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

Ricevimento: libero, scrivetemi a lnenzi@units.it

Tutors:

- Nicholas Pearson: nicholasandrea.pearson@phd.units.it
- Michele Alessi: michele.alessi@phd.units.it
- Ermes Aviano: ermes.aviano@phd.units.it

Chi sono

Laura Nenzi

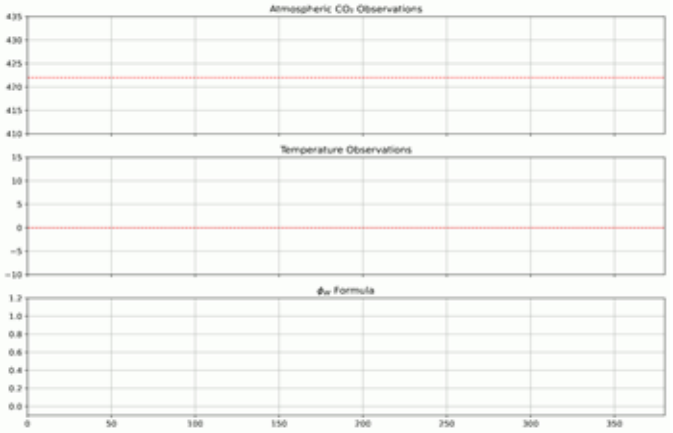
- Professore Associato in Ingegneria Informatica al Dipartimento di Ingegneria ed Architettura
- Laurea in Matematica, Phd in Computer Science
- Ricerca in Metodi Formali e Machine Learning

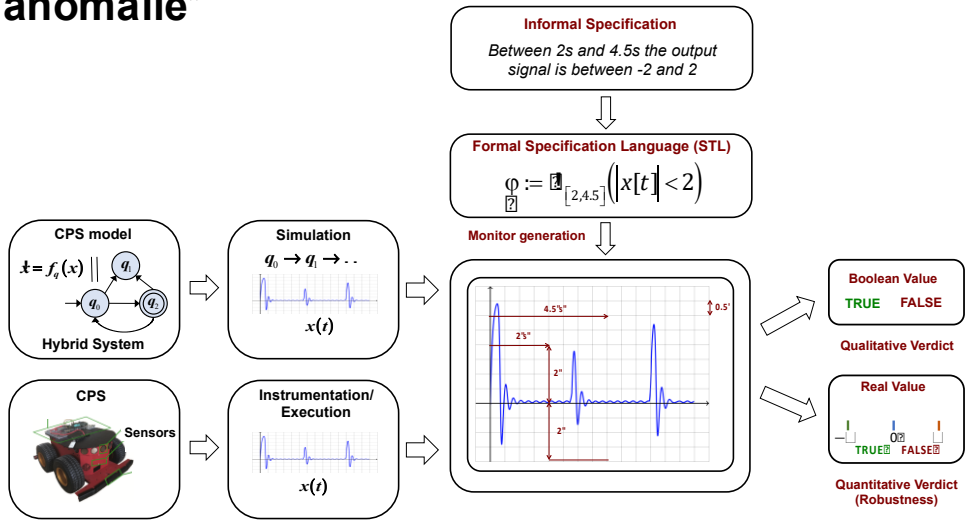
Ufficio c3 2.55

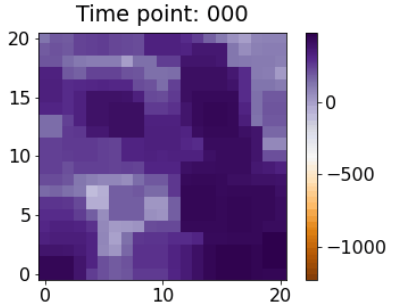
Mail: lnenzi@units.it

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**”



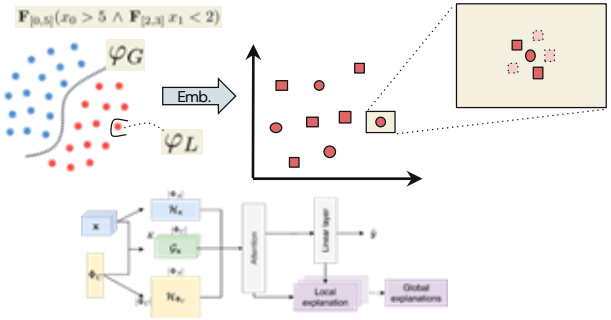


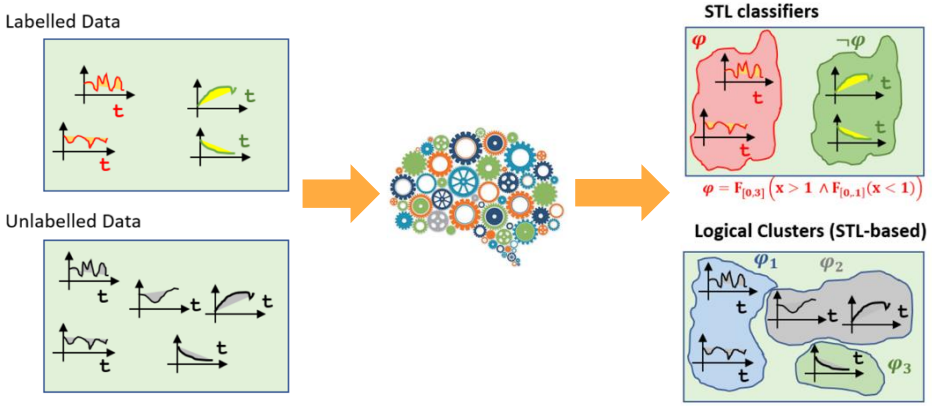


$$Crowdedness > K \rightarrow \Diamond_{d \leq 1} Crowdedness < K$$

Cosa Faccio: Explainable AI

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





"Ii 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 program-mazione.

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
    - 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 >= 18, obbligatorio se il voto è 16,17 ed obbligatorio a discrezione del docente se vengono rilevate dele 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

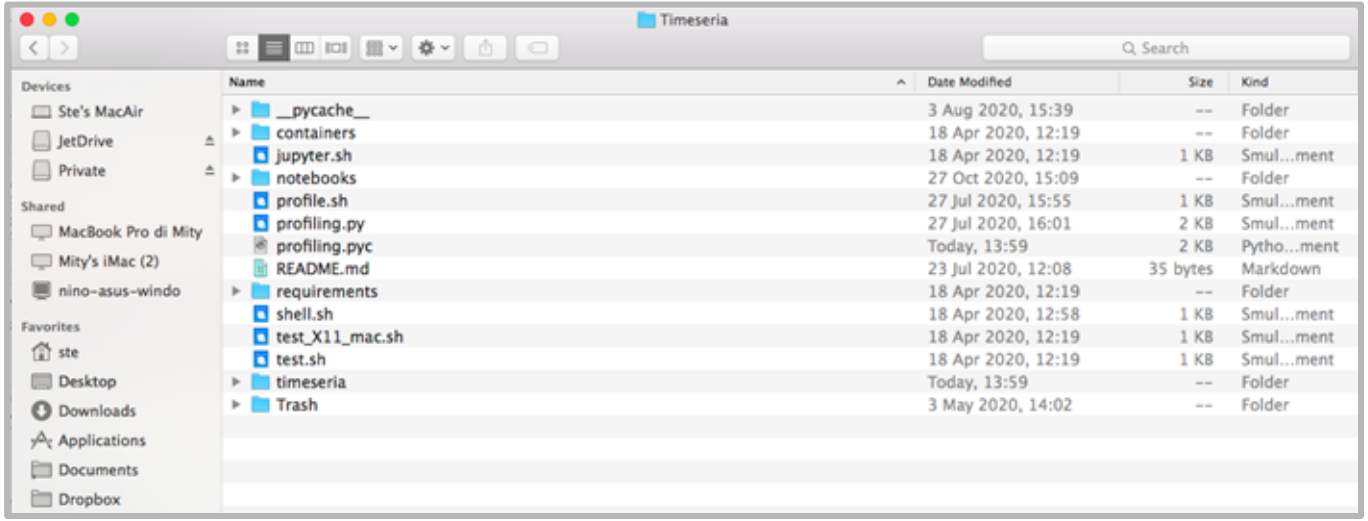
Iniziamo! ...con gli strumenti

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

- Abraham Lincoln

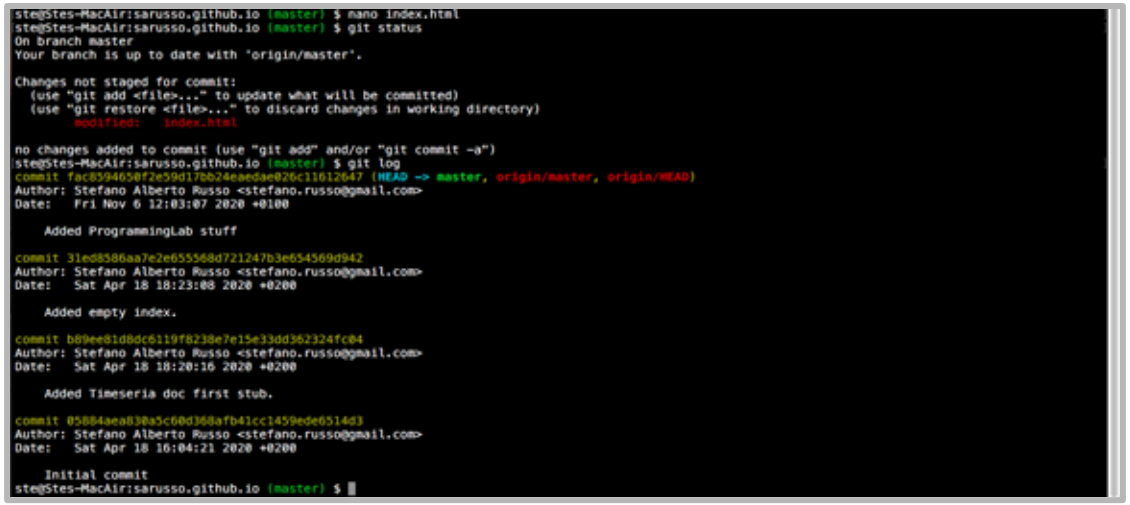
Strumenti: il File Manager

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.



