

# RSE day 3



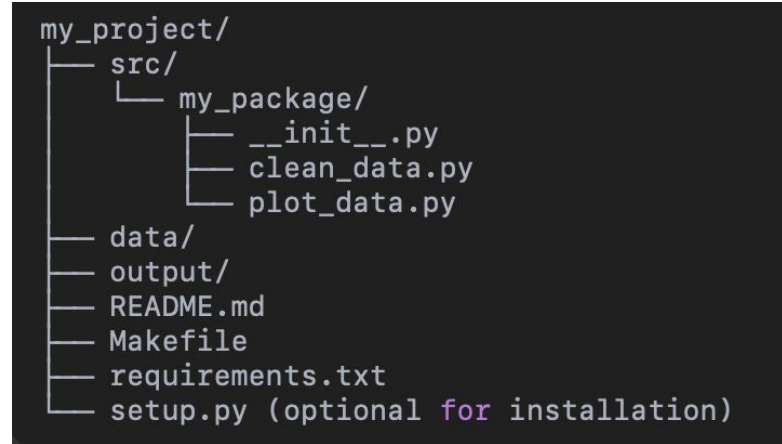
DIGITAL RESEARCH  
ACADEMY

# Packaging

Python:

- Can be versioned
- Can be uploaded to pip
- Can be (re) used by yourself and others
- A folder that contains an empty `__init__.py` file is considered a package.
- Functions can be imported using:

```
from .clean_data import clean_dataset
from .plot_data import plot_study_hours
```



```
from my_package.clean_data import clean_dataset
```

# Running python scripts from the command line

- Running functions from python scripts from the command line requires another “dunder” method, `__main__`
- 
- This method (similar to other main methods) provides an entry point for execution and allows to run the file from the command line
- 
- Additional arguments can be passed using the ``sys`` standard library.

```
```python
if __name__ == "__main__":
    import sys

    clean_data(sys.argv[1], sys.argv[2]) #
    adjust name if needed
```
```

<https://realpython.com/python-magic-methods/>

# User interaction with your code

Files that help users to interact with your code:

- [Contributing.md](#): How can users contribute?
- [License.md](#): How can the code be re-used?
- Install information: How can the software be installed?
- User tutorials: Give examples on how to use the code.

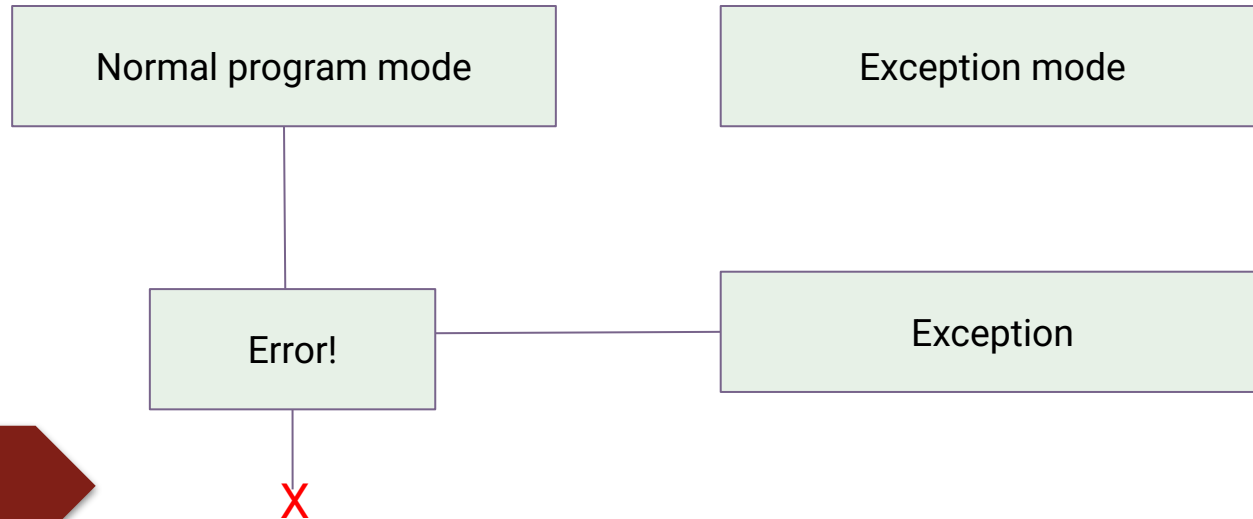
Example repositories:

Nilearn

PhysIO

# Error handling

- The purpose of errors is to **halt/disrupt the program** if something goes wrong/some condition is not met.
- The program switches from normal mode to error mode. All operations are being paused to handle the exception.



# Error handling

## Different types of errors:

- Syntax errors - > parser throws an error due to incorrect syntax
- Exceptions - > syntactically correct code results in an error

```
>>> print(0 / 0))
File "<stdin>", line 1
    print(0 / 0))
            ^
SyntaxError: unmatched ')'
```

```
>>> print(0 / 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

# Error handling

- In Python, exceptions follow the **raise Exception + Message** format:

```
number = 10
if number > 5:
    raise Exception(f"The number should not exceed 5. ({number=})")
print(number)
```

- To be more specific, python (and many other languages) have a variety of build in [exception types](#)

You can write your own exceptions if you don't find a fitting one (requires OOP).

# Error handling

- In MATLAB, exceptions are created via instances of MException objects that are created during the runtime of the program and consists of:
  - An errorID
  - An error Message
- The throw() statement is used to raise the corresponding exception

```
function indexIntoArray(A,idx)

