DMarlon Studio

User manual

# DMarlon Studio manual

The complete guide to the visual IDE for building Python apps

Version 1.0.19 · 2026-11-05

dmarlonstudio.com

# Contents


1 **Introduction and installation**
What DMarlon Studio is · Requirements · Installation · Where everything is stored

2 **Getting started**
The main window · Theme and language · Your first window in five steps

3 **The Designer**
Projects · The Designer screen · Components · Moving, aligning and editing · Window position · Events and code · Generated Python and export · Run, stop and see the output · Included examples · Data connections · HTML view: real websites

4 **AI in the Designer**
Chat with Claude Code · Ask the AI · AI budget · GitHub

5 **The Editor**
Opening a folder · Explorer and tabs · Find and replace · Terminal · Source control (git) · Claude: implement · AI chat

6 **Databases**
Connections · Tables and queries

7 **API client**
Requests · Collections, variables and history

8 **FTP**
Servers and files

9 **Diagrams**
Flowcharts and database diagrams

10 **Settings**
Connect to Claude (MCP) · Artificial intelligence · Language, appearance and version

11 **Keyboard shortcuts**
Designer · Editor

12 **Troubleshooting**
Python not found, or no tkinter · Flask is missing · The app won't open (unsigned) · AttributeError: 'Form1' object has no attribute 'root' · Claude Code was not found

13 **Roadmap**
Targets · Richer projects

Chapter 1

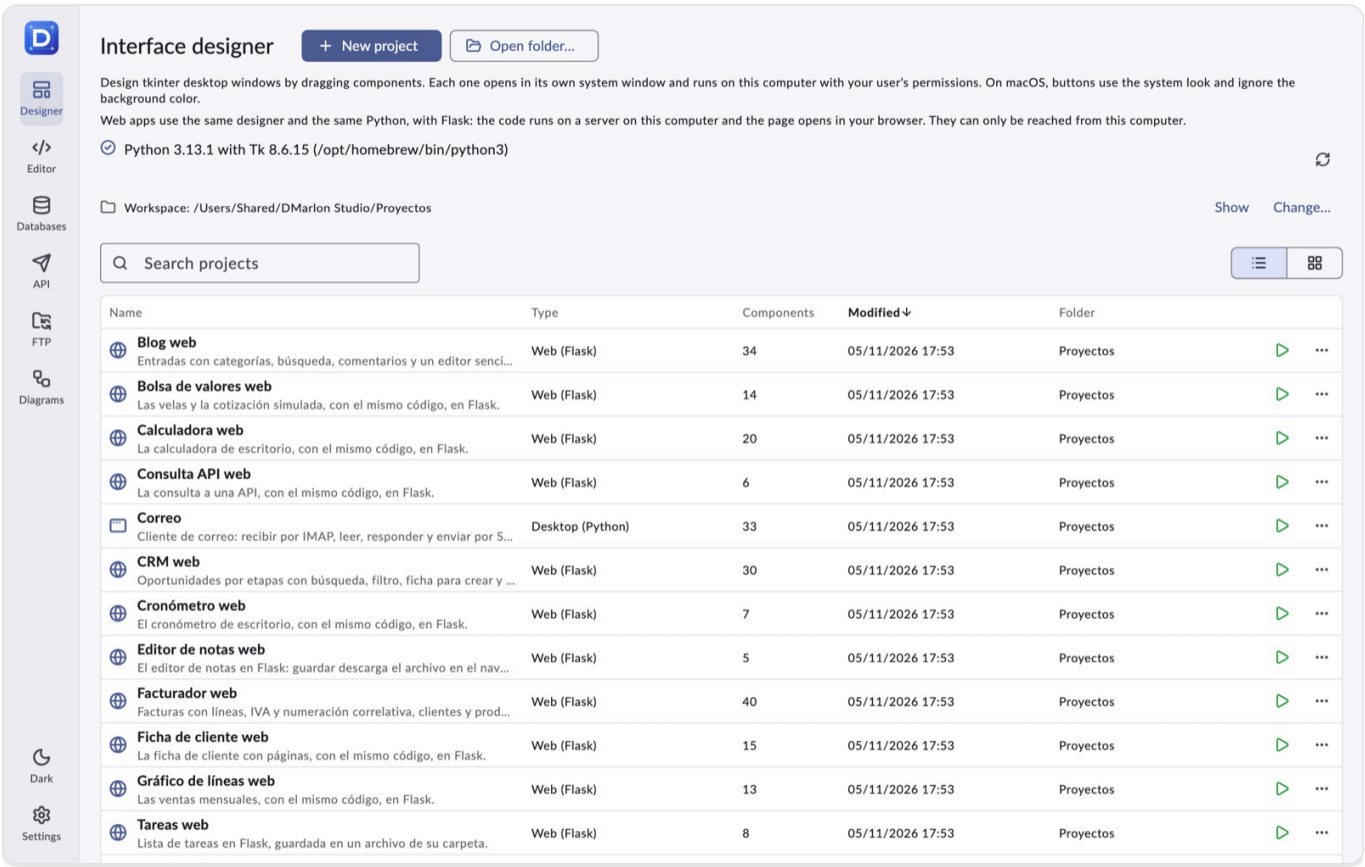
# Introduction and installation

What DMarlon Studio is, what it needs and where it keeps everything.

## What DMarlon Studio is

DMarlon Studio is a visual desktop IDE. You design windows by dragging components onto a form, adjust their properties in an inspector and write the code of each event in Python. **F9** runs the window on your computer.

Today the same designer builds two kinds of app: desktop windows with **tkinter** and web apps with **Flask**. Around the designer there is a VS Code-style code editor with git and a terminal, a database manager, an API client, an FTP client, a diagram editor and AI assistants (Claude by default).




The Designer: palette on top, tree and inspector on the left and the form in the center.

## Requirements

| To... | You need |
|---|---|
| Use the studio | macOS 10.15 or later, or 64-bit Windows 10/11. |

| To… | You need |
|---|---|
| Run desktop windows | Python 3 with tkinter. On macOS: `brew install python-tk` or the python.org installer. On Windows: the python.org installer with the **tcl/tk and IDLE** option ticked. |
| Run web apps | Flask in that same Python: `python3 -m pip install flask` (on Windows, `py -3 -m pip install flask`). |
| Version control in the Editor | git installed on the computer. |
| Claude Code chat in the Designer | Claude Code installed and signed in (optional). |
| AI assistants | An API key for the provider chosen in **Settings** (optional). |

ⓘ The studio finds Python on its own: on the PATH, in the Homebrew folders and in the python.org install, and keeps the one with the newest Tk. The result is shown on the Designer's project list.

## Installation

Download the zip for your system from the downloads section of dmarlonstudio.com.

1. **macOS**: unzip `DMarlonStudio-macOS.zip` and move **DMarlon Studio** to the Applications folder. The first time, open the app with right-click → **Open**: it is not signed.
2. **Windows**: unzip the whole `DMarlonStudio-Windows.zip` wherever you like (not in Program Files) and run `DMarlonStudio.exe`. If “Windows protected your PC” appears, click **More info** → **Run anyway**.

## Where everything is stored

| What | Where |
|---|---|
| Designer projects | One folder per project in `~/Documents/DMarlon Studio/Proyectos` (it can be changed in the Designer). They can be versioned with git. |
| Settings, connections, API requests, history and diagrams | `studio.sqlite`, in the app's data folder: `~/Library/Application Support/com.dmarlonstudio.studio` on macOS and `%APPDATA%\DMarlon Studio\DMarlon Studio` on Windows. |
| Deleted projects | The `.papelera` folder inside the workspace: nothing is erased for good. |

Each project folder holds `project.dmsproj` (the design, as JSON), the window's code (`Form1.py`, named after the form), a `.gitignore` and, once run, `build/` with the generated Python.

