Fundamentals: Expressions and Assignment

A typical Python program is made up of one or more statements, which are executed in a Python console (also known as a shell) for their side effects—e.g., to input some values, write some values to a file, display values to the user, etc.

There are many kinds of Python statements. But the most fundamental is assignment. An assignment statement has the following form, where var denotes a variable and exp denotes an expression:

\[
\text{var} = \text{exp}
\]

(Spoken as: ‘var is assigned exp’ or ‘var gets exp’.)

To execute an assignment, the console first evaluates exp and creates an object\(^1\) to represent the value of exp. The console then associates the name var with this object.\(^2\) After executing the assignment, var can be used in other expressions to stand for this value.

Every object has a type. The type determines how the console represents the object in memory, how the object can be used in a program, and how the console displays the object to the user. You can learn an object’s type using Python’s type function—for example, evaluating the expression type(1.5) returns the type float.

This exercise explores these concepts in more detail. It also requires you to experiment in the console and load and run a Python program.

Part (a): (Expressions and types) A literal expression stands for a fixed value. The table below illustrates four kinds of literals, their types, and the values that they stand for.

<table>
<thead>
<tr>
<th>Literal</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>int</td>
<td>The integers 18 and -10,000</td>
</tr>
<tr>
<td>-10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>float</td>
<td>The decimal numbers 3.5 and -0.0003</td>
</tr>
<tr>
<td>-3e-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'THIS IS AN EX-PARROT!!'</td>
<td>str</td>
<td>The sequence of 22 characters between the single quotes (including spaces and punctuation)</td>
</tr>
<tr>
<td>&quot;No, 'e's uh,... resting.&quot;</td>
<td>str</td>
<td>The sequence of 24 characters between the double quotes (including spaces and punctuation)</td>
</tr>
<tr>
<td>&quot;&quot;&quot;I never! Yes you did!&quot;&quot;&quot;&quot;</td>
<td>str</td>
<td>The sequence of 21 characters between the triple quotes (including spaces, punctuation, and a new-line character (\n))</td>
</tr>
<tr>
<td>True</td>
<td>bool</td>
<td>The Boolean values true and false</td>
</tr>
<tr>
<td>False</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

\(^1\) An object is a representation in computer memory of a real-world value, such as an integer or a real number.

\(^2\) More precisely, the console associates var with an available location in memory and stores the object at that location.
In an expression, a variable stands for the value that was last assigned to it. For example, executing \( x = 5 \) associates the variable \( x \) with the value 5. After this assignment, evaluating the expression \( x \) returns a 5 and evaluating the expression \( \text{type}(x) \) returns \text{int}.

More complex expressions are formed by applying operators and functions to other expressions, called \textit{arguments}. For example, after the assignment \( x = 5 \), evaluating the expression \( x + 2 \) returns 7 – the value produced by adding the values of the arguments, \( x \) and 2.

With a partner, bring up Spyder. \textit{Press the Variable explorer tab in the top right pane.} Follow the instructions below discussing answers to the questions with your partner; put a pink “tent” on your monitor if you are uncertain of an answer or have other questions.

1. In the console, type each literal below followed by the \texttt{enter}-key, and note what the console displays for each:
   a. \texttt{1500.0}
   b. \texttt{1.5e3}
   c. \texttt{1.5e20}
   d. \texttt{15000000000000000000.0} (there are 19 zeros before the decimal point)
   e. \texttt{15000000000000000008.0} (there are 18 zeros before the 8)
   f. \texttt{150000000000000000000000000}
   g. \texttt{1500000000000000000008}

   Q: What can you conclude from the results produced by (d) and (e)?

   Q: What can you conclude from the results produced by (f) and (g)?

   Q: How do you think the console determines whether to display a \texttt{float} value using the exponent notation or as a decimal number (a series of digits containing a period)?

2. In the console, enter the identifier \( x \) (i.e., type \( x \) followed by the \texttt{enter}-key)
   Q: Why does the console display an error?

3. Enter the assignment \( x = 1.5e3 \).
   Q: Why doesn’t the console display anything?

   Q: What side effect occurred and how does Spyder show this side effect?

   Q: What does the console display if you now enter \( x \)?

