



2026 Dig-IT Lab Online Summer School

## Hands-On Workshop

**Design of an efficient and resilient building  
considering global warming**

with an open-source interactive energy management  
toolbox

author: [stephane.ploix@grenoble-inp.fr](mailto:stephane.ploix@grenoble-inp.fr)

## Agenda (starting time @T0=8:15, Stockholm time)

[8:15, T0] - Short introduction to the workshop (15')

[8:30, T0+15'] - The **baterm** capabilities (30')

[9:00, T0+45'] - Subject to solve (15')

[9:15, T0+1h'] - **Work in 10 groups with regular visits of your professor (6h)\***

[16:00, T0+8h15] - **Final presentation of each group results (1 student per group) and discussions (2h)**

[18:00, T0+10h15] - End of the workshop

\* you can take breaks during this period, in particular the lunch break from [12:30, T0+4h15] till [13:45, T0+5h30] (1h15')

\* Save your replies to each question in a separate document.

**If your university requests evidence of your attendance, you can attach the results of this workshop to the certificate of attendance provided by KTH.**

General introduction:

## **What is a building? A system providing services to its occupants**

- protection from the weather
- thermal comfort for its occupants
- storage of goods
- cooking means
- quietness during night
- local production of electricity and heat
- washing means (bodies, clothes, dishes, ...)
- parking for vehicles and recharging means
- grid system services
- ...

# What is a smart building?

I) A system providing smart-services to its occupants

- listening to its occupants
- learning from physics and occupants' feedback
- revealing, suggesting actions or, possibly, acting for building performance
- monitoring and predicting building performance
- ...

II) A smartly designed building system

- this is the topic of this workshop

## What is **baterm**?

- a **command line tool** to design **efficient building systems**.
- fast learning curve.
- a set of tools to:
  - analyze a **context** i.e. what is around **a building and/or a PV plant**?
  - simulate the **building performance**
  - simulate a **PV plant**
  - analyze the joint performance: **building + PV plant**

# Fast design but precise simulation model

- heat transfer by conduction, convection and radiation
- solar gains through windows, walls and PV panels considering solar masks
- close (building itself, surrounding buildings) and far (mountains) shadings
- air infiltration and forced ventilation
- HVAC system for cooling and heating
- types of roofs
- thermal inertia
- cell connections in photovoltaic panels and impact of temperature
- mounting type of PV panels
- ...

# Understanding what matters in building design

Building efficiency is not just a matter of insulation

- location, shape and direction of the building (including roof)
- adjacency to other buildings
- location and glazing surfaces
- materials used
- tuning of the HVAC system
- occupancy and comfort demands
- PV plant and autonomy
- ...

Thanks to `baterm`, it's easy to understand what matters in building design

## batem or baterm ?

- `baterm` is part of `batem` .
- `batem` is a Python module containing:
  - tutorials, named `slidesXX-short_name.pdf`
  - workshops, named `workshopsXX-short_name.ipynb`
- Each tutorial has a corresponding workshop (a Jupyter Notebook) to learn by practice.
- `batem` is also a tool for research to experiment new ideas

The tutorials and workshops are:

- 00: introduction (no workshop, 4h)
- 01: modeling the Sun - Earth system and their effects (4h)
- 02: modeling climate changes and global warming (4h)
- 03: modeling human comfort and energy impacts (4h)
- 04: building physics (4h)
- 05: modeling inertia and dynamical phenomena (4h)
- 06: HVAC system for cooling and heating (4h)
- 07: fitting models of existing buildings (4h)
- 08: managing buildings to improve their energy performance (4h)
- 09: modeling photovoltaic plants (4h)

- 10: simulating energy communities (4h)
- 11: graph theory to model ski-resort systems (4h)
- 12: using multi-agent systems to model skiers (4h)
- 13: strategy for good compromises between comfort and energy performance in a ski-resort (4h)
- 14: pleiades for thermal dynamic simulation (8h)
- 15: fast design of building systems with `baterm` (this tutorial, 8h)

Each session can be followed alone.

# Installation

You need Python (3.13 best) accessible from the command line, then:

```
python3 -m pip install -U batem or python3.exe -m pip install -U batem
```

Then go to your working directory and type `batem` in the command line  
(or `python3 -m batem.batem` or for Windows: `python3.exe -m batem.batem`)

Note that the command line is: `Terminal.app` on macOS, `PowerShell.exe` on Windows  
and `Terminal` (or other) on Linux

`batem` source code is available on GitLab: [https://gricad-gitlab.univ-grenoble-alpes.fr/batem/batem\\_all](https://gricad-gitlab.univ-grenoble-alpes.fr/batem/batem_all)

**Note that the first time you work with a location, there might be an important delay for downloading the weather data: don't interrupt the process.**

On Windows, if the commands `python3.exe` or `pip3.exe` are not found, it's likely because these files are not in the `PATH` environment variable.

You can add them to the `PATH` by:

1. Opening the System Properties
2. Clicking on the "Environment Variables" button
3. Adding the path of the Python installation directory (e.g., `C:\Python313` ) to the "Path" variable
4. Clicking "OK" to save the changes
5. Open a new terminal and check if the commands are found.

