



TASK

Getting Started With Your Bootcamp

Visit our website

WELCOME TO THE GETTING STARTED WITH YOUR BOOTCAMP TASK!

Well done on making one of the most important decisions of your life. Moving into a tech career can at times feel scary, but we are here to walk you through your programming journey.

We will lay the foundation for your success by setting up your development environment and introducing you to core programming concepts that will serve you for the rest of your life.

FIRST SET UP YOUR DEVELOPMENT ENVIRONMENT

Before we get going, please consult the '**Additional Reading – Installation, Sharing, and Collaboration**' guide that accompanies this task. This reading will provide you with detailed instructions on setting up Visual Studio Code and Python on macOS, Windows, and Linux. It includes steps for downloading, installing, and updating environment paths, and installing Microsoft's Python extension. It also offers alternatives for unsupported operating systems and solutions for common troubleshooting issues.

Please note: It is crucial to consult this reading and set up your development environment before beginning any practical coding components later on.

THE START OF YOUR FIRST COMPUTER PROGRAM

Let's learn how to create your first computer program by first exploring the concept of pseudocode. Although pseudocode is not a formal programming language, it will ease your transition into the world of programming. Pseudocode makes creating programs easier because, as you may have guessed, programs can sometimes be complex and long, so preparation is key. It is difficult to find errors without understanding the complete flow of a program, and writing pseudocode helps you to establish this high-level understanding.

You will also be introduced to the Python programming language. Python is a widely used programming language, and 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!, NASA, and Reddit (which is primarily written in Python).

Python is a high-level programming language, along with 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.

This task is a gentle introduction to Python, where you will be asked to create a simple program. In doing so, you will become familiar with the structure of a Python program.

Lastly, you will also be introduced to the concept of variables and the more complex programming problems that can be solved using these. 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. 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.

INTRODUCTION TO PSEUDOCODE

What is pseudocode? And why do you need to know this? Well, being a programmer means that you will often have to visualise a problem and know how to implement the steps to solve a particular conundrum. This process is known as writing pseudocode.

Pseudocode is a simplified, human-readable description of a program or algorithm, designed for understanding rather than execution. It doesn't follow strict syntax rules, making it accessible to anyone, regardless of their programming language expertise. Pseudocode helps programmers conceptualise algorithms and plan their implementation in any programming language. It's also useful for collaborating on program design with team members and clients before actual coding begins.

WHAT IS AN ALGORITHM?

Simply put, an algorithm is a step-by-step method of solving a problem. To understand this better, it might help to consider an example that is not algorithmic. When you learned to multiply single-digit numbers, you probably memorised the multiplication table for each number (say n) all the way up to $n \times 10$. In effect, you memorised 100 specific solutions. That kind of knowledge is not algorithmic. But along the way, you probably recognised a pattern and made up a few tricks.

For example, to find the product of n and 9 (when n is less than 10), you can write $n - 1$ as the first digit and $10 - n$ as the second digit (e.g., $7 \times 9 = 63$). This trick is a general solution for multiplying any single-digit number by 9. That's an algorithm! Similarly, the techniques you learned for addition with carrying, subtraction with borrowing, and long division are all algorithms.

Algorithms don't require intelligence to execute; they follow a set of rules mechanically. However, creating an algorithm involves breaking down complex problems into logical steps, which requires computational thinking. Typically, an algorithm has an input and an output.

Now, let's see an example of writing a password validation algorithm in pseudocode.



Problem

Write an algorithm that asks a user to input a password, and then stores the password in a variable (something you will learn more about later in this lesson) called **password**. For now, just think of a variable as

a name for a container to store some information in. Once the user's chosen password has been stored, the algorithm must request input from the user. If the input does not match the password the user originally set, the algorithm must store the incorrect passwords in a list until the correct password is entered. At that point, it must print out the value of the variable **password** (i.e., the user's chosen password that has been stored in the password variable/container), as well as the incorrect passwords.

Pseudocode solution:

```
request input from the user
store input into variable called "password"
request second input from the user
if the second input is equal to "password"
    output the "password" and the incorrect inputs (which should be none
    at this point)
if the second input is not equal to "password"
    repeatedly request input until input matches "password"
    store the non-matching input for later output
when the input matches "password"
    output "password" and output all incorrect inputs.
```

In the practical tasks at the end of this lesson, you will practice creating high-level solutions to simple problems using pseudocode, as seen above.

INTRODUCTION TO PYTHON

Python is a powerful, widely used programming language known for its simplicity and ease of use. Its syntax (the way the code is written) is designed to be readable, which can make it easier for beginners to learn and use.

