What to do
and what not to do

A plea from the sanity of your TAs

How to project - 1

As a general guide - we are looking for something “cool and useful”

- It should not be trivial
- Complexity for its own sake WILL NOT win you marks

Set out a clear goal before starting to code. Time spent researching helpful packages and data sources at the beginning of the project prevents wasted time coding!

How to project 2 - Chemical Boogaloo

Before starting a project you should satisfy these conditions

- Do the data source and packages required actually exist?
  - If not, how quickly can you implement the required features?
- What happens if you can’t complete the whole project?
  - You should aim to have a Minimum Viable Product to which you can add functionality for a better grade
- Is the proposed project actually useful or new… If you can easily find the functionality *via* Google search, don’t make that your project.

How to structure packages

```
my_package/
│
├── .github/                  # GitHub-specific configurations
│   └── workflows/            # CI/CD configuration
│       ├── tests.yml
│       └── publish.yml
│
├── src/                      # Source code directory
│   └── my_package/           # Main package directory
│       ├── __init__.py       # Package initialization
│       ├── core.py           # Core functionality
│       └── utils.py          # Utility functions
│
├── tests/                    # Test directory
│   ├── __init__.py
│   ├── test_core.py
│   └── test_utils.py
```

You should broadly follow this structure

All non-test code should be under **src/my_package/**

Don’t name your package my_package or any other generic placeholder name

How to structure packages

```
my_package/
│
├── .github/                  # GitHub-specific configurations
│   └── workflows/            # CI/CD configuration
│       ├── tests.yml
│       └── publish.yml
│
├── src/                      # Source code directory
│   └── my_package/           # Main package directory
│       ├── __init__.py       # Package initialization
│       ├── core.py           # Core functionality
│       └── utils.py          # Utility functions
│
├── tests/                    # Test directory
│   ├── __init__.py
│   ├── test_core.py
│   └── test_utils.py
```

__init__.py

```python
# src/my_package/__init__.py

# Import and expose specific functions from internal modules
from .core import calculate_distance, Point
from .utils import format_output

# Define package-level constants
VERSION = "1.0.0"

# You can even define functions directly in __init__.py
def get_version():
    return VERSION
```

How structuring code works

__init__.py

```python
# src/my_package/__init__.py

# Import and expose specific functions from internal modules
from .core import calculate_distance, Point
from .utils import format_output

# Define package-level constants
VERSION = "1.0.0"

# You can even define functions directly in __init__.py
def get_version():
    return VERSION
```

How to use this code from outside the package

```python
import my_package

# Access functions exposed in __init__.py
my_package.calculate_distance(point1, point2
my_package.format_output(result)
my_package.get_version()

# Import specific components
from my_package import Point, calculate_dist
```

Accessing code in sub folders

```
src/
└── my_package/
    ├── __init__.py
    ├── core.py
    ├── utils.py
    └── advanced/
        ├── __init__.py
        ├── algorithms.py
        └── visualization.py
```

__init__.py

```python
# src/my_package/__init__.py

# Expose core functionality
from .core import calculate_distance, Point

# Expose selected functions from subfolder
from .advanced.algorithms import find_optimal_path
from .advanced.visualization import plot_results

# Now users can do:
# from my_package import find_optimal_path, plot_results
```

Accessing code in sub folders

__init__.py

```
# External script

# Import from subfolder
from my_package.advanced import algorithms
algorithms.find_optimal_path(graph)

# Import specific function from subfolder module
from my_package.advanced.visualization import plot_results
plot_results(data)
```

Accessing the code

```
# src/my_package/__init__.py

# Expose core functionality
from .core import calculate_distance, Point

# Expose selected functions from subfolder
from .advanced.algorithms import find_optimal_path
from .advanced.visualization import plot_results

# Now users can do:
# from my_package import find_optimal_path, plot_results
```

Where to put data

Under the root directory of the package, you should put your data in a data/ directory like so.

