Writing clear code

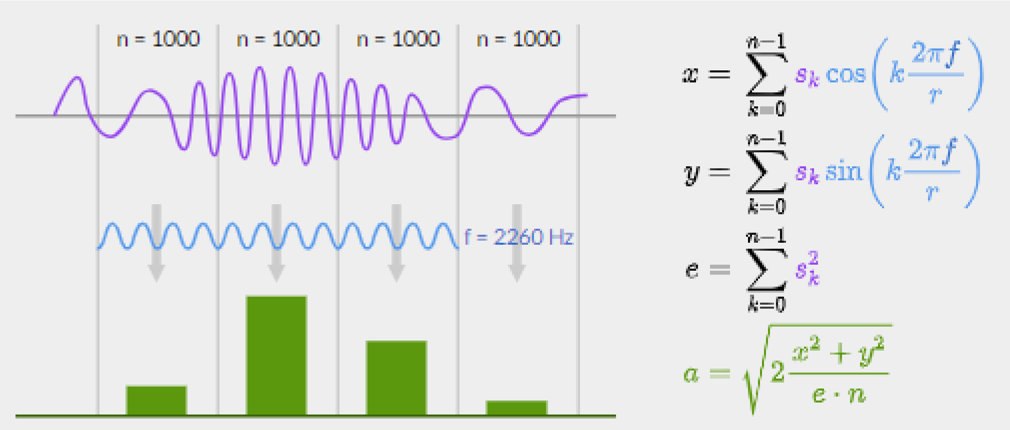
Good practices

- Informative variable and function names
- Minimize hard-coded variables
- Reduce code repetition
- Document code
- Cite sources
- Follow a particular *style guide*. There can be many style guides for the same language.

Style guides

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

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

```
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

Suisse exercise

```
xMin = np.min(x)
yMin = np.min(y)
xIndex = (x - xMin) / 200
yIndex = (y - yMin) / 200
xIndexInt = xIndex.astype('int32')
yIndexInt = yIndex.astype('int32')
xCount = xIndexInt.max() + 1
yCount = yIndexInt.max() + 1
xp = xMin + np.arange(0, xCount) * 200
yp = yMin + np.arange(0, yCount) * 200
```

```
dx = 200
dy = 200
xmin = x.min()
ymin = y.min()
xidx = ((x - xmin) / dx).astype('int32')
yidx = ((y - ymin) / dy).astype('int32')
xdim = xidx.max() + 1
ydim = yidx.max() + 1
xp = xmin + np.arange(0, xdim) * dx
yp = ymin + np.arange(0, ydim) * dy
```

Cryptogramme exercise

