Schedule

Theme	Description
Introduction to testing	We will present the motivation (why) and basics (how) behind testing.
Demo: unit tests and refactoring in Python and R	We will show some code examples of written tests. This includes discussing the Python and R packages suitable for testing. Through an example code we show the process of refactoring and adding tests. Here we also discuss the cases when testing is appropriate with the involvement of the participants.
Break	
Practical: refactor and test code in groups	You will apply presented packages and write tests for either their own or example functions provided by us.
Practical: discussion	Insights and questions will be discussed with the whole room.
(Optional) Automated tests via GitHub Actions	If time allows, we will demonstrate how tests can be automated through popular open-source platforms, such as GitHub Actions

Before starting

https://github.com/OBIWOW/OBiWoW-2024/tree/main/09-Monday/improving-software-quality-in-bioinformatics-through-testing

Why testing?

How are you testing your own code?

How are you testing your own code?

- Here we are going to discuss:
  - Systematical
  - Reusable
  - Automated

  tests that aim to ensure the correct implementation.

- Not the model / science / biology.

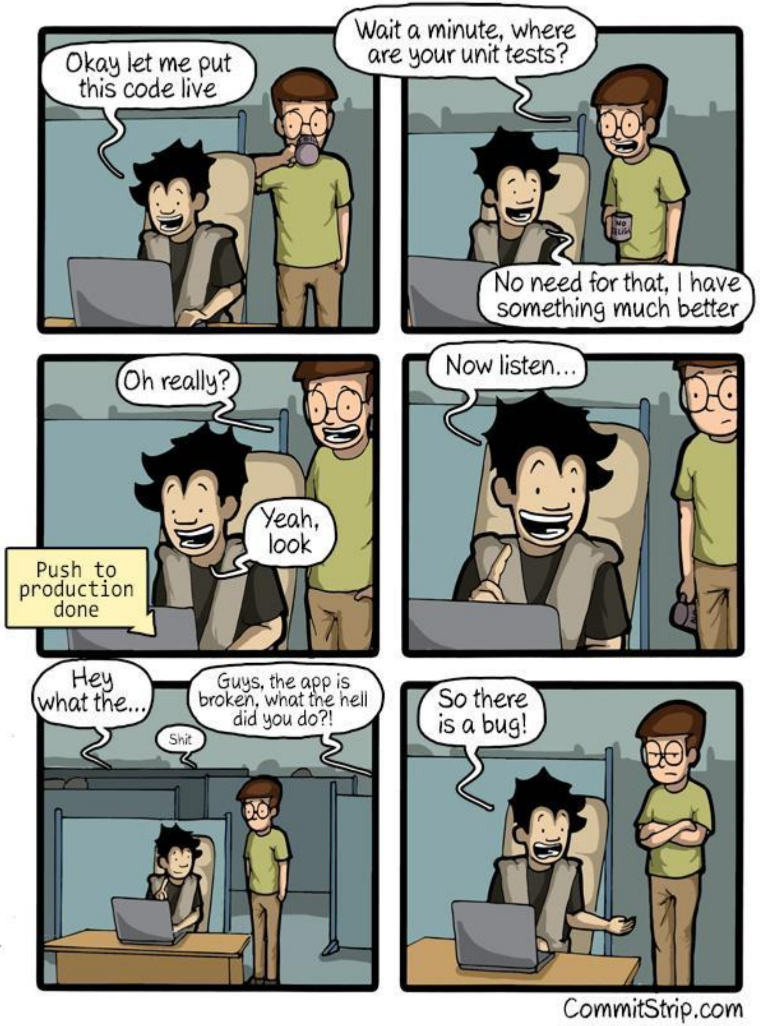
Bit more clarification before we begin

Testing the implementation

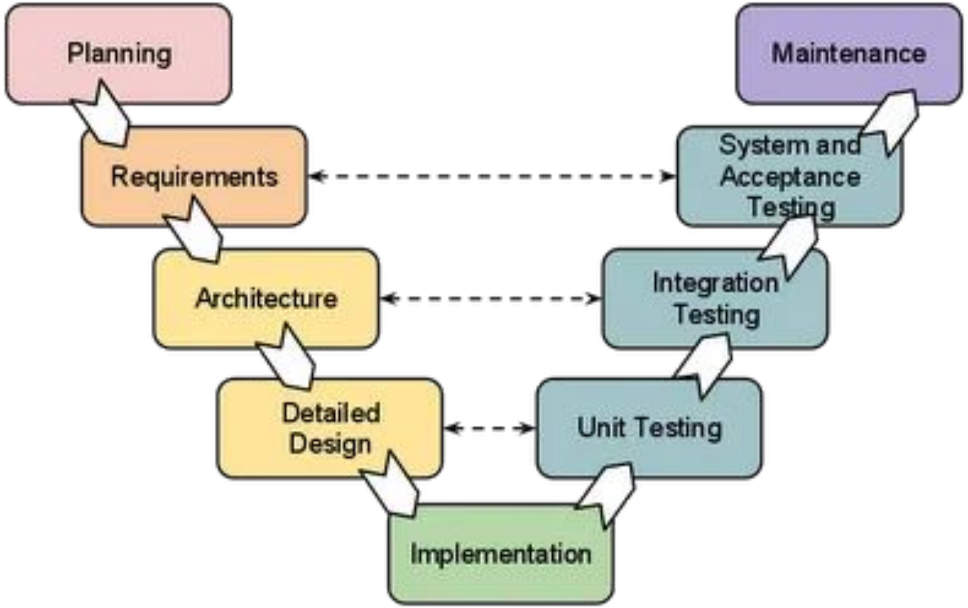
- **How does my code behave given a specific input?**
- Collected in a test folder outside of source
- Does not run with the main code (a.k.a. not “shipped”)

