Chapter 1

Introduction

What is programming?

Programming is giving a series of instructions to a computer. This series of instructions is known as Algorithm. Here is an example:

```
take two steps forward,
try opening the door
if it's unlocked, open the door,
if it isn't, turn left
```

It is fairly common for educators to teach programming using "hands-on examples", in which you're simply copying someone, without actually learning how to think or use the tools at your disposal. I disagree with this approach and prefer to add a little bit of information with each module, expanding our knowledge and teaching the student how to find solutions to problems using the tools they've learned.

Do I need to know math?

A common fear among those who are interested in programming is the *myth* that you need to know math to program. Yes, I've said *myth*. And the reason why I've said it is that most times programming will not require anything but basic math. It is important to understand that inside the computer science field there are many professions. Some will require a higher knowledge of math, others will barely require any. Just because something involve numbers, it doesn't mean it requires math.

I think I'm too old to learn how to code

Another *myth* that is very common is that to become a good programmer you need to start learning it very early in life. Programming is something you learn by doing, by practicing. And you'll grow with each day of doing it. No one is born a programmer. It is a skill acquired through study and practice.

Why python?

Python is a general-purpose language, which means it is a language used in many fields, including data science, machine learning, software development, web development, automation, among others. On top of this, its syntax is similar to English, which makes it less intimidating than other languages for those who are beginning their journey.

About the course

The course was developed in such way that each chapter will build on the previous. Because of this, everything we see will be utilized in the future, so it is crucial that you do not move to the next module until the topics are well understood.

We'll cover other topics besides python throughout the course, even if not deeply, because we'll need this knowledge to work on our projects.

Do not limit yourself to the exercises of this course. Programming is like LEGO. It is a creative exercise. In this course you'll learn all the tools and knowledge needed to create everything you can imagine, as well as the ability to learn another programming language.

About the exercises

There are multiple ways of doing the same thing in programming. Some codes are more optimized than others, but as long as everything works, it's good. The answers to the exercises can be found on [this repository](#).

Try to do the exercises yourself and, if you can't, check my answers.

Bonus: How does a computer work?

Each piece of a computer has a specific job. Here, I want to clarify a little bit about what each of them do, in a superficial manner. This will help us clarify some things later on.

Motherboard

It connects all the pieces of the computer and made them work together.

CPU | Processor

It is the brain of the computer. It is capable of processing an enormous amount of information per second.

GPU | Graphics Card

It processes the images and videos on a computer, such as games, tv-shows, the image in your monitor.

HDD | SSD | NVMe

This is the long term memory of a computer. The data is **persistent**, which means that even after you turn off the computer, the data on your HD will still exist. It is where you'll store photos, videos, software, games, etc.

RAM | Random Access Memory

RAM is the *short term memory* of a computer. It is extremely fast when compared to the speed of an HD, however, it is much smaller in size. When we run a software, open a video, do something, all the data needed to run these things is stored on RAM, when we close these programs, they're removed from memory. Because of that, when you open a new program, its data is placed where there is free space within the RAM. The location of the data is not permanent.

Python download and installation

Python can be downloaded from the official website.

> ***Important:*** Download python 3 (or the most updated version) to guarantee that the codes on this course will function correctly.

Windows

The windows installation is simple, like any other software. Select "Add python.exe to PATH" and then just click "next" and wait.

> **Important:** Check the "Add python.exe to PATH" checkbox before installing.

MacOS

For the installation on MacOS I recommend this link from [freeCodeCamp](). Feel free to use a different source, maybe another website or a youtube video.

Linux

Python comes installed with most Linux distributions. I do not recommend updating python because some libraries might break due to difference in versions.

IDE installation

Once installed, python can run from the terminal of the computer. However, for the this first part of the course, we'll install an IDE (Integrated Development Environment) to program.

> **Note:** You can write your code even on Microsoft Word or Notepad, and execute it from the terminal. An IDE is a software made specifically to run certain languages, coming prepared with everything you need and optimized for this purpose. There are other text editors and IDEs you can use.

PyCharm Community Edition

I will use PyCharm as my IDE for the first part of the course. The download can be done from [this link]().

> **Note:** There are two versions, PyCharm Professional Edition, which is paid, and PyCharm Community Edition, which is free. We'll be using the Community Edition.

Once installed, create a new project and wait a little. The first time a project is created, the IDE requires some time to organize things.

Visual Studio Code

VSCode is a text editor optimized for coding and it utilizes plugins to improve the life of the programmer. If you are using and older or slower computer, I recommend using VSCode with the python plugin. It is easy to find on google how to install said plugins. The download can be done from [this link](#).