```c
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);
```

```c
#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

- Use functions to carry out common set of instructions

- 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
```

if you are doing a lot of copy-paste of code, consider that there may be room for improvement

Documenting code

Why document code?

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

Who is your audience?

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

code is for the machine
documentation is for the human

How to document code

README file to provide project overview

In script/code files

- function “docstrings” (Python)
- comments (inline, comment blocks, header)
- simple, “self-explanatory” code parts do not need redundant documentation

Literate programming tools

- Jupyter notebooks / Quarto documents (Python, R, Julia, GNU Octave, …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

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

README file

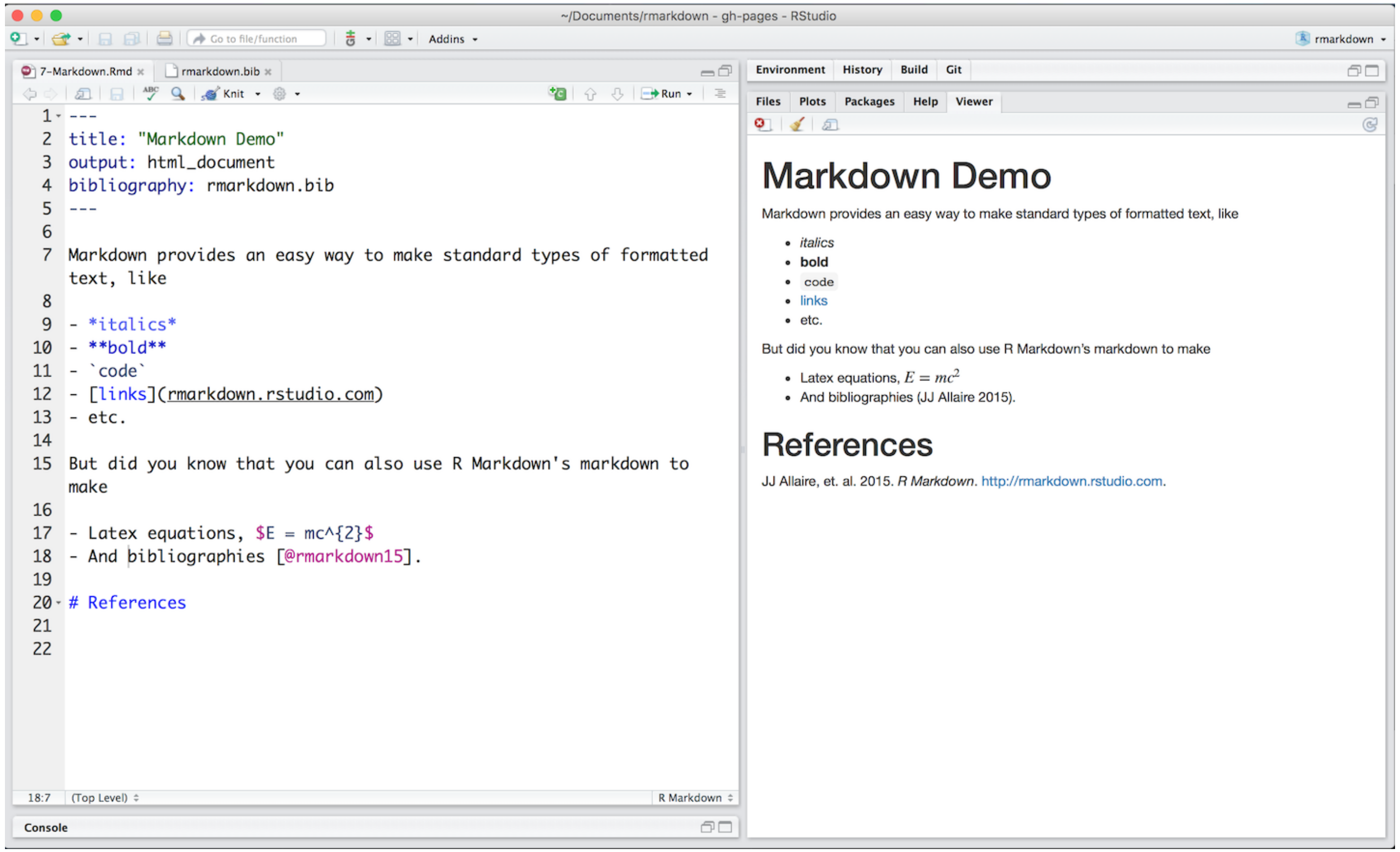
- What is the project about
- Project description
  - what your program does
  - choice of tools used
  - existing features and those you wish to add (or could have added)
- How to install and run the program
- Use cases for the program
- Credits

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

Markdown

- Lightweight *markup* language
- 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

Docstring examples

