Python Style and Environment Reproducibility

CWP Computing Resources:

https://github.com/aissah/cwp_coding_resources.git

Considerations for reproducible and maintainable Python code

- **Project Directory Structure**
- **Version control**
- **Documentation**
- **Testing**
- **Style**
- **Licensing**

Python Style

PEP 8 gives the coding conventions for the Python code comprising the standard library in the main Python distribution. Some style considerations included in PEP8 are:

- **Naming conventions**
- **Code layout**: *Indentation*, *Imports*, *Blank lines*, *Line length* ...
- **Docstrings**
- **Whitespaces**

Tools for Maintaining Style

There are various tools available to help you maintain a consistent style in your Python code. Some of the most popular are:

- **Linters**: tools that analyze code for potential errors and style issues.
  Some popular linters: **pylint**, **flake8**, and **pydocstyle**.
- **Formatters**: tools that automatically format your code according to some style guide.
  Some popular formatters: **black**, **autopep8**, and **yapf**.
- **Ruff** is both a linter and a formatter for Python code.

1. Use linting and formatting tools to maintain a consistent style in your Python code

Using Ruff

Ruff is easy to use and can be installed using conda/pip:

```
pip install ruff
conda install ruff
```

To lint a Python file using Ruff, you can run the command:

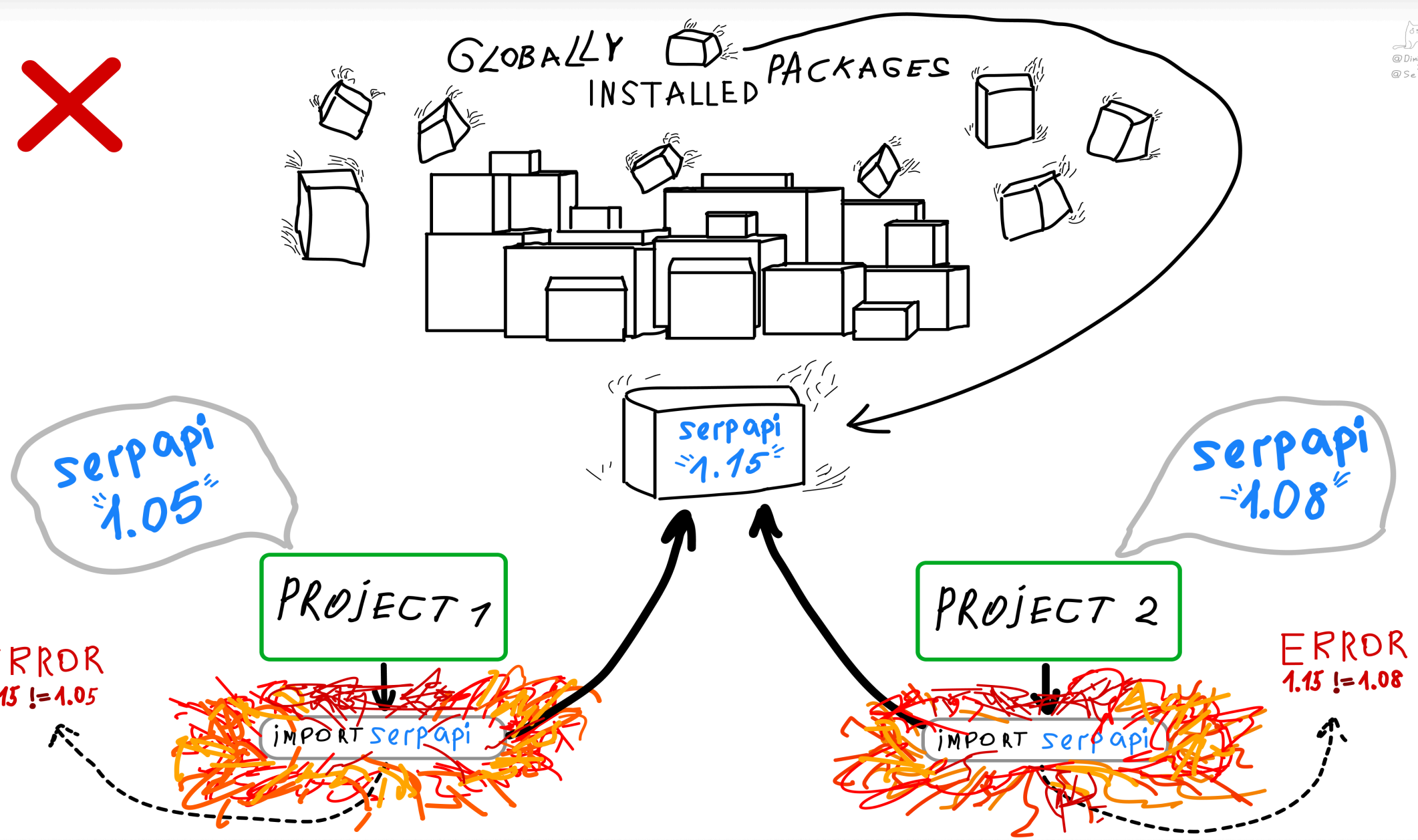
```
ruff check <filename>
```

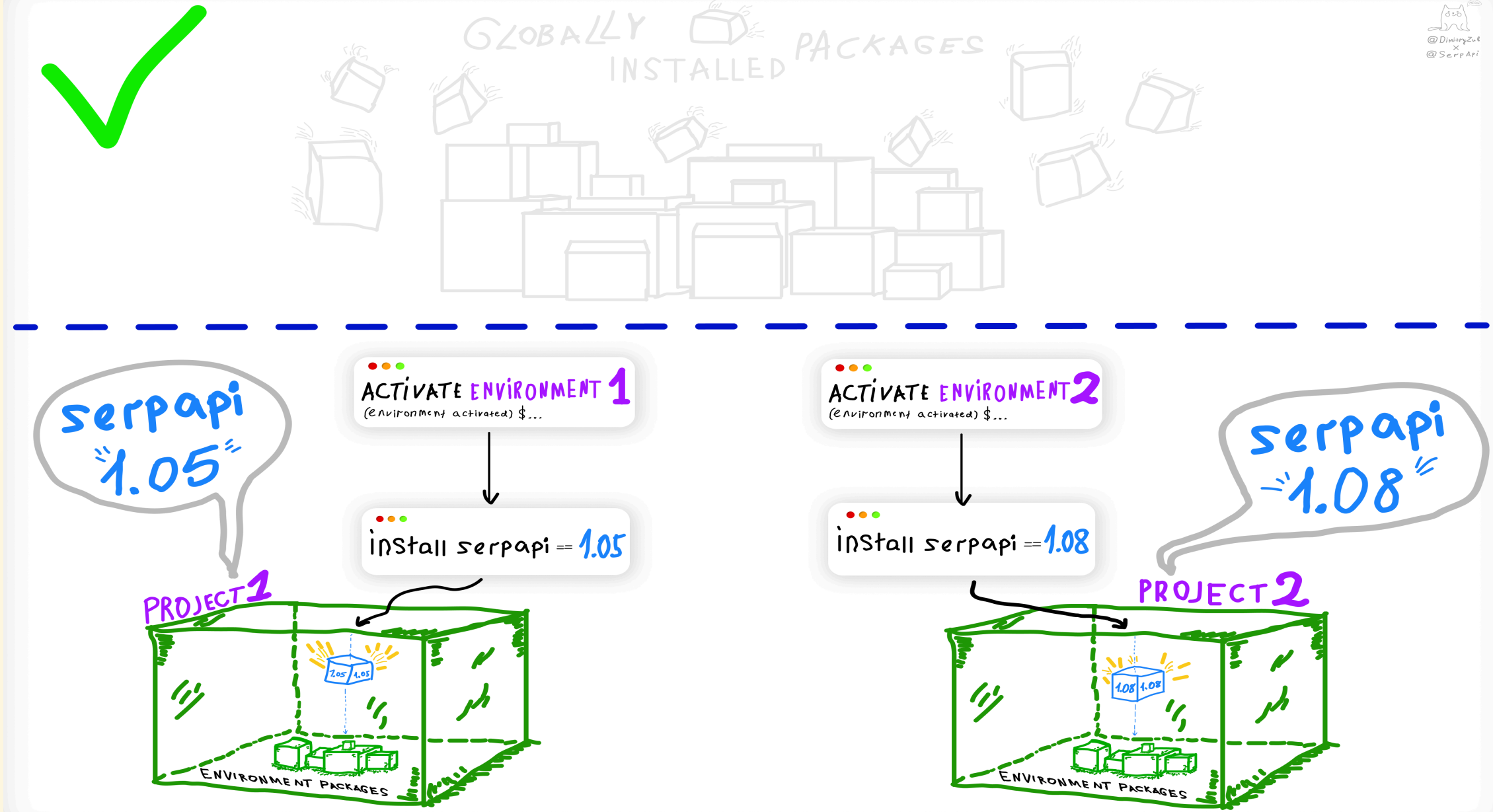
To format a Python file using Ruff, you can run the command:

```
ruff format <filename>
```

