esade

Advanced Python

Class 5

Pepe Bonet Giner

17th January 2023

Index Class 5

Topic 1: Recap Class 4

Topic 2: GitHub II. CD/CI

Topic 3: Makefile

Topic 4: Multiprocessing

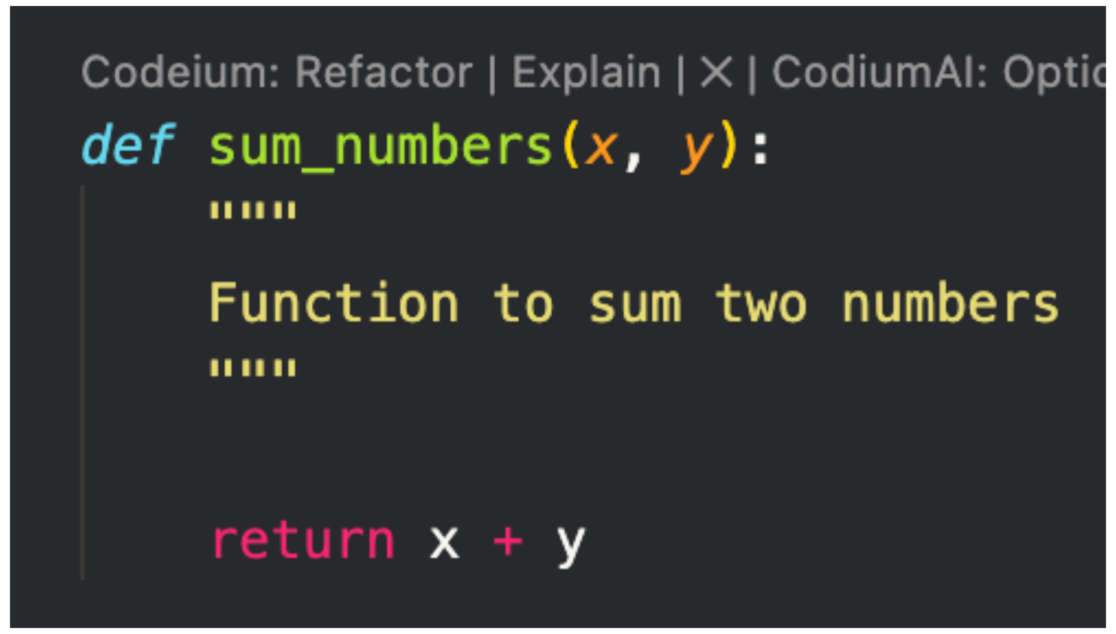
esade

Topic 1: Recap Class 4

Code Testing (Pytest & Unittests)

Testing in Python refers to the process of systematically checking and validating that individual units of source code, such as **functions** or methods, work correctly.

