ICS 214 IT Workshop III (Python)
IIIT Kottayam

Session 2 - The Development Toolkit

Project Dependencies, Virtual Environments, and Git

Instructor: Anmol Krishan Sachdeva

Tuesday | December 6, 2022

Anmol Krishan Sachdeva
Hybrid Cloud Architect, Google
MSc Advanced Computing
University of Bristol, United Kingdom
LinkedIn: greatdevaks
Twitter: @greatdevaks

- International Tech Speaker
- Distinguished Guest Lecturer
- Represented India at reputed International Hackathons
- Deep Learning Researcher
- 8+ International Publications
- Google, Microsoft, IBM, and HP Certified Professional
- ALL STACK DEVELOPER
- Mentor

Agenda

- An Introduction to ***pip*** Package Management
    - Listing
    - Installing
    - Removing
    - Upgrading
    - Searching
    - Inspecting
    - Version Pinning
- Handling Multiple Packages
- Virtual Environments
- Development Hygiene and IDE
- Other Tools
- Git Basics

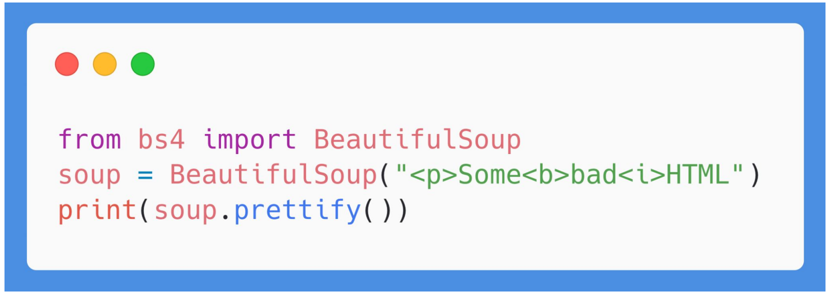
Understanding the Need for Package Management

- Programs for solving real-world problems often require Third-Party Libraries/Modules/Packages
  - ***How to reference Third-Party Libraries/Modules/Packages?***
    - ***Third-Party Module's Source Code Injection/Loading/Reference?***
- The concept of ***Distribution Packages*** makes it easy to solve the problem
  - Bundle the Python code
  - Publish it in form of a distributable *(a release; a **versioned** archive)*
    - ***Why versioned?***
      - New features, patches, bug fixes may have to be introduced to the Distribution Package
      - End-users / consumers of the Distribution Package should be able to decide on which version to use

Enter *pip*

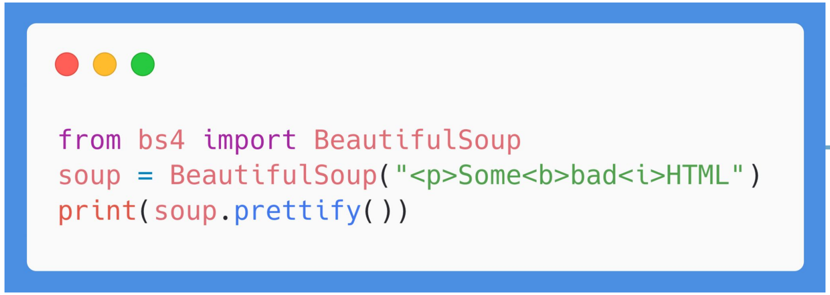
- Standard Package Manager for Python
- Helps ***install*** and ***manage*** Distribution Packages that aren't part of the ***Python Standard Library*** ***(Batteries Included Philosophy)***
- Comes pre-installed with the latest Python distributions/versions
  - If ***pip*** is not already installed on the machine, the recommended way is to use the Operating System's Package Manager (like ***apt*** for Ubuntu and ***brew*** for macOS)
  - Alternatively, ***pip-installer.org*** ***(PYPA)*** can be used to install ***pip***
- **Notes for Anaconda users**
  - Anaconda uses ***conda*** as the main package manager
  - ***pip*** also is packaged along and is supported well
  - ***conda*** is able to manage non-Python Distribution Packages as well

Example 1:
BeautifulSoup Package Import

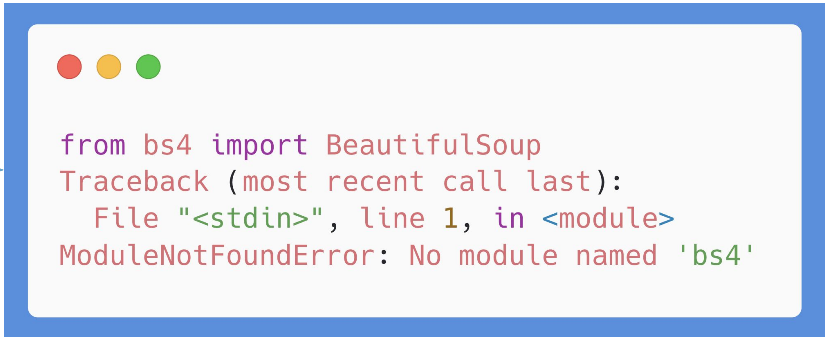


```python
from bs4 import BeautifulSoup
soup = BeautifulSoup("<p>Some<b>bad<i>HTML")
print(soup.prettify())
```

