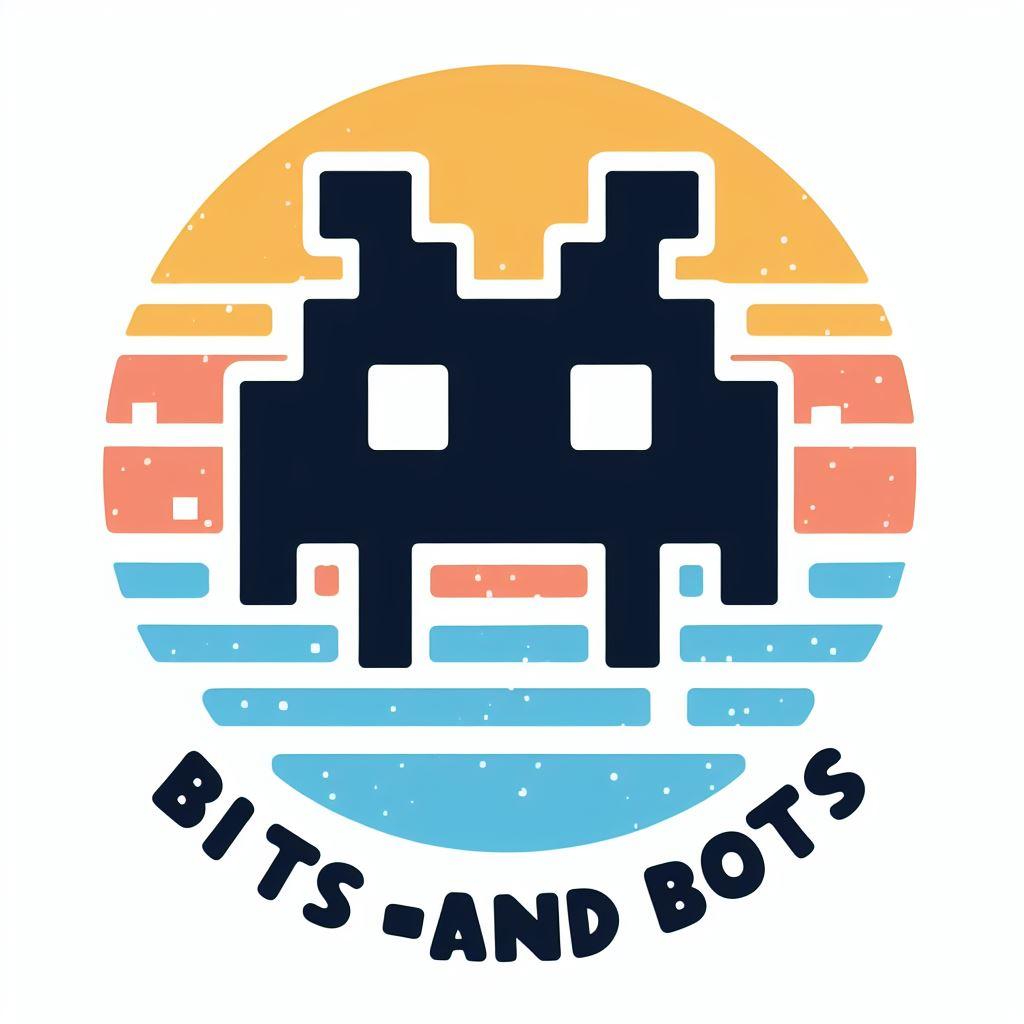
**Guide to building games with Python**

With the Bits and Bots study group



**Table of contents**

[Introduction](#_ypw07qedyjtn)

[Module overview](#_71bwouslg7a0)

[External reading and learning materials](#_aascxcjg934i)

[Module 1: Installation and Python basics](#_ky1wbyestwr9)

[1. Installing Python](#_fhza70vqhw5h)

[2. Installing an IDE (Integrated Development Environment)](#_f3qsnshu1ioy)

[3. Basic programming concepts](#_mtry74shftz3)

[3. 1 Expressions](#_6cfah6j1boes)

[3.2 Data types](#_3dhkowlg1c28)

[3.3 Variables](#_z5ckelb4yghl)

[3. 4 Functions](#_hidvirjm3sjb)

[Module 2: Your first games](#_12ssgtzhtvws)

[Guess the Number](#_13828uutmh13)

[Rock, Paper, Scissors](#_59l866i6fgid)

[Review](#_e3pwwrry1kxn)

[Module 3: Functions, Loops, and Debugging](#_yfdyb7wmmhmw)

[Module 4: Lists and Dictionaries](#_2p6rx88cxvmp)

[Module 5: From A(I) to B(agels)](#_qu4nxlj5uauu)

[Module 6: Coordinates and a Treasure Hunt](#_xpr26qew75fz)

[Module 7: Advanced AI and Coordinates](#_socm4dy86mis)

[Module 8: Pygame](#_gj05s84iggy)

# Introduction

Hello, world!

When you start programming, you’ll find the above phrase more often. A programming language is a way to make manual work more easier and efficient by automating it. It is a way to let your computer ‘speak’ to the world and execute some tasks for you. Programming skills are mostly used to automate big amounts of work, to make it faster. A simple task that takes you hours or even days by hand, can be automated to take up merely seconds.

In order to program you do not need any mathematical skills per se. You should be creative and comfortable with working with your computer. But first and foremost, you have to learn a new language. Because when it comes to programming, you have to be very precise in telling your computer what to do. And that is something you can literally trace back in the code with phrases like: “If [this thing occurs], then [do this] or else [do that]”. If this makes you uncomfortable, do not worry. One of the fun things about programming is that you cannot really break something. If there's a mistake in your code, your computer will indicate where the mistake occurred because that is the exact point your machine didn’t know what to do anymore. For most error messages, you can find the answer online what the exact mistake is. But more on that later.

This guide will provide a step-by-step walkthrough on how to download and install python. It also helps you through the first steps of learning python. The course is made by beginners in Python and focused on beginners and to have fun while learning Python. In different modules and expert sessions, you will become more familiar with Python and learn some new skills that you can apply later on in your day to day work.

And now for something completely different: We have integrated a gaming element in this course! There is a lot of material out there and available online, which we will refer to a lot. The most exercises will evolve around a game, like Rock, Paper, Scissors and Hangman. At the end of this course you will:

* Have an understanding of the basic principles in Python, like variables, data types, and functions.
* Be able to write and read code and have an idea how you could use scripts in your day to day work.
* Make games in Python that you can show off to your colleagues.
* Have made the first steps in developing some digital skills.

