Welcome!

Hello, I'm Margherita

- 2nd DSDM cohort (graduated 2023)
- Since then working as a research assistant with Hannes
  - Institutional disruption
  - Data provision for organisations in Kenya
- Before that
  - Ashoka
  - Monitoring & evaluation
  - Spain & Jordan
  - Teach First & Discover Academy
  - Greece & India

Two truths and a lie

1. I am preparing for an ironman
2. I have performed music at a Parisian museum
3. I grew up across four countries

Two truths and a lie

On a piece of paper, write your name and two truths and a lie.

Session overview

1	Fri, Sep 6, 2024	**VSC, virtual environments**
2	Mon, Sep 9, 2024	**Version control**
3	Tue, Sep 10, 2024	**EDA & feature creation in python**
4	Thu, Sep 12, 2024	**EDA & feature creation in python (continued)**
5	Fri, Sep 13, 2024	**RStudio, importing and exploring data (EDA) in R**
6	Tue, Sep 17, 2024	**EDA live challenge**

Session 1:
VSC and virtual environments

DSDM Brushup Course - Coding - September 2024

Margherita Philipp

You will be able to

Session 1

- Explain libraries & virtual environments
- Set up virtual environments with conda or a manager of your choice
- Create and use requirements.txt files
- Explain the difference between .py vs .ipynb

Session 2

- Explain version control
- Pull from/ push to GitHub from the terminal

What is a “library”?

What is a “library”?

- structured **collection of functions and classes** that **can be imported** and used in other functions, classes or programs in order to **reduce the time** required to code.
- allow developers (and/or Data Scientists) to access pre-written frequently used pieces of code **without having to write them from scratch** every time they are needed.
- allow other developers (or themselves) to **focus on new functionalities** or custom usages of the predefined functions and classes.

An ideal library (and code in general): DRY

DRY

Don’t

Repeat

Yourself

"Every piece of knowledge must have a **single, unambiguous, authoritative representation** within a system". The principle has been formulated by Andy Hunt and Dave Thomas in their book *The Pragmatic Programmer*.

- Successful application of DRY principle: a **modification** of any single element of a system **does not require a change** in other logically unrelated elements.
- Also: elements that are logically related all **change predictably** and uniformly, and are thus kept in sync.

Library examples

We use many libraries to perform analysis and are highly dependent on other people's code.

Open Source libraries are commonly available in Github. Can you list examples?

Library examples

We use many libraries to perform analysis and are highly dependent on other people's code.

Open Source libraries are commonly available in Github. Examples:

- pandas
- numpy
- scikit-learn
- scipy
- datetime
- os
- matplotlib
- seaborn
- xgboost
- tensorflow
- pytorch
- nltk
- statsmodels
- pyspark
- flake8
- pytest
- jupyter
- regex

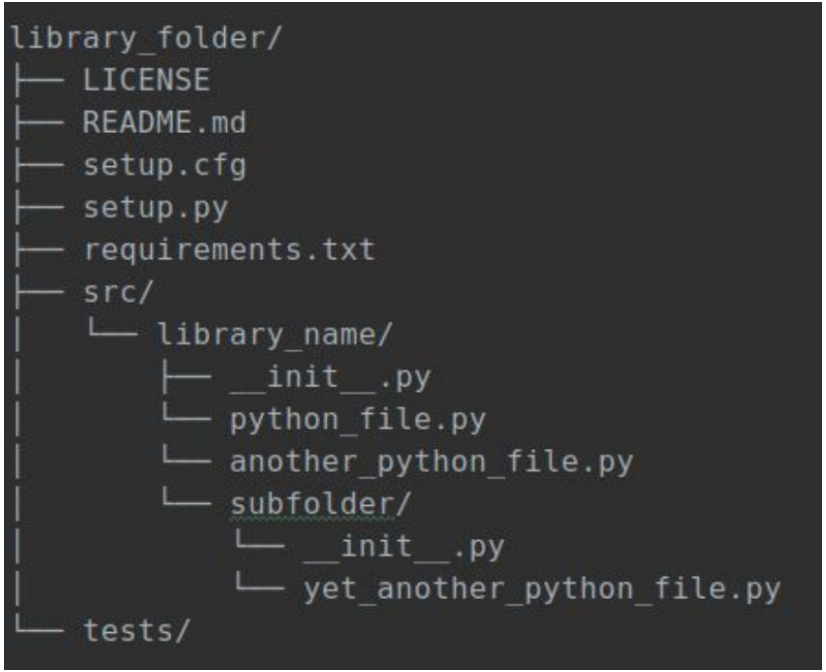
Installing libraries

To install a library from Pypi, we use: **pip install**

We can create our own libraries! To install locally, share with colleagues or even publish to Open Source (Pypi).

Extra: For development of our own libraries, there is a way to install the library and be able to modify it and use the current state of the library when importing it. It creates a “direct access” to the folder.

You will trial this with Roger :)



Under the hood of pip install

In Python, when we install a library, a copy of that library folder is stored in the site-packages folder of a Python interpreter in our computer.

Those libraries will be copied inside the path shown running “**which python**” (or which python3) in the terminal.