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 indicat

Comandi shell - File e Directory Manipulation

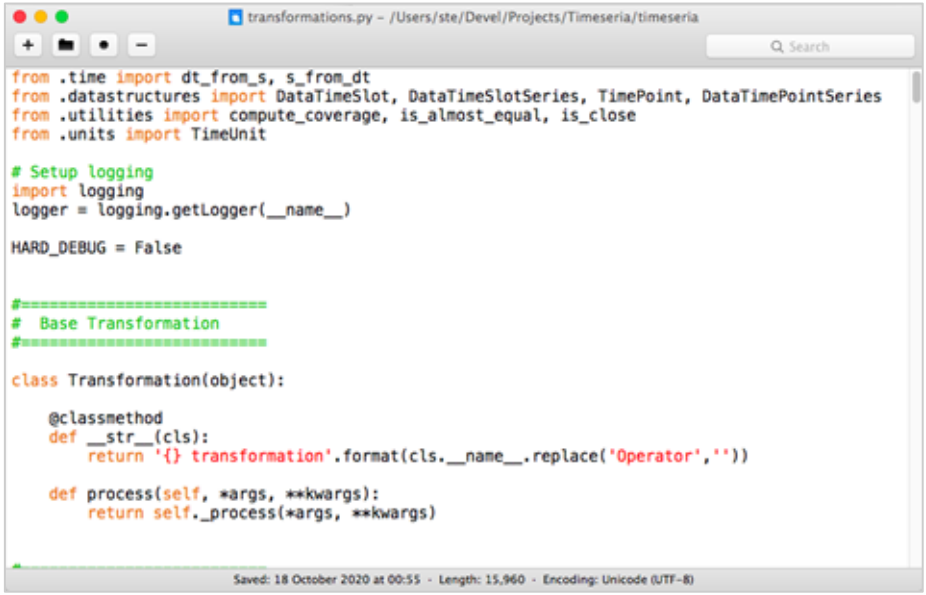
- **mkdir dirName**: crea una sub-directory con il nome indicato nella cwd

- **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

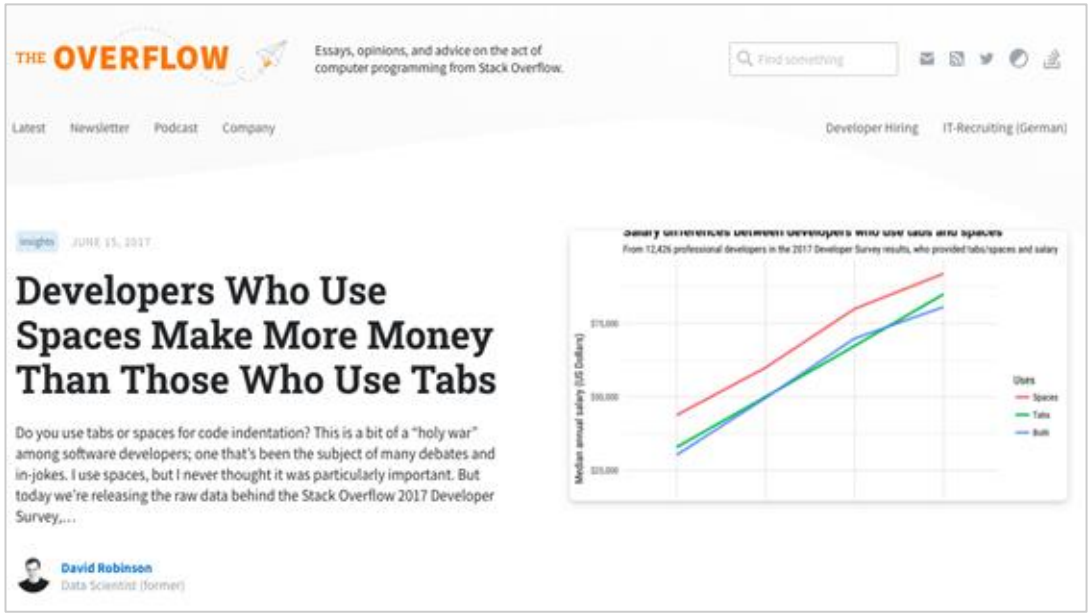
È un programma con cui scrivere il codice. A differenza di un programma di videoscrittura (come Microsoft Word), che formatta il testo con font, dimensioni e colori, un editor di testo è progettato principalmente per gestire testo semplice, senza elementi di formattazione avanzati. Gli editor del codice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



Strumenti: l'Editor del codice

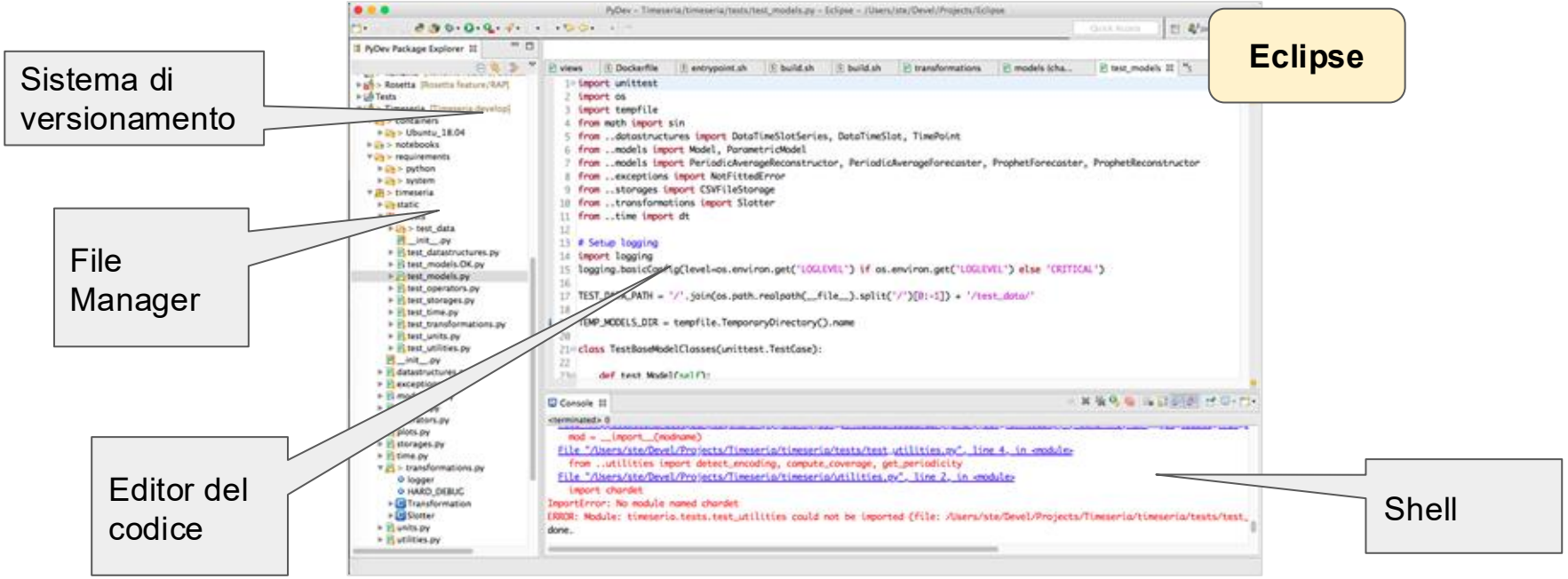
Nota: è tassativo impostare l'editor a usare gli spazi e non i tab.

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



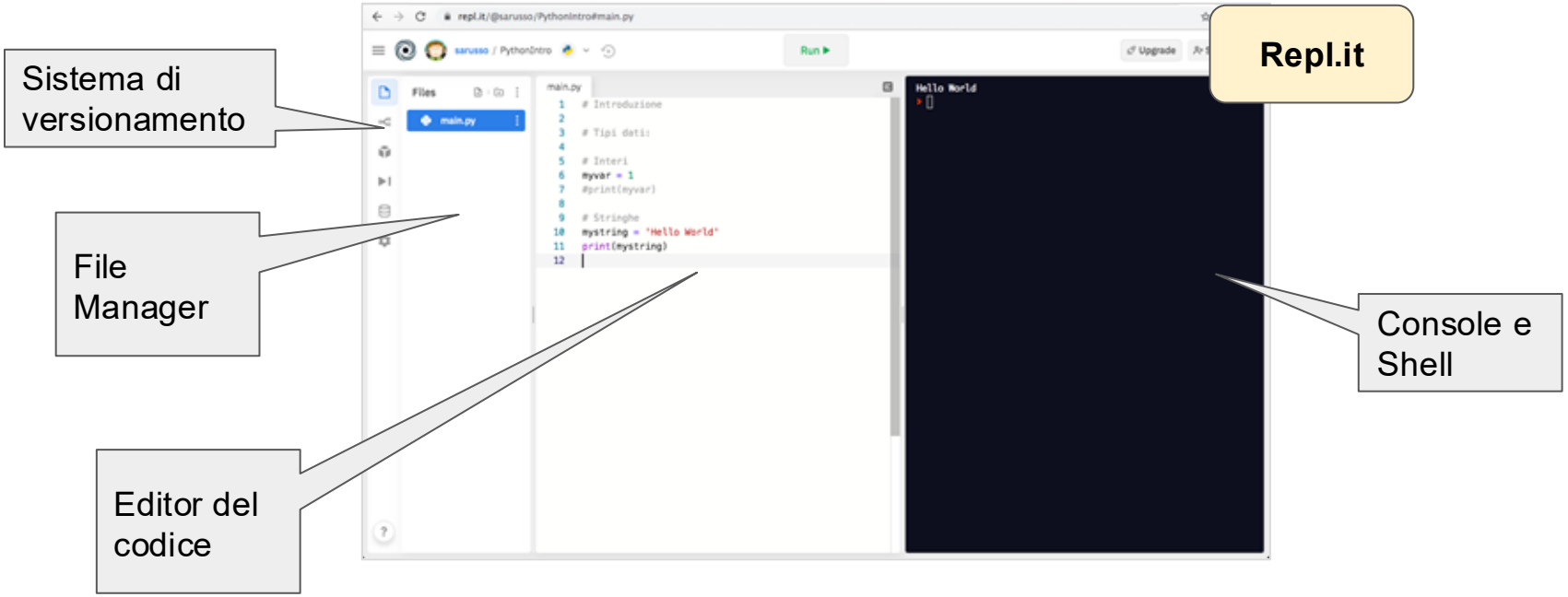
Strumenti: l'IDE (Integrated Development Environment)

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.



