Simple Programs with Python

1 Programs

A program consists of data definitions and instructions that manipulate the data. These are:

- *Data definitions*, which indicate the data to be manipulated by the instructions.

- A *sequence of instructions*, which perform the computations on the data in order to produce the desired results.

2 Data Definitions

The data in a program consists of one or more data items. These are manipulated or transformed by the computations (computer operations). In Python, each data definition is specified by assigning a value to a variable and has:

- A reference, which is a *variable* with a unique *name* to refer to the data item

- A *value* associated with it

The name of the reference (variable) to a data item is an *identifier* and is defined by the programmer; it must be different from any *keyword* in the programming language.

2.1 Data Objects

In Python, the data items are known as *data objects* and every variable references a data object. If a the value associated with a data object does not change, then the the data object is said to be *immutable*, otherwise it is *mutable*.

The three most important attributes of a data object are:

- the identity, which is the location (address) of the data object in the computer memory;

- the type, which defines the operations are allowed for the data object;

- the value, which can be changed (mutable) or not (immutable).
2.2 Variables

As mentioned previously, a variable is a reference to a data object and the name of the variable is used in a program for uniquely identifying the variable and are known as an identifier. The special text words or symbols that indicate essential parts of a programming language are known as keywords. These are reserved words and cannot be used for any other purpose.

A problem that calculates the area of a triangle uses four variables, example of the names for these variables are: \textit{a, b, c}, and \textit{area}.

2.3 Using Data Objects and Variables

In the following listing of Python commands, the first three commands include three assignments to variables \textit{x}, \textit{y}, and \textit{z}. The fourth Python command uses the Python function \texttt{id()} to get the address of the data object referenced by variable \textit{x} and this address is 19290088. Note that the address of the referenced object with variables \textit{y} and \textit{z} is the same, because these two variables refer to the same data object. After changing the value of variable \textit{y}, the reference is different because now variable \textit{y} refers to a different data object. Note that the `#` symbol is used to include a comment on a source line and has no effect on the instruction.

```python
>>> x = 5.33
>>> y = 6
>>> z = y  # these now refer to the same data object
>>> id(x)  # get identity of data object
19290088
>>> id(y)
19257084
>>> id(z)
19257084
>>> y = y + 1
>>> id(y)
19257072
>>> id(z)
19257084
>>> z = z + 1
>>> id(z)
19257072
>>> type(x)
<type 'float'>
>>> type(y)
<type 'int'>
```
2.4 Basic Data Types

The fundamental data types are classified into the three categories:

- Numeric
- Text
- Boolean

The numeric types are further divided into two basic types, integer, float. Values of integer type are those that are countable to a finite value, for example, age, number of parts, number of students enrolled in a course, and so on. Values of type float have a decimal point; for example, cost of a part, the height of a tower, current temperature in a boiler, a time interval. These values cannot be expressed as integers.

In the python commands of the previous example, the python function type() is used to get the type variable \( x \) and of variable \( y \). Note that the type of variable \( x \) is float and the type of variable \( y \) is int.

Text data items are of type string and consist of a sequence of characters. The values for this types of data items are text values. An example of a string as the text value: 'Welcome!'.

The third data type is used for data objects whose values can take any of two truth-values (True or False); these data objects are of type bool.

3 Simple Python Programs

A very simple program consists of data definitions and a sequence of instructions. The script mode is normally used for writing Python programs. Instructions are written into a text file using an appropriate text editor such as gedit on Linux and Notepad++ on Windows. The text file with the source code is known as a script and has a .py extension.

An instruction performs a specific manipulation or computation on the data, it is written as a language statement in the program.

3.1 The Assignment Statement

As discussed in previous examples, the assignment statement is the most fundamental statement (high-level instruction); its general form is:

\[
\text{variable.name} = \text{expression}
\]
The assignment operator is denoted by the = symbol and on the left side of this operator a variable name must always be written. On the right side of the assignment operator, an expression is written. The Python interpreter evaluates the expression on the right-hand side of the assignment and the result is assigned to the variable on the left-hand side of the assignment operator.

In the following example, the first Python statement is a simple assignment that assigns the value 34.5 to variable $x$. The second assignment statement is a slightly more complex assignment that performs an addition of the value of variable $x$ and the constant 11.38. The result of the addition is assigned to variable $y$.

$$
x = 34.5 \\
y = x + 11.38
$$

### 3.2 Basic Input and Output Instructions

Input and output statements are used to read (input) data values from the input device (e.g., the keyboard) and write (output) data values to an output device (the computer screen).

