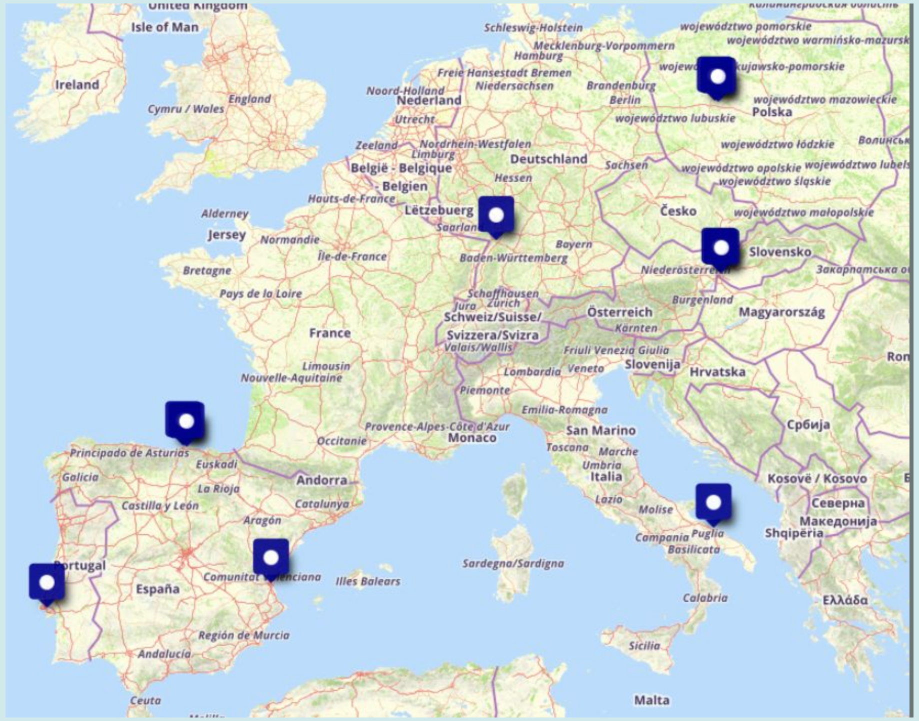
Log, debug and test!

B. Esteban

Co-funded by the European Union

AI4 eosc Co-funded by the European Union



CSIC Consejo Superior de Investigaciones Científicas

KIT Karlsruher Institut für Technologie

LIP

Ústav Informatiky Slovenská Akadémia Vied

Universitat Politècnica de València

INFN Istituto Nazionale di Fisica Nucleare

PSNC

WODR

MicroStep-MIS

predictia

AI4EOSC

Artificial Intelligence for the #EOSC

- Evolution of the DEEP Hybrid DataCloud platform
- HORIZON-INFRA-2021-EOSC-01-04 call
- Runs September 1st 2022 – August 2025 (36 months)
- 7 academic partners
  + 2 SME
  + 1 non-profit organization

Advanced features for distributed, federated, composite learning, metadata provenance, MLOps, event-driven data processing, and provision of AI/ML/DL services

Objectives

Objective 1
Why to log?

Get useful information about program state and errors in development and production runtimes.
Helps to improve your program/service.

Objective 2
Why to test?

Reproduce program states and evaluate correct program behaviour.
Helps to improve your program/service.

Objective 3
Debug errors and bugs without dying in the attempt.

Goal
Make robust code with low errors.
Easily find and solve bugs.
Edit code without breaking requirements.

Co-funded by the European Union

Logging Cookbook in AI4EOSC

- Print to stdio (print command) output for users, not program status.
- Do not return program status via API, sensible information might leak. (Passwords, emails, IPs, user ids, etc.)
- Log program status through the terminal is generally safe.
- If you catch exceptions with try, log the error before continue the program.

Recommended links:

- Logging HOWTO: https://docs.python.org/3/howto/logging.html
- Logging Cookbook for Python3: https://docs.python.org/3/howto/logging-cookbook.html

Logging Flow

Two main components of logging (remember):

