Files

You will learn to read and write data with files.

Lesson 1 (Write to a file)**.**
Writing to a file is almost as easy as displaying a sentence on the screen. On the left is a program that writes two lines in a file called `my_file.txt`; on the right is the resulting file that is displayed in a text editor.

```
fi = open("my_file.txt","w")

fi.write("Hello world!\n")

line = "Hi there.\n"
fi.write(line)

fi.close()
```

```
Hello world!
Hi there.
```

Explanations.

- The command `open` allows you to open a file. The first argument is the name of the file. The second argument here is `"w"` to say that you want to write to the file.
- We do not work with the file name, but with the value returned by the function `open`. Here we have named this object file `fi`. It is with this variable `fi` that we work now.
- We now write in the file almost as we would display a sentence on the screen. The instruction is `fi.write()` where the argument is a string of characters.
- To switch to the next line, you must add the line terminator character `"\n"`.
- It is important to close your file when you are finished writing. The instruction is `fi.close()`.
- The data to be written are strings, so to write a number, you must first transform it with `str(number)`.

Lesson 2 (Read a file)**.**
It is just as easy to read a file. Here is how to do it (left) and the display by `Python` on the screen (right).

```
fi = open("my_file.txt","r")

for line in fi:
    print(line)

fi.close()
```

```
Hello world!

Hi there.
```

Explanations.

- The command `open` is this time called with the argument `"r"` (for read), it opens the file in reading.
- We work again with an object file named here `fi`.
- A loop goes through the entire file line by line. Here we just ask for the display of each line.
- We close the file with `fi.close()`.
- The data read is a string, so if you want a number, you must first transform it with `int(string)` (for an integer) or `float(string)` (for a decimal number).

Activity 1 (Read and write a file)**.**

Goal: write a file of grades, then read it to calculate the averages.

1. Generates at random a grades file, named `grades.txt`, which is composed of lines with the structure:

   ```
   Firstname Name grade1 grade2 grade3
   ```

   For example:

   ```
   Tintin Vador 15.0 5.0 19.0
   Bill Croft 15.0 14.5 10.5
   Hermione Skywalker 10.5 7.0 19.5
   Lara Parker 12.5 13.0 14.5
   Hermione Croft 11.5 18.5 9.5
   Robin Vador 8.0 8.0 11.0
   ```

   Hints.
   - Build a list of first names `list_firstnames = ["Alice","Tintin","James",...]`. Then choose a first name at random by the instruction `firstname = choice(list_firstnames)` (you have to import the module `random`).
   - Same thing for names!
   - For a grade, you can choose a random number with the command `randint(a,b)`.
   - **Please note!** Don't forget to convert the numbers into a string to write it to the file: `str(grade)`.
2. Read the file `grades.txt` that you produced. Calculate the average of each person's grades and write the result to a file `averages.txt` where each line is of the form:

   ```
   Firstname Name average
   ```

   For example:

```
Tintin Vador 13.00
Bill Croft 13.33
Hermione Skywalker 12.33
Lara Parker 13.33
Hermione Croft 13.17
Robin Vador 9.00
```

Hints.

- For each line read from the file `grades.txt`, you get the data as a list by the command `line.split()`.
- **Please note!** The data read is a string. You can convert a string `"12.5"` to the number `12.5` by the instruction `float(string)`.
- To convert a number to a string with only two decimal places after the dot, you can use the command `'{0:.2f}'.format(number)`.
- Don't forget to close all your files.

Lesson 3 (Files with format *csv*)**.**
The format *csv* (for *comma-separated values*) is a very simple text file format containing data. Each line of the file contains a series of data (numbers or text). On the same line the data are separated by a comma (hence the name of the format, even if other separators are possible).

