

# Programming Lab

Lezione 1

*Introduzione e strumenti*

Stefano Alberto Russo

# Benvenuti

**Corso:** Laboratorio di programmazione (3 CFU) 

→ Nome esteso del corso comprendente la parte del prof.

Caravagna: *Introduzione alla programmazione e laboratorio*

**Docente:** Stefano Alberto Russo (io 🙌)

**Sito Web (repository):** [github.com/sarusso/ProgrammingLab/](https://github.com/sarusso/ProgrammingLab/)

**Ricevimento:** libero, scrivetemi a [stefanoalberto.russo@phd.units.it](mailto:stefanoalberto.russo@phd.units.it)

**Tutors:** Valentina Blasone e Lucrezia Valeriani

# Chi sono

Uno che ha fatto un po' di tutto e ne ha viste di cotte e di crude.

Ho studiato prima informatica e poi fisica computazionale.

Sono stato 3 anni al CERN lavorando su Data Science e sistemi Big Data

Poi sono tornato a Trieste a lavorare in una startup dove facevamo soluzioni di monitoraggio energetico, sia Hardware che Software

Poi ho deciso di fondare la mia di startup, tra Londra e Hong Kong (Machine Learning per monitoraggi infrastrutturali e ambientali, abbiamo lavorato anche sul Morandi).

Ad oggi sono a fare ricerca su “science platforms” all'INAF

# Argomenti del corso, lezione per lezione

- 1) Intro del corso e strumenti da "laboratorio": la shell, Git, gli IDE, Repl.it etc.
- 2) Python: tipi dati, costrutti, funzioni, moduli, be pythonic.
- 3) Interagire con i file ed il formato CSV.
- 4) Gli oggetti in Python
- 5) Le eccezioni ed il flusso try-except
- 6) Controllo degli input e sanitizzazione

# Argomenti del corso, lezione per lezione

7) Testing e unit tests

8) Lavorare veramente 1: creiamo un modello

9) Lavorare veramente 2: fittiamo un modello

10) Lavorare veramente 3: valutiamo un modello

11) Esercitazione esame

12) Esercitazione esame

# Modalità d'esame

**Nota:** che voi vi sappiate districare tra i vari strumenti che vedremo nel corso (Git e testing in particolare) è argomento d'esame, non è un extra!

## **Esame:**

- Compito di esame rilasciato qualche giorno prima dell'appello
- Esame orale in cui si discute lo svolgimento del compito
  - Svolgimento libero, a casa
  - Testing automatico con unit-testing
  - Il compito va consegnato via mail o come link a repository Git + commit hash

# Perchè questo corso (fatto in questo modo)

Concetto: non farvi perdere tempo a far funzionare le cose nei prossimi anni.

Non è un buon uso del vostro tempo

Meglio fare “cose fiche”, no?

..ma serve un po' di sforzo all'inizio (leggi: questo corso 😊)

# Perchè questo corso - un'analogia

Mario monta la sua nuova TV alla svelta, tira una teleferica per il cavo dell'alimentazione, la attacca al muro col Patafix, e poi disabilita la chiave del WiFi perchè non riesce a configurarla sulla TV.

Mario passa le prossime serate a litigare con cavi mangiati dal cane, con la TV sbilenca e con il WiFi lento perchè intanto gliel'hanno fregato i vicini.



# Perchè questo corso - un'analogia

Bill invece, investe un paio d'ore nel montare la sua nuova TV e passa il cavo per bene, fa un buco col trapano per il supporto, e configura il WiFi correttamente.

Bill passa le prossime serate a godersi la sua serie preferita su Netflix in pace.

ecco, siate come Bill.

# Perchè questo corso

“ma io ho fretta”

“ma io voglio solo fare una cosa rapida”

“ma io non sono un ingegnere software!”

“ma io non ho le basi”

“ma io voglio iniziare subito a fare codice”

# Perchè questo corso

..cosa vi aspettate che io dica ora?

3..

2..

1..

# Perchè questo corso

...che in realtà avete ragione.

Perchè in effetti io ho omesso un dettaglio, torniamo alla slide di prima

# Perchè questo corso

Bill invece, investe un paio d'ore nel montare la sua nuova TV e passa il cavo per bene, fa un buco col trapano per il supporto, e configura il WiFi correttamente.

Bill passa le prossime serate a godersi la sua serie preferita su Netflix in pace.

ecco, siate come Bill\*.

***\* se sapete che poi paga!***

# Perchè questo corso

Ecco, lo scopo di questo corso è di mostrarvi che se facciamo le cose per bene nel gestire il codice poi tutto ciò paga. Un po' come montare la TV correttamente. E vi converrà farlo sempre, perchè non perderete tempo su cose stupide.

