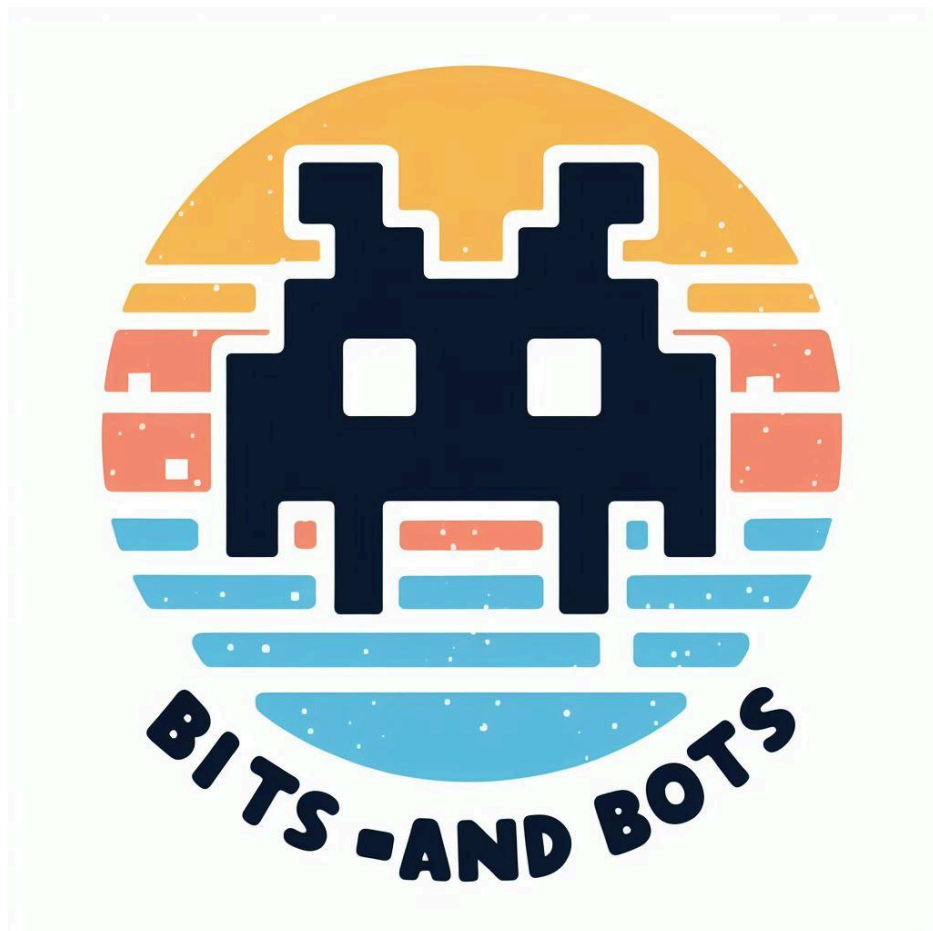


# Guide to building games with Python

With the Bits and Bots study group



# Table of contents

<b>Introduction</b>	<b>3</b>
Module overview	4
Module 1: Installation and Python basics	4
Module 2: Your first games	4
Module 3: Functions, Loops and Debugging	5
Module 4: Lists and Dictionaries	5
Module 5: From A(I) to B(agels)	5
Module 6: Coordinates and a Treasure Hunt	5
Module 7: Advanced AI and Coordinates	5
Module 8: Pygame	6
External reading and learning materials	7
<b>Module 1: Installation and Python basics</b>	<b>8</b>
Installing Python	8
Installing an IDE (Integrated Development Environment)	10
Basic programming concepts	13
Expressions	14
Data types	14
Variables	15
Functions	17
<b>Module 2: Your first games</b>	<b>18</b>
Read, Create, and Review	18
Getting started: Creating a Flowchart	18
Recreate Guess the Number	19
Recreate Rock, Paper, Scissors	20
<b>Module 3: Functions, Loops, and Debugging</b>	<b>21</b>
Recap of Module 2	21
Dragon Realm	22
Adapting Dragon Realm	22
Debugging Code	23
Try and fix the code	23
<b>Module 4: Lists and Dictionaries</b>	<b>24</b>
Recap of Module 3	24
Hangman	25
GitHub	25
<b>Module 5: From A(I) to B(agels)</b>	<b>28</b>
Recap of Module 4	28
Tic-Tac-Toe	29

The Bagel Deduction Game	29
Summertime Tips	30
<b>Module 6: Coordinates and a Treasure Hunt</b>	<b>31</b>
Recap of module 5	31
Cartesian coordinate system	32
Sonar Treasure Hunt	32

# 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
<b>Summer recess</b>	
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](#).

### Module 1: Installation and Python basics

In this module we will:

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

### Module 2: Your first games

In this module we will:

1. Build our first games: Guess the Number and Rock, paper, scissors
2. 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:

1. Learn how to make your own functions
2. Learn how to work with loops
3. 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:

1. Design a program with flowcharts
2. Learn about lists
3. Learn about dictionaries
4. Make a hangman game

Optional:

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

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

In this module we will:

1. Work with AI
2. Learn about augmented assignment operators
3. Learn about string interpolation

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

In this module we will:

1. Learn coordinates
2. Learn about data structures

## **Module 7: Advanced AI and Coordinates**

In this module we will:

1. Work with a more advanced AI
2. Work with the Cartesian coordinate system

## Module 8: Pygame

In this module we will:

1. Create a graphics window
2. 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*.<sup>1</sup> Be able to understand the concepts covered in those chapters.

**Further learning:**

- For extra understanding: read chapter 1 of *Automate the Boring Stuff*.<sup>2</sup> 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*.<sup>3</sup>
- If you want to do more reading and practising on the basics, check out: [Python Tutorial \(w3schools.com\)](https://www.w3schools.com/python/). 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.