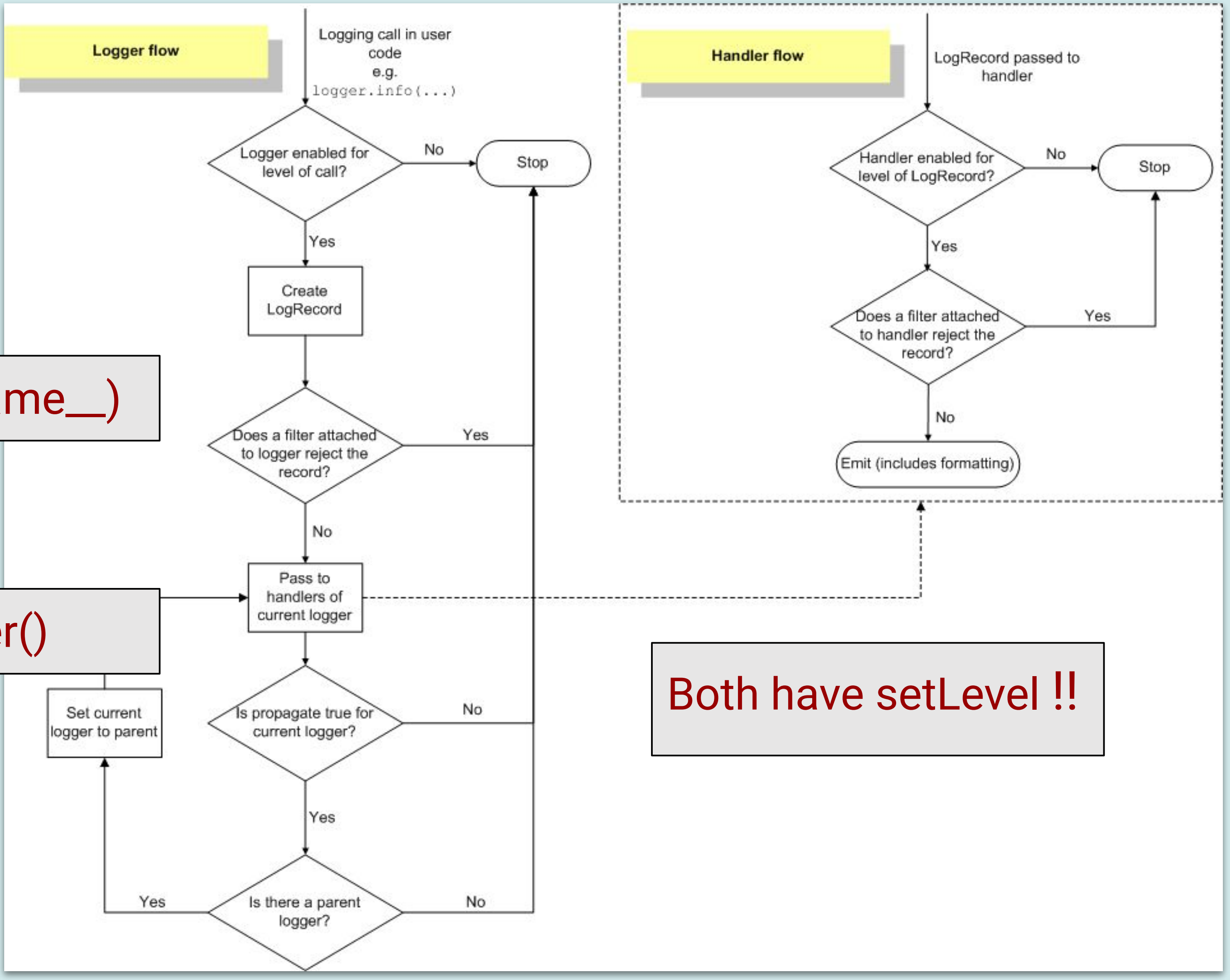
- Loggers:

```
logger = logging.getLogger(__name__)
```

- Handlers:
  (Not so important)

```
handler = logging.StreamHandler()
```

Why? There is normally a default handler for console or your web library.


Logger flow
Logging call in user code e.g. logger.info(...)
Logger enabled for level of call?
No
Stop
Yes
Create LogRecord
Does a filter attached to logger reject the record?
Yes
No
Pass to handlers of current logger
Set current logger to parent
Is propagate true for current logger?
No
Yes
Is there a parent logger?
Yes
No
Handler flow
LogRecord passed to handler
Handler enabled for level of LogRecord?
No
Stop
Yes
Does a filter attached to handler reject the record?
Yes
No
Emit (includes formatting)

Both have setLevel !!

AI4 | eosc Co-funded by the European Union

How to start logging?

- Generic scripts → use “logging.basicConfig”: logging.basicConfig(format=format, level=verbosity)
- Web frameworks → config file, e.i.; https://flask.palletsprojects.com/en/2.3.x/logging/#basic-configuration
- DEEPaaS → config file (also) → [debug = true] https://docs.deep-hybrid-datacloud.eu/projects/deepaas/en/stable/install/configuration/sample.html