Perchè di fatto, voi avrete a che fare col codice per tutto il corso di laurea, e probabilmente per buona parte della vostra vita.

Agli statistici in sala: anche voi! Il mestiere sta cambiando tanto, non vorrete essere già vecchi vero?

# Organizzazione delle ore

**Prima ora:** teoria (Python, principalmente)

**Seconda ora:** pratica (con me e gli assistenti)

**Oggi, alla fine della prossima ora dovreste tutti sapere\*:**

- 1) Come si usa Repl.it configurato con Bash
- 2) Come si fa un commit da Repl su GitHub
- 3) Come si esegue uno script Python dentro Repl

*\*liberi di usare Python e Git sul vostro computer, ovviamente.*

# Iniziamo! ...con gli strumenti

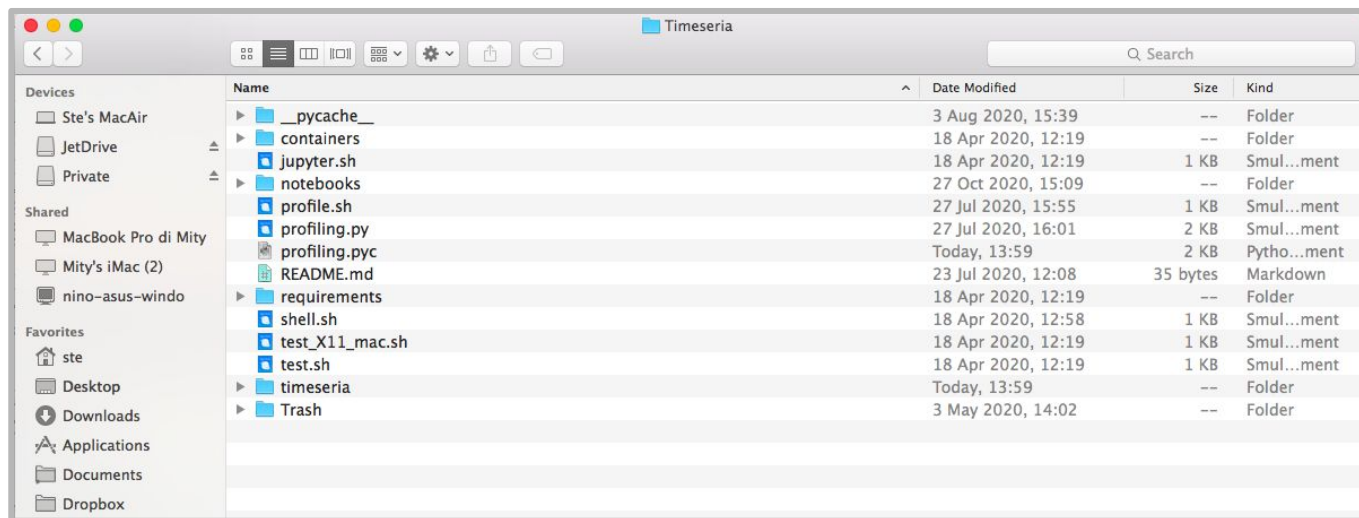
“Datemi sei ore per abbattere un’ albero e ne spenderò le prime quattro per affilare l’ascia”

*- Abraham Lincoln*



# Strumenti: il File Manager

E' quello con cui si vedono le cartelle e i files del computer. Configuratelo per vedere anche i file nascosti e le estensioni!



# Strumenti: la Shell (anche Terminale / Console)

E' quella cosa con cui si interagisce con il computer in via testuale, senza bottoni che automatizzano le cose. E' come si fanno le cose sul serio senza usare un ambiente preconfezionato. Su sistemi Unix in genere è "bash"

```
ste@Stes-MacAir:sarusso.github.io (master) $ nano index.html
ste@Stes-MacAir:sarusso.github.io (master) $ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
ste@Stes-MacAir:sarusso.github.io (master) $ git log
commit fac859465072e59d17bb24aeadae026c11612647 (HEAD -> master, origin/master, origin/HEAD)
Author: Stefano Alberto Russo <stefano.russo@gmail.com>
Date:   Fri Nov 6 12:03:07 2020 +0100

    Added ProgrammingLab stuff

commit 31ed8586aa7e2e655568d721247b3e654569d942
Author: Stefano Alberto Russo <stefano.russo@gmail.com>
Date:   Sat Apr 18 18:23:08 2020 +0200

    Added empty index.

commit b89ee81d8dc6119f8238e7e15e33dd362324fc04
Author: Stefano Alberto Russo <stefano.russo@gmail.com>
Date:   Sat Apr 18 18:20:16 2020 +0200

    Added Timeseria doc first stub.

commit 05884aea830a5c60d368afb41cc1459ede6514d3
Author: Stefano Alberto Russo <stefano.russo@gmail.com>
Date:   Sat Apr 18 16:04:21 2020 +0200

    Initial commit
ste@Stes-MacAir:sarusso.github.io (master) $
```

