Python

Module 1

Installation and Python Basics

Bits and Bots
study group

Content

This module

For this module, make sure you:

- Read the guide and make sure you understand everything and have Python and an IDE available.
- Read and work through the introduction and chapter 1 of *Automate the Boring Stuff.*[1] This chapter will overlap with the screenshots on how to install Python. It covers the basics (functions, variables, etc.) and has practice questions at the end of each chapter.
- Read and work through chapter 1 and 2 of *Invent Your Own Computer Games With Python.*[2] Be able to understand the concepts covered in those chapters.

Further learning:

- Chapters 1 and 2 of Michael Dawsons, *Python Programming for the Absolute Beginner*.[3]
- If you want to do more reading and practising on the basics, check out: Python Tutorial (w3schools.com). Here you can check out multiple assignments and try out the basics for yourself.

Don't just read, make sure to test everything in your own Integrated Development Environment (IDE)!

This guide and the books that we use will overlap. We have found that repeating learning materials helps to make you remember and understand better. We therefore advise not to finish this module in one day, but to spread the workload over multiple days. Remember to make notes.

Installing Python

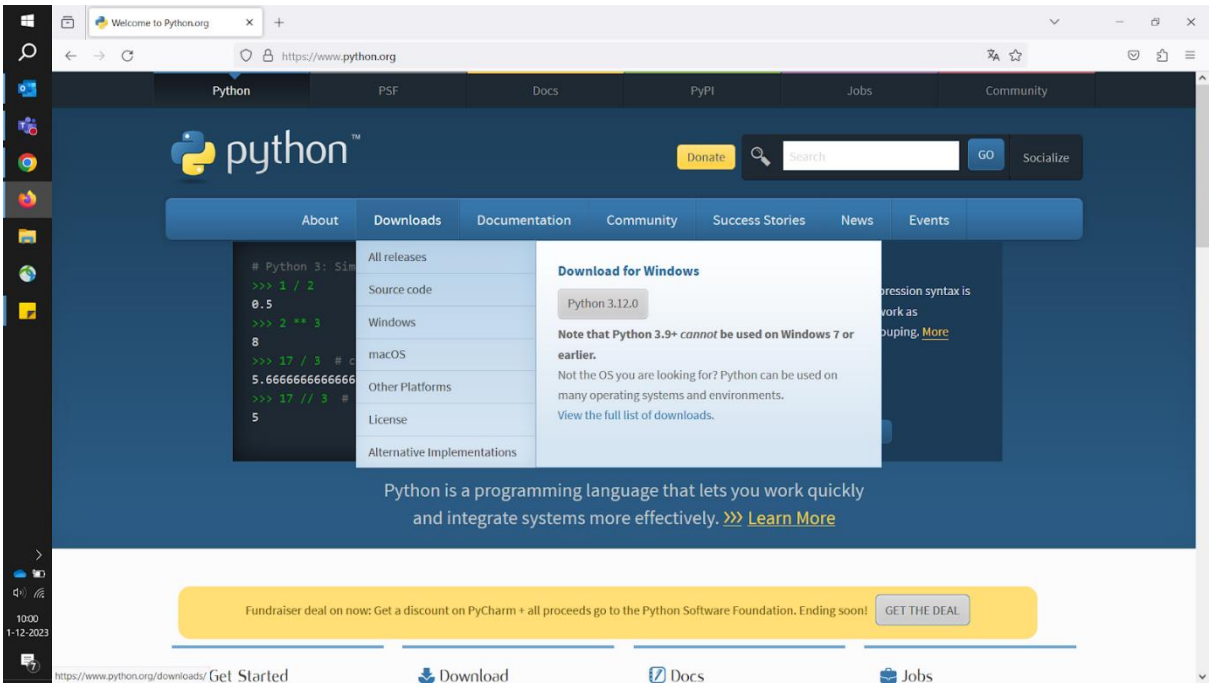
Install Python on your computer. It's recommended to install the latest Python version. Be sure to download a version of Python 3 (for example 3.12.0).

- Note: we recommend installing Python on your laptop or computer. However, we understand that this is not always possible. This could be the case if you use a government laptop. You can use an online version via: Online Python - IDE, Editor, Compiler, Interpreter (online-python.com), but we cannot guarantee everything will work spotless during the later modules.

Everything you need to know about Python, can be found on the official website: https://www.python.org/. Just to help you get started, we include some screenshots in this guide on how to install python on your computer. In this example we show Python for Windows.

