


MODERN PYTHON DEVELOPMENT

Viktor Ahlqvist

 /vikahl/modern-python-development

- "Modern" Python versions
 - Python 3.6 – 2016-12-23 (security updates until 2021)
 - Python 3.7 – 2018-06-27
 - Python 3.8 – 2019-10-14
- The examples will work on Linux/Mac, most often on Windows

AGENDA

Installation

Tools and code quality

Type hinting

Testing

Package as a module

Automate tests

Project template with cookiecutter

- Package which will be used as an example
- Extracts *TODO* notes from code and returns it as JSON
- Good as an example, limited "real" usage

/vikahl/todo-extractor

INSTALLATION

SYSTEM INSTALL AND VIRTUALENV

Three advices

1. Do not install anything in the system Python
2. Use *virtual environments*
3. Have a system for managing multiple Python versions

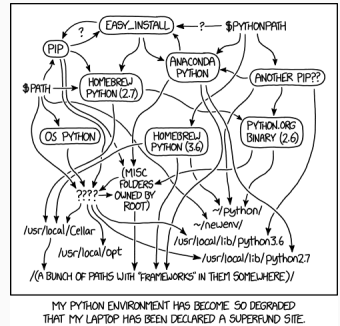


Figure 1: Randall Munroe, xkcd 1987

REMEMBER

- You cannot see which packages that was installed directly and which was installed as dependencies
- Cannot, in an easy way, lock dependencies in multiple levels

- Local package installations per environment/project
- Easy "uninstallation" by removing the folder
- Create in the shell or through IDE and activate with activation script `python3 -m venv venv`

Do not forget to ignore the folder in git!

PYENV - MANAGE VERSIONS

- Different Python versions per project
- Install versions independent on the system package manager
- Automatic activation in specific folders with `.python-version` file
- Install with plugins for virtualenv support

```
viktor@aldertech:~$ python3 -V
Python 3.7.3
viktor@aldertech:~$ cd todoextract/
(todo-extractor) viktor@aldertech:~/todoextract$ python3 -V
Python 3.8.0
(todo-extractor) viktor@aldertech:~/todoextract$ pyenv shell 3.5.3
viktor@aldertech:~/todoextract$ python3 -V
Python 3.5.3
—
```

PIPX - MANAGE "BINARIES"

- Isolate Python program in separate venv
- Can both install and run directly
- Easy updating with `pipx upgrade-all`
- *Possible to have multiple versions of the program installed at the same time, but needs to manually symlink then*

Install it with `pip3 install pipx`

Is hopefully *the only* package that is installed in system Python

TOOLS AND CODE QUALITY

- Programs that analyse the code for potential errors
- PEP 8 – *Style guide for Python code*

- Syntax, style, docstrings, static analysis
- Whiny, but catches a lot
- Will probably need config

```
C0326: No space allowed around keyword  
argument assignment  
C0114: Missing module docstring  
W0102: Dangerous default value [] as  
argument  
C0116: Missing function or method  
docstring  
E0602: Undefined variable 'my_list'
```

Your code has been rated at -12.50/10

```
def add_2(arg = []):  
    arg.append(2)  
    print(arg)  
    return my_list
```

- Similar analysis as Pylint
- Covers less means less whiny
- Easier to use in existing code bases

```
E251 unexpected spaces around keyword /  
parameter equals
```

```
F821 undefined name 'my_list'
```

```
def add_2(arg = []):  
    arg.append(2)  
    print(arg)  
    return my_list
```

- Security oriented, rates issues with severity and confidence
- Searches in the code and included packages

```
>> Issue: [B502:ssl_with_bad_version] ssl.wrap_socket call with
insecure SSL/TLS protocol version identified, security issue.
    Severity: High    Confidence: High
    Location: ./ssl-insecure-version.py:4
    More Info: https://bandit.readthedocs.io/en/latest/plugins/b502...
3
4         ssl.wrap_socket(ssl_version=ssl.PROTOCOL_SSLv2)
5         SSL.Context(method=SSL.SSLv2_METHOD)
```

AUTO FORMATTERS : BLACK

- Formats the code automatically
- *The uncompromising code formatter*
 - line length
 - normalization of quotes
- Guarantees the same execution of code after formatting
- Alternatives: YAPF, autoformatter

My advice: choose Black and focus on delivering value instead of bike shedding

LINTERS OCH AUTO FORMATTERS

RUN LINTERS AND AUTO FORMATTERS AT PR AND
DEMAND THAT THE ARE SUCCESSFUL BEFORE MERGE

TYPE HINTING

TYPE HINTING AND TYPE ANNOTATIONS

- Python is still dynamically typed
- Makes it easier for developers and IDE
- Developed driven by Dropbox with MyPy
- Static type checking (linting) with type checkers

```
def prepare_plot(value: int, unit: str) -> dict
```