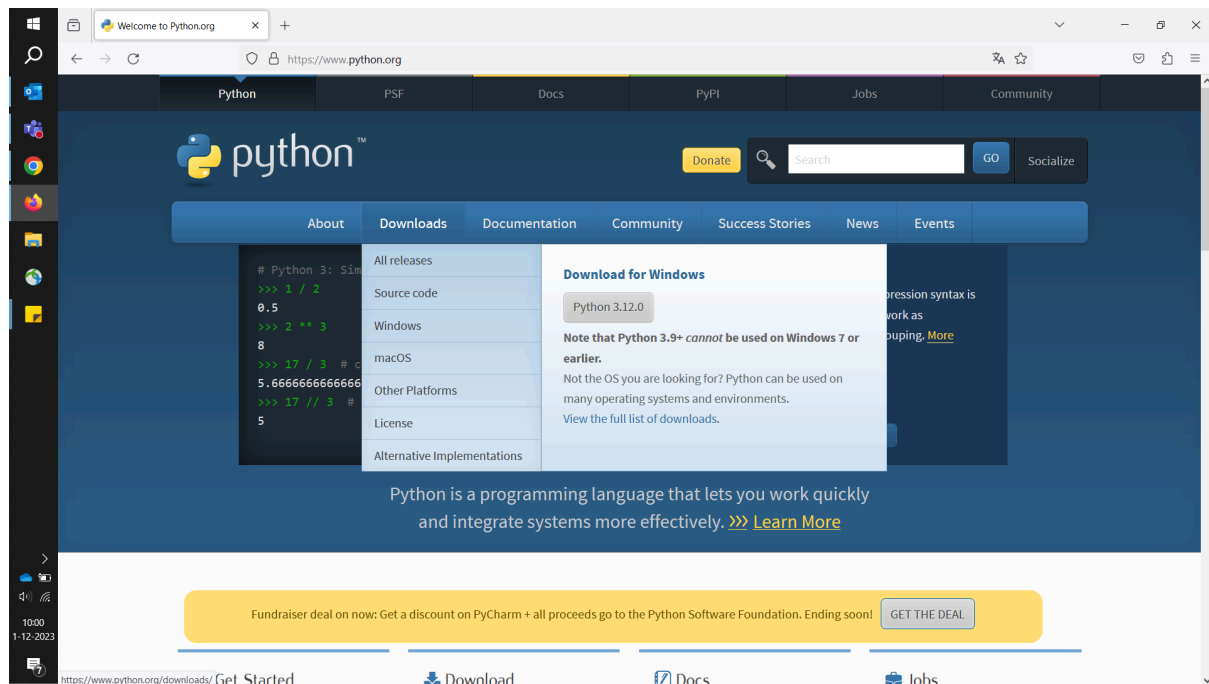
---

<sup>1</sup> <https://inventwithpython.com/invent4thed/>

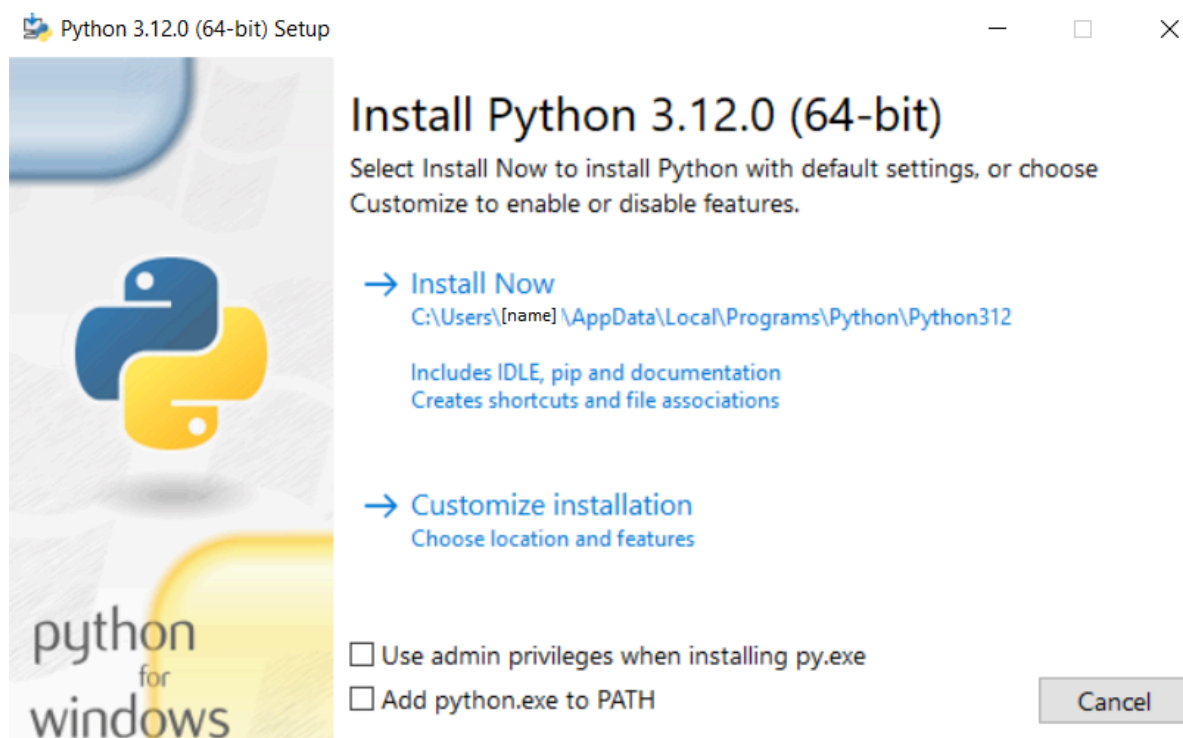
<sup>2</sup> [Automate the Boring Stuff with Python](#)

<sup>3</sup> [Python Programming for the Absolute Beginner \(3rd Edition\) \(nibmehub.com\)](#)