Chapter 2

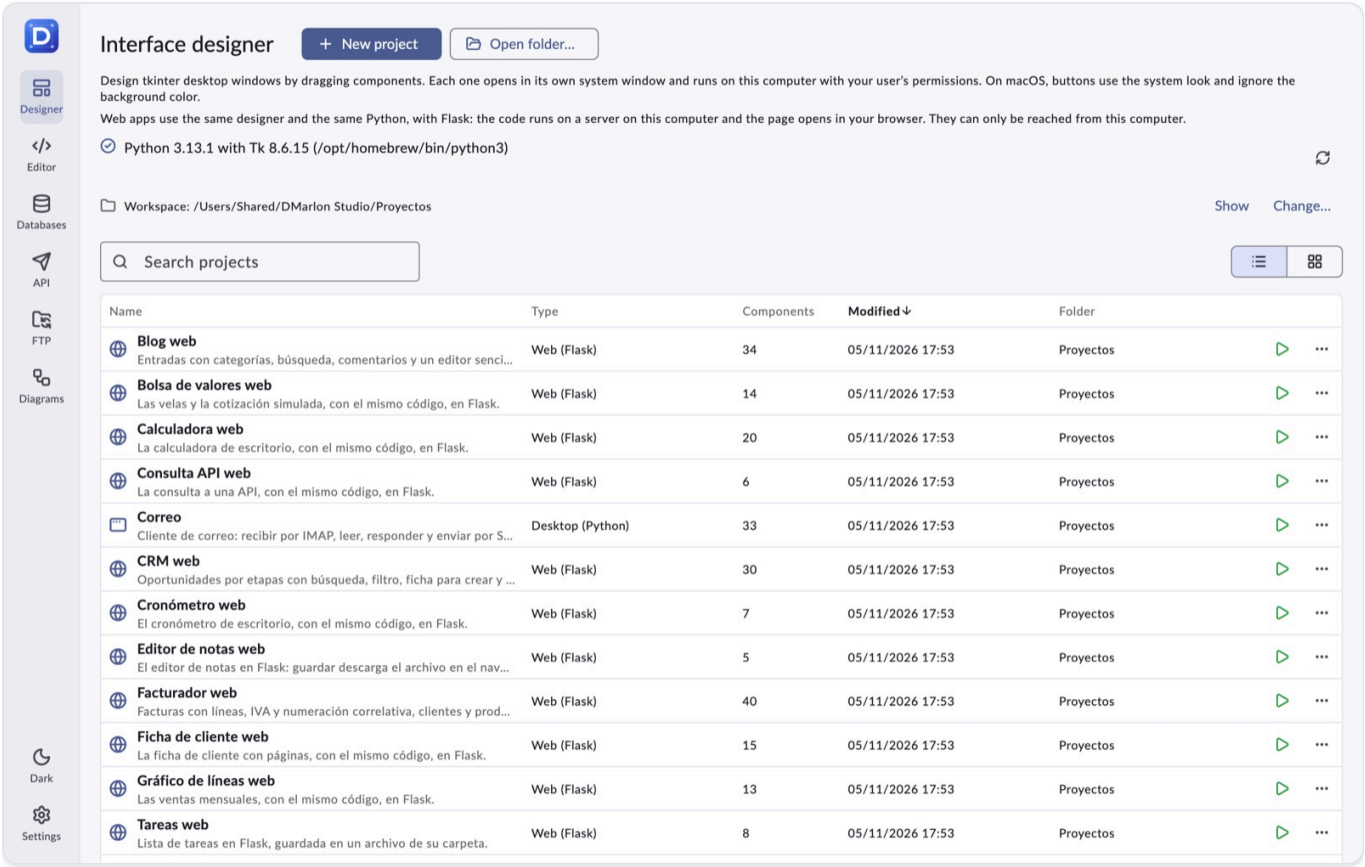
# Getting started

The main window, the modules, the theme and the language.

## The main window

On the left is the module bar: **Designer**, **Editor**, **Databases**, **API**, **FTP**, **Diagrams** and, at the bottom, the **Dark**/**Light** button and **Settings**.

Each module is created the first time you open it and keeps its state: you can go from the Designer to the Editor and back without losing anything unsaved.




The project list, with the examples that come with the studio.

## Theme and language

- The **Dark**/**Light** button on the bar switches the theme at once. In **Settings → Appearance** you can also choose **System** to follow your computer.
- **Settings → Language** switches between **Español** and **English** without closing anything. The first time, the system language is used.

## Your first window in five steps

1. In **Designer**, click **New project**, type a name and choose **Desktop (Python)**.
2. Drag a **Label**, a **Text field** and a **Button** from the palette onto the form.
3. Select the button and change its **Text** in the inspector, for example to “Greet”.
4. Double-click the button: the `Button1_click` method is created and the code opens. Type `self.Label1.config(text="Hello, " + self.Entry1.get())`.
5. Press **F9**. The window opens in its own system window.

Chapter 3

# The Designer

Projects, components, inspector, events, generated code and running.

## Projects

The project list shows the workspace projects and those opened from other folders. At the top you see the Python found (Python and Tk versions) and the button to **Look for Python again**.

It is shown as a **list** with columns (**Name**, **Type**, **Components**, **Modified** and **Folder**) sorted by clicking their title, or as **Cards**. It has **Search projects** and is split into pages (**Per page**: 10, 15, 25 or 50). Double-click opens a project. Right-clicking a project offers **Open**, **Run**, **Show the folder**, **GitHub…**, **AI budget**, **Duplicate** and **Delete**; right-clicking the background, **Add new** › **Desktop project (Python)**, **Web project (Flask)** or **Open folder…**.

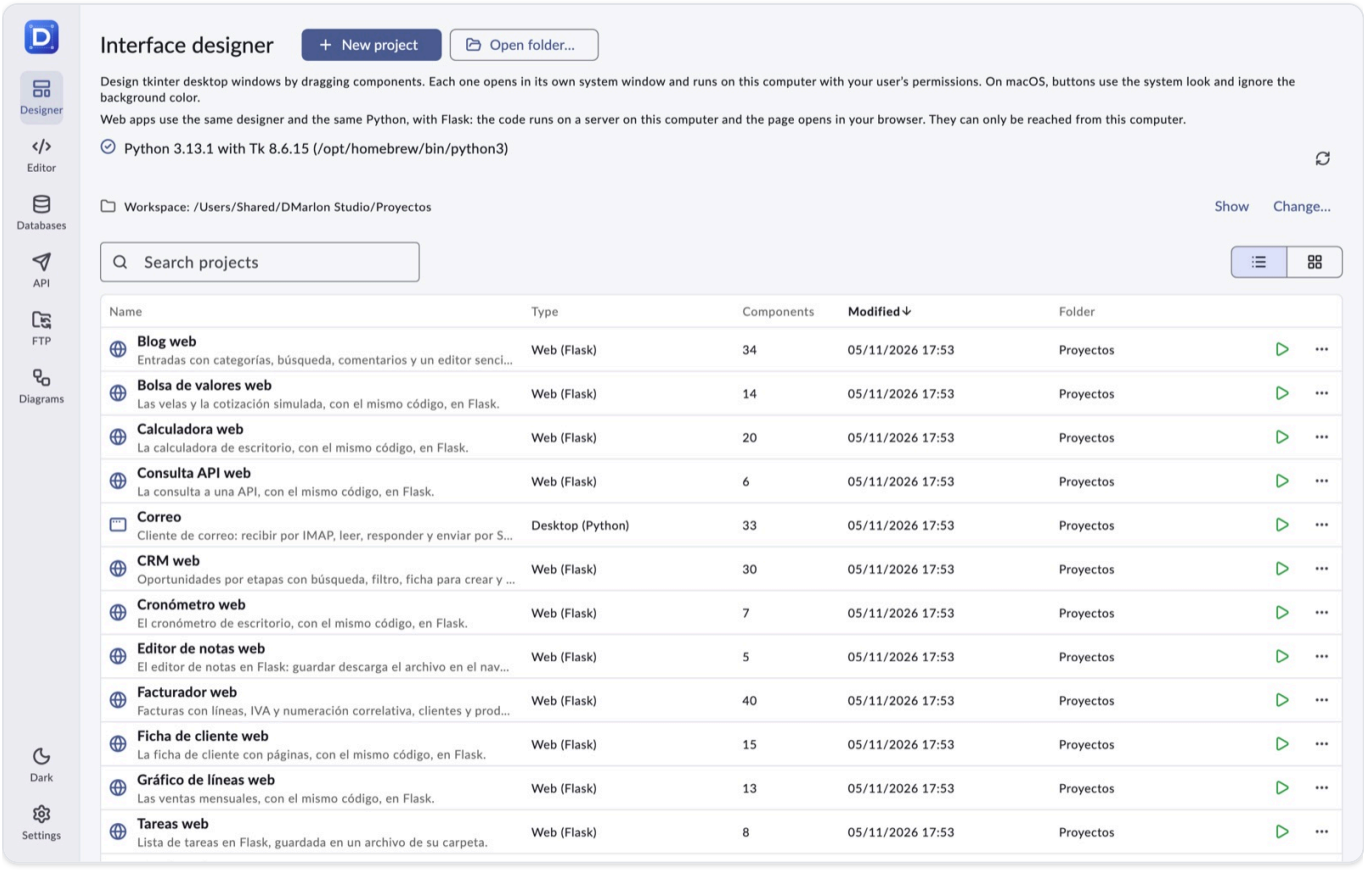
- **New project**: asks for the name and the target, **Desktop (Python)** or **Web (Flask)**. The target cannot be changed later.
- **Open folder…**: opens a project stored in any other folder; it stays on the list as a recent one.
- **Workspace**: **Show** opens it in Finder or Explorer and **Change…** picks another folder.
- On each card: **Run**, **Show the folder** and, under **More actions**, **GitHub…**, **AI budget**, **Duplicate** and **Delete** (moves the project to the workspace trash).
- **Tabs**: each open project has its tab at the top, with a dot when there are unsaved changes. Several can be open at once; the **Projects** tab goes back to the list without closing anything. Close them with the ✕ or a middle click; they ask whether to save.

> ⓘ The studio copies the example projects into the workspace (new ones also appear in existing workspaces). See the full list in **Included examples**.

## The Designer screen

- **Toolbar**: back to the projects, application name (click to rename it), **Save**, **Undo**, **Run (F9)** in green, **Stop** in red, **Open in browser** (web), **Show the project folder**, **Export the .py file**, **GitHub**, **AI budget**, **Ask the AI**, **Chat with Claude Code**, alignment and centering, **Snap to grid** and **Show output**.
- **Output console**: at the bottom; drag its top edge to resize it, minimize it with the arrow, and **Copy the output** copies every message.
- **Palette**: a bar of icons; hovering shows the component's name. Drag it onto the form or add it with a double click.
- **Tree**: the window and its components, with those inside a frame or a page indented.

- **Inspector**: **Properties** and **Events** of the selected component or, if none, of the window.
- **Center**: **Design**, **Code** and **Generated Python** tabs. **F12** switches between design and code.




The example calculator in the Designer.

