

# More about data types

Previously, you explored data types in Python. A **data type** is a category for a particular type of data item. You focused on string, list, float, integer, and Boolean data. These are the data types you'll work with in this course. This reading will expand on these data types. It will also introduce three additional types.

## String

In Python, **string data** is data consisting of an ordered sequence of characters. Characters in a string may include letters, numbers, symbols, and spaces. These characters must be placed within quotation marks. These are all valid strings:

```
"updates needed"
```

```
"20%"
```

```
"5.0"
```

```
"35"
```

```
"**/**/**"
```

```
""
```

**Note:** The last item (""), which doesn't contain anything within the quotation marks, is called an empty string.

You can use the `print()` function to display a string. You can explore this by running this code:

```
print("updates needed")
```

The code prints

```
updates needed
```

You can place strings in either double quotation marks (") or single quotation marks ('). The following code demonstrates that the same message prints when the string is in single quotation marks:

```
print('updates needed')
```

The code prints.

```
updates needed
```

**Note:** Choosing one type of quotation marks and using it consistently makes it easier to read your code. This course uses double quotation marks.

# List

In Python, **list data** is a data structure that consists of a collection of data in sequential form. Lists elements can be of any data type, such as strings, integers, Booleans, or even other lists. The elements of a list are placed within square brackets, and each element is separated by a comma. The following lists contains elements of various data types:

```
[12, 36, 54, 1, 7]
```

```
["eraab", "arusso", "drosas"]
```

```
[True, False, True, True]
```

```
[15, "approved", True, 45.5, False]
```

```
[]
```

**Note:** The last item [], which doesn't contain anything within the brackets, is called an empty list.

You can also use the *print()* function to display a list:

```
print([12, 36, 54, 1, 7])
```

The code prints

```
[12, 36, 54, 1, 7]
```

This displays a list containing the integers *12*, *36*, *54*, *1*, and *7*.

# Integer

In Python, **integer data** is data consisting of a number that does not include a decimal point. These are all examples of integer data:

```
-100
```

```
-12
```

```
-1
```

```
0
```

```
1
```

```
20
```

```
500
```

Integers are not placed in quotation marks. You can use the *print()* function to display an integer. When you run this code, it displays

```
print(5)
```

output:

```
5
```

You can also use the *print()* function to perform mathematical operations with integers. For example, this code adds two integers:

```
print(5 + 2)
```

output

```
7
```

The result is 7. You can also subtract, multiply, or divide two integers.

# Float

**Float data** is data consisting of a number with a decimal point. All of the following are examples of float data:

```
-2.2
```

```
-1.34
```

```
0.0
```

```
0.34
```

Just like integer data, float data is not placed in quotation marks. In addition, you can also use the *print()* function to display float data or to perform mathematical calculations with float data. You can run the following code to review the result of this calculation:

```
print(1.2 + 2.8)
```

Output

```
4.0
```

The output is *4.0*.

**Note:** Dividing two integer values or two float values results in float output when you use the symbol `/`:

```
print(1/4)
```

```
print(1.0/4.0)
```

Output

```
0.25
```

```
0.25
```

The output of both calculations is the float value of *.25*.

If you want to return a whole number from a calculation, you must use the symbol `//` instead:

```
print(1//4)
print(1.0//4.0)
```

Output

```
0
0.0
```

It will round down to the nearest whole number. In the case of `print(1//4)`, the output is the integer value of `0` because using this symbol rounds down the calculation from `.25` to the nearest whole number. In the case of `print(1.0//4.0)`, the output is the float value of `0.0` because it maintains the float data type of the values in the calculation while also rounding down to the nearest whole number.

# Boolean

Boolean data is data that can only be one of two values: either *True* or *False*.

You should not place Boolean values in quotation marks. When you run the following code, it displays the Boolean value of *True*:

```
print(True)
```

Output

```
True
```

You can also return a Boolean value by comparing numbers. Because `9` is not greater than `10`, this code evaluates to *False*:

```
print(9 > 10)
```

Output

```
False
```

Additional data types

In this course, you will work with the string, list, integer, float and Boolean data types, but there are other data types. These additional data types include tuple data, dictionary data, and set data.

# Tuple

**Tuple data** is a data structure that consists of a collection of data that cannot be changed. Like lists, tuples can contain elements of varying data types.

A difference between tuple data and list data is that it is possible to change the elements in a list, but it is not possible to change the elements in a tuple. This could be useful in a cybersecurity context. For example, if software identifiers are stored in a tuple to ensure that they will not be altered, this can provide assurance that an access control list will only block the intended software.

The syntax of a tuple is also different from the syntax of a list. A tuple is placed in parentheses rather than brackets. These are all examples of the tuple data type:

```
("wjaffrey", "arutley", "dkot")
```

```
(46, 2, 13, 2, 8, 0, 0)
```

```
(True, False, True, True)
```

```
("wjaffrey", 13, True)
```

**Pro tip:** Tuples are more memory efficient than lists, so they are useful when you are working with a large quantity of data.

# Dictionary

**Dictionary data** is data that consists of one or more key-value pairs. Each key is mapped to a value. A colon (:) is placed between the key and value. Commas separate key-value pairs from other key-value pairs, and the dictionary is placed within curly brackets ({}).

Dictionaries are useful when you want to store and retrieve data in a predictable way. For example, the following dictionary maps a building name to a number. The building name is the value, and the number is the key. A colon is placed after the key.

```
{ 1: "East",
```

```
2: "West",
```

```
3: "North",
```

```
4: "South" }
```

# Set

In Python, **set data** is data that consists of an unordered collection of unique values. This means no two values in a set can be the same.

Elements in a set are always placed within curly brackets and are separated by a comma. These elements can be of any data type. This example of a set contains strings of usernames:

```
{"jlanksy", "drosas", "nmason"}
```

## Key takeaways

It's important for security analysts who program in Python to be familiar with various Python data types. The data types that you will work with in this course are string, list, integer, float and Boolean. Additional data types include tuple, dictionary, and set. Each data type has its own purpose and own syntax.

---

Revision #4

Created 9 December 2023 01:44:21 by naruzkurai

Updated 19 December 2023 03:38:35 by naruzkurai