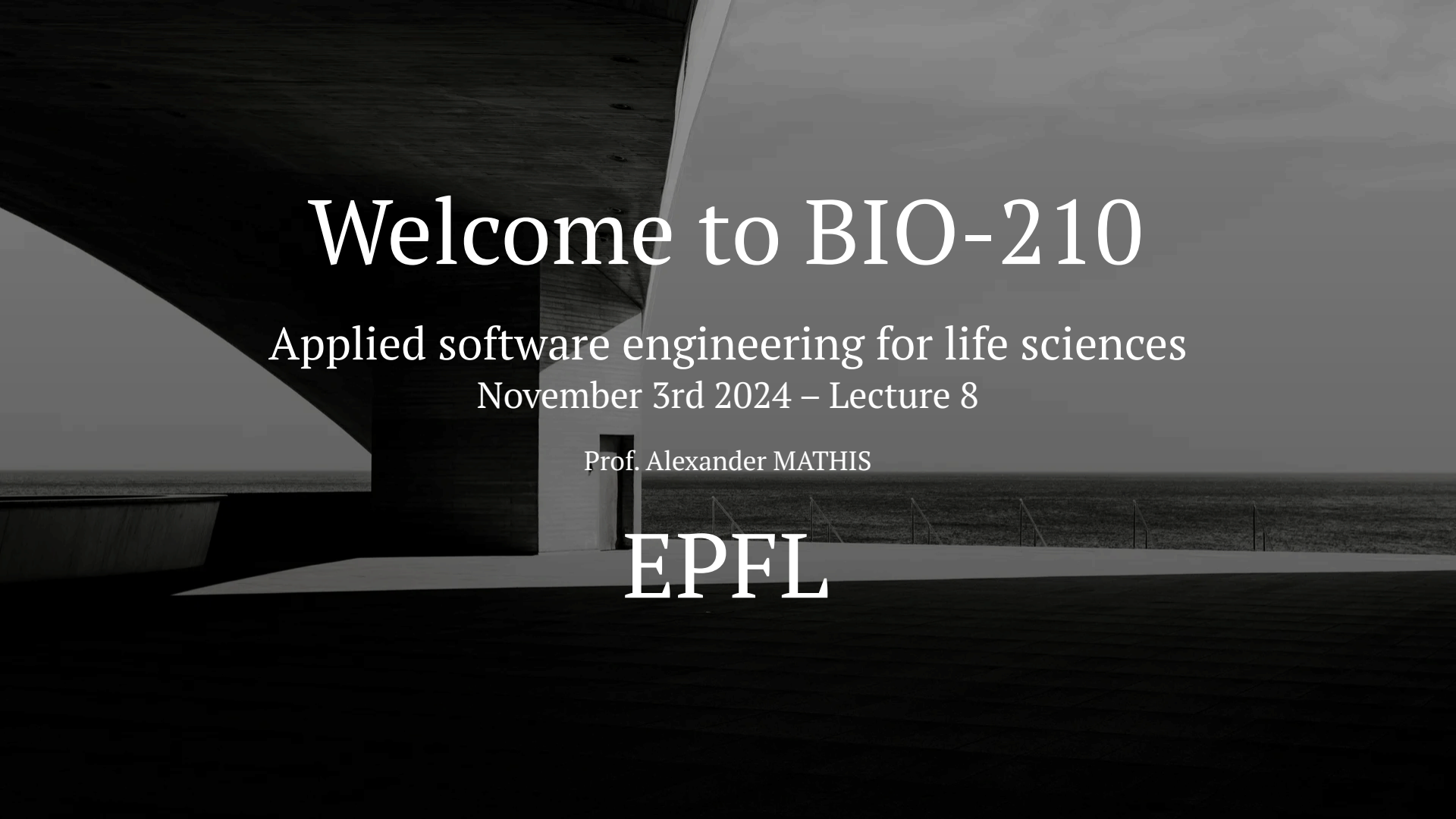


Welcome to BIO-210

Applied software engineering for life sciences

November 3rd 2024 – Lecture 8

Prof. Alexander MATHIS

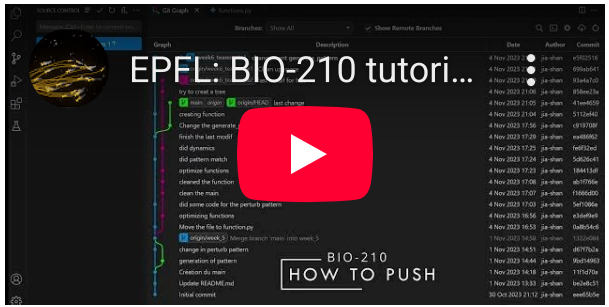
EPFL

Announcements

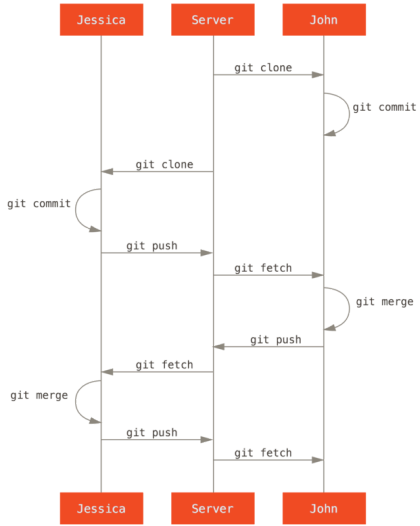
- you should have received a code review by the SA. Please let me know if you did not.
- v3 was due today (no code review, but discuss with your student assistants today)
- Try to submit in time, as we will have to reduce your score (for graded parts, e.g. v4 next week!)

