HyperionDev

Variables – Storing Data in Programs

Task

Visit our website

Introduction

Now that you have completed your first Python program, let's consider variables in a little more depth. To be able to perform calculations and instructions, we need a place to store values in the computer's memory. This is where variables come in. A variable is a way to store information. It can be thought of as a type of container that holds information.

By the end of this lesson, you will understand how to declare and use variables, allowing you to solve more complex programming problems. You will also learn how to handle different types of variables and convert between them as needed.

Declaring variables

Variables in programming work the same as variables in mathematics. We use them in calculations to hold values that can be changed. In maths, variables are named using letters, like x and y. In programming, you can name variables whatever you like, as long as you do not pick something that is a keyword (also known as a "reserved" word) in the programming language. It is best to name them something useful and meaningful to the program or calculation you are working on. For example, num_learners could contain the number of learners in a class, or total_amount could store the total value of a calculation.

In Python, we use the following format to create a variable and assign a value to it:

```
variable_name = value_you_want_to_store
```

For example:

```
num = 2
```

In the code above, the variable named num is assigned the integer (or whole number) 2. Hereafter, when you type the "word" num, the program will refer to the appropriate space in memory where the variable is stored, and retrieve the value 2 that is stored there.

We use variables to hold values that can be changed (can vary). You can name a variable anything you like as long as you follow the rules shown below. However, as previously stated, giving your variables meaningful names is good practice.

Below is an example of bad naming conventions vs good naming conventions.

- `my_name = "Tom"` `# Good variable name`
- `variableOne = "Tom"` `# Bad variable name`
- `string_name = "Tom"` `# Good variable name`
- `h4x0r = "Tom"` `# Bad variable name`

Here, `my_name` and `string_name` are examples of descriptive variables as they reveal what they are and what content they store, whereas `variableOne` and `h4x0r` are terrible names because they are not descriptive.

Variable naming rules

Let's think a bit more about how to name variables. As previously mentioned, it is very important to give variables descriptive names that reference the value being stored. Here are the naming rules in Python (these can differ in other programming languages):

1. Variable names must start with a letter or an underscore.
2. The remainder of the variable name can consist of letters, numbers, and underscores.
3. Variable names are case sensitive so "Number" and "number" are each different variable names.
4. You cannot use a Python keyword (reserved word) as a variable name. A reserved word has a fixed meaning and cannot be redefined by the programmer. For example, you would not be allowed to name a variable "import" since Python already recognises this as a keyword. The same is true of the keyword "if".

Variable naming style guide

The way you write variable names will vary depending on the programming language you are using. For example, the **Java** style guide recommends the use of camel case – where the first letter is lowercase, but each subsequent word is capitalised with no spaces in between (e.g., "thisIsAGoodExampleOfCamelCase").

The style guide provided for **Python** code, **PEP 8**, recommends the use of snake case – all lowercase with underscores in between instead of spaces (e.g. "this_is_a_good_example_of_snake_case"). You should use this type of variable naming for your Python tasks.

In maths, variables only deal with numbers, but in programming we have many different types of variables and many different types of data. Each variable data type is specifically created to deal with a specific type of information.

Variable data types

There are five major types of data that variables can store. These are **strings**, **chars**, **integers**, **floats**, and **Booleans**.

- **string**: A string consists of a combination of characters. For example, it can be used to store the surname, name, or address of a person. It can also store numbers, but when numbers are stored in a string you cannot use them for calculations without changing their data type to one of the types intended for numbers.
- **char**: Short for character. A char is a single letter, number, punctuation mark, or any other special character. This variable type can be used for storing data like the grade symbol (A–F) of a pupil. Moreover, strings can be thought of (and treated by functions) as lists of chars in situations in which this approach is useful.
- **integer**: An integer is a whole number, or number without a decimal or fractional part. This variable type can be used to store data like the number of items you would like to purchase, or the number of students in a class.
- **float**: We make use of the float data type when working with numbers that contain decimals or fractional parts. This variable type can be used to store data like measurements or monetary amounts.
- **Boolean**: Can only store one of two values, namely TRUE or FALSE.