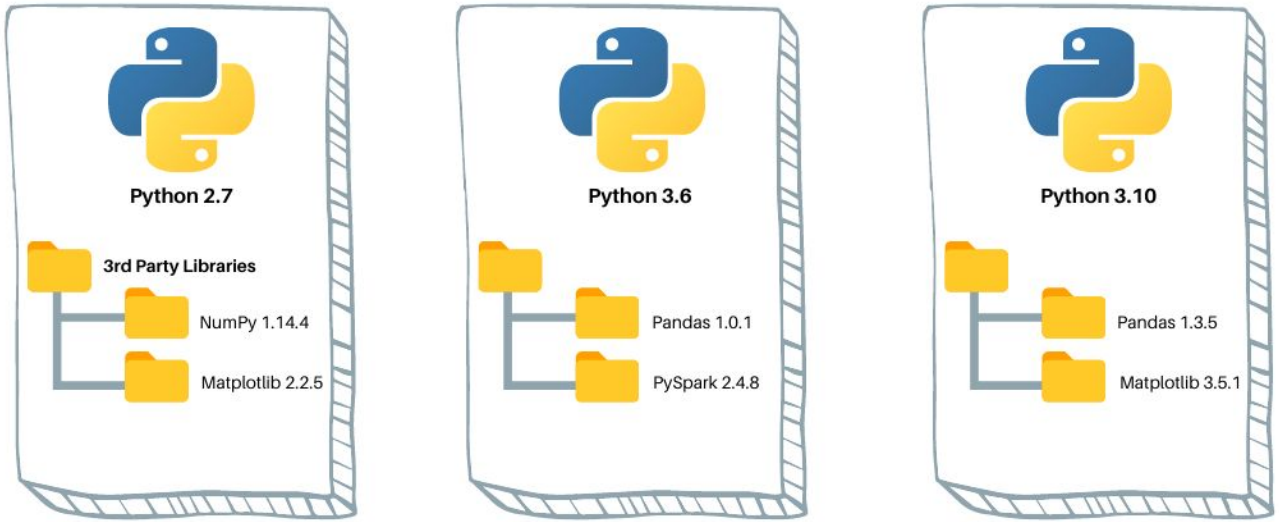
➜ Try this now.

If we use the same Python interpreter for all projects, all projects will use the same version of Python and the same version of the libraries. Many times that’s far from ideal.

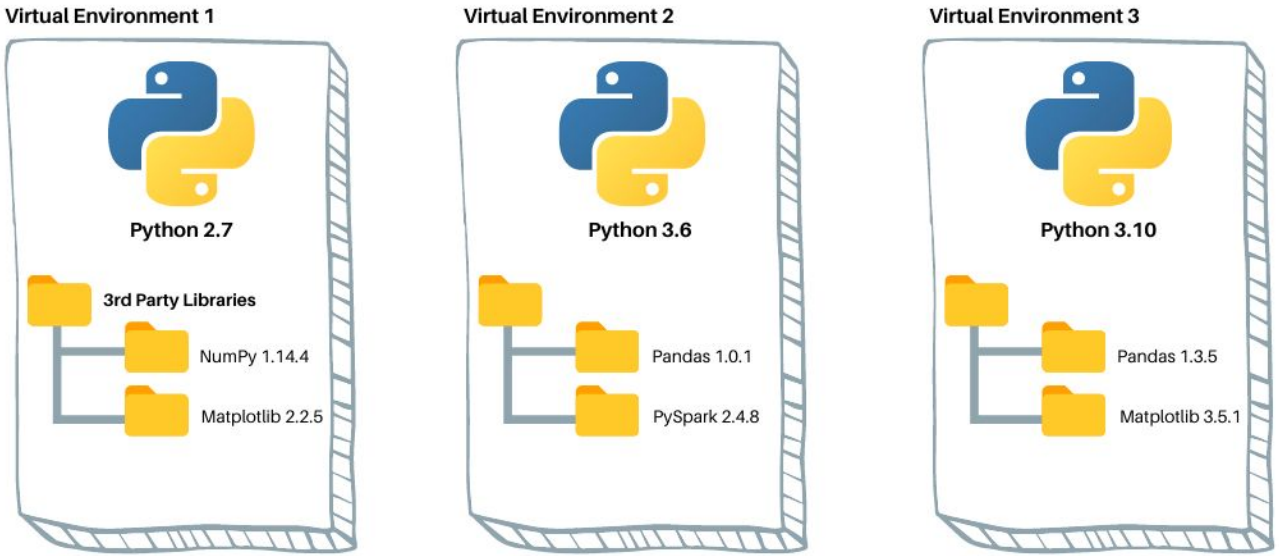
➜ Why?

When libraries create problems

Imagine you are working on three different projects that need different library versions. Do you have to upgrade and downgrade your libraries each time?



Solution: virtual environments



Virtual environment options

What virtual environments managers are you aware of or familiar with?

Virtual environment options

You can create and handle virtual environments (virtual envs) in different ways:

- **venv**: It is a module in the default Python system to manage virtual envs. It does not allow multiple Python versions.
- **pyenv**: It isolates Python versions, not a virtual env tool. used to install Python versions.
- **virtualenv**: It is the classical way to manage virtual envs. Might be a bit old fashioned.
- **virtualenvwrapper**: It is an extension to virtualenv. More comfortable than virtualenv.
- **pipenv**: A new way to manage virtual envs in a project oriented manner. Getting traction.
- **conda**: Part of Anaconda distribution and go tool if you work using Anaconda. Simple to use but occupies a lot of space (up to 10 times more than others).

Demo

1. Folder
2. VSC
3. Terminal location and conda env
4. Install pandas
5. Create requirements.txt

Venvs with conda

1	Create an environment	`conda create -n venvname`
2	Get list of environments	`conda info --envs`
3	Get list of all packages installed in an environment	`conda list (-n venvname)` `pip3 list`
4	Activate an environment	`conda activate venvname`
5	Install something within an activated environment	`conda install package -v(e.g.2.2.3)` `pip3 install package -v(e.g.2.2.3)`
6	Deactivate an environment	`conda deactivate venvname`
7	Create a requirements file: stores all of the packages and dependencies installed in current environment	`conda list -e > requirements.txt` `pip3 freeze > requirements.txt`
8	Install from a requirements file	`conda install --file requirements.txt` `pip3 install -r requirements.txt`
9	Remove a conda environment	`conda remove -n venvname --all`

Command line

- cd .. - change directory: up one directory
- ls - list: all items in the current folder
- cd Doc ➜ - go into folder starting with Doc (end with tab)
- mkdir *dirname* - make directory: called trial1
- rmdir *dirname* - remove directory

You'll do more of this with Roger...

15m Task: library management with conda

1. Create a folder called brushup_files
    a. Extra: do it via the command line
2. Open that location in VSC
3. From terminal in VSC, create a virtual environment called brushup_env
4. Install a version of pandas (latest is 2.2.2)
5. Create a requirements.txt file. Is it saved in the brushup_files folder?
6. Are there any direct paths in requirements.txt? Replace them with package versions.
7. Close this environment and create another called “trial”
8. Install requirements.txt file
9. Manually downgrade pandas in requirements.txt and try to install it now.
10. Done?
    a. Upload a screenshot of the your terminal showing that you are in the brushup_file folder with the trial active and the error message visible
    b. See if someone might benefit from your help.