A function to test

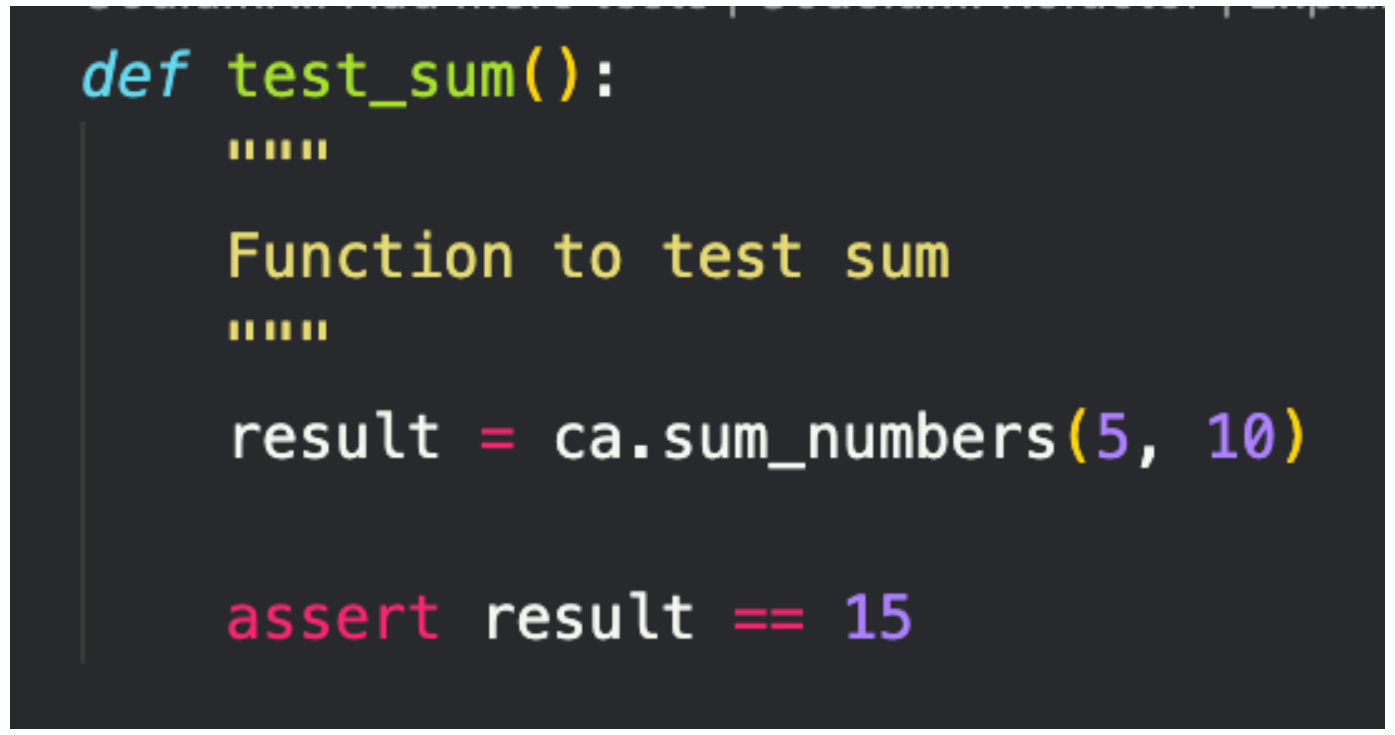


Codeium: Refactor | Explain | X | CodiumAI: Optic

```python
def sum_numbers(x, y):
    """
    Function to sum two numbers
    """

    return x + y
```

A test for the function

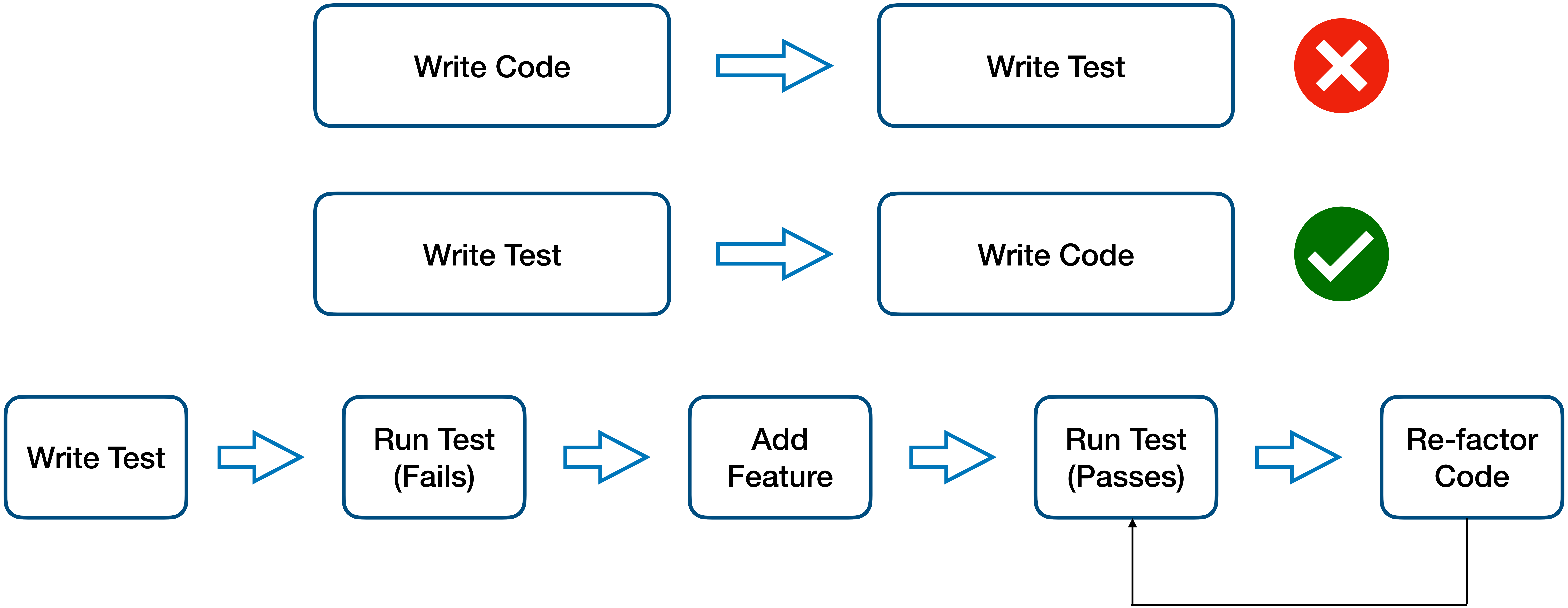


```python
def test_sum():
    """
    Function to test sum
    """
    result = ca.sum_numbers(5, 10)

    assert result == 15
```

Test Driven Development (TDD)

- Development Practice



Write Code → Write Test ✗

Write Test → Write Code ✓

Write Test → Run Test (Fails) → Add Feature → Run Test (Passes) → Re-factor Code

Code Linting

Linting is to run a program that analyzes Python code for various errors and potential problems. It checks for programmatic and stylistic errors, helping developers maintain a clean and consistent codebase that adheres to best practices.

flake8

Pylint

✦✦✦✦✧ Star your Python code!

