TASK

Variables — Storing Data in Programs

Visit our website

Introduction

WELCOME TO THE VARIABLES TASK!

Now that you have completed your first Python program, this task will introduce you to the concept of variables and more complex programming problems that can be solved using them. *Variable* is a computer programming term that is used to refer to the storage locations for data in a program. Each variable has a name which can be used to refer to some stored information known as a *value*. By completing this task you will gain an understanding of variables and how to declare and assign values to them, as well as the different types of variables and how to convert between types.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to log in to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

A note from the Hyperion Team

Now that you are a little more familiar with Python and creating basic programs, we would like to show you some stuff to help you on your journey to becoming a seasoned programmer.

Creating excellent content requires good tools and equipment. This applies equally well to programming. There are some great tools and resources available online that you can start using as soon as possible, if you have not already, to make the coding process just that much more convenient. **Here** is a link to the Hyperion Blog where you will find essential utilities & resources for programmers.

WHAT ARE VARIABLES?

To be able to perform these 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.

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 don't pick something that is a keyword 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 code to create a variable and assign a value to it:

```
variable_name = value_you_want_to_store
```

Check out this 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 and retrieve the value 2 that is stored there.

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

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

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

A note from our coding mentor
Jared

Variable Naming Rules

It is very important to give variables descriptive names that reference the value being stored. Here are the naming rules:

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* print *since Python already recognises this as a keyword.*

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 each variable type deals with a specific set 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.

- **char:** short for character. A char is a single letter, number, punctuation mark or any other special character. It can be used for storing the grade symbol (A-F) of a pupil, for example.

- **integer:** an integer is a whole number, or number without a decimal or fractional part. For example, it can be used to store the number of items you would like to purchase, or the number of students in a class.

- **float:** we make use of a Float data type when working with numbers that contain decimals. For example, it can be used to store 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 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 we would make use of **strings**. When dealing with decisions that have only two possible outcomes you would use **booleans**, as the scenario would 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. This is known as "weak-typing". This is because Python detects the variable's data type by reading how data is assigned to the variable:

- 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.

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.

Take heed that types can be converted from one to another. 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- prints string 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 by writing **`int(number_string)`**. We'll explain this further in the next section!

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

```
mystery_1 = "10"
mystery_2 = 10.6
```

```
mystery_3 = "ten"
mystery_4 = True

print(type(mystery_1))
print(type(mystery_2))
print(type(mystery_3))
print(type(mystery_4))
```

Output:

```
<class 'str'>
<class 'int'>
<class 'str'>
<class 'bool'>
```

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

CASTING

In the string printing example above, you saw something we called *casting*. Casting basically 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 it to convert to and then use the corresponding function.

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

```
number = 30
number_str = "10"
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.

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

```
num_days = int(input("How many days did you work this month?"))
pay_per_day = float(input("How much is your pay per day?"))
salary = num_days * pay_per_day
print("My salary for the month is USD:{}".format(salary))
```

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

Take note of what is in the brackets on line 4 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/brackets will be replaced by the value in the variable specified in the brackets after the *format* method. You will learn more about this in the next task. Let's briefly turn our attention to the benefits of using the f-string in comparison to the format method.

Working with the f-string

The syntax for working with the f-string is quite similar to what is shown above in the format method. Notice that we declare the variables upfront and we don't need to tag on the .format method at the end of our string. Also note the ***f*** at the beginning of the string:

```
num_days = 28
pay_per_day = 50
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 values inside string literals. If you'd like to learn a little more about f-strings, you can **read more about them here**.

If you wanted to use the str.format() method, you could do so as follows:

Example 1: insert values using index references

```
print("You worked {0} this month and earned ${1} per day".format(num_days = 22,
pay_per_day = 50))
```

Example 2: insert values using empty placeholders

```
print("You worked {} this month and earned ${} per day".format(num_days = 22,
pay_per_day = 50))
```

Hey there, have you heard about Alan Turing?

Alan Turing (1912 – 1954) was a British mathematician, logician and cryptographer. He is considered by many to be the father of modern computer science. He designed and built some of the earliest electronic, programmable, digital computers.

Father of modern day computing

During the Second World War, Alan Turing was recruited by the military to head a classified mission at Bletchley Park. This mission was to crack the Nazi's Enigma machine code which was used to send secret military messages. Many historians believe that breaking the Enigma code was key to bringing the war to an end in Europe. Turing published a paper in 1936 that is now recognised as the foundation of computer science.

- ***Masood Gool,*** *Mentor*

Instructions

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

You may run **example.py** to see the output. The instructions on how to do this are inside the 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.py** file for more information).

You are not required to read the entirety of **Additional Reading.pdf**. It is purely for extra reference. That said, don't simply disregard it!

Compulsory Task 1

Follow these steps:

- Create a new Python file in this folder called **variables.py**
- Inside it, write Python code declaring and assigning values for each variable type.
- Please first provide pseudo code as comments.
- Print each variable on separate lines.

Compulsory Task 2

Follow these steps:

- Create a new Python file in this folder called **details.py**
- Use an *input* command to get the following information from the user.
    - Name
    - Age
    - House number
    - Street name
- Print out a single sentence containing all the details of the user.
- For example:
    - This is John Smith he is 28 years old and lives at house number 42 on Hamilton Street.

Compulsory Task 3

Follow these steps:

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

Optional Bonus Task

Follow these steps:

- Create a new Python file in this folder called **optional_task.py**
- Use input to get any two numbers from the user.
- Store these numbers in variables called *num1* and *num2*.
- Now swap these two numbers around. The number stored in *num1* should now be stored in *num2*, and the number stored in *num2* should now be stored in *num1*.
- Print out the values of *num1* and *num2* before the swap, and the values of *num1* and *num2* after the swap.

Thing(s) to look out for:

1. Make sure that you have installed and set up all programs correctly. You have set up **Dropbox** correctly if you are reading this, but **Python or your chosen editor** may not be installed correctly.
2. If you are not using Windows, please ask one of our expert code reviewers for alternative instructions.

Completed the task(s)?

Ask an expert code reviewer to review your work!

Review work

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.