SDEV 1001

Programming Fundamentals

Making Decisions in Programming - 1 and 2

Expectations - What I expect from you

- No Late Assignments

- No Cheating

- Be a good classmate

- Don’t waste your time

- Show up to class

Agenda

On the right is what we will cover today.

Welcome — First PyPI Project
Why use virtual environments?
What we'll accomplish
Step 1 — Create the virtual environment
Step 2 — Activate the virtual environment
Step 3 — Package choices (new examples)
Install the packages
Save dependencies to requirements.txt
Example — using python-dotenv for config
Example — generate a simple HTML report with yattag
Benefits — why these examples matter
Dependency pinning tips
Deactivate the virtual environment
Recap — what we covered
Try it yourself — quick instructions
Next steps (optional)

Welcome — First PyPI Project

- Quick demo: create a virtual environment, install packages, use them, and save dependencies
- Goal: practical steps you can repeat for any Python project

Why use virtual environments?

- Keep project dependencies isolated
- Prevent version conflicts between projects
- Makes projects reproducible and easier to deploy

What we’ll accomplish

- Create a virtual environment
- Install two small PyPI packages (different examples from the README)
- Use them in tiny scripts
- Save dependencies to a requirements file

Step 1 — Create the virtual environment

- Command:

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

- Notes: - I used `.venv` (convention used by many tools) - Creates a local Python environment in the project folder

Step 2 — Activate the virtual environment

- Windows (cmd.exe):

```
.venv\Scripts\activate
```

- macOS / Linux:

```
source .venv/bin/activate
```

- Prompt will change to show the environment name

Step 3 — Package choices (new examples)

- Example packages:
  - `python-dotenv` — read .env files for configuration
  - `yattag` — simple HTML generation for tiny reporting

Why these?

- `python-dotenv` shows good configuration practice
- `yattag` demonstrates using third-party helpers to generate output

Install the packages

```
pip install python-dotenv yattag
```

- Installing both in one command saves time
- `pip` will also install any needed dependencies automatically

Save dependencies to requirements.txt

```
pip freeze > requirements.txt
```

- Creates `requirements.txt` listing installed packages and exact versions
- Good for sharing or redeploying the project

Example — using python-dotenv for config

- Run:

```
python report_config.py
```

- Create `.env`:

```
APP_TITLE="My Demo App"
AUTHOR="Ada Learner"
```

- Then create `report_config.py`:

```python
from dotenv import load_dotenv
import os

load_dotenv()  # reads .env into environment variables

def get_config():
    return {
        "title": os.getenv("APP_TITLE", "Untitled"),
        "author": os.getenv("AUTHOR", "Unknown")
    }

if __name__ == "__main__":
    print(get_config())
```

Example — generate a simple HTML report with yattag

- Run:

```
python make_report.py
```

- Create `make_report.py`:

```python
from yattag import Doc
from report_config import get_config

def build_report(data):
    doc, tag, text = Doc().tagtext()
    cfg = get_config()

    with tag('html'):
        with tag('body'):
            with tag('h1'):
                text(cfg["title"])

    return doc.getvalue()

if __name__ == "__main__":
    sample = ["Point A: Install venv", "Point B: Use dotenv
    print(build_report(sample))
```

Benefits — why these examples matter

- `python-dotenv` shows configuration separation (no secrets in code)
- `yattag` shows leveraging small tools to generate HTML quickly
- Both illustrate the common pattern: install, import, use

Dependency pinning tips

- `pip freeze` lists everything, including transitive deps
- For apps, that’s fine; for libraries, prefer a minimal `requirements.in` and use a lockfile tool
- Tools: `pip-tools`, `poetry`, `pipenv` (optional next steps)

Deactivate the virtual environment

- When finished:

```
deactivate
```

- Restores your normal shell prompt

Recap — what we covered

- Create and activate a venv
- Install packages (`python-dotenv`, `yattag`)
- Use them in small scripts
- Save dependencies with `pip freeze`
- Deactivate when done

Try it yourself — quick instructions

1. Create a new folder for the demo
2. Run:

```
python -m venv .venv
.venv\Scripts\activate
pip install python-dotenv yattag
pip freeze > requirements.txt
```

3. Add `.env`, `report_config.py`, `make_report.py` from slides 4. Run `python make_report.py` and view the output

Next steps (optional)

- Explore `pip-tools` to manage top-level dependencies only
- Try `poetry` for project and dependency management
- Publish a tiny package to test PyPI workflow (advanced)