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

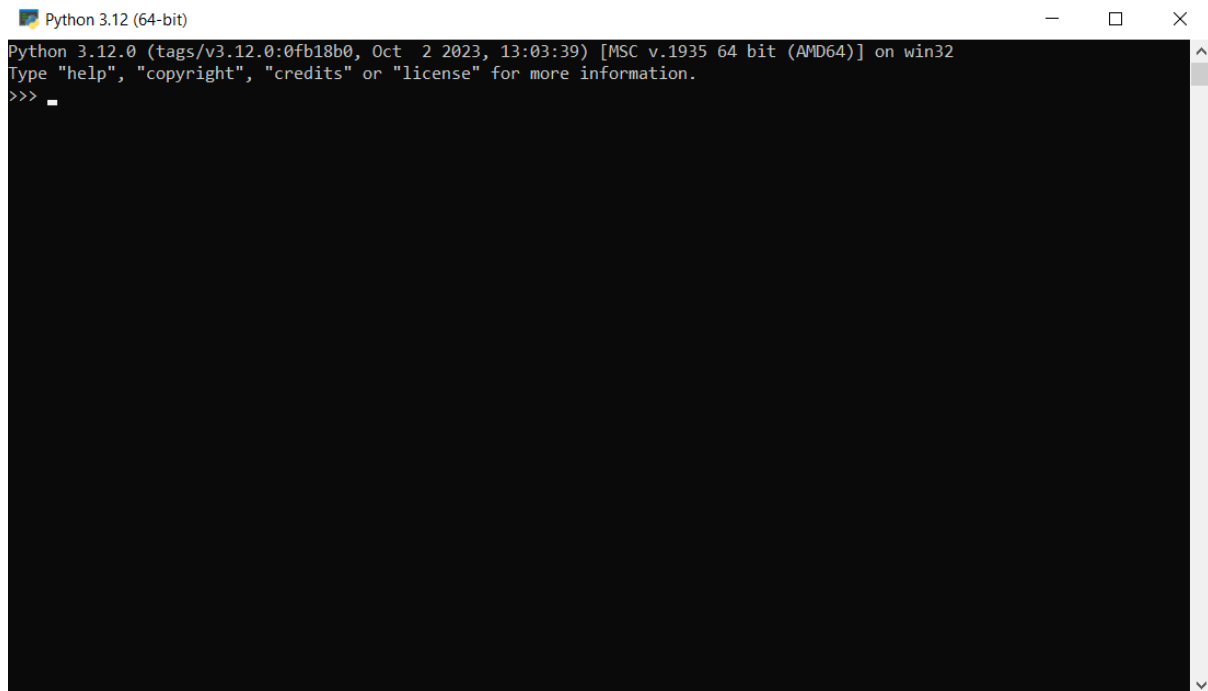


Next, check your downloads folder. Double click the Python installer. The following screen should appear:



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:



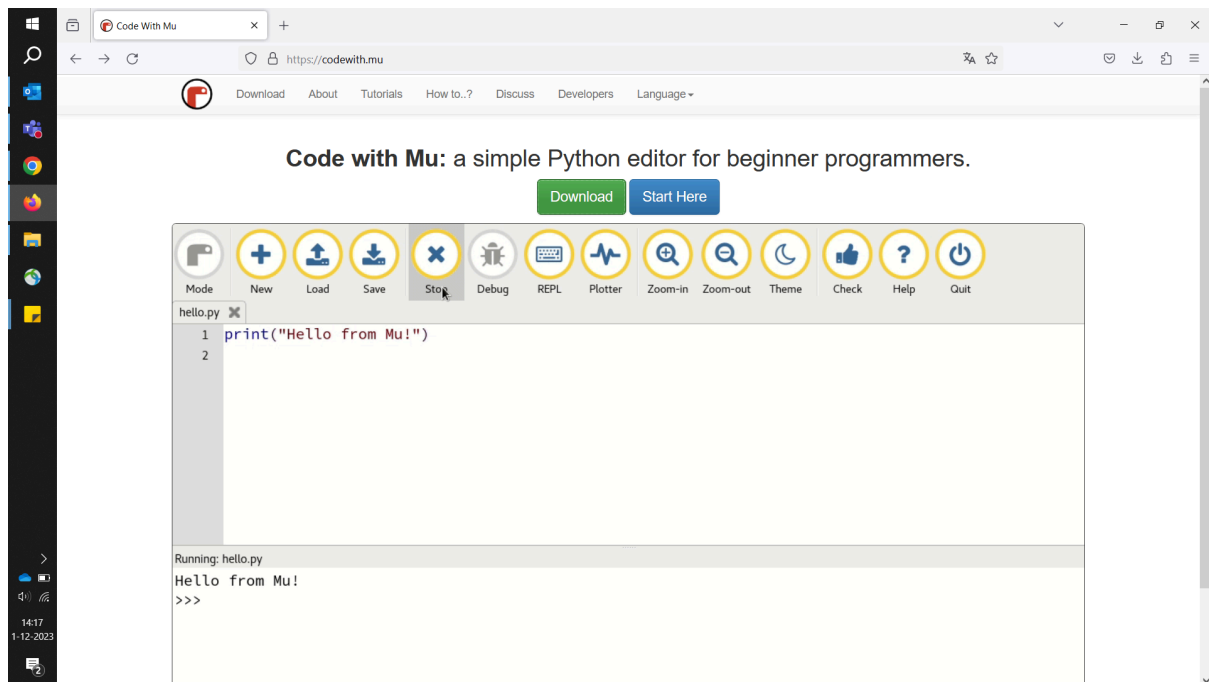
Now you'll be able to run Python programs on your computer. If you want to enter your programs, you'll need a different software. In the following step, we will explain what your options are and how you can install software that suits your needs best.