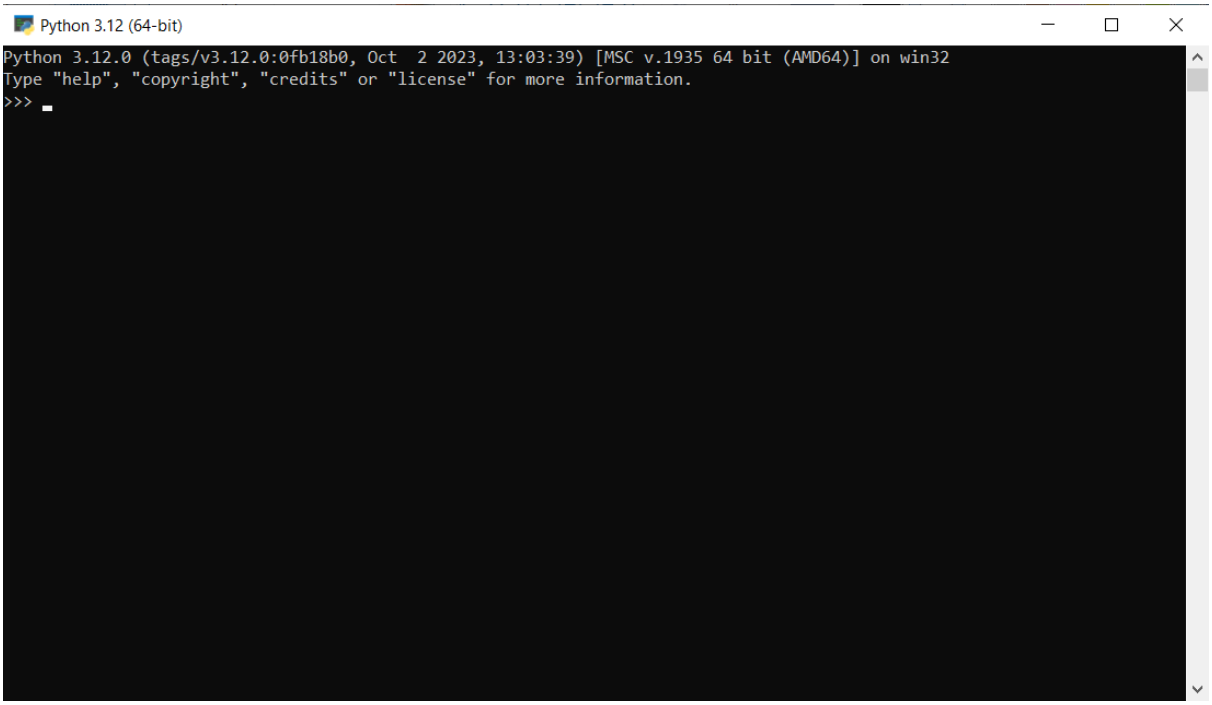
1. First, go to https://www.python.org/ and hover over downloads. Click on the button directly under 'Download for Windows'.

[1] Automate the Boring Stuff with Python
[2] https://inventwithpython.com/invent4thed/
[3] Python Programming for the Absolute Beginner (3rd Edition) (nibmehub.com)



2. Next, check your downloads folder. Double click the Python installer. The following screen should appear:

3. Click 'Install Now'. A pop up will show that installation is finished. As you can see above, Python is installed in a folder named 'AppData'. This is a hidden folder in Windows, but don't worry. You can find Python by entering 'py' in your search box, and select it. Once opened, you'll see the following window:



Now you'll be able to run Python programs on your computer. If you want to enter your programs, you'll need a different software. In the following step, we will explain what your options are and how you can install software that suits your needs best.

Installing an IDE (Integrated Development Environment)

It is possible to write your code in a text editor and run it on the command line directly. For now, we will use an interface to write code in. The reasoning behind this, is that most of the examples you will follow to create games in other modules, will use an IDE as well. There are multiple editors available, such as:

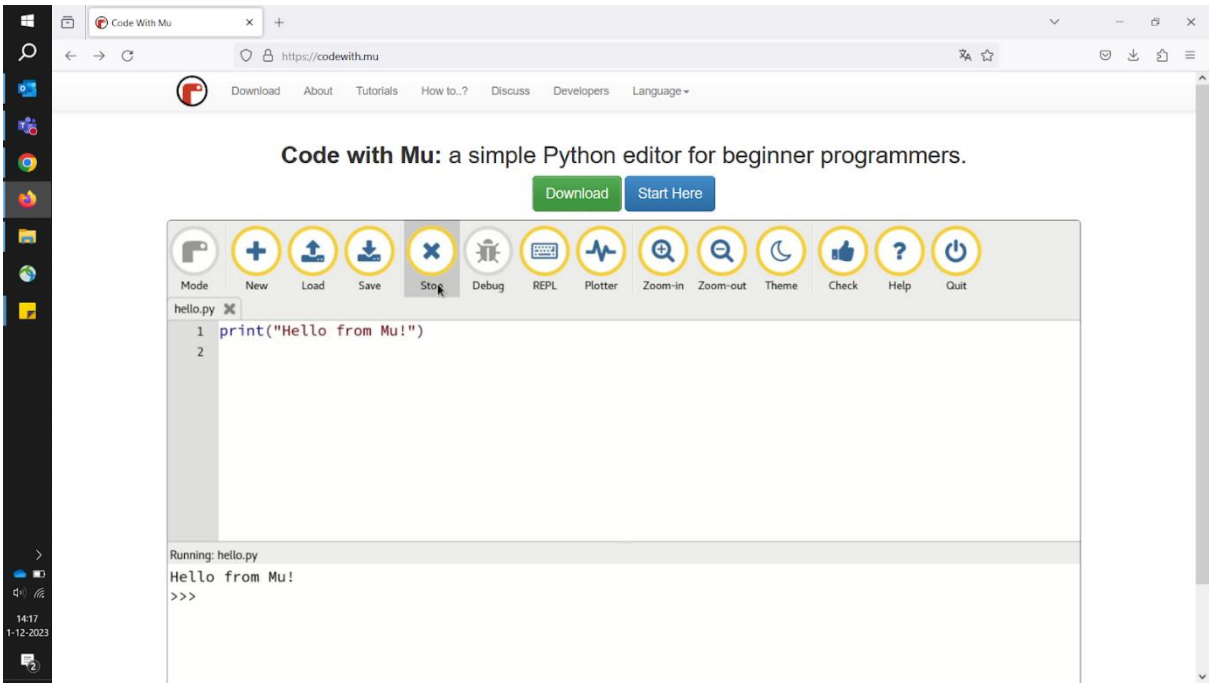
- An IDE like Visual Studio Code or IDLE (only for Python). Helps with syntax highlighting and understanding the language. IDLE is installed along with Python. You can find it by entering 'IDLE' in the search box. IDLE will be used in the learning materials, provided in later modules.