Sanity checks

- **Does the input complies with the code?**
- a.k.a. defensive programming
- Written in the source code
- Runs when the main code runs

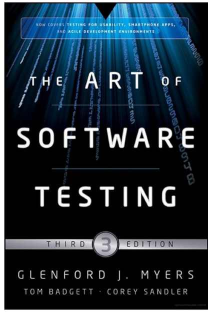

Okay let me put this code live
Wait a minute, where are your unit tests?
No need for that, I have something much better
Oh really?
Yeah, look
Push to production done
Now listen...
Hey what the...
Shit
Guys, the app is broken, what the hell did you do?!
So there is a bug!
CommitStrip.com

Software engineering, how to think about levels of tests



Principles of Testing

- Intent of finding errors - you should expect to find errors
- The expected result of a test case needs to be defined
- Test results should actually be checked
- Test cases must cover invalid and unexpected input conditions
- Create permanent test cases (regression testing - to not bring back old bugs when adding new features)
- Errors tend to cluster - if you found errors already, there’s a good chance to get more
- Untestable code is usually of poor quality



Myers, Glenford J., Corey Sandler, and Tom Badgett. *The art of software testing*. Hoboken, N.J: John Wiley & Sons, 2012.

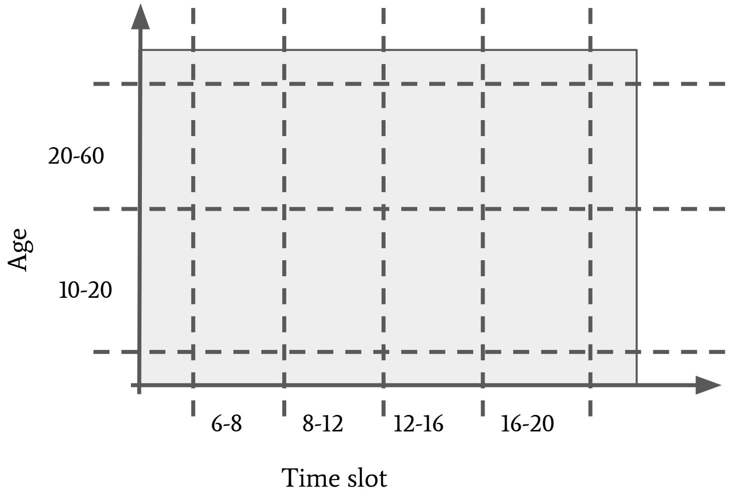
Levels of testing

- **Unit test:** Test individual piece of code (method or class)
- **Integration test:** When putting the units together, one has to make sure that their interaction does not produce error
- **Regression test:** Whenever any part of the system changes (maybe just fixing a bug) all test should run again
- **Acceptance test:** Testing with a customer to make sure that the application actually does what the customer wants