Materials for GIT

- Jennifer Shan developed a video tutorial on how to use git in Visual Studio Code



Reminder: Multiple-developer git workflow

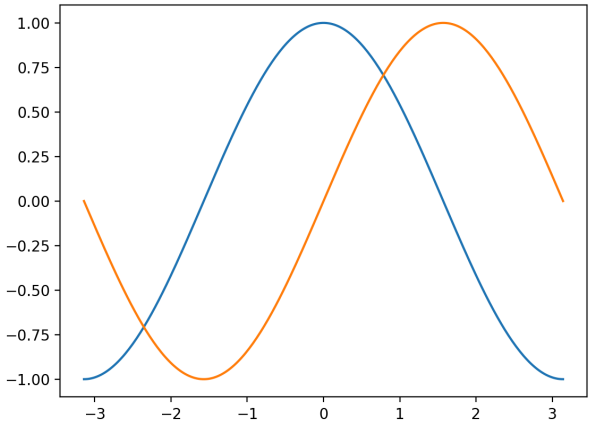


Source: git docs

If you are still struggling, ask the SAs & TAs for tutoring!

🧩🤔: How will this plot look?

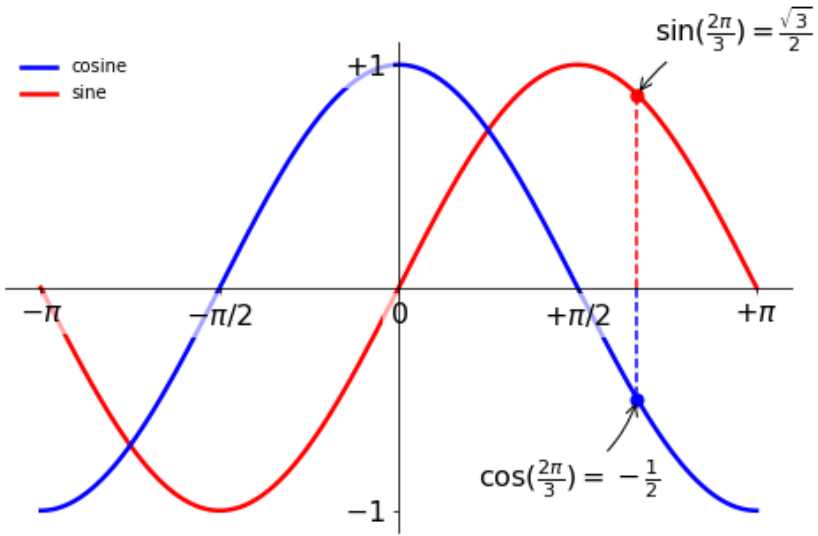
```
1   X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
2   C, S = np.cos(X), np.sin(X)
3   plt.plot(X,C)
4   plt.plot(X,S)
5   plt.show()
```



Source

However, don’t trust the defaults (especially if you want to share your plots!)

just a few more lines of code …



Source

Reminder: Check out the Matplotlib gallery

Tons of visual examples with code, e.g. matlab-unchained

🧩🤔: what should be in docstrings?

- **Short summary** (for basic, simple functions)

```
def add(a, b):
   """The sum of two numbers.

   """
```

- **Extended summary** contains among others:
  - a simple description (clarify functionality, not implementation details those belong to Notes)
  - parameters
  - returns
  - examples
  - notes
  - references

Details are here

Questions?

Testing

Why tests?

- Catch bugs before they become problems
- Make refactoring safer
- Document expected behavior
- Enable confident collaboration

Reminder: demo project

Consider a shortened version of `LotkaVolterraModel.py` (from here)

```python
import numpy as np

def dX_dt(X, a=1.0, b=0.1, c=1.5, d=0.75):
    """
    DOCSTRINGS OMMITTED!
    """
    return np.array([a * X[0] - b * X[0] * X[1], -c * X[1] + d * b * X[0] * X[1]])

def population_equilibrium(a=1.0, b=0.1, c=1.5, d=0.75):
    """
    DOCSTRINGS OMMITTED!
    """
    return np.zeros(2), np.array([c / (d * b), a / b])
```

Question: What is a population equilibrium?

Different forms of importing a module

Importing the module:

```
>>> import LotkaVolterraModel
```

Importing a specific function (from the module)

```
>>> from LotkaVolterraModel import dX_dt    # Import a specific function
```

Importing all functions (and `module.func1` is not required)

```
>>> from LotkaVolterraModel import *        # Import all functions
```

Importing the module (best way!)

```
>>> import LotkaVolterraModel
# Using functions (output not shown)
>>> LotkaVolterraModel.dX_dt(np.array([0,1]))
>>> LotkaVolterraModel.population_equilibrium()
```

Importing a specific function (from the module)

```
>>> from LotkaVolterraModel import dX_dt    # Import a specific function
>>> dX_dt(np.array([0,1]))                  # Output not shown
>>> population_equilibrium()
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-9ef8bb27b9eb> in <module>
----> 1 population_equilibrium()

NameError: name 'population_equilibrium' is not defined
```

Note: In this case, `LotkaVolterraModel.` is not required for using fuctions, but only imported functions are available.

Importing all functions (and `module.f1` is not required)

```
>>> from LotkaVolterraModel import *       # Import all functions (but namespace collision possible!)
>>> dX_dt(np.array([0,1]))                 # All functions are available
>>> population_equilibrium()               # No error, all functions availabe...
```

Note: unexpected namespace collision possible (as you import all functions)

Questions?

A (first) hand crafted test ...

```python
import numpy as np

def dX_dt(X, a=1.0, b=0.1, c=1.5, d=0.75):
    """
    DOCSTRINGS OMMITTED!
    """
    return np.array([a * X[0] - b * X[0] * X[1], -c * X[1] + d * b * X[0] * X[1]])

if __name__ == "__main__":
    if dX_dt(np.ones(2),1,0.1,1.5,.75) == np.array([ 0.9  , -1.425]):
        print("works!!")
```

🧩🤔 : What happens here and how can we solve it?

```
alex@mac demo-project % python3 LotkaVolterraModel.py
Traceback (most recent call last):
  File "/Users/alex/Code/Teaching/demo-project/LotkaVolterraModel.py", line 130, in <module>
    if dX_dt(np.ones(2),1,0.1,1.5,.75) == np.array([ 0.9  , -1.425]):
ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()
```

Fixing the comparison

```
import numpy as np

def dX_dt(X, a=1.0, b=0.1, c=1.5, d=0.75):
    """
    DOCSTRINGS OMMITTED!
    """
    return np.array([a * X[0] - b * X[0] * X[1], -c * X[1] + d * b * X[0] * X[1]])

if __name__ == "__main__":
    if (dX_dt(np.ones(2),1,0.1,1.5,.75) == np.array([ 0.9  , -1.425])).all():
        print("works!!")
```

```
alex@mac demo-project % python3 LotkaVolterraModel.py
works!!
```

… but such hand-crafted tests are clunky …

Doctests

`Doctest` is part of the Python Standard Library.

- The doctest module searches for strings that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown.

This allows:

- To check that a module's docstrings are up-to-date by verifying that all interactive examples still work as documented.
- To perform regression testing by verifying that interactive examples from a test file or a test object work as expected.

Using doctests I: Simply add examples

```
def dX_dt(X, a=1.0, b=0.1, c=1.5, d=0.75):
    """
    Computes the growth rate of fox and rabbit populations based on
    system state (X) and parameters (a,b,c,d)

    Parameters
    ----------
    X : array or tuple
        [prey_count, predator_count]
    Returns
    -------
    numpy array
        [change of prey_count, change of predator_count]

    Examples
    -------
    >>> dX_dt(np.ones(2),1,0.1,1.5,.75)
    array([ 0.9  , -1.425])
    >>> dX_dt(np.zeros(2),1,0.1,1.5,.75)
    array([0., 0.])
    """
    return np.array([a * X[0] - b * X[0] * X[1], -c * X[1] + d * b * X[0] * X[1]])
```

Using doctests II: running the tests

```
1  import numpy as np
2
3  def dX_dt(X, a=1.0, b=0.1, c=1.5, d=0.75):
4    ....
5
6  if __name__ == "__main__":
7      import doctest               # Importing the library
8      print("Starting doctests")   # not required (just for clarity in output)
9      doctest.testmod()            # Running the doctests (here for this module)
```

Using doctests III: output

```
alex@mac  demo-project % python3 LotkaVolterraModel.py
Starting doctests           # No output if they work!
# Running with detailed output:
alex@mac  demo-project % python3 LotkaVolterraModel.py -v
Starting doctests
Trying:
    dX_dt(np.ones(2),1,0.1,1.5,.75)
Expecting:
    array([ 0.9  , -1.425])
ok
Trying:
    dX_dt(np.zeros(2),1,0.1,1.5,.75)        # zero is a fixpoint
Expecting:
    array([0., 0.])
ok
1 items had no tests:
    __main__
1 items passed all tests:
   2 tests in __main__.dX_dt
2 tests in 2 items.
2 passed and 0 failed.
Test passed.
```

Putting an error (on purpose)...

```
1   def dX_dt(X, a=1.0, b=0.1, c=1.5, d=0.75):
2       """
3       OMMITTED!
4
5       Examples
6       -------
7       >>> dX_dt(np.zeros(2),1,0.1,1.5,.75)
8       array([0., 1.])
9       """
10      return np.array([a * X[0] - b * X[0] * X[1], -c * X[1] + d * b * X[0] * X[1]])
```

As `dX_dt(0) = 0`...

Running the tests again...

```
alex@mac  demo-project % python3 LotkaVolterraModel.py -v
Starting doctests
Trying:
    dX_dt(np.zeros(2),1,0.1,1.5,.75)
Expecting:
    array([0., 1.])
**********************************************************************
File "/Users/alex/Code/Teaching/demo-project/LotkaVolterraModel.py", line 35, in __main__.dX_dt
Failed example:
    dX_dt(np.zeros(2),1,0.1,1.5,.75)
Expected:
    array([0., 1.])
Got:
    array([0., 0.])
1 items had no tests:
    __main__
**********************************************************************
1 items had failures:
   1 of   1 in __main__.dX_dt
1 tests in 2 items.
0 passed and 1 failed.
***Test Failed*** 1 failures.
```

Questions?