Example. Here is a file that contains the names, first names, years of birth, height and number of Nobel Prize medals:

```
CURIE,Marie,1867,1.55,2
EINSTEIN,Albert,1879,1.75,1
NOBEL,Alfred,1833,1.70,0
```

Activity 2 (csv format)**.**

Goal: write a data file, with csv *format, then read it for a graphic display.*

1. Generates a `sales.csv` file of sales figures (randomly drawn) from a sports brand.
   Here is an example:

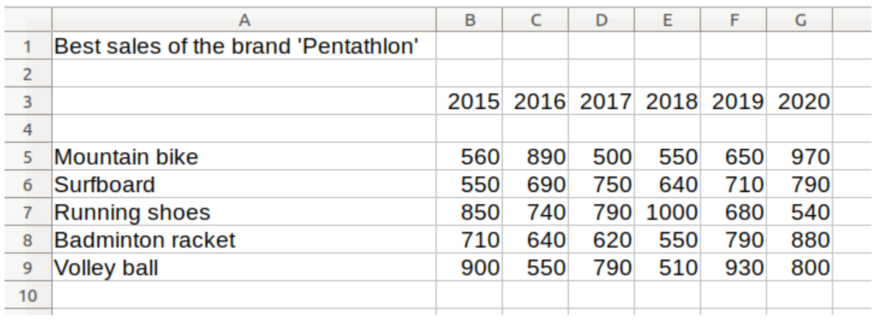
```
Best sales of the brand 'Pentathlon'

,2015,2016,2017,2018,2019,2020

Mountain bike,560,890,500,550,650,970
Surfboard,550,690,750,640,710,790
Running shoes,850,740,790,1000,680,540
Badminton racket,710,640,620,550,790,880
Volley ball,900,550,790,510,930,800
```

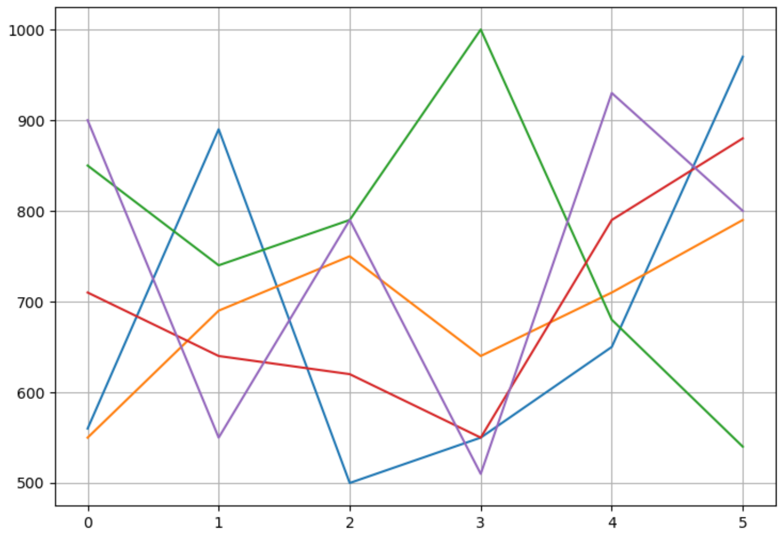
- The data starts from the fifth line.

- The produced file respects the format *csv* and must be readable by *LibreOffice Calc* for example.



	A	B	C	D	E	F	G
1	Best sales of the brand 'Pentathlon'						
2							
3		2015	2016	2017	2018	2019	2020
4							
5	Mountain bike	560	890	500	550	650	970
6	Surfboard	550	690	750	640	710	790
7	Running shoes	850	740	790	1000	680	540
8	Badminton racket	710	640	620	550	790	880
9	Volley ball	900	550	790	510	930	800
10							

2. Reads the file `sales.csv` to display the sales curves.



Hints.

- The `matplotlib` package allows you to easily display graphics, it is often called with the instruction:

```
import matplotlib.pyplot as plt
```

- Here is how to view two data lists `mylist1` and `mylist2`:

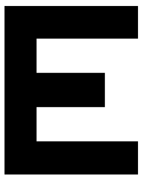
```
plt.plot(mylist1)
plt.plot(mylist2)
plt.grid()
plt.show()
```

Lesson 4 (Images *bitmap*).
There is a simple file format, called the *bitmap* format, which describes an image pixel by pixel. This format is available in three variants depending on whether the image is in black and white, grayscale or color.