- Mu editor software. This is software where you'll enter your programs, much like the way you type in a text processor. Al Sweigart explains this thoroughly in the practical programming guide for beginners: 'Automate the boring stuff with Python'. We will provide a step-by-step install guide with screenshots on the next pages.

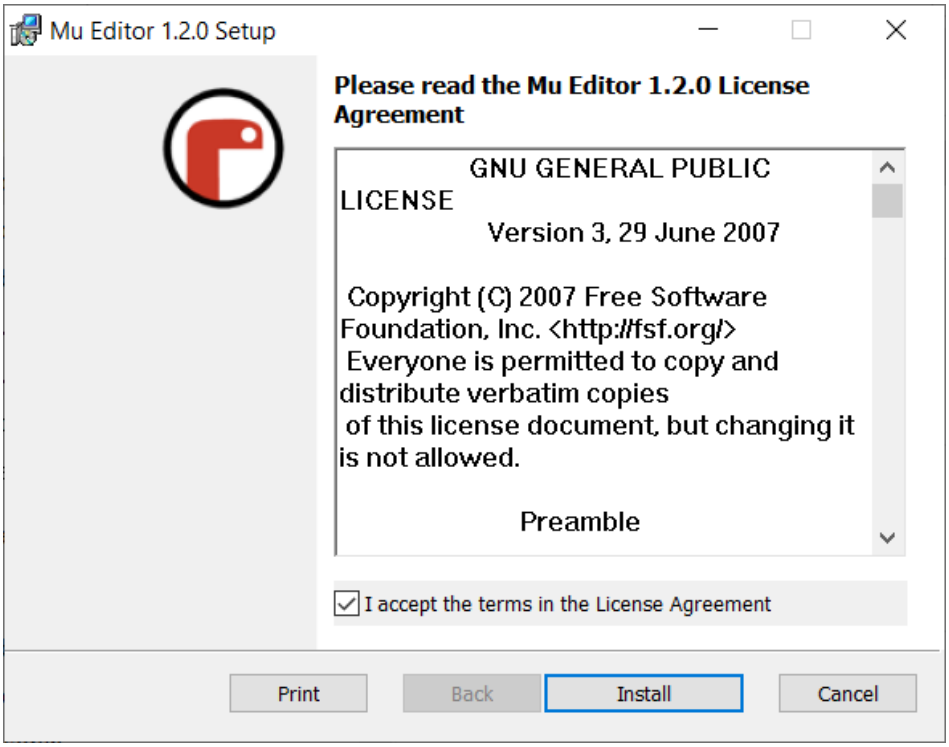
Installing the MU editor

You can use IDLE, which is already installed with Python or Mu. We do not really have a preference, but Mu is suggested for Python beginners. To install Mu, go to

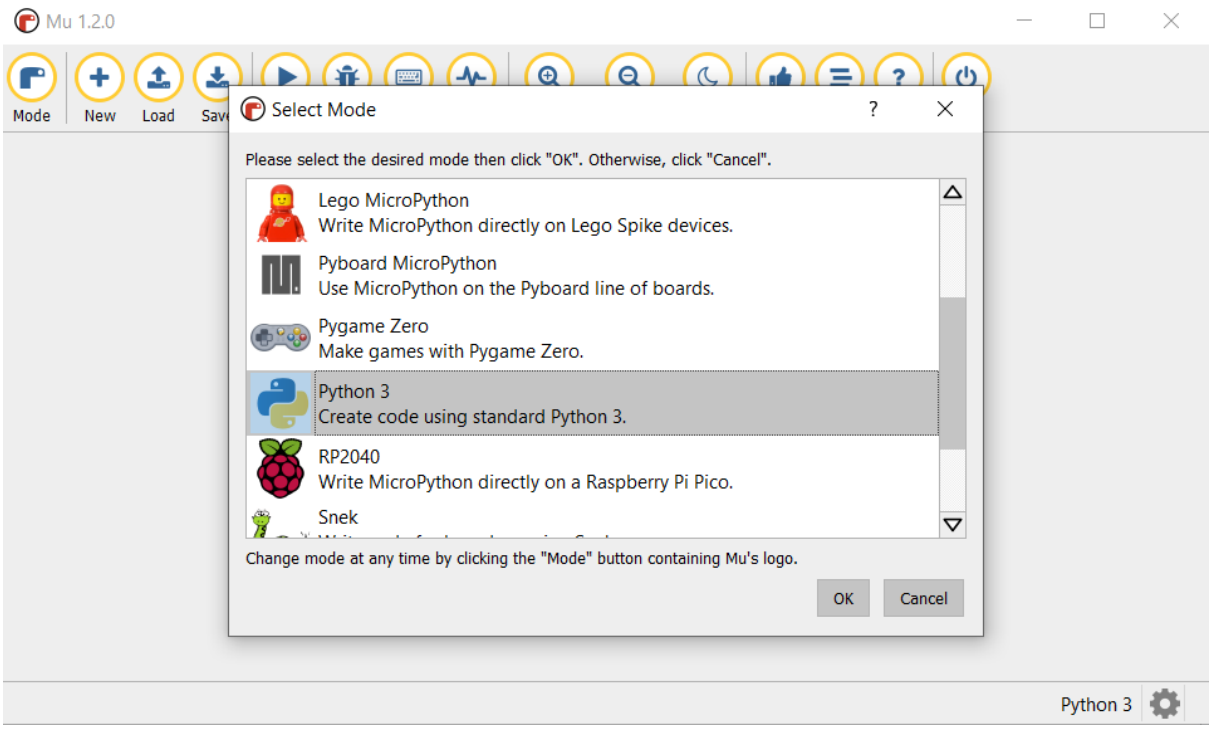
https://codewith.mu/ and click the green download button as shown below. With the redirect to the next page you can choose different operating systems. We show Windows here.



