IBM Developer
SKILLS NETWORK

Writing Your First Python Code

Estimated time needed: **25** minutes

Objectives

After completing this lab you will be able to:

- Write basic code in Python
- Work with various types of data in Python
- Convert the data from one type to another
- Use expressions and variables to perform operations

Table of Contents

Estimated time needed: **25 min**

Say "Hello" to the world in Python

When learning a new programming language, it is customary to start with an "hello world" example. As simple as it is, this one line of code will ensure that we know how to print a string in output and how to execute code within cells in a notebook.

[Tip]: To execute the Python code in the code cell below, click on the cell to select it and press `Shift` + `Enter`.

In [1]:
```python
# Try your first Python output

print('Hello, Python!')
```
```
Hello, Python!
```

After executing the cell above, you should see that Python prints `Hello, Python!`. Congratulations on running your first Python code!

[Tip:] `print()` is a function. You passed the string `'Hello, Python!'` as an argument to instruct Python on what to print.

What version of Python are we using?

There are two popular versions of the Python programming language in use today: Python 2 and Python 3. The Python community has decided to move on from Python 2 to Python 3, and many popular libraries have announced that they will no longer support Python 2.

Since Python 3 is the future, in this course we will be using it exclusively. How do we know that our notebook is executed by a Python 3 runtime? We can look in the top-right hand corner of this notebook and see "Python 3".

We can also ask Python directly and obtain a detailed answer. Try executing the following code:

In [3]:
```python
# Check the Python Version

import sys
print(sys.version)
```
```
3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
```

[Tip:] `sys` is a built-in module that contains many system-specific parameters and functions, including the Python version in use. Before using it, we must explictly `import` it.

Writing comments in Python

In addition to writing code, note that it's always a good idea to add comments to your code. It will help others understand what you were trying to accomplish (the reason why you wrote a given snippet of code). Not only does this help **other people** understand your code, it can also serve as a reminder **to you** when you come back to it weeks or months later.

To write comments in Python, use the number symbol `#` before writing your comment. When you run your code, Python will ignore everything past the `#` on a given line.

In [4]:
```python
# Practice on writing comments

print('Hello, Python!') # This line prints a string
# print('Hi')
```
```
Hello, Python!
```

After executing the cell above, you should notice that `This line prints a string` did not appear in the output, because it was a comment (and thus ignored by Python).

The second line was also not executed because `print('Hi')` was preceded by the number sign (`#`) as well! Since this isn't an explanatory comment from the programmer, but an actual line of code, we might say that the programmer *commented out* that second line of code.

Errors in Python

Everyone makes mistakes. For many types of mistakes, Python will tell you that you have made a mistake by giving you an error message. It is important to read error messages carefully to really understand where you made a mistake and how you may go about correcting it.

For example, if you spell `print` as `frint`, Python will display an error message. Give it a try:

In [5]:
```python
# Print string as error message

frint("Hello, Python!")
```
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-313a1769a8a5> in <module>
      1 # Print string as error message
      2
----> 3 frint("Hello, Python!")

NameError: name 'frint' is not defined
```

The error message tells you:

1. where the error occurred (more useful in large notebook cells or scripts), and
2. what kind of error it was (NameError)

Here, Python attempted to run the function `frint`, but could not determine what `frint` is since it's not a built-in function and it has not been previously defined by us either.

You'll notice that if we make a different type of mistake, by forgetting to close the string, we'll obtain a different error (i.e., a `SyntaxError`). Try it below:

In [6]:
```python
# Try to see built-in error message

print("Hello, Python!)
```
```
  File "<ipython-input-6-f0b5a635e1a2>", line 3
    print("Hello, Python!)
                          ^
SyntaxError: EOL while scanning string literal
```

Does Python know about your error before it runs your code?

Python is what is called an *interpreted language*. Compiled languages examine your entire program at compile time, and are able to warn you about a whole class of errors prior to execution. In contrast, Python interprets your script line by line as it executes it. Python will stop executing the entire program when it encounters an error (unless the error is expected and handled by the programmer, a more advanced subject that we'll cover later on in this course).

Try to run the code in the cell below and see what happens:

In [7]:
```python
# Print string and error to see the running order

print("This will be printed")
frint("This will cause an error")
print("This will NOT be printed")
```
```
This will be printed
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-af59af1b345d> in <module>
      2
      3 print("This will be printed")
----> 4 frint("This will cause an error")
      5 print("This will NOT be printed")