Demo and “solutions”

Below are all the terminal commands up to installing the requirements.
The packages for which I needed to replace file locations with versions were:
- altgraph==0.17.2
- future==0.18.2
- macholib==1.15.2
- six==1.15.0

margheritaphilipp@Margheritas-MacBook-Air brushup % conda info --envs
margheritaphilipp@Margheritas-MacBook-Air brushup % conda create -n brushup_env
margheritaphilipp@Margheritas-MacBook-Air brushup % conda activate brushup_env
(brushup_env) margheritaphilipp@Margheritas-MacBook-Air brushup % pip3 install pandas=2.2.0
(brushup_env) margheritaphilipp@Margheritas-MacBook-Air brushup % pip3 list
(brushup_env) margheritaphilipp@Margheritas-MacBook-Air brushup % conda deactivate
margheritaphilipp@Margheritas-MacBook-Air brushup % conda create -n trial
margheritaphilipp@Margheritas-MacBook-Air brushup % conda activate trial
(trial) margheritaphilipp@Margheritas-MacBook-Air brushup % pip3 install -r requirements.txt

Session 2:
Version control systems

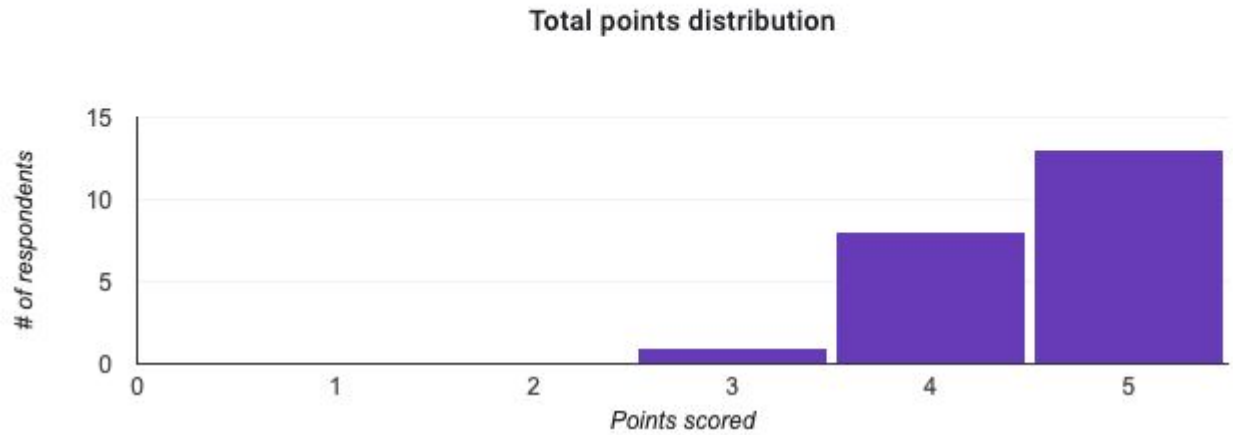
(VCS ≠ VSC)

DSDM Brushup Course - Coding - September 2024

Margherita Philipp

Quiz results

Average	Median	Range
4.55 / 5 points	5 / 5 points	3 - 5 points



Questions from session 1

1. I've done all the steps you told us and in the last one I was able to manually downgrade pandas to version 2.2.1 without getting any error. I don't know if that means I made a mistake or if it's alright.
2. From what I understood in class, we should install Pandas using the pip command instead of Conda. So, to create the requirements.txt, we also use pip instead of Conda?

   P.D (When created by Conda, the directory gets loaded with folders.)

3. When I try to install Pandas, it shows me that it is already installed, so I guess that is for the installation that we were asked to do before the course started, but we also installed a bunch of other stuff from the prompt of our OS so every time that I create a requirements file, it will be included either the way I needed it for that specific project or not. Is there a way to fix that?
4. The requirements.txt, when created, is located outside the folder of my virtual environment. Is it supposed to be like that? Shouldn't I have a requirements.txt for every environment?
5. The requirements.txt created by Pip and by Conda have different content. Why is that, and which one should I use?

ex_5.py 1, M | ex_1.py | README.md U | requirements.txt brushup_files | requirements.txt brushup U

brushup_files > requirements.txt

```
contourpy==1.3.0
cycler==0.12.1
fonttools==4.53.1
kiwisolver==1.4.5
matplotlib==3.9.2
mpmath==1.3.0
numpy==2.1.0
packaging==24.1
pandas==2.2.0
pillow==10.4.0
pyparsing==3.1.4
python-dateutil==2.9.0.post0
pytz==2024.1
six==1.16.0
sympy==1.13.2
tzdata==2024.1
```

EXPLORER

DSDM
- brushup
- brushup_files
  - requirements.txt
- docs
- venv
- requirements.txt

PROBLEMS 1 | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | COMMENTS

zsh
zsh brushup_files

```
  Using cached contourpy-1.3.0-cp312-cp312-macosx_11_0_arm64.whl.metadata (5.4 kB)
ERROR: Cannot install -r requirements.txt (line 1), -r requirements.txt (line 5), -r requirements.txt (line 9) and numpy==2.1.0 because these
 package versions have conflicting dependencies.

The conflict is caused by:
    The user requested numpy==2.1.0
    contourpy 1.3.0 depends on numpy>=1.23
    matplotlib 3.9.2 depends on numpy>=1.23
    pandas 2.2.0 depends on numpy<2 and >=1.26.0; python_version >= "3.12"

To fix this you could try to:
1. loosen the range of package versions you've specified
2. remove package versions to allow pip attempt to solve the dependency conflict

ERROR: ResolutionImpossible: for help visit https://pip.pypa.io/en/latest/topics/dependency-resolution/#dealing-with-dependency-conflicts

[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: pip install --upgrade pip
(trial) → brushup_files git:(main) ✗ pip list
Package         Version
--------------- ---------
contourpy       1.3.0
cycler          0.12.1
fonttools       4.53.1
kiwisolver      1.4.5
matplotlib      3.9.2
mpmath          1.3.0
numpy           2.1.0
packaging       24.1
pillow          10.4.0
```

