```
print obj
print obj1, obj2, obj3
print "My name is %s" % name
```

Notes:

- To print multiple items, separate them with commas. The print statement inserts a blank between objects.
- The print statement automatically appends a newline to output. To print without a newline, add a comma after the last object, or use "sys.stdout", for example:
  ```
  print 'Output with no newline',
  ```
  which will append a blank, or:
  ```
  import sys
  sys.stdout.write("Some output")
  ```
- To re-define the destination of output from the print statement, replace sys.stdout with an instance of a class that supports the write method. For example:
  ```
  import sys

  class Writer:
      def __init__(self, filename):
          self.filename = filename
      def write(self, msg):
          f = file(self.filename, 'a')
          f.write(msg)
          f.close()

  sys.stdout = Writer('tmp.log')
  print 'Log message #1'
  print 'Log message #2'
  print 'Log message #3'
  ```

More information on the print statement is at The print statement -- http://docs.python.org/reference/simple_stmts.html#the-print-statement.

1.5.2 Assignment statement

The assignment operator is =.

Here are some of the things you can assign a value to:

- A name (variable)
- An item (position) in a list. Example:
  ```
  >>> a = [11, 22, 33]
  >>> a
  [11, 22, 33]
  >>> a[1] = 99
  >>> a
  [11, 99, 33]
  ```
- A key in a dictionary. Example:
  ```
  >>> names = {}
  >>> names['albert'] = 25
  >>> names
  ```

```
{'albert': 25}
```

- A slice in a list. Example:

```
>>> a = [11, 22, 33, 44, 55, 66, 77, ]
>>> a
[11, 22, 33, 44, 55, 66, 77]
>>> a[1:3] = [999, 888, 777, 666]
>>> a
[11, 999, 888, 777, 666, 44, 55, 66, 77]
```

- A tuple or list. Assignment to a tuple or list performs unpacking. Example:

```
>>> values = 111, 222, 333
>>> values
(111, 222, 333)
>>> a, b, c = values
>>> a
111
>>> b
222
>>> c
333
```

  Unpacking suggests a convenient idiom for returning and capturing a multiple arguments from a function. Example:

```
>>> def multiplier(n):
...     return n, n * 2, n * 3
...
>>>
>>> x, y, z = multiplier(4)
>>> x
4
>>> y
8
>>> z
12
```

  If a function needs to return a variable number of values, then unpacking will *not do*. But, you can still return multiple values by returning a container of some kind (for example, a tuple, a list, a dictionary, a set, etc.).

- An attribute. Example:

```
>>> class A(object):
...   pass
...
>>> c = A()
>>>
>>> a = A()
>>> a.size = 33
>>> print a.size
33
>>> a.__dict__
{'size': 33}
```

1.5.3 import statement

Things to know about the `import` statement:

- The import statement makes a module and its contents available for use.
- The import statement *evaluates* the code in a module, but only the first time that any given module is imported in an application.
- All modules in an application that import a given module share a single copy of that module. Example: if modules A and B both import module C, then A and B share a single copy of C.

Here are several forms of the import statement:

- Import a module. Refer to an attribute in that module:
```
import test
print test.x
```
- Import a specific attribute from a module:
```
from test import x
from othertest import y, z
print x, y, z
```
- Import all the attributes in a module:
```
from test import *
print x
print y
```
Recommendation: Use this form sparingly. Using `from mod import *` makes it difficult to track down variables and, thus, to debug your code
- Import a module and rename it. Import an attribute from a module and rename it:
```
import test as theTest
from test import x as theValue
print theTest.x
print theValue
```

A few comments about import:

- The import statement also evaluates the code in the imported module.
- But, the code in a module is only evaluated the first time it is imported in a program. So, for example, if a module mymodule.py is imported from two other modules in a program, the statements in mymodule will be evaluated only the first time it is imported.
- If you need even more variety that the import statement offers, see the imp module. Documentation is at imp - Access the import internals -- http://docs.python.org/library/imp.html#module-imp. Also see the `__import__()` built-in function, which you can read about here: Built-in Functions: __import() -- http://docs.python.org/library/functions.html#__import__.

More information on import is at Language Reference: The import statement -- http://docs.python.org/reference/simple_stmts.html#the-import-statement.

1.5.4 assert statement

Use the assert statement to place error checking statements in your code. Here is an example: