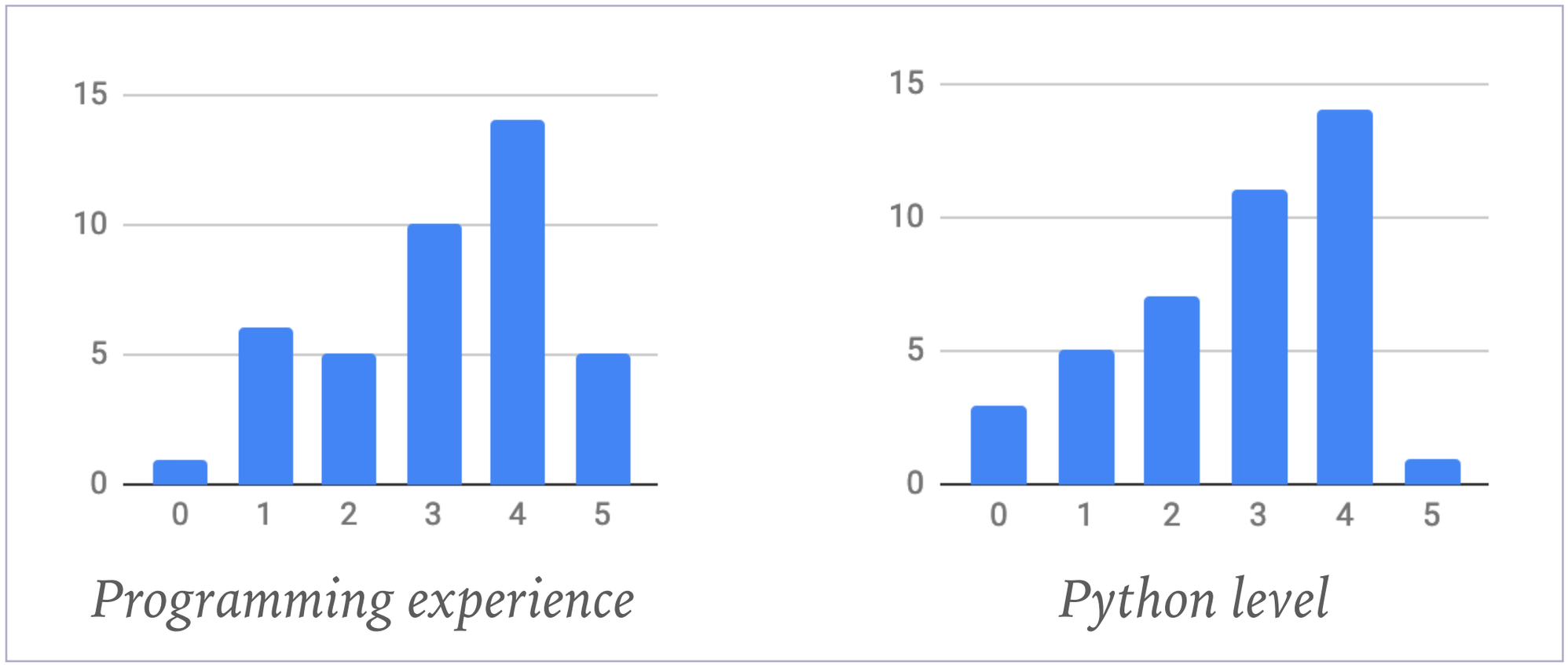
HYPOCRITICAL

# A ~~PRACTICAL~~ INTRODUCTION TO SOFTWARE TESTING IN PYTHON

*Habits for writing code that's less wrong*

# ABOUT YOU

- There are about 60 of you.
- There were 41 replies to the questionnaire.




*Programming experience* *Python level*

- 29 have not used tests, 12 have (but some “long ago”).
- So… this course is not for absolute beginners.

# MATRIX MOMENT

- Take the red pill and you can never again say you didn't know you should be testing your code (and data).
- Take the blue pill and you'll wake up in your old life, with plausible deniability.

# ABOUT ME

- PhD in computer science. Thesis on fitness landscapes, search algorithms.
- Programming for 45 years, including daily UNIX for 40 years.
- Four start-up companies.