## Module overview

| **Module** | **Dates** |
| --- | --- |
| 1 | 15 Feb - 15 March |
| 2 | 15 March - 15 April |
| 3 | 15 April - 15 May |
| 4 | 15 May - 15 June |
| 5 | 15 June - 15 July |
| **Summer recess** | |
| 6 | 15 September - 15 October |
| 7 | 15 October - 15 November |
| 8 | 15 November - 15 December |

Note: for dates of monthly meetings, have a look at the [General Guide](https://docs.google.com/document/d/1ub6CTB4U5yM4khYQjNtGiHaFIhlpLv9MH2LebAswDJY/edit).

**Module 1: Installation and Python basics**

In this module we will:

* Learn how to install Python and an integrated development environment.
* Learn the basic principles of Python (variables, data types, statements, functions, and loops).
* Learn to use the interactive shell to write and run your programs.
* Write your first program

**Module 2: Your first games**

In this module we will:

* Build our first games: Guess the Number and Rock, paper, scissors
* Learn to recognise and use variables, functions, and loops

End Goal: Applying our basic knowledge to create two games and understanding how they work.

Extra: make the Joke-telling program and add a preservation joke

**Expert session**

Meetup: expert session from Daniel Steinmeier (Using basic Python: why it is useful and how to apply it in your digital preservation work). Date: TBA.

**Module 3: Functions, Loops and Debugging**

In this module we will:

* Learn how to make your own functions
* Learn how to work with loops
* Learn how to debug code

End Goal: Understand how debugging works and create a new game with loops and functions.

**Module 4: Lists and Dictionaries**

In this module we will:

* Design a program with flowcharts
* Learn about lists
* Learn about dictionaries
* Make a hangman game

Optional:

* Give your game a theme (digital preservation or digital archiving).

**Module 5: From A(I) to B(agels)**

In this module we will:

* Work with AI
* Learn about augmented assignment operators
* Learn about string interpolation

**Module 6: Coordinates and a Treasure Hunt**

In this module we will:

* Learn coordinates
* Learn about data structures

**Module 7: Advanced AI and Coordinates**

In this module we will:

* Work with a more advanced AI
* Work with the Cartesian coordinate system

**Module 8: Pygame**

In this module we will:

* Create a graphics window
* Create and adding graphics

# External reading and learning materials

Al Sweigart, Invent your own computer games with Python (2016), available online: <https://inventwithpython.com/invent4thed/chapter0.html>

Al Sweigart, Automate the boring stuff with Python. Practical programming for total beginners (2020), available online: <https://automatetheboringstuff.com/>

Michael Dawson, Python programming for the absolute beginner (2010), available online: <https://nibmehub.com/opac-service/pdf/read/Python%20Programming%20for%20the%20Absolute%20Beginner-%203rd%20Edition.pdf>

Python Tutorial with exercises, W3Schools, online free course: <https://www.w3schools.com/python/default.asp>

Official Python Beginners Guide, <https://wiki.python.org/moin/BeginnersGuide>

# Module 1: Installation and Python basics

**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, chapter 1, and chapter 2 of *Invent Your Own Computer Games With Python.*[[1]](#footnote-0) Be able to understand the concepts covered in those chapters.

**Further learning:**

* For extra understanding: read chapter 1 of *Automate the Boring Stuff.*[[2]](#footnote-1) This chapter will also cover the basics (functions, variables, etc.) and has practice questions.
* Chapters 1 and 2 of Michael Dawsons, *Python Programming for the Absolute Beginner*.[[3]](#footnote-2)
* If you want to do more reading and practising on the basics, check out: [Python Tutorial (w3schools.com)](https://www.w3schools.com/python/default.asp). 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)!

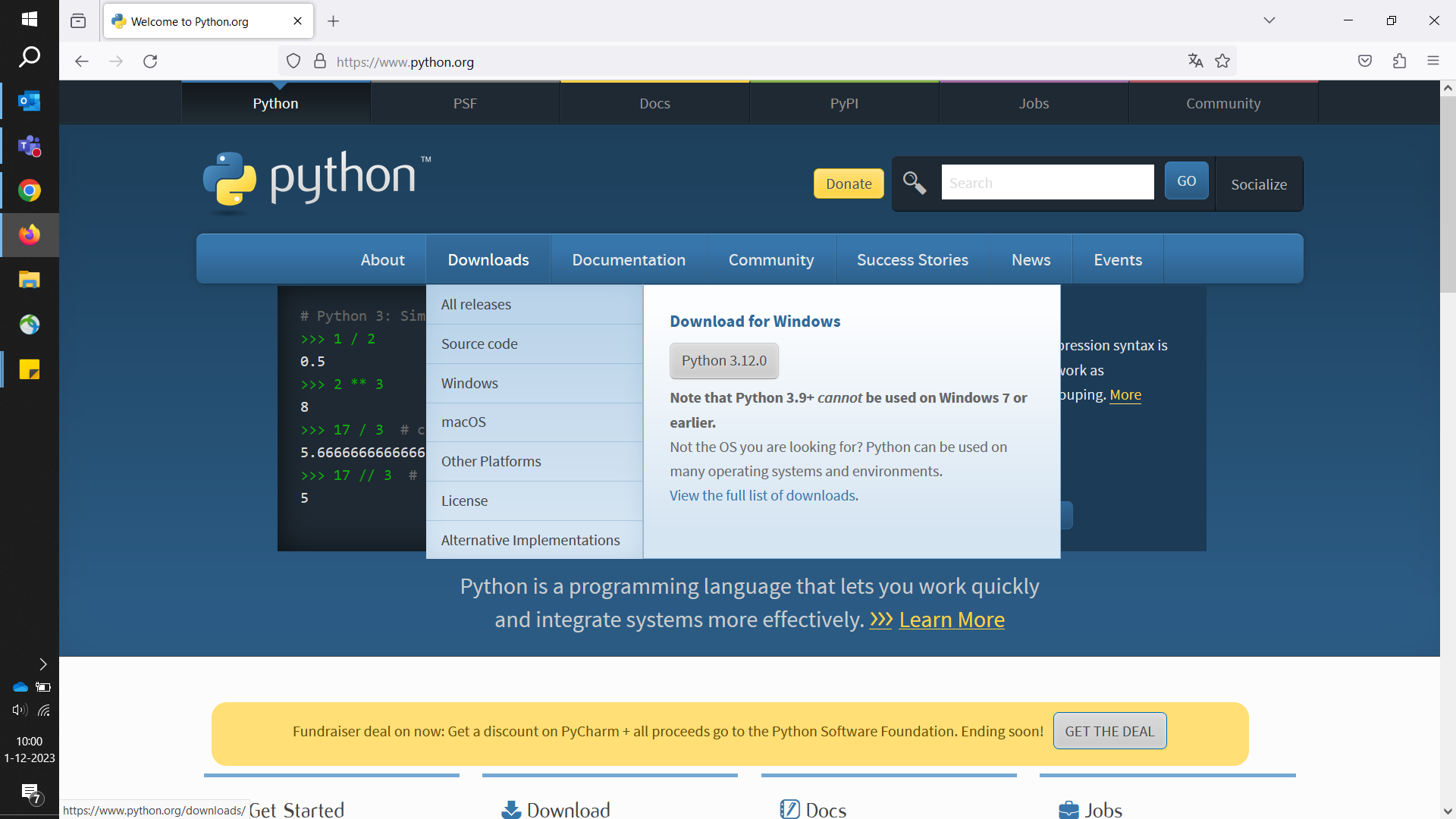
## Installing Python

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).