## Strumenti: l'Editor del codice

E' quella cosa con cui scrivete il codice. Nota: è tassativo impostare l'editor a usare gli spazi e non i tab. Indentazione a 4 spazi per Python (come vedremo)

A screenshot of a code editor window titled "transformations.py - /Users/ste/Devel/Projects/Timeseria/timeseria". The editor has a light gray background and a search bar at the top right containing "Q Search". The code is written in Python and includes comments. It defines imports from the datetime module, DataTimeSlot, DataTimeSlotSeries, TimePoint, DateTimePointSeries, compute\_coverage, is\_almost\_equal, is\_close, and TimeUnit. It sets up logging with a logger named \_\_name\_\_. A constant HARD\_DEBUG is set to False. There are green dashed lines indicating a section break. A class Transformation(object) is defined with two methods: \_\_str\_\_(cls) which returns a string representation of the transformation, and process(self, \*args, \*\*kwargs) which calls self.\_process(\*args, \*\*kwargs). At the bottom, there is a status bar showing "Saved: 18 October 2020 at 00:55 · Length: 15,960 · Encoding: Unicode (UTF-8)".

```
from .time import dt_from_s, s_from_dt
from datastructures import DataTimeSlot, DataTimeSlotSeries, TimePoint, DateTimePointSeries
from utilities import compute_coverage, is_almost_equal, is_close
from units import TimeUnit

# Setup logging
import logging
logger = logging.getLogger(__name__)

HARD_DEBUG = False


=====
# Base Transformation
=====


class Transformation(object):

    @classmethod
    def __str__(cls):
        return '{} transformation'.format(cls.__name__.replace('Operator', ''))






    def process(self, *args, **kwargs):
        return self._process(*args, **kwargs)
```

Saved: 18 October 2020 at 00:55 · Length: 15,960 · Encoding: Unicode (UTF-8)

# Strumenti: l'Editor del codice

**THE OVERFLOW** 

Essays, opinions, and advice on the art of computer programming from Stack Overflow.




Latest Newsletter Podcast Company Developer Hiring IT-Recruiting (German)

insights JUNE 15, 2017

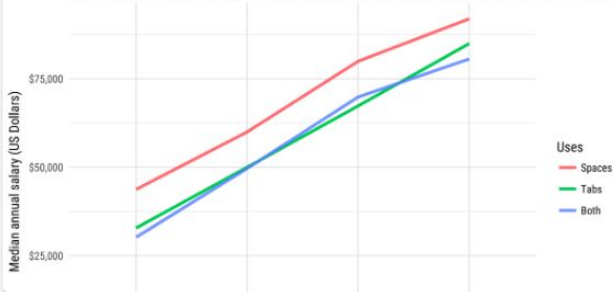
## Developers Who Use Spaces Make More Money Than Those Who Use Tabs

Do you use tabs or spaces for code indentation? This is a bit of a “holy war” among software developers; one that’s been the subject of many debates and in-jokes. I use spaces, but I never thought it was particularly important. But today we’re releasing the raw data behind the Stack Overflow 2017 Developer Survey,...

**David Robinson**  
Data Scientist (former)

### Salary differences between developers who use tabs and spaces

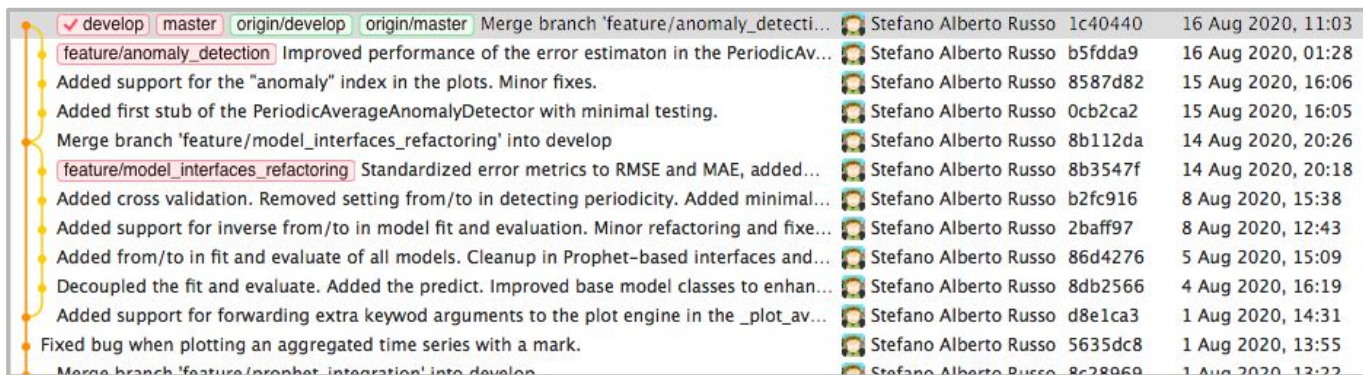
From 12,426 professional developers in the 2017 Developer Survey results, who provided tabs/spaces and salary