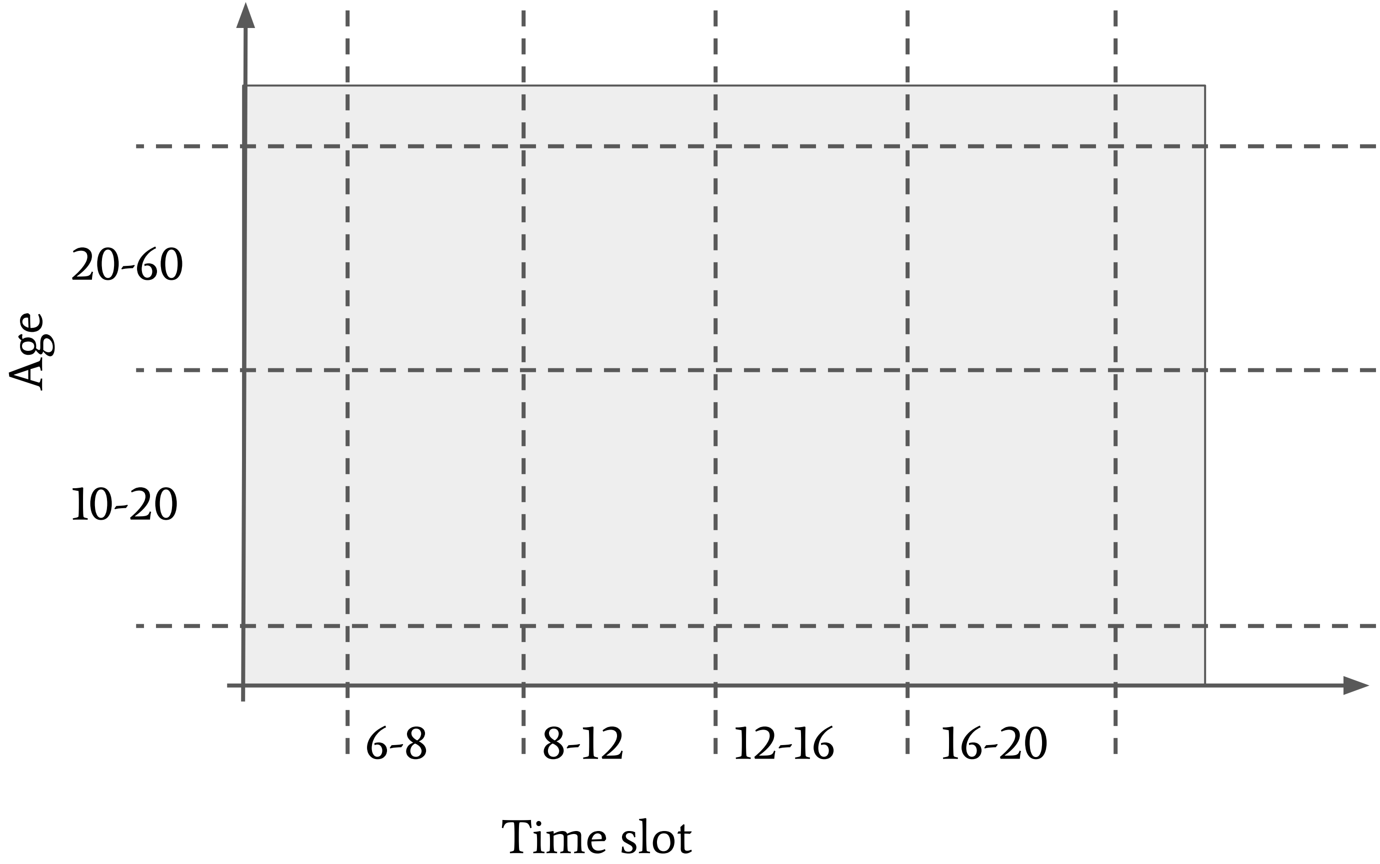
What to test?

Example:

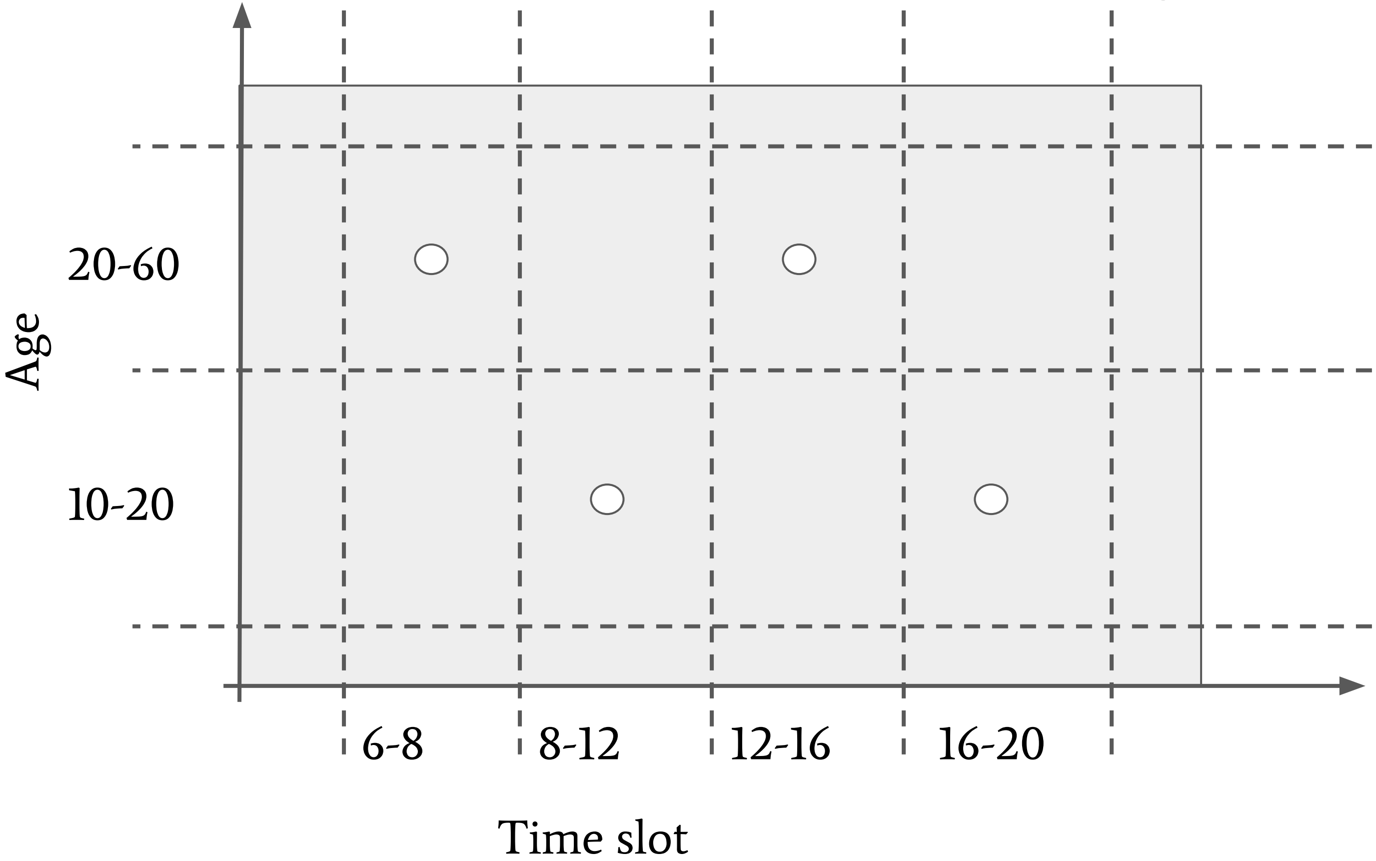
- There is a code that calculates the entrance fee for a swimming pool. There are two different age groups with different prices and 4 time slot per day that also affects the price.
- How would you test your system?