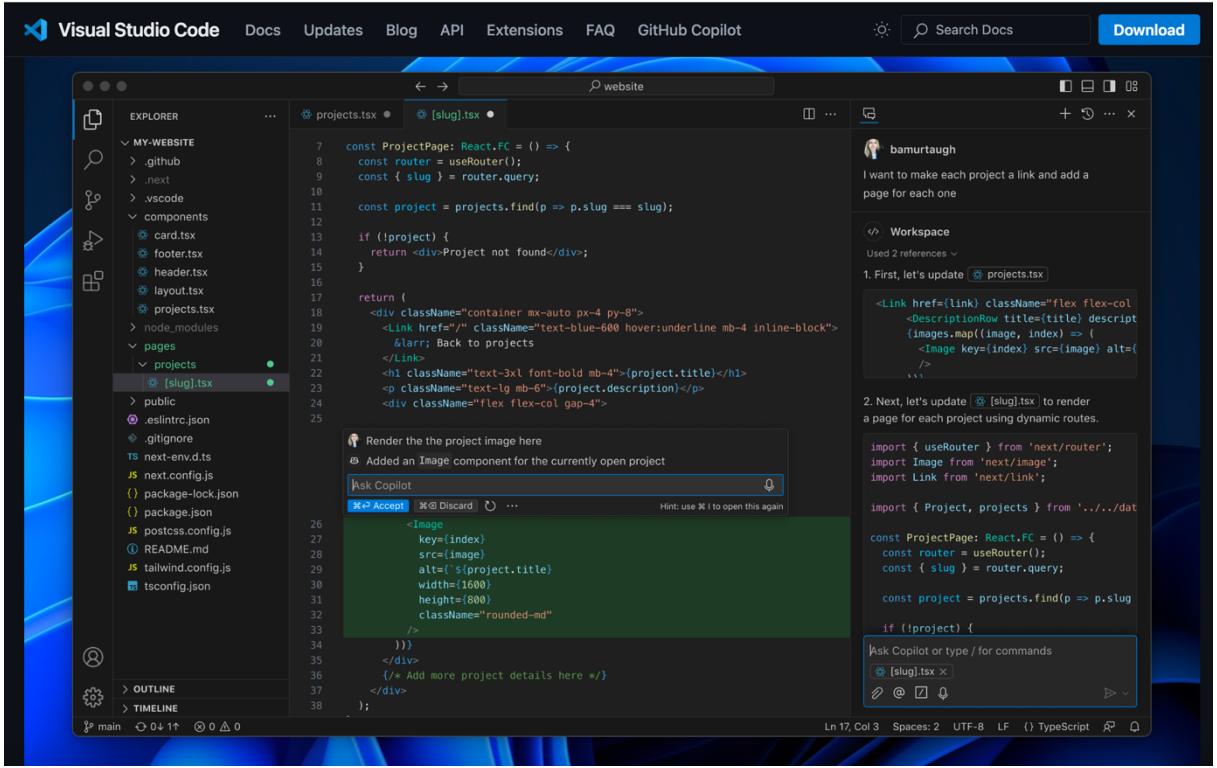
Strumenti: l'IDE (Integrated Development Environment)

- 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)

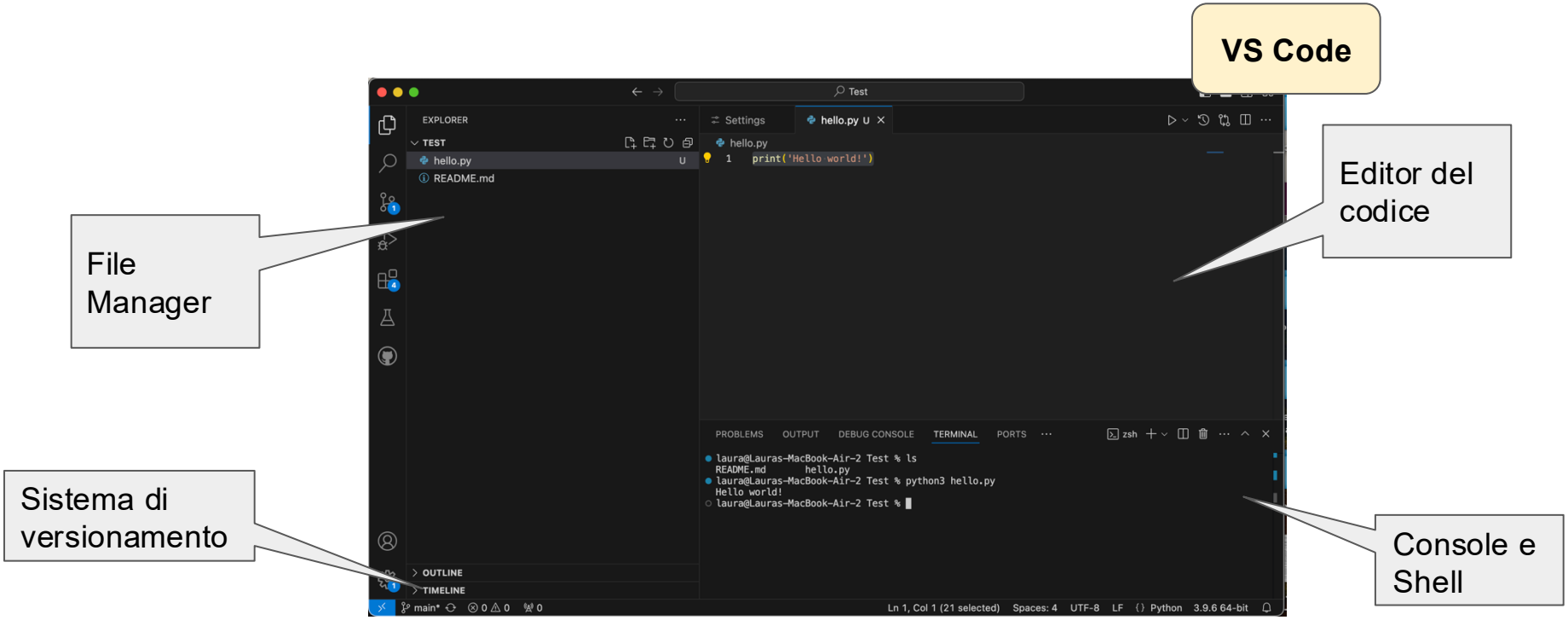
- È un editor di codice sorgente sviluppato da Microsoft. È gratuito, open-source e multipiattaforma (disponibile per Windows, macOS e Linux)



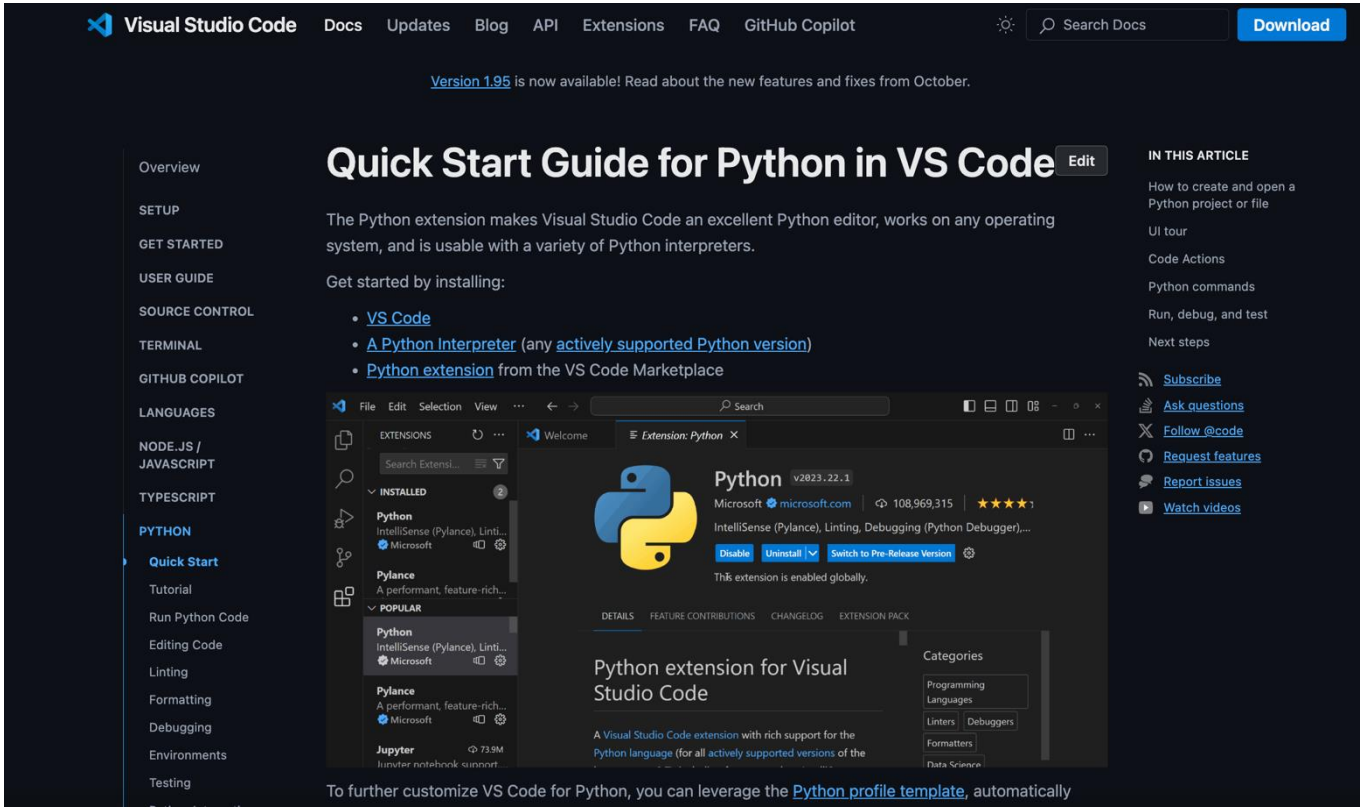
VS Code non è un IDE ma...

- 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

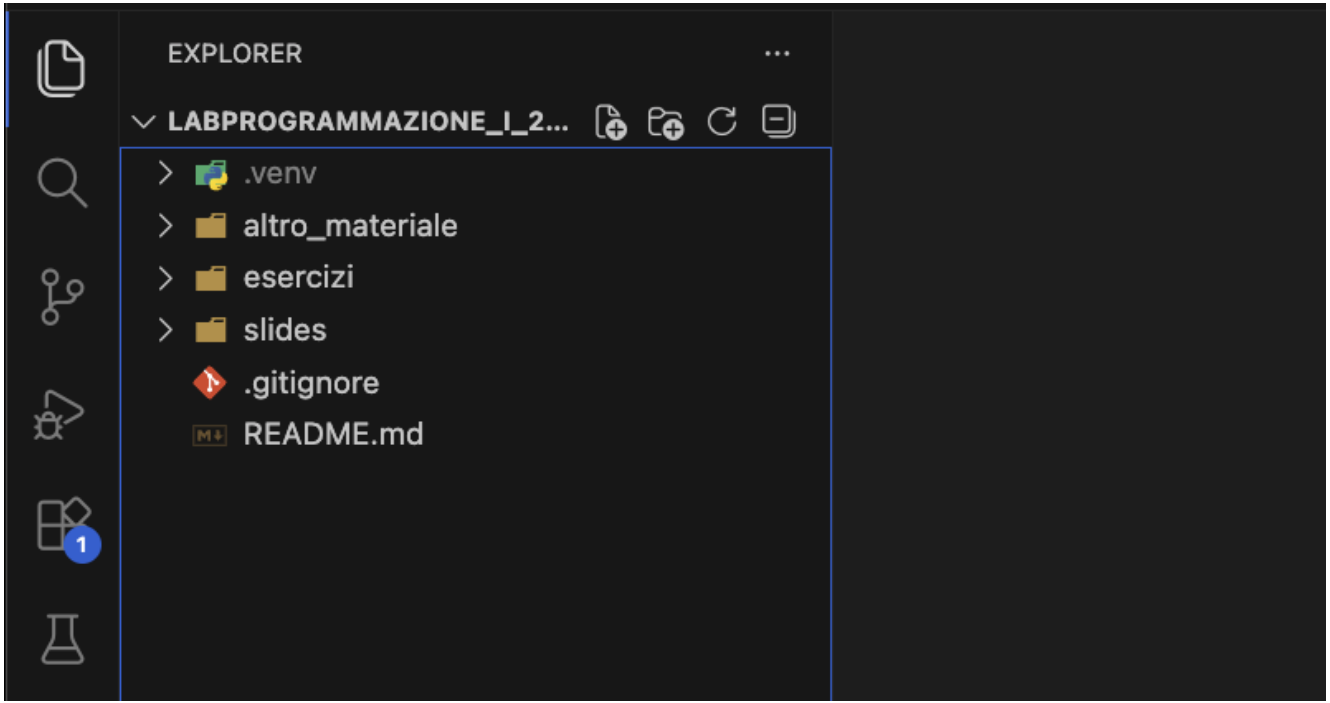


Ambiente Virtuale

- 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.



