WELCOME

** This document helps in maintaining standards and I believe in pushing the boundaries of what's possible, reaching new heights, and setting standards that go beyond the ordinary flow . Feel free to adjust the implemented details based on your requirement .

BEYOND BASICS ‘ BEST PRACTICES IN SDLC

AGENDA

- Styling guides
- Branching and versioning
- Ensuring quality
- Dockerizing
- CI / CD
- Security
- Documentation
- Testing

Why coding standards

- Reduced Debugging Time
- Improved Code Review
- Easy Refactoring and Maintanence
- Improved code quality

NAMING CONVENTIONS

venzo

Module Name

- Keep it as short as simple
- Do not elaborate the process in file naming .
- Use underscore for lengthy name for better readability
- example :
  - dataprocessingfile.py ✗
  - data_process.py ✓

Code Entities

- Start each word with an uppercase letter, and don't use underscores to separate words
- For function and method names should be in lowercasde with underscore seperated .

```python
# don't do this
class process_data:

    def cleanincomingdata():
        pass

# do this instead
class ProcessData:

    def clean_data():
        pass
```

Identifiers

- Use descriptive naming .
- Give pronounceable names .
- Avoid using confusing abbreviations

```python
# don't do this
days = 12
name = request.data

# do this instead
DUE_DAYS = 12
customer_name = request.data
```

ORGANIZING IMPORTS

Organizing rules

- Don't make it one liners

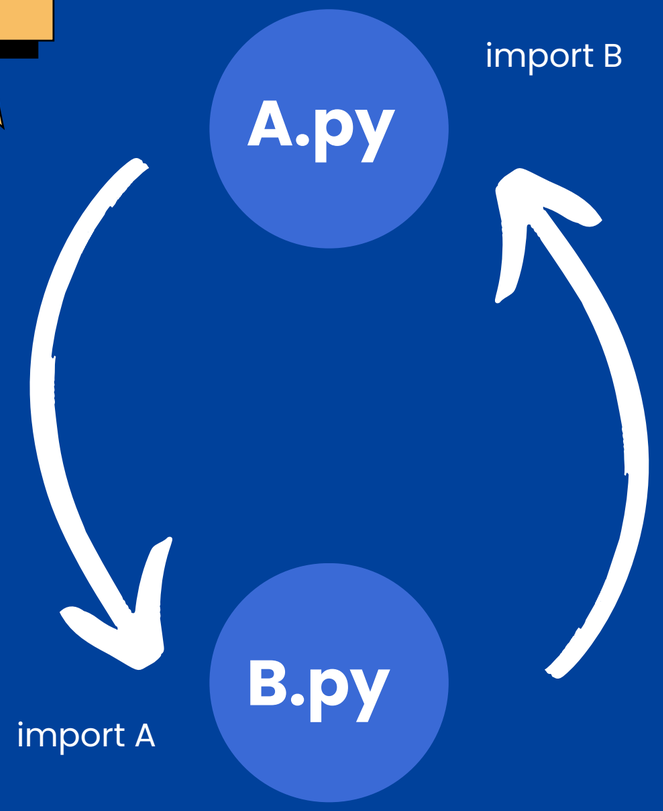
```
# don't do this
import os , import contextlib , import zlib


# do this instead
import os
import contextlib
import zlib
```

- Don't make circular import
- Follow step down import

```
# don't do this
from app import src
from typing import List
import os

# do this instead
import os    # standard import
from typing import List  #package or library import
from app.src import somefunc #app imports
```

- Bring dependencies file closer to our source folder .
- use **isort** for improts ordering

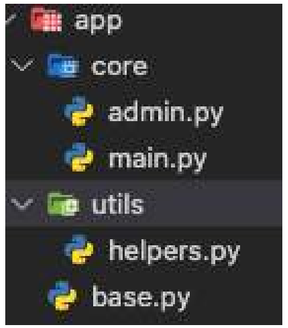
Don't do this inorder to end up with circular import error



FOLDERING TIPS

Do’s

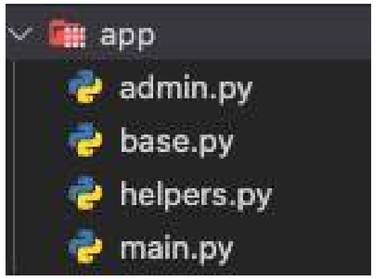
- Do use subfolders to further organize content



- Follow a Consistent Naming Convention
- Don't Overcomplicate
- Don’t add version names

Don’ts

- Don’t overload all the files in a single folder



- Dont give unrelated folder name
- Don't Overcomplicate
- Don’t add version names

BRANCHING AND VERSIONING

Why low branches

- Simplifies the repository structure, reduces complexity, and makes it easier to understand the flow of changes.
- Provides a clear release history and tags management .

- Development might be less isolated, and conflicts may arise when merging larger changes.
- Challenging to maintain a stable master branch.

Why More branches

- Makes it easier to manage multiple ongoing tasks concurrently.
- Each branch represents a specific feature or bug fix.