Example 1: ModuleNotFoundError



```
from bs4 import BeautifulSoup
soup = BeautifulSoup("<p>Some<b>bad<i>HTML")
print(soup.prettify())
```



```
from bs4 import BeautifulSoup
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'bs4'
```

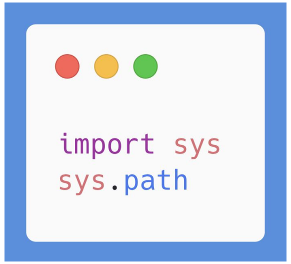
Common *pip* Commands

- Listing Packages: `` `python -m pip list` ``
- Searching Packages: [The Python Package Index (PYPI)]
- Installing Packages: `` `python -m pip install requests` ``
- Installing Specific Version of a Package: `` `python -m pip install requests==1.x.x` ``
- Uninstalling Packages: `` `python -m pip uninstall requests` ``
- Inspecting Packages: `` `python -m pip show requests` ``
- Listing Outdated Packages: `` `python -m pip list -O` ``
- Upgrading Packages: `` `python -m pip -U requests` ``

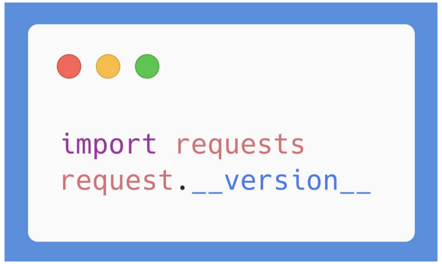
Indian Institute of Information Technology Kottayam

Handling Multiple Packages

- Installing Multiple Packages: `` `python -m pip install requests bs4` ``
- Uninstalling Multiple Packages: `` `python -m pip uninstall -y requests bs4` ``
    - **Note:** Installation of a Package takes care of installing its dependencies but uninstallation doesn't take care of dependencies
- Use **`*sys*`** package for getting information on the import paths/directories for the packages



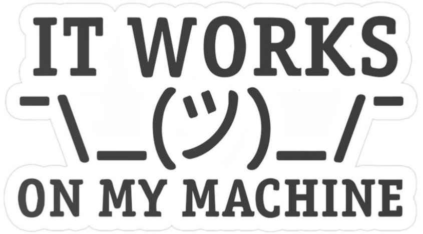
```
import sys
sys.path
```



```
import requests
request.__version__
```

Understanding the Need for Virtual Environments

- *What if multiple Python versions are present on the machine?*
- *What happens if multiple Python Programs (essentially Projects in real-world) are there?*
  - *Should all use the globally defined Distribution Packages?*
- *What if some Distribution Packages are to be restricted to specific Python Projects?*
- *What happens if these multiple Python Projects want to utilize N different Distribution Packages?*
  - *What if some Python Projects have requirement for common Distribution Packages but of different versions?*
- *What if there are X people who want to collaboratively build some Python Project?*
  - Developers may be running different Operating Systems
  - There may be different Python versions installed on everyone's machines
  - Some developers may have old Distribution Package versions running
- *What happens if multiple users are using the same machine for development?*
- *What if a Python Project needs to be tested against different package versions?*



Enter Virtual Environments

- Provides independent and isolated Python interpreter for your Python Project
- Isolated ***Python*** and ***pip*** versions can be maintained, helping maintain development hygiene by not polluting the global packages

Virtual Environment Creation

- `` `mkdir ~/<preferred_directory>/venvs` ``
- `` `cd ~/<preferred_directory>/venvs` ``
- `` `python -m venv <virtual_env_name>` ``
- `` `source <virtual_env_name>/bin/activate` ``
- `` `python -m pip install <package_name>` ``
- `` `deactivate` # For getting out of the Virtual Environment’s scope ``

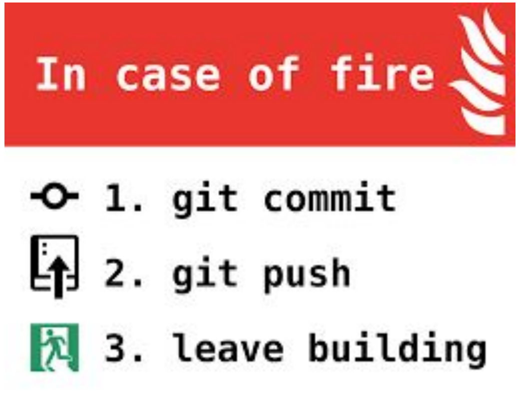
Working with Real-World Projects

- Pin the package versions when committing the code
  `` `python -m pip freeze > requirements.txt` ``
- Install the packages by referring to the requirements when collaborating on projects
  `` `python -m pip install -r requirements.txt` ``

Beyond *pip* and *venv*

- Unified package management and virtual environment concept
  - Poetry
  - Pipenv
- Capable of handling sub-dependencies for different packages requesting the same dependent package but having different versions

Git Basics: Version Control System (VCS)