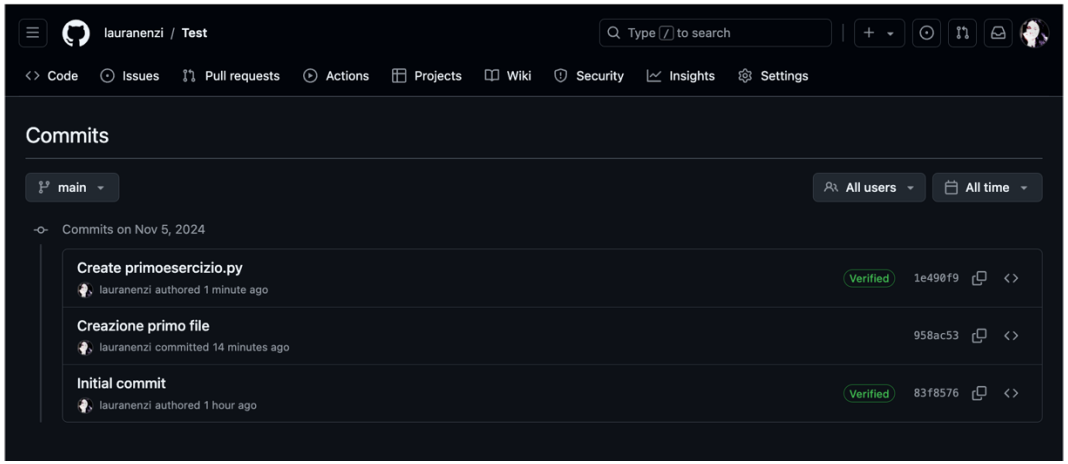
Ambiente Virtuale: conda

- 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.



Tutorial di Michele Rispoli (tutor dell'anno scorso):
https://github.com/drpOpZ/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.

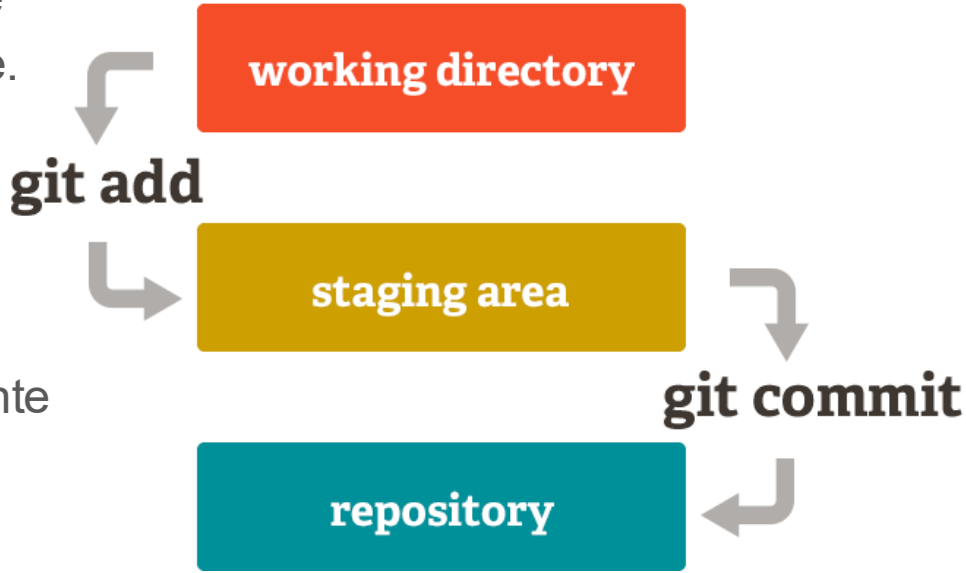
Git: Repository (o repo)

- 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.

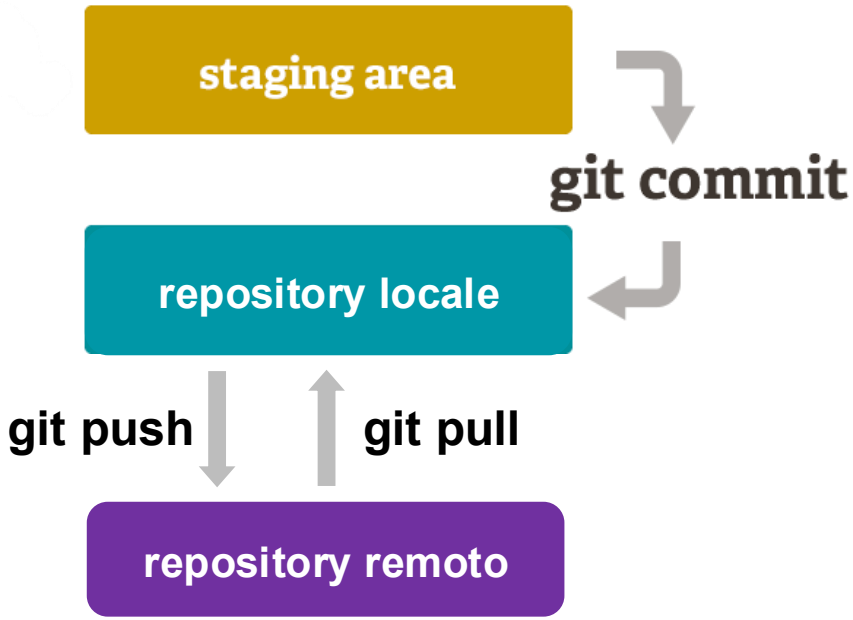


Git: Pull e Push

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.

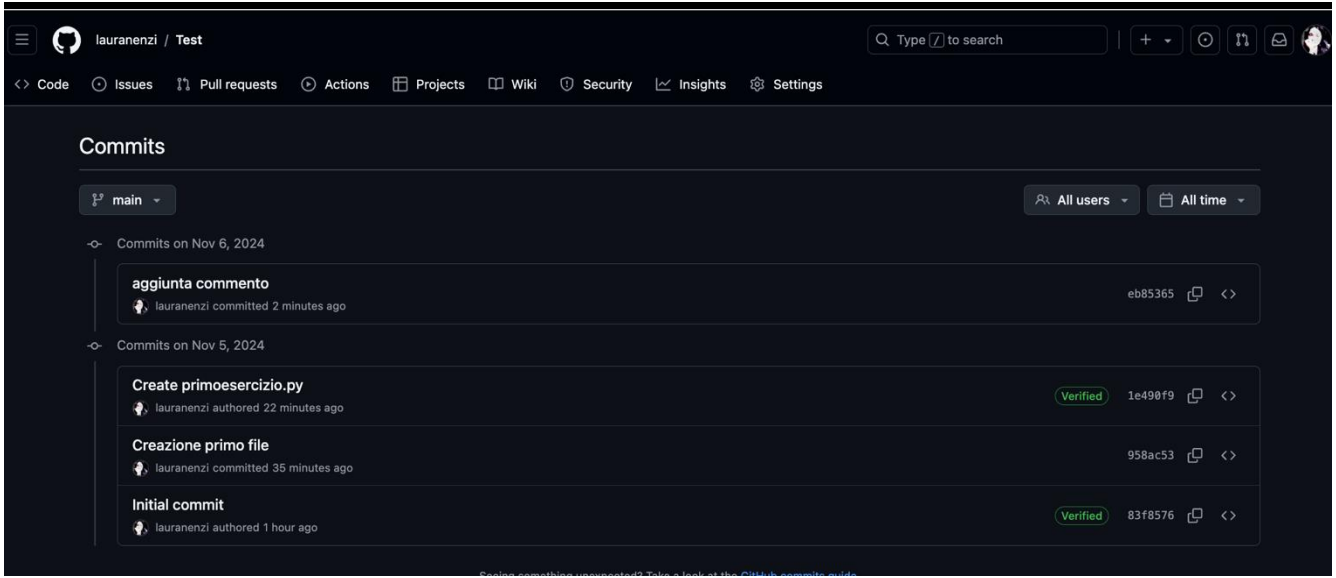


GitHub

- 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.



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)

