PY0101EN-1-1-Write_your_first_python_code

December 26, 2021

1 Writing Your First Python Code

1.1 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

```
<ul>
    <li>
        <a href="https://#hello">Say "Hello" to the world in Python</a>
        <ul>
            <li><a href="https://version/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_c
            <li><a href="https://comments/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_
            <li><a href="https://errors/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_co
            <li><a href="https://python_error/?utm_medium=Exinfluencer&utm_source=Exinfluencer&
            <li><a href="https://exercise/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_
        </ul>
    </li>
    <li>
        <a href="https://#types_objects">Types of objects in Python</a>
        <ul>
            <li><a href="https://int/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_conte
            <li><a href="https://float/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_con
            <li><a href="https://convert/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_c
            <li><a href="https://bool/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_cont
            <li><a href="https://exer_type/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm
        </ul>
    </li>
    <li>
        <a href="https://#https://exp/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_cont
        <ul>
            <li><a href="exp">Expressions</a></li>
            <li><a href="https://exer_exp/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_
            <li><a href="https://var/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_conte
            <li><a href="https://exer_exp_var/?utm_medium=Exinfluencer&utm_source=Exinfluencer&
        </ul>
    </li>
</ul>
<p>
</p>
```

Say “Hello” to the world in Python

[1]:
```
print('Hello, Python!')
```

```
Hello, Python!
```

```
[Tip:] <code>print()</code> is a function. You passed the string <code>'Hello, Python!'</code>
```

What version of Python are we using?

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.

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

[1]:
```
import sys
print(sys.version)
```

```
3.7.6 | packaged by conda-forge | (default, Mar 23 2020, 23:03:20)
[GCC 7.3.0]
```

```
[Tip:] <code>sys</code> is a built-in module that contains many system-specific parameters and
```

Writing comments in Python

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). also serve as a reminder to you when you come back to it weeks or months later.

[3]:
```
# Practice on writing comments

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

```
Hello, Python!
```

Errors in Python

[4]:
```
# Print string as error message

frint("Hello, Python!")
```

```
          ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-4-313a1769a8a5> in <module>
         1 # Print string as error message
         2
   ----> 3 frint("Hello, Python!")


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

The error message tells you:

where the error occurred (more useful in large notebook cells or scripts), and

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:

```
[5]: # Try to see built-in error message

     print("Hello, Python!)
```

```
         File "<ipython-input-5-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 called as interpreted language. Compiled languages examine your entire program at compile time, & 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 entire program when it encounters error (unless the error is expected and handled by the programmer, a more advanced subject that we'll cover later on in this course)

```
[6]: # 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-6-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

[7]:
```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
print("Hello World")
```

```
Hello World
```

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

```
Hello World!
```

Types of objects in Python

Python is object-oriented language. There are many different types of objects in Python. 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.

[9]:
```
# Integer

11
```

[9]:
```
11
```

[10]:
```
# Float

2.14
```

[10]:
```
2.14
```

[11]:
```
# String

"Hello, Python 101!"
```

[11]:
```
'Hello, Python 101!'
```

[12]:
```
# Type of 12

type(12)
```

[12]:
```
int
```

[13]:
```
# Type of 2.14

type(2.14)
```

[13]:
```
float
```

[14]:
```
# Type of "Hello, Python 101!"

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

[14]:
```
str
```

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

[15]:
```
float
```

Integers

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

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

type(-1)
```

[16]:
```
int
```

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

type(4)
```

```
[17]: int
```

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

      type(0)
```

```
[18]: int
```

Floats

Floats represent real numbers; they are 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:

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

      type(1.0) # Notice that 1 is an int, and 1.0 is a float
```

```
[19]: float
```

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

      type(0.56)
```

```
[21]: float
```

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

      sys.float_info
```

```
[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:

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

      type(2)
```

```
[23]: int
```

Converting integers to floats

Let's cast integer 2 to float:

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

float(2)
```

[24]:
```
2.0
```

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

type(float(2))
```

[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):

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

int(1.1)
```

[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():

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

int('1')
```

[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:

[28]:
```
# 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
    ----> 3 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:

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

float('1.2')
```

[29]:
```
1.2
```

```
[Tip:] Note that strings can be represented with single quotes (<code>'1.2'</code>) or double
```

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?

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

str(1)
```

[30]:
```
'1'
```

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

str(1.2)
```

[31]:
```
'1.2'
```

Boolean data type

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

[32]:
```
# Value true

True
```

[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").

[33]:
```
# Value false

False
```

[33]:
```
False
```

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

[34]:
```
# Type of True

type(True)
```

[34]:
```
bool
```

[35]:
```
# Type of False

type(False)
```

[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:

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

int(True)
```

[36]:
```
1
```

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

bool(1)
```

[37]:
```
True
```

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

bool(0)
```

[38]:
```
False
```

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

float(True)
```

[39]:
```
1.0
```

Exercise: Types

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

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

[40]:
```
float
```

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

[42]:
```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
type(6//2)
```

[42]:
```
int
```

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:

[43]:
```
# Addition operation expression

43 + 60 + 16 + 41
```

[43]:
```
160
```

[44]:
```
# Subtraction operation expression

50 - 60
```

[44]:
```
-10
```

[45]:
```
# Multiplication operation expression

5 * 5
```

[45]:
```
25
```

[46]:
```
# Division operation expression

25 / 5
```

[46]: 5.0

[47]:
```
# Division operation expression

25 / 6
```

[47]: 4.166666666666667

[48]:
```
# Integer division operation expression

25 // 5
```

[48]: 5

[49]:
```
# Integer division operation expression

25 // 6
```

[49]: 4

Exercise: Expression

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

[50]:
```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
160/60
```

[50]: 2.6666666666666665

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).

[51]:
```
# Mathematical expression

30 + 2 * 60
```

[51]: 150

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

[52]:
```
# Mathematical expression

(30 + 2) * 60
```

[52]: 1920

Variables

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

```
[53]: # 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:

```
[54]: # Print out the value in variable

      x
```

```
[54]: 160
```

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

```
[55]: # Use another variable to store the result of the operation between variable␣
       ↪and value

      y = x / 60
      y
```

```
[55]: 2.6666666666666665
```

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

```
[56]: # Overwrite variable with new value

      x = x / 60
      x
```

```
[56]: 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:

```
[57]: # Name the variables meaningfully

      total_min = 43 + 42 + 57 # Total length of albums in minutes
      total_min
```

```
[57]: 142
```

```
[58]: # Name the variables meaningfully

      total_hours = total_min / 60 # Total length of albums in hours
      total_hours
```

```
[58]: 2.3666666666666667
```

```
[59]: # Complicate expression

      total_hours = (43 + 42 + 57) / 60  # Total hours in a single expression
      total_hours
```

```
[59]: 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

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

```
[60]: 7
```

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

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

```
[61]: 10
```

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

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

```
[62]: 17
```