

# LABORATORIO DI PROGRAMMAZIONE I

Lezione 1

*Introduzione e strumenti*

Laura Nenzi

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

Modulo del corso: INTRODUZIONE ALLA PROGRAMMAZIONE E LABORATORIO (15 CFU) che comprende Modulo prof. Caravagna (9 CFU) e Laboratorio di programmazione II (3 CFU)

**Docente:** Laura Nenzi (io)

**Sito Web (repository):** [https://github.com/lauranenzi/LabProgrammazione\\_I\\_2026](https://github.com/lauranenzi/LabProgrammazione_I_2026)

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

**Tutors:**

- Nicholas Pearson: [nicholasandrea.pearson@phd.units.it](mailto:nicholasandrea.pearson@phd.units.it)
- Michele Alessi: [michele.alessi@phd.units.it](mailto:michele.alessi@phd.units.it)
- Ermes Aviano: [ermes.aviano@phd.units.it](mailto:ermes.aviano@phd.units.it)

# Chi sono

Laura Nenzi

- Professore Associato in Ingegneria Informatica al Dipartimento di Ingegneria ed Architettura
- Laurea in Matematica, Phd in Computer Science
- Ricerca in Metodi Formali e Machine Learning

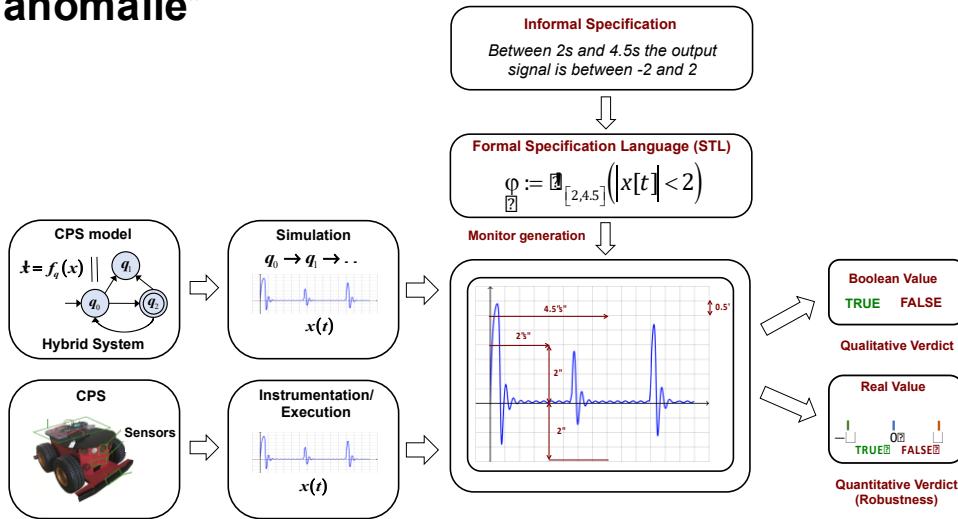


Ufficio c3 2.55

Mail: [lnenzi@units.it](mailto:lnenzi@units.it)

# Cosa Faccio: Runtime Verification

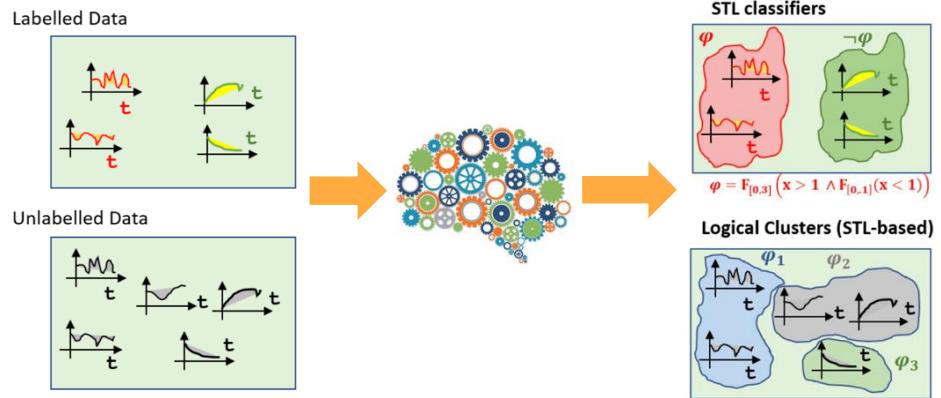
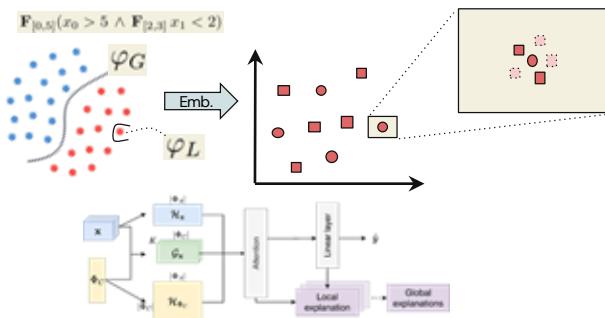
“framework per specificare **formule di logica temporali e monitorarle**, in particolare progettando algoritmi e identificando istanze di dati che non soddisfano le formule come anomalie”



$$\text{Crowdedness} > K \rightarrow \diamond_{d \leq 1} \text{Crowdedness} < K$$

# Cosa Faccio: Explainable AI

“impariamo formule logiche da dati di serie temporali che catturano i comportamenti essenziali di sistemi complessi e incerti”



“li integriamo inoltre in un framework **neuro-simbolico** per guidare gli output dei modelli generativi, garantendo che rispettino specifiche logiche e forniscano una classificazione interpretabile”

# Argomenti del corso

Questo modulo di laboratorio si concentra interamente su Python. I primo modulo è propedeutico a questo quindi viene assunto che lo studente sia già familiare con i costrutti e gli operatori di base di un linguaggio di programmazione.

- 1) Intro del corso e strumenti da "laboratorio": la shell, Git, gli IDE, etc.
- 2) Intro a Python: tipi dati, costrutti, funzioni, moduli, interazione con dati, be pythonic.
- 3) Programmazione ad Oggetti
- 4) Le eccezioni, il flusso try-except ed il controllo degli input e sanitizzazione
- 5) (Jupyter Notebook)

# Obiettivo del Corso

- Capacità di scrivere un programma completo in Python.
- Saper scegliere in generale il modello di programmazione opportuno per ciascun problema
- Imparare delle “buone pratiche” di programmazione

# Consiglio: Limitate l'uso dell'IA

- **L'intelligenza artificiale (IA) scrive codice, ma non capisce il vostro problema.** Senza esercizio non sviluppate il “debug mentale”, cioè la capacità di leggere un algoritmo e chiedervi: ha senso? copre i casi limite? cosa succede se l'input è sporco?
- **Programmare non è scrivere righe, è pensare in modo strutturato.** Bisogna decidere cosa fare prima di decidere come farlo. Gli esercizi allenano la decomposizione del problema, potete vederla come una ginnastica cognitiva.
- Per cui vi consiglio vivamente di provare a fare gli esercizi usando l'IA in maniera limitata.

# Informazioni Utili

- Sito web della comunità italiana di Python: <https://www.python.it/>
- Tutorial ufficiale (scegliete la versione di Python che avete installato):  
<https://docs.python.org/it/3/tutorial/index.html>
- A. Downey J. Elkner C. Meyers, How to Think Like a Computer Scientist,  
Learning with Python.
  - <https://allendowney.github.io/ThinkPython/>
  - traduzione italiana (versione pdf) di Andrea Zanella:  
[https://github.com/AllenDowney/ThinkPythonItalian/blob/master/thinkpython\\_italian.pdf](https://github.com/AllenDowney/ThinkPythonItalian/blob/master/thinkpython_italian.pdf)
  - traduzione italiana (versione web) di Alessandro Pocaterra:  
<https://www.python.it/doc/Howtothink/Howtothink-html-it/index.htm>

# Modalità d'esame

- Esame scritto al computer in un'aula d'informatica dell'università. Le aule di informatica dove farete l'esame avranno computer tutti forniti direttamente con Visual Studio Code.
- La connessione internet non sarà presente ma potrete accedere alla pagina del corso su Moodle, [link](#)
- Il giorno dell'esame troverete sulla pagine del corso su Moodle il testo dell'esame e dovrete caricare sulla stessa piattaforma l'elaborato finale. È essenziale quindi che chiunque voglia fare l'esame sia iscritto al corso del laboratorio su Moodle.

# Modalità d'esame

- Possono sostenere l'esame solo gli studenti che hanno passato il modulo del prof. Caravagna.
- L'orale è facoltativo se lo scritto è  $\geq 18$ , obbligatorio se il voto è 16,17 ed obbligatorio a discrezione del docente se vengono rilevate delle irregolarità
- La prova orale si sostiene solo nell'appello in cui si svolge lo scritto, ed in date e orari concordati col docente.
- Chi verrà visto copiare sarà escluso dalle due prove successive e se reiterà il comportamento potrebbe avere conseguenze anche più gravi

# Organizzazione delle ore

Ogni lezione è di 3 ore. Non sempre ma in linea di massima faremo:

- **Prima ora:** pratica (con me e gli assistenti)
- **Seconda/Terza ora:** Teoria

**Oggi, alla fine della lezione dovrete tutti sapere:**

- 1) Come si esegue uno script Python dentro a VSC
- 2) Come si fa un commit da VSC su GitHub