Simple doctests for the major functions can be found in the demo-code.

- if you have multiple doctests (for multiple functions, it runs for all functions)
- make sure to have quick and simple tests
- make sure you update them (if necessary), when you change or expand features

🧩🤔: How to implement factorial in Python?

```
1  def factorial(n):
2    result = 1
3    factor = 2
4    while factor <= n:
5        result *= factor
6        factor += 1
7    return result
```

🧩🤔: What doctests should we add?

```python
# example_lecture8.py
def factorial(n):
    """
    Returns factorial of n, for positive integers.

    Examples          # NOT necessary (but good for docstrings)
    --------
    >>> [factorial(n) for n in range(11)]
    [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]
    >>> factorial(22)
    1124000727777607680000

    """
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Running the tests

```
alex@mac % python3 example_lecture8.py -v
Trying:
    [factorial(n) for n in range(11)]
Expecting:
    [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]
ok
Trying:
    factorial(22)
Expecting:
    1124000727777607680000
ok
1 items had no tests:
    __main__
1 items passed all tests:
   2 tests in __main__.factorial
2 tests in 2 items.
2 passed and 0 failed.
Test passed.
```

🧩🤔: what is going on here?

```
In [1]: import numpy as np

In [2]: X=np.ones(33)

In [3]: X.append(3)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-3-8be15a984d9d> in <module>
----> 1 X.append(3)

AttributeError: 'numpy.ndarray' object has no attribute 'append'
```

What are tracebacks?

A Python traceback is a report summarizing function calls in your code. When your program runs into an exception, Python will display the current traceback to help you diagnose the problem. Typical errors are `AttributeError ImportError IndexError KeyError NameError SyntaxError TypeError ValueError`. Examples:

```
In [1]: a,b=(2,3,4)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1-6ac6ff6f93e3> in <module>
----> 1 a,b=(2,3,4)

ValueError: too many values to unpack (expected 2)
In [2]: a=[2,4,5,6,33]
In [3]: a[22]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-3-b29842c8f55d> in <module>
----> 1 a[22]

IndexError: list index out of range
```

Read them from bottom-to-top!

Checking inputs: Raising exceptions

```
1   def factorial_withwarnings(n):    # Renamed for convenience (in my script)
2       import math
3       if not n >= 0:
4           raise ValueError("n must be >= 0")
5       if math.floor(n) != n:
6           raise ValueError("n must be an integer")
7       if n+1 == n:  # catch a value like 1e300;
8           raise OverflowError("n too large")
9
10      result = 1
11      factor = 2
12      while factor <= n:
13        result *= factor
14        factor += 1
15      return result
```

Doctests against error messages

Adding this function to example_lecture8.py

```
def factorial_withwarnings(n):
    '''
    >>> factorial_withwarnings(-1)
    Traceback (most recent call last):
      ...
    ValueError: n must be >= 0

    Factorials of floats are OK, but the float must be an exact integer:
    >>> factorial_withwarnings(30.1)
    Traceback (most recent call last):
      ...
    ValueError: n must be exact integer
    >>> factorial_withwarnings(1e100)
    Traceback (most recent call last):
      ...
    OverflowError: n too large
    '''
    CODE OMMITTED (see previous page)
```

Test report