## Components

Every component has a **Name**, a position and a size. Almost all accept **Background color**, **Text color** and font (**Font**, **Font size**, **Bold**, **Italic**), and the mouse events (**Click**, **Double click**, **Right click**, **Mouse enter**, **Mouse leave**). Typing components add the keyboard ones (**Focus in**, **Focus out**, **Key down**, **Key up**, **Enter key**).

| Component | Own properties | Main events |
|---|---|---|
| **Label** (`Label`) | Text, Alignment, Justification, Relief, Line wrap | Mouse |
| **Button** (`Button`) | Text, State, Relief | **On press** (`Button1_click`) |
| **Text field** (`Entry`) | Initial text, Justification, Mask with (e.g. `*` for passwords), State, Linked table and Linked field | Keyboard and mouse |
| **Password** (`Password`) | Placeholder text, Button to show the password, Maximum length, State | Keyboard and mouse |

| Component | Own properties | Main events |
|---|---|---|
| **Image** ( `Image` ) | File (in the project's `images` folder), Fit (original size, whole without distortion, fill and crop, stretch), Border | Mouse |
| **Long text** ( `Text` ) | Initial text (several lines), Line wrap, State | Keyboard and mouse |
| **Checkbox** ( `Checkbutton` ) | Text, State, Checked | **On press** |
| **Option** ( `Radiobutton` ) | Text, State, Group, Value, Checked | **On press** |
| **List** ( `Listbox` ) | Items (one per line), Selection (browse, single, multiple, extended) | **On select** |
| **Drop-down list** ( `Combobox` ) | Items (one per line), Initial text, Justification, State (normal, readonly, disabled) | **On select** |
| **Slider** ( `Scale` ) | Title, From, Until, Step, Value, Orientation, Show value | **On change** (receives the value) |
| **Frame** ( `Frame` ) | Relief, Border. Holds other components | Mouse |
| **Canvas** ( `Canvas` ) | Background color. For drawing from code | Mouse, **Drag**, **Button release**, **Mouse move** |
| **Data table** ( `Treeview` ) | Data source, SQL query, Columns (field\|title\|width), Limit, Sequence, Load on open | **On select** |
| **Progress bar** ( `Progressbar` ) | Value, Maximum, Mode (determinate or indeterminate), Orientation | Mouse |
| **Page control** ( `PageControl` ) | Pages (one per line), Active page. Each page holds its own components | **On page change** |
| **Status bar** ( `StatusBar` ) | Text. Always takes the bottom edge of the window | Mouse |
| **Ribbon** ( `Ribbon` ) | Tabs, groups and buttons (indented; “Button \| icon”), Active tab | **On button press**, **On tab change** |
| **Menu** ( `Menu` ) | Options (one per line; indent = submenu; `-` = separator). Non-visual: shown at the top of the window | **On choosing an option** (receives the option) |
| **Timer** ( `Timer` ) | Interval (ms), Active. Non-visual | **Every interval** |

| Component | Own properties | Main events |
|---|---|---|
| **API integration** (`RestClient`) | Base URL, Authentication (none, Bearer token, basic or a key in a header; the secret, preferably in an environment variable), Headers, Timeout. Non-visual | **When a call finishes**, **If a call fails** |
| **Tray icon** (`TrayIcon`) | Hover text, Image, Menu (one option per line). Non-visual | **On icon click**, **On choosing a menu option** |
| **Open dialog** (`OpenDialog`) | Title, Types ("Text\|*.txt"), Default extension, Initial folder, Choose several files. Non-visual | **On choosing a file** |
| **Save dialog** (`SaveDialog`) | Title, Types, Default extension, Initial folder, File name, Ask before overwriting. Non-visual | **On choosing a file** |
| **AI assistant** (`AiAssistant`) | Claude model, Instructions, Effort, Maximum answer length. Non-visual. From code: `ask`, `chat` (with memory; `reset` starts another), `summarize`, `translate`, `classify` (one label from a list), `extract` (data into a dictionary) and their `*_async` variants, which don't freeze the window. Uses Anthropic's official SDK (`pip install anthropic`); when run from the studio it gets the key from **Settings** through the `ANTHROPIC_API_KEY` variable | **When the answer arrives**, **If the request fails** |
| **PDF report** (`PdfReport`) | Document title, File name, Orientation, Author, Page numbers. Non-visual. From code: `title`, `heading`, `text`, `table` (with totals and the header repeated on every page), `image` (JPEG), `line`, `page_break`; then `save`, `save_dialog`, `open` or `download` (web). Nothing to install | — |
| **Excel report** (`ExcelReport`) | File name, First sheet. Non-visual. From code: `sheet`, `header`, `write_row`, `write_rows` (text, numbers, dates and booleans with their type; frozen header with a filter); then `save`, `save_dialog`, `open` or `download` (web) | — |
| **Chart** (`Chart`) | Kind (line, area, bar or candles), Title, Axis titles, Legend, Grid, Decimals, Value prefix and suffix, Maximum points. From code: `set_series`, `add_point`, `set_candles`, `add_candle`, `clear` | **On clicking a point** |
| **Data connection** (`Connection`) | Connection (one of those saved in **Databases**). Non-visual | — |

Menu, Progress bar, Status bar, Timer and Page control arrive with version 1.0.5; Password, Image, Ribbon, API integration, Tray icon and the Open and Save dialogs, with 1.0.6; the Chart, with 1.0.8. In a web app, components without an equivalent say so in the inspector.

## Moving, aligning and editing

- Click selects; dragging moves; the handles resize. With **Snap to grid** on, everything fits the grid.
- The arrow keys move the selection one grid step; with Alt (Option), one pixel; with Shift, they resize.
- With two or more components selected, **Align left**, **Center horizontally**, **Align right**, **Align top**, **Center vertically**, **Align bottom**, **Same width** and **Same height** become available (relative to the first one selected).
- With one or more selected, **Center in the window (horizontally)** and **Center in the window (vertically)** center them as a block in the window or in their frame.
- Right-click offers **Cut**, **Copy**, **Paste**, **Delete**, **Bring to front** and **Send to back**.
- **Copy between projects**: copied items go to the system clipboard, with their children and **the code of their events**. Paste them into another project (or after restarting the studio): components get free names and their methods are renamed with them, never overwriting an existing one. Helper methods called by those events must be copied by hand.
- **Undo** (⌘Z / Ctrl+Z) reverts each design change. If the code was edited by hand after a step, that step restores the design but not the code, and you are told so.

## Window position

The window's **Position** property decides where it opens: **Centered on screen** (new projects), **Default** (the system decides) or **Manual**, with **Left** and **Top**.

## Events and code

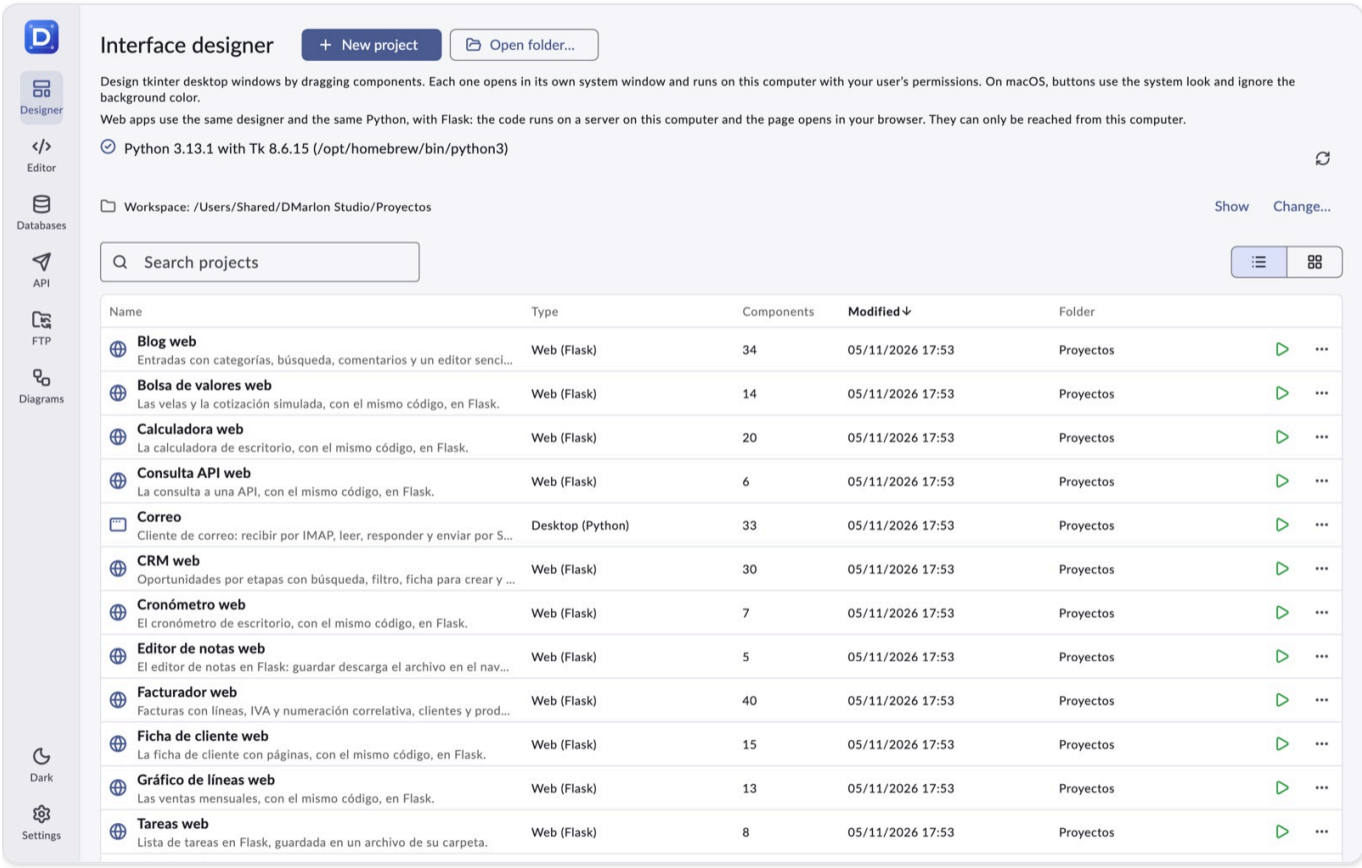
The window's code is a Python class named after the form that inherits from `FormDesign`, the class the studio generates. `self.root` is the window and each component is `self.<Name>`.

