

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

25-06_bioinfo_ws_unit_testing Public

Edit Pins Watch 1 Fork 0 Star 0

main 1 Branch 0 Tags

Go to file

Add file <> Code

marrip fix: rm proxy

.devcontainer fix: rm proxy

LICENSE Initial commit

README.md Initial commit

README AGPL-3.0 license

25-06_bioinfo_ws_unit_testing

codespace template for unit testing workshop

Local Codespaces

Codespaces

Your workspaces in the cloud

On current branch

shiny space eureka 3d

main No changes

Create a codespace on main

Codespace usage for this repository is paid for by marrip.

About

codespace template for unit testing workshop

Readme

AGPL-3.0 license

Activity

Custom properties

0 stars

1 watching

0 forks

Report repository

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

Dockerfile 100.0%

Suggested workflows

Based on your tech stack

First exercise

← →

25-06_bioinfo_ws_unit_testing [Codespaces: shiny space eureka]

👤

☰

EXPLORER

...

📁

25-06_BIOINFO_WS_UNIT_TESTING [CODESPA...

> .devcontainer

🔑 LICENSE

📄 README.md

🔍

🔗

🔧

📁

🐙

👤

⚙️

> GLIEDERUNG

> ZEITACHSE

[Vorschau] README.md

×

25-06_bioinfo_ws_unit_testing

codespace template for unit testing workshop

PROBLEME

AUSGABE

DEBUGGING-KONSOLE

TERMINAL

PORTS

bash

+

⌵

📄

🗑️

...

⬆️

×

```
root@codespaces-defab7:/workspaces/25-06_bioinfo_ws_unit_testing#
```

<> Codespaces: shiny space eureka

🔗 main ↺

⊗ 0 ⚠️ 0 🗨️ 0

👤 Layout: German

🔔

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>"
```

- and we push them to GitHub:

```
$ 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>"
```

- and we push them to GitHub:

```
$ 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>"
```

- and we push them to GitHub:

```
$ 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](#)):

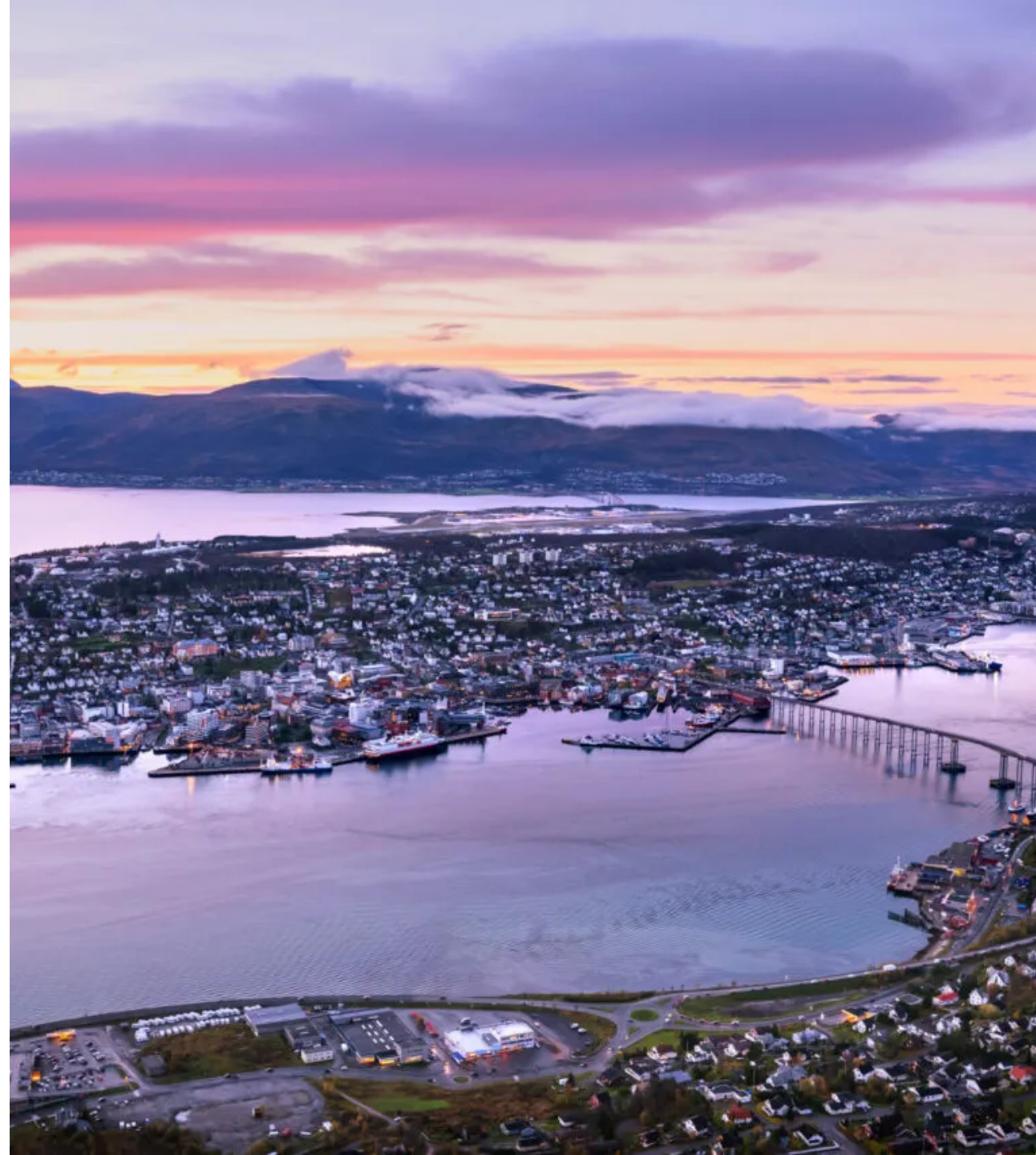
```
$ git add third/tests/third_test.py  
$ git commit -m "test: <your commit message>"
```