% 1) Detect the error.
try
    A(idx)
catch

    % 2) Construct an MException object to represent the error.
    errID = 'MYFUN:BadIndex';
    msg = 'Unable to index into array.';
    baseException = MException(errID,msg);

    % 3) Store any information contributing to the error.
    if nargin < 2
        causeException = MException('MATLAB:notEnoughInputs','Not enough input arguments.');
```

---

```
        baseException = addCause(baseException,causeException);

    % 4) Suggest a correction, if possible.
    if(nargin > 1)
        exceptionCorrection = matlab.lang.correction.AppendArgumentsCorrection('1');
        baseException = baseException.addCorrection(exceptionCorrection);
    end

    throw(baseException);
end
```

Error using indexIntoArray  
Unable to index into array.

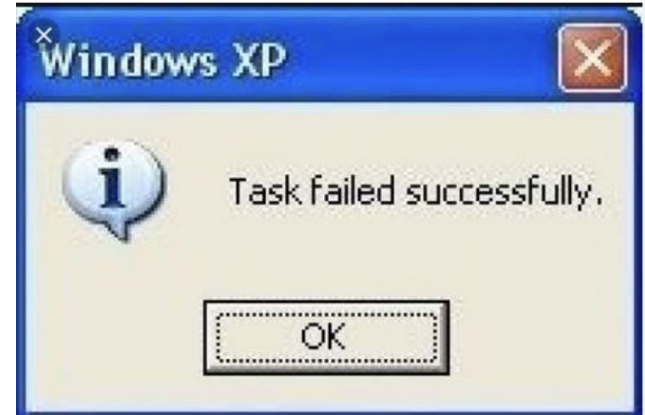
Caused by:  
Error using assert  
Indexing array is not numeric.  
Indexing array is too large.



# Good error messages

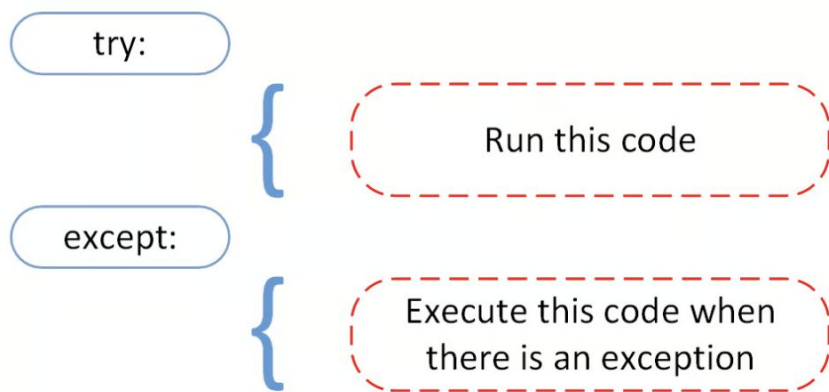
What makes a good error message?

- Clear: Say **what went wrong**
- Specific: Say **where** it went wrong
- Helpful: Suggest **what to do next**, if possible
- Polite: Don't blame the user

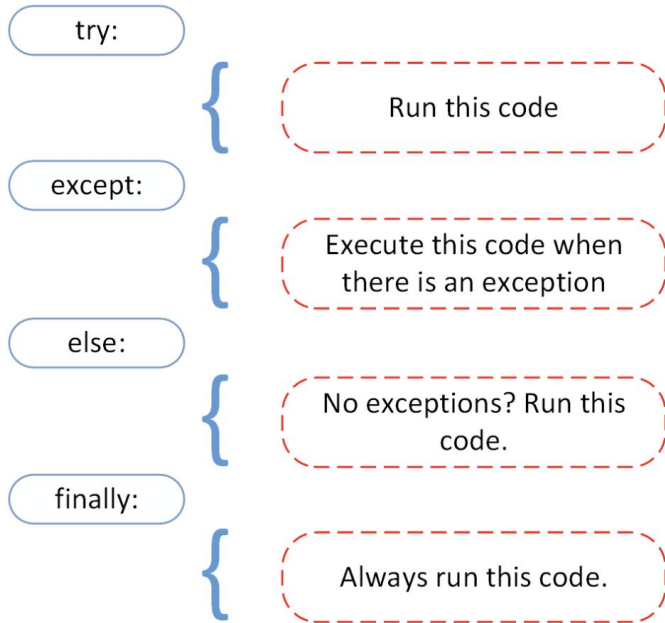


# Error handling

Exceptions can be wrapped into try - except blocks:



# Error handling



```
# ...  
  