NameError: name 'frint' is not defined
```

Exercise: Your First Program

Generations of programmers have started their coding careers by simply printing "Hello, world!". You will be following in their footsteps.

In the code cell below, use the `print()` function to print out the phrase: `Hello, world!`

In [8]:
```python
# Write your code below. Don't forget to press Shift+Enter to execute the cell
print("Hello, world!")
```
```
Hello, world!
```

▶ Click here for the solution

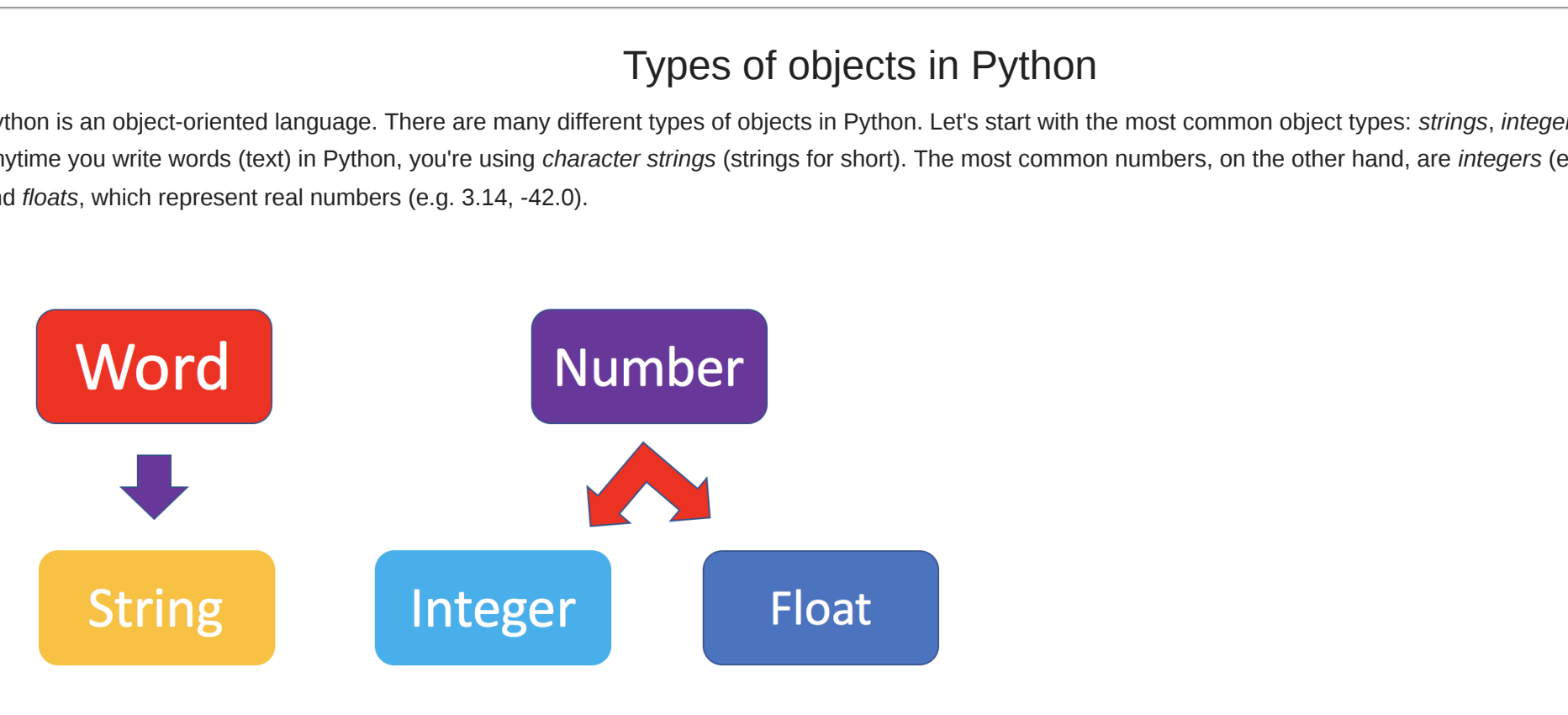
Now, let's enhance your code with a comment. In the code cell below, print out the phrase: `Hello, world!` and comment it with the phrase `Print the traditional hello world` all in one line of code.

In [9]:
```python
# Write your code below. Don't forget to press Shift+Enter to execute the cell
print("Hello, world!") # Print the traditional hello world
```
```
Hello, world!
```

▶ Click here for the solution

Types of objects in Python

Python is an object-oriented language. There are many different types of objects in Python. Let's start with the most common object types: *strings*, *integers* and *floats*. Anytime you write words (text) in Python, you're using *character strings* (strings for short). The most common numbers, on the other hand, are *integers* (e.g. -1, 0, 100) and *floats*, which represent real numbers (e.g. 3.14, -42.0).



The following code cells contain some examples.

In [10]:
```python
# Integer