- and we push them to GitHub:

```
$ git push
```

Thank you for your attention!

Day 1 done!



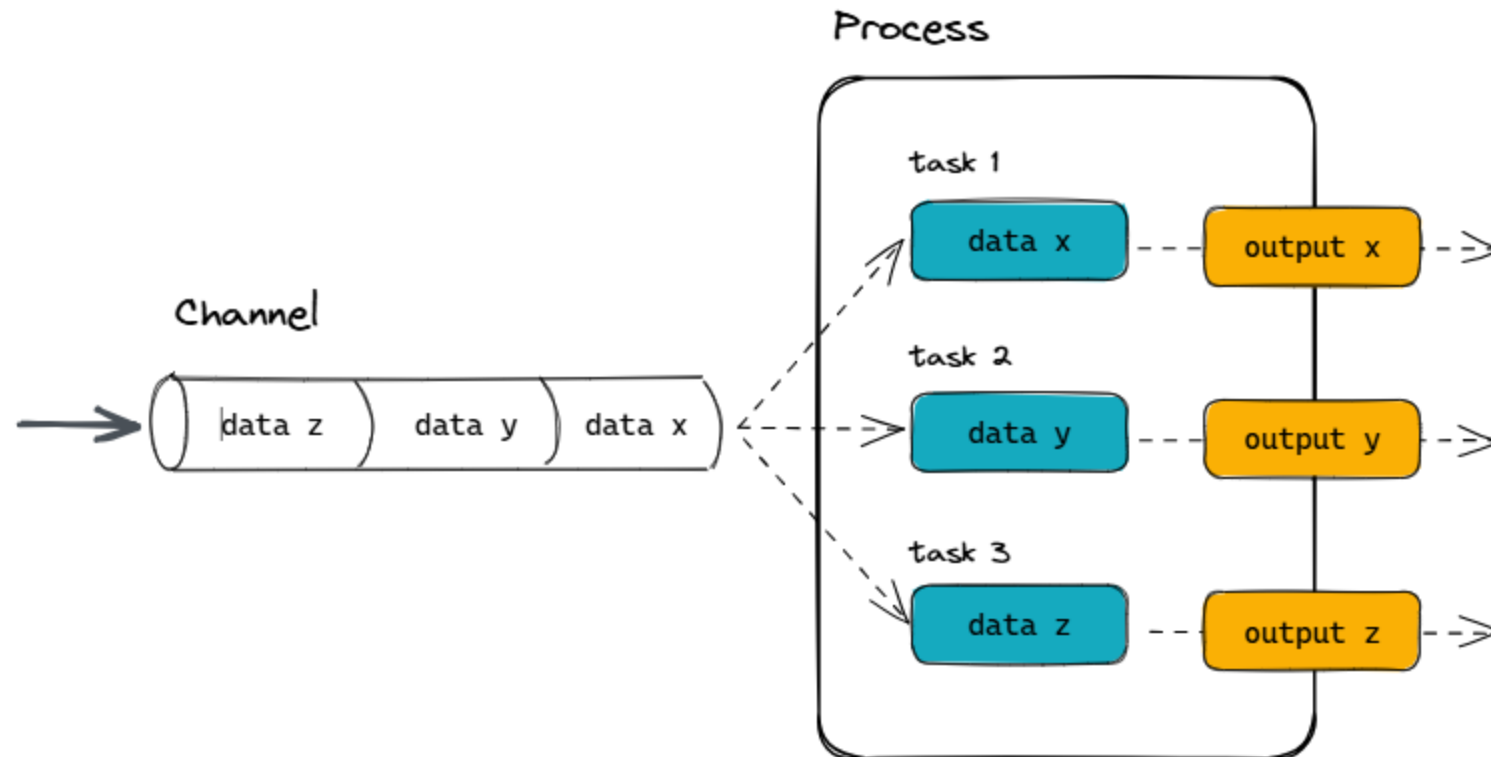
2. Nextflow

What is nextflow?

- workflow orchestration engine, domain-specific language (in contrast to general-purpose language, e.g. python)
- easy to write data-intensive computational workflows
- extension of groovy which is a superset of Java
- core features:
 - portability and reproducibility
 - scalability of parallelization and deployment
 - integration of existing tools, systems, and industry standards

