1.2 Using the Python Console

For the remainder of this chapter, we'll use the Python console to illustrate some foundational concepts in Python. We strongly encourage you to follow along by typing the Python code that we present into the Python console on your own computer, to begin getting used to the rhythm of using the console to experiment with Python.

Entering code in the Python console

To get you started, here is a brief introduction to the Python console. When we first start it, we see the following:

```
>>>
```

The text `>>>` is called the console **prompt**: the Python console is "prompting" us to type in some Python code to execute. If we type in a simple arithmetic expression,

```
>>> 4 + 5
```

and press Enter, we see the following output:

```
>>> 4 + 5
9
```

The interpreter took our bit of code, `4 + 5`, and calculated its value, `9`, and displayed that value to us. Let's formalize this idea with some terminology that we'll use throughout the course.

Expressions, literals, and operators

A piece of Python code that produces a value is called an **expression**, and the act of calculating the value of an expression is called **evaluating** the expression.[1]

[1] Evaluating an expression is something that you, a human, can also do! For example, if you see `4 + 5` and think "Oh, that's `9`", you just evaluated the expression.

The expression `4 + 5` looks simple enough, but technically it is formed from two smaller expressions—the numbers `4` and `5` themselves. We can ask Python to evaluate each of these expressions, though the result is not very interesting.

```
>>> 4
4
>>> 5
5
```

A Python **literal** is the simplest kind of Python expression: it is a piece of code that represents the exact value as written. For example, `4` is an integer literal representing the number 4.

What about the `*` symbol? We know from our study of mathematics that `*` means multiplication, but what is it, technically, in the Python programming language? A literal, expression, or something else? Let's try to evaluate it in the Python console (typing in `*` and pressing Enter):[2]

[2] This example will make *much* more sense if you're following along on the Python console on your computer!

```
>>> *
Traceback (most recent call last):
  ... lots of text omitted ...
  File "<input>", line 1
    *
     ^
SyntaxError: invalid syntax
```

This is our very first Python **error message**, telling us that something went wrong when the Python interpreter tried to execute our code. In this case, we received a **syntax error** (which the interpreter calls `SyntaxError`), which means that our code was not properly structured. This should make sense: just writing `*` by itself is not a valid multiplication operation, as `*` expects both an expression on its left and right sides to multiply.

So `*` by itself is not a valid Python expression. Instead, `*` is an example of an **operator**, which is a symbol in a programming language that represents a specific computation to perform. In the Python programming language, operators are not expressions, but are used to build up larger expressions from smaller ones, just as we use `*` to write the expression `4 * 5`. As we'll see later in this chapter, most operators in Python are **binary** operators, meaning they are used to combine two expressions together. Addition (`+`), subtraction (`-`), and multiplication (`*`) are all examples of binary operators.

In summary,[3] the *expression* `4 * 5` consists of two smaller expressions, the *literals* `4` and `5`, joined together with the arithmetic *operator* `*`, representing multiplication. We'll devote the rest of this chapter to exploring the different kinds of data we can represent in Python beyond just numbers. We'll see how to write literals for different data types and what computations we can perform on them using different operators.

[3] We're deliberately using all of the terminology we introduced in this section. Pause at each italicized word and make sure you understand it (or go back up to review its definition).