```python
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))
```

```matlab
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
```

Some examples of well-documented code

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

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)

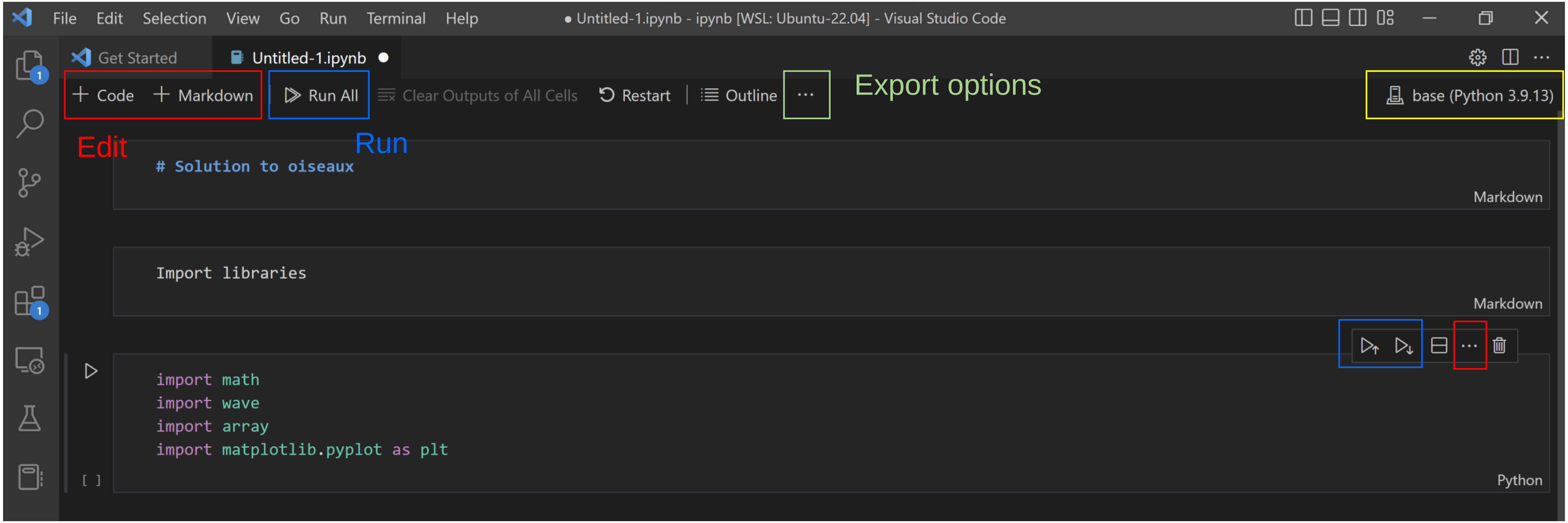
Export to multiple formats

- 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

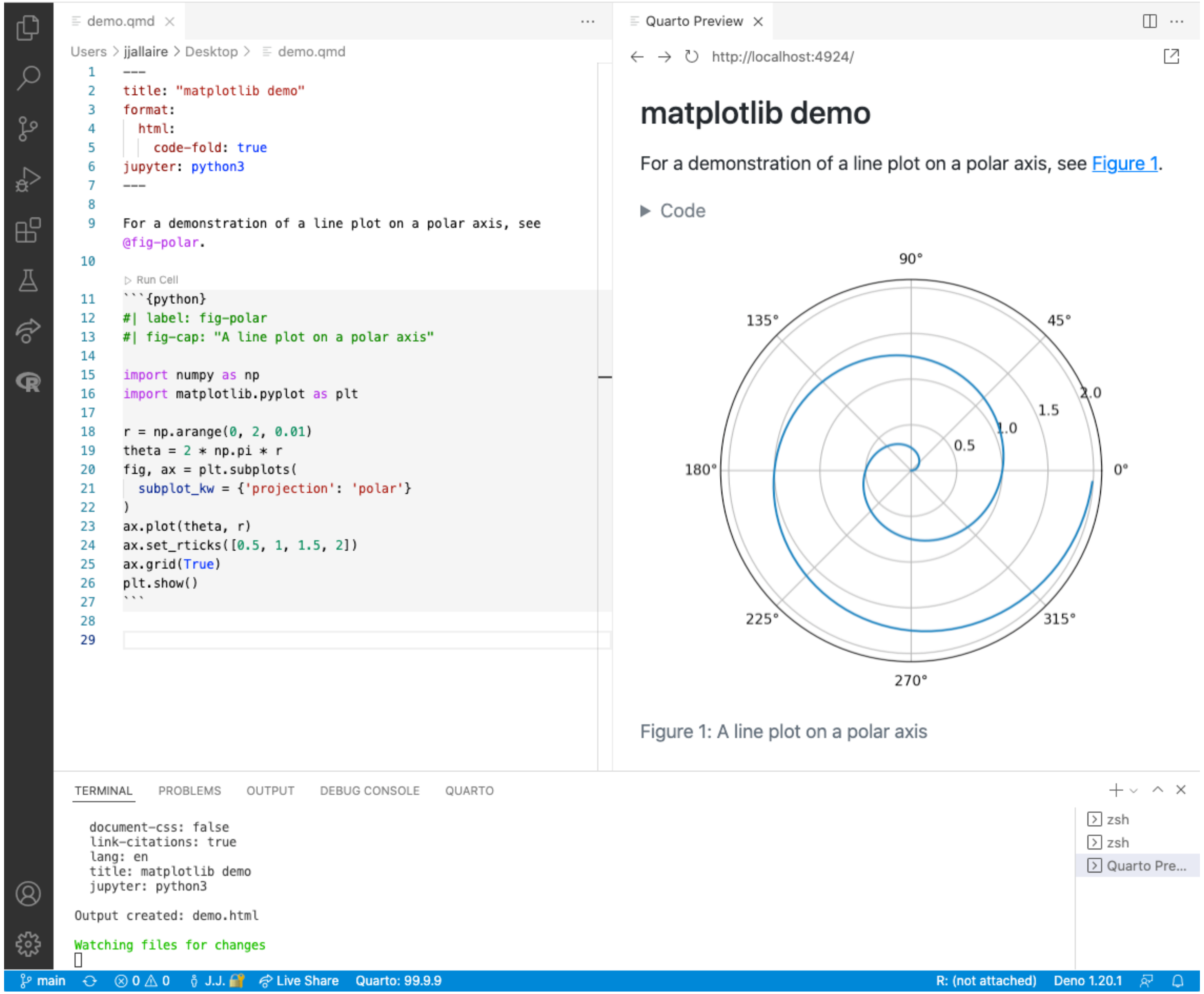

