


Setup and Usage



Next Level intervals.icu

Step by Step

Python – Git – Visual Studio Code – GitHub – Run it

SECTION 01

Install the software

Python, Visual Studio Code, Git, and the project repository

What we will install

- **Python** – runtime that executes the sync scripts
- **Visual Studio Code** – editor for working with the project files
- **Git** – downloads and updates the repository
- **The repository** – intervals-icu-sync from GitHub

Instructions target Windows. macOS steps are similar.

Step 1 – Install Python

- Open python.org/downloads in your browser
- Click **Download Python 3.x** (latest stable release)
- Run the installer when the download finishes
 - ▶ **Tick “Add python.exe to PATH”** at the bottom – this is critical
 - ▶ Click **Install Now** and accept the User Account Control (UAC) prompt
- Wait for the installer to finish, then click **Close**

Verify – Python is ready

- Open a new **PowerShell** or **Command Prompt** window
- Run these two commands:

```
python --version
```

```
pip --version
```
- Both should print a version number (e.g. Python 3.12.x)
- **Error “not recognized”?** Re-run the installer and tick **Add to PATH**

Step 2 – Install Visual Studio Code

- Open **code.visualstudio.com**
- Click **Download for Windows** (User Installer, 64-bit)
- Run the installer and accept the license
 - ▶ Keep the default install location
 - ▶ **Tick all four “Other” options**: add to PATH, register file types, “Open with Code” context menus
- Click **Install**, then **Finish** to launch VS Code

Step 3 – Install Git

- Open git-scm.com/download/win
- The 64-bit standalone installer downloads automatically
- Run the installer – **defaults are fine**, just click **Next** through
 - ▶ Default editor: leave as **Vim** or pick **VS Code**
 - ▶ PATH option: keep **Git from the command line and also from 3rd-party software**
- Click **Install**, then **Finish**

Verify – VS Code is ready

- VS Code should open automatically – you see the **Welcome** tab
- From any folder, right-click → **Open with Code** should appear
- From the terminal, the code command should launch the editor:

```
code --version
```
- **Recommended:** install the **Python** extension by Microsoft (Extensions panel, Ctrl+Shift+X)

Verify – Git is ready

- Open a **new** PowerShell window (so it picks up the new PATH)
- Run:
`git --version`
- Should print something like `git version 2.4x.x`
- **Set your identity** (used for any commits you make):
`git config --global user.name "Your Name"`
`git config --global user.email "you@example.com"`

Step 4 – Clone the repository

- Open the repo on GitHub:
`https://github.com/rbrands/interval-sync`
- Click the green **Code** button → copy the **HTTPS** URL
- In PowerShell, change to where you want the project:
`cd C:\Users\<you>\source`
- Clone the repo:
`git clone https://github.com/rbrands/interval-sync.git`
- A new folder `interval-sync` appears with all source files

Step 5 – Open the repo in VS Code

- **Option A – from the terminal**
 `cd intervals-icu-sync`
 `code .`
- **Option B – from VS Code**
 - ▶ File → **Open Folder...** → pick the `intervals-icu-sync` folder
 - ▶ Click **Yes, I trust the authors** in the workspace trust dialog
- If prompted, install the recommended **Python** extension
- Open `README.md` for project-specific configuration

Step 6 – Pull updates from GitHub

- **Option A – in the terminal**

```
cd intervals-icu-sync  
git pull
```

- Fetches the latest commits from GitHub and merges them into your local branch.

- **Option B – in VS Code**

- ▶ Left sidebar → **Source Control** (Ctrl+Shift+G)
- ▶ Top right ... menu → **Pull**
- ▶ Or click the ↻ **Sync** icon next to the branch name in the status bar

Tip: Commit your local changes first, or save them with `git stash`, otherwise you may run into conflicts.

Note: The repo includes a `CHANGELOG.md` with release notes and a `VERSION` file showing the current version.