- Double-clicking a component opens its main event (`Button1_click`, `Listbox1_select`…) and creates it if needed.
- On the inspector's **Events** tab you type or choose the method for each event; **Go to code** jumps to it.
- Renaming a component renames its default methods and the references in the code.
- The window has its own events: **On open** (`form_create`), **On close**, **Key down** and **On resize**, and its properties **Title**, **Background color**, **Resizable** and **Always on top**.

```
class Form1(FormDesign):
    def form_create(self):
        self.total = 0
        self.StatusBar1.config(text="Ready")

    def Button1_click(self):
        self.total += 1
        self.Label1.config(text=f"Clicked {self.total} times")
```

> ⚠ Put what should happen at startup in the **On open** event (`form_create`). If you define your own `__init__`, call `super().__init__()` first.

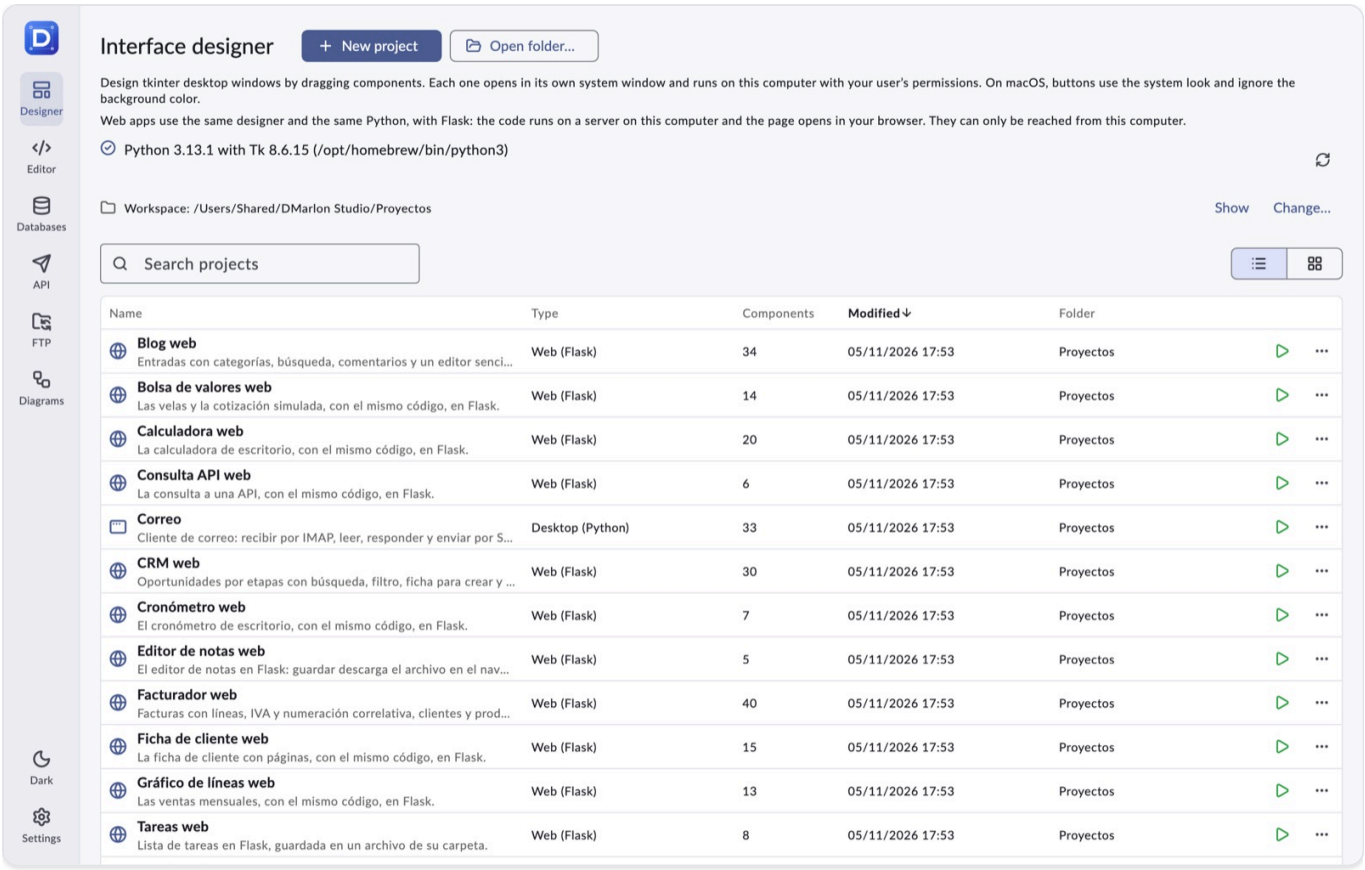



The Code tab: the window class with its handlers.

## Generated Python and export

The **Generated Python** tab shows, read-only, the complete file that runs: the design turned into code plus your class. When you run, it is written to `build/<project>.py` inside the project folder.

**Export the .py file** saves that file wherever you want. It works with any Python 3 with tkinter (or Flask, for web apps) without the studio.




The generated Python: not edited, rebuilt from the design and the code.

## Run, stop and see the output

- **Run (F9)** saves the project and launches the window in its own system window, with your user's permissions.
- **Stop** closes the running window or server.
- **Show output** opens the panel with what the app writes (`print`, Python errors) and the Python used.
- For a web app, the studio starts the Flask server on this computer and opens the browser; **Open in browser** opens it again. It can only be reached from this computer.
- If a Python package is missing (Flask, the Anthropic SDK, a database driver or pystray), the studio says so before running and offers **Install**: it installs it with the pip of the same Python that runs the project and then carries on. You can also **Copy the command** to do it by hand.

## Included examples

They all keep their data in the project folder and work without any setup. Those marked (web) also exist as a Flask app.

| Example | What it shows |
|---|---|
| **Calculator** (web) | Buttons, events and state |
| **Unit converter** (web) | Drop-down lists and computing while typing |
| **Tasks** (web) | A list with adding, removing and saving |
| **Stopwatch** (web) | Timer, menu, progress and status bars |
| **Customer form** (web) | Page control, lists and validation |
| **Notes editor** (web) | Ribbon and the Open and Save dialogs |
| **API query** (web) | Background API integration and a data table |
| **Notepad** | Classic menu, open, save, undo and a status bar with line and column |
| **Diary** | Contacts and appointments in SQLite, search and pages |
| **Currency converter** | Real exchange rates (ECB) with API integration and an offline fallback |
| **The snake** | A game on a canvas with a timer and a saved high score |
| **Music player** | Playlist, volume and progress (best with `pygame`) |
| **Monthly sales** (web) | Line, area or bar chart and a live mode |
| **Stock market** (web) | Candles, moving average and volume with live simulated data |
| **Windows and dialogs** | Messages, questions, asking for data, another window that returns a result, a modal dialog and non-modal windows |
| **Text assistant** (web) | Claude to summarize, translate, classify and extract data, and a chat with memory |
| **Reports** (web) | A sales table exported to PDF and Excel |
| **3D viewer** | 3D shapes drawn on a canvas: mouse rotation, animation, lighting and wireframe mode |
| **Email** | Receive over IMAP and send over SMTP with attachments, without freezing the window |
| **CRM web**, **Invoicing web** | Complete web apps with SQLite |
| **Web blog** | A real website made with the HTML view: home page, articles, comments, newsletter, accounts with login and sign-up, and a dashboard |
| **Web shop** | Electronics shop: deals, cart, accounts, simulated payment, “My account” and a sales and stock dashboard |

| Example | What it shows |
| --- | --- |
| **Odoo / ERPSqlite contacts** (web) | Connecting to an Odoo over XML-RPC |

## Data connections

1. Save the database in the **Databases** module (SQLite, PostgreSQL or MySQL/MariaDB).
2. Add a **Data connection** to the form and choose that connection in its **Connection** property.
3. Add a **Data table**, choose the connection in **Data source** and write the **SQL query** and the **Columns** (`field|title|width`).
4. To edit the selected row, link a **Text field** with **Linked table** and **Linked field**.

From code, `self.Connection1.query(sql, params)` returns rows and `self.Connection1.execute(sql, params)` changes data (always with `?` parameters, never by joining text). `self.Treeview1_refresh()` reloads the table.

> Passwords are never written to the generated file: they reach the process in `DMSTUDIO_DB_…` environment variables. An exported file reads them from the environment.

## HTML view: real websites

The **HTML view** shows HTML written by your code. With **Full page (web)** switched on, the whole page is that view: no form frame, its own CSS, like a normal website. The **Web blog** and **Web shop** examples are made this way.

- `self.Site.set_html(html, top=True)` changes the content (`top` goes back to the top of the page).
- A link or button with `data-action="post" data-arg="7"` calls the **On clicking a link or sending a form** event with an `HtmlAction`: `action.name` is `post` and `action.arg` is `7`.
- A `<form data-action="search">` sends its fields: read them with `action.get("q")` or in `action.form`.
- On the desktop, the view shows simple HTML (headings, paragraphs, lists, links and forms) in a text box; CSS styles only apply on the web.

```
import html

class Form1(FormDesign):
    def form_create(self):
        self.Site.set_html('<h1>Hello</h1><a href="#" data-
action="greet">Greet</a>')

    def Site_action(self, action):
        if action.name == "greet":
            name = html.escape(action.get("name", "world"))
            self.Site.set_html("<h1>Hello, %s</h1>" % name, top=True)
```

⚠ The HTML view paints HTML as is: always escape what the visitor writes with `html.escape`. The page only runs the engine's own scripts, so a `<script>` inside the HTML does not run.

