Python Essentials 1: Module 2

Data types, variables, basic input-output operations, basic operators

In this module, you will learn:

how to write and run simple Python programs;
what Python literals, operators, and expressions are;
what variables are and what are the rules that govern them;
how to perform basic input and output operations.

(part 4)

What are variables?

It seems fairly obvious that Python should allow you to encode literals carrying number and text values.

You already know that you can do some arithmetic operations with these numbers: add, subtract, etc. You'll be doing that many times.

But it's quite a normal question to ask how to store the results of these operations, in order to use them in other operations, and so on.

How do you save the intermediate results, and use them again to produce subsequent ones?

Python will help you with that. It offers special "boxes" (containers) for that purpose, and these boxes are called variables - the name itself suggests that the content of these containers can be varied in (almost) any way.

What does every Python variable have?

    a name;
    a value (the content of the container)

Let's start with the issues related to a variable's name.

Variables do not appear in a program automatically. As developer, you must decide how many and which variables to use in your programs.

You must also name them.

If you want to give a name to a variable, you must follow some strict rules:

    the name of the variable must be composed of upper-case or lower-case letters, digits, and the character _ (underscore)
    the name of the variable must begin with a letter;
    the underscore character is a letter;
    upper- and lower-case letters are treated as different (a little differently than in the real world - Alice and ALICE are the same first names, but in Python they are two different variable names, and consequently, two different variables);
    the name of the variable must not be any of Python's reserved words (the keywords - we'll explain more about this soon).

Correct and incorrect variable names

Note that the same restrictions apply to function names.
Python does not impose restrictions on the length of variable names, but that doesn't mean that a long variable name is always better than a short one.

Here are some correct, but not always convenient variable names:
MyVariable, i, t34, Exchange_Rate, counter, days_to_christmas, TheNameIsSoLongThatYouWillMakeMistakesWithIt, _.

Moreover, Python lets you use not only Latin letters but also characters specific to languages that use other alphabets.
These variable names are also correct:
Adiós_Señora, sûr_la_mer, Einbahnstraße, переменная.

And now for some incorrect names:

10t (does not begin with a letter), Exchange Rate (contains a space)

NOTE

The PEP 8 -- Style Guide for Python Code recommends the following naming convention for variables and functions in Python:
    variable names should be lowercase, with words separated by underscores to improve readability (e.g., var, my_variable)
    function names follow the same convention as variable names (e.g., fun, my_function)
    it's also possible to use mixed case (e.g., myVariable), but only in contexts where that's already the prevailing style, to retain backwards compatibility with the adopted convention.

Keywords
Take a look at the list of words that play a very special role in every Python program.
```
[
  'False', 'None', 'True', 'and',   'as', 'assert', 'break', 'class', 'continue', 'def', 'del',
  'elif',  'else',  'except', 'finally', 'for', 'from',  'global', 'if',   'import', 'in', 'is',
  'lambda', 'nonlocal', 'not',   'or',    'pass', 'raise', 'return', 'try',   'while',  'with', 'yield'
]
```

They are called keywords or (more precisely) reserved keywords. They are reserved because you mustn't use them as names: neither for your variables, nor functions, nor any other named entities you want to create.

The meaning of the reserved word is predefined, and mustn't be changed in any way.

Fortunately, due to the fact that Python is case-sensitive, you can modify any of these words by changing the case of any letter, thus creating a new word, which is not reserved anymore.

For example - you can't name your variable like this:

import

You mustn't have a variable named in such a way - it is prohibited. But you can do this instead:

Import
These words might be a mystery to you now, but you'll soon learn the meaning of them.

Creating variables

What can you put inside a variable?

Anything.

You can use a variable to store any value of any of the already presented kinds, and many more of the ones we haven't shown you yet.

The value of a variable is what you have put into it. It can vary as often as you need or want. It can be an integer one moment, and a float a moment later, eventually becoming a string.

Let's talk now about two important things - how variables are created, and how to put values inside them (or rather - how to give or pass values to them).

REMEMBER

A variable comes into existence as a result of assigning a value to it. Unlike in other languages, you don't need to declare it in any special way.

If you assign any value to a nonexistent variable, the variable will be automatically created. You don't need to do anything else.

The creation (or otherwise - its syntax) is extremely simple: just use the name of the desired variable, then the equal sign (=) and the value you want to put into the variable.

Take a look at the snippet:

```
var = 1
print(var)
```

It consists of two simple instructions:

> The first of them creates a variable named var, and assigns a literal with an integer value equal to 1.
> The second prints the value of the newly created variable to the console.

