Variables and Data Types & What They Can Do

Introduction to Python Programming

# Variables

This module begins with a look at *variables*. Variables are how values are stored and used in code. The term ‘values’ means any kind of data that will be used in your code. This could be as simple as a number, a word, or a more complicated data structure.

For an example of how variables are used, consider a simple program that reads in the first word of a text file and then writes that word to a second text file. When the program reads the word in from the first file, it needs to temporarily store that word in memory before it writes it to the second file. This value needs to be given a name so that it can be referred to and used in the code, and this is what variables allow for. Variables allow for storage of values in memory and then name them so that they can be referred to later. For example, in the script described above, first\_word might be used as the variable name.

It is important to always use good naming conventions when naming variables, and the name should be an accurate name. For example, if our code uses a variable to store a person’s age, it would be best to use a variable name such as age or person\_age, etc., and not a variable name such as x, number\_value, or persons\_height.

Now, let’s consider various data types and their uses.

# Data Types

There are different *data types* for different kinds of data as seen in the examples below.

* **Strings** (str) are used to store text
* **Integers** (int) are used to store integers
* **Floats** (float) are used to store numeric values
* **Lists** (list) are used to store a list of values
* **Booleans** (bool) are used to store a *true* or *false* value
* **Dictionaries** (dict) are used to store key-value pairs

Note that the data type determines more than just *how* data can be stored; it also determines *what can be done* with that data. For example, we can multiply *float* (numeric) variables together, but it does not make sense to multiply *strings* (text) together. We can convert a *string* to lowercase, but it does not make sense to convert an *integer* to lowercase. We can add items to a *list* data type, but we cannot add items to a *Boolean*, and so on.

Data types (also known as *data structures*, especially when discussing lists and dictionaries) are important. As programmers develop skills, they will grow in understanding of these data types and the implications of choosing one over the other. However, it’s always best to start with a grasp of the basics, so let’s take a look at seven of the most commonly used data types.

# Seven Data Types

As we consider each data type, we will look at some examples of how to create them in Python. To create a variable in Python, all we need to do is to write the name of the variable, an equals (=) sign, and then the value we want to assign that variable name to.

We will cover this more in course lectures, but for now, here are some examples:

|  |
| --- |
| my\_name = 'Will' my\_weight = 180 |

## Strings

The *string* data type is used to store text data. They can be as small as one character, such as ‘a’, or they can be as large as an entire book. Strings can be used to store various data such as names, set values, or tags (think of analyzing Instagram tags). For example, strings can be used to store the names of managers in a program application that helps schedule employees in various stores locations.

Here are some example strings:

|  |
| --- |
| cats\_name = 'Rose' street\_name = 'goldfinch ln' |

## Floats and Ints

*Floats* and *ints* are both used to store numerical data. The *int* data type is used to store integers like 5, 6, and 37. An example application of this could be to store someone’s age.

*Floats* are used to store numerical values that are NOT integers. Some examples of floats are:

|  |
| --- |
| my\_float = 10.14375 degrees\_celsius = -0.1 |

## Lists

*Lists* are used to store multiple values in a specific order. Lists can contain any kind of data, including other lists (i.e., lists of lists). Lists can be useful for storing a collection of file paths, names, or even runs scored during each inning of a baseball game. Lists are mutable which means they can have items added and removed. They can also have items updated.

Some examples of lists are:

|  |
| --- |
| my\_list = [1, 5, 6, 7] my\_friends = ['John', 'Sally', 'Mary', 'Carol'] a\_random\_list = [-5, 'Hello', [1, 'a'], 17.129] |

## Tuples

*Tuples* are very similar to lists; they are used to store a collection of values in order. The main difference is that, unlike lists which are mutable, tuples are immutable. This means they cannot be updated after they are created. Therefore, you cannot add additional items, delete items, or update items.

Some examples of tuples are:

|  |
| --- |
| coordinates = (-1, 2.3) customer\_1 = ('Jane Smith', 28, '1702 Goldcoast Ln', 'Miami', 'FL') |

## Dictionaries

*Dictionaries* are similar to lists and tuples in that they contain a collection of data. However, they do not have an order. Instead, the items are stored by a name. This name is also known as a *key*, and dictionaries are often described as a collection of *key-value* pairs.

Suppose there is a dictionary with two key-value pairs. The first key is ‘name’ and the value associated with it is ‘Jane’. The second key is ‘age’, and the value associated with it is ‘33’. Note how this is different from a list. With a list, you would access an item by its order in the list. For example, you might access the 1st or 5th item. With a dictionary, however, there is no order, which means there is no 1st or 5th item. You would access the item by its key, such as ‘name’ or ‘age’. (We will see further examples of this in the lectures.)

Dictionaries also cannot have two keys of the same name, and while keys can be more than just strings, they cannot be mutable variables like lists.

Some example dictionaries are:

|  |
| --- |
| user\_1 = {'name': 'Jose', 'occupation': 'doctor', age: 45} survey\_results = {'survey\_name': 'satisfaction', 'name\_of\_customer': 'Liko', 'satisifaction\_with\_food': 5, 'satisifcation\_with\_service': 4} |

## Sets

*Sets* are also a collection of data and are similar to lists or tuples. However, they do not have order and there cannot be duplicate entries. They are like a dictionary but with only keys and no values. This may not seem very useful. However, sets are extremely fast when determining if an item is in them.

For example, imagine there was a group of 1,000,000 user IDs for an application, and we wanted to know if any of another 10,000 users were in the group of 1,000,000 or not. This would be a slow process if the 1,000,000 users were stored in a list. However, it would be much faster using a set. Using sets in this and similar scenarios can *dramatically* improve the speed of code.

The reason sets are faster is that the location in memory where a value in the set will be stored is determined by the value itself. Consider a set with the value ‘Hello’ in it. ‘Hello’ will be stored in only one location in memory. This is why there cannot be duplicate values, as they both would be stored in the exact same location and would overwrite each other. Now, because ‘Hello’ is stored in an exact location, if we need to ask if ‘Hello’ is in the set, the computer does not need to search through each value. The computer knows exactly where to look for ‘Hello’ because it can only be in one place. Lists are different, and because they have an order, the computer doesn’t know where ‘Hello’ is. It could be at any of the spots in the list. In the worst-case scenario, the computer would need to search through all of the items in the list to find ‘Hello’. Also, sets, like lists, cannot contain variables that are mutable.

Here are some examples of sets:

|  |
| --- |
| user\_ids = {'b69dth3n', '98ddfhn3', '9dj3n48f', '0p4nd72m'} fruits = {'apple', 'banana', 'pear'} my\_dictionary = {5, 9, 10, 11,'hello'} |

## Booleans

Booleans are very simple. A Boolean is simply the value *true* or *false*. Booleans can be created directly by indicating if the value is true or false, or they can be the result of comparison operators. We will see more comparison operations in the course lectures.

Below are some examples of Booleans:

|  |
| --- |
| my\_boolean\_1 = **True** my\_boolean\_2 = 5 < 2 |