## The basic commands of **baterm**

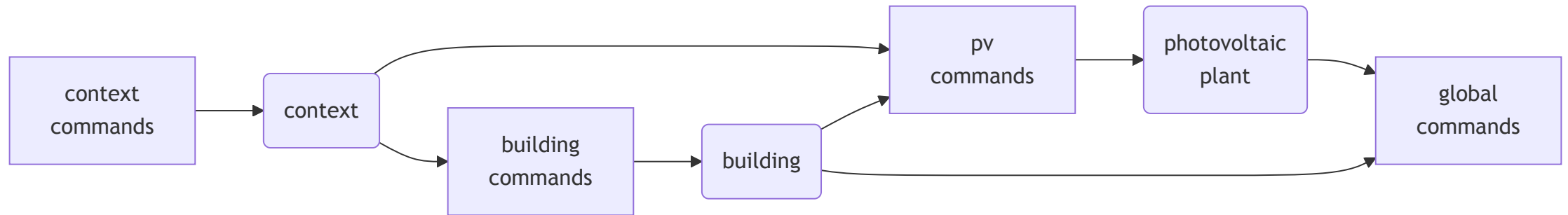
| command      | description                                                       |
|--------------|-------------------------------------------------------------------|
| help         | show help for a command ( <code>help alias</code> , for instance) |
| alias        | list all aliases (replace a string by another)                    |
| dir          | list all directories in the working folder                        |
| cd           | change the working directory                                      |
| pwd          | print the current working directory                               |
| history      | show the history                                                  |
| manual       | show the manual                                                   |
| quit or exit | quit the program                                                  |

| <b>command</b>        | <b>description</b>              |
|-----------------------|---------------------------------|
| shell                 | run a shell command             |
| macro                 | list all macros (batem scripts) |
| → macro create        | create or overwrite a new macro |
| → macro delete        | delete a macro                  |
| → macro list          | list all macros                 |
| load                  | load a macro (baterm script)    |
| run                   | run a macro (baterm script)     |
| shortcuts             | list available shortcuts        |
| ls                    | list all files                  |
| pyrun or run_pyscript | run a Python script             |

## The batem commands

| command        | description          |
|----------------|----------------------|
| lectures       | list all lectures    |
| → lectures 15  | show the lecture 15  |
| workshops      | list all workshops   |
| → workshops 15 | show the workshop 15 |

# Commands are structured around 3 objects: context, building and photovoltaic plant.



A **context** is a location with a weather and possible solar **side masks** (modeling for instance, surrounding buildings).

# The context commands

A context is mainly defined by a location:

- a latitude North and a longitude East in decimal degrees.

It is used to load the historical weather data from <http://open-meteo.com>, to get:

- the **elevation** of the location
- the far masks due to **surrounding mountains**.

A context can be created with the command `context new`.

Default values are proposed. Then it can be modified with `context edit`.

Note the tip on the right hand side of each value: it is a documentation for the value.

The name appearing in the location field is used to save the collected data to avoid re-downloading them each time.

The context data are shown below: the historical weather data (from January 1st, 1980 till nowadays) will be saved in `Grenoble.json`.

|                          |                                                           |                                                                                                                                                   |
|--------------------------|-----------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| latitude_north_deg       | <input type="text" value="45.1916"/>                      | <input data-bbox="1635 291 1768 334" type="button" value="?"/>                                                                                    |
| longitude_east_deg       | <input type="text" value="5.7174"/>                       | <input data-bbox="1635 362 1768 405" type="button" value="?"/>                                                                                    |
| location                 | <input type="text" value="Grenoble"/>                     | <input data-bbox="1635 434 1768 476" type="button" value="?"/>                                                                                    |
| albedo                   | <input type="text" value="0.1"/>                          | <input data-bbox="1635 505 1768 548" type="button" value="?"/>                                                                                    |
| pollution                | <input type="text" value="0.1"/>                          | <input data-bbox="1635 576 1768 619" type="button" value="?"/>                                                                                    |
| side_masks               | <div><div></div></div>                                    | <input data-bbox="1635 648 1768 691" type="button" value="?"/>                                                                                    |
| simulated_year           | <input type="text" value="2022"/>                         | <input data-bbox="1635 948 1768 991" type="button" value="?"/>                                                                                    |
| year_average_temperature | <input type="text" value="14.83 °C (used as TZ:ground)"/> | <input data-bbox="1635 1019 1768 1062" type="button" value="?"/>                                                                                  |
|                          |                                                           | <div><input data-bbox="1368 1098 1556 1140" type="button" value="Cancel"/><input data-bbox="1582 1098 1768 1140" type="button" value="OK"/></div> |

**Albedo** is the average reflectivity of the surface: 0 means no reflection, 1 means full reflection. It is used to compute the solar radiation on the surface. (default value is 0.1)

The **pollution coefficient** is used to compute the solar radiation on surfaces.

The pollution level is a value between 0 and 1, 0 means no pollution, 1 means maximum pollution. (default value is 0.1)

**Sidemasks** are used to model the surrounding buildings. They are defined by a list of rectangular surfaces with their dimensions and angles.

The simulated year is the year of the historical weather data to be used. It should be between 1980 and the year before the current year because full year data is needed for the simulation.

The year average temperature is the average temperature of the year. It is used to compute the ground temperature but it can be changed if needed.

A documentation for the side-masks can be found in the file [doc\\_sidemasks](#).

Here is an example of a side-mask (several side-masks can be defined ; they just have to be separated by a comma):

```
{  "name": "left",
    "x_center": 2.5,
    "y_center": -20.0,
    "width": 15.0,
    "height": 6.0,
    "elevation": 0.0,
    "exposure_deg": 180.0,
    "slope_deg": 90.0,
    "normal_angle_with_south_deg": 0.0,
    "normal_rotation_angle_deg": null,
    "tangent_x": NaN,
    "tangent_y": NaN,
    "facade_normal_x": NaN,
    "facade_normal_y": NaN
}
```

