What is Python?

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on PCs under MS-DOS, Windows, Windows NT, and OS/2.

Why can't I use an assignment in an expression?

Many people used to C or Perl complain that they want to use this C idiom:

```
while (line = readline(f)) {
...do something with line...
}
```

where in Python you're forced to write this:

```
while True:
line = f.readline()
if not line:
break
...do something with line...
```

The reason for not allowing assignment in Python expressions is a common, hard-to-find bug in those other languages, caused by this construct:

```
if (x = 0) {
```

```
...error handling...
}
else {
...code that only works for nonzero x...
}
```

The error is a simple typo: x = 0, which assigns 0 to the variable x, was written while the comparison x == 0 is certainly what was intended.
Many alternatives have been proposed. Most are hacks that save some typing but use arbitrary or cryptic syntax or keywords, and fail the simple criterion for language change proposals: it should intuitively suggest the proper meaning to a human reader who has not yet been introduced to the construct.
An interesting phenomenon is that most experienced Python programmers recognize the "while True" idiom and don't seem to be missing the assignment in expression construct much; it's only newcomers who express a strong desire to add this to the language.
There's an alternative way of spelling this that seems attractive but is generally less robust than the "while True" solution:

```
line = f.readline()
while line:
...do something with line...
line = f.readline()
```

The problem with this is that if you change your mind about exactly how you get the next line (e.g. you want to change it into sys.stdin.readline()) you have to remember to change two places in your program -- the second occurrence is hidden at the bottom of the loop.
The best approach is to use iterators, making it possible to loop through objects using the for statement. For example, in the current version of Python file objects support the iterator protocol, so you can now write simply:

```
for line in f:
... do something with line...
```

Is there a tool to help find bugs or perform static analysis?
Yes.
PyChecker is a static analysis tool that finds bugs in Python source code and warns about code complexity and style.

Pylint is another tool that checks if a module satisfies a coding standard, and also makes it possible to write plug-ins to add a custom feature.

How do you set a global variable in a function?
Did you do something like this?

```
x = 1 # make a global
def f():
print x # try to print the global
...
for j in range(100):
if q>3:
x=4
```

Any variable assigned in a function is local to that function. unless it is specifically declared global. Since a value is bound to x as the last statement of the function body, the compiler assumes that x is local. Consequently the print x attempts to print an uninitialized local variable and will trigger a NameError.
The solution is to insert an explicit global declaration at the start of the function:

```
def f():
global x
print x # try to print the global
...
for j in range(100):
if q>3:
x=4
```

In this case, all references to x are interpreted as references to the x from the module namespace.

What are the rules for local and global variables in Python?
In Python, variables that are only referenced inside a function are implicitly global. If a variable is assigned a new value anywhere within the function's body, it's assumed to be a local. If a variable is ever assigned a new value inside the function, the variable is implicitly local, and you need to explicitly declare it as 'global'.
Though a bit surprising at first, a moment's consideration explains this. On one hand, requiring global for assigned variables provides a bar against unintended side-effects. On the other hand, if global was required for all global references, you'd be using global all the time. You'd have to declare as global every reference to a builtin function or to a component of an imported module. This clutter would defeat the usefulness of the global declaration for identifying side-effects.

How do I share global variables across modules?
The canonical way to share information across modules within a single program is to create a special module (often called config or cfg). Just import the config module in all modules of your application; the module then becomes available as a global name. Because there is only one instance of each module, any changes made to the module object get reflected everywhere. For example:

config.py:

```
x = 0 # Default value of the 'x' configuration setting
```

mod.py:

```
import config
config.x = 1
```

main.py:

```
import config
import mod
print config.x
```

Note that using a module is also the basis for implementing the Singleton design pattern, for the same reason.

How can I pass optional or keyword parameters from one function to another?

Collect the arguments using the * and ** specifier in the function's parameter list; this gives you the positional arguments as a tuple and the keyword arguments as a dictionary. You can then pass these arguments when calling another function by using * and **:

```
def f(x, *tup, **kwargs):
...
kwargs['width']='14.3c'
...
g(x, *tup, **kwargs)
```

In the unlikely case that you care about Python versions older than 2.0, use 'apply':

```
def f(x, *tup, **kwargs):
...
kwargs['width']='14.3c'
...
apply(g, (x,)+tup, kwargs)
```

How do you make a higher order function in Python?

You have two choices: you can use nested scopes or you can use callable objects. For example, suppose you wanted to define linear(a,b) which returns a function f(x) that computes the value a*x+b. Using nested scopes:

```
def linear(a,b):
def result(x):
return a*x + b
return result
```

Or using a callable object:

```
class linear:
def __init__(self, a, b):
self.a, self.b = a,b
def __call__(self, x):
return self.a * x + self.b
```

In both cases:

```
taxes = linear(0.3,2)
```

gives a callable object where taxes(10e6) == 0.3 * 10e6 + 2.

The callable object approach has the disadvantage that it is a bit slower and results in slightly longer code. However, note that a collection of callables can share their signature via inheritance:

```
class exponential(linear):
# __init__ inherited
```

```
def __call__(self, x):
return self.a * (x ** self.b)
```

Object can encapsulate state for several methods:

```
class counter:
value = 0
def set(self, x): self.value = x
def up(self): self.value=self.value+1
def down(self): self.value=self.value-1

count = counter()
inc, dec, reset = count.up, count.down, count.set
```

Here inc(), dec() and reset() act like functions which share the same counting variable.

How do I copy an object in Python?
In general, try copy.copy() or copy.deepcopy() for the general case. Not all objects can be copied, but most can.

Some objects can be copied more easily. Dictionaries have a copy() method:

```
newdict = olddict.copy()
```

Sequences can be copied by slicing:

```
new_l = l[:]
```

How can I find the methods or attributes of an object?
For an instance x of a user-defined class, dir(x) returns an alphabetized list of the names containing the instance attributes and methods and attributes defined by its class.

How do I convert a string to a number?
For integers, use the built-in int() type constructor, e.g. int('144') == 144. Similarly, float() converts to floating-point, e.g. float('144') == 144.0.

By default, these interpret the number as decimal, so that int('0144') == 144 and int('0x144') raises ValueError. int(string, base) takes the base to convert from as a second optional argument, so int('0x144', 16) == 324. If the base is specified as 0, the number is interpreted using Python's rules: a leading '0' indicates octal, and '0x' indicates a hex number.

Do not use the built-in function eval() if all you need is to convert strings to numbers. eval() will be significantly slower and it presents a security risk: someone could pass you a Python expression that might have unwanted side effects. For example, someone could pass __import__('os').system("rm -rf $HOME") which would erase your home directory.

eval() also has the effect of interpreting numbers as Python expressions, so that e.g. eval('09') gives a

syntax error because Python regards numbers starting with '0' as octal (base 8).

How can my code discover the name of an object?
Generally speaking, it can't, because objects don't really have names. Essentially, assignment always binds a name to a value; The same is true of def and class statements, but in that case the value is a callable. Consider the following code:

```
class A:
pass

B = A

a = B()
b = a
print b
<__main__.A instance at 016D07CC>
print a
<__main__.A instance at 016D07CC>
```

Arguably the class has a name: even though it is bound to two names and invoked through the name B the created instance is still reported as an instance of class A. However, it is impossible to say whether the instance's name is a or b, since both names are bound to the same value.

Generally speaking it should not be necessary for your code to "know the names" of particular values. Unless you are deliberately writing introspective programs, this is usually an indication that a change of approach might be beneficial.

In comp.lang.python, Fredrik Lundh once gave an excellent analogy in answer to this question:

The same way as you get the name of that cat you found on your porch: the cat (object) itself cannot tell you its name, and it doesn't really care -- so the only way to find out what it's called is to ask all your neighbours (namespaces) if it's their cat (object)...

....and don't be surprised if you'll find that it's known by many names, or no name at all!

Is there an equivalent of C's "?:" ternary operator?
No.

How do I convert a number to a string?
To convert, e.g., the number 144 to the string '144', use the built-in function str(). If you want a hexadecimal or octal representation, use the built-in functions hex() or oct(). For fancy formatting, use the % operator on strings, e.g. "%04d" % 144 yields '0144' and "%.3f" % (1/3.0) yields '0.333'. See the library reference manual for details.

How do I modify a string in place?
You can't, because strings are immutable. If you need an object with this ability, try converting the string to a list or use the array module:

```
>>> s = "Hello, world"
>>> a = list(s)
>>>print a
['H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd']
>>> a[7:] = list("there!")
>>>".join(a)
'Hello, there!'


>>> import array
>>> a = array.array('c', s)
>>> print a
array('c', 'Hello, world')
>>> a[0] = 'y' ; print a
array('c', 'yello world')
>>> a.tostring()
'yello, world'
```

How do I use strings to call functions/methods?

There are various techniques.

* The best is to use a dictionary that maps strings to functions. The primary advantage of this technique is that the strings do not need to match the names of the functions. This is also the primary technique used to emulate a case construct:

```
def a():
pass

def b():
pass

dispatch = {'go': a, 'stop': b} # Note lack of parens for funcs

dispatch[get_input()]() # Note trailing parens to call function
```

*

Use the built-in function getattr():

```
import foo
getattr(foo, 'bar')()
```

Note that getattr() works on any object, including classes, class instances, modules, and so on.