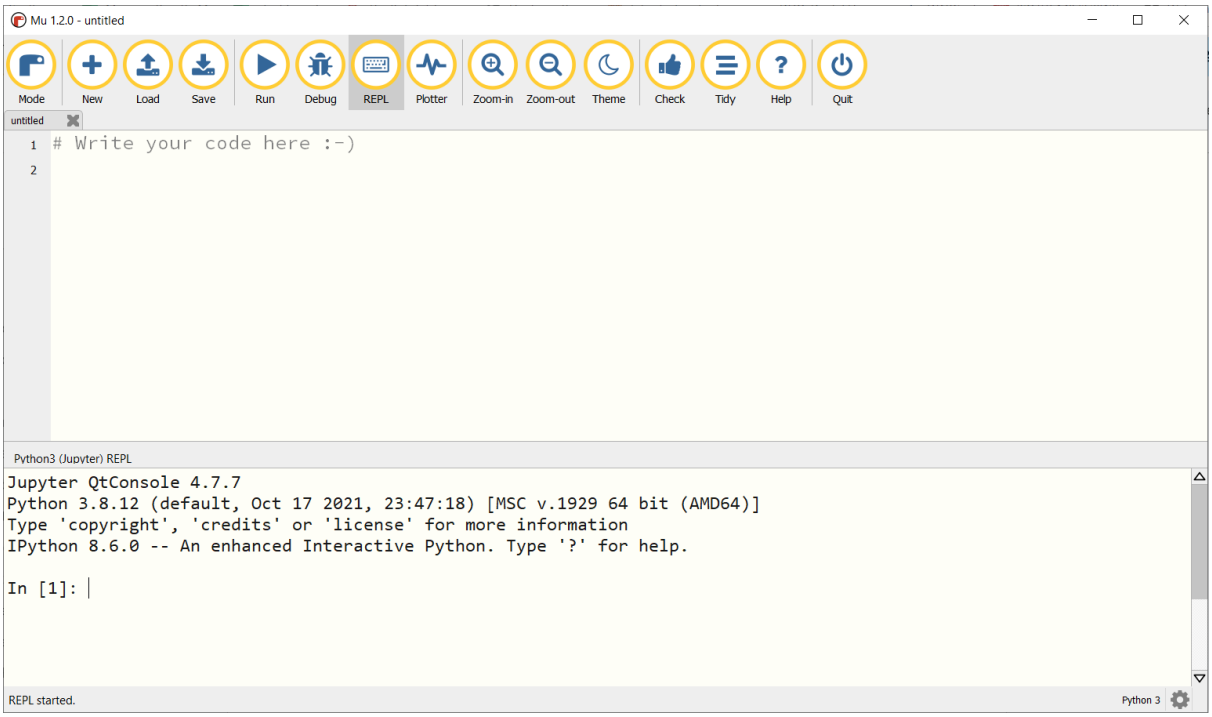
Find the installer in your downloads folder and run the file by double-clicking it. Accept the terms and install the file.



Once installed, you can start Mu by entering ‘Mu’ in the search box. If you run Mu for the first time, a Select Mode window will appear. You can select a number of options. This can always be altered later, but choose **Python 3** for the first time, as shown below.



Now that you have Mu installed, click on the REPL button at the top of the screen. At the lower half of the window, an interactive shell is opened.



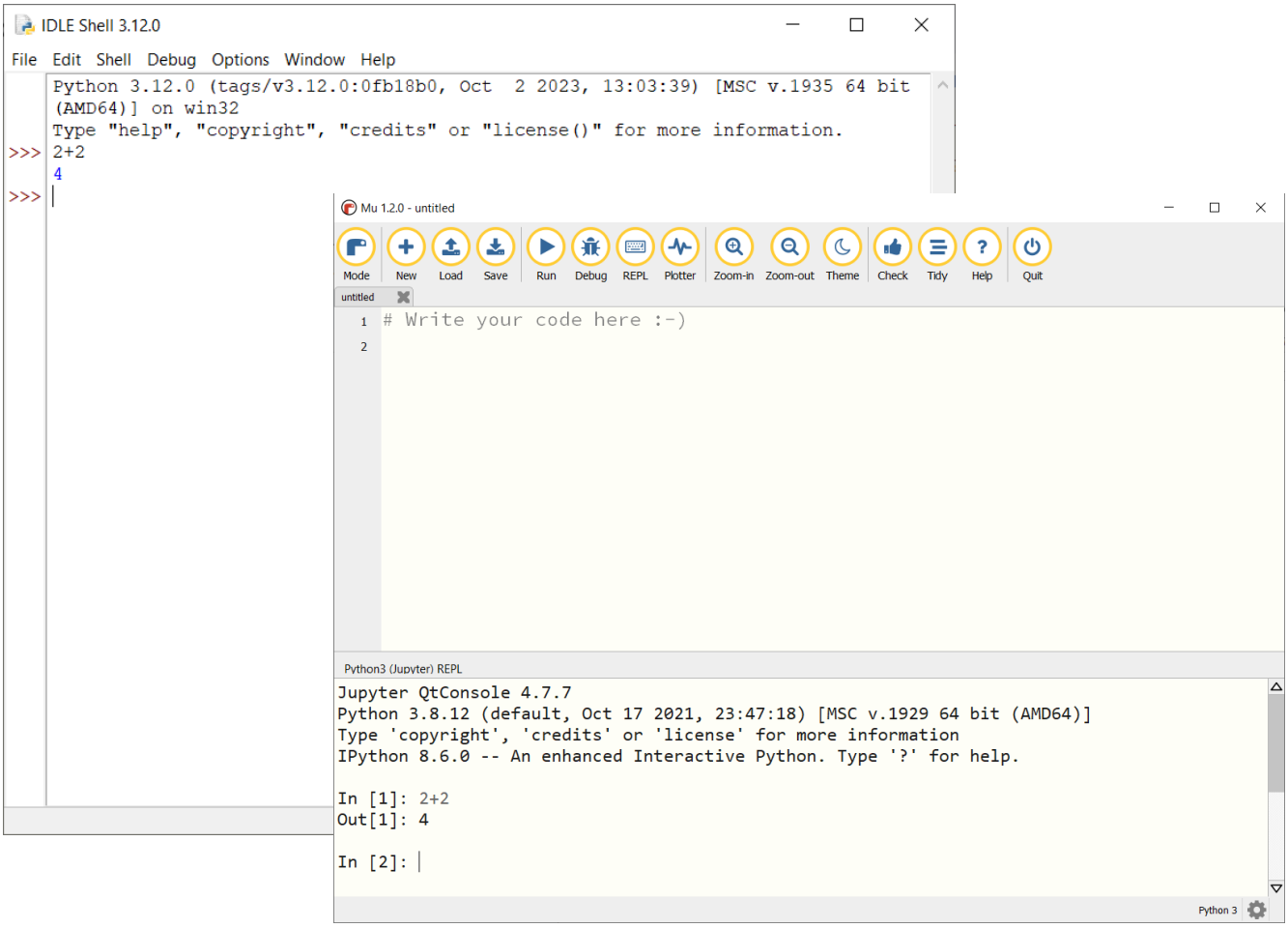
Try it out! The `In [1]:` part is called a prompt. You can give your computer an instruction here. Type the following, to give your first command:

```
print('Hello, world!')
```

