HyperionDev

Your First Computer Program and Using Variables

Task

Visit our website

Introduction

In this task, we will start with the basics of programming using Python, one of the most popular and accessible programming languages today. Python is used by developers worldwide for various applications, from simple scripts to complex machine-learning algorithms. Its simple syntax and readability make it an ideal language for beginners.

This lesson will guide you through setting up your programming environment, writing your first Python scripts, and running them to see the immediate results of your code. We'll introduce you to fundamental concepts like variables – essential tools for storing and manipulating data in your programs. By the end of this task, 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.

What is programming?

Programmers write code statements to create programs. Programs are executable files that perform the instructions given by the programmer.

Code can be written in different programming languages, such as Python, Java, JavaScript, and C++. In this course, you will start by learning Python.

Each coding language has a different file type. For example, after writing Python code you need to save it in a Python file. A Python file has the following file naming format:

filename.**py**

The filename can be any valid filename and **.py** is the file extension.

You can then “run” the Python file. In this process, the Python program you have written is executed and displays the outcomes that may result based on what the code statements say.

There is more information about how to run Python files in the example file (**example_variable.py**) that accompanies this task. But **before we get there**, we need to talk about why we’re using Python, show you how to write basic code in Python, and perform some basic operations.

Why Python?

Python is a widely used programming language. It is consistently ranked in the top 10 most popular programming languages as measured by the **TIOBE programming community index**. Many familiar organisations make use of Python, such as Wikipedia, Google, Yahoo!, the National Aeronautics and Space Administration (NASA), and Reddit (which is written entirely in Python).

Python is a high-level programming language, similar to other popular languages such as Java, C#, and Ruby. High-level programming languages are closer to human languages than machine code. They're called "high level" as they are several steps removed from the actual code that runs on a computer's processor.

Python is hot!

The demand for Python programmers is only growing. Python boasts the highest year-on-year increase in terms of demand by employers (as reflected in job descriptions online) as well as popularity among developers. Python developers are also one of the highest-paid categories of programmers! The demand for Python is only set to grow further with its extensive use in analytics, data science, and machine learning.

In comparison with Java, Python is a more recent, efficient, and arguably faster programming language. The syntax (the way the code is written) is, nonetheless, very similar to Java. Here are a few more reasons to use Python:

- **Simple, yet powerful:** looking at languages like C++ and Java can flummox and scare the beginner, but Python is intuitive, with a user-friendly way of presenting code. Python's conciseness and economy of language allow for speedy development and less hassle over useful tasks. This makes Python easy on the eyes and mind.

- **From child's play to big business:** while Python is simple enough to be learned quickly (even by kids), it is also powerful enough to drive many big businesses. Python is used by some of the biggest tech firms such as Google, Yahoo!, Instagram, Spotify, and Dropbox, which should speak volumes about the job opportunities out there for Python developers.

- **Python is on the web:** Python is a very appealing language of choice for web development. Sites such as Pinterest and Instagram make use of the versatility, rapidity, and simplicity of Django (a web development framework written in Python).

- **Dropbox was built using Python:** Dropbox must save massive quantities of files while supporting a similarly massive degree of user growth. Did you know that 99.9% of Dropbox code is written in Python? Using Python has helped Dropbox gain more than a hundred million users. With only a few hundred lines of Python code, they were able to scale their user numbers dramatically, proving the utility of the language!

How do you write code?

Before you get started, we suggest you install Visual Studio Code (VS Code) and use it as your integrated development environment (IDE) to open all text files (.txt) and Python files (.py).

Setting up VS Code

Please read the following instructions in full before following them:

1. Visit **Visual Studio Code**.

2. Download the version of VS Code that matches your operating system (OS). Alternatively, you can follow the instructions stated in the following links for the corresponding operating system families:

   - macOS: **https://code.visualstudio.com/docs/setup/mac**
   - Linux: **https://code.visualstudio.com/docs/setup/linux**
   - Windows: **https://code.visualstudio.com/docs/setup/windows**

   Note: Unix-like operating systems such as macOS and Linux often come with a preinstalled version of Python. It is generally discouraged to use the distributions of Python that are shipped with macOS as they may either be outdated or have customisations that might give you issues further down the line.

3. Once you have VS Code installed, you will need to install the Python extension for VS Code. Visit the **Python tutorial** page and follow the link provided in the guidelines to install the **Python extension for VS Code**.