The situation you are faced with will determine which variable type you need to use. For example, when dealing with money or mathematical calculations you would likely use **integers** or **floats**. When dealing with sentences or displaying instructions to the user you would make use of **strings**. You could also use **strings** to store data like telephone numbers that are numerical but will not be used for calculations. When dealing with decisions that have only two possible outcomes you would use **Booleans**, as the scenario could only either be True or False.

Variables store data and the type of data that is stored by a variable is intuitively called the data type. In Python, we do not have to declare the data type of the variable when we declare the variable (unlike certain other languages). This is known as “weak typing” and makes working with variables easier for beginners.

Python detects the variable's data type by reading how data is assigned to the variable, as follows:

- Strings are detected by quotation marks " ".
- Integers are detected by the lack of quotation marks and the presence of digits or other whole numbers.
- Floats are detected by the presence of decimal point numbers.
- Booleans are detected by being assigned a value of either True or False.

So, if you enter numbers, Python will automatically know you are using integers or floats. If you enter a sentence, Python will detect that it is storing a string. If you want to store something like a telephone number as a string, you can indicate this to Python by putting it in quotation marks, e.g., phone_num = “082 000 0000”.

You need to take care when setting a string with numerical information. For example, consider this:

```
number_str = "10"
print(number_str * 2) # Prints 1010, i.e., prints number_str twice
print(int(number_str) * 2) # Prints 20 because the string 10 is cast to number 10
```

Watch out here! Since you defined 10 within quotation marks, Python figures this is a string. It's not stored as an integer even though 10 is a number, as numbers can also be made into a string if you put them between quotation marks. Now, because 10 is declared as a string here, we will be unable to do any arithmetic calculations with it – the program treats it as if the numbers are letters. In the above example, when we ask Python to print the string times 2, it helpfully prints the string twice. If we want to print the value of the number 10 times 2, we have to cast the string variable to an integer (convert it) by writing int(number_string). Take heed that all variable types can be converted from one to another, not just ints and strings.

There is also a way that you can determine what data type a variable is using the type() built-in function. For example:

```
# Assigning a string "10" to the variable mystery_1
mystery_1 = "10"

# Assigning a floating-point number 10.6 to the variable mystery_2
mystery_2 = 10.6

# Assigning a string "ten" to the variable mystery_3
mystery_3 = "ten"

# Assigning the Boolean value True to the variable mystery_4
mystery_4 = True

# Use the print() and type() functions to check the data types.
print(type(mystery_1))
print(type(mystery_2))
print(type(mystery_3))
print(type(mystery_4))
```

Output:

```
<class 'str'>
<class 'float'>
<class 'str'>
<class 'bool'>
```

The output shows us the data type of each variable in the inverted commas.

We can also separate large integer numbers with an underscore '_' for ease of reading.

Input:

```
# Separate large numbers with underscores
number = 15_000_300
new_number = number + 300

# Print new number and its type
print(new_number)
print(type(number))
```

Output:

```
15000600
<class 'int'>
```

Casting

Let's return to the concept of changing variable types from one to another. In the string-printing example above, you saw something we called "casting". Casting means taking a variable of one particular data type and "turning it into" another data type. Putting the 10 in quotation marks will automatically convert it into a string, but there is a more formal way to change between variable types. This is known as casting or type conversion.

Casting in Python is pretty simple to do. All you need to know is which data type you want to convert to and then use the corresponding function.

- **str()** – converts a variable to a string
- **int()** – converts a variable to an integer
- **float()** – converts a variable to a float

```
# Assigning the integer value 30 to the variable number
number = 30

# Assigning the string "10" to the variable number_str
number_str = "10"

# Printing the result of adding the integer value of number_str to number
print(number + int(number_str)) # Prints 40
```

This example converts **number_str** into an integer so that we can add two integers together and print the total. We cannot add a string and an integer together. (Are you curious what would happen if you didn't cast **number_str** to a string? Try copying the code in the block above, pasting it into VS Code, and running it. What happens?)

