Let's Code Python

Programming Fundamentals

Module 01

Duration: 4 days (23 & 30 April, 7 & 14 May 2022)

Time: 9am to 12pm

Delivery Method: Virtual Instructor Lead Training (VILT)

Course Credits: N/A (Certificate of completion will be awarded at the end)

Google Classroom code: svpqjll

Google Meet code: meet.google.com/xjt-nqjz-oev

Table of Contents

Module 01

1. The Basic Elements of Python

A Python ***program***, sometimes called a ***script***, is a sequence of definitions and commands. These definitions are evaluated, and the commands are executed by the Python interpreter in something called the ***shell***. Typically, a new shell is created whenever the execution of a program begins. Usually, a window is associated with the shell.

**start Python Shell

A command often called a statement, instructs the interpreter to do something. For example, the statement **print(**'Python rules!'**)** instructs the interpreter to call the function ***print***, which will output the string Python rules! to the window associated with the shell.

The sequence of commands:

- **print(**'Python rules!'**)**
- **print(**'Python is cool!'**)**
- **print(**'Python rules,', 'Python is cool!'**)**

causes the interpreter to produce the output:

- Python rules!
- Python is cool!
- Python rules, Python is cool!

Notice that two values were passed to print in the third statement. The print function takes a variable number of arguments separated by commas, and prints them, separated by a space character, in the order in which they appear.

1.1 Objects, Expressions, and Numerical Types

Objects are the core things that Python programs manipulate. Every object has a ***type*** that defines the kinds of things that programs can do with that object. Types are either scalar or non-scalar. ***Scalar*** objects are indivisible. Think of them as the atoms of the language. ***Non-scalar*** objects, for example, strings, have internal structure.

Many types of objects can be denoted by ***literals*** in the text of a program. For example, the text 2 is a literal representing a number, and the text 'abc' is a literal representing a string.

Python has four types of scalar objects:

- ***int*** is used to represent integers. Literals of type int are written in the way we typically denote integers (e.g., -3 or 5 or 10002).
- ***float*** is used to represent real numbers. Literals of type float always include a decimal point (e.g., 3.0 or 3.17 or -28.72).
- ***bool*** is used to represent the Boolean values ***True*** and ***False***.
- ***None*** is a type with a single value.

Objects and ***operators*** can be combined to form ***expressions***, each of which evaluates to an object of some type. We will refer to this as the ***value*** of the expression. For example, the expression 3 + 2 denotes the object 5 of type int, and the expression 3.0 + 2.0 denotes the object 5.0 of type float.

The **==** operator is used to test whether two expressions evaluate to the same value and the **!=** operator is used to test whether two expressions evaluate to different values. A single = means something quite different, as we will see in Section 1.2. Be forewarned, you will make the mistake of typing "=" when you meant to type "==". Keep an eye out for this error.

The symbol **>>>** is a shell prompt indicating that the interpreter is expecting the user to type some Python code into the shell. The line below the line with the prompt is produced when the interpreter evaluates the Python code entered at the prompt, as illustrated by the following interaction with the interpreter:

```
>>> 3 + 2
5
>>> 3.0 + 2.0
5.0
>>> 3 != 2
True
```

The built-in Python function type can be used to find out the type of an object:

```
>>> type(3)
<type 'int'>
>>> type(3.0)
<type 'float'>
```

i+j is the sum of i and j. If i and j are both of type int, the result is an int. If either of them is a float, the result is a float.
i–j is i minus j. If i and j are both of type int, the result is an int. If either of them is a float, the result is a float.
i*j is the product of i and j. If i and j are both of type int, the result is an int. If either of them is a float, the result is a float.
i//j is integer division. For example, the value of 6//2 is the int 3 and the value of 6//4 is the int 1. The value is 1 because integer division returns the quotient and ignores the remainder. If j == 0, an error occurs.
i/j is i divided by j. In Python 3, the / operator, performs floating point division. For example, the value of 6/4 is 1.5. If j == 0, an error occurs. (In Python 2, when i and j are both of type int, the / operator behaves the same way as // and returns an int. If either i or j is a float, it behaves like the Python 3 / operator.)
i%j is the remainder when the int i is divided by the int j. It is typically pronounced "i mod j," which is short for "i modulo j."
ij** is i raised to the power j. If i and j are both of type int, the result is an int. If either of them is a float, the result is a float. The comparison operators are == (equal), != (not equal), > (greater), >= (at least), <, (less) and <= (at most).

Figure 1.1 Operators on types int and float

The arithmetic operators have the usual precedence.
For example, * binds more tightly than +, so the expression x + y * 2 is evaluated by first multiplying y by 2 and then adding the result to x.
The Order of evaluation can be changed by using parentheses to group subexpressions, e.g., (x + y) * 2 first adds x and y, and then multiplies the result by 2.

The primitive operators on type bool are **and**, **or**, and **not**:

- ***a and b*** is True if both a and b are True, and False otherwise.
- ***a or b*** is True if at least one of a or b is True, and False otherwise.
- ***not a*** is True if a is False, and False if a is True.