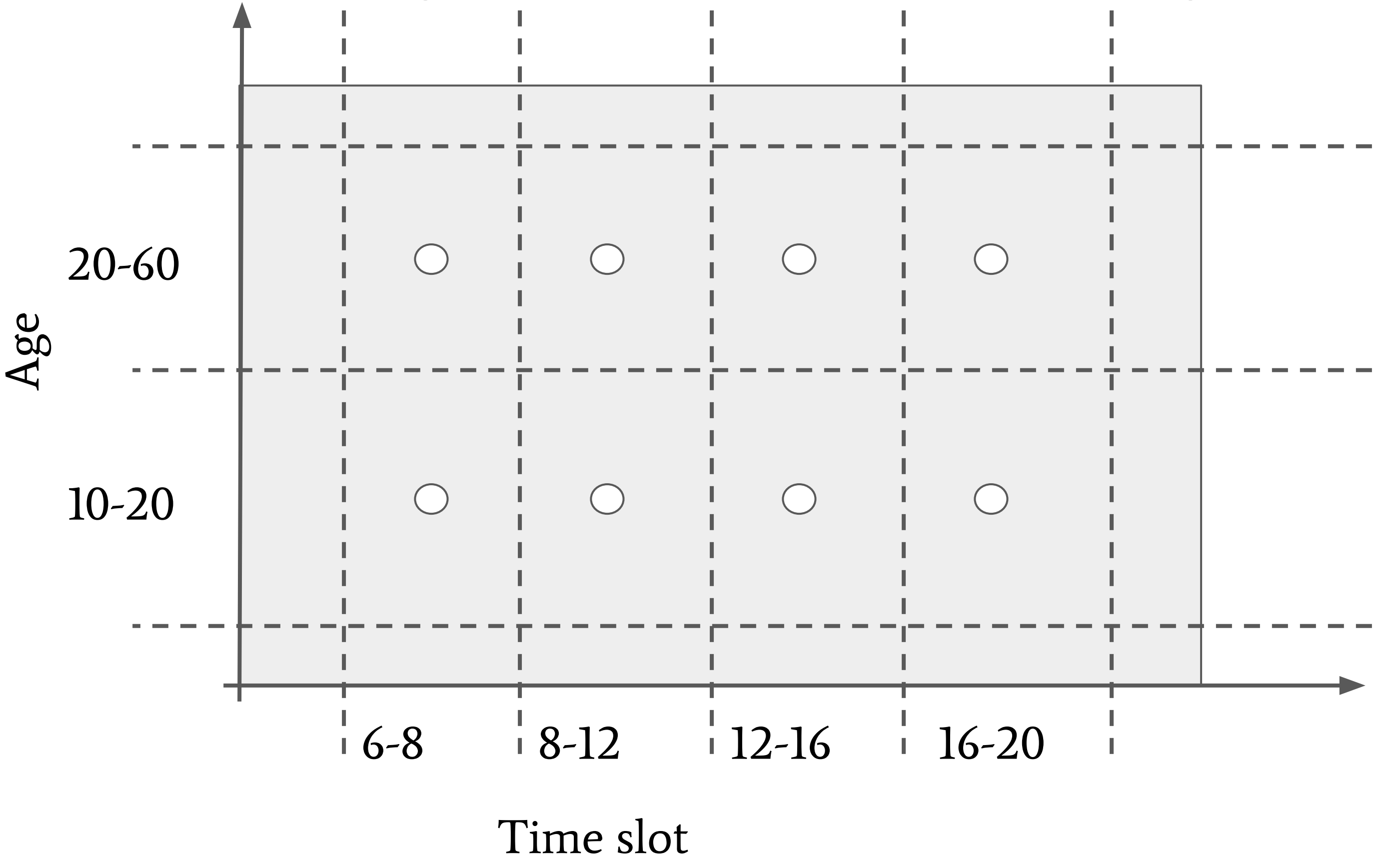
What to test?