   Ensure that you are directed to install the latest stable version of Microsoft's Python extension available at the **VS extension marketplace** so that you get tooltips and other useful tooling to help you as you program. When you select to install the extension, a window will open in VS Code where you will be prompted to complete the installation.

   You can double-check that the Python extension has been installed in VS Code by navigating to the panel on the left-hand side and clicking on "Extensions".

4. Once the extension is installed for VS Code, revert back to the **Python tutorial** page and follow the guidelines to **install a Python interpreter** for your OS.

   In order to select a Python interpreter, a workspace folder needs to be opened in VS Code. While the tutorials page provides instructions on how to do this in the command terminal, we've provided simpler steps below:

   - Create an empty folder on your PC.
   - In VS, use File > Open Folder to open the project folder. Then use File > New File and select "Python file" from the dropdown options.
   - Now that your workspace is set up, you can follow the instructions to **select a Python interpreter**.

5. Continue to follow the guidelines in the tutorial to **verify the Python installation** and **configure and run the debugger**.

Take note

Ensure that, if you are on Windows or macOS, you have installed the latest stable version of Python using the prescribed means above. If the installation was successful it will output the version number.

Ensure that, if you are on Linux and the prepackaged Python version is not the latest stable version, you get a package provider for your operating system that uses the latest stable version. Being behind by one or two minor versions is fine on operating systems such as Fedora. However, on operating systems such as Ubuntu, we strongly recommend that you use a PPA (Personal Package Archive) such as **deadsnakes**.

Note: Please avoid the Flatpak or Snap versions as they give you troubleshooting problems. Only proceed with flatpaks if you are sure of how they work.

For all operating systems, ensure that your environment paths are up to date for your installation.

6. Once you have installed VS Code, visit the **Python overview** page to learn how to use VS Code with Python. If you've never programmed before, we strongly recommend that you watch the **introductory videos**.

7. If you encounter any problems with the Python extension installation please view the "Common Questions" available on each of the corresponding OS setup pages, or visit the following links:

   - **Setup**
   - **Python extension discussions Q&A**
   - **Python documentation on the VS Code site**
   - **VS Code on Stack Overflow**

   If you encounter any problems with the Python software installation please visit:

   - **Windows help**
   - **Mac help**
   - **Unix help**

   If the problem persists, please reach out to our support team for further assistance.

8. Opt-out telemetry in VS Code refers to the practice of automatically collecting usage data from users of the software, and sharing it with the VS Code developers to help them identify bugs, improve performance, and make other changes that will benefit users. If you have concerns about your privacy, data, and usage habits, please **turn this feature off using the instructions available here**.

Now that you have your development environment set up, you are ready to start coding!

Learning the rules

All programming languages have syntax rules. Syntax is the "spelling and grammar setup" of a programming language and determines how you write correct, well-formed code statements. If you make a mistake by breaking the “spelling and grammar” rules for coding in Python, this is called a syntax error.

Consider the following Python code. We’ll get into more detail about what it means later, but for now, it’s enough to know that this code will output "Hello World".

```
print("Hello, World!")
```

A common syntax error you could make in this code would be forgetting to add a closing quotation mark (") at the end of the “Hello World”. The text inside the brackets is a string. With strings, all opening quotation marks (") require a closing one – the opening one shows the string is starting, and the closing one shows where it ends.

Another common syntax error that you could make in the example above is forgetting to add a closing bracket “)”. Remember that all opening brackets “(“ require a matching closing one, “)”!

Any program you write must be exactly correct. All code is case-sensitive. This means that “Print” is not the same as “print”. If you enter an invalid Python command, misspell a command, or misplace a punctuation mark, you will get a syntax error when trying to run your Python program.

Errors appear in the Python shell when you try to run a program and it fails. Be sure to read all errors carefully to discover what the problem is. Error reports in the Python shell will even tell you what line of your program had an error. The process of resolving errors in code is known as debugging.

Creating output

You may want your program to display or output information to the user. The most common way to view program output is to use the print function. To use print, we enter the print command followed by one or more arguments in brackets. Let's first take a moment to understand what arguments are, as well as two other new terms, parameters and variables.

A **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 that can be used to refer to some stored information known as a **value**.

Functions in Python are blocks of code that perform one or more specific task. A **parameter** is a variable in a function definition that tells you the sort of data you will need to give the function to work with when it runs. When a function is actually called (in other words, the programmer instructs the block of code making up the function to run), the **arguments** are the data you pass into the function's parameters (i.e., the actual data you give to the function to work on when it runs).