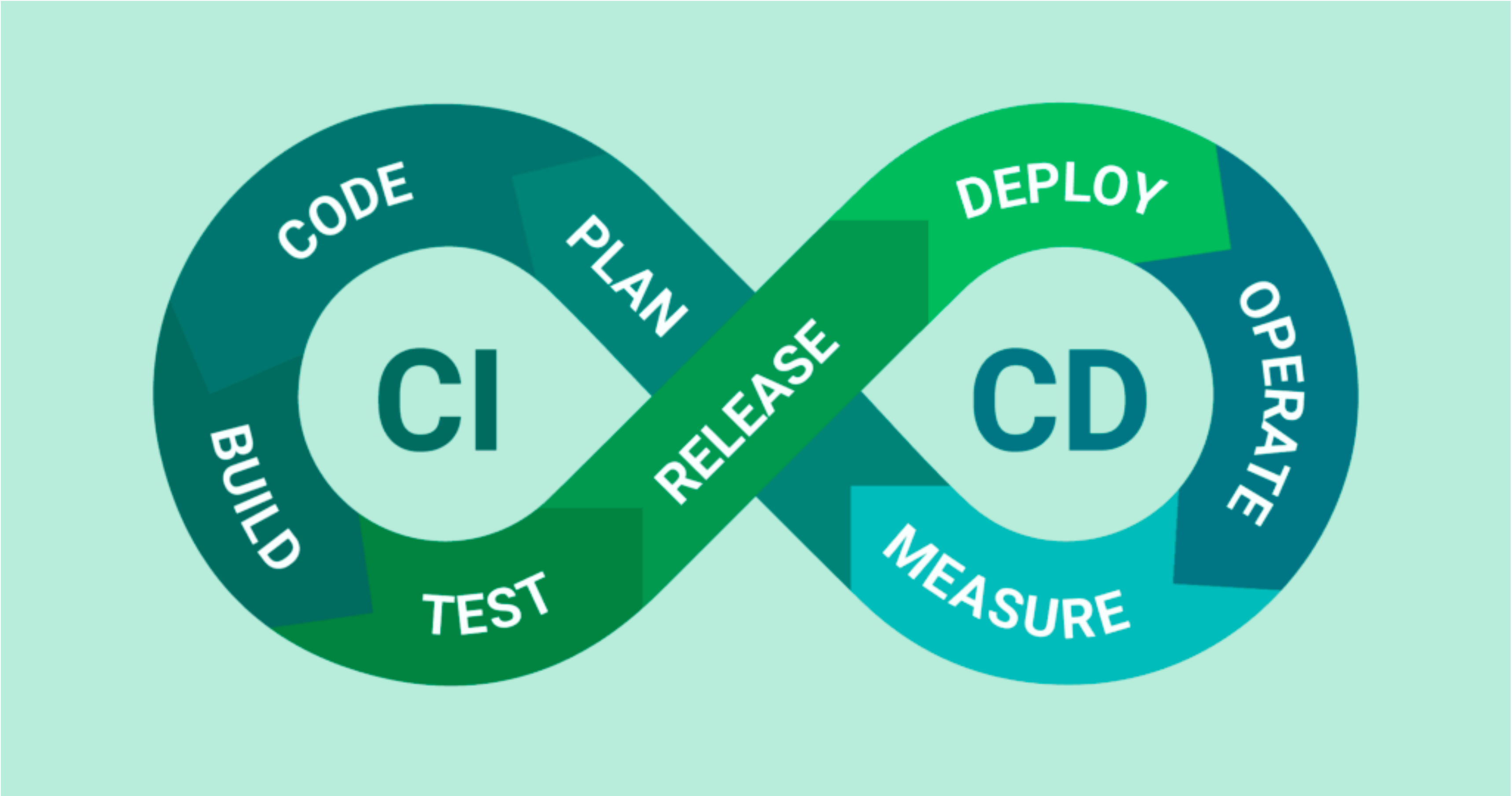
Topic 2: GitHub II
CD/CI

What is CI / CD ?

- **Continuous Improvement (CI)** refers to the practice of regularly integrating code changes from multiple developers into a shared repository.
    - The goal is to detect issues and conflicts early in the development cycle and ensure that the codebase remains stable.
- **Continuous Deployment (CD)** focuses on automating the deployment of applications to various environments (e.g., staging, production) after passing the CI stage.
    - It involves automating the steps required to build, test, and deploy software, reducing manual effort and ensuring consistent and reliable deployments.

GitHub

GitHub Actions



Steps to be taken

1. **Version Control:** Store your codebase in a GitHub repository hosted on GitHub
2. **Automated Testing:** Set up automated testing frameworks, such as unit tests or integration tests, that run whenever code changes are made.
3. **Continuous Integration:** Use GitHub Actions to automatically build and test your code whenever changes are pushed to the repository. This ensures that new code is integrated smoothly with the existing codebase and helps catch any errors or conflicts.
4. **Code Reviews:** Encourage code reviews by peers or senior developers to ensure code quality, adherence to best practices, and identify potential improvements.

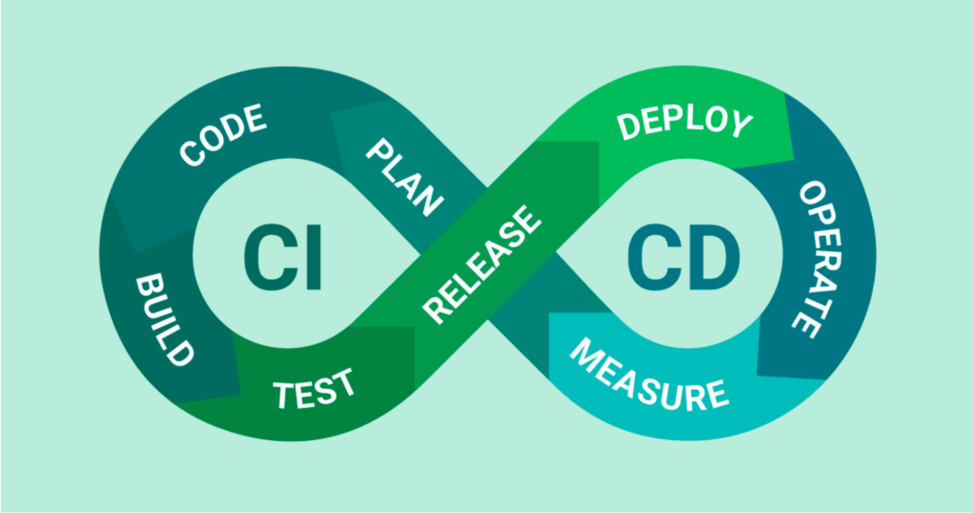
Exercise

- Whenever we merge branches and ask for a pull request, we can ask for code review from peers. This corresponds to a CI step

GitHub

Desktop

What are the benefits?