## 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](#)'. 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.

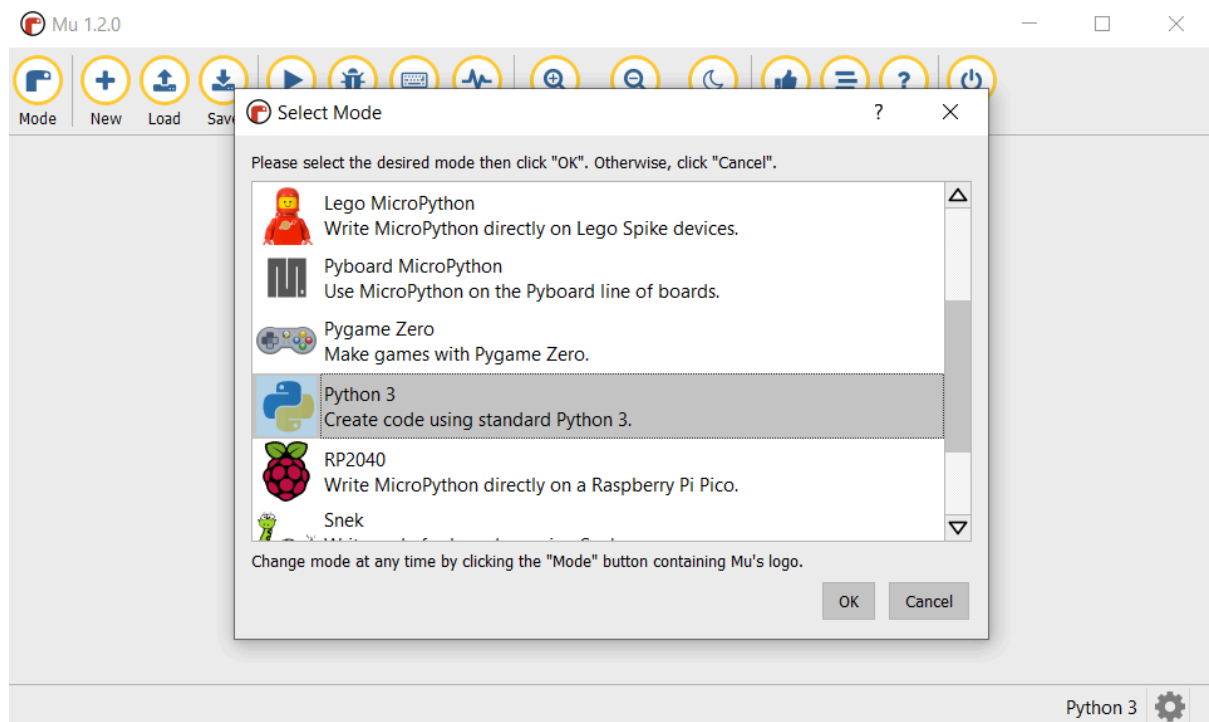
To install Mu, go to <https://codewith.mu/> and click the green download button as shown below. With the redirect to the next page you can choose different operating systems. We show Windows here.



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.



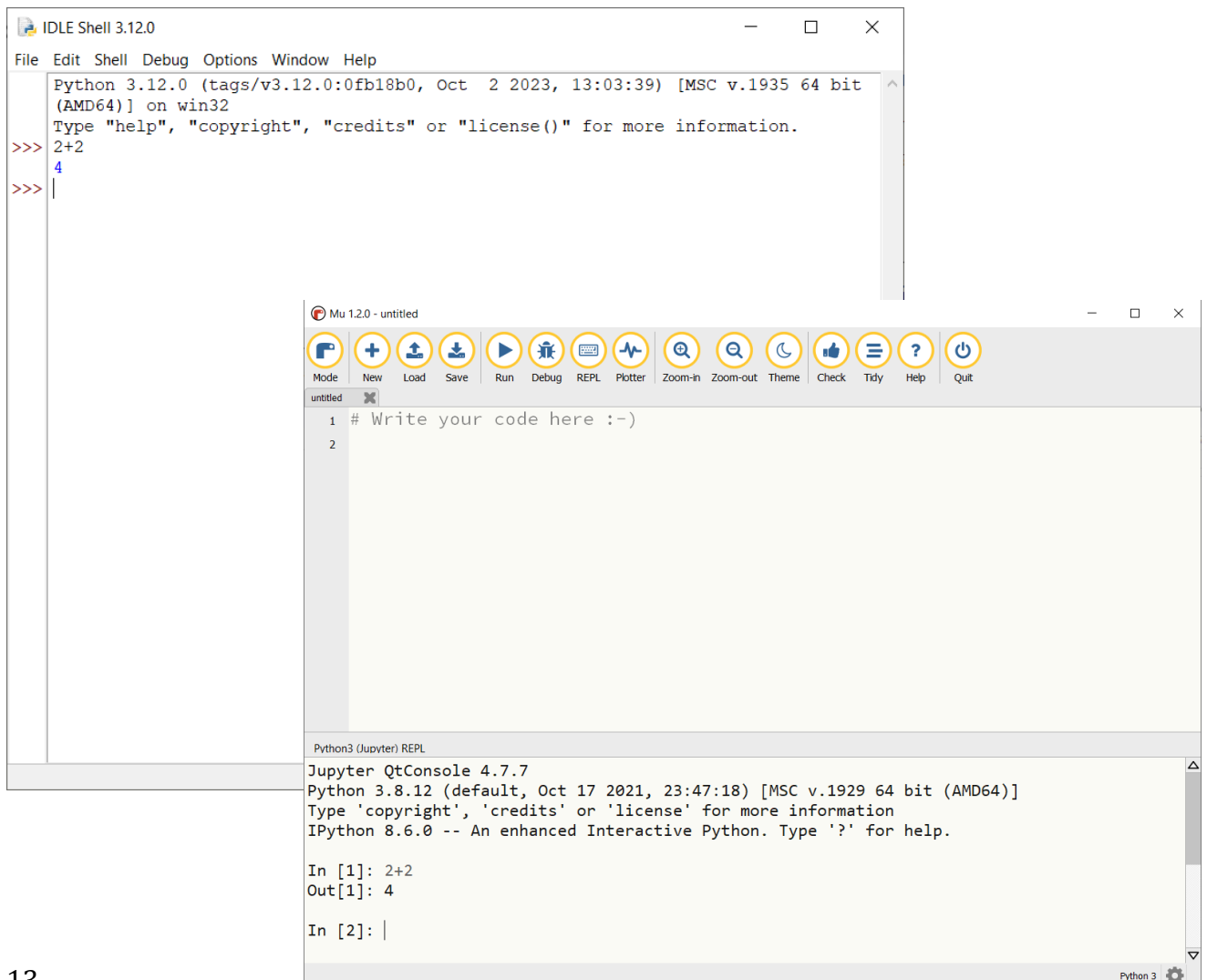
**Try it out! The `In [1]:` part is called a prompt. You can give your computer an instruction here. Type the following, to give your first command:**

```
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.



## 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.<sup>4</sup>

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

## 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*).

---

<sup>4</sup> <https://automatetheboringstuff.com/2e/chapter1/>

## 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.



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> #Example starts below
>>> x=36
>>> x
36
>>> Hello='Me'
>>> Hello
'Me'
>>> hello='You'
>>> hello
'You'
>>> Hello
'Me'
>>>
```

