1-Introduction

April 20, 2023

1 Introduction to programming

1.1 What is programming?

When you are programming, you are giving a computer the instructions to perform one or several tasks. For example, I will tell the computer to perform a sum here:

[2]:
```
63-35
```

[2]:
```
28
```

As you can see, I have given the computer two initial data (63 and 35), which we call **inputs**, and the instruction to substract 35 to 63 (defined by the - symbol). As instructed, the computer has returned the result of this calculation, the **output**.

Therefore, we can think of programming as follows: `inputs + instructions = output`

1.2 What is the purpose?

The idea is that the computer **performs a task for us**. The task can be, for example, a complicated mathematical calculation, or even a simple one like in the example. It can also be a different type of task, like displaying an image or sorting data. These are examples of basic tasks, but it can be more complex, like managing a server or running a genome analysis pipeline.

One of the main advantages of programming is that it allows us to **automate tasks**. In addition, a computer programme will usually give us the same answer from the same input data, which is not so easy when we perform a lab experiment. This means that, in theory, it should be **more reproducible**. However, in practice, you need access to the instructions that have been fed to the computer, not just the inputs and outputs, and these methods are often not published.

1.3 How does it work?

This is quite simple: you feed the inputs and the instructions to the computer, then this code (**source code**) is processed by a **compiler**, which "translates" it to a language that the computer can understand (**object code**). Usually, a programme is made up of several pieces of code called **modules**. These modules are put together by a **linker**. Often, even single-module programmes need to be linked after they are combined, because linkers also replace symbolic links ("fake" addresses that you create to point to the true addresses of files) with true addresses. After the compiler and the linker have done their work, you get the so-called **machine code**, the final instructions that your machine will execute.

To summarize, the process works as follows:

1. Writing source code
2. Compiling code
3. Linking code
4. Executing code

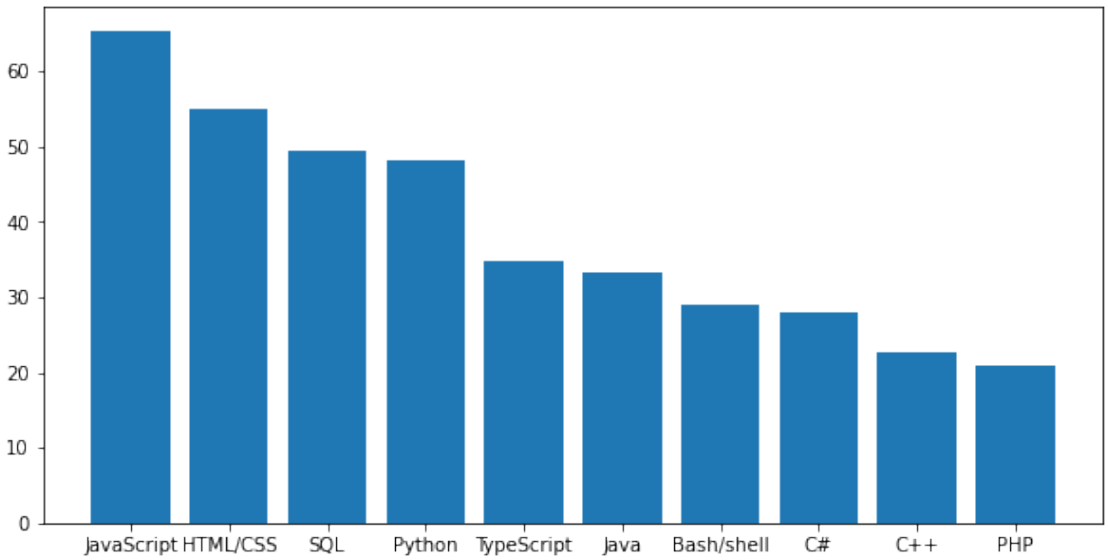
Reference: https://www.webopedia.com/definitions/compilation/

1.4 Why Python?

Well, why not? Perhaps I'm not being very convincing here… let's look at some numbers:

According to Stack Overflow's 2022 Developer Survey, Python is the fourth most used programming language, with a 48.07% of respondents using Python. Among those who are learning to code, the number goes up to 58.38%. Let's display it more nicely:

[13]:
```
from matplotlib import pyplot as plt
languages = {"JavaScript": 65.36, "HTML/CSS": 55.08, "SQL": 49.43, "Python": 48.
↪07, "TypeScript": 34.83, "Java": 33.27, "Bash/shell": 29.07, "C#": 27.98,␣
↪"C++": 22.55, "PHP": 20.87}
fig, ax = plt.subplots(figsize = (10, 5))
plt.bar(languages.keys(), height = languages.values());
```



Note that here I'm only displaying the 10 most used programming languages. If you want to learn more, you can check the survey here.

1.4.1 Well, if JavaScript is the most used language, why am I not learning JavaScript?

You have to take into account that different programming languages are popular in different fields. For example, JavaScript, PHP and HTML/CSS are widely used for web development, which is not the purpose of this course. But of course, if you want to develop your own website, you should look into these programming languages! SQL is used to manage databases, so it is also very useful if you want to be a web administrator. And C++ is very popular in technical fields.