Once installed, open the program, click in "Open folder" and select the folder in which your files will be. After this, create a file `filename.py` by clicking with the right button of the mouse on the "EXPLORER" panel, on the left side of the screen.

Later in the course we'll utilize VSCode for some projects.

Chapter 1: The Basics

My first program

It is standard that the first thing we make is a little program that says *"Hello, world"* to confirm that everything works properly, and this is what we're going to do. Some IDE's or text editors automatically create a python file when you start a project, some don't. If your file was not created, make a file called "hello.py". The `.py` at the end is what makes this a python file. Write:
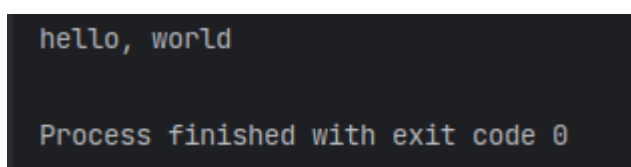
```python
print("Hello, world")
```

> **Note:** It is important to use quotation marks in the message.

And execute the program by clicking on the **play** button at the top of the screen:

This should be the result:

There you go! You'e just made your first python program. Simple, right?

```
print()
```

Is a function that prints on screen whatever data is between parenthesis.

> **Note:** We'll study functions deeply later. For now, don't worry about them.

Interpreted x compiled languages

You might have heard that computers do not understand anything but zeros and ones. This is true, they only understand binary code. Then how is it possible that the computer has understood what I wrote? Behind the scenes, python reads the code and "explains" it to the computer.

There are two types of languages, the **compiled** ones and the **interpreted** ones.

Languages such as java, C and C++ are compiled, meaning the code written is converted to a language the computer understands and then we can execute the program. We compile it once and execute it multiple times.

Other languages such as python and javascript are interpreted. Every time we run the code, the interpreter reads it and "explains" it to the computer. And it will happen whenever you run it again. When we open a website, for example, the browser reads the website's code and creates it on screen.

Variables

A variable is a little space in memory (RAM) where you'll store a data. This data can be a name, a date, a number, etc. It sounds confusing, I know. Let's make it a little clearer using an example. We'll make a variable called `message`:

```
message = "Hello, world"
print(message)
```

In this example, we're separating a little space in memory and calling it `message`, then we get the data `Hello, world` and put it inside of that space. Next, we call the function `print()` again:

```
hello, world

Process finished with exit code 0
```

As we can see, we have the same result. Python printed the value assigned to the variable `message`.

> ***Important:*** The equal sign `=` is not the same as the mathematical sign. We'll see the comparative equal sign later. In python `=` is called *"Assign operator"*. It gets a value and assign it to a variable.

Okay, but where is this used? Think of the games you've played. The number of lives, your HP, the name of the characters, their attributes, the ammo. All of this data is stored in variables.

Variables can have any name, however, there are some rules that must be followed when choosing them:

- The name of the variables cannot contain spaces.
- They can only contain letters, numbers and underscores, but they can only begin with a letter or underscore. The names cannot begin with a number.
- The ideal is that names are descriptive and help understand the type of information the variable is storing.
- Avoid using words that are reserved by python, like `print`.

> ***Note:*** There are some naming conventions. Some of them are:
>
> - **camelCase**: The first word starts with a lowercase letter and the rest of the words start with an uppercase letter
> - **PascalCase**: The first letter of every word is uppercase
> - **snake_case**: All the words are lowercase and separated by underscores
> - **kebab-case**: All the words are lowercase and separated by a hyphen
>
> Python recommends snake_case

Python reads code from top to bottom

Programming languages, in general, read code from top to bottom. In the following code, we try to print a variable that has still not been read by python:

```python
print(f"Hello, {nome}")
nome = "João"
```

When we try to run the code, python raises an error:

```
Traceback (most recent call last):
  File "/home/joao/PycharmProjects/test_project/main.py", line 2, in <module>
    print(f"Hello, {nome}")
NameError: name 'nome' is not defined. Did you mean: 'None'?


Process finished with exit code 1
```

Here, python gives us some information about where it believes the problem is. It informs us the file and line. Then it says the name of the error it's found:

```
NameError: name 'nome' is not defined. Did you mean: 'None'?
```

It informs us that the word "nome" has not been defined. It doesn't know this word or what it means. Then it asks "Did you mean: 'None'?"

The importance of errors

When python can't understand what you're trying to do, it will raise an error and tell you what is wrong, or try to do so. Read the error carefully and don't be afraid of googling it. As important as learning how to code, it is to learn how to debug, which is the act of fixing problems within your code.