The **Web blog** and **Web shop** come with user accounts: **Sign up**, **Log in** and **Log out**. Passwords are stored with PBKDF2 and a salt, never in clear text, and after five failed attempts you have to wait a minute. The first account to sign up is the administrator and sees the **Dashboard**: in the blog, posts, comments and user roles; in the shop, sales by category, orders and stock. In the shop, every customer sees their orders in **My account**.

Chapter 4

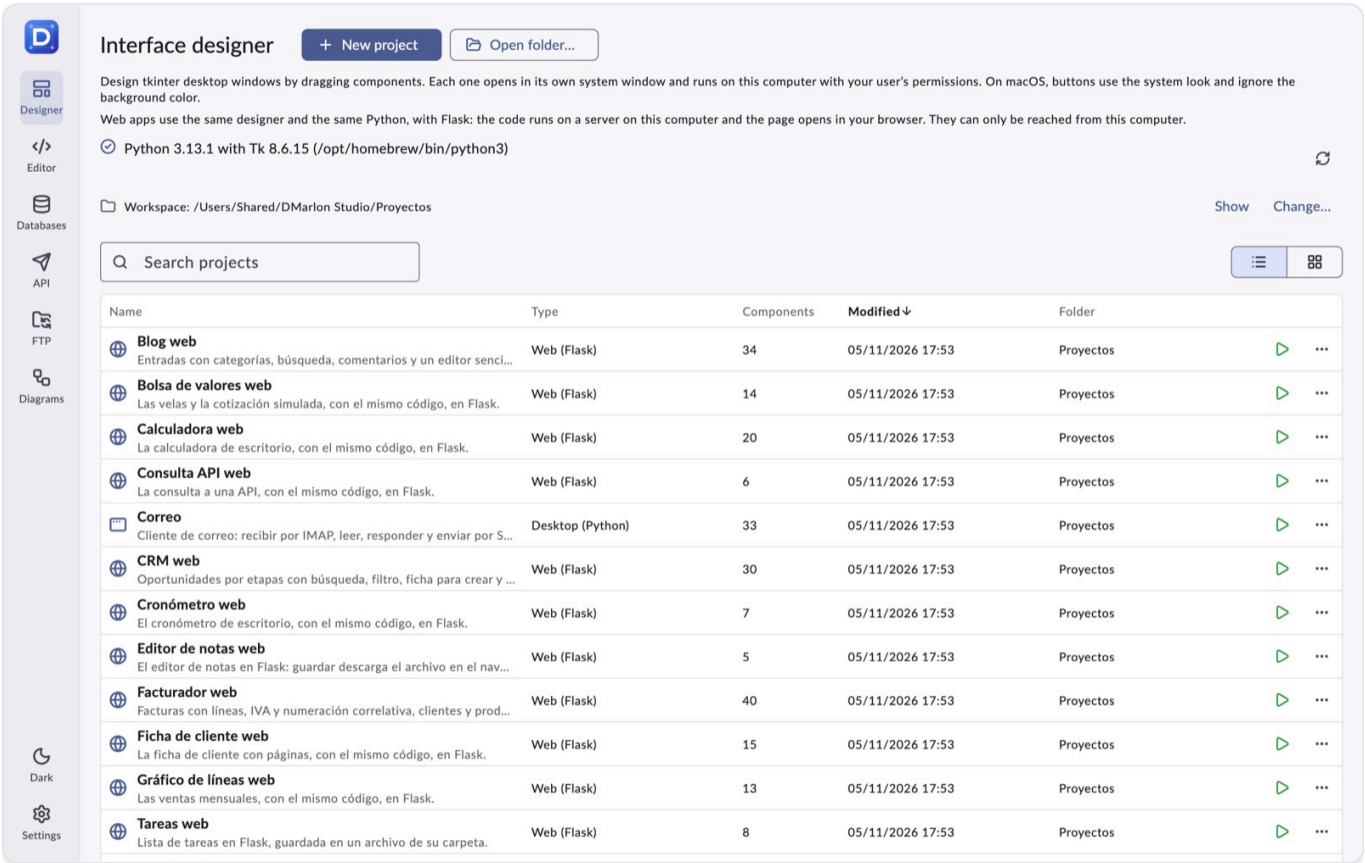
# AI in the Designer

Chat with Claude Code and “Ask the AI”.

## Chat with Claude Code

The **Chat with Claude Code** button (the robot on the toolbar) opens a panel on the right. Describe the change and press Enter (Shift+Enter adds a new line).

- It uses the Claude Code installed on your computer and your session: no API key needed.
- Before each message the studio saves the project. Claude Code reads and edits the design (`project.dmsproj`) and the window's code, without a terminal.
- When it finishes, the new design and code are loaded into the Designer as a single step that can be undone.
- The conversation continues between messages; the trash button starts a new one.
- The 📎 clip attaches images (a sketch, a screenshot) or text files: Claude Code reads them without them entering the project.




The Claude Code panel, to the right of the form.

## Ask the AI

**Ask the AI** uses the provider set in **Settings** (Claude by default) and has three modes:

- **Switch** changes the current window (“Add a phone field and align the buttons”).
- **New window** designs a window from scratch; you can **Attach a sketch** (an image).
- **Code of** a component: with a component selected, it writes the method for its main event.

The answer arrives as an **AI proposal**: you review it before applying it and, once applied, it is undone in one step.

## AI budget

The wallet button (**AI budget**) shows what the project has spent on Claude: by tool (Claude Code, Claude: implement, Ask the AI, chat), by model and by day, with the latest calls and their tokens.

- You can set a **Cap (US$)**, **Monthly** or **Total**. It warns at 80% and, with **Block when used up**, the AI stops answering in that project until you raise it.
- Costs use Anthropic API pricing. With a Claude subscription, Claude Code's cost is its API equivalent, not what you are charged.

## GitHub

- The Designer's **GitHub** button connects your account with a token (kept only on this computer).
- **Publish on GitHub** creates the repository (private or public), makes the first commit and pushes it.
- Once published: it lists the changes, asks for the **Commit message** and **Commit and push**; also **Pull changes** and **Open on GitHub**.

Chapter 5

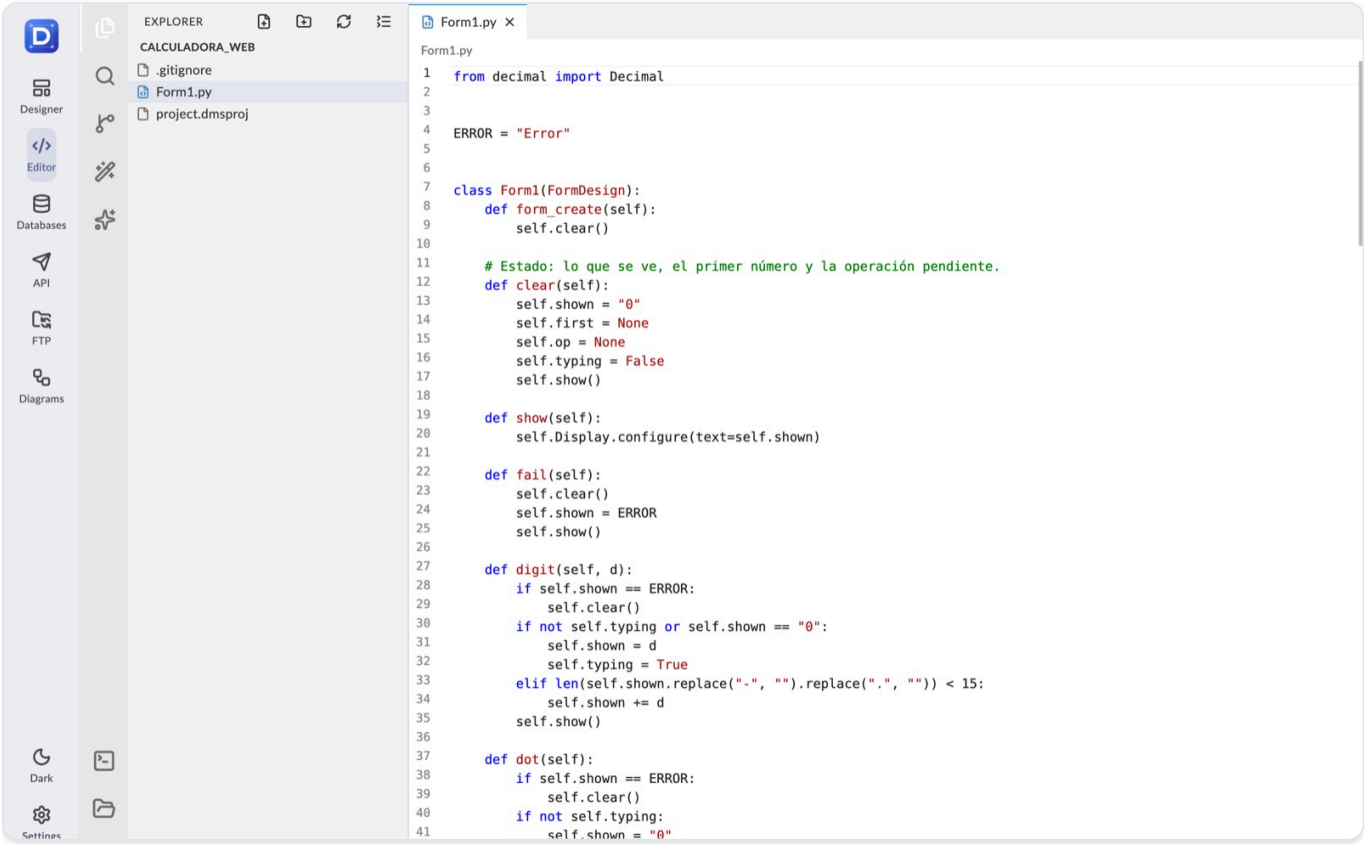
# The Editor

Code editor with explorer, search, terminal, git and AI.