11
```
Out[10]: `11`

In [11]:
```python
# Float

2.14
```
Out[11]: `2.14`

In [12]:
```python
# String

"Hello, Python 101!"
```
Out[12]: `'Hello, Python 101!'`

You can get Python to tell you the type of an expression by using the built-in `type()` function. You'll notice that Python refers to integers as `int`, floats as `float`, and character strings as `str`.

In [13]:
```python
# Type of 12

type(12)
```
Out[13]: `int`

In [14]:
```python
# Type of 2.14

type(2.14)
```
Out[14]: `float`

In [ ]:
```python
# Type of "Hello, Python 101!"

type("Hello, Python 101!")
```

In the code cell below, use the `type()` function to check the object type of `12.0`.

In [15]:
```python
# Write your code below. Don't forget to press Shift+Enter to execute the cell
type(12.0)
```
Out[15]: `float`

▶ Click here for the solution

Integers

Here are some examples of integers. Integers can be negative or positive numbers:

-4	-3	-2	-1	0	1	2	3	4

We can verify this is the case by using, you guessed it, the `type()` function:

In [16]:
```python
# Print the type of -1

type(-1)
```
Out[16]: `int`

In [17]:
```python
# Print the type of 4

type(4)
```
Out[17]: `int`

In [18]:
```python
# Print the type of 0

type(0)
```
Out[18]: `int`

Floats

Floats represent real numbers; they are a superset of integer numbers but also include "numbers with decimals". There are some limitations when it comes to machines representing real numbers, but floating point numbers are a good representation in most cases. You can learn more about the specifics of floats for your runtime environment, by checking the value of `sys.float_info`. This will also tell you what's the largest and smallest number that can be represented with them.

Once again, can test some examples with the `type()` function:

In [19]:
```python
# Print the type of 1.0

type(1.0) # Notice that 1 is an int, and 1.0 is a float
```
Out[19]: `float`

In [20]:
```python
# Print the type of 0.5

type(0.5)
```
Out[20]: `float`

In [21]:
```python
# Print the type of 0.56

type(0.56)
```
Out[21]: `float`

In [22]:
```python
# System settings about float type

sys.float_info
```
Out[22]:
```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

Converting from one object type to a different object type

You can change the type of the object in Python; this is called typecasting. For example, you can convert an *integer* into a *float* (e.g. 2 to 2.0).