And that's it! You're all set to start programming with Python and use IDLE or Mu to start writing your code. If you start a new code, make sure to save your file as *.py* to be able to alter and run it later. But what to type exactly? Just as you need to learn different words to make sentences in order to be able to have a conversation, you need to learn some basic expressions and functions first, in order to be able to give your computer commands.

Basic programming concepts

As said in the introduction, you do not need to be a mathematician. Python can do the maths for you, but you have to know what is what exactly. So let's do some basic maths. We all know 2 plus 2 equals 4. Below are the two interfaces, IDLE and Mu, that show us the same thing.



Expressions

`In [1]:` in Mu and >>> in IDLE are called ***prompts***. This is where your instructions to your computer begins. Above we told the computer it had to add 2 to 2. It did, and told us the answer: 4. In Python, 2 + 2 is called an ***expression***, which is the most basic kind of programming instruction in the language. Expressions consist of ***values*** (such as 2) and ***operators*** (such as +), and they can always evaluate (that is, reduce) down to a single value. That means you can use expressions anywhere in Python code that you could also use a value.

Below you will find all the math operators in Python, with thanks to Al Sweigart.[4]

Operator	Operation	Example	Evaluates to . . .
**	Exponent	`2 ** 3`	8
%	Modulus/remainder	`22 % 8`	6
//	Integer division/floored quotient	`22 // 8`	2
/	Division	`22 / 8`	2.75
*	Multiplication	`3 * 5`	15
-	Subtraction	`5 - 2`	3
+	Addition	`2 + 2`	4

There are multiple operators in Python. On the website of W3Schools, you can find them all, and try out for yourself what they do.

Data types

A ***data type*** is a category for values. So the value 2 in the math example form above is a data type with the name ***integer***. Integers are always whole numbers, negative or positive. So -1 or 3 or even 500 is an integer. Numbers with a decimal point, such as 3.25 are called ***floating-point numbers*** (or ***floats***). So 42 is an integer, but the value 42.0 would be a floating-point number.

Another common data type in Python, next to integers and floats, are ***strings***. These are text values, like the Hello, World! example that we have seen in the introduction. However, strings are always surrounded by single quotes, so Python knows where the string starts and ends. You even can have strings with no characters in it (*blank string* or *empty string*).