Your setup is ready – verification checklist

- ✓ **Python** `python --version` prints 3.x
- ✓ **VS Code** `code --version` prints a version & commit
- ✓ **Git** `git --version` prints 2.x and identity is configured
- ✓ **Repository** `intervals-icu-sync` folder exists locally
- ✓ **VS Code** opens the project, Python extension is active

Next: follow the project README to install dependencies and configure your intervals.icu API key.

SECTION 02

Prepare Python

Virtual environment, dependencies, and your API key

Step 7 – Create a virtual environment

Isolates the project's Python packages from the rest of your system.

RUN IN THE VS CODE TERMINAL

```
python -m venv .venv
```

```
.venv\Scripts\activate (Windows)
```

```
source .venv/bin/activate (macOS / Linux)
```

You will see (.venv) at the start of the prompt once active. VS Code detects .venv automatically and activates it in new terminals.

Step 8 – Install requirements

Downloads every Python package the project needs.

RUN IN THE VS CODE TERMINAL

```
pip install -r requirements.txt
```

Make sure (.venv) is shown in the prompt — otherwise packages land in the wrong place.

Step 9 – Configure your API key

Tells the scripts which intervals.icu account to talk to.

RUN IN THE VS CODE TERMINAL

```
cp .env.example .env
```

THEN OPEN .env AND SET

API_KEY – in intervals.icu under **Settings** → **Developer Settings**

ATHLETE_ID – also under **Settings** → **Developer Settings** (e.g. i12345)

SECTION 03

Prepare Coach Logic

System prompt, athlete profiles, and coaching domain knowledge

Coaching Logic – two directories

The coaching system is split across two directories that you assemble before talking to your AI coach.

`prompts/system_prompt.md`

The base system prompt for the LLM. Contains a placeholder block where you insert the matching athlete profile.

`coach-logic/`

Modular documentation of the coaching domain knowledge — training philosophy, decision logic, fueling rules, zones, schema and example workouts.

System prompt – swap in the athlete profile

The base prompt prompts/system_prompt.md contains a placeholder:

```
## Athlete Profile
```

```
<<INSERT ATHLETE / DISCIPLINE BLOCK HERE>>
```

Before passing the prompt to the coach, copy the contents of the matching `discipline_*.md` file into that block:

File	Athlete type
discipline_climber.md	Climber / FTP-focused athlete
discipline_criterium.md	Criterium racer / W' and repeatability focus
discipline_roadrace.md	Road racer / aerobic durability and FTP focus
discipline_marathon.md	Mountain marathon rider / aerobic durability + climbing endurance focus

coach-logic/ – domain knowledge modules

Modular markdown files describing the coaching domain knowledge — share them with your AI coach alongside the system prompt.

File	Content
training_philosophy.md	Underlying training principles based on Joe Friel
coach_logic.md	Coaching logic, data interpretation and decision framework
decision_engine.md	How the coach makes training decisions based on input data
fueling_rules.md	Fueling evaluation rules and their coaching impact
training_zones.md	Power, HR and RPE zone definitions used by the coach
input_schema.md	Description of the JSON input schema passed to the coach
workouts.md	Example workouts (VO2max, threshold, endurance) with dose levels and tags

Where to put it – ChatGPT and Claude

ChatGPT

Create a Project

- Sidebar → **Projects** → **New project**
- Open the project

Instructions

Paste the contents of `system_prompt.md` (with the discipline block already inserted).

Files

Upload all files from `coach-logic/` here. Easy to update.

Claude

Create a Project

- Sidebar → **Projects** → **New project**
- Open the project settings

Custom instructions

Paste the contents of `system_prompt.md` (with the discipline block already inserted).

Project knowledge

Upload all files from `coach-logic/` here.

SECTION 04

Typical Workflow

How you use the tool every week/daily with your AI coach

Step 1 – Prepare the week and share with your coach

Pull the latest data from intervals.icu and hand it to your AI coach.

RUN IN THE VS CODE TERMINAL

```
python ./scripts/prepare_week_for_coach.py
```

Generates `data/processed/coach_input_{monday}.json` for the current week.

SHARE WITH YOUR COACH

Send the file to ChatGPT, Claude or another assistant and discuss the outcome and the plan for the week.

