# Computing with a digital twin

A first Python lab for sport scientists: student guide

Thomas Gomez – LBEPS, Université d'Évry Paris-Saclay & Icam Grand Paris Sud

**How to use this guide**

Keep this guide open next to your browser. Follow it **in order**. Each step says what to click, what to type and what you should see. You do not need any knowledge of programming. If you are stuck for more than 5 minutes, ask your neighbour, then the teacher. **Today we do not use AI assistants**: the goal is that you understand every line yourself.

## Contents

**1 What you will do** **2**

**2 Before the session (5 minutes)** **3**

**3 Opening a notebook in Google Colab** **3**

**4 Working in a notebook** **4**

4.1 Text cells and code cells . . . . . 4

4.2 The four kinds of cells in our notebooks . . . . . 5

4.3 Editing . . . . . 5

4.4 When something goes wrong . . . . . 5

**5 Python in two pages** **5**

5.1 The six building blocks . . . . . 5

5.2 Arithmetic . . . . . 6

5.3 NumPy: computing on a whole table at once . . . . . 6

5.4 Reading an error message . . . . . 6

**6 Session 1, notebook by notebook** **6**

**7 Saving figures and putting them in a document** **7**

**8 Mini-project report** **8**

**9 Session 2: the runner** **8**

**10 Session 3: learning from data** **9**

**11 Backup: JupyterLite, no account needed** **10**

**12 Glossary** **10**

# 1 What you will do

During the talk you saw the four steps of every digital twin: **data**, **model**, **calibration**, **prediction** (with its uncertainty). In this lab you go through these four steps yourself, by writing and running small Python programs.

| Notebook | What you do | Time |
|---|---|---|
| *Session 1: the cyclist's twin* | | |
| 0 First steps in Colab | run a cell, read an error message, save your work | 15 min |
| 1 Python basics | variables, lists, loops, conditions, functions, figures | 40 min |
| 2 The critical power test | compute CP and $W'$ from three tests, predict | 35 min |
| 3 A simple digital twin | simulate an interval session second by second | 40 min |
| 4 Uncertainty | noisy measurements, 1000 virtual test campaigns | 25 min |
| 5 Mini-project | *coach for a day*: a report with three figures | rest of the session |
| *Session 2: the runner* | | |
| 6 Running a 1500 m | simulate a runner, try pacing strategies | 75 min |
| 7 Mini-project | the pacing challenge: a report with three figures | 60 min |
| *Session 3: learning from data* | | |
| 8 Learning from examples | decision tree, random forest, your own learning loop, neural network | 90 min |
| 9 A physics-informed network | a network that respects an equation | 45 min |
| 10 Mini-project | *physics or data?*: a report with three figures | rest of the session |

You will work in a **notebook**: a web page that mixes text and small pieces of Python code that you run one by one. The notebooks run in **Google Colab**, a free service: the computation is done on Google's computers and you only need a web browser.

## 2 Before the session (5 minutes)

**Checklist**

1. A laptop (a tablet works, a phone is too small) with **Chrome**, **Firefox**, **Edge** or **Safari**.
2. A **Google account**. If you have a Gmail address, you have one. Otherwise create one for free at `https://accounts.google.com/signup` (choose "For my personal use"). Write down your password!
3. Check that you can open `https://colab.research.google.com` and sign in.
4. Have a text editor to write your report: Word, LibreOffice Writer or Google Docs (free, in the browser).

**Tip**

No Google account and you do not want one? Use the backup in Section 11 (JupyterLite): it works without any account, directly in the browser.

## 3 Opening a notebook in Google Colab

**Tip**

**Fastest way**: open `https://thomas-gomez.fr/python-lab/`, click **Open in Colab** on the notebook, then **Copy to Drive**, and work in the tab *Copy of ...*. The printed sheet *Getting started, click by click* (also on the lab page) shows every click, with drawings of the screens, for Notebooks 0 and 1.

If the teacher gives you a Google Drive link or a zip file instead, follow way A or way B.