Nobody knows everything about a language. The more we use it, the more we learn. But even so, it is a very big topic. This is why basically everything in programming is documented. Within the documentation we can find, in detail, how to use certain tools and how the language runs behind the scenes.

Besides google and stackoverflow, now we also have the aid of AIs such as ChatGPT, among others, to help with programming. They're excellent tools that will explain and help you write code.

> **Important:** It is common to see students copying code generated by AI and pasting them into projects. If you don't understand the code, it can break parts of your program and even add vulnerabilities to it. Use AI as a tool only.

Data Types

There are many data types, and each is treated by the language in a different way behind the scenes. The interaction between data will depend on their type.

Some languages are more strict and force us to specify the type of data a variable will take. These are known as *Statically Typed Languages*, and some examples are: Java, C, C++, C#, Kotlin, Go.

Other languages are less demanding and check the type of data within the variable when the program runs. These are known as *Dynamically Typed Languages*, and some examples are: Python, JavaScript, Ruby, PHP.

We've already created variables and assigned data to them, like this:

```
message = "Hello, world"
```

As we can see, we did not have to explicitly tell python the type of data we're assigning to the variable. It can figure out the type. But in other languages, such as java, we do. Here is an example of a variable being created in java, a statically typed language:

```java
String message = "Hello, world";
```

In java we have to tell it what is the type of data the variable should expect.

But if python can recognize them, why do I need to know all the different types? Each type has their own rules of usage and interaction. We'll learn more about these when we talk about them in the following sections.
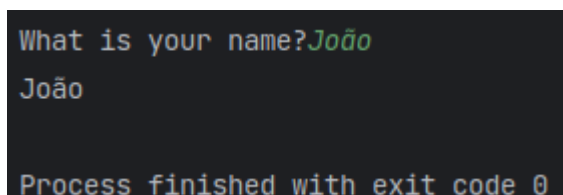
String

The data of type `string` are alphanumeric characters, which means they're letters and numbers. They appear between quotation marks (double `"` or single `'`).

> **Note:** When a word is typed without quotation marks, python sees them as commands.

Let's make a variable `name` and this time let's ask the user to type their name with the function `input`:

```python
name = input("What's your name?")
print(name)
```

When we run the code, this is what we'll see:

```
What is your name?João
João

Process finished with exit code 0
```

```python
input()
```

Is a function that writes on screen the message in-between parenthesis then gets what the user types and stores it in a variable.

Ignoring characters

As we can see, when running the code, the user's answer was right next to the question. This can be fixed with a space, but we can also add a line break using a backwards slash `\` to add special characters. For example:

```python
name = input("What is your name?\n")
```

> **Note:** `\n` in the middle of a string causes a line break.

The backwards slash is also used to ignore a character that follows it, as we'll see next.

Should I use single or double quotation marks?

Both work the same way: They determine where the string begins and where it ends. Let's assume that you want to use quotation marks within a string:

```python
"And then he said "wow""
```

As we can see, the colors are different to let us know that python will complain and raise an error. It can't understand that "wow" is a part of the string because the first quotation marks determine where the string begins and the second where it ends.

One possible solution is to use the backwards slash, as we've seen:

```python
"And then he said \"wow\""
```

In this case, the colors haven't changed. Python understands that the quotation marks around "wow" should be treated as a part of the string, because they're in front of a backwards slash.

Another way to deal with this situation is to utilize single quotation marks:

```python
'And then he said \"wow\"'
```

Here, the single quotation marks determine where the string begins and where it ends, so the double quotation marks are treated as a part of the string.

Concatenation

Concatenation is the act of putting data together. There are multiple ways of doing this in python, we'll see some of them:

The most common way of concatenation is by using the `+` operator.

```python
name = input("What is your name?\n")
print("Hello, " + name)
```

In this case, we are concatenating the string `"Hello, "` and the string inside the variable `name`, which was given by the user.


```
What is your name?
João
Hello, João

Process finished with exit code 0
```

Another way of concatenating variables is by separating them with a comma:

```python
name = input("What is your name?\n")
print("Hello,", name)
```

> **Note:** In this case, python adds a space between the string and the variable.

As you're probably already imagining, concatenating multiple variables in a huge text would be a lot of work. There is a better way of concatenating in python: using an `f string`.

```python
name = input("What is your name?\n")
print(f"Hello, {name}")
```

In the `f string` all the variables are between curly braces in the middle of the string. Python understands that they're variables and replaces them with their values.

Exercise 1

1. What is the result of the following expression:

```python
print("11" + "11")
```

2. What's the difference between Dynamically Typed Languages and Statically Typed Languages?
3. Madlibs is a game in which a list of words is given and added to a story. Make a program that will ask the user for 1 name and 2 adjectives, then add them to a sentence.