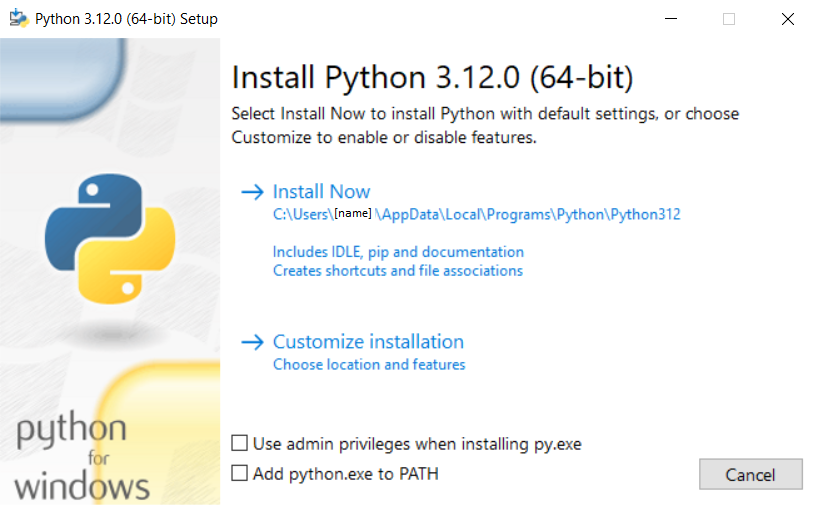
* Note: we recommend installing python on your laptop or computer. However, we understand that this is not always possible. This could be the case if you use a government laptop. You can use an online version via: [Online Python - IDE, Editor, Compiler, Interpreter (online-python.com)](https://www.online-python.com/), 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.

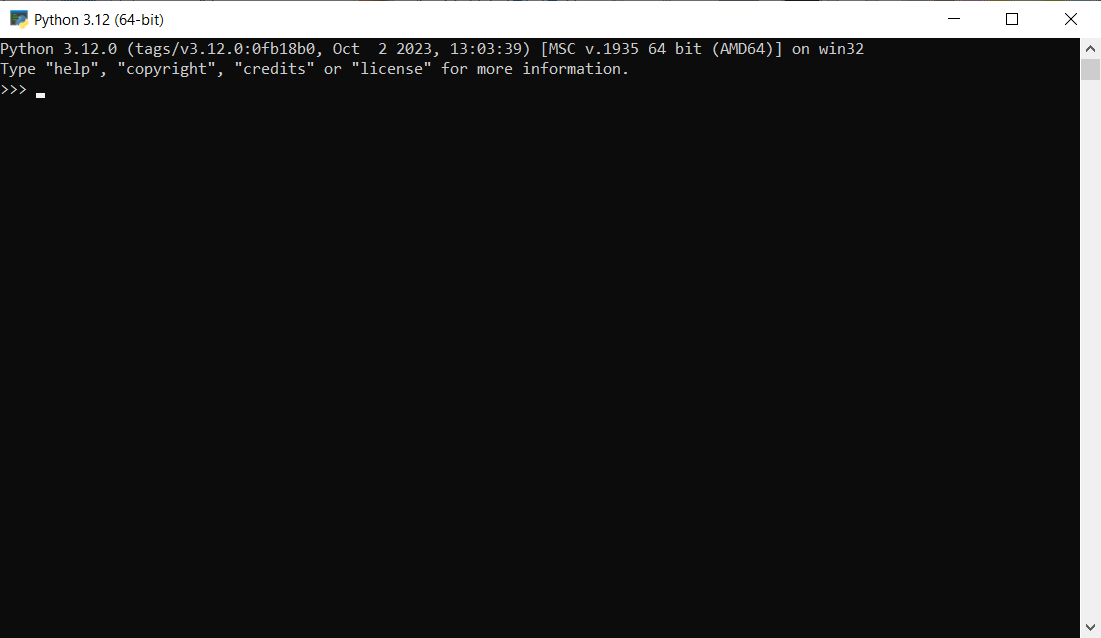
First, go to <https://www.python.org/> and hover over downloads. Click on the button directly under ‘Download for Windows’.



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



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:



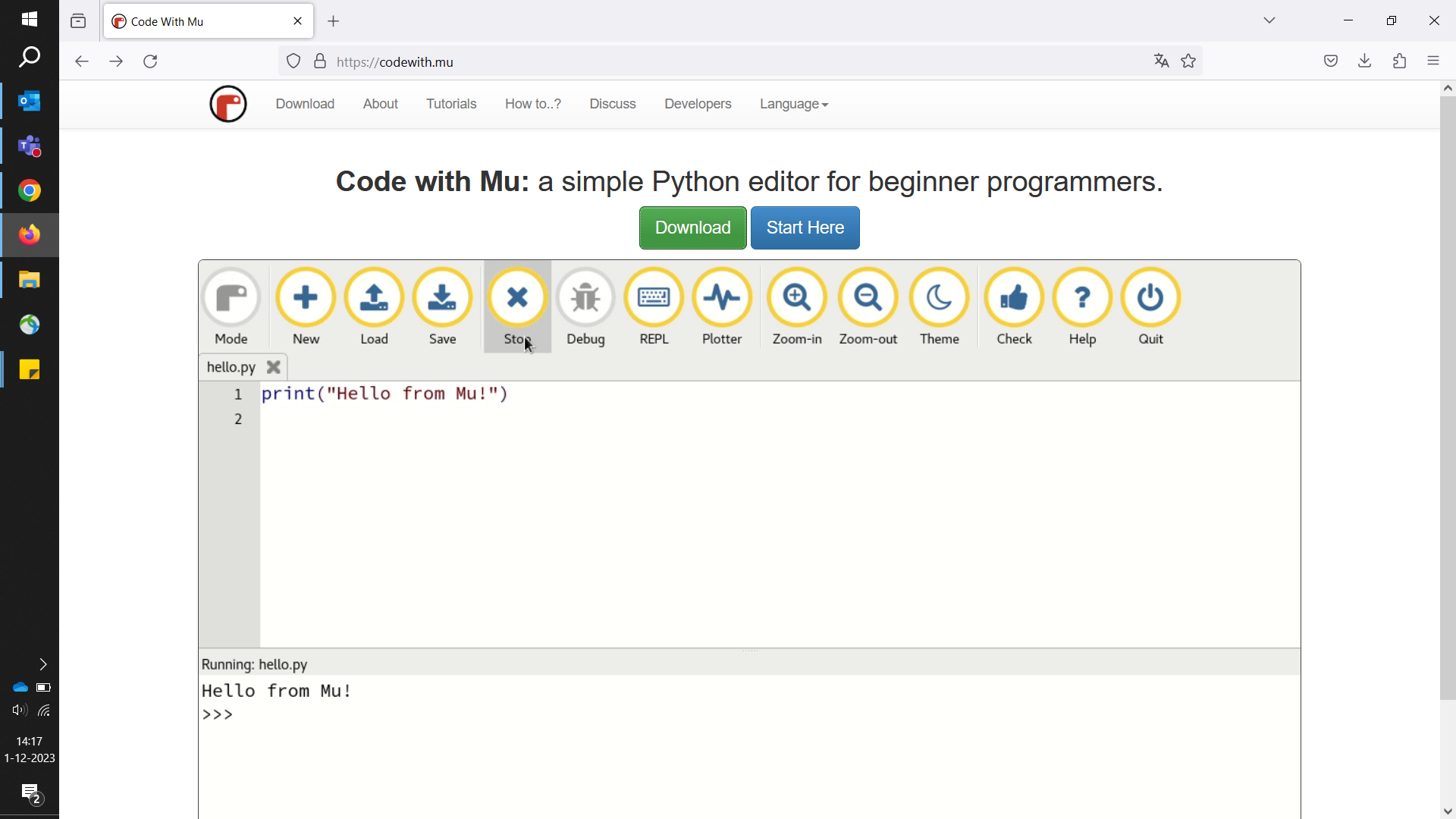
Now you’ll be able to run Python programs on your computer. If you want to enter your programs, you’ll need a different software. In the following step, we will explain what your options are and how you can install software that suits your needs best.