Uses	Median annual salary (US Dollars)
Spaces	~\$65,000
Tabs	~\$55,000
Both	~\$55,000

# Strumenti: il sistema di versionamento (Git)

E' quella cosa dove viene tenuta traccia di tutte le modifiche che avete fatto nel codice. Usate SEMPRE un sistema di versionamento, mai che vada Dropbox (che ha la history). Git è la soluzione più indicata.



✓ develop	master	origin/develop	origin/master	Merge branch 'feature/anomaly_detecti...	Stefano Alberto Russo	1c40440	16 Aug 2020, 11:03
feature/anomaly_detection				Improved performance of the error estimaton in the PeriodicAv...	Stefano Alberto Russo	b5fdda9	16 Aug 2020, 01:28
				Added support for the "anomaly" index in the plots. Minor fixes.	Stefano Alberto Russo	8587d82	15 Aug 2020, 16:06
				Added first stub of the PeriodicAverageAnomalyDetector with minimal testing.	Stefano Alberto Russo	0cb2ca2	15 Aug 2020, 16:05
				Merge branch 'feature/model_interfaces_refactoring' into develop	Stefano Alberto Russo	8b112da	14 Aug 2020, 20:26
feature/model_interfaces_refactoring				Standardized error metrics to RMSE and MAE, added...	Stefano Alberto Russo	8b3547f	14 Aug 2020, 20:18
				Added cross validation. Removed setting from/to in detecting periodicity. Added minimal...	Stefano Alberto Russo	b2fc916	8 Aug 2020, 15:38
				Added support for inverse from/to in model fit and evaluation. Minor refactoring and fixe...	Stefano Alberto Russo	2baff97	8 Aug 2020, 12:43
				Added from/to in fit and evaluate of all models. Cleanup in Prophet-based interfaces and...	Stefano Alberto Russo	86d4276	5 Aug 2020, 15:09
				Decoupled the fit and evaluate. Added the predict. Improved base model classes to enhan...	Stefano Alberto Russo	8db2566	4 Aug 2020, 16:19
				Added support for forwarding extra keywod arguments to the plot engine in the _plot_av...	Stefano Alberto Russo	d8e1ca3	1 Aug 2020, 14:31
				Fixed bug when plotting an aggregated time series with a mark.	Stefano Alberto Russo	5635dc8	1 Aug 2020, 13:55
				Merge branch 'feature/prophet_integration' into develop	Stefano Alberto Russo	8c28969	1 Aug 2020, 13:22

Tutorial di Michele Rispoli (tutor dell'anno scorso):

[https://github.com/drOpZ/proglab2021-tutors/blob/master/git\\_quickstart.md](https://github.com/drOpZ/proglab2021-tutors/blob/master/git_quickstart.md)

# Strumenti: l'IDE (Integrated Development Environment)

E' un sistema che integra in modo integrato File Manager, Editor del codice, la Shell, il sistema di versionamento e altre funzionalità come il debugger.