GitHub Actions

GitHub

- **Faster Time to Market**: Automating processes reduces manual effort and speeds up the delivery of software updates.

- **Improved Code Quality:** Continuous integration and automated testing catch errors early, ensuring code quality and reducing the risk of introducing bugs.

- **Collaboration and Feedback:** Code reviews and feedback from peers help improve code quality and foster collaboration among team members.

- **Reliable Deployments (Less human mistakes):** Automating the deployment process minimizes the risk of configuration errors and ensures consistent deployments across environments.

CI/CD Demo

- Let’s implement a simple CI pipeline using GitHub Actions to test and lint our code automatically

GitHub

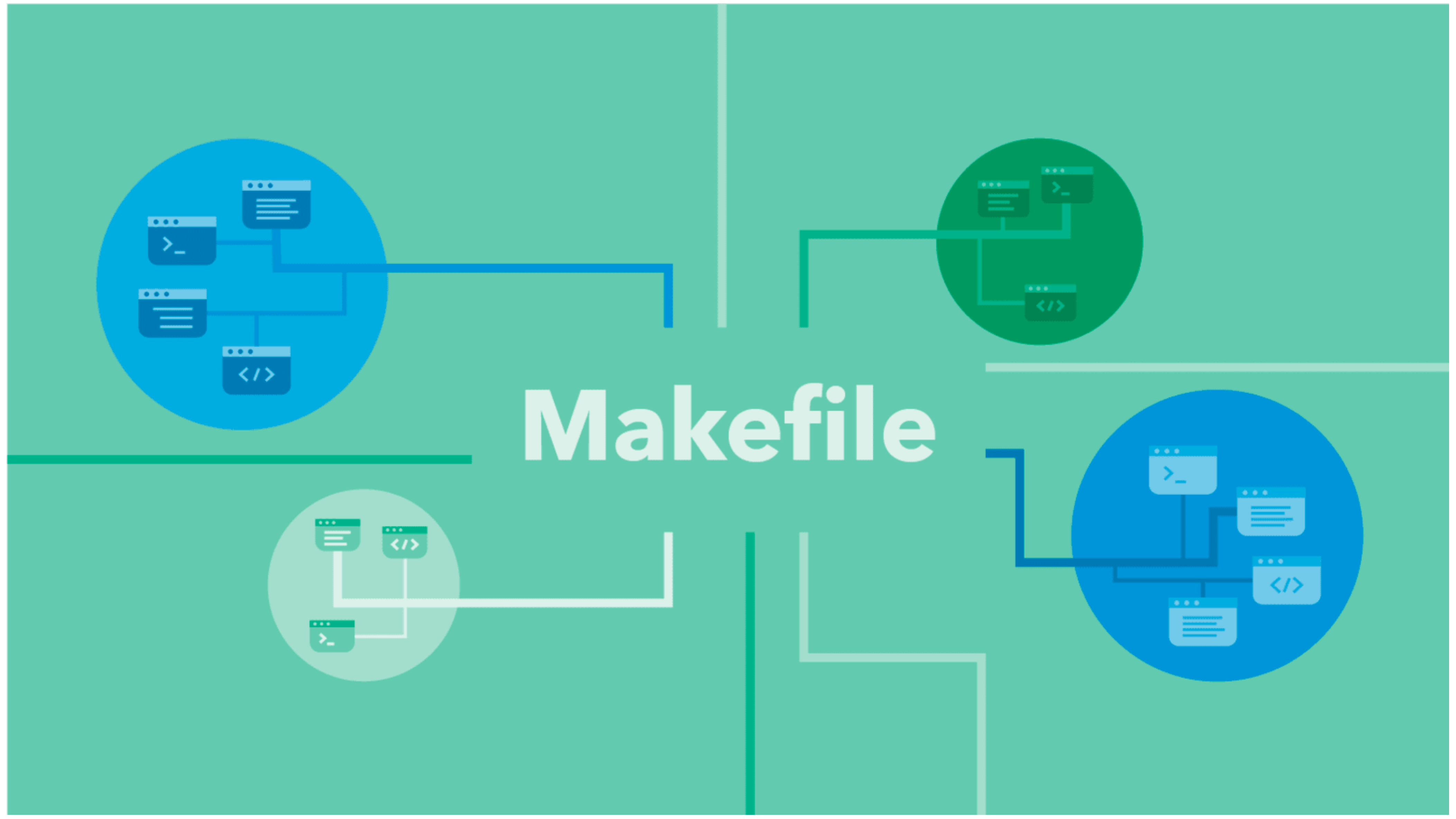
Desktop

esade

Topic 3: Makefile

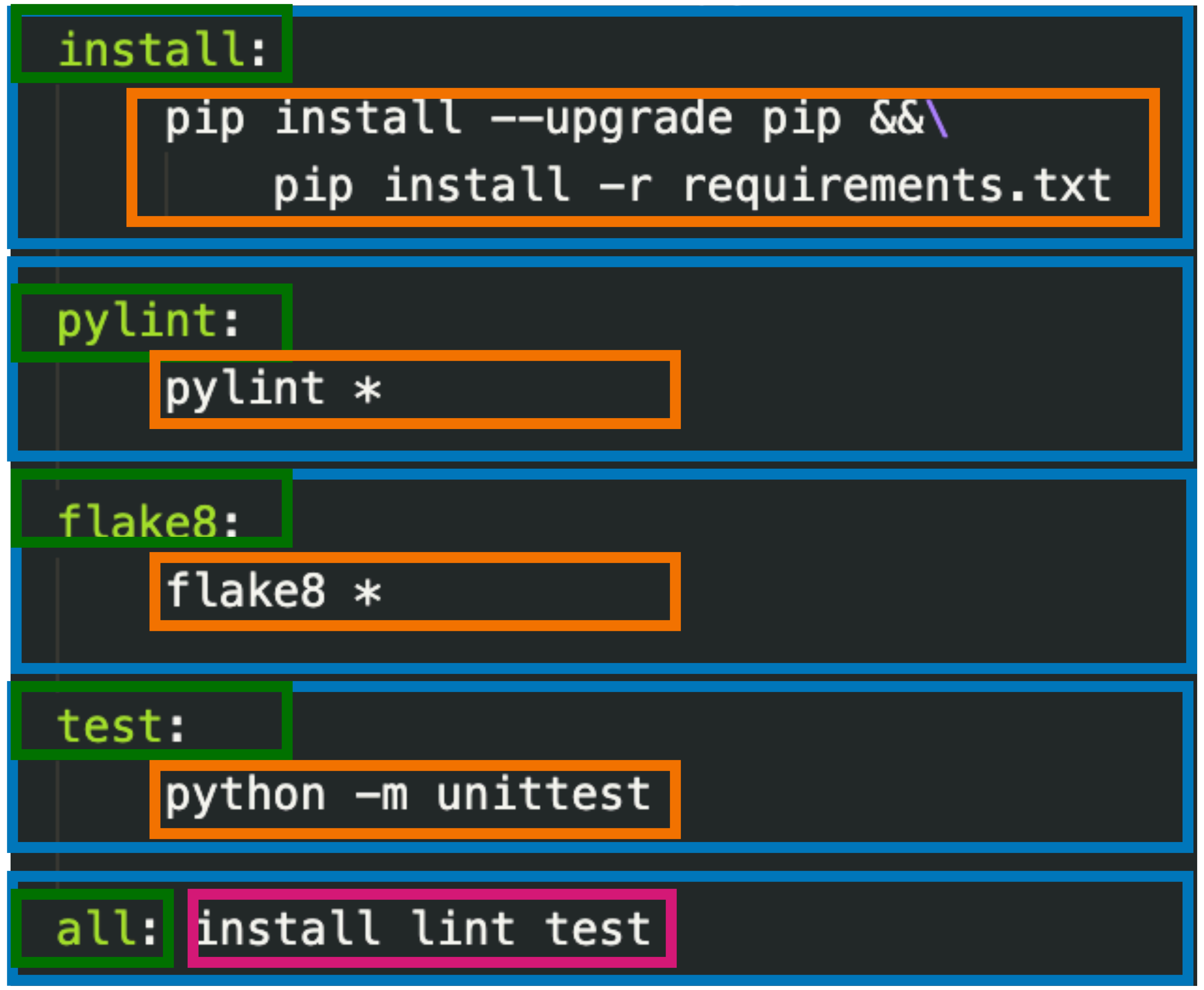
Makefile

- Makefile is a special file used to automate the process of compiling and building software by using “**make**”



Makefile

- Every blue box is a rule
- Every rule has a target (a name) (Green box)
- Every name can have dependencies (pink)
- Every rule and name has a command to be run (orange boxes)



```
install:
	pip install --upgrade pip &&\
		pip install -r requirements.txt

pylint:
	pylint *

flake8:
	flake8 *

test:
	python -m unittest

all: install lint test
```