Black and white picture, format "pbm".
The image is described by 0 and 1.
Here is an example: the file `image_bw.pbm` on the left (read as a text file) and on the right its visualization

(using an image reader, here very enlarged).

```
P1
4 5
1 1 1 1
1 0 0 0
1 1 1 0
1 0 0 0
1 1 1 1
```

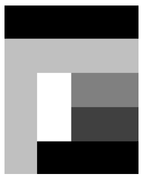


Here is the description of the format:

- First line: the id `P1`.
- Second line: the number of columns, then the number of rows (here 4 columns and 5 rows).
- Then the color of each pixel line by line: 1 for a black pixel, 0 for a white pixel. Warning: this is contrary to the usual convention!

Grayscale image, format "pgm".
The image is described by different values for different grayscales. Here is an example: the file `image_gray.pbm` on the left and on the right its visualization.

```
P2
4 5
255
  0   0   0   0
192 192 192 192
192 255 128 128
192 255  64  64
192   0   0   0
```

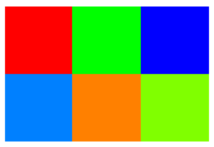


Here is the description of the format:

- First line: the identifier is this time `P2`.
- Second line: the number of columns, then the number of rows.
- Third line: the maximum value of the grayscale (here 255).
- Then the grayscale of each pixel line by line: this time 0 for a black pixel, the maximum value for a white pixel and intermediate values give intermediate grays.

Image in colors, format "ppm".
The image is described by three values per pixel: one for red, one for green, one for blue.
Here is an example: the file `image_col.ppm` on the left and on the right its visualization.

```
P3
3 2
255
255   0   0      0 255 0      0   0 255
  0 128 255    255 128 0    128 255   0
```



Here is the description of the format:

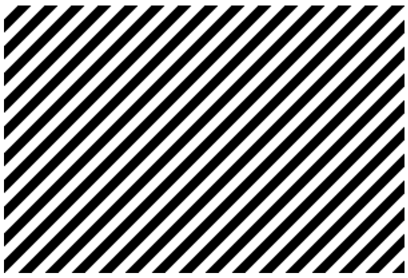
- First line: the id is now `P3`.

- Second line: the number of columns, then the number of rows.
- Third line: the maximum value of the color levels (here 255).
- Then each pixel is described by 3 numbers: the level of red, green and blue (RGB system). For example, the first pixel is encoded by $(255, 0, 0)$ so it is a red pixel.

Activity 3 (Images *bitmap*).

Goal: define your own images pixel by pixel.

1. Generate a file `image_bw.pbm` that represents a black and white image (e.g. of size 300×200) according to the following pattern:

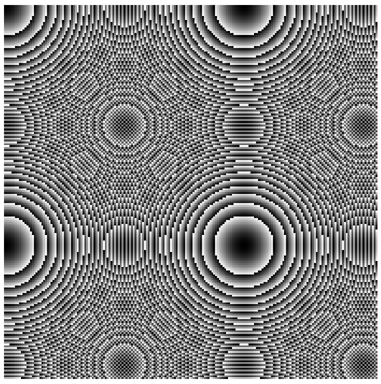


Hints. If i designates the line number and j the column number (from the top left), then the pixel in position (i, j) is white if $i + j$ is between 0 and 9, or between 20 and 29, or between 40 and 49,... This is obtained by the formula :

```
col = (i+j)//10 % 2
```

which returns 0 or 1 as desired.