Note: print() has yet another side to it - it can handle variables too. Do you know what the output of the snippet will be?

Using variables

You're allowed to use as many variable declarations as you need to achieve your goal, like this:

```
var = 1
account_balance = 1000.0
client_name = 'John Doe'
print(var, account_balance, client_name)
print(var)
```

You're not allowed to use a variable which doesn't exist (in other words, a variable that was not assigned a value).

This example will cause an error:

```
var = 1
print(Var)
```

We've tried to use a variable named Var, which doesn't have any value (note: var and Var are different entities, and have nothing in common as far as Python's concerned).

REMEMBER

You can use the print() function and combine text and variables using the + operator to output strings and variables, e.g.:

```
var = "3.8.5"
print("Python version: " + var)
```

Assigning a new value to an already existing variable

How do you assign a new value to an already created variable? In the same way. You just need to use the equal sign.

The equal sign is in fact an assignment operator. Although this may sound strange, the operator has a simple syntax and unambiguous interpretation.

It assigns the value of its right argument to the left, while the right argument may be an arbitrarily complex expression involving literals, operators and already defined variables.

Look at the code below:

```
var = 1
print(var)
var = var + 1
print(var)
```

The code sends two lines to the console:

```
1
2
```

The first line of the snippet creates a new variable named var and assigns 1 to it.

The statement reads: assign a value of 1 to a variable named var.

We can say it shorter: assign 1 to var.

Some prefer to read such a statement as: var becomes 1.

The third line assigns the same variable with the new value taken from the variable itself, summed with 1. Seeing a record like that, a mathematician would probably protest - no value may be equal to itself plus one. This is a contradiction. But Python treats the sign = not as equal to, but as assign a value.

So how do you read such a record in the program?

Take the current value of the variable var, add 1 to it and store the result in the variable var.

In effect, the value of variable var has been incremented by one, which has nothing to do with comparing the variable with any value.

Do you know what the output of the following snippet will be?

```
var = 100
var = 200 + 300
print(var)
```

Solving simple mathematical problems

Now you should be able to construct a short program solving simple mathematical problems such as the Pythagorean theorem:

The square of the hypotenuse is equal to the sum of the squares of the other two sides.

The following code evaluates the length of the hypotenuse (i.e., the longest side of a right-angled triangle, the one opposite of the right angle) using the Pythagorean theorem:

```
a = 3.0
b = 4.0
c = (a ** 2 + b ** 2) ** 0.5
print("c =", c)
```

Note: we need to make use of the ** operator to evaluate the square root as:

$\sqrt{x} = x^{(1/2)}$

and

$c = \sqrt{a^2 + b^2}$

Can you guess the output of the code?

LAB

Estimated time
10 minutes

Level of difficulty
Easy

Objectives
becoming familiar with the concept of storing and working with different data types in Python;
experimenting with Python code.
Scenario
Here is a short story:

Once upon a time in Appleland, John had three apples, Mary had five apples, and Adam had six apples. They were all very happy and lived for a long time. End of story.

Your task is to:

    create the variables: john, mary, and adam;
    assign values to the variables. The values must be equal to the numbers of fruit possessed by John, Mary, and Adam respectively;
    having stored the numbers in the variables, print the variables on one line, and separate each of them with a comma;
    now create a new variable named total_apples equal to addition of the three former variables.
    print the value stored in total_apples to the console;
    experiment with your code: create new variables, assign different values to them, and perform various arithmetic operations on them (e.g., +, -, *, /, //, etc.). Try to print a string and an integer together on one line, e.g., "Total number of apples:" and total_apples.

Shortcut operators

It's time for the next set of operators that make a developer's life easier.

Very often, we want to use one and the same variable both to the right and left sides of the = operator.

For example, if we need to calculate a series of successive values of powers of 2, we may use a piece like this:

```
x = x * 2
```

You may use an expression like this if you can't fall asleep and you're trying to deal with it using some good, old-fashioned methods:

```
sheep = sheep + 1
```

Python offers you a shortened way of writing operations like these, which can be coded as follows:

```
x *= 2
sheep += 1
```

Let's try to present a general description for these operations.

If op is a two-argument operator (this is a very important condition) and the operator is used in the following context:

variable = variable op expression

It can be simplified and shown as follows:

variable op= expression

Take a look at the examples below. Make sure you understand them all.

```
i = i + 2 * j ⇒ i += 2 * j
```

```
var = var / 2 ⇒ var /= 2
```

```
rem = rem % 10 ⇒ rem %= 10
```

```
j = j - (i + var + rem) ⇒ j -= (i + var + rem)
```

```
x = x ** 2 ⇒ x **= 2
```

LAB

Estimated time
10 minutes

Level of difficulty
Easy

Objectives
becoming familiar with the concept of, and working with, variables;
performing basic computations and conversions;
experimenting with Python code.
Scenario
Miles and kilometers are units of length or distance.