#### 3.2.1 Output Statement

In Python, the `print` statement is used for the output of variables and text strings. This output statement writes the value of one or more variables to the output device. The variables do not change their values. The general form of the output statement in Python is:

```python
print ⟨ data_list ⟩
```

For example, in the following Python statements, the line will simply display the value of variable $y$ on the screen. The second output displays the string literal “value of x= ”, followed by the value of variable $x$.

```python
print y
print "value of x= ", x
```

Note that the `print` instruction is a statement in Python 2, it is a function in Python 3 and is written as:

```python
print (y)
print ("value of x= ", x)
```
3.2.2 Input Statements

The *input* statement reads a value of a variable from the input device (e.g., the keyboard). This statement is written with the function *input*, for of a single data value and assign to a variable. The following two lines of pseudo-code include the general form of the input statement and an example that uses the *read* statement to read a value of variable *y*.

\[
\langle \text{var\_name} \rangle = \text{input} (\langle \text{string\_lit} \rangle)
\]

The following example displays the string literal “Enter value of \( y \): ” and reads the value of variable \( y \).

\[
y = \text{input} (“Enter value of } y: ”)
\]

3.3 Example Scripts with Input/Output

The following script computes 75% of the value of variable \( y \). The name of the script is *prog01.py*.

\[
y = 34.5 \\
\text{print (} y \text{)} \\
y = y \times 0.75 \\
\text{print(} y \text{)}
\]

At the Linux prompt, the Python interpreter is run with script *prog01.py*. The interpreter will execute every Python command in sequence (one after the other) and the results are displayed on the screen.

\[
$\text{python prog01.py} \\
34.5 \\
25.875
\]

The next script has more output but it carries out the same computations. The first line of the script is only a comment, it starts with the pound (\#) symbol. The name of this second script is *prog02.py*.

\[
# \text{This script computes 75\% of the value of } y \\
y = 34.5 \\
\text{print } “\text{Initial value of } y: “, y \\
y = y \times 0.75 \\
\text{print } “\text{Final value of } y: “, y
\]

© 2016 J. M. Garrido
This script is started by invoking the python interpreter with the name of the script, prog02.py.

```
$ python prog02.py
Initial value of y:  34.5
Final value of y:  25.875
```

The third script performs the same computations as the first two scripts. The main difference is that it inputs the value of variable \( y \); in other words, the user of the program will enter the value of \( y \).

```
# This script computes 75% of the value of y
y = input("Enter initial value of y: ")
print "Initial value of y: ", y
y = y * 0.75
print "Final value of y: ", y
```

This script is started by invoking the python interpreter with the name of the script, prog03.py.

```
$ python prog03.py
Enter initial value of y: 34.5
Initial value of y: 34.5
Final value of y: 25.875
```

4 A Simple Problem: Temperature Conversion

This section revisits and implements in Python the temperature conversion problem, which was discussed in the previous chapter. The solution and implementation is derived by following a basic sequence of steps.

The problem: given the value of the temperature in degrees Celsius, compute the corresponding value in degrees Fahrenheit and show this result.

4.1 Mathematical Model

The mathematical representation of the solution to the problem, the formula expressing a temperature measurement \( F \) in Fahrenheit in terms of the temperature measurement \( C \) in Celsius is:

\[
F = \frac{9}{5} C + 32
\]
The solution to the problem is the mathematical expression for the conversion of a temperature measurement in Celsius to the corresponding value in Fahrenheit. The mathematical formula expressing the conversion assigns a value to the desired temperature in the variable $F$, the dependent variable. The values of the variable $C$ can change arbitrarily because it is the independent variable. The mathematical model uses real numbers to represent the temperature readings in various temperature units.

### 4.2 Computational Model

The computational model is derived by implementing the mathematical model in a program using the Python programming language. This model is developed using a Terminal window Linux. In a similar manner to the previous examples, the computational model is developed by writing a Python program as a script using the *gedit* text editor, then executing the Python interpreter with the script.

The Python program is very simple and Listing 4.1 shows the complete source code.