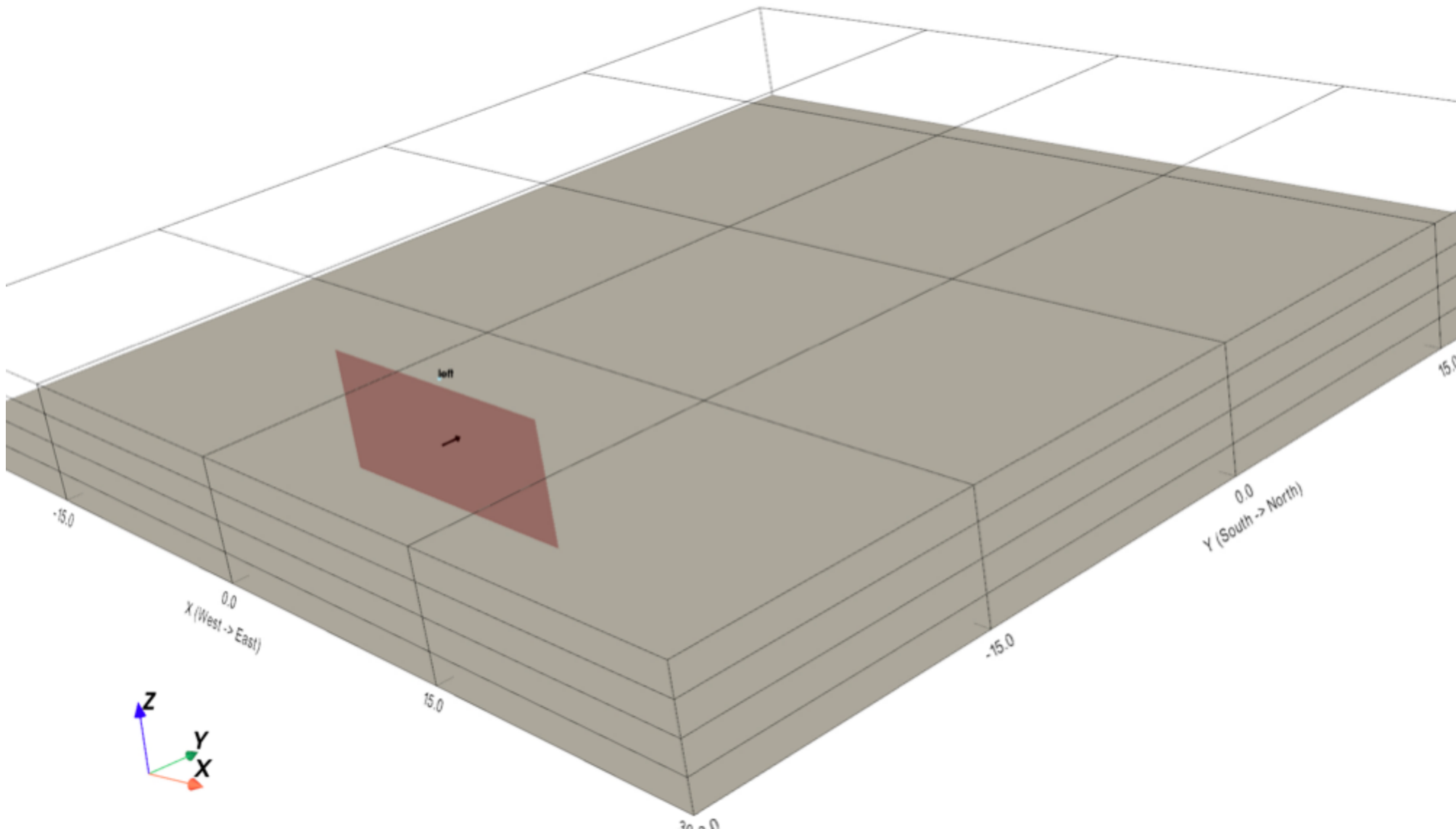
Here, side-masks are used to model vertical rectangular surfaces (slope =  $90^\circ$ ) touching the ground (elevation = 0).

Only the lower middle position (at the center of the edge cutting the ground) has to be defined ( $x\_center$ ,  $y\_center$ ) and the dimension of the side (width, height). Finally the direction of the normal has to be defined:  $exposure\_deg=0$  (or  $\pm 180^\circ$  or  $360^\circ$ ) means South or North,  $exposure\_deg= \pm 90^\circ$  means East or West.

A side-mask for a wall requires only 5 values to be defined:

```
{ "name": "mask_name",  
  "x_center": ?,  
  "y_center": ?,  
  "width": ?,  
  "height": ?,  
  "elevation": 0.0,  
  "exposure_deg": ?,  
  "slope_deg": 90.0,  
}
```