### Way A: the link opens a Google Drive folder (recommended)

**Step by step**

1. Click the link. A Google Drive page opens with the folders `session1`, `session2` and `session3`. Sign in if Google asks you to.
2. Double-click `session1`, then double-click the file `00_first_steps_in_colab.ipynb`.
3. Drive shows a preview (sometimes a page of strange text). At the top of the page, click **Open with** (or the small arrow next to it) and choose **Google Colaboratory**.
   - If **Google Colaboratory** is not in the list: click **Connect more apps**, search for `Colaboratory`, click **Install**, then try again.
4. The notebook opens in Colab. It belongs to the teacher: you can run it but your changes would not be saved. Make it yours: menu **File** → **Save a copy in Drive**.
5. A new tab opens with a notebook called `Copy of 00_first_steps_in_colab.ipynb`. **Work in this copy.** It is saved automatically in your Google Drive, in a folder called `Colab Notebooks`.

### Way B: the link opens a web page with a zip file

**Step by step**

1. Click the link (for example `https://thomas-gomez.fr/python-lab/`) and download `python-lab-notebooks.zip`.
2. Find the file in your `Downloads` folder and unzip it (Windows: right-click → **Extract All**; Mac: double-click).
3. Go to `https://colab.research.google.com` and sign in.
4. A window "Open notebook" appears (if not: menu **File** → **Upload notebook**). Click the tab **Upload**, then **Browse**, and choose `00_first_steps_in_colab.ipynb` in the unzipped folder.
5. The notebook opens. It is already your own copy, saved in your Google Drive (`Colab Notebooks` folder).

Repeat the same steps for each new notebook (01, 02, . . . ). Next time, you find all your copies in Google Drive, folder `Colab Notebooks`, or in Colab with **File** → **Open notebook**.

# 4 Working in a notebook

## 4.1 Text cells and code cells

A notebook is a column of **cells**:

- **text cells**: explanations, questions, instructions. You read them.
- **code cells**: grey boxes with Python code. You **run** them; the result appears just below.

**Running a code cell**

1. Click once inside the grey box.
2. Click the round ▶ button on its left, **or** press `Shift` + `Enter`.
3. Wait: while the cell runs, the button turns into a spinning circle. When it is done, a small number or a green tick appears, and the result is shown under the cell.

The first time, Colab may say *"Warning: This notebook was not authored by Google"*: click **Run anyway**. It may take 10 to 20 seconds to start (Colab is switching on a computer for you).

### 4.2 The four kinds of cells in our notebooks

| The cell starts with... | What you do |
|---|---|
| `# SETUP CELL` | Run it first, at the top of each notebook. You do not need to understand it. |
| (normal code) | Read it, run it, look at the result. |
| `# YOUR TURN` or a `...` | An exercise: replace the `...` by your code, then run. |
| `check(...)` | Tells you whether your answer is right: `Correct!`, `Not yet` with a hint, or `Replace the ...`. |

**Watch out**

**Order matters.** Python remembers what was run *before*. If you open a notebook again later, or if you get a `NameError` about something that exists above, run the cells again from the top: menu **Runtime** → **Run before**, or **Runtime** → **Run all**.

### 4.3 Editing

- Change a number in a code cell and run it again: this is how you explore. Nothing can break.
- To add a cell: hover between two cells and click **+ Code** or **+ Text**.
- To undo: Ctrl + Z (Mac: Cmd + Z).
- To edit a text cell (for your answers): double-click on it; click outside to finish.

### 4.4 When something goes wrong

**Reset in three clicks**

1. Menu **Runtime** → **Restart session** (Python forgets everything).
2. Menu **Runtime** → **Run all** (runs all the cells again from the top).
3. Still stuck? Reload the page. Your copy is in Google Drive.

## 5 Python in two pages

### 5.1 The six building blocks

```
power = 250                      # a VARIABLE: a named box that holds a value
powers = [300, 350, 450]         # a LIST: several values (positions start at 0)
print(powers[0])                 # shows 300

for p in powers:                 # a LOOP: repeat for each value of the list
    print(p, "W")                # the lines inside the loop are shifted by 4 spaces

if power > 260:                  # a CONDITION: choose what to do
    print("above CP")
else:
    print("below CP")
```

```
def work(p, t):                  # a FUNCTION: a recipe you can reuse
    return p * t
print(work(300, 60))             # shows 18000

plt.plot([0, 1, 2], [3, 5, 4])   # a FIGURE: y against x
plt.xlabel("Time (s)")
plt.show()
```

### 5.2 Arithmetic

| | | | |
|---|---|---|---|
| + add | - subtract | * multiply | / divide |
| ** power: `2 ** 3` is 8 | `round(x, 1)` rounds | `min(a, b)`, `max(a, b)` | `abs(x)` absolute value |

Decimal numbers are written with a **point**: `6.5`, never `6,5` (a comma separates two values).