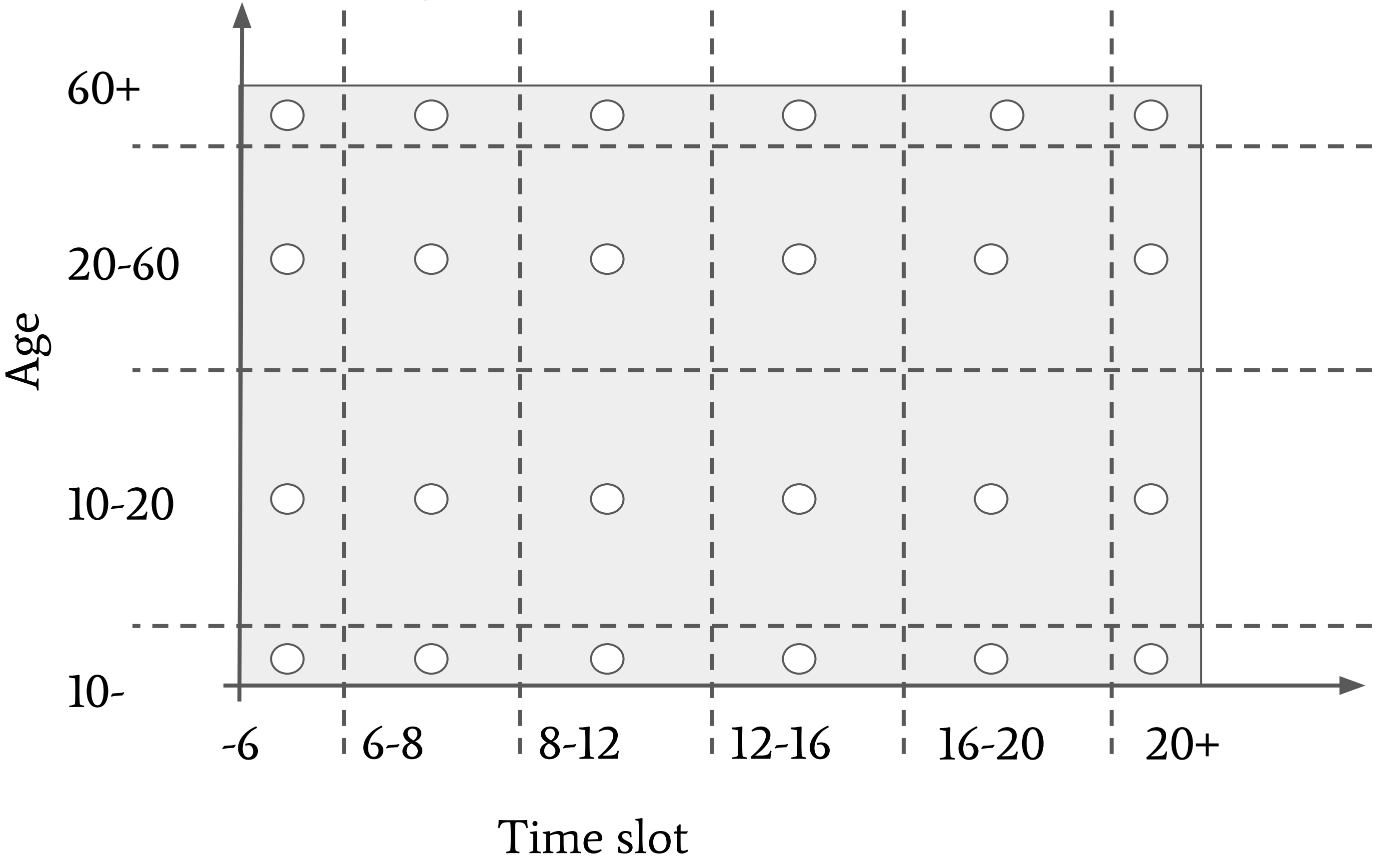
What to test? - Weak equivalence class testing



What to test? - Strong equivalence class testing



What to test? - Strong robust equivalence class testing



How is it relevant for us?

How is it relevant for us?

- What if my code has only a few functions?
  - Test those functions
  - Consider organizing your code to be more modular
  - Assertions can be useful on their own when processing data (eg. assertthat::assert_that(dim(enhancers)[1]>0))

How is it relevant for us?

- What if my code has only a few functions?
  - Test those functions
  - Consider organizing your code to be more modular
  - Assertions can be useful on their own when processing data (eg. assertthat::assert_that(dim(enhancers)[1]>0))
- How can I test my big data analysis code?
  - Maybe you don’t need extensive testing, especially if using borrowed functions (eg. base R, sklearn)
  - However, you can consider simulating or randomizing data to ensure you understand the behaviour of the tools

How is it relevant for us?