try:  
    linux_interaction()  
except RuntimeError as error:  
    print(error)  
else:  
    try:  
        with open("file.log") as file:  
            read_data = file.read()  
    except FileNotFoundError as fnf_error:  
        print(fnf_error)  
finally:  
    print("Cleaning up, irrespective of any exceptions.")
```

# Debugging

Debugging = the process of identifying, isolating, and fixing errors (bugs) in your code.

In Jupyter/Colab:

- `%debug` after an exception opens a post-mortem debugger.
- `%pdb` on automatically triggers the debugger on errors.

In IDEs:

- VS Code, Matlab have full visual debugging support (breakpoints, step into, watch variables)

# Debugging

Breakpoints:

- Mark locations where code should be stopped. Program execution is halted, the current state of all variables can be observed (watch variables)

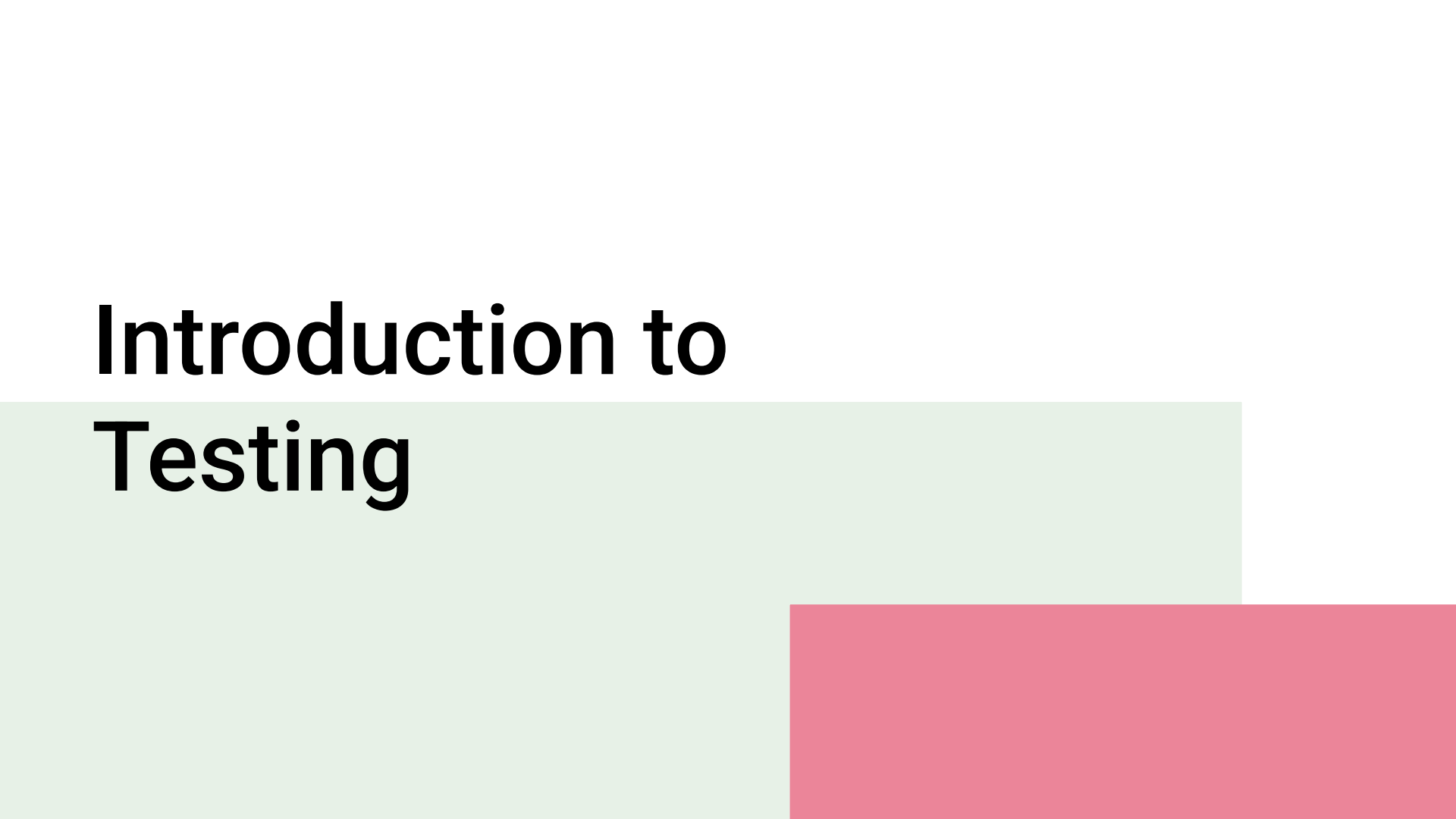
Step into:

- Allows to sequentially progress with the program (for example, step into the next function)

# Outline

| DAY 3                           |       |   |       |          |
|---------------------------------|-------|---|-------|----------|
| Topic                           | Time  |   |       | Duration |
| Welcome + Introduction          | 9:00  | - | 9:15  | 0:15     |
| Introduction to testing         | 9:15  | - | 9:45  | 0:30     |
| Break                           | 9:45  | - | 10:00 | 0:15     |
| Testing exercise                | 10:00 | - | 10:45 | 0:45     |
| Break                           | 10:45 | - | 11:00 | 0:15     |
| A very quick introduction to Cx | 11:00 | - | 11:35 | 0:35     |
| Wrap-up                         | 11:35 | - | 12:00 | 0:25     |

# Introduction to Testing

The background features a light green rectangular area on the left and a pink rectangular area on the right, both positioned below the title.

# It's normal to make mistakes...



Science

Current Issue

First release papers

Archive

About ▾

Submit manuscript

GET OUR E-ALERTS

🏠 | NEWS OF THE WEEK



## A Scientist's Nightmare: Software Problem Leads to Five Retractions

GREG MILLER [Authors Info & Affiliations](#)

SCIENCE • 22 Dec 2006 • Vol 314, Issue 5807 • pp. 1856-1857 • DOI: [10.1126/science.314.5807.1856](https://doi.org/10.1126/science.314.5807.1856)

📄 2,921 🗣️ 120



Until recently, Geoffrey Chang's career was on a trajectory most young scientists only dream about. In 1999, at the age of 28, the protein crystallographer landed a faculty position at the prestigious Scripps Research Institute in San Diego, California. The next year, in a ceremony at the White House, Chang received a Presidential Early Career Award for Scientists and Engineers, the country's highest honor for young researchers. His lab generated a stream of high-profile papers detailing the molecular structures of important proteins embedded in cell membranes.

Then the dream turned into a nightmare. In September, Swiss researchers published a

### CURRENT ISSUE





# It's normal to make mistakes...

WIRED

SECURITY POLITICS GEAR THE BIG STORY BUSINESS SCIENCE CULTURE IDEAS MERCH

SIGN IN

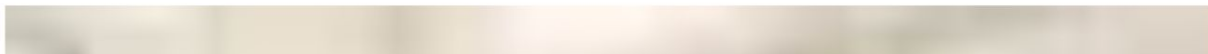
SUBSCRIBE

LISA GROSSMAN

NOV 10, 2010 7:00 AM

## Nov. 10, 1999: Metric Math Mistake Muffed Mars Meteorology Mission

1999: A disaster investigation board reports that NASA's Mars Climate Orbiter burned up in the Martian atmosphere because engineers failed to convert units from English to metric. Mars Photo Galleries: Where Will Next Mars Rover Land? Exotic New Mars Images From Orbiting Telephoto Studio Strange Places on Mars: What Do You Want to See Next? [...]



# It's normal to make mistakes ...

Comment | [Open access](#) | Published: 23 August 2016