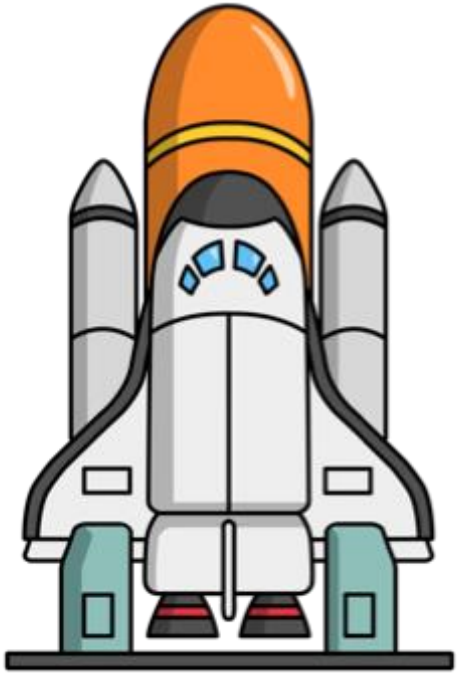
Esempio: trova i numeri in una lista minori di 5 e stampali

```
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

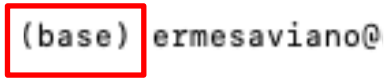
- Su https://www.anaconda.com/download/success scegliere il Graphical Installer e seuigre le istruzioni

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:**



- **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

- Inizializiamo conda da terminale:.
  Su macOS e Linux: `conda init`

  Su Windows:

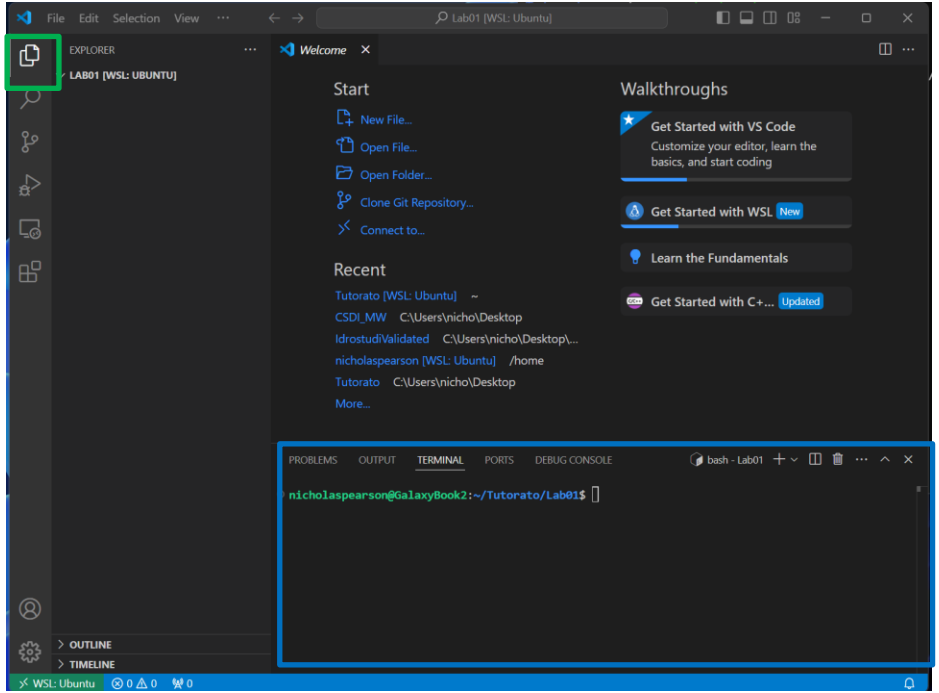
  - Apri dal menu Start: **Anaconda PowerShell Prompt**
  - Dentro questo terminale scrivi `conda init powershell`

- 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

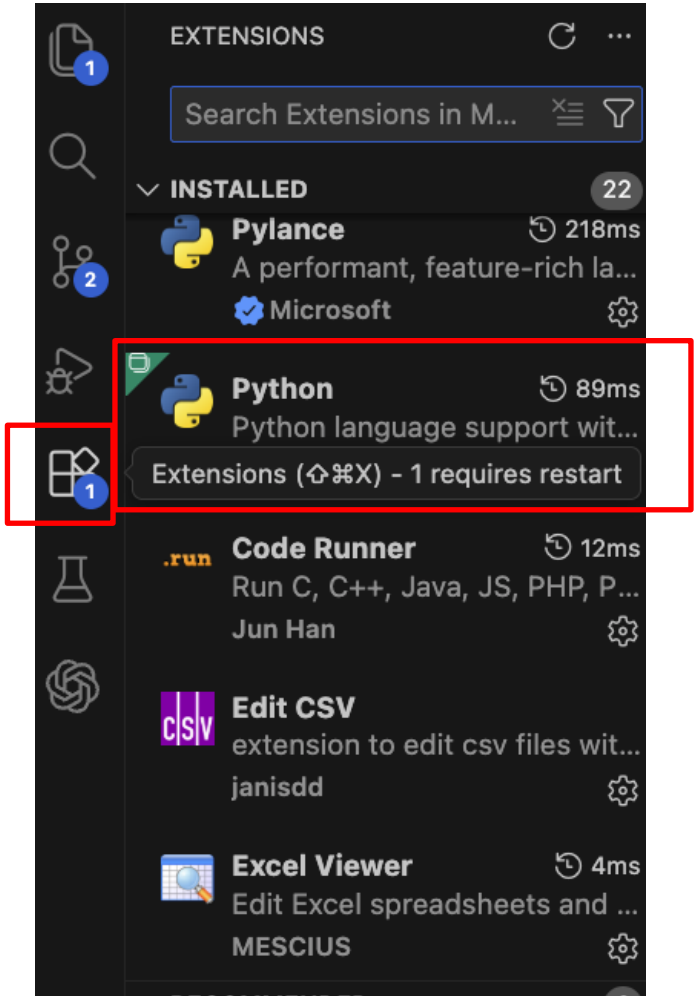
È possibile ci sia un messaggio di errore sul terminale in vscode.

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.

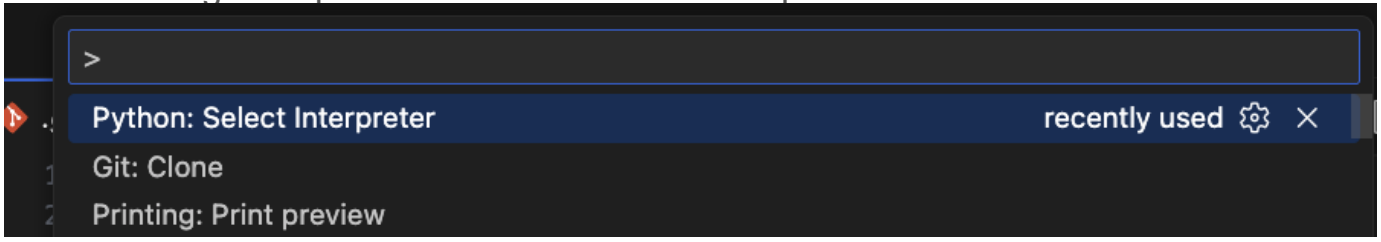
- Verificare che conda sia presente:
```
conda --version
```
- Creare un ”virtual enviroment”
```
conda create -n nome_ambiente python=3.12
```
- Attivare l’ambiente
```
conda activate nome_ambiente
```
- Per disattivare
```
conda deactivate
```

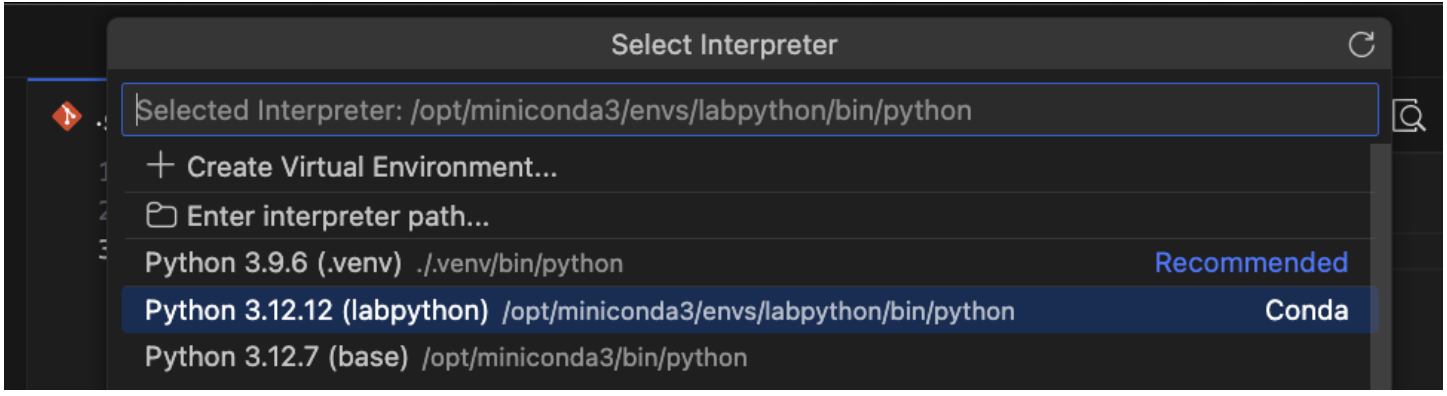
Quando è attivo vedrete una cosa tipo
```
(labpython) lauranenzi@iMacLaura LabProgrammazione_I_2026 %
```

Setup dell’ambiente

4b) Collegare VS Code all’ambiente:

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

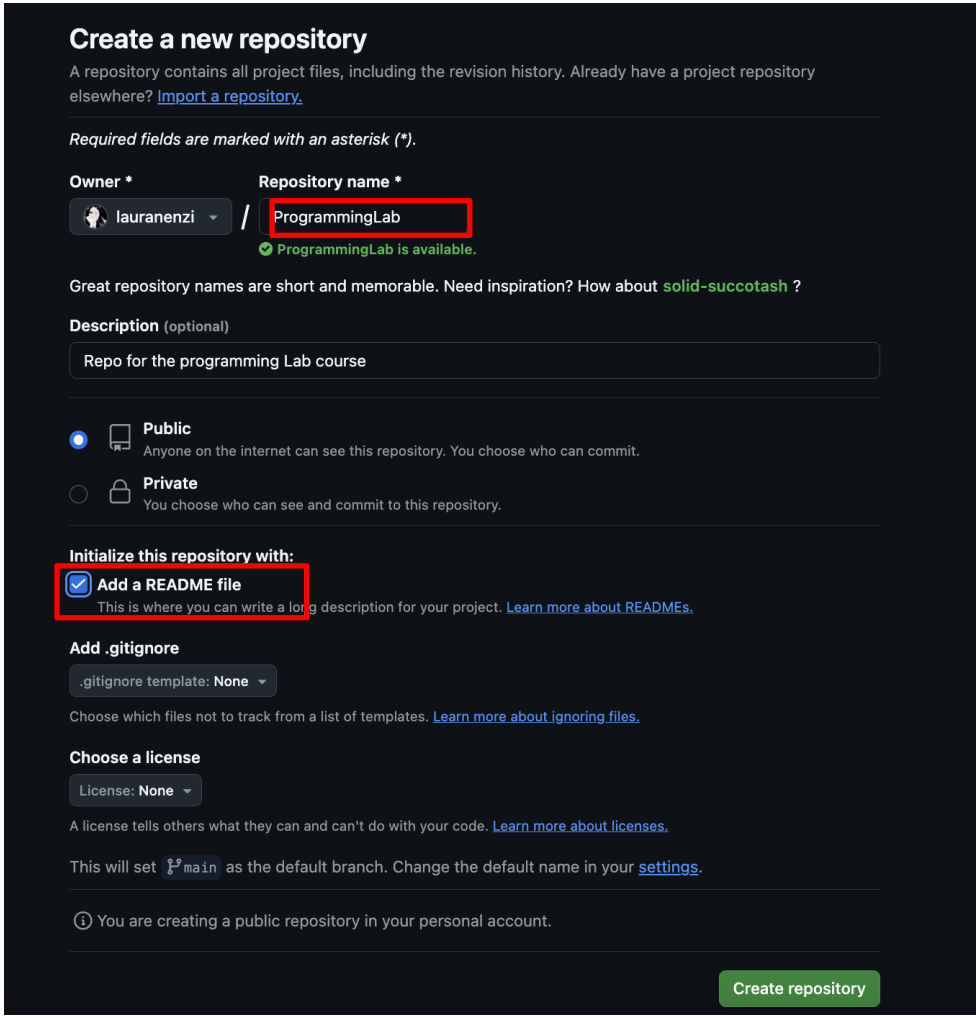




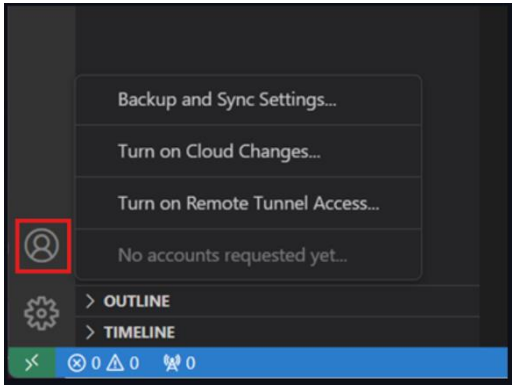
Setup dell’ambiente

5a) Registratevi su GitHub se non lo siete già

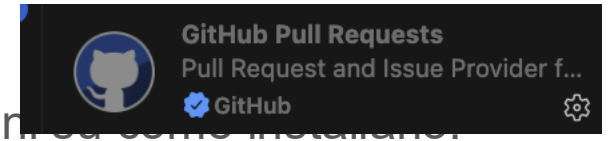
5b) Createvi un repository su GitHub chiamato “ProgrammingLab”



Setup dell’ambiente



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

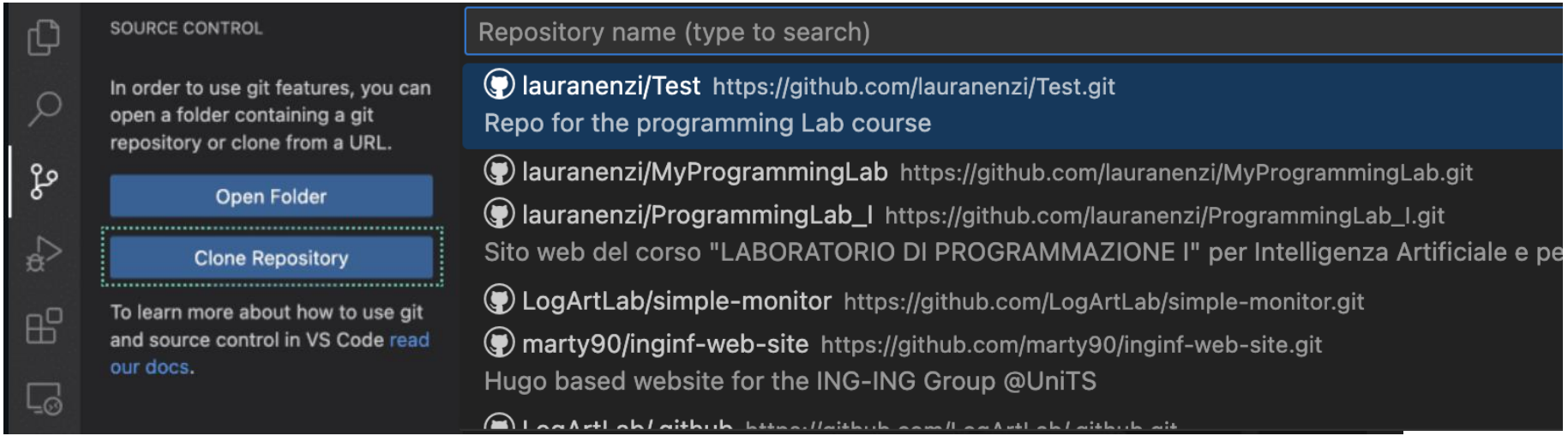


- Aggiungere l’estensione: Github Pull Requests
- Se Git è mancante, vengono mostrate le istruzioni su come installare. Sito git https://git-scm.com/downloads