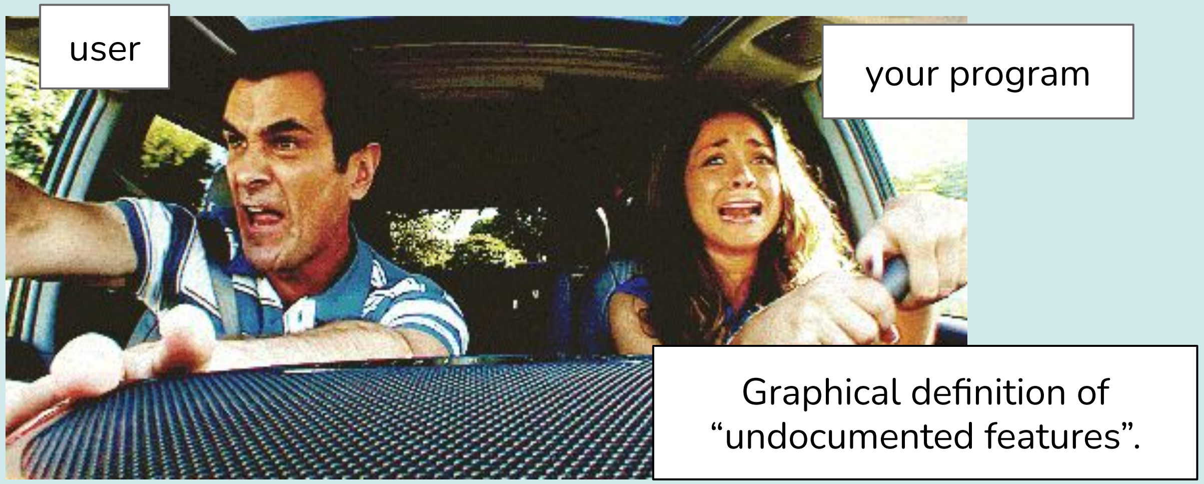
Small time for demo: advanced api

Logging Exercise:

- Clone repository from: https://github.com/BorjaEst/ai4eosc-exercises
- *Create/activate virtual environment with your favorite tool.
- Install model; use “pip install -e .”
- Open the model script to generate data at: “ai4eosc_exercises/data/create_dataset.py”.
- Edit the script to print log information in a file if “--debug” argument is true.
- Execute the script.
  python -m ai4eosc_exercises.data.create_dataset --debug my_data.txt
- Find the errors.

A bit of knowledge about TDD

- Software programming practice – Methodology where requirements are converted to test cases before software is fully developed.
- Origin – Developed by Kent Beck in the late 1990's as part of Extreme Programming.
- Relies on testing – A procedure intended to establish the quality, performance, or reliability of something, especially before it is taken into widespread use



A Software **DEFECT** / **BUG** / **FAULT** is a condition in a software product which does not meet a software requirement or end-user expectation.

