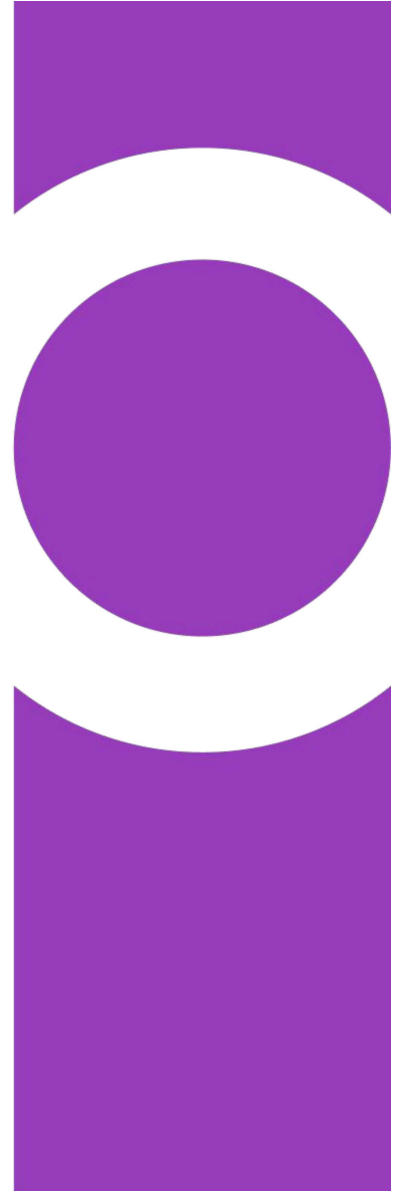


# Python to Django: Essentials



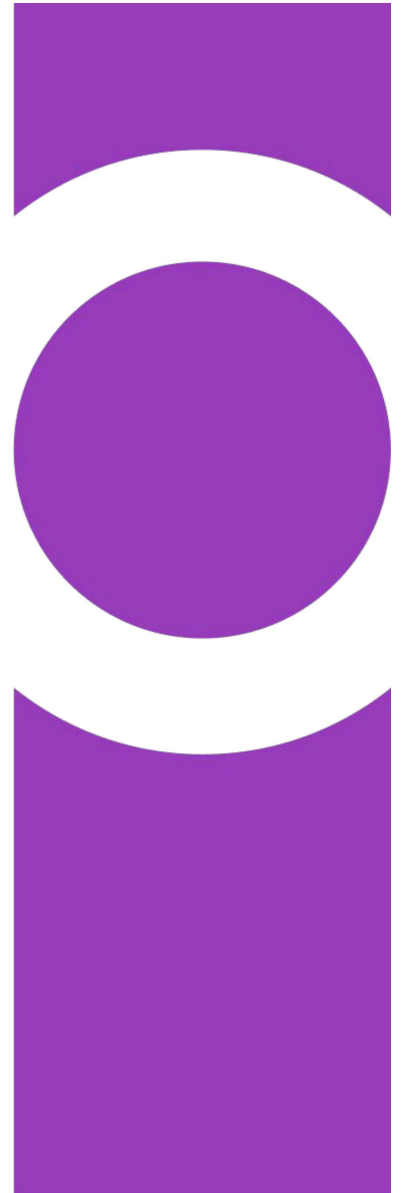
# Python Basics

- **Understand core concepts:** variables, data types (int, float, str, list, dict, tuple), and basic operations.
- **Control flow:** if-else statements, loops (for, while), and conditional expressions.
- **Functions:** def, return, \*args and \*\*kwargs, lambda functions.
- **Modules and Packages:** import custom and built-in modules, creating packages.
- **Exception handling:** try-except blocks, raising exceptions, finally clause.
- **File operations:** open, read, write, with-context managers.