Numbers

There are two numeric data types: int (integer) and float (floating-point number). Integers are whole numbers, while floats are numbers with decimal places.

Data type	Examples
Int	-2, -1, 0, 1, 2, 3, 10, 45, 67, 106
Float	-1.32 , -1.0 , 0.0 , 1.57, 43.5, 100.0

We can make calculations using python, and for this we utilize operators, just like in math. They are:

Operator	Name	What it does
+	add	adds the number
-	subtract	subtracts the number
*	multiplication	multiplies the number
/	division	divides the number
%	modulus	gives you the remainder of the division
**	exponentiation	raises the number to a power
//	floor division	divides the number and gives you the result rounded to the closest lowest number

Some are quite simple and you probably remember them, but others not so much. Then let's take a look on some examples to clarify their usage.

Modulus

This operator divides the numbers and gives us the remainder of the division. What will be the result of the expression below?

```python
print(11 % 2)
```

That's right! The result is 1.

It divides 11 by 2 and the remainder is 1.

Exponentiation

In case you don't remember, exponentiation is the act of multiplying a number by itself one or more times.

```python
print(2 ** 3)
```

Is the same as 2^3 or $2 \cdot 2 \cdot 2$, which is equal to 8.

```
8

Process finished with exit code 0
```

Floor Division

With this operator, the numbers are divided and python rounds them to the closest lowest number.

```python
print(10 // 3)
```

The result of 10 divided by 3 is 3.333, however, when we utilize the floor division, python gives 3 as the answer.

```
3

Process finished with exit code 0
```

Int e Float

In certain languages, you can only execute operators with data of the same type. In python this is not the case. the language tries to understand what you're trying to do and tries to give you the most precise answer possible.

Because of that, it whenever we try to do an calculation with an `int` and a `float`, the result will be a `float`. Python understands that there is a high chance the result of this calculation will be a number with decimal points, so it converts the result to `float`.

```python
print(5.0 + 3)
```

```
8.0

Process finished with exit code 0
```

In some cases, as in the division, this also occur. Even if the division is between two integers. Python knows that there is a good chance that the result of a division will be a number with decimal points.

```python
print(4 / 2)
```

```
2.0

Process finished with exit code 0
```

Number of decimal places

Sometimes python will give a result with many decimal places. This is normal and it happens in many languages. Python always tries to deliver the most precise answer, which is a little hard given how computers deal with numbers.

```python
print(0.2 + 0.1)
```

```
0.30000000000000004

Process finished with exit code 0
```

But what if I want a number with a specific amount of decimal places? In this case, we can format the result.

```python
number = 1 + 0.9887984
print(f"Non-formatted result: {number}")
print(f'Formatted result: {"%.2f" % number}')
```

Here we're trying to add an `int` and a `float`, so we know that the result will be a `float` and python will try to make it as precise as it can. On the third line, notice that the variable is not alone. We've also passed `"%.2f" % variable_name`, which will get the value passed to the variable and round it to two decimal places.

> *Note:* The `"%.2f"` must always be between quotation marks. The `2` is the amount of decimal places you want.

and this will be the result:

```
Not formatted result: 1.9887983999999999
Formatted result: 1.99

Process finished with exit code 0
```

Order of mathematical operations

Just like in math, in python the order of operations must also be respected.

```python
number = 5 + (2 + 3) ** 2
```

First we'll calculate the numbers between parenthesis, then the exponentiation and finally, we add.

```
30

Process finished with exit code 0
```

Numbers with underscore

Big numbers are often hard to read. Python solves this problem by letting us use underscores to separate the numbers, making them more readable.

```python
big_number = 4_540_000_000
print(big_number)
```

```
4540000000

Process finished with exit code 0
```

Boolean

Boolean is a data type with only two possible values: True and False.

```python
true_bool = True
false_bool = False
```

Pretty easy, huh?

I know that it doesn't seem too relevant yet, but boolean is extremely important and we'll see their usage soon.

Revisiting variables

Now that we already have a solid understanding of variables and data types, we can go a little deeper into these topics.

Up to this point we've only assigned a value to a variable and used it. But, as the name suggests, it is variable. Which means we can alter it's value at any moment.

```python
number = 100
number = 10
print(number)
```

In this example, first we make a variable `number` and assign it the integer 100. Nothing new. Then we modify the value of the variable to 10. When we run the code, this is the result we get:

```
10

Process finished with exit code 0
```

Python reads the code from top to bottom, remember? So first it creates the variable, then python modifies it and finally, prints its current value on screen, which is 10.