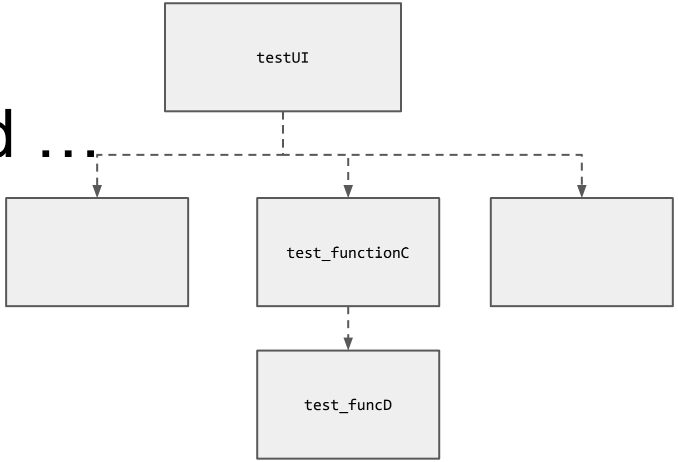
- What if my code has only a few functions?
  - Test those functions
  - Consider organizing your code to be more modular
  - Assertions can be useful on their own when processing data (eg. assertthat::assert_that(dim(enhancers)[1]>0))
- How can I test my big data analysis code?
  - Maybe you don't need extensive testing, especially if using borrowed functions (eg. base R, sklearn)
  - However, you can consider simulating or randomizing data to ensure you understand the behaviour of the tools
- When should I bother with this at all?
  - Quite easy to transform your ad-hoc tests to unit tests, but probably not needed during experimenting with tools and ideas
  - At some point (eg. when you settled on a method) it is a good idea to refactor your code and while doing so, it is a perfect opportunity to include tests

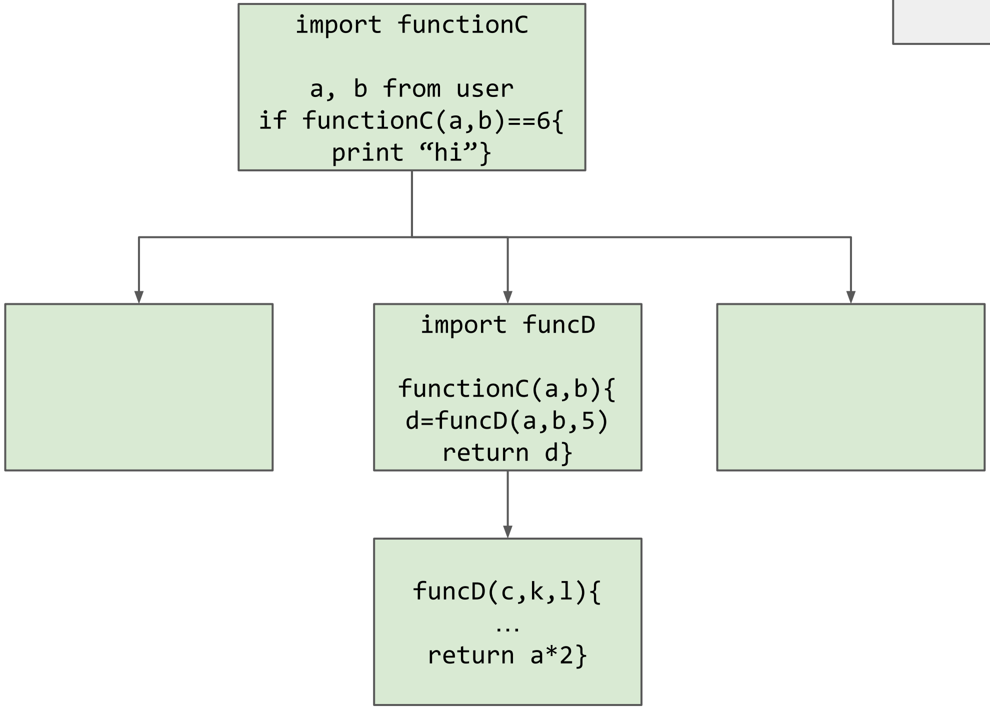
Basic intuition about unit testing

```
functionA(int a, int b){
c = a ** b
c -= a + b*b
return c
}
```