You can see the result with the command `context 3d :`



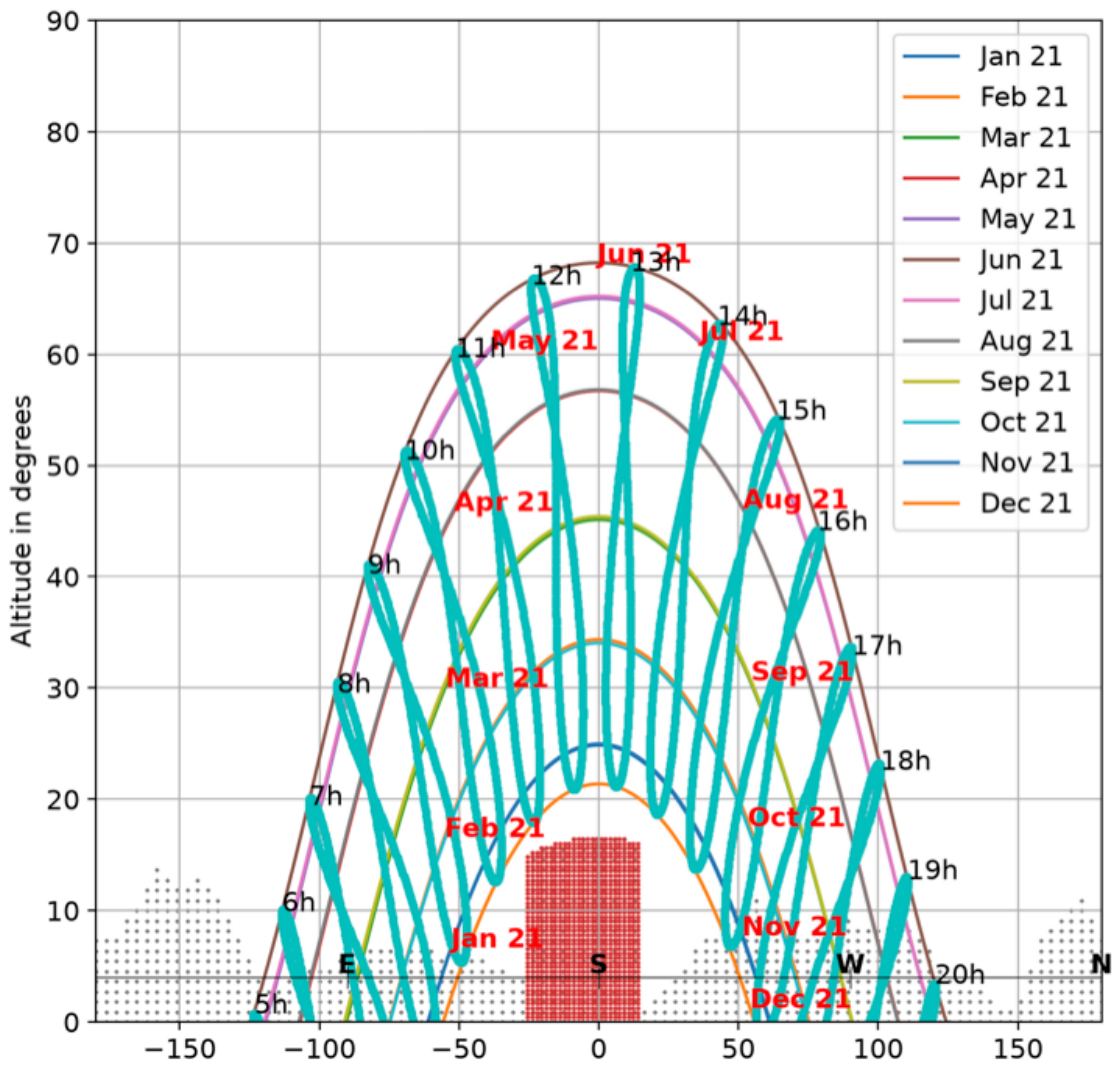
The heliodon, representing the sun path each 21st of months of the year, is shown with the command `context heliodon` knowing that the observer is located at the (0,0,0) position i.e. at the specified gps coordinates.

The red shape is the defined side-mask (sun is hidden behind it when it passes at this position).

The gray mask stands for the surrounding mountains.

As we can see, this mask is never hiding the sun.

Here is an example of heliodon for the context:



| <b>command</b>    | <b>description</b>                          |
|-------------------|---------------------------------------------|
| context           | show the current context                    |
| context delete    | delete a context                            |
| context new       | create a new context                        |
| context edit      | edit a context                              |
| context heliodon  | show the heliodon                           |
| context 3d        | show the 3d view of the context             |
| context sidemasks | show the features of the defined side masks |

Many other tools are available to analyze the context:

| <b>command</b>      | <b>description</b>                                                            |
|---------------------|-------------------------------------------------------------------------------|
| context rain        | graphical tool to analyze the rain data                                       |
| context temperature | graphical tool to analyze the temperature data                                |
| context wind        | graphical tool to analyze the wind data                                       |
| context solar       | graphical tool to analyze the solar data                                      |
| context psychro     | psychrometric chart to analyze how the weather is comfortable                 |
| context evolution   | graphical tool to analyze the evolution of the outdoor temperature since 1980 |

| <b>command</b> | <b>description</b>                          |
|----------------|---------------------------------------------|
| context vars   | list all variables available in the context |
| context plot   | plot any variables of the context           |

# The building commands

A building is defined by its geometry (footprint, number of floors, heights), its envelope (wall / roof / floor / glazing compositions), its HVAC settings (setpoints, heating and cooling powers, air renewal) and its occupancy.

It needs a context first: the weather and the solar masks come from it.

A building can be created with `building new`. Default values are proposed. Then it can be modified with `building edit`.

As for the context, each field has a tip on the right-hand side.

The building is saved as a `.building` JSON file under `./data` (for instance `building save myhouse` → `./data/myhouse.building`).

The footprint is a list of 2D points in meters: `[0,0],[10,0],[10,5],[0,5]`. Axes: **+X East, +Y North, +Z up**. It can be visualized with the command `building footprint`.

Side indices start at 0 and follow the footprint edges. Use `building footprint` to check them. Sides can be visualized with the command `building footprint` too.

Glazing ratios can be one value same for all façades, or one value per side. It represents the fraction of the side that is glazed.

Compositions are stacks of layers **from indoor to outdoor**, for example:

```
["plaster",0.013],["insulation",0.15],["brick",0.1]
```

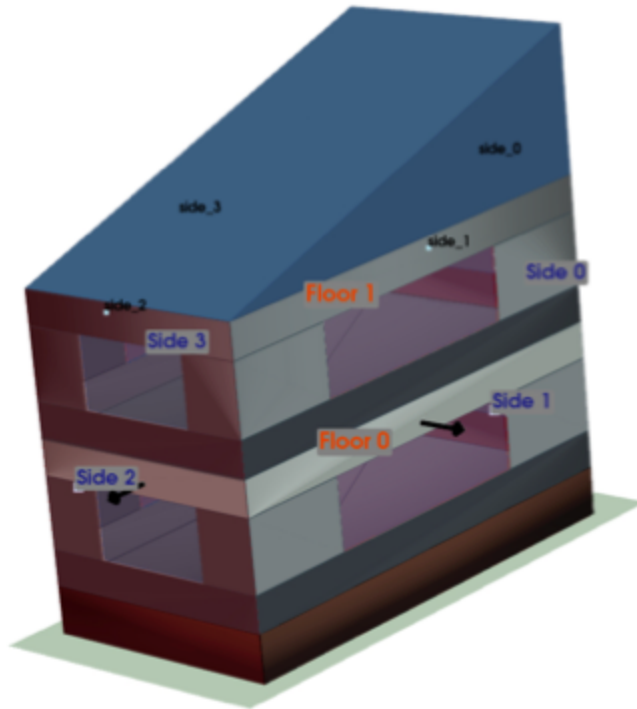
Material names come from `propertiesDB.xlsx` (see `material list thermal`) but only the ones with a star in front are loaded. If you load another one, use the command `material load <a_name> <row_number>`.