Here are a few more reasons to use Python:

- **Simple, yet powerful:** Looking at languages like C++ and Java can confuse and scare the beginner, but Python is intuitive, with a user-friendly way of presenting code. Python's succinctness 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).



Know this – 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 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.



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. 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. Information about how to run Python files is given in the example file (**example_first_program.py**) that accompanies this task. We will now show you how to write some basic code in Python and perform some basic operations.

THE PRINT() FUNCTION

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 parentheses. Let’s first take a moment to understand what arguments are, as well as two other terms: **variables** and **parameters**.

Remember from earlier in this task that 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 and methods in Python are blocks of code designed to perform one or more specific tasks. 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). The same is true of a method.

Now that you understand these basic concepts, let's review how **print** works. We enter the **print** command into a Python file followed by one or more arguments in parentheses. 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**. Consider the Python statement below:

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

When you run this 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.

SYNTAX RULES

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

A common syntax error you could make in the **print** statement we looked at above is forgetting to add a closing quotation mark at the end of the string argument ("). In 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 **print** example above is forgetting to add a closing parenthesis "). Remember that all opening parentheses "(" require a matching closing one, ")".

Any program you write must be exactly correct, as 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.

HOW TO GET INPUT

Sometimes you want a user to enter data that will be used by your program through the keyboard. 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.

```
name = input("Enter your name: ")
```

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

WHAT ARE 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 to 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
```


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 out **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 in the next section. However, as previously stated, giving your variables meaningful names is good practice.

Below are examples of bad naming conventions compared to 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.



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 something to help you on your journey to becoming a seasoned programmer.

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. Head to the [HyperionDev Blog](#) where you will find essential utilities and resources for programmers.



RULES FOR NAMING 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, as 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. The same is true of the keyword **input**.

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.

VARIABLE DATA TYPES

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.

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

Integer: `42` (used to store whole numbers)

Float: `3.14` (used to store numbers with decimals)

Boolean: **True** or **False** (used to store logical values)

- **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 of a pupil (A–F). 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 be either 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 = "082 123 4567"`.

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, which is the string printed twice.
print(int(number_str) * 2) # Prints 20 because the string 10 is cast to
number 10
```

Watch out here! When you enclose the number **10** in quotation marks, Python interprets it as a string instead of an integer, preventing arithmetic operations. For instance, printing `"10" * 2` results in **1010** instead of **20**. To perform calculations with the numeric value **10**, convert the string variable to an integer using `int(number_string)`. Remember, Python allows conversion between different variable types, not just integers and strings.

There is also a way that you can determine what data type a variable is by using the built-in `type()` 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

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 converting 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 you can 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
- **bool()** – converts a variable to a Boolean

```
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. Are you curious about 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 to see what happens.

Now, remove the **int()** function and run the following code to see what happens if you don't cast **number_str** to an integer:

```
number = 30
number_str = "10"
print(number + number_str) # This will cause a TypeError
```

You should see a **`TypeError`** because you cannot add a string and an integer directly together. However, a similar error would be encountered when trying to add together two different data types without using casting to ensure they are the same data type. Keep in mind that you do not need to worry about having an in-depth understanding of errors at this point, as you will learn more about them later.

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.

```
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 parentheses on line four above. When working with strings, we are able to put variables into our strings with the **`format`** method (a method in programming is like a tool that does specific tasks or changes data in a program). 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 parentheses after the **`format`** method.

Now it's time to try your hand at a few **practical tasks**.

Instructions

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 tasks 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're not likely to get it right the first time. Errors happen, and when they do, you need to know how to handle them. If you get stuck, read the "**Defensive Programming**" additional reading that accompanies this task.



Take note:

The tasks below 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.

When you select "Request Review", the task is automatically complete - you do not need to wait for it to be reviewed by a mentor.

You will then 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.

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 your 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 **input()** and then print out (output) the name.
- Use the same input and output approach to take in a user's age and print it out.
- Finally, print the string "Hello World!" on a new line. Note: 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. This is because each **print** statement automatically inserts an "enter", or newline instruction, at the end.

Be sure to place files for submission inside your **task folder** and click "**Request review**" on your dashboard.

Auto-graded task 2

Follow these steps:

- Create a new Python file in this folder called **conversion.py**.
- As in the previous practical task, 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.

Be sure to place files for submission inside your **task folder** and click "**Request review**" on your dashboard.



Rate us

Share your thoughts

HyperionDev 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.