## Gene name errors are widespread in the scientific literature

[Mark Ziemann](#), [Yotam Eren](#) & [Assam El-Osta](#) 

[Genome Biology](#) **17**, Article number: 177 (2016) | [Cite this article](#)

**160k** Accesses | **2665** Altmetric | [Metrics](#)

### Abstract

---

The spreadsheet software Microsoft Excel, when used with default settings, is known to convert gene names to dates and floating-point numbers. A programmatic scan of leading genomics journals reveals that approximately one-fifth of papers with supplementary Excel gene lists contain erroneous gene name conversions.

# So you better expect them: Tests

## What are Software Tests?

- Software tests are functions that certify the functionality of your code.
- They form a separate code base with your code and are run **independently/after/on** your code base

## Verification vs. validation

- **Validation:** This is the process of ensuring the input and output parameters of a product. It answers this question: *Are we building the product correctly?*
- **Verification:** This is the process of checking whether the product fulfills the specified requirements. Verifications answer question: *Are we building the right product?*
- Software test aim at verification, not at validation

# Assumptions and silent failure

The issue:

- The correct execution of these programmes is based on **assumptions**
  - Example 1: the order of columns in the spreadsheet
  - Example 2: the unit of the measurement
  - Example 3: the correct naming of the columns

THESE ASSUMPTIONS WERE NOT TESTED

WORSE: the program executed and failed silently.

TESTS allow us to check assumptions and to FAIL LOUDLY when the assumptions are not met

# Commonly used validation methods that are not tests

Print statements

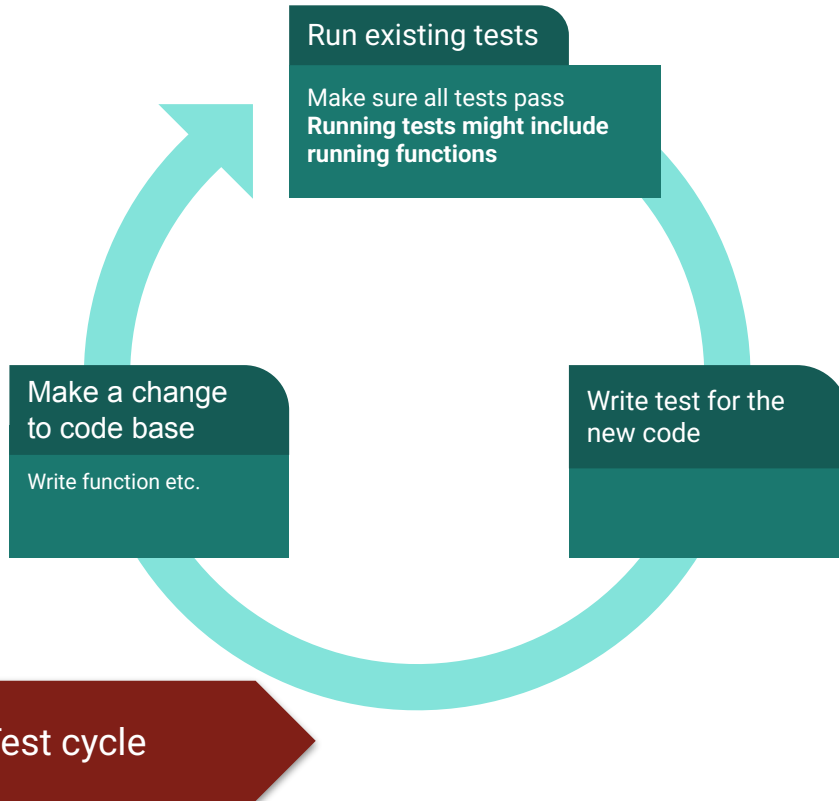
Plotting images/Visual inspection

Abort statements

Issues:

- Might get lost in noise of terminal output
- Require expert knowledge to validate them
- All of them are executed at runtime and cannot be run separately from the code

# The test cycle



- Tests run you functions independently of what is before and after
- This enforces modularity on your functions & code

# Different types of tests - as part of the test cycle

Unit testing

Integration testing

Regression testing

System testing

Smoke testing

Runtime testing

# Unit tests

A type of testing in which the logic of the smallest components or objects of software is tested. Tests small parts of your code. This is predominantly the first type of testing performed in the development life cycle.

Example: You want to make a ball pen

- You test the cap, the ball, the ink etc SEPARATELY



# Unit tests

A type of testing in which the logic of the smallest components or objects of software is tested. Tests small parts of your code. This is predominantly the first type of testing performed in the development life cycle.

## Pros:

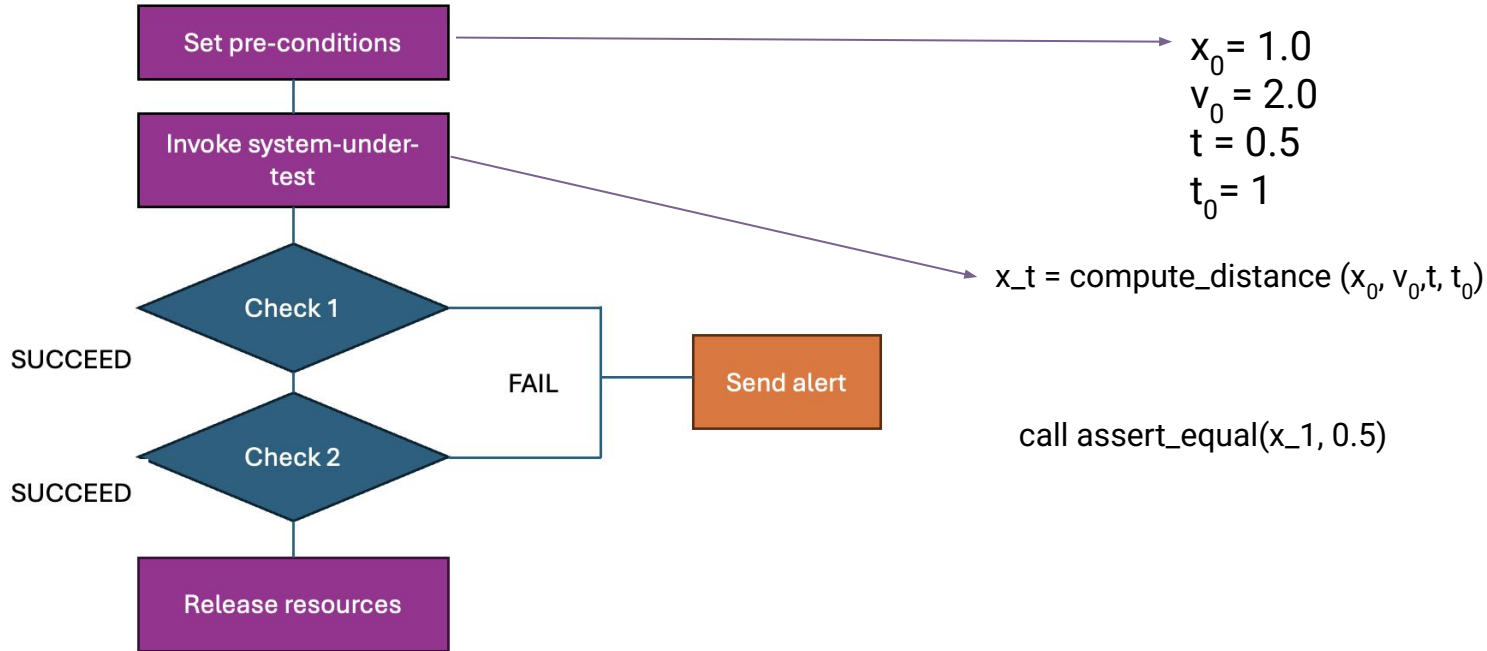
- Small, easy to execute
- Easy to write IF code base is organized in small functions
- Help to locate bugs precisely, as small part of code base is tests
- Hermetic

## Cons:

- Does not test how different parts of the code base interact with each other
- Harder to write if code is complex

# Anatomy of a unit test

$$x(t) = x_0 + v_0(t - t_0) + 1/2a(t - t_0)^2$$



# Unit tests - practically

What are attributes of good unit tests

- Silent in the case of success
- Automated and repeatable
- Independent (no side effects)
- Transparent (obvious, but not tautological)
- Narrow/precise
- Orthogonal (one bug => one failing test)
- small/frugal (memory, resources)

Should cover the entire code base