This is used in several places in the standard library, like this:

```
class Foo:
def do_foo(self):
```

```
...
def do_bar(self):
...

f = getattr(foo_instance, 'do_' + opname)
f()
```

*
Use locals() or eval() to resolve the function name:

```
def myFunc():
print "hello"

fname = "myFunc"

f = locals()[fname]
f()

f = eval(fname)
f()
```

Note: Using eval() is slow and dangerous. If you don't have absolute control over the contents of the string, someone could pass a string that resulted in an arbitrary function being executed.

Is there an equivalent to Perl's chomp() for removing trailing newlines from strings?
Starting with Python 2.2, you can use S.rstrip("\r\n") to remove all occurences of any line terminator from the end of the string S without removing other trailing whitespace. If the string S represents more than one line, with several empty lines at the end, the line terminators for all the blank lines will be removed:

```
>>> lines = ("line 1 \r\n"
... "\r\n"
... "\r\n")
>>> lines.rstrip("\n\r")
"line 1 "
```

Since this is typically only desired when reading text one line at a time, using S.rstrip() this way works well.

For older versions of Python, There are two partial substitutes:

* If you want to remove all trailing whitespace, use the rstrip() method of string objects. This removes all trailing whitespace, not just a single newline.
* Otherwise, if there is only one line in the string S, use S.splitlines()[0].

Is there a scanf() or sscanf() equivalent?
Not as such.

For simple input parsing, the easiest approach is usually to split the line into whitespace-delimited words using the split() method of string objects and then convert decimal strings to numeric values using int() or float(). split() supports an optional "sep" parameter which is useful if the line uses something other than whitespace as a separator.

For more complicated input parsing, regular expressions more powerful than C's sscanf() and better suited for the task.

Is there a scanf() or sscanf() equivalent?

Not as such.

For simple input parsing, the easiest approach is usually to split the line into whitespace-delimited words using the split() method of string objects and then convert decimal strings to numeric values using int() or float(). split() supports an optional "sep" parameter which is useful if the line uses something other than whitespace as a separator.

For more complicated input parsing, regular expressions more powerful than C's sscanf() and better suited for the task. 1.3.9 What does 'UnicodeError: ASCII [decoding,encoding] error: ordinal not in range(128)' mean?

This error indicates that your Python installation can handle only 7-bit ASCII strings. There are a couple ways to fix or work around the problem.

If your programs must handle data in arbitrary character set encodings, the environment the application runs in will generally identify the encoding of the data it is handing you. You need to convert the input to Unicode data using that encoding. For example, a program that handles email or web input will typically find character set encoding information in Content-Type headers. This can then be used to properly convert input data to Unicode. Assuming the string referred to by value is encoded as UTF-8:

value = unicode(value, "utf-8")

will return a Unicode object. If the data is not correctly encoded as UTF-8, the above call will raise a UnicodeError exception.

If you only want strings converted to Unicode which have non-ASCII data, you can try converting them first assuming an ASCII encoding, and then generate Unicode objects if that fails:

```
try:
x = unicode(value, "ascii")
except UnicodeError:
value = unicode(value, "utf-8")
else:
# value was valid ASCII data
pass
```

It's possible to set a default encoding in a file called sitecustomize.py that's part of the Python library. However, this isn't recommended because changing the Python-wide default encoding may cause third-party extension modules to fail.

Note that on Windows, there is an encoding known as "mbcs", which uses an encoding specific to your current locale. In many cases, and particularly when working with COM, this may be an appropriate default encoding to use.

How do I convert between tuples and lists?
The function tuple(seq) converts any sequence (actually, any iterable) into a tuple with the same items in the same order.

For example, tuple([1, 2, 3]) yields (1, 2, 3) and tuple('abc') yields ('a', 'b', 'c'). If the argument is a tuple, it does not make a copy but returns the same object, so it is cheap to call tuple() when you aren't sure that an object is already a tuple.

The function list(seq) converts any sequence or iterable into a list with the same items in the same order. For example, list((1, 2, 3)) yields [1, 2, 3] and list('abc') yields ['a', 'b', 'c']. If the argument is a list, it makes a copy just like seq[:] would.

What's a negative index?
Python sequences are indexed with positive numbers and negative numbers. For positive numbers 0 is the first index 1 is the second index and so forth. For negative indices -1 is the last index and -2 is the penultimate (next to last) index and so forth. Think of seq[-n] as the same as seq[len(seq)-n].

Using negative indices can be very convenient. For example S[:-1] is all of the string except for its last character, which is useful for removing the trailing newline from a string.

How do I iterate over a sequence in reverse order?
If it is a list, the fastest solution is

```
list.reverse()
try:
for x in list:
"do something with x"
finally:
list.reverse()
```

This has the disadvantage that while you are in the loop, the list is temporarily reversed. If you don't like this, you can make a copy. This appears expensive but is actually faster than other solutions:

```
rev = list[:]
rev.reverse()
for x in rev:
<do something with x>
```

If it's not a list, a more general but slower solution is:

```
for i in range(len(sequence)-1, -1, -1):
x = sequence[i]
<do something with x>
```

A more elegant solution, is to define a class which acts as a sequence and yields the elements in reverse order (solution due to Steve Majewski):

```
class Rev:
def __init__(self, seq):
self.forw = seq
def __len__(self):
return len(self.forw)
def __getitem__(self, i):
return self.forw[-(i + 1)]
```

You can now simply write:

```
for x in Rev(list):
<do something with x>
```

Unfortunately, this solution is slowest of all, due to the method call overhead.

With Python 2.3, you can use an extended slice syntax:

```
for x in sequence[::-1]:
<do something with x>
```

How do you remove duplicates from a list?
If you don't mind reordering the list, sort it and then scan from the end of the list, deleting duplicates as you go:

```
if List:
List.sort()
last = List[-1]
for i in range(len(List)-2, -1, -1):
if last==List[i]: del List[i]
else: last=List[i]
```

If all elements of the list may be used as dictionary keys (i.e. they are all hash able) this is often faster

```
d = {}
for x in List: d[x]=x
List = d.values()
```

How do you make an array in Python?
Use a list:
["this", 1, "is", "an", "array"]

Lists are equivalent to C or Pascal arrays in their time complexity; the primary difference is that a Python list can contain objects of many different types.

The array module also provides methods for creating arrays of fixed types with compact representations, but they are slower to index than lists. Also note that the Numeric extensions and others define array-like structures with various characteristics as well.

To get Lisp-style linked lists, you can emulate cons cells using tuples:

```
lisp_list = ("like", ("this", ("example", None) ) )
```

If mutability is desired, you could use lists instead of tuples. Here the analogue of lisp car is lisp_list[0] and the analogue of cdr is lisp_list[1]. Only do this if you're sure you really need to, because it's usually a lot slower than using Python lists.

How do I create a multidimensional list?
You probably tried to make a multidimensional array like this:

```
A = [[None] * 2] * 3
```

This looks correct if you print it:

```
>>> A
[[None, None], [None, None], [None, None]]
```

But when you assign a value, it shows up in multiple places:

```
>>> A[0][0] = 5
>>> A
[[5, None], [5, None], [5, None]]
```

The reason is that replicating a list with * doesn't create copies, it only creates references to the existing objects. The *3 creates a list containing 3 references to the same list of length two. Changes to one row will show in all rows, which is almost certainly not what you want.

The suggested approach is to create a list of the desired length first and then fill in each element with a newly created list:

```
A = [None]*3
for i in range(3):
A[i] = [None] * 2
```

This generates a list containing 3 different lists of length two. You can also use a list comprehension:

```
w,h = 2,3
A = [ [None]*w for i in range(h) ]
```

Or, you can use an extension that provides a matrix datatype; Numeric Python is the best known.

How do I apply a method to a sequence of objects?
Use a list comprehension:

```
result = [obj.method() for obj in List]
```

More generically, you can try the following function:

```
def method_map(objects, method, arguments):
"""method_map([a,b], "meth", (1,2)) gives [a.meth(1,2), b.meth(1,2)]"""
nobjects = len(objects)
methods = map(getattr, objects, [method]*nobjects)
return map(apply, methods, [arguments]*nobjects)
```

I want to do a complicated sort: can you do a Schwartzman Transform in Python?
Yes, it's quite simple with list comprehensions.

The technique, attributed to Randal Schwartz of the Perl community, sorts the elements of a list by a metric which maps each element to its "sort value". To sort a list of strings by their uppercase values:

```
tmp1 = [ (x.upper(), x) for x in L ] # Schwartzman transform
tmp1.sort()
Usorted = [ x[1] for x in tmp1 ]
```

To sort by the integer value of a subfield extending from positions 10-15 in each string:

```
tmp2 = [ (int(s[10:15]), s) for s in L ] # Schwartzman transform
tmp2.sort()
Isorted = [ x[1] for x in tmp2 ]
```

Note that Isorted may also be computed by

```
def intfield(s):
return int(s[10:15])

def Icmp(s1, s2):
return cmp(intfield(s1), intfield(s2))

Isorted = L[:]
Isorted.sort(Icmp)
```

but since this method calls intfield() many times for each element of L, it is slower than the Schwartzman Transform.

How can I sort one list by values from another list?
Merge them into a single list of tuples, sort the resulting list, and then pick out the element you want.

```
>>> list1 = ["what", "I'm", "sorting", "by"]
>>> list2 = ["something", "else", "to", "sort"]
>>> pairs = zip(list1, list2)
>>> pairs

[('what', 'something'), ("I'm", 'else'), ('sorting', 'to'), ('by', 'sort')] >>> pairs.sort()
>>> result = [ x[1] for x in pairs ]
>>> result
['else', 'sort', 'to', 'something']
```

An alternative for the last step is:

```
result = []
for p in pairs: result.append(p[1])
```

If you find this more legible, you might prefer to use this instead of the final list comprehension. However, it is almost twice as slow for long lists. Why? First, the append() operation has to reallocate memory, and while it uses some tricks to avoid doing that each time, it still has to do it occasionally, and that costs quite a bit. Second, the expression "result.append" requires an extra attribute lookup, and third, there's a speed reduction from having to make all those function calls.