Let's try it:

In [23]:
```python
# Verify that this is an integer

type(2)
```
Out[23]: `int`

Converting integers to floats

Let's cast integer 2 to float:

In [24]:
```python
# Convert 2 to a float

float(2)
```
Out[24]: `2.0`

In [25]:
```python
# Convert integer 2 to a float and check its type

type(float(2))
```
Out[25]: `float`

When we convert an integer into a float, we don't really change the value (i.e., the significand) of the number. However, if we cast a float into an integer, we could potentially lose some information. For example, if we cast the float 1.1 to integer we will get 1 and lose the decimal information (i.e., 0.1):

In [26]:
```python
# Casting 1.1 to integer will result in loss of information

int(1.1)
```
Out[26]: `1`

Converting from strings to integers or floats

Sometimes, we can have a string that contains a number within it. If this is the case, we can cast that string that represents a number into an integer using `int()`:

In [27]:
```python
# Convert a string into an integer

int('1')
```
Out[27]: `1`

But if you try to do so with a string that is not a perfect match for a number, you'll get an error. Try the following:

In [28]:
```python
# Convert a string into an integer with error

int('1 or 2 people')
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-28-b78145d165c7> in <module>
      1 # Convert a string into an integer with error
----> 2 int('1 or 2 people')

ValueError: invalid literal for int() with base 10: '1 or 2 people'
```

You can also convert strings containing floating point numbers into float objects:

In [29]:
```python
# Convert the string "1.2" into a float

float('1.2')
```
Out[29]: `1.2`

[Tip:] Note that strings can be represented with single quotes (`'1.2'`) or double quotes (`"1.2"`), but you can't mix both (e.g., `"1.2'`).

Converting numbers to strings

If we can convert strings to numbers, it is only natural to assume that we can convert numbers to strings, right?

In [30]:
```python
# Convert an integer to a string

str(1)
```
Out[30]: `'1'`

And there is no reason why we shouldn't be able to make floats into strings as well:

In [31]:
```python
# Convert a float to a string

str(1.2)
```
Out[31]: `'1.2'`

Boolean data type

Boolean is another important type in Python. An object of type *Boolean* can take on one of two values: `True` or `False`:

In [32]:
```python
# Value true

True
```
Out[32]: `True`

Notice that the value `True` has an uppercase "T". The same is true for `False` (i.e. you must use the uppercase "F").

In [33]:
```python
# Value false

False
```
Out[33]: `False`

When you ask Python to display the type of a boolean object it will show `bool` which stands for *boolean*:

In [34]:
```python
# Type of True

type(True)
```
Out[34]: `bool`

In [35]:
```python
# Type of False

type(False)
```
Out[35]: `bool`

We can cast boolean objects to other data types. If we cast a boolean with a value of `True` to an integer or float we will get a one. If we cast a boolean with a value of `False` to an integer or float we will get a zero. Similarly, if we cast a 1 to a Boolean, you get a `True`. And if we cast a 0 to a Boolean we will get a `False`. Let's give it a try:

In [36]:
```python
# Convert True to int

int(True)
```
Out[36]: `1`

In [37]:
```python
# Convert 1 to boolean

bool(1)
```
Out[37]: `True`

In [38]:
```python
# Convert 0 to boolean

bool(0)
```
Out[38]: `False`

In [39]:
```python
# Convert True to float

float(True)
```
Out[39]: `1.0`

Exercise: Types

What is the data type of the result of: `6 / 2`?

In [40]:
```python
# Write your code below. Don't forget to press Shift+Enter to execute the cell
type(6/2)
```
Out[40]: `float`

▶ Click here for the solution

What is the type of the result of: `6 // 2`? (Note the double slash `//`.)