OUTLINE

```
Collecting pandas==2.2.1 (from -r requirements.txt (line 4))
  Using cached pandas-2.2.1-cp312-cp312-macosx_11_0_arm64.whl.metadata (19 kB)
Requirement already satisfied: python-dateutil==2.9.0.post0 in ./trial/lib/python3.12/site-packages (from -r requirements
txt (line 5)) (2.9.0.post0)
Requirement already satisfied: pytz==2024.1 in ./trial/lib/python3.12/site-packages (from -r requirements.txt (line 6)) (
024.1)
Requirement already satisfied: six==1.16.0 in ./trial/lib/python3.12/site-packages (from -r requirements.txt (line 7)) (1
16.0)
Requirement already satisfied: soupsieve==2.6 in ./trial/lib/python3.12/site-packages (from -r requirements.txt (line 8))
(2.6)
Requirement already satisfied: tzdata==2024.1 in ./trial/lib/python3.12/site-packages (from -r requirements.txt (line 9))
(2024.1)
INFO: pip is looking at multiple versions of pandas to determine which version is compatible with other requirements. Thi
 could take a while.
ERROR: Cannot install -r requirements.txt (line 4) and numpy==2.1.1 because these package versions have conflicting depen
encies.

The conflict is caused by:
    The user requested numpy==2.1.1
    pandas 2.2.1 depends on numpy<2 and >=1.26.0; python_version >= "3.12"

To fix this you could try to:
1. loosen the range of package versions you've specified
2. remove package versions to allow pip attempt to solve the dependency conflict

ERROR: ResolutionImpossible: for help visit https://pip.pypa.io/en/latest/topics/dependency-resolution/#dealing-with-depe
dency-conflicts
```

~/Desktop/brushup_file

File Edit Selection View Go Run Terminal Help

brushup_files

EXPLORER

BRUSHUP_FILES

requirements.txt

requirements.txt

```
Bottleneck ==1.3.7
mkl-service==2.4.0
mkl_fft ==1.3.8
mkl_random ==1.2.2
numexpr ==2.8.4
numpy ==1.25.2
pandas ==1.3.0
python-dateutil ==2.8.2
pytz ==2024.1
setuptools==72.1.0
six ==1.16.0
tzdata ==2024a
wheel==0.43.0
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: \ warning  libmamba Added empty dependency for problem type SOLVER_RULE_UPDATE
failed

LibMambaUnsatisfiableError: Encountered problems while solving:
  - package pandas-1.3.0-py37hd77b12b_0 requires python >=3.7.1,<3.8.0a0, but none of the providers can be installed

Could not solve for environment specs
The following packages are incompatible
├─ pandas 1.3.0  is installable with the potential options
│  ├─ pandas 1.3.0 would require
│  │  └─ python >=3.7.1,<3.8.0a0 , which can be installed;
│  ├─ pandas 1.3.0 would require
│  │  └─ python >=3.8,<3.9.0a0 , which can be installed;
│  └─ pandas 1.3.0 would require
│     └─ python >=3.9,<3.10.0a0 , which can be installed;
└─ pin-1 is not installable because it requires
   └─ python 3.11.* , which conflicts with any installable versions previously reported.
```

OUTLINE

You will be able to

Session 1

- Explain libraries & virtual environments
- Set up virtual environments with conda or a manager of your choice
- Create and use requirements.txt files
- Explain the difference between .py vs .ipynb

