

EXERCISE SESSION 4

BIO 210

TAs: Alessandro Marin Vargas, Alberto Chiappa,
Sophia Becker and Lucas Stoffl

October 18th, 2021

EXECUTION PROBLEMS?

- Many of you might have encountered the problem that the cell cannot run anymore remaining stuck on [*].

```
[*]: 3 + 4
```

```
[*]: print(33.0 % 4)
```

It is possible that you are running an infinite loop within the kernel and that is why it can't complete the execution.

SOLUTION



Restart the kernel

WHAT IS A KERNEL?

A notebook kernel is a “**computational engine**” that executes the code contained in a Notebook document. Specifically, it uses an ipython kernel, which executes python code. The code runs on a server which can run locally on your machine or in the cloud (e.g. Noto). The results are turned into HTML and incorporated into the page you’re writing.

- When you open a Notebook, the associated kernel is automatically launched.
- When you run a cell, the kernel performs the corresponding computation and produces the results.

Sometimes the kernel might get stuck for different reasons. Therefore, you can apply the following suggestions:

- 1) Press “**Interrupt**” from the action bar (the stop symbol) and try to run again the cell
- 2) If it does not work, press “**Restart**” which will restart the kernel. Now, you have to run again all the cells from the first one.
- 3) If it still does not work, **restart the jupyter server**
 - If you are on noto, close the session and start a new one
 - If you have started jupyter lab from the terminal just press **ctrl + c** and call “jupyter lab” or “jupyter notebook” again

WINDOWS USERS

1. Go to: <https://gitforwindows.org/>
2. Download the latest version and start the installer
3. Follow the Git Setup wizard
4. Open the windows command prompt (or Git Bash)
5. Type: **git --version** → to verify Git was installed

MAC USERS

1. Navigate to: <https://sourceforge.net/projects/git-osx-installer/files/git-2.23.0-intel-universal-mavericks.dmg/download>
2. Download the latest version and start the installer
3. Follow the instructions
4. Open the command prompt "terminal"
5. Type: **git --version** → to verify Git was installed.

LINUX USERS

Git is already pre-installed on Linux, so you do not need to install it.

WINDOWS USERS

1. Follow the instructions of the following link: <https://docs.anaconda.com/anaconda/install/windows/>
2. Open anaconda prompt
3. Type: **python --version** → to verify python is installed

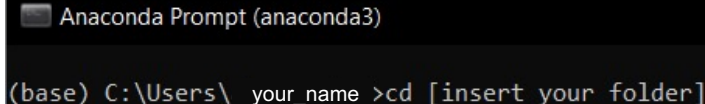
MAC USERS

1. Follow the instructions of the following link: <https://docs.anaconda.com/anaconda/install/mac-os/>
2. Open terminal
3. Type: **python --version** → to verify python is installed

LINUX USERS

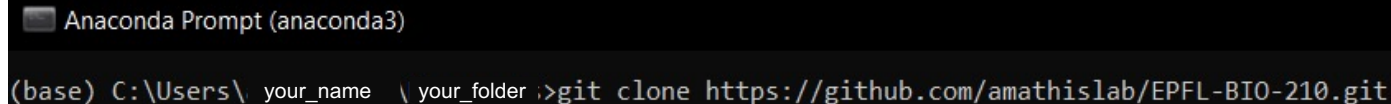
1. Follow the instructions of the following link: <https://docs.anaconda.com/anaconda/install/linux/>
2. Open terminal
3. Type: **python --version** → to verify python is installed

Go in the folder in which you want to clone the class repository using (**cd [insert your folder]**):



```
Anaconda Prompt (anaconda3)  
  
(base) C:\Users\ your_name >cd [insert your folder]
```

Clone the class repository using (**git clone** <https://github.com/amathislab/EPFL-BIO-210.git>):



```
Anaconda Prompt (anaconda3)  
  
(base) C:\Users\ your_name \ your_folder ;>git clone https://github.com/amathislab/EPFL-BIO-210.git
```

In previous lectures, you have seen the functions related to different Python packages. When developing a project, you might need to install a specific set of packages. Usually, you might have different projects with different requirements, and you do not want to have a conflict between packages. Therefore, a possible solution consists in using **Anaconda** and **virtual environments**.

Anaconda is a distribution of Python used for package management and deployment. Conda analyses the current environment (including everything currently installed and related version limitations) and works out how to install a compatible set of dependencies.

When Anaconda is installed, it comes with Python, around 250 packages and the virtual environment manager.

CREATE A VIRTUAL ENVIRONMENT

conda can be used to create environments to isolate your projects. This means that each project can have its own dependencies, regardless of what dependencies every other project has.

Create a virtual environment giving it a name **env_name**

```
conda create -n env_name
```

If you want to use a specific version of Python, you can specify it as:

```
conda create -n env_name python=3.8
```

Once you have generated an environment, you have to access it.

```
conda activate env_name
```

You should observe that on the left of the terminal (*base*) changed in (*env_name*). This means that you are in the virtual environment now. Here, you can install the packages that are required for your specific project.

For example, let's install jupyter lab in our virtual environment:

```
conda install -c conda-forge jupyterlab
```

Remaining inside the generated virtual environment, we can navigate to the class repository.

```
cd [path_to_EPFL-BIO-210]
```

Now, we can access to the notebooks using jupyter lab:

```
jupyter lab
```

Once, you have finished to work with your project you can exit your virtual environment with the following command:

```
conda deactivate env_name
```

Questions?