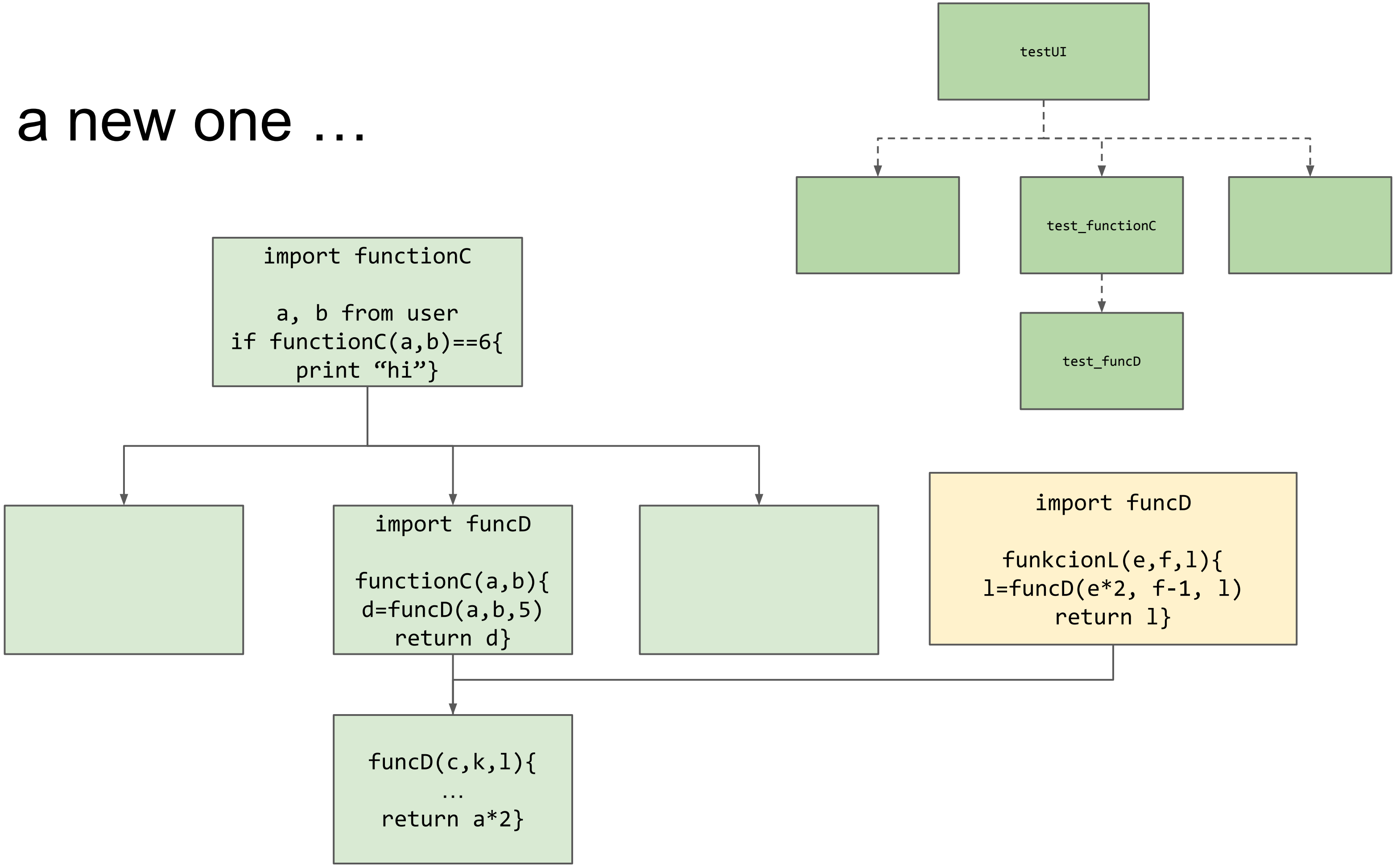
```
import functionA

test_functionA(){

a = 2
b = 5
expected_c = 5
actual_c = functionA(a=a, b=b)

assert expected_c == actual_c
}
```

