Here's a detailed summary of the uv (pronounced "you-vee") tool and its **main commands** for both *projects* and *scripts* (based on its official docs). I'll highlight the key commands you'll likely see in the video you referenced, and what the "project" vs "script" distinction means.

What is "uv"

- uv is a high-speed package & project manager for Python — combining functionality of pip, virtualenv, pyenv, poetry, etc. (Real Python)
- It can manage **projects** (directories of code, with `pyproject.toml`, build system, etc) *and* **scripts** (stand-alone `.py` files) with their own inline metadata or dependencies. (docs.astral.sh)
- "Project" in this context means a directory meant to develop a full Python package/app (can be installable, build/distribute). (mathspp.com)
- "Script" means a standalone file you want to execute (maybe small utility) where you can declare inline dependencies, Python version, and run it via uv without manually managing virtualenvs. (docs.astral.sh)

Key Commands for Projects

Here are some of the main commands you'll encounter with uv when dealing with **projects**:

- `uv init [<name>]` — initialize a new project. Eg: `uv init myproject` (maybe with flags like `--app`, `--package`). (DataCamp)
- `uv sync` — sync/install dependencies in the project, set up the environment according to `pyproject.toml` + lockfile. (til.simonwillison.net)
- `uv add <pkg>` — add a dependency to the project (updates `pyproject.toml`, lockfile). (DataCamp)
- `uv remove <pkg>` — remove a dependency. (DataCamp)

- `uv lock` — lock dependencies (create/refresh `uv.lock`). (docs.astral.sh)
- `uv run <command or script>` — run a command inside the project's environment. For example: `uv run pytest` or `uv run python main.py`. (til.simonwillison.net)
- `uv build` — build your project into distribution artifacts (sdist/wheel). (DataCamp)
- `uv publish [--token <token>]` — publish your built package to PyPI or other index. (DataCamp)
- `uv self update` — update uv itself (when installed via installer). (Real Python)
- `uv python install <version>` or `uv python pin <version>` — manage and set the Python version for the project. (GitHub)

Key Commands for Scripts

When dealing with **scripts** (standalone `.py`), uv also has specific commands and flags. Here are the major ones:

- `uv run example.py` — run a script without dependencies. (docs.astral.sh)
- `uv run --with <dependency> example.py` — run a script and dynamically specify dependencies for that run. E.g., `uv run --with rich example.py` if script uses `rich`. (docs.astral.sh)
- `uv run --python <version> example.py` — run the script specifying a target Python version. (docs.astral.sh)
- `uv init --script example.py [--python <version>]` — initialize inline metadata in a script (for dependencies, Python version) so the script becomes self-contained. (docs.astral.sh)
- `uv add --script example.py <deps…>` — declare dependencies inline for the script. Eg: `uv add --script example.py requests rich`. (docs.astral.sh)

- `uv lock --script example.py` — lock dependencies for the script (generates `example.py.lock`). (docs.astral.sh)
- `uv run --no-project example.py` — run the script ignoring the project environment (i.e., don't install the current project) if you are in a project directory. (docs.astral.sh)

There are also tool-/CLI-specific commands around "tool" invocation (for running arbitrary tools) such as:

- `uvx <tool>` (alias for `uv tool run <tool>`) — run a tool in an isolated environment. (Medium)
- `uv tool install <tool>` — install a tool for persistent use. (Medium)

"Project" vs "Script" — clarification

- A **project** under uv is a directory with a `pyproject.toml`, optionally `build-system`, dependencies etc. You develop your application or library here. uv manages its environment, version, building/publishing.
- A **script** is a single file (or small set of files) you want to run quickly without necessarily building a full package. uv still manages dependencies/env for you, making it reproducible.
- The educator in the video probably meant that when you are inside a "project" uv may treat things differently (for example, `uv run` may install your current project by default) vs when you run just a "script" you might use `--no-project` or inline metadata. Example: In script guide: "If you use `uv run` in a project … it will install the current project before running the script. If your script does not depend on the project, use the `--no-project` flag." (docs.astral.sh)