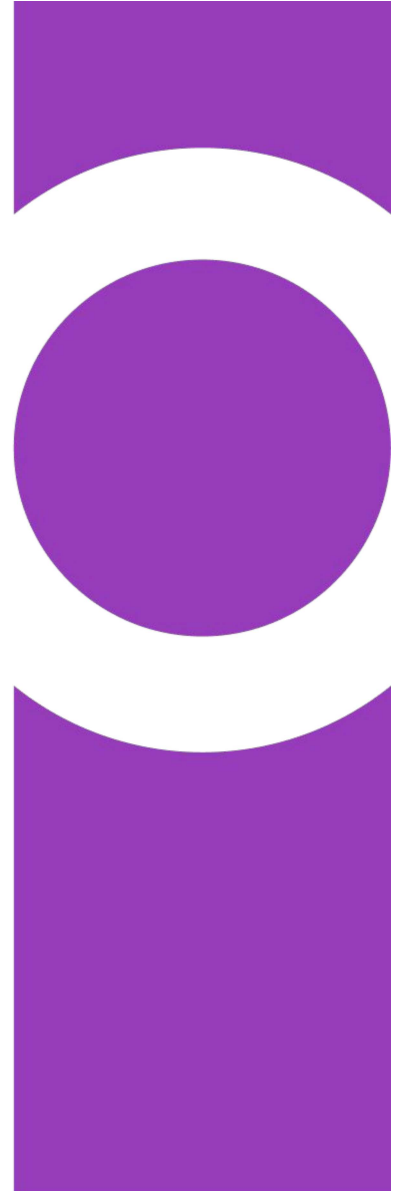
# VSCode Overview

- Install **Python** & **Django** extensions
- Use **integrated terminal** for venv and CLI tools
  - Set project interpreter via **.vscode/settings.json**
  - Configure **launch.json** for debugging workflows
  - Enable **linting** (pylint/flake8) and **formatting** (black/prettier)
  - Use **IntelliSense** for smart code hints and navigation



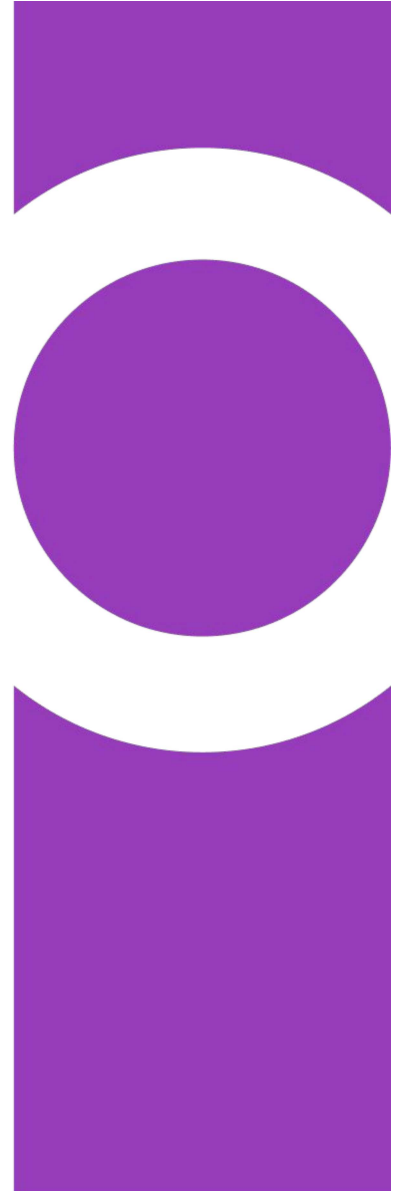
# Environment Setup

- Install Python from official source and verify using ``python --version``.
- Set up virtual environments using ``venv`` or ``virtualenv`` to isolate project dependencies.
- Use pip to install required packages and generate ``requirements.txt``.
- Use ``.env`` files to manage environment-specific variables.



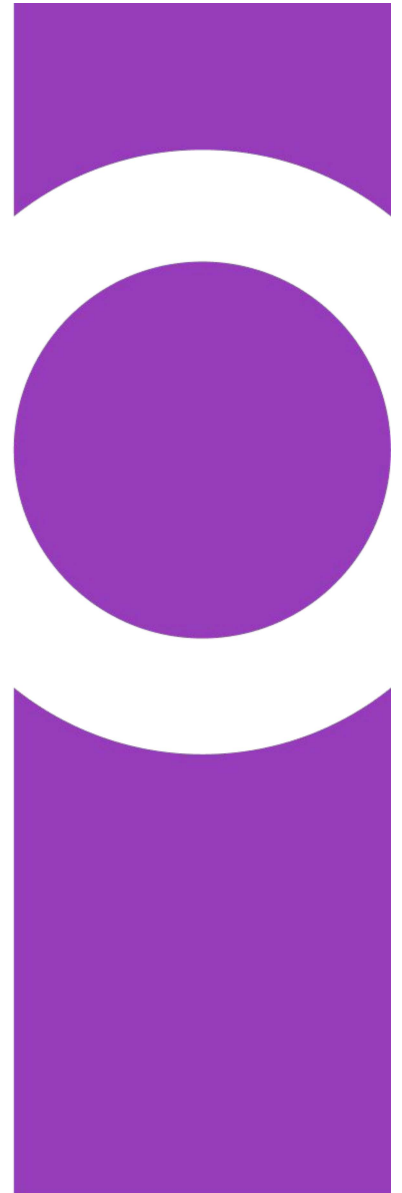
# Frameworks: Pros & Cons

- **Flask:** Micro-framework, flexible but requires manual setup for DB, auth, etc.
- **FastAPI:** Modern, high performance using async, but smaller ecosystem.
- **Pyramid:** Highly configurable but steeper learning curve.
- **Django:** Opinionated and full-featured, great for quick MVP and production apps.



