Writing clear code

Good practices

Style

- Informative variable and function names
- Minimize hard-coded variables
- Reduce code repetition
- Be consistent – one way to do this is to follow a particular *style guide* (there can be many style guides for the same language)

Documentation

- What the purpose of the code is (what problem does it solve)
- What the code does
- Why the code is written this way
- Cite sources

Organize code

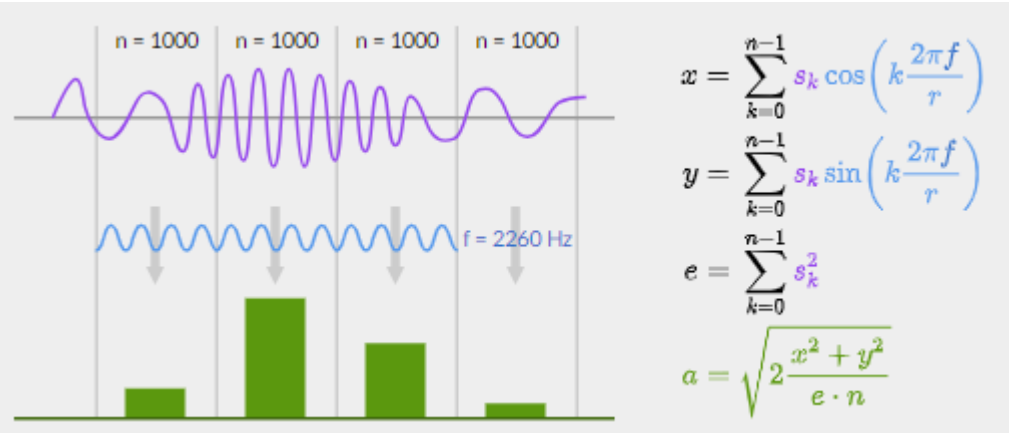
- If long, split into multiple files
- Arrange according to logical units

Style guides

- Google Python Style Guide
- Numpy example
- C
- MATLAB

Within the permissiveness of the language, need to stick to self-consistent conventions for readability

Informative variables and function names



Which is easier to read? Depends on who you ask.

- Scientists/mathematicians – closer to mathematical symbols and expressions
- Software developers – (English) words

Both are accepted.

```
def fourier(s, r, f):
    n = len(s)
    x = 0
    y = 0
    e = 0
    for k in range(n):
        x += s[k] * cos(k * 2 * pi * f / r)
        y += s[k] * sin(k * 2 * pi * f / r)
        e += s[k]**2
    return sqrt(2 * (x ** 2 + y ** 2) / (e * n))
```

```
def fourier(signal, framerate, freq):
    x = 0.0
    y = 0.0
    e = 0.0
    for i, s in enumerate(signal):
        x += s * math.cos(i * 2 * math.pi * freq / framerate)
        y += s * math.sin(i * 2 * math.pi * freq / framerate)
        e += s * s
    return math.sqrt(2 * (x * x + y * y) / (e * len(signal)))
```

Minimize hard-coded variables

cryptogramme exercise

```
char cryptogramme[] = "T-LS-AOS--EEOIOAAUENTUOLUILRPDSA-S.LUNV-ENEGT-T
// Initialiser le message à ******...
char message[57];
for (int i = 0; i < 56; i++) {
    message[i] = '*';
}

// Dechiffrer
int position = 0;
for (int i = 0; i < 56; i++) {
    // Avancer de 17 cases vides
    int avancer = 17;
    while (avancer > 0) {
        position += 1;
        if (position == 56) position = 0;
        if (message[position] == '*') avancer -= 1;
    }

    // Mettre le caractère dans cette case
    char c = cryptogramme[i];
    message[position] = c;
}

// Terminer la chaine de caractères, et l'afficher
message[56] = 0;
printf("Message: %s\n", message);
```

```
#include <stdio.h>
#define NCHAR 56
#define NADV 17

int main() {

  char *cryptogramme = "T-LS-AOS--EEOIOAAUENTUOLUILRPDSA-S.LUNV-ENEGT-T-E-NLLSBE";

  // Initialiser le message à ******...
  char message[NCHAR+1];
  for (int i = 0; i < NCHAR; i++) {
    message[i] = '*';
  }

  // Dechiffrer
  int pos = 0;
  for (int i = 0; i < NCHAR; i++) {
    int adv = 0;
    while(adv < NADV) {
      pos = (pos + 1) % NCHAR; // % is the modulo operator
      if(message[pos] == '*') adv++;
    }

    // Mettre le caractère dans cette case
    message[pos] = cryptogramme[i];
  }

  // Terminer la chaine de caractères, et l'afficher
  message[NCHAR] = 0;
  printf("Message: %s\n", message);

  return 0;

}
```