In [41]:
```python
# Write your code below. Don't forget to press Shift+Enter to execute the cell
type(6//2)
```
Out[41]: `int`

▶ Click here for the solution

Expression and Variables

Expressions

Expressions in Python can include operations among compatible types (e.g., integers and floats). For example, basic arithmetic operations like adding multiple numbers:

In [42]:
```python
# Addition operation expression

43 + 60 + 16 + 41
```
Out[42]: `160`

We can perform subtraction operations using the minus operator. In this case the result is a negative number:

In [43]:
```python
# Subtraction operation expression

50 - 60
```
Out[43]: `-10`

We can do multiplication using an asterisk:

In [44]:
```python
# Multiplication operation expression

5 * 5
```
Out[44]: `25`

We can also perform division with the forward slash:

In [45]:
```python
# Division operation expression

25 / 5
```
Out[45]: `5.0`

In [ ]:
```python
# Division operation expression

25 / 6
```

As seen in the quiz above, we can use the double slash for integer division, where the result is rounded down to the nearest integer:

In [46]:
```python
# Integer division operation expression

25 // 5
```
Out[46]: `5`

In [47]:
```python
# Integer division operation expression

25 // 6
```
Out[47]: `4`

Exercise: Expression

Let's write an expression that calculates how many hours there are in 160 minutes:

In [48]:
```python
# Write your code below. Don't forget to press Shift+Enter to execute the cell
160/60
```
Out[48]: `2.6666666666666665`

▶ Click here for the solution

Python follows well accepted mathematical conventions when evaluating mathematical expressions. In the following example, Python adds 30 to the result of the multiplication (i.e., 120).

In [49]:
```python
# Mathematical expression

30 + 2 * 60
```
Out[49]: `150`

And just like mathematics, expressions enclosed in parentheses have priority. So the following multiplies 32 by 60.

In [50]:
```python
# Mathematical expression

(30 + 2) * 60
```
Out[50]: `1920`

Variables

Just like with most programming languages, we can store values in variables, so we can use them later on. For example:

In [51]:
```python
# Store value into variable

x = 43 + 60 + 16 + 41
```

To see the value of `x` in a Notebook, we can simply place it on the last line of a cell:

In [52]:
```python
# Print out the value in variable

x
```
Out[52]: `160`

We can also perform operations on `x` and save the result to a new variable:

In [53]:
```python
# Use another variable to store the result of the operation between variable and value

y = x / 60
y
```
Out[53]: `2.6666666666666665`

If we save a value to an existing variable, the new value will overwrite the previous value:

In [54]:
```python
# Overwrite variable with new value

x = x / 60
x
```
Out[54]: `2.6666666666666665`

It's a good practice to use meaningful variable names, so you and others can read the code and understand it more easily:

In [55]:
```python
# Name the variables meaningfully

total_min = 43 + 42 + 57 # Total length of albums in minutes
total_min
```
Out[55]: `142`

In [56]:
```python
# Name the variables meaningfully

total_hours = total_min / 60 # Total length of albums in hours
total_hours
```
Out[56]: `2.3666666666666667`

In the cells above we added the length of three albums in minutes and stored it in `total_min`. We then divided it by 60 to calculate total length `total_hours` in hours. You can also do it all at once in a single expression, as long as you use parenthesis to add the albums length before you divide, as shown below.

In [57]:
```python
# Complicate expression

total_hours = (43 + 42 + 57) / 60  # Total hours in a single expression
total_hours
```
Out[57]: `2.3666666666666667`

If you'd rather have total hours as an integer, you can of course replace the floating point division with integer division (i.e., `//`).

Exercise: Expression and Variables in Python

What is the value of `x` where `x = 3 + 2 * 2`

In [58]:
```python
x = 3 + 2 * 2# Write your code below. Don't forget to press Shift+Enter to execute the cell
x = 3 + 2 * 2
```

▶ Click here for the solution

What is the value of `y` where `y = (3 + 2) * 2`?

In [63]:
```python
# Write your code below. Don't forget to press Shift+Enter to execute the cell
y = (3+2)*2
y
```
Out[63]: `10`

▶ Click here for the solution

What is the value of `z` where `z = x + y`?

In [66]:
```python
# Write your code below. Don't forget to press Shift+Enter to execute the cell
z = x+y
z
```
Out[66]: `17`

▶ Click here for the solution

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python.

Author

Joseph Santarcangelo

Other contributors

Mavis Zhou

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-01-10	2.1	Malika	Removed the readme for GitShare
2020-08-26	2.0	Lavanya	Moved lab to course repo in GitLab