# x_make_py_venv_x â€” Interpreter Furnace Manual

This furnace provisions and maintains virtual environments across the interpreter matrix we rely on. It installs runtimes, bootstraps virtualenvs, hydrates dependencies, and syncs tooling metadata so every pipeline runs against the exact Python version it expects.

## Mission Log

- Discover or install interpreters through `uv`, `pyenv-win`, the Windows `py` launcher, or the active executable.
- Forge virtual environments under deterministic names like `.venv-3.12.6` in the chosen root.
- Run `ensurepip --upgrade`, optionally upgrade pip, and install dependency sets from auto-detected or user-provided requirement files.
- Update `.python-version`, `tox.ini`, and any auxiliary manifests so the test matrix and developer tooling stay aligned.

## Instrumentation

- Python 3.11 or newer to drive the script.
- Optional managers: `uv`, `pyenv-win`. The furnace can bootstrap `uv` with `--bootstrap-uv` if absent.

## Operating Procedure

```
powershell Set-Location C:\x_runner_x C:\x_runner_x\.venv\Scripts\python.exe
x_make_py_venv_x\x_cls_make_py_venv_x.py \ 3.12.6 3.11 --tool uv --env-root .venvs \ --bootstrap-uv --update-
tox --write-python-version
```

Key switches: - `VERSION ...`: Interpreter versions to guarantee. - `--tool {auto,uv,pyenv,py,current}`: Force a specific acquisition path (defaults to `auto`). - `--env-root PATH`: Target directory for environments. - `--requirements PATH`: Additional requirements files to install (repeatable). - `--default-requirements PATH`: Extra auto-detection hints (defaults include `requirements.txt` and `x_0_make_all_x/requirements.txt`). - `--package PACKAGE`: Packages to install in every environment (repeatable). - `--write-python-version`: Emit `.python-version` pinned to the first interpreter. - `--update-tox`: Rewrite `tox.ini` envlists to match provisioned versions. - `--skip-pip-upgrade`: Skip the pip upgrade step after `ensurepip`. - `--no-auto-requirements`: Disable automatic requirements discovery. - `--bootstrap-uv`: Install `uv` via `pip` when missing. - `--dry-run`: Show the plan without executing. - `--verbose`: Echo each command before execution.

## Example Workflow

```
powershell Set-Location C:\x_runner_x C:\x_runner_x\.venv\Scripts\python.exe
x_make_py_venv_x\x_cls_make_py_venv_x.py \ 3.12.6 --tool uv --env-root .venvs \ --bootstrap-uv --write-python-
version --verbose
```

This sequence installs `uv` if required, ensures Python 3.12.6 exists, builds `.venvs\.venv-3.12.6`, upgrades pip, installs detected requirements, and pins `.python-version`.

### NiceGUI Prototype Status

The NiceGUI prototype is retired. The PySide6 control center under `x_0_make_all_x/interface/gui/app.py` is the supported interface; this furnace no longer installs NiceGUI by default.

## Updating Tox and `.python-version`

```
powershell C:\x_runner_x\.venv\Scripts\python.exe x_make_py_venv_x\x_cls_make_py_venv_x.py \ 3.12 3.11 --tool
uv --env-root .venvs \ --update-tox --write-python-version
```

This keeps `tox.ini` and `.python-version` synchronized with the provisioned interpreters.

## Dry-Run Preview

```
powershell C:\x_runner_x\.venv\Scripts\python.exe x_make_py_venv_x\x_cls_make_py_venv_x.py \ 3.12.6 --tool uv
--env-root .venvs --dry-run --verbose
```

Use dry-run mode to review steps in CI logs or before changing shared environments.

---

Document additional workflows by capturing real command output. The furnace operates deterministically; the more evidence we archive, the cleaner the lab stays.