Reduce code repetition (1)

Use functions to carry out common set of instructions

- When several lines of code are repeated with few modifications, wrap it in a function

Appropriate control structures

```
if conditionTrue
    x = 0;
    y = 0;
    z = 0;
else
    x = 0;
    y = 0;
    z = 1;
end
```

```
x = 0;
y = 0;
if conditionTrue
    z = 0;
else
    z = 1;
end
```

which is better?

Reduce code repetition (2)

Use lists / cell arrays to store similar information and iterate over them, so that the same code to maximize code reuse.

éolienne exercise

```
speed_pully = readmatrix('vent-pully.csv')
speed_chr = readmatrix('vent-chur.csv')
speed_jungfraujoch = readmatrix('vent-jungfraujoch.csv')

power_pully = power_vectorized(speed_pully)
power_chur = power_vectorized(speed_chur)
power_jungfraujoch = power_vectorized(speed_jungfraujoch)

{...more operations...}
```

```
sites = {'pully', 'chur', 'jungfraujoch'};
for ix = 1:numel(sites)
    speed = readmatrix(sprintf('vent-%s.csv', sites{ix}))
    power = power_vectorized(speed)
    {...more operations...}
end
```

Documenting code

Why document code?

- Describe how to use the code
- Describe what the code does
- Describe why decisions were taken
- Who to contact

code is for the machine
documentation is for the human

Who is your audience?

- Team members
- Project manager
- Broader community
- **Yourself 6 months from now**

generating documentation is probably one of the best uses of generative AI (co-pilot) tools in coding

How to document code

README file to provide project overview

In script/code files

- Function documentation (“docstrings” in Python)
- Comment block at top of script describing what it does
- Inline comments to describe why decisions were taken
- Simple, “self-explanatory” code parts do not need redundant documentation

Example files using literate programming tools

- Jupyter notebooks / Quarto documents (Python, R, Julia, GNU Octave, MATLAB …C?)
- Live Editor (MATLAB)

Documented scripts vs. notebooks - which to use?

- **Code development**: script/code files with comments, docstrings
- **Demonstration of code use (show input/output) or juxtapose with LaTeX equations**: literate programming tools

For ENG-270 projects, extra example files beyond the report is not necessary.

However, FYI it is possible to mix text and code using literate programming tools - generate tables and figures to be displayed in the report while hiding the code.

https://realpython.com/documenting-python-code

README (or README.md)