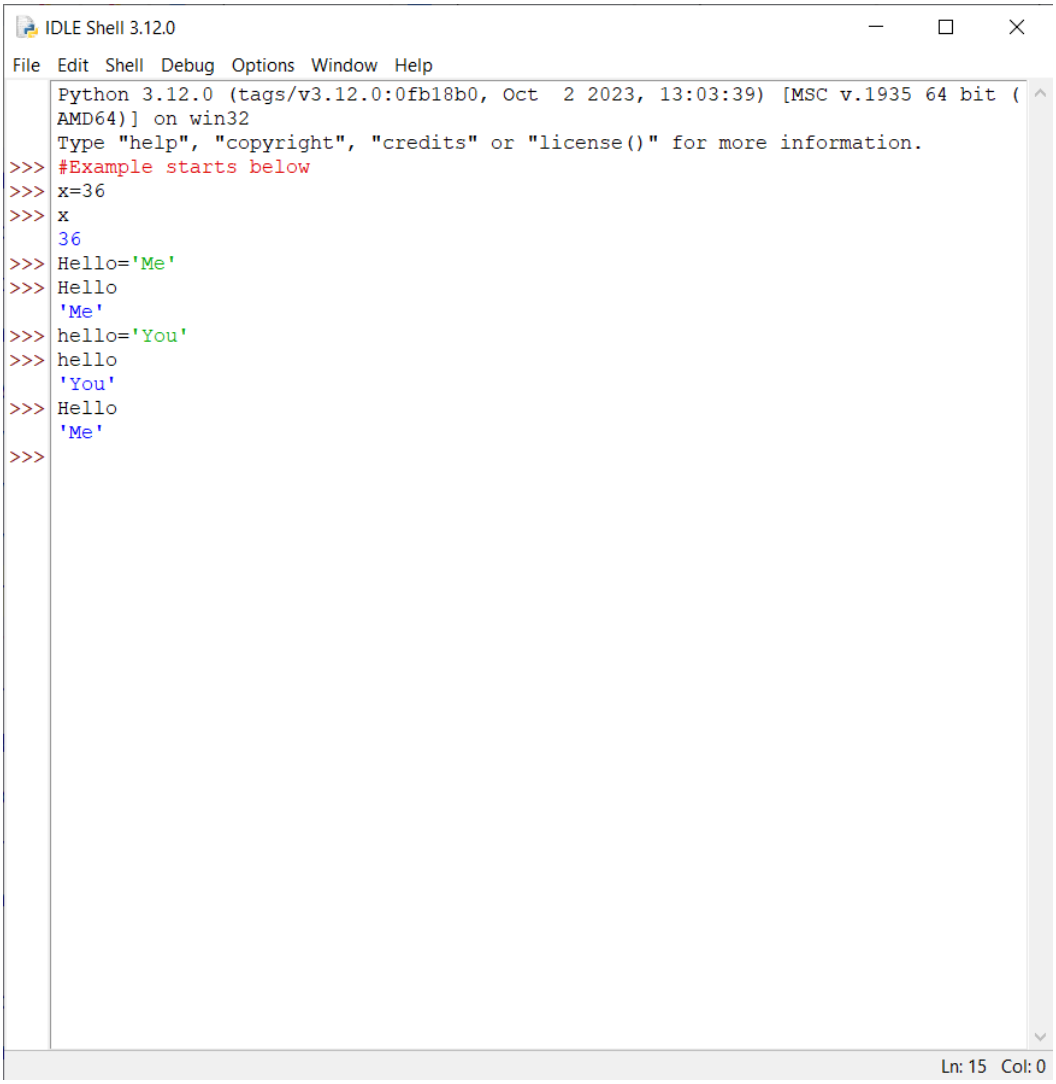
[4] https://automatetheboringstuff.com/2e/chapter1/

More on data types can be found here. It also teaches you a trick to help you find the data type of an object, using the `type() function`.

Variables

A variable is a container for storing single values. It will be created at the moment you assign a value. So this could be a number (integer) or text (string). You assign a value to a variable by using an equal sign. The variable can have almost any name you choose, but a descriptive name will help make your code more readable. Imagine that you moved to a new house and labelled all the boxes with 'Stuff'. It would be hard to find anything back.

In the screenshot below you can see some code. It contains variables and values. Can you name what is what? The answers are shown below the screenshot.

