Guided Exercise 1

Annexes

Collective Code Ownership and Coding Standards



**Software Development**

February 2025

Degree in Computer Engineering - Double Degree in Computer Science and ADE

**TABLE OF CONTENTS**

[**APPENDIX A. OVERVIEW OF THE TOOLS 2**](#_vz4ff4a8eoqc)

[TOOLS TO USE 2](#_3h2jz57pp36f)

[IDE: PyCharm 2](#_3dracw2jy1a)

[Virtual Environments (Virtual Environment - venv) 2](#_ktyjp9p9y8as)

[Package Management 4](#_ys5yc4gy9ka2)

[Code analyser: Pylint 4](#_cr6478dq96u2)

[Version control system: GIT and GITHUB 5](#_lzpsmzm6jvtb)

[**ANNEX B. INSTALLATION AND CONFIGURATION OF THE ENVIRONMENT 6**](#_ertk7tvplsmz)

[Step 1: Installation of PyCharm Community 6](#_yinqtqiuwhmv)

[Step 2: Install Pylint 6](#_qg2wi3rmk4ch)

[Step 3: Register a project in GitHub 7](#_ot2pusmc0zvg)

[Step 3.0: Register in GitHub 7](#_q11m623is4f8)

[Step 3.1: Create the project repository on GitHub 7](#_r0o29ciuu7db)

[Step 3.2: Give the team access to the repository 11](#_7fjlsdcna0px)

[Step 3.3: Create local repositories 12](#_gkb42cugstml)

[Step 4: Configure the Python Interpreter 18](#_z042b2g147r)

[Step 5: Add code to the repository 20](#_si8b9eu6vo6h)

[Step 5.1: Create a test program 21](#_v1ri9n5f4ebx)

[Step 5.2: Install and use the static analyzer 21](#_jrh7jcxx084s)

[Step 5.3: Dealing with the virtual environment 23](#_kdcoksbgisri)

[Step 5.4: Changes and version control 28](#_435jgq2j2wxc)

[Download code from the remote repository 32](#_uouhe1pk8bvf)

[Step 6: Insert the code of UC3MTravel 36](#_qpt3dcpey11h)

[**ANNEX C. STEPS TO ESTABLISH A CODING REGULATION 39**](#_bv8gjy9fr6q4)

[Step 1. Standard for the Organization of Files 42](#_dx7d1ju2qa8m)

[Source Code Files 42](#_uqqsrqmaoh1j)

[Class Files 43](#_q52ke7hz9zcw)

[Step 2: Standard for Names and Variables 43](#_k3cqyjbzw3li)

[Classes and Members of Classes 43](#_z50bxpvd3lbi)

[Visibility 43](#_3vsigtukdwz1)

[Step 3: Standard for Methods 44](#_ths29g4rhmnz)

[Method structure 44](#_jbbddhizhqmb)

[Indentation and braces 44](#_a82w63t0x4z8)

[Method definitions 44](#_y9putd2uc2wb)

[Step 4: Standard for exception handling 44](#_4gai34mibvr)

[Step 5: Other standards 45](#_xeyi5ayc0dfd)

[Parameters splitting 45](#_o58n2ech6epd)

[Step 6: Standard for the application design 45](#_l05x7xjtk1vx)

[Step 7: Modify the Pylint configuration 46](#_3c21kog04i8)

[Step 8: Review the source code and update the repository 46](#_681r0cdgj80x)

# APPENDIX A. OVERVIEW OF THE TOOLS

This session will focus on the tools for managing changes in the source code (version control) and the tools that support continuous integration.

## TOOLS TO USE

### IDE: PyCharm

An Integrated Development Environment (IDE), is a computer application that provides comprehensive services to facilitate software development for the programmer.

We are going to use PyCharm. That IDE is available in the virtual rooms, and you can download the community edition for free. You can also get the education version using your institutional email (UC3M). PyCharm has integrated the connection with the version control systems used in this and further guided exercises (see below).

### Virtual Environments (Virtual Environment - venv)

A virtual development environment or simply a virtual environment is a mechanism that allows managing "isolated spaces" where you can install different versions of Python programs and packages.

The scripts installed in a virtual environment are isolated from those installed in other virtual environments or the global Python configuration.

The virtual environment is a directory tree that contains Python executable files, scripts, libraries and configuration files necessary to execute Python programs.

You will create a virtual environment for each project we work on, and we will configure them through the tools provided by our IDE, PyCharm.

The virtual environment cannot be shared because it contains specific configurations for the computer. We must take this into account when configuring the environment.

### Package Management

A package management system, also known as a package manager, is a collection of tools used to automate installing, updating, configuring, and removing software packages.

In software, a dependency is an application or a library required by another program to function properly. For this reason, it is commonly said that a program depends on certain applications or libraries.

We need a package manager to add the libraries required to the virtual environment. In fact, we must include in the virtual environment all the libraries required for our project. If we share our project, the team members should install all the dependencies in their virtual environments to run the software.

There are different managers: pip is the preferred program, and since Python 3.4 it is included by default with the Python binary installers. From PyCharm it is possible to manage packages from the user interface. Using pip commands, we can generate a file with the list of references of the project. When other programmers download the project to another computer, you can use that reference file to configure the virtual environment.

### Code analyser: Pylint

The static analysis of the code refers to the process of evaluating the source code without running it. Based on this analysis we obtain information that allows us to improve the code without changing the functionality.

In this guided exercise we will use a static code analyzer, Pylint (Python linting). "Linting" is the process of checking code to detect style and program errors that escape the usual parsing done by the compiler. The term's origin is derived from the name of the Lint tool, created to detect incorrect, suspicious or incompatible code between different C architectures.

Pylint allows you to identify errors and symptoms (code smells) in Python code according to standard coding rules written in a style guide, accepted and implemented by the entire team.

One of the most used style guides in the Python world is PEP8 (Python Enhancement Proposals 8). It is the standard that Pylint uses by default and the one that we will use in the course. You can find the information at the following links:

* [PEP8](https://www.python.org/dev/peps/pep-0008/)\_ https://www.python.org/dev/peps/pep-0008/
* [PEP8 version translated into Spanish](http://recursospython.com/pep8es.pdf): <http://recursospython.com/pep8es.pdf>

The Pylint code analyzer will run integrated into the PyCharm IDE.

### Version control system: GIT and GITHUB

Version control systems (VCS) are programs that aim to control changes in the development of any software, allowing to know the current status of a project, the changes made to any of its pieces, who made the changes, etc.

VCS use repositories that are directories where the files of a project are stored. They can store in the repository code files, images, audios and any element related to the project.

Git is a free software version control system (VCS) developed by Linus Torvalds (who also created Linux). It is very popular and works for all types of projects. It is part of other well-known solutions: GitHub, GitLab, etc.

GitHub is a project management and version control system and a social networking platform designed for developers. It is one of the largest online repositories for collaborative work in the world that allows you to collaborate with other people from all over the world, plan projects and track work.

The code can be stored publicly or privately. In the latter case, it is necessary to have a payment account or account for an organization with special agreements with GitHub, such as the uc3m.

GitHub is widespread in the developers' world, so we have considered it convenient to use in this subject so that it allows you to know it. Although we are going to use GitHub, we suggest you become familiar with Git. GitHub is available from the following URL: https://www.github.com

PyCharm, integrates a Git client to access the Git repositories on GitHub.

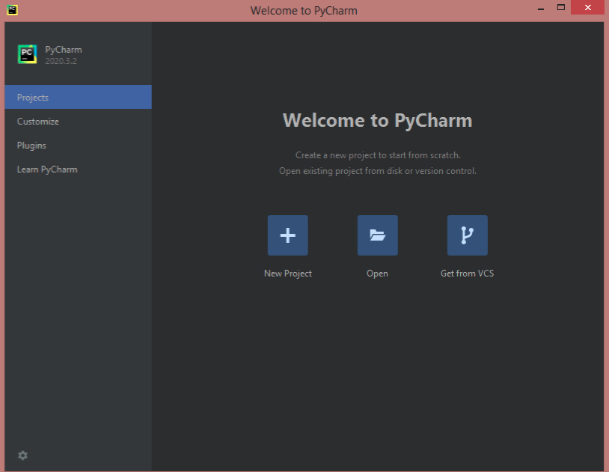
# ANNEX B. INSTALLATION AND CONFIGURATION OF THE ENVIRONMENT

## Step 1: Installation of PyCharm Community

To install this IDE you have to download it from the official website: Download PyCharm: https://www.jetbrains.com/es-es/pycharm/download

Once on the page, you have to select the environment (Windows, Mac, Linux), download the Community edition (it is free) and follow the instructions provided by the Company.