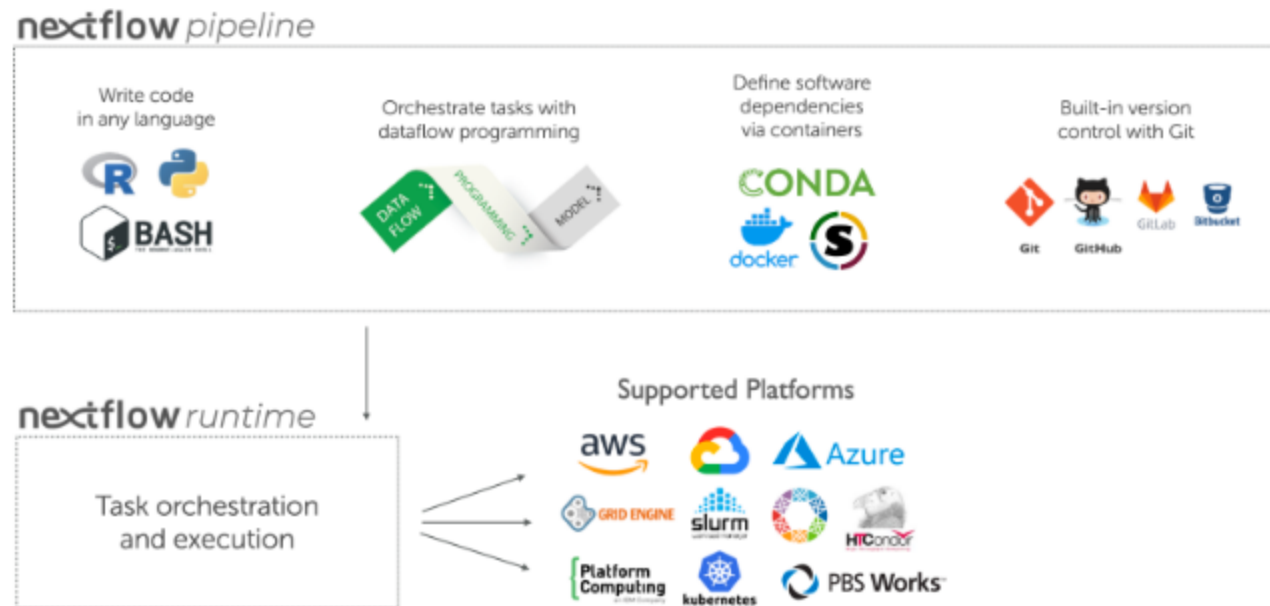
Processes and channels

- different processes joined together - each written in any language that can be executed by Linux platform
- independently and isolated processes - not sharing common (writable) state
- communication via asynchronous first-in, first-out (FIFO) queues, called `channels`



Execution abstraction

- process = *what* is executed <-> executor = *how* it is executed
- provides abstraction between workflow's functional logic and underlying execution system/runtime
- workflow runs seamlessly on local computer, HPC cluster or cloud



Let's get started

- go to https://github.com/InPreD/25-06_bioinfo_ws_nextflow

The screenshot shows the GitHub repository page for '25-06_bioinfo_ws_nextflow' by user 'marrip'. The repository is public and has 1 branch and 0 tags. The 'Code' dropdown menu is open, showing options for 'Local' and 'Codespaces'. The 'Codespaces' section includes a '+ Create a codespace on main' button. The repository description is 'codespace template for nextflow workshop'. The file list includes '.devcontainer', '.gitignore', 'LICENSE', and 'README.md'. The right sidebar shows repository details like 'About', 'Releases', 'Packages', and 'Languages'.

25-06_bioinfo_ws_nextflow (Public)

Edit Pins Watch 1 Fork 0 Star 0

main 1 Branch 0 Tags

Go to file

Add file <> Code

marrip chore: add gitignore file

.devcontainer chore: add devcontainer

.gitignore chore: add gitignore

LICENSE Initial commit

README.md Initial commit

README AGPL-3.0 license

25-06_bioinfo_ws_nextflow

codespace template for nextflow workshop

About

codespace template for nextflow workshop

Readme

AGPL-3.0 license

Activity

Custom properties

0 stars

1 watching

0 forks

Report repository

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

Dockerfile 100.0%

Suggested workflows

Based on your tech stack

- create a branch for your work:

```
$ git checkout -b nextflow-<your name>
```

- create a new file `hello_world.nf`
- write a workflow which outputs a file containing "Hello World!"

```
#!/usr/bin/env nextflow

/*
 * Use echo to print 'Hello World!' to a file
 */
process sayHello {

    output:
        path 'output.txt'

    script:
        """
        echo 'Hello World!' > output.txt
        """
}

workflow {

    // emit a greeting
    sayHello()
}
```

- try to run it

```
$ nextflow run hello_world.nf
```

```
root@52abc3ecdd95 /w/25-06_bioinfo_ws_nextflow (main)# nextflow run hello_world.nf
Nextflow 25.04.1 is available - Please consider updating your version to it

NEXTFLOW ~ version 24.10.4

Launching `hello_world.nf` [boring_rubens] DSL2 - revision: 595ea09581

executor > local (1)
[33/7106de] process > sayHello [100%] 1 of 1 ✓
```

- check if you can find `work/33/7106de/output.txt`
- explore the other files that are in `work/33/7106de/`

- add a directory to which results should be published

```
process sayHello {  
    publishDir 'results', mode: 'copy'  
    output:  
        path 'output.txt'  
    ...  
}
```

- add a greeting variable

```
process sayHello {  
    publishDir 'results', mode: 'copy'  
  
    input:  
        val greeting  
  
    output:  
        path 'output.txt'  
  
    script:  
        """  
        echo '$greeting' > output.txt  
        """  
}  
  
workflow {  
    // emit a greeting  
    sayHello(params.greeting)  
}
```

```
$ nextflow run hello_world.nf --greeting 'Heisann!'
```