the given object, possibly with a given encoding, and a given mode for handling errors ('strict', 'replace', or 'ignore'; 'strict' is the default).
<code>vars([object])</code>	Returns a dictionary representing the local scope, or a dictionary corresponding to the attributes of the given object (do not modify the returned dictionary, as the result of such a modification is not defined by the language reference).
<code>xrange([start,]stop[, step])</code>	Similar to <code>range</code> , but the returned object uses less memory, and should be used only for iteration.
<code>zip(sequence1, ...)</code>	Returns a list of tuples, where each tuple contains an item from each of the supplied sequences. The returned list has the same length as the shortest of the supplied sequences.

Table B-4. *List Methods*

Method	Description
<code>aList.append(obj)</code>	Equivalent to <code>aList[len(aList):len(aList)] = [obj]</code> .
<code>aList.count(obj)</code>	Returns the number of indices <code>i</code> for which <code>aList[i] == obj</code> .
<code>aList.extend(sequence)</code>	Equivalent to <code>aList[len(aList):len(aList)] = sequence</code> .
<code>aList.index(obj)</code>	Returns the smallest <code>i</code> for which <code>aList[i] == obj</code> (or raises a <code>ValueError</code> if no such <code>i</code> exists).
<code>aList.insert(index, obj)</code>	Equivalent to <code>aList[index:index] = [obj]</code> if <code>index >= 0</code> ; if <code>index < 0</code> , object is prepended to the list.
<code>aList.pop([index])</code>	Removes and returns the item with the given index (default <code>-1</code>).
<code>aList.remove(obj)</code>	Equivalent to <code>del aList[aList.index(obj)]</code> .
<code>aList.reverse()</code>	Reverses the items of <code>aList</code> in place.
<code>aList.sort([cmp][, key][, reverse])</code>	Sorts the items of <code>aList</code> in place (stable sorting). Can be customized by supplying a comparison function, <code>cmp</code> ; a key function, <code>key</code> , which will create the keys for the sorting; and a reverse flag (a Boolean value).

Table B-5. *Dictionary Methods*

Method	Description
<code>aDict.clear()</code>	Removes all the items of <code>aDict</code> .
<code>aDict.copy()</code>	Returns a copy of <code>aDict</code> .
<code>aDict.fromkeys(seq[, val])</code>	Returns a dictionary with keys from <code>seq</code> and values set to <code>val</code> (default <code>None</code>). May be called directly on the dictionary type, <code>dict</code> , as a class method.
<code>aDict.get(key[, default])</code>	Returns <code>aDict[key]</code> if it exists; otherwise, it returns the given default value (default <code>None</code>).
<code>aDict.has_key(key)</code>	Checks whether <code>aDict</code> has the given key.
<code>aDict.items()</code>	Returns a list of (<code>key</code> , <code>value</code>) pairs representing the items of <code>aDict</code> .
<code>aDict.iteritems()</code>	Returns an iterable object over the same (<code>key</code> , <code>value</code>) pairs as returned by <code>aDict.items()</code> .
<code>aDict.iterkeys()</code>	Returns an iterable object over the keys of <code>aDict</code> .
<code>aDict.itervalues()</code>	Returns an iterable object over the values of <code>aDict</code> .
<code>aDict.keys()</code>	Returns a list of the keys of <code>aDict</code> .

Continued

Table B-5. *Continued*

Method	Description
<code>aDict.pop(key[, d])</code>	Removes and returns the value corresponding to the given key, or the given default, <code>d</code> .
<code>aDict.popitem()</code>	Removes an arbitrary item from <code>aDict</code> and returns it as a (key, value) pair.
<code>aDict.setdefault(key[, default])</code>	Returns <code>aDict[key]</code> if it exists; otherwise, it returns the given default value (default <code>None</code>) and binds <code>aDict[key]</code> to it.
<code>aDict.update(other)</code>	For each item in <code>other</code> , adds the item to <code>aDict</code> (possibly overwriting existing items). Can also be called with arguments similar to the dictionary constructor, <code>aDict</code> .
<code>aDict.values()</code>	Returns a list of the values in <code>aDict</code> (possibly containing duplicates).

Table B-6. *String Methods*

Method	Description
<code>string.capitalize()</code>	Returns a copy of the string in which the first character is capitalized.
<code>string.center(width[, fillchar])</code>	Returns a string of length <code>max(len(string), width)</code> in which a copy of <code>string</code> is centered, padded with <code>fillchar</code> (the default is space characters).
<code>string.count(sub[, start[, end]])</code>	Counts the occurrences of the substring <code>sub</code> , optionally restricting the search to <code>string[start:end]</code> .
<code>string.decode([encoding[, errors]])</code>	Returns decoded version of the string using the given encoding, handling errors as specified by errors ('strict', 'ignore', or 'replace').
<code>string.encode([encoding[, errors]])</code>	Returns the encoded version of the string using the given encoding, handling errors as specified by errors ('strict', 'ignore', or 'replace').
<code>string.endswith(suffix[, start[, end]])</code>	Checks whether <code>string</code> ends with <code>suffix</code> , optionally restricting the matching with the given indices <code>start</code> and <code>end</code> .
<code>string.expandtabs([tabsize])</code>	Returns a copy of the string in which tab characters have been expanded using spaces, optionally using the given <code>tabsize</code> (default 8).
<code>string.find(sub[, start[, end]])</code>	Returns the first index where the substring <code>sub</code> is found, or <code>-1</code> if no such index exists, optionally restricting the search to <code>string[start:end]</code> .

Method	Description
<code>string.index(sub[, start[, end]])</code>	Returns the first index where the substring <code>sub</code> is found, or raises a <code>ValueError</code> if no such index exists, optionally restricting the search to <code>string[start:end]</code> .
<code>string.isalnum()</code>	Checks whether the string consists of alphanumeric characters.
<code>string.isalpha()</code>	Checks whether the string consists of alphabetic characters.
<code>string.isdigit()</code>	Checks whether the string consists of digits.
<code>string.islower()</code>	Checks whether all the case-based characters (letters) of the string are lowercase.
<code>string.isspace()</code>	Checks whether the string consists of whitespace.
<code>string.istitle()</code>	Checks whether all the case-based characters in the string following non-case-based letters are uppercase and all other case-based characters are lowercase.
<code>string.isupper()</code>	Checks whether all the case-based characters of the string are uppercase.
<code>string.join(sequence)</code>	Returns a string in which the string elements of <code>sequence</code> have been joined by <code>string</code> .
<code>string.ljust(width[, fillchar])</code>	Returns a string of length <code>max(len(string), width)</code> in which a copy of <code>string</code> is left-justified, padded with <code>fillchar</code> (the default is space characters).
<code>string.lower()</code>	Returns a copy of the string in which all case-based characters have been lowercased.
<code>string.lstrip([chars])</code>	Returns a copy of the string in which all <code>chars</code> have been stripped from the beginning of the string (the default is all whitespace characters, such as spaces, tabs, and newlines).
<code>string.partition(sep)</code>	Searches for <code>sep</code> in the string and returns (<code>head</code> , <code>sep</code> , <code>tail</code>).
<code>string.replace(old, new[, max])</code>	Returns a copy of the string in which the occurrences of <code>old</code> have been replaced with <code>new</code> , optionally restricting the number of replacements to <code>max</code> .
<code>string.rfind(sub[, start[, end]])</code>	Returns the last index where the substring <code>sub</code> is found, or <code>-1</code> if no such index exists, optionally restricting the search to <code>string[start:end]</code> .
<code>string.rindex(sub[, start[, end]])</code>	Returns the last index where the substring <code>sub</code> is found, or raises a <code>ValueError</code> if no such index exists, optionally restricting the search to <code>string[start:end]</code> .

Continued

Table B-6. *Continued*

Method	Description
<code>string.rjust(width[, fillchar])</code>	Returns a string of length <code>max(len(string), width)</code> in which a copy of <code>string</code> is right-justified, padded with <code>fillchar</code> (the default is space characters).
<code>string.rpartition(sep)</code>	Same as <code>partition</code> , but searches from the right.
<code>string.rstrip([chars])</code>	Returns a copy of the string in which all chars have been stripped from the end of the string (the default is all whitespace characters, such as spaces, tabs, and newlines).
<code>string.rsplit([sep[, maxsplit]])</code>	Same as <code>split</code> , but when using <code>maxsplit</code> , counts from right to left.
<code>string.split([sep[, maxsplit]])</code>	Returns a list of all the words in the string, using <code>sep</code> as the separator (splits on all whitespace if left unspecified), optionally limiting the number of splits to <code>maxsplit</code> .
<code>string.splitlines([keepends])</code>	Returns a list with all the lines in <code>string</code> , optionally including the line breaks (if <code>keepends</code> is supplied and is true).
<code>string.startswith(prefix[, start[, end]])</code>	Checks whether <code>string</code> starts with <code>prefix</code> , optionally restricting the matching with the given indices <code>start</code> and <code>end</code> .
<code>string.strip([chars])</code>	Returns a copy of the string in which all chars have been stripped from the beginning and the end of the string (the default is all whitespace characters, such as spaces, tabs, and newlines).
<code>string.swapcase()</code>	Returns a copy of the string in which all the case-based characters have had their case swapped.
<code>string.title()</code>	Returns a copy of the string in which all the words are capitalized.
<code>string.translate(table[, deletechars])</code>	Returns a copy of the string in which all characters have been translated using <code>table</code> (constructed with the <code>maketrans</code> function in the <code>string</code> module), optionally deleting all characters found in the string <code>deletechars</code> .
<code>string.upper()</code>	Returns a copy of the string in which all the case-based characters have been uppercased.
<code>string.zfill(width)</code>	Pads <code>string</code> on the left with zeros to fill <code>width</code> .

Statements

This section gives you a quick summary of each of the statement types in Python.

Simple Statements

Simple statements consist of a single (logical) line.

Expression Statements

Expressions can be statements on their own. This is especially useful if the expression is a function call or a documentation string.

Example:

```
"This module contains SPAM-related functions."
```

Assert Statements

Assert statements check whether a condition is true and raise an `AssertionError` (optionally with a supplied error message) if it isn't.

Example:

```
assert age >= 12, 'Children under the age of 12 are not allowed'
```

Assignment Statements

Assignment statements bind variables to values. Multiple variables may be assigned to simultaneously (through sequence unpacking) and assignments may be chained.

Examples:

```
x = 42                                # Simple assignment
name, age = 'Gumby', 60               # Sequence unpacking
x = y = z = 10                        # Chained assignments
```

Augmented Assignment Statements

Assignments may be augmented by operators. The operator will then be applied to the existing value of the variable and the new value, and the variable will be rebound to the result. If the original value is mutable, it may be modified instead (with the variable staying bound to the original).

Examples:

```
x *= 2                                # Doubles x
x += 5                                # Adds 5 to x
```

The pass Statement

The pass statement is a “no-op,” which does nothing. It is useful as a placeholder, or as the only statement in syntactically required blocks where you want no action to be performed.

Example:

```
try: x.name
except AttributeError: pass
else: print 'Hello', x.name
```

The del Statement

The del statement unbinds variables and attributes, and removes parts (positions, slices, or slots) from data structures (mappings or sequences). It cannot be used to delete values directly, because values are only deleted through garbage collection.

Examples:

```
del x                # Unbinds a variable
del seq[42]          # Deletes a sequence element
del seq[42:]          # Deletes a sequence slice
del map['foo']        # Deletes a mapping item
```

The print Statement

The print statement writes one or more values (automatically formatted with str, separated by single spaces) to a given stream, with sys.stdout being the default. It adds a line break to the end of the written string unless the print statement ends with a comma.

Examples:

```
print 'Hello, world!' # Writes 'Hello, world\n' to sys.stdout
print 1, 2, 3          # Writes '1 2 3\n' to sys.stdout
print >> somefile, 'xyz' # Writes 'xyz' to somefile
print 42,              # Writes '42 ' to sys.stdout
```

The return Statement

The return statement halts the execution of a function and returns a value. If no value is supplied, None is returned.

Examples:

```
return                # Returns None from the current function
return 42              # Returns 42 from the current function
return 1, 2, 3         # Returns (1, 2, 3) from the current function
```

The yield Statement

The yield statement temporarily halts the execution of a generator and yields a value. A generator is a form of iterator and can be used in for loops, among other things.

Example:

```
yield 42              # Returns 42 from the current function
```

The raise Statement

The raise statement raises an exception. It may be used without any arguments (inside an except clause, to re-raise the currently caught exception), with a subclass of Exception and an optional argument (in which case, an instance is constructed), or with an instance of a subclass of Exception.

Examples:

```
raise                                # May only be used inside except clauses
raise IndexError
raise IndexError, 'index out of bounds'
raise IndexError('index out of bounds')
```

The break Statement

The break statement ends the immediately enclosing loop statement (for or while) and continues execution immediately after that loop statement.

Example:

```
while True:
    line = file.readline()
    if not line: break
    print line
```

The continue Statement

The continue statement is similar to the break statement in that it halts the current iteration of the immediately enclosing loop, but instead of ending the loop completely, it continues execution at the beginning of the next iteration.

Example:

```
while True:
    line = file.readline()
    if not line: break
    if line.isspace(): continue
    print line
```

The import Statement

The import statement is used to import names (variables bound to functions, classes, or other values) from an external module. This also covers from __future__ import ... statements for features that will become standard in future versions of Python.

Examples:

```
import math
from math import sqrt
from math import sqrt as squareroot
from math import *
```

The global Statement

The `global` statement is used to mark a variable as global. It is used in functions to allow statements in the function body to rebind global variables. Using the `global` statement is generally considered poor style and should be avoided whenever possible.

Example:

```
count = 1
def inc():
    global count
    count += 1
```

The exec Statement

The `exec` statement is used to execute strings containing Python statements, optionally with a given global and local namespace (dictionaries).

Examples:

```
exec 'print "Hello, world!'"
exec 'x = 2' in myglobals, mylocals # ... where myglobals and mylocals are dicts
```

Compound Statements

Compound statements contain groups (blocks) of other statements.

The if Statement

The `if` statement is used for conditional execution, and it may include `elif` and `else` clauses.

Example:

```
if x < 10:
    print 'Less than ten'
elif 10 <= x < 20:
    print 'Less than twenty'
else:
    print 'Twenty or more'
```

The while Statement

The `while` statement is used for repeated execution (looping) while a given condition is true. It may include an `else` clause (which is executed if the loop finishes normally, without any `break` or `return` statements, for instance).

Example:

```
x = 1
while x < 100:
    x *= 2
print x
```

The for Statement

The for statement is used for repeated execution (looping) over the elements of sequences or other iterable objects (objects having an `__iter__` method that returns an iterator). It may include an else clause (which is executed if the loop finishes normally, without any break or return statements, for instance).

Example:

```
for i in range(10, 0, -1):
    print i
print 'Ignition!'
```

The try Statement

The try statement is used to enclose pieces of code where one or more known exceptions may occur, and enables your program to trap these exceptions and perform exception-handling code if an exception is trapped. The try statement can combine several except clauses (handling exceptional circumstances) and finally clauses (executed no matter what; useful for cleanup).

Example:

```
try:
    1/0
except ZeroDivisionError:
    print "Can't divide anything by zero."
finally:
    print "Done trying to calculate 1/0"
```

The with Statement

The with statement is used to wrap a block of code using a so-called context manager, allowing the context manager to perform some setup and cleanup actions. For example, files can be used as context managers, and they will close themselves as part of the cleanup.

Note In Python 2.5, you need `from __future__ import with_statement` for the with statement to work as described.

Example:

```
with open("somefile.txt") as myfile:
    dosomething(myfile)
# The file will have been closed here
```


Function Definitions

Function definitions are used to create function objects and to bind global or local variables to these function objects.

Example:

```
def double(x):  
    return x*2
```

Class Definitions

Class definitions are used to create class objects and to bind global or local variables to these class objects.

Example:

```
class Doubler:  
    def __init__(self, value):  
        self.value = value  
    def double(self):  
        self.value *= 2
```



Online Resources

As you learn Python, the Internet will serve as an invaluable resource. This appendix describes some of the web sites that may be of interest to you as you are starting out. If you are looking for something Python-related that isn't described here, I suggest that you first check the official Python web site (<http://python.org>), and then use your favorite web search engine, or the other way around. There is a lot of information about Python online; chances are you'll find something. If you don't, you can always try `comp.lang.python` (described in this appendix). If you're an IRC user (see <http://irchelp.org> for information), you might want to check out the `#python` channel on `irc.freenode.net`.

Python Distributions

Several Python distributions are available. Here are some of the more prominent ones:

Official Python distribution (<http://python.org/download>): This comes with a default integrated development environment called IDLE (for more information, see <http://docs.python.org/lib/idle.html>).

ActivePython (<http://activestate.com>): This is ActiveState's Python distribution, which includes several nonstandard packages in addition to the official distribution. This is also the home of Visual Python, a Python plug-in for Visual Studio .NET.

Jython (<http://www.jython.org>): Jython is the Java implementation of Python.

IronPython (<http://www.codeplex.com/Wiki/View.aspx?ProjectName=IronPython>): IronPython is the C# implementation of Python.

MacPython (<http://homepages.cwi.nl/~jack/macpython/index.html>): MacPython is the Macintosh port of Python for older versions of Mac OS. The new Mac version can be found on the main Python site (<http://python.org>). You can also get Python through MacPorts (<http://macports.org>).

pywin32 (<http://sf.net/projects/pywin32/>): These are the Python for Windows extensions. If you have ActivePython installed, you already have all these extensions.

Python Documentation

Answers to most of your Python questions are most likely somewhere on the python.org web site. The documentation can be found at <http://python.org/doc>, with the following subdivisions:

Python Tutorial (<http://python.org/doc/tut>): This is a relatively simple introduction to the language.

Python Reference Manual (<http://python.org/doc/ref>): This document contains a precise definition of the Python language. It may not be the place to start when learning Python, but it contains precise answers to most questions you might have about the language.

Python Library Reference (<http://python.org/doc/lib>): This is probably the most useful piece of Python documentation you'll ever find. It describes all (or most) of the modules in the standard Python library. If you are wondering how to solve a problem in Python, this should be the first place you look—perhaps the solution already exists in the libraries.

Extending and Embedding the Python Interpreter (<http://python.org/doc/ext>): This is a document that describes how to write Python extension modules in the C language, and how to use the Python interpreter as a part of larger C programs. (Python itself is implemented in C.)

Macintosh Library Modules (<http://python.org/doc/mac>): This document describes functionality specific to the Macintosh port of Python.

Python/C API Reference Manual (<http://python.org/doc/api>): This is a rather technical document describing the details of the Python/C application programming interface (API), which enables C programs to interface with the Python interpreter.

Two other useful documentation resources are Python Documentation Online (<http://pydoc.org>) and `pyhelp.cgi` (<http://starship.python.net/crew/theller/pyhelp.cgi>), which allow you to search the standard Python documentation. If you want some “recipes” and solutions provided by the Python community, the Python Cookbook (<http://aspn.activestate.com/ASPN/Python/Cookbook>) is a good place to look.

The future of Python is decided by the language's Benevolent Dictator For Life (BDFL), Guido van Rossum, but his decisions are guided and informed by so-called Python Enhancement Proposals, which may be accessed at <http://python.org/dev/peps>. Various HOWTO documents (relatively specific tutorials) can be found at <http://python.org/doc/howto>.

Useful Toolkits and Modules

One source for finding software implemented in Python (including useful toolkits and modules you can use in your own programs) is the Vaults of Parnassus (<http://www.vex.net/parnassus>); another is the Python Package Index (<http://pypi.python.org/pypi>). If you can't find what you're looking for on either of these sites, try a standard web search, or perhaps take a look at freshmeat (<http://freshmeat.net>) or SourceForge (<http://sf.net>).

Table C-1 lists the URLs of some of the most well-known GUI toolkits available for Python. For a more thorough description, see Chapter 12. Table C-2 lists the URLs of the third-party packages used in the ten projects (Chapters 20–29).

Table C-1. *Some Well-Known GUI Toolkits for Python*

Toolkit	URL
Tkinter	http://python.org/topics/tkinter/doc.html
wxPython	http://www.wxpython.org
PythonWin	http://sf.net/projects/pywin32/
Java Swing	http://java.sun.com/docs/books/tutorial/uiswing
PyGTK	http://www.pygtk.org
PyQt	http://www.thekompany.com/projects/pykde

Table C-2. *The Third-Party Modules Used in This Book's Ten Projects*

Package	URL
Psycopg	http://initd.org/pub/software/psycopg/
MySQLdb	http://sourceforge.net/projects/mysql-python
Pygame	http://www.pygame.org
PyXML	http://sourceforge.net/projects/pyxml
ReportLab	http://www.reportlab.org

Newsgroups, Mailing Lists, and Blogs

An important forum for Python discussion is the Usenet group `comp.lang.python`. If you're serious about Python, skimming this group regularly can be quite useful. Its companion group, `comp.lang.python.announce`, contains announcements about new Python software (including new Python distributions, Python extensions, and software written using Python).

Several official mailing lists are available. For instance, the `comp.lang.python` group is mirrored in the `python-list@python.org` mailing list. If you have a Python problem and need help, simply send an email to `help@python.org` (assuming that you've exhausted all other options, of course). For learning about programming in Python, the tutor list (`tutor@python.org`) may be useful. For information about how to join these (and other) mailing lists, see <http://mail.python.org/mailman/listinfo>.

A couple of useful blogs are Unofficial Planet Python (<http://planetpython.org>) and The Daily Python-URL (<http://pythonware.com/daily>).



Python 3.0

This book describes mainly the language defined by Python version 2.5. Python version 3.0 (and its companion “transition” release, 2.6) isn’t all that different. Most things work just as they did before, but the language cleanups introduced mean that some existing code will break.