README file

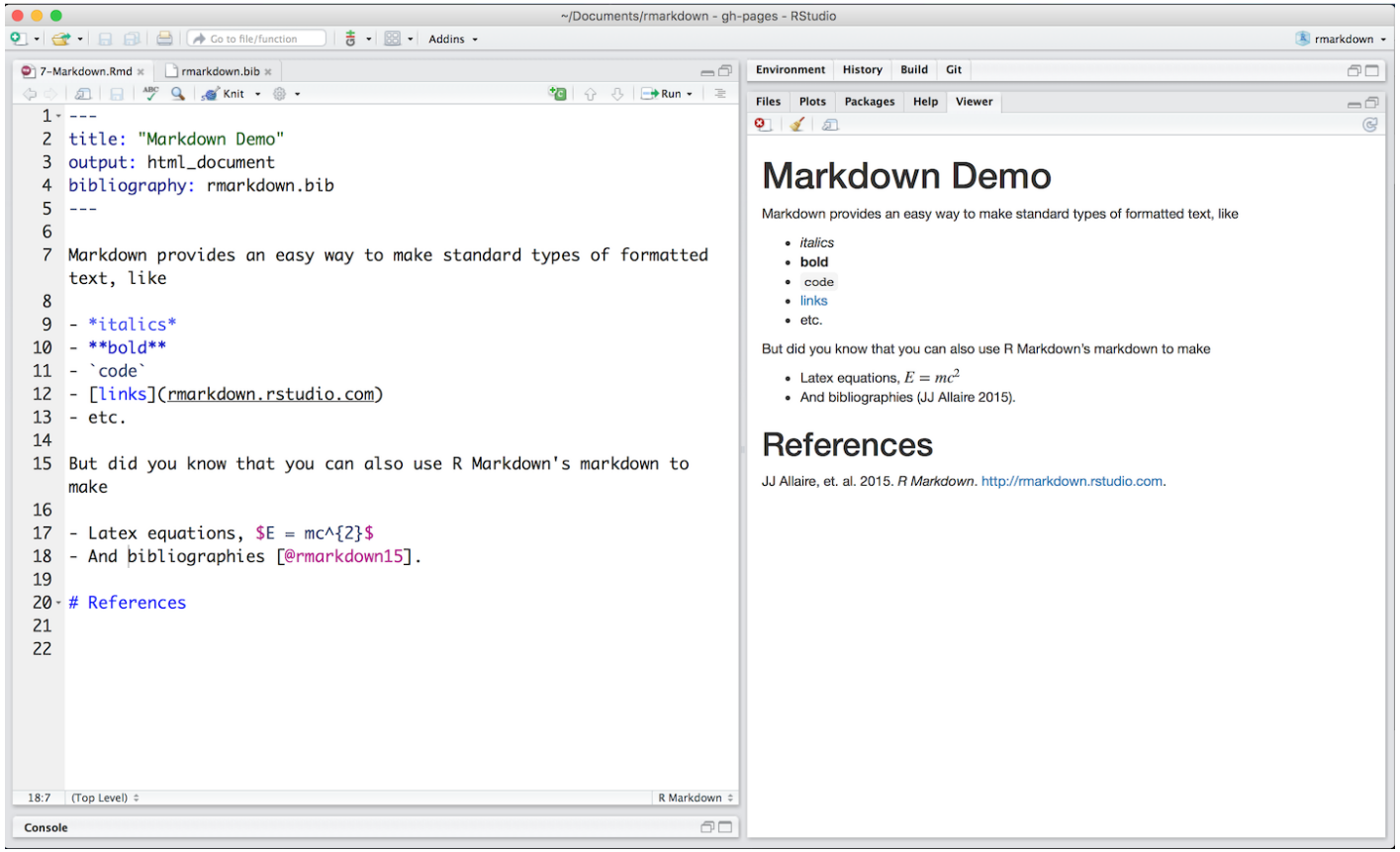
- What is the project about
- Project description - what your program does
- How to install prerequisite tools, build, and run the program
- Credits

- README (with no extension) is a plain text file (ASCII or Unicode)
- README.md is a Markdown file.

Markdown

- Lightweight *markup* language
- ".md" file is just another text file
- Used in README.md and Jupyter notebooks/ Quarto documents
- Add *formatting elements* to plain text documents
- *Rendering* generates formatted text
- Many different variants or *flavors*
  - Pandoc
  - GitHub
  - CommonMark
  - Markdown Extra
  - …

Example of Pandoc Markdown



https://rmarkdown.rstudio.com/lesson-8.html
https://www.markdownguide.org/getting-started/

Markdown flavors

- README.md should be written in GitHub Markdown
- Jupyter notebooks and Quarto documents should be written in Pandoc Mardown
- Pandoc Markdown offers greater capabilities – e.g., including automated generation of bibliographies from citations

For simple documents, the two differ mostly in the header

- GitHub Markdown:
  ```
  # Project Title
  ```
- Pandoc Markdown:
  ```
  ---
  title: Project Title
  ---
  ```