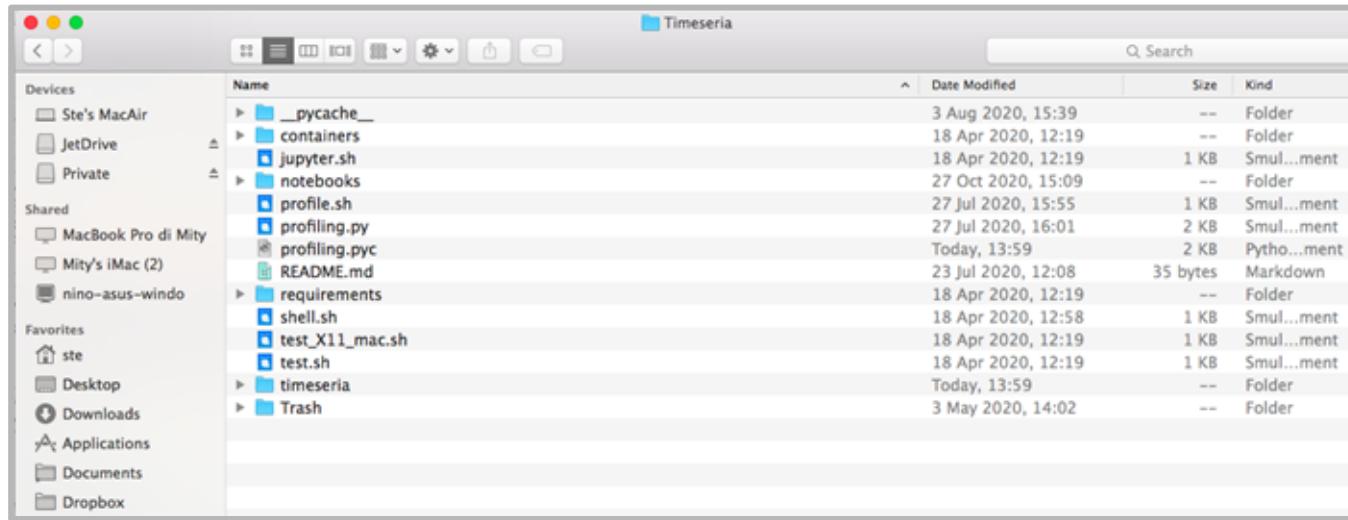
# Iniziamo! ...con gli strumenti

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

- *Abraham Lincoln*

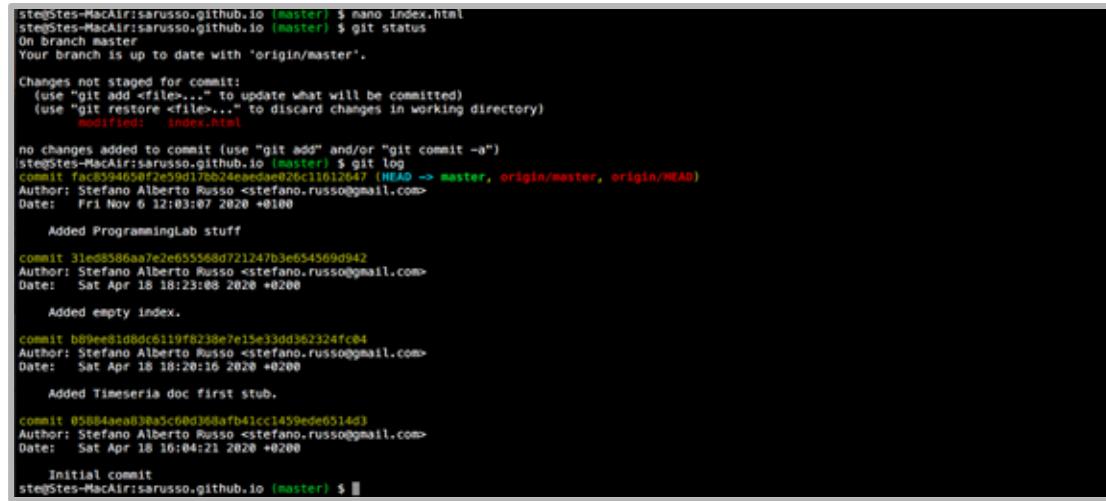
# Strumenti: il File Manager

È quello con cui si vedono le cartelle e i files del computer. Può essere configurato per vedere anche i file nascosti e le estensioni! Esempi sono Windows Explorer (Esplora file) su Windows, il Finder su MacOS e Nautilus o Dolphin sui sistemi Linux.



# Strumenti: la Shell (anche Terminale / Console)

È un programma che permette di interagire con il computer in via testuale attraverso una Command-Line Interface (CLI), senza bottoni che automatizzano le cose. È come si fanno le cose sul serio senza usare un ambiente preconfezionato. Su sistemi Unix in genere è “BASH”. Nei sistemi Windows sono il Prompt dei Comandi (CMD) e la PowerShell.



```
ste@Stes-MacAir:sarussuo.github.io (master) $ man index.html
ste@Stes-MacAir:sarussuo.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:sarussuo.github.io (master) $ git log
commit fac8594650f2e59d1760c4eeedae026c11612647 [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 b69ee81d8dc6119f8238e7e15e33dd362324fc04
Author: Stefano Alberto Russo <stefano.russo@gmail.com>
Date:   Sat Apr 18 18:20:16 2020 +0200

    Added Timeseria doc first stub.

commit 05884aea930a9c00d368aafbd1c1459ede6514d3
Author: Stefano Alberto Russo <stefano.russo@gmail.com>
Date:   Sat Apr 18 16:04:21 2020 +0200

    Initial commit

ste@Stes-MacAir:sarussuo.github.io (master) $
```

# Comandi shell - Navigazione di base

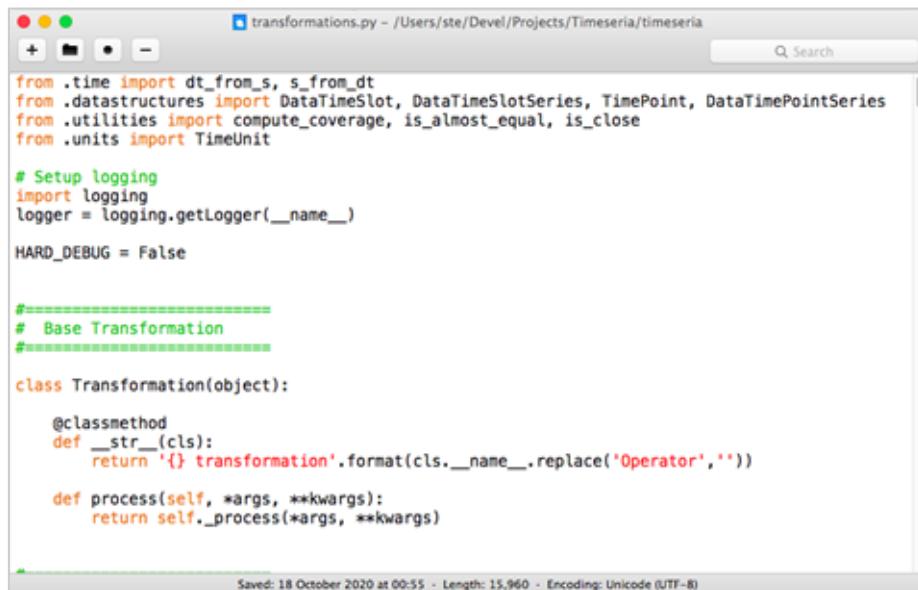
- **ls**: restituisce l'elenco dei file e delle cartelle nella directory corrente
- **cd dirName**: comando utilizzato per spostarsi da una directory ad un'altra  
**cd ..**: comando utilizzato per tornare alla directory superiore  
**cd -**: comando utilizzato per tornare alla directory precedente  
**cd**: comando utilizzato per tornare alla directory madre o root
- **pwd**: restituisce la current working directory  
**man commandName**: restituisce il manuale per il comando indicato

# Comandi shell - File e Directory Manipulation

- **mkdir dirName**: crea una sub-directory con il nome indicato nella cwd
- **cp sourcePath destPath**: copia un file o una directory dal path sorgente a quello destinazione  
**mv sourcePath destPath**: sposta un file o una directory dal path sorgente a quello destinazione
- **rm filePath**: elimina il file indicato  
**rm -r dirPath**: elimina la directory indicata

# Strumenti: l'Editor del codice

È un programma con cui scrivere il codice. A differenza di un programma di videoscrittura (come Microsoft Word), che formatta il testo con font, dimensioni e colori, un editor di testo è progettato principalmente per gestire testo semplice, senza elementi di formattazione avanzati. Gli editor del codice hanno colori specifici per aiutare a programmare.



The screenshot shows a Mac OS X-style code editor window titled "transformations.py - /Users/ste/Devel/Projects/Timeseria/timeseria". The code is written in Python and defines a class named Transformation. The code includes imports for time, datastructures, utilities, and units modules, as well as a logging setup. It also defines a constant HARD\_DEBUG and a class Transformation with methods \_\_str\_\_ and process. The code uses color-coded syntax highlighting where keywords like from, import, def, and class are in blue, while identifiers and strings are in black. Comments are in green, and strings are in red. The status bar at the bottom indicates the file was saved on October 18, 2020, at 00:55, has a length of 15,960 bytes, and is encoded in Unicode (UTF-8).

```
from .time import dt_from_s, s_from_dt
from .datastructures import DataTimeSlot, DataTimeSlotSeries, TimePoint, DataTimePointSeries
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)

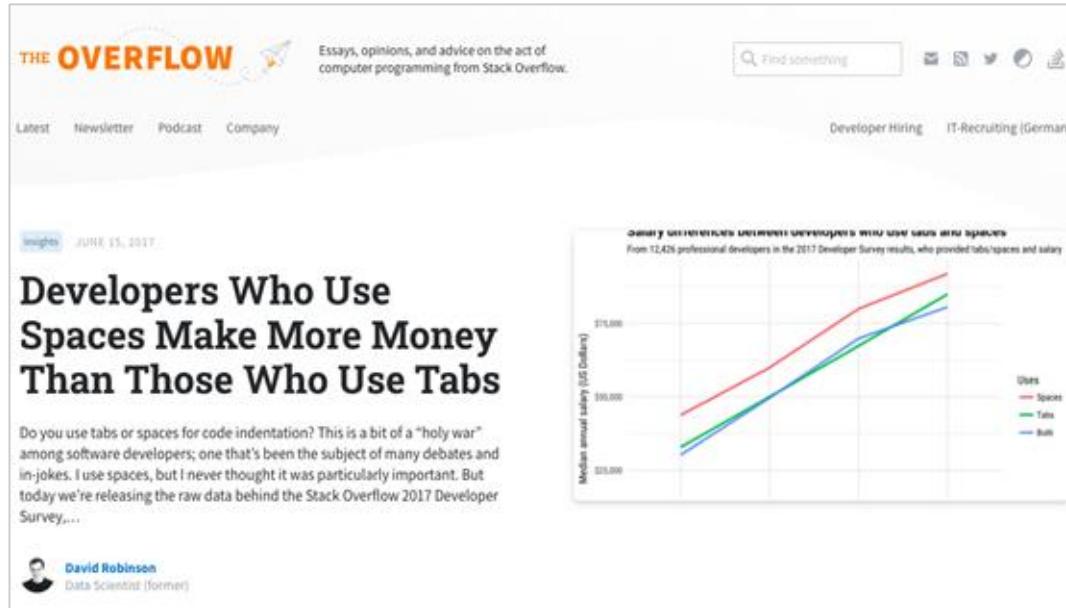

```

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

# Strumenti: l'Editor del codice

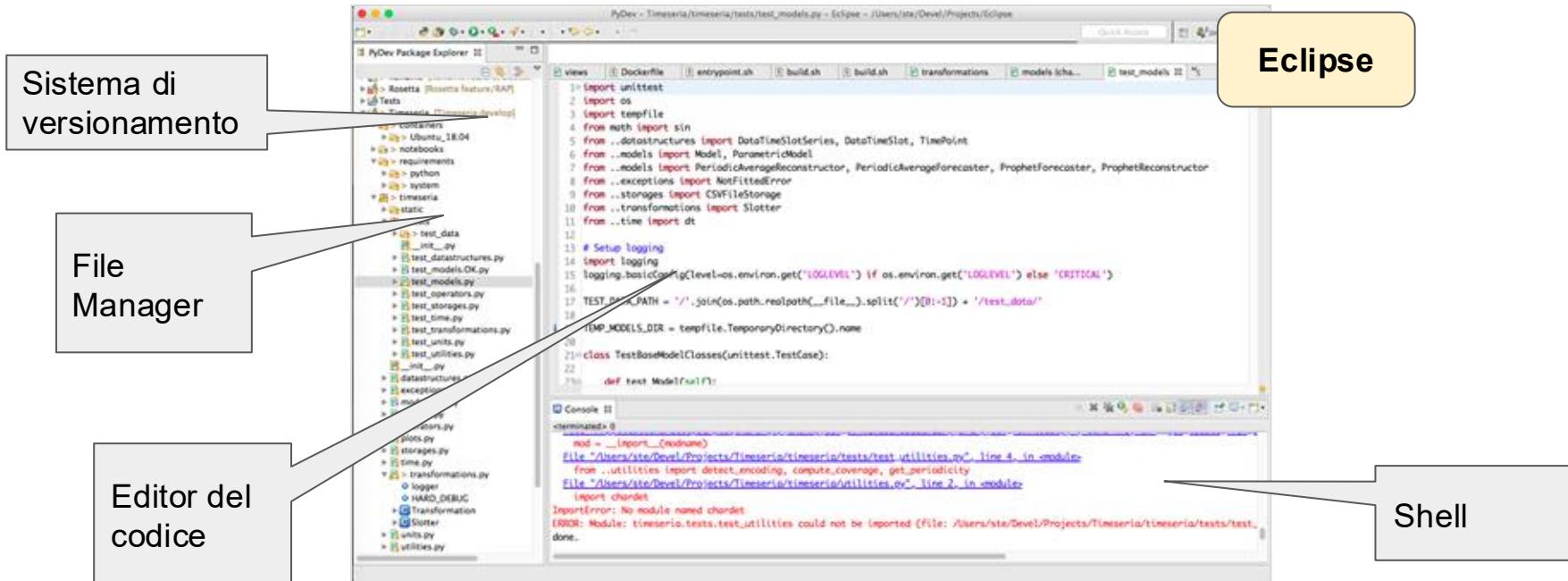
Nota: è tassativo impostare l'editor a usare gli spazi e non i tab.