1.2 Variables and Assignment

Variables provide a way to associate names with objects. Consider the code

```
pi = 3
radius = 11
area = pi * (radius**2)
radius = 14
```

It first ***binds*** the names pi and radius to different objects of type int.10 It then binds the name area to a third object of type int. This is depicted in the left panel of Figure 1.2.

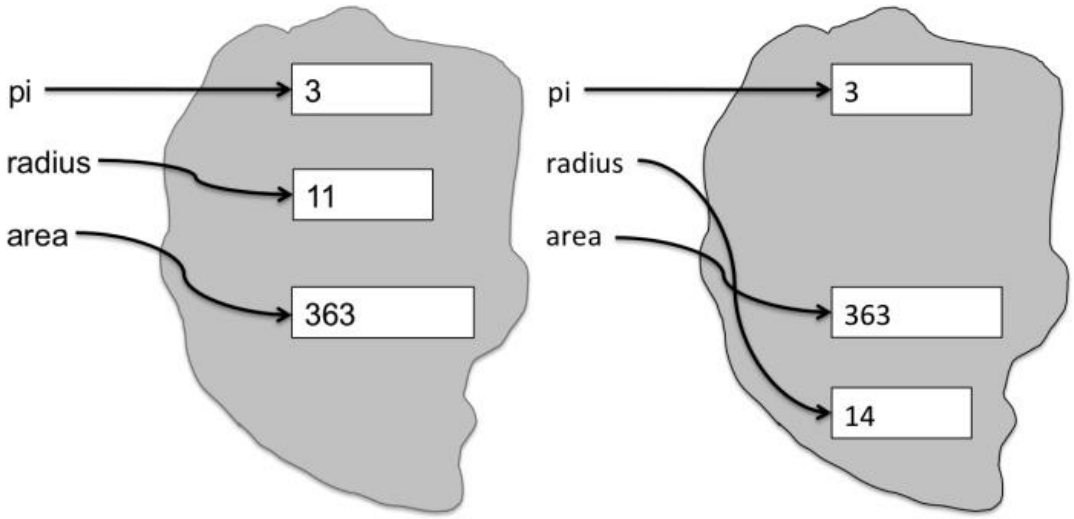

Figure 1.2 Binding of variables to objects

If the program then executes radius = 14, the name radius is rebound to a different object of type int, as shown in the right panel of Figure 1.2. Note that this assignment has no effect on the value to which area is bound. It is still bound to the object denoted by the expression 3 * (11 ** 2).

In Python***, a variable is just a name***, nothing more. Remember this—it is important. An ***assignment*** statement associates the name to the left of the = symbol with the object denoted by the expression to the right of the =. Remember this too. An object can have one, more than one, or no name associated with it.

Perhaps we shouldn't have said, "a variable is just a name." Despite what Juliet said, names matter. Programming languages let us describe computations in a way that allows machines to execute them. This does not mean that only computers read programs. As you will soon discover, it's not always easy to write programs that work correctly. Experienced programmers will confirm that they spend a great deal of time reading programs to understand why they behave as they do. It is therefore of critical importance to write programs in such a way that they are easy to read. An apt choice of variable names plays an important role in enhancing readability.

Consider the two code fragments

a = 3.14159 b = 11.2 c = a*(b**2)	pi = 3.14159 diameter = 11.2 area = pi*(diameter**2)

As far as Python is concerned, they are not different. When executed, they will do the same thing. To a human reader, however, they are quite different. When we read the fragment on the left, there is no a priori reason to suspect that anything is amiss. However, a glance at the code on the right should prompt us to be suspicious that something is wrong. Either the variable should have been named radius rather than diameter, or diameter should have been divided by 2.0 in the calculation of the area.

In Python, variable names can contain uppercase and lowercase letters, digits (but they cannot start with a digit), and the special character _. Python variable names are case-sensitive e.g., **Julie** and **julie** are different names.

Finally, there are a small number of ***reserved words*** (sometimes called ***keywords***) in Python that have built-in meanings and cannot be used as variable names. Different versions of Python have slightly different lists of reserved words.

The reserved words in Python 3 are:

and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, nonlocal, None, not, or, pass, raise, return, True, try, while, with, and yield.

Another good way to enhance the readability of code is to add comments. Text following the symbol **#** is not interpreted by Python. For example, one might write

```
side = 1 #length of sides of a unit square

radius = 1 #radius of a unit circle

#subtract the area of the unit circle from the area of the unit square

areaC = pi*radius**2

areaS = side*side

difference = areaS – areaC
```

Python allows multiple assignments.

The statement

x, y = 2, 3

binds x to 2 and y to 3. All the expressions on the right-hand side of the assignment are evaluated before any bindings are changed. This is convenient since it allows you to use multiple assignments to swap the bindings of two variables.

For example, the code

```
x, y = 2, 3
x, y = y, x
print('x =', x)
print('y =', y)
will print
x = 3
y = 2
```