Using Command-Line Arguments

For Introduction to Programming Using Python
By Y. Daniel Liang

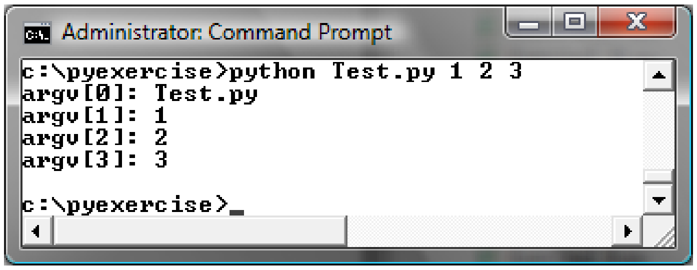
You can pass command-line arguments in a Java/C++ program. You can do the same thing in Python. The arguments passed from a command line will be stored in sys.argv, which is a list of strings. Listing 1 gives a simple test program that displays all the arguments passed from the command line:

Listing 1 Test.py

```
import sys

for i in range(0, len(sys.argv)):
    print("argv[" + str(i) + "]: " + sys.argv[i])
```

As shown in Figure 1, the arguments are passed from the command line separated by space. The Python source code filename is treated as the first argument in the command line.



```
c:\pyexercise>python Test.py 1 2 3
argv[0]: Test.py
argv[1]: 1
argv[2]: 2
argv[3]: 3

c:\pyexercise>_
```

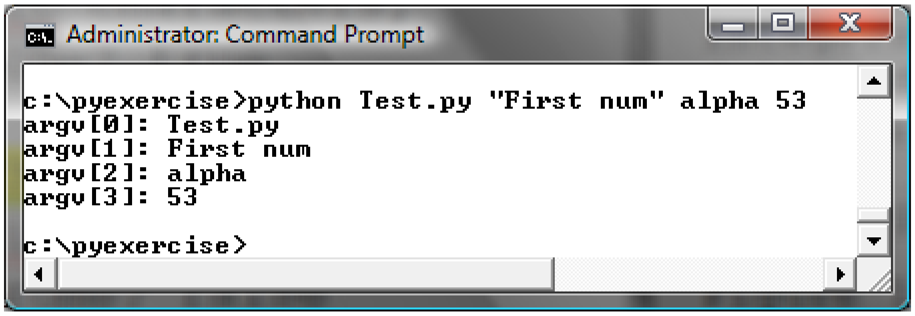
Figure 1

The arguments are passed from the command line separated by spaces.

The arguments must be strings, but they don't have to appear in quotes on the command line. The strings are separated by a space. A string that contains a space must be enclosed in double quotes. Consider the following command line:

```
python Test.py "First num" alpha 53
```

It starts the program with four strings: "Test.py", "First num" and alpha, and 53, a numeric string, as shown in Figure 2. Note that 53 is actually treated as a string. You can use "53" instead of 53 in the command line.



```
c:\pyexercise>python Test.py "First num" alpha 53
argv[0]: Test.py
argv[1]: First num
argv[2]: alpha
argv[3]: 53

c:\pyexercise>
```

Figure 2

The argument must be enclosed in quotes if it contains spaces.

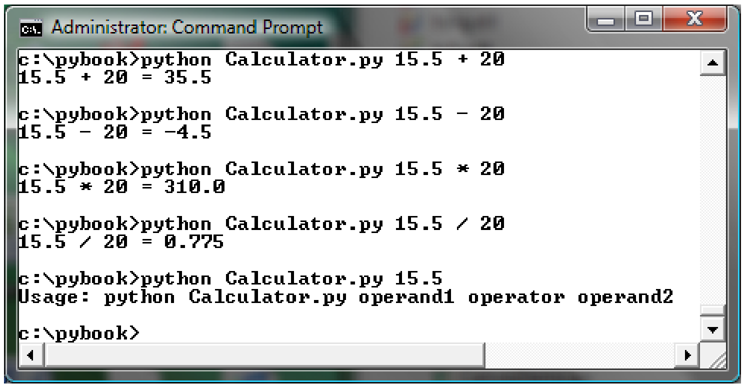
Listing 2 presents a program that performs binary operations on integers. The program receives three arguments: an integer followed by an operator and another integer. For example, to add two integers, use this command:

```
python Calculator.py 1 + 2
```

The program will display the following output:

```
1 + 2 = 3
```

Figure 3 shows sample runs of the program.



```
c:\pybook>python Calculator.py 15.5 + 20
15.5 + 20 = 35.5

c:\pybook>python Calculator.py 15.5 - 20
15.5 - 20 = -4.5

c:\pybook>python Calculator.py 15.5 * 20
15.5 * 20 = 310.0

c:\pybook>python Calculator.py 15.5 / 20
15.5 / 20 = 0.775

c:\pybook>python Calculator.py 15.5
Usage: python Calculator.py operand1 operator operand2

c:\pybook>
```

Figure 3

The program takes three arguments (operand1 operator operand2) from the command line and displays the expression and the result of the arithmetic operation.

Here are the steps in the program:

1. Check argv to determine whether three arguments have been provided in the command line. If not, terminate the program using sys.exit().

2. Perform a binary arithmetic operation on the operands argv[1] and argv[3] using the operator specified in argv[2].

Listing 2 Calculator.py

```
import sys

# Check number of strings passed
if len(sys.argv) != 4:
    print("Usage: python Calculator.py operand1 operator 
operand2")
    sys.exit()

# Determine the operator
if sys.argv[2][0] == '+':
    result = eval(sys.argv[1]) + eval(sys.argv[3])
elif sys.argv[2][0] == '-':
    result = eval(sys.argv[1]) - eval(sys.argv[3])
elif sys.argv[2][0] == '*':
    result = eval(sys.argv[1]) * eval(sys.argv[3])
elif sys.argv[2][0] == '/':
    result = eval(sys.argv[1]) / eval(sys.argv[3])

# Display result
print(sys.argv[1] + ' ' + sys.argv[2] + ' ' + sys.argv[3] +
    " = " + str(result))
```