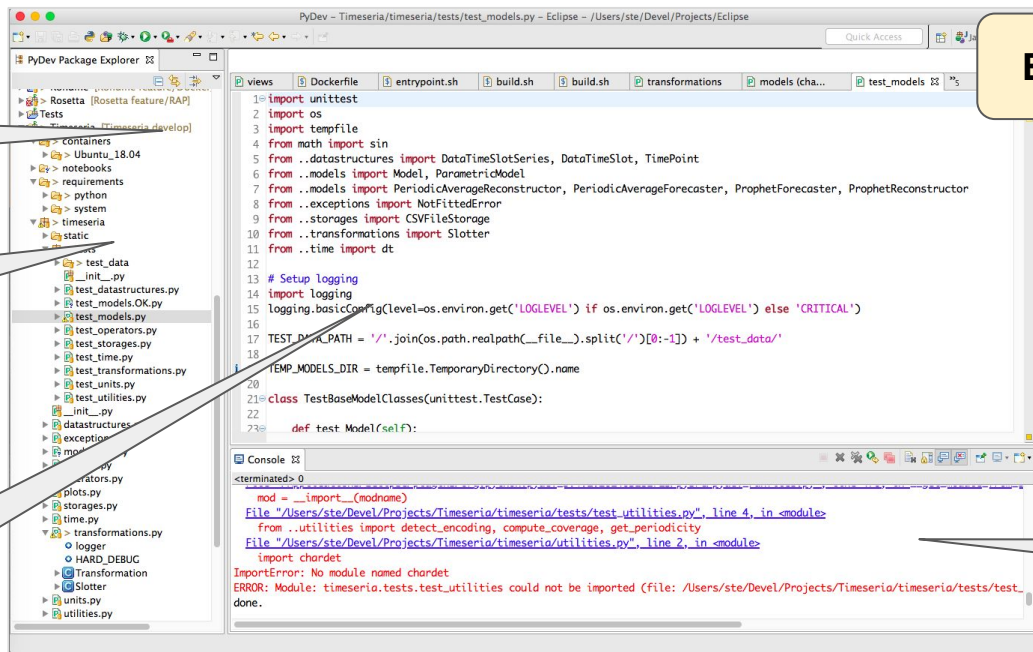
Sistema di  
versionamento

File  
Manager

Editor del  
codice

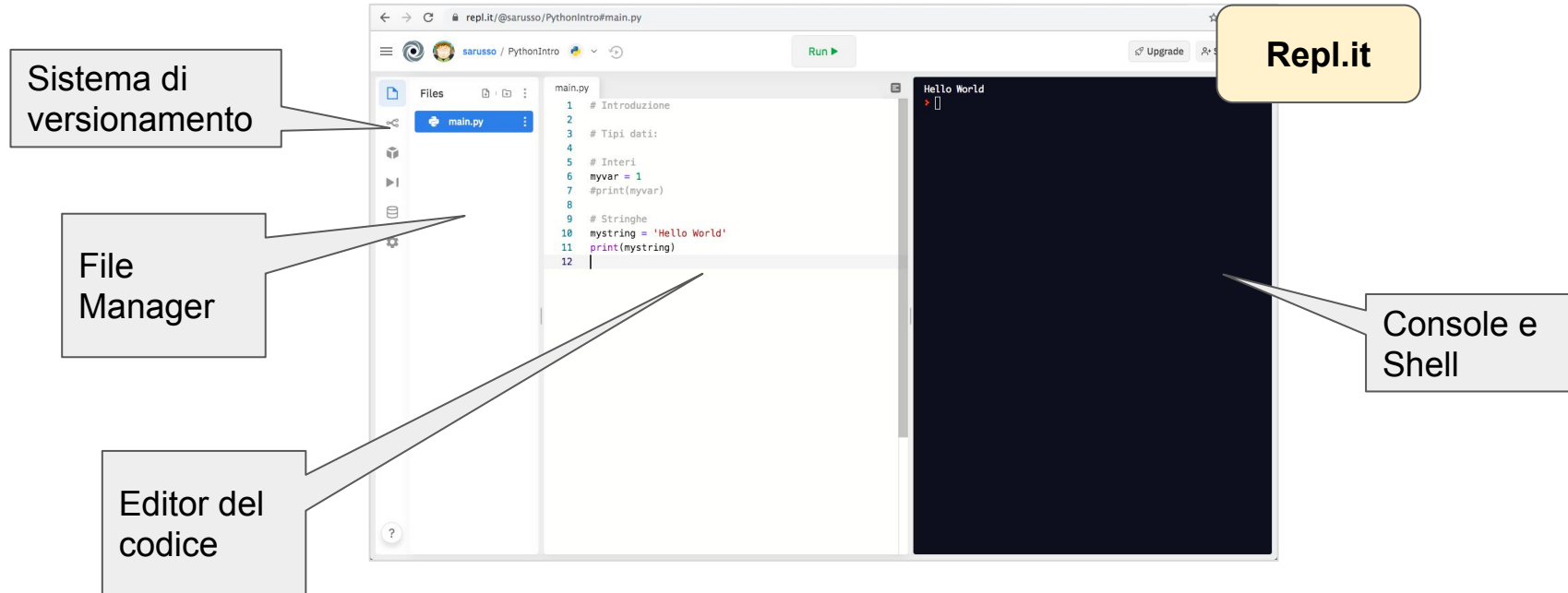
Eclipse

Shell



# Strumenti: l'IDE (Integrated Development Environment)

E' un sistema che integra in modo integrato File Manager, Editor del codice, la Shell, il sistema di versionamento e altre funzionalità come il debugger.



