Summary

- MANY ways to solve the same problem

- Some are preferred over others
    - Readability*
        - parsimonious – no more variables and functions defined than necessary
        - short - also less place to introduce bugs
        - closer to mathematical formulae
        - declarative (describes *what* is being computed rather than *how* it’s being computed)
    - Efficiency
        - vectorization
        - avoid defining same operation in multiple places

- *There are two different ways in which code can be readable, and they can be orthogonal:
    - easy to read and understand what the whole program or sections of the program is doing at a glance
    - easy to read and understand what each step is doing

Wishful programming

- To solve the problem, think about how you want to transform the data and what operations you need to perform to transform them

```
output = operation(input_data1, input_data2, ..., parameter1, parameter2, ...)
```

- Write out the operations without filling in the details (pseudo-code)

```
signal, framerate = readWavFile(inputwav)
time_freq, amplitude = apply_fourier(signal, framerate, frequency)
time_power, amplitude_ave = average(time_freq, amplitude)
events = detect_events(time_power, amplitude_ave, threshold)
```

- Often programming can be done without the computer

- (90% of coding is spent debugging)

Use functions

Nested loops (often bad)

```python
N = len(signal)
framerate = 44100 # s^-1
nframes = 1000
freq = 2260        # s^-1 (Hertz = per second)
time = []          # s
power = []
for i in range(0, N, nframes):
    x = 0
    y = 0
    e = 0
    imax = min(i+nframes, N)
    for k in range(i, imax):
        x += signal[k] * cos(k * 2 * pi * freq / framerate)
        y += signal[k] * sin(k * 2 * pi * freq / framerate)
        e += signal[k]**2
    a = sqrt(2 * (x**2 + y**2) / (e * nframes))
    time.append(i / framerate)
    power.append(a)
```

Encapsulate operation in a function

```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))
```

```python
framerate = 44100 # s^-1
nframes = 1000
freq = 2260        # s^-1 (Hertz = per second)
time = []          # s
power = []
for i in range(0, len(signal), nframes):
    time.append(i / framerate)
    power.append(fourier(signal[i : (i+nframes)], framerate, freq))
```

Oiseaux exercise

Vectorization

Python/NumPy

MATLAB

Not vectorized

```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 [fourier] = fourierfct(signal, framerate, freq)
    %% Vérification des paramètres
    if nargin ~= 3
        error('This fonction expects 3 arguments : the signal, the frame rate, and the frequency');
    end

    %% Fonction
    len = length(signal);
    x = 0;
    y = 0;
    e = 0;
    fourier = 0;
    signal = double(signal);

    for i = 1:length(signal)
        x = x + signal(i) * cos(i * 2 * pi * freq / framerate);
        y = y + signal(i) * sin(i * 2 * pi * freq / framerate);
        e = e + signal(i) * signal(i);
        fourier = sqrt(2 * (x * x + y * y) / (e * len));
    end
end
```

Vectorized

```python
import numpy as np
from math import cos, sin, sqrt
def fourierfct(s, f, r):
    """
    fourierfct calculates the Fourier transform of the signal

    Parameters
    s: signal
    f: frequency
    r: framerate

    Returns
    a: amplitude
    """
    n = s.size;
    k = np.arange(n);
    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));
    return a
```

```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
```

Vectorization

Vectorized method 1

```
vitesseMin = 2.5;
vitesseMax = 6.5;

segm1 = vitesse > vitesseMin & vitesse <= vitesseMax;
segm2 = vitesse > vitesseMax;

puissances = zeros(size(vitesse));
puissances(segm1) = (vitesse(segm1) - vitesseMin) * 1.5;
puissances(segm2) = 6;
```

Not vectorized; not *initialized*

```
vitesseMin = 2.5;
vitesseMax = 6.5;

for i = 1:length(vitesse)
    if vitesse(i) < vitesseMin
        puissances(i) = 0;
    elseif vitesse(i) > vitesseMax
        puissances(i) = 6;
    else
        puissances(i) = (vitesse(i) - vitesseMin) * 1.5;
    end
end
```

Vectorized method 2

```
puissances = (vitesse - vitesseMin) * 1.5;
puissances(vitesse <= vitesseMin) = 0;
puissances(vitesse > vitesseMax) = 6;
```

Vectorized method 3

```
segm0 = vitesse <= vitesseMin;
segm2 = vitesse > vitesseMax;
segm1 = ~(segm0 | segm2); % or, segm0 + segm2 < 1
puissances = segm0 .* 0 + segm1 .* (vitesse - vitesseMin) .* 1.5 + segm2 .* 6;
```

Suisse exercise

	Python/NumPy	**MATLAB**

Read

```python
dhm200 = np.genfromtxt('DHM200.xyz', delimiter = ' ', dtype = None)
x = dhm200[:, 0]
y = dhm200[:, 1]
h = dhm200[:, 2]
```

```matlab
fid = fopen('data/DHM200.xyz', 'r');
dhm = fscanf(fid, '%f %f %f', [3 Inf])';
fclose(fid);

x = dhm(:,1);
y = dhm(:,2);
h = dhm(:,3);
```

could have also used readmatrix()

Build index

```python
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
```

```matlab
dx = 200;
dy = 200;
xmin = min(x);
ymin = min(y);
xidx = int16((x - xmin) / dx) + 1;
yidx = int16((y - ymin) / dy) + 1;
xdim = max(xidx);
ydim = max(yidx);
xp = xmin + double(1:xdim)*dx;
yp = xmin + double(1:ydim)*dy;
```

Create and fill grid

```python
altitudes = np.full((ydim, xdim), np.nan)
altitudes[yidx, xidx] = h
```

```matlab
altitudes = NaN(ydim, xdim);
altitudes(sub2ind(size(altitudes), yidx, xidx)) = h;
```

Plot

```python
plt.axes().set_aspect('equal')
plt.pcolormesh(xp, yp, altitudes, shading = 'auto')
```

```matlab
imagesc(xp, yp, altitudes)
colorbar
axis xy     % Y-axis orientation
axis image % Set X, Y aspect ratio
xlabel('x-coordinate (m)')
ylabel('y-coordinate (m)')
```