## Opening a folder

With no folder open you get **Open folder…**, **Open DMarlon Studio** (the studio's documents folder), **Clone repository…** and the list of **Recent** folders.

If a project is open in the Designer, the Editor shows its code and its generated Python as **Project open in the Interface designer**: what you type here is the same text as in the Code tab.




The Editor: activity bar, explorer and tabs.

## Explorer and tabs

- The activity bar holds **Explorer**, **Search**, **Source control**, **Claude: implement** and **AI chat**, with the terminal and **Open folder** at the bottom.
- The explorer offers **New file**, **New folder**, **Rename**, **Delete**, **Copy path**, **Open in terminal** and **Collapse all**.

- Tabs have syntax highlighting for dozens of languages and line numbers. ⌘S / Ctrl+S saves; closing something unsaved asks first.
- Anything unsaved is kept when you switch modules.

## Find and replace

- **Find in file** with **Replace** and **Replace all**.
- The **Search** panel goes through every file in the folder, with **Match case** and **Regular expression**; a click opens the file at that line.

## Terminal

⌘J / Ctrl+J shows or hides the terminal, which runs commands in the open folder (`git status`, `python3 app.py`, `npm install` …). The up and down arrows go through the history and **Stop** ends the running command.

## Source control (git)

- **Changes**: modified files with their diff; type the **Commit message** and use **Commit all** or **Commit and push**. Also **Amend last commit** and **Undo last commit**.
- **Push**, **Pull** and **Check the remote for changes (fetch)**, with what is pending each way.
- **History** of the branch or of **All branches**: view the commit, **Revert this commit**, **Apply to the current branch (cherry-pick)**, create a branch or a tag from it and **Reset the branch to this commit…**.
- Local and remote **Branches**: create, switch, **Merge**, rename and delete (also on the remote).
- **Stash changes** to switch branches and bring them back later.
- **Remotes…**, **Initialize repository**, **Clone repository…** and **Publish to GitHub…** (public or private) with your **GitHub account** (a token with the “repo” scope).

## Claude: implement

**Claude: implement** asks Claude for a change through the Anthropic API (Claude Opus 5.5 by default). Claude explores the open folder (lists, reads and searches files) and proposes changes to one or more files.

- Nothing is written until you apply it: the **Proposed changes** list shows each file with its added and removed lines; a click opens the diff with **Apply** and **Discard**.

- **Apply all** writes every change; if a file changed on disk while Claude was working, you are warned instead of it being overwritten.
- It needs the Anthropic key in **Settings → Artificial intelligence**.
- The 📎 clip attaches images and text or code files to the request; the wallet button opens the folder's **AI budget**.

## AI chat

The **AI chat** answers about the open file with the provider from **Settings**. If it proposes the whole file, **Apply to the file** puts it in the tab (unsaved) and **Copy the code** copies it.

In all three chats Enter sends, Shift+Enter adds a line break and the 📎 clip attaches images (PNG, JPG, GIF or WebP up to 5 MB) and text files (up to 300 KB).

Chapter 6

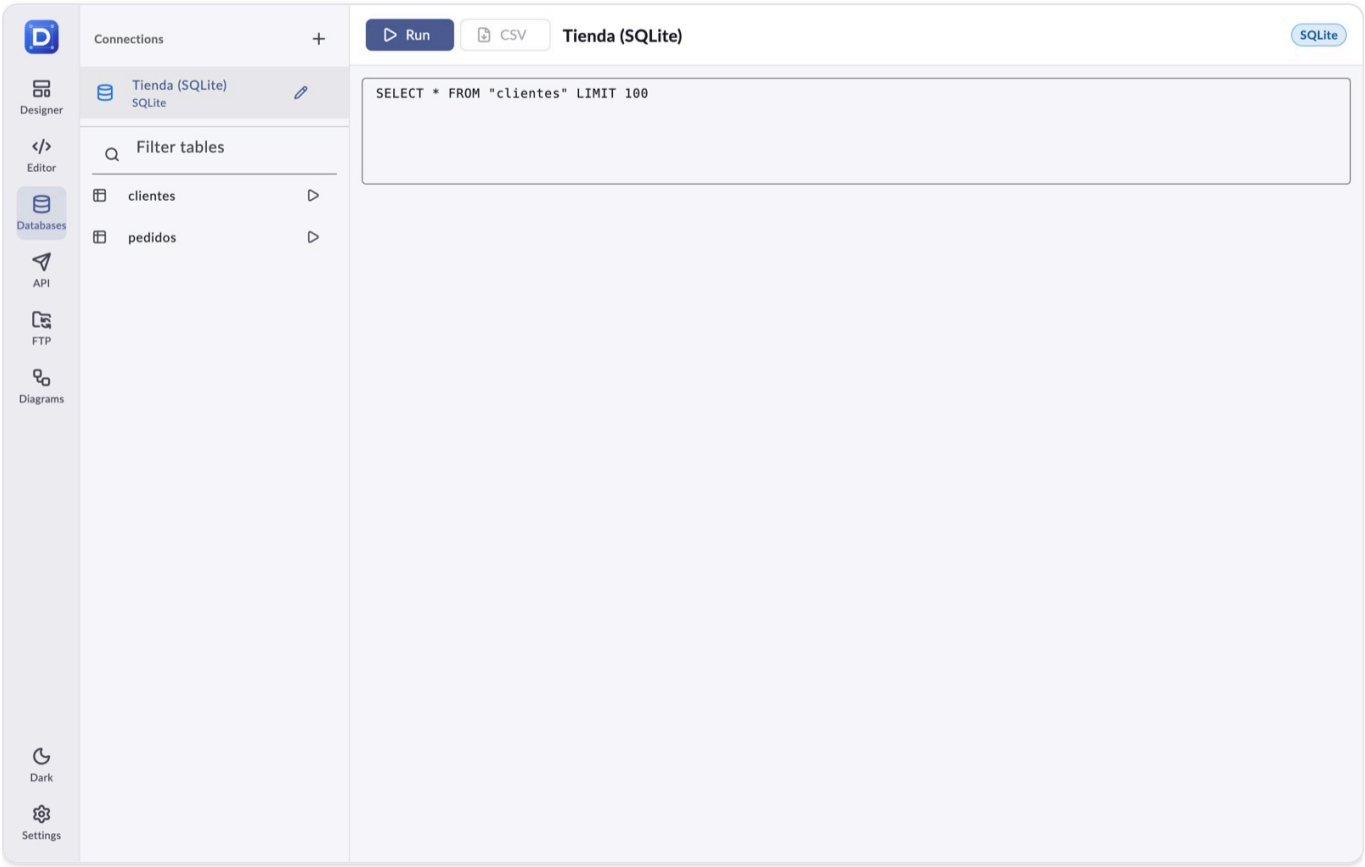
# Databases

Connections, tables, SQL queries and CSV export.

## Connections

**+** adds a SQLite, PostgreSQL, MySQL / MariaDB or MongoDB connection, with server, port, database, user, password and **Encrypted connection (SSL)** depending on the type. **Test connection** checks it can be reached.

The connections saved here are the ones the Designer's windows can use through the **Data connection** component.




A SQLite database with its tables and the query editor.

## Tables and queries

- The table list can be narrowed with **Filter tables**; each table opens a sample query.
- **Run** executes the query in the editor and shows the result in a grid.
- Anything that changes data asks for confirmation first.

- **CSV** saves the full result to a file.