Python segue la convenzione PEP 8 (le linee guida ufficiali di stile) che raccomanda **4 spazi per livello di indentazione**



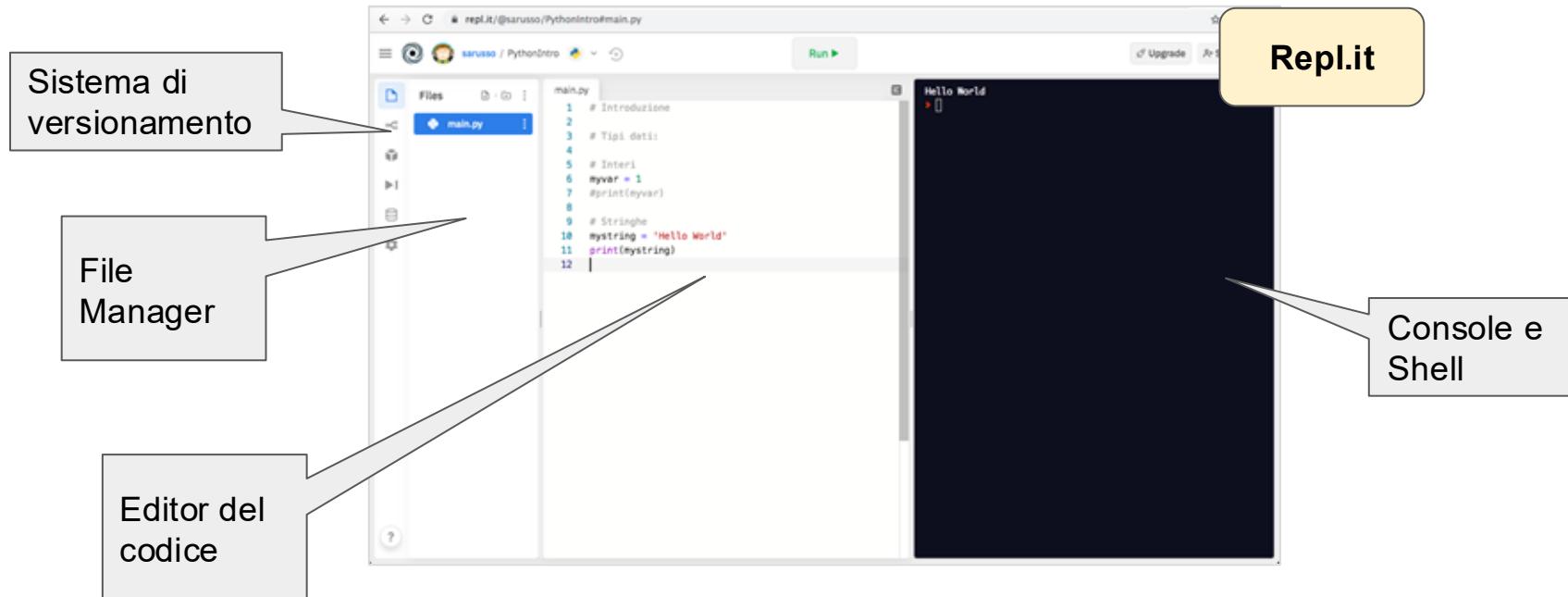
# Strumenti: l'IDE (Integrated Development Environment)

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



# Strumenti: l'IDE (Integrated Development Environment)

- Il tutto preconfigurato per uno o più linguaggi di programmazione specifici (ad esempio, PyCharm per Python, IntelliJ IDEA per Java).



# Visual Studio Code (VS Code)

- È un editor di codice sorgente sviluppato da Microsoft. È gratuito, open-source e multiplattforma (disponibile per Windows, macOS e Linux)

The screenshot shows the Visual Studio Code interface. The code editor displays a TypeScript file named [slug].tsx. The file contains code for a Next.js project, specifically a dynamic route page. The GitHub Copilot feature is active, providing suggestions for rendering project images. The Explorer sidebar shows the project structure, including components, pages, and a projects folder. The bottom status bar indicates the current file is [slug].tsx, and the code is written in TypeScript.

```
const ProjectPage: React.FC = () => {
  const router = useRouter();
  const { slug } = router.query;

  const project = projects.find(p => p.slug === slug);

  if (!project) {
    return <div>Project not found</div>;
  }

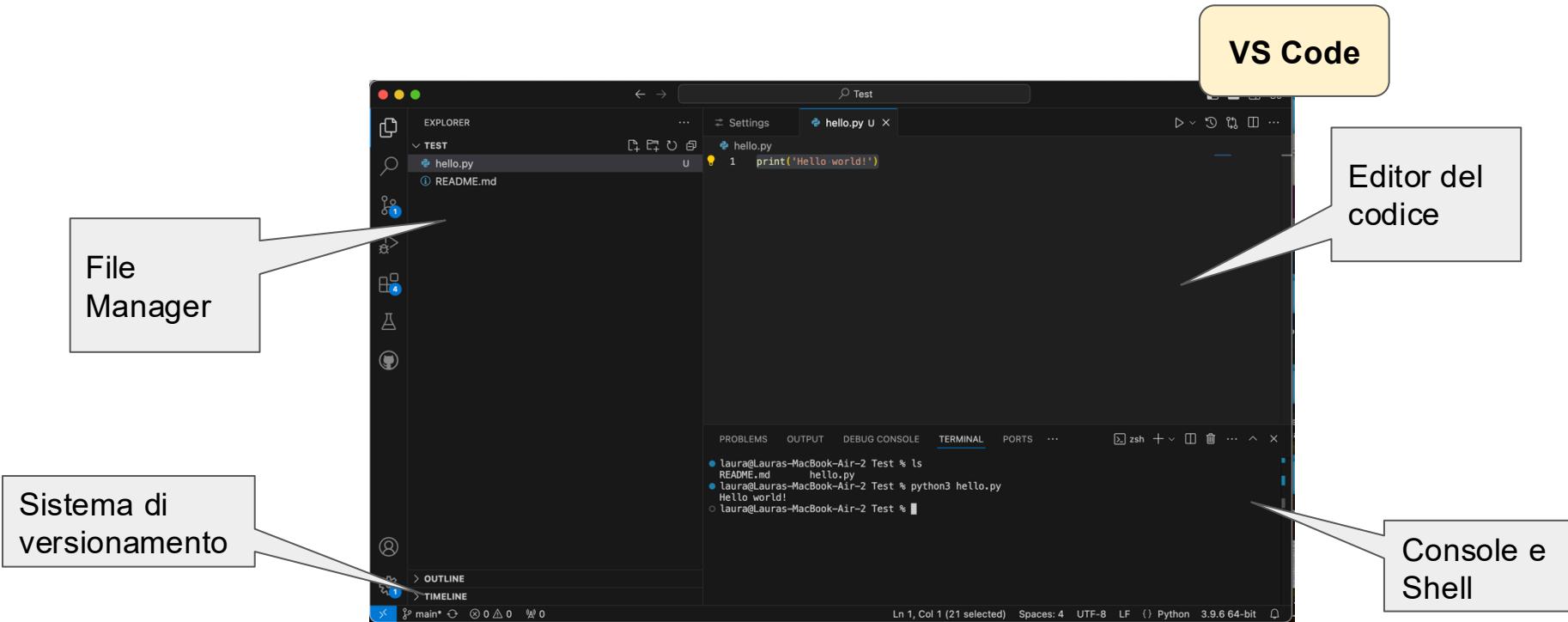
  return (
    <div className="container mx-auto px-4 py-8">
      <Link href="/" className="text-blue-600 hover:underline mb-4 inline-block">
        <Image alt="Back to projects" />
        Back to projects
      </Link>
      <h1 className="text-3xl font-bold mb-4">{project.title}</h1>
      <p className="text-lg mb-6">{project.description}</p>
      <div className="flex flex-col gap-4">
        <Image alt="Project image" key={index} src={image} alt={image.alt} />
        Render the project image here
      </div>
    </div>
  );
}

export default ProjectPage;
```

# VS Code non è un IDE ma...

- VS Code è molto flessibile grazie al suo sistema di **estensioni**.
- Questo significa che può essere configurato per comportarsi quasi come un IDE completo aggiungendo estensioni, e.g. per il debugging, il controllo della versione, l'integrazione di database e altro.
- Gli **IDE** di solito offrono tutte queste funzionalità già pronte, senza necessità di aggiungere estensioni.
- Quindi, pur non essendo un IDE completo, VS Code può essere configurato per offrire un'esperienza simile

# VS Code usato come IDE



# VS Code per Python

Visual Studio Code Docs Updates Blog API Extensions FAQ GitHub Copilot Download

[Version 1.95](#) is now available! Read about the new features and fixes from October.

## Quick Start Guide for Python in VS Code Edit

The Python extension makes Visual Studio Code an excellent Python editor, works on any operating system, and is usable with a variety of Python interpreters.