## 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.

* Mu editor software. This is software where you’ll enter your programs, much the way you type in a word processor. Al Sweigart explains this thoroughly in the practical programming guide for beginners: ‘[Automate the boring stuff with Python](https://automatetheboringstuff.com/)’. We will provide a step-by-step install guide with screenshots on the next pages.
* 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.

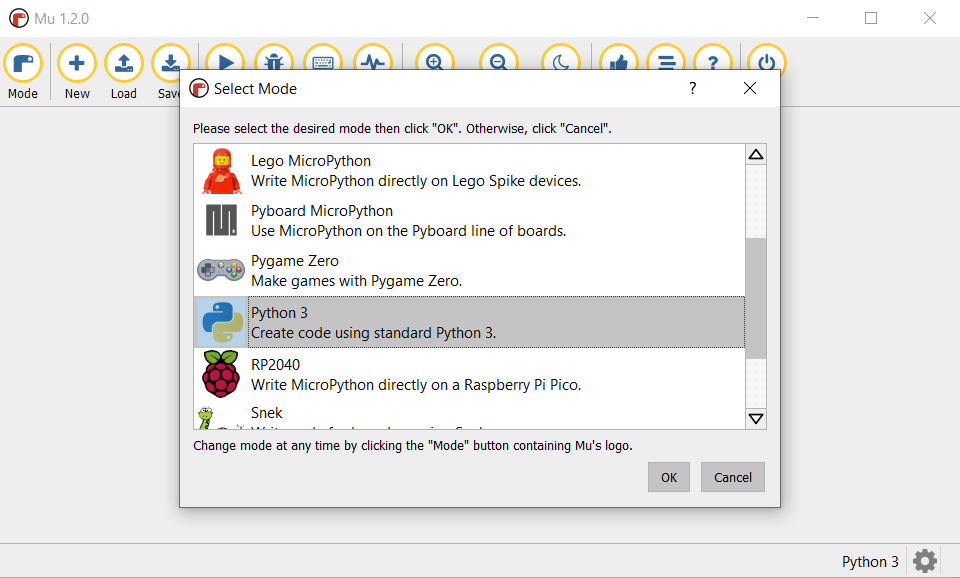
To install Mu, go to <https://codewith.mu/> and click the green download button as shown below. With the redirect to the next page you can choose different operating systems. We show Windows here.



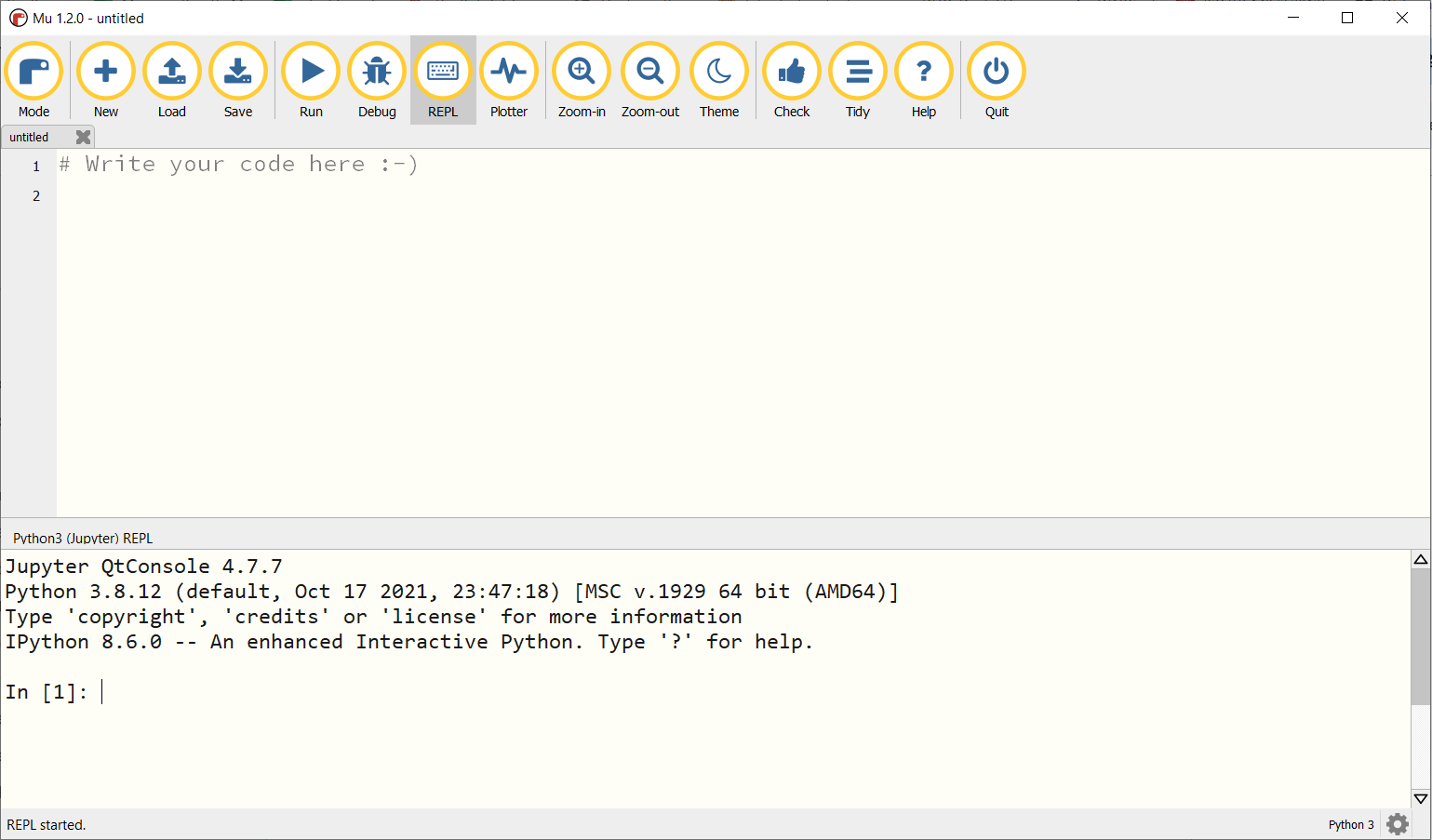
Find the installer in your downloads folder and run the file by double-clicking it. Accept the terms and install the file.



