[Python Virtual Environments and Package Management] [cheatsheet]

1. Virtual Environment Creation

- Create a virtual environment using venv: `python -m venv myenv`
- Create a virtual environment using virtualenv: `virtualenv myenv`
- Create a virtual environment with a specific Python version: `virtualenv -p /usr/bin/python3.8 myenv`
- Create a virtual environment using conda: `conda create --name myenv`
- Create a virtual environment with specific packages: `conda create --name myenv numpy pandas`

2. Virtual Environment Activation

- Activate a virtual environment (Windows): `myenv\Scripts\activate`
- Activate a virtual environment (macOS/Linux): `source myenv/bin/activate`
- Activate a conda environment: `conda activate myenv`
- Activate a virtual environment using PyCharm: `File > Settings > Project > Python Interpreter > Add > Existing Environment`
- Activate a virtual environment using VS Code: `Python: Select Interpreter`

3. Virtual Environment Deactivation

- Deactivate a virtual environment: `deactivate`
- Deactivate a conda environment: `conda deactivate`

4. Package Installation

- Install a package using pip: `pip install package_name`
- Install a specific version of a package: `pip install package_name==1.0.0`
- Install packages from a requirements file: `pip install -r requirements.txt`
- Install a package using conda: `conda install package_name`
- Install a specific version of a package using conda: `conda install package_name=1.0.0`
- Install packages from a conda environment file: `conda env create -f environment.yml`
- Install a package from a local file: `pip install path/to/package.whl`
- Install a package from a URL: `pip install https://example.com/package.tar.gz`
- Install a package in editable mode: `pip install -e /path/to/package`

By: Waleed Mousa

- Install a package with extra dependencies: `pip install package_name[extra]`
- Install a package with a specific Python version: `python3.8 -m pip install package_name`
- Install a package globally: `sudo pip install package_name`

5. Package Uninstallation

- Uninstall a package using pip: `pip uninstall package_name`
- Uninstall multiple packages: `pip uninstall package1 package2 package3`
- Uninstall a package and its dependencies: `pip uninstall package_name -y`
- Uninstall a package using conda: `conda remove package_name`

6. Package Upgrade

- Upgrade a package using pip: `pip install --upgrade package_name`
- Upgrade all packages: `pip list --outdated --format=freeze | grep -v '^\-e' | cut -d = -f 1 | xargs -n1 pip install -U`
- Upgrade a package using conda: `conda update package_name`
- Upgrade all packages in a conda environment: `conda update --all`

7. Package Search

- Search for a package using pip: `pip search package_name`
- Search for a package using conda: `conda search package_name`

8. Package Information

- Show information about a package using pip: `pip show package_name`
- List all installed packages using pip: `pip list`
- List outdated packages using pip: `pip list --outdated`
- Show information about a package using conda: `conda info package_name`
- List all installed packages using conda: `conda list`
- List packages in a specific conda environment: `conda list -n myenv`

9. Requirements File

- Create a requirements file: `pip freeze > requirements.txt`
- Create a requirements file for a specific virtual environment: `pip freeze -l > requirements.txt`

- Create a requirements file with specific packages: `pip freeze | grep -i 'package1\|package2' > requirements.txt`
- Install packages from a requirements file: `pip install -r requirements.txt`
- Uninstall packages from a requirements file: `pip uninstall -r requirements.txt -y`
- Update packages in a requirements file: `pip install -r requirements.txt --upgrade`
- Check if requirements are satisfied: `pip check`

10. Conda Environment File

- Create a conda environment file: `conda env export > environment.yml`
- Create a conda environment from an environment file: `conda env create -f environment.yml`
- Update a conda environment from an environment file: `conda env update -f environment.yml`
- Remove a conda environment: `conda env remove -n myenv`
- List all conda environments: `conda env list`
- Clone a conda environment: `conda create --name myclone --clone myenv`

11. Package Version Management

- Install a specific version of a package: `pip install package_name==1.0.0`
- Install a minimum version of a package: `pip install 'package_name>=1.0.0'`
- Install a maximum version of a package: `pip install 'package_name<2.0.0'`
- Install a package with version range: `pip install 'package_name>=1.0.0,<2.0.0'`
- Install a prerelease version of a package: `pip install --pre package_name`
- Install a package from a specific source: `pip install --index-url https://example.com/simple/ package_name`
- Install a package with a specific build: `pip install 'package_name==1.0.0+123abc'`