Session 2

- Explain version control
- Pull from/ push to GitHub from the terminal

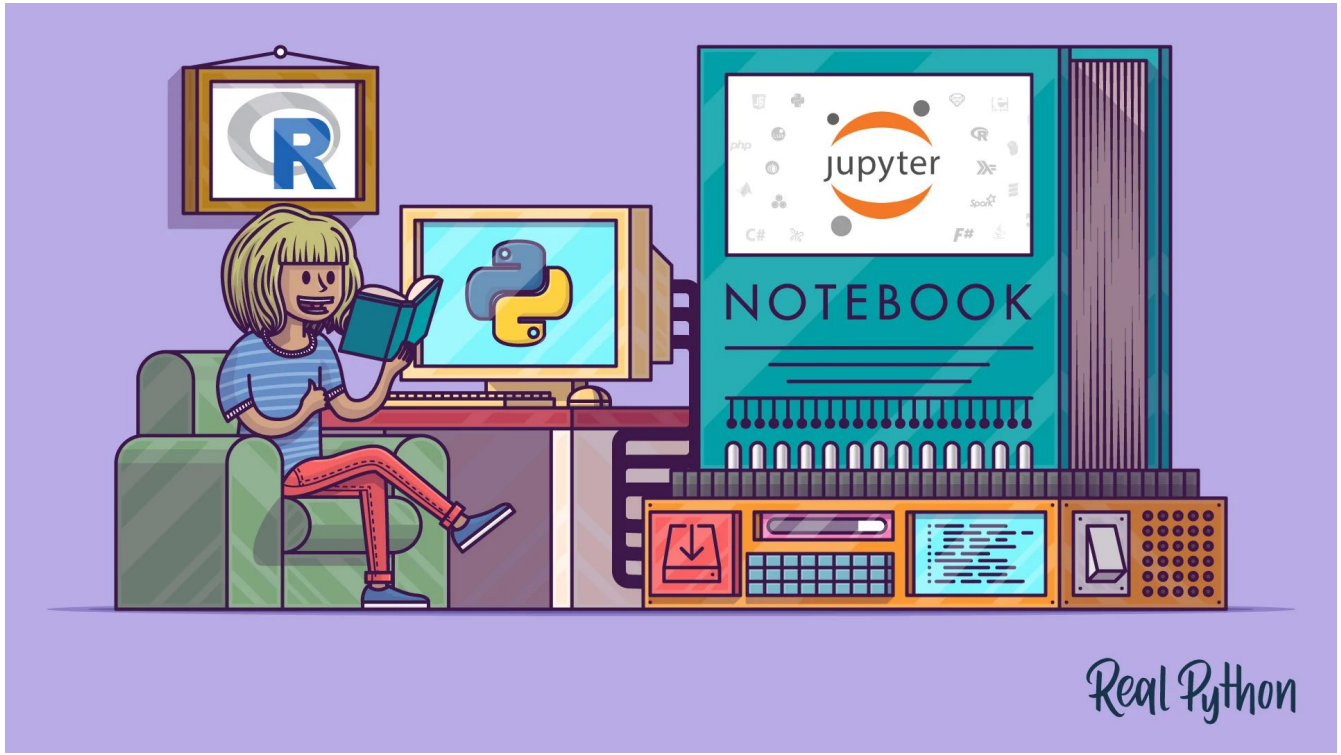
Truths and lies interlude

Alejandro	Delgado Tello
Aleksandr	Smolin
Anastasiia	Chernavskaia
Angad Singh	Sahota
Blanca	Jimenez
Deepak	Malik
Denis	Shadrin
Enzo	Infantes
Ferran	Boada Bergadà
Hannes	Schiemann
Julián	Romero
Lucia	Sauer
Maria	Simakova Mariukha

Maria Jose	Aleman Hernandez
Marta	Sala
Matias	Borrell
Moritz	Peist
Nicolas	Rauth
Noemi	Lucchi
Pablo	Fernández
Simon	Vellin
Soledad	Monge
Tarang	Kadyan
Viktoria	Gagua
Wei	Sun

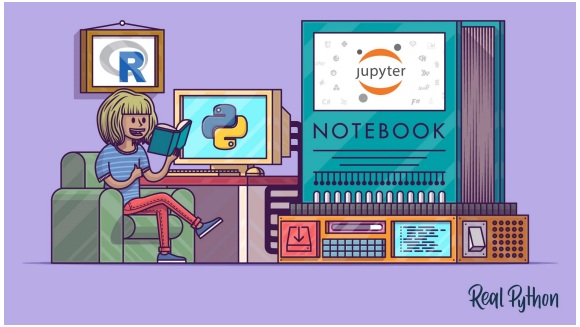
What's the difference? .py vs .ipynb

-



.py vs .ipynb

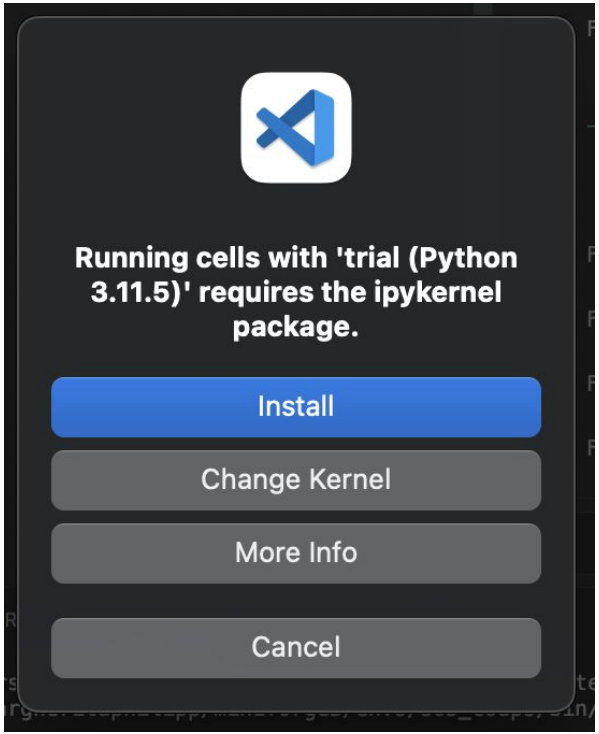
- .py file is a regular Python programming file (a plain text file that contains just code).
- .ipynb file is a Python notebook file that contains notebook code alongside execution results and other internal settings.



10m Task: library management with conda

1. In your brushup folder, create file called **temperature.py**
2. Select the kernel environment to be brushup_env (bottom right of VSC)
3. Create a function that converts celsius to fahrenheit
   - ❏ def c_to_f(celsius):
   - ❏ fahrenheit = celsius * 9/5 + 32
   - ❏ return fahrenheit
   - ❏ cel=15
   - ❏ print(c_to_f(cel))
4. Save the file.
5. Run the file.
   - ❏ via VSC “run”
   - ❏ in terminal: python3 temperature.py
6. Now create a **temp.ipynb** that contains the same function.
7. Select the environment (top right of VSC)
8. Can you run it immediately? What is missing?
9. Done?
   - ❏ Expand your function
   - ❏ E.g. can you ensure only number are accepted?
   - ❏ Can you allow for a list of inputs that are all converted?
   - ❏ Can you make it print a statement about “x degrees celsius is equal to y degrees fahrenheit”?

What you need to run notebooks




```
(trial) margheritaphilipp@Margheritas-MacBook-Air trial1 % pip install ipykernel
```

Motivation for version control

1. How can multiple developers **collaborate** on the same project without overwriting each other's work?
2. How can you **track changes and revert** to a previous version if something goes wrong?
3. How can you **safely experiment** with new features without affecting the main codebase?