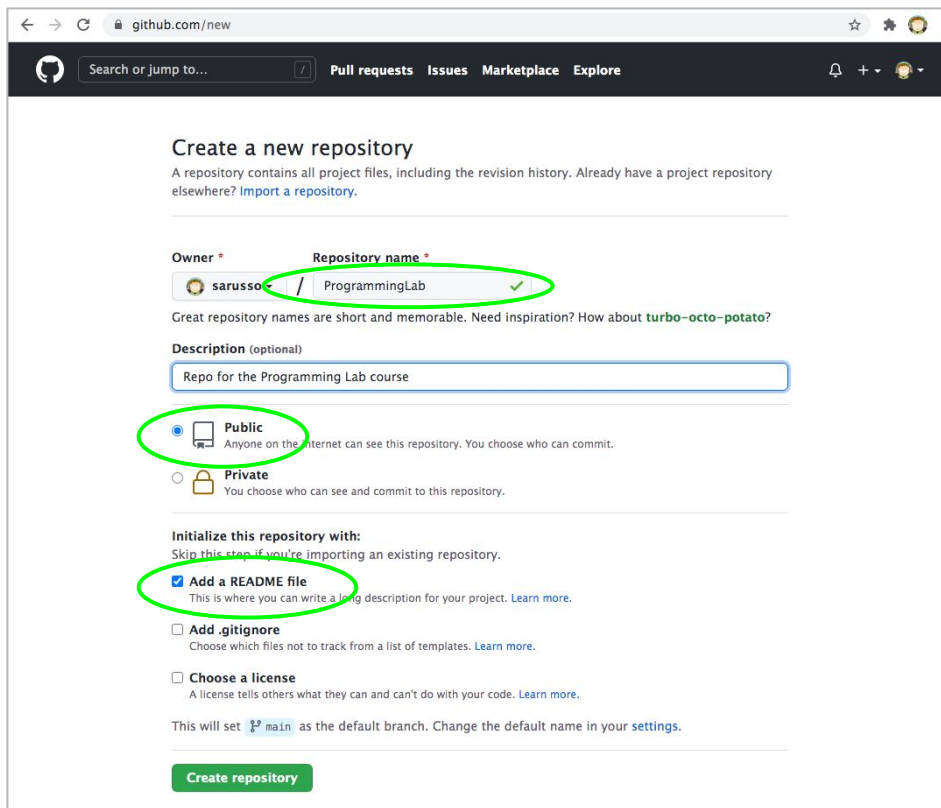
# Setup dell'ambiente

- 1) Registratevi su GitHub se non lo siete già
- 2) Createvi un repository su GitHub chiamato "ProgrammingLab"
- 3) Registratevi su Repl (repl.it) se non lo siete già
- 4) Create un nuovo Repl importando il repository "ProgrammingLab".  
Autorizzate l'app Repl su GitHub se vi viene chiesto di farlo.  
→ settate Python come linguaggio
- 5) Infine impostate repl in modo che usi 4 spazi come indentazione

*Seguono degli screenshot di alcuni passaggi*



# Setup dell'ambiente



The screenshot shows the GitHub 'Create a new repository' page. Several elements are highlighted with green circles:

- The **Repository name** field, which contains 'ProgrammingLab' and has a green checkmark.
- The **Description** field, which contains 'Repo for the Programming Lab course'.
- The **Public** radio button, which is selected.
- The **Add a README file** checkbox, which is checked.

The page includes the following text and elements:

**Create a new repository**  
A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

**Owner \*** sarusso / **Repository name \*** ProgrammingLab ✓

Great repository names are short and memorable. Need inspiration? How about [turbo-octo-potato](#)?

**Description (optional)**  
Repo for the Programming Lab course

☒ **Public**  
Anyone on the Internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

**Initialize this repository with:**  
Skip this step if you're importing an existing repository.

☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more](#).