Once installed, when we open it we will see the following screen:



You can adjust the user interface in File-> Settings-> Appearance-> Behavior)

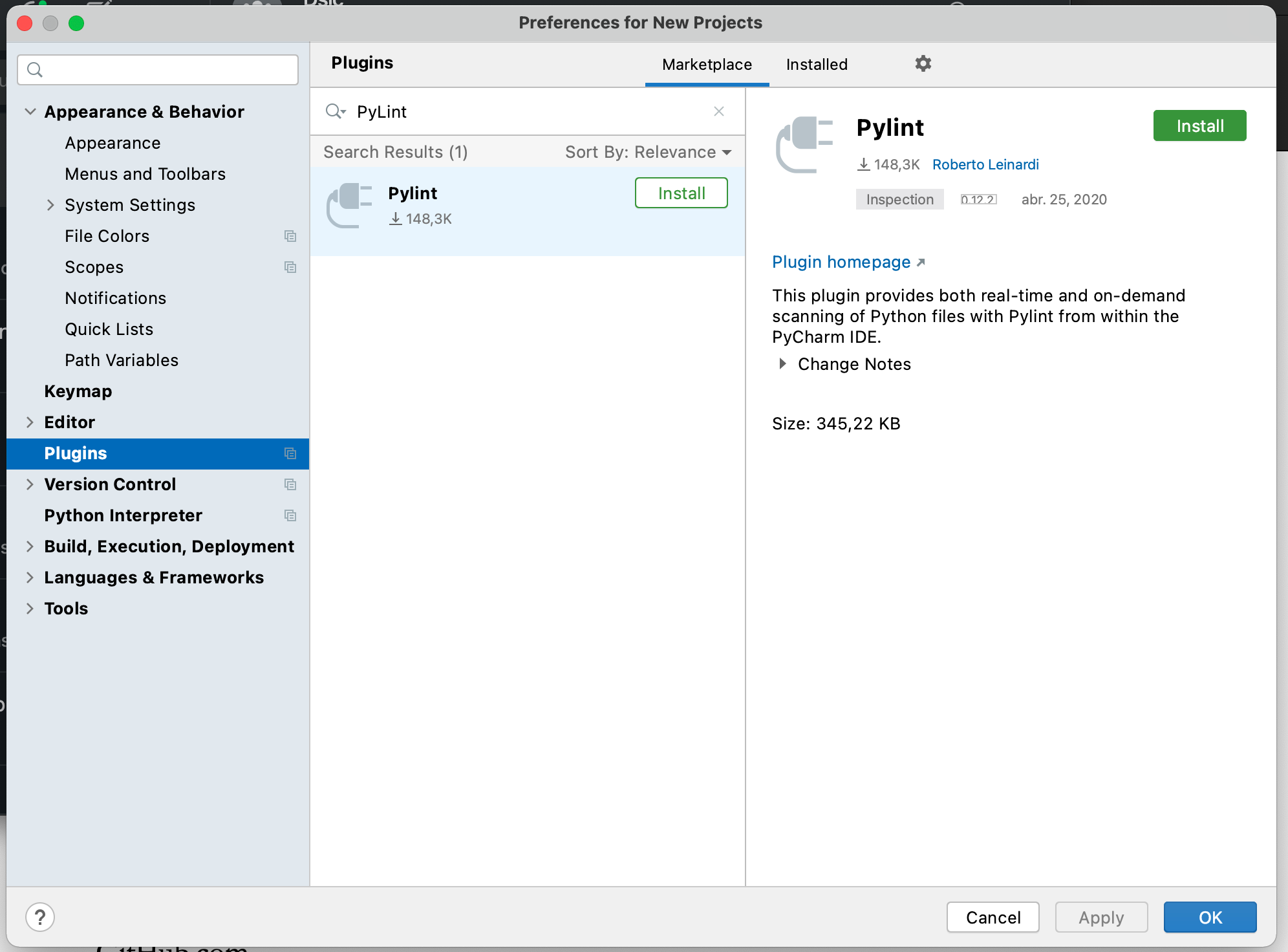
## Step 2: Install Pylint

As we have already indicated, we will use a code analyzer that is Pylint integrated into the IDE.

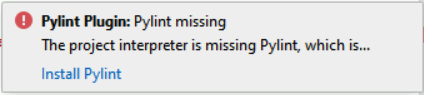
First of all you have to install Pylint in the PyCharm environment:

File-> Settings-> Plugin

We look for Pylint and install it.



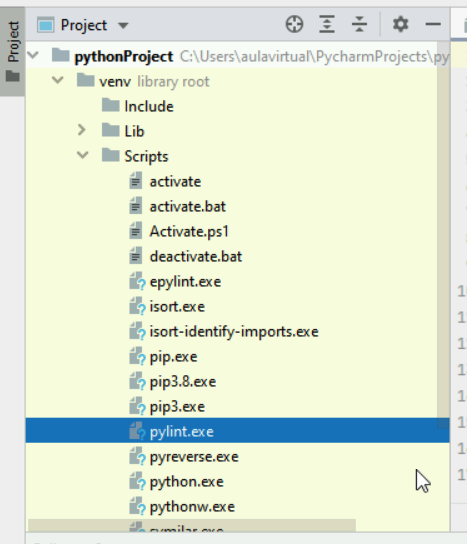
After installing the plugin, you should check whether the Pylint executable is available. Otherwise you should install it.



To install pylint, open the PyCharm terminal and execute the following command:

pip install pylint

If you are working in a virtual environment, the executable of Pylint will not be available for other projects, so you have to check it everytime you configure a new project.



## 

## Step 3: Register a project in GitHub

### Step 3.0: Register in GitHub

First of all you have to create your GitHub account, using the institutional email address ([1XXXXXX@alumnos.uc3m.es](mailto:1XXXXXX@alumnos.uc3m.es)). Your github user must be your student number (1XXXXXX).

The two members of the team must have an account.

### Step 3.1: Create the project repository on GitHub

In GitHub we are going to create the projects we need for the course. Each project will have a repository with all the required files (current and previous versions).

You have to create several repositories that will contain the guided exercises and the final practice.

These repositories will be called as follows:

* GYY.2024.TXX.GEZ

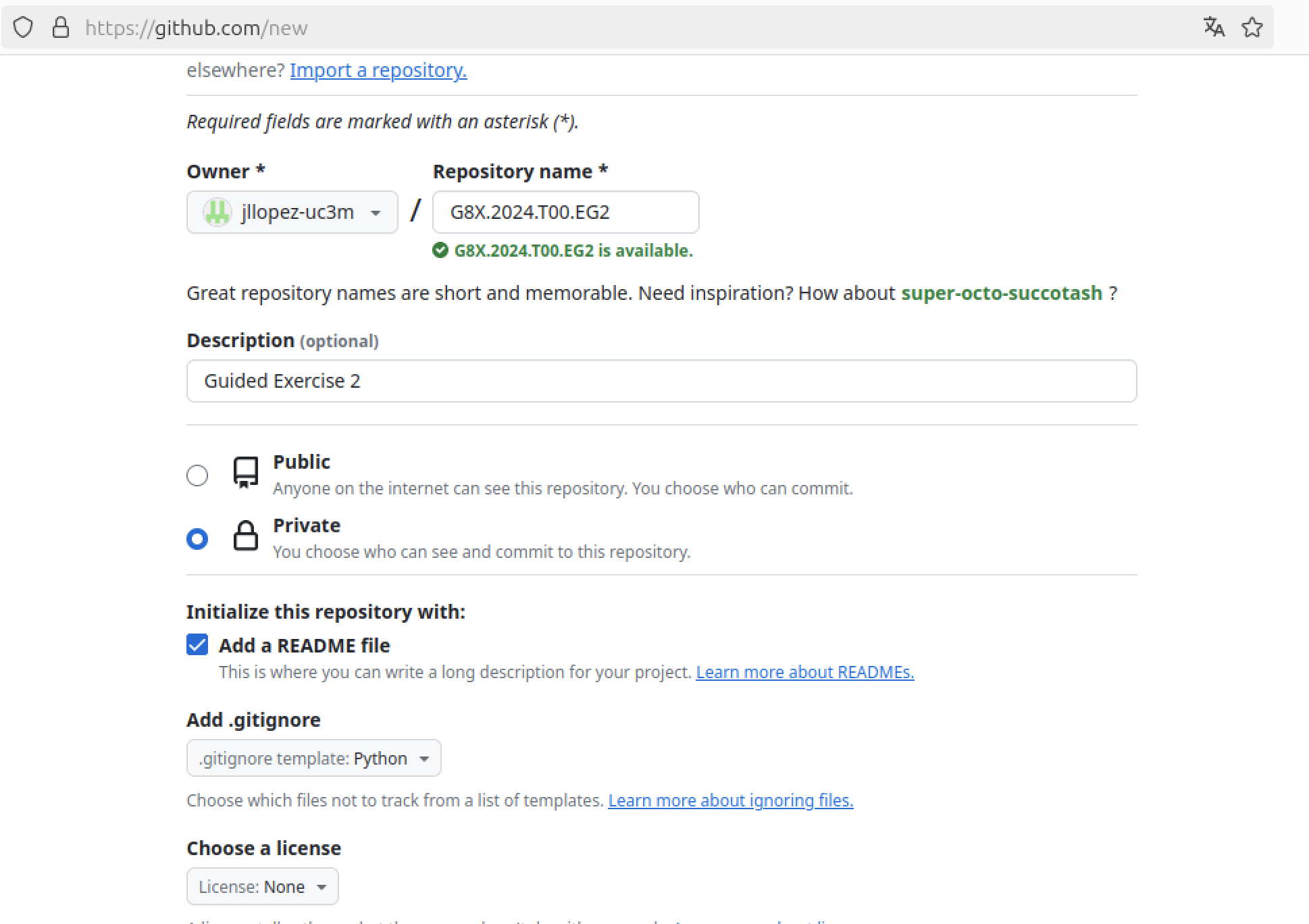
Where:

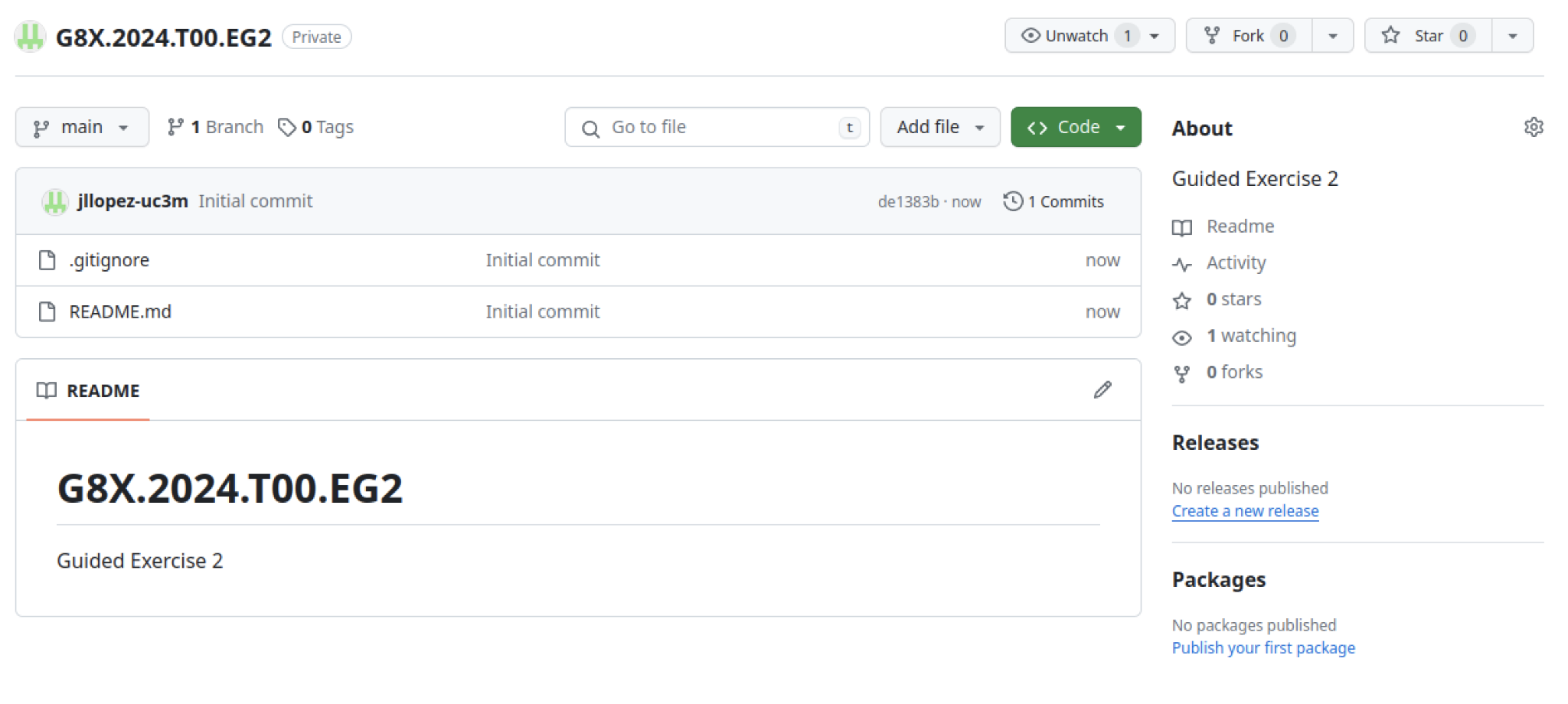
* YY is the class group, for example, class 50 is G50.
* XX is the group number that the teacher has assigned you.
* EGZ indicates Guided Exercise Z, where Z is the number of the guided exercise. Guided exercises 2, 3. (GE2 and GE3) will be uploaded here.

To create a GitHub repository you have to click on **New Repository**.



Type the repository name according to the convention that has been indicated previously, mark the repository as **Private** and check the item **Add a Readme file**. Optionally you can add a new .gitignore file choosing the option “Python” to avoid future issues. If you don’t generate the .gitignore file now, you can add it later. The .gitignore file contain list of files and folders you don’t want to control. These files and folders usually contain local configurations, compilations or libraries.





Thus we have created a repository. We have also included two files (Readme file and .gitignore) and made our first commit.

**Note:** The README files on GitHub contain information about other files in the repository. It is the form you provide for software documentation.

If you have created the .gitignore file, you can clic on it and take a look at the content. Notice that it includes several files and folders that won’t be controlled by git. Yo have to make sure that the foldes “venv” and “.idea” are included in the .gitignore. You can see that the “venv” folder is included (among others), and in the last line contains specific folders for pycharm project. However the line referred to the .idea folder is commented. You have to uncomment this line. To do so, you have to clic in the edit button (see next image).



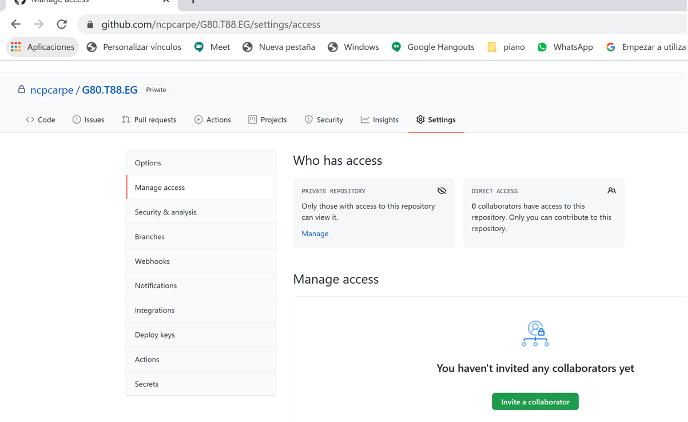
Once you are editing the file, you must uncomment the last line and clic on “Commit changes” (see next image).



### Step 3.2: Give the team access to the repository

Next, you must invite your team partner and your teacher to the repository. Click on Settings / Manage access. There you can invite those people you want to have access to this repository. These people will receive an email to accept the invitation (in less than 7 days) and they will now be able to see and clone said repository.





### 

### Step 3.3: Create local repositories

Each programmer works in the local repository (using Git) and can upload or download files from the central remote GitHub repository, where all the contributors integrate their changes.

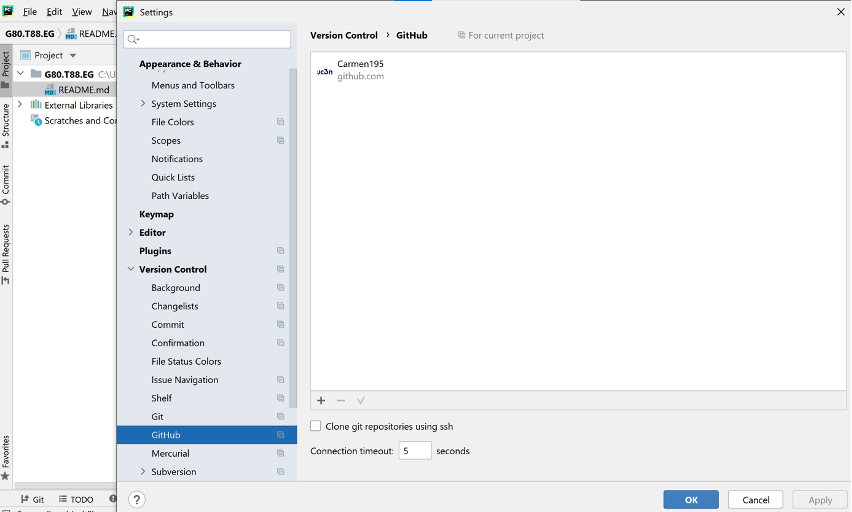
Therefore, once we have a repository on GitHub, either because we have created it or because we have been invited, we must create the local repository in our computer to start working.