Your choice of programming language should depend on your needs, on your goal and on what is already available. As a Bioinformatician, you might need to learn several programming languages. Bash will be essential to you if you are using a Linux computer, and you might also want to learn R if you need a specific R package, e.g., to do differential expression analyses with DESeq2.

1.5 How do I get started?

Good that you asked! In this course, you will be using Anaconda. Follow the link to Anaconda or search for it with your preferred browser. Then, click on the `Download` button. You should be prompted to install the version of Anaconda that best suits your operative system.

Once that you install Anaconda, in your command line you will be in the `base` environment. Now you should create a new environment from the course repository. First you need to clone this repository to your local computer. To do so run the following command: `https://github.com/SentientVirus/ATTC_biopython.git`

Now, you should create a conda environment from the `biopython.yml` file. From your `base` environment, from the command line, also knows as shell or, if you have a Windows computer, through Anaconda Prompt, type `conda env create -f biopython.yml`.

This environment contains all the packages you will need from the course, even though some of them, such as Pandas or Biopython, are not default Python packages and have to be installed (`conda install ..`). In general terms, you shouldn't install anything to the `base` environment, but work in a separate environment.

Now, let's activate the environment that we created. The first line of the `.yml` file sets the name of the environment; in this case, it is called `bioinfo`: `conda activate bioinfo`

You can open Anaconda Navigator and check for applications on the `bioinfo` channel. You will see that Jupyter Notebook is one of those applications. Look for Jupyter Notebook and click on launch, then try to run this notebook!

1.6 Jupyter Notebook

1.6.1 What is it?

The Jupyter Notebooks, as their name indicates, are notebooks that combine text (this one has so much text that I might as well turn it into an essay…) and code! They were created to facilitate sharing documents. As you can see, it is super easy to comment code, you can even add walls of text. See this paragraph? It's a wall. There are no bigger walls than this one. Bear with me. Continue reading, I promise you it will be worth it… or maybe not. Actually, not at all. Sorry about this, I just wanted to make a point. Blah blah blah. And I could keep going. Yes, this is a threat. Okay, maybe this is enough. Maybe not.

Ejem, as I was saying, Jupyter Notebooks were created with the idea to make them collaborative, but they have other advantages. For example, you can organize your code in chunks and run it in a different order. Let me give you an example:

[33]:
```
num = 9 #Set a starting value for num
```

[35]:
```
num -= 1 #Remove 1 from the value of num
```

[34]:
```
num *= 3 #Multiply num by 3
```

[36]:
```
print(num) #Show the final value of num
```

```
26
```

Hmm… Shouldn't the result be 24? What has happened? Maybe it's time you run this notebook and find out…

1.6.2 Wait, how do I run the notebook? Isn't this a PDF?

The tutorials in this course are written in Jupyter Notebook. As you can see from the tutorial PDFs, one of the functionalities of Jupyter notebooks is that they can be converted to PDFs, but we won't cover this during the course. Once that you have installed Jupyter Notebook, you can run all the tutorials directly on the notebooks!

As you have seen, Jupyter Notebook can be run from Anaconda Navigator, but you can also run it through the shell (or through Anaconda Prompt if you have a Windows computer) by simply typing:

```
jupyter notebook
```

OBS! The environment that is activated when you run this command must have `jupyter notebook` installed. When you run the remaining tutorials of this course, make sure that you activate the `bioinfo` environment before you open the notebook, as the packages that you will need are installed there! Otherwise you will have to re-install the packages manually. You should also keep in mind that Jupyter Notebook will be run on the directory where you execute the command, so make sure you run it in the directory where you keep your notebooks!

1.7 Python IDEs

Integrated Development Environments (IDEs) are software applications designed to help you develop your code. The idea is that you should write and test-run your code using the interface that is provided by these applications. The simplest Python IDE is the **Integrated Development and Learning Environment (IDLE)**. To run it, you can use the following command (in the shell):

```
python -m idlelib <your script>
```

OBS! Replace `<your script>` with the name of the file that you want to run. You can try creating an empty file, `test.py`, and paste the following code in it:

[39]:
```
import os #Loading module
print(os.getcwd()) #Get current working directory (cwd) and print it
```

```
/Users/marmo435/Courses/ATTC_4/ATTC_biopython
```

Include the comments as well! You'll see that the `import` and `print` commands, as well as the comments, use a different color scheme! Then you can click on `Run`, and you'll see how it is run in the Python console. This code should return your current working directory. Feel free to play with the settings or write your own code if you feel confident to do so.

1.7.1 Other IDEs

Anaconda also includes Python IDEs, **Spyder** and **PyCharm**. If you want to try them, you can install them in an Anaconda environment, either through the `conda install` command or through Anaconda Navigator. Keep in mind that these IDEs are more advanced than IDLE, so it might be a little overwhelming before you have been introduced to basic concepts in Python and we will be using Jupyter Notebook to run the code for this course. However, if you want, you can copy the code in these notebooks into an IDE and try its different functionalities.