Get started by installing:

- [VS Code](#)
- [A Python Interpreter \(any actively supported Python version\)](#)
- [Python extension](#) from the VS Code Marketplace

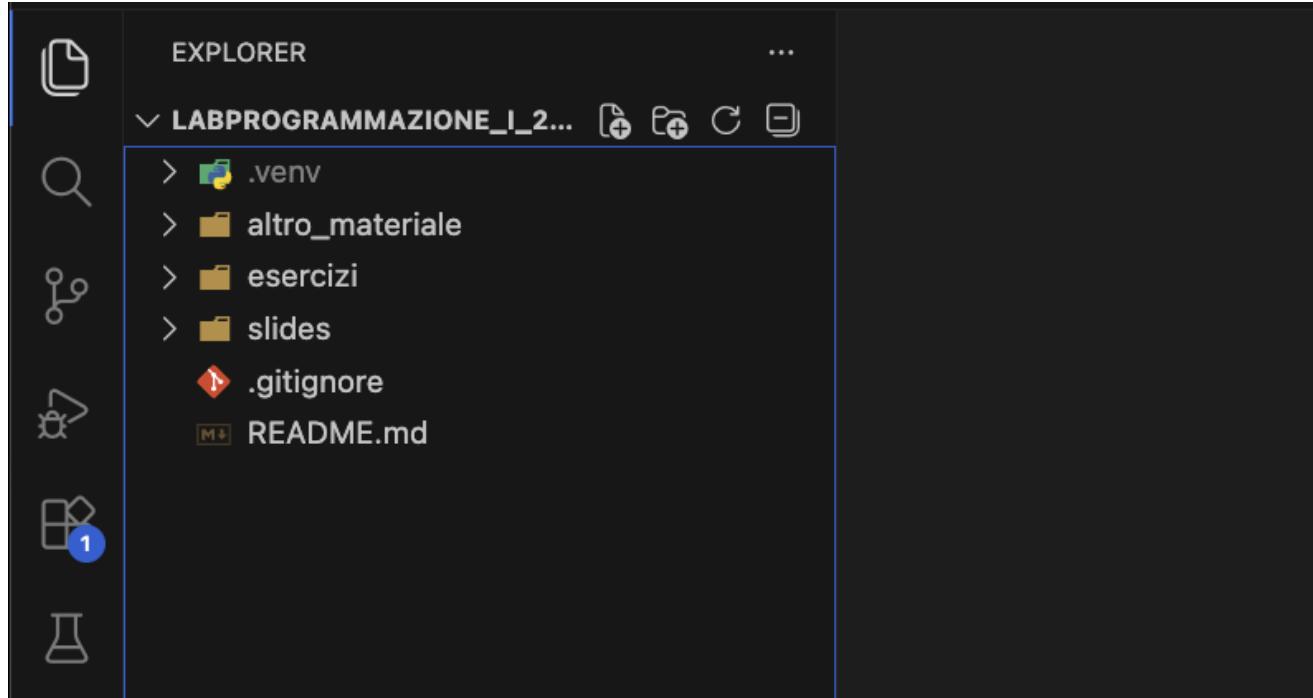
The screenshot shows the VS Code Marketplace interface. A search bar at the top has 'Extension: Python' typed into it. Below the search bar, the Python extension card is displayed, featuring the Python logo, the text 'v2023.22.1', the developer 'Microsoft', and a star rating of 5 stars. Below the card, there's a link to 'Disable', 'Uninstall', and 'Switch to Pre-Release Version'. A note says 'This extension is enabled globally.' At the bottom of the card, there are tabs for 'DETAILS', 'FEATURE CONTRIBUTIONS', 'CHANGELOG', and 'EXTENSION PACK'. To the right of the card, there's a sidebar titled 'IN THIS ARTICLE' with links to various Python-related topics like 'How to create and open a Python project or file', 'UI tour', 'Code Actions', etc. At the very bottom of the screenshot, a footer message reads: 'To further customize VS Code for Python, you can leverage the [Python profile template](#), automatically'.

# Ambiente Virtuale

- Un ambiente virtuale è uno spazio isolato che contiene:
  - una versione specifica di Python
  - le librerie installate per un progetto
- Ogni progetto può avere il suo ambiente indipendente.
- Progetti diversi possono richiedere:
  - versioni diverse di Python
  - versioni diverse delle stesse librerie
- Un ambiente virtuale permette di lavorare su un progetto senza modificare il resto del sistema.

# Ambiente Virtuale: venv

- È lo strumento integrato in Python per creare ambienti virtuali.



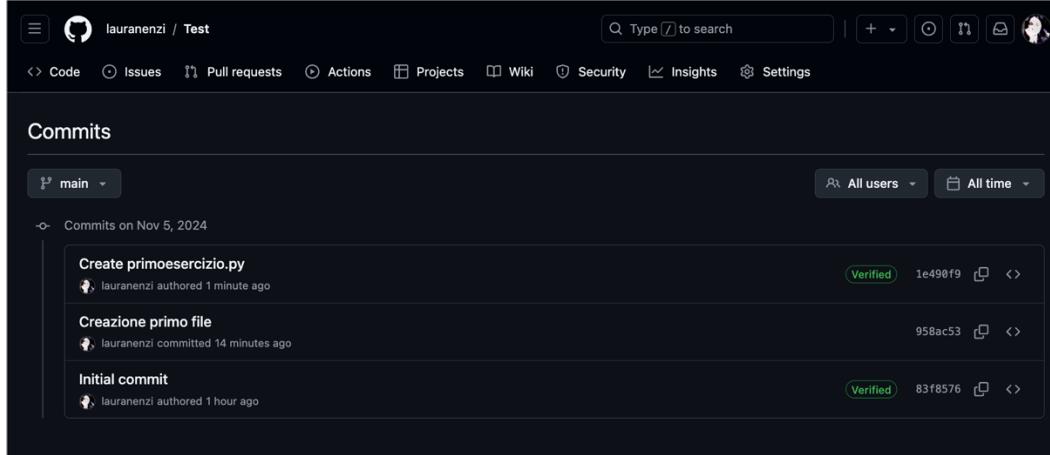


# Ambiente Virtuale: conda

- Conda è un programma per creare ambienti virtuali
- Un gestore di pacchetti e ambienti open source che installa, aggiorna e rimuove pacchetti con le relative dipendenze in ambienti isolati, consentendo più versioni dello stesso software senza conflitti.
- Miniconda è una distribuzione minimale di Anaconda che contiene:
  - il gestore di ambienti (conda)
  - una versione base di Python
  - il minimo indispensabile per iniziare

# Il sistema di versionamento (Git)

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



The screenshot shows a GitHub repository interface for a user named 'lauranenzi'. The repository name is 'Test'. The main navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A search bar at the top right allows users to search for specific commits or files. Below the search bar, there are filters for 'All users' and 'All time'. The main content area is titled 'Commits' and displays the commit history for the 'main' branch. The commits listed are:

- Create primoesercizio.py** (Verified) - authored by lauranenzi 1 minute ago. Commit hash: 1e490f9
- Creazione primo file** - committed by lauranenzi 14 minutes ago. Commit hash: 958ac53
- Initial commit** (Verified) - authored by lauranenzi 1 hour ago. Commit hash: 83f8576

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)

# Il sistema di versionamento (Git)

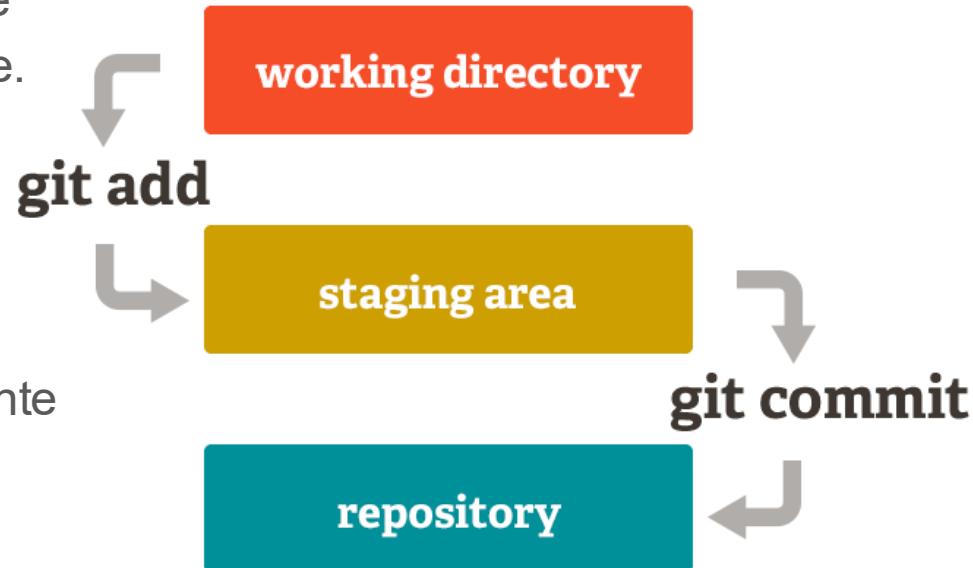
- Ti permette di tenere traccia dello stato di una cartella (cioé dei file al suo interno) in vari istanti del tempo
- Ti consente di sincronizzare più copie della cartella, che possono essere su uno stesso computer o su più computer
- È un programma, esattamente come word, adobe reader o steam.
- Oltre alla versione stand-alone, numerosi software di sviluppo permettono di "integrarlo" al loro interno (tramite dei plug-in, ad esempio), permettendo di utilizzarlo direttamente dalla loro interfaccia.

# Git: Repository (o repo)

- È la cartella che utilizza il sistema di versionamento git.
- Al suo interno si trova la sottocartella .git, in cui Git tiene i file per il suo funzionamento (è una cartella nascosta, l'interfaccia potrebbe non mostrartela).
- Se ti trovi in un repository puoi utilizzare i comandi di git per eseguire operazioni

# Git: Staging e Commit

- **Staging** è l'area temporanea dove metti le modifiche che vuoi salvare. Ti permette di selezionare esattamente quali modifiche includere nel prossimo commit.
- **Commit** è il salvataggio permanente delle modifiche nel repository. Ogni commit è un punto di salvataggio che crea una cronologia del progetto.



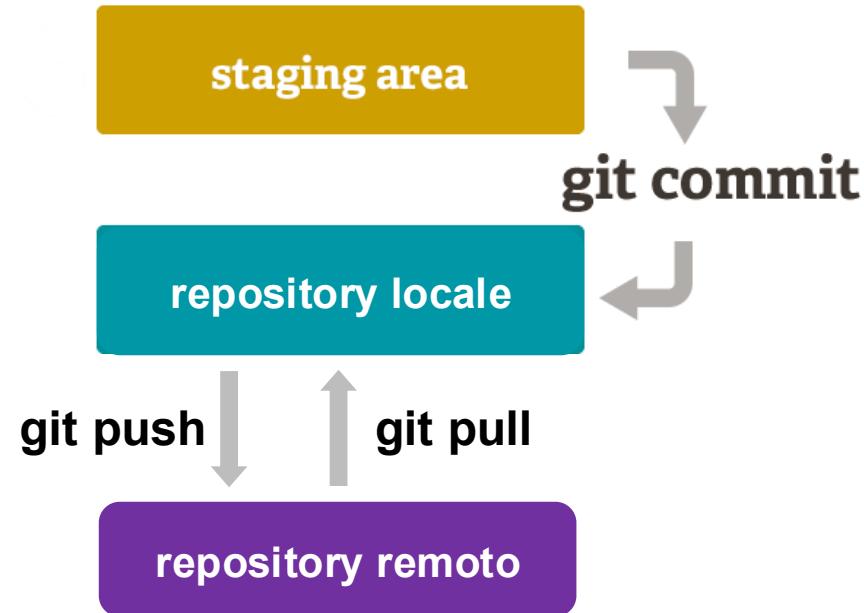
# Git: Pull e Push

Come si sincronizzano più repository. Ci sarà un repository remoto e dei repository locali (quelli sul tuo computer).

- **Push** invia i commit locali al repository remoto (es. GitHub).  
Permette di condividere il proprio lavoro con altri

- **Pull** scarica e integra nel repository locale le modifiche presenti nel repository remoto.

Serve per sincronizzarsi con il lavoro degli altri.



