

## Set 3: Variables

- Skill 3.1: Explain the purpose of variables
- Skill 3.2: Apply proper naming conventions to variables
- Skill 3.3: Create variables to store integers
- Skill 3.4: Apply math operations to variables
- Skill 3.5: Interpret integer, float, and String variable types
- Skill 3.6: Create multi-line Strings
- Skill 3.7: Concatenate variables
- Skill 3.8: Apply compound operators
- Skill 3.9: Convert variable types
- Skill 3.10: Identify common errors associated with variables

### Skill 3.1: Explain the purpose of variables

#### Skill 3.1 Concepts

Variables are containers for storing data values and are essential for controlling the memory in our programs. You can think of variables like a box that can hold a value.



Every Python variable has the following,

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

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

### Skill 3.1 Exercise 1

### Skill 3.2: Apply proper naming conventions to variables

#### Skill 3.2 Concepts

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;
- 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).

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.

Below are some examples of correct and incorrect variable names. Notice that Python lets you use not only Latin letters but also characters specific to languages that use other alphabets.

Correct	Incorrect
MyVariable i l t34 Exchange_Rate counter days_to_christmas TheNameIsTooLongAndHardlyReadable - Adiós_Señora sûr_la_mer Einbahnstraße переменная.	123 1i my variable my\$variable score# !important

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 camel case (e.g., myVariable), but only in contexts where that's already the prevailing style, to retain backward compatibility with the adopted convention.

Take a look at the list of words that play a very special role in every Python program.

## Python Keywords

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

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**

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.

### Skill 3.2 Exercise 1

## Skill 3.3: Create variables to store integers

### Skill 3.3 Concepts

Unlike many other languages, Python has no command for declaring a variable. A variable is created the moment you first assign a value to it. In the example, 15 has been assigned the variable spam.

```
spam = 15;
```



To check the value of a variable you can print it,

Code	Output
print(spam)	15

You can change the value stored in a variable by entering another assignment statement. For example,

```
spam = 15          #spam has a value of 15
spam = 3          #spam is now 3
```

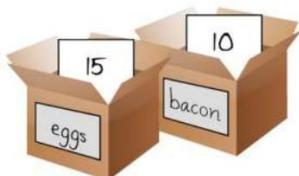
The value of spam was originally 15, but it was replaced or overwritten with the value of 20, then overwritten again with the value of 3. This process of overwriting is illustrated below,



You can create as many variables as you need in your programs. For example, let's assign different values to two variables named eggs and bacon, like so:

```
bacon = 10
eggs = 15
```

Now the bacon variable has 10 inside it, and eggs has 15 inside it. Each variable is its own box with its own value, like shown below.



### Skill 3.3 Exercise 1

## Skill 3.4: Apply math operations to variables

### Skill 3.4 Concepts

Integers in python can be manipulated using the basic math operations you are familiar with,

Operator	Action
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Integer division
%	Modulus
**	Exponentiation

In the examples below, we apply the math operations to change the value stored in the variable spam.

Code	Output
spam = 15 spam = spam + 5 spam = spam/5 spam = spam*3 spam = spam - 2 spam = spam // 3 spam = spam ** 2 spam = spam/2 print(spam)	4.5

### Skill 3.4 Exercise 1

## Skill 3.5: Interpret integer, float, and String variable types

### Skill 3.5 Concepts

#### Integers and floats

Integers (or ints for short) are whole numbers such as 4, 99, and 0. Floating point numbers (or floats for short) are fractions or numbers with decimal points like 3.5, 42.1, and 5.0. In Python, the number 5 is an integer, but 5.0 is a float. These numbers are called values. Consider the following example,

Code	Output
print(8 * 3 / 2 + 2 + 7 - 9)	12.0

Notice that the / division operator evaluates to a float value, as in 24 / 2 evaluating to 12.0. Math operations with float values also evaluate to float values, as in 12.0 + 2 evaluating to 14.0.

## Strings

In Python, text values are called strings. String values can be used just like integer or float values. You can store strings in variables. In code, string values start and end with a single quote (' ) or double quote(" ).

```
spam = 'hello'
```

The quotes tell Python where the string begins and ends. They are not part of the string value's text. Now if you type spam into the interactive shell, you will see the contents of the spam variable. Remember, Python evaluates variables to the value stored inside the variable. In this case, this is the string 'hello'.

Code	Output
spam = 'hello' print(spam)	hello

Strings can have any keyboard character in them and can be as long as you want. Below are all examples of string variable types,

```
a = 'hello'  
b = 'Hi there! '  
c = 'KITTENS'  
d = '7 apples, 14 oranges, 3 lemons'  
f = 'Anything not pertaining to elephants is irrelephant.'  
g = 'A long time ago, in a galaxy far, far away...'  
h = '0*&#wY%*&OCfsdY0*&gfc%Y0*&%3yc8r2'
```

### Skill 3.5 Exercise 1

### Skill 3.6: Create multi-line Strings

#### **Skill 3.6 Concepts**

Python strings are very flexible, but if we try to create a String that occupies multiple lines we find ourselves face-to-face with a SyntaxError. Python offers a solution: multi-line strings. By using three quote-marks ("""" or """) instead of one, we tell the program that the string doesn't end until the next triple-quote. This method is useful if the string being defined contains a lot of quotation marks and we want to be sure we don't close it prematurely.