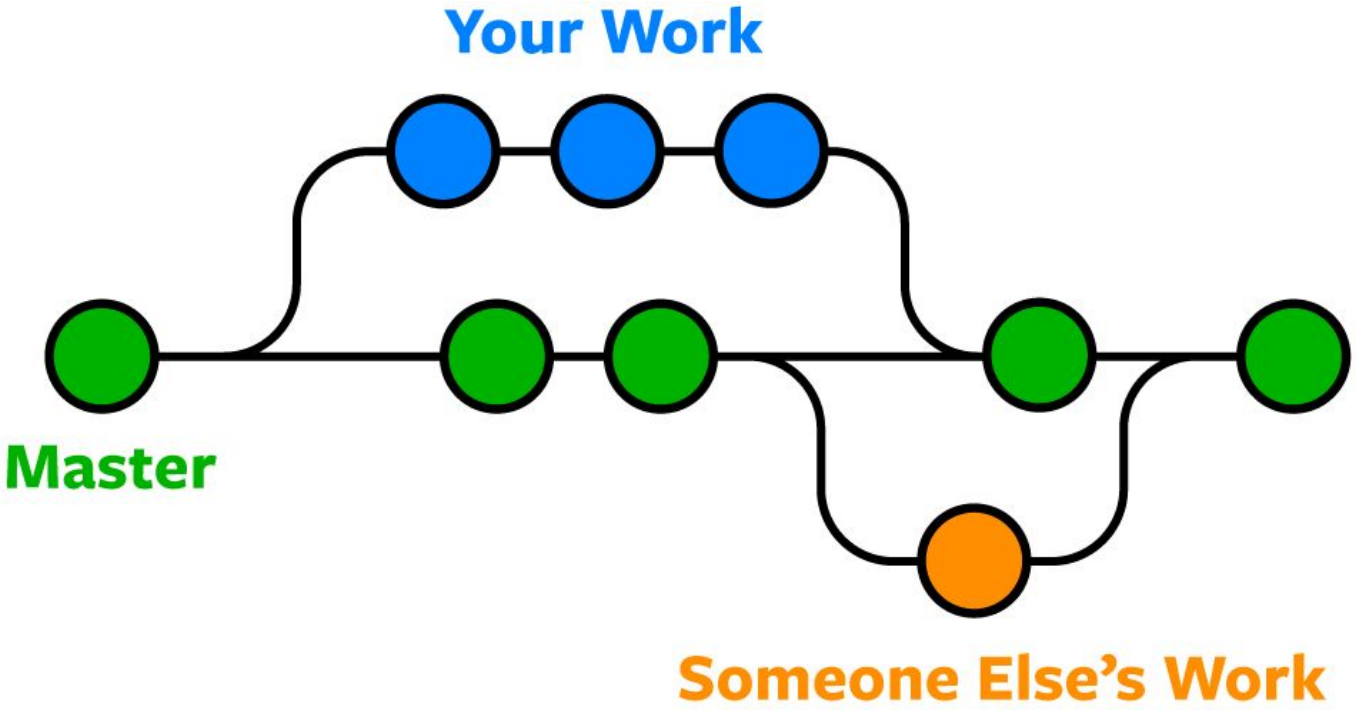
Why version control?

- Allow multiple **collaborators** to work on the same project **remotely** by having a different version in their computer.
- Allow **merging the changes** done by multiple collaborators into a single state.
- Reduce possibilities of **errors and conflicts** meanwhile project development through traceability to every small change.
- Allow **revision and validation** of code modifications before merging them.
- Allow **recovery** in case of any disaster or mistake.
- Informs us about **Who, What, When, Why** changes have been made.

Version control with git

- Git is a is a **free and open source** distributed version control system.
- A version control system (VCS) is a software that helps in recording changes made to files by keeping a **track record** of modifications done to the code.
- With a VCS, we can keep track of the history of a file without saving all the different versions in a way that we can reconstruct any previous state.