What is a class?
A class is the particular object type created by executing a class statement. Class objects are used as templates to create instance objects, which embody both the data (attributes) and code (methods) specific to a datatype.

A class can be based on one or more other classes, called its base class(es). It then inherits the attributes and methods of its base classes. This allows an object model to be successively refined by inheritance. You might have a generic Mailbox class that provides basic accessor methods for a mailbox, and subclasses such as MboxMailbox, MaildirMailbox, OutlookMailbox that handle various specific mailbox formats.

What is a method?
A method is a function on some object x that you normally call as x.name(arguments...). Methods are defined as functions inside the class definition:

```
class C:
def meth (self, arg):
return arg*2 + self.attribute
```

What is self?
Self is merely a conventional name for the first argument of a method. A method defined as meth(self, a, b, c) should be called as x.meth(a, b, c) for some instance x of the class in which the definition occurs; the called method will think it is called as meth(x, a, b, c).

How do I check if an object is an instance of a given class or of a subclass of it?
Use the built-in function isinstance(obj, cls). You can check if an object is an instance of any of a

number of classes by providing a tuple instead of a single class, e.g. isinstance(obj, (class1, class2, ...)), and can also check whether an object is one of Python's built-in types, e.g. isinstance(obj, str) or isinstance(obj, (int, long, float, complex)).

Note that most programs do not use isinstance() on user-defined classes very often. If you are developing the classes yourself, a more proper object-oriented style is to define methods on the classes that encapsulate a particular behaviour, instead of checking the object's class and doing a different thing based on what class it is. For example, if you have a function that does something:

```
def search (obj):
if isinstance(obj, Mailbox):
# ... code to search a mailbox
elif isinstance(obj, Document):
# ... code to search a document
elif ...
```

A better approach is to define a search() method on all the classes and just call it:

```
class Mailbox:
def search(self):
# ... code to search a mailbox

class Document:
def search(self):
# ... code to search a document

obj.search()
```

What is delegation?
Delegation is an object oriented technique (also called a design pattern). Let's say you have an object x and want to change the behavior of just one of its methods. You can create a new class that provides a new implementation of the method you're interested in changing and delegates all other methods to the corresponding method of x.

Python programmers can easily implement delegation. For example, the following class implements a class that behaves like a file but converts all written data to uppercase:

```
class UpperOut:
def __init__(self, outfile):
self.__outfile = outfile
def write(self, s):
self.__outfile.write(s.upper())
def __getattr__(self, name):
return getattr(self.__outfile, name)
```

Here the UpperOut class redefines the write() method to convert the argument string to uppercase before calling the underlying self.__outfile.write() method. All other methods are delegated to the

underlying self.__outfile object. The delegation is accomplished via the __getattr__ method; consult the language reference for more information about controlling attribute access.

Note that for more general cases delegation can get trickier. When attributes must be set as well as retrieved, the class must define a __settattr__ method too, and it must do so carefully. The basic implementation of __setattr__ is roughly equivalent to the following:

```
class X:
...
def __setattr__(self, name, value):
self.__dict__[name] = value
...
```

Most __setattr__ implementations must modify self.__dict__ to store local state for self without causing an infinite recursion.

How do I call a method defined in a base class from a derived class that overrides it?
If you're using new-style classes, use the built-in super() function:

```
class Derived(Base):
def meth (self):
super(Derived, self).meth()
```

If you're using classic classes: For a class definition such as class Derived(Base): ... you can call method meth() defined in Base (or one of Base's base classes) as Base.meth(self, arguments...). Here, Base.meth is an unbound method, so you need to provide the self argument.

How can I organize my code to make it easier to change the base class?
You could define an alias for the base class, assign the real base class to it before your class definition, and use the alias throughout your class. Then all you have to change is the value assigned to the alias. Incidentally, this trick is also handy if you want to decide dynamically (e.g. depending on availability of resources) which base class to use. Example:

```
BaseAlias = <real base class>
class Derived(BaseAlias):
def meth(self):
BaseAlias.meth(self)
```

How do I create static class data and static class methods?
Static data (in the sense of C++ or Java) is easy; static methods (again in the sense of C++ or Java) are not supported directly.

For static data, simply define a class attribute. To assign a new value to the attribute, you have to explicitly use the class name in the assignment:

```
class C:
count = 0 # number of times C.__init__ called
```

```
def __init__(self):
C.count = C.count + 1

def getcount(self):
return C.count # or return self.count
```

c.count also refers to C.count for any c such that isinstance(c, C) holds, unless overridden by c itself or by some class on the base-class search path from c.__class__ back to C.

Caution: within a method of C, an assignment like self.count = 42 creates a new and unrelated instance vrbl named "count" in self's own dict. Rebinding of a class-static data name must always specify the class whether inside a method or not:

```
C.count = 314
```

Static methods are possible when you're using new-style classes:

```
class C:
def static(arg1, arg2, arg3):
# No 'self' parameter!
...
static = staticmethod(static)
```

However, a far more straightforward way to get the effect of a static method is via a simple module-level function:

```
def getcount():
return C.count
```

If your code is structured so as to define one class (or tightly related class hierarchy) per module, this supplies the desired encapsulation.

How can I overload constructors (or methods) in Python?

This answer actually applies to all methods, but the question usually comes up first in the context of constructors.

In C++ you'd write

```
class C {
C() { cout << "No arguments\n"; }
C(int i) { cout << "Argument is " << i << "\n"; }
}
```

in Python you have to write a single constructor that catches all cases using default arguments. For example:

```
class C:
```

```
def __init__(self, i=None):
if i is None:
print "No arguments"
else:
print "Argument is", i
```

This is not entirely equivalent, but close enough in practice.

You could also try a variable-length argument list, e.g.

```
def __init__(self, *args):
....
```

The same approach works for all method definitions.

How do I find the current module name?

A module can find out its own module name by looking at the predefined global variable __name__. If this has the value '__main__', the program is running as a script. Many modules that are usually used by importing them also provide a command-line interface or a self-test, and only execute this code after checking __name__:

```
def main():
print 'Running test...'
...

if __name__ == '__main__':
main()
__import__('x.y.z') returns
```

Try:

```
__import__('x.y.z').y.z
```

For more realistic situations, you may have to do something like

```
m = __import__(s)
for i in s.split(".")[1:]:
m = getattr(m, i)
```

When I edit an imported module and reimport it, the changes don't show up. Why does this happen?

For reasons of efficiency as well as consistency, Python only reads the module file on the first time a module is imported. If it didn't, in a program consisting of many modules where each one imports the same basic module, the basic module would be parsed and re-parsed many times. To force rereading of a changed module, do this:

```
import modname
reload(modname)
```

Warning: this technique is not 100% fool-proof. In particular, modules containing statements like

from modname import some_objects

will continue to work with the old version of the imported objects. If the module contains class definitions, existing class instances will not be updated to use the new class definition. This can result in the following paradoxical behavior:

```
>>> import cls
>>> c = cls.C() # Create an instance of C
>>> reload(cls)
<module 'cls' from 'cls.pyc'>
>>> isinstance(c, cls.C) # isinstance is false?!?
False
```

The nature of the problem is made clear if you print out the class objects:

```
>>> c.__class__
<class cls.C at 0x7352a0>
>>> cls.C
<class cls.C at 0x4198d0>
```

Where is the math.py (socket.py, regex.py, etc.) source file?
There are (at least) three kinds of modules in Python:

1. modules written in Python (.py);
2. modules written in C and dynamically loaded (.dll, .pyd, .so, .sl, etc);
3. modules written in C and linked with the interpreter; to get a list of these, type:
import sys
print sys.builtin_module_names

How do I make a Python script executable on Unix?
You need to do two things: the script file's mode must be executable and the first line must begin with #! followed by the path of the Python interpreter.

The first is done by executing chmod +x scriptfile or perhaps chmod 755 scriptfile.
The second can be done in a number of ways. The most straightforward way is to write

```
#!/usr/local/bin/python
```

as the very first line of your file, using the pathname for where the Python interpreter is installed on your platform.

If you would like the script to be independent of where the Python interpreter lives, you can use the "env" program. Almost all Unix variants support the following, assuming the python interpreter is in a directory on the user's $PATH:

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

Don't do this for CGI scripts. The $PATH variable for CGI scripts is often very minimal, so you need to use the actual absolute pathname of the interpreter.

Occasionally, a user's environment is so full that the /usr/bin/env program fails; or there's no env program at all. In that case, you can try the following hack (due to Alex Rezinsky):

```
#! /bin/sh
""":"
exec python $0 ${1+"$@"}
"""
```

The minor disadvantage is that this defines the script's __doc__ string. However, you can fix that by adding

```
__doc__ = """...Whatever..."""
```

Why don't my signal handlers work?

The most common problem is that the signal handler is declared with the wrong argument list. It is called as

```
handler(signum, frame)
```

so it should be declared with two arguments:

```
def handler(signum, frame):
...
```

How do I test a Python program or component?

Python comes with two testing frameworks. The doctest module finds examples in the docstrings for a module and runs them, comparing the output with the expected output given in the docstring.

The unittest module is a fancier testing framework modelled on Java and Smalltalk testing frameworks.