Ln: 15 Col: 0

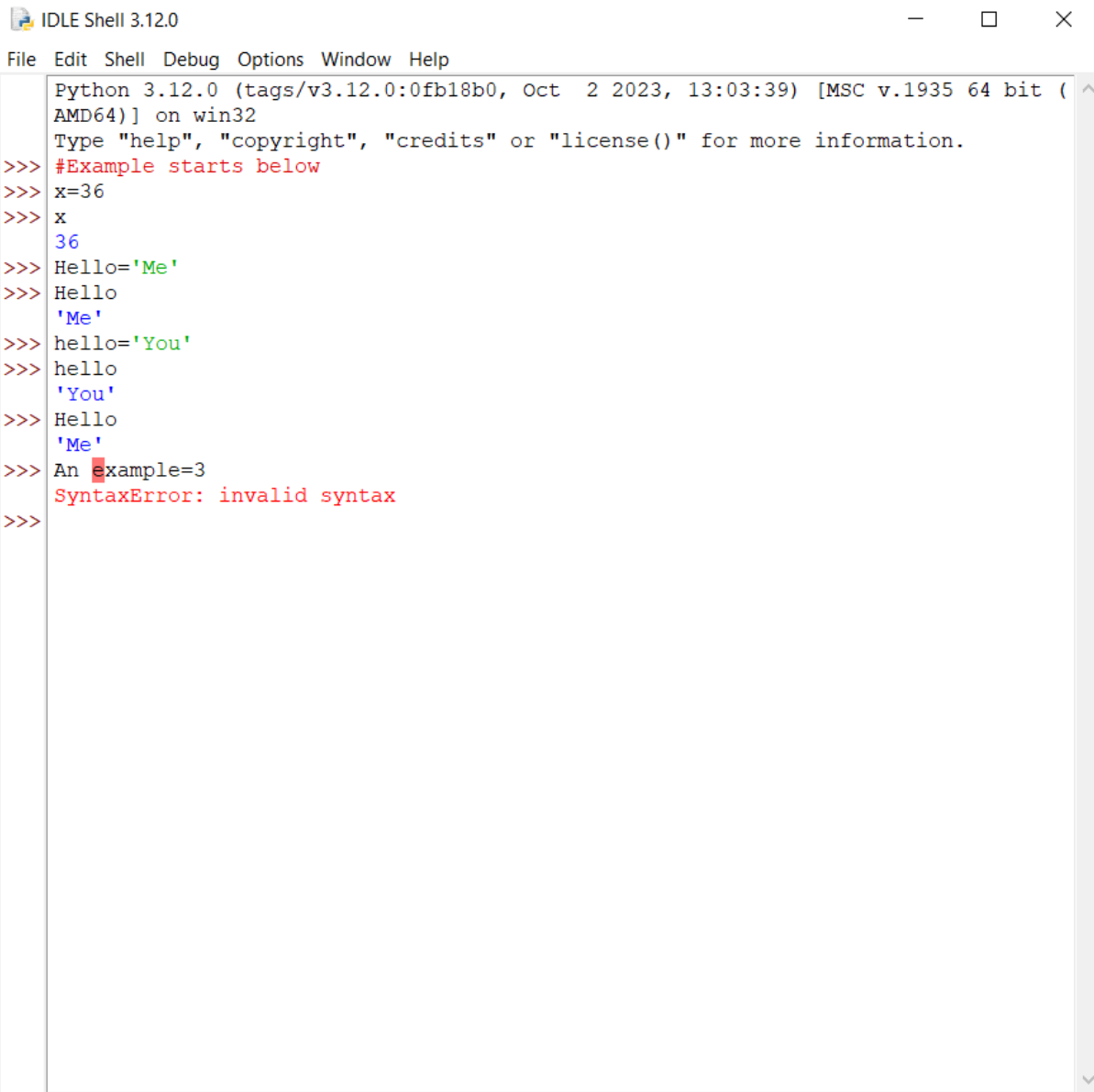
In the screenshot above, there are three variables defined: x, Hello and hello. They all have their own value, but two data types are being used. Me and You are both strings. You can recognize this by the single quote that is used. The number 36 is an integer.



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.