Equations in Markdown

```
**The Cauchy-Schwarz Inequality**
$$\left( \sum_{k=1}^n a_k b_k \right)^2 \leq \left( \sum_{k=1}^n a_k^2 \right) \left(
\sum_{k=1}^n b_k^2 \right)$$
```

or

```
**The Cauchy-Schwarz Inequality**

```math
\left( \sum_{k=1}^n a_k b_k \right)^2 \leq \left( \sum_{k=1}^n a_k^2 \right) \left(
\sum_{k=1}^n b_k^2 \right)
```
```

renders

The Cauchy-Schwarz Inequality

$$\left(\sum_{k=1}^{n} a_k b_k\right)^2 \leq \left(\sum_{k=1}^{n} a_k^2\right) \left(\sum_{k=1}^{n} b_k^2\right)$$

https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/writing-mathematical-expressions

Code documentation

Function documentation

Main elements:

- Description
- Input arguments
- Return value
- Example usage

```
function a = fourierfct(s, f, r)
% fourierfct calculates the Fourier transform of the signal
%
% Parameters
% s: signal
% f: frequency
% r: framerate
%
% Returns
% a: amplitude
n = length(s);
k = 0:n-1;
x = sum(s .* cos(k .* 2 .* pi .* f ./ r));
y = sum(s .* sin(k .* 2 .* pi .* f ./ r));
e = sum(s .^ 2);
a = sqrt(2 * (x ^ 2 + y ^ 2) / (e * n));
end
```

```
from math import pi, cos, sin, sqrt
def fourier(s, r, f):
    """
    fourier calculates the Fourier transform of the signal

    Parameters
    s: signal
    f: frequency
    r: framerate

    Returns
    amplitude of the desired frequency
    """
    n = len(s)
    x = 0
    y = 0
    e = 0
    for k in range(n):
        x += s[k] * cos(k * 2 * pi * f / r)
        y += s[k] * sin(k * 2 * pi * f / r)
        e += s[k]**2
    return sqrt(2 * (x ** 2 + y ** 2) / (e * n))
```

How to add help in MATLAB: https://ch.mathworks.com/help/matlab/matlab_prog/add-help-for-your-program.html
Python docstring and suggested contents: https://pandas.pydata.org/docs/development/contributing_docstring.html

Header block (comment block) at top of file – - *optional* -

In case the file gets distributed without the original context (i.e., renamed / separated from the rest of the code):

- File name
- What module this file belongs to
- What the file does in general terms (purpose of the code)
- Licensing terms
- Authors

Note this is the part that often fails to get updated when the code changes! Some programmers will use text processing techniques to automatically generate these headers.

Some examples of well-documented code

- NumPy (Python, C), example
- Flask (Python), example
- OpenBSD (C), example
- Redis (C), example
- Also git, linux kernel, … (Python)

Notebooks

Jupyter notebooks

- Extension is .ipynb (originally: Interactive Python Notebook)
- .ipynb are actually **JSON** files
- Originally interfaced through web browser
- Now interfaces are available through editors (VS Code, JupyterLab Desktop)

.ipynb notebooks consist of two types of cells:

1. **text** (markdown format). markdown is a *markup* language
2. **code** (select Python "kernel")

Work interactively (edit/run)