Bearing in mind that 1 mile is equal to approximately 1.61 kilometers, complete the program in the editor so that it converts:

miles to kilometers;
kilometers to miles.
Do not change anything in the existing code. Write your code in the places indicated by ###. Test your program with the data we've provided in the source code.

```
kilometers = 12.25
miles = 7.38

miles_to_kilometers = ###
kilometers_to_miles = ###

print(miles, "miles is", round(miles_to_kilometers, 2), "kilometers")
print(kilometers, "kilometers is", round(kilometers_to_miles, 2), "miles")
```

Pay particular attention to what is going on inside the print() function. Analyze how we provide multiple arguments to the function, and how we output the expected data.

Note that some of the arguments inside the print() function are strings (e.g., "miles is", whereas some other are variables (e.g., miles).

TIP

There's one more interesting thing happening there. Can you see another function inside the print() function? It's the round() function. Its job is to round the outputted result to the number of decimal places specified in the parentheses, and return a float (inside the round() function you can find the variable name, a comma, and the number of decimal places we're aiming for). We're going to talk about functions very soon, so don't worry that everything may not be fully clear yet. We just want to spark your curiosity.

After completing the lab, open Sandbox, and experiment more. Try to write different converters, e.g., a USD to EUR converter, a temperature converter, etc. - let your imagination fly! Try to output the results by combining strings and

variables. Try to use and experiment with the round() function to round your results to one, two, or three decimal places. Check out what happens if you don't provide any number of digits. Remember to test your programs.

Experiment, draw conclusions, and learn. Be curious.

Expected output
7.38 miles is 11.88 kilometers
12.25 kilometers is 7.61 miles

LAB

Estimated time
10-15 minutes

Level of difficulty
Easy

Objectives
becoming familiar with the concept of numbers, operators, and arithmetic operations in Python;
performing basic calculations.
Scenario
Take a look at the code in the editor: it reads a float value, puts it into a variable named x, and prints the value of a variable named y. Your task is to complete the code in order to evaluate the following expression:

3x^3 - 2x^2 + 3x - 1

The result should be assigned to y.

Remember that classical algebraic notation likes to omit the multiplication operator - you need to use it explicitly. Note how we change data type to make sure that x is of type float.

Keep your code clean and readable, and test it using the data we've provided, each time assigning it to the x variable (by hardcoding it). Don't be discouraged by any initial failures. Be persistent and inquisitive.

Test Data
Sample input

x = 0
x = 1
x = -1

Expected Output

y = -1.0
y = 3.0
y = -9.0

Key takeaways

1. A variable is a named location reserved to store values in the memory. A variable is created or initialized automatically when you assign a value to it for the first time. (2.1.4.1)

2. Each variable must have a unique name - an identifier. A legal identifier name must be a non-empty sequence of characters, must begin with the underscore(_), or a letter, and it cannot be a Python keyword. The first character may be followed by underscores, letters, and digits. Identifiers in Python are case-sensitive. (2.1.4.1)

3. Python is a dynamically-typed language, which means you don't need to declare variables in it. (2.1.4.3) To assign values to variables, you can use a simple assignment operator in the form of the equal (=) sign, i.e., var = 1.

4. You can also use compound assignment operators (shortcut operators) to modify values assigned to variables, e.g., var += 1, or var /= 5 * 2. (2.1.4.8)

5. You can assign new values to already existing variables using the assignment operator or one of the compound operators, e.g.: (2.1.4.5)

```
var = 2
print(var)

var = 3
print(var)

var += 1
print(var)
```

6. You can combine text and variables using the + operator, and use the print() function to output strings and variables, e.g.: (2.1.4.4)

```
var = "007"
print("Agent " + var)
```

Exercise 1

What is the output of the following snippet?

```
var = 2
var = 3
print(var)
```

Exercise 2

Which of the following variable names are illegal in Python?

```
my_var
m
101
```

averylongvariablename
m101
m 101
Del
del
Check
Exercise 3

What is the output of the following snippet?

```
a = '1'
b = "1"
print(a + b)
```

Exercise 4

What is the output of the following snippet?

```
a = 6
b = 3
a /= 2 * b
print(a)
```