Listing 4.1: Temperature conversion program.

```python
1 ""
2 Program : tconvctof.py
3 Author : Jose M Garrido
4 Date : 5-12-2014
5 Description : Read value of temperature Celsius from
6 console, convert to degrees Fahrenheit, and display
7 value of this new temperature value on the output
8 console */
9 ""
10
11 C = input("Enter value of temp in Celsius: ")
12 F = C * (9.0/5.0) + 32.0 # temperature in Fahrenheit
13 print "Value of temperature in Celsius: ", C
14 print "Temperature in Fahrenheit: ", F
```

Lines 1–9 is part of a multi-line comment. The computation of the temperature in Fahrenheit is performed in line 12 using an assignment statement. The following listing shows the interpretation of the commands in the script *tconvctof.py* by executing the Python interpreter.

```
$ python tconvctof.py
Enter value of temp in Celsius: 25.0
```

© 2016 J. M. Garrido
Value of temp in Celsius: 25.0
Temperature in Fahrenheit: 77.0

This procedure can be repeated several times to compute the Fahrenheit temperature starting with a given value of 10.0 for the temperature in Celsius and then repeating in increments of 5.0 degrees Celsius. The last computation is for a given value of 45.0 degrees Celsius.

Table ?? shows the values of temperature in Celsius from 5.0 to 45.0 used to compute the corresponding temperature in Fahrenheit. This is a short set of results of the original problem. Figure ?? shows a plot of the values of temperature computed.

Table 1: Celsius and Fahrenheit temperatures.

<table>
<thead>
<tr>
<th>Celsius</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fahrenheit</td>
<td>41</td>
<td>50</td>
<td>59</td>
<td>68</td>
<td>77</td>
<td>86</td>
<td>95</td>
<td>104</td>
<td>113</td>
</tr>
</tbody>
</table>

Figure 1: Plot of the values of temperature Celsius and Fahrenheit.
5 Distance Between Two Points

5.1 Problem statement

The following problem requires computing the distance between two points in a Cartesian plane. A program is to be developed that computes this distance, given the values of the coordinates of the two points.

5.2 Analysis of the Problem

A Cartesian plane consists of two directed lines that perpendicularly intersect their respective zero points. The horizontal directed line is called the $x$-axis and the vertical directed line is called the $y$-axis. The point of intersection of the $x$-axis and the $y$-axis is known as the origin and is denoted by the letter O.

Figure ?? shows a Cartesian plane with two points, $P_1$ and $P_2$. Point $P_1$ is defined by two coordinate values $(x_1, y_1)$ and point $P_2$ is defined by the coordinate values $(x_2, y_2)$.

5.3 Design of the Solution

The horizontal distance between the two points, $\Delta x$, is computed by the difference $x_2 - x_1$. Similarly, the vertical distance between the two points is denoted by $\Delta y$ and is computed by the difference $y_2 - y_1$. The distance, $d$, between two points $P_1$ and $P_2$ in a Cartesian plane is calculated with the following mathematical expression:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figure 2: Horizontal and vertical distances between two points.
\[ d = \sqrt{\Delta x^2 + \Delta y^2} \]

A detailed design in an algorithm follows:

1. Read the values of the coordinates for point \( P_1 \) from input device (keyboard)
2. Read the values of the coordinates for point \( P_2 \) from the input device
3. Compute the horizontal distance, \( \Delta x \), between the two points:
   \[ \Delta x = x_2 - x_1 \]
4. Compute the vertical distance, \( \Delta y \), between the two points:
   \[ \Delta y = y_2 - y_1 \]
5. Compute the distance, \( d \), between the two points:
   \[ d = \sqrt{\Delta x^2 + \Delta y^2} \]
6. Display the value of the distance between the two points, on the output device (video screen)

### 5.4 Implementation

This phase implements the design by coding a program in Python, running the Python interpreter with the corresponding script, and testing the program. Listing 5.1a shows the source program, which is stored in file `distpoints.py`.

```
1 ""
2 # Program : distpts.py
3 # Author : Jose M Garrido, January, 20, 2014.
4 # Description : This program computes the distance
5 # between two points in a Cartesian plane.
6 ""
7 import math
8 x1 = input("Enter value of x-coordinate of P1: ")
9 y1 = input("Enter value of y-coordinate of P1: ")
10 print "Coordinates of P1: ", x1, y1
```