### 5.3 NumPy: computing on a whole table at once

```
times = np.array([145, 280, 510])
powers = np.array([400, 330, 300])
work = powers * times                          # multiplies value by value
t = np.linspace(0, 600, 100)                   # 100 values from 0 to 600
slope, intercept = np.polyfit(times, work, 1) # best straight line through the points
```

### 5.4 Reading an error message

An error message is red and ends with a line like `NameError: name 'Hello' is not defined`. **Read the last line first**; the arrow `-->` above it shows the line of your code that failed.

| Last line | Usual cause and fix |
|---|---|
| `NameError` | a typo in a name (Python is case sensitive: `Power` ≠ `power`), missing quotes around a text, or a cell above not run. |
| `SyntaxError` | a missing bracket `)`, comma or colon `:` at the end of a `for`, `if` or `def` line; or a decimal comma. |
| `IndentationError` | the spaces at the start of a line are wrong: lines inside a loop or an `if` are shifted by exactly 4 spaces. |
| `TypeError` | a `...` left in your code, or a text used as a number. |
| `ZeroDivisionError` | a division by zero (for example a power exactly equal to CP). |

## 6 Session 1, notebook by notebook

### Notebook 0 – First steps in Colab (15 min)

Run your first cells, use Python as a calculator, meet an error message on purpose, save a copy. **You should see:** `Correct!` after the exercise on the duration of a ride.

### Notebook 1 – Python basics (40 min)

The six building blocks with sport numbers (power, mass, heart rate). Four exercises, each with an automatic check. The last exercise downloads your first figure: **check that the file** `my_first_figure.png` **is in your Downloads folder** (see Section 7).

### Notebook 2 – The critical power test (35 min)

Alex did three rides at constant power until exhaustion. You plot the work against the time, see a straight line, and let the computer find it:

$$\text{work} = W' + \text{CP} \times \text{time}.$$

The slope is the critical power CP; the intercept is the reserve $W'$. Then you predict the time to exhaustion at any power above CP: time $= W'/(P - \text{CP})$. **You should find:** CP close to 261 W and $W'$ close to 19.9 kJ.

### Notebook 3 – A simple digital twin (40 min)

The reserve $W'$ is a tank. Each second, it loses $P - \text{CP}$ joules above CP and refills a little below CP. The computer repeats this rule second by second and finds when the tank is empty (exhaustion). You then simulate interval sessions and look for the hardest session the athlete can complete.

### Notebook 4 – Uncertainty (25 min)

The computer repeats a virtual test campaign 1000 times with random errors on the times and recalibrates each time. The spread of the CP values is the uncertainty. You compare test designs: **which tests teach the most about the athlete?**

### Notebook 5 – Mini-project: coach for a day

See Section 8.

## 7 Saving figures and putting them in a document

**Save a figure from a notebook**

1. In the cell that draws the figure, add `save_figure("a_name")` **just before** `plt.show()`:

```
plt.plot(times, powers, "o")
plt.xlabel("Time (s)")
save_figure("fig1_calibration")
plt.show()
```

2. Run the cell. In Colab the image `fig1_calibration.png` is downloaded to your computer (folder `Downloads`). The first time, the browser may ask whether the site may download files: click **Allow**.
3. If nothing is downloaded: click the **folder icon** in the left bar of Colab, find the file, click the three dots next to it, then **Download**.
4. Quick alternative: right-click on the figure in the notebook and choose **Save image as...**.

**Insert the figure in your report**

- **Word / LibreOffice**: **Insert** → **Pictures (Image)** → **This device**, choose the `.png` file.
- **Google Docs**: **Insert** → **Image** → **Upload from computer**.
- **PowerPoint**: **Insert** → **Pictures**.

Add a caption under each figure: "Figure 1. . . . " with one sentence that says what is shown.

**Watch out**

Files saved *inside* Colab (left folder icon) are deleted when the session ends. Download your figures before leaving.

# 8 Mini-project report

Each pair coaches one athlete (A, B, C or D, given by the teacher). Work in `05_mini_project_coach`: all the tools are loaded by its setup cell; fill in the cells marked `YOUR TURN`.