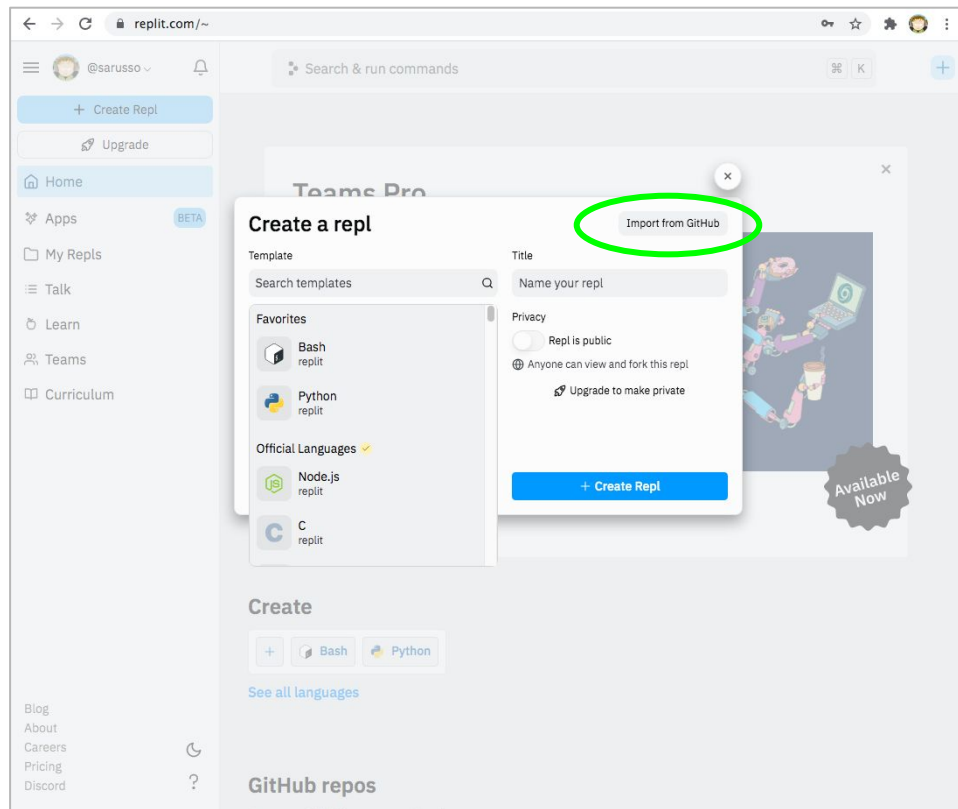
☐ **Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more](#).

☐ **Choose a license**  
A license tells others what they can and can't do with your code. [Learn more](#).

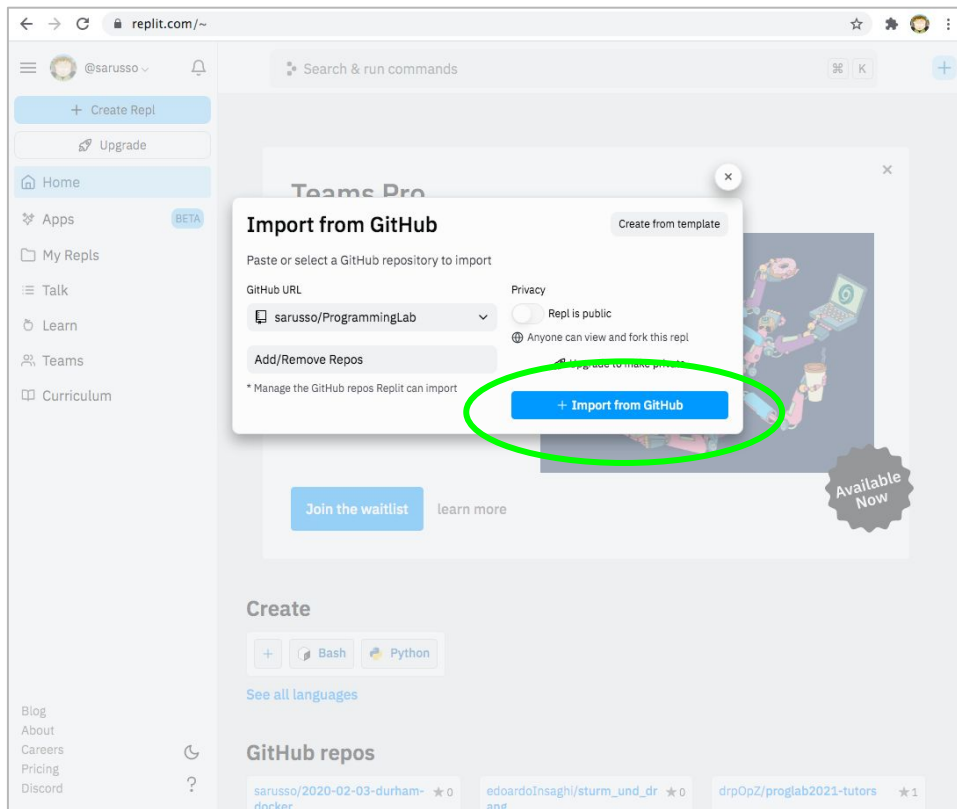
This will set [main](#) as the default branch. Change the default name in your [settings](#).