softwaretestingfundamentals.com/defect

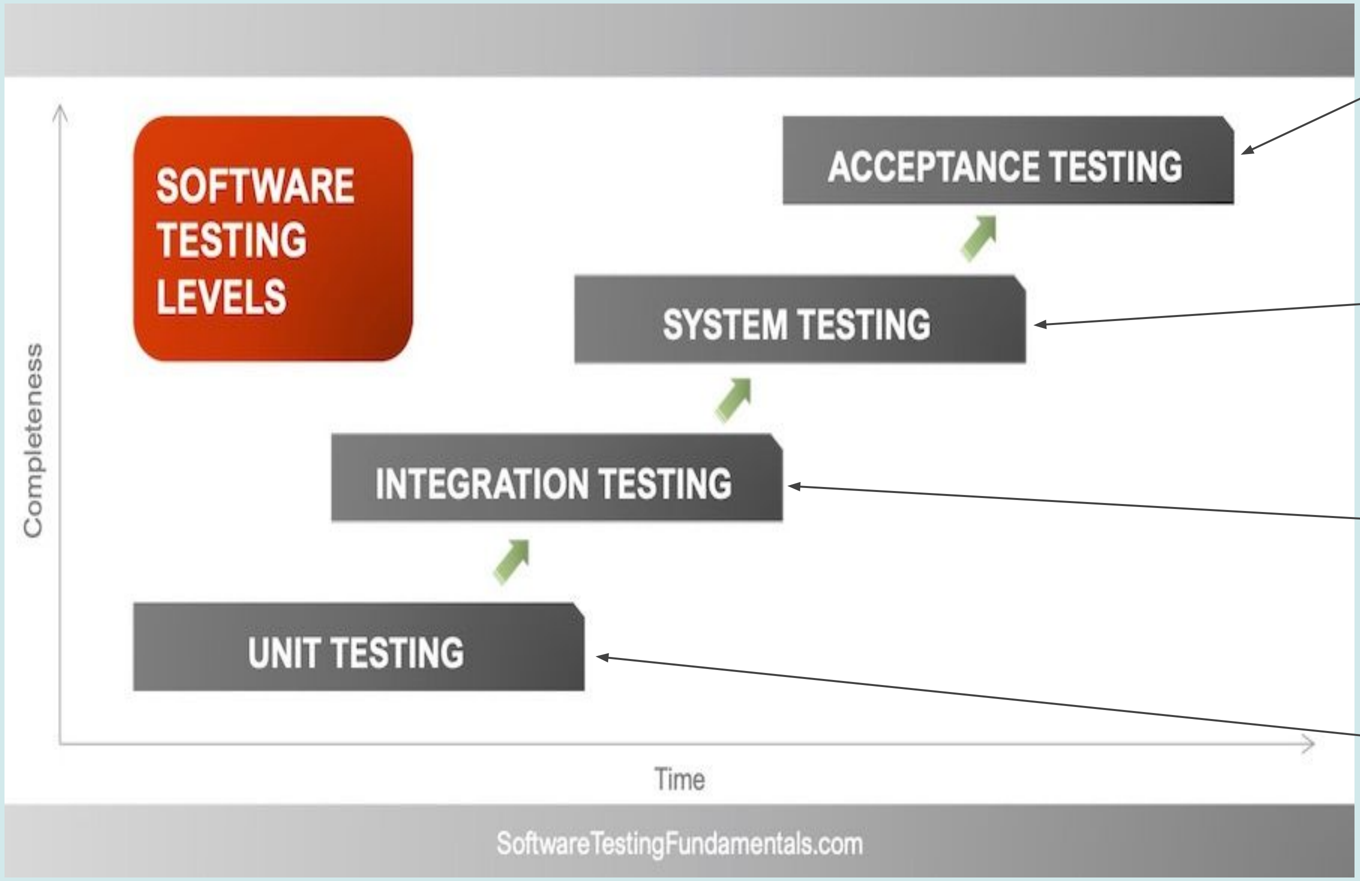
Testing Cookbook in AI4EOSC

- Pytest (library), Unittest (library) and unit testing (method).
  Are not the same! You should choose between unittest or pytest.
- Write generally tests as software requirements.
- Parametrization is generally better than 100% coverage.
- Using tox, helps you to ensure that it will run (almost) everywhere.
- CICD to ensure code contributions are always tested.

Recommended links:

- pytest: helps you write better programs: https://docs.pytest.org
- Python Unit testing framework: https://docs.python.org/3/library/unittest.html

Testing levels where you should test



Is AI4EOSC a nice framework?
Yes! (But I am not responsible of this)

Is DEEPaaS working?
Sure! (Also not responsible)

Does your software work in DEEPaaS?
Hope so 🤔 (Kind of responsible)

Does your software do what you want?
(Unfortunately I am responsible of this)

Co-funded by the European Union

Testing Flow and Scopes

Two main components of pytest testing (remember):

- Fixtures:

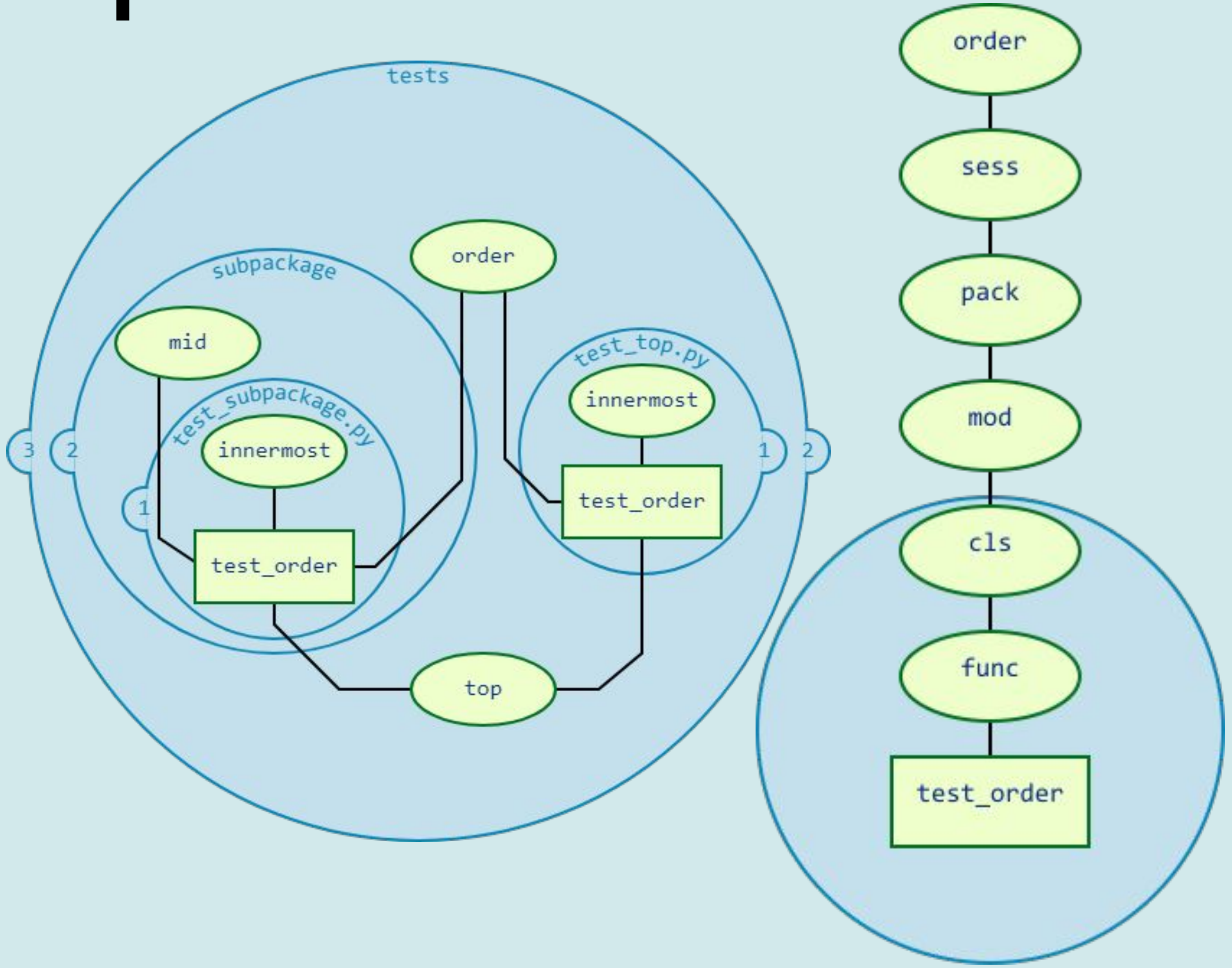
```
@pytest.fixture(scope=<scope>)
```

- Tests:

```
def test_<something>(<fixtures>):
```

Complicated? Do not worry, let's start by simply: Fixture == Setup so:

Fixture1 -> Fixture2 -> [test1, test2, test3]



https://docs.pytest.org/en/7.4.x/reference/fixtures.html#reference-fixtures

How to start testing?

- Simple testing → use “python -m pytest tests”:
  Run in local, does not handle installation of requirements.
- tox - automation project → use “tox -e <environment>”:
  Tests installation and execution of tests in different environments.
- CICD (Jenkins/github actions/etc.) → Just commit and push:
  Tests run in a remote machine automatically.

Small time for demo: advanced api

Testing Exercise:

- Clone repository from: https://github.com/BorjaEst/ai4eosc-exercises
- Install test requirements;
  “pip install -r requirements-test.txt .”
- Test your metadata completing tests at:
  “ai4eosc_exercises/tests/test_metadata/”.
- Create tests for predictions at:
  “ai4eosc_exercises/tests/test_predict/”.
- Prediction tests are currently using “test_dataset_1.txt” as unique input file, edit the fixtures for predictions to test also “test_dataset_2.txt”
- Can you repeat the steps for “tests/test_training/”?.

Debug is easy with the correct tools

- Log program status with python and DEEPaaS logging.
- Tests that point to the requirements that are failing.
- Debugger tools like breakpoints, to stop program execution.
- Python profilers to test your code efficiency.

-> Use IDE or Python pdb with:

```
> python -m pdb myscript.py
```

Recommended links:

- Debugging with vscode: https://code.visualstudio.com/docs/editor/debugging
- The Python Debugger: https://docs.python.org/3/library/pdb.html