2. Generate a file `image_gray.pgm` that represents a grayscale image (for example of size 200×200 with 256 grayscale) according to the following pattern:

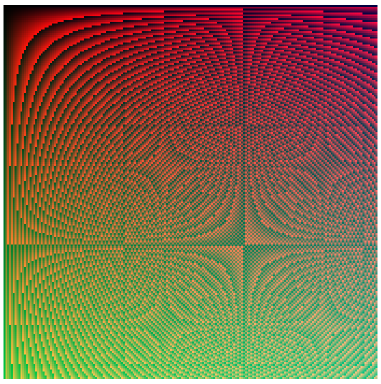


Hints. This time the formula is:

```
col = (i**2 + j**2) % 256
```

which returns an integer between 0 and 255.

3. Generate a file `image_col.ppm` that represents a color image (e.g. of size 200×200, with 256 red, green and blue levels) according to the following pattern:

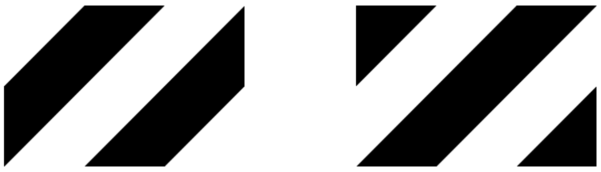


Hints. This time the formula is:

```
 R = (i*j) % 256
  G = i % 256
B = (i + j)//3 % 256
```

which gives the red, green and blue levels of the pixel (i, j).

4. Write a function `inverse_black_white(filename)` that reads a black and white image file `.pbm` and creates a new file in which the white pixels have turned black and vice versa.
   Example: on the left the start image, on the right the finish image.

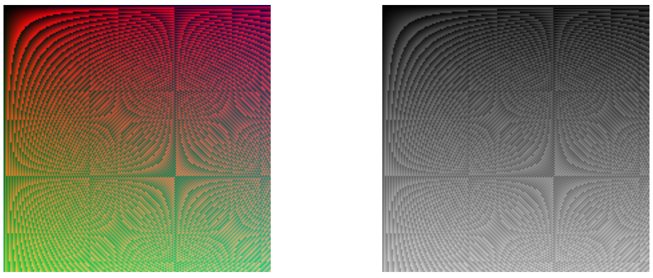


5. Write a function `colors_to_gray(filename)` that reads a color image file in `.ppm` format and creates a new file in `.pgm` format in which the color pixels are transformed into grayscale.
   You can use the formula:

$$g = 0,21 \times R \ + \ 0,72 \times G \ + \ 0,07 \times B$$

   where
   - R, G, B are the red, green and blue levels of the colored pixel,
   - g is the grayscale of the transformed pixel.

   Example: on the left the starting image in color, on the right the arrival image in grayscale.



Activity 4 (Distance between two cities).

Goal: read the coordinates of the cities and write the distances between them.

1. **Distance in the plan.**
   Write a program that reads a file containing the coordinates (x, y) of cities, then calculates and writes in another file the distances (on the map) between two cities.

The formula for the distance between two points (x_1, y_1) and (x_2, y_2) of the plan is:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

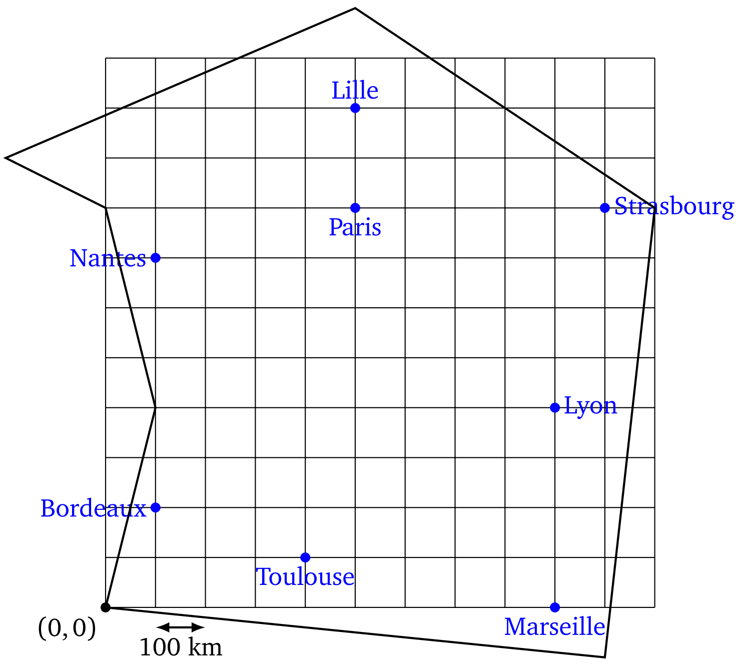
Example. Here is an example of an input file:

```
Paris 500 800
Lille 500 1000
Nantes 100 700
Marseille 900 0
```