If you’re transitioning from older code to Python 3.0, a couple of tools can come in quite handy. First, Python 2.6 comes with optional warnings about 3.0 incompatibilities (run Python with the `-3` flag). If you first make sure your code runs without errors in 2.6 (which is largely backward-compatible), you can refactor away any incompatibility warnings. (Needless to say, you should have solid unit tests in place before you do this; see Chapter 16 for more advice on testing.) Second, Python 3.0 ships with an automatic refactoring tool called `2to3`, which can automatically upgrade your source files. (Be sure to back up or check in your files before performing any large-scale transformations.) If you wish to have both 2.6 and 3.0 code available, you could keep working on the 2.6 code (with the proper warnings turned on), and generate 3.0 code when it’s time for releasing.

Throughout the book, you’ll find notes about things that change in Python 3.0. This appendix gives a more comprehensive set of pointers for moving to the world of 3.0. I’ll describe some of the more noticeable changes, but not everything that is new in Python 3.0. There are many changes, both major and minor. Table D-1 (which is based on the document *What’s New in Python 3.0?*, by Guido van Rossum), at the end of this appendix, lists quite a few more changes and also refers to relevant PEP documents, when applicable (available from <http://python.org/dev/peps>). Table D-2 lists some other sources of further information.

Strings and I/O

The following sections deal with new features related to text. Strings are no longer simply byte sequences (although such sequences are still available), the `input/print` pair has been revamped slightly, and string formatting has had a major facelift.

Strings, Bytes, and Encodings

The distinction between text and byte sequences is significantly cleaned up in Python 3.0. Strings in previous versions were based on the somewhat outmoded (yet still prevalent) notion that text characters can easily be represented as single bytes. While this is true for English and most western languages, it fails to account for ideographic scripts, such as Chinese.

The Unicode standard was created to encompass all written languages, and it admits about 100,000 different characters, each of which has a unique numeric code. In Python 3.0, `str` is, in fact, the `unicode` type from earlier versions, which is a sequence of Unicode characters. As there is no unique way of encoding these into byte sequences (which you need to do in order to perform disk I/O, for example), you must supply an encoding (with UTF-8 as the default in most cases). So, text files are now assumed to be encoded versions of Unicode, rather than simply arbitrary sequences of bytes. (Binary files are still just byte sequences, though.) As a consequence of this, constants such as `string.letters` have been given the prefix `ascii_` (for example, `string.ascii_letters`) to make the link to a specific encoding clear.

To avoid losing the old functionality of the previous `str` class, there is a new class called `bytes`, which represents immutable sequences of bytes (as well as `bytearray`, which is its mutable sibling).

Console I/O

There is little reason to single out console printing to the degree that it has its own statement. Therefore, the `print` statement is changed into a function. It still works in a manner very similar to the original statement (for example, you can print several arguments by separating them with commas), but the stream redirection functionality is now a keyword argument. In other words, instead of writing this:

```
print >> sys.stderr, "fatal error:", error
```

you would write this:

```
print("fatal error:", error, file=sys.stderr)
```

Also, the behavior of the original `input` no longer has its own function. The name `input` is now used for what used to be `raw_input`, and you need to explicitly say `eval(input())` to get the old functionality.

New String Formatting

Strings now have a new method, called `format`, which allows you to perform rather advanced string formatting. The fields in the string where values are to be spliced in are enclosed in braces, rather than prefaced with a `%` (and braces are escaped by using double braces). The replacement fields refer to the arguments of the `format` method, either by numbers (for positional arguments) or names (for keyword arguments):

```
>>> "{0}, {1}, {x}".format("a", 1, x=42)
'a 1 42'
```

In addition, the replacement fields can access attributes and elements of the values to be replaced, such as in `"{foo.bar}"` or `"{foo[bar]}"`, and can be modified by format specifiers similar to those in the current system. This new mechanism is quite flexible, and because it allows classes to specify their own format string behavior (through the magic `__format__` method), you will be able to write much more elegant output formatting code.

Classes and Functions

Although none of the changes are quite as fundamental as the introduction of new-style classes, Python 3 has some goodies in store in the abstraction department: functions can now be annotated with information about parameters and return values, there is a framework for abstract base classes, metaclasses have a more convenient syntax, and you can have keyword-only parameters and nonlocal (but not global) variables.

Function Annotation

The new function annotation system is something of a wildcard. It allows you to annotate the arguments and the return type of a function (or method) with the values of arbitrary expressions, and then to retrieve these values later. However, what this system is to be used for is not specified. It is motivated by several practical applications (such as more fine-grained docstring functionality, type specifications and checking, generic functions, and more), but you can basically use it for anything you like.

A function is annotated as follows:

```
def frozzbozz(x: foo, y: bar = 42) -> baz:
    pass
```

Here, `foo`, `bar`, and `baz` are annotations for the positional argument `x`, the keyword argument `y`, and the return value of `frozzbozz`, respectively. These can be retrieved from the dictionary `frozzbozz.func_annotations`, with the parameter names (or "return" for the return value) as keys.

Abstract Base Classes

Sometimes you might want to implement only *parts* of a class. For example, you may have functionality that is to be shared among several classes, so you put it in a superclass. However, the superclass isn't really complete and shouldn't be instantiated by itself—it's only there for others to inherit. This is called an *abstract base class* (or simply an *abstract class*). It's quite common for such abstract classes to define nonfunctional methods that the subclasses need to override. In this way, the base class also acts as an interface definition, in a way.

You can certainly simulate this with older Python versions (for example, by raising `NotImplementedError`), but now there is a more complete framework for abstract base classes. This framework includes a new metaclass (`ABCMeta`), and the decorators `@abstractmethod` and `@abstractproperty` for defining abstract (that is, unimplemented) methods and properties, respectively. There's also a separate module (`abc`) that serves as a "support framework" for abstract base classes.

Class Decorators and New Metaclass Syntax

Class decorators work in a manner similar to function decorators. Simply put, instead of the following:

```
class A:
    pass
A = foo(A)
```

you could write this:

```
@foo
class A:
    pass
```

In other words, this lets you do some processing on the newly created class object. In fact, it may let you do many of the things you might have used a metaclass for in the past. But in case you need a metaclass, there is even a new syntax for those. Instead of this:

```
class A:
    __metaclass__ = foo
```

you can now write this:

```
class A(metaclass=foo):
    pass
```

For more information about class decorators, see PEP 3129 (<http://python.org/dev/peps/pep-3129>), and for more on the new metaclass syntax, see PEP 3115 (<http://python.org/dev/peps/pep-3115>).

Keyword-Only Parameters

It's now possible to define parameters that must be supplied as keywords (if at all). In previous versions, any keyword parameter could also be supplied as a positional parameter, unless you used a function definition such as `def foo(**kws):` and processed the `kws` dictionary yourself. If a keyword argument was required, you needed to raise an exception explicitly when it was missing.

The new functionality is simple, logical, and elegant. You can now put parameters after a `varargs` argument:

```
def foo(*args, my_param=42): ...
```

The parameter `my_param` will never be filled by a positional argument, as they are all eaten by `args`. If it is to be supplied, it must be supplied as a keyword argument. Interestingly, you do not even need to give these keyword-only parameters a default. If you don't, they become *required* keyword-only parameters (that is, not supplying them would be an error). If you don't want the `varargs` argument (`args`), you could use the new syntactical form, where the `varargs` operator (`*`) is used without a variable:

```
def foo(x, y, *, z): ...
```

Here, `x` and `y` are required positional parameters, and `z` is a required keyword parameter.

Nonlocal Variables

When nested (static) scopes were introduced in Python, they were *read-only*, and they have been ever since; that is, you can access the local variables of outer scopes, but you can't rebind them. There's a special case for the global scope, of course. If you declare a variable to be global

(with the `global` keyword), you can rebind it globally. Now you can do the same for outer, non-global scopes, using the `nonlocal` keyword.

Iterables, Comprehensions, and Views

Some other new features include being able to collect excess elements when unpacking iterables, constructing dictionaries and sets in a manner similar to list comprehension, and creating dynamically updatable views of a dictionary. The use of iterable objects has also extended to the return values of several built-in functions.

Extended Iterable Unpacking

Iterable unpacking (such as `x, y, z = iterable`) has previously required that you know the exact number of items in the iterable object to be unpacked. Now you can use the `*` operator, just for parameters, to gather up extra items as a list. This operator can be used on any one of the variables on the left-hand side of the assignment, and that variable will gather up any items that are left over when the other variables have received their items:

```
>>> a, *b, c, d = [1, 2, 3, 4, 5]
>>> a, b, c, d
(1, [2, 3], 4, 5)
```

