

# Programming fundamentals with Python

## Session 1

Pepe García [jgarciah@faculty.ie.edu](mailto:jgarciah@faculty.ie.edu)

2021-10-05



SCHOOL OF  
SCIENCE &  
TECHNOLOGY

# Plan for today

- Course introduction

# Plan for today

- Course introduction
- Start using VS Code



# Plan for today

- Course introduction
- Start using VS Code
- Learn a bit about CLI tools



# Plan for today

- Course introduction
- Start using VS Code
- Learn a bit about CLI tools
- Learn about version control systems



# Plan for today

- Course introduction
- Start using VS Code
- Learn a bit about CLI tools
- Learn about version control systems
- Introduction to Git



# Course introduction

- **Professor:** Pepe García
- **Email:** jgarciah@faculty.ie.edu
- **Please, ask me anything**



# Course introduction. Objectives

The objectives for this course are that:

- Students learn how to use industry grade tools for programming (git, pip...)
- Students will further their knowledge of the Python programming language.
- Students will understand how to apply programming to resolve real world problems.



# Course introduction. Grading

Section	Score %
Final Exam	25 %
Workgroups	25 %
Individual Work	40 %
Class Participation	10 %

In this course we'll use alpha grading (Honors, Excellence, Proficiency, Pass, Fail), using a normal distribution of grades and percentiles 15, 50, 85, 100.

If someone is 2 \* low, they get an automatic **Fail**

# Course introduction. Important dates

- Session 10th: Workgroup assignment is sent out (you'll have some days to work on it)
- Session 13th: Individual assignment is out (you'll have some days to work on it)
- Session 15th: Final exam

Questions?



Since Spyder is not working correctly for us, we'll start by changing to VS Code editor.

<https://code.visualstudio.com/>



## Demo

Let's see how to use VS Code:

- Opening a folder to start working on it (we can drag and drop the folder to the editor)
- Installing the Python extension
- Creating files (`File > New File`, `File > Open...`)



# Installing Git

If you don't have it installed, you can get it from  
**<https://git-scm.com/downloads>**



# Command line

The command line allows users to navigate the computer and manage it. We can do almost the same things with the command line and a graphical user interface.

```
Windows\system32\cmd.exe - ping 192.168.1.1 -t
from 192.168.1.1: bytes=32 time=1ms TTL=100
from 192.168.1.1: bytes=32 time=1ms TTL=100
from 192.168.1.1: bytes=32 time=167ms TTL=100
from 192.168.1.1: bytes=32 time=2ms TTL=100
from 192.168.1.1: bytes=32 time=2ms TTL=100
from 192.168.1.1: bytes=32 time=1ms TTL=100
Request timed out.
from 192.168.1.1: bytes=32 - MISCOMPARE at offset 1 - time=2ms
Request timed out.
from 192.168.1.1: bytes=32 time=4ms TTL=100
from 192.168.1.1: bytes=32 time=5ms TTL=100
from 192.168.1.1: bytes=32 time=387ms TTL=100
from 192.168.1.1: bytes=32 time=2ms TTL=100
from 192.168.1.1: bytes=32 time=2ms TTL=100
from 192.168.1.1: bytes=32 time=1ms TTL=100
from 192.168.1.1: bytes=32 time=1ms TTL=100
from 192.168.1.1: bytes=32 time=1ms TTL=100
from 192.168.1.1: bytes=32 time=108ms TTL=100
Request timed out.
```

## Disclaimer

In this slide set, every time you see a \$ at the beginning of the line it means that that's a command to be written in the terminal.





## Disclaimer

In this slide set, every time you see a **\$** at the beginning of the line it means that that's a command to be written in the terminal.

## Disclaimer 2

If you're on Mac, we will use the **Terminal** for today's session, if you're on Windows, please open **Git Bash**.