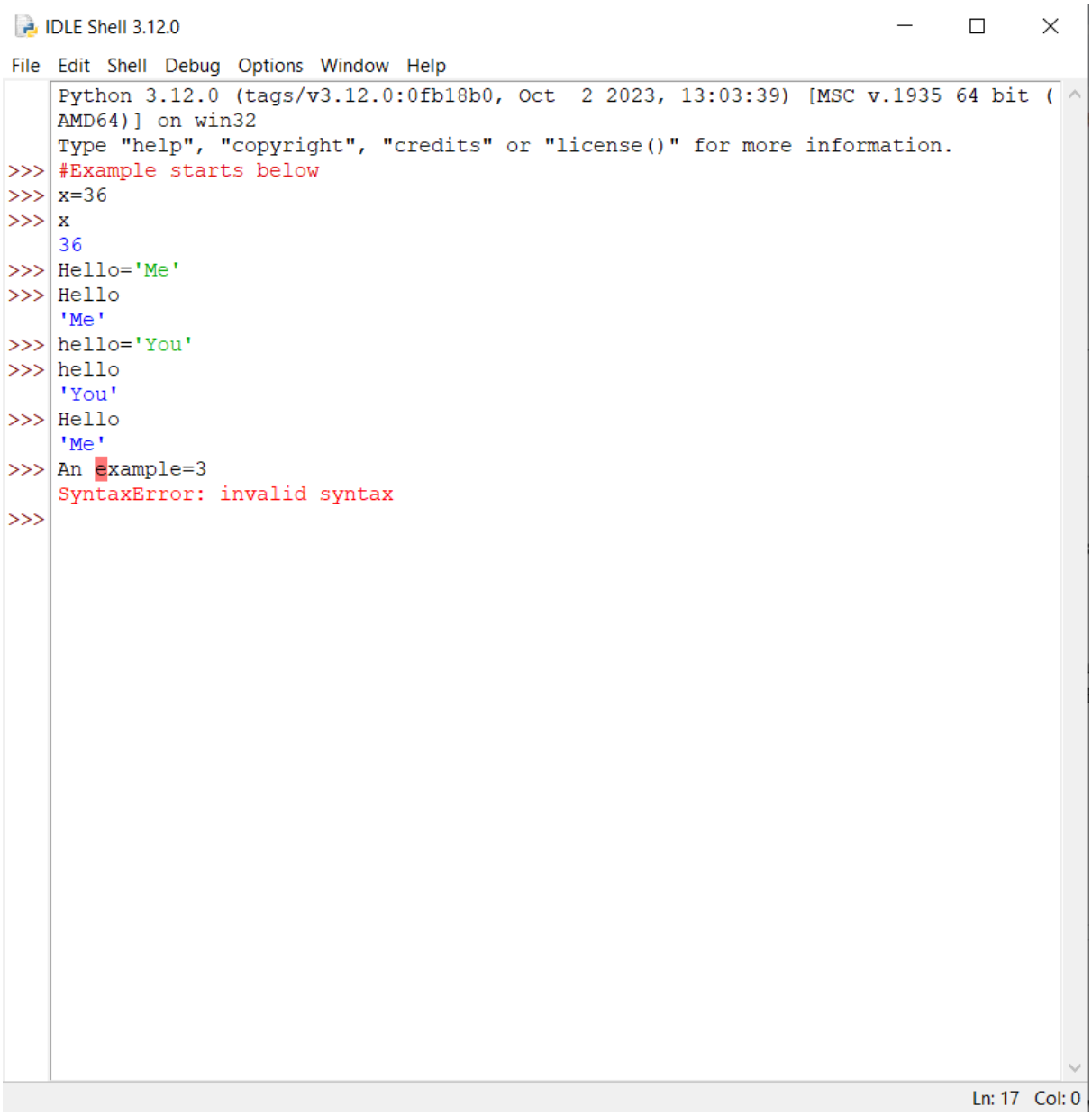


In the screenshot above, there are three variables defined: x, Hello and hello. They all have their own value, but two data types are being used. Me and You are both strings. You can recognize this by the single quote that is used. The number 36 is an integer.

Did you see that variables are case sensitive? This means that the variable hello will keep the value You and is not overwritten by the variable Hello. If you would write `Hello='You'`, it would no longer print 'Me' as an outcome.

Try it out! Rewrite the code in your own interface and play with different names for variables and different values

Perhaps you tried to create a variable, like the one below.



The SyntaxError (shown in red) tells you something went wrong. In this case, it is the name of the variable. You can choose almost anything you like, but there are some exceptions:

- It can only be one word with no spaces.
- It can use only letters, numbers and the underscore (_) character.

- It can't begin with a number.

To practice more with variables, check out W3Schools.

Functions

In Python you have different ***functions***. It tells the computer what to do exactly. And if you understand strings and functions, you will be able to make programs that interact with users. This is important because text is the main way the user and the computer will communicate with each other. The user enters text through the keyboard with the `input()` function, and the computer displays text on the screen with the `print()` function.

You gave your first command to your computer a little while back. It said: `print('Hello, world!').` If you have tried it, you saw that the outcome was just Hello, world! - so what this function did was print the string value inside its parentheses on the screen. There were no parentheses or single quotes. The quotes only mark where the string begins and ends; they are not part of the string value. So, with the **`print()` function**, you say that Python is calling this function and the string value is being passed to the function. A value that is passed to a function is called an ***argument***.

Another useful function that probably will come in handy when you create a game, is the **`input()` function**. This function waits for the user to type some text on the keyboard and press enter. The text string that the user enters becomes the value that the function call evaluates to. Function calls can be used in expressions anywhere a value can be used. The value that the function call evaluates to is called the return value.

To learn more about functions and try creating one yourself, you can have a look at W3Schools.

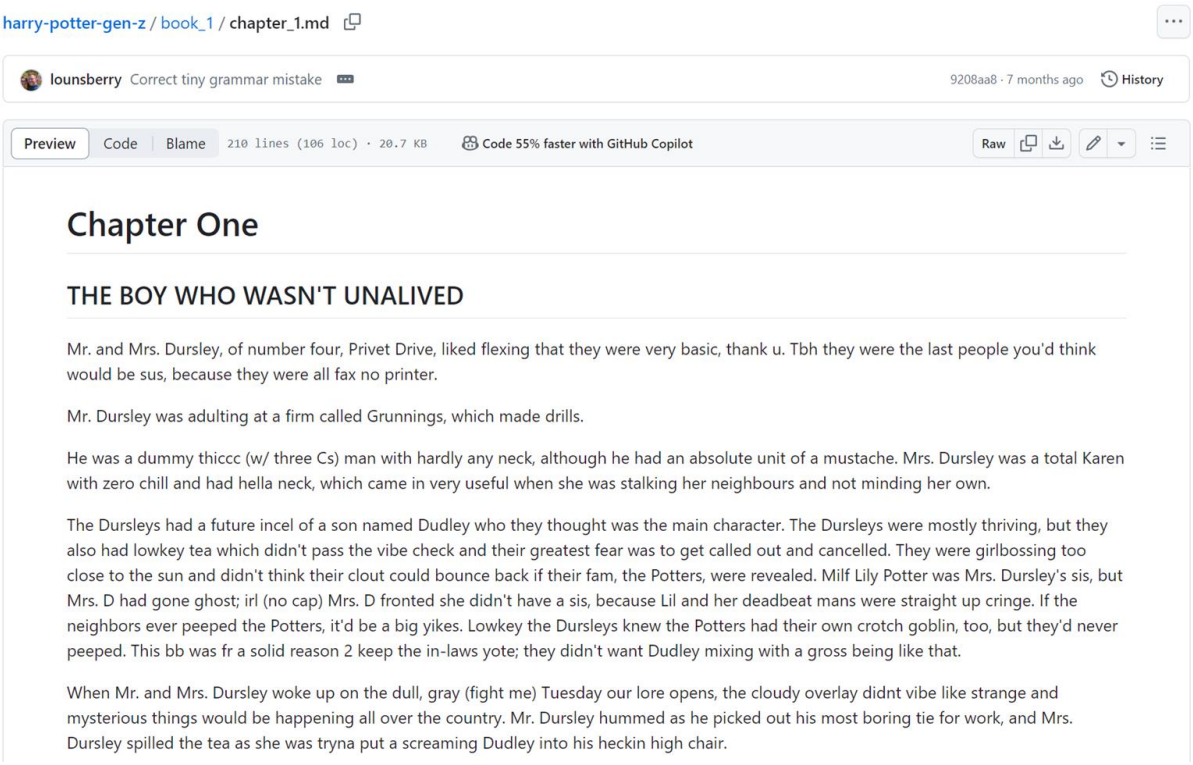
Your First Program

Now that you have installed Python and learned about the basic concepts, it is time to write your first program, save it as a Python file and run it on your computer. In order to do this, follow the instructions of 'Your First Program' from *Automate the Boring Stuff*.