# GitHub

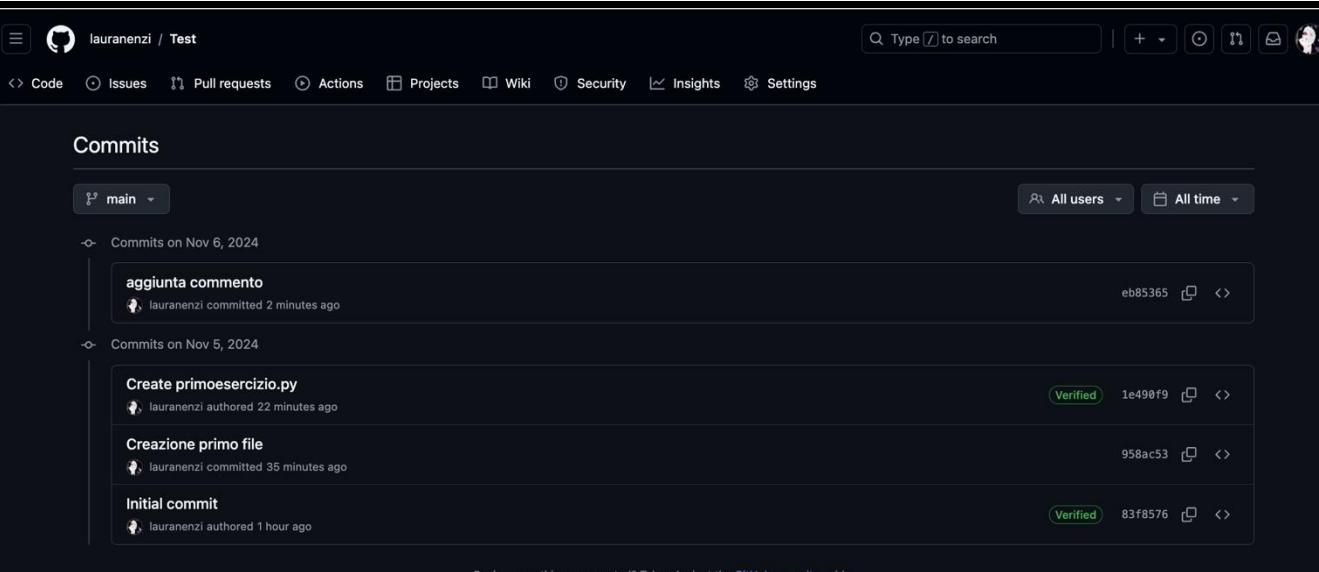


- GitHub è, in superficie, un sito web dove metti il codice.
- In profondità, è un gigantesco laboratorio collaborativo basato su **Git**
- A git aggiunge:
  - repository condivisi
  - gestione collaborativa
  - issue per discutere problemi o idee
  - hosting di siti statici
  - ambienti cloud tipo Codespaces

# GitHub



- Un repository non è solo una cartella con file. È una struttura con memoria. Ogni modifica è tracciata, attribuita, comparabile.
- Ogni commit è identificato da un hash che dipende dal contenuto.



A screenshot of a GitHub repository's commits page. The repository is named "lauranenzi / Test". The main navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A search bar at the top right contains the placeholder "Type / to search". Below the navigation, there are dropdown menus for "All users" and "All time". The main content area is titled "Commits" and shows two commit lists. The first list is for Nov 6, 2024, containing one commit: "aggiunta commento" by lauranenzi, committed 2 minutes ago, with a hash of eb85365. The second list is for Nov 5, 2024, containing three commits: "Create primoesercizio.py" by lauranenzi, authored 22 minutes ago and verified (hash 1e490f9); "Creazione primo file" by lauranenzi, committed 35 minutes ago (hash 958ac53); and "Initial commit" by lauranenzi, authored 1 hour ago and verified (hash 83f8576). At the bottom of the page, a note reads: "Seeing something unexpected? Take a look at the GitHub commits guide."

# Pseudo Codice (parentesi)

Lo pseudo-codice sarà il vostro migliore amico, ancora prima di Python.

Fare pseudo-codice vuol dire scrivere, in linguaggio naturale (Italiano/Inglese), quello che dovrebbe fare il programma, con un minimo di sintassi.

Non ci si focalizza sui dettagli nello pseudo-codice!

Ovvero, non ci si focalizza sul **come**, ma sul **cosa** fare.

# Pseudo Codice (parentesi)

Esempio: trova i numeri in una lista minori di 5 e stampali

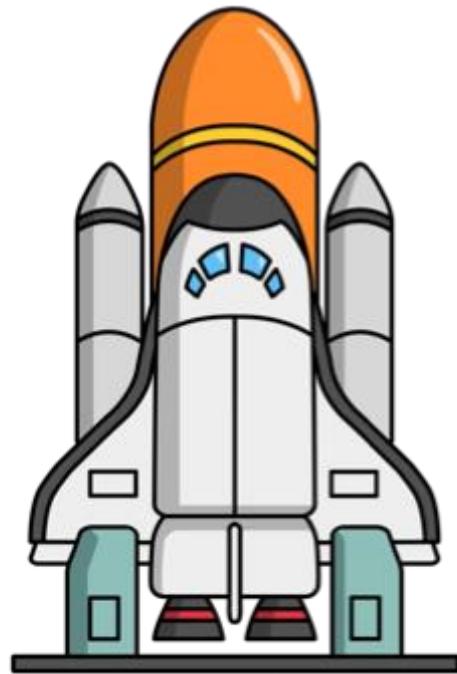
```
data una lista di numeri;  
  
per ogni elemento della lista:  
    se l'elemento è minore di 5:  
        stampa l'elemento
```

# Python: un codice minimale

Esempio: trova i numeri in una lista minori di 5 e stampali

```
number_list = [13,12,34,4,51,8,27,18]

for item in number_list:
    if item < 5:
        print(item)
```



Cominciamo

# Step generali da seguire

I seguenti step sono comuni ad entrambe le opzioni, cambia solamente il modo in cui vengono eseguiti.

1. Scaricare miniconda <https://www.anaconda.com/download/success>
2. Aprire VSC ed installare l'estensione per Python
3. Creare un virtual environment su VSC tramite linea di comando o interfaccia grafica
4. Registrarsi su github e Collegare il proprio account su vscode
5. Creare un repository e clonarlo sul vostro computer

# Come capire se il mio macOS ha chip INTEL?

- Aprire menù a tendina nella parte superiore dello schermo avvicinando il puntatore;
- Cliccare sull'icona della mela in alto a sinistra;
- Cliccare sulla voce "Informazioni su questo Mac";
- Verificare nella voce "Chip": se il chip è "Apple" seguire 1a), se, invece, il chip è "Intel" seguire 1b);

# 1a) Installare miniconda su macOS con chip Apple

- Su <https://www.anaconda.com/download/success> scegliere il Graphical Installer e seguirne le istruzioni

# 1b) Installare miniconda su macOS con chip INTEL

- Le seguenti istruzioni sono prese dalla seguente pagina web ([link](#)):
  - Aprire una nuova finestra del terminale
  - Copiare, incollare ed eseguire il seguente comando: `mkdir -p ~/miniconda3`
  - Copiare, incollare ed eseguire il seguente comando: `curl https://repo.anaconda.com/miniconda/Miniconda3-py310_25.5.1-0-MacOSX-x86_64.sh -o ~/miniconda3/miniconda.sh`
  - Copiare, incollare ed eseguire il seguente comando: `bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3`
  - Copiare, incollare ed eseguire il seguente comando: `rm ~/miniconda3/miniconda.sh`
  - Dopo l'installazione, chiudi e riapri il tuo terminale o ricaricalo eseguendo il seguente comando: `source ~/miniconda3/bin/activate`
  - Dopo, inizializza conda su tutte le shell disponibili eseguendo il comando seguente: `conda init --all`

# 1b) Installare miniconda su macOS con chip INTEL

- Se tutto funziona correttamente sul terminale apparirà la voce "base" accanto al proprio nome utente come nel seguente esempio:

```
(base) ermesaviano@
```

- Per il controllo finale sulla correttezza dell'installazione digitate ed eseguite da terminale il seguente comando "conda --version";
- Dovreste ottenere in output la versione installata di conda (in questo caso particolare la 25.5.1 cioè l'ultima versione disponibile di miniconda per il vostro pc –macOS con chip INTEL–)

# 1c) Installare miniconda su Windows

- Installare miniconda seguendo le istruzioni di questo link

<https://www.anaconda.com/docs/getting-started/miniconda/install#windows-powershell>

- Dovrete eseguire le tre righe di codice dal terminale powershell su VSC

Una volta che le tre righe di codice (eseguite una ad una) sono state completate, possiamo proseguire creando il primo virtual environment.

# Setup dell'ambiente

- Inizializziamo conda da terminale:

Su macOS e Linux: `conda init`

Su Windows:

- Apri dal menu Start: **Anaconda PowerShell Prompt**
- Dentro questo terminale scrivi `conda init powershell`

- Chiudere e riaprite il terminale e controllare

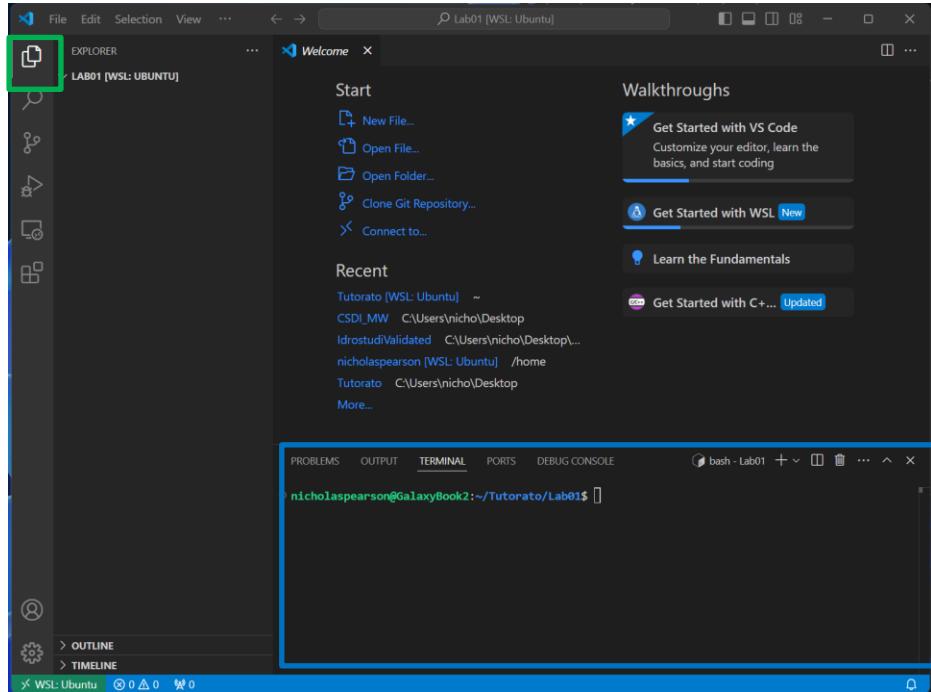
`conda --version`

# Setup dell'ambiente

2) Aprire VS Code dopo averlo installato (dovreste averlo già se no vedete qui:

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

- Aprire un terminale cliccando su:  
**Terminal >> New Terminal**
- Da qui potete scegliere se navigare l'ambiente utilizzando **l'interfaccia grafica** di VS Code oppure utilizzando **comandi shell nel terminale**