- Institute of Virology, Charité Universitätsmedizin, Berlin.
- Zoology at the University of Cambridge.
- Computational virology for about 12 of the years since 2004.
- terence.jones@charite.de

# SIMPLE OBVIOUSLY-CORRECT CODE

```python
from datetime import datetime


def getToday():
    """
    Return today's date as a "YYYY-MM-DD" string.
    """
    return datetime.today().strftime('%Y-%m-%d')


def getYear(date):
    """
    Return the year as a "YYYY" string.

    @param date: A "YYYY-MM-DD" date string, as returned by getToday.
    """
    return date.split('-')[0]


def getDay(date):
    """
    Return the day as a "DD" string.

    @param date: A "YYYY-MM-DD" date string, as returned by getToday.
    """
    return date.split('-')[-1]
```

# RUNNING THE SIMPLE OBVIOUSLY-CORRECT CODE

get-today.py

```
1  #!/usr/bin/env python
2
3  from dates import getToday
4
5  print('Today is', getToday())
```

```
$ ./get-today.py
Today is 2022-06-09
```

# SOME HANDY SCRIPTS THAT USE THE LIBRARY

## get-day.py

```python
#!/usr/bin/env python

import sys
from dates import getToday, getDay

try:
    today = sys.argv[1]
except IndexError:
    today = getToday()

print('The day is', getDay(today))
```

## get-year.py

```python
#!/usr/bin/env python

import sys
from dates import getToday, getYear

try:
    today = sys.argv[1]
except IndexError:
    today = getToday()

print('The year is', getYear(today))
```

## is-it-still-2002.py

```python
#!/usr/bin/env python

from dates import getToday, getYear

if int(getYear(getToday())) == 2020:
    print('It is still 2020!')
else:
    print('2020 is but a memory!')
```

# COURSE OVERVIEW

- Thanks to Dr. Franziska Hufsky, scientific coordinator of EVBC, among other things.
- Two days, three hours a day.
- Both days divided into “why” and “how” and discussion.
- Initially, “why” is more important than “how”.
- Please do interrupt, but not too much!

# DAY ONE

- A quick look at some simple code. ✓
- Intro.
- Bioinformatic weirdness.
- Attitude - the most important factor?
- Why to test - what's in it for you?
- Testing frameworks and harnesses.
- Using pytest.
- Some code and tests.

# INTRO

- Testing is a huge subject. Can't do it all at once. Awareness grows on you.
- My background and motivation.
- Learning to program is learning a craft.
- I put "practical" in the title because I'm no expert.
- What do I mean by practical? Teaching bottom up. Often learning the hard way.
- High-level understanding and commitment is more important than knowing how to do it
- The state of the world (in bioinformatics software).
- Example code bases - have a look around.
- Testing is (necessarily?) a team sport.
- Actually this a course about being careful.
- Breadth rather than depth - not too many details.

# YOU KNOW WHAT'S REALLY WEIRD?

- Bioinformatics is full of complicated tasks, like string matching.
- But, bioinformatics software is very often not tested at all.
- The hypocrisy black hole - it's worse than politics.
- Many people know we should be testing, but not many people do it.
- In virology people are super careful in the lab, but not at the keyboard.
- Confidence sometimes seems inversely proportional to test failure likelihood.
- The more something is urgent, the faster we go, and the less we test.


*Hernán Kirsten*

# BUT IT CAN BE DONE

- Margaret Hamilton, age 33.
- Director of the Software Engineering Division of the MIT Instrumentation Laboratory.
- Developed on-board flight software for NASA's Apollo program.

https://en.wikipedia.org/wiki/Margaret_Hamilton_(software_engineer)

# ATTITUDE

- Pretend you're in the lab.
- Your cat is going to walk over your keyboard while your editor is open. You will fall asleep momentarily at the keyboard while typing. You will change something accidentally in your editor and not see it.
- Your code *will* have bugs.
- Testing is not just writing tests but also having the right mindset. Learning to question things, knowing what to test, behaving so that errors are less likely.
- I.e., learning how to make fewer mistakes, to reduce the probability that you will make mistakes or at least that you will overlook them.
- Debugging is to a large extent something from the past. Change in mindset from "fix the bug" to "fix the bug and make sure it never happens again”.
- If you're not writing tests as you debug, you're doing it wrong!