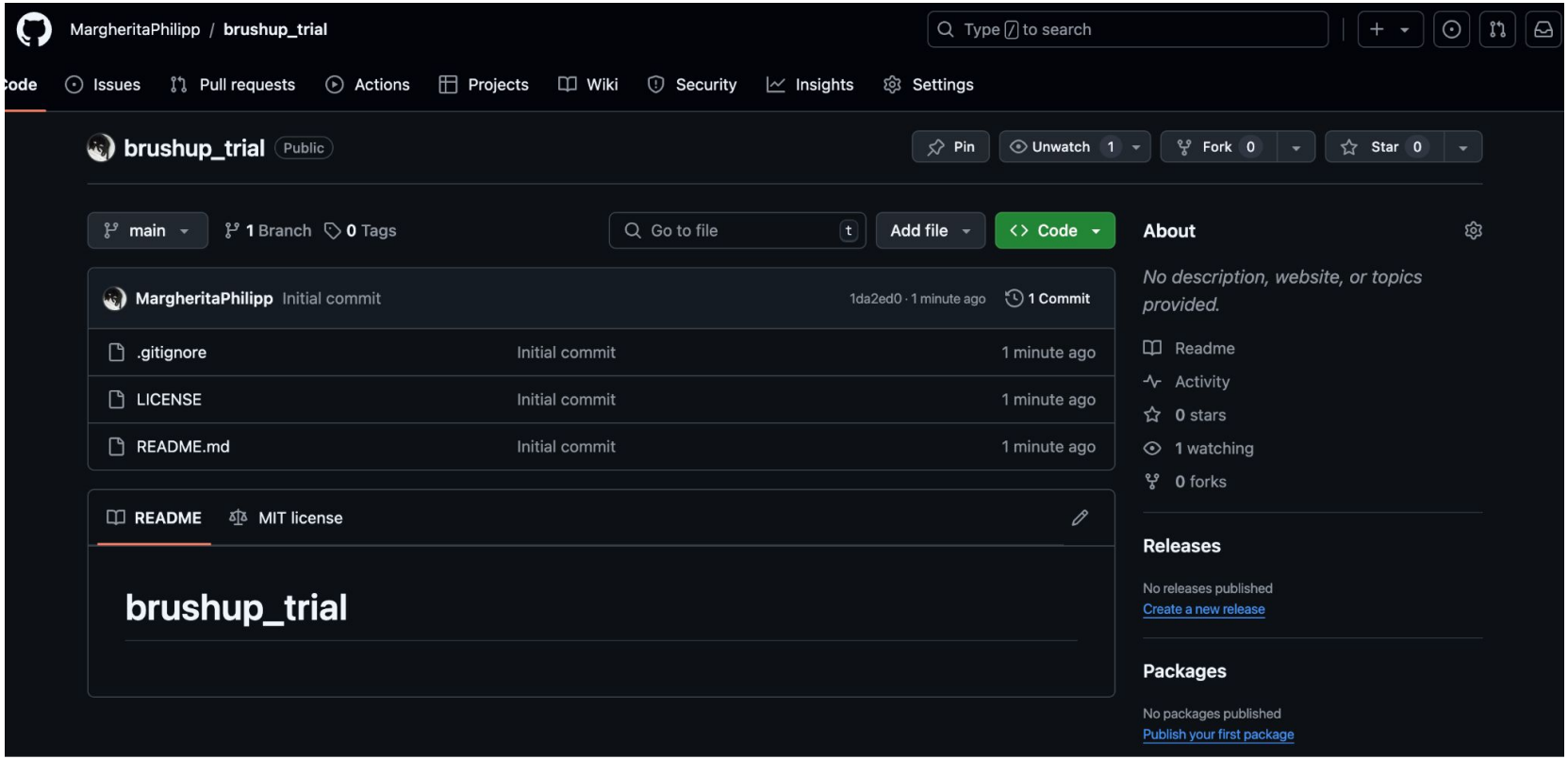
Git magic with branches



10m Task: github and git

1. If you don't have one: create a github account (and if needed, install/ enable git: https://docs.github.com/en/get-started/getting-started-with-git/set-up-git)
2. Create a repository call brusup_git
   - ❏ Public
   - ❏ With readme
   - ❏ Gitignore template "python"
   - ❏ License MIT
3. Done?
   - ❏ Can you explore/ explain: What is the difference between clicking on the .gitignore vs the "initial commit" in the same line?
   - ❏ Does anyone need your help?
   - ❏ What is the readme file for?
   - ❏ What does gitignore do?
   - ❏ What is the maximum file size you can upload to github?
   - ❏ Might anyone benefit from your help?

Git magic with branches



Version control lingo

Version control systems components:

1. A **repository**:

   It can be thought of as a database of changes. It contains all the edits and historical versions (snapshots) of the project.

2. **Copy of Work** (sometimes called as checkout): It is the personal copy of all the files in a project.

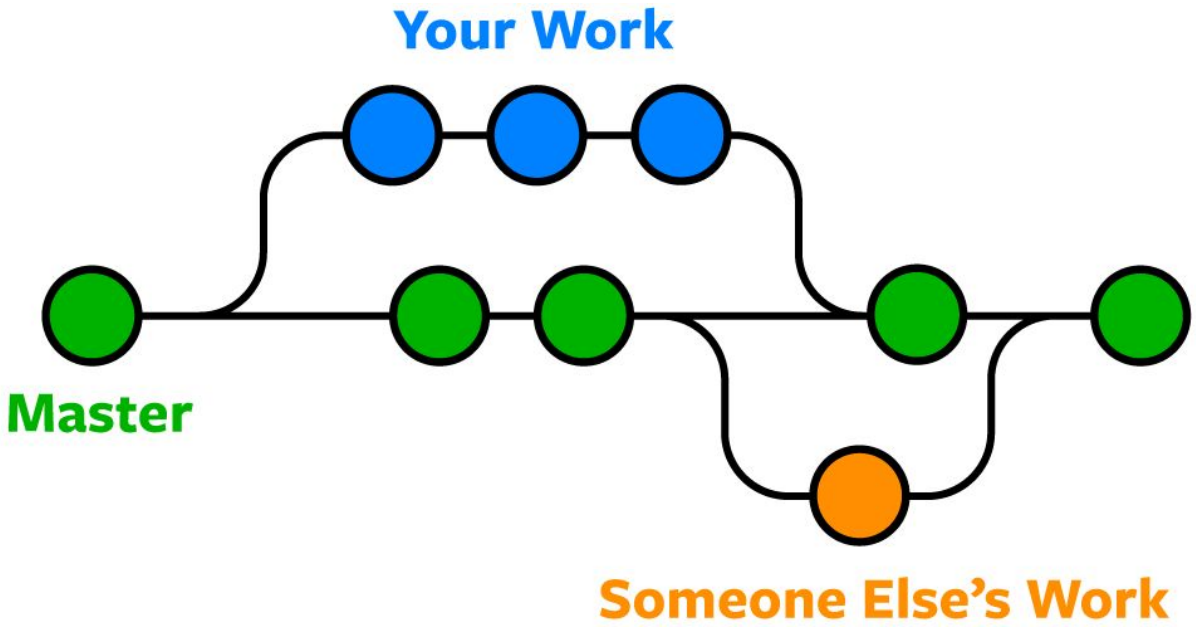
   You can edit to this copy, without affecting the work of others and you can finally commit your changes to a repository when you are done making your changes.

Git cheat sheet 1

1	Initialise git	`git init`
2	Link to repository	`git remote add origin git@github.com:username/reponame`
3	Show the state of modified files (untracked, staged, unstaged)	`git status`
4	**Pull** changes done in a specific branch from the remote repository	`git pull origin <branch>`
5	**Create** a branch and move to it	`git checkout -b <name_branch>`
6	**Stage** modifications of a file/directory to prepare them to commit	`git add <file>`
7	**Commit** (save) the staged changes with a message	`git commit -m "<message>"`
8	Change state of directory to a certain branch	`git checkout <branch>`
9	**Push** committed changes to specific branch on remote repository	`git push origin <branch>`
10	Unlink repository	`git remote remove origin`

```
git init

git status

git remote add origin git@github.com:margheritaphilipp/brushup_trial

git status

git pull origin main

git status

git add print_hello.py

git status

git commit - "adding py file"

git status

git checkout -b mph

git status

git push origin mph
```

Where does what happen?



1. Pull origin main
2. Push origin branch
3. (Stage and) Commit

Extra:

What "nono" is being illustrated here (going against best practice)?

What could "pull origin Elses_Work" look like in the graphic?

May require setting up a Secure SHell (SSH)

1. Check if you have an SSH key: **`ls -al ~/.ssh`**

   Look for files like *id_rsa* or *id_ed25519 [part of error message]*. If they exist, proceed to step 2. If they don't, go to step 3.

2. Add your SSH key to the SSH agent:

   Start the SSH agent in the background: **`eval "$(ssh-agent -s)"`**

   Add your SSH private key to the agent: `ssh-add ~/.ssh/id_rsa`

3. Generate a new SSH key (if you don't have one): **`ssh-keygen -t ed25519 -C "your_email@example.com"`**

   Press Enter to accept the default file location, and then choose whether to set a **passphrase**.

   If you're using an older system that doesn't support ed25519, you can generate an RSA key instead: `ssh-keygen -t rsa -b 4096 -C "your_email@example.com"`

4. Add the SSH key to GitHub: **`cat ~/.ssh/id_ed25519.pub`**

   Copy the result into GitHub under Settings > SSH and GPG keys > New SSH key ➡ Paste the key into the "Key" field and give it a recognizable title.