# Listing files

We can **list files** in a folder using the **ls** command.

```
$ ls
```

```
Desktop  Documents Downloads Library  Movies      Music      Pi
```



# Changing directories

We change directories (move around) using **cd**.

```
$ ls
```

```
Desktop    Documents Downloads Library    Movies    Music    Pi
```

```
$ cd Desktop
```

# Changing directories

We can go to *upper* directories using **cd ..**

```
$ ls
```

```
Desktop  Documents Downloads Library  Movies  Music  Pi
```

```
$ cd Desktop
```

```
$ cd ..
```

```
$ ls
```

```
Desktop  Documents Downloads Library  Movies  Music  Pi
```

# Getting current directory

We can see where we are with the **pwd** command

```
$ pwd  
/Users/pepe
```

```
$ cd Desktop
```

```
$ pwd  
/Users/pepe/Desktop
```

**pwd** stands for print working directory



# Creating directories

One can create directories using the `mkdir` command:

```
$ pwd  
/Users/pepe
```

```
$ mkdir hello_dolly
```

```
$ cd hello_dolly
```

```
$ pwd  
/Users/pepe/hello_dolly
```



## Why do we need version control software?

Have you ever found yourselves with a bunch of copies of a file (an assignment maybe?) that you save to not lose what you've created?



**Version control** is the process of handling programs, versions, changes, and differences in files.

With **version control** systems we can see:

- **Who** made changes

The version control system we're going to use in this course is **git**.



**Version control** is the process of handling programs, versions, changes, and differences in files.

With **version control** systems we can see:

- **Who** made changes
- To **which files**

The version control system we're going to use in this course is **git**.

**Version control** is the process of handling programs, versions, changes, and differences in files.

With **version control** systems we can see:

- **Who** made changes
- To **which files**
- **When** did they do it

The version control system we're going to use in this course is **git**.

**Version control** is the process of handling programs, versions, changes, and differences in files.

With **version control** systems we can see:

- **Who** made changes
- To **which files**
- **When** did they do it
- **Why** did they do it

The version control system we're going to use in this course is **git**.

Git terminology can be very broad, but we'll focus on the parts that matter



# Working directory

The **working directory** is the folder in which our code will be. The contents of this folder will be controlled by **git**.

Whenever we're happy about the state of a file, we move it to the **staging area**. In the **staging area** we save files that are ready to be saved.



The **local repository** is the place in which we store all the changes made to all the files of our projects, over time.



# Creating our first repository

## Practice (5 mins)

- Create a folder called **my-first-repo** in your desktop
- Navigate to it using the terminal (**cd**)
- Open **VS Code**, create a python file and save it in **my-first-repo** folder
- In the terminal, initialize the repository with **git init**





# Git operations

We can always see the status of our repository:

```
$ git status
```

```
On branch master
```

```
No commits yet
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
file.py
```

```
nothing added to commit but untracked files present (use "git add" to track)
```



# Git operations

We can use **git add file.py** to add the file to the staging area, in which we store the files ready to be committed.

```
$ git add file.py
```

```
$ git status
```

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

```
new file:   file.py
```



When there is a meaningful change we want to save, we use **git commit** to save it to our local repository.

We use **git commit -m “message”** and try to use a meaningful description of the changes we just made.

```
$ git commit -m "add file.py to git"
[master (root-commit) 123cd8b] add file.py to git
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file.py
```



# Git concepts



SCHOOL OF  
SCIENCE &  
TECHNOLOGY

One of the most powerful features of **git** is handling changes. Let's add this function to our **file.py**.

```
def func(a, b):  
    return a + b
```



# Git operations

And let's see the changes now! **git diff**

Git will show the lines we added with a **+** sign before, and those we removed with a **-** sign

```
$ git diff
```

```
diff --git a/file.py b/file.py
index e69de29..c09bd0e 100644
--- a/file.py
+++ b/file.py
@@ -0,0 +1,3 @@
+
+def func(a, b):
+    return a + b
(END)
```

In the picture we can see we added three lines



# Commit the last changes

## Practice

Now, let's commit our latest changes



# Git operations

Other of the cool features of **git** is watching the history of our repository. With `git log` we will see a log of all the changes that happened to our repository.

```
$ git log
```

```
commit 123cd8b45ae31065cdd7cf0ecd8ce83b444886db (HEAD -> master)
Author: Pepe García <pepe@pepegar.com>
Date:   Mon Nov 11 23:55:49 2019 +0100
```

```
add file.py to git
```





Now, let's create an account in Github!

Go to [github.com](https://github.com) and create an account. (if you've one already, that's OK)

Images and inspiration drawn from

**How to teach Git**

**Learn git concepts, not commands**