We have two alternatives:

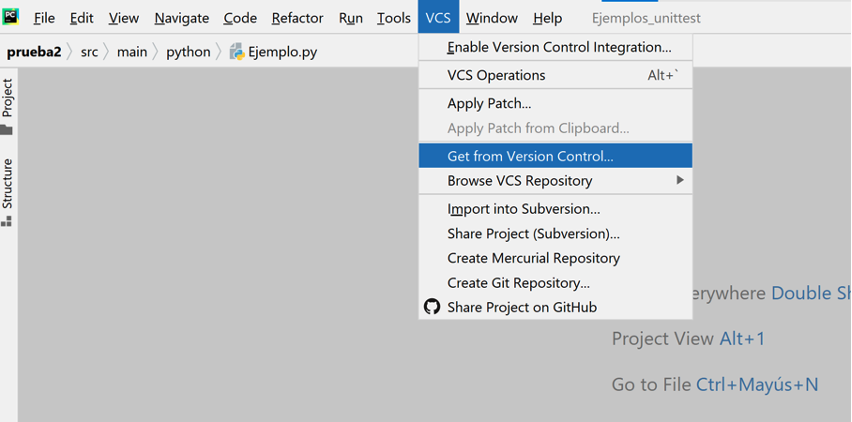
* Clone it configuring PyCharm.
* Clone it using a GitHub URL.

**Clone the repository configuring PyCharm settings**

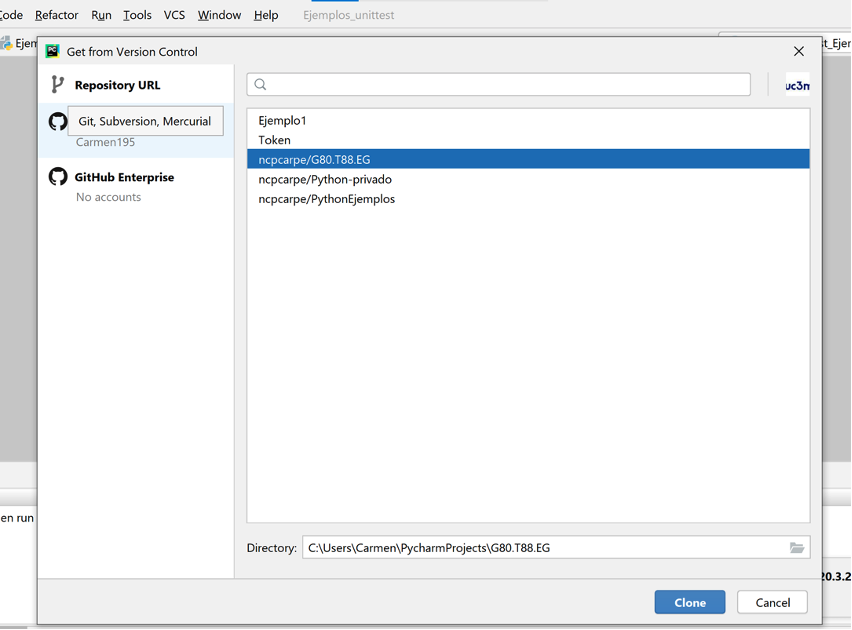
The picture below shows how to configure your GitHub credentials in Pycharm.



Once you have configured your GitHub account on PyCharm, go to **VCS / Get From Version Control**



In the following window you can select the GitHub repository you want to clone.



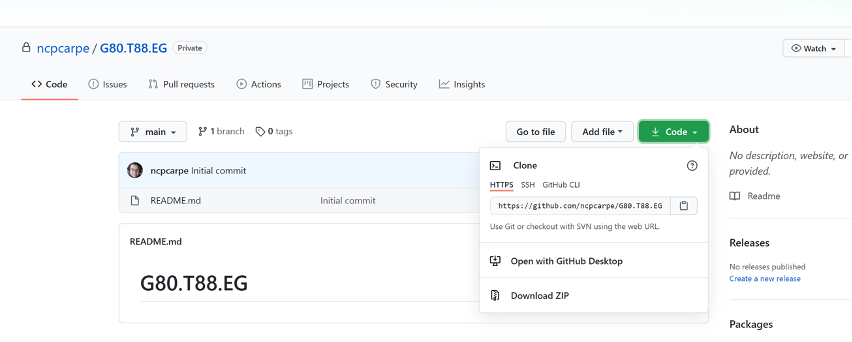
Finally click the Clone button (maybe the GIT client is not installed yet, in this case you must install it).

Depending on the configuration, PyCharm may ask whether you want to show it in the same window or a new one, and finally, it shows the cloned repository.



**Clone the repository using a GitHub URL**

Another option is to obtain the URL of the repository. In the main page of the GitHub repository, we click on Code and copy the https address.



Once copied, go to PyCharm and in **VCS / Get From Version Control**, click on **Repository URL,** select Git in **Version Control**, and paste the URL.



Finally click on the Clone button. If you don't have Git installed, you will receive an error message with an option to install it.

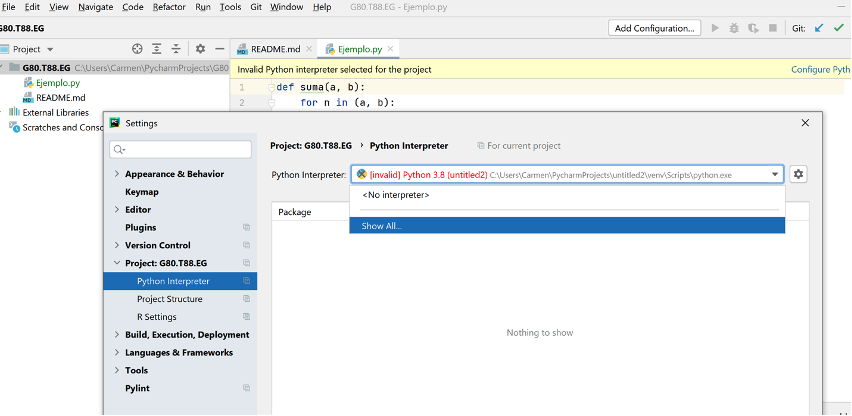
Whichever chosen method, at the end of the process, the local repository is linked to the remote one, and we can now download and upload code.

## 

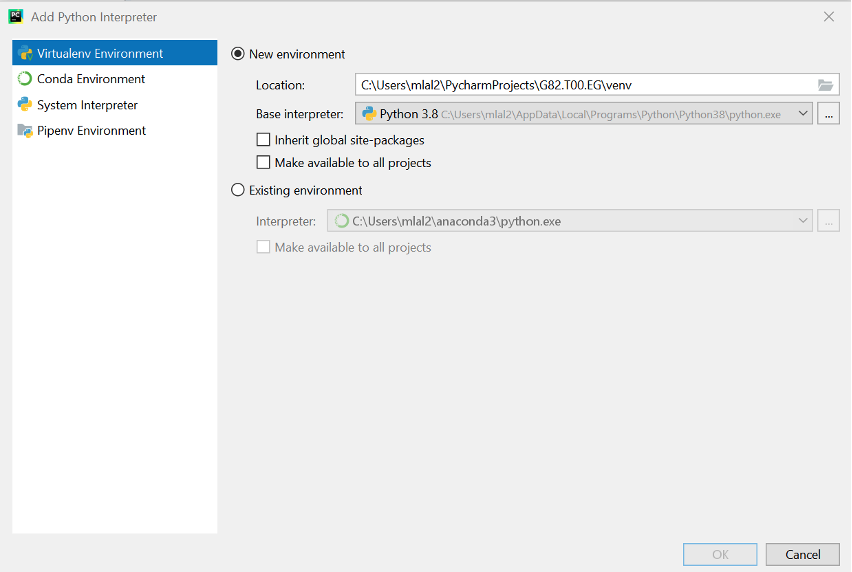
## Step 4: Configure the Python Interpreter

In this step you can choose the Python interpreter for your project. If you have several versions of the Python interpreter, choose the most appropriate for your project. You can configure the Python Interpreter in a virtual environment or not, but it is generally recommended to use virtual environments to avoid interference with other projects and libraries. **In this course we will work with virtual environments**.

Go to **Settings → Project: GXX.2023.TYY.EGx → Python Interpreter**



In the icon  select **Add, and** select **Virtualenv Environment** in the left bar. Then configure the virtual environment selecting the most adequate interpreter (in our case Python 3.8)



In case you don’t need a virtual environment, select **Configure Python interpreter,** and choose the interpreter for your project.