© 2016 J. M. Garrido
x2 = input("Enter value of x-coordinate of P2: ")
y2 = input("Enter value of y-coordinate of P2: ")
print "Coordinates of P2: ", x2, y2

dx = x2 - x1

# compute vertical distance between points
dy = y2 - y1

# compute the distance between the points
d = math.sqrt(dx ** 2 + dy ** 2)

# display result
print "Distance between P1 and P2: ", d

The following listing shows the Python interpretation of the script distpts.py with the input values shown.

```
$ python distpts.py
Enter value of x-coordinate of P1: 2.25
Enter value of y-coordinate of P1: 1.5
Coordinates of P1: 2.25 1.5
Enter value of x-coordinate of P2: 1.3
Enter value of y-coordinate of P2: 0.45
Coordinates of P2: 1.3 0.45
Horizontal and vertical distances: -0.95 -1.05
Distance between P1 and P2: 1.41598022585
```

6 General Structure of a Python Program

A typical program in the Python language has the general structure as shown in Figure ???. It consists of several parts:

1. The import commands are optional but they are present in almost all Python programs. Each of these uses the import statement and allows the program access to the definitions and code in the specified Python module.
2. Global data, which may consist of assignments of values to variables, in a
similar manner as described previously. These are global data because they
can be used by all functions in the program.

3. Definition of functions. This is an optional component of a Python program
but it is almost always present. When present, one or more functions are
defined in this part of the program. In Figure ??, a function is defined with
name $A$.

4. Definition of classes. This is another optional component in a Python program.
A class definition allows the program to create objects of that class. When
present, one or more classes are defined in this part of the program.

5. The instructions are Python statements that invoke or call the functions in the
program and/or in the imported modules. These instructions can also create
and manipulate objects using the class definitions in the program and/or in
the imported modules.

Function definitions can be programmer-defined functions that are invoked (or
called) in the program. The other functions that can be called are the built-in func-

Figure 3: General structure of a Python program.
tions provided by standard Python interpreter libraries or by other Python modules. A library is a collection of related function definitions and/or class definitions that may also include data.

A function starts executing when it is called by another function or by an instruction in the program. Before a function can be called in a Python program, a function definition is required. Once a function is defined, it can be called or invoked by any other function or by an instruction in the program.

7 Simple Functions

A Python program is normally decomposed into modules, and these are divided into classes and functions. A function carries out a specific task in a program.

As mentioned previously, data in a function is known only to that function—the scope of the data is local to the function. The local data in a function has a limited lifetime; it only exists during execution of the function.

7.1 Function Definitions

A simple Python program consists of functions and instructions that call or invoke the various functions. Figure 7.1 illustrates the general structure of a function in the Python language.

![Figure 4: Structure of a python function.](image)

In the source code, the general syntactical form of a function in the Python programming language is written as follows:

```python
def <function_name> ( [parameters] ) :
    [ local declarations ]
    [ executable language statements ]
```

© 2016 J. M. Garrido
The relevant internal documentation of the function definition is described in one or more lines of comments, which begins with the characters ("""") and ends with ("""").

The local data definitions in the function are optional. The instructions implement the body of the function. The following Python source code shows a simple function for displaying a text message on the screen.

```python
def show_message () :
    ""
    This function displays a message on the screen.
    ""
    print("Computing data")
```

This is a very simple function and its only purpose is to display a text message on the screen. This function does not declare parameters and the type of this function is *void* to indicate that this function does not return a value.

### 7.2 Function Calls

The name of the function is used when calling or invoking the function by some other function. The function that calls another function is known as the calling function; the second function is known as the called function. When a function calls or invokes another function, the flow of control is altered and the second function starts execution immediately.

![Figure 5: A function calling another function](image)

When the called function completes execution, the flow of control is transferred back (returned) to the calling function and it continues execution from the point after it called the second function.
The figure illustrates an instruction calling function *functb*. After completing its execution, function *functb* returns the flow of control to the instruction that performed the call.

An example of this kind of function call is the call to function *show_message*, discussed previously. In Python, the statement that calls a simple function uses the function name and an empty parentheses pair. For example, the call to function *show_message* is written as:

```
show_message()
```

Listing 2 shows a Python program that defines function *show_message* then calls the function. This program is stored in file *shmessp.py*.

Listing 2: Python program that defines and calls a function.

```python
# Program : shmessp.py
# Author : Jose M Garrido, May 28 2014.
# Description : Define and call a simple function.

def show_message():
    
    This function displays a message
    on the screen
    
    print "Computing results ..... 

y = input("Enter a number: ")
sqw = y * y
show_message()
print "square of the number is: ", sqw
```

The function is defines in lines 6–11 and the function is called in line 15. The following listing shows the Python interpretation of the script *shmessp.py*.

```
$ python shmessp.py
Enter a number: 12
Computing results ..... 
square of the number is: 144
```