## Roofs

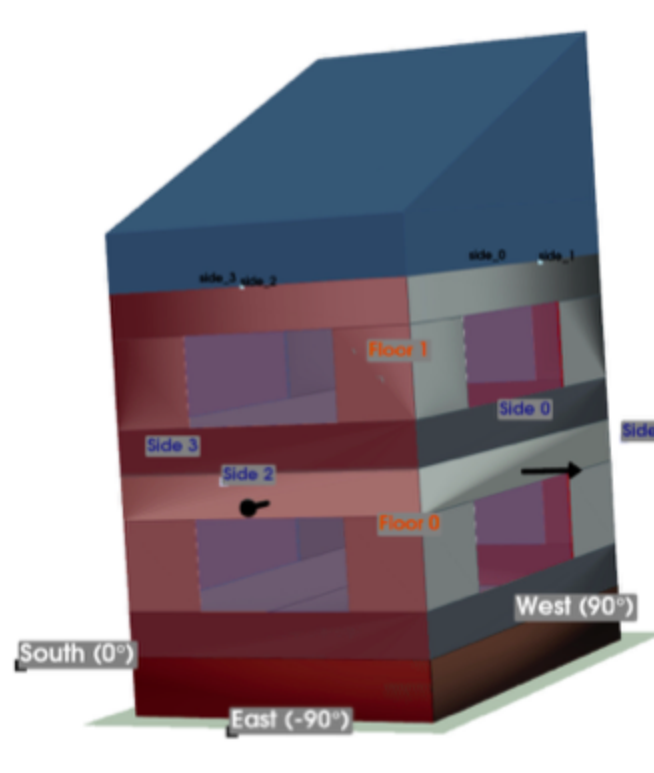
A roof can be extruded along a given height: `roof_extrusion: 1.0`. There will be an additional 1 m strip in roof material in the continuity of the walls.

It can have a wall ratio: `roof_wall_ratio: 0.4`. It represents the fraction of the side that is considered as a wall. The rest is considered as a roof.

Moreover, if `attic` is set to `True`, a passive attic zone (no HVAC, glazing, or occupants) is defined: it constitutes a buffer zone between the last inhabited floor and the attic: the last floor composition is `lowup_floor_composition`.



attic: False  
 extrusion: 0  
 wall\_ratio: 0.0



attic: True (not visible on the 3D plot)  
 extrusion: 1  
 wall\_ratio: 0.4

In the building form, 2 kinds of roof shapes can be defined: the aperiodic roofs (periodic=False) and the periodic roofs (periodic=True). Defaults keep a flat ceiling:

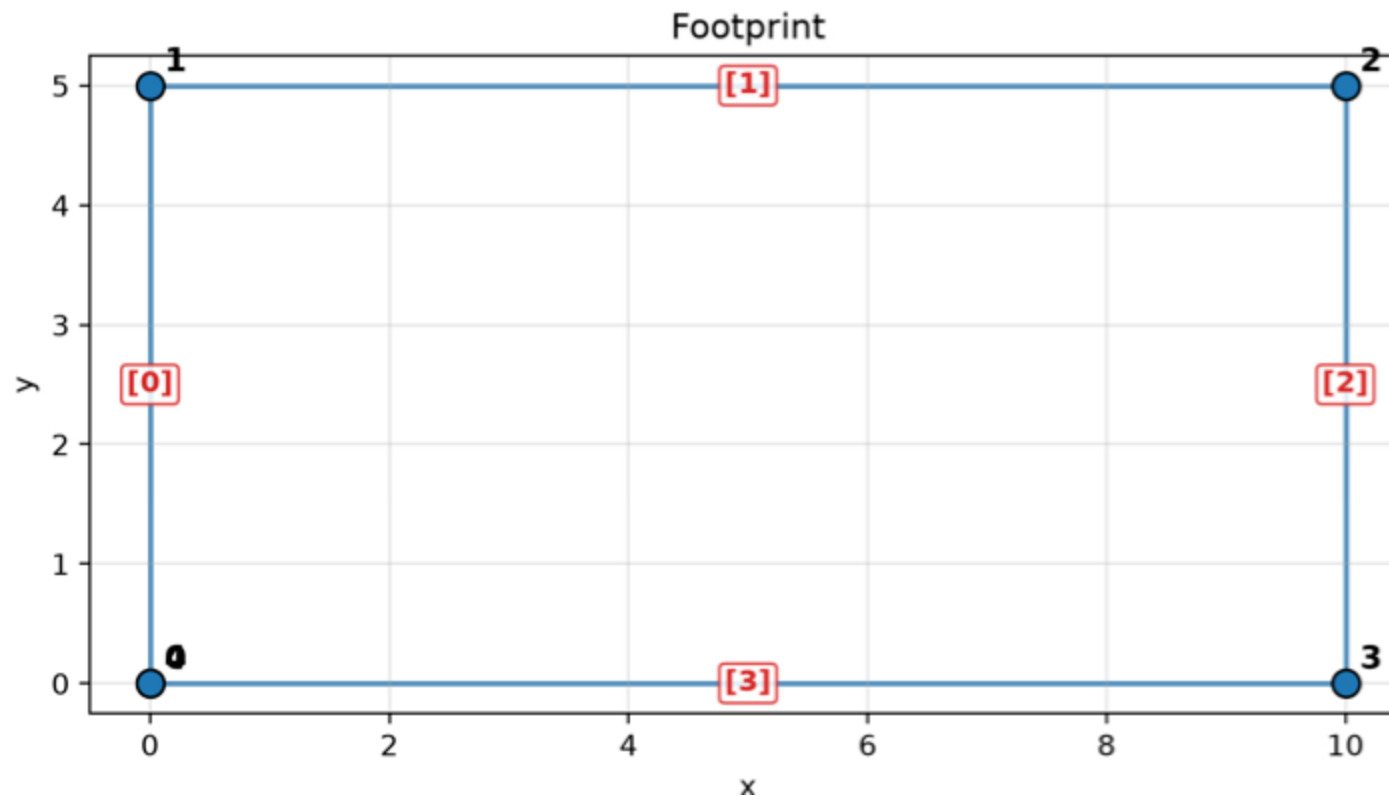
`"roof_points": []` and do not depend on the kind of roof shape.