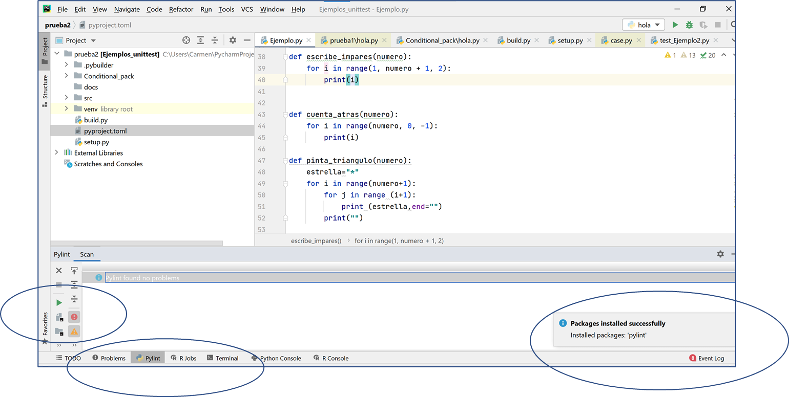
## Step 5: Add code to the repository

### Step 5.1: Create a test program

In your project, add a new python file and write a simple Python example program.

### Step 5.2: Install and use the static analyzer

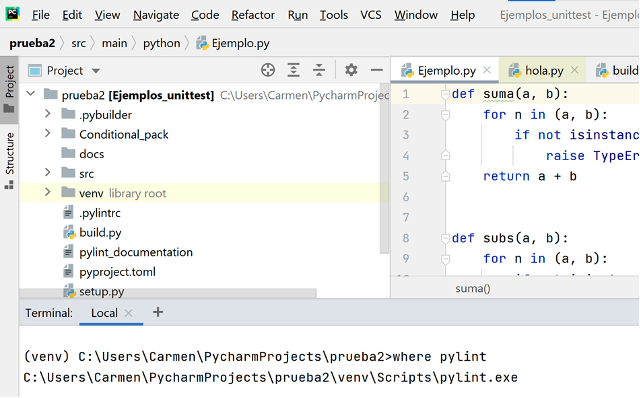
In a previous step we already installed the PyLint plugin. Now it is time to configure it. First of all, make sure that it is installed in your virtual environment.



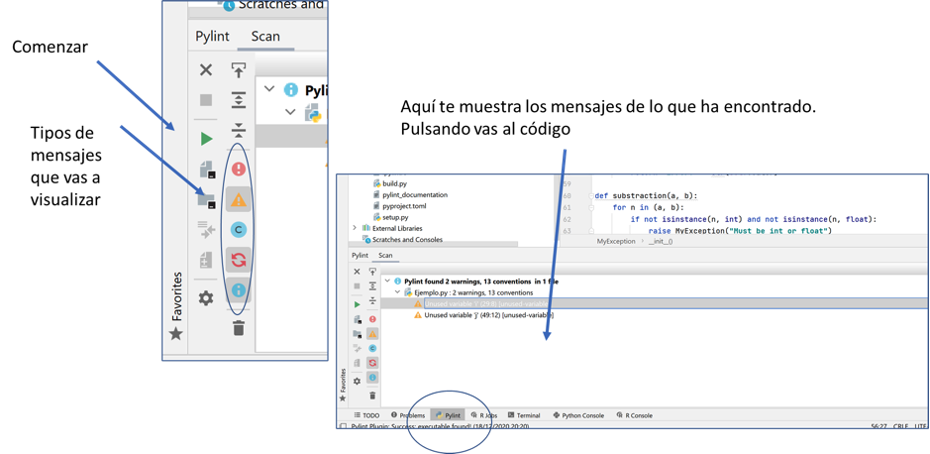
From PyLint, press **Run**. If the Pylint executable file is not installed you will receive an error message: in this case install it. Next, go to **Settings** to verify and / or configure the path. Press **Test** to verify the installation.



You can also check where it is from the **Terminal** with the command “**where Pylint”**.



To verify the current file, click on the green triangle.



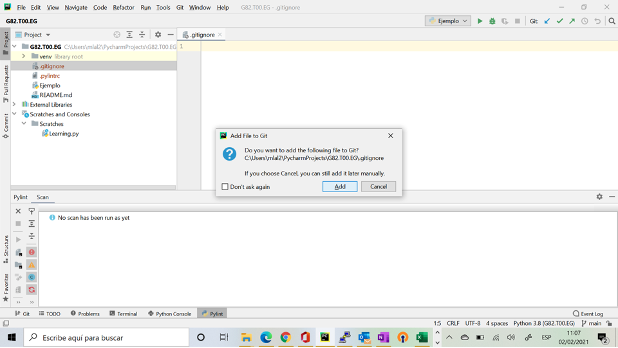
### Step 5.3: Dealing with the virtual environment

If you are using a virtual environment, it cannot be shared because it causes configuration problems in other computers. Therefore exclude it from the git repository.

To prevent those files from being sent to the git repository, you must configure the **.gitignore** file, **if you didn’t create it at** [**step 3.1**](#_r0o29ciuu7db)**.**

That is a text file that contains a list of files and folders we want to ignore and, therefore, exclude from the local and remote repository. You must place this file in the **root directory of the project**.

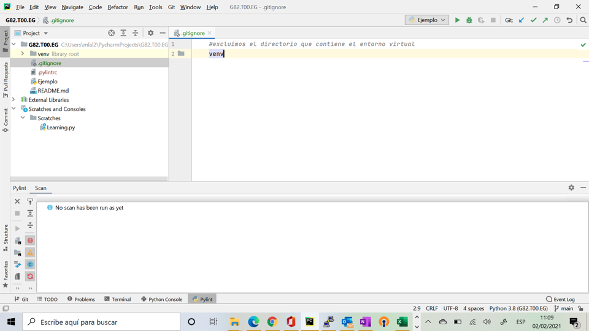
Therefore in the root folder of the project, click **File → New**, choose **file** and name the new file as **.gitignore**



Depending on the configuration, you can add this new file to the Git repository.

In **.gitignore** you can add in each line the name of a file or folder that you do not want to include in the git repository. You can include comments preceded by the character "#", and use wildcard characters such as "\*".

Initially, the file is blank. To avoid sending the folder venv, add its name in a line of the file.



If you want to configure your virtual environment in other machine, you need a list of the references included on it. To get a list of our references, execute the command from the PyCharm terminal:

pip freeze> requirements.txt

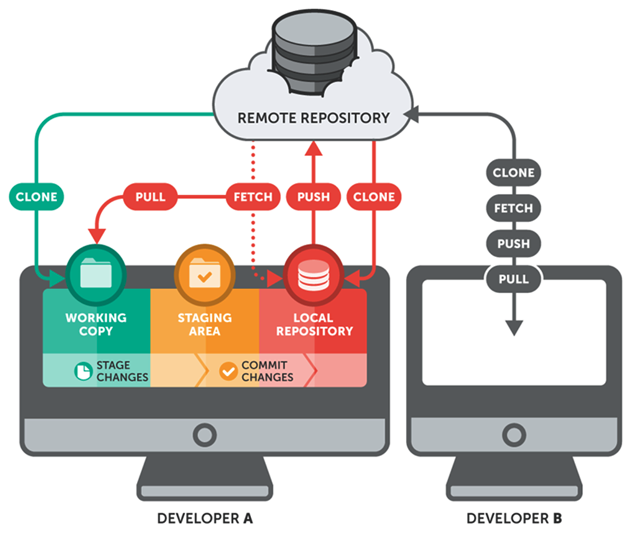
In this way you obtain a list with all the references in a text file. Then, you can automatically install on another computer using the command

pip install - r requirements.txt

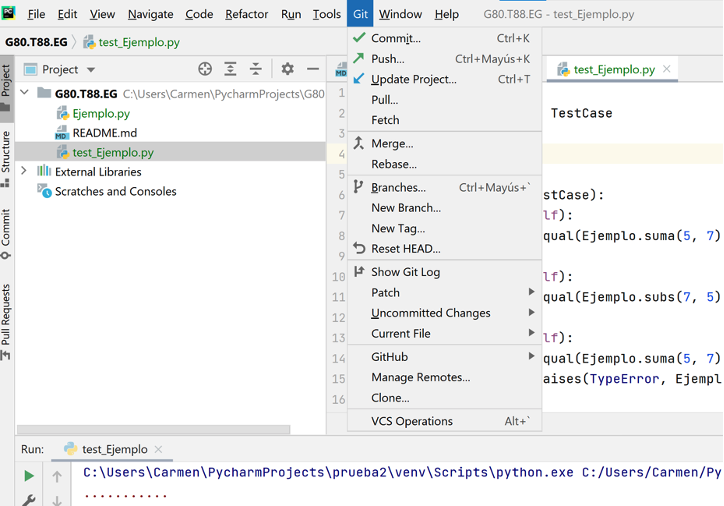
### Step 5.4: Changes and version control

You have to become familiar familiar with the common actions for the version management of code in your project:

* Uploading your code to a local and remote repository.
* Downloading code from the remote repository.
* Integrating changes.