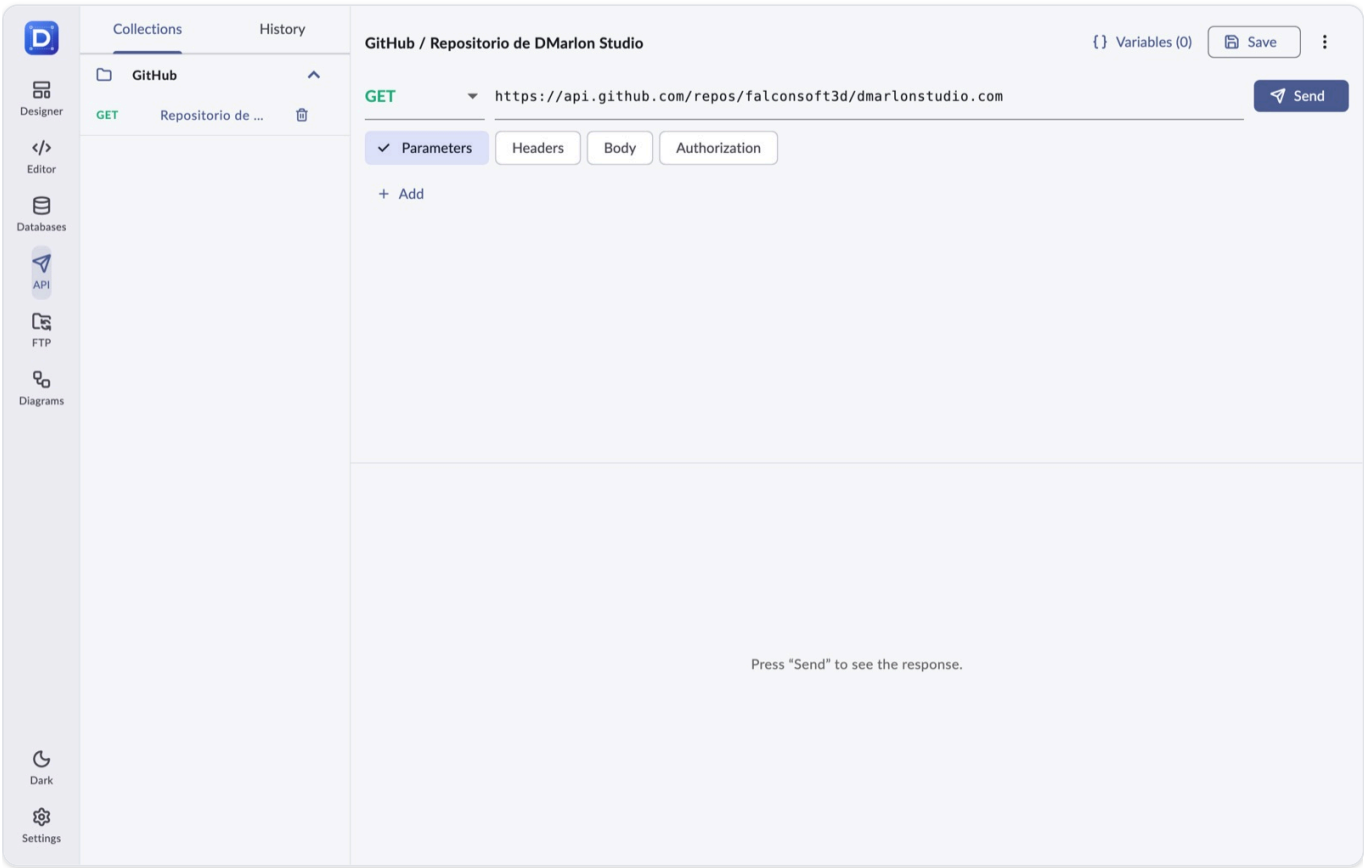
Chapter 7

# API client

HTTP requests with collections, variables and history.

## Requests

- Pick the method (GET, POST, PUT, PATCH, DELETE…), type the URL and click **Send**.
- **Parameters**, **Headers**, **Body** (JSON, form or text) and **Authorization** (none, Bearer or Basic) tabs.
- The response shows the status code, the time, the headers and the body.
- **Copy as curl** gets the request ready for the terminal.




A request saved in the GitHub collection.

## Collections, variables and history

- **Save** stores the request in a **Collection**; **Save as new…** makes a copy.
- **Environment variables…** defines values such as `base_url` or `token`, used in the URL, headers and body as `{{base_url}}`.
- The **History** keeps the last 100 requests sent; **Clear history** empties it.

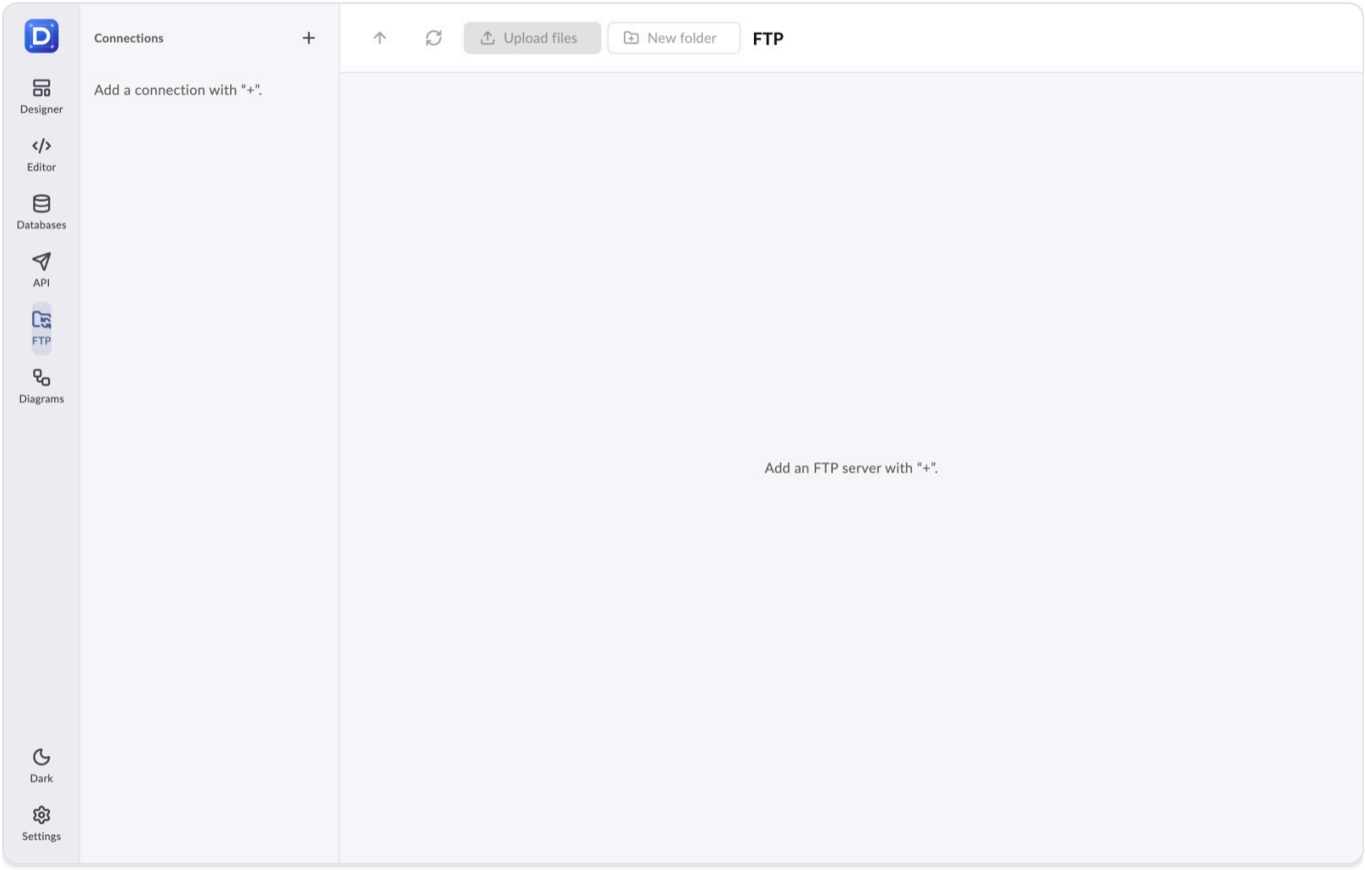
Chapter 8

# FTP

FTP, FTPES and FTPS servers.

## Servers and files

- **+** adds a server with its **Security**: **FTP (unencrypted)**, **FTPES (explicit TLS)** or **FTPS (implicit TLS)**.
- Choose a server to browse its folders; **Parent folder** goes up a level.
- **Upload files**, **Download**, **New folder**, **Rename…** and **Delete** (folders, with their contents).




The FTP client, ready to add a server.

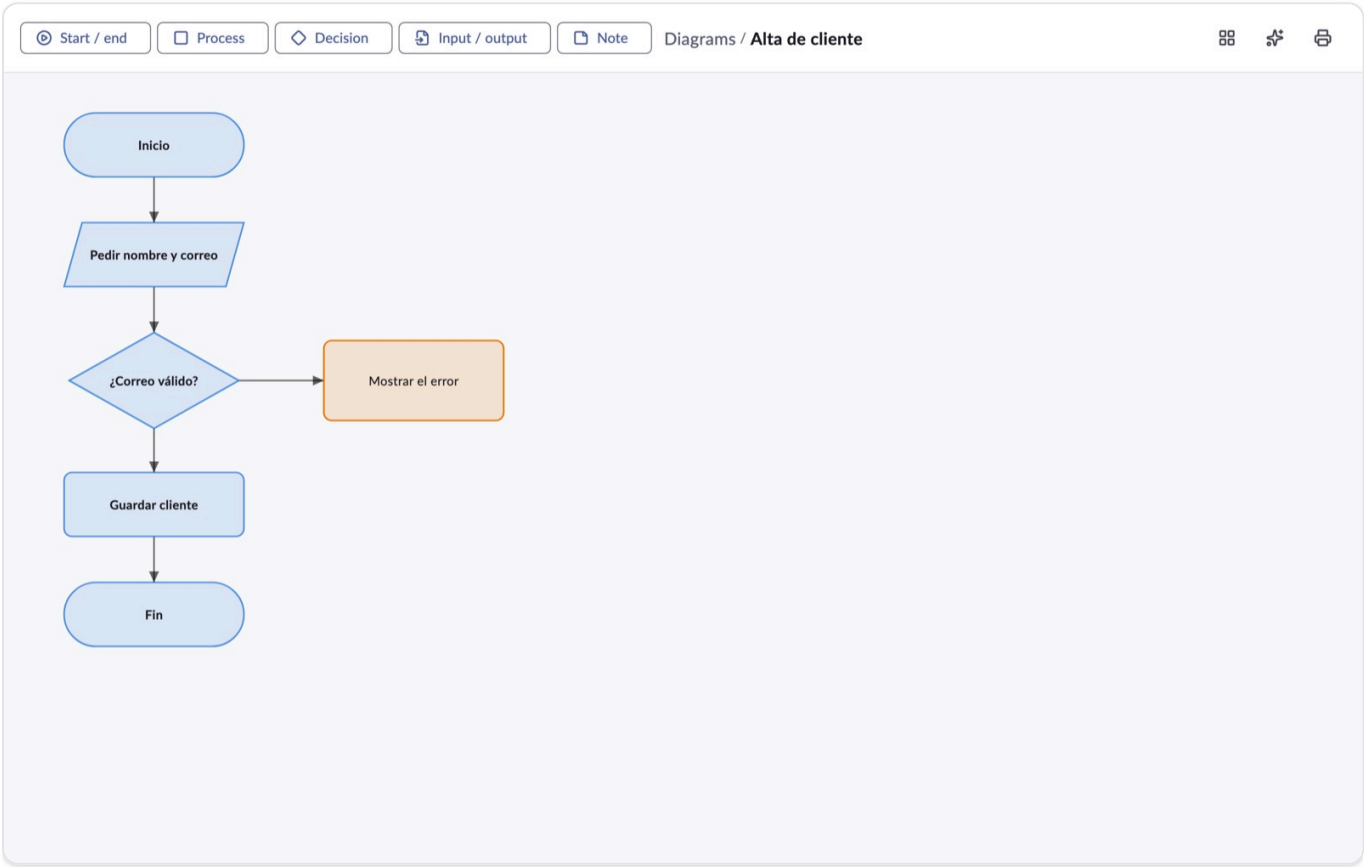
Chapter 9

# Diagrams

Flowcharts and database diagrams, with AI and PDF.