Dictionary and Set Comprehension

It is now possible to construct dictionaries and sets using virtually the same comprehension syntax as for list comprehensions and generator expressions:

```
>>> {i:i for i in range(5)}
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4}
>>> {i for i in range(10)}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

The last result also demonstrates the new syntax for sets (see the section “Some Minor Issues,” later in this appendix).

Dictionary Views

You can now access different *views* on dictionaries. These views are collection-like objects that change automatically to reflect updates to the dictionary itself. The views returned by `dict.keys` and `dict.items` are set-like, and cannot include duplicates, while the views returned by `dict.values` can. The set-like views permit set operations.

Iterator Return Values

Several functions and methods that used to return lists now return more lazy iterable objects instead. Examples include `range`, `zip`, `map`, and `filter`.

Things That Have Gone

Some functions will simply disappear in Python 3.0. For example, you can no longer use `apply`. Then again, with the `*` and `**` operators for argument splicing, you don't really need it. Another notable example is `callable`. With it gone, you now have two main options for finding out whether an object is callable: you can check whether it has the magic method `__callable__`, or you can simply try to call it (using `try/except`). Other examples include `execfile` (use `exec` instead), `reload` (use `exec` here, too), `reduce` (it's now in the `functools` module), `coerce` (not needed with the new numeric type hierarchy), and `file` (use `open` to open files).

Some Minor Issues

The following are some minor issues that might trip you up:

- The old (and deprecated) form of the inequality operator, `<>`, is no longer allowed. You should write `!=` instead (which is common practice already).
- Backquotes won't work anymore. You should use `repr` instead.
- Comparison operators (`<`, `<=`, and the like) won't allow you to compare incompatible types. For example, you can no longer check whether 4 is greater than "5" (this is consistent with the existing rules for addition).
- There is a new syntax for sets: `{1, 2, 3}` is the same as `set([1, 2, 3])`. However, `{}` is still an empty dictionary. Use `set()` to get an empty set.
- Division is now real division! In other words, `1/2` will give you 0.5, not 0. For integer division, use `1//2`. Because this is a "silent error" (you won't get any error messages if you try to use `/` for integer division), it can be insidious.

The Standard Library

The standard library is reorganized quite a bit in Python 3.0. A thorough discussion can be found in PEP 3108 (<http://www.python.org/dev/peps/pep-3108>). Here are some examples:

- Several modules are removed. This includes previously deprecated modules (such as `mimetools` and `md5`), platform-specific ones (for IRIX, Mac OS, and Solaris), and some that are hardly used (such as `mutex`) or obsolete (such as `bsddb185`). Important functionality is generally preserved through other modules.
- Several modules are renamed, to conform to PEP 8: Style Guide for Python Code (<http://www.python.org/dev/peps/pep-0008>), among other things. For example, `copy_reg` is now `copyreg`, `ConfigParser` is `configparser`, `cStringIO` is dropped, and `StringIO` is added to the `io` module.
- Several modules have been grouped into packages. For example, the various HTTP-related modules (such as `httplib`, `BaseHTTPServer`, and `Cookie`) are now collected in the new `http` packages (as `http.client`, `http.server`, and `http.cookies`).

The idea behind these changes is, of course, to tidy things up a bit.

Other Stuff

As I mentioned at the beginning of this appendix, Python 3.0 has a lot of new features. Table D-1 lists many of them, including some I haven't discussed in this appendix. If there's something specific that's tripping you up, you might want to take a look at the official documentation or play around with the `help` function. See also Table D-2 for some sources of further information.

Table D-1. *Important New Features in Python 3.0*

Feature	Related PEP
<code>print</code> is a function.	PEP 3105
Text files enforce an encoding.	
<code>zip</code> , <code>map</code> , and <code>filter</code> return iterators.	
<code>dict.keys()</code> , <code>dict.values()</code> , and <code>dict.items()</code> return views, not lists.	
The <code>cmp</code> argument is gone from <code>sorted</code> and <code>list.sort</code> . Use <code>key</code> instead.	PEP 3100
Division is now true division: <code>1/2 == 0.5</code> .	PEP 238
There is only one string type, <code>str</code> , and it's equivalent to the Python 2.x unicode type.	
The <code>basestring</code> class is removed.	
The new bytes type is used for representing binary data and encoded text.	PEP 3137
bytes literals are written as <code>b"abc"</code> .	PEP 3137
UTF-8 is the default Python source encoding. Non-ASCII identifiers are permitted.	PEP 3120
<code>StringIO</code> and <code>cStringIO</code> are superseded by <code>io.StringIO</code> and <code>io.BytesIO</code> .	PEP 0364
New built-in string formatting replaces the <code>%</code> operator.	PEP 3101
Functions can have their parameters and return type annotated.	PEP 3107
Use <code>raise Exception(args)</code> , not <code>raise Exception, args</code> .	PEP 3109
Use <code>except MyException as identifier</code> , not <code>except MyException, identifier</code> .	PEP 3110
Classic/old-style classes are gone.	
Set metaclass with <code>class Foo(Base, metaclass=Meta):</code> .	PEP 3115
Abstract classes, <code>@abstractmethod</code> , and <code>@abstractproperty</code> are added.	PEP 3119
Class decorators, similar to function decorators, are added.	PEP 3129
Backquotes are gone. Use <code>repr</code> .	
<code><></code> is gone. Use <code>!=</code> .	
<code>True</code> , <code>False</code> , <code>None</code> , <code>as</code> , and <code>with</code> are keywords (they can't be used as names).	
<code>long</code> is renamed to <code>int</code> , and is now the only integer type, but without the <code>L</code> .	PEP 237
<code>sys.maxint</code> is gone, as there is no longer a maximum.	PEP 237

Continued

Table D-1. *Continued*

Feature	Related PEP
<code>x < y</code> is now an error if <code>x</code> and <code>y</code> are of incompatible types.	
<code>__getslice__</code> and friends are gone. Instead, <code>__getitem__</code> is called with a slice.	
Parameters can be specified as <i>keyword-only</i> .	PEP 3102
After <code>nonlocal x</code> , you can assign to <code>x</code> in an outer (nonglobal) scope.	PEP 3104
<code>raw_input</code> is renamed to <code>input</code> . For the old input behavior, use <code>eval(input())</code> .	PEP 3111
<code>xrange</code> is renamed to <code>range</code> .	
Tuple parameter unpacking is removed. <code>def foo(a, (b, c)):</code> won't work.	PEP 3113
<code>next</code> in iterators is renamed <code>x.__next__</code> . <code>next(x)</code> calls <code>x.__next__</code> .	PEP 3114
There are new octal literals. Instead of <code>0666</code> , write <code>0o666</code> .	PEP 3127
There are new binary literals. <code>0b1010 == 10</code> . <code>bin()</code> is the binary equivalent to <code>hex()</code> and <code>oct()</code> .	PEP 3127
Starred iterable unpacking is added, as for parameters: <code>a, b, *rest = seq</code> or <code>*rest, a = seq</code> .	PEP 3132
<code>super</code> may now be invoked without arguments, and will do the right thing.	PEP 3135
<code>string.letters</code> and friends are gone. Use <code>string.ascii_letters</code> .	
<code>apply</code> is gone. Replace <code>apply(f, x)</code> with <code>f(*x)</code> .	
<code>callable</code> is gone. Replace <code>callable(f)</code> with <code>hasattr(f, "__call__")</code> .	
<code>coerce</code> is gone.	
<code>execfile</code> is gone. Use <code>exec</code> instead.	
<code>file</code> is gone.	
<code>reduce</code> is moved to the <code>functools</code> module.	
<code>reload</code> is gone. Use <code>exec</code> instead.	
<code>dict.has_key</code> is gone. Replace <code>d.has_key(k)</code> with <code>k in d</code> .	
<code>exec</code> is now a function.	

Table D-2. *Sources of Information for Python 2.6 and 3.0*

Name	URL
Python v3.0 Documentation	http://docs.python.org/dev/3.0
What's New in Python 3.0?	http://docs.python.org/dev/3.0/whatsnew/3.0.html
PEP 3000: Python 3000	http://www.python.org/dev/peps/pep-3000
Python 3000 and You	http://www.artima.com/weblogs/viewpost.jsp?thread=227041

Index

■ Symbols and Numerics

- != (not equal to) operator, 580
- # sign, comments, 116
- #! character sequence, 21, 22
 - adding pound bang line, 329
- % character, string formatting, 53, 54, 56
 - changes in Python 3.0, 600, 605
- % (remainder) operator, 580
- & (bitwise and) operator, 580
- * (multiplication) operator, 580
- * (parameter splicing) operator, 126, 127, 129
- Python 3.0, 602, 603, 604, 606
- ** (exponential) operator, 580
- ** (keyword splicing) operator, 128, 129, 604
- + (unary plus) operator, 580
- += operator, 522
- (unary minus) operator, 580
- / (division) operator, 580
- == (equality) operator, 15, 93, 569, 580
- ^ (bitwise exclusive or) operator, 580
- __(double underscores), 151
- | (bitwise or) operator, 580
- ~ (bitwise negation) operator, 580
- <, <= (less than) operators, 580
- << (left shift) operator, 580
- >, >= (greater than) operators, 580
- >> (right shift) operator, 580
- 2to3 (automatic refactoring tool), 599

■ A

- ABBREV.txt file, 300, 301, 302
- ABCMeta metaclass, 601
- abs function, 16, 30, 581
- abspath function, 494
- abstract classes, Python 3.0, 601, 605
- abstraction, 139, 571
 - see also* OOP (object-oriented programming)
 - changes in Python 3.0, 601
 - classes, 147–156
 - creating functions, 115–117

- documenting functions, 116
- encapsulation, 145–147
- Fibonacci numbers program, 113–114
- inheritance, 147
- interfaces, 156–157
- making code reusable, 212
- parameters, 117–130
- polymorphism, 142–145
- program structure, 114
- recursion, 133–139
- scoping, 131–133
- value of abstraction, 121
- accept method, socket class, 306
- access attribute
 - publisher handler, mod_python, 342, 343
- accessor methods, 187
 - as attributes of property function, 188
 - private attributes, 151
- Acrobat Reader, getting, 425
- action method, rule objects
 - instant markup project, 412, 414
- ActivePython, 6, 595
- actual parameters *see* arguments
- add function, operator module, 144
- add method
 - chat server project, 479
 - set type, 229
 - wx.BoxSizer class, 285
- addDestination method, NewsAgent class, 459
- addFilter method, Parser class, 414, 415
- adding, sequences, 37
- addition operator (+), 37
- address family, stream socket, 306
- addRule method, Parser class, 414, 415
- addSource method, NewsAgent class, 459
- Adobe Acrobat Reader, getting, 425
- Albatross, 344
- algorithms, 9, 29
- alignment, string formatting, 56, 58
- all function, 581

- all variable, 219
- allow_reuse_address attribute
 - SimpleXMLRPCServer class, 527
- altsep variable, os module, 224
- and operator
 - Boolean operators, 96
 - short-circuit logic, 574
 - operator precedence, 579
- Anjuta environment, 6
- announcements
 - comp.lang.python.announce group, 597
- any function, 581
- Apache web server
 - configuring to use, 338
 - conflicting configuration definitions, 339
 - dynamic web pages with CGI, 328
 - mod_python, 336–343
- apilevel property, Python DB API, 294
- APIs
 - Python Database API, 294–298
 - Python/C API, 375–380, 596
- App class, wx module *see* wx.App class
- append method, lists, 43, 522, 585
- append mode, open function (files), 262
- appending to dictionaries, 71, 79
- appendleft method, deque type, 232
- application frameworks
 - web application frameworks, 343
- apply function, 140, 581
 - changes in Python 3.0, 604, 606
- Arachno Python environment, 6
- arcade game project, 547–567
 - banana about to be squished, 566
 - Banana class, 558
 - banana.png file, 556
 - config.py file, 556
 - further exploration, 567
 - Game class, 565
 - game states, 556
 - GameOver class, 564
 - goals, 548
 - Info class, 563
 - implementations, 551–556
 - Level class, 560
 - LevelCleared class, 564
 - objects.py file, 556, 557
 - Paused class, 561
 - preparations, 551
 - pygame module, 548
 - Pygame, 548–551
 - pygame.display module, 549
 - pygame.event module, 550
 - pygame.font module, 550
 - pygame.image module, 551
 - pygame.locals module, 549
 - pygame.mouse module, 550
 - pygame.sprite module, 550
 - Squish opening screen, 566
 - squish.py file, 556, 559
 - SquishSprite class, 557
 - StartUp class, 564
 - State class, 559
 - tools, 548–551
 - Weight class, 558
 - weight.png file, 554, 556
 - weight.pny file, 554
- archive files
 - wrapping modules as, 386–387
- args parameter/object, 377, 378
- argument splicing, Python 3.0, 604
- arguments
 - calling functions without, 572
 - command-line arguments, 223
 - levels of configuration, 398
 - default arguments, 572
 - function parameters and, 118
 - methods, 573
 - named arguments, 572
 - printing arguments, using in reverse order, 223
- argv variable, sys module, 222, 223
 - levels of configuration, 398
- arithmetic operators, 9
 - precedence, 580
- arithmetic sequence, 184
- arraysize attribute, cursors, 297
- as clause
 - changes in Python 3.0, 605
 - import statement, 85
- ascii constants, string module, 60
- ASCII encoding error
 - handling special characters, 451
- asctime function, time module, 233
- assert method, TestCase class, 356
- assert statements, 97, 118, 589

- assertAlmostEqual method, TestCase class, 356
 - assertEqual method, TestCase class, 356
 - using instead of failUnless, 360
 - AssertionError class, 589
 - assertions, 97, 111
 - assertNotAlmostEqual method, TestCase class, 356
 - assertNotEqual method, TestCase class, 356
 - assertRaises method, TestCase class, 356
 - assignment (=) operator, 569
 - assignments, 15, 85–88, 589
 - augmented assignments, 87, 589
 - chained assignments, 87
 - changing lists, 41
 - description, 13, 111
 - sequence unpacking, 85–87
 - slice assignments, lists, 42
 - asterisk width specifier, 59
 - async_chat class
 - chat server project, 473
 - collect_incoming_data method, 473, 475
 - found_terminator method, 473, 475
 - handle_close method, 475
 - push method, 475
 - set_terminator method, 473, 475
 - asynchat module, 310
 - async_chat class, 473
 - chat server project, 470, 473
 - asynchronous I/O
 - multiple connections, 312
 - Twisted framework, 316–319
 - with select and poll, 313–316
 - asyncore module, 310
 - chat server project, 470
 - dispatcher class, 471
 - loop method, 472
 - tools for chat server project, 470
 - Atom, 345
 - Atox, 424
 - attribute methods, 191–192
 - see individual method names
 - attribute reference precedence, 580
 - AttributeError class, 162
 - checking if object has specific attribute, 172
 - __getattr__ method, 192
 - attributes, 146, 573
 - accessing attributes of objects, 150–152
 - accessor methods defining, 187–188
 - binding to functions, 150
 - checking if object has specific attribute, 172
 - double underscores in attribute name, 116
 - encapsulation, 146
 - magic attributes, 116
 - object-oriented design, 157
 - private attributes, 151
 - screen scraping using HTMLParser, 326
 - special attributes, 116
 - attrs argument, handle_starttag, 326
 - atx, 424
 - augmented assignments, 87, 589
 - auth/auth_realn attributes
 - publisher handler, mod_python, 342, 343
 - autoexec.bat file, 98, 216
 - automated tests, 351
 - automatic checkers
 - limits to capabilities of, 361
 - PyChecker/PyLint tools, 359–362, 364
 - automatic refactoring tool (2to3), 599
 - AWT (Abstract Window Toolkit), 290
- ## B
- backquotes, Python 3.0, 604, 605
 - backslash character (\)
 - escaping quotes, 23
 - escaping, regular expressions, 243
 - raw strings, 27, 28
 - backticks
 - representing strings, 25
 - backtracking, generators
 - solving Eight Queens problem, 200–201
 - backup parameter, input function, 226
 - BaseRequestHandler class
 - SocketServer module, 311
 - bases attribute, 155
 - issubclass method, 154
 - basestring class, Python 3.0, 605
 - basestring function, 581
 - BasicTextParser class, 422
 - “batteries included” phrase, 221
 - BBCode, 424
 - bdist command, Distutils, 387
 - formats switch, 387
 - rpm format, 387
 - wininst format, 387, 388
 - Beautiful Soup module, 327–328
 - Berkeley DB, 515

- Binary constructor, Python DB API, 298
 - binary literals, Python 3.0, 606
 - binary mode, open function (files), 262
 - binary search
 - recursive function for, 136–138
 - BINARY value, Python DB API, 298
 - bind method, socket class, 306
 - chat server project, 471, 472
 - Bind method, widgets, 286, 291
 - binding parameters, 572
 - BitTorrent, 517, 518
 - bitwise operators, 580
 - BlackAdder environment, 6
 - blit function, 549
 - blitting, 549
 - blocking, 306
 - blockquote element, bulletin board project, 504
 - blocks, 88, 111, 569
 - finding blocks of text, 406–407
 - blocks generator
 - instant markup project, 406
 - blogs, 597
 - Boa Constructor environment, 6
 - body method, NNTP class, 455, 457
 - bool function, 581
 - Boole, George, 89
 - Boolean operators, 38, 95–96
 - short-circuit logic, 96
 - Boolean values, 89–90
 - Boost.Python, 371
 - bottlenecks
 - extending Python, 365, 380
 - bound methods, 150
 - calling unbound superclass constructor, 180
 - BoxSizer class, wx module *see* wx.BoxSizer class
 - break statements, 102, 591
 - else clause, try/except statement, 168
 - extracting subject of an article, 457
 - infinite recursion, 134
 - using with for and while loops, 105
 - while True/break idiom, 104–105, 271
 - broadcast method, Node class
 - chat server project, 479
 - XML-RPC file sharing project, 521, 522, 525, 532
 - browsers
 - open function, webbrowser module, 225
 - buffering argument, open function (files), 263
 - buffers
 - closing files after writing, 267
 - updating files after writing, 268
 - bugs *see* debugging
 - build command, Distutils, 384, 385
 - build subdirectory, Distutils, 385
 - build_ext command, Distutils, 389
 - built-in functions, 16, 581–584
 - built-in string formatting, Python 3.0, 605
 - bulletin board project, 499–515
 - creating database, 501
 - cursor object, 503–504
 - database password, 503
 - edit.cgi script, 507, 510–511
 - further exploration, 515
 - hidden inputs, 510
 - implementations, 502–514
 - main page, 513
 - main.cgi script, 506, 507–508
 - message composer, 514
 - message viewer, 514
 - preparations, 500–502
 - requirements, 500
 - save.cgi script, 507, 511–513
 - simple.main.cgi script, 505
 - testing, 513
 - tools, 500
 - view.cgi script, 506, 508–510
 - Button class, wx module *see* wx.Button class
 - buttons
 - adding button to frame, 281
 - Bind method, widgets, 286
 - event handling, 286
 - setting button label, 282
 - setting button size/position, 283
 - wx.EVT_BUTTON symbolic constant, 286
 - bytearray class, Python 3.0, 600
 - bytes class, Python 3.0, 600
 - bytes literals, Python 3.0, 605
 - bytes type, Python 3.0, 599, 605
 - BytesIO, Python 3.0, 605
- ## C
- c (%) conversion specifier, 57
 - C extensions, 371
 - extending Python, 369–380
 - C programming
 - deallocating objects, 376

- extending Python for improved speed, 365–366
- importing existing (shared) C libraries, 370
- including C/C++ directly in Python code, 370
- Python/C API, 375
 - reference manual, 596
- C# class
 - IronPython extending Python, 368
- C++
 - enabling interoperability Python/C++, 371
 - including C/C++ directly in Python code, 370
- caching
 - XML-RPC file sharing project, 534
- callable function, 115, 157, 159, 581
 - Python 3.0, 604, 606
- callback method, Handler class, 411
- callback methods, HTMLParser, 325
- callproc method, cursors, 297
- Canvas class, pdfgen module, 427
- capitalize method, strings, 586
- capwords function, string module, 63, 66
- cat command, files, 265
- catching exceptions, 163–170
 - catching all exceptions, 167, 169
 - catching exception object, 166
 - catching many exceptions in one block, 166
 - description, 173
 - raising exceptions again, 164–165
 - try/except statement, 163–169
 - using more than one except clause, 165–166
- ceil function, 17, 30
- cElementTree, 437
- center method, strings, 586
- CGI (Common Gateway Interface)
 - bulletin board project, 502, 506
 - edit.cgi script, 507, 510–511
 - main.cgi script, 506, 507
 - save.cgi script, 507, 511–513
 - simple_main.cgi script, 505
 - view.cgi script, 506, 508–510
 - CGI handler, mod_python, 336, 338–339
 - CGI script, 331
 - debugging with cgitb, 331–332
 - description, 347
 - dynamic web pages with, 328–336
 - adding the pound bang (!) line, 329
 - preparing web server, 328–329
 - setting file permissions, 329–330
 - getting information from CGI script, 335
 - HTML form, 334–336
 - input to CGI script, 333
 - invoking CGI scripts without forms, 334
 - performance using CGI handler, 339
 - remote editing with CGI project, 489–498
 - running CGI script, 339
 - security risks, 330
 - using cgi module, 333
 - cgi file name extension, 329, 339
- cgi module
 - description, 310
 - dynamic web pages with CGI, 328, 333
 - FieldStorage class, 333
 - remote editing with CGI project, 489, 490
- cgi-bin subdirectory, 329
- cgitb module
 - debugging with, 331–332
 - enable function, 331, 347
 - remote editing with CGI project, 490
 - tracebacks, 502
- chained assignments, 87
- chained comparison operators, 93
- character sets, 243
- characters event handler
 - XML parsing project, 440, 441
- chat server project, 469–487
 - advantages of writing, 469
 - asynchat module, 473
 - asyncore module, 471
 - ChatServer class, 471–473
 - ChatSession class, 473–475
 - collecting data (text) coming from client, 473
 - command interpretation, 477–478
 - further enhancement, 486
 - implementations, 471–485
 - listening on port for incoming connections, 471
 - new server, 480–485
 - preparations, 470–471
 - requirements, 469
 - rooms, 478–480
 - tools, 470
- chat services, 469
- ChatRoom class, 479, 483
 - look command, 480, 485
 - say command, 480, 485
 - who command, 480, 485

- ChatServer class, 480, 484, 471–473
- ChatSession class, 480, 484, 473–475
 - enter method, 480
- checkIndex function, 185
- CherryPy, 344
- chmod command, UNIX, 330
- choice function, random module, 144, 159, 235
- chr function, 95, 112, 581
- chunks attribute
 - screen scraping using HTMLParser, 326
- clamp method, rectangles, 556
- class attribute
 - finding out class of an object, 155
- class decorators, Python 3.0, 601, 605
- class definition statement, 594
- class keyword, 573
- class methods, 189–191
 - cls parameter, 190
 - self parameter, 189
- class namespace, 152–153
- class scope variable, 153
- class statement, 149
 - self parameter, 149
 - superclasses, 153
- classes, 147–148, 158, 573
 - abstract classes, 601
 - accessing attributes of objects, 150–152
 - accessor methods, 151
 - built-in exception classes, 162
 - changes in Python 3.0, 605
 - class decorators, 601
 - class namespace, 152–153
 - class statement, 149
 - classes and types, 147
 - creating, 148–149
 - new-style classes, 206
 - custom exception classes, 163
 - defining, 573
 - distinguishing methods from functions, 150
 - exception classes, 162, 163
 - inheritance, 141, 147–155, 159
 - instances, 147
 - isinstance method, 155
 - interfaces, 156–157
 - metaclasses, 176
 - method definitions, 149
 - mix-in classes, 444–446
 - naming conventions, 148
 - new-style/old-style classes, 149, 175, 206
 - implementing properties with old-style classes, 191
 - Python version, 3.0, 176
 - object-oriented design, 157, 158
 - objects and, 147, 155