# Integration tests

A testing technique in which the communication logic of the individual software components or services are combined and tested as a group.

While unit tests focus on the functionality of individual parts, the focus of integration tests should be on the correct **interaction** between different parts.

Important when different people work on different parts of a system.

## Pros:

- Tests the interaction between parts, not only parts themselves

## Cons:

- Give only broad idea of where the failure happened or what caused it

# Integration tests

A testing technique in which the communication logic of the individual software components or services are combined and tested as a group.

While unit tests focus on the functionality of individual parts, the focus of integration tests should be on the correct interaction between different parts.

Important when different people work on different parts of a system.

Example: You want to make a ball pen

- You write with the ball pen

# Types of integration test approaches

- Big Bang: the integration of all units of a system are tested at the same time
  - Common when testers get a full new code base
- Top down: the integration of greater top units are tested first, then the integration of smaller bottom units
- Bottom up: the integration of smaller units is tested first, then the integration of higher level units/parts
  
- test stubs might (mocks, fixtures) need to be used, as the focus is on integration (aka the test should fail due to the interaction of different parts, not because of a failure of a lower order test)

# Regression tests

Style of testing that focuses on retesting after changes are made or a bug has been found. The results of tests after the changes are compared to the results before, and errors are raised if these are different.

- Very obviously important when you work in a team
- Written when code is functioning: record the output and save it
  - This output is then compared against the output of running the code later, after changes have been made

Example: You want to make a pen:

- You change the ink. So you write before and after to see whether the pen still works as before

# Regression tests

- Bug regression: We retest a specific bug that has been allegedly fixed.
- Old fix regression testing: We retest several old bugs that were fixed, to see if they are back. (This is the classical notion of regression: the program has regressed to a bad state.)
- General functional regression: We retest the project broadly, including areas that worked before, to see whether more recent changes have destabilized working code

## Pros:

- Good sanity check when working with more than one person

## Cons:

- Regression tests show that the code produces correct results, not necessarily that the code is working correctly
- You need to update the reference examples when the code changes fundamentally



# System tests (end-to-end testing)

Run a program from end to end.

Can be automated, for example: run every night

Pros:

- Easy to implement
- Works well even with complex code

Cons:

- Can be resource-consuming (time, memory) etc to run

# Smoke tests

A testing technique that is primarily used to check the core features of a product when there is a change introduced or before releasing it to a wider audience.

Run at the beginning of a test cycle

Checks for the bare minimum and big red flags

- Does the code compile

Pros:

- Easy to implement

Cons:

- Not specific
- Does not check whether results are correct

# Runtime testing

Tests that are run as part of the program itself

If statements that have an abort part