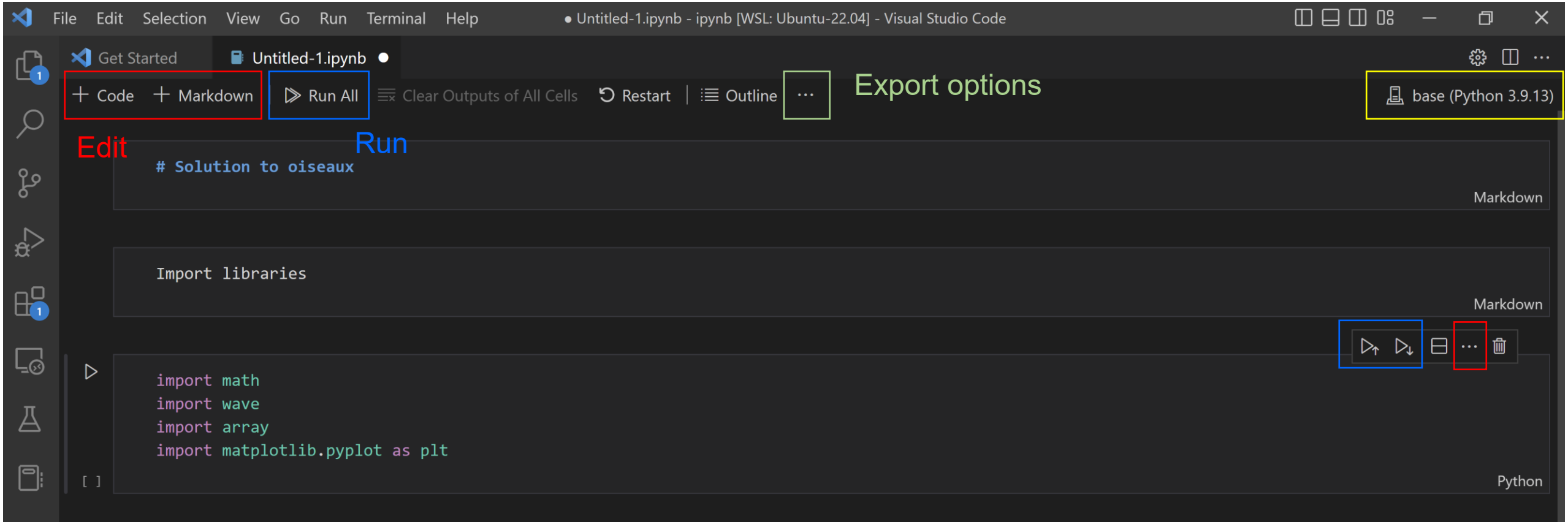
Export to multiple formats

- PDF
- HTML

https://code.visualstudio.com/docs/datascience/jupyter-notebooks

Jupyter notebooks in VS Code

Install Jupyter Extension; Start New File → Jupyter notebook



oiseaux_nb.ipynb

File Edit View


```
{
  "cells": [
    {
      "cell_type": "markdown",
      "id": "a991b31d",
      "metadata": {},
      "source": [
        "Import libraries.\n"
      ]
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "id": "ed63822a",
      "metadata": {},
      "outputs": [],
      "source": [
        "from math import sin, cos, pi, sqrt\n",
        "import wave\n",
        "import array\n",
        "import matplotlib.pyplot as plt\n",
        "from functools import reduce, partial"
      ]
    },
    {
      "cell_type": "markdown",
      "id": "2bc02ad6",
      "metadata": {},
      "source": [
        "Define Python functions for Fourier analysis.\n",
        "\n",
        "\n",
        "```{=tex}\n",
        "\\begin{align}\n",
        "x &= \\sum_0^{n-1} s_k \\cos \\left(k \\frac{2\\pi f}{r}\\right)
\\\\\\\n",
        "y &= \\sum_0^{n-1} s_k \\sin \\left(k \\frac{2\\pi f}{r}\\right)
```