Write your tests as requirements

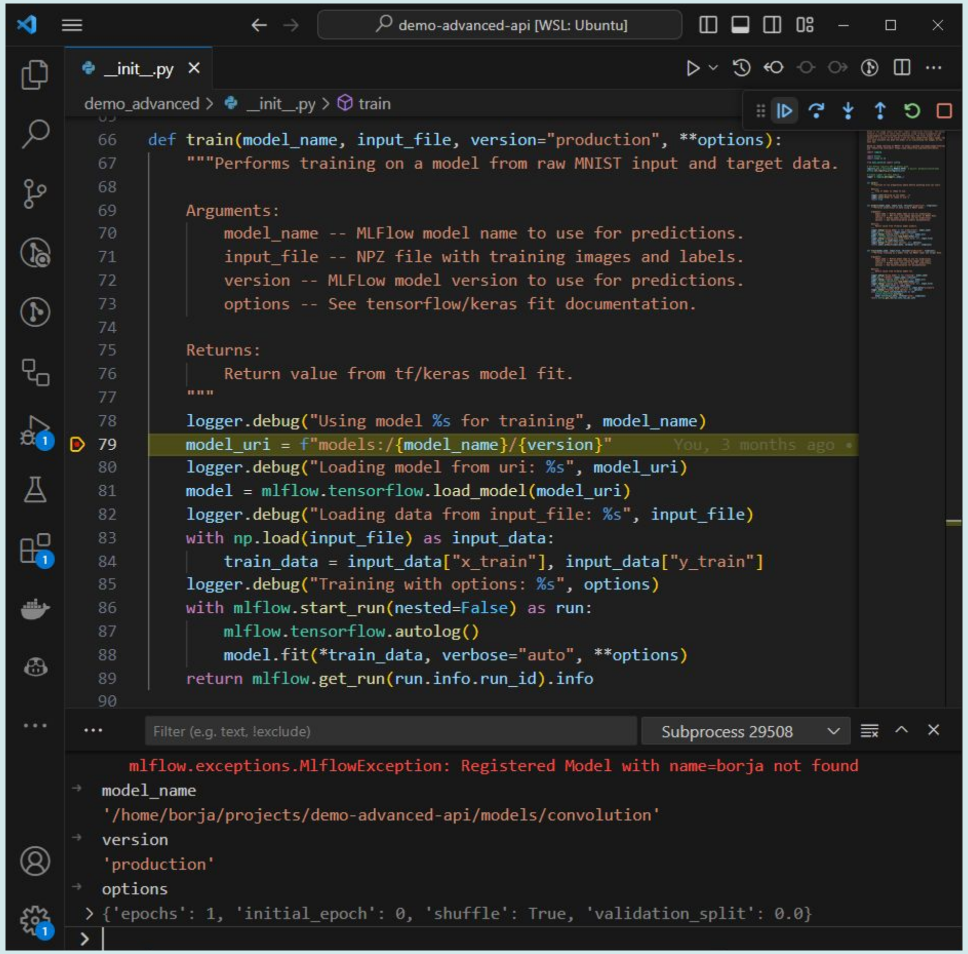
pizza_requirements/test_toppins.py

```
from pizza_factory import ingredients
from pytest import fixture

class TestPepperoni: # ----------------------------> Test case expressed as ‘class’
    @fixture(scope="class")
    def pepperoni(self): # ---------------------------> Fixture for case set up
        return ingredients.Pepperoni()

    def test_is_red(self, pepperoni): # -----------> Test case function/check
        assert pepperoni.color == "red"

    def test_is_round(self, pepperoni): # -------> Test case function/check
        assert pepperoni.shape == "round"
```

Will help you know what you cannot provide to your users.

If you do changes, you know can control the side effects.

What is a debugging breakpoint?


demo-advanced-api [WSL: Ubuntu]
__init__.py
demo_advanced > __init__.py > train
You, 3 months ago
Filter (e.g. text, !exclude)
Subprocess 29508

```python
def train(model_name, input_file, version="production", **options):
    """Performs training on a model from raw MNIST input and target data.

    Arguments:
        model_name -- MLFlow model name to use for predictions.
        input_file -- NPZ file with training images and labels.
        version -- MLFLow model version to use for predictions.
        options -- See tensorflow/keras fit documentation.

    Returns:
        Return value from tf/keras model fit.
    """
    logger.debug("Using model %s for training", model_name)
    model_uri = f"models:/{model_name}/{version}"
    logger.debug("Loading model from uri: %s", model_uri)
    model = mlflow.tensorflow.load_model(model_uri)
    logger.debug("Loading data from input_file: %s", input_file)
    with np.load(input_file) as input_data:
        train_data = input_data["x_train"], input_data["y_train"]
    logger.debug("Training with options: %s", options)
    with mlflow.start_run(nested=False) as run:
        mlflow.tensorflow.autolog()
        model.fit(*train_data, verbose="auto", **options)
    return mlflow.get_run(run.info.run_id).info
```