### Aperiodic roof is defined by:

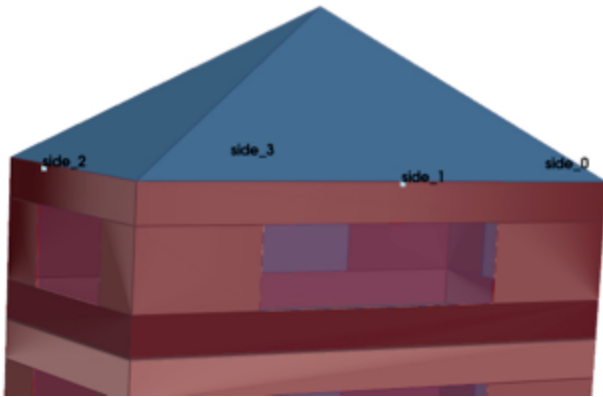
- `roof_points` — ridge / apex `(x, y, z)` in meters (same axes as the footprint in meters) define a convex hull on the roof surface where `z=0` stands for the top of the building without roof but keeping the footprint.
- `roof_extrusion` — height of the vertical roof base above the top storey [m]
- `roof_wall_ratio` — lower fraction of each vertical roof side counted as **wall** (0..1); the rest is **roof**
- `attic` — if true, a passive attic zone (no HVAC, glazing, or occupants)

Check the shape with `building 3d` .

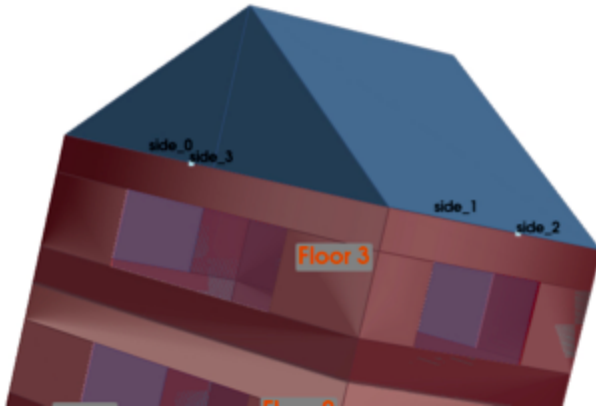
For example, the following roofs are defined for a building whose footprint is `[0,0,0]` , `[10,0,0]` , `[10,5,0]` , `[0,5,0]` .



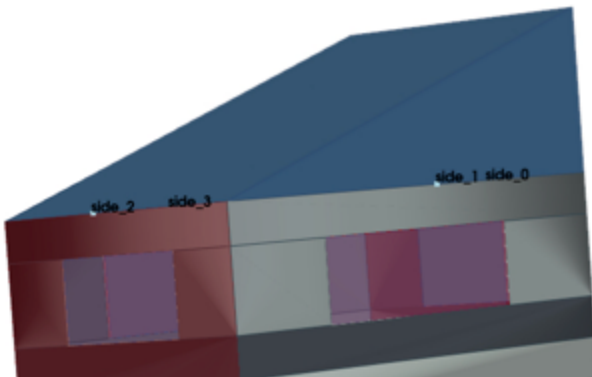
A pointed roof can be defined by one point: `[5, 2.5, 3]` .



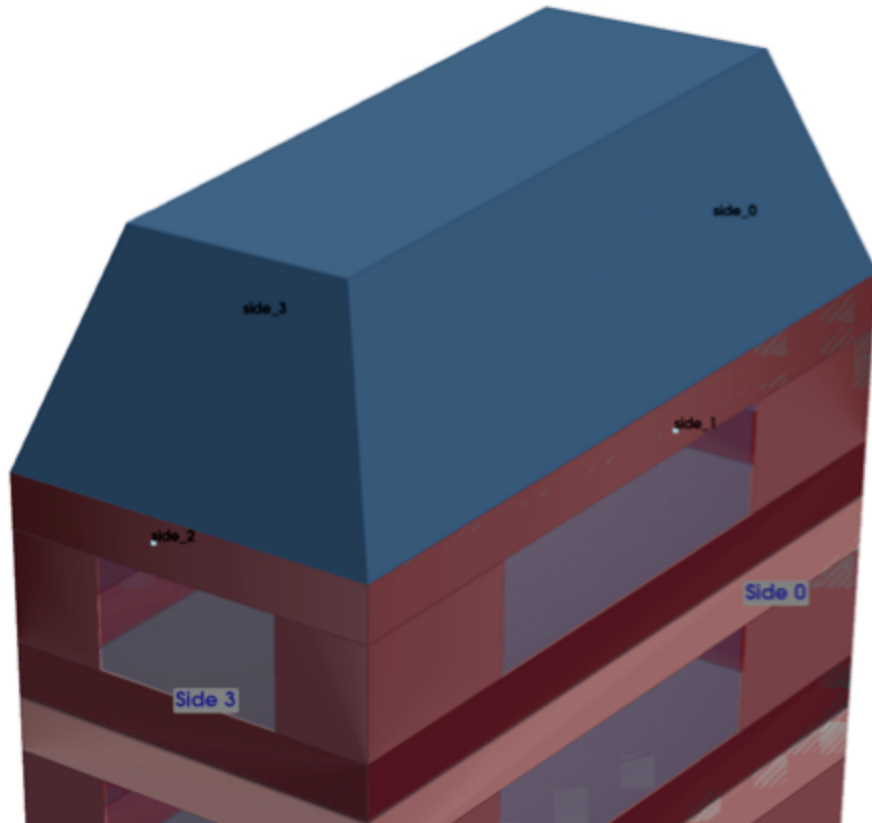
A hip roof can be defined by two points:  $[5, 0, 3]$ ,  $[5, 5, 3]$  .



