Practical session 01

Look ma, I've published a package!

High Performance Computing for Data Science - SISSA, 2023-2024

Pasquale Claudio Africa, Konstantin Karchev

24 Apr 2024

Part 1: Setting up your development environment

1. **Install required tools**:
    - Ensure Python is installed on your system.
    - Install Git and create a GitHub account if you haven't already.
    - Install your preferred text editor or IDE (e.g., Visual Studio Code, PyCharm).
2. **Create a virtual environment**:
    - Open your terminal or command prompt.
    - Navigate to your project directory and run `python -m venv env`.
    - Activate the virtual environment: `source env/bin/activate`.
    - Install essential packages: `pip install pytest sphinx`.

Part 2: Developing the Python package

1. **Create the package structure**:
    - Inside your project directory, create a new directory for your package (e.g., `mypackage/`).
    - Inside `mypackage/`, create an `__init__.py` file and other module files (e.g., `core.py`).
2. **Write basic module code**:
    - Implement a simple function in `core.py` - for example, a function to calculate the mean of a list of numbers.

Part 3: Using Pytest for unit testing

1. **Write tests**:
    - Create a `tests/` directory.
    - Inside `tests/`, create test files starting with `test_` (e.g., `test_core.py`).
    - Write test functions to check the correctness of your package functions.
2. **Run tests**:
    - Execute the tests using pytest by running `pytest` in your terminal.

Part 4: Using Sphinx for documentation

1. **Setup Sphinx**:
    - Run `sphinx-quickstart` in your project directory.
    - Follow the prompts to set up your project (e.g., project name, author).
2. **Document your code**:
    - Use reStructuredText to write documentation in the `docs/` directory.
    - Include docstrings in your module files for automatic documentation generation.
3. **Build the documentation**:
    - Run `make html` inside the `docs/` directory to generate HTML documentation.

Part 5: Using Git for version control

1. **Initialize Git repository**:
    - Run `git init` to initialize a new Git repository.
    - Add a `.gitignore` file to exclude the `env/` directory and other non-essential files (e.g., `*.pyc`, `__pycache__/`).
2. **Commit and push your changes**:
    - Add your changes using `git add .`
    - Commit the changes with `git commit -m "Initial commit"`
    - Link your local repository to GitHub with `git remote add origin [your-repository-url]`.
    - Push your commits with `git push -u origin main`.

Part 6: Configure GitHub Actions

1. **Create GitHub Actions workflow**:
    - In your GitHub repository, create a `.github/workflows/` directory.
    - Add workflow file(s) (e.g., `python-package.yml`) to run tests across multiple Python versions.
2. **Set up actions for documentation and deployment**:
    - Add a workflow file to build documentation and push it to GitHub Pages.
    - Add a workflow file to publish your package to Test PyPI upon tagging a release.

Part 7: Finalizing and review

1. **Tag a release**:
    - After testing and final revisions, tag your release using `git tag v1.0.0`.
    - Push tags with `git push --tags`.
2. **Publish on Test PyPI**:
    - Configure a proper `pyproject.toml` file for distribution.
    - Run `python -m twine upload --repository testpypi dist/*` to upload the package.

Solution

See https://github.com/pcafrica/python_package_example/ .