- The complexity of managing and merging numerous branches may increase.
- Increased Maintenance and complex in solving merge conflicts

Versioning Methodology

SEMVER

- Semantic Versioning
- Follows as Major , Minor , Patch
- E.g.,
  - LCL-v1.0.0
  - BETA-v1.1.0
  - PRD-v.1.1.1
- alternative not recommended
  - Calver , Rc's , beta

ENSURING QUALITY

- Follow the coding standards
- Do a code review
- Use version control
- Devil is in the details (Refactoring)
  - Consistent naming
  - Better Error Handling
  - Making better documentation
- Pre-commit before the actual commit
- Break into pieces
- Performance profiling
- Ensure concurrency and dead lock detection

DOCKERIZING

- Use official images and non vulnerable
- Seperate dependencies
- Add environment variable
- utilize ignore.
- Avoid unnecessary WORKDIR
- multi stage builds
- health checks
- Security scanning

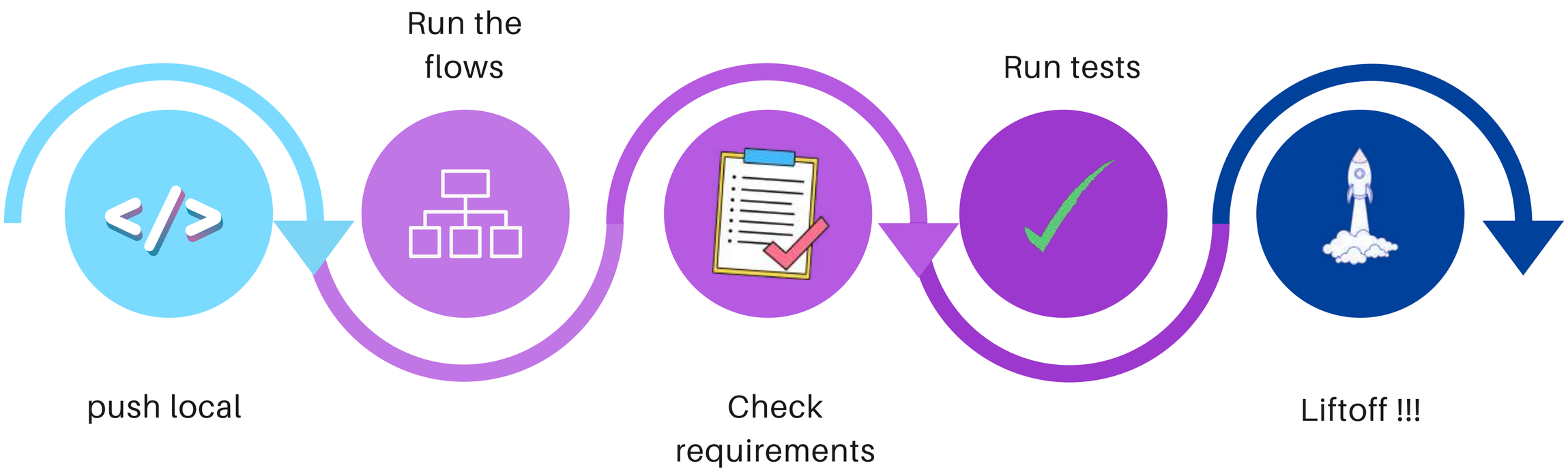
dockerfile > ...

```dockerfile
FROM python:3.8-slim

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir gunicorn \
    && pip install --no-cache-dir -r requirements.txt \
    && pip install --no-cache-dir uvicorn

RUN python manage.py collectstatic --noinput

RUN python manage.py test testing/folder

CMD ["python","manage.py","runserver","0.0.0.0:8000"]
```

CI / CD

- Automating multi branch build
- Fast builds
- Add Rollback mechanism
- Pipeline as code .
- Collab and communicate
- Reduce less configuration drift
- Treat your code as Iac
- Post deployment alerts
- Manage deployment and chaos

Example flow



SECURITY

- Use LTS version for framework or package
- Keep dependencies updated
- Enable CSRF effectively
- Encrypt env and credentials
- Encrypt env and credentials
- Use MFA and 2FA
- Monitor incidents and Audits
- Regular patching and updating
- Harden server configuration

DOCUMENTATION

venzo

Do’s

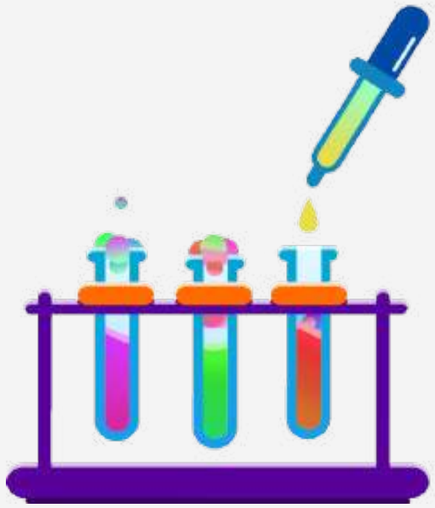
- Provide Clear Comments
- Document at the top
- Meaningful names
- Keep Documentation Updated
- Use md , rst effectively .
- Add sample code if needed .
- High-Level Overviews

Don’ts

- Writing redundant comments
- Don't Overload Comments
- Don't Write Unmaintainable Documentation
- Don’t add Temporary in comments unless and until its important

E.g., Readme , Sphinx ,mkdocs ,swagger ,Gitbook

TESTING



- Use TTD and BDD methods
- Write the Minimum Code to Pass the Test
- Refactor the code base with Confidence
- Write step by step flows
- Test independent flows wisely
- Use crisp and clear name e.,g ., TestUserLoginFlow
- Focus on Unit first testing .
- Maintain consistent rule and style
- implement load testing

CONCLUSION

- Consistency is the key
- Continuous learning and improvement
- Readability == Maintainability
- Empowering Future Generations
- Always Fail and Fix

Thank You