 - OOP, 147–156
 - overriding methods, 206
 - property function, 189
 - specifying superclasses, 153–154
 - subclasses, 147, 148
 - subclassing built-in classes, 175
 - superclasses, 147
 - multiple superclasses, 155–156
- classmethod function, 581
- clear method, dictionaries, 74–75, 585
- clear method, Group class
 - arcade game project, 550, 552
- Clearsilver, 341
- Client class
 - GUI client project, 539, 542
 - fetchHandler, 538, 540, 541, 544
 - OnInit method, 538, 539, 541, 543
 - XML-RPC file sharing project, 528, 533
- clients
 - chat server project, 470, 471
 - GUI client project, 537–545
 - XML-RPC file sharing project, 527–528
- close function, fileinput module, 226
 - finding sender of e-mail, 253
- close method, connections, 296, 300
 - bulletin board project, 503
- close method, cursors, 297
- close method, files, 264, 267
- close method, generators, 199
- closeHandler, Java Swing, 290
- closing files, 267–268
- clpa_server variable, 458
- cls parameter, class methods, 190
- cmath module, 18
- Cmd class, cmd module, 527
 - modeling command interpretation on, 477
- cmd module, 259
 - Cmd class, 477, 527
 - XML-RPC file sharing project, 519
- cmp argument, sort method, 605

- cmp function, 52, 581
 - making comparisons, 93
- code
 - making code reusable, 212
 - reading source code to explore modules, 221
 - source code checking, 359
- code coverage, testing, 351
- Code Crusader environment, 6
- code fatigue, 395
- Code Forge environment, 6
- code reuse, 212
- coerce function, Python 3.0, 604, 606
- collect_incoming_data method
 - chat server project, 473, 475
- collections
 - see also* mappings; sequences
 - collections module, 231
- combine function, 132
- command interpretation
 - chat server project, 477–478
 - modeling on Cmd class, 477
- command prompt
 - running scripts from, 20
- CommandHandler class
 - chat server project, 478, 481
- command-line arguments, 223
 - levels of configuration, 398
- command-line switches, 398
- command-line tools
 - using with subprocess module, 360
- commands
 - cat command, 265
 - import command, 17
 - pipe characters linking, 265
 - python command, 265
 - sort command, 265
- commas
 - separating print statements with, 83–84
- comments, 22, 570
 - documenting functions, 116
- commit method, connections, 296, 300
 - bulletin board project, 503
 - save.cgi script, 511
- Common Gateway Interface *see* CGI
- comp.lang.python group, 597
- comparison operators, 92–95
 - chaining, 93
 - comparing incompatible types, 92
 - comparing sequences, 95
 - comparing strings, 94
 - equality operator, 93
 - identity operator, 93–94
 - in operator, 94
 - is operator, 93–94
 - membership operator, 94
 - Python 3.0, 604, 606
- compile function, re module, 245
- compiling extensions, Distutils, 388–389, 390
- complex function, 581
- complex numbers, 18
- Complex type, 579
- components *see* widgets
- comprehensions, 603
- computer games
 - arcade game project, 547–567
- concatenating strings, 24
- condition method, rule objects, 412, 413, 414
- conditional operator, 96
- conditional statements, 88–97
 - assertions, 97
 - Boolean operators, 95–96
 - comparison operators, 92–95
 - conditional execution, 90
 - conditions, 92–96
 - description, 111
 - elif clauses, 91
 - else clauses, 90
 - if statements, 90
 - nesting blocks, 91
 - short-circuit logic, Boolean operators, 574
- config.py file, 397
 - arcade game project, 556
- configparser module, 397, 398
 - renamed modules in Python 3.0, 604
- configuration, 396–398
 - description, 394, 401
 - levels of, 398
- configuration files, 396–398
 - dividing into sections, 397
- conflict function, Eight Queens problem, 202
- connect function, Python DB API, 300, 304
 - parameters, 296
- connect method, socket class, 306

- connection object, 296, 303
 - bulletin board project, 503, 511
 - connectionLost event handler, 317
 - connectionMade event handler, 317
 - connections
 - bulletin board project, 502
 - chat server project, 471, 472
 - network programming, 311–316
 - Python DB API, 296, 303
 - console I/O, Python 3.0, 600
 - constants, 396
 - string module, 60
 - symbolic constants, 396
 - constructors, 176–181, 575
 - creating, 177
 - default parameters, 574
 - description, 206
 - init method, 177, 575
 - overriding, 177–179
 - using super function, 180, 181
 - Python DB API, 297
 - unbound superclass constructor, 179–180
 - containers, 32
 - content handlers
 - creating, 439, 441
 - dispatcher mix-in classes, 446
 - ContentHandler class
 - XML parsing project, 451
 - xml.sax.handler module, 439
 - Content-type header
 - dynamic web pages with CGI, 331
 - context managers, 268
 - continual rewriting syndrome, 395
 - continue statements, 103, 591
 - using with for and while loops, 105
 - control structures, 569
 - conversion flags, 56
 - conversion specifiers, 56
 - % character, 56
 - dictionaries, 73
 - field width, 56, 59
 - minimum field width, 57
 - precision, 54, 56, 57
 - string formatting, 54–59
 - tuples, 56
 - types, 56, 57
 - conversion types, string formatting, 56, 57
 - conversions
 - between numbers and strings, SQLite, 303
 - convert function, 301
 - convert method, surface objects, 554
 - Cookie module, 310
 - cookie-cutter code
 - automating, 377
 - cookielib module, 310
 - copy function, copy module, 220
 - copy method, dictionaries, 75, 585
 - count method, lists, 43, 585
 - count method, strings, 586
 - CounterList class, 186, 187
 - coverage
 - code coverage, 351
 - test coverage, 351, 352
 - CPython, extending, 367, 369–371
 - cracking, vs. hacking, 1
 - CREATE TABLE command
 - bulletin board project, 501
 - create_socket method
 - chat server project, 471, 472
 - cStringIO, Python 3.0, 605
 - csv module, 258
 - ctypes library, 370
 - cursor method, connections, 296, 300
 - cursor objects, 296
 - in bulletin board project, 503–505
 - cursors, Python DB API, 296–297, 303
 - attributes, 297
 - bulletin board project, 503
 - methods, 296
 - custom exception classes, 163, 173
 - CXX *see* PyCXX
 - cyclic garbage, 377
- ## D
- %d conversion specifier, 56
 - Daily Python-URL blog, 597
 - Dalke, Andrew
 - Sorting Mini-HOWTO, 49
 - data
 - analyzing many forms of numeric data, 370
 - fetching data from Internet, 432
 - data structures, 31, 570
 - containers, 32
 - dequeues, 231–232
 - heaps, 230–231

- lists, 40–49
- mappings and dictionary type, 69
- sequences, 31–40
- sets, 228–229
- stacks, 45
- tuples, 49–51
- Database API *see* Python Database API
- database parameter
 - connect function, Python DB API, 296
- DatabaseError exception, Python DB API, 295
- databases
 - compact table-based databases, 293
 - food database application, 300–303
 - importing data into, 301
 - key-value databases, 293
 - object databases, 293
 - popular commercial choices, 293
 - Python Database API, 294–298
 - relational databases, 293
 - supported by Python packages, 293
- DataError exception, Python DB API, 295
- datagram socket, 306
- dataReceived event handler, 317
- Date constructor, Python DB API, 298
- DateFromTicks constructor, Python DB API, 298
- dates
 - fields of Python date tuples, 233
- datetime module, 234, 258, 456
- DATETIME value, Python DB API, 298
- DB API *see* Python Database API
- deallocating objects, 376
- Debian Linux, installing Python, 4
- debugging
 - anticipating code changes, 351
 - cgibt module, 331–332
 - PythonDebug directive, 339, 340
 - remote editing with CGI project, 490
- decode method, strings, 586
- decorators
 - abstract classes, Python 3.0, 601
 - changes in Python 3.0, 605
 - class decorators, Python 3.0, 601
 - description, 190
- deep copy, dictionaries, 76
- deepcopy function, copy module, 76, 220
- def statements, 115, 571
 - class namespace, 152
 - documenting functions, 116
 - generator-function component, 198
- default arguments, 572
- default values, parameters, 124
 - using empty lists as, 575
- defaultdict dictionary, 232
- defaultStart/defaultEnd methods
 - XML parsing project, 445, 447
- deferred execution, Twisted, 317
- definitions
 - class definitions, 594
 - function definitions, 594
- del method, 177
- del operation, dictionaries, 71
- del statements, 107–108, 590
 - deleting elements from lists, 41
 - description, 112
 - using for cleanup operation, 170
- delattr function, 581
- __delattr__ method, 191
- delitem method, 182, 184
- deque module, 259
- deque type, 231–232
 - collections module, 231
- deques, 231–232
- description attribute, cursors, 297
- descriptor protocol, 189
- design
 - object-oriented design, 157–158
- destructors
 - __del__ method, 177
- __dict__ attribute
 - avoiding endless looping, 192
 - __getattr__ method trap, 192
 - seeing all values stored in objects, 157
- dict function, 71, 81, 581
- dictfetchall method, cursor object
 - bulletin board project, 503, 504
 - SQLite alternative, 505
- dictfetchone method, cursor object
 - bulletin board project, 503
- dictionaries, 571
 - ** operator, 127, 128
 - accessing dictionary items, 76
 - adding items to, 71, 72
 - assigning value to new key, 71
 - checking if key exists, 78

dictionaries (*continued*)

- constructing from other mappings, 71
 - conversion specifiers, 73
 - creating, 70
 - creating with values of None, 76
 - deep copy of, 76
 - defaultdict, 232
 - empty dictionary, 604
 - globals function, 132
 - iterating over, 100
 - keys, 121
 - keys and values, 70
 - locals function, 132
 - membership, 71, 72
 - modules mapping, 222
 - overwriting same key items from another, 80
 - precedence, 581
 - removing all items from, 74
 - removing arbitrary value from, 79
 - returning all items of, 78
 - returning list of keys, 78
 - returning list of values, 80
 - returning value of specified key, 79
 - shallow copy of, 75
 - string formatting with, 73, 81
 - subclassing dict type, 185–187
 - uses, 69
- dictionary comprehension, Python 3.0, 603
- dictionary methods, 74–80, 585–586
- clear, 74–75, 585
 - copy, 75, 585
 - fromkeys, 76, 585
 - get, 76–78, 585
 - has_key, 78, 585, 606
 - items, 78, 585
 - iteritems, 78, 585
 - iterkeys, 78, 585
 - itervalues, 80, 585
 - keys, 78, 585
 - pop, 79, 586
 - popitem, 79, 586
 - setdefault, 79, 106, 586
 - update, 80, 586
 - values, 80, 586
- dictionary type
- deepcopy function, 76
 - in operation, 71
 - key related operators, 71
 - mappings and, 69
 - operations, 71–73
 - purpose of, 69–70
 - syntax, 70
 - types for keys, 72
 - uniqueness of values, 70
 - using, 70
- dictionary views, Python 3.0, 603
- difflib library, 258
- digests, passwords, 494
- digits constant, string module, 60
- dir function, 260, 581
- exploring modules, 218
- directory element
- XML parsing project, 437
- directory list
- XML parsing project, 448
- discussion forum
- bulletin board project, 499–515
- dispatch method
- XML parsing project, 445
- Dispatcher class
- bind method, 471, 472
 - chat server project, 471
 - create_socket method, 471
 - garbage collection, 480
 - handle_accept method, 471, 472, 475, 480
 - listen method, 471, 472
 - set_reuse_addr method, 473
 - XML parsing project, 445, 448
- display method, Level class, 561
- display method, State class, 560
- display module, pygame, 549
- dist subdirectory, Distutils, 387
- distribute method, NewsAgent class, 459, 462
- distributing operators, 128–129, 604
- distribution formats, 387
- distributions
- ActivePython, 595
 - alternative Python distributions, 5–7
 - distributing Python packages, 383
 - IronPython, 595
 - Jython, 595
 - MacPython, 595
 - Official Python Distribution, 595

- Python distributions, 595
 - pywin32, 595
 - Distutils toolkit, 383–386
 - bdist command, 387
 - build command, 384, 385
 - build subdirectory, 385
 - build_ext command, 389
 - compiling extensions, 388–389
 - description, 383, 390
 - dist subdirectory, 387
 - install command, 385
 - installing, 384
 - lib subdirectory, 385
 - py_modules directive, 386
 - py2exe extension, 389–390
 - register command, 390
 - sdist command, 386, 387
 - setup function, 384, 391
 - setup.py script, 383, 384, 385, 387, 390
 - setuptools project, 384
 - SWIG, 375, 389
 - uninstall command, 385, 388
 - wrapping modules as archive file, 386–387
 - division, 9, 10
 - division (/) operator, 580
 - double slash (//) operator, 10
 - integer division, Python 3.0, 604, 605
 - rounding, 16
 - divmod function, 581
 - Django, 343, 344
 - do_exit method, Client class
 - XML-RPC file sharing project, 533
 - do_fetch method, Client class
 - GUI client project, 538
 - XML-RPC file sharing project, 533
 - do_logout method, chat server project, 479
 - do_look method, chat server project, 480
 - do_say method
 - chat server project, 480
 - XML-RPC file sharing project, 527
 - do_who method, chat server project, 480
 - doc attribute
 - exploring modules, 220
 - function attributes, 116
 - doc parameter, property function, 189
 - docstrings, 116, 220
 - exploring modules, 220
 - doctest module, 352, 353–355, 364
 - testmod function, 353, 354, 364
 - Document Object Model (DOM), 439
 - documentation
 - creating graphics and documents in PDF, 425
 - exploring modules, 220–221
 - Macintosh library modules, 596
 - Python, 596
 - DOM (Document Object Model), 439
 - DOS, handling whitespace for, 225
 - double slash operator, 10
 - double underscores (__)
 - making method or attribute private, 151
 - double-clicking, 21
 - double-ended queues (deques), 231–232
 - draw method, 549, 550, 552
 - drawToFile method, renderPDF class, 428
 - dsn parameter
 - connect function, Python DB API, 296
 - duck typing, 145
 - dynamic web pages
 - screen scraping, 321
 - dynamic web pages with CGI, 328–336
- ## E
- %E, %e conversion specifiers, 57
 - Eclipse environment, 6
 - edit.cgi script
 - bulletin board project, 510, 511
 - description, 507
 - link from main.cgi, 507
 - link from view.cgi, 508
 - testing, 513
 - remote editing with CGI project, 492–494, 496
 - editing
 - remote editing with CGI project, 489–498
 - eggs, Python, 384
 - Eight Queens problem, 200–206
 - ElementTree, 437
 - dealing with XML in Python, 439
 - elif clauses, if statements, 91, 592
 - else clauses
 - if statements, 90, 592
 - try/except statement, 168–169
 - combining try/except/finally/else, 170
 - description, 173
 - using in loops, 105
 - email, finding sender of, 251–253

- email addresses filter
 - instant markup project, 418
- email module, 310
- EmailDestination class
 - news gathering project, 468
- empty dictionary, 604
- empty lists, 37
 - using as default value, 575
- empty set, 604
- enable function, cgibt module, 331, 347
- encapsulation, 145–147, 573
 - accessing attributes of objects, 150–152
 - accessor methods, 187
 - attributes, 146
 - description, 141, 158
 - extending Python, 366
 - state, 147
- encode method, strings, 586
- encoding, Python 3.0, 599, 605
- end method
 - MatchObjects, re module, 248
- end method, Handler class
 - instant markup project, 410, 411
- endElement event handler
 - XML parsing project, 440, 441, 445
- endless loop trap
 - setattr method, 192
- EndSession exception
 - chat server project, 479
- endswith method, strings, 586
- ensureDirectory method
 - XML parsing project, 447, 448
- enter method
 - context managers, 268
- enter method, ChatSession class, 480
- enumerate function, 102, 112, 582
- environ mapping, os module, 223, 224
- environment variables
 - description, 216
 - environ mapping, os module, 224
 - levels of configuration, 398
 - PYTHONPATH, 215
 - setting in UNIX and Mac OS X, 216
 - setting in Windows, 216
- EOF command, 527
- Epytext
 - markup systems and web sites, 424
- equality (==) operator, 15, 93, 569, 580
- eric environment, 6
- Error exception, Python DB API, 295
- error handling
 - exceptions, Python DB API, 295
- error messages *see* tracebacks
- errors
 - see also* exceptions
 - AttributeError class, 162
 - catching Pygame-specific errors, 549
 - distinguishing from failures in unittest, 357
 - inappropriate type used, 183
 - index outside range, 183
 - IndexError class, 162
 - IOError class, 162
 - KeyError class, 162
 - NameError class, 162
 - NotImplementedError exception, 224
 - stderr stream, sys module, 222
 - SyntaxError class, 163, 254
 - TypeError class, 163
 - ValueError class, 163
 - ZeroDivisionError class, 161, 163
- escape function, re module, 245, 247
- escaping quotes, 23–24
- escaping special characters
 - regular expressions, 242
- EtText, 424
- eval function, 112, 582
 - sample template system, 254
- eval statements, 110
 - description, 112
 - scope, 111
- event handling
 - Bind method, widgets, 286
 - button events, 286
 - chat server project, 471
 - closeHandler, Java Swing, 290
 - connectionLost event handler, 317
 - connectionMade event handler, 317
 - dataReceived event handler, 317
 - description, 291
 - HTMLParser callback methods, 325
 - load function, 286
 - rawDataReceived event handler, 318
 - save function, 286
 - screen scraping using HTMLParser, 326

- when event handler is called, 286
- writing Twisted server, 317
- wx.EVT_BUTTON symbolic constant, 286
- wxPython GUI toolkit, 286
- XML parsing project, 439–441, 448–450
- event module, pygame, 550
- event-driven networking framework, 316
 - writing Twisted server, 317
- events, 286
 - polling event constants in select module, 315
 - XML parsing project, 439–441
- except block
 - Python 3.0, 605
 - Python DB API, 295
- except clause, try statement, 593
 - see also* try/except statements
 - catching all exceptions, 167
 - catching exception object, 166
 - catching many exceptions in one block, 166
 - description, 173
 - trapping KeyError exception, 172
 - using more than one except clause, 165–166
- Exception class, 162
 - catching all exceptions, 169
 - raise statement, 162
- exception objects, 161, 173
 - catching, 166
- exceptions, 161, 576
 - see also* errors
 - built-in exception classes, 162, 163
 - catching exceptions, 163–170
 - catching all exceptions, 167, 169
 - catching exception object, 166
 - catching many exceptions in one block, 166
 - danger of catching all exceptions, 167
 - description, 173
 - raising exceptions again, 164–165
 - try/except statement, 163–169
 - connecting to NNTP servers, 455
 - custom exception classes, 163, 173
 - doing something after exceptions, 169–170
 - EndSession exception, 479
 - exception hierarchy, 295
 - exception objects, 161, 173
 - functions and, 164, 170–171, 173
 - GeneratorExit exception, 199
 - indicating everything worked, 168–169
 - NotImplementedError exception, 224
 - Python DB API, 295
 - raise statement, 162–163
 - raising exceptions, 161–163, 173
 - StopIteration exception, 192
 - SyntaxError exception, 254
 - try/except statement, 163–169
 - using more than one except clause, 165–166
 - warnings, 173
 - XML-RPC file sharing project, 528–529
 - Zen of, 171–173
- exec statements, 109–110, 592
 - changes in Python 3.0, 606
 - description, 112
 - replacing reload function functionality using, 211
 - sample template system, 254
 - scope, 111
- execfile function, 582
 - Python 3.0, 604, 606
- executable binaries, 390
- executable Windows programs
 - creating with py2exe, 389–390
- execute method, cursors, 297, 301, 302
 - bulletin board project, 503
- executemany method, cursors, 297, 301
- executing programs, 19–20
- execv function, 224
- exit command
 - XML-RPC file sharing project, 527
- exit function, sys module, 222
- exit method
 - context managers, 268
- expandtabs method, strings, 586
- exponentiation operator (**), 11
 - compared to pow function, 16
 - precedence, 580
- expression statements, 589
- expressions, 9–12, 579–588
 - compared to statements, 13
 - description, 29
 - evaluating expression strings, 254
 - logical expressions, 573
 - precedence, 580
- extend method
 - deque type, 232
 - lists, 44, 585

- extending Python, 365–366
 - architecture, 366
 - CPython, 367
 - encapsulation, 366
 - extending CPython, 369–371
 - extension approaches, 380
 - identifying bottlenecks, 365
 - IronPython, 367–369
 - Jython, 367–369
 - Python extensions, 380
 - SWIG, 371–375
 - using Python/C API, 380, 375–380
 - framework for extensions, 377–378
 - hand-coded palindrome module, 379–380
 - reference counting, 376–377
 - wrapping legacy code, 366
 - writing C extensions, 369–380
- extendleft method
 - deque type, 232
- extensions
 - compiling extensions, 388–389, 390
 - framework for, 377–378
 - py2exe extension, Distutils, 389–390
- Extreme Programming, 349, 393
- F**
- %F conversion specifier, 57
- %f conversion specifier, 54, 57
- factorial function, 135
- factorials
 - recursive function to calculate, 134
- Factory class
 - twisted.internet.protocol module, 317
- fail method, TestCase class, 357
- failIf method, TestCase class, 357
- failIfAlmostEqual method, TestCase class, 356
- failIfEqual method, TestCase class, 356
- failUnless method, TestCase class, 356
 - using assertEqual instead of, 360
- failUnlessAlmostEqual method, TestCase class, 356
- failUnlessEqual method, TestCase class, 356
- failUnlessRaises method, TestCase class, 357
- failures
 - distinguishing from errors in unittest, 357
- False value (Boolean), 89
 - changes in Python 3.0, 605
- Fault class, xmlrpclib module, 528, 529
- feed method
 - instant markup project, 410
- feeds *see* RSS feeds
- fetch command
 - XML-RPC file sharing project, 527
- fetch method, Node class
 - XML-RPC file sharing project, 520, 521, 524, 526, 527, 531
- fetchall method, cursors, 297, 302
 - bulletin board project, 503, 504
- fetchHandler, Client class
 - GUI client project, 538, 540, 541, 544
- fetchmany method, cursors, 297
- fetchone method, cursors, 297
 - bulletin board project, 503
- fget/fset/fdel parameters, property function, 189
- Fibonacci numbers program, 113
- field width, string formatting, 56, 57, 59
- FieldStorage class, cgi module, 333
- file function, 275, 582
 - Python 3.0, 604, 606
- file iterators, 272–274
- file locking
 - remote editing with CGI project, 497
- file methods, 263–270
 - close method, 264, 267
 - examples using, 268–270
 - flush method, 268
 - read method, 264
 - readline method, 266
 - readlines method, 266
 - seek method, 266
 - tell method, 266
 - write method, 264
 - writelines method, 267
 - xreadlines method, 272
- file permissions
 - dynamic web pages with CGI, 329–330
- file property, modules
 - exploring modules via source code, 221
- file sharing, 517
 - adding GUI client to Python program, 537–545
 - XML-RPC file sharing project, 517–535
- filecmp module
 - news gathering project, 468
- fileinput module, 225–227
 - description, 259

- finding sender of e-mail, 252
- functions, 225
- lazy line iteration with `fileinput`, 272
- sample template system, 255, 256
- `filelineno` function, `fileinput` module, 226
- `filename` function, `fileinput` module, 226
- files
 - closing files, 267–268, 274
 - file types, 274
 - file-like objects, 263, 274
 - finding file name, `wxPython`, 286
 - iterating over file contents, 270–274
 - byte by byte, 270–271
 - description, 274
 - file iterators, 272–274
 - iterating over lines in very large file, 272
 - lazy line iteration with `fileinput`, 272
 - one line at a time, 271
 - reading everything first, 271
 - without storing file object in variable, 273
 - modes, 261–263, 274
 - opening files, 261–263
 - buffering, 263
 - changes to file in text mode, 262
 - description, 274
 - pipng output, 264–265
 - random access, 266
 - read/write/append/binary modes, 261
 - reading and writing, 264
 - closing files after, 267
 - reading files, 274
 - reading lines, 266, 274
 - streams, 274
 - universal newline support mode, 263
 - using as context managers, 268
 - validating file names, XML-RPC, 529–534
 - writing files, 274
 - updating files after writing, 268
 - writing lines, 266, 274
- Filter class, 154
- filter function, 138, 139, 140, 582
 - changes in Python 3.0, 605
 - instant markup project, 414
- filters
 - fetching data from Internet, 432