Another hip roof:  $[0, 0, 3]$ ,  $[0, 5, 3]$  :



A ridge roof can be defined by 4 points:  $[1,1,3]$ ,  $[9,1,3]$ ,  $[9,4,3]$ ,  $[1,4,3]$  .



## Periodic roof shapes

Set `periodic: True` to build a repeating height profile instead of a convex hull.

In this mode, `roof_points` are **not** `(x, y, z)` apexes. They are breakpoints of **one period**:

```
(fraction, z)
```

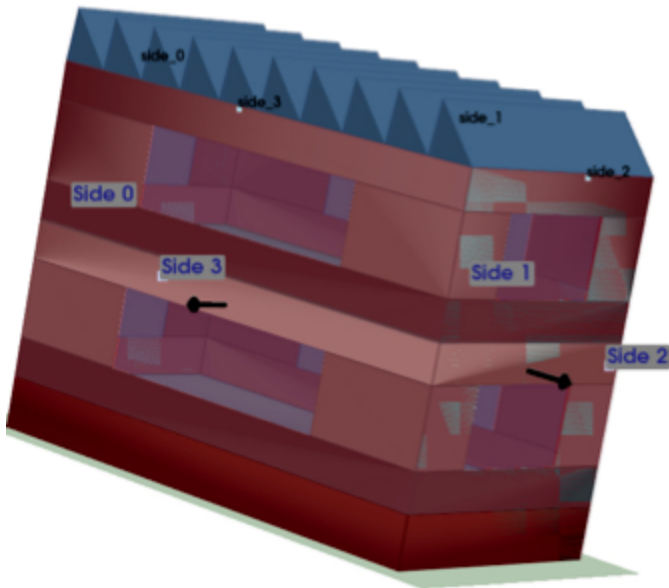
- `fraction` — position along one period ( `0` = start, `1` = end of the period)
- `z` — roof height above the top of the extrusion [m]

That profile is repeated `periodic_n_repeats` times along `periodic_direction` ( `"x"`, `"y"`, or a 2D vector `[dx, dy]` ).

`roof_extrusion`, `roof_wall_ratio` and `attic` work as for aperiodic roofs.

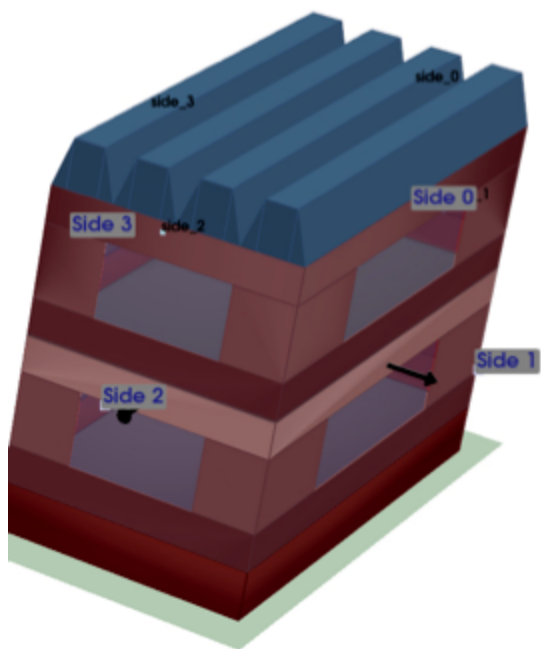
Saw-tooth / shed periods (10 repeats along x ):

```
"periodic": true,  
"periodic_n_repeats": 10,  
"periodic_direction": "x",  
"roof_points": [[0, 0], [0.1, 1.0], [1, 0]],  
"roof_extrusion": 0
```



Trapezoid periods (4 repeats):

```
"periodic": true,  
"periodic_n_repeats": 4,  
"periodic_direction": "y",  
"roof_points": [[0, 0], [0.25, 1], [0.75, 1], [1, 0]]
```



Check with `building 3d` . Prefer this mode for industrial saw roofs or any shape that repeats along one axis.

## Optional features:

- attic: set `attic: true` in the roof fields; needs the intermediate floor composition `lowup_floor`
- basement: when `base_elevation > 0`, a `basement_ground` composition is required
- wall contacts: neighbouring heated volumes on a façade (force glazing to 0 on the contacting floors)
- solar protections: `max_unshaded_protections` per side; 90° means no solar protection on that axis and 45° means the sun is hidden when it passes this altitude

Typical design loop:

```
context load project  
building new  
building 3d  
building simulate --quick  
building results  
building edit  
building simulate  
building save project
```

| command              | description                                                 |
|----------------------|-------------------------------------------------------------|
| building             | show a summary of the current building                      |
| building new         | create a new building (GUI)                                 |
| building edit        | edit the current building (GUI)                             |
| building load <path> | load a <code>.building</code> file from <code>./data</code> |
| building save <path> | save the building to <code>./data</code>                    |
| building delete      | forget the building in memory (or delete a file)            |

| <b>command</b>     | <b>description</b>                                               |
|--------------------|------------------------------------------------------------------|
| building footprint | plan view with side indices                                      |
| building 3d        | 3D view of the building                                          |
| building heliodons | sun paths seen from central glazing of building façades          |
| building simulate  | hourly thermal simulation over the year specified in the context |
| building results   | HVAC totals + comfort indicators                                 |
| building vars      | list building-related variables                                  |
| building plot      | plot any building variables                                      |

Parametric design is done with `building variation ...`.