## Flowcharts and database diagrams

- **Flowchart**: **Start / end**, **Process**, **Decision**, **Input / output** and **Note** shapes. Press → next to a shape to connect it to another; connections can have a label (e.g. “Yes”).
- **Database**: tables with their columns, **Primary key** and foreign keys, and the relations between tables. **Diagram SQL** shows the SQL that creates it.
- **From a database** draws the diagram of a saved connection (optionally, only the tables starting with a prefix).
- **With AI** creates the diagram from a description and **Change with AI** modifies it.
- **Arrange** lays the shapes out again and **Export to PDF** saves it for printing or sharing.




A flowchart for registering a customer.

Chapter 10

# Settings

Language, appearance and artificial intelligence.

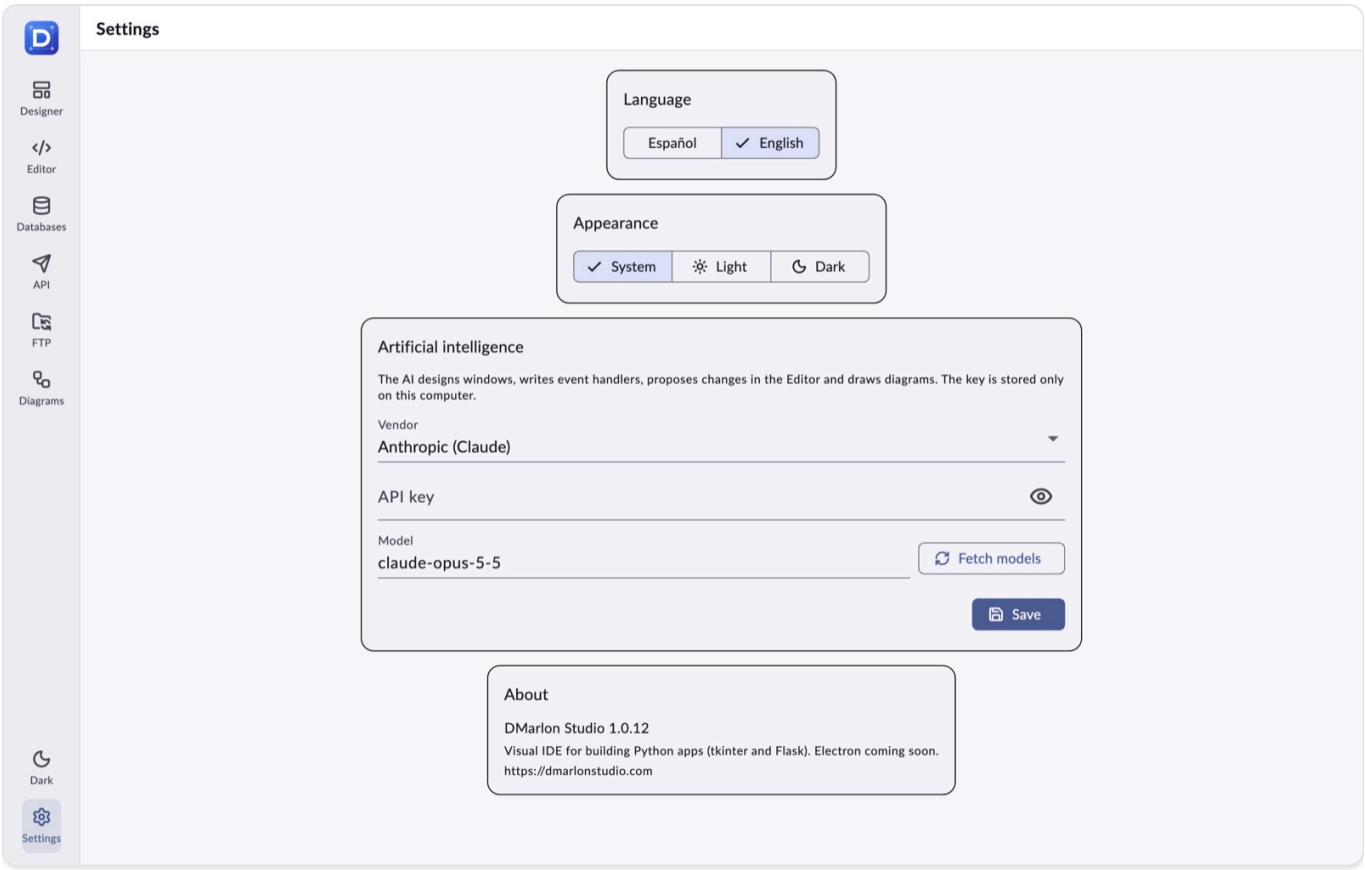
## Connect to Claude (MCP)

- In **Settings → Connect to Claude (MCP)** you turn on the **MCP server**: while DMarlon Studio is open, Claude (Claude Code or another MCP client) can work with your projects.
- Tools: `list_projects`, `get_project`, `list_components`, `create_project`, `add_component`, `update_component`, `remove_component`, `update_code`, `generate_python`, `run_project`, `get_output` and `stop_project`.
- It only answers on this computer (`127.0.0.1`) and with the **Token**. **Add to Claude Code** registers it in one click; you can also copy the `claude mcp add` command shown.
- If a project is open in the Designer with no pending changes, it reloads by itself with what Claude does; with unsaved changes, Claude is told to save them first.

## Artificial intelligence

Choose the provider (**Vendor**): Anthropic (Claude), OpenAI, Gemini, xAI, DeepSeek, Mistral, Groq, Cohere, OpenRouter, Perplexity, Ollama or a compatible custom API. Type the **API key** and the **Model**; **Fetch models** lists those on your account. Claude (`claude-opus-5-5`) is the default provider.

The key is stored on this computer only. Each provider remembers its key and model when you switch between them.




Settings: language, appearance, artificial intelligence and version.

## Language, appearance and version

- **Language**: Español or English, instantly.
- **Appearance**: System, Light or Dark.
- **About** shows the installed version.

Chapter 11

# Keyboard shortcuts

Every Designer and Editor shortcut.

## Designer

| Shortcut | Action |
| --- | --- |
| F9 | Run |
| F12 | Switch between design and code |
| ⌘S / Ctrl+S | Save the project |
| ⌘Z / Ctrl+Z | Undo |
| ⌘C, ⌘X, ⌘V (Ctrl on Windows) | Copy, cut and paste components |
| Delete / Backspace | Delete the selection |
| Arrows | Move one grid step (with Alt, one pixel) |
| Shift + arrows | Resize |
| Double-click on the palette | Add the component |
| Double-click on a component | Open its main event |
| Enter in a chat | Send (Shift+Enter: new line) |

## Editor

| Shortcut | Action |
| --- | --- |
| ⌘S / Ctrl+S | Save the file |
| ⌘J / Ctrl+J | Show or hide the terminal |
| ↑ / ↓ in the terminal | Go through the command history |
| Enter in a chat | Send (Shift+Enter: new line) |

Chapter 12

# Troubleshooting

The most common messages and how to solve them.

## Python not found, or no tkinter

- **Python was not found**: install it (python.org or Homebrew) and click **Look for Python again** on the project list.
- **Found … but it does not include tkinter**: on macOS, `brew install python-tk`; on Linux, `sudo apt install python3-tk`; on Windows, reinstall Python ticking **tcl/tk and IDLE**.
- **Tk 8.5 is old and does not honor the colors of the design**: that is the Python bundled with macOS. Install one from python.org or Homebrew.

## Flask is missing

Web apps need Flask in the same Python. The **Flask is missing** message shows the exact command, for example `python3 -m pip install flask`.

## The app won't open (unsigned)

- **macOS**: the first time, right-click DMarlon Studio → **Open** → **Open**.
- **Windows**: on “Windows protected your PC”, **More info** → **Run anyway**.

## AttributeError: 'Form1' object has no attribute 'root'

This happened when the window's code defined its own `__init__` without calling `super().__init__()`: that method replaced the one that creates the window. Since version 1.0.5 the window and its components are created before your `__init__`, and the **On open** event runs after it.

```
class Form1(FormDesign):
    def __init__(self):
        super().__init__()   # first
        self.current = "0"

    # Even better: the On open event
    def form_create(self):
        self.current = "0"
```

## Claude Code was not found

Install Claude Code from claude.com/claude-code and run `claude` once in a terminal to sign in. The studio looks for it on the PATH and in `~/.local/bin`.

Chapter 13

# Roadmap

What comes next.

## Targets

You choose the target when creating a project; it cannot be changed later:

| Target | Status |
| --- | --- |
| Python + tkinter (desktop) | Available |
| Python + Flask (web) | Available |

## Richer projects

- Several forms per project, with a **Project** node in the tree.
- The application icon, set in the project.
- Building executables (PyInstaller for Python).