 - instant markup project, 409, 413, 418
- `filterwarnings` function, 174
- finally clause, try statement, 169–170, 173, 593
 - combining try/except/finally/else, 170
- find method, strings, 60, 586
- findall function, re module, 245, 246
 - finding sender of e-mail, 252
- findall method, regular expressions
 - news gathering project, 462
- firewalls, network programming and, 305
- `firstDisplay` method
 - arcade game project, 560, 562
- flag argument, `wx.BoxSizer` class, 285
- flags
 - conversion specifiers, 56
- flags parameter, 247
- flip function, arcade game project, 549, 552
- float function, 30, 582
 - food database application, 300
- Float type, 579
- floating-point numbers, 10
- floats, 10
- floor function, 16, 30
- flush method, files, 268
- Font function, arcade game project, 550
- font module, `pygame`, 550
- food database application, 300–303
 - creating and populating tables, 301–302
 - `food_query.py`, 303
 - `importdata.py`, 301
 - searching and dealing with results, 302–303
- footers
 - `writeFooter` method, 446
- for loops, 99, 570
 - Fibonacci numbers program, 113
 - generators
 - iter method, 193
 - iterable files, 272
 - list comprehension, 105–106
 - recursive generators, 196
- for statements, 593
- forking, 312
 - chat server project, options for, 469
 - multiple connections, 312
 - `SocketServer` module, 313
- form tag, action attribute, 492
- formal parameters, 118
- format function, bulletin board project, 504, 507

- format method, strings
 - changes in Python 3.0, 600
- format strings, Python/C API, 378
- formats switch
 - bdist/sdist commands, Distutils, 387
- formatting strings
 - changes in Python 3.0, 600
 - conversion specifiers, 54–59
 - string formatting operator, 53
- forms
 - invoking CGI scripts without, 334
 - writing HTML forms, 334–336
- forums, online Python resources, 597
- found_terminator method
 - chat server project, 473, 475, 479
- Frame class, wx module *see* wx.Frame class
- frames
 - adding button to frame, 281
 - setting frame size, 283
 - setting frame title, 282
 - wx.Frame class, 281
 - wxPython GUI toolkit creating, 281
- frameworks
 - event-driven networking framework, 316
 - framework for extensions, 377–378
 - SocketServer framework, 310–311, 319
 - Twisted framework, 316–319, 320
 - web application frameworks, 343
- freshmeat.net, 596
- from module import statement, 17
 - reasons not to use, 18
- fromkeys method, dictionaries, 76, 585
- frozenset function, 582
- frozenset type
 - immutability, 229
- ftplib module, 310
- function annotation, Python 3.0, 601
- function attributes
 - doc attribute, 116
- function call precedence, 580
- function definition statement, 115, 594
- functional programming, 138–139, 140
- functional requirements
 - requirement specification, 350
- functions, 29, 571–572
 - see* individual function names
 - as values in Python, 572
 - binding attributes to, 150
 - built-in functions, 16, 581–584
 - calling functions without arguments, 572
 - containing yield statement, 195
 - creating, 115–117
 - defining functions in modules, 212
 - testing modules, 212–214
 - distinguishing methods from, 150
 - documenting, 116
 - ending functions, 117
 - exceptions and, 164, 170–171, 173
 - extinct functions in Python 3.0, 604
 - flags parameter, 247
 - formal parameters, 118
 - from module import statement, 17
 - function definition, 139
 - functions without return values, 117
 - generator-function, 198, 207
 - local naming, 120
 - methods compared, 43
 - nested scopes, 133
 - number of scopes/namespaces, 131
 - object-oriented design, 157
 - parameter/return type annotation, 601, 605
 - parameters, 16, 117–130, 139
 - changing, 118–120
 - collecting parameters, 125–128
 - distributing operators, 128–129, 604
 - examples using, 129–130
 - gathering operators, 125–128, 602, 603, 604, 606
 - immutability, 123
 - keyword parameters, 123–125
 - keyword-only parameters, 602, 606
 - passing parameters to functions, 572
 - reasons for changing, 120–122
 - values, 118
 - parts of recursive function, 134
 - recursion, 133–139, 140
 - recursive functions, 134
 - return value, 16
 - caution using if statements, 117
 - return value is None, 117
 - type objects, 17
- functools module, 258
- future module, 10, 19

G


- %G, %g conversion specifiers, 57
- games, arcade game project, 547–567
 - Game class, 565
 - game states, 556
 - GameOver class, 564
- garbage collection
 - cyclic garbage, 377
 - del method, 177
 - gc module, 377
 - reference counting, 377
 - unbound objects, 480
- gathering operators, 125–128, 602, 603, 604, 606
- gc module, 377
- generator comprehension, 196
- GeneratorExit exception
 - close method, generators, 199
- generators, 194–200
 - backtracking and, 200–201
 - close method, 199
 - components of, 198
 - description, 207
 - generator-function, 198, 207
 - generator-iterator, 198
 - making generators, 195
 - methods, 198–199
 - recursive generators, 196–197
 - return statement, 195, 198
 - send method, 198
 - simulating, 199
 - solving Eight Queens problem, 200–206
 - throw method, 199
 - yield expression, 198, 199
 - yield statement, 195, 198
- Gentoo Linux
 - installing Python, 4
- get function, arcade game project, 550, 552
- GET method
 - getting information from CGI script, 335
- get method, dictionaries, 76–78, 585
- get_surface function, arcade game project, 549, 552
- __getattr__ method__, 191, 192
 - raising AttributeError, 192
- getattr function, 157, 159, 582
 - chat server project, 478
 - checking whether object has specific attribute, 173
 - working with getattr method, 192
- __getattribute__ method, 191
 - accessing dict attribute, 192
- getdefaultencoding function, sys module, 451
- gethostname function, socket class, 306, 307
- getitem method, 182
 - changes in Python 3.0, 606
 - overriding, 186
 - simulating slicing, sequences, 185
 - subclassing list type, 186
- getItems method, NewsAgent class
 - news gathering project, 459, 461, 462
- getName accessor method
 - private attributes, 151
- getopt module, 259
 - news gathering project, 468
- getPort function
 - XML-RPC file sharing project, 523, 530
- getrandbits function, random module, 235
- getslice method, Python 3.0, 606
- GetValue method
 - load event, wxPython, 286
 - save event, wxPython, 286
- getvalue method, FieldStorage class
 - input to CGI script, 333
- global keyword, 133, 603
- global scope, 131
 - exceptions and functions, 170
 - rebinding variables in outer scopes, 133
- global statements, 592
- global variables
 - avoiding, 240
 - bugs when referencing, 132
 - constants, 396
 - object-oriented design, 157
 - Python DB API, 294–295
 - rebinding, 132
 - shadowing, 132
 - treating objects as abstract, 146
- globals function, 132, 582
- gmtime function, time module, 234
- Gnutella, 517
- gopherlib module, 310
- Graphical User Interfaces *see* GUIs
- graphics
 - creating graphics in PDF/Python, 425

- graphics creation project, 425–434
 - constructing PolyLine objects, 429–430
 - drawing with ReportLab, 427–429
 - fetching data from Internet, 432
 - further exploration, 434
 - implementations, 427–434
 - preparations, 426
 - prototype for sunspots_proto.py, 430–431
 - ReportLab package, 425
 - tools, 426
 - using LinePlot Class, 432–434
- graphics package, reportlab module, 427
- graphics-generating package
 - graphics creation project, 426
- graphs
 - definitions and further information, 201
- greater than operators, 580
- greedy patterns, 250
 - finding sender of e-mail, 252
- grokking, 218
- Group class, arcade game project, 550
 - clear method, 552
 - update method, 552
- group method
 - connecting to NNTP servers, 455
 - MatchObjects, re module, 248
- group numbers, regular expressions
 - using in substitution string, 249
- groups
 - re module, 247–249
 - Usenet groups, 597
- Grutatxt, 424
- GTK platform
 - PyGTK GUI toolkit, 278
- GUI client project, 537–545
 - Client class, 539, 542
 - further exploration, 545
 - implementations, 538–545
 - ListableNode class, 541
 - preparations, 538
 - requirements, 537
 - tools, 537
- GUI platforms, 291, 277
- GUI toolkits
 - chat server project, 486
 - choosing between, 278
 - description, 277

- for Jython, 290
- list of GUI toolkits for Python, 597
- popular GUI Toolkits for Python, 277
- Swing, 290
- Tk/Tkinter, 289
- wxPython, building text editor, 278–288
- GUIs (Graphical User Interfaces), 291

■ H

- hacking
 - cracking compared, 1
- halting theorem, 361
- halts module, 362
- handle method
 - arcade game project, 560, 562
 - forking and threading, 313
- handle method, Node class
 - XML-RPC file sharing project, 521, 522, 525, 531
- handle_accept method
 - chat server project, 471, 472, 475, 480
- handle_charref method, HTMLParser, 325
- handle_close method
 - chat server project, 475, 480
- handle_comment method, HTMLParser, 325
- handle_data method, HTMLParser, 325, 326
- handle_decl method, HTMLParser, 325
- handle_endtag method, HTMLParser, 325, 327
- handle_entityref method, HTMLParser, 325
- handle_pi method, HTMLParser, 325
- handle_startendtag method, HTMLParser, 325
- handle_starttag method, HTMLParser, 325, 326
- Handler class, instant markup project, 410–411, 418
 - callback/start/end/sub methods, 410, 411
- handler module, xml.sax, 439
- handlers
 - CGI handler, mod_python, 336, 338–339
 - creating content handler, 439–441
 - instant markup project, 409–411
 - mod_python handler framework, 336
 - PSP handler, mod_python, 336, 339–341
 - publisher handler, mod_python, 336, 341–343
- handlers.py, instant markup project, 418
- has_key method, dictionaries, 78, 585, 606
- hasattr function, 157, 159, 582
 - replacing callable function, 115
 - working with getattr method, 192
- hashlib module, 258

- head method, NNTP class, 455, 457
- header file
 - framework for extensions, 377
 - SWIG, 372, 373
- headers
 - writeHeader method, 446
- heading rules
 - instant markup project, 415
- HeadingRule class
 - instant markup project, 416, 420
- HeadlineHandler class
 - XML parsing project, 441
- heapify function, 230, 231
- heappop function, 230, 231
- heappush function, 230
- heapq module, 230–231, 259
- heapreplace function, 230, 231
- heaps, 230–231
- hello method, Node class
 - XML-RPC file sharing project, 520, 521, 524, 526, 528, 531, 534
- help function, 116, 582
 - description, 30, 260
 - exploring modules, 219–220
- help switch, Distutils, 384
- hex function, 582
- hexadecimal numbers, 12
- hidden form elements
 - remote editing with CGI project, 494
- hidden inputs, 510
 - bulletin board project, 513
- host parameter, connect function, 296
- hotshot module, 363
- htaccess file, 338, 339, 341
- HTML
 - automatically marking up plain-text file, 403
 - creating HTML pages, 442–444
 - fixing ill-formed HTML, 322
 - index.html file, 492
 - introduction to, 403
 - parsing, 322
 - writing HTML forms, 334–336
 - XHTML advantages over, 325
- HTMLDestination class
 - news gathering project, 460, 462
- htmllib module
 - parsing HTML, 322
- HTMLParser class, 325–327
 - callback methods, 325
 - event-handling methods, 325
 - screen scraping using, 326
- HTMLRenderer class
 - instant markup project, 409, 411, 419
- httplib module, 310
- 
- %i conversion specifier, 56
- id column, messages table
 - bulletin board project, 502
 - view.cgi script, 508
- id function, 582
- identity operator, 93–94
- IDEs for Python, 6
- IDLE interactive Python shell, 2
 - IDEs for Python, 6
 - saving and executing programs, 19
- if statements, 15, 90, 569, 592
 - catching exceptions, 164
 - caution when return value is None, 117
 - try/except statement compared, 171, 173
- I/O, Python 3.0, 600, 605
- image attribute, Sprite class, 554
- image module, pygame, 551
- imaginary numbers, 18
- imaplib module, 310
- immutability
 - frozenset type, 229
 - lists, 119
 - parameters, 123
 - set type, 229
 - strings/numbers/tuples, 119
 - using is operator with immutable values, 94
- implementation of projects
 - structure of projects in this book, 402
- import command, 17
 - from module import statement, 17
- import statements, 591
 - as clause, 85
 - description, 111
 - fetching functions from external modules, 209
 - importing something as something else, 84–85
 - open function, 85
- importdata.py, 301
- importing modules, 259
- import-only-once behavior, modules, 211

- in operation, 71
- in operator, 38, 39, 94
 - operator precedence, 580
- include_dirs variable, 299
- incompatibility warnings
 - transitioning to Python 3.0, 599
- indentation, 569
 - blocks, 88
- index method, lists, 44, 138, 585
- index method, strings, 587
- index, sequences
 - checkIndex function, 185
 - description, 31
 - illegal type of index used, 184
- index.html file
 - remote editing with CGI project, 492, 496
- IndexError class, 162
 - index outside range, 183, 184
- indexing, lists, 570
- indexing, sequences, 32–34
- inequality operator
 - Python 3.0, 604, 605
- infinite recursion, 134
- inflate method, rectangles, 555, 556
- Info class, arcade game project, 563
- information-gathering agent
 - news gathering project, 453
- inheritance, 147–155
 - description, 141, 159
 - multiple inheritance, 156, 575
 - multiple superclasses, 155–156
 - object-oriented design, 157
 - overriding methods and constructors, 177, 178
 - specifying superclasses, 153–154
 - subclassing list/dict/str types, 185
- init file
 - making Python recognize packages, 217
- init function, pygame module, 549, 552
- __init__ method, 575
 - calling unbound superclass constructor, 180
 - making init magic, 177
 - using super function, 180
- initialization functions
 - naming conventions, 380
- initializing methods *see* constructors
- inner scope *see* local scope
- Inno Setup installer, 388
- inplace parameter, input function, 226, 227
- input
 - compared to raw_input, 26
 - fileinput module, 225–227
 - getting input from users, 14–15
 - hidden inputs, 510
 - stdin stream, sys module, 222
- input function, 14, 30, 570, 582
 - changes in Python 3.0, 600
- input function, fileinput module, 226
 - backup parameter, 226
 - inplace parameter, 226, 227
- insert method, lists, 45, 585
- inside attribute, ListRule class
 - instant markup project, 417
- inside function
 - XML-RPC file sharing project, 530
- inspect module
 - publisher handler, mod_python, 342
 - seeing all values stored in objects, 157
- install command, Distutils, 385
 - compiling extensions, 388
- installations, Python, 1–7
 - on Linux/UNIX, 3–5
 - on Macintosh, 5
 - on Windows, 1–3
- installers
 - alternative installers, 388
 - creating Windows installer, 387
 - Inno Setup, 388
 - introduction, 390
 - McMillan installer, 388
 - Windows installer technology, 388
 - writing install scripts with Distutils, 383–386
- instances of classes, 147
 - isinstance method, 155
- instant markup project, 403–424
 - adding markup, 407–408
 - components, 409
 - filters, 413, 418
 - final program, 418–422
 - finding blocks of text, 406–407
 - further exploration, 423–424
 - goals, 404
 - Handler class, 410–411
 - handlers, 409–411
 - implementations, 406–422

- Parser class, 413–415
- parsers, 413–415
- preparations, 405–406
- Rule class, 413
- rules, 412–413, 415–418
- tools, 404
- int function, 30, 113, 570, 582
- integer type, 579
- integers
 - integer division, 9
 - large integers, 11–12
 - long integers, 11–12
 - numbers containing leading zeros, 70
- IntegrityError exception, Python DB API, 295
- interactive interpreter, 7–9
 - saving programs, 19
- interface file, SWIG, 372, 373
- InterfaceError exception, 295
- interfaces, 156–157, 159
- InternalError exception, 295
- interoperability
 - enabling, Python/C++, 371
- interpreter, interactive, 7–9
- I/O, asynchronous I/O with select and poll, 313–316
- IOError class, 162
- IRC, 595
- IronPython, 595
 - alternative Python distributions, 6, 7
 - description, 380
 - extending Python, 367–369
- is not operator, 580
- is operator, 93–94
 - operator precedence, 580
- isalnum method, strings, 587
- isalpha method, strings, 587
- isdigit method, strings, 587
- isdir function, os.path module
 - XML parsing project, 447
- isfirstline function, fileinput module, 226
- isinstance function, 142, 159, 582
 - using in checkIndex function, 185
- isinstance method, 155
- islower method, strings, 587
- isspace method, strings, 587
- isstdin function, fileinput module, 226
- issubclass function, 159, 582
- issubclass method, 154
- istitle method, strings, 587
- isupper method, strings, 587
- item access, 182–187
 - sequence and mapping protocol, 182–185
 - subclassing list/dict/str types, 185–187
- item access precedence, 580
- items method, dictionaries, 78, 100, 585
 - changes in Python 3.0, 605
- __iter__ method, 192–194, 207
 - for loops, 193
- iter function, 194, 207, 582
- iterable unpacking, Python 3.0, 603, 606
- iteration
 - definition, 192
 - for loops, 99
 - iterable object, 99
 - iterating over dictionaries, 100
 - iterating over file contents, 270–274
 - byte by byte, 270–271
 - file iterators, 272–274
 - iterating over lines in large file, 272
 - lazy line iteration with fileinput, 272
 - one line at a time, 271
 - reading everything first, 271
 - without storing file object in variable, 273
 - iterating over sequences, 32
 - iterating over string-like objects, 197
 - looping, 99
 - numbered iteration, 101
 - parallel iteration, 100–101
 - reversed iteration, 102
 - sorted iteration, 102
 - StopIteration exception, 192
 - utilities, 100–102
 - while loops, 98
- iterator protocol, 192–194
- iterator return values, Python 3.0, 603
- iterators, 192–194
 - changes in Python 3.0, 605
 - description, 207
 - file iterators, 272–274
 - generators and, 195
 - introduction, 175
 - iterator protocol, 192–194
 - making sequences from, 194
 - returning, 197

- iteritems method, dictionaries, 78, 585
- iterkeys method, dictionaries, 78, 585
- itertools module, 258
- itervalues method, dictionaries, 80, 585

J

- Java class, Jython extending Python, 367
- Java Swing *see* Swing GUI toolkit
- JavaBean properties
 - Jython extending Python, 368
- JavaScript Object Notation (JSON), 346
- join function, os.path module
 - XML parsing project, 447
- join method, strings, 61, 587
 - example using, 223
 - performance, 255
- JSON (JavaScript Object Notation), 346
- just-in-time compiler for Python, 370
- Jython, 595
 - alternative Python distributions, 6, 7
 - description, 380
 - extending Python, 367–369
 - GUI toolkits for, 290
 - JavaBean properties, 368
- jythonc command, 367

K

- KDevelop environment, 6
- key argument of sort method, lists, 48
 - changes in Python 3.0, 605
- key related operations, dictionaries, 71
- key types, dictionaries, 71
- KeyError exception, 162
 - trapping with except clause, 172
- keys
 - inappropriate type used, 183
 - sequence key is negative integer, 183
- keys method, dictionaries, 78, 100, 585
 - changes in Python 3.0, 605
- keys, dictionaries, 121
 - checking if key exists, 78
 - types for keys, 72
- keyword arguments/parameters, 123–125
 - ** (keyword splicing) operator, 126
 - combining with positional parameters, 124
 - keyword-only parameters, 602, 606

- sort method, lists, 48
 - using with wx constructors, 282

- Komodo environment, 6

L

- label argument, wx.Button constructor, 282
- lambda expressions, 139
- lambda operator, 579
- languages
 - object-oriented languages, 141
- large integers, 11–12
- LaTeX
 - markup system, 404
 - typesetting system, 434
- layout mechanisms, 291
- lazy evaluation, Boolean operators, 96
- lazy line iteration, 272
- left shift operator, 580
- len function, 40, 52, 582
- __len__ method, 182, 184
- len operation, dictionaries, 71
- less command, UNIX, 457
- less than operators, 580
- letters constant, string module, 60
 - changes in Python 3.0, 606
- Level class, arcade game project, 560
- LevelCleared class, arcade game project, 564
- lib subdirectory, Distutils, 385
- libraries
 - ctypes library, 370
 - difflib library, 258
 - importing existing (shared) C libraries, 370
 - Macintosh library modules, 596
 - Python library reference, 596
 - standard library modules, 259
 - Tidylib, 324
- library_dirs variable, 299
- line method, Canvas class, 427
- line numbers
 - adding to Python script, 227
- line separators *see* newline character
- lineno function, fileinput module, 226
- LinePlot Class
 - graphics creation project, 432–434
- LineReceiver class, Twisted framework
 - chat server project, options for, 469

- LineReceiver protocol
 - twisted.protocols.basic module, 318
- lines
 - constructing PolyLine objects, 429–430
- lines generator
 - instant markup project, 406
- linesep variable, os module, 224
- Linux
 - installing Python on Linux/UNIX, 3–5
- list comprehension, 105–106, 112
 - exploring modules, 218
 - generator comprehension and, 196
 - using, 139
- list constructor
 - making lists from iterators, 194
- list function, 40, 52, 583
- list item rules
 - instant markup project, 415
- list method, ListableNode class
 - GUI client project, 541, 542
- list methods, 43–49, 585
 - append method, 43, 585
 - count method, 43, 585
 - extend method, 44, 585
 - index method, 44, 585
 - insert method, 45, 585
 - pop method, 45–46, 585
 - remove method, 46, 585
 - reverse method, 46, 585
 - sort method, 47–49, 585
- list rules
 - instant markup project, 415
- ListableNode class, GUI client project, 541
 - list method, 541, 542
- listen method
 - chat server project, 471, 472
 - socket class, 306
- listenTCP function, reactor module, 318, 320
- ListItemRule class
 - instant markup project, 416, 420
- ListRule class
 - instant markup project, 417, 420
- lists, 40–49, 570
 - adding items compared to dictionaries, 72
 - appending object to end of, 43
 - appending values to end of, 44
 - assigning to slices, 42
 - assigning values to, 41
 - changing lists, 41
 - copying entire list, 47
 - counting occurrences of elements in, 43
 - deleting elements from, 41
 - deleting slices, 42
 - empty lists, 37
 - finding first occurrence of a value, 44
 - immutability, 119
 - indexing, 570
 - initialization, 37
 - inserting elements, 42
 - inserting object into, 45
 - making lists from iterators, 194
 - making lists from other lists, 105
 - multiple references to same list, 47
 - operations on, 41–42
 - precedence, 581
 - removing an element from, 45
 - removing first occurrence of value, 46
 - reversed function, 47
 - reversing elements in, 46
 - selecting all elements, 35
 - selecting elements from start/end, 35
 - slicing, 119, 571
 - sorted function, 48
 - sorting into new list, 47
 - sorting original list, 47
 - subclassing list type, 185–187
 - tuples compared, 31
- literal values, 579
- ljust method, strings, 587
- load function
 - arcade game project, 551
 - event handler, 286
- local scope, 131
 - parameters, 118
- locals function, 132, 583
- locals module, pygame, 549, 552
- localtime function, 233, 234, 455
- logging, 399, 400, 401
- logging module, 258, 399–400
- logical expressions, 573
 - short-circuit logic, 574
- logical operators *see* Boolean operators
- login command, chat server project, 479, 485, 486
- LoginRoom class, chat server project, 479, 480, 482

- logout command, chat server project, 479, 485, 486
- LogoutRoom class, chat server project, 479, 480, 483
- logRequests value
 - XML-RPC file sharing project, 523
- long function, 30, 583
- long integer type, 579
- long integers, 11–12
 - changes in Python 3.0, 605
- long strings, 26–27
- look command, chat server project, 480, 485
- lookup function, 121
- loop method, chat server project, 472
- loops, 97–105
 - breaking out of, 102–105
 - while True/break idiom, 104, 105
 - description, 112
 - __dict__ attribute avoiding endless looping, 192
 - for loops, 99
 - iterating over dictionaries, 100
 - iteration utilities, 100–102
 - numbered iteration, 101
 - parallel iteration, 100, 101
 - reversed iteration, 102