For testing, it helps to write the program so that it may be easily tested by using good modular design. Your program should have almost all functionality encapsulated in either functions or class methods -- and this sometimes has the surprising and delightful effect of making the program run faster (because local variable accesses are faster than global accesses). Furthermore the program should avoid depending on mutating global variables, since this makes testing much more difficult to do.

The "global main logic" of your program may be as simple as

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

at the bottom of the main module of your program.

Once your program is organized as a tractable collection of functions and class behaviours you should write test functions that exercise the behaviours. A test suite can be associated with each module which automates a sequence of tests. This sounds like a lot of work, but since Python is so terse and flexible it's surprisingly easy. You can make coding much more pleasant and fun by writing your test functions in parallel with the "production code", since this makes it easy to find bugs and even design flaws earlier.

"Support modules" that are not intended to be the main module of a program may include a self-test of the module.

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

Even programs that interact with complex external interfaces may be tested when the external interfaces are unavailable by using "fake" interfaces implemented in Python.

None of my threads seem to run: why?
As soon as the main thread exits, all threads are killed. Your main thread is running too quickly, giving the threads no time to do any work.

A simple fix is to add a sleep to the end of the program that's long enough for all the threads to finish:

```
import threading, time

def thread_task(name, n):
for i in range(n): print name, i

for i in range(10):
T = threading.Thread(target=thread_task, args=(str(i), i))
T.start()


time.sleep(10) # <----------------------------!
```

But now (on many platforms) the threads don't run in parallel, but appear to run sequentially, one at a time! The reason is that the OS thread scheduler doesn't start a new thread until the previous thread is blocked.

A simple fix is to add a tiny sleep to the start of the run function:

```
def thread_task(name, n):
time.sleep(0.001) # <---------------------!
for i in range(n): print name, i
```

```
for i in range(10):
T = threading.Thread(target=thread_task, args=(str(i), i))
T.start()

time.sleep(10)
```

Instead of trying to guess how long a time.sleep() delay will be enough, it's better to use some kind of semaphore mechanism. One idea is to use the Queue module to create a queue object, let each thread append a token to the queue when it finishes, and let the main thread read as many tokens from the queue as there are threads.

How do I parcel out work among a bunch of worker threads?
Use the Queue module to create a queue containing a list of jobs. The Queue class maintains a list of objects with .put(obj) to add an item to the queue and .get() to return an item. The class will take care of the locking necessary to ensure that each job is handed out exactly once.

Here's a trivial example:

```
import threading, Queue, time

# The worker thread gets jobs off the queue. When the queue is empty, it
# assumes there will be no more work and exits.
# (Realistically workers will run until terminated.)
def worker ():
print 'Running worker'
time.sleep(0.1)
while True:
try:
arg = q.get(block=False)
except Queue.Empty:
print 'Worker', threading.currentThread(),
print 'queue empty'
break
else:
print 'Worker', threading.currentThread(),
print 'running with argument', arg
time.sleep(0.5)

# Create queue
q = Queue.Queue()

# Start a pool of 5 workers
for i in range(5):
t = threading.Thread(target=worker, name='worker %i' % (i+1))
t.start()

# Begin adding work to the queue
```

```
for i in range(50):
q.put(i)

# Give threads time to run
print 'Main thread sleeping'
time.sleep(5)
```

When run, this will produce the following output:

Running worker Running worker Running worker Running worker Running worker Main thread sleeping Worker <Thread(worker 1, started)> running with argument 0 Worker <Thread(worker 2, started)> running with argument 1 Worker <Thread(worker 3, started)> running with argument 2 Worker <Thread(worker 4, started)> running with argument 3 Worker <Thread(worker 5, started)> running with argument 4 Worker <Thread(worker 1, started)> running with argument 5 ...

How do I delete a file? (And other file questions...)
Use os.remove(filename) or os.unlink(filename);

How do I copy a file?
The shutil module contains a copyfile() function.

How do I read (or write) binary data?
or complex data formats, it's best to use the struct module. It allows you to take a string containing binary data (usually numbers) and convert it to Python objects; and vice versa.

For example, the following code reads two 2-byte integers and one 4-byte integer in big-endian format from a file:

```
import struct

f = open(filename, "rb") # Open in binary mode for portability
s = f.read(8)
x, y, z = struct.unpack(">hhl", s)
```

The '>' in the format string forces big-endian data; the letter 'h' reads one "short integer" (2 bytes), and 'l' reads one "long integer" (4 bytes) from the string.

How do I run a subprocess with pipes connected to both input and output?
Use the popen2 module. For example:

```
import popen2
fromchild, tochild = popen2.popen2("command")
tochild.write("input\n")
tochild.flush()
output = fromchild.readline()
```

How can I mimic CGI form submission (METHOD=POST)?
I would like to retrieve web pages that are the result of POSTing a form. Is there existing code that

would let me do this easily?

Yes. Here's a simple example that uses httplib:

```
#!/usr/local/bin/python

import httplib, sys, time

### build the query string
qs = "First=Josephine&MI=Q&Last=Public"

### connect and send the server a path
httpobj = httplib.HTTP('www.some-server.out-there', 80)
httpobj.putrequest('POST', '/cgi-bin/some-cgi-script')
### now generate the rest of the HTTP headers...
httpobj.putheader('Accept', '*/*')
httpobj.putheader('Connection', 'Keep-Alive')
httpobj.putheader('Content-type', 'application/x-www-form-urlencoded')
httpobj.putheader('Content-length', '%d' % len(qs))
httpobj.endheaders()
httpobj.send(qs)
### find out what the server said in response...
reply, msg, hdrs = httpobj.getreply()
if reply != 200:
sys.stdout.write(httpobj.getfile().read())
```

Note that in general for URL-encoded POST operations, query strings must be quoted by using urllib.quote(). For example to send name="Guy Steele, Jr.":

```
>>> from urllib import quote
>>> x = quote("Guy Steele, Jr.")
>>> x
'Guy%20Steele,%20Jr.'
>>> query_string = "name="+x
>>> query_string
'name=Guy%20Steele,%20Jr.'
```

How do I send mail from a Python script?
Use the standard library module smtplib.

Here's a very simple interactive mail sender that uses it. This method will work on any host that supports an SMTP listener.

```
import sys, smtplib

fromaddr = raw_input("From: ")
toaddrs = raw_input("To: ").split(',')
```

```
print "Enter message, end with ^D:"
msg = ''
while 1:
line = sys.stdin.readline()
if not line:
break
msg = msg + line

# The actual mail send
server = smtplib.SMTP('localhost')
server.sendmail(fromaddr, toaddrs, msg)
server.quit()
```

A Unix-only alternative uses sendmail. The location of the sendmail program varies between systems; sometimes it is /usr/lib/sendmail, sometime /usr/sbin/sendmail. The sendmail manual page will help you out. Here's some sample code:

```
SENDMAIL = "/usr/sbin/sendmail" # sendmail location
import os
p = os.popen("%s -t -i" % SENDMAIL, "w")
p.write("To: receiver@example.com\n")
p.write("Subject: test\n")
p.write("\n") # blank line separating headers from body
p.write("Some text\n")
p.write("some more text\n")
sts = p.close()
if sts != 0:
print "Sendmail exit status", sts
```

How do I avoid blocking in the connect() method of a socket?
The select module is commonly used to help with asynchronous I/O on sockets.

Are there any interfaces to database packages in Python?
Yes.

Python 2.3 includes the bsddb package which provides an interface to the BerkeleyDB library. Interfaces to disk-based hashes such as DBM and GDBM are also included with standard Python.

How do I generate random numbers in Python?
The standard module random implements a random number generator. Usage is simple:

```
import random
random.random()
```

This returns a random floating point number in the range [0, 1).

Can I create my own functions in C?
Yes, you can create built-in modules containing functions, variables, exceptions and even new types in

C.

Can I create my own functions in C++?
Yes, using the C compatibility features found in C++. Place extern "C" { ... } around the Python include files and put extern "C" before each function that is going to be called by the Python interpreter. Global or static C++ objects with constructors are probably not a good idea.

How can I execute arbitrary Python statements from C?
The highest-level function to do this is PyRun_SimpleString() which takes a single string argument to be executed in the context of the module __main__ and returns 0 for success and -1 when an exception occurred (including SyntaxError). If you want more control, use PyRun_String(); see the source for PyRun_SimpleString() in Python/pythonrun.c.

How can I evaluate an arbitrary Python expression from C?
Call the function PyRun_String() from the previous question with the start symbol Py_eval_input; it parses an expression, evaluates it and returns its value.

How do I extract C values from a Python object?
That depends on the object's type. If it's a tuple, PyTupleSize(o) returns its length and PyTuple_GetItem(o, i) returns its i'th item. Lists have similar functions, PyListSize(o) and PyList_GetItem(o, i).

For strings, PyString_Size(o) returns its length and PyString_AsString(o) a pointer to its value. Note that Python strings may contain null bytes so C's strlen() should not be used.

To test the type of an object, first make sure it isn't NULL, and then use PyString_Check(o), PyTuple_Check(o), PyList_Check(o), etc.

There is also a high-level API to Python objects which is provided by the so-called 'abstract' interface -- read Include/abstract.h for further details. It allows interfacing with any kind of Python sequence using calls like PySequence_Length(), PySequence_GetItem(), etc.) as well as many other useful protocols.

How do I call an object's method from C?
The PyObject_CallMethod() function can be used to call an arbitrary method of an object. The parameters are the object, the name of the method to call, a format string like that used with Py_BuildValue(), and the argument values:

```
PyObject *
PyObject_CallMethod(PyObject *object, char *method_name,
char *arg_format, ...);
```

This works for any object that has methods -- whether built-in or user-defined. You are responsible for eventually Py_DECREF'ing the return value.