Let’s build our own Makefile

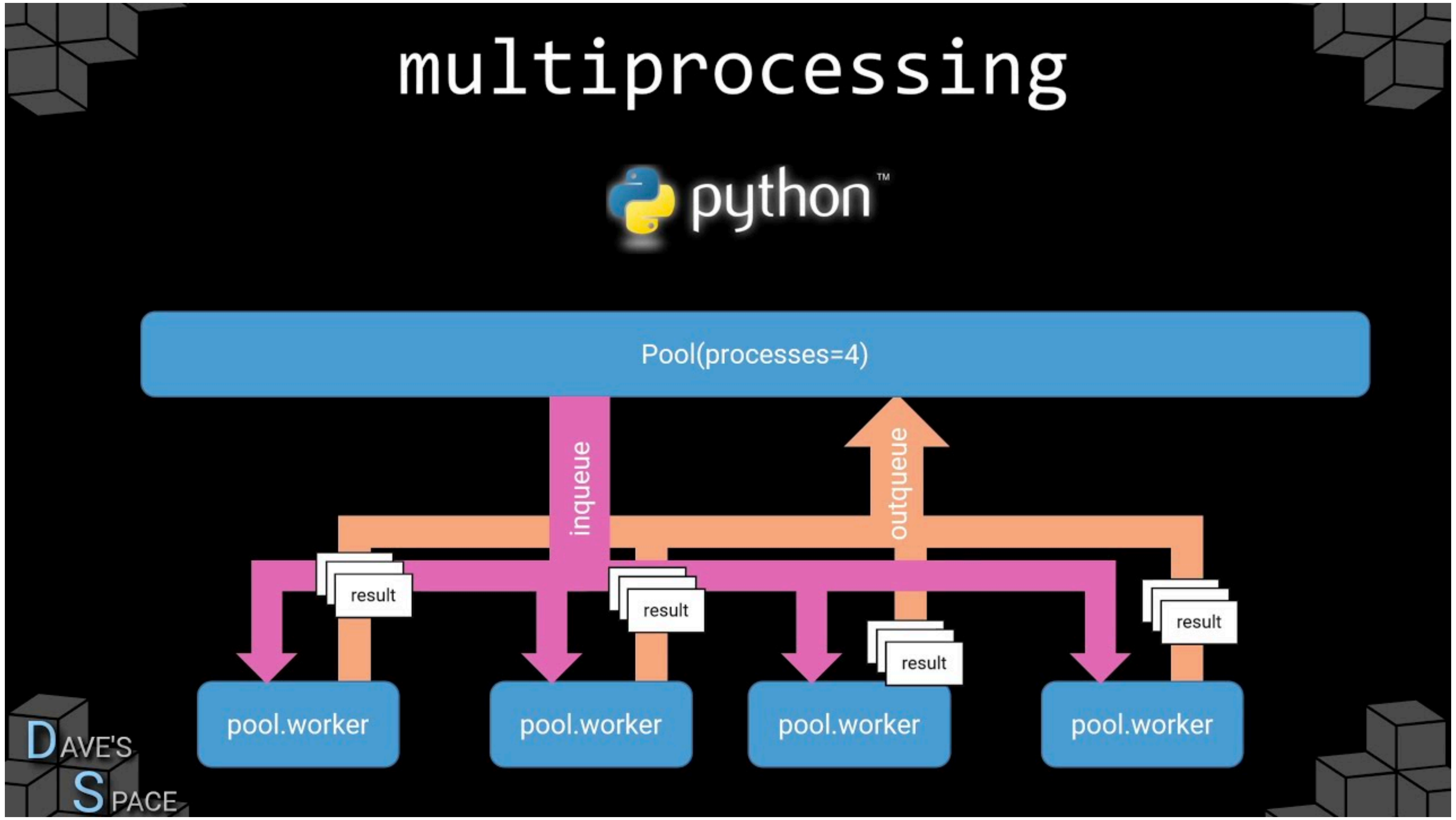
- Let’s build a makefile that can summarize the task we need to do in a single file and we can call it from everywhere