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.



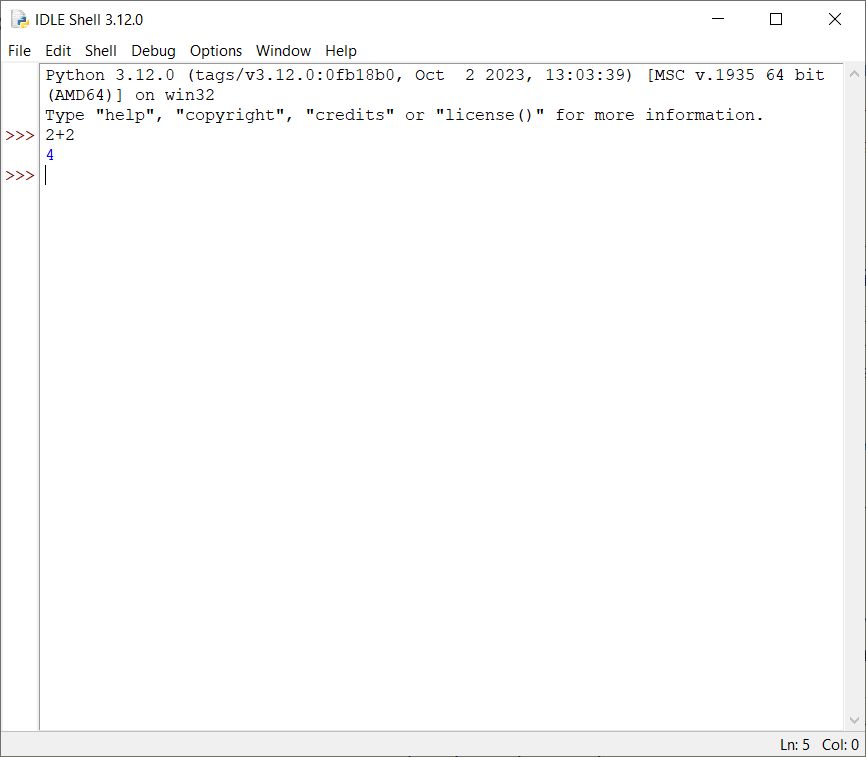
**Try it out! The In [1]: part is called a prompt. You can give your computer an instruction here. Type the following, to give your first command:**

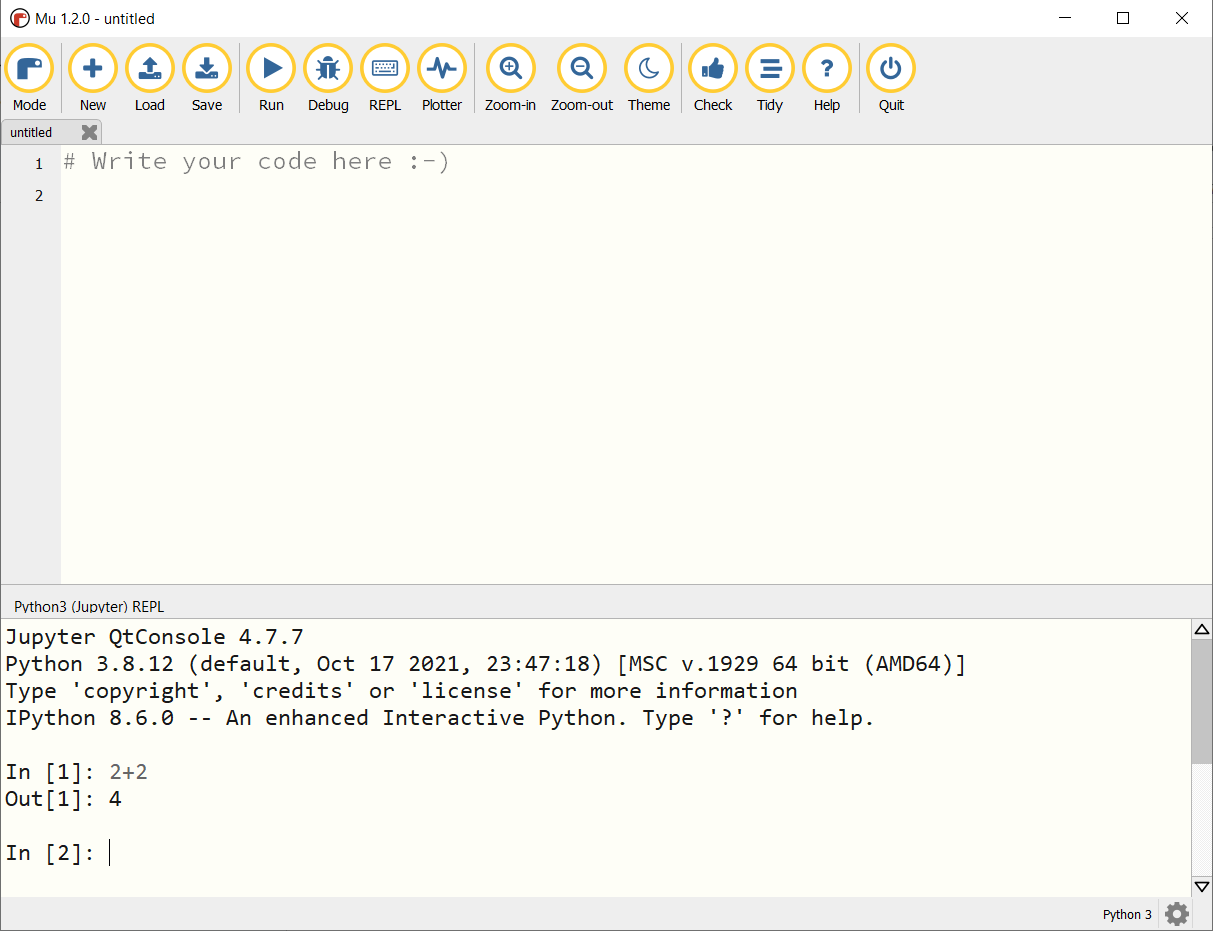
print(‘Hello, world!’)