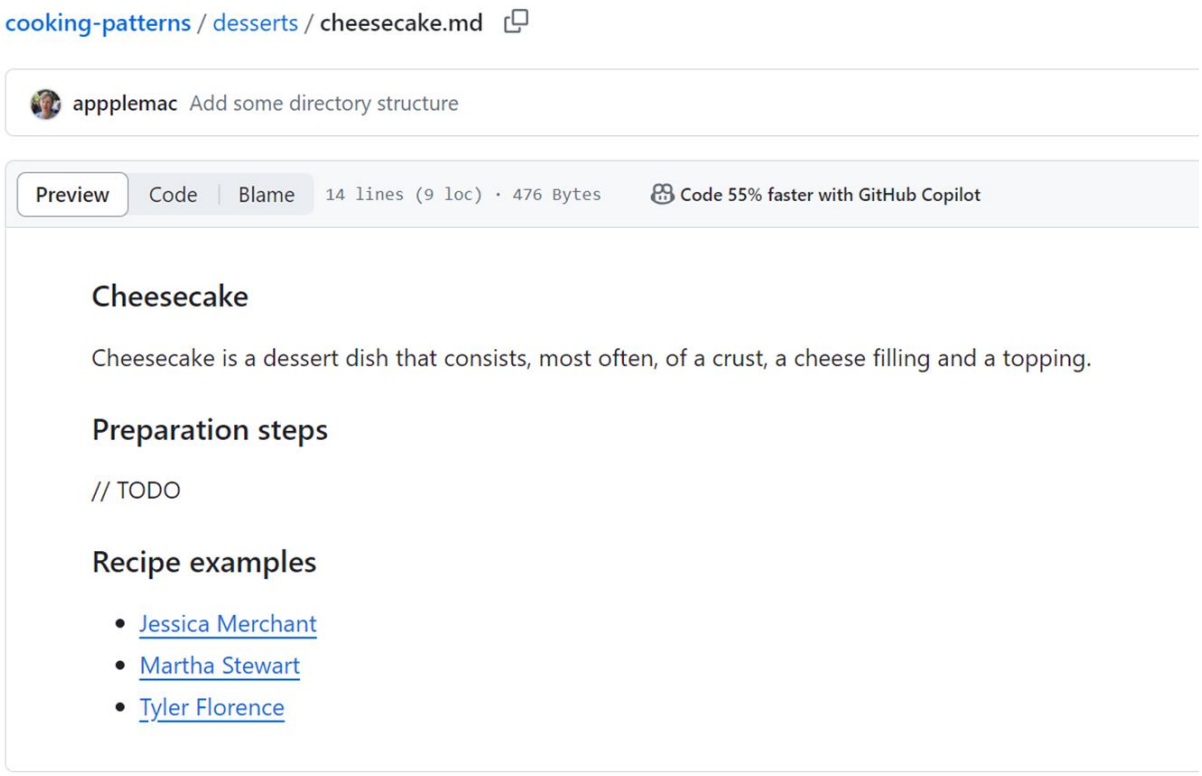
GitHub

GitHub is a platform that allows anyone to create, store, manage, and share content. Oftentimes this is code. But that is not the only thing present in GitHub. Here are some of the use cases for GitHub:

- Software development
- Fanfiction
- Zines
- Community/crowdsourcing projects
- Finding file samples
- Educational resources
- Reading lists
- Datasets
- Blog posts
- Documentation



Here you can see some fanfiction that is being written with a community. But you can also have recipes on there:



As digital archivists, we often use open-source tools to aid us with e.g. identifying and validating digital objects or harvesting websites. Here are some useful examples of GitHub repositories used in the field that you can find:

- https://github.com/openpreserve/jhove
- https://github.com/digital-preservation/droid
- https://github.com/webrecorder/browsertrix-crawler
- https://github.com/noord-hollandsarchief (see the repositories in this account)

So why is GitHub great to use? GitHub allows people to work collaboratively on any project, with tracking and control settings favouring non-linear workflows and multiple contributions at a time. By tracking everything, there is also a lot of data integrity. Additionally, you can use GitHub to let the developers know you have an issue/something is not working.

There are a lot of people working with digital archives that use GitHub. This is the reason that we found it very important for everyone to learn how to use it. Additionally, to show you that it is not just a scary programmer platform.

GitHub Exercise

For this module, we want you to:

1. Create a GitHub account if you do not have one:
    - To start with, read more about GitHub here. One thing to bear in mind is that GitHub, while an amazing tool does come with a lot of jargon. A great explanation for some of the terminology around GitHub can be found here.
    - To create your first GitHub account you will first need to register at www.github.com and follow the instructions provided here.

2. Follow the GitHub tutorials:
    - Try out a few features with your new account. This will be creating a repository, creating a pull request, creating branches and merging a pull request. If these terms seem strange then that is because they are slightly. A link to the tutorial you will be following for the first stages of GitHub can be accessed here.
    - If you are enjoying learning and wish to practice further then you can follow the tutorial on the GitHub skills page 'Introduction to GitHub'

Note that both steps can also be done after the monthly meeting on GitHub in April. In this meeting we will go over GitHub terminology and give a demo.