**Report template (1 to 2 pages)**

**Title:** Digital twin of athlete _____ – names of the pair.
**1. Calibration.** CP $=$ . . . W, $W' =$ . . . kJ. *Figure 1*: work against time, with the fitted line. One sentence: are the three points close to the line?
**2. Prediction.** A small table: power (W) and predicted time to exhaustion (s) at 1.1, 1.3 and 1.6 $\times$ CP.
**3. Training session.** Describe your session (warm-up, efforts, recoveries, number of repetitions). *Figure 2*: power and $W'$ tank. One sentence: what would you tell the athlete?
**4. Uncertainty.** Typical error (standard deviation) on CP $=$ . . . W. *Figure 3*: histogram. Which extra test would make CP more precise? (Use Notebook 4.)
**5. Limits.** One limit of this model (for example: what it says, or does not say, below CP).

Save the report as PDF (**File** → **Save as** / **Download** → **PDF**) and hand it in as the teacher asks.

# 9 Session 2: the runner

## Notebook 6 – Running a 1500 m (75 min)

A runner is described by four quantities that change during the race: position, speed, propulsive force (per kg) and anaerobic energy. Every 0.1 s:

- the speed increases with the push and decreases with friction;
- the anaerobic energy is refilled by the aerobic power (oxygen) and used by the push $\times$ speed;
- when the energy is empty, the runner can only use the aerobic power and slows down.

This is a simplified version of the model of Aftalion & Trélat (2021). A **strategy** is a Python function that gives the target speed at each distance; the function `race` turns it into a race and a final time. You try even paces, a fast start and a final sprint, and use a loop to find the best even pace.

### Notebook 7 – Mini-project: the pacing challenge (60 min)

1. Your best 1500 m strategy, compared with the optimal one computed by mathematicians (Figure 1).
2. The best even pace for 800, 1500 and 3000 m (Figure 2).
3. A different runner: change one physiological parameter and see what it changes (Figure 3).
4. A short report with the three figures and answers to the questions of the notebook (same format as in Section 8).

## 10 Session 3: learning from data

In sessions 1 and 2, the rules of the model were written by scientists. In **machine learning**, the computer is given **examples** (an input and the right answer) and finds a rule that reproduces them. Every learning tool is used in three steps:

```
model = make_forest()   →   model.fit(inputs, answers)   →
                        model.predict(new_inputs)
```

### Notebook 8 – Learning from examples (90 min)

Alex did 40 exhaustion tests. You keep 30 to **learn** (training set) and 10 to **test** (test set), then you train:

- a **decision tree**: yes/no questions on the power ("is the power below 325 W?"); its prediction is a staircase;
- a **random forest**: 100 trees trained on random parts of the data, whose answers are averaged;
- your own **learning loop**: you learn CP and $W'$ of Alex by gradient descent, the loop that is inside every `fit`:

$$L(a,b) = \frac{1}{n}\sum_i (b + a\,t_i - y_i)^2, \qquad a \leftarrow a - \eta\,\frac{\partial L}{\partial a}, \qquad b \leftarrow b - \eta\,\frac{\partial L}{\partial b},$$

  and you see what the learning rate $\eta$ does (too small: slow; too large: the loss explodes);
- a **neural network**: neurons that multiply by weights, add and bend; the same loop adjusts its 500 weights (the gradients come from the chain rule: back-propagation).

You measure the mean error on the test set, see **overfitting** (a deep tree learns the noise by heart), and compare the three methods with the critical power model, also outside the powers of the data.

### Notebook 9 – A physics-informed network (45 min)

Oxygen uptake rises at the start of an exercise following $\tau\,dV/dt = V_{\text{end}} - V$. With only 6 measurements in the first 90 s, you train the same small network twice: on the data only, and on the data *plus* the equation (a PINN). The second one predicts the whole curve and learns the time constant $\tau$ of the athlete.

**Tip**

In JupyterLite, the first setup cell of notebooks 8 and 10 downloads the learning library (about 30 seconds). Wait for the green tick before going on. Each training of notebook 9 takes a second or two: be patient.