And that’s it! You’re all set to start programming with Python and use 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.

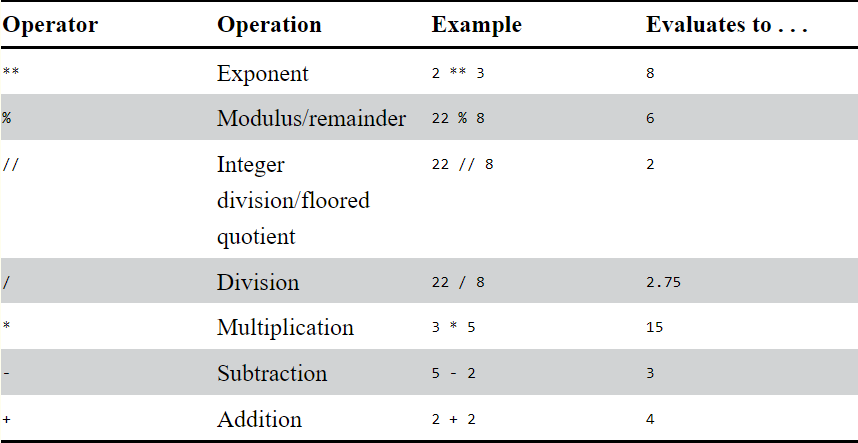




### 3. 1 Expressions

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]](#footnote-3)



### 3.2 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*).

### 3.3 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.

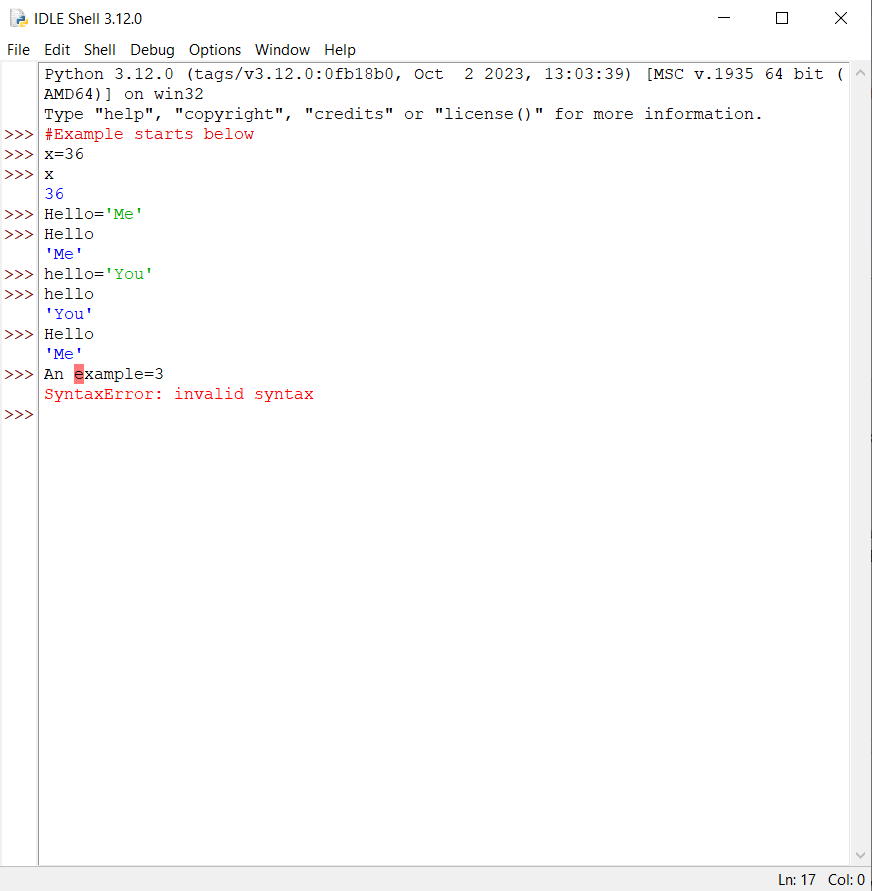


In the screenshot above, there are three variables defined: x, Hello and hello. They all have their own value, but two data types are being used. Me and You are both strings. You can recognize this by the single quote that is used. The number 36 is an integer.

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 down Hello=’You’, it would no longer print ‘Me’ as 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.

### 3. 4 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.

The above can be a bit overwhelming, so let’s bring everything we mentioned above together in Module 2 and create your first games!

# Module 2: Your first games

**For this module, make sure you:**

* Make sure you understand everything from Module 1. We will continue with the Python basics that we’ve learned.
* Read chapter 2 of *Automate the Boring Stuff.*[[5]](#footnote-4) This chapter will also cover some of the same concepts (booleans, comparison operators, importing modules, etc.) and has practice questions.
* Read module 2 of this guide completely, before working with the learning materials. This way, you know what is expected of you in this module and what the exercises are.
* Read and work through chapter 3 of *Invent Your Own Computer Games With Python.*[[6]](#footnote-5) Be able to understand the concepts covered in those chapters.

**Further learning (optional):**

* If you want to do more reading and practising on the basics, check out: [Python Tutorial (w3schools.com)](https://www.w3schools.com/python/default.asp). Here you can check out multiple assignments and try out the basics for yourself.
* Chapter 3 of Michael Dawsons, *Python Programming for the Absolute Beginner.*

## Read, create, and review

It is time to create your first games! To review what you have done and learned, it is important to write down your process. So, for example, if an error occurs, make sure to get a screenshot of it so you can discuss it in the group on Discord or during a monthly meeting. Also write down what you have done to fix the problem.

When learning something new, repetition is important. Try to remember to review what you have done and see if you can recognise and understand which variables, data types, statements, and functions you have used. Are you able to explain what every line of code does, in your own words?

Moreover, you can discuss your games with other members of the study group in the Discord channels. For example: What errors did occur? And how did you solve them?

## Getting started: Creating a Flowchart

We will follow Al Sweigarts ‘Invent your own computer games with Python’ again. In [chapter 3](https://inventwithpython.com/invent4thed/chapter3.html) you will learn more about functions, the ***boolean*** data type and about ***flow control statements***. Also, we will import a module, like we have seen in the expert session with Daniel (see the slides in Discord, main channel). In this case, we will import the ***Random module***. By importing this module, the program can call the randint() function. This function will come up with a random number for the player to guess.