- Assicuratevi di riavviare VS Code in seguito
- Come dicono qui, settate il vostro username ed email da terminale (va bene di vscode)

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

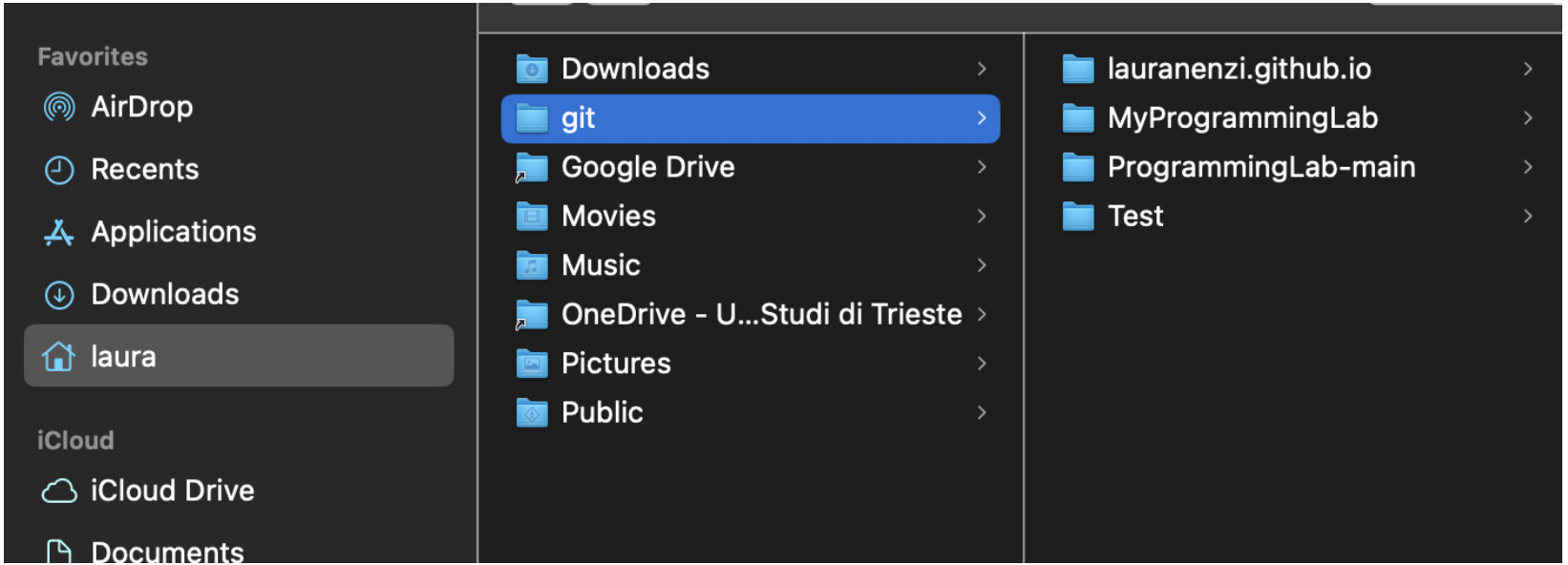
Setup dell’ambiente

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



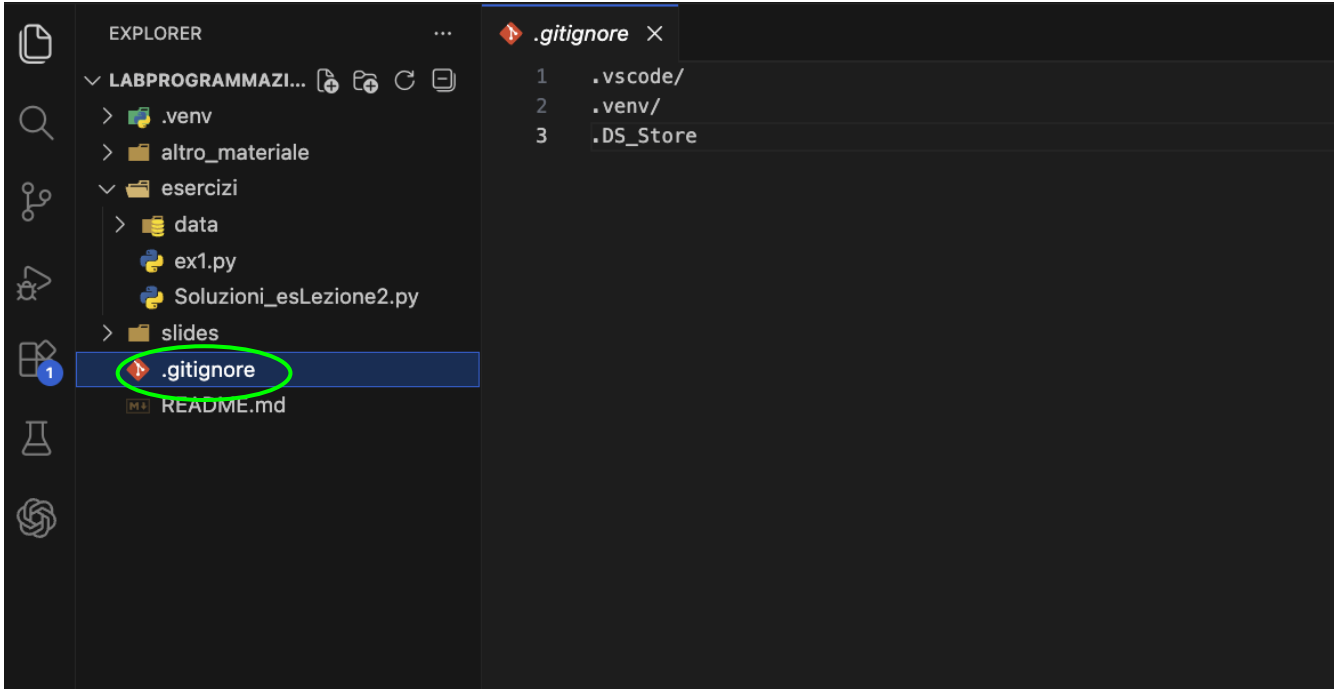
Setup dell’ambiente

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



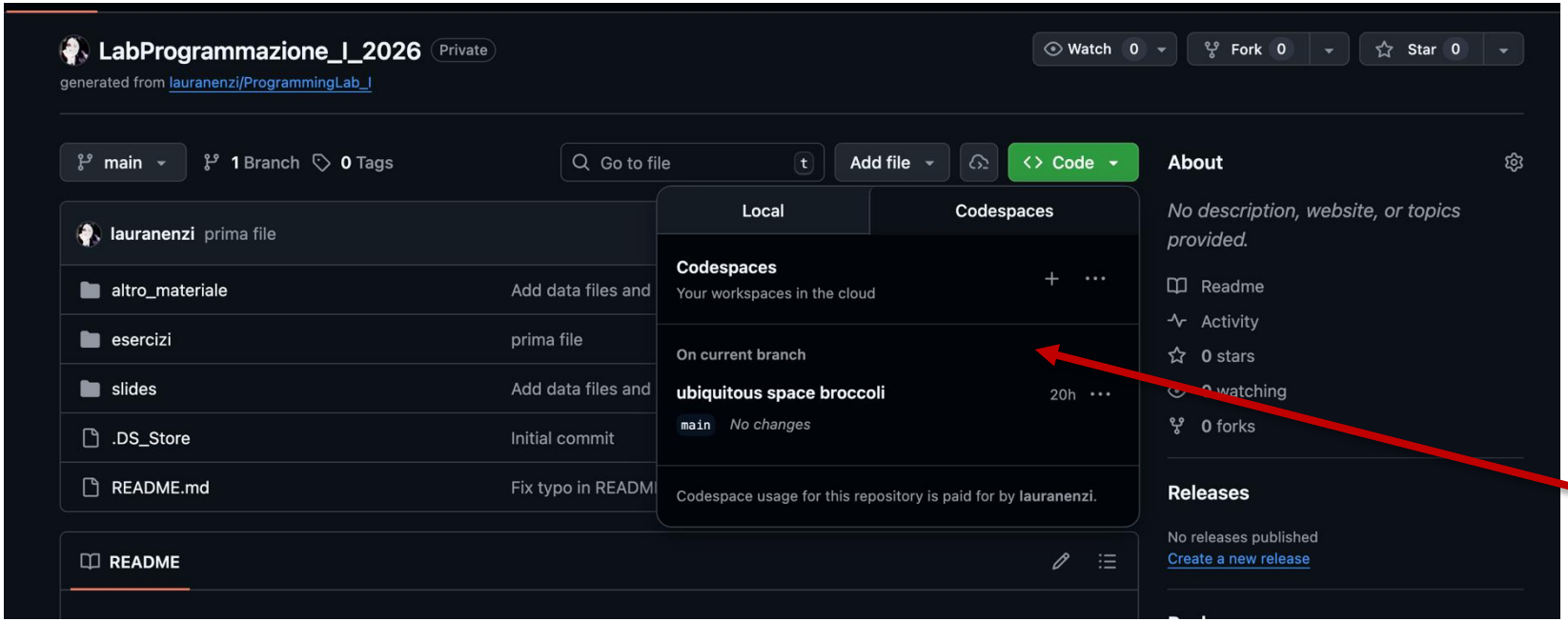
Setup dell’ambiente

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



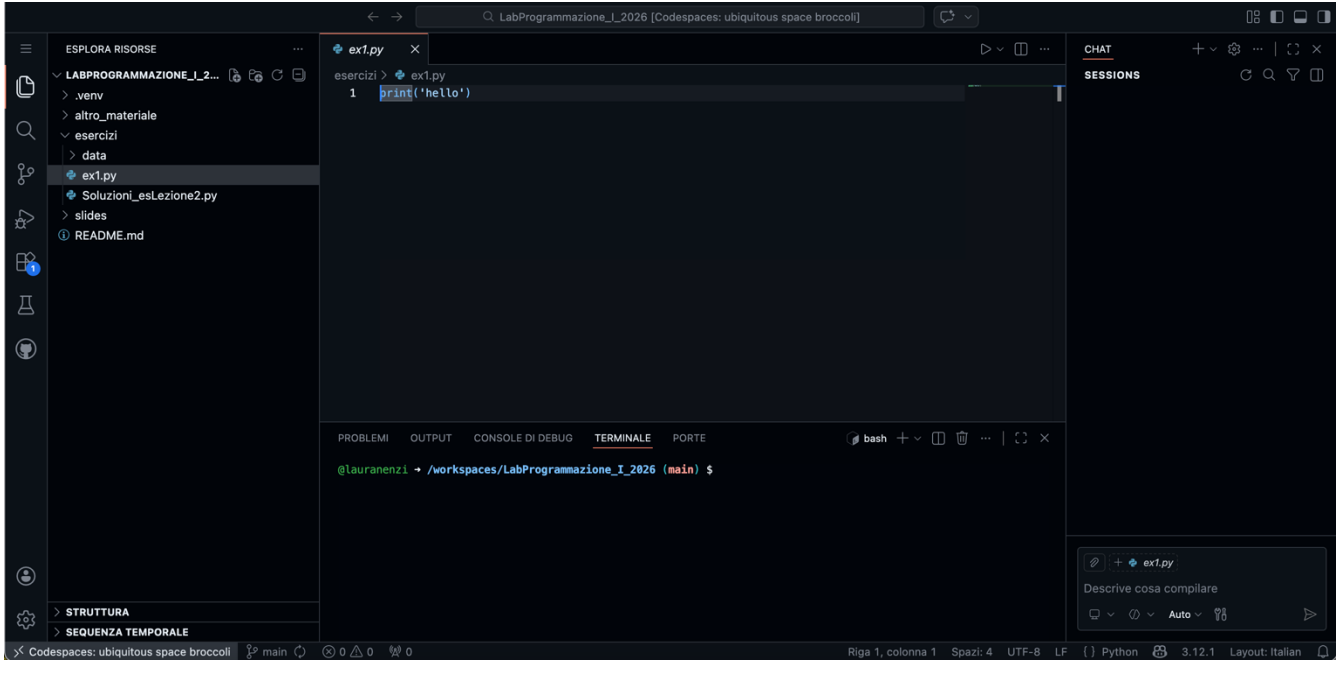
Codespaces

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



Codespaces

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



Primi comandi

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

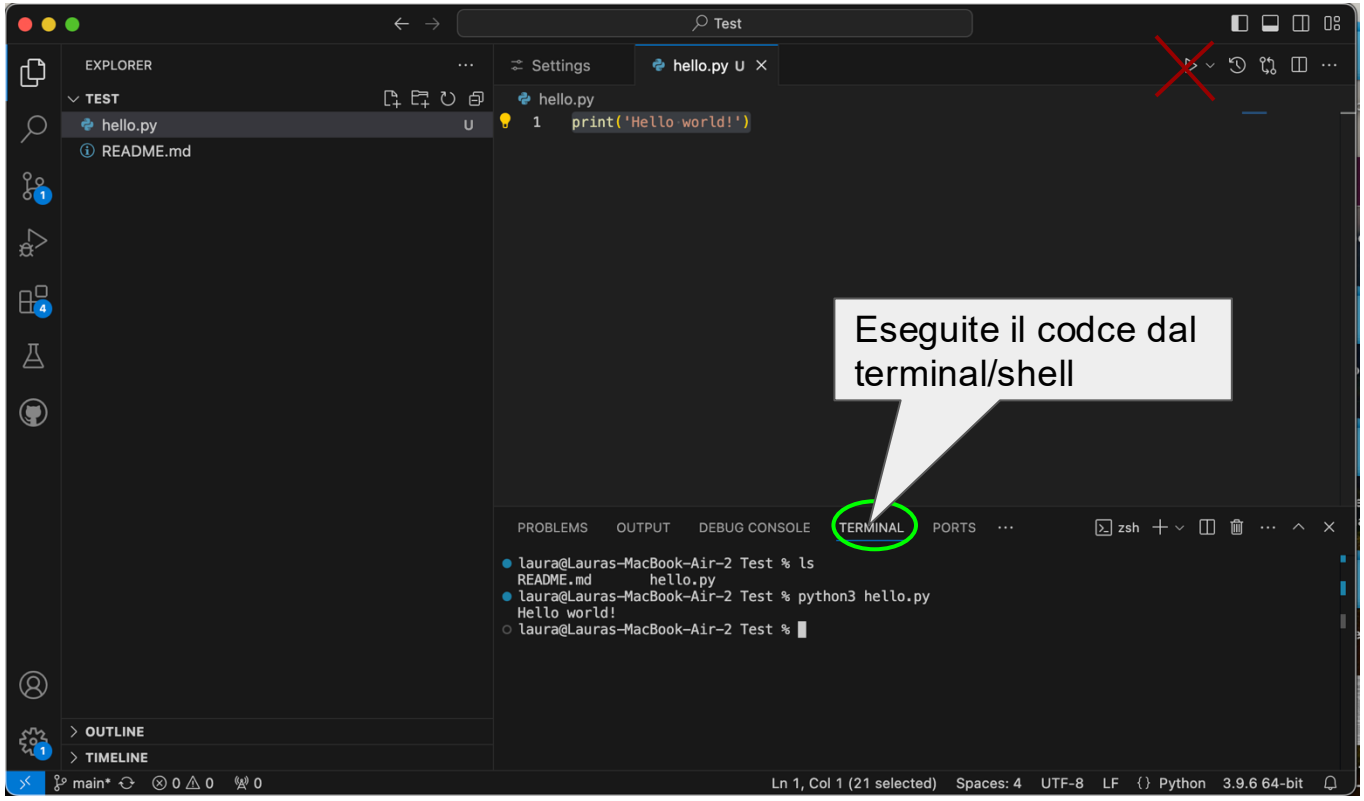
```
print('Hello world!')
```

Poi, eseguiamo lo script dalla shell:

```
python hello.py
```

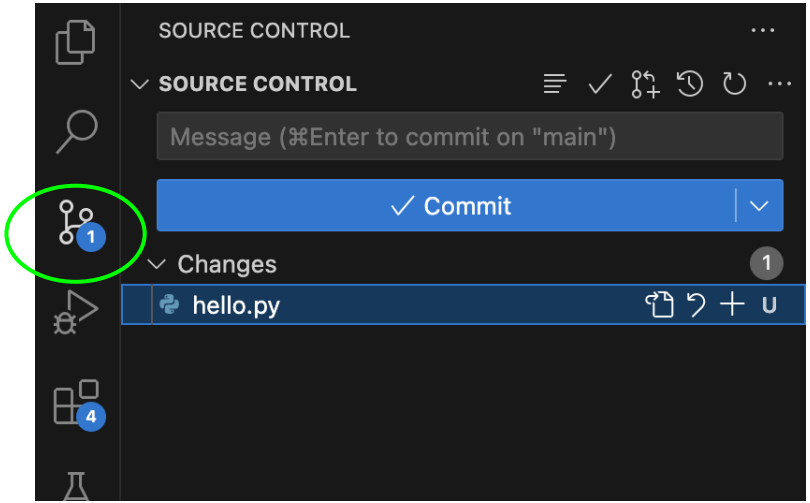
Assicuratevi di essere nella cartella dove c’è il file!!!

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.



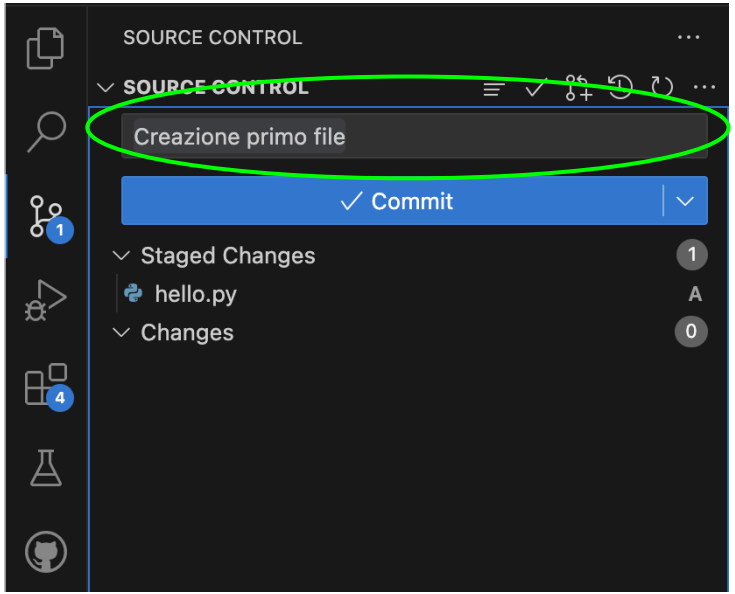
Staging

- 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.

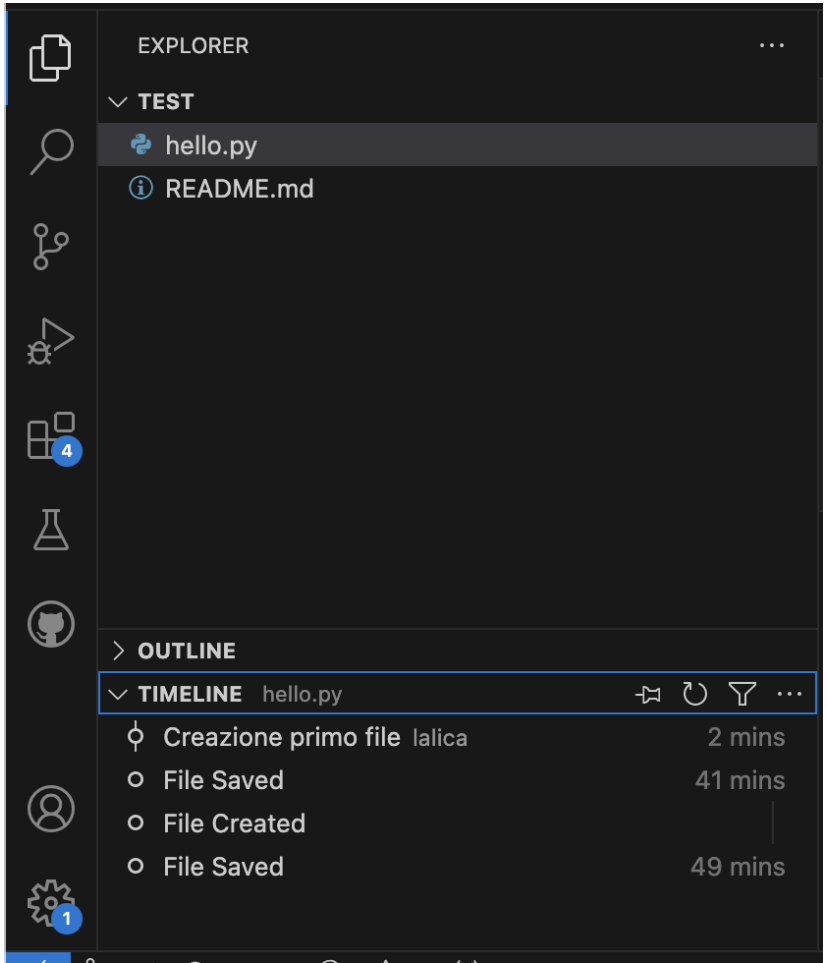
Commit

- 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.



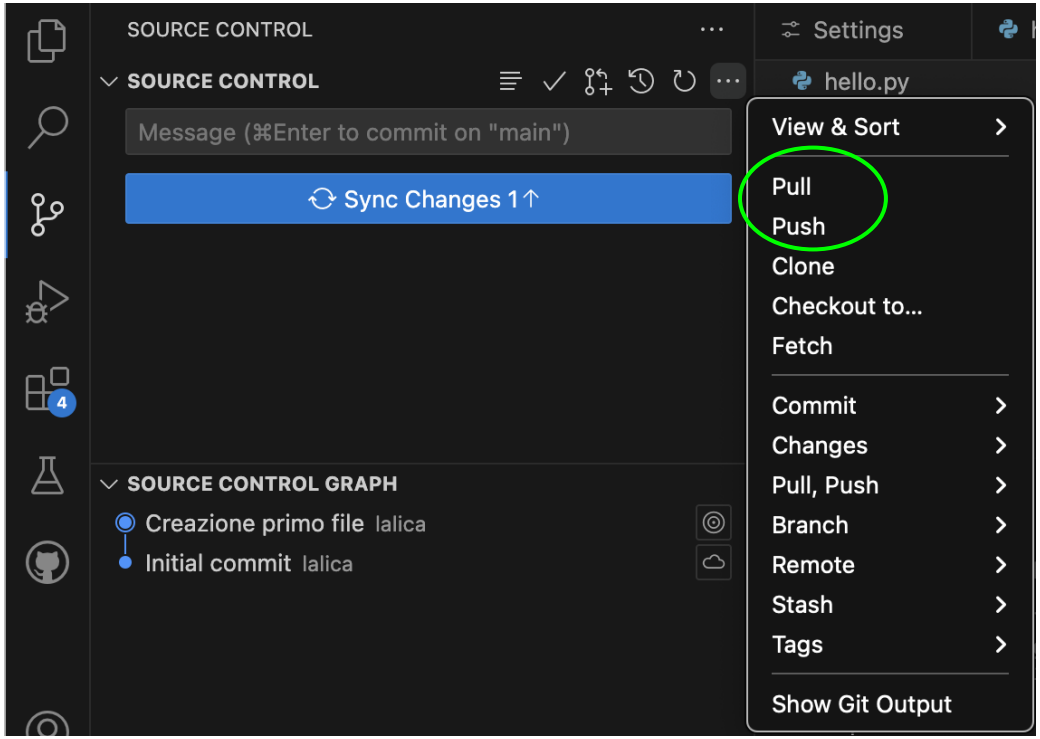
Timeline

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

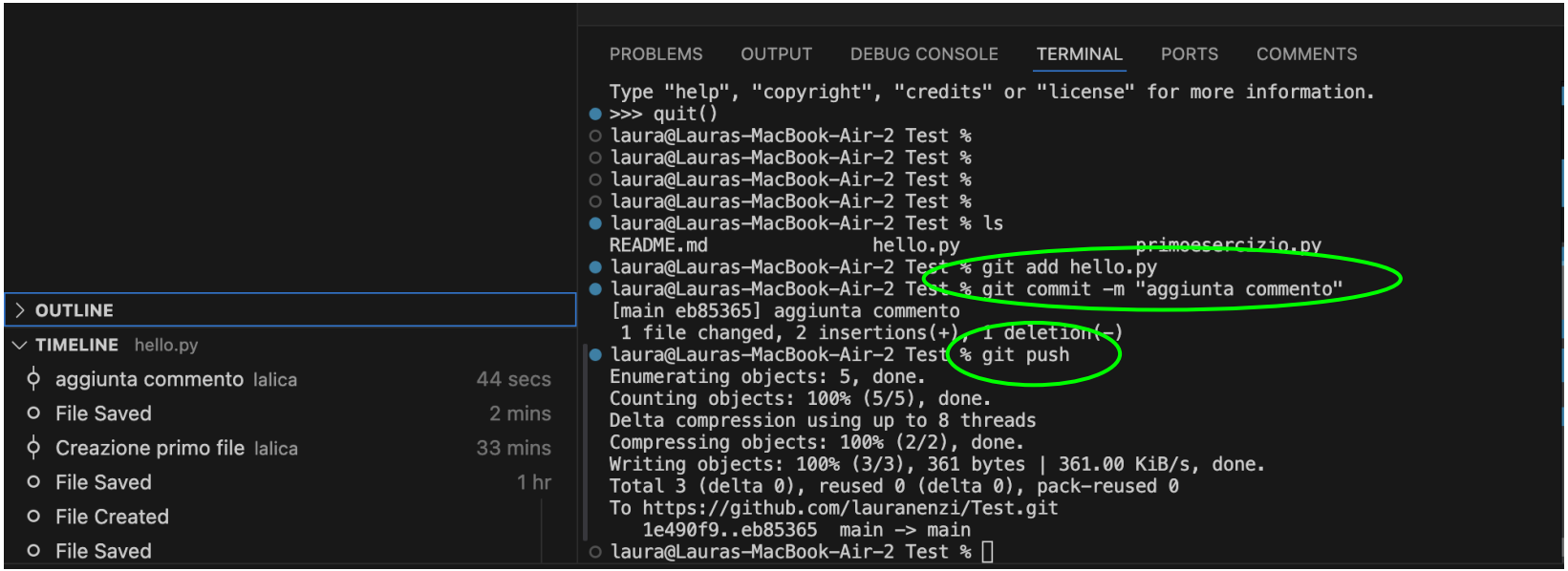


Push and Pull

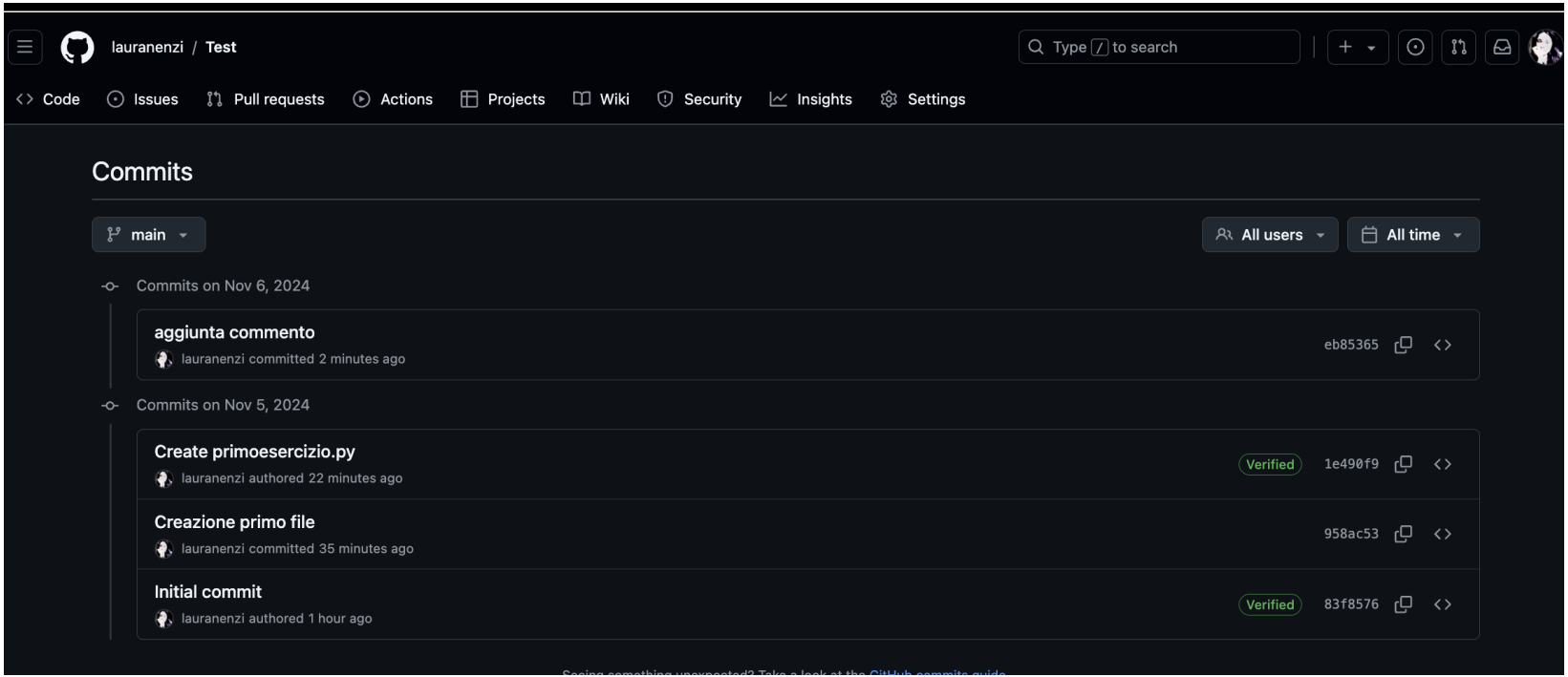
- Push carica le tue modifiche sul remoto.
- Pull scarica le modifiche dal remoto.
- È possibile accedere ai comandi Push e Pull dal menu Source Control.



Git da terminale



Cronologia su Github



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** *Pesonal Access Token* **generato e copiato prima (NON la password personale d'accesso).**