 - sorted iteration, 102
 - list comprehension, 105–106
 - using else clauses in, 105
 - while loops, 98
- lower method, strings, 62, 95, 241, 457, 587
- lowercase constant, string module, 60
- lstrip method, strings, 587

M

- m switch
 - making programs available as modules, 212
- Mac OS X
 - setting environment variables, 216
- Macintosh
 - installing Python on, 5
- Macintosh library modules, 596
- MacPython, 595
- magic attributes
 - __dict__ attribute, 192
- magic methods, 575
 - advanced use of, 187
 - constructors, 176–181
 - __del__ method, 177
 - __delattr__ method, 191
 - __delitem__ method, 182, 184

- __getattr__ method, 191, 192
- __getattribute__ method, 191, 192
- __getitem__ method, 182
- __init__ method, 177
- introduction, 175, 206
- item access, 182–187
- __iter__ method, 192–194
- iterator protocol, 192–194
- __len__ method, 182, 184
- modules, 576
- __next__ method, 192
- __nonzero__ method, 182
- overriding methods and constructors, 177–179
 - calling unbound superclass constructor, 179–180
 - using super function, 180–181
- property function, 189
- sequence and mapping protocol, 182–185
- __setattr__ method, 191, 192
- __setitem__ method, 182
- subclassing list/dict/str types, 185–187
- mailbox module, 310
- mailcap module, 310
- mailing lists, 597
- main chat room, chat server project, 479
- main function
 - calling from another function, 240
 - unittest module, 356, 364
- main page, bulletin board project, 513
- main value
 - testing modules, 213
- main.cgi script, bulletin board project, 506, 507–508
 - link to edit.cgi, 507
 - link to view.cgi, 507, 508
 - testing, 513
- MainLoop method, wx.App class, 281
- makedirs function, os module
 - XML parsing project, 447
- maketrans function, string module, 60, 65, 66
- MANIFEST.in file, 387
- map function, 138, 140, 583
 - changes in Python 3.0, 605
- mappings
 - constructing dictionaries from, 71
 - deleting element associated with key, 182
 - description, 32, 81, 182, 206
 - dictionaries and, 69

- environ mapping, 223, 224
 - modules mapping, 222
 - returning number of key-value pairs contained in, 182
 - returning value of key, 182
 - sequence and mapping protocol, 182–185
 - storing value for key, 182
- Markdown
- markup systems and web sites, 424
- markup
- instant markup project, 403–424
- markup systems, 424
- markup.py program, instant markup project, 421
- Martelli, Alex, 106
- Python Cookbook, 96
- match function, re module, 245, 246
- MatchObjects class, re module, 247–249
- methods, 248
- Matplotlib/pylab, 434
- max function, 40, 52, 583
- MAX_HISTORY_LENGTH constant
- peer-to-peer file sharing, 522
- maxint value, sys
- changes in Python 3.0, 605
- maxsplit argument
- split function, re module, 246
- McMillan installer, 388
- membership, 51
- checking membership with sets, 228
 - dictionaries, 71, 72
 - sequences, 38–39
- membership operator, 94
- memory leaks, 376
- message composer, bulletin board project, 514
- Message object, email module
- news gathering project, 461
- message viewer, bulletin board project, 514
- message_from_string function
- news gathering project, 461
- messages table, bulletin board project
- columns described, 502
 - creating, 501
- __metaclass__ attribute
- creating new-style classes, 206
 - finding out class of an object, 155
 - new-style/old-style classes, 175
 - old-style and new-style classes, 149
 - property function, 188
- metaclass syntax, Python 3.0, 602, 605
- metaclasses, 176
- Metakit, 515
- method resolution order (MRO), 156
- methods, 51, 573
- see* individual method names
 - accessor methods, 151, 187
 - arguments, 573
 - bound methods, 150
 - calling, 43, 573
 - calling overridden version, 206
 - class methods, 189–191
 - constructors, 176–181
 - dictionaries, 585–586
 - distinguishing methods from functions, 150
 - functions compared, 43
 - generator methods, 198–199
 - list methods, 43–49
 - lists, 585
 - magic methods *see* magic methods
 - making method or attribute private, 151
 - MatchObjects, re module, 248
 - method definitions, 149
 - object-oriented design, 157, 158
 - overriding, 177–179
 - overriding in subclasses, 148, 156
 - polymorphism and, 143
 - static methods, 189–191
 - string methods, 60–66
 - strings, 586–588
- methods, files, 263–270
- mhlib module, 310
- microthreads, 312
- min function, 40, 52, 583
- minimum field width, string formatting, 56, 57, 59
- mix-in classes, 159
- dispatcher mix-in classes, 444–446
- mktime function, time module, 233, 234
- mod_python framework, 336–343, 347
- CGI handler, 336, 338–339
 - configuring Apache, 338
 - installing, 337
 - PSP handler, 336, 339–341
 - publisher handler, 336, 341–343
- mode argument, open function (files), 261–263
- modes, files, 261–263, 274

- modulator tool, 371
 - modules, 29, 209–221, 259, 576
 - see* individual module names
 - `__all__` variable, 219
 - “batteries included” phrase, 221
 - checking if module exists, 218
 - creating and locating, 209–210
 - defining functions in, 212–214
 - `dir` function, 218
 - documentation, 220, 221
 - exploring, 218–221
 - extinct modules in Python 3.0, 604
 - help function, 219–220
 - import statement, 209
 - importing, 17, 209, 210, 259
 - import-only-once behavior, 211
 - magic methods, 576
 - main purpose of, 210
 - making code reusable, 212
 - making modules available, 214–216
 - making programs available as, 212
 - mapping, `sys` module, 222
 - modifying `sys.path` to specify location, 210, 214
 - naming file containing module code, 217
 - networking, 305–310
 - newly released third-party modules, 7
 - packages, 259
 - packaging in packages, 217
 - packaging in Python 3.0, 604
 - permissions affecting save location, 214
 - programming, 17–19
 - putting modules in existing `sys.path`, 214–215
 - reasons for not doing, 215
 - `py_modules` directive, `Distutils`, 386
 - `pyc` file extension, 210
 - `pygame` modules, 548–551
 - reading source code, 221
 - reloading, 211
 - renamed modules in Python 3.0, 604
 - specifying module location in `PYTHONPATH`, 215–216
 - standard library modules, 221–259
 - switching between database modules, 294
 - test code contained in, 259
 - testing, 212–214
 - third-party modules, 597
 - twisted modules, 317, 318
 - wrapping modules as archive file, 386–387
 - writing extension modules for Python, 370
 - modulo operator
 - list comprehension, 105
 - string formatting, 66
 - modulus operator, 11
 - mouse cursor
 - `pygame.mouse` module, 550
 - mouse module, `pygame`, 550
 - `move` method, `rectangles`, 555
 - MRO (method resolution order), 156
 - `µTidyLib`, 324
 - multiple connections
 - network programming, 311–316
 - multiple inheritance, 156, 575
 - multiple superclasses, 155–156
 - multiplication operator, 37, 580
 - multiplying, sequences, 37–38
 - mutable objects *see* immutability
 - `mxTidy`, 324
 - MySQL database
 - bulletin board project, 500, 501
 - `MySQLdb` module, 597
 - bulletin board project, 500
- ## N
- name variable, testing modules, 213
 - name argument, open function (files), 261
 - named arguments, 572
 - named value, Python DB API, 295
 - `NameError` class, 162
 - namespaces
 - see also* scopes
 - class namespace, 152–153
 - class statement, 149
 - number of namespaces, 131
 - using `exec` and `eval`, 109, 111
 - naming conventions
 - classes, 148
 - making method or attribute private, 152
 - initialization functions, 380
 - object-oriented design, 158
 - symbolic constants, 396
 - variables, 13
 - `wx` module methods, 281
 - nan value, 18
 - negative numbers
 - `sqrt` function, 18

- nested scopes, 133
 - instant markup project, 410
- nesting blocks, if statements, 91
- Network News Transfer Protocol *see* NNTP
- network programming
 - asynchronous I/O with select and poll, 313–316
 - chat server project, 469
 - event-driven networking framework, 316
 - firewalls, 305
 - forking and threading with SocketServer, 313
 - introduction, 305
 - multiple connections, 311–316
 - opening remote files, 308
 - port numbers, 307
 - retrieving remote files, 309
 - sockets, 470
 - synchronous network programming, 306
 - Twisted framework, 316–319
- networking modules, 305–310
- newline character
 - changes on opening in text mode, 262
 - platforms using other line separators, 267
- NEWNEWS command
 - NNTP server supporting, 454, 456
- newnews method, NNTP class, 455, 456
- news gathering project, 453–468
 - automatically generated news page, 461
 - downloading messages from newsgroups, 455
 - flexible news-gathering agent, 463
 - further exploration, 467
 - goals, 454
 - implementations, 455–467
 - NewsAgent class, 459
 - news page with more than one source, 463
 - preparations, 454–455
 - tools, 454
- newsgroups
 - downloading messages from, 455
 - online resources, 597
- NewsItem class
 - news gathering project, 459
- newsreaders *see* NNTP clients
- `__next__` method, 192
 - changes in Python 3.0, 606
 - iter method returning iterator, 192, 207
 - object implementing, 193
- nextfile function, fileinput module, 226
 - finding sender of e-mail, 253
- nextset method, cursors, 297
- nextState method, LevelCleared class
 - arcade game project, 564
- nlargest function, heapq module, 230, 231
- NNTP (Network News Transfer Protocol), 453
- NNTP class
 - body method, 455, 457
 - head method, 455, 457
 - instantiating, 455
 - newnews method, 455, 456
- NNTP clients, 453
- NNTP constructor
 - connecting to servers, 454
- NNTP servers
 - '211' string beginning, 455
 - '411' string beginning, 455
 - connecting to servers, 454
 - description, 453
 - downloading messages from newsgroups, 455
 - main network of, 453
 - news gathering project, 454
- nntplib library
 - news gathering project, 453, 454, 455
- nntplib module, 310
- NNTPSource class
 - news gathering project, 461, 462
- Node class, GUI client project, 541
 - updateList method, 541, 543
- Node class, XML-RPC file sharing project, 520, 530
 - broadcast method, 521, 522, 525, 532
 - constructor, 520
 - fetch method, 520, 521, 524, 531
 - handle method, 521, 522, 525, 531
 - hello method, 520, 521, 524, 531
 - implementing, 520–525
 - query method, 520, 521, 522, 524, 530
 - start method, 523, 524, 531
 - stopping and restarting node, 527
- None value, 37
 - changes in Python 3.0, 605
 - return value, functions, 117
 - using None as placeholder, 574
- None value, Boolean values, 89
- nongreedy patterns *see* greedy patterns

- nonlocal keyword, 133
 - Python 3.0, 602, 606
 - nonzero method, 182
 - nose
 - alternatives to unit test tools, 355
 - not equal to operator, 580
 - not in operator, 580
 - not operator, 96, 580
 - NotImplementedError exception, 224
 - NotSupportedError exception, 295
 - nsmallest function, heapq module, 230, 231
 - NUMBER value, Python DB API, 298
 - numbered iteration, 101
 - numbers, 9–12
 - complex numbers, 18
 - floating-point numbers, 10
 - hexadecimal numbers, 12
 - imaginary numbers, 18
 - immutability, 119
 - nan value, 18
 - numbers containing leading zeros, 70
 - octal numbers, 12
 - numeric arrays
 - analyzing many forms of numeric data, 370
 - numeric value, Python DB API, 295
 - NumPy, 370, 548
- O**
- %o conversion specifier, 56
 - object function, 583
 - object-oriented design, 157–158, 159
 - object-oriented languages, 141
 - Smalltalk, 151
 - object-oriented programming *see* OOP
 - objects, 572–575
 - accessing attributes of objects, 150–152
 - checking if object has specific attribute, 172
 - classes, 147–156
 - classes and objects, 147
 - deleting, 107
 - description, 141, 158
 - encapsulation, 573, 145–147
 - exception objects, 161
 - finding out class of an object, 155
 - file-like objects, 263, 274
 - inheritance, 575, 147–154, 155
 - MatchObjects, re module, 247
 - methods, 573
 - object-oriented design, 157
 - polymorphism, 142–145
 - forms of, 144
 - methods and, 143
 - private attributes, 151
 - referencing not owning, 376
 - referring to the object itself, 149
 - seeing all values stored in objects, 157
 - treating objects as abstract, 146
 - objects.py file, arcade game project, 556, 557
 - oct function, 583
 - octal literals, Python 3.0, 606
 - octal numbers, 12
 - Official Python Distribution, 595
 - offset parameter
 - seek method, files, 266
 - OnInit method, Client class
 - GUI client project, 538, 539, 541, 543
 - online chatting *see* chat server project
 - online resources, 595–597
 - OOP (object-oriented programming)
 - classes, 147–156, 158
 - distinction between types and classes, 148
 - encapsulation, 141, 145–147, 158
 - inheritance, 141, 147–154, 155, 159
 - interfaces, 156–157, 159
 - objects, 141, 158
 - polymorphism, 141, 142–145, 158
 - forms of, 144
 - summary of key concepts, 158–159
 - open function, files, 261–263, 275
 - binary mode, 262
 - buffering argument, 263
 - mode argument, 261–263
 - name argument, 261
 - open function, import statement, 85
 - open function, shelve module, 238
 - open function, webbrowser module, 225
 - opening files, 261–263
 - open function, 583
 - operating systems
 - os module, 223–225
 - OperationalError exception, 295
 - operations
 - dictionaries, 71–73
 - lists, 41–42

- sequences, 32–40
 - adding, 37
 - checking membership, 38–39
 - indexing, 32–34
 - multiplying, 37–38
 - slicing, 34–37
 - strings, 53
 - tuples, 50
 - operator module
 - add function, 144
 - operators
 - * (parameter splicing) operator, 126, 127
 - ** (keyword splicing) operator, 128
 - += operator, 522
 - adding sequences, 37
 - arithmetic operators, 9
 - assignment operator, 13, 15
 - Boolean operators, 38, 95–96
 - comparison operators, 92–95
 - conditional operator, 96
 - distributing operators, 128–129, 604
 - double slash operator, 10
 - equality operator, 15, 93
 - exponentiation operator (**), 11
 - gathering operators, 125–128, 602, 603, 604, 606
 - identity operator, 93–94
 - in operator, 38, 39, 94
 - is operator, 93–94
 - logical operators, 96
 - membership operator, 94
 - modulo operator, 66
 - modulus operator, 11
 - multiplying sequences, 37
 - parameter operators, 125–129
 - precedence, 579–581
 - repetition operators, 250
 - splicing operators, 129
 - string formatting operator, 53
 - ternary operator, 96
 - optimization
 - extending Python for speed, 365–366
 - profiling, 362–363
 - optparse module, 259
 - levels of configuration, 398
 - news gathering project, 468
 - or operator
 - Boolean operators, 96
 - short-circuit logic, 574
 - operator precedence, 579
 - finding union of two sets, 228
 - ord function, 95, 112, 583
 - os module, 223–225, 259
 - functions and variables, 223
 - makedirs function, 447
 - os.path module, XML parsing project, 447
 - outer scope *see* global scope
 - output
 - piping output, 264–265
 - stdout stream, sys module, 222
 - overriding
 - description, 206
 - getitem method, 186
 - methods and constructors, 177–179
 - calling unbound superclass constructor, 179–180
 - using super function, 180–181
 - methods in subclasses, 148, 156
- ## P
- package manager
 - installing Python on Linux/UNIX, 4
 - packages, Python, 217–218
 - announcing/publishing, 390
 - centralized index of, 390
 - description, 259
 - distributing, 383
 - files and directories layout, 217
 - graphics-generating package, 426
 - grouping modules in, 217
 - making Python recognize, 217
 - module packaging in Python 3.0, 604
 - Python Package Index, 384
 - packaging programs
 - creating Linux RPM packages, 387
 - creating Windows installer, 387
 - distribution formats, 387
 - Distutils, 383–386
 - introduction, 383
 - wrapping modules as archive file, 386–387
 - page element, XML parsing project, 437, 442
 - painting pretty picture project *see* graphics creation project

- palindrome module
 - hand-coded using Python/C API, 379–380
- palindromes, 372
 - program to recognize, 372–375
- Panel class, wx module *see* wx.Panel class
- ParagraphRule class
 - instant markup project, 417, 421
- parallel iteration, 100–101
- parameter operators, 125–129
 - Python 3.0, 602, 603, 604, 606
- parameters, functions, 16, 117–130
 - actual parameters, 118
 - annotation, 601, 605
 - arguments, 118
 - changing, 118–120
 - collecting parameters, 125–128
 - combining positional/keyword parameters, 124
 - default values, 124
 - description, 139
 - distributing operators, 128–129, 604
 - examples using parameters, 129–130
 - formal parameters, 118
 - gathering operators, 125–128, 602, 603, 604, 606
 - immutability, 123
 - keyword parameters, 123–125, 602, 606
 - keyword-only parameters, 602, 606
 - local naming, 120
 - local scope, 118
 - modifying parameters, 123
 - parameter operators, 125–128
 - passing parameters to functions, 572
 - positional parameters, 123
 - default values, 124
 - reasons for changing, 120–122
 - rebinding parameters, 123, 572
 - self parameter, 150
 - values, 118
- paramstyle property, Python DB API, 294, 295
- parent argument, wx constructors, 281
- parse function, xml.sax module, 439
- parse method, instant markup project, 414
- Parser class, instant markup project, 413–415, 421
 - addFilter method, 414, 415
 - addRule method, 414, 415
 - parse method, 414
- parsers, instant markup project, 409, 413–415
- parsing XML project *see* XML parsing project
- parsing, HTML, 322
 - HTMLParser class/module, 325–327
- partition method, strings, 587
- pass statements, 107, 112, 590
- passthrough variable
 - XML parsing project, 448
- password handling, 494
- password parameter
 - connect function, Python DB API, 296
- passwords
 - bulletin board project, 503
- path configuration files, 216
- path submodule, os module, 223, 447
- path variable, sys module, 222
 - modifying to specify module location, 210, 214
 - putting modules in existing sys.path, 214–215
 - reasons for not doing, 215
 - search path (list of directories), 214
 - using PYTHONPATH alongside, 216
- pathsep variable, os module, 223, 224
- patterns
 - greedy patterns, 250
 - re module functions, 245
- Paused class, arcade game project, 561
- PDF (Portable Document Format) files
 - editing, 425
 - getting PDF reader, 425
 - graphics creation project, 425
- pdfgen module, ReportLab package, 427
- PDFs, drawing with ReportLab, 427
- peer-to-peer file sharing
 - GUI client project, 544
 - MAX_HISTORY_LENGTH constant, 522
 - XML-RPC file sharing project, 517–535
- peer-to-peer systems, 517
- performance
 - join method, strings, 255
 - using CGI handler, 339
- period (dot) character
 - regular expression wildcards, 242
- permissions
 - dynamic web pages with CGI, 329–330
 - saving modules, 214
- Pilgrim, Mark, 345
- pipe characters, 265
- piping output, files, 264–265

- placeholders
 - using None as placeholder, 574
- PlainDestination class
 - news gathering project, 460, 462
- plain-text markup, 403
 - markup systems, 424
- platform variable, sys module, 222
- platforms, GUI, 277
- Platypus, ReportLab package, 434
- playful programming, 393
- Plone, 344
- poll function, select module, 320
 - asynchronous I/O with, 315–316
- poll object
 - register/unregister methods, 315
- POLLXYZ events
 - polling event constants in select module, 315
- PolyLine class
 - constructing PolyLine objects, 429
- polymorphism, 142–145
 - description, 141, 158, 182
 - duck typing, 145
 - forms of, 144
 - interfaces, 156
 - isinstance function or, 155
 - methods and, 143
 - repr function, 145
 - subclassing list/dict/str types, 185
 - types, 145
 - use of isinstance function, 185
- pop functions
 - heappop function, 230, 231
- pop method, dictionaries, 79, 586
- pop method, lists, 45–46, 585
- popen function, 224
 - running Tidy, 324
- popitem method, 79, 586
 - sequence unpacking, 86
- poplib module, 310
- port numbers
 - chat server project, 470, 471, 473
 - network programming, 307
 - numbers requiring administrator privileges, 470
- pos argument, wx.Button constructor, 283
- positional parameters
 - combining with keyword parameters, 124
 - default values, 124
 - description, 123
 - gathering operators, 125–128, 602, 603, 604, 606
 - keyword-only parameters, 602, 606
- POST method
 - getting information from CGI script, 335
 - remote editing with CGI project, 490
- PostgreSQL database
 - bulletin board project, 500, 501
- pound bang (#), 21, 22
 - dynamic web pages with CGI, 329
- pow function, 16, 30, 583
- power (exponential) operator, 11
- power function, 135
- powers
 - recursive function to calculate, 135
- pprint function, 215
- precedence, operators, 579–581
- precision, string formatting, 54, 56, 57
- preparations for projects
 - structure of projects in this book, 402
- print function, Python 3.0, 605
- print statement
 - changes in Python 3.0, 600
- print statements, 14, 111, 590
 - separating with commas, 83–84
- printable constant, string module, 60
- printing
 - pretty-printing function, 215
 - using arguments in reverse order, 223
- priority queues
 - heaps, 230–231
- private attributes, 151
- problem descriptions for projects
 - structure of projects in this book, 401
- procedures
 - functions without return values, 117
 - remote procedure calls
 - REST and RPC, 346
 - SOAP, 346
 - with Pyro, 346
 - XML-RPC, 345
- profile module, 258, 363
 - run method, 363, 364
- profiling, 359, 362–363, 364
 - hotshot/profile/timeit modules, 363

programming

see also OOP (object-oriented programming)

- algorithms, 9
- books about programming, 400–401
- built-in functions, 16, 581–584
- comments, 22
- configuration, 394, 396–398, 401
- configuration files, 396–398
- dictionary methods, 74–80, 585–586
- expressions, 9–12, 579–588
- flexibility in, 393–394, 401
- functional programming, 138–139, 140
- functions, 16
- input, 14–15
- list methods, 43–49, 585
- literal values, 579
- logging, 399–400, 401
- making scripts behave like programs, 20–22
- minimum requirements, 400
- modules, 17–19
- operator precedence, 579–581
- playful programming, 393
- prototyping, 394–395, 401
- pseudocode, 136
- Python reference, 579–594
- Python tutorial, 569–577
- requirement specification, 350–351
- saving and executing programs, 19–20
- statements, 13–14, 589–594
- string methods, 60–66, 586–588
- strings, 22–29
- symbolic constants, 396
- test-driven programming, 349–352, 364
- testing, 394
- text editor, 19
- variables, 13

ProgrammingError exception, Python DB API, 295

programs

- abstraction and program structure, 114
- building Windows executable programs, 383
- creating executables with py2exe, 389–390
- description, 29
- importing programs as modules, 209
- making programs available as modules, 212
- packaging, 383
 - Distutils, 383–386

projects

- arcade game project, 547–567
- bulletin board project, 499–515
- chat server project, 469–487
- graphics creation project, 425–434
- GUI client project, 537–545
- instant markup project, 403–424
- news gathering project, 453–468
- remote editing with CGI project, 489–498
- structure of projects in this book, 401
- XML parsing project, 435–452
- XML-RPC file sharing project, 517–535

properties

- accessor methods defining attributes, 187–188
- creating properties, 188
 - `__getattr__`/`__setattr__` methods, 191–192
 - property function, 188–189
- implementing with old-style classes, 191
- introduction, 175
- new-style/old-style classes, 175

property function, 188–189, 207, 583

- calling with arguments, 189
- descriptor protocol, 189
- `__get__`/`__set__` methods as attributes of, 188
- magic methods, 189
- new-style/old-style classes, 206

proportion argument

- Add method, `wx.BoxSizer` class, 285

protocol attribute, Factory class

- writing Twisted server, 317

protocol module, 317

protocols, 182

- descriptor protocol, 189
- iterator protocol, 192–194
- sequence and mapping protocol, 182–185

prototyping, 394–395, 401

- case against rewriting, 395
- extending Python for improved speed, 365

pseudocode, 136, 569

PSP (Python Server Pages), 339

- PSP tags, 340

psp file name extension, 340

PSP handler, `mod_python`, 336, 339–341