```
alex@mac % python3 example_lecture8.py -v
Trying:
    [factorial(n) for n in range(11)]
Expecting:
    [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]
ok
.... OMMITTED ...
Trying:
    factorial_withwarnings(-1)
Expecting:
    Traceback (most recent call last):
      ...
    ValueError: n must be >= 0
ok
.... OMMITTED ...
1 items had no tests:
    __main__
2 items passed all tests:
   2 tests in __main__.factorial
   3 tests in __main__.factorial_withwarnings
5 tests in 3 items.
5 passed and 0 failed.
Test passed.
```

Summary of doctests

- Doctest allows you to write simple test routines, with practically zero overhead
- Doctests motivates the addition of examples, which improves your documentation
- Doctests make sure the documentation is up-to-date
- Not ideal to support complex function testing (as docs get cluttered, etc.)

For more info, see doctest docs

Pytest: a powerful testing framework

The library Pytest

- makes it easy to write small tests,
- yet also scales to support complex functional testing for applications and libraries.

Note: you can also test sizes/dimensions, return types, etc.

Pytest is not in the standard library. Check out the installation guide – you can install it (in the terminal) by typing:

```
alex@mac  % pip install pytest
```

Let's look at some examples!

A simple example

Writing tests (store this file below as `test_example.py`)

```python
def myfun(x):
  return 3*x

def test_myfun():
  assert myfun(6) == 18                    # The assert statement is the core of pytest
  assert myfun('EPFL') == 'EPFLEPFLEPFL'
  assert myfun(3.0) == 9.0
```

If you run `pytest` in the terminal ... (all `test_XYZ.py` and `XYZ_test.py` are excecuted)

```
alex@mac  % pytest
================================== test session starts ==================================
platform darwin -- Python 3.8.8, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /Users/alex/Code/Teaching/demo-project/abc
plugins: anyio-2.2.0
collected 1 item

test_example.py .                                                                [100%]

=================================== 1 passed in 0.01s ===================================
```

Putting an error … `assert myfun(3.0) == 9.1`

```
alex@mac  % pytest
================================= test session starts ==================================
platform darwin -- Python 3.8.8, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /Users/alex/Code/Teaching/demo-project/abc
plugins: anyio-2.2.0
collected 1 item

test_example.py F                                                                [100%]

======================================== FAILURES =========================================
_____________________________________ test_myfun ______________________________________

    def test_myfun():
      assert myfun(6) == 18
      assert myfun('EPFL') == 'EPFLEPFLEPFL'
>     assert myfun(3.0) == 9.1
E     assert 9.0 == 9.1
E      +  where 9.0 = myfun(3.0)

test_example.py:7: AssertionError
================================= short test summary info ==================================
FAILED test_example.py::test_myfun - assert 9.0 == 9.1
================================== 1 failed in 0.07s ===================================
```

Running the tests for the demo-project

Check out how we added tests in the demo-project!

```
alex@mac  % pytest
================================== test session starts ===================================
platform darwin -- Python 3.8.8, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /Users/alex/Code/Teaching/demo-project
plugins: anyio-2.2.0
collected 9 items

test_LVM.py .........                                                            [100%]

==================================== warnings summary ====================================
test_LVM.py::test_main
  /Users/alex/Code/Teaching/demo-project/test_LVM.py:84: DeprecationWarning: the imp module is deprecated in fa
    import imp

-- Docs: https://docs.pytest.org/en/stable/warnings.html
============================== 9 passed, 1 warning in 0.34s ==============================
```

All passed ...

🧩🤔: How are arguments matched to functions in python?

Python functions allow highly flexible calling patterns (Argument-matching modes).

- *Positionals*: matched left to right
- *Keywords*: matched by argument name; `name = value` syntax
- *Defaults*: specify values for optional arguments (that do not need to be passed)
- (*Varargs collecting*: pass arbitrarily many positional or keyword arguments)

Reminders:

```
>>> def f(x,y=2,z=3): print(x,y,z)    # x required, y and z optional!
>>> f(1)         # using defaults
(1,2,3)