### Notebook 10 – Mini-project: physics or data? (rest of the session)

1. Error of the forest, the network and the physics model against the number of training tests (Figure 1).
2. Error of the network and of the PINN against the number of oxygen measurements (Figure 2).
3. Time constant $\tau$ learned by the PINN against the number of measurements (Figure 3).
4. A short report with the three figures and your advice to a coach: when to use machine learning, a physiological model, or both.

## 11 Backup: JupyterLite, no account needed

If Colab does not work for you (no Google account, blocked network), use JupyterLite: Python runs *inside* your browser.

**Step by step**

1. Open `https://thomas-gomez.fr/python-lab/` and click **JupyterLite**.
2. Wait until the page is loaded (the first time it can take a minute: your browser downloads Python).
3. In the file list on the left, double-click `session1`, then the notebook you want.
4. Run cells exactly as in Colab: `Shift` + `Enter` or the ▶ button at the top.
5. Figures: `save_figure("name")` saves the image in the file list on the left; right-click it → **Download**.

**Watch out**

JupyterLite keeps your work *in this browser only*. Download your notebook (**File** → **Download**) and your figures before leaving, and do not clear the browser data.

## 12 Glossary

**Algorithm** a list of precise steps that solves a problem (a recipe).

**Back-propagation** computing the gradient of a network with the chain rule, from the loss back to each weight.

**Calibration** finding the parameters of a model that make it fit the data of one athlete.

**Cell** a box of a notebook: text, or code that you run.

**Compiled language**
: the whole program is translated once into machine code before running (C, C++, Fortran): fast.

**CP**
: critical power: the highest power that can be held for a long time without using the limited reserve $W'$.

**Digital twin**
: a model of one athlete, calibrated on their data, used to predict and to decide.

**Extrapolation**
: predicting outside the range of the data used to build the model.

**Function**
: a named recipe in a program, with inputs and a result (`def ... return ...`).

**Gradient**
: the derivatives of the loss with respect to each weight: it points uphill.

**Gradient descent**
: the way a model learns: change the weights by small steps in the direction that lowers the error.

**Interpreted language**
: the program is translated and run line by line by an interpreter (Python, R): easy to try, slower loops.

**Learning rate**
: $\eta$, the size of the steps of gradient descent.

**Least squares**
: the method that finds the line (or the parameters) closest to the data points.

**Loop**
: repeating some instructions (`for`, `while`).

**Loss**
: the error that the training makes as small as possible (for example the mean squared error).

**Machine learning**
: methods in which the computer learns a rule from examples instead of being given it.

**Model**
: equations that describe how the state of the athlete changes, with parameters.

**Monte Carlo**
: repeating a computation many times with random numbers to see the spread of the results.

**Neural network**
: a learning model made of neurons (multiply by weights, add, bend) organised in layers.

**Notebook**
: a page that mixes text and code (`.ipynb` file).

**Overfitting**
: learning the training examples by heart, noise included, so that new examples are predicted badly.

**Parameter**
: a fixed number that describes one athlete (CP, $W'$).

**PINN**
: physics-informed neural network: a network trained on the data and on an equation at the same time.

**Random forest**
: many decision trees, each trained on a random part of the data; their answers are averaged.

**Simulation**
: computing the evolution of the model step by step in time.

**Standard deviation**
: the typical distance between a value and the average: a measure of uncertainty.

**State**
: a quantity that changes during the effort (energy left, speed).

**Training / test set**
: the examples used to learn, and the examples kept apart to check what was learned.

**Variable**
: a named box in the computer's memory (`power = 250`).

$W'$
: "W prime": the size of the limited reserve of energy used above CP, in joules.

References. Monod H, Scherrer J (1965) Ergonomics 8:329–338, doi:10.1080/00140136508930810. Skiba PF et al. (2012) Med Sci Sports Exerc 44:1526–1532, doi:10.1249/MSS.0b013e3182517a80. Aftalion A, Trélat E (2021) J Math Biol 83:9, doi:10.1007/s00285-021-01632-z. Boillet A et al. (2024) Sci Rep 14:5470, doi:10.1038/s41598-024-56042-0. Breiman L (2001) Mach Learn 45:5–32, doi:10.1023/A:1010933404324. Poole DC, Jones AM (2012) Compr Physiol 2:933–996, doi:10.1002/cphy.c100072. Raissi M, Perdikaris P, Karniadakis GE (2019) J Comput Phys 378:686–707, doi:10.1016/j.jcp.2018.10.045.