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> #Example starts below
>>> x=36
>>> x
36
>>> Hello='Me'
>>> Hello
'Me'
>>> hello='You'
>>> hello
'You'
>>> Hello
'Me'
>>> An example=3
SyntaxError: invalid syntax
>>>
```

Ln: 17 Col: 0

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.

## 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 read the learning materials provided carefully. Do not forget the additional learning materials as well.

## 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*.<sup>5</sup> 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*.<sup>6</sup> 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/). Here you can check out multiple assignments and try out the basics for yourself.
- Chapter 3 of Michael Dawson's, *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 Sweigart's 'Invent your own computer games with Python' again. In [chapter 3](#) 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

---

<sup>5</sup> [Automate the Boring Stuff with Python](https://automatetheboringstuff.com/)

<sup>6</sup> <https://inventwithpython.com/invent4thed/>

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](#) from 'Automate the Boring Stuff' and recreate the game, but you can also skip to: "A Short Program: Rock, Paper, Scissors"<sup>7</sup>. 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 every line of code does.

---

<sup>7</sup> Near the end of chapter 2 of Automate the Boring Stuff.

# Module 3: Functions, Loops, and Debugging

**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*<sup>8</sup> and chapter 3 of *Invent Your Own Computer Games With Python*.<sup>9</sup>
  - 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*<sup>10</sup>
- Read chapter 6 of *Invent Your Own Computer Games With Python*<sup>11</sup> and chapter 11 of *Automate the Boring Stuff*<sup>12</sup> 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/). 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://www.cmdchallenge.com/)

## 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.

---

<sup>8</sup> [Automate the Boring Stuff with Python](https://www.automatetheboringstuff.com/)

<sup>9</sup> <https://inventwithpython.com/invent4thed/>

<sup>10</sup> [Chapter 5 - Dragon Realm \(inventwithpython.com\)](https://inventwithpython.com/chapter5/)

<sup>11</sup> [Chapter 6 - Using the Debugger \(inventwithpython.com\)](https://inventwithpython.com/chapter6/)

<sup>12</sup> [Automate the Boring Stuff with Python](https://www.automatetheboringstuff.com/)

## Dragon Realm

In part 1 of this module, you will create another game: Dragon Realm ([chapter 5](#)). 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](#) and [chapter 11](#) 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 and fix the code

Now, apply what you've just learned. [Here](#) 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:

- Recap what you have learned in the last module. Check your own notes, or read the summaries provided at the end of the chapters of Al Sweigart books.
- Read the guide and make sure you understand everything. Make sure to start with the GitHub assignment first.
- Read chapters [4](#) (lists) and [5](#) (dictionaries) of *Automate the Boring Stuff*.
- Read and work through chapter 7, 8 and 9 of *Invent Your Own Computer Games With Python* while following the guide. 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/). Here you can check out multiple assignments and try out the basics for yourself.
- Chapter 5 of Michael Dawson's, [Python Programming for the Absolute Beginner](#).

## Recap of Module 3

In the previous module, you have been working with new functions and loops. The `while` loop will only run if a certain condition is `true`. With the `sleep()` function you are able to build up the suspense in a game, as we have seen in Dragon Realm. Furthermore, you have learned how to create your own function. Just a reminder: A function is a small program within your code, that only runs when the function is called. By using functions you create small pieces of code to structure your program into sections. If you understand how functions work and how to use them, you can save yourself some time, because you can reuse functions. Also, it is easier to fix bugs.

Using a debugger is great to understand what your program does, because it lets you step through your code line by line. The `logging` module facilitates you to have a closer look at your code when it's running. Furthermore, we have seen three type of bugs:

- Syntax errors: Typos in the code like forgetting a quotation mark or a bracket.
- Runtime errors: Bugs that happen while the program is running. The traceback gives an error message showing the line containing the problem
- Semantic errors: These bugs don't crash the program, but prevent it from doing what the programmer intended it to do. The program could crash later on, but it won't be immediately obvious where the semantic bug happened.

# Hangman

In this module, we will create a hangman. Because this is a more advanced game, you first have to start designing the hangman in a flowchart, before you can start writing the code for the game.

This module covers two chapters of Al Sweigarts 'Invent Your Own Computer Games with Python':

- Chapter [7](#) : flowchart and ASCII art and
- Chapter [8](#): lists and methods
- There is also a third chapter dedicated to extending the hangman game, if you would like to do that as well: [chapter 9](#) (dictionary data type). An extra challenge in that case would be to make it more digital preservation themed. That would also make it more fun to share in the group, but also with your colleagues.

# 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             | • Educational resources |
| • Fan fiction                      | • Reading lists         |
| • Zines                            | • Datasets              |
| • Community/crowdsourcing projects | • Blog posts            |
| • Finding file samples             | • Documentation         |

harry-potter-gen-z / book\_1 / chapter\_1.md

lounsberry Correct tiny grammar mistake 9208aa8 · 7 months ago History

Preview Code Blame 210 lines (106 loc) · 20.7 KB Code 55% faster with GitHub Copilot Raw

## Chapter One

### THE BOY WHO WASN'T UNALIVED

Mr. and Mrs. Dursley, of number four, Privet Drive, liked flexing that they were very basic, thank u. Tbh they were the last people you'd think would be sus, because they were all fax no printer.

Mr. Dursley was adulting at a firm called Grunnings, which made drills.

He was a dummy thiccc (w/ three Cs) man with hardly any neck, although he had an absolute unit of a mustache. Mrs. Dursley was a total Karen with zero chill and had hellas neck, which came in very useful when she was stalking her neighbours and not minding her own.

The Dursleys had a future incel of a son named Dudley who they thought was the main character. The Dursleys were mostly thriving, but they also had lowkey tea which didn't pass the vibe check and their greatest fear was to get called out and cancelled. They were girlbossing too close to the sun and didn't think their clout could bounce back if their fam, the Potters, were revealed. Milf Lily Potter was Mrs. Dursley's sis, but Mrs. D had gone ghost; irl (no cap) Mrs. D fronted she didn't have a sis, because Lil and her deadbeat mans were straight up cringe. If the neighbors ever peeped the Potters, it'd be a big yikes. Lowkey the Dursleys knew the Potters had their own crotch goblin, too, but they'd never peeped. This bb was fr a solid reason 2 keep the in-laws yote; they didn't want Dudley mixing with a gross being like that.

When Mr. and Mrs. Dursley woke up on the dull, gray (fight me) Tuesday our lore opens, the cloudy overlay didnt vibe like strange and mysterious things would be happening all over the country. Mr. Dursley hummed as he picked out his most boring tie for work, and Mrs. Dursley spilled the tea as she was tryna put a screaming Dudley into his heckin high chair.

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

cooking-patterns / desserts / cheesecake.md

applemac Add some directory structure

Preview Code Blame 14 lines (9 loc) · 476 Bytes Code 55% faster with GitHub Copilot