To call, e.g., a file object's "seek" method with arguments 10, 0 (assuming the file object pointer is "f"):

```
res = PyObject_CallMethod(f, "seek", "(ii)", 10, 0);
if (res == NULL) {
```

```
... an exception occurred ...
}
else {
Py_DECREF(res);
}
```

Note that since PyObject_CallObject() always wants a tuple for the argument list, to call a function without arguments, pass "()" for the format, and to call a function with one argument, surround the argument in parentheses, e.g. "(i)".

How do I catch the output from PyErr_Print() (or anything that prints to stdout/stderr)?
In Python code, define an object that supports the write() method. Assign this object to sys.stdout and sys.stderr. Call print_error, or just allow the standard traceback mechanism to work. Then, the output will go wherever your write() method sends it.

The easiest way to do this is to use the StringIO class in the standard library.

Sample code and use for catching stdout:

```
>>> class StdoutCatcher:
... def __init__(self):
... self.data = ''
... def write(self, stuff):
... self.data = self.data + stuff
...
>>> import sys
>>> sys.stdout = StdoutCatcher()
>>> print 'foo'
>>> print 'hello world!'
>>> sys.stderr.write(sys.stdout.data)
foo
hello world!
```

How do I access a module written in Python from C?
You can get a pointer to the module object as follows:

```
module = PyImport_ImportModule("<modulename>");
```

If the module hasn't been imported yet (i.e. it is not yet present in sys.modules), this initializes the module; otherwise it simply returns the value of sys.modules["<modulename>"]. Note that it doesn't enter the module into any namespace -- it only ensures it has been initialized and is stored in sys.modules.

You can then access the module's attributes (i.e. any name defined in the module) as follows:

```
attr = PyObject_GetAttrString(module, "<attrname>");
```

Calling PyObject_SetAttrString() to assign to variables in the module also works.

How do I interface to C++ objects from Python?
Depending on your requirements, there are many approaches. To do this manually, begin by reading the "Extending and Embedding" document. Realize that for the Python run-time system, there isn't a whole lot of difference between C and C++ -- so the strategy of building a new Python type around a C structure (pointer) type will also work for C++ objects.

How do I tell "incomplete input" from "invalid input"?
Sometimes you want to emulate the Python interactive interpreter's behavior, where it gives you a continuation prompt when the input is incomplete (e.g. you typed the start of an "if" statement or you didn't close your parentheses or triple string quotes), but it gives you a syntax error message immediately when the input is invalid.

In Python you can use the codeop module, which approximates the parser's behavior sufficiently. IDLE uses this, for example.

The easiest way to do it in C is to call PyRun_InteractiveLoop() (perhaps in a separate thread) and let the Python interpreter handle the input for you. You can also set the PyOS_ReadlineFunctionPointer to point at your custom input function. See Modules/readline.c and Parser/myreadline.c for more hints.

However sometimes you have to run the embedded Python interpreter in the same thread as your rest application and you can't allow the PyRun_InteractiveLoop() to stop while waiting for user input. The one solution then is to call PyParser_ParseString() and test for e.error equal to E_EOF, which means the input is incomplete). Here's a sample code fragment, untested, inspired by code from Alex Farber:

```
#include <Python.h>
#include <node.h>
#include <errcode.h>
#include <grammar.h>
#include <parsetok.h>
#include <compile.h>

int testcomplete(char *code)
/* code should end in \n */
/* return -1 for error, 0 for incomplete,
1 for complete */
{
node *n;
perrdetail e;

n = PyParser_ParseString(code, &_PyParser_Grammar,
Py_file_input, &e);
if (n == NULL) {
if (e.error == E_EOF)
return 0;
return -1;
}
```

```
PyNode_Free(n);
return 1;
}
```

Another solution is trying to compile the received string with Py_CompileString(). If it compiles without errors, try to execute the returned code object by calling PyEval_EvalCode(). Otherwise save the input for later. If the compilation fails, find out if it's an error or just more input is required - by extracting the message string from the exception tuple and comparing it to the string "unexpected EOF while parsing". Here is a complete example using the GNU readline library (you may want to ignore SIGINT while calling readline()):

```
#include <stdio.h>
#include <readline.h>

#include <Python.h>
#include <object.h>
#include <compile.h>
#include <eval.h>

int main (int argc, char* argv[])
{
int i, j, done = 0; /* lengths of line, code */
char ps1[] = ">>> ";
char ps2[] = "... ";
char *prompt = ps1;
char *msg, *line, *code = NULL;
PyObject *src, *glb, *loc;
PyObject *exc, *val, *trb, *obj, *dum;

Py_Initialize ();
loc = PyDict_New ();
glb = PyDict_New ();
PyDict_SetItemString (glb, "__builtins__",
PyEval_GetBuiltins ());

while (!done)
{
line = readline (prompt);

if (NULL == line) /* CTRL-D pressed */
{
done = 1;
}
else
{
```

```
i = strlen (line);

if (i > 0)
add_history (line);
/* save non-empty lines */

if (NULL == code)
/* nothing in code yet */
j = 0;
else
j = strlen (code);

code = realloc (code, i + j + 2);
if (NULL == code)
/* out of memory */
exit (1);

if (0 == j)
/* code was empty, so */
code[0] = '\0';
/* keep strncat happy */

strncat (code, line, i);
/* append line to code */
code[i + j] = '\n';
/* append '\n' to code */
code[i + j + 1] = '\0';

src = Py_CompileString (code, " <stdin>", Py_single_input);

if (NULL != src)
/* compiled just fine - */
{
if (ps1 == prompt ||
/* ">>> " or */
'\n' == code[i + j - 1])
/* "... " and double '\n' */
{
/* so execute it */
dum = PyEval_EvalCode ((PyCodeObject *)src, glb, loc);
Py_XDECREF (dum);
Py_XDECREF (src);
free (code);
code = NULL;
if (PyErr_Occurred ())
PyErr_Print ();
```

```
prompt = ps1;
}
}
/* syntax error or E_EOF? */

else if (PyErr_ExceptionMatches (PyExc_SyntaxError))
{
PyErr_Fetch (&exc, &val, &trb);
/* clears exception! */

if (PyArg_ParseTuple (val, "sO", &msg, &obj) &&
!strcmp (msg, "unexpected EOF while parsing")) /* E_EOF */
{
Py_XDECREF (exc);
Py_XDECREF (val);
Py_XDECREF (trb);
prompt = ps2;
}
else
/* some other syntax error */
{
PyErr_Restore (exc, val, trb);
PyErr_Print ();
free (code);
code = NULL;
prompt = ps1;
}
}
else
/* some non-syntax error */
{
PyErr_Print ();
free (code);
code = NULL;
prompt = ps1;
}

free (line);
}
}

Py_XDECREF(glb);
Py_XDECREF(loc);
Py_Finalize();
exit(0);
}
```

How do I run a Python program under Windows?
This is not necessarily a straightforward question. If you are already familiar with running programs from the Windows command line then everything will seem obvious; otherwise, you might need a little more guidance. There are also differences between Windows 95, 98, NT, ME, 2000 and XP which can add to the confusion.

Unless you use some sort of integrated development environment, you will end up typing Windows commands into what is variously referred to as a "DOS window" or "Command prompt window". Usually you can create such a window from your Start menu; under Windows 2000 the menu selection is "Start | Programs | Accessories | Command Prompt". You should be able to recognize when you have started such a window because you will see a Windows "command prompt", which usually looks like this:

C:\>

The letter may be different, and there might be other things after it, so you might just as easily see something like:

D:\Steve\Projects\Python>

depending on how your computer has been set up and what else you have recently done with it. Once you have started such a window, you are well on the way to running Python programs.

You need to realize that your Python scripts have to be processed by another program called the Python interpreter. The interpreter reads your script, compiles it into bytecodes, and then executes the bytecodes to run your program. So, how do you arrange for the interpreter to handle your Python?

First, you need to make sure that your command window recognises the word "python" as an instruction to start the interpreter. If you have opened a command window, you should try entering the command python and hitting return. You should then see something like:

```
Python 2.2 (#28, Dec 21 2001, 12:21:22) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You have started the interpreter in "interactive mode". That means you can enter Python statements or expressions interactively and have them executed or evaluated while you wait. This is one of Python's strongest features. Check it by entering a few expressions of your choice and seeing the results:

```
>>> print "Hello"
Hello
>>> "Hello" * 3
HelloHelloHello
```

Many people use the interactive mode as a convenient yet highly programmable calculator. When you

want to end your interactive Python session, hold the Ctrl key down while you enter a Z, then hit the "Enter" key to get back to your Windows command prompt.

You may also find that you have a Start-menu entry such as "Start | Programs | Python 2.2 | Python (command line)" that results in you seeing the >>> prompt in a new window. If so, the window will disappear after you enter the Ctrl-Z character; Windows is running a single "python" command in the window, and closes it when you terminate the interpreter.

If the python command, instead of displaying the interpreter prompt >>>, gives you a message like:

'python' is not recognized as an internal or external command,
operable program or batch file.

or:

Bad command or filename

then you need to make sure that your computer knows where to find the Python interpreter. To do this you will have to modify a setting called PATH, which is a list of directories where Windows will look for programs. You should arrange for Python's installation directory to be added to the PATH of every command window as it starts. If you installed Python fairly recently then the command

dir C:\py*

will probably tell you where it is installed; the usual location is something like C:\Python23. Otherwise you will be reduced to a search of your whole disk ... use "Tools | Find" or hit the "Search" button and look for "python.exe". Supposing you discover that Python is installed in the C:\Python23 directory (the default at the time of writing), you should make sure that entering the command

c:\Python23\python

starts up the interpreter as above (and don't forget you'll need a "CTRL-Z" and an "Enter" to get out of it). Once you have verified the directory, you need to add it to the start-up routines your computer goes through. For older versions of Windows the easiest way to do this is to edit the C:\AUTOEXEC.BAT file. You would want to add a line like the following to AUTOEXEC.BAT:

PATH C:\Python23;%PATH%

For Windows NT, 2000 and (I assume) XP, you will need to add a string such as

;C:\Python23

to the current setting for the PATH environment variable, which you will find in the properties window of "My Computer" under the "Advanced" tab. Note that if you have sufficient privilege you might get a choice of installing the settings either for the Current User or for System. The latter is preferred if you want everybody to be able to run Python on the machine.

If you aren't confident doing any of these manipulations yourself, ask for help! At this stage you may want to reboot your system to make absolutely sure the new setting has taken effect. You probably won't need to reboot for Windows NT, XP or 2000. You can also avoid it in earlier versions by editing the file C:\WINDOWS\COMMAND\CMDINIT.BAT instead of AUTOEXEC.BAT.

You should now be able to start a new command window, enter python at the C:> (or whatever) prompt, and see the >>> prompt that indicates the Python interpreter is reading interactive commands.

Let's suppose you have a program called pytest.py in directory C:\Steve\Projects\Python. A session to run that program might look like this:

C:\> cd \Steve\Projects\Python
C:\Steve\Projects\Python> python pytest.py

Because you added a file name to the command to start the interpreter, when it starts up it reads the Python script in the named file, compiles it, executes it, and terminates, so you see another C:\> prompt. You might also have entered

C:\> python \Steve\Projects\Python\pytest.py

if you hadn't wanted to change your current directory.

Under NT, 2000 and XP you may well find that the installation process has also arranged that the command pytest.py (or, if the file isn't in the current directory, C:\Steve\Projects\Python\pytest.py) will automatically recognize the ".py" extension and run the Python interpreter on the named file. Using this feature is fine, but some versions of Windows have bugs which mean that this form isn't exactly equivalent to using the interpreter explicitly, so be careful.

The important things to remember are:
1. Start Python from the Start Menu, or make sure the PATH is set correctly so Windows can find the Python interpreter.

python

should give you a '>>>" prompt from the Python interpreter. Don't forget the CTRL-Z and ENTER to terminate the interpreter (and, if you started the window from the Start Menu, make the window disappear).

2. Once this works, you run programs with commands:

python {program-file}

3. When you know the commands to use you can build Windows shortcuts to run the Python interpreter on any of your scripts, naming particular working directories, and adding them to your menus. Take a look at

python --help

if your needs are complex.
4. Interactive mode (where you see the >>> prompt) is best used for checking that individual statements and expressions do what you think they will, and for developing code by experiment.

How do I make python scripts executable?
On Windows 2000, the standard Python installer already associates the .py extension with a file type (Python.File) and gives that file type an open command that runs the interpreter (D:\Program Files\Python\python.exe "%1" %*). This is enough to make scripts executable from the command prompt as 'foo.py'. If you'd rather be able to execute the script by simple typing 'foo' with no extension you need to add .py to the PATHEXT environment variable.

On Windows NT, the steps taken by the installer as described above allow you to run a script with 'foo.py', but a longtime bug in the NT command processor prevents you from redirecting the input or output of any script executed in this way. This is often important.

The incantation for making a Python script executable under WinNT is to give the file an extension of .cmd and add the following as the first line:

@setlocal enableextensions & python -x %~f0 %* & goto :EOF

How do I debug an extension?
When using GDB with dynamically loaded extensions, you can't set a breakpoint in your extension until your extension is loaded.

In your .gdbinit file (or interactively), add the command:

br _PyImport_LoadDynamicModule

Then, when you run GDB:

$ gdb /local/bin/python
gdb) run myscript.py
gdb) continue # repeat until your extension is loaded
gdb) finish # so that your extension is loaded
gdb) br myfunction.c:50
gdb) continue

Where is Freeze for Windows?
"Freeze" is a program that allows you to ship a Python program as a single stand-alone executable file. It is not a compiler; your programs don't run any faster, but they are more easily distributable, at least to platforms with the same OS and CPU.

Is a *.pyd file the same as a DLL?

Yes, .

How can I embed Python into a Windows application?
Embedding the Python interpreter in a Windows app can be summarized as follows:

1. Do _not_ build Python into your .exe file directly. On Windows, Python must be a DLL to handle importing modules that are themselves DLL's. (This is the first key undocumented fact.) Instead, link to pythonNN.dll; it is typically installed in C:\Windows\System. NN is the Python version, a number such as "23" for Python 2.3.

You can link to Python statically or dynamically. Linking statically means linking against pythonNN.lib, while dynamically linking means linking against pythonNN.dll. The drawback to dynamic linking is that your app won't run if pythonNN.dll does not exist on your system. (General note: pythonNN.lib is the so-called "import lib" corresponding to python.dll. It merely defines symbols for the linker.)

Linking dynamically greatly simplifies link options; everything happens at run time. Your code must load pythonNN.dll using the Windows LoadLibraryEx() routine. The code must also use access routines and data in pythonNN.dll (that is, Python's C API's) using pointers obtained by the Windows GetProcAddress() routine. Macros can make using these pointers transparent to any C code that calls routines in Python's C API.

Borland note: convert pythonNN.lib to OMF format using Coff2Omf.exe first.

2. If you use SWIG, it is easy to create a Python "extension module" that will make the app's data and methods available to Python. SWIG will handle just about all the grungy details for you. The result is C code that you link into your .exe file (!) You do _not_ have to create a DLL file, and this also simplifies linking.

3. SWIG will create an init function (a C function) whose name depends on the name of the extension module. For example, if the name of the module is leo, the init function will be called initleo(). If you use SWIG shadow classes, as you should, the init function will be called initleoc(). This initializes a mostly hidden helper class used by the shadow class.

The reason you can link the C code in step 2 into your .exe file is that calling the initialization function is equivalent to importing the module into Python! (This is the second key undocumented fact.)

4. In short, you can use the following code to initialize the Python interpreter with your extension module.

```
#include "python.h"
...
Py_Initialize(); // Initialize Python.
initmyAppc(); // Initialize (import) the helper class.
PyRun_SimpleString("import myApp") ; // Import the shadow class.
```

5. There are two problems with Python's C API which will become apparent if you use a compiler other

than MSVC, the compiler used to build pythonNN.dll.

Problem 1: The so-called "Very High Level" functions that take FILE * arguments will not work in a multi-compiler environment because each compiler's notion of a struct FILE will be different. From an implementation standpoint these are very _low_ level functions.

Problem 2: SWIG generates the following code when generating wrappers to void functions:

```
Py_INCREF(Py_None);
_resultobj = Py_None;
return _resultobj;
```

Alas, Py_None is a macro that expands to a reference to a complex data structure called _Py_NoneStruct inside pythonNN.dll. Again, this code will fail in a mult-compiler environment. Replace such code by:

```
return Py_BuildValue("");
```

It may be possible to use SWIG's %typemap command to make the change automatically, though I have not been able to get this to work (I'm a complete SWIG newbie).

6. Using a Python shell script to put up a Python interpreter window from inside your Windows app is not a good idea; the resulting window will be independent of your app's windowing system. Rather, you (or the wxPythonWindow class) should create a "native" interpreter window. It is easy to connect that window to the Python interpreter. You can redirect Python's i/o to _any_ object that supports read and write, so all you need is a Python object (defined in your extension module) that contains read() and write() methods.

How do I use Python for CGI?

On the Microsoft IIS server or on the Win95 MS Personal Web Server you set up Python in the same way that you would set up any other scripting engine.

Run regedt32 and go to:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\ScriptMap

and enter the following line (making any specific changes that your system may need):

```
.py :REG_SZ: c:\\python.exe -u %s %s
```

This line will allow you to call your script with a simple reference like: http://yourserver/scripts/yourscript.py provided "scripts" is an "executable" directory for your server (which it usually is by default). The "-u" flag specifies unbuffered and binary mode for stdin - needed when working with binary data.

In addition, it is recommended that using ".py" may not be a good idea for the file extensions when used in this context (you might want to reserve *.py for support modules and use *.cgi or *.cgp for

"main program" scripts).

In order to set up Internet Information Services 5 to use Python for CGI processing, please see the following links:

http://www.e-coli.net/pyiis_server.html (for Win2k Server) http://www.e-coli.net/pyiis.html (for Win2k pro)

Configuring Apache is much simpler. In the Apache configuration file httpd.conf, add the following line at the end of the file:

ScriptInterpreterSource Registry

Then, give your Python CGI-scripts the extension .py and put them in the cgi-bin directory.

How do I emulate os.kill() in Windows?
Use win32api:

```
def kill(pid):
"""kill function for Win32"""
import win32api
handle = win32api.OpenProcess(1, 0, pid)
return (0 != win32api.TerminateProcess(handle, 0))
```

Why does os.path.isdir() fail on NT shared directories?
The solution appears to be always append the "\" on the end of shared drives.