It sweeps one design parameter (or a full set) and opens figures with:

- left: HVAC energy (kWh/year)
- right: comfort percentiles on a reference floor

Examples:

```
building variation
building variation heat_recovery
building variation glazing_ratio side 0
building variation solar_protection side 0
```

Scalars include heat recovery, air renewal, solar factor, setpoints, rotation, ...

Figures are also saved under `.baterm/variation_figures/`.

Full syntax: `help building variation`.

| command                            | description                      |
|------------------------------------|----------------------------------|
| building variation                 | full parametric sweep            |
| building variation <param>         | sweep one parameter              |
| building variation ...             | limit to a few cases (fast)      |
| material list thermal              | browse thermal materials         |
| material list glazing              | browse glazing materials         |
| material thermal load <name> <row> | load a material into the session |

## The pv (for photovoltaic) commands

A photovoltaic plant is defined by its orientation, tilt, mounting type, module efficiency and sizing.

It also needs a **context** (site + weather). If no context is loaded, `pv new` opens the context editor first.

A plant can be created with `pv new`, then modified with `pv edit`.

It is saved as a `.pv` JSON file under `./data` (for instance `pv save roof` → `./data/roof.pv`).

The plant embeds a copy of the context used at creation time. When you already have a session context, baterra keeps that period so building and PV stay on the same horizon.

## Orientation conventions:

- `exposure_deg` : **0 = South**, +90 = West, -90 = East, 180 = North
- `slope_deg` : **0 = facing the ground**, 90 = vertical, 180 = facing the sky  
(a classic 35° tilt facing the sky corresponds to `slope_deg = 145` )

## Sizing: give **exactly one** of:

- `number_of_panels`
- `peak_power_kw`
- `ground_surface_m2`

Mounting type ( `flat` , `saw` , `arrow` , ...) changes self-shading between rows. See `slides01-sun.pdf` for more details.

## Typical PV loop:

```
context load project
pv new
pv 3d
pv best angles
pv simulate --quick
pv simulate
pv save project
```

`pv best angles` searches exposure/slope for maximum annual DC production of the current plant.

`pv simulate` writes collector powers and the total `PV:power_W` on the shared DataProvider.

| command        | description                                           |
|----------------|-------------------------------------------------------|
| pv             | show a summary of the current PV plant                |
| pv new         | create a new plant (GUI; needs context)               |
| pv edit        | edit the current plant (GUI)                          |
| pv load <path> | load a <code>.pv</code> file from <code>./data</code> |
| pv save <path> | save the plant to <code>./data</code>                 |
| pv delete      | forget the plant in memory (or delete a file)         |
| pv dir         | list <code>.pv</code> files in <code>./data</code>    |

| <b>command</b>        | <b>description</b>                     |
|-----------------------|----------------------------------------|
| pv 3d                 | 3D layout of the arrays                |
| pv heliodon           | sun path for the plant site            |
| pv best angles        | best exposure / slope for annual yield |
| pv simulate [--quick] | hourly PV production                   |
| pv results            | annual totals + building–PV indicators |
| pv vars               | list PV-related variables              |
| pv plot               | plot PV variables                      |

After both `building simulate` and `pv simulate`, `pv results` prints indicators such as:

- self-consumption
- self-sufficiency (self-production)
- dependency
- year autonomy
- NEEG (net energy exchange with the grid)
- autonomous day ratio

These indicators need the building electricity demand `PELEC:building#sim` and the PV production `PV:power_W`.

# The global commands

Global commands join the **building** and the **PV plant** on the same DataProvider.

They answer questions such as:

- when is production larger than demand?
- what is the annual electricity bill with a given tariff?
- how to export all series for further analysis?

```
flowchart LR
    B[building simulate] --> D[DataProvider]
    P[pv simulate] --> D
    D --> E[enercost]
    D --> F[balance]
    D --> X[export]
```

## balance

`balance` draws a **day × month heatmap** of the daily net balance:

**balance = PV production – building electricity demand**

- **green**: surplus (PV covers the day)
- **red**: deficit (grid import needed)

Needs both simulations first:

```
building simulate  
pv simulate  
balance
```

## enercost

enercost computes the annual electricity cost from:

- `PELEC:building#sim` (building demand)
- `PV:power_W` (PV production)

Default tariff is French **HP/HC** (off-peak roughly 22:00–06:00).

It writes series such as `ENERCOST:hour#sim`, `GRID_IMPORT:kWh#sim`,  
`GRID_EXPORT:kWh#sim`.

```
enercost
enercost hp_hc
enercost base --base 0.25 --export 0.12
```

Typical joined workflow:

```
context load mysite  
building load myhouse  
building simulate  
pv load roof  
pv simulate  
pv results  
balance  
enercost  
export my_study
```

`export <name>` writes all DataProvider series to `./<name>.xlsx` in the working directory.

| <b>command</b>                        | <b>description</b>                        |
|---------------------------------------|-------------------------------------------|
| balance                               | day×month heatmap PV – PELEC              |
| enercost                              | annual electricity bill (default HP/HC)   |
| enercost hp_hc                        | French peak / off-peak tariff             |
| enercost base --base ... --export ... | flat buy / sell prices (€/kWh)            |
| export <name>                         | export all series to Excel                |
| status                                | show loaded context / building / PV       |
| clear                                 | forget context, building and PV in memory |
| vars                                  | list variables (session DataProvider)     |
| plot                                  | plot any session variables                |

## Wrap-up

Remember the dependency chain:

1. **context** (site + weather + masks)
2. **building** and/or **pv** (both need the context)
3. **simulate** each side
4. **global** tools: `balance`, `enercost`, `export`

Use `help <command>` at any time, and `manual` to open the PDF user manual.