# Setup dell'ambiente - Windows

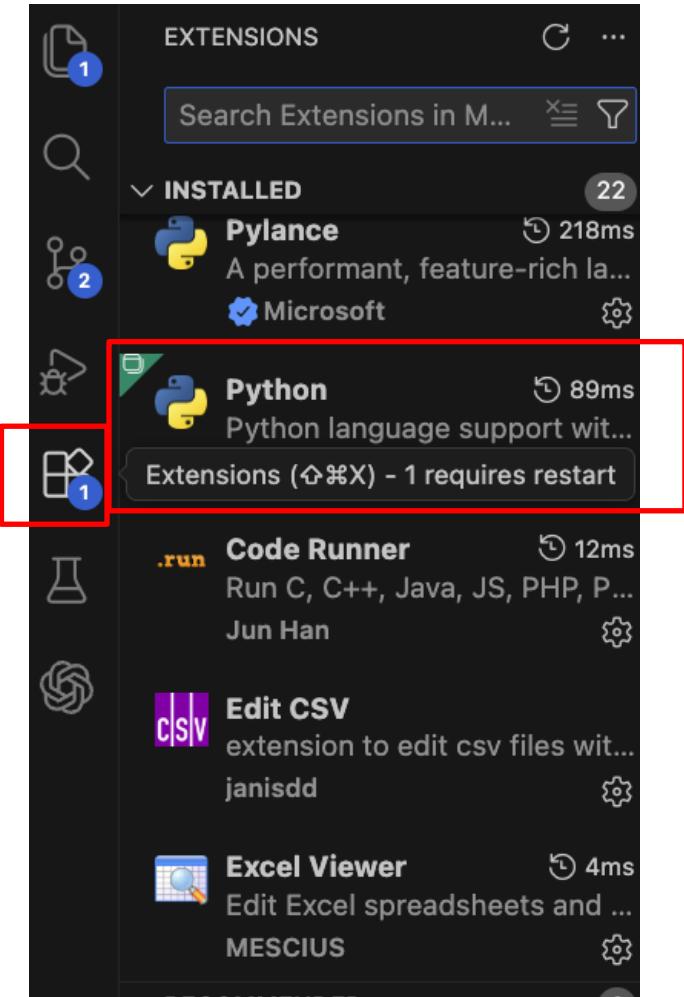
È possibile ci sia un messaggio di errore sul terminale in vscode.

E' sufficiente eseguire questa linea di codice nel vostro terminale **powershell** per risolvere.

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

# Setup dell'ambiente

3) Scaricare l'estensione di Python per VSC



# Setup dell'ambiente

4) Create un “virtual environment”, i.e. un ambiente virtuale.

Una volta attivato quell’ambiente, tutti i pacchetti che installi successivamente sono isolati dagli altri ambienti, incluso l’ambiente globale dell’interprete, riducendo molte complicazioni che possono sorgere da conflitti tra versioni dei pacchetti.

# Setup dell'ambiente

4a) Settare un interprete python ed un ambiente virtuale con miniconda da terminale.

- Verificare che conda sia presente: `conda --version`
- Creare un "virtual environment" `conda create -n nome_ambiente python=3.12`
- Attivare l'ambiente `conda activate nome_ambiente`
- Per disattivare `conda deactivate`

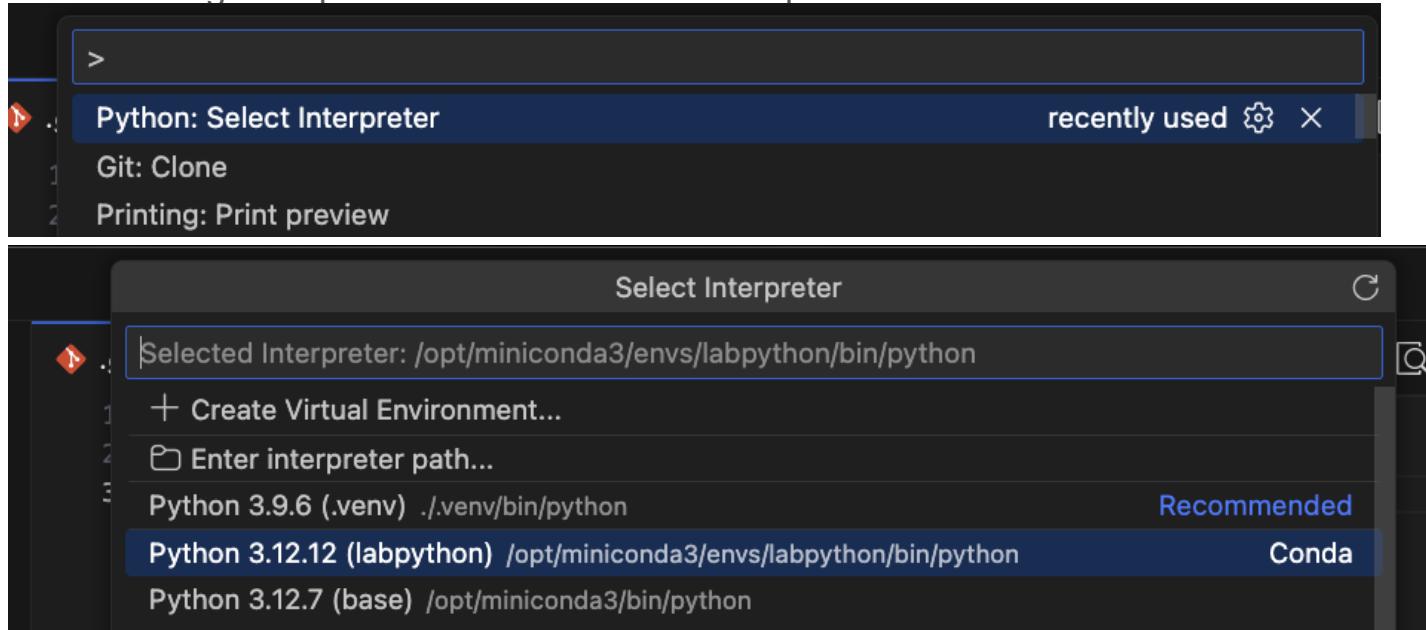
Quando è attivo vedrete una cosa tipo

○ (labpython) lauranenzi@iMacLaura LabProgrammazione\_I\_2026 %

# Setup dell'ambiente

## 4b) Collegare VS Code all'ambiente:

- Scrivere **Python: Select Interpreter** nel Command Palette
- Scegliere quello con il nome scelto nel percorso.



# Setup dell'ambiente

5a) Registratevi su GitHub se non lo siete già

5b) Createvi un repository su GitHub chiamato “ProgrammingLab”

**Create a new repository**

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

Required fields are marked with an asterisk (\*).

Owner \* lauranenzi / Repository name \* ProgrammingLab ✓ ProgrammingLab is available.

Great repository names are short and memorable. Need inspiration? How about [solid-succotash](#) ?

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

Add .gitignore  
.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license  
License: **None**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

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

ⓘ You are creating a public repository in your personal account.

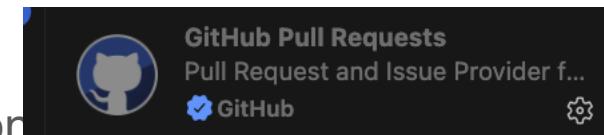
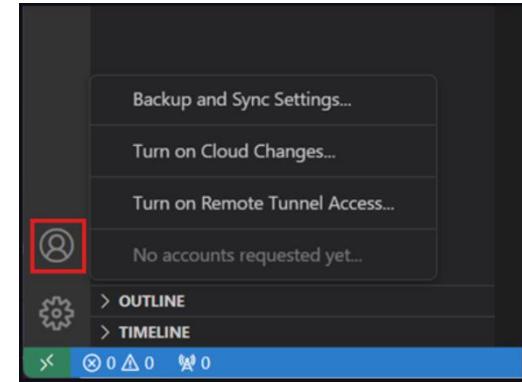
**Create repository**

# Setup dell'ambiente

6) Accedere a VS Code col vostro account GitHub nel menu Account in basso a destra della barra Attività.

- Aggiungere l'estensione: Github Pull Requests
- Se Git è mancante, vengono mostrate le istruzioni. Sito git <https://git-scm.com/downloads>
- Assicuratevi di riavviare VS Code in seguito
- Come dicono [qui](#), settate il vostro username ed email da terminale (va bene di vscode)

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```



# Setup dell'ambiente

7) Scrivere il comando "Git: Clone" nel Command Palette e selezionare il Clone Repository nel Source Control. Scegliete quindi il repository "ProgrammingLab" che dovrebbe comparire tra i vostri repository

The screenshot shows the VS Code interface with the "Source Control" sidebar open. The "Clone Repository" button is highlighted with a green dashed border. The main area displays a search bar and a list of repositories. The first repository listed is "lauranenzi/Test" with the URL "https://github.com/lauranenzi/Test.git". Below it is a description: "Repo for the programming Lab course". The second repository listed is "lauranenzi/MyProgrammingLab" with the URL "https://github.com/lauranenzi/MyProgrammingLab.git". The third repository listed is "lauranenzi/ProgrammingLab\_I" with the URL "https://github.com/lauranenzi/ProgrammingLab\_I.git". A partial description for this repository is visible: "Sito web del corso \"LABORATORIO DI PROGRAMMAZIONE I\" per Intelligenza Artificiale e pe". The fourth repository listed is "LogArtLab/simple-monitor" with the URL "https://github.com/LogArtLab/simple-monitor.git". The fifth repository listed is "marty90/inginf-web-site" with the URL "https://github.com/marty90/inginf-web-site.git". A partial description for this repository is visible: "Hugo based website for the ING-ING Group @UniTS". The bottom of the list shows the start of another repository: "LogArtLab/github" with the URL "https://github.com/LogArtLab/github.git".

SOURCE CONTROL

In order to use git features, you can open a folder containing a git repository or clone from a URL.

Open Folder

Clone Repository

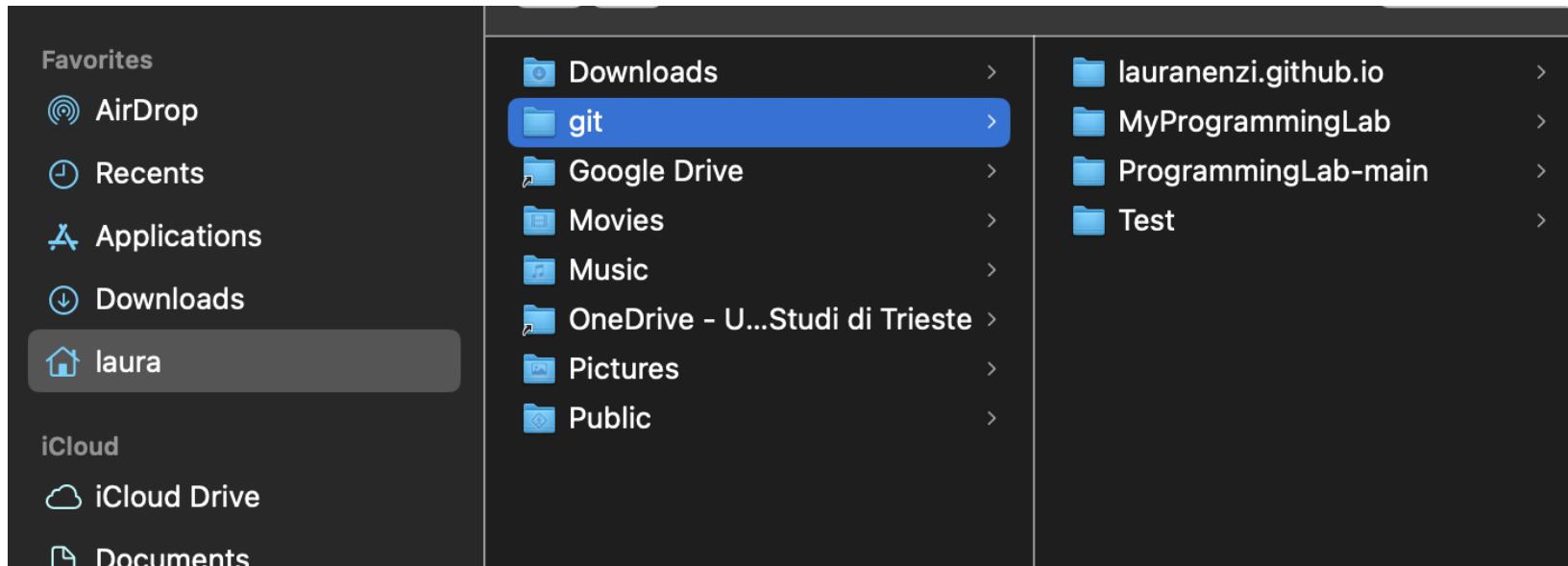
To learn more about how to use git and source control in VS Code [read our docs](#).