Let’s start with Guess the Number! Before writing the code for this game, look at the figure below. This is how the output will look like after you run your code and play the Guess the Number game. The player’s input is marked in bold.

Before you write this code, create a flowchart for the game.

Answer the following question: What is it exactly that you will tell your computer to do?

Hello! What is your name?  
**Albert**Well, Albert, I am thinking of a number between 1 and 20.  
Take a guess.  
**10**Your guess is too high.  
Take a guess.  
**2**Your guess is too low.  
Take a guess.  
**4**Good job, Albert! You guessed my number in 3 guesses!

We highly recommend creating a flowchart for your games, so you’ll understand your code better. See for example figure 2-1 in chapter 2 of *Automate the Boring Stuff* or Daniels flowchart on slide 7 (expert session 1).

## Recreate Guess the Number

Now that you have made a flowchart, it is time to have a look at chapter 3 of *Invent Your Own Computer Games With Python.* When working through the chapter, remember: Don’t copy the code, but type every line down by yourself in IDLE or Mu and save your file. Run it and see what happens. Then closely read the chapter with your program at hand.

Both IDLE and Mu will give you some hints when writing your code. They show different colouring for functions and statements and provide hints on closing your functions and strings. They also allow you to press tab to insert four spaces or provide the spaces automatically. You can try both IDLE and Mu out and see what suits you best.

Don’t forget to review what you’ve done, as explained at the beginning of this module.

## Recreate Rock, Paper, Scissors

With similar principles that you have learned and used in the Guess the Number game, you can create another game: Rock, Paper, Scissors. It is recommended to (re)read [chapter 2](https://automatetheboringstuff.com/2e/chapter2/) from ‘Automate the Boring Stuff’ and recreate the game, but you can also skip to: “A Short Program: Rock, Paper, Scissors”[[7]](#footnote-6). Again, start with creating a flowchart for the game, and type the whole code by yourself afterwards. Do not copy it from the screen.

Below you can see the output of Rock, Paper, Scissors. The lines in bold are the input that the player gives to the computer, to tell their move.

ROCK, PAPER, SCISSORS  
0 Wins, 0 Losses, 0 Ties  
Enter your move: (r)ock (p)aper (s)cissors or (q)uit  
**p**PAPER versus...  
PAPER  
It is a tie!  
0 Wins, 1 Losses, 1 Ties  
Enter your move: (r)ock (p)aper (s)cissors or (q)uit  
**s**SCISSORS versus...  
PAPER  
You win!  
1 Wins, 1 Losses, 1 Ties  
Enter your move: (r)ock (p)aper (s)cissors or (q)uit  
**q**

Don’t forget to review what you’ve done, as explained at the beginning of this module. Identify which variables, data types, statements and functions you have used. Try to explain what errors occurred (if any) and how to solve them. Explain briefly in your own words what everly line of code does.

# Module 3: Functions, Loops, and Debugging

**Recap of module 2**

It is advised to create a flow chart for every script or game you create, especially when you are just getting started with Python. This way, you make it obvious what you want your program to do, like skip (else, if or elif statements) or repeat instructions (while and for loops). With the help of the Boolean values True or False, the computer knows how to handle parts of your code.

In the first two modules, we have seen various built-in functions like print() and input(). Python also has a standard library that contains a set of modules that you can import in your code. For example the random module. This module has been used to create the first two games.

**For this module, make sure you:**

* Read module 3 in the guide. Make sure you understand the concepts that will be used in this module and that you know what is expected of you.
  + If needed, read the summaries at the end of chapter 2 of *Automate the Boring Stuff*[[8]](#footnote-7) and chapter 3 of *Invent Your Own Computer Games With Python.*[[9]](#footnote-8)
  + If you haven’t done so already, it can be wise to make notes while working through the modules. This can be your own summary.
* Read chapter 5 of *Invent Your Own Computer Games With Python*[[10]](#footnote-9)
* Read chapter 6 of *Invent Your Own Computer Games With Python[[11]](#footnote-10)* and chapter11 of *Automate the Boring Stuff* [[12]](#footnote-11) on debugging code.

**Further learning (optional):**

* If you want to do more reading and practising on the basics, check out: [Python Tutorial (w3schools.com)](https://www.w3schools.com/python/default.asp). Here you can check out multiple assignments and try out the basics for yourself.
* You can reread Chapter 3 of Michael Dawsons, *Python Programming for the Absolute Beginner*, about while loops and else/elif clauses, if necessary.
* Want to try out some commands that you can use in the command prompt on Windows? Check out the: [Command Challenge! (cmdchallenge.com)](https://cmdchallenge.com/)

## Dragon Realm

In part 1 of this module, you will create another game: Dragon Realm ([chapter 5](https://inventwithpython.com/invent4thed/chapter5.html)). In this game, the player is in a land full of dragons. The dragons all live in caves with their large piles of collected treasure. Some dragons are friendly and share their treasure. Other dragons are hungry and eat anyone who enters their cave. The player approaches two caves, one with a friendly dragon and the other with a hungry dragon, but doesn’t know which dragon is in which cave. The player must choose between the two.

To make this game, you will have to import libraries, create your own functions and use *else/if* and *while* loops. A new function (the sleep() function) is introduced, and you will work with the and, or and not Boolean operators again.

Before you read chapter 5, create a flowchart for the game, based on the output below.

This is what Dragon Realm looks like when it’s run. The player’s input is in bold.

You are in a land full of dragons. In front of you, you see two caves.  
In one cave, the dragon is friendly and will share his treasure with you. The other dragon is greedy and hungry, and will eat you on sight.   
Which cave will you go into? (1 or 2)  
**1**You approach the cave...  
It is dark and spooky...  
A large dragon jumps out in front of you! He opens his jaws and...  
Gobbles you down in one bite!   
Do you want to play again (yes or no) **no**

After you made your flowchart, open chapter 5 and work through it. Did your flowchart look the same as the one provided in the chapter? What were the differences? Try to figure out what would and wouldn’t work in the game if you have missed a step.

### Adapting Dragon Realm

Let’s face it: it’s hard to compete with a game that has a dragon in it! But here’s a little challenge. Use the same code of Dragon Realm, but alter the text completely, to make this game more suitable for a digital archive or preservation themed game. Or try and add another choice in the game. Do not forget to share this on the Discord server with the other members!