- The *typing* module gives more possibilities
- `typing.Union[str, int]`
- `typing.List[str]`
- `typing.Dict[str, set]`
- `typing.Iterable`
- `typing.Any`
- ...
- Classes are also types
- Use strings to annotate if the type is not available

- Literal defines unique values

```
def print_length(length, unit=typing.Literal["m", "ft"])
```

- Final for values that *should not* be re-defined

```
user_id: typing.Final[int] = 24
```

- @typing.final for classes and methods that *should not* be inherited or re-defined

NEW IN 3.8 : TYPEDDICT : PEP 589

```
class OpKoKo(typing.TypedDict):  
    edition: str  
    start_date: datetime.date  
  
current = OpKoKo(edition="19.2", date=datetime.date(2019, 11, 8))  
# ...  
def printable_program(current: OpKoKo) -> str:
```

- Better control of types in a dict
- The object is a normal dict
- *Similar to* named tuple *or* dataclasses

NEW IN 3.8 : PROTOCOLS : PEP 544

- Static duck typing
- Specifies methods/attributes that an object *should* have

```
class ChessPiece(typing.Protocol):  
    name: str  
    def move(self) -> None:  
        pass  
  
def play(piece: ChessPiece, x: int, y: int) -> None:  
    piece.move(x, y)
```

TESTING

- Fantastic test framework, better than Unittest from standard lib
- Can run Unittest tests
- Read Brian Okkens *Python Testing with pytest* (Pragmatic bookshelf)
- "Pure" asserts: `assert data is not None`
- Tests functions named `test_` placed in files with same prefix

FIXTURES

- Functions that are run before (and sometimes after) tests
- Several exists, easy to extend
- Use for temp files, mock objects, database setup, ...
- Can be scoped to only run once per session, class, module, function (default)

```
def test_parse_config(tmpdir):  
    config = tmpdir / "config.json"  
    config.write_text(json.dumps(TEST_CONFIG))  
    parser = Parser.from_config(config_file=str(config))  
    assert parser.parsed_config == TEST_CONFIG
```

WRITE YOUR OWN FIXTURES

- Decorate functions with `@pytest.fixture()` decorator
- Share between tests in `conftest.py` file (loaded as plugin by Pytest)

```
@pytest.fixture()
def user_config():
    return {'username': 'kalle', 'birthdate': '1987-01-01'}

def test_user_age(user_config):
    user = User(user_config)
    age = # ... calculate expected age
    assert user.age == age
```

FIXTURES FOR SETUP AND TEARDOWN

- Fixtures can be used for both setup and teardown
- Code after `yield` is guaranteed to run

```
@pytest.fixture()
def tasks_db(tmpdir):
    # Setup : start db
    tasks.start_tasks_db(str(tmpdir), 'tiny')

    yield # this is where testing is happening

    # Teardown : stop db
    tasks.stop_tasks_db()
```

Figure 2: Brian Okken, Python testing with pytest, p. 51

PACKAGE AS A MODULE

- There are multiple ways – I recommend Flit
- Start with `flit init`
- Configurable in `pyproject.toml`
PEP 518 – *Specifying minimum build system requirements for Python projects*

- Does not support `requirements.txt` → flytta beroenden
- Install with sym links/`pth`-files for TDD

PUBLISH THE MODULE

- Test against `test.pypi.org`
- Configure your own servers in `~/.pypirc`
- `flit --repository testpypi publish`

Install the module with `pipx install todo-extractor`

AUTOMATE TESTS

- Framework for managing git pre-commit hooks
- Written in Python, but not limited to Python
- Configured in `.pre-commit` YAML file
- Must be installed manually by each developer, does not replace CI

EXAMPLE CONFIGURATION

```
repos:
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v2.3.0
  hooks:
    - id: pretty-format-json
    - id: check-json
- repo: https://github.com/psf/black
  rev: stable
  hooks:
    - id: black
- repo: https://github.com/pre-commit/mirrors-pylint
  rev: master
  hooks:
    - id: pylint
```


- Automates tests with defined test environments
- Unique venvs disconnected from development
- Easily enables testing in multiple Python versions
- Can be run in CI

PROJECT TEMPLATE WITH COOKIECUTTER

- Framework to create projects from Jinja2 templates
- Written in Python, but not limited to Python
- Can create whole file structure with folders

/vikahl/python-template

FINISHED, QUESTIONS?

/VIKAHL/MODERN-PYTHON-DEVELOPMENT

/VIKAHL/TODO-EXTRACTOR

/VIKAHL/PYTHON-TEMPLATE

MODERN PYTHON DEVELOPMENT

Viktor Ahlqvist

 /vikahl/modern-python-development

SOURCES AND INSPIRATION

- Python bytes (podcast)
- Test and code (podcast)
Especially episode 80 *From Python script to maintainable package* and 81 *TDD with Flit*
- Talk Python to me (podcast)
- Real Python (realpython.com)
- docs.python.org