Now that you understand these basic concepts, let's review how print works. You've seen the following code before:

```python
# This code will output "Hello, World!"
print("Hello, World!")
```

Code hack

Did you notice the code above included some text that describes what the code does? This line is a comment.

Comments are useful because they help explain what the code does, why certain choices were made, and they can suggest what needs to be improved. This is especially helpful when working with others or when you need to understand your own code after some time. Essentially, comments make it easier for anyone (including your future self) to understand and work with the code, making them a very important part of programming.

In Python, comments are used to write notes right inside the code. To make comments in Python you start the line with the # symbol. This tells Python to ignore that line when running the program.

We enter the print command into a Python file followed by one or more arguments in brackets. In programming, a **command** is an instruction given by a user telling a computer to do something. Together, a command and an argument are known as a statement.

```python
# This code will output "Hello, World!"
print("Hello, World!")
```

When you run the above program, the computer will output the argument “Hello, World!” that was passed into the input parameter. Note that the argument is enclosed in double quotes ("..."). This is because "Hello, World!" is a type of variable called a **string**, i.e., a list of characters. You'll learn more about strings and other variable types later in this task.

Note that the Python shell (the window that is displayed when you run a Python program) only shows the output of the program. Other statements in your code that don't create output will be executed but not displayed in the Python shell.

Getting input

Sometimes you want a user to enter data through the keyboard that will be used by your program. To do this, you use the input command.

When the program runs, the input command, which can be seen in the example below, will show the text "Enter your name: " in the output box of the program. The program will then halt until the user enters something with their keyboard and presses enter.

```python
# Prompt the user to input their name
name = input("Enter your name: ")
# Store the user's input in the “name” variable
```

The variable name stores what the user entered into the box as a string. Storing and declaring variables doesn't produce any output.

Declaring variables

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. In the example above, the input command was used to tell the computer to take whatever the user typed before they pressed enter and place that in a container as a variable of the **string** type.

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 (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 keywords "if" and "else".

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 **char**acter. 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'>
```

Extra resource

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. There are a host of **free resources** on the HyperionDev Blog, including essential utilities and resources for programmers.

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

```python
# 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.

```python
# 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'll 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 don’t 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 here**.

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: insert values using index references

```
# Printing a formatted string with placeholders for variables(using indexes)
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

```
# Printing a formatted string with placeholders for variables(using placeholders)
print("You worked {} this month and earned ${} per day".format(num_days = 22,
pay_per_day = 50))
```

What do you think the advantage might be of using index reference

Extra resource

The ‘Zen of Python’, written in 1999 by Tim Peters, mentions all the software principles that influence the design of the Python language.

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one — and preferably only one — obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
*Although never is often better than *right* now.*
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea — let's do more of those!

What’s next?

This lesson is continued in the example files (**example_first_program.py** and **example_variables.py**) provided in this task folder. Open these files 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).

You are not required to read the entirety of the **additional reading**. It is purely for extra reference. That said, do take a look – you could find it very useful!

Now it’s 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 the task 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 for this task, which you can use as a self-assessment tool. 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:

- Create a new Python file in the folder for this task, and call it **hello_world.py**.
- First, provide pseudocode as comments in your Python file, outlining how you will solve this problem (you'll need to read the rest of this practical task first of course!).
.
- Now, inside your **hello_world.py** file, write Python code to take in a user's name using the input() function and then print out the name.
- Use the same input() function and output approach to take in a user's age and print it out.
- Finally, print the string "Hello, World!" on a new line (the new line will happen by default if you use a separate print statement to the one you used immediately above to print out the age, because each print statement automatically inserts an "enter", or newline instruction, at the end).

Auto-graded task 2

Follow these steps:

- Create a new Python file in the folder for this task, and call it **details.py**.
- As in task 1, please first provide pseudocode as comments in your Python file, outlining how you will solve this problem.
- 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.
```

Auto-graded task 3

Follow these steps:

- Create a new Python file in this folder called **conversion.py**.
- As in the previous practical tasks, please first provide pseudocode as comments in your Python file, outlining how you will solve this problem.
- 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.

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

Take note

Make sure that you have installed and set up Python and your editor correctly.

If you are not using Windows, please ask one of our mentors for alternative instructions.

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.