## Debugging Code

Part 2 of this module revolves around debugging. As explained in the introduction, you have to be very precise in telling the computer what to do. When you make a tiny mistake or a typo, like forgetting to end a string with a quote, the computer will give you an error message. These errors are called bugs. In Python the computer will indicate where the program stopped running. In this module, you will learn more about bugs and how to fix them. This might not be a very fun part of the module, but it can be very helpful to understand Python better. A detailed explanation can be found in Al Sweigarts ‘Invent Your Own Computer Games with Python’, [chapter 6](https://inventwithpython.com/invent4thed/chapter6.html) and [chapter 11](https://automatetheboringstuff.com/2e/chapter11/) in ‘Automate the Boring Stuff’, of the same author.

Chapter 6 covers the types of bugs that can occur and how to use a debugger. This is a program that lets you go through your code one line at a time in the same order that Python executes each instruction. There’s a bit of overlap with chapter 11, but the first half of chapter 11 explains how you can get more detailed information about the bugs and how to use the logging module that enables you to display log messages on your screen as your program runs. It helps you understand what’s happening in your program and in what order.

### Try to Fix the Code!

Now, apply what you’ve just learned. [Here](https://www.codingame.com/playgrounds/64843/beginner-python-concepts/debugging) you can find a code with Syntax Errors. Can you fix them? If you’re up for it, try to fix the Logic Errors as well. For the exercise with the syntax errors, make notes during the progress. Before running and altering the code, have a look at it. Can you already identify some of the bugs?

# 

# Module 4: Lists and Dictionaries

**For this module, make sure you:**

* Read the guide and make you sure you understand everything
* Read chapters 4 and 5 of *Automate the Boring Stuff.*[[13]](#footnote-12)
* Read and work through chapter 7, 8 and 9 of *Invent Your Own Computer Games With Python.*[[14]](#footnote-13) Be able to understand the concepts covered in those chapters.

**Further learning (optional):**

* If you want to do more reading and practising on the concepts that were described in the work for this module, check out: [Python Tutorial (w3schools.com)](https://www.w3schools.com/python/default.asp). Here you can check out multiple assignments and try out the basics for yourself.
* Chapter 5 of Michael Dawsons, *Python Programming for the Absolute Beginner*.

In this module, we will create a hangman. Because this is a more advanced game, you first have to start designing the hangman, before you can start writing the code for the game. This module covers two chapters ([7](https://inventwithpython.com/invent4thed/chapter7.html) and [8](https://inventwithpython.com/invent4thed/chapter8.html)) of Al Sweigarts ‘Invent Your Own Computer Games with Python’, and there’s a third chapter dedicated to extending the hangman game, if you would like to do that as well ([chapter 9](https://inventwithpython.com/invent4thed/chapter9.html)).

# Module 5: From A(I) to B(agels)

**For this module, make sure you:**

* Read the guide and make you sure you understand everything
* Read chapter of *Automate the Boring Stuff.*[[15]](#footnote-14)
* Read and work through chapter 10 of *Invent Your Own Computer Games With Python.*[[16]](#footnote-15) Be able to understand the concepts covered in those chapters.

**Further learning (optional):**

* If you want to do more reading and practising on the concepts that were described in the work for this module, check out: [Python Tutorial (w3schools.com)](https://www.w3schools.com/python/default.asp). Here you can check out multiple assignments and try out the basics for yourself.
* Chapter 6 of Michael Dawsons, *Python Programming for the Absolute Beginner*.

# Module 6: Coordinates and a Treasure Hunt

**For this module, make sure you:**

* Read the guide and make you sure you understand everything
* Read and work through chapter 12 and 13 of *Invent Your Own Computer Games With Python.*[[17]](#footnote-16) Be able to understand the concepts covered in those chapters.

**Further learning (optional):**

* If you want to do more reading and practising on the concepts that were described in the work for this module, check out: [Python Tutorial (w3schools.com)](https://www.w3schools.com/python/default.asp). Here you can check out multiple assignments and try out the basics for yourself.
* Chapter 14: Caesar Cipher

# Module 7: Advanced AI and Coordinates

**For this module, make sure you:**

* Read the guide and make you sure you understand everything
* Read and work through chapter 15 and 16 of *Invent Your Own Computer Games With Python.*[[18]](#footnote-17) Be able to understand the concepts covered in those chapters.

**Further learning (optional):**

* If you want to do more reading and practising on the concepts that were described in the work for this module, check out: [Python Tutorial (w3schools.com)](https://www.w3schools.com/python/default.asp). Here you can check out multiple assignments and try out the basics for yourself.

# Module 8: Pygame

**For this module, make sure you:**

* Read the guide and make you sure you understand everything
* Read and work through chapter 17, 18, 19, 20 and 21 of *Invent Your Own Computer Games With Python.*[[19]](#footnote-18) Be able to understand the concepts covered in those chapters.
* Or read and recreate the pizza game from chapter 11, Michael Dawsons, *Python Programming for the Absolute Beginner*.

1. <https://inventwithpython.com/invent4thed/> [↑](#footnote-ref-0)
2. [Automate the Boring Stuff with Python](https://automatetheboringstuff.com/) [↑](#footnote-ref-1)
3. [Python Programming for the Absolute Beginner (3rd Edition) (nibmehub.com)](https://nibmehub.com/opac-service/pdf/read/Python%20Programming%20for%20the%20Absolute%20Beginner-%203rd%20Edition.pdf) [↑](#footnote-ref-2)
4. <https://automatetheboringstuff.com/2e/chapter1/> [↑](#footnote-ref-3)
5. [Automate the Boring Stuff with Python](https://automatetheboringstuff.com/2e/chapter2/) [↑](#footnote-ref-4)
6. <https://inventwithpython.com/invent4thed/> [↑](#footnote-ref-5)
7. Near the end of chapter 2 of Automate the Boring Stuff. [↑](#footnote-ref-6)
8. [Automate the Boring Stuff with Python](https://automatetheboringstuff.com/2e/chapter2/) [↑](#footnote-ref-7)
9. <https://inventwithpython.com/invent4thed/> [↑](#footnote-ref-8)
10. [Chapter 5 - Dragon Realm (inventwithpython.com)](https://inventwithpython.com/invent4thed/chapter5.html) [↑](#footnote-ref-9)
11. [Chapter 6 - Using the Debugger (inventwithpython.com)](https://inventwithpython.com/invent4thed/chapter6.html) [↑](#footnote-ref-10)
12. [Automate the Boring Stuff with Python](https://automatetheboringstuff.com/2e/chapter11/) [↑](#footnote-ref-11)
13. [↑](#footnote-ref-12)
14. <https://inventwithpython.com/invent4thed/> [↑](#footnote-ref-13)
15. [↑](#footnote-ref-14)
16. <https://inventwithpython.com/invent4thed/> [↑](#footnote-ref-15)
17. <https://inventwithpython.com/invent4thed/> [↑](#footnote-ref-16)
18. <https://inventwithpython.com/invent4thed/> [↑](#footnote-ref-17)
19. <https://inventwithpython.com/invent4thed/> [↑](#footnote-ref-18)