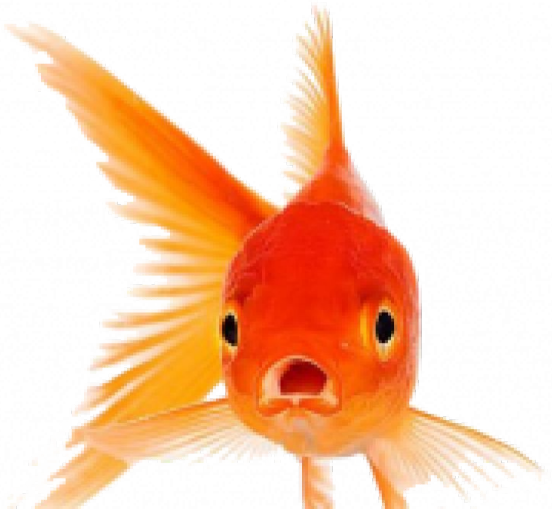
```
my_package/
├── pyproject.toml      # Modern package configuration
├── README.md           # Project documentation
├── src/                # Source code directory
│   └── my_package/     # Actual package code
├── data/               # Data directory
│   ├── raw/            # Raw data files
│   └── processed/      # Processed data files
```

Here is the example structure for non-code files.

README: This documents how to install the package, what data users might need to download etc

data: Stores all smallish data files needed for the code to run.
If you need large files - speak to us about how to implement it!

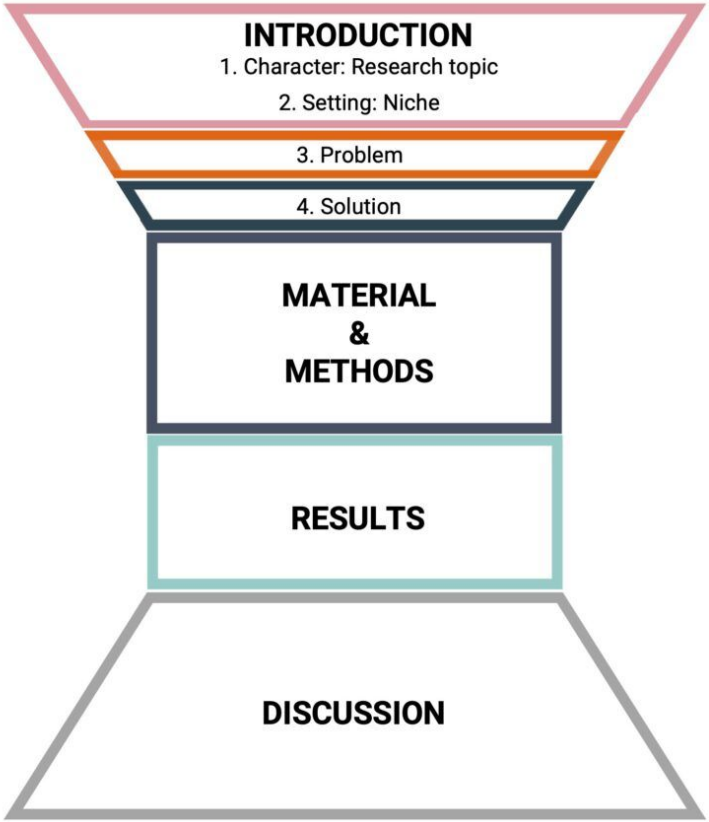
How to present



- A presentation tells a **story**
  - Introduce
  - Motivate
  - Approach/methods
    - Illustrate with flowcharts and diagrams
  - Results
  - Discussion
- Assume your audience is a **goldfish**
- Don’t get **stuck** in the details
  - **NOT** a description of every function
- Use visuals to **support** your statements
  - If you include screenshots of only code, you will **fail** the presentation
- Speak clearly, make eye-contact with the audience
  - If you read entirely from notes, you will **fail** the presentation

How to report

- Written in a jupyter notebook (**.ipynb**)
  - Look up markdown for formatting the text
- Structure it like a **scientific report**
  - Explain the relevant chemistry (Intro)
  - Complete sentences
  - Use titles, subtitles etc.
  - Include references if needed
- **Communicate** what you did and what your package solves
  - It is not a diary (“then we did this, then this, …”)
  - Implementation details only if necessary
- **Import** functions **only**
  - Don’t show the function but show what it achieves



Your TAs

Rebecca

- Drugs
- Proteins
- 3D representations

Sarina

- Transition metal complexes
- Catalysis
- QM calculations

Daniel

- Reaction prediction
- Retrosynthesis
- Large Language Models