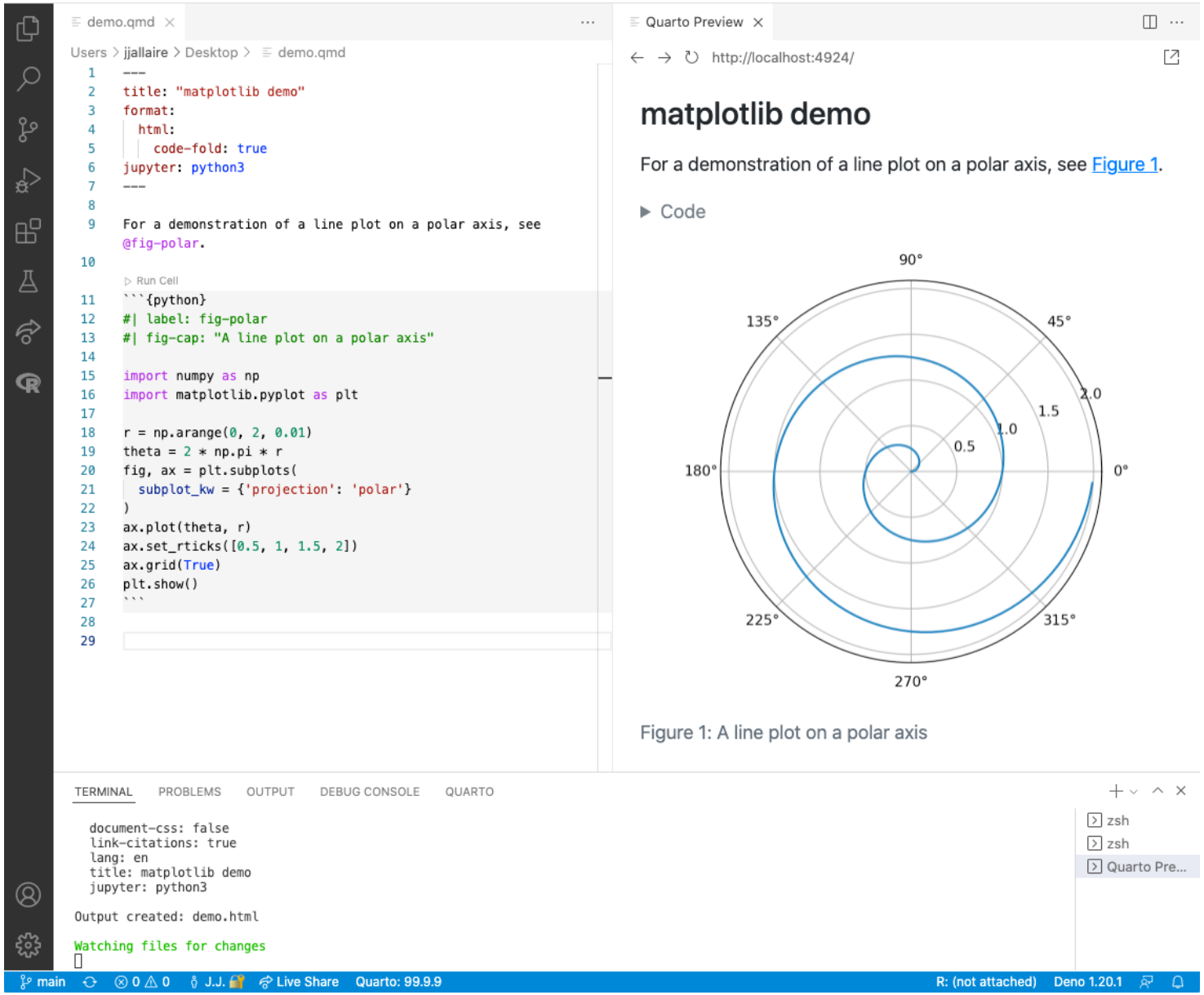

Ln 22, Col 7 | 100% | Unix (LF) | UTF-8

An .ipynb file is just a JSON file

Quarto

new kid in town

- Plain text file, but markdown rather than JSON
  → better version control
- Uses *Jupyter kernel* like Jupyter notebooks, also Mkernel for MATLAB available
- Export to slides, books, etc.



https://quarto.org/docs/tools/vscode.html
https://quarto.org/docs/visual-editor/vscode/

oiseaux_qu.qmd

File Edit View

````
---
title: "Oiseaux exercise"
echo: false
---

Import libraries.