Below is an example of a multi-line String.

Code	Output
<pre>leaves_of_grass = """ Poets to come! orators, singers, musicians to come! Not to-day is to justify me and answer what I am for, But you, a new brood, native, athletic, continental, greater than before known, Arouse! for you must justify me. """</pre>	<p>Poets to come! orators, singers, musicians to come! Not to-day is to justify me and answer what I am for, But you, a new brood, native, athletic, continental, greater than before known, Arouse! for you must justify me.</p>

In the above example, we assign a famous poet's words to a variable. Even though the quote contains multiple linebreaks, the code works!

If a multi-line string isn't assigned a variable or used in an expression it is treated as a comment.

A comment is a note or explanation in the source code of a computer program. They are added with the purpose of making the code easier for ourselves or other developers to understand in the future, and they are generally ignored by compilers and interpreters.

### Skill 3.6 Exercise 1

#### **Skill 3.7: Concatenate variables**

##### **Skill 3.7 Concepts**

Concatenation means to combine or attach things together. In Python this is done with the plus (+) sign.

Consider the following example. Below, the variables initialized to "Hello" and "good buddy" are concatenated and assigned to a new variable c. The program prints Hellogood buddy to the screen.

```
mm = "Hello"
nx = "good buddy"
c = mm + nx
print(c) #prints Hellogood buddy... notice no space between o & g
```

A space between the words "Hello" and "good" could have been achieved by concatenating an empty string between these words as shown below,

```
mm = "Hello"
nx = "good buddy"
print(mm+ " " + nx) #prints Hello good buddy..notice the space
```

The example below also illustrates another way a space could have been achieved,

```
print("Hello" + " good buddy") #prints Hello good buddy
```

Unlike some other languages like java and javascript, it is not possible to concatenate a String with a numeric variable as follows.

```
x = 17
s = "Was haben wir gemacht?" #German for "What have we done"
combo = s + "" + x
print(combo) #results is an error
```

The example below prompts the user for their first and last name, then prints a message to the console.

```
firstName = input('Enter your first name')
lastName = input('Enter your last name')
fullName = firstName + ' ' + lastName
print('Hello' + fullName)
```

### Skill 3.7 Exercise 1

### **Skill 3.8: Apply compound operators**

#### **Skill 3.8 Concepts**

A compound assignment operator is an operator that performs a calculation and an assignment at the same time. In the below example, x can be re-assigned explicitly using  $x = x + 5$ ; x can also be re-assigned using the addition compound operator.

```
x = 10
x = x + 5      #x is 15
x += 5         #x is now 20
```

Compound operators can be applied to all arithmetic operations. How this is done is illustrated below,

- Syntax Example                          Simplified meaning
- a.  $x += 3;$        $\rightarrow$        $x = x + 3;$
- b.  $x -= y - 2;$        $\rightarrow$        $x = x - (y - 2);$
- c.  $*=$   
 $z *= 46;$        $\rightarrow$        $z = z * 46;$
- d.  $/=$   
 $p /= x-z;$        $\rightarrow$        $p = p / (x-z);$
- e.  $\%=$   
 $j \%= 2$        $\rightarrow$        $j = j \% 2;$

The compound operator can also be applied to Strings.

Code	Output
<pre>someString = "a" someString += "b" someString += "c"  print(someString)</pre>	abc

### Skill 3.8 Exercise 1

#### **Skill 3.9: Convert variable types**

##### **Skill 3.9 Concepts**

Previously we saw that the following code resulted in an error because Python does not allow for concatenating strings with numerical values.

```
x = 17
s = "Was haben wir gemacht?" #German for "What have we done"
combo = s + "" + x
print(combo) #results is an error
```

It is possible to convert data types. This process is also referred to as *casting*. Below are some examples,

Code	Output
<pre>x = 5.0 print(int(x))</pre>	5
<pre>x = 2 print(float(x))</pre>	2.0
<pre>a = 2 b = "I have " c = " cats." print(b + str(a) + c)</pre>	I have 2 cats.

Notice in the last example above we were able to combine an integer value with strings by converting it to a string using the `str()` function.

It is also possible to convert a string to a number, as long as the value in quotes is a valid number. You will get an error, however, if the value in between the quotes is not a valid number.

Code	Output
<pre>x = "5.0" print(float(x)/2)</pre>	2.5
<pre>x = "2" print(int(x) + 10)</pre>	12
<pre>x = "two" print(int(x) + 10)</pre>	ValueError: invalid literal for int() with base 10: 'two'

### [Skill 3.9 Exercise 1](#)

### [Skill 3.10: Identify common errors associated with variables](#)

#### **Skill 3.10 Concepts**

You cannot use a variable before an assignment statement creates it. Python will give you a *NameError* because no such variable by that name exists yet. Mistyping the variable name also causes this error.

Code	Output
<pre>print(someVariable)</pre>	print(myVariable) NameError: name 'myVariable' is not defined
<pre>someVariable = "This is a string variable!" print(someVarible)</pre>	print(someVariable) NameError: name 'someVarible' is not defined

In the first example, we tried to print `someVariable` before it was defined. In the second example, `someVariable` was mistyped in the print statement.