In the Git menu of PyCharm you can find different operations.



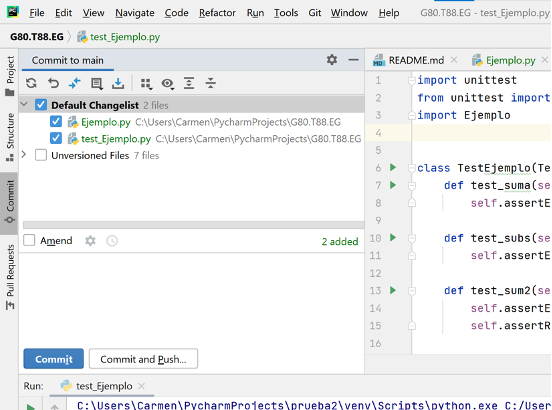
**Including files in the local and / or remote repository**

After writing a program or at the end of a working session, you should add the code to the code repository. The commands we have to use are:

* Commit. This command closes a local version with the changes made.
* Push. Close the version and upload it to the remote repository GitHub.

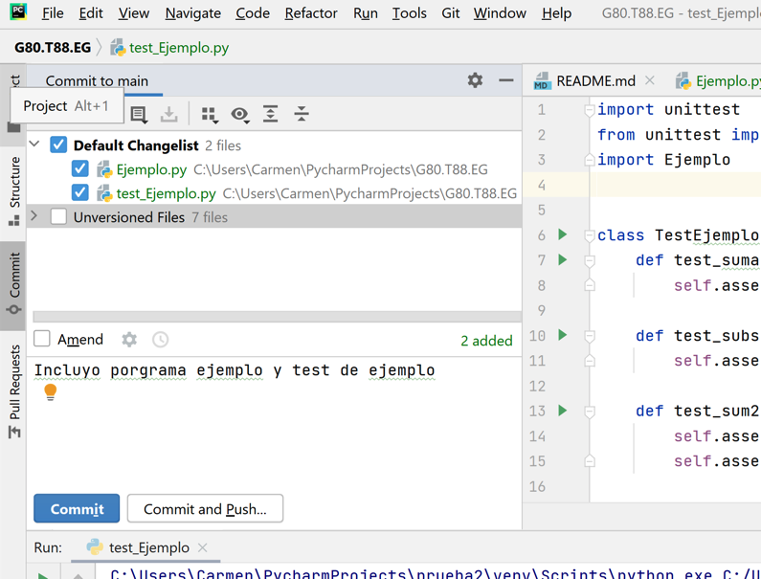
**Commit**

Click **Git-> Commit**



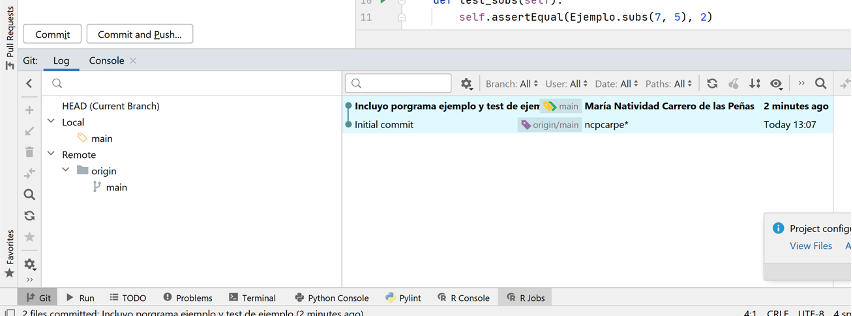
The **Default Changelist** shows the list of changes of the files already registered in the repository. The files to be included in the repository are checked. In the textbox **Amend** you must include a short explanatory text to identify that **Commit**.

The list **Unversioned files** shows files that are not yet included in the repository, that is, there is not a previous version of these files. You can include them (or part or them) checking each file.

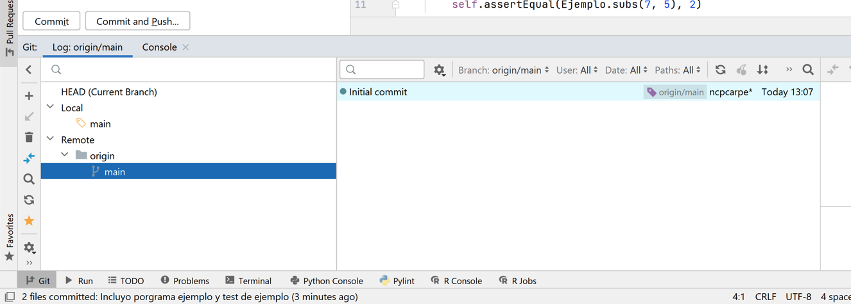


Once you have included the explanatory text you can **Commit** (close version locally) or **Commit and Push** (close version and upload it to GitHub).

In the tab below, **Log**, you can see the status of the local repository (left side of the tab) and the remote one (right side of the tab).



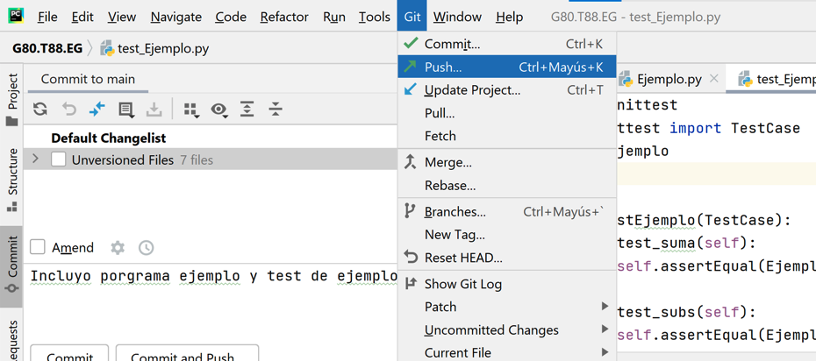
In the window below, you can see that the initial file is on the remote repository.

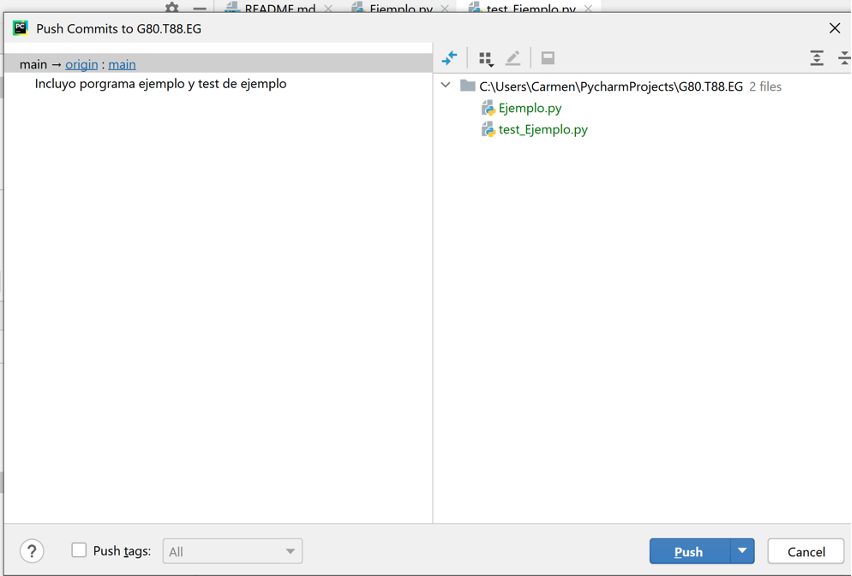


**Push**

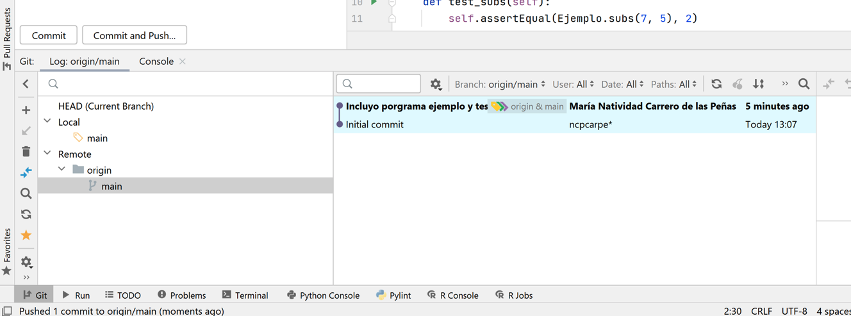
After committing, the changes are registered in the local repository in your computer, but they have not yet been sent to the GitHub server. When Pushing, the changes of the local repository are sent to the server and will be available for the rest of the users of the repository.

In the menu **Git → Push** you can see the files to be pushed as well as the description of the corresponding commit.

****

****

When you click **Push**, you can verify that the same files you have in the local repository appears on the remote one.