5. Check the connection: **`ssh -T git@github.com`**
6. Try pulling again: **`git pull origin main`**

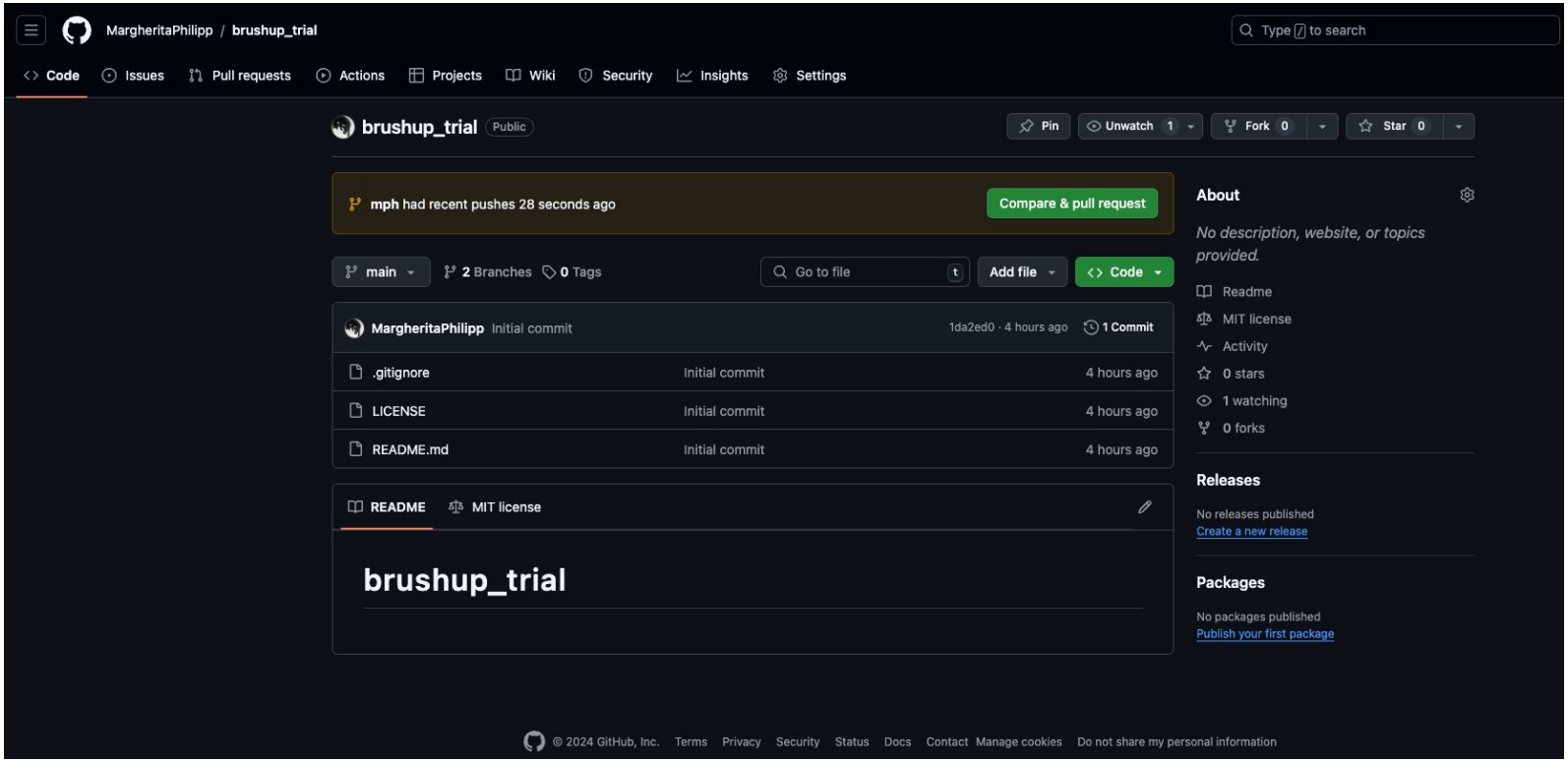
15m task: start pulling and pushing

1. Initialise git
2. If needed, set up Secure SHell (SSH)
3. Pull origin from main branch
4. Create a branch
5. Add the files temperature.py and temp.ipynb
6. Commit with a message
7. Push your branch
8. Done?
   - ❑ Submit a screenshot of your github
   - ❑ How can you now create a pull request?
   - ❑ What might be best practice here?
   - ❑ Can you add a collaborator to your github?
   - ❑ Do you know how .py and .ipynb behave differently when collaborating via github?
   - ❑ Can you add a data folder to your local folder that does not get synced to the repository?
   - ❑ What git commands were not included in the cheatsheet?
   - ❑ Might anyone benefit from your help?

Git cheat sheet 2

1	View the history of commits in a repository	`git log`
2	Undo a commit that has already been pushed to a shared repository	`git revert`
3	Merges changes from one branch to another without creating a merge commit (best to stay away from this)	`git rebase`
4	Downloads new data but doesn't modify working files	`git fetch`
5	Create a copy of an existing Git repository	`git clone <repo> <directory>` (https://github.com/username/repository.git)
6	git stash	`git stash`
		?

Example screenshot



Not a fan of git in terminal?

Best practices

- Make incremental, small changes
- Keep commits atomic
- Develop using branches
- Write descriptive commit messages
- Obtain feedback through code reviews
- Don’t alter published history

Truths and lies interlude

Alejandro	Delgado Tello
Aleksandr	Smolin
Anastasiia	Chernavskaia
Angad Singh	Sahota
Blanca	Jimenez
Deepak	Malik
Denis	Shadrin
Enzo	Infantes
Ferran	Boada Bergadà
Hannes	Schiemann
Julián	Romero
Lucia	Sauer
Maria	Simakova Mariukha

Maria Jose	Aleman Hernandez
Marta	Sala
Matias	Borrell
Moritz	Peist
Nicolas	Rauth
Noemi	Lucchi
Pablo	Fernández
Simon	Vellin
Soledad	Monge
Tarang	Kadyan
Viktoria	Gagua
Wei	Sun