```{python}
from math import sin, cos, pi, sqrt
import wave
import array
import matplotlib.pyplot as plt
from functools import reduce, partial
```

Define Python functions for Fourier analysis.

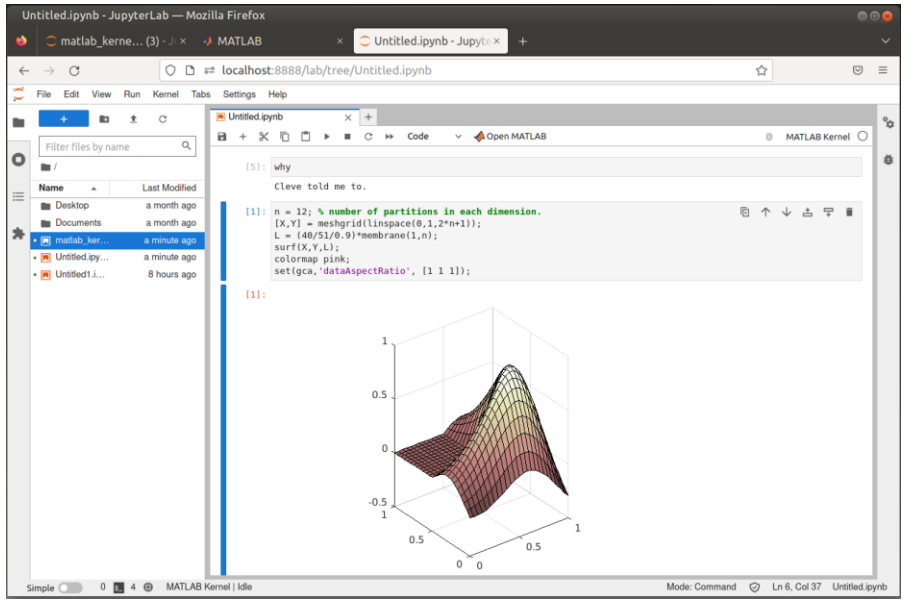
```{=tex}
\begin{align}
x &= \sum_0^{n-1} s_k \cos \left(k \frac{2\pi f}{r}\right) \\
y &= \sum_0^{n-1} s_k \sin \left(k \frac{2\pi f}{r}\right) \\
e &= \sum_0^{n-1} s_k^2\\
a &= \sqrt{2 \frac{x^2 + y^2}{e \cdot n}}
\end{align}
```
```{python}
def readWavFile(filename):
    wav = wave.open(filename)
    framerate = wav.getframerate()
    bytes = wav.readframes(wav.getnframes())
    signal = array.array('h', bytes).tolist()
    return signal, framerate

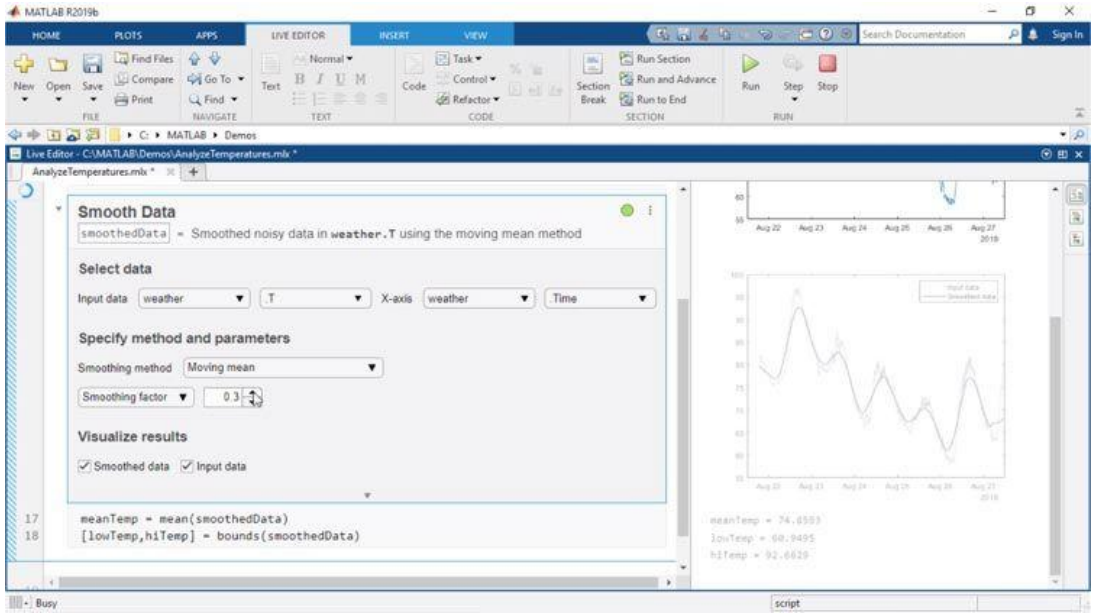
def fourier(signal, r, f):
    seq = range(len(signal))
    x = reduce(lambda acc, k: acc + signal[k] * cos(k * 2 * pi * f / r),
seq, 0)
````

Ln 1, Col 1 | 100% | Unix (LF) | UTF-8

An .qmd file is just a Markdown file
````