Psyco, 370

psycopg module, 597

- bulletin board project, 500

pth file extension, 216

- publisher handler, `mod_python`, 336, 341–343
- pump function, arcade game project, 550
- punctuation constant, `string` module, 60
- push functions
 - heappush function, 230
- push method, chat server project, 475, 479
- PuTTY software, 471
- py file extension, 576
 - naming file containing module code, 217
 - running CGI script, 339
- py.test
 - alternatives to unit test tools, 355
- `Py_BuildValue` function, 378, 381
- `Py_DECREF` macro, 376, 381
- `Py_INCREF` macro, 376, 378, 381
- `py_modules` directive, `Distutils`, 386
- `Py_None` object, 378
- py2exe extension
 - building Windows executable programs, 383
 - `Distutils`, 389–390
 - Inno Setup installer, 388
- `PyArg_ParseTuple` function, 378, 381
- `PyArg_ParseTupleAndKeywords` function, 378, 381
- pyc file extension, 210
- pychecker/pylint commands, 360
- `PyChecker/PyLint` tools, 359–362, 364
- `PyCXX`, 371
- pyformat value, Python DB API, 295
- Pygame documentation, 547
- pygame module, 548, 597
- pygame modules, functions of
 - `blit`, 549
 - `flip`, 549, 552
 - `Font`, 550
 - `get`, 550, 552
 - `get_surface`, 549, 552
 - `init`, 549, 552
 - `load`, 551
 - `pump`, 550
 - `set_caption`, 549
 - `set_mode`, 549, 552
 - `set_visible`, 552
 - `Surface`, 548
 - `update`, 549, 552
- Pygame tool, arcade game project, 548–551
 - catching Pygame-specific errors, 549
- `pygame.display` module, 549, 552
- `pygame.event` module, 550, 552
- `pygame.font` module, 550
- `pygame.image` module, 551
- `pygame.locals` module, 549
 - importing constants from, 552
- `pygame.mouse` module, 550, 552
- `pygame.sprite` module, 550
- PyGTK GUI toolkit, 278, 597
- pylab, `Matplotlib`, 434
- Pylons, 343, 344
- `PyObject` type, 377
- PyPI (Python Package Index), 384, 390, 596
- PyPy, 370
- PyQt GUI toolkit, 278, 597
 - SIP tool, 371
- Pyrex, 370
- Pyro
 - remote procedure calls with, 346
- pyRXP
 - dealing with XML in Python, 439
- `PySimpleApp` class, `wx` module *see* `wx.PySimpleApp` class
- PySQLite, 298, 304
 - downloading and installing, 299
- Python
 - see also* programming
 - adding line numbers to script, 227
 - alternative distributions, 5–7
 - built-in functions, 16, 581–584
 - `cmath` module, 18
 - comments, 22
 - compiling from sources, 4–5
 - converting values to strings, 24
 - creator of, 19
 - dictionary methods, 74–80, 585–586
 - distinction between types and classes, 148
 - enabling interoperability Python/C++, 371
 - expressions, 9–12, 579–588
 - extending, 365–366
 - extension approaches, 380
 - IronPython, 367–369, 380
 - Jython, 367–369, 380
 - Python/C API, 375–380
 - SWIG, 371–375, 380
 - writing C extensions, 369–380
 - functional programming, 140
 - functions, 117

Python (*continued*)

- GUI platforms for, 291
 - GUI toolkits, 597
 - IDEs for Python, 6
 - IDLE interactive Python shell, 2
 - including C/C++ directly in Python code, 370
 - installing on Windows, 1–3
 - installing Python on Linux/UNIX, 3–5
 - installing Python on Macintosh, 5
 - interactive interpreter, 7–9
 - interpreter, 9
 - just-in-time compiler for, 370
 - large integers, 12
 - list methods, 43–49, 585
 - literal values, 579
 - making scripts behave like normal programs, 20–22
 - mod_python, 336–343
 - modules, 17–19
 - operator precedence, 579–581
 - popular GUI Toolkits for, 277
 - private attributes, 151
 - release information, 7
 - RPython, 370
 - running scripts from command prompt, 20
 - Stackless Python, 312
 - statements, 13–14, 589–594
 - string methods, 60–66, 586–588
 - strings, 22–29
 - third-party modules, 597
 - web application frameworks, 343
 - web tutorial, 569
 - writing extension modules for, 370
- Python 3.0, 599–606
- abstract classes, 601
 - argument splicing, 604
 - automatic refactoring tool (2to3), 599
 - class decorators, 601
 - comparing incompatible types, 604
 - console I/O, 600
 - dictionary comprehension, 603
 - dictionary views, 603
 - extinct functions, 604
 - extinct modules in, 604
 - function annotation, 601
 - inequality operator, 604
 - integer division, 604
 - iterable unpacking, 603
 - iterator return values, 603
 - keyword-only parameters, 602, 606
 - metaclass syntax, 602
 - module packaging in, 604
 - new features in, 605
 - nonlocal variables, 602
 - renamed modules in, 604
 - set comprehension, 603
 - set syntax, 604
 - sources of information for, 606
 - standard library, 604
 - string formatting, 600
 - strings/bytes/encodings, 599
 - transitioning from older code to, 599
- Python/C API
- creating built-in types and classes, 378
 - deallocating objects, 376
 - extending Python using, 375–380
 - format strings, 378
 - framework for extensions, 377–378
 - hand-coded palindrome module, 379–380
 - reference counting, 376–377
 - writing extension modules for Python, 370
- python command, 3, 265
- Python Cookbook, 596
- Alex Martelli, 96
- Python Database API (Python DB API), 294–298
- apilevel property, 294
 - bulletin board project, 502, 503
 - connections, 296, 502
 - constructors and special values, 297
 - cursors, 296–297, 503
 - description, 293, 303
 - exceptions, 295
 - global variables, 294–295
 - paramstyle property, 294, 295
 - switching between database modules, 294
 - threadsafety property, 294
 - types, 297–298
- Python distributions online, 595
- Python documentation online, 596
- Python eggs, 384
- Python Enhancement Proposals, 596
- Python extensions, 380
- Python help (pyhelp.cgi), 596
- “Python Imaging Library not available” warning, 429

- Python interpreter
 - extending and embedding, 596
- Python library reference, 596
- Python Package Index *see* PyPI
- Python reference, 579–594
- Python reference manual, 596
- Python Server Pages *see* PSP
- Python web site, 595
- PythonDebug directive, 339, 340
- Python/C API reference manual, 596
- PYTHONPATH environment variable, 215–216, 224, 225
- Pythonwin environment, 6
- PythonWin GUI toolkit, 278, 597
- pyw file extension, 217, 288
- pywin32, 595
- PyX package, 426, 434
- PyXML module, 597
 - installing, 437

Q

- qmark value, Python DB API, 295
 - importing data into databases, 302
- Qnew command-line switch, 10
- Qt platform, PyQt GUI toolkit, 278
- query method, Node class
 - XML-RPC file sharing project, 520, 521, 522, 524, 526, 530
- queues
 - deque, 231–232
 - heaps, 230–231
- QUIT event
 - arcade game project, 552
- quit function, servers, 457
- Quixote, 344
- quote/quote_plus functions, urllib module, 309, 320
- quotes
 - escaping quotes, 23–24
 - single-quoted strings, 23–24

R

- %r conversion specifier, 57
- raise statement, exceptions, 162–163, 173, 591
 - changes in Python 3.0, 605
- raising exceptions, 161–163, 173
 - raising exceptions again, 164–165
- random access, files, 266

- random data
 - urandom function, 224
- random function, random module, 235
- random library
 - choice function, 144
- random module, 234–238, 260
 - choice function, 159
- randomString function
 - XML-RPC file sharing project, 528, 532
- randrange function, random module, 235, 236
- range function, 99, 100, 101, 112, 583
- raw strings, 27–28
- raw_input
 - compared to input, 26
- raw_input function, 30, 570, 583
 - changes in Python 3.0, 600, 606
 - ignoring return value, 238
 - reading strings, 113
- rawDataReceived event handler, 318
- RDF (Resource Description Framework), 345
- RDF Site Summary, 345
- re module, 242–257, 260
 - see also* regular expressions
 - compile function, 245
 - escape function, 245, 247
 - findall function, 245, 246, 252
 - finding sender of e-mail, 251–253
 - flags parameter, 247
 - functions, 245
 - VERBOSE flag, 249
 - groups, 247–249
 - match function, 245, 246
 - MatchObjects, 247–249
 - sample template system, 253–257
 - screen scraping, 321
 - search function, 245
 - split function, 245, 246
 - sub function, 245, 247, 249, 250
 - using group numbers in substitution string, 249
- reactor module
 - listenTCP function, 318
- read method, files, 264
 - examples using file methods, 268
 - iterating over file contents with, 270
 - reading entire file before iterating, 271
- read mode, open function (files), 262

- reading files, 264, 274
 - closing files after reading, 267
- reading lines, files, 266, 274
- readline method, files, 266
 - examples using file methods, 269
 - iterating over file contents with, 271
- readlines method, files, 266
 - examples using file methods, 269
 - reading entire file before iterating, 271
 - xreadlines method and, 272
- rebinding
 - global variables, 132
 - local and global scopes, 131, 132
 - variables in outer scopes, 133
- receiveItems method, NewsAgent class, 459
- rect attribute, Sprite class, 554
- rectangle objects
 - clamp method, 556
 - inflate method, 555, 556
 - move method, 555
- recursion, 133–139
 - infinite recursion, 134
 - parts of recursive function, 134
 - recursive definitions, 134
 - recursive functions, 134, 140
 - binary search example, 136–138
 - calculating factorials example, 134
 - calculating powers example, 135
 - value of, 136
- recursive generators, 196, 197
- recv method, socket class, 307
- reduce function, 139, 140, 583
 - Python 3.0, 604, 606
- reduce method, set type, 229
- refactoring
 - 2to3 (automatic refactoring tool), 599
 - news gathering project, 453
- reference counting
 - borrowed references, 376, 377
 - deallocating objects, 376
 - decrementing reference count, 376
 - extending Python using Python/C API, 376–377
 - garbage collection, 377
 - incrementing reference count, 376
- references
 - Python library reference, 596
 - Python reference, 579–594
 - Python reference manual, 596
 - Python/C API reference manual, 596
- REFERENCES keyword, PostgreSQL
 - CREATE TABLE command, 501
- register command, Distutils, 390
- register method, poll object, 315
- register_function method
 - SimpleXMLRPCServer class, 519
- register_instance method
 - SimpleXMLRPCServer class, 519, 523
- regular expressions
 - see also* re module
 - character sets, 243
 - denoting beginning/end of string, 244
 - description, 242
 - escaping special characters, 242
 - findall method, news gathering project, 462
 - finding sender of e-mail, 252
 - instant markup project, 409, 411
 - making readable, 249
 - re module, 242–257, 260
 - repeating patterns, 244
 - sample template system, 253–257
 - screen scraping, 322
 - specifying alternative matches, 243
 - subpatterns, 243–244
 - transforming into pattern object, 245
 - wildcards, 242
- relational databases
 - tutorial/reading on, 293
- release information, 7
- reload function, 260, 583
 - modules, 211
 - Python 3.0, 604, 606
 - replacing functionality using exec, 211
- remainder operator, 580
- remote editing with CGI project, 489–498
 - controlling file access, 493
 - debugging, 490
 - edit.cgi script, 492–494, 496
 - further exploration, 497
 - index.html file, 492, 496
 - implementations, 490–496
 - preparations, 490
 - requirements, 489
 - running the editor, 496
 - save.cgi script, 492, 494–495, 496

- tools, 490
- view.cgi script, 497
- remote procedure calls *see* RPC
- remove method, chat server project, 479
- remove method, lists, 46, 585
- remove method, set type, 229
- renderPDF class
 - drawToFile method, 428
- RenderUpdates class
 - draw method, 549, 550
- repetition operators, 250
- replace method, strings, 63, 587
- reply_to column, messages table
 - bulletin board project, 502, 504, 506
 - edit.cgi script, 510
 - testing, 513
 - view.cgi script, 508
- reportlab module, 597
 - graphics package, 427
 - importing, 426
- ReportLab package
 - constructing PolyLine objects, 429–430
 - description, 425
 - documentation for, 426
 - downloading, 426
 - drawing with, 427–429
 - first prototype for sunspots_proto.py, 430–431
 - LinePlot class, 432–434
 - pdfgen module, 427
 - Platypus, 434
 - reasons for choosing, 426
- repr function, 25, 30, 573, 583
 - polymorphism, 145
- representational state transfer (REST), 346
- requirement specification
 - functional requirements, 350
 - test-driven programming, 350–351
- reset method, Weight class
 - arcade game project, 558
- Resource Description Framework (RDF), 345
- resources
 - online resources, 595
 - Python 3.0, 606
- REST (representational state transfer), 346, 535
- reStructuredText, 424
- return statement, 116, 572, 590
 - ending functions, 117
 - generators, 195, 198
 - infinite recursion, 134
- return value, functions, 16
 - annotation, 601, 605
 - functions without return values, 117
 - iterator return values, Python 3.0, 603
 - return value is None, 117
 - caution using if statements, 117
- reverse argument of sort method, lists, 49
- reverse function, 223
- reverse method, lists, 46, 585
- reversed function, 47, 52, 102, 112, 223, 583
- reversed iteration, 102
- rewriting
 - case against rewriting, 395
- rfind method, strings, 587
- Rich Site Summary, 345
- right shift operator, 580
- rindex method, strings, 587
- rjust method, strings, 588
- robotparser module, 310
- rollback method, connections, 296
- Room class, chat server project, 481
- rooms, chat server project, 478–480
 - LoginRoom class, 479
 - LogoutRoom class, 479
 - main chat room, 479
- Rossum, Guido van, 278
- round function, 16, 30, 584
- rounding, division, 16
- rowcount attribute, cursors, 297
- ROWID value, Python DB API, 298
- rpartition method, strings, 588
- RPC (remote procedure calls)
 - REST and RPC, 346
 - SOAP, 346
 - XML-RPC, 345
- rpm format
 - bdist command, Distutils, 387
- RPMs
 - creating Linux RPM packages, 387
 - XML parsing project, 437
- RPython, 370
- rsplit method, strings, 588
- RSS (Really Simple Syndication), 345
- RSS feeds, 345
 - client program handling feeds, 345

RSS feeds (*continued*)

Scrape 'N' Feed, 328

Universal Feed Parser, 345

rstrip method, strings, 227, 588

Rule class/object, instant markup project, 413, 419

condition/action methods, 412, 414

rules, instant markup project, 409, 412–413, 415–418

run function, reactor module, 318, 320

run method, Game class

arcade game project, 565

run method, profile module, 363, 364

runDefaultSetup function

news gathering project, 462

S

%s conversion specifier, 54

safe_substitute method, 55

sample function, random module, 235

save function, event handler, 286

save.cgi script

bulletin board project, 507, 511–513

remote editing with CGI project, 492, 494–495, 496

saving programs, 19–20

SAX (Simple API for XML), 435

dealing with XML in Python, 439

XML parsing project, 435, 438, 442

sax module, xml

parse function, 439

SAX parser

XML parsing project, 436

say command, chat server project, 480, 485

scope, 131–133

see also namespaces

class scope variable, 153

description, 140

global scope, 131

local scope, 131

parameters, 118

nested scopes, 133

instant markup project, 410

number of scopes, 131

rebinding global variables, 132

using exec and eval, 109, 111

Scrape 'N' Feed, 328

scrapping *see* screen scraping

screen scraping, 321–328, 346

Beautiful Soup module, 327–328

HTMLParser callback methods, 325

Tidy, 322–324

using HTMLParser module, 325–327

web services, 344–346

XHTML, 325

scripts

adding line numbers to, 227

behaving like normal programs, 20–22

running from command prompt, 20

saving and executing programs, 19–20

scroll bars, text controls, 284

sdist command, Distutils, 386, 387

formats switch, 387

search function, re module, 245

second implementations of projects

structure of projects in this book, 402

second system syndrome

case against rewriting, 395

security

CGI security risks, 330

password digests, 494

PythonDebug directive, 340

using exec and eval, 109

seek method, files, 266

select function, select module

asynchronous I/O, 312, 314–315

avoiding forking and threading, 312

description, 320

select module, poll function

asynchronous I/O, 315–316

polling event constants in select module, 315

self parameter

calling unbound superclass constructor, 180

class methods, 189

class statement, 149

distinguishing methods from functions, 150

framework for extensions, 377, 378

static methods, 189

send method, generators, 198

send method, socket class, 307

sender column, messages table

bulletin board project, 502

sep variable, os module, 223, 224

separators

altsep variable, 224

linesep variable, 224

pathsep variable, 223, 224

sep variable, 223, 224

- sequence unpacking
 - assignment statements, 85–87
 - file iterators, 274
 - popitem method, 86
- sequences, 31–40, 51, 182, 206
 - accessing individual elements, 32
 - accessing ranges/slices of elements, 34
 - adding, 37
 - arithmetic sequence, 184
 - built-in sequence types, 31
 - checking membership, 38–39
 - close function, 226
 - comparing, 95
 - concatenating, 37
 - creating infinite sequence, 183
 - deleting element associated with key, 182
 - empty lists, 37
 - finding number of elements in, 40
 - finding smallest/largest elements in, 40
 - immutable sequences, 49
 - indexing, 31, 32–34
 - illegal type of index used, 184
 - initialization, 37
 - iterating over, 32
 - key is negative integer, 183
 - lists, 40–49
 - making from iterators, 194
 - mapping protocol and, 182–185
 - multiplying, 37–38
 - operations, 32–40
 - returning number of elements contained in, 182
 - returning value of key, 182
 - slicing, 34–37, 119
 - simulating, 185
 - specifying step length between elements, 36
 - storing value for key, 182
 - tuples, 49–51
- SERIAL keyword, PostgreSQL
 - CREATE TABLE command, 501
- serve_forever method
 - SimpleXMLRPCServer class, 519, 523
- server sockets, 319
- ServerProxy class
 - XML-RPC file sharing project, 520
- servers
 - connecting to, 454
 - forking server, 313
 - SocketServer module, 310–311
 - SocketServer-based servers, 317
 - threading server, 313
 - writing Twisted server, 317–319
- service provider, web services, 344
- service requester, web services, 344
- set attr method, 191, 192
- Set class instances, 228
- set comprehension, Python 3.0, 603
- set function, 584
- set methods, 187, 188
- set type
 - add method, 229
 - frozenset type and, 229
 - immutability, 229
 - reduce method, 229
 - remove method, 229
 - sets module and, 228
 - union method, 228, 229
- set_caption function, arcade game project, 549
- set_mode function, arcade game project, 549, 552
- set_reuse_addr method, chat server project, 473
- set_terminator method, chat server project, 473, 475
- set_visible function, arcade game project, 552
- __setattr__ method, 191, 192
- setattr function, 157, 159, 584
- setdefault method, dictionaries, 79, 106, 586
- setdefaultencoding function, sys module, 451
- Setext, 424
- setinputsizes method, cursors, 297
- __setitem__ method, 182
- setName method, private attributes, 151
- setoutputsize method, cursors, 297
- sets, 228–229
 - empty set, 604
 - new syntax in Python 3.0, 604
- sets module, 228–229, 259
- SetSizer method, wx.Panel class, 284
- setup function, Distutils, 384, 391
- setup script, Distutils, 383, 384
- setup.py script, Distutils, 384, 385, 387
 - commands to run setup.py, 390
- setuptools project, 384
- SetValue method
 - load event, wxPython, 286

- sgmllib module, 322
- sha module, 343
 - remote editing with CGI project, 494
- shadowing
 - locals function, 132
- shallow copy, dictionaries, 75
- shebang, 21
- shelve module, 238–241, 260
 - modifying objects, 239
 - open function, 238
- shift operator precedence, 580
- short-circuit logic, Boolean operators, 96, 574
- Show method, wx.Frame class, 281
- shuffle function, random module, 235
- signs (+/-), string formatting, 58
- Simple API for XML *see* SAX
- simple generators *see* generators
- Simple Wrapper and Interface Generator *see* SWIG
- simple_main.cgi script
 - bulletin board project, 505
- SimpleWebSource class
 - news gathering project, 462
- SimpleXMLRPCServer class, 519
 - allow_reuse_address attribute, 527
 - register_function method, 519
 - register_instance method, 519, 523
 - registering Node with, 521
 - serve_forever method, 519, 523
- SimpleXMLRPCServer module, 310, 518
- single-quoted strings, 23–24
- SIP tool, 371
- site-packages directory
 - executing path configuration files, 216
 - putting modules in existing sys.path, 215
- size argument, setting button positions using, 283
- sizers, 284–285
 - BoxSizer class, 284
 - layout mechanisms, 291
 - using relative coordinates, 284
- Slashdot, 499
- sleep function, time module, 233, 234
- slice function, sequences, 185
- slicing
 - lists, 42, 571
 - precedence, 580
 - sequences, 34–37
 - simulating, 185
- Smalltalk, 151
- SmartASCII, 424
- smtpd/smtplib modules, 310
- SOAP/SOAPy, 346
- socket class, socket module, 306
 - accept method, 306
 - bind method, 306
 - connect method, 306
 - gethostname function, 306, 307
 - listen method, 306
 - recv method, 307
 - send method, 307
- socket module, 306–308, 319
 - socket class, 306
 - tools for chat server project, 470
- socket server
 - connecting to, 470
- sockets
 - chat server project
 - bind method, 471
 - create_socket method, 471, 472
 - datagram socket, 306
 - description, 319
 - network programming, 470
 - stream socket, 306
 - types of, 306
- SocketServer framework, 319
- SocketServer module, 310–311
 - BaseRequestHandler class, 311
 - classes, 311
 - forking and threading with, 313
 - SocketServer-based servers, 317
 - StreamRequestHandler class, 311
- sort command, files, 265
- sort method, lists, 47–49, 585
 - cmp built-in function, 48
 - key argument, 48
 - keyword arguments, 48
 - reverse argument, 49
- sorted function, 48, 52, 102, 112, 584
 - keyword arguments, 49
- sorted iteration, 102
- Sorting Mini-HOWTO, 49
 - Andrew Dalke, 49

- source code
 - encoding in Python 3.0, 605
 - exploring modules, 221
- source code checking, 359
 - PyChecker/PyLint tools, 359–362, 364
- SourceForge, 596
- span method
 - MatchObjects, re module, 248
- special attributes *see* magic attributes
- special characters
 - character sets, regular expressions, 243
 - escaping, regular expressions, 242
- special methods *see* magic methods
- special values, Python DB API, 297, 304
- speed
 - extending Python to improve, 365–366
- splicing operators, 129
 - argument splicing, Python 3.0, 604
- split function, re module, 245, 246
- split method, strings, 63, 588
 - food database application, 300
- splitlines method, strings, 588
- Sprite class, pygame.sprite module
 - arcade game project, 550
 - image attribute, 554
 - rect attribute, 554
- sprite module, pygame, 550
- Spyce, 341, 344
- SQL
 - tutorial/reading on, 293
- SQLite, 298, 304
 - bulletin board project, 500
 - creating database in, 501
 - conversions between numbers and strings, 303
- sqrt function, 18, 30
- stack trace
 - catching exceptions, 167
 - exceptions and functions, 170
- Stackless Python, 312
 - alternative Python distributions, 6, 7
- stacks, 45
- standard library modules, 221–259
 - see* individual modules
 - opening/closing standard library files, 221
 - Python 3.0, 604
- StandardError exception, Python DB API, 295
- starred iterable unpacking, 603, 606
- start method
 - MatchObjects class, 248
 - Handler class, 410, 411
 - Node class, 523, 524, 528, 531
- startElement event handler
 - XML parsing project, 440, 441, 445
- startfile function, os module, 225
- startPage method
 - XML parsing project, 448
- startswith method, strings, 588
- Startup class, arcade game project, 564
- startUp method, test fixture, 356
- state, encapsulation, 147
- State class, arcade game project, 559
- state variables
 - screen scraping using HTMLParser, 326
- statements, 13–14, 589–594
 - assert statements, 97, 589
 - assertions, 111
 - assignment statements, 85–88, 111, 589
 - blocks, 88, 111
 - break statement, 102, 591
 - class statement, 149, 594
 - compared to expressions, 13
 - conditional statements, 88–97, 111
 - assertions, 97
 - Boolean operators, 95–96
 - comparison operators, 92–95
 - elif clause, 91
 - else clause, 90