Repository name (type to search)

- lauranenzi/Test <https://github.com/lauranenzi/Test.git>  
Repo for the programming Lab course
- lauranenzi/MyProgrammingLab <https://github.com/lauranenzi/MyProgrammingLab.git>
- lauranenzi/ProgrammingLab\_I [https://github.com/lauranenzi/ProgrammingLab\\_I.git](https://github.com/lauranenzi/ProgrammingLab_I.git)  
Sito web del corso "LABORATORIO DI PROGRAMMAZIONE I" per Intelligenza Artificiale e pe
- LogArtLab/simple-monitor <https://github.com/LogArtLab/simple-monitor.git>
- marty90/inginf-web-site <https://github.com/marty90/inginf-web-site.git>  
Hugo based website for the ING-ING Group @UniTS
- LogArtLab/github <https://github.com/LogArtLab/github.git>

# Setup dell'ambiente

7) Salvare il repository localmente sul computer. Vi consiglio di creare una cartella git dove salvate tutti i repository localmente.



# Setup dell'ambiente

8a) Dovete aggiungere un file `.gitignore` dove potete mettere tutti i file che non vanno sincronizzati

The screenshot shows the VS Code interface with two main panes. The left pane, titled 'EXPLORER', displays a file tree for a project named 'LABPROGRAMMAZI...'. It contains several folders and files, including '.venv', 'altro\_materiale', 'esercizi' (which contains 'data', 'ex1.py', and 'Soluzioni\_esLezione2.py'), 'slides', and 'README.md'. A green circle highlights the '.gitignore' file, which is selected and has a blue border. The right pane, titled '.gitignore', shows the contents of the file:

```
1  .vscode/
2  .venv/
3  .DS_Store
```

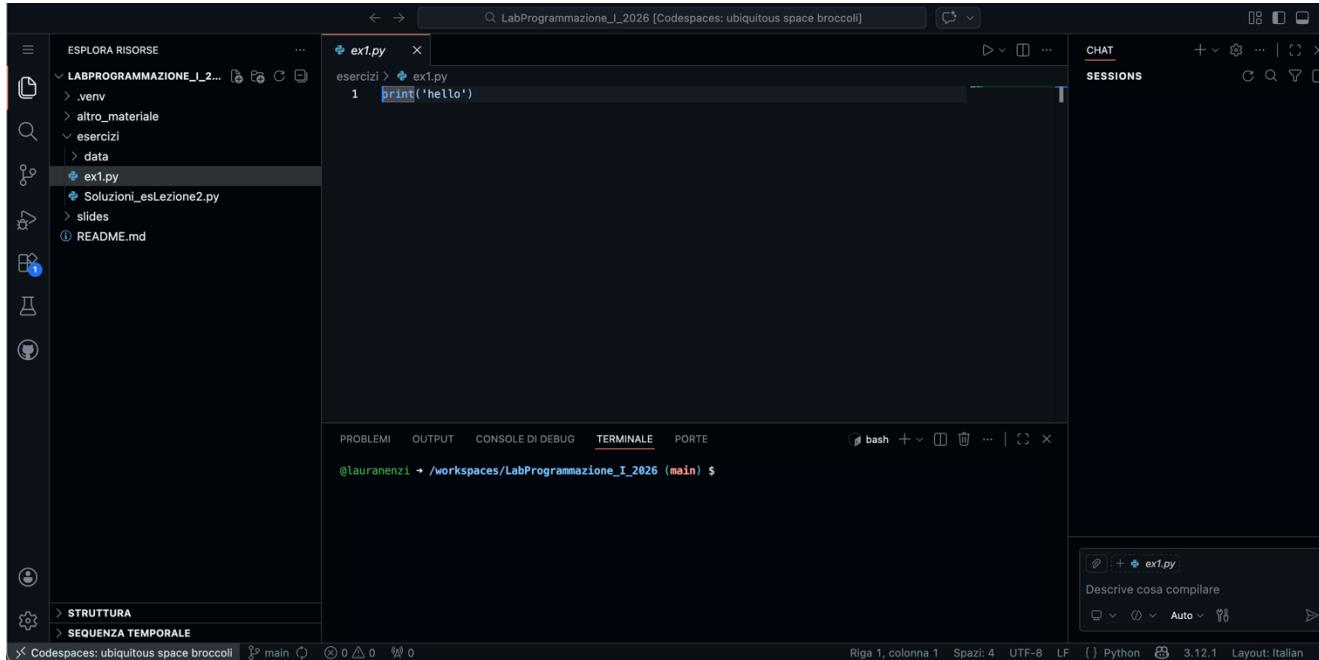
# Codespaces

- Potete anche lavorare con codespaces direttamente online. **Dovete però sapere usare VSC localmente!**

The screenshot shows a GitHub repository page for "LabProgrammazione\_I\_2026". The repository is private and was generated from [lauranenzi/ProgrammingLab\\_I](#). The main navigation bar includes "main", "1 Branch", "0 Tags", a search bar, and a green "Code" button. Below the navigation, there's a list of files and folders: "prima file", "altro\_materiale", "esercizi", "slides", ".DS\_Store", "README.md", and "README". On the right side, there's a "Codespaces" section with tabs for "Local" and "Codespaces". The "Codespaces" tab is active, showing "Your workspaces in the cloud". It lists one workspace: "ubiquitous space broccoli" (20h ago), which is currently on the "main" branch and has "No changes". A red arrow points from the text "On current branch" to the workspace entry. To the right of the workspace list is an "About" section with details like "No description, website, or topics provided.", "Readme", "Activity", "0 stars", "0 watching", "0 forks", and a "Releases" section.

# Codespaces

- L'ambiente creato è come quello che create localmente su VSC. Dovete comunque selezionare l'interprete e sincronizzare l'ambiente su GitHub



# Primi comandi

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

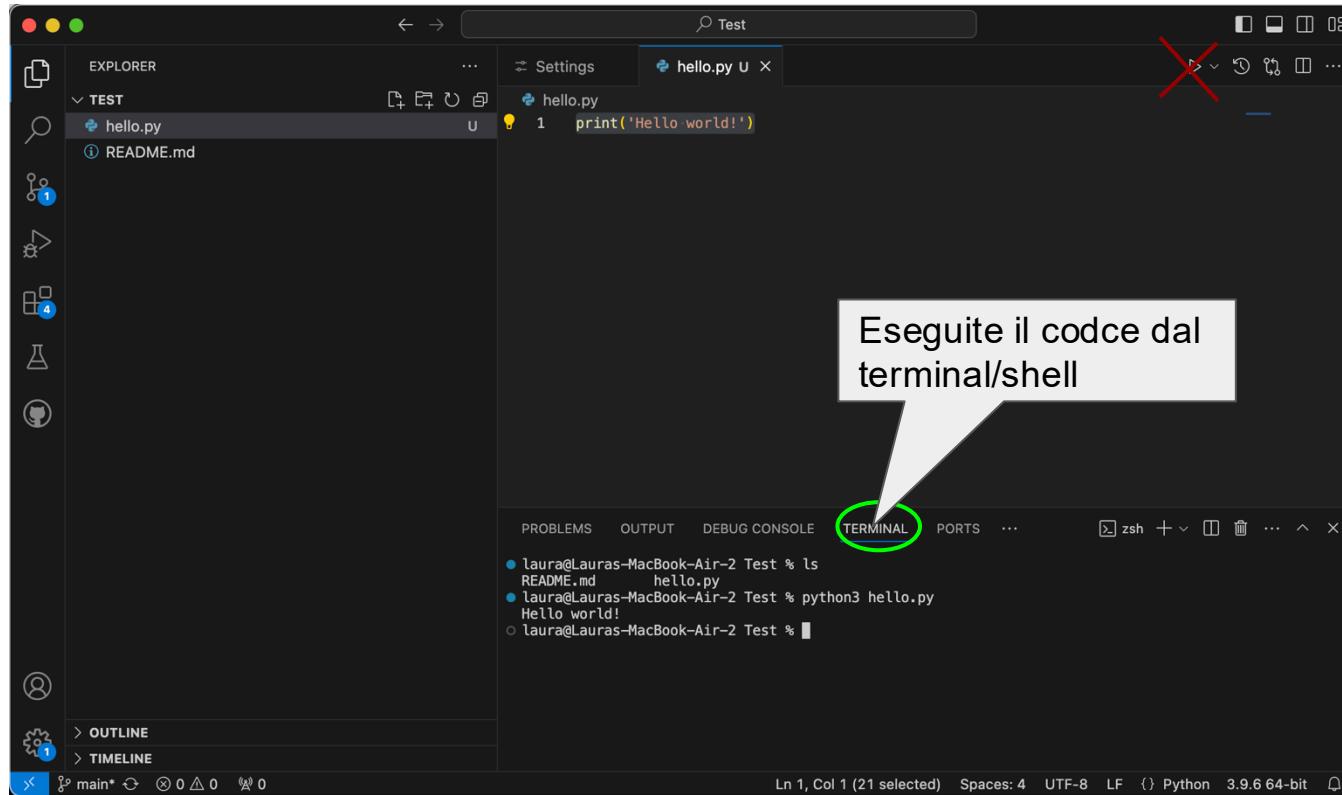
```
print('Hello world!')
```

Poi, eseguiamo lo script dalla shell:

```
python hello.py
```

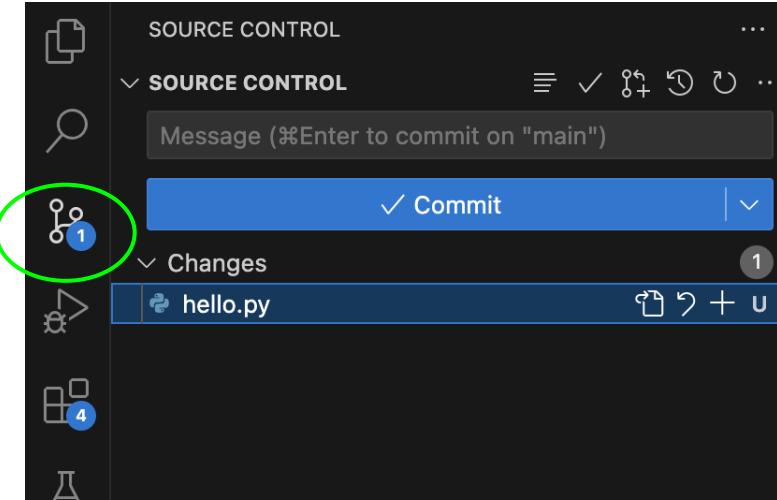
Assicuratevi di essere nella cartella dove c'è il file!!!