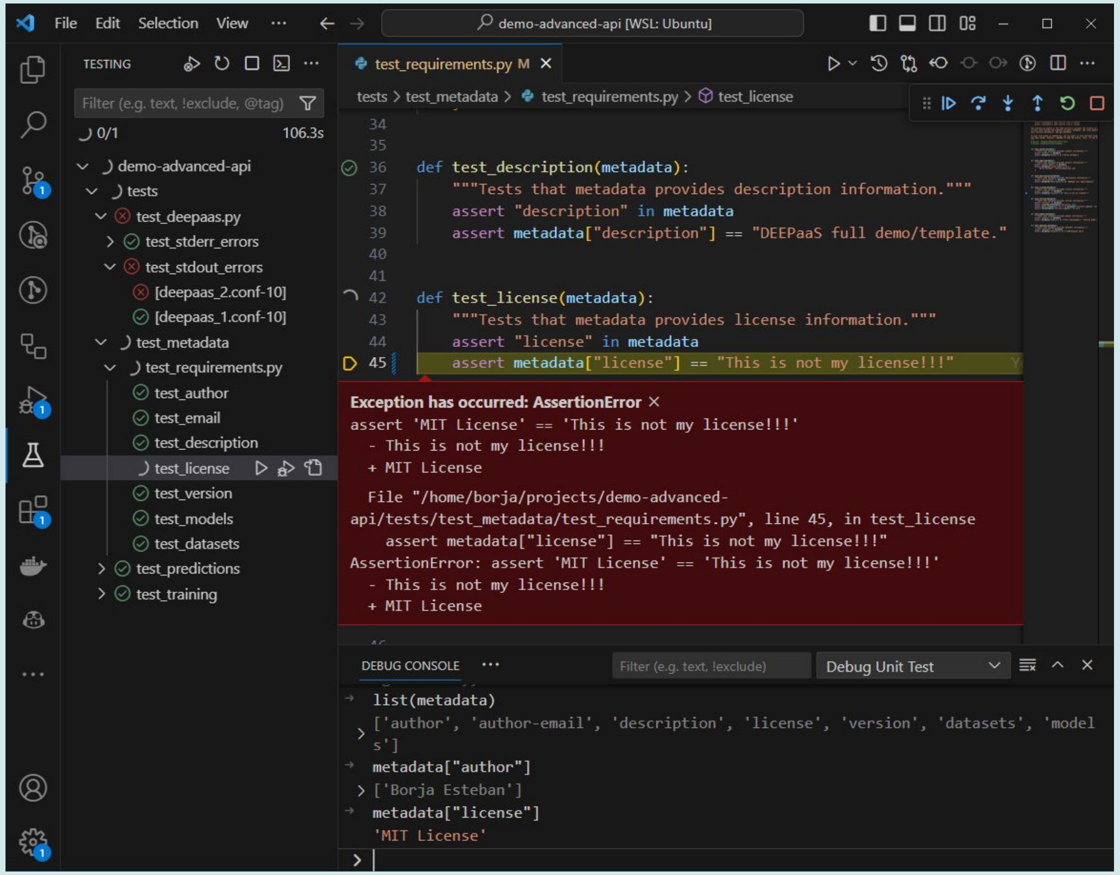
```
mlflow.exceptions.MlflowException: Registered Model with name=borja not found
→ model_name
'/home/borja/projects/demo-advanced-api/models/convolution'
→ version
'production'
→ options
> {'epochs': 1, 'initial_epoch': 0, 'shuffle': True, 'validation_split': 0.0}
>
```

Execution stops where you need:

- Defined red dots in the code.
- Defined commands in the code.
- When an exception is raised.
- When an exception is uncaught.
- etc.

Then you can print variables in the console and even execute commands.

You can easily integrate debugger tools with testing in most IDEs.



```python
def test_description(metadata):
    """Tests that metadata provides description information."""
    assert "description" in metadata
    assert metadata["description"] == "DEEPaaS full demo/template."


def test_license(metadata):
    """Tests that metadata provides license information."""
    assert "license" in metadata
    assert metadata["license"] == "This is not my license!!!"
```

```
Exception has occurred: AssertionError
assert 'MIT License' == 'This is not my license!!!'
  - This is not my license!!!
  + MIT License
  File "/home/borja/projects/demo-advanced-api/tests/test_metadata/test_requirements.py", line 45, in test_license
    assert metadata["license"] == "This is not my license!!!"
AssertionError: assert 'MIT License' == 'This is not my license!!!'
  - This is not my license!!!
  + MIT License
```

```
list(metadata)
['author', 'author-email', 'description', 'license', 'version', 'datasets', 'models']
metadata["author"]
['Borja Esteban']
metadata["license"]
'MIT License'
```

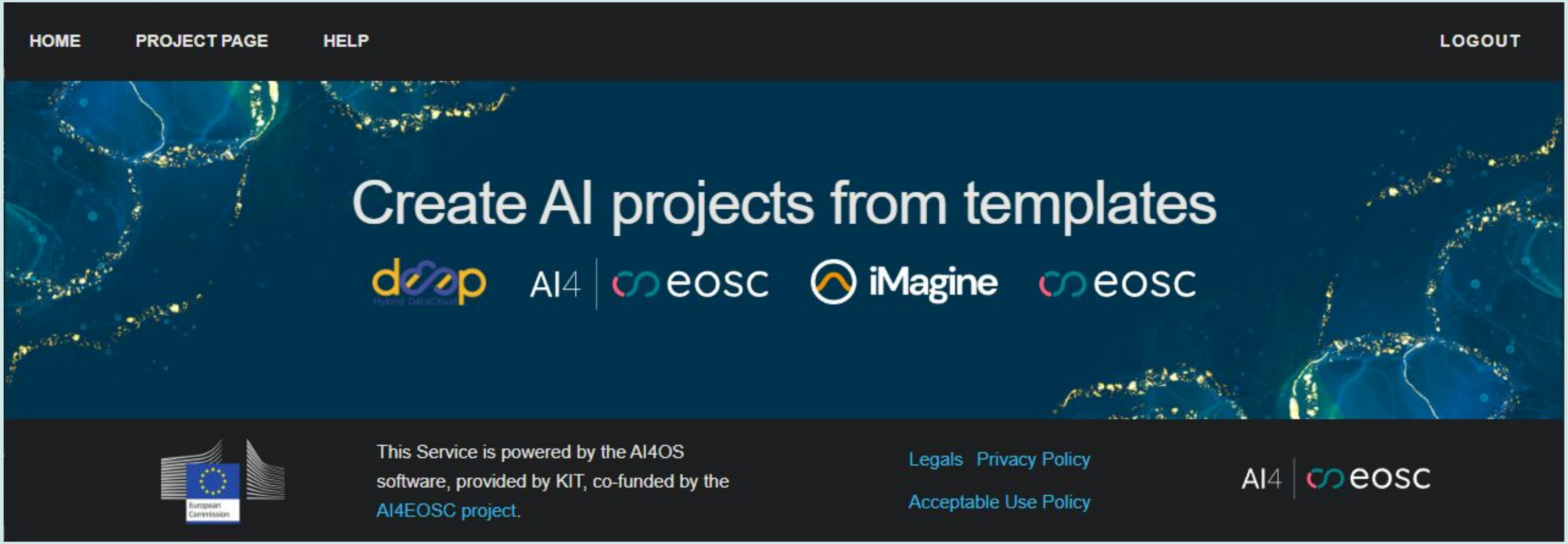
Time to see it in action?