In the image above you can see the local and remote repositories on the left side of the window. On the right side you can see the commits, the user who made it and the comments included for each one.

#### Download code from the remote repository

You can download the code from the remote repository to your local one. For example, your teammate has uploaded changes, and you want to update your code with his changes.

You have to coordinate with your partner to make changes in different modules because if you both are updating the same module, you will have to validate the changes.

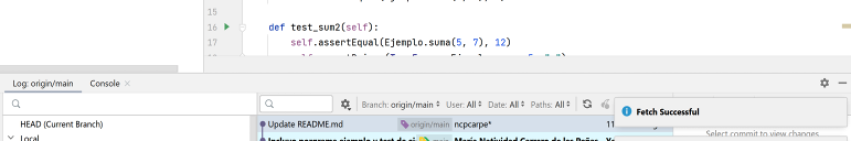
Take into account that if you are going to upload changes, you must synchronize your local repository with the remote one before.

So before downloading the changes from the remote repository, you must consider two actions:

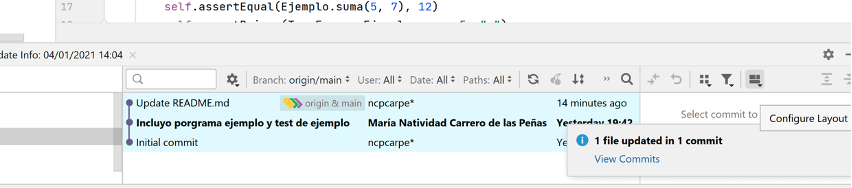
* **Fetch**: this command downloads the remote repository changes but doesn’t merge these changes with our code. In this way, you can compare the content of the remote repository with the local one.
* **Pull**: this command downloads the remote repository to the local one and merges the code, (merge in Git terminology), with our local version.
* **Update Project**: this command executes a fetch and then a pull, that is, it checks for changes and updates the local repository.

Let's try these commands:

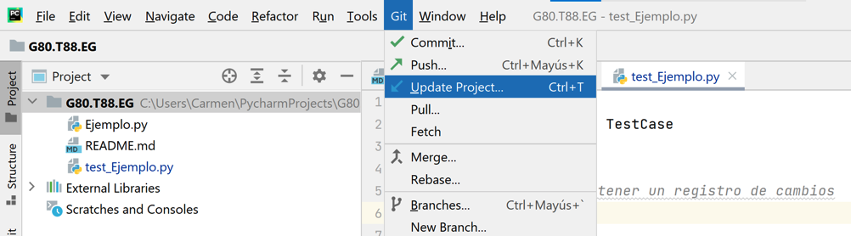
Executing the Fetch command you can see whether there are changes made by another collaborator.



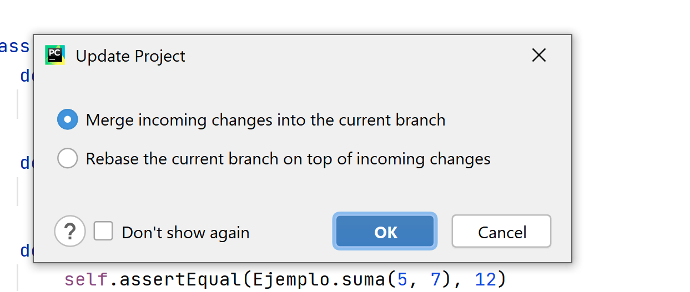
The local repository is updated when you execute the **Pull command**. In case you have several branches, you have to confirm where you want to bring the changes.



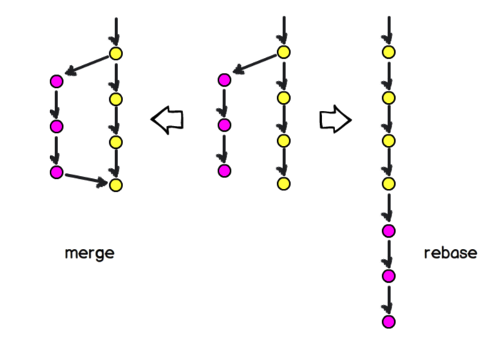
Update Project directly updates the local project with the content on the server.



When you update the project you can choose between two options: **Merge** or **Rebase**.



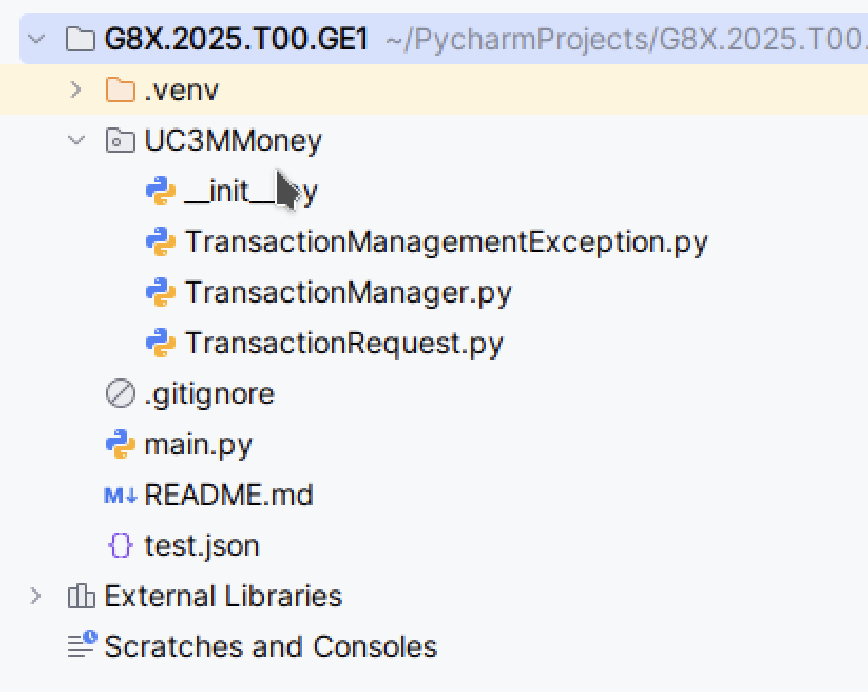
What both **Merge** and **Rebase** do is integrate the changes. The difference is in how the commit history looks. The following image shows the differences between both operations.



* **Merge** generates a commit with the merged code, but you can still see the yellow developer's commits and those made by the purple developer.
* **Rebase** puts the purple developer commits behind the yellow ones.

## Step 6: Insert the code of UC3MMoney

Finally you have to add the code of the UC3MMoney project. We are going to work in this guided exercise. Unzip the file with the code and copy the files main.py, test.json as well as the folder UC3MMoney to the main folder of your project. The result must look like this:

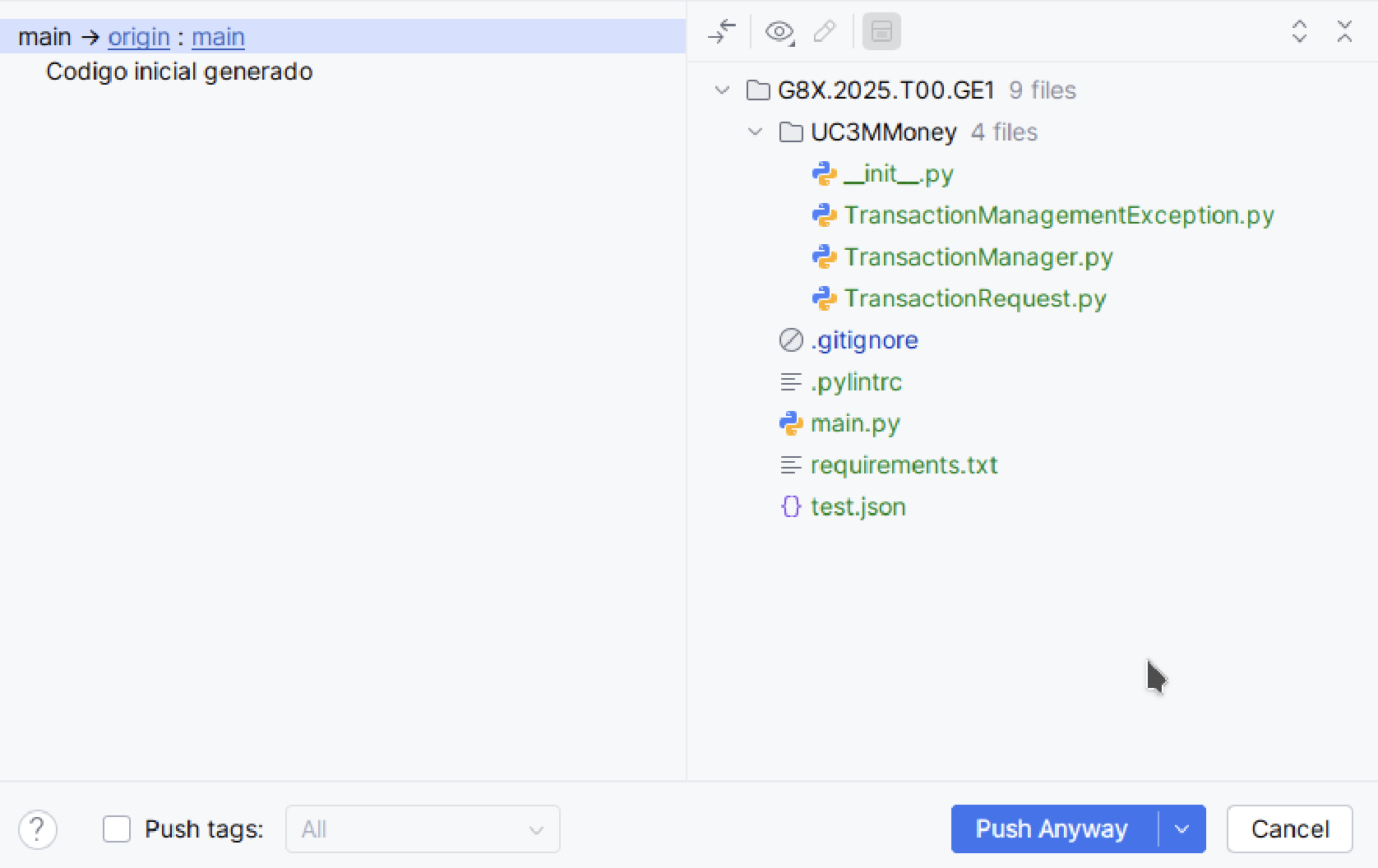


Depending on the configuration you have, PyCharm may ask you whether you want to add the files to git.

Remember that you must update the references.txt file in order to include all the references you need. Execute in the terminal the following command:

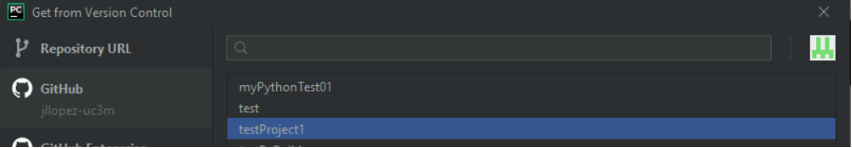
pip freeze> requirements.txt

Finally you can **commit** **and push** the files.



Once all the files are on the server, the other member of the team must download the project and configure it.

To do this, the steps to connect to the Git repository and download the project must be repeated.



When PyCharm downloads the project, it will detect that there is no virtual environment and it will ask you whether you want to create one. Also, if you previously created the requirements.txt file, PyCharm will detect the project references and install them automatically. In case of problems with the automatic configuration of the virtual environment, follow the steps in section 5.3 to create the virtual environment, and then, if you have the requirements.txt file with the list of all the references, execute the following command in the terminal.

pip install -r requirements.txt

In this way, the virtual environment is configured with the same references that we originally had. For this reason, It is crucial to execute the “pip freeze” command before committing to make sure we have the latest version of the references available.

# ANNEX C. STEPS TO ESTABLISH A CODING REGULATION

When starting a software project, it is necessary to establish a coding standard, accepted and implemented by the whole team, to make sure that all the programmers of the project work in a coordinated way. The intended purpose of this standard is to help build programs that are correct, understandable and easy to maintain. The complete source code should reflect a similar style as if a single programmer had written the entire system at one time.

When establishing a coding standard, we should make a distinction between rules and recommendations. A rule is a statement that must be complied with, while a recommendation is a valid rule in general, but that can admit or require exceptions.

One of the most used style guides in the Python world is PEP8

* PEP 8 – Style Guide for Python Code: <https://peps.python.org/pep-0008/>
* PEP 8 in Spanish - Style Guide for Python Code | Python resources: <http://www.recursospython.com/pep8es.pdf>

As part of the guided exercise, a coding standard must be defined and implemented so that you can validate your source code using the analyzer Pylint.

The steps shown below are indicative. Each team is free to add more elements to the coding standard or to dispense with those it deems unnecessary.

## Step 1. Standard for the Organization of Files

### Source Code Files

* The standard should include a rule to determine the copyright legend format that all the files should have. For example, a commented line in the source may include the date, the author and the company to which the person who developed the system belongs.
* The standard should establish a rule to determine how to manage all the source files, as well as the rest of the files of interest (executables, function libraries, etc.). In the practices of the subject, the GitLab service deployed for this purpose will manage the version control of the files.

### Class Files

* The standard should establish whether a separate file is used for each class. It should also consider the name of those files.
* The standard should determine whether each class file includes a header to provide some information about it, how to use the class and other considerations to ease its subsequent revision/update.

## Step 2: Standard for Names and Variables

### Classes and Members of Classes

* The standard should include rules for the naming of classes, interfaces and data types defined by the developer.
* The standard should set out the rules for the naming of fields, methods, properties and constants.

### Visibility

* The standard should establish whether the variables (instance or static) can be public or properties should be used instead.
* The standard should also include rules about the format that makes up the source code. For example, it should be considered whether the programmer must include only one statement per line and the alignment of consecutive declarations and initializations (like in a table).
* The standard should set out where to initialize the fields or variables. The rules should also consider that "for" loops require an index and that this index must be initialized
* The standard should determine whether the "this" keyword must be added when referencing class' attributes in a method, in order to distinguish them from the local variables.

## Step 3: Standard for Methods

### Method structure

* The standard should determine whether or not there is a maximum size for methods. It should also provide guidelines for organizing method body into logical blocks, including descriptive comments..

### Indentation and braces

* The standard should establish the type of indentation to be applied to the source code.
* The standard should include recommendations on how to place the ‘(’, ‘)’ in loops, conditional and other sentences.

### Method definitions

* In case the parameters of methods do not fit in the same line of the declaration, the standard should set out how they will be declared.
* The standard should set out recommendations for the inclusion of a description before the declaration of a method. At this point, the standard may include different rules depending on whether the method is public or private.

## Step 4: Standard for exception handling

* The standard may give considerations to aspects such as the detection of errors in the input parameters, or whether the exceptions can be handled by the components themselves or by the main program.
* The standard should include rules for dealing with methods with high possibilities of throwing an exception.

## Step 5: Other standards

### Parameters splitting

* The standard should set out rules to determine the splitting of parameters and the placement of the parenthesis for the function in relation to its name. The rules should also consider the possibility of applying special rules for “for”, “while”, “if”, and other sentences.

## Step 6: Standard for the application design

The standard should include the design patterns that are necessary and relevant for the development of the proposed application. It should also cover how to implement each of these patterns and the instructions for their use.

**Each team must define its coding standard and include it in a PDF document named** “**CodingStandard.pdf**”**. This document must be included in the root directory of the eclipse project to be generated in this guided exercise.**

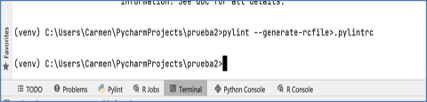
The coding standard will have a cover page, a table of contents and it will be organised into sections. For example, a section could be "Names and Variables": within this section, the rules associated with names and variables will be described. For each rule, you have to include examples of how to apply it and examples of how not to apply it. Finally, if the rule can be implemented in pylint, include also the rule you have included or modified.

## Step 7: Modify the Pylint configuration

Next you must configure the code analyzer to validate your project according to your coding standard.

To do this, open the PyCharm Terminal and execute the following command to save the configuration in a text file called **.pylintrc**:

pylint --generate-rcfile > .pylintrc



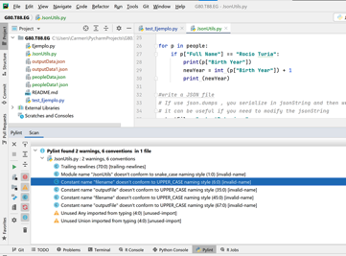
Once the file is generated, you can edit it to include the changes. You must consider between 10 or 15 rules.

You must be sure that in the Pylint Settings you have configured the path to this file.



**The modified .pylintrc text file must also be included in the project directory, as well as a screenshot with the result of the execution of Pylint before and after applying the coding standard.**

**Below we include a screenshot example before correcting the errors and/or warnings raised by Pylint.**



## Step 8: Review the source code and update the repository

After defining the coding standard, the team members should check that they comply with the established rules and recommendations, introducing the required changes for the code available in Aula Global to comply with the previously established code regulations.

After verifying that the latest version is in the local computer, students must apply the changes in the source code that are necessary to satisfy the previously defined coding standard. To achieve this, each team must check out beforehand the files to be modified.

After making the appropriate changes, the source code must be compiled in the local environment to verify that there are no new defects introduced in the project. Once the collection of the code does not produce any error, the modified files can be protected and uploaded to the repository. We recommended adding a comment before applying a “commit” in order to document the changes made to the code.