# Primi comandi



# Source Control

- Puoi accedere alla “Source Control view” dalla Barra dell’attività che elenca tutti i file modificati nel tuo spazio di lavoro.



# Staging

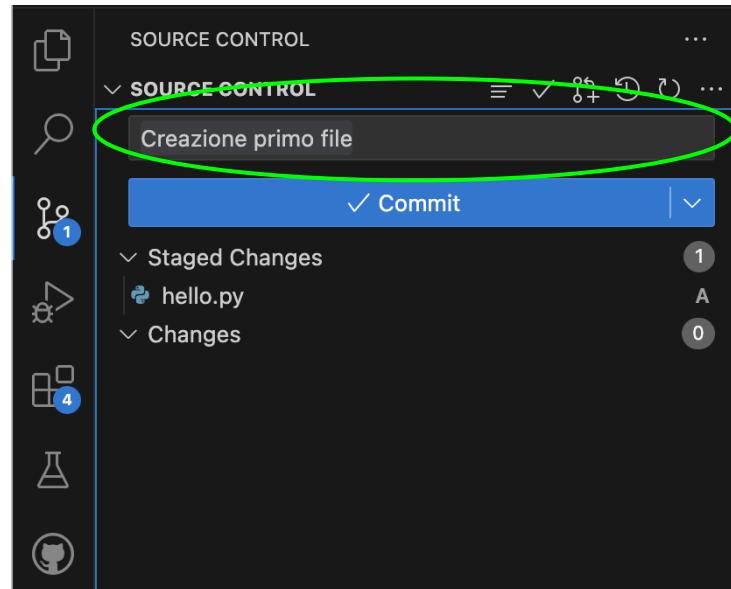
- Quando si seleziona un file, l'editor mostra una vista diff che evidenzia le modifiche del file rispetto al file precedentemente modificato.



- Per mettere in staging un file, seleziona l'icona + (più) accanto al file nella vista Controllo sorgente. Questo aggiunge il file alla sezione Modifiche in staging, indicando che verrà incluso nel prossimo commit.

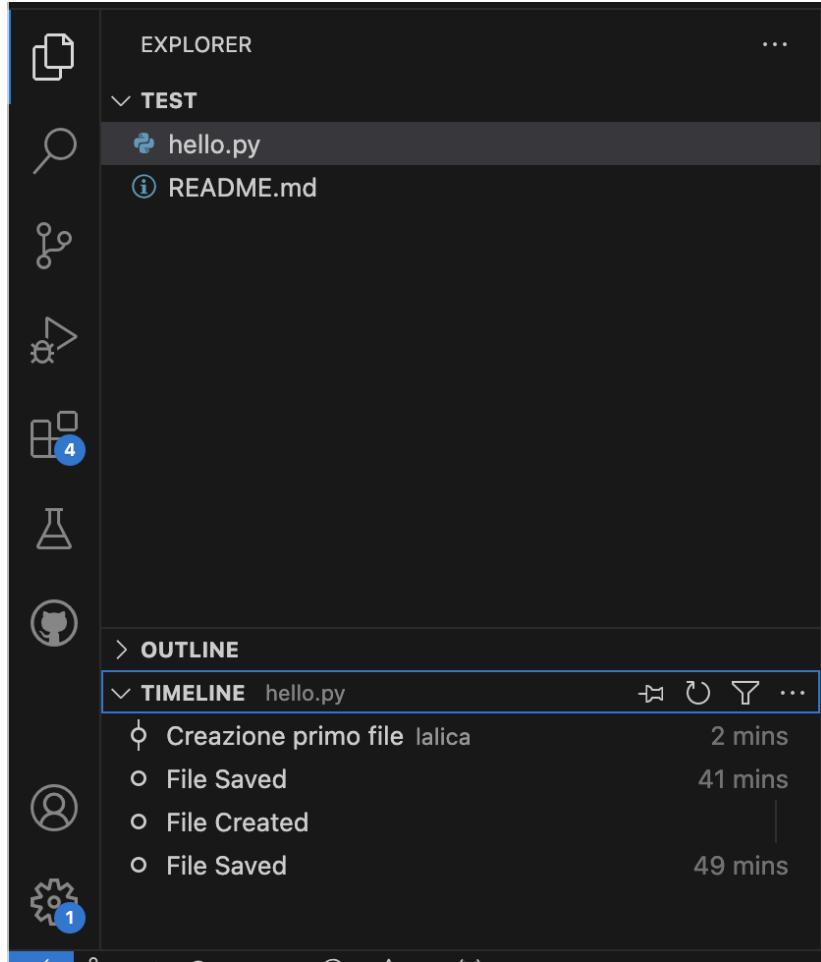
# Commit

- Per eseguire il commit delle modifiche in stage, digita un messaggio di commit nella casella di testo superiore, quindi seleziona il pulsante Commit. Questo salva le modifiche nel repository Git locale, consentendoti di ripristinare le versioni precedenti del codice se necessario.
- Suggerimento: eseguite il commit delle modifiche in anticipo e spesso. Ciò rende più facile tornare alle versioni precedenti del tuo codice, se necessario.



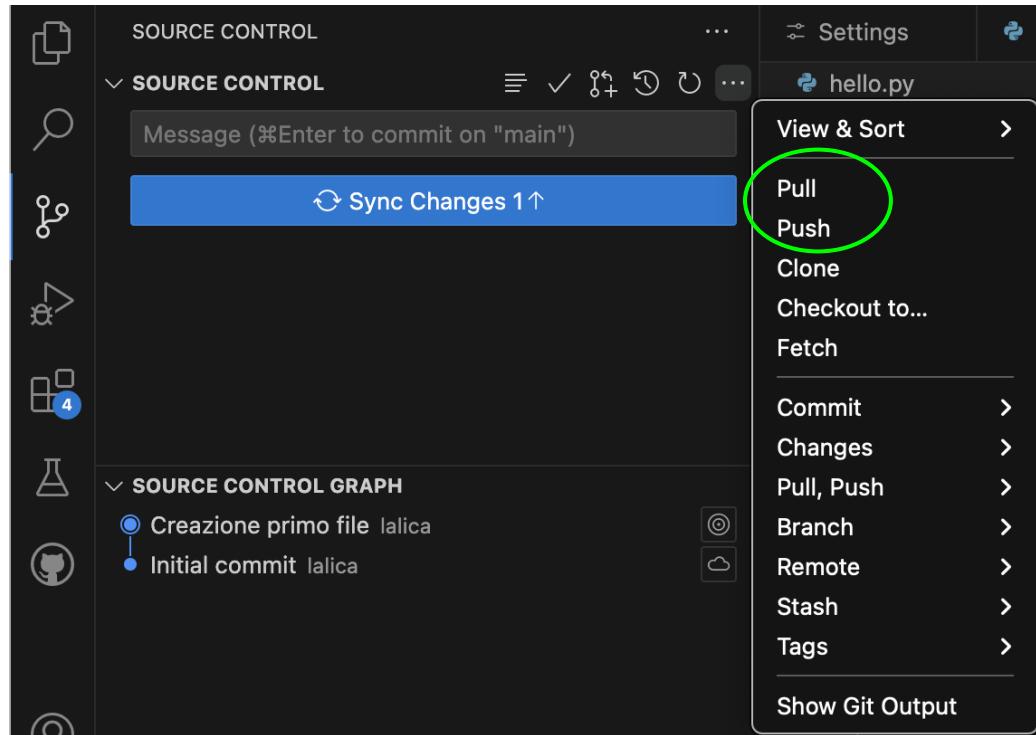
# Timeline

- È possibile navigare e rivedere tutte le modifiche dei file locali nella vista “Timeline” disponibile nella parte inferiore della vista Esplora.



# Push and Pull

- Push carica le tue modifiche sul remoto.
- Pull scarica le modifiche dal remoto.
- È possibile accedere ai comandi Push e Pull dal menu Source Control.



# Git da terminale

The screenshot shows a terminal window integrated into a code editor interface. On the left, there's an 'OUTLINE' section with a tree view of a 'hello.py' file. The main area is a terminal window with the following content:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    COMMENTS  
Type "help", "copyright", "credits" or "license" for more information.  
● >>> quit()  
○ laura@Lauras-MacBook-Air-2 Test %  
○ laura@Lauras-MacBook-Air-2 Test %  
○ laura@Lauras-MacBook-Air-2 Test %  
○ laura@Lauras-MacBook-Air-2 Test %  
● laura@Lauras-MacBook-Air-2 Test % ls  
 README.md          hello.py          primoessercizio.py  
● laura@Lauras-MacBook-Air-2 Test % git add hello.py  
● laura@Lauras-MacBook-Air-2 Test % git commit -m "aggiunta commento"  
 [main eb85365] aggiunta commento  
   1 file changed, 2 insertions(+), 1 deletion(-)  
● laura@Lauras-MacBook-Air-2 Test % git push  
 Enumerating objects: 5, done.  
 Counting objects: 100% (5/5), done.  
 Delta compression using up to 8 threads  
 Compressing objects: 100% (2/2), done.  
 Writing objects: 100% (3/3), 361 bytes | 361.00 KiB/s, done.  
 Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
 To https://github.com/lauranenzen/Test.git  
   1e490f9..eb85365 main -> main  
○ laura@Lauras-MacBook-Air-2 Test %
```

The terminal output is annotated with several green circles highlighting specific lines: the command `git commit`, the commit message, the command `git push`, and the output of the push operation.

# Cronologia su Github

The screenshot shows a GitHub repository interface with the following details:

- Repository:** lauranenzi / Test
- Branch:** main
- Commits on Nov 6, 2024:**
  - aggiunta commento** (Commit ID: eb85365) - authored by lauranenzi 2 minutes ago.
- Commits on Nov 5, 2024:**
  - Create primoesercizio.py** (Commit ID: 1e490f9) - authored by lauranenzi 22 minutes ago. Status: Verified.
  - Creazione primo file** (Commit ID: 958ac53) - authored by lauranenzi 35 minutes ago.
  - Initial commit** (Commit ID: 83f8576) - authored by lauranenzi 1 hour ago. Status: Verified.

At the bottom, there is a note: "See something unexpected? Take a look at the [GitHub commit guide](#).

# Altre Estensioni di vscode utili

- rainbow indent

# Backup slides

# Come clonare una repository GitHub (via HTTPS)

- **Creazione Personal Access Token (PAT):**
  - Accedi a GitHub e clicca sulla tua foto profilo (in alto a destra) e seleziona "Settings";
  - Scorri fino a "Developer settings" nel menù a sinistra;
  - Seleziona "Personal access tokens" e poi la voce "Tokens (classic)";
  - Clicca su "Generate new token" e poi su "Generate new token (classic)";
  - Assegna un nome, imposta la scadenza e seleziona lo scope "repo" per accedere ai repository privati;
  - Clicca su "Generate token" e COPIA il codice generato (inizia con ghp\_) e salvalo da qualche parte. Nota: non potrai più vederlo su GitHub.

# Come clonare una repository GitHub (via HTTPS)

- **Clonazione via HTTPS:**
  - **Vai sul repository che vuoi clonare;**
  - **Clicca sul pulsante verde "Code" e copia l'URL HTTPS;**
  - **Apri il terminale e digita "git clone ed incolla l'URL HTTPS copiato precedentemente" ed esegui il comando;**
  - **Quando richiesto, inserisci il tuo username GitHub;**
  - **Quando viene richiesta la password, incolla il Personal Access Token generato e copiato prima (NON la password personale d'accesso).**