```
>>> import os
>>>os.path.isdir( '\\\\rorschach\\public')
0
>>>os.path.isdir( '\\\\rorschach\\public\\')
1
```

It helps to think of share points as being like drive letters. Example:

k: is not a directory
k:\ is a directory
k:\media is a directory
k:\media\ is not a directory

The same rules apply if you substitute "k:" with "\conkyfoo":

\\conky\foo is not a directory
\\conky\foo\ is a directory
\\conky\foo\media is a directory
\\conky\foo\media\ is not a directory

Web Python

Some host providers only let you run CGI scripts in a certain directory, often named cgi-bin. In this case all you have to do to run the script is to call it like this:

http://my_server.tld/cgi-bin/my_script.py

The script will have to be made executable by "others". Give it a 755 permission or check the executable boxes if there is a graphical FTP interface.

Some hosts let you run CGI scripts in any directory. In some of these hosts you don't have to do anything do configure the directories. In others you will have to add these lines to a file named .htaccess in the directory you want to run CGI scripts from:

```
Options +ExecCGI
AddHandler cgi-script .py
```

If the file does not exist create it. All directories below a directory with a .htaccess file will inherit the configurations. So if you want to be able to run CGI scripts from all directories create this file in the document root.

To run a script saved at the root:

http://my_server.tld/my_script.py

If it was saved in some directory:

http://my_server.tld/some_dir/some_subdir/my_script.py

Make sure all text files you upload to the server are uploaded as text (not binary), specially if you are in Windows, otherwise you will have problems.

The classical "Hello World" in python CGI fashion:

```
#!/usr/bin/env python
print "Content-Type: text/html"
print
print """\
<html>
<body>
<h2>Hello World!
</body>
</html>
"""
```

To test your setup save it with the .py extension, upload it to your server as text and make it executable before trying to run it.

The first line of a python CGI script sets the path where the python interpreter will be found in the

server. Ask your provider what is the correct one. If it is wrong the script will fail. Some examples:

```
#!/usr/bin/python
#!/usr/bin/python2.3
#!/usr/bin/python2.4
```

It is necessary that the script outputs the HTTP header. The HTTP header consists of one or more messages followed by a blank line. If the output of the script is to be interpreted as HTML then the content type will be text/html. The blank line signals the end of the header and is required.

```
print "Content-Type: text/html"
print
```

If you change the content type to text/plain the browser will not interpret the script's output as HTML but as pure text and you will only see the HTML source. Try it now to never forget. A page refresh may be necessary for it to work.

Client versus Server

All python code will be executed at the server only. The client's agent (for example the browser) will never see a single line of python. Instead it will only get the script's output. This is something realy important to understand.

When programming for the Web you are in a client-server environment, that is, do not make things like trying to open a file in the client's computer as if the script were running there. It isn't.

How to Debugging in python?
Syntax and header errors are hard to catch unless you have access to the server logs. Syntax error messages can be seen if the script is run in a local shell before uploading to the server.

For a nice exceptions report there is the cgitb module. It will show a traceback inside a context. The default output is sent to standard output as HTML:

```
#!/usr/bin/env python
print "Content-Type: text/html"
print
import cgitb; cgitb.enable()
print 1/0
```

The handler() method can be used to handle only the catched exceptions:

```
#!/usr/bin/env python
print "Content-Type: text/html"
print
import cgitb
try:
f = open('non-existent-file.txt', 'r')
```

except:
cgitb.handler()

There is also the option for a crude approach making the header "text/plain" and setting the standard error to standard out:

```
#!/usr/bin/env python
print "Content-Type: text/plain"
print
import sys
sys.stderr = sys.stdout
f = open('non-existent-file.txt', 'r')
```

Will output this:

```
Traceback (most recent call last):
File "/var/www/html/teste/cgi-bin/text_error.py", line 6, in ?
f = open('non-existent-file.txt', 'r')
IOError: [Errno 2] No such file or directory: 'non-existent-file.txt'
```

Warning: These techniques expose information that can be used by an attacker. Use it only while developing/debugging. Once in production disable it.

How to make Forms in python?
The FieldStorage class of the cgi module has all that is needed to handle submited forms.

```
import cgi
form = cgi.FieldStorage() # instantiate only once!
```

It is transparent to the programmer if the data was submited by GET or by POST. The interface is exactly the same.

* Unique field names :

Suppose we have this HTML form which submits a field named name to a python CGI script named process_form.py:

```
<html><body>
<form method="get" action="process_form.py">
Name: <input type="text" name="name">
<input type="submit" value="Submit">
</form>
</body></html>
```

This is the process_form.py script:

```
#!/usr/bin/env python
import cgi
form = cgi.FieldStorage() # instantiate only once!
name = form.getfirst('name', 'empty')

# Avoid script injection escaping the user input
name = cgi.escape(name)

print """\
Content-Type: text/html\n
<html><body>
<p>The submited name was "%s"</p>
</body></html>
""" % name
```

The getfirst() method returns the first value of the named field or a default or None if no field with that name was submited or if it is empty. If there is more than one field with the same name only the first will be returned.

If you change the HTML form method from get to post the process_form.py script will be the same.

* Multiple field names:

If there is more than one field with the same name like in HTML input check boxes then the method to be used is getlist(). It will return a list containing as many items (the values) as checked boxes. If no check box was checked the list will be empty.

Sample HTML with check boxes:

```
<html><body>
<form method="post" action="process_check.py">
Red<input type="checkbox" name="color" value="red">
Green<input type="checkbox" name="color" value="green">
<input type="submit" value="Submit">
</form>
</body></html>
```

And the corresponding process_check.py script:

```
#!/usr/bin/env python
import cgi
form = cgi.FieldStorage()

# getlist() returns a list containing the
# values of the fields with the given name
colors = form.getlist('color')
```

```
print "Content-Type: text/html\n"
print '<html><body>'
print 'The colors list:', colors
for color in colors:
print '<p>', cgi.escape(color), '</p>'
print '</body></html>'
```

* File Upload;

To upload a file the HTML form must have the enctype attribute set to multipart/form-data. The input tag with the file type will create a "Browse" button.

```
<html><body>
<form enctype="multipart/form-data" action="save_file.py" method="post">
<p>File: <input type="file" name="file"></p>
<p><input type="submit" value="Upload"></p>
</form>
</body></html>
```

The getfirst() and getlist() methods will only return the file(s) content. To also get the filename it is necessary to access a nested FieldStorage instance by its index in the top FieldStorage instance.

```
#!/usr/bin/env python
import cgi
form = cgi.FieldStorage()

# A nested FieldStorage instance holds the file
fileitem = form['file']

# Test if the file was uploaded
if fileitem.filename:
open('files/' + fileitem.filename, 'w').write(fileitem.file.read())
message = 'The file "' + fileitem.filename + '" was uploaded successfully'
else:
message = 'No file was uploaded'

print """\
Content-Type: text/html\n
<html><body>
<p>%s</p>
</body></html>
""" % (message,)
```

The Apache user must have write permission on the directory where the file will be saved.

* Big File Upload

To handle big files without using all the available memory a generator can be used. The generator will return the file in small chunks:

```
#!/usr/bin/env python
import cgi
form = cgi.FieldStorage()

# Generator to buffer file chunks
def fbuffer(f, chunk_size=10000):
while True:
chunk = f.read(chunk_size)
if not chunk: break
yield chunk

# A nested FieldStorage instance holds the file
fileitem = form['file']

# Test if the file was uploaded
if fileitem.filename:
f = open('files/' + fileitem.filename, 'w')

# Read the file in chunks
for chunk in fbuffer(fileitem.file):
f.write(chunk)
f.close()
message = 'The file "' + fileitem.filename + '" was uploaded successfully'
else:
message = 'No file was uploaded'

print """\
Content-Type: text/html\n
<html><body>
<p>%s</p>
</body></html>
""" % (message,)
```

How to use Cookies for Web python ?
HTTP is said to be a stateless protocol. What this means for web programmers is that every time a user loads a page it is the first time for the server. The server can't say whether this user has ever visited that site, if is he in the middle of a buying transaction, if he has already authenticated, etc.

A cookie is a tag that can be placed on the user's computer. Whenever the user loads a page from a site the site's script can send him a cookie. The cookie can contain anything the site needs to identify that user. Then within the next request the user does for a new page there goes back the cookie with all the

pertinent information to be read by the script.

* Set the Cookie;

There are two basic cookie operations. The first is to set the cookie as an HTTP header to be sent to the client. The second is to read the cookie returned from the client also as an HTTP header.

This script will do the first one placing a cookie on the client's browser:

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

# This is the message that contains the cookie
# and will be sent in the HTTP header to the client
print 'Set-Cookie: lastvisit=' + str(time.time());

# To save one line of code
# we replaced the print command with a '\n'
print 'Content-Type: text/html\n'
# End of HTTP header

print '<html><body>'
print 'Server time is', time.asctime(time.localtime())
print '</body></html>'
```

The Set-Cookie header contains the cookie. Save and run this code from your browser and take a look at the cookie saved there. Search for the cookie name, lastvisit, or for the domain name, or the server IP like 10.1.1.1 or 127.0.0.1.

The Cookie Object

The Cookie module can save us a lot of coding and errors and the next pages will use it in all cookie operations.

```
#!/usr/bin/env python
import time, Cookie

# Instantiate a SimpleCookie object
cookie = Cookie.SimpleCookie()

# The SimpleCookie instance is a mapping
cookie['lastvisit'] = str(time.time())

# Output the HTTP message containing the cookie
print cookie
print 'Content-Type: text/html\n'
```

```
print '<html><body>'
print 'Server time is', time.asctime(time.localtime())
print '</body></html>'
```

It does not seem as much for this extremely simple code, but wait until it gets complex and the Cookie module will be your friend.