# ATTITUDE

- Admit your mistakes. Engender trust.
- Beyond schadenfreude.
- Celebrate that someone who makes so many errors is allowed to talk about them and that we have such good tools that you can still be a good programmer whose final code has few errors.
- You can't make yourself any smarter, but you can change the way you work, then relax and embrace your inner idiot.
- Have to do it in a team?
- Know someone will look at your code and complain if you've been lazy.

# BENEFITS OF TESTING

- The consequences of buggy code can be very serious.
- It feels great to know your code is perhaps correct.
- It will make you more confident and relaxed.
- Eliminate regression failures.
- It allows easy refactoring.
- Tests serve as documentation.
- Errors and uncertainties don't add, they multiply. But you can tame the complexity with linear tests.
- You can’t easily revert papers or code that people are using.

# TEST SUITES AND TEST RUNNERS

- Unittest: https://docs.python.org/3/library/unittest.html
- pytest: https://docs.pytest.org/en/7.1.x/

```
$ pip install pytest
```

pytest

Search Go

## About pytest

pytest is a mature full-featured Python testing tool that helps you write better programs.

## Contents

Home
Get started
How-to guides
Reference guides

# pytest: helps you write better programs

The `pytest` framework makes it easy to write small, readable tests, and can scale to support complex functional testing for applications and libraries.

`pytest` requires: Python 3.7+ or PyPy3.

**PyPI package name**: pytest

## A quick example

```python
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```

# DAY TWO

- Code coverage.
- Testing data.
- Passing file pointers, patching, skipping, mocking open.
- Limit use of Jupyter notebooks.
- Structure code to be easier to test.
- Pytest plugins, fixtures, etc.
- Writing functions and classes to support testing.

# TESTING DATA

- Not a traditional area of testing, but there's a ton you can do.
- Test data you're given - you will almost always find problems.
- Test your dictionaries and other static data structures.
- Test the output files you generate.
- Make it slightly annoying to deliberately change things, in order to detect accidental changes.
- Writing these tests is generally very easy.
- I have many examples.

# MOCKING OPEN

- If you can, avoid it by passing already open “files”.
- If not, use the mock function.
- What if you have to mock opening of several files?
- Examples.

# JUPYTER NOTEBOOKS

- You can't easily write tests in a notebook.
- It's way too easy / tempting to be lazy.
- You can't easily diff Jupyter notebook files.
- It's hard to do shared work with them (git is good at merging code, not so good at merging Jupyter JSON files)
- Sometimes there is weirdness.
- It's too easy to wind up with a notebook that no longer runs, and hard to figure out why.
- They are full of global variables!
- There is no separation of concerns (scripts, libraries, tests).

# JUPYTER NOTEBOOKS, IF YOU REALLY MUST

- Write functions inside Jupyter but then put them in files and import them in Jupyter (and write tests for them).
- Dynamically load changed code

```
%load_ext autoreload
%autoreload 2
```

- Run the notebook from scratch on the command line

```
env MY_JUPYTER_VAR=1 jupyter nbconvert \
    --execute notebook.ipynb --to html --stdout \
    --ExecutePreprocessor.timeout=180 > notebook.stdout
```

- Then write tests that check on what is done in the notebook (e.g., if it saves files, test what's in the files)

# CODING FOR TESTING

- First write a failing test, then make it pass.
- Write simple tests.
- Write fast tests (don't touch the files, network, databases).
- Structure code in a way that can more easily be tested.
- Divide into scripts, libraries, tests.
- Separate data from code.
- Functions that work on files should take an already open file pointer. Then you can easily test them with `StringIO`.

# (SLIGHTLY) MORE ADVANCED PYTEST

- Use pytest fixtures (warning: implicit magic).
- Use `-k` to run tests with names matching an expression.
- Use `--capture=tee-sys` to see standard output / error.
- To run tests on multiple CPUs, use `pytest -n auto`
  See https://pypi.org/project/pytest-xdist/
- Code coverage `--cov=module --cov-report html:cov_html`
  See https://pypi.org/project/pytest-cov/
- Use `--lf` (or `--last-failed`) to only re-run the last failures.
- Use `--ff` (or `--failed-first`) to run the last failures first and then the rest of the tests.