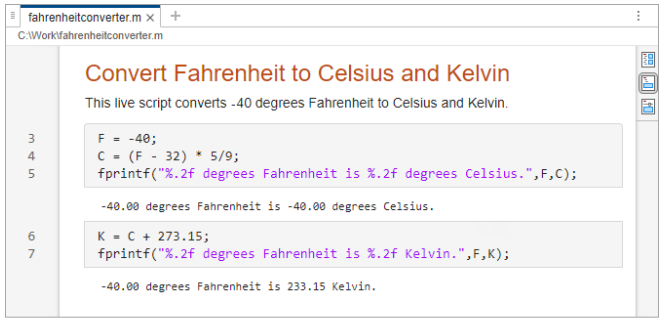
Jupyter or MATLAB Live Editor?

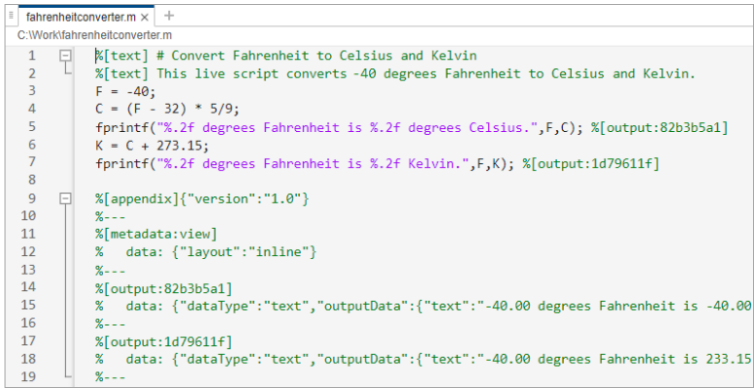




https://ch.mathworks.com/products/reference-architectures/jupyter.html
https://blogs.mathworks.com/matlab/2023/01/30/official-mathworks-matlab-kernel-for-jupyter-released/
https://ch.mathworks.com/products/matlab/live-editor.html

MATLAB Live Code formats





.mlx format: zipped XML (hierarchical) files

- not good for version control (with git)
- effectively useful only with MATLAB editor

.m format (since R2025a): plain text file

- Edit in MATLAB Live Editor
- good for version control
- can open and read (and edit) with any text editor

https://ch.mathworks.com/help/matlab/matlab_prog/live-script-file-format.html
https://ch.mathworks.com/help/matlab/matlab_prog/plain-text-file-format-for-live-scripts.html

Testing

Not major emphasis for solo projects; projects with short lifespan

Do not need for ENG-270 projects

Just FYI that this exists

https://realpython.com/python-testing/

Testing

Examples of tests in software development

- ***unit tests*** for functions to check that a certain output is produced for a given input. (Important for large codebases where someone else might add a feature or rewrite your function.)
- ***integration tests*** to check that components work with each other
- ***performance tests*** to measure speed and scalability

Why write tests?

- check that your code runs the same on another machine
- check that the rest of the code doesn't break when you or someone else rewrites your function to speed up or change the underlying algorithm

Example function

Original function

```
def euclidean_distance(x, y):
    return sum(map(lambda x, y: (x - y) ** 2, x, y)) ** .5
```

Test function

```
from math import cos, pi
def test_euclidean_distance():
    value = 1 / cos(45 / 180 * pi)
    tolerance = 1e-10
    assert abs(euclidean_distance([1, 0], [0, 1]) - value) < tolerance
```

When to write tests?

- **after** the function is finished is defined so that users/machines beside yourself can understand your intention (*conventional approach*)

- **before** writing the function to specify what it should do (*test-driven development*)

- (AI could be used to write tests for your functions, or to write the function that fulfills your test)

Final remarks

Main points

- Strive to write readable code
- Reduce repeated code
  - If changes are necessary, only need to change in fewest number of places
- Documentation – *for the project*, provide minimum set of instructions to reproduce your results and understand what is there
  - README to give overview
    - How to install tools
    - How to build
    - How to run
  - Function inputs/outputs for the main functions
  - Inline comments if some surprising decisions are taken (otherwise don't clutter)
- Support provenance from raw data to results