* Retrieve the Cookie;

The returned cookie will be available as a string in the os.environ dictionary with the key 'HTTP_COOKIE':

```
cookie_string = os.environ.get('HTTP_COOKIE')
```

The load() method of the SimpleCookie object will parse that string rebuilding the object's mapping:

```
cookie.load(cookie_string)
```

Complete code:

```
#!/usr/bin/env python
import Cookie, os, time

cookie = Cookie.SimpleCookie()
cookie['lastvisit'] = str(time.time())

print cookie
print 'Content-Type: text/html\n'

print '<html><body>'
print '<p>Server time is', time.asctime(time.localtime()), '</p>'

# The returned cookie is available in the os.environ dictionary
cookie_string = os.environ.get('HTTP_COOKIE')

# The first time the page is run there will be no cookies
if not cookie_string:
print '<p>First visit or cookies disabled</p>'

else: # Run the page twice to retrieve the cookie
print '<p>The returned cookie string was "' + cookie_string + '"</p>'

# load() parses the cookie string
cookie.load(cookie_string)
# Use the value attribute of the cookie to get it
```

```
lastvisit = float(cookie['lastvisit'].value)

print '<p>Your last visit was at',
print time.asctime(time.localtime(lastvisit)), '</p>'

print '</body></html>'
```

When the client first loads the page there will be no cookie in the client's computer to be returned. The second time the page is requested then the cookie saved in the last run will be sent to the server.

* Morsels

In the previous cookie retrieve program the lastvisit cookie value was retrieved through its value attribute:

```
lastvisit = float(cookie['lastvisit'].value)
```

When a new key is set for a SimpleCookie object a Morsel instance is created:

```
>>> import Cookie
>>> import time
>>>
>>> cookie = Cookie.SimpleCookie()
>>> cookie
<SimpleCookie: >
>>>
>>> cookie['lastvisit'] = str(time.time())
>>> cookie['lastvisit']
<Morsel: lastvisit='1159535133.33'>
>>>
>>> cookie['lastvisit'].value
'1159535133.33'
```

Each cookie, a Morsel instance, can only have a predefined set of keys: expires, path, commnent, domain, max-age, secure and version. Any other key will raise an exception.

```
#!/usr/bin/env python
import Cookie, time

cookie = Cookie.SimpleCookie()

# name/value pair
cookie['lastvisit'] = str(time.time())

# expires in x seconds after the cookie is output.
# the default is to expire when the browser is closed
```

```
cookie['lastvisit']['expires'] = 30 * 24 * 60 * 60


# path in which the cookie is valid.
# if set to '/' it will valid in the whole domain.
# the default is the script's path.
cookie['lastvisit']['path'] = '/cgi-bin'

# the purpose of the cookie to be inspected by the user
cookie['lastvisit']['comment'] = 'holds the last user\'s visit date'

# domain in which the cookie is valid. always stars with a dot.
# to make it available in all subdomains
# specify only the domain like .my_site.com
cookie['lastvisit']['domain'] = '.www.my_site.com'

# discard in x seconds after the cookie is output
# not supported in most browsers
cookie['lastvisit']['max-age'] = 30 * 24 * 60 * 60

# secure has no value. If set directs the user agent to use
# only (unspecified) secure means to contact the origin
# server whenever it sends back this cookie
cookie['lastvisit']['secure'] = ''

# a decimal integer, identifies to which version of
# the state management specification the cookie conforms.
cookie['lastvisit']['version'] = 1

print 'Content-Type: text/html\n'

print '<p>', cookie, '</p>'
for morsel in cookie:
print '<p>', morsel, '=', cookie[morsel].value
print '<div style="margin:-1em auto auto 3em;">'
for key in cookie[morsel]:
print key, '=', cookie[morsel][key], '<br />'
print '</div>
'
```

Notice that print cookie automatically formats the expire date.

How to use Sessions for Web python ?
Sessions are the server side version of cookies. While a cookie persists data (or state) at the client,

sessions do it at the server. Sessions have the advantage that the data do not travel the network thus making it both safer and faster although this not entirely true as shown in the next paragraph

The session state is kept in a file or in a database at the server side. Each session is identified by an id or session id (SID). To make it possible to the client to identify himself to the server the SID must be created by the server and sent to the client and then sent back to the server whenever the client makes a request. There is still data going through the net, the SID.

The server can send the SID to the client in a link's query string or in a hidden form field or as a Set-Cookie header. The SID can be sent back from the client to the server as a query string parameter or in the body of the HTTP message if the post method is used or in a Cookie HTTP header.

If a cookie is not used to store the SID then the session will only last until the browser is closed, or the user goes to another site breaking the POST or query string transmission, or in other words, the session will last only until the user leaves the site.

* Cookie Based SID:

A cookie based session has the advantage that it lasts until the cookie expires and, as only the SID travels the net, it is faster and safer. The disadvantage is that the client must have cookies enabled.

The only particularity with the cookie used to set a session is its value:

```
# The sid will be a hash of the server time
sid = sha.new(repr(time.time())).hexdigest()
```

The hash of the server time makes an unique SID for each session.

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

import sha, time, Cookie, os

cookie = Cookie.SimpleCookie()
string_cookie = os.environ.get('HTTP_COOKIE')

# If new session
if not string_cookie:
# The sid will be a hash of the server time
sid = sha.new(repr(time.time())).hexdigest()
# Set the sid in the cookie
cookie['sid'] = sid
# Will expire in a year
cookie['sid']['expires'] = 12 * 30 * 24 * 60 * 60
# If already existent session
else:
cookie.load(string_cookie)
```

```
sid = cookie['sid'].value

print cookie
print 'Content-Type: text/html\n'
print '<html><body>'

if string_cookie:
print '<p>Already existent session</p>'
else:
print '<p>New session</p>'

print '<p>SID =', sid, '</p>'
print '</body></html>'
```

In every page the existence of the cookie must be tested. If it does not exist then redirect to a login page or just create it if a login or a previous state is not required.

* Query String SID;

Query string based session:

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

import sha, time, cgi, os

sid = cgi.FieldStorage().getfirst('sid')

if sid: # If session exists
message = 'Already existent session'
else: # New session
# The sid will be a hash of the server time
sid = sha.new(repr(time.time())).hexdigest()
message = 'New session'

qs = 'sid=' + sid
print """\
Content-Type: text/html\n
<html><body>
<p>%s</p>
<p>SID = %s</p>
<p><a href="./set_sid_qs.py?sid=%s">reload</a></p>
</body></html>
""" % (message, sid, sid)
```

To mantain a session you will have to append the query string to all the links in the page.

Save this file as set_sid_qs.py and run it two or more times. Try to close the browser and call the page again. The session is gone. The same happens if the page address is typed in the address bar.

* Hidden Field SID;

The hidden form field SID is almost the same as the query string based one, sharing the same problems.

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

import sha, time, cgi, os

sid = cgi.FieldStorage().getfirst('sid')

if sid: # If session exists
message = 'Already existent session'
else: # New session
# The sid will be a hash of the server time
sid = sha.new(repr(time.time())).hexdigest()
message = 'New session'

qs = 'sid=' + sid

print """\
Content-Type: text/html\n
<html><body>
<p>%s</p>
<p>SID = %s</p>
<form method="post">
<input type="hidden" name=sid value="%s">
<input type="submit" value="Submit">
</form>
</body><html>
""" % (message, sid, sid)
```

* The shelve module;

Having a SID is not enough. It is necessary to save the session state in a file or in a database. To save it into a file the shelve module is used. The shelve module opens a file and returns a dictionary like object which is readable and writable as a dictionary.

```
# The shelve module will persist the session data
# and expose it as a dictionary
session = shelve.open('/tmp/.session/sess_' + sid, writeback=True)
```

The SID is part of file name making it a unique file. The apache user must have read and write permission on the file's directory. 660 would be ok.

The values of the dictionary can be any Python object. The keys must be immutable objects.

```
# Save the current time in the session
session['lastvisit'] = repr(time.time())

# Retrieve last visit time from the session
lastvisit = session.get('lastvisit')
```

The dictionary like object must be closed as any other file should be:

```
session.close()
```

* Cookie and Shelve;

A sample of how to make cookies and shelve work together keeping session state at the server side:

```
#!/usr/bin/env python
import sha, time, Cookie, os, shelve

cookie = Cookie.SimpleCookie()
string_cookie = os.environ.get('HTTP_COOKIE')

if not string_cookie:
sid = sha.new(repr(time.time())).hexdigest()
cookie['sid'] = sid
message = 'New session'
else:
cookie.load(string_cookie)
sid = cookie['sid'].value
cookie['sid']['expires'] = 12 * 30 * 24 * 60 * 60

# The shelve module will persist the session data
# and expose it as a dictionary
session = shelve.open('/tmp/.session/sess_' + sid, writeback=True)

# Retrieve last visit time from the session
lastvisit = session.get('lastvisit')
if lastvisit:
message = 'Welcome back. Your last visit was at ' + \
time.asctime(time.gmtime(float(lastvisit)))
# Save the current time in the session
session['lastvisit'] = repr(time.time())
```

```
print """\
%s
Content-Type: text/html\n
<html><body>
<p>%s</p>
<p>SID = %s</p>
</body></html>
""" % (cookie, message, sid)

session.close()
```

It first checks if there is a cookie already set. If not it creates a SID and attributes it to the cookie value. An expiration time of one year is established.

The lastvisit data is what is maintained in the session.