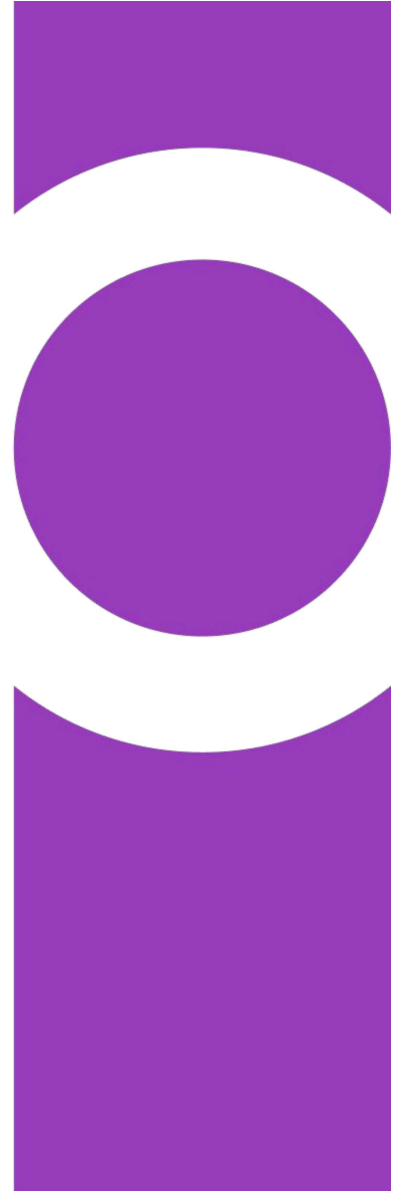
# Why Django?

- Django is a high-level, secure, and scalable framework with batteries-included.
- Built-in admin panel, authentication system, and ORM make development faster.
- Ideal for rapid development and clean, pragmatic design.
- Excellent documentation and large community support.



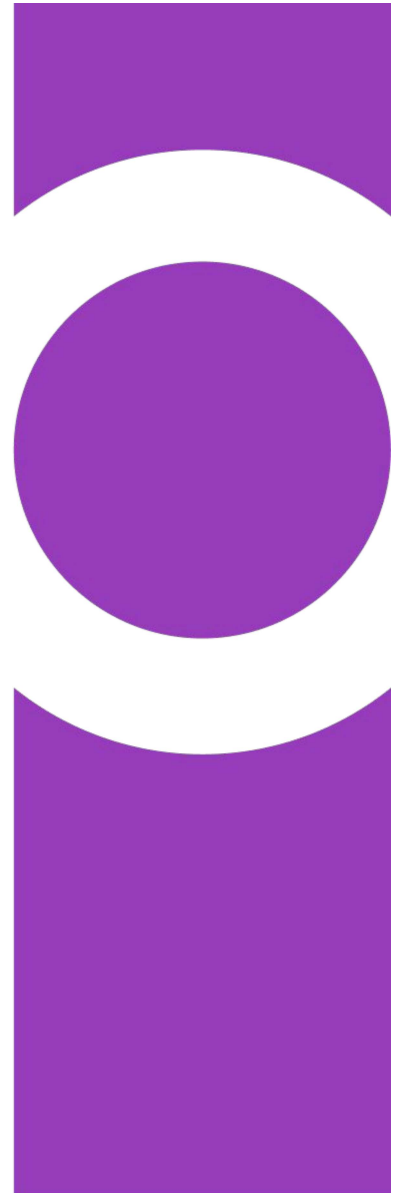
# Django Project Structure

- ``manage.py``: entry point to Django project commands.
- ``settings.py``: configuration for DB, apps, middleware, static files.
- ``urls.py``: central routing config using ``path()`` and ``include()``.
- **App structure**: `models.py` (DB schema), `views.py` (logic), `templates/`, `static/`, `migrations/`.



# Static & Media File Management

- Static files: CSS, JS, images placed in static/ and served with ``collectstatic``.
- ``STATIC_URL`` and ``STATICFILES_DIRS`` defined in settings.py.
- Media files: User uploads served via ``MEDIA_URL`` and saved to ``MEDIA_ROOT``.
- Configure URL patterns to serve media during development using ``static()`` from `django.conf.urls.static`.





# URL Routing

- `path()` maps URL patterns to views; `re_path()` allows regex routes.
- **Modular URLs:** each app has its own `urls.py` included in the root config.
- Named routes improve readability and reverse URL resolution.
- Use `include()` to maintain scalable URL configurations.