You can also convert the variable type entered via input(). **By default, anything entered into an input() is a string**. To convert the input to a different data type, simply use the desired casting function.

```
# Prompting the user to input the number of days worked this month and converting it to an integer
num_days = int(input("How many days did you work this month?"))

# Prompting the user to input the pay per day and converting it to a float
pay_per_day = float(input("How much is your pay per day?"))

# Calculating the salary by multiplying the number of days worked by the pay per day
salary = num_days * pay_per_day

# Printing the calculated salary with a formatted string
print("My salary for the month is USD:{}".format(salary))
# Explanation provided below
```

When writing programs, you will have to decide what variables you will need.

Take note of what is in the brackets on line four above. When working with strings, we are able to put variables into our strings with the format method. To do this, we use curly braces `{ }` as placeholders for our values. Then, after the string, we put `.format(variable_name)`. When the code runs, the curly braces will be replaced by the value in the variable specified in the brackets after the format method.

Working with the f-string

The f-string is another approach to including variables in strings. The syntax for working with the f-string is quite similar to what is shown above in the format method example.

Notice that we declare the variables upfront and we do not need to tag on the format method at the end of our string the way we did for the format method above. Also, note the **f** at the beginning of the string:

```
num_days = 28
pay_per_day = 50

# Use f-strings to format the strings
print(f"I worked {num_days} days this month. I earned ${pay_per_day} per day.")
```

Output:

```
“I worked 28 days this month. I earned $50 per day.”
```

f-strings provide a less verbose way of interpolating (inserting) values inside string literals. You can **read more about f-strings** in the Python documentation.

If you wanted to use the str.format() method in the same scenario as the f-string example, you could do so as follows:

Example 1: Using .format() with positional arguments

```
# The {0} will be replaced by the first argument (22), and {1} by the second (50)
print("You worked {0} this month and earned ${1} per day".format(22, 50))
```

Example 2: Using .format() with keyword arguments

```
# The {num_days} and {pay_per_day} are replaced by values passed with matching keyword names
print("You worked {num_days} this month and earned ${pay_per_day} per day".format(num_days=22, pay_per_day=50))
```

What do you think the advantage might be of using index references/ positional arguments?

What's next?

This lesson is continued in the example file (**example_variables.py**) provided in this task folder. Open this file using VS Code. The context and explanations provided in the examples should help you better understand some simple basics of Python.

You may run the examples to see the output. The instructions on how to do this are inside each example file. Feel free to write and run your own example code before attempting the task to become more comfortable with Python.

Try to write comments in your code to explain what you are doing in your program (read the example files for more information, and to see examples of how to write comments).

Now it is time to try your hand at a few practical tasks.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

After you click "Request review" on your student dashboard, you will receive a 100% pass grade if you've submitted the task.

When you submit the task, you will receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer. Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission. Please take a moment to complete the survey.

Once you've done that, feel free to progress to the next task.

Auto-graded task 1

Follow these steps:

1. Create a new Python file in the folder for this task, and call it **details.py**.
2. Use an `input()` command to get the following information from the user.
    - Name
    - Age
    - House number
    - Street name
3. Print out a single sentence containing all the details of the user.
4. For example:

```
This is John Smith. He is 28 years old and lives at house number
42 on Hamilton Street.
```

Auto-graded task 2

Follow these steps:

1. Create a new Python file in this folder called **conversion.py**.
2. Declare the following variables:
    - `num1 = 99.23`
    - `num2 = 23`
    - `num3 = 150`
    - `string1 = “100”`
3. Convert them as follows:
    - `num1` into an integer
    - `num2` into a float
    - `num3` into a String
    - `string1` into an integer
4. Print out all the variables on separate lines.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.