```
population = population + people_born - people_died

// test that the population is positive
if (population < 0):
    error( 'The number of people can never be negative' )
```

# Runtime testing

internal checks within functions that verify that their inputs and outputs are valid

```
function add_arrays( array1, array2 ):  
  
    // test that the arrays have the same size  
    if (array1.size() != array2.size()):  
        error( 'The arrays have different sizes!' )  
  
    output = array1 + array2  
  
    if (output.size() != array1.size()):  
        error( 'The output array has the wrong size!' )  
  
    return output
```

# Runtime testing

internal checks within functions that verify that their inputs and outputs are valid

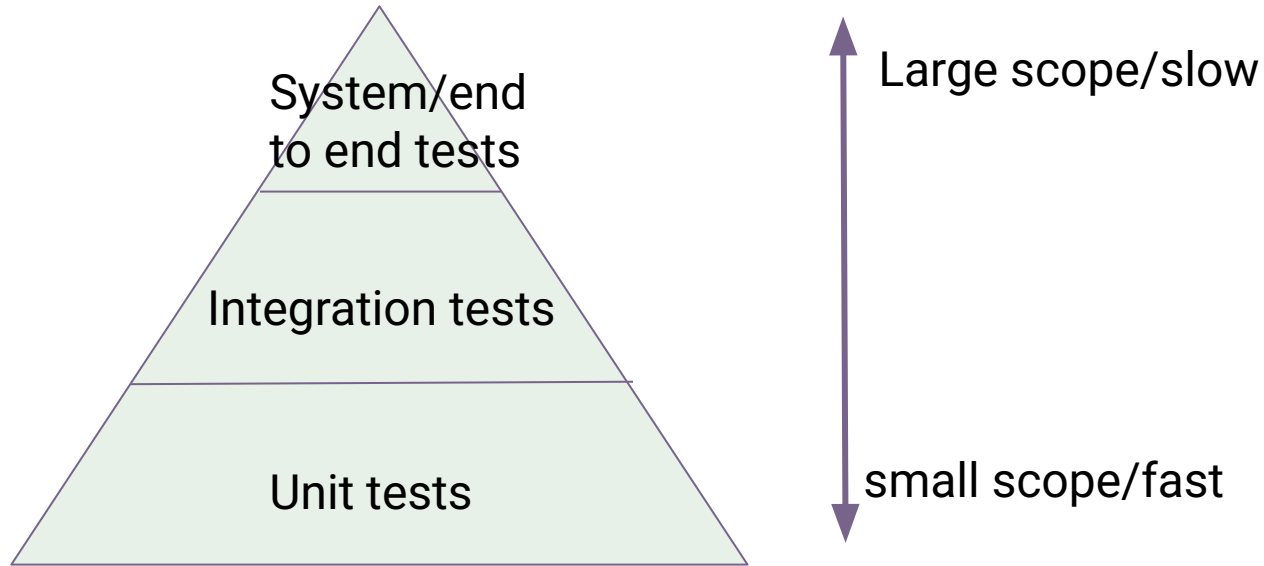
## Pros:

- Run within the program, so can catch problems caused by logic errors or edge cases.
- Makes it easier to find the cause of the bug by catching problems early.
- Catching problems early also helps prevent them escalating into catastrophic failures. It minimises the blast radius.

## Cons

- Tests can slow down the program.
- What is the right thing to do if an error is detected? How should this error be reported? Exceptions are a recommended route to go with this.
- Program needs to be run

# Hierarchy of tests



# Testing Frameworks

|        | Single case           | Mocking    | Property based testing   | Mutation testing |
|--------|-----------------------|------------|--------------------------|------------------|
| C++    | Catch2 & GoogleTest   | GoogleTest | rapidcheck               |                  |
| Rust   | Cargo tests (builtin) | Mockall    | Quickcheck & proptest    | Cargo-mutants    |
| Python | Pytest & Unittest     | Mock       | Hypothesis               | Mutatest         |
| Java   | JUnit                 | Mockito    | jqwik & junit-quickcheck | Pitest           |
| Julia  | Unit Testing          | Mocking    |                          |                  |

Tests can also be run via language agnostic frameworks, such as make and snakemake

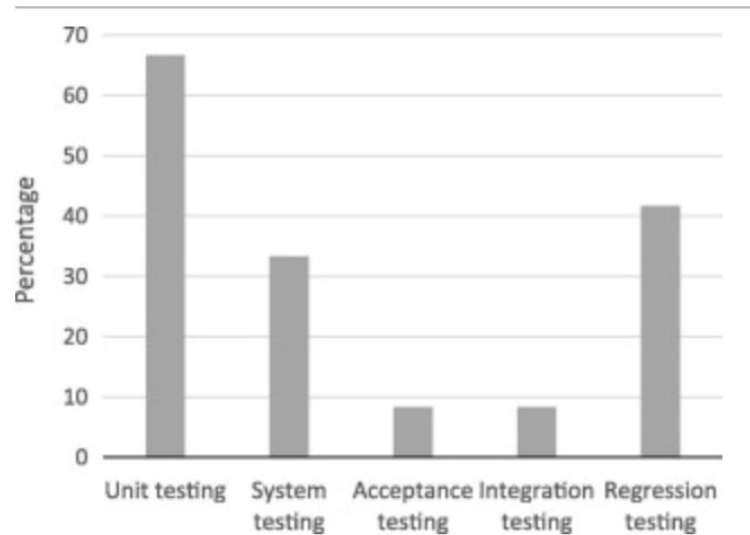
# Test driven development

- Starts with a TEST
- 
1. Assumption: what is the result/output that my code should deliver?
  2. Write the test.
  3. Write the code.
  4. Run the test.
  5. Proceed with 1.

Aims for FULL test coverage



# Software tests used in scientific research



Kanewala, U. & Bieman, J. M. Testing scientific software: A systematic literature review. *Inf. Softw. Technol.* **56**, 1219–1232 (2014).

# Setting up your test environment

- Tests live in a separate folder to your code base
- They are run independently from your code base