And here is the output file produced by the program:

```
                 Paris       Lille      Nantes   Marseille
Paris                0         200         412         894
Lille              200           0         500        1077
Nantes             412         500           0        1063
Marseille          894        1077        1063           0
```

We read on this file that the distance between Lille and Marseille is 1077 kilometers.
Below is the map of France that provided (very approximate) data for the input file. The origin is at the bottom left, each side of a square represents 100 km. For example, in this reference, Paris has the coordinates $(500, 800)$.



2. **Distance on the sphere.**
On Earth, the distance between two cities is a distance on a *great circle* along the surface of the sphere and not along a straight line. This is the distance a plane travels to connect two cities.
Write a program that reads the latitudes and longitudes of cities, then calculates and writes to another file the distances (on the Earth's surface) between two cities.
Example. Here is an example of an input file:

```
Paris 48.853 2.350
New-York 40.713 -74.006
Vancouver 49.250 -123.119
Lima -12.043 -77.0282
Hong-Kong 22.286 114.158
Addis-Abeba 9.0250 38.747
```

And here is the output file produced by the program:

```
                   Paris    New-York   Vancouver        Lima   Hong-Kong Addis-Abeba
Paris                  0        5837        7924       10253        9629        5573
New-York            5837           0        3905        5874       12959       11207
Vancouver           7924        3905           0        8168       10257       13298
Lima               10253        5874        8168           0       18371       13001
Hong-Kong           9629       12959       10257       18371           0        8135
Addis-Abeba         5573       11207       13298       13001        8135           0
```

Implementation and explanations.

- The input file contains the latitude (noted φ) and longitude (noted λ) in degrees for each city. For example, Paris has a latitude of $\varphi = 48.853$ degrees and a longitude of $\lambda = 2.350$ degrees.
- For the formulas, it will be necessary to use the angles in radians. The formula for converting degrees to radians is:

$$\text{angle in radians} = \frac{2\pi}{360} \times \text{angle in degrees}$$

- **Formula for an approximate distance.**
  There is a simple formula that gives a good estimate for the shortest distance between two points of a sphere with a radius of R. First, set:

$$x = (\lambda_2 - \lambda_1) \cdot \cos\left(\frac{\varphi_1 + \varphi_2}{2}\right) \quad \text{and} \quad y = \varphi_2 - \varphi_1$$

  The approximate distance is then

$$\tilde{d} = R\sqrt{x^2 + y^2}$$

  where (φ_1, λ_1) and (φ_2, λ_2) are the latitudes/longitudes of two cities expressed in radians.
- **Exact distance formula.**
  The bravest can use the exact formula to calculate the distance. First, set:

$$a = \left(\sin\left(\frac{\varphi_1 + \varphi_2}{2}\right)\right)^2 + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \left(\sin\left(\frac{\lambda_2 - \lambda_1}{2}\right)\right)^2$$

  The exact distance is then:

$$d = 2 \cdot R \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

  where $\operatorname{atan2}(y, x)$ is the function “arctangent” which is obtained by the command `atan2(y,x)` from the module `math`.
- For the Earth radius we will take $R = 6371$ km.