# **Bioinformatics** session

3rd Annual workshop on bioinformatics and variant interpretation in InPreD

https://inpred.github.io/25-06\_bioinfo\_ws/bioinfo\_ws



## 1. Unit testing

## What is unit testing?

- test smallest piece of code that can be logically isolated in software application (function, subroutine, method)
- the smaller the better more granular view of what is going on; also faster
- should not cross systems (database, filesystem, network) -> integration and functional tests

## **Example**

```
# calculator.py
def add(x, y):
    """add numbers"""
    return x + y
```

```
# test_calculator.py
import calculator

def test_add():
    assert calculator.add(1, 2) == 3
```

## Why do we need unit testing?

- early defect detection
- code quality improvement
- facilitates refactoring
- faster development cycles
- better documentation
- enables more frequent releases

## How to design a unit test?

- identify the unit (function, method)
- what is its functionality?
- what is the input (correct and incorrect)?
- how to handle incorrect input? (edge cases, invalid data)
- what does it return?
- positive and negative results should be tested

## Set up unit testing for your functions

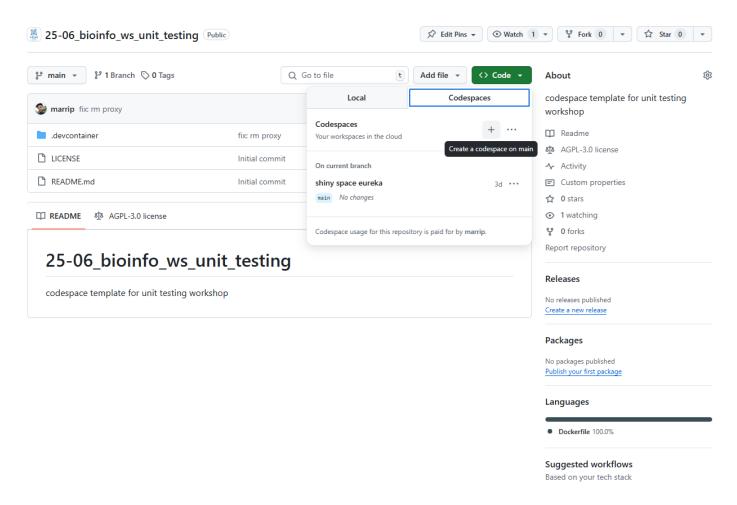
install pytest

```
$ pip install pytest
```

- add your function to a module at my\_module/my\_module.py
- add your unit test at my\_module/tests/my\_module\_test.py
- in the test file import your module from my\_module.my\_module import my\_function

#### First exercise

• go to https://github.com/InPreD/25-06\_bioinfo\_ws\_unit\_testing



### First exercise



#### First exercise

- pytest was already installed in the codespace
- the suggested layout was already applied
- create a branch for your work:

```
$ git checkout -b unit-tests-<your name>
```

- start with the first exercise in first/tests/first\_test.py
- whenever you are done, commit your changes (use commit message conventions):

```
$ git add first/tests/first_test.py
$ git commit -m "test: <your commit message>"
```

```
$ git push --set-upstream origin unit-tests-<your name>
```

## Handle exceptions in unit tests

- functions can raise exceptions and we would like to test for those
- import pytest to have access to raises()
- add with -block to handle the exception:

```
import calculator
import pytest

def test_add_exception():
    with pytest.raises(TypeError):
        assert add("one", "two") == None
```

#### **Second exercise**

- continue with the second exercise in second/tests/second\_test.py
- whenever you are done, commit your changes (use commit message conventions):

```
$ git add second/tests/second_test.py
$ git commit -m "test: <your commit message>"
```

```
$ git push
```

## Make unit tests table-driven by using parametrize

- having more than one test case results in repeating a lot of code (one function per test case)
- to condense this as much as possible (ideally one unit test per function), we can use the pytest decorator parametrize
- again, import pytest to gain access to the decorator
- add the decorator @pytest.mark.parametrize as a header to your function
- define the required variables (input, exception, output)
- add your test cases as a list of tuples (one tuple per case)
- also use nullcontext from the module contextlib to account for cases without exceptions

```
import calculator
import pytest
from contextlib import nullcontext
@pytest.mark.parametrize(
    "x, y, exception, want",
        (1, 2, nullcontext(), 3),
        ("one", "two", pytest.raises(TypeError), None)
def test_add(x, y, exception, want):
    with exception:
        assert add(x, y) == want
```

#### Third exercise

- continue with the third exercise in third/tests/third\_test.py
- whenever you are done, commit your changes (use commit message conventions):

```
$ git add third/tests/third_test.py
$ git commit -m "test: <your commit message>"
```

```
$ git push
```

## Use github action to automatically run tests on push

• add .github/workflows/main.yml:

```
name: Python test
on: push
jobs:
  test:
    name: Run unit tests
    runs-on: ubuntu-latest
    steps:
        name: Check out the repo
        uses: actions/checkout@v4
        name: Set up Python 3.12.8
        uses: actions/setup-python@v4
        with:
          python-version: 3.12.8
        name: Install dependencies
        run: pip install -r requirements.txt
        name: Unit testing
        uses: pavelzw/pytest-action@v2
        with:
          verbose: true
          emoji: true
          job-summary: true
          custom-arguments: -q
          click-to-expand: true
          report-title: 'Bioinfo workshop unit testing'
```

## Use github action to automatically run tests on push

• if you don't want to write all of that, merge the branch containing the file into your branch:

\$ git merge add-github-action

#### Fourth exercise

- write unit tests for the functions in fourth/fourth.py
- whenever you are done, commit your changes (use commit message conventions):

```
$ git add third/tests/third_test.py
$ git commit -m "test: <your commit message>"
```

```
$ git push
```

## 2. Nextflow