When you have all of these units tested …

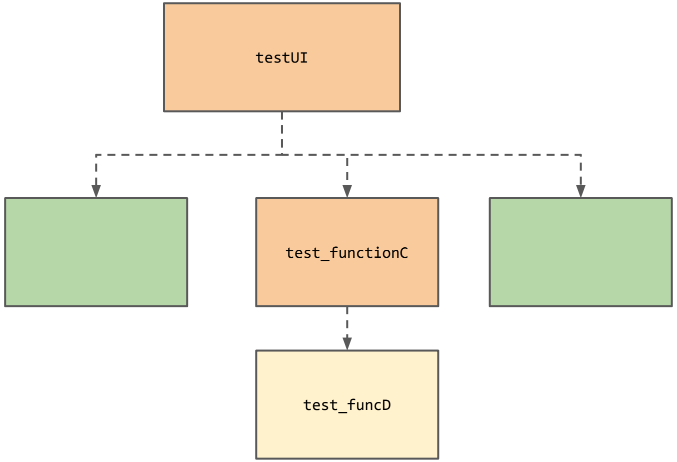


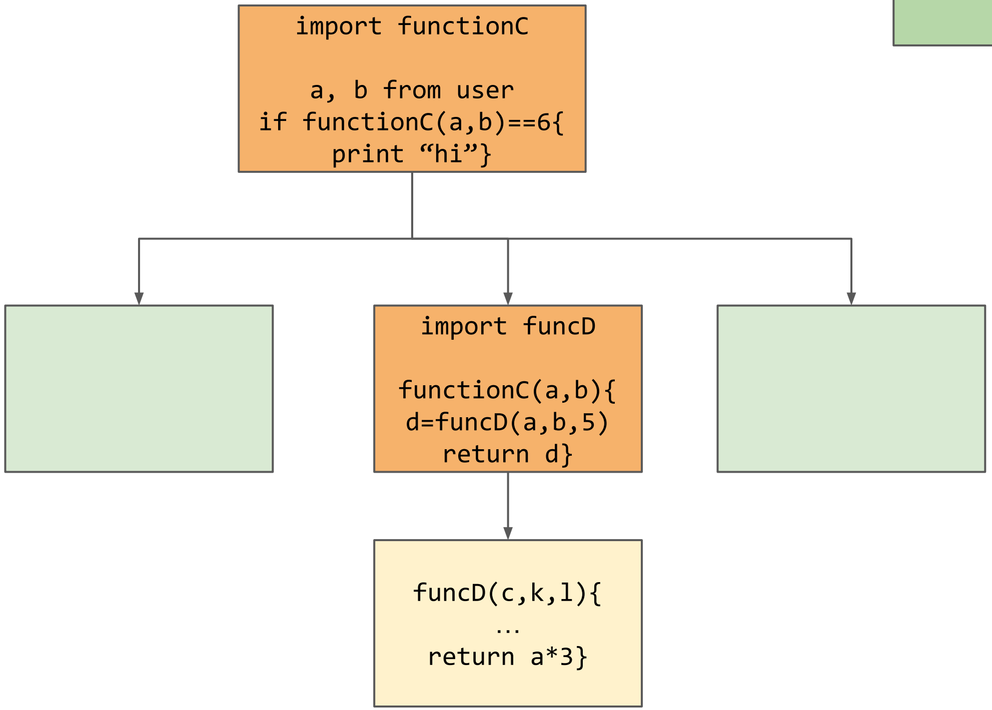


…adding a new one …

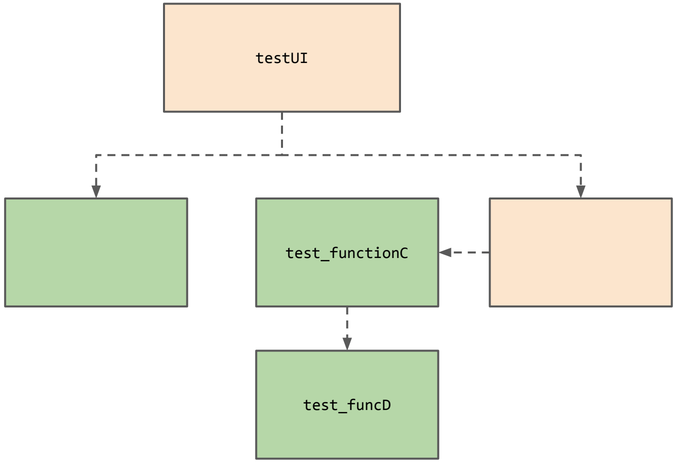


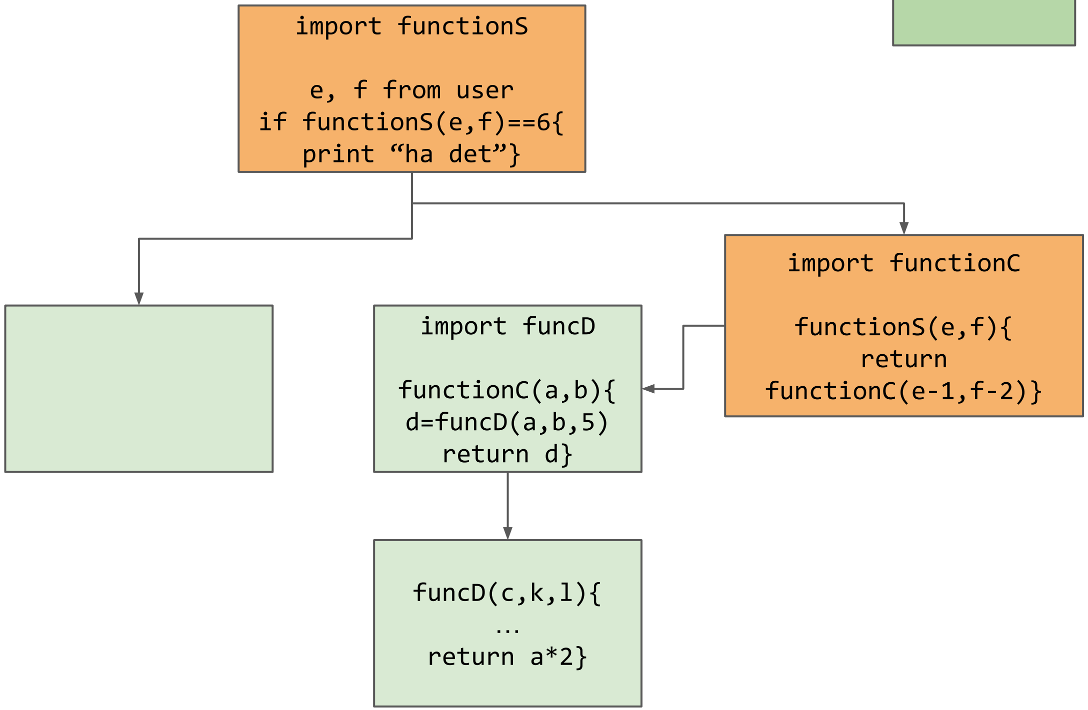
… or changing existing…





…or reorganizing is not that risky anymore





Some examples

- Python
  - Let’s take a look
  - https://docs.python.org/3/library/unittest.html
- R
  - Let’s take another look
  - https://r-pkgs.org/testing-basics.html

Let's start testing!

https://github.com/OBIWOW/OBiWoW-2024/tree/main/09-Monday/improving-software-quality-in-bioinformatics-through-testing

	Python / R
A	Usually does scripting
B	Usually writes functions
C	Likes to think about better/best ways of testing

resources:

https://docs.pytest.org/en/stable/

https://r-pkgs.org/testing-basics.html

Further information

- [Continuous integration]
- [Test driven development]
- [Mocking]
- Simulated data
- Debugging (what to do when you found a bug)

Questions?

Please fill in the evaluation form