 - if statement, 90
 - nesting blocks, 91
 - continue statement, 103, 591
 - def statement, 115, 116
 - del statement, 41, 107–108, 112, 590
 - deleting objects, 107
 - description, 29
 - doing nothing, 107
 - eval statement, 110, 112
 - exec statement, 109–110, 112, 592
 - expression statements, 589
 - for statement, 593
 - function definition statement, 115, 594
 - global statements, 592
 - if statement, 15, 592
 - import statements, 84–85, 111, 591

- statements (*continued*)
 - loops, 97–105, 112
 - breaking out of, 102–105
 - for loop, 99
 - iteration, 100–102
 - using else clause in, 105
 - while loop, 98
 - pass statement, 107, 112, 590
 - print statement, 111, 590
 - separating with commas, 83–84
 - raise statement, 162–163, 591
 - return statement, 116, 590
 - try statements, 593
 - while statement, 592
 - while True/break idiom, 104–105
 - with statement, 267, 593
 - yield statement, 590
- static methods, 189–191
 - self parameter, 189
- staticmethod function, 584
- stderr stream, sys module, 222, 263
- stdin stream, sys module, 222, 263
 - file iterators, 273
 - script counting words in, 265
- stdout class
 - write method, 318
- stdout stream, sys module, 222, 263
- StopIteration exception, 192
- store function, 122
- str function, 25, 30, 584
- str type, Python 3.0, 600, 605
- stream redirection functionality
 - changes in Python 3.0, 600
- stream socket, 306
- StreamRequestHandler class, 311
- streams, chat server project, 477
- streams, files, 263, 274
- strftime function, time module, 233, 455, 456
- String constructor
 - drawing with ReportLab, 428
- string formatting
 - % character, 53, 54, 56
 - changes in Python 3.0, 600, 605
 - dictionaries, 73, 81
- string methods, 60–66, 586–588
 - capitalize, 586
 - center, 586
 - count, 586
 - decode, 586
 - encode, 586
 - endswith, 586
 - expandtabs, 586
 - find, 60, 586
 - index, 587
 - isalnum/isalpha/isdigit, 587
 - islower/isspace, 587
 - join, 61, 223, 255, 587
 - ljust, 587
 - lower, 62, 95, 241, 457, 587
 - lstrip, 587
 - partition, 587
 - replace, 63, 587
 - rfind, 587
 - rindex, 587
 - rjust, 588
 - rpartition, 588
 - rsplit, 588
 - rstrip, 227, 588
 - safe_substitute, 55
 - split, 63, 300, 588
 - splitlines, 588
 - startswith, 588
 - strip, 64, 241, 300, 588
 - substitute, 55
 - swapcase, 588
 - title, 63, 588
 - translate, 60, 64–66, 588
 - upper, 95, 588
 - zfill, 588
- string module, 55
 - capwords function, 63, 66
 - constants, 60
 - letters constant, 60, 606
 - maketrans function, 65, 66
- String type, 579
- STRING value, Python DB API, 298
- StringIO, Python 3.0, 605
- strings, 22–29
 - changing to lowercase, 62
 - comparing, 94
 - concatenating, 24
 - converting values to, 24
 - escaping quotes, 23–24
 - evaluating expression strings, 254

- executing/evaluating on the fly, 108
- finding substrings, 60
- formatting, 53–59
 - conversion specifiers, 54–59
 - conversion types, 56, 57
 - precision specifiers, 54
 - Python 3.0, 600
 - signs/alignment/zero-padding, 58
 - string formatting operator, 53
 - width and precision, 57
- immutability, 53, 119
- input compared to raw_input, 26
- long strings, 26–27
- modulo operator, 66
- non-english strings, 66
- numbers containing leading zeros, 70
- operations, 53
- precedence, 581
- Python 3.0, 599
- raw strings, 27–28
- removing whitespace, 64
- repr function, 25
- representing, 24–25
- single-quoted strings, 23–24
- subclassing str type, 185–187
- template strings, 55
- Unicode strings, 28–29
- using group numbers in substitution string, 249
- strip method, strings, 64, 241, 588
 - food database application, 300
- strptime function, time module, 233, 234
- style parameter
 - wx.BoxSizer constructor, 285
 - wx.TextCtrl constructor, 283
- sub function, re module, 245, 247
 - instant markup project, 407, 408, 409, 411
 - sample template system, 254
 - using group numbers in substitution string, 249, 250
- sub method, Handler class
 - instant markup project, 410, 411
- subclasses, 147, 148
 - inheritance, 154–155
 - issubclass method, 154
 - overriding methods, 148, 156, 177
- subclassing
 - list/dict/str types, 185–187
- subject column, messages table
 - bulletin board project, 502
- subpatterns
 - finding sender of e-mail, 252
 - groups, re module, 247
- subpatterns, regular expressions, 243–244
- subprocess module, 224, 371
 - running Tidy, 324
 - using command-line tools, 360
- substitute method, 55
- substitutions
 - using group numbers in substitution string, 249
- sum function, 140, 584
- sunspots example
 - fetching data from Internet, 432
 - final sunspot program (sunspots.py), 433
 - first implementation, 431
 - first prototype, 430
 - implementations, 427–434
 - introduction, 425
 - preparations, 426
 - second implementation, 434
 - using LinePlot class, 432, 434
- super function, 180–181, 207, 584
 - changes in Python 3.0, 606
 - new-style/old-style classes, 175, 176, 206
 - subclassing list type, 186
 - using when multiple superclasses, 181
- superclasses
 - calling unbound superclass constructor, 179–180
 - description, 147
 - multiple inheritance, 156
 - multiple superclasses, 155–156
 - overriding methods and constructors, 177
 - overriding methods in subclasses, 156
 - specifying, 153–154
- Surface function, arcade game project, 548
- surface objects, 548
 - convert method, 554
- swapcase method, strings, 588
- SWIG (Simple Wrapper and Interface Generator), 371–375, 380
 - automating compilation, 375
 - c++ option, 373
 - compiling, 373
 - Distutils using, 389
 - header file, 373

SWIG (Simple Wrapper and Interface Generator)
(*continued*)

- installing, 371
- interface file, 373
- linking, 374
- program to recognize palindromes, 372–375
- python option, 373
- running, 373
- using Distutils, 375
- using SWIG, 372
- wrapping code, 375
- Swing GUI toolkit, 278, 597
 - example illustrating, 288
 - Jython and, 290
- switches
 - command-line switches, 398
- symbolic constants, 396
- synchronous network programming, 306
- SyntaxError exception, 163
 - sample template system, 254
- sys module, 222–223, 259
 - functions and variables, 222
 - getdefaultencoding function, 451
 - path variable
 - modifying to specify module location, 210, 214
 - putting modules in existing sys.path, 214–215
 - search path (list of directories), 214
 - using PYTHONPATH alongside, 216
 - setdefaultencoding function, 451
- sys.maxint, Python 3.0, 605
- system function, os module, 223, 224
- SystemRandom class, 234

T

- tab characters, indenting with, 88
- tables
 - CREATE TABLE command, 501
- tags
 - HTMLParser callback methods, 325
- tar command
 - compiling Python from sources, 4
- tar files
 - sdist command, Distutils, 387
- TCPServer class, SocketServer module, 311
- tearDown method, test fixture, 356
- tell method, files, 266
- telnet command, chat server project, 470
- telnetlib module, 310
- Template class, string module, 55, 74
- template strings, 55
- templates, 253–257
- terminator, chat server project, 473
- ternary operator, 96
- test code, modules, 259
- test coverage, 351, 352
- test fixture, 356
- TestCase class, unittest module, 355
 - instantiating all subclasses of, 356
 - methods, 356–357
- test-driven programming, 349–352, 364
 - anticipating code changes, 351
 - automated tests, 351
 - key steps in process, 352
 - making code fail test, 352
 - requirement specification, 350–351
 - simple test program, 350
 - unittest module, 353
- testing
 - alternatives to unit test tools, 355
 - anticipating code changes, 351
 - automated testing, 394
 - beyond unit testing, 358–363
 - bulletin board project, 513
 - code coverage, 351
 - doctest module, 352, 353–355, 364
 - minimum requirements, 400
 - modules, 212–214
 - profiling, 359, 362–363, 364
 - PyChecker/PyLint tools, 359–362, 364
 - requirement specification, 350–351
 - source code checking, 359
 - test-driven programming, 349–352, 364
 - tools for testing, 352–358
 - unit testing, 349
 - unittest module, 352, 355–358, 364
- testmod function, doctest module, 353, 354, 364
- TeX typesetting program, 404, 426
- text
 - finding blocks of, 406–407
- text column, messages table
 - bulletin board project, 502
- text controls
 - creating, 283
 - creating text area, 283

- horizontal scroll bar, 284
- multiline text area, 284
- text editor
 - selecting for programming, 19
 - wxPython GUI toolkit building, 279–288
 - creating application object, 280
 - creating frames (windows), 281
 - creating widgets (components), 281
 - event handling, 286
 - finding file name, 286
 - importing wx module, 280
 - improving layout, 284–285
 - interface elements, 280
 - minimal requirements for text editor, 279
 - positions and sizes, 283
 - putting text into text area, 286
 - titles and labels, 282
- text files
 - changes in Python 3.0, 600
 - changes on opening in text mode, 262
- text parameter, CGI, 490
- textAnchor argument, String constructor, 428
- TextCtrl class, wx module *see* wx.TextCtrl class
- Textile, 424
- threading, 312
 - chat server project, options for, 469
 - microthreads, 312
 - multiple connections, 312
 - SocketServer module, 313
 - XML-RPC file sharing project, 528, 534
- threading module
 - XML-RPC file sharing project, 519
- threading server, 313
- threadsafety property, Python DB API, 294
- throw method, generators, 199
- Tidy, 322–324
 - getting Tidy library, 324
 - μTidyLib, 324
 - mxTidy, 324
 - using command-line Tidy, 324
 - using HTMLParser, 325
- Tidylib, 324
- Time constructor, Python DB API, 298
- time function, time module, 233, 234, 455
- time module, 232–234, 259, 454
 - functions, 233, 455
- TimeFromTicks constructor, Python DB API, 298
- timeit module, 234, 258, 363
- Timestamp constructor, Python DB API, 298
- TimestampFromTicks constructor, Python DB API, 298
- TinyFugue, chat server project, 471
- title argument, wx.Frame constructor, 282
- title method, strings, 63, 588
- title rules, instant markup project, 415
- TitleRule class, instant markup project, 416, 420
- Tk GUI toolkit, 289
- Tk platform, Tkinter GUI toolkit, 277
- Tkinter GUI toolkit, 277, 289, 597
 - choosing between GUI toolkits, 278
 - example illustrating, 288
- toolkits *see* GUI toolkits
- tools for projects
 - Pygame tool, 548–551
 - structure of projects in this book, 401
- trace module, 258
- trace.py program, 351
- tracebacks, 161
 - cgitb module, 502
- transactions, 296
- translate method, strings, 60, 64–66, 588
- translation tables, 65
- trapping exceptions *see* catching exceptions
- trees, 201
- True value
 - Boolean values, 89
 - changes in Python 3.0, 605
 - while True/break idiom, 104–105
- truth, Boolean values, 89
- try statements, 593
- try/except statements, 163–169, 576
 - catching all exceptions, 169
 - danger of, 167
 - catching exception object, 166
 - catching many exceptions in one block, 166
 - checking whether object has specific attribute, 172
 - combining try/except/finally/else, 170
 - else clause, 168–169, 173
 - finally clause, 170
 - if/else compared, 171, 173
 - trapping KeyError exception, 172
 - using more than one except clause, 165–166

- try/finally statement, 169, 173
 - calling exit function in, 222
 - closing database, 241
 - closing files, 267
 - tuple function, 50, 52, 584
 - tuple parameter unpacking, Python 3.0, 606
 - tuples, 49–51
 - conversion specifiers, 56
 - distributing operator, 128, 604
 - empty tuple, 49
 - fields of Python date tuples, 233
 - finding out if object is tuple, 142
 - gathering operator, 126, 604
 - immutability, 119
 - lists compared, 31
 - tuple operations, 50
 - uses of, 51
 - writing tuple with single value, 50
 - TurboGears, 343, 344
 - tutorial, Python, 569–577, 596
 - Twisted framework, 316–319, 320
 - chat server project, options for, 469
 - deferred execution, 317
 - downloading and installing, 317
 - remote procedure calls with, 346
 - SOAP toolkit, 346
 - web application frameworks, 344
 - writing Twisted server, 317–319
 - twisted.internet.protocol module
 - Factory class, 317
 - twisted.protocols.basic module
 - LineReceiver protocol, 318
 - txt2html, 424
 - type function, 159, 584
 - type objects, 17
 - TypeError class, 163
 - inappropriate key type used, 183, 184
 - recursive generators, 196, 197
 - types, 569
 - see also* classes
 - bool type, 90
 - classes and, 147, 148
 - conversion specifiers, 57
 - deque type, 231–232
 - dictionary type, 69
 - duck typing, 145
 - polymorphism, 145
 - Python DB API, 297–298, 304
 - string formatting, 56, 57
- ## U
- %u conversion specifier, 56
 - UDPServer class, SocketServer module, 311
 - unary operators, 580
 - unbound methods
 - calling unbound superclass constructor, 180
 - underscores
 - magic methods, 575
 - making method or attribute private, 151, 573
 - UnhandledQuery class
 - XML-RPC file sharing project, 528, 530
 - unichr function, 584
 - unicode function, 584
 - Unicode strings, 28–29
 - Unicode type, 579
 - changes in Python 3.0, 600, 605
 - uniform function, random module, 235
 - uninstall command, Distutils, 385, 388
 - union method, set type, 228, 229
 - unit testing, 349
 - alternatives to unit test tools, 355
 - unittest module, 352, 355–358, 364
 - distinguishing errors and failures, 357
 - main function, 356, 364
 - TestCase class, 355, 356
 - test-first, code-later programming, 353
 - Universal Feed Parser, 345
 - universal newline support mode, files, 263
 - UNIX
 - installing mod_python on, 337
 - installing Python on, 3–5
 - levels of configuration, 398
 - making executable script in, 576
 - setting environment variables, 216
 - UnixDatagramServer class, 311
 - UnixStreamServer class, 311
 - unknown method, chat server project, 479
 - Unofficial Planet Python blog, 597
 - unpacking
 - iterable unpacking, Python 3.0, 603
 - sequence unpacking, 85–87
 - starred iterable unpacking, 603, 606
 - tuple parameter unpacking, 606
 - unquote function, urllib module, 309, 320
 - unquote_plus function, urllib module, 309, 320

- unregister method, poll object, 315
- update method, dictionaries, 80, 586
- upper method, strings, 95, 588
- uppercase constant, string module, 60
- urandom function, os module, 224
- urlcleanup function, urllib module, 309
- urlencode function, urllib module, 309, 320, 334
- urlfile.txt file, GUI client project, 540
- urllib module, 308–309, 319
 - news gathering project, 453
 - quote function, 309
 - quote_plus function, 309
 - screen scraping, 321
 - invoking CGI scripts without forms, 334
 - unquote function, 309
 - unquote_plus function, 309
 - urlcleanup function, 309
 - urlencode function, 309
 - urlopen function, 308, 309, 432
 - urlretrieve function, 309
- urllib2 module, 308–309, 319
- urlopen function, urllib module, 308, 309, 320
 - graphics creation project, 432
- urlparse module, 310
 - XML-RPC file sharing project, 519
- urlretrieve function, urllib module, 309, 320
- URLs filter
 - instant markup project, 418
- urls.txt file
 - XML-RPC file sharing project, 534
- Usenet, 453
- Usenet groups, 597
- user parameter
 - connect function, Python DB API, 296
- UserList/UserDict/UserString
 - subclassing list/dict/str types, 185
- users
 - getting input from users, 14–15
- users dictionary, 479
- UTF-8, Python 3.0, 600, 605
- util.py block generator
 - instant markup project, 406

V

- ValueError class, 163
- values
 - literal values, 579

- None, 37
 - seeing all values stored in objects, 157
 - special values, Python DB API, 297, 304
- values method, dictionaries, 80, 100, 586
 - changes in Python 3.0, 605
- van Rossum, Guido, 278
- variables, 13, 29, 131, 569
 - all variable, 219
 - altsep variable, 224
 - argv variable, 222, 223
 - environ mapping, 223, 224
 - environment variables, 216
 - global variables, Python DB API, 294–295
 - linesep variable, 224
 - modules mapping, 222
 - naming conventions, 13
 - nonlocal variables, Python 3.0, 602
 - path variable, 222
 - pathsep variable, 223, 224
 - platform variable, 222
 - rebinding variables in outer scopes, 133
 - scopes, 140
 - scoping, 131–133
 - sep variable, 223, 224
 - stderr stream, 222
 - stdin stream, 222
 - stdout stream, 222
- vars function, 131, 584
- VERBOSE flag, re module functions, 249
- version control
 - remote editing with CGI project, 497
- versions, Python DB API, 294
- view.cgi script
 - bulletin board project, 506, 508–510
 - link from main.cgi, 507, 508
 - link to edit.cgi, 508
 - testing, 513
 - remote editing with CGI project, 497
- views
 - dictionary views, Python 3.0, 603
- virtual tea party *see* chat server project
- VisualWx environment, 6

W

- Warning exception, Python DB API, 295
- warnings, 173
- Weave, 370

- web application frameworks, 343, 347
- web development
 - mod_python, 336–343
- web forms *see* forms
- web pages
 - dynamic web pages with CGI, 328–336
 - adding pound bang (!) line, 329
 - CGI script, 331
 - CGI security risks, 330
 - debugging with cgith, 331–332
 - HTML form, 334–336
 - invoking CGI scripts without forms, 334
 - preparing web server, 328–329
 - setting file permissions, 329–330
 - using cgi module, 333
 - screen scraping, 321–328
 - Beautiful Soup module, 327–328
 - Tidy, 322
 - using HTMLParser, 325–327
 - using web services, 344–346
 - XHTML, 325
- web programming
 - dynamic web pages with CGI, 328–336
 - mod_python, 336–343
 - screen scraping, 321–328
 - Beautiful Soup module, 327–328
 - Tidy, 322–324
 - using web services, 344–346
- web server
 - dynamic web pages with CGI, 328–329
- Web Service Description Language (WSDL), 345
- web services, 344–346, 347
 - remote procedure calls with XML-RPC, 345
 - RSS, 345
 - service provider, 344
 - service requester, 344
 - SOAP, 346
- web sites
 - generating from single XML file, 435
 - XML parsing project, 437
- web tutorial, Python, 569
- web.py, 344
- web-based bulletin board *see* bulletin board project
- webbrowser module, 225
- website element, XML parsing project, 437
- website.py file, XML parsing project, 448
- website.xml file, XML parsing project, 438
- WebsiteConstructor class, 449, 451
- Webware, 341, 344
- Weight class, arcade game project, 558
- weight.png file, arcade game project, 554, 556
- weight.pny file, arcade game project, 554
- whence parameter
 - seek method, files, 266
- while loops, 98, 569
 - ignoring return value of raw_input function, 238
 - iterating over file contents with read(), 270
 - iterating over file contents with readline(), 271
- while statements, 592
- while True/break idiom, 104–105
 - iterating over file contents with read(), 271
- whitespace
 - handling for DOS, 225
 - VERBOSE flag, re module functions, 249
- who command, chat server project, 480, 485
- widgets
 - wxPython GUI toolkit creating, 281
- widgets, text editor
 - Bind method, 286
- width of field, string formatting, 56, 57, 59
- WikiCreole, 424
- WikiMarkupStandard, 424
- wikis
 - remote editing with CGI project, 489
- wiki-style markup systems, 424
- Wikitext, 424
- wildcards, regular expressions, 242
- Windows
 - installing mod_python on, 337
 - installing Python on, 1–3
 - setting environment variables, 216
- windows *see* frames
- Windows Installer file, 3
- Wingware environment, 6
- wininst format
 - bdist command, Distutils, 387, 388
- with statement, 267
 - changes in Python 3.0, 605
 - closing files, 267, 274
 - context managers, 268

- with statements, 593
- wrapper code
 - SWIG, 372, 375, 380
 - wrapping legacy code, 366
- wrapping modules as archive file, 386, 387
- write method
 - save event, wxPython, 286
- write method, files, 264, 269
- write method, stdout class
 - writing Twisted server, 318
- write mode, open function (files), 262
- writeback parameter, shelve.open function, 239
- writeFooter method
 - XML parsing project, 446, 448, 451
- writeHeader method
 - XML parsing project, 446, 448
- writelines method, files, 267, 269
 - XML parsing project, 451
- writing files, 264, 274
 - closing files after writing, 267
 - updating files after writing, 268
- writing lines, files, 266, 274
- WSDL (Web Service Description Language), 345
- wx module
 - importing, 280
 - method naming conventions, 281
 - style facets, 284
 - using keyword arguments with wx constructors, 282
- wx.ALL flag, 285
- wx.App class
 - creating application object, 280
 - GUI client project, 538
 - MainLoop method, 281
- wx.BoxSizer class
 - Add method, 285
 - building text editor, 284
 - horizontal or vertical style, 285
 - style argument, 285
 - using relative coordinates, 284
- wx.Button class
 - adding button to frame, 281
 - label argument, 282
 - parent argument, 281
 - pos (position) argument, 283
 - size argument, 283
- wx.EVT_BUTTON symbolic constant, 286
- wx.EXPAND flag, 285
- wx.Frame class
 - building text editor, 281
 - parent argument, 281
 - Show method, 281
 - size argument, 283
 - title argument, 282
 - windows as instances of, 281
- wx.HORIZONTAL/wx.VERTICAL values, 285
- wx.HSCROLL value, 284
- wx.LEFT/wx.RIGHT flags, 285
- wx.Panel class
 - building text editor, 284
 - SetSizer method, 284
- wx.PySimpleApp class
 - creating application object, 281
- wx.TE_MULTILINE value, 284
- wx.TextCtrl class
 - building text editor, 283
 - style parameter, 283
- wx.TOP/wx.BOTTOM flags, 285
- wxDesigner environment, 6
- wxGlade environment, 6
- wxPython GUI toolkit, 277, 291, 597
 - building text editor, 279–288
 - creating application object, 280
 - creating frames (windows), 281
 - creating widgets (components), 281
 - event handling, 286
 - importing wx module, 280
 - improving layout, 284–285
 - interface elements, 280
 - minimal requirements for, 279
 - positions and sizes, 283
 - titles and labels, 282
 - using relative coordinates, 284
 - choosing between GUI toolkits, 278
 - demo distribution, 279
 - downloading, 278
 - example illustrating, 289
 - GUI client project, 537
 - installing, 279
- wxWindows platform
 - wxPython GUI toolkit, 277

X

%X, %x conversion specifiers, 57

XHTML

advantages over HTML, 325

XML, 435

uses of, 436

XML parsing project, 435–452

creating content handler, 439

creating HTML pages, 442–444

creating simple content handler, 441

dispatcher mix-in classes, 444, 446

events/event handlers, 439–441, 448–450

factoring out header/footer/default handling, 446

further exploration, 451

goals, 436

handling special characters, 450

implementations, 438–451

installing, PyXML, 437

parsing XML file, 439

preparations, 437–438

SAX parser, 436

Simple API for XML (SAX), 435, 438

support for directories, 447

tools, 436–437

xml.sax module

parse function, 439

xml.sax.handler module

ContentHandler class, 439

XMLDestination class

news gathering project, 468

XML-RPC

remote procedure calls with, 345

XML-RPC file sharing project, 517–535

adding GUI client, 537–545

avoiding loops, 518

connecting to nodes, 518

creating client interface, 527–528

exceptions, 528–529

further exploration, 534

implementations, 519–534

Node class, 520–525

node communication, 518

preparations, 519

requirements, 518

tools, 518

validating file names, 529–534

XML-RPC server

SimpleXMLRPCServer module, 310

xmlrpclib module, 310, 518

XML-RPC file sharing project, 520, 527

Fault class, 528, 529

XPath, 325

xrange function, 100, 101, 112, 584

changes in Python 3.0, 606

xreadlines method, files

lazy line iteration with, 272

Y

YAML

markup systems and web sites, 424

yield expression, generators, 198, 199

yield statement, generators, 195, 198

generator-function, 198, 207

yield statements, 590

Z

Zawinski, Jamie, 242

ZeroDivisionError class, 161, 163, 576

catching with more than one except clause, 165

muffling, 164, 165

zero-padding, string formatting, 58

zeros

numbers containing leading zeros, 70

zfill method, strings, 588

zip files

sdist command, Distutils, 387

zip function, 101, 112, 584

changes in Python 3.0, 605

constructing PolyLine objects, 430

“zlib not available” warning, 429

Zope, 341, 343, 344

ZSI, SOAP toolkit, 346