   Q: What does the console display if you now enter \( \text{type}(x) \)?
4. Follow the instructions in the handout “Downloading Python Scripts (programs) from the CTL website” to create a CTL folder that contains a Week01 subfolder, and then download someAssignments.py into the subfolder. Next, follow the instructions in the handout “Opening and Working on Python Scripts (programs) in Spyder” to open someAssignments.py in Spyder. (Do not run it yet.)

The Spyder Editor (left pane) sheet should now contain a 3-line program containing 6 assignments.

Q: What is displayed if you now enter y into the console? Why do you think this is?

Q: Based on that experiment, what do you think the console will display if you enter the expression a * b into the console?

5. Press the big green arrow in the toolbar to run (execute) the program.

Q: How does Spyder show the effect of running the program?

Q: What is displayed if you now enter the following expressions into the console:
   a. a * b
   b. y * b
   c. s + t
   d. s + x

6. To correctly form expressions that use functions and operators, you need to know the Python typing rules. Typing rules indicate what types of arguments an operator can be applied to and the type of the value it returns. In this step, you will experiment in the console to discover some typing rules for a few useful Python operators.

Some examples to get you started: In 5 (a) above, since a is an int and b is an int and since executing a * b returns an int, the result of (a) suggests that multiplying two int values produces an int value. This is expressed as the typing rule: int * int -> int

Similarly, the result of 5 (b) suggests that multiplying a float by an int produces a float. This is expressed as the typing rule: float * int -> float

The result in 5 (c) suggests the typing rule: str + str -> str

In contrast, evaluating the expression in (d) produces an error. This is expressed by the rule: str + float -> ERROR
Beside each expression below, write the typing rule suggested by entering the expression into the console:

\[ a - x \]

\[ b ** a \]

\[ b ** y \]

\[ b // a \]

\[ b / a \]

\[ b % a \]

\[ y // a \]

\[ y % a \]

\[ s * t \]

7. To correctly form expressions that contain many operators, you need to know the **precedence** of operators and how operators **associate**. The following table shows the precedence of the Python operators that we've used so far, from highest (power) to lowest (addition and subtraction):

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ x**y ]</td>
<td>power</td>
</tr>
<tr>
<td>[ -x, +x ]</td>
<td>negative and positive</td>
</tr>
<tr>
<td>[ x*y, x/y, x//y, x%y ]</td>
<td>multiplication, division, quotient, and remainder</td>
</tr>
<tr>
<td>[ x+y, x-y ]</td>
<td>addition and subtraction</td>
</tr>
</tbody>
</table>

Most Python operators at the same precedence level associate from left to right. The one exception is the power operator, ***, which associates from right to left.
For example:

\[
1 - 3 \times 2 + 7 = 1 - 6 + 7 = -5 + 7 = 2
\]

\[
2^{2^2} = 2^{9} = 512
\]

Assume the following assignments were previously entered in the console: \( A = 2 \) and \( B = 3 \)

Fill in the table with the value of each expression (do these on paper!).

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A + B ) // 2</td>
<td></td>
</tr>
<tr>
<td>(- B - A + 2 \times A)</td>
<td></td>
</tr>
<tr>
<td>( B \times A / A \times B)</td>
<td></td>
</tr>
<tr>
<td>(10 \times B \times A \times -0.5)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>((A + B) ) // 2</td>
<td></td>
</tr>
<tr>
<td>((- (B - A) + 2) \times A)</td>
<td></td>
</tr>
<tr>
<td>(B \times A / (A \times B))</td>
<td></td>
</tr>
<tr>
<td>((10 \times B) \times (A \times -0.5))</td>
<td></td>
</tr>
</tbody>
</table>

After filling in the table, download the file precedence.py from this week’s Artifacts section to your CTL > Week01 folder as a Py File (.py). Open precedence.py in Spyder and execute it to check your answers. (Refer to the handouts on downloading and opening Python scripts, if needed.)

8. For each type, Python provides a type constructor, which is a function for creating values of that type; the type constructor name is the same as the type name. Test this out in the console by executing expressions such as the following.

\(\text{int}(2.9999e2)\)

\(\text{float}(3 \times 75)\)

\(\text{float}(2.9999e2)\)

\(\text{str}(3)\)

\(\text{str}(2.9999e2)\)

\(\text{str}(3/10**15)\)