>>> f(1,4)       # overwriting defaults by positional variable
(1,4,3)
>>> f(1,4,5)
(1,4,5)
# Mixed keyword and default example:
>>> f(1,z=55)    # a gets 1 by position, others by keyword
(1,2,55)
```

Coverage

How do you know if your testing is comprehensive?

- Coverage.py is a tool for measuring code coverage of Python programs.
- Coverage.py monitors your program, counting which parts of the code have been executed, then analyzes the source code to identify code that could have been executed, but was not. It reports the fraction of code that is tested (i.e. the **code coverage**)
- For testing, code coverage measures, *how well your tests are covering your source code*. Note, it does not measure the quality of your tests – this needs to be ensured by the programmer, i.e. you!

Installation:

```
1   alex@mac % pip install coverage
```

Measuring and reporting test coverage

```
alex@mac  % coverage run -m pytest
=================================== test session starts ====================================
platform darwin -- Python 3.9.7, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /Users/alex/Code/Teaching/demo-project
collected 9 items

test_LVM.py .........                                                              [100%]

==================================== warnings summary ======================================
test_LVM.py::test_main
  /Users/alex/Code/Teaching/demo-project/test_LVM.py:82: DeprecationWarning: the imp module is deprecated in fa
    import imp

-- Docs: https://docs.pytest.org/en/stable/warnings.html
============================== 9 passed, 1 warning in 0.20s ================================
alex@mac  % coverage report
Name                    Stmts   Miss  Cover
-------------------------------------------
LotkaVolterraModel.py      17      0   100%
test_LVM.py                48      0   100%
-------------------------------------------
TOTAL                      65      0   100%
```

Software development

- With a powerful set of tests incl. functional tests, one can automatically test the code base
- Tests can be run cross-platform, *and* with different dependencies
- *Performance regressions* can be avoided during further development

E.g. for DeepLabCut, we use GitHub actions to install and run our test-suite

Example test-run for a PR: https://github.com/DeepLabCut/DeepLabCut/pull/2374

Discussion: How do you assure high quality for your tests?

- write simple, readable tests
- write deterministic tests (or fix seeds)
- test one aspect per test (give them clear names!)

Further reading

- Pytest examples
- Effective python testing with Pytest
- Testing for numpy
- Integrating tests into the developing cycle is a popular appraoch in software development called: test-driven development

Software testing is an important aspect of software engineering, here we only scratched the surface by introducing doctests, unit tests (with pytest) and coverage. For further reading start with the Wikipedia article on software testing

	Date	Topic	Software version	Software releases	Feedback
0	08/09/2025	Python introduction I			
1	15/09/2025	Python introduction II			
2	22/09/2025	Public holiday			
3	29/09/2025	Git & GitHub (+ installation)			
4	06/10/2025	Project introduction	v1		
5	13/10/2025	Functionify	v2	v1	
6	20/10/2025	EPFL fall break			
7	27/10/2025	Visualization & documentation	v3	**v2**	code review (API)
8	03/11/2025	Unit-tests, functional tests	v4	v3	
9	10/11/2025	Code refactoring	v5	v4	graded (tests)
10	17/11/2025	Profiling & code optimization	v6	v5	code review
11	24/11/2025	Object oriented programming	v7	v6	graded (speed)
12	01/12/2025	Model analysis & project report	v8	v7	code review (OO)
13	08/12/2025	Work on project (no class)			
14	15/12/2025	Wrap up		v8	graded (project)

Best test-suite prize!

Grading:

- we will grade coverage, and quality of randomly selected tests
- announced in week 11 (as part of *v4* grading).

Today's summary

- testing (doctests, pytest, coverage, CI)
- Do you want an additional, open-source coding task? -> add a unit-test to DeepLabCut

Concepts	Definitions
Imports	Difference of `module.function()`, `function()`, etc.
Traceback	Report summarizing function calls, common for raising errors
Doctests	Allows automatic testing of examples in docstrings
Pytests	Allows creation of flexible and comprehensive unit-testing
Coverage	Assesses how much of your code is tested

Try out the commands in the python shell/notebooks! Practice is key.

After lunch:

- Monday 13 - 15: exercises working on v4 of your project
- No office hours today, please email me if you have questions