```
{
  "cells": [
    {
      "cell_type": "markdown",
      "id": "a991b31d",
      "metadata": {},
      "source": [
        "Import libraries.\n"
      ]
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "id": "ed63822a",
      "metadata": {},
      "outputs": [],
      "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
- 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
````

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

Testing

In software development,

- *unit tests* are written 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* are written to check that components work with each other

Contrived example for testing builtin function *sum()*

Python

```python
import unittest


class TestSum(unittest.TestCase):

    def test_sum(self):
        self.assertEqual(sum([1, 2, 3]), 6, "Should be 6")

    def test_sum_tuple(self):
        self.assertEqual(sum((1, 2, 2)), 6, "Should be 6")

if __name__ == '__main__':
    unittest.main()
```

Shell

```
$ python test_sum_unittest.py
.F
======================================================================
FAIL: test_sum_tuple (__main__.TestSum)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_sum_unittest.py", line 9, in test_sum_tuple
    self.assertEqual(sum((1, 2, 2)), 6, "Should be 6")
AssertionError: Should be 6

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

“Test-driven development”

- Some software engineers are adamant about developing programs around functionality that can be tested.
- May be suitable for large projects with multiple team members, or for code bases that are expected to have a long lifetime.
- Writing tests is time consuming (considering multitudes of “edge cases”), and some functions are hard to test. For simple programs, this is a typically a low priority.
- Results should be tested against intuition nonetheless (though this is no guarantee that the program is functioning correctly).
- **Testing will not be emphasized in this class**, but you should be aware that formal testing frameworks exist.

Other considerations – structuring your files

- Project directory structure
  - data
  - code (libraries and main code)
  - documentation
  - …

- Raw data should remain untouched, but imported and transformed into outputs (results) by your code
  - data at intermediate levels of processing can be saved if necessary

- *See other lectures on automation*