## Cheesecake

Cheesecake is a dessert dish that consists, most often, of a crust, a cheese filling and a topping.

### Preparation steps

// TODO

### Recipe examples

- [Jessica Merchant](#)
- [Martha Stewart](#)
- [Tyler Florence](#)

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.

For this module, we want you to:

1. Create a GitHub account if you do not have one.
2. Follow this tutorial: <https://github.com/skills/introduction-to-github>
3. Try one of the following things with [the Bits and Bots repository](#)<sup>13</sup>:
  - Add your own game. You can place it in its own repository and link to it in the text.
  - Add something to the command prompt guide. You will notice it is a .md file. This means that it is written using Markdown. You can use [this website](#) to see how it works and to practise.
  - Add a batch script to the command prompt folder.
  - Help us improve the repository. You can do this by raising an issue.
    - Issues are suggested improvements, tasks or questions related to the repository. Issues can be created by anyone (for public repositories), and are moderated by repository collaborators. Each issue contains its own discussion thread. You can also categorise an issue with labels and assign it to someone.
    - What makes a good issue is being very clear, adding detail to it and if possible, to provide a solution for it.

---

<sup>13</sup> We will also be using this repository to add some guides from the in-depth sessions here so you can add to these to make them more extensive.

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

For this module, make sure you:

- Recap what you have learned in module 4. Check your own notes, or read the summaries provided at the end of chapters 7, 8 and 9 of 'Invent Your Own Computer Games with Python'.
- Read the guide and make you sure you understand everything
- Read and work through chapter 10 and 11 of *Invent Your Own Computer Games With Python*.<sup>14</sup> 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/). Here you can check out multiple assignments and try out the basics for yourself.
- Chapter 6 ('Functions: The Tic-Tac-Toe Game') of Michael Dawsons, *Python Programming for the Absolute Beginner*.<sup>15</sup>

## Recap of Module 4

In the previous module, you have created your GitHub account and added something to the Bits and Bots repository. By doing so, you are actively creating lasting resources for the study group that others can benefit from as well. GitHub is a very useful platform to share your work, get help with problems and offer solutions to other peoples' bugs. Furthermore, you have made the Hangman game in the previous module.

Before creating Hangman, you have experienced that it can be beneficial to make a flowchart beforehand. This way, you can make changes to your program more easily and identify problems by thinking about how the code works. In order to create the Hangman, you have used ASCII art. But also, two new data types are introduced: Lists and dictionaries. **Lists** are values that can contain other values. You can recognize a list in your code by the square brackets, `[]`. You can use methods while using lists. **Methods** are functions attached to a value. Two of methods from module 4 are repeated here: You made up words that can be guessed in the game (in a list) and typed all those words only separated with a space. The `split()` method evaluates to a list with each word in the string as a single list item. The `append()` method adds an argument to the end of the list.

---

<sup>14</sup> <https://inventwithpython.com/invent4thed/>

<sup>15</sup> [Python Programming for the Absolute Beginner \(3rd Edition\)](#)

Like lists, a **dictionary** is a mutable collection of many values. The dictionary is typed with braces `{ }` in a code. Dictionaries are, unlike lists, unordered and that's why they can't be sliced like lists. Dictionaries are useful because you can map one item (the key) to another (the value), as opposed to lists, which simply contain a series of values in order. Values inside a dictionary are accessed using square brackets just as with lists.

## Tic-Tac-Toe

Tic-Tac-Toe is normally played with two people. One player is X and the other player is O. Players take turns placing their X or O. If a player gets three of their marks on the board in a row, column, or diagonal, they win. When the board fills up with neither player winning, the game ends in a draw. For the game you will create, you won't be using any new concepts, but you will work with AI, because your opponent is the computer.

Read and work through [chapter 10](#) of *Invent Your Own Computer Games With Python*. Also try and compare the script with how the game is made in [Chapter 6](#) ('Functions: The Tic-Tac-Toe Game') of Michael Dawsons, *Python Programming for the Absolute Beginner*.

## The Bagel Deduction Game

This game is a deduction game in which the player tries to guess a random three-digit number, with no repeating digits, that is generated by the computer. After each guess, the computer gives the player three types of clues:

- Bagels: None of the three digits guessed is in the secret number
- Pico: One of the digits is in the secret number, but the guess has the digit in the wrong place.
- Fermi: The guess has a correct digit in the correct place.

The computer can give multiple clues, which are sorted in alphabetical order. If the secret number is 456 and the player's guess is 546, the clues would be "fermi pico pico". The "fermi" is from the 6 (correct place) and "pico pico" are from the 4 and 5 (correct digits, wrong place).

Read and work through [chapter 11](#) of *Invent Your Own Computer Games With Python*.

## Summertime Tips

Since this module has a somewhat longer timespan, here are some extra tips for the summer if you want more practice:

- Practising with GitHub

Practice with GitHub by placing all your games in either your own repository (and linking to it in ours) or placing them directly into the Bits and Bots repository. It is recommended that if you are hoping to make lots of changes to your game that you keep them within your own repository for better access and control.

- Adapt and improve

Try and adapt some games to make them directed to the field and add some extra functionality where you can.

- Adapt existing code

In the expert session by Remco van Veenendaal he demonstrated his [box](#) which was made by starting with code someone else made (the 3D box) and adapted it to his needs. Even adapting code can really help you get to grips with learning Python. In the future it can happen that you want to tweak something a bit more towards your own purpose instead of starting something from scratch.

Examples you can try:

- Try to adapt the [checksum batch script](#).
- Take inspiration from the [script Daniel Steinmeier](#) provided us when analysing Twitter exports. He used JSON files for this, if you have some JSON files, try to export some information from it. You can try and use chapter [16](#) (Working with CSV files and JSON data) of [Automating the Boring Stuff](#).

- Try some Practice Projects from one of the chapters of [Automate the Boring Stuff](#).

In the previous module you have started working with lists and dictionaries. Practise a bit more by completing the Practice Projects of chapter [4](#) (Lists) and [5](#) (Dictionaries and Structuring Data). You can find the practice projects at the end of each chapter.

# Module 6: Coordinates and a Treasure Hunt

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

- Recap what you have learned in module 5. Check your own notes, or read the summaries provided at the end of chapters 10 and 11 of 'Invent Your Own Computer Games with Python'.
- 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*.<sup>16</sup> 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/tutorial/). Here you can check out multiple assignments and try out the basics for yourself.
- Chapter 14: Caesar Cipher: The program in this chapter is not really a game, but you will learn how to convert normal English into a secret code.