Since python is a dynamically typed language, we can even change the type of the variable to string and python would understand it.

```python
number = 100
number = "hello"
print(number)
```

```
hello

Process finished with exit code 0
```

More assign operators

Let's imagine that we have a variable `number` of value 2 and we want to add 10 to its value. How would we do that?

```python
number = 2
number = number + 10
print(number)
```

It looks a little confusing, right? But let's go through it step by step. First, we create a variable and assign it the value 2. Then, we assign to the variable `number` the value of `number` + 10, meaning that the new value of `number` will be its current value (2) plus the integer 10. Resulting in:

```
12

Process finished with exit code 0
```

Another way of writing this would be:

```python
number = 2
number += 10
print(number)
```

Here are some other assign operators:

Operator	What it does
+=	adds to the old value and assigns the result to the variable
-=	subtracts from the old value and assigns the result to the variable
*=	multiplies by the old value and assigns the result to the variable
/=	divides by the old value and assigns the result to the variable
%=	calculates the division and assigns the remainder to the variable
**=	raises the number to the power and assings the result to the variable

Type Casting

Type casting is the act of changing the type of a variable.

While python allows us to work with integers and floats at the same time, even if they're different data types, this doesn't happen when with strings and numbers. If we try, python will raise an error.

```python
print("10" + 10)
```

```
Traceback (most recent call last):
  File "/home/joao/PycharmProjects/test_project/main.py", line 2, in <module>
    print("10" + 10)
TypeError: can only concatenate str (not "int") to str

Process finished with exit code 1
```

Python is complaining and saying that it can only concatenate `string` (str) to `string`, and not `int`.

User input

When we use the function `input()` to get some information from the user, this data always comes in the `string` format, even if the user has types a number. To deal with this situation we need to convert the data. And how do we do that?

There are a few functions we can use:

```python
int(data)
```

turns the data into an integer.

17

```
float(data)
```

turns the data into a float.

```
str(data)
```

turns the data into a string.

An use case example:

```
number = input("type a number:\n")
number = int(number)
```

or, if you'd like to make your code more concise:

```
number = int(input("type a number:\n"))
```

Exercise 2

1. Make a program that asks the user to type 2 numbers and print their sum.
2. Make a program that asks the user's name and the year they were born, and shows a sentence with their name and how old they are/will be this year.
3. Make a program that asks the user for two numbers and show their division with 3 decimal places.

Constants

Just like variables, constants are spaces in memory where data is stored. The difference is that the value assigned to a constant cannot be changed when the program is running.

When you are sure that the value of a data will not be altered during the execution of the program, you can write it as a constant. This way, the language will know that the value will not change and, behind the scenes, will make optimizations to work in a more efficient manner.

Okay, but how do I do this in python? So, python does not have a constant data type. However, since this data type exists in other languages, we follow the same naming conventions and make its name capitalized. This way we make it clear to other programmers

or to ourselves when we revisit our code, that the variable should be treated as a constant. For example:

```
CHARACTER_NAME = Lonk
```

Comments

Comments are lines of code ignored by python, which we write for ourselves or for other programmers. Wait, this sounds confusing. Let me explain it with an example:

Imagine that you worked on a project a year ago and now you want to improve it. When opening the project, you can read all that code and decipher what each line is doing, but it would be a lot faster and easier if there were comments in the code explaining what each block of code does, or explaining how you've solved a certain problem.

Comments are essential and make our lives much easier. When we are part of a team, they allow us to clarify to other programmers what our code is doing; When we revisit our old projects, they allows us to understand what and how we did them.

In python, everything that's written in front of a `#` is treated as a comment.

```
# This is a comment.
# and this is too!
```

We can also use triple quotation marks to make multi-line comments.

```
"""
This is a comment.
and this is too!

Everything written here is treated as a comment.
"""
```

The Zen of Python

The Zen of Python is a text written by Tim Peters that describes the philosophy followed by the community to write good python code.

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
```

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

I'll comment only about a few principles. Some may not be understandable yet, but they will be later.

```
Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Readability counts.
```

When we are beginning our programming journey, we don't know what is good and what is bad. Today we see many videos on social media telling us to write code a certain way because "it is more professional", when, in fact, you're needlessly making your code harder to read. A readable, beautiful, easy to understand and simple code is always preferred and more professional than the opposite.

And, finally, perhaps the most important principle:

```
Now is better than never.
```

Often times we want to learn as much as we can before starting. We feel we're not ready, that we don't know enough. However, you don't need to know everything to take the first step. More important that writing the perfect code, is to write a code that works. Later, if you want, you can revisit it and improve it.

Don't leave it for later. Do what you can with what you have now.