12. Package Dependency Management

- Show package dependencies: `pip show package_name`
- List package dependencies: `pip show package_name | grep Requires`
- Show package dependents: `pip show -f package_name`
- List package dependents: `pip show -f package_name | grep Required-by`

- Install package dependencies: `pip install -r <(pip show -f package_name | grep Requires | cut -d ':' -f 2)`

13. Virtual Environment Management

- List all virtual environments: `ls -d */`
- Remove a virtual environment: `rm -rf myenv`
- Copy a virtual environment: `cp -r myenv myenv_copy`
- Rename a virtual environment: `mv myenv myenv_new`
- Create a virtual environment with a specific name: `python -m venv myproject_env`
- Create a virtual environment with a specific path: `python -m venv /path/to/myenv`
- Create a virtual environment with a specific Python version: `virtualenv -p python3.8 myenv`
- Create a virtual environment with a specific package: `virtualenv myenv && source myenv/bin/activate && pip install package_name`

14. Package Distribution

- Create a source distribution: `python setup.py sdist`
- Create a wheel distribution: `python setup.py bdist_wheel`
- Create a universal wheel distribution: `python setup.py bdist_wheel --universal`
- Upload a package to PyPI: `twine upload dist/*`
- Upload a package to TestPyPI: `twine upload --repository testpypi dist/*`
- Register a package on PyPI: `python setup.py register`
- Install a package from PyPI: `pip install package_name`
- Install a package from TestPyPI: `pip install --index-url https://test.pypi.org/simple/ package_name`

15. Package Development

- Create a new package: `mkdir mypackage && cd mypackage`
- Initialize a package: `python setup.py init`
- Create a setup.py file: `touch setup.py`
- Define package metadata in setup.py: `from setuptools import setup; setup(name='mypackage', version='1.0.0', packages=['mypackage'])`
- Create a package directory: `mkdir mypackage`
- Create an __init__.py file: `touch mypackage/__init__.py`

- Add package modules: `touch mypackage/module1.py mypackage/module2.py`
- Install package in editable mode: `pip install -e .`
- Build package distribution: `python setup.py sdist bdist_wheel`

16. Package Testing

- Run tests using unittest: `python -m unittest discover`
- Run tests using pytest: `pytest`
- Run tests with coverage: `pytest --cov=mypackage`
- Generate coverage report: `pytest --cov=mypackage --cov-report=html`
- Run tests with verbose output: `pytest -v`
- Run tests with specific markers: `pytest -m marker_name`
- Run tests with specific names: `pytest test_file.py::test_function`
- Run tests with a specific Python version: `python3.8 -m pytest`
- Run tests in parallel: `pytest -n 4`
- Run tests with a timeout: `pytest --timeout=60`

17. Package Documentation

- Generate documentation using Sphinx: `sphinx-quickstart`
- Build documentation: `sphinx-build -b html docs/source docs/build`
- Generate API documentation: `sphinx-apidoc -o docs/source mypackage`
- Serve documentation locally: `python -m http.server --directory docs/build`
- Host documentation on Read the Docs: `https://readthedocs.org/projects/mypackage/`

18. Package Continuous Integration

- Set up Travis CI: `.travis.yml`
- Set up CircleCI: `.circleci/config.yml`
- Set up Jenkins: `Jenkinsfile`
- Set up GitHub Actions: `.github/workflows/main.yml`
- Run tests on CI: `pytest`
- Build package on CI: `python setup.py sdist bdist_wheel`
- Deploy package on CI: `twine upload dist/*`

19. Package Continuous Deployment

- Deploy to PyPI on Git tag: `deploy: provider: pypi`

- Deploy to PyPI on successful build: `deploy: provider: pypi on: branch: master`
- Deploy to TestPyPI on Git tag: `deploy: provider: pypi server: https://test.pypi.org/legacy/`
- Deploy to Heroku: `git push heroku master`
- Deploy to AWS Elastic Beanstalk: `eb deploy`
- Deploy to Google Cloud Functions: `gcloud functions deploy myfunction`

20. Package Management Best Practices

- Use virtual environments for each project
- Keep dependencies up to date
- Pin package versions in requirements.txt
- Use a package manager like pip or conda
- Follow semantic versioning for package releases
- Provide comprehensive documentation
- Write unit tests for package functionality
- Use continuous integration and deployment
- Choose appropriate package distribution formats
- Publish packages to a package repository
- Monitor package usage and feedback
- Handle package deprecation and migration