**Create repository**

# Setup dell'ambiente



# Setup dell'ambiente



# Primi comandi (1)

Creiamo adesso uno script "hello.py" con dentro il contenuto:

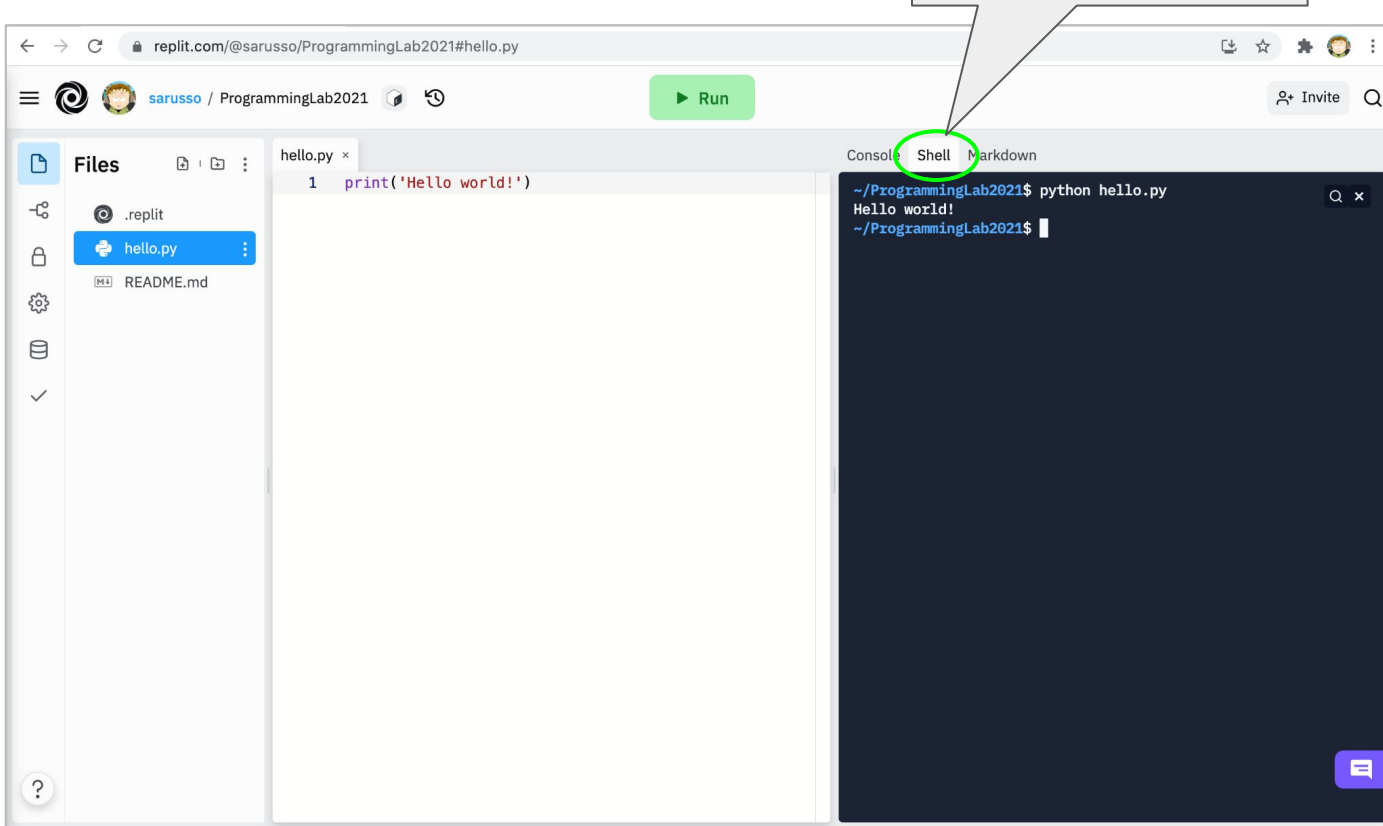
```
print('Hello world!')
```

Poi, eseguiamo lo script

```
$ python hello.py  
Hello world!
```

# Primi comandi (1)

p.s. usate questa tab per eseguire il codice



The screenshot displays the Replit web interface for a project named 'ProgrammingLab2021'. The left sidebar shows the file explorer with files: '.replit', 'hello.py', and 'README.md'. The main editor area shows the code in 'hello.py':

```
1 print('Hello world!')
```

A green 'Run' button is located at the top right of the editor. Below the editor, the console is open with three tabs: 'Console', 'Shell', and 'Markdown'. The 'Shell' tab is selected and highlighted with a green circle. The console output shows the command and its result:

```
~/ProgrammingLab2021$ python hello.py  
Hello world!  
~/ProgrammingLab2021$
```

## Primi comandi (2)

Ora committiamo e pushamo questo script: dal menu in basso a sinistra, clicchiamo su “Git”, poi su “Commit All & Push”.

→ in realtà verranno aggiunti anche due files interni di Repl, solo per il primo commit.

Infine, testate il vostro script su <http://autograding.live> !