```
your_project/  
├── your_package/  
│   └── my_module.py  
├── tests/  
│   └── test_my_module.py  
├── requirements.txt  
└── pytest.ini  # Optional config
```

# Running your tests

| Context                        | Purpose/Use  | Tool/Example  |
|--------------------------------|--|---|
| <b>Assert statements</b>       | Quick internal checks within the code during development | <code>assert x &gt; 0, "x must be positive"</code><br>(Python)                  |
| <b>Manual running</b>          | Run the script manually to see if asserts raise errors   | <code>python script.py</code>   |
| <b>Tests from command line</b> | Structured test framework with richer reporting          | <code>pytest test_module.py</code>  |
| <b>Automatic test run</b>      | Integrate tests into build or workflow pipelines         | Run tests via <code>make test</code> in a Makefile                              |
| <b>IDE-integrated testing</b>  | Run tests with UI support and environment control        | Configure test environment in <b>VS Code</b>                                    |
| <b>CI with GitHub Actions</b>  | Automatically run tests on every push or pull request    | <code>.github/workflows/test.yml</code> with <code>pytest</code> or other tools |

# Assert - examples

- Assert is part of the standard library (python, matlab)
- Most languages have the equivalent of a assert statement
- Assert statements can be part of a test function, but can also be integrated into the code

```
assert x > 0, "x must be positive"
```

```
assert condition, message
```

```
assert x > 0
```

- The message is optional and is thrown in case of error
- asserts pass SILENTLY and fail LOUDLY

# Inline testing - assert

-

| Method                               | Equivalent to                 |
|--------------------------------------|-------------------------------|
| <code>.assertEqual(a, b)</code>      | <code>a == b</code>           |
| <code>.assertTrue(x)</code>          | <code>bool(x) is True</code>  |
| <code>.assertFalse(x)</code>         | <code>bool(x) is False</code> |
| <code>.assertIs(a, b)</code>         | <code>a is b</code>           |
| <code>.assertIsNone(x)</code>        | <code>x is None</code>        |
| <code>.assertIn(a, b)</code>         | <code>a in b</code>           |
| <code>.assertIsInstance(a, b)</code> | <code>isinstance(a, b)</code> |

`.assertIs()`, `.assertIsNone()`, `.assertIn()`, and `.assertIsInstance()` all have opposite methods, named `.assertIsNot()`, and so forth.

# Setting up your test environment in VS code

1. **Install the Python extension** for VS Code (if you haven't already).
2. Open your project folder in VS Code.
3. Press Ctrl+Shift+P (windows) Cmd+Shift+P and search for Python: Configure Tests.
4. Select your test framework (e.g., unittest or pytest): in this case, select unittest.
5. It'll auto-discover your test files and functions.

Troubleshoot:

> Developer Reload

Run tests by cli

# Test fixtures and mocks

## Fixture

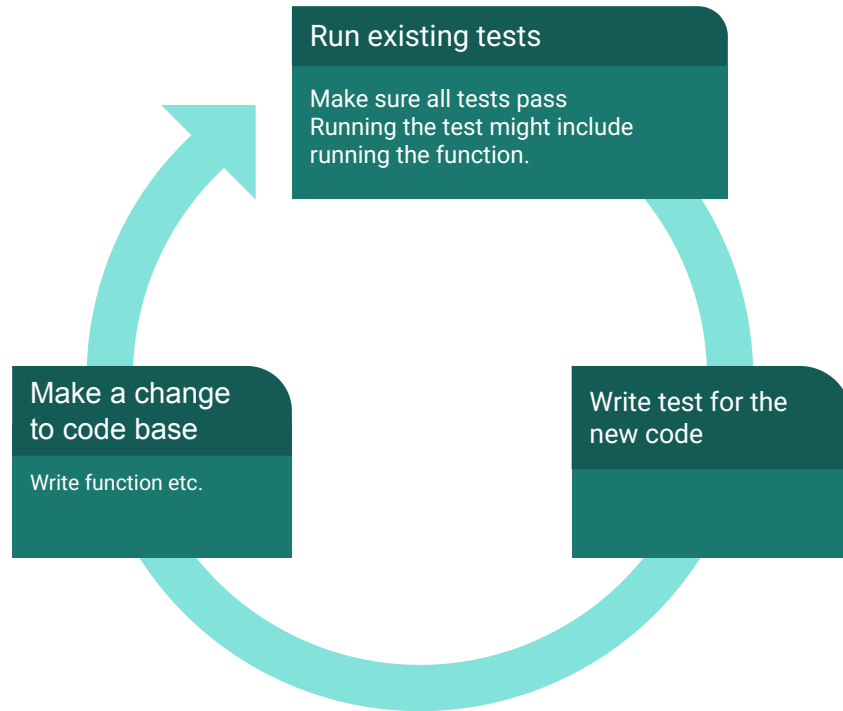
- Prepares the environment **before a test runs**
- Cleans up **after** the test
- Provides **consistent input/data/state**

## Mock

- Simulates behaviour of a dependency
- For example: API call

# Fixtures

- Your tests run your functions independently of what comes before or after
- Fixtures set the test up (provide parameters, data etc) and clean up after it





# Example of a Fixture (pytest)

Function to be tested:

```
import matplotlib.pyplot as plt

def plot_data(df):
    plt.plot(df["x"], df["y"])
    plt.xlabel("x")
    plt.ylabel("y")
    return plt
```

Test with fixture (providing df)

```
# test_my_module.py
import pytest
import pandas as pd
from my_module import plot_data

@pytest.fixture
def example_dataframe():
    # This fixture creates reusable test data
    return pd.DataFrame({
        "x": [1, 2, 3],
        "y": [2, 4, 6]
    })

def test_plot_data_runs(example_dataframe):
    plt = plot_data(example_dataframe)
    assert plt.gca().has_data()
```

# Pain points of software testing (in academia)

- Scientific software is often meant to establish new knowledge - known truth to test against
- Long execution times or pipelines
- Lack of knowledge of researchers about how to efficiently construct tests and prepare their code for tests
- Two code bases to maintain
- Incident model in academia: little focus on code, let alone additional code (tests)

# Setting up your test environment

- create a new environment with conda or venv
- activate
- install (pip), pytest coverage, pytest-cov
- set PYTHONPATH=.

**15 min break**

# Activity 1 (40')

Creating tests

Testing exercise

**15 min break**

**CI/CD**

The background features a light green rectangle on the left and a pink rectangle on the right, both extending from the bottom of the slide.

**What do we need CI/CD for?**



# CI/CD

CI: continuous integration

- Runs tests, checks whether the code works etc

CD: continuous deployment

- Once all checks have passed, the code gets deployed, website gets built etc.

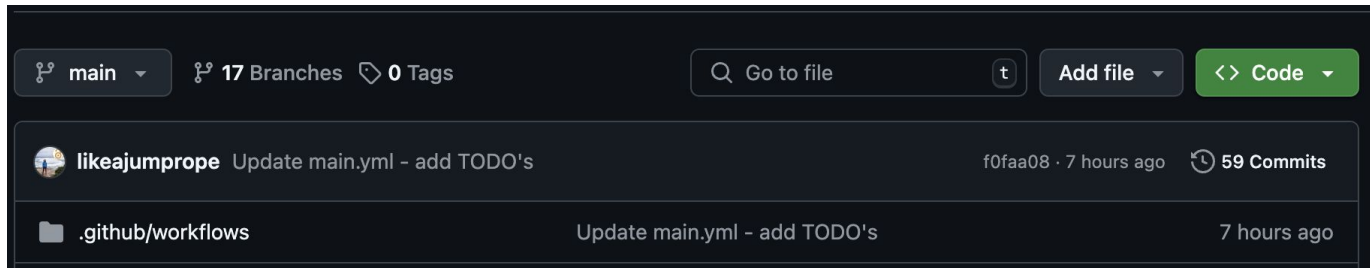
CI/CD: pipeline for robust output

# Github actions

- **GitHub Actions** are **automated workflows** that run when something happens in your GitHub repo — like pushing code, opening a pull request, or publishing a release.
- Often used as part of CI/CD pipeline
- Field: DevOps
- Examples:
  - Run tests every time code is pushed to main
  - Run quarto on push and render markdown to html on push
  - Open an issue every month
  - Close an issue with merging of a PR

# Github actions

Orchestrated via yaml.file in a .github/workflow folder



Structure: key - value pairs

Can be run

- On webhook (push, issue, merge etc)
- Scheduled
- Manually

```
parent:
  key1: value1
  key2: value2
child:
  key1: value1
```

# Github actions

The workflow is scheduled, but you can run it manually via github actions.

The screenshot shows the GitHub Actions interface for a repository. The top navigation bar includes links for Code, Issues (20), Pull requests, Discussions, Actions (highlighted with a red box), Projects (1), Wiki, Security, Insights, and Settings. On the left sidebar, the 'Actions' section is active, showing 'All workflows' and a list of workflows. The 'Create Monthly Newsletter Issue' workflow is highlighted with a red box. Below this, there are links for Management, Caches, Attestations, Runners, Usage metrics, and Performance metrics. The main content area displays the 'Create Monthly Newsletter Issue' workflow details. It shows '5 workflow runs' and a search bar for 'Filter workflow runs'. A message states: 'This workflow has a workflow\_dispatch event trigger.' To the right of this message is a 'Run workflow' button (highlighted with a red box). Below the message, there are two workflow runs listed. The first run is successful (green checkmark) and is titled 'Create Monthly Newsletter Issue' with the description 'Create Monthly Newsletter Issue #5: Manually run by likeajumprope'. The second run is failed (red X) and is also titled 'Create Monthly Newsletter Issue' with the description 'Create Monthly Newsletter Issue #4: Manually run by likeajumprope'. A dropdown menu is open, showing 'Use workflow from' with 'Branch: main' selected and a 'Run workflow' button (highlighted with a red box). The bottom right corner shows a timer icon and '9s'.

# Activity 2:

## Creating a Github action workflow

Do Activity 2:

Introduction to Github actions.

**WRAP UP**