## Recap of module 5

The Python concepts that were being used in module 5 were familiar; they've been introduced and adapted in earlier modules. The new games in module 5, Tic-Tac-Toe and The Bagel Deduction Game, have one thing in common: the computer (AI) is your opponent. Tic-Tac-Toe is a relatively simple game, because there are not as many moves possible as is the case with chess for example. The AI checks which move to make to win the game, or to block the opponent, which is quite a simple algorithm to follow for the computer.

The Bagels deduction game introduced some new functions and methods: `shuffle()`, `sort()` and `join()`.

- The `shuffle()` function is used to create the secret number that has to be guessed.
- The `sort()` method adds an extra difficulty to the game, as it sorts the clues. If the clue was 'Pico', 'Fermi', 'Pico', that would tell the player that the centre digit of the guess is in the correct position. Since the other two clues are both Pico, the player would know that all they have to do to get the secret number is swap the first and third digits.

---

<sup>16</sup> <https://inventwithpython.com/invent4thed/>



- The `join()` method is sort of like the opposite of the `split()` string method that we have used in module 4 (Hangman game). While `split()` returns a list from a split-up string, `join()` returns a string from a combined list.

## Cartesian coordinate system

Most of the programming doesn't require understanding a lot of maths. But sometimes it is needed to mark a certain position, for example on chess boards. Chapter 12 of *Invent Your Own Computer Games With Python* goes over some simple mathematical concepts that we will use in later games. In two-dimensional (2D) games, the graphics on the screen can move left or right and up or down. These games need a way to translate a place on the screen into integers the program can deal with. This is where the Cartesian coordinate system comes in. Coordinates are numbers that represent a specific point on the screen. These numbers can be stored as integers in your program's variables.

### Additional learning resources:

- To get some more information on the Cartesian coordinate system, check out [this easy to follow video](#).
- Chapter 12 also covers absolute values and the `abs()` function. W3schools has a [lesson](#) that you can follow and that also contains two examples for you to experiment with.

Now that you understand some of the basics, read and work through chapter [12](#).

### Python Turtle library

Python has a pre-installed library, Turtle, that can be used to create mini games and animations. Check out this [Python Turtle guide](#) and see how this is used in practice by creating a [turtle race game](#) that also uses the cartesian coordinate system!

## Sonar Treasure Hunt

To create the Sonar Treasure Hunt, you need to use the coordinate system that you have learned about in chapter 12. Sonar is a technology that ships use to locate objects under the sea. For this game, the player drops sonar devices at various places in the ocean to locate sunken treasure chests. The sonar devices in this game tell the player how far away the closest treasure chest is, but not in what direction. By placing multiple sonar devices, the player can figure out the location of the treasure chest.

In chapter 13 you will be organising your data into data structures. You have done this already in Module 4 where you have used lists and dictionaries. Read [this article](#) to read up on these data structures, but also on the more advanced types such as trees and graphs. You will also use the following methods and functions in the chapter:

- `remove()` list method - experiment with this method using [this lesson](#)
- `isdigit()` string method - experiment with this method using [this lesson](#)
- `sys.exit()` function. This function is different from your normal `quit()` and `exit()` functions. Find out what the difference is in [this article](#).

# Module 7: Advanced AI and Coordinates

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

- Recap what you have learned in module 6. Check your own notes, or read the summaries provided at the end of chapters 12 and 13 of 'Invent Your Own Computer Games with Python'.
- Read and work through chapter [15](#) and [16](#) of *Invent Your Own Computer Games With Python*.<sup>17</sup> Be able to understand the concepts covered in those chapters.

**Further learning (optional):**

- More about the `bool()` function via this [blogpost](#).
- Read more about the `round()` function [here](#).
- For those who are interested in taking the Reversegam a step further, take a look at [this version](#) where a graphical user interface is added! The [code](#) is available on GitHub.

## Recap of module 6

By learning more about the Cartesian Coordinate System in the previous module, some simple mathematical concepts were introduced that we will build upon further. The Cartesian coordinate system is needed to describe where a certain position is located in a two-dimensional area. Coordinates have two numbers: the x-coordinate and the y-coordinate. The x-axis runs left and right, and the y-axis runs up and down. Your computer screen is made up of pixels, the smallest dot of colour a screen can show. It's common for computer screens to use a coordinate system that has the origin (0, 0) at the top-left corner and that increases going down and to the right. There are no negative coordinates.

Chapter 12 also taught us what the absolute value of numbers is: The number without the minus sign in front of it. Therefore, positive numbers do not change, but negative numbers become positive. For example, the absolute value of -4 is 4 and absolute value of 5 (which is already positive) is just 5. Python's `abs()` function returns the absolute value of an integer.

The Sonar Treasure Hunt game that you recreated during module 6, is the first game we've made in the Python track that uses the Cartesian coordinate system. This is needed, as the board has 900 spaces. The Cartesian coordinate system makes all these spaces manageable. The locations in games that use this system can be stored in a list of lists in which the first index is the x-coordinate and the second index is the y-coordinate.

---

<sup>17</sup> <https://inventwithpython.com/invent4thed/>

These data structures (such as the ones used for the ocean and treasure chest locations) make it possible to represent complex concepts as data, and your game programs become mostly about modifying these data structures.

## The Reversegam Game

[Reversegam](#), also known as Reversi or Othello, is a two-player board game that is played on a grid. Therefore you will make use of the Cartesian coordinate system with x- and y-coordinates. This version of the game will have a computer AI that is more advanced than the Tic-Tac-Toe AI from [Chapter 10](#). In fact, this AI is so good that it will probably beat you almost every time you play.

It is a challenge to make this game more preservation themed (we dare you!), but it is a good game to practise more with using coordinates. In the game, the players choose a colour: black or white. Each game starts with already two tiles on the board. Every time you add a tile on the board, any of the opponent's tiles that are between the new tile and the other tiles of the player's colour are flipped. The game ends when the board is full. The player with the most tiles in their colour on the board wins.

Reversegam uses all the programming concepts that you have learned so far, with the exception of one new function: `bool()`. The description in the chapter is very brief, so if you want to learn more about the `bool()` function, read this [blogpost](#).

## Reversegam AI Simulation

The reason why the computer beats a human opponent so much, is not because the computer is smarter than you, but because it can calculate the possibilities for moves much faster. [Chapter 16](#) of *Invent Your Own Computer Games With Python* therefore helps us to recreate the Reversegam game step by step, but this time the computer will be its own opponent.

New in this version of Reversegam, is Python built-in `round()` function. This function rounds a floating-point number to the nearest integer number. More on this function, how to use it, and the errors that can occur can be found in this [blogpost](#).