Step 2 – Get the plan back and upload it

Take your coach's weekly plan and push it to intervals.icu.

ASK YOUR COACH FOR A JSON PLAN

Format is described in the system prompt. Save the JSON as `data/plan/week_plan.json`.

RUN IN THE VS CODE TERMINAL

```
python ./scripts/upload_plan.py
```

Uploads the plan to intervals.icu.

SECTION 05

MCP Server (optional)

Skip the file copy-paste — let the AI call the tools directly

What is MCP – and why use it

The **Model Context Protocol** lets the AI assistant call your scripts directly instead of you copy-pasting JSON files.

WITHOUT MCP

- Run script → export JSON → paste into chat → copy plan back → run upload script

WITH MCP

- The AI fetches data, generates a plan, and uploads it — all in one conversation
- Server: `scripts/mcp_server.py`, built with FastMCP
- Save and upload only run after explicit user confirmation

MCP tools – what the AI can call

The server exposes six tools that map directly to the manual scripts and data files.

Tool	Description
prepare_week_data	Runs the full pipeline and consolidates the result
get_coach_input	Returns coach_input_{monday}.json (defaults to current week)
get_fueling_analysis	Returns the fueling analysis JSON for a week
get_latest_metrics	Returns the most recent metrics (CTL, ATL, TSB, FTP, HRV...)
save_week_plan	Validates and saves the plan to data/plans/week_plan.json
upload_week_plan	Uploads week_plan.json to intervals.icu (supports dry_run, clear)

End-to-end – one conversation

What happens after you say *"Analyse my training week and create a plan."*

1. **prepare_week_data** – fetch & consolidate data from intervals.icu
2. **get_coach_input** – load data into the AI's context
3. AI generates plan, you review and confirm
4. **save_week_plan** – validate and save to data/plans/week_plan.json
5. **upload_week_plan** – push the plan to the intervals.icu calendar

Tip: pass `dry_run=true` to preview the upload without making API calls.

Claude Desktop – simplest setup

Claude Desktop is the only ready-to-use desktop client supporting local MCP servers. ChatGPT requires extra setup (next slides).

1. EDIT THE CONFIG FILE

Windows %APPDATA%\Claude\claude_desktop_config.json

macOS ~/Library/Application Support/Claude/claude_desktop_config.json

```
{
  "mcpServers": {
    "intervals-icu-coach": {
      "command": "C:\\...\\.venv\\Scripts\\python.exe",
      "args": ["C:\\...\\scripts\\mcp_server.py"]
    }
  }
}
```

Use the absolute path to the .venv Python — not the system Python. Claude Desktop starts the server on launch.

ChatGPT (1/2) – expose the server via HTTPS

ChatGPT MCP connectors require a public HTTPS SSE endpoint — localhost is not reachable from ChatGPT.

STEP A – START THE SSE SERVER

```
./start_mcp_server.ps1
```

Server runs on `http://127.0.0.1:8765/sse`

STEP B – CLOUDFLARE TUNNEL

- Install: `winget install --id Cloudflare.cloudflare`
- Create a free Cloudflare account, add a domain, switch nameservers
- Dashboard → **Networking** → **Tunnels** → create tunnel and install connector with the provided token
- Add public hostname pointing to `http://127.0.0.1:8765`
- Result: `https://intervals-icu-mcp-local.example.com/sse`

ChatGPT (2/2) – add the connector

MCP connectors are available in selected ChatGPT plans and may still be in beta.

STEP C – ENABLE DEVELOPER MODE

- Open ChatGPT → **Profile** → **Settings** → **Apps**
- Enable **Developer Mode** (or **Custom MCP Connectors**, depending on version)

STEP D – ADD THE MCP CONNECTOR

- **Settings** → **Apps** → **Add MCP Connector**
- **Name:** Intervals.icu Coach
- **URL:** `https://intervals-icu-mcp-local.example.com/sse`
- **Authentication:** None
- Save — ChatGPT will detect the server and expose its tools