esade

Topic 4: Multiprocessing

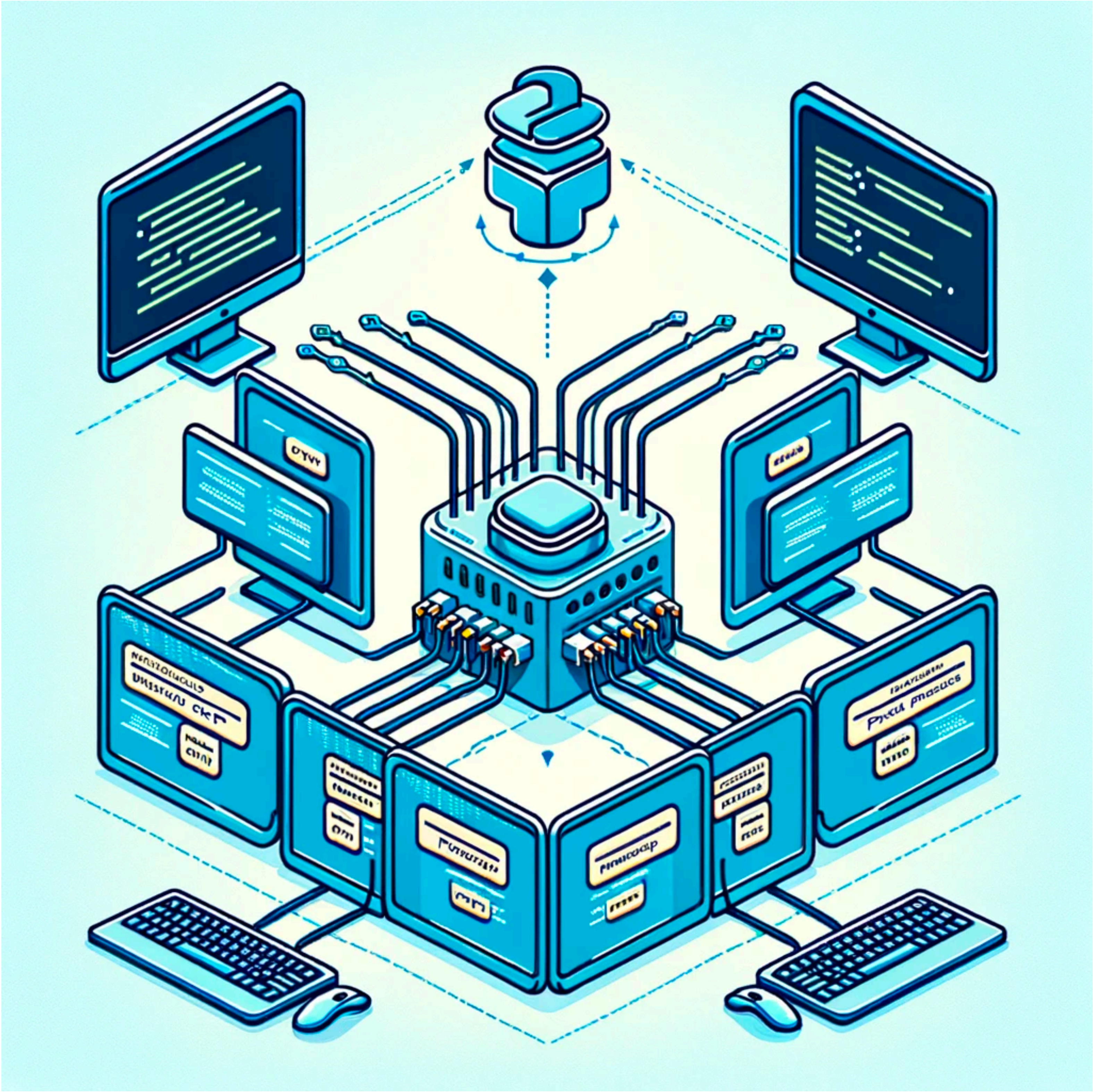
What is Multiprocessing in Python

Essentially is a tool to parallelize tasks to take advantage of multiple CPUs to reduce computation time



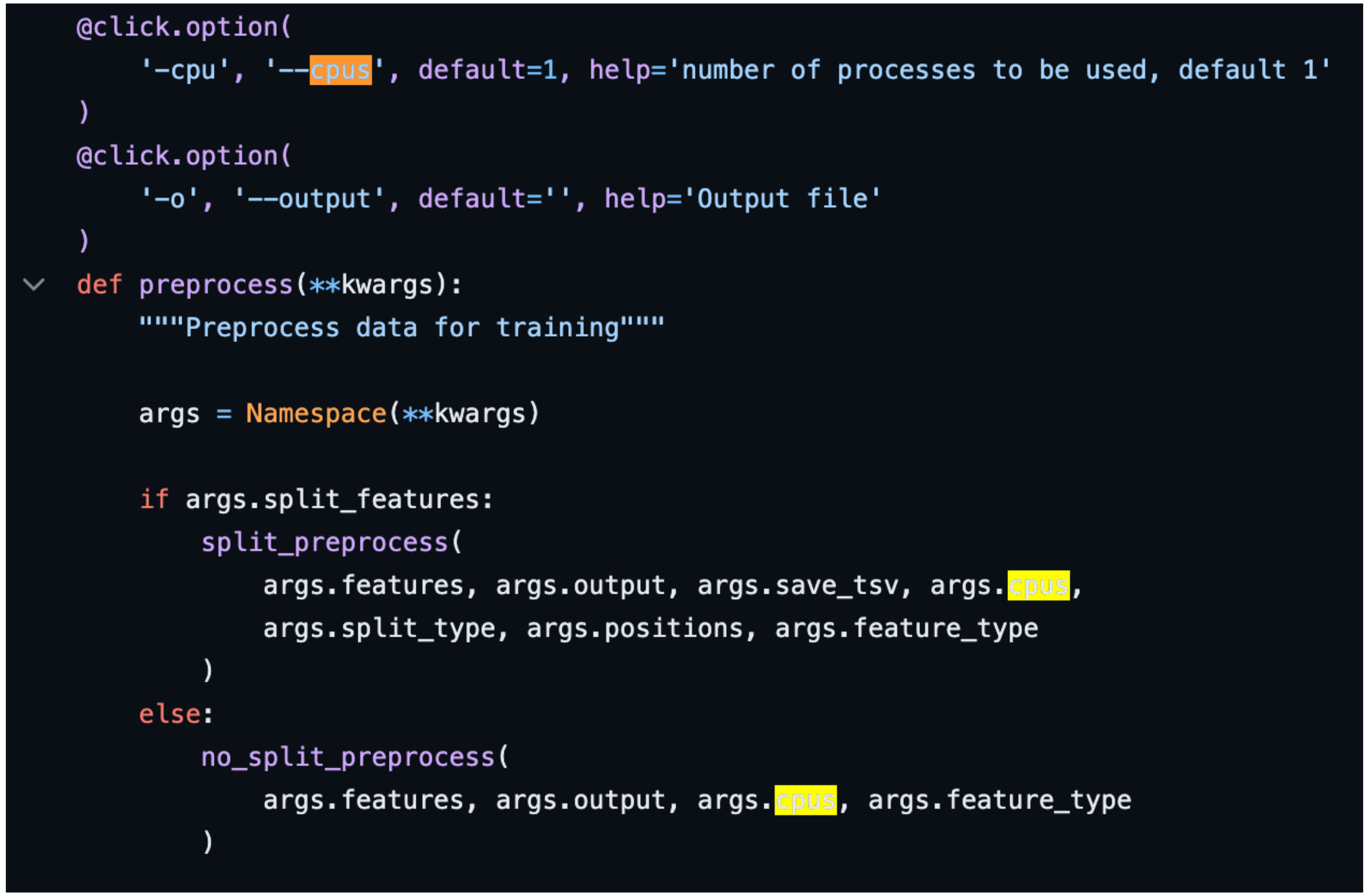
Multiprocessing Example

- Understand a multiprocessing example to calculate the factorial number



When have I used multiprocessing?

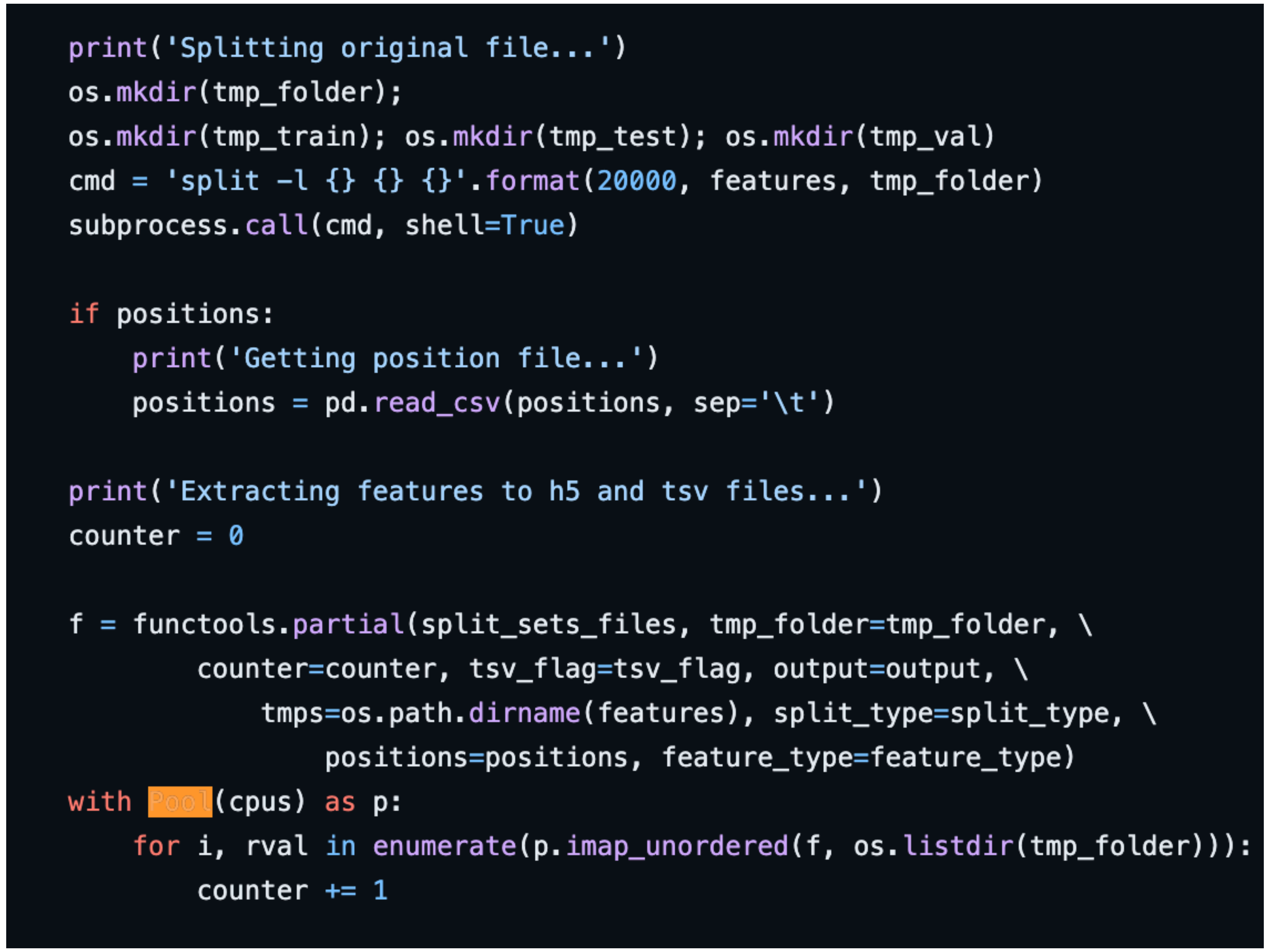
- Especially to work with enormous datasets and perform data cleaning and processing.
    - I “chop” the data into pieces, and every part is processed separately
- Otherwise, libraries have already build in parameters to use multiprocessing



```python
@click.option(
    '-cpu', '--cpus', default=1, help='number of processes to be used, default 1'
)
@click.option(
    '-o', '--output', default='', help='Output file'
)
def preprocess(**kwargs):
    """Preprocess data for training"""

    args = Namespace(**kwargs)

    if args.split_features:
        split_preprocess(
            args.features, args.output, args.save_tsv, args.cpus,
            args.split_type, args.positions, args.feature_type
        )
    else:
        no_split_preprocess(
            args.features, args.output, args.cpus, args.feature_type
        )
```



```python
print('Splitting original file...')
os.mkdir(tmp_folder);
os.mkdir(tmp_train); os.mkdir(tmp_test); os.mkdir(tmp_val)
cmd = 'split -l {} {} {}'.format(20000, features, tmp_folder)
subprocess.call(cmd, shell=True)

if positions:
    print('Getting position file...')
    positions = pd.read_csv(positions, sep='\t')

print('Extracting features to h5 and tsv files...')
counter = 0

f = functools.partial(split_sets_files, tmp_folder=tmp_folder, \
        counter=counter, tsv_flag=tsv_flag, output=output, \
            tmps=os.path.dirname(features), split_type=split_type, \
                positions=positions, feature_type=feature_type)
with Pool(cpus) as p:
    for i, rval in enumerate(p.imap_unordered(f, os.listdir(tmp_folder))):
        counter += 1
```

The reality of Multiprocessing

- The reality is that most likely you will not have to build your own multiprocess.
- In most cases, when needed you could just pass it as an argument to for instance train your machine learning model on more cpus or gpus.
- I have only used it for specific scenarios and when I have to I look up how to do it.