Small time for demo: advanced api

Debugging Exercise:

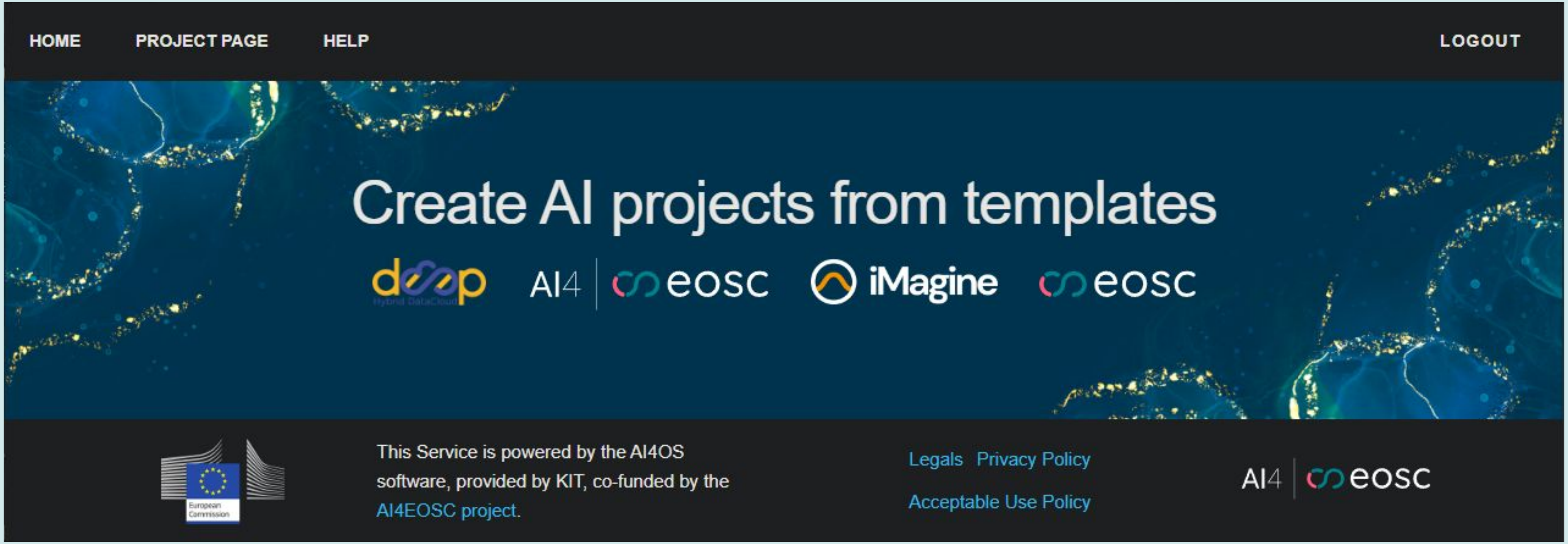
- Clone repository from: https://github.com/BorjaEst/ai4eosc-exercises
- Can you rewrite your tests and fixtures to make them look like model requirements?
- Pause execution when testing test_emails.
  If you are not using an IDE, use: `pdb.set_trace()`
  Can you tell the value for metadata[“license”] ?
- Run DEEPaaS with a debugger. Open the browser at the local URL and call for “GET /models/ai4eosc_exercises”
  Can you pause the execution when calling the method?
- Can you use “logger.debug” to print information when calling again “GET /models/ai4eosc_exercises”. What is missing?
  Hint: Look at “.vscode/launch.json” -> “Line 12””.

Time for questions



This Service is powered by the AI4OS software, provided by KIT, co-funded by the AI4EOSC project.

Thank you for your time!



This Service is powered by the AI4OS software, provided by KIT, co-funded by the AI4EOSC project.

FAQ

- Flask returns debug info in web page!
  Yes, but for frontend debug purposes, DEEPaaS API is not a frontend framework (Yet).