Virtual Environments

- Virtual environments are a way to create isolated environments for Python projects.
- They allow you to install packages and dependencies without affecting the system-wide Python installation.


GLOBALLY INSTALLED PACKAGES
serpapi "1.15"
serpapi "1.05"
serpapi "1.08"
PROJECT 1
PROJECT 2
import serpapi
import serpapi
ERROR
1.15 != 1.05
ERROR
1.15 != 1.08


GLOBALLY INSTALLED PACKAGES
@DimitryZub
×
@SerpApi
ACTIVATE ENVIRONMENT 1
(environment activated) $...
install serpapi == 1.05
serpapi 1.05
PROJECT 1
ENVIRONMENT PACKAGES
ACTIVATE ENVIRONMENT 2
(environment activated) $...
install serpapi == 1.08
serpapi 1.08
PROJECT 2
ENVIRONMENT PACKAGES

2. Create a new virtual environment for each project

Tools for Environment and Dependency Management

Some tools available to help you manage the environment and dependencies of a project:

- **pip and venv**
- **conda**
- **uv**
- **poetry**

Using conda

To create a new virtual environment using conda, run the command:

```
conda create --name myenv
```

To activate the virtual environment:

```
conda activate myenv
```

To install packages in the virtual environment:

```
conda install <package>
```

Environment Reproducibility

Files that can help you recreate the environment in which a piece of software was developed or tested in Python:

- **requirements.txt**: pip preferred format for specifying dependencies
- **environment.yml**: conda preferred format for specifying dependencies
- **pyproject.toml**: pep 518 compliant file for specifying dependencies and project configuration

3. Create environment files to achieve environment reproducibility

Using environment.yml and conda

To create an environment.yml file using conda, run the command:

```
conda env export --name myenv > environment.yml
```

To use the environment.yml file to recreate the environment, run the command:

```
conda env create --file environment.yml
```

Using requirements.txt and pip

To create a requirements.txt file using pip, run the command:

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

To use the requirements.txt file to recreate the environment, run one of these commands:

```
pip install -r requirements.txt
```

4. A less adopted faster way to manage virtual environments

Using uv

To start a new project using uv, run the command:

```
uv init
```

To install a package using uv, run the command:

```
uv add <package>
```

To create a requirements.txt file using uv, run the command:

```
uv pip compile pyproject.toml -o requirements.txt
```

Summary

- Use a linter and formatter (Ruff) to maintain a consistent style in your Python code
- Use virtual environments to create isolated environments for Projects
- Use requirements.txt, pyproject.toml, or environment.yml to achieve environment reproducibility in Python

Discussion

Python Style - extra

PEP 8 is a style guide for Python Code. Some of the style considerations when writing Python code included in PEP8 are:

- **Naming conventions**: Naming conventions are important in Python. They help to make the code more readable and understandable.
- **Code layout**: Code layout is important in Python. It helps to make the code more readable and understandable.
- **Indentation**: Indentation is important in Python. It helps to make the code more readable and understandable.
- **Comments**: Comments are important in Python. They help to make

Tools for Maintaining Style - extra

There are various tools available to help you maintain a consistent style in your Python code. Some of the most popular are:

- **Linters**: Linters are tools that analyze your code for potential errors and style issues. They can help you catch bugs and improve the quality of your code. Some popular linters for Python are pylint, flake8, and pydocstyle.
- **Formatters**: Formatters are tools that automatically format your code according to a specific style guide. They can help you maintain a consistent style in your code. Some popular formatters for Python are black, autopep8, and yapf.

Virtual Environment and Dependency Management

There are various tools available to help you manage the environment and dependencies of a project. Some of the most popular are:

- **pip and venv**: The Python package installer
- **poetry**: Python dependency management and packaging made easy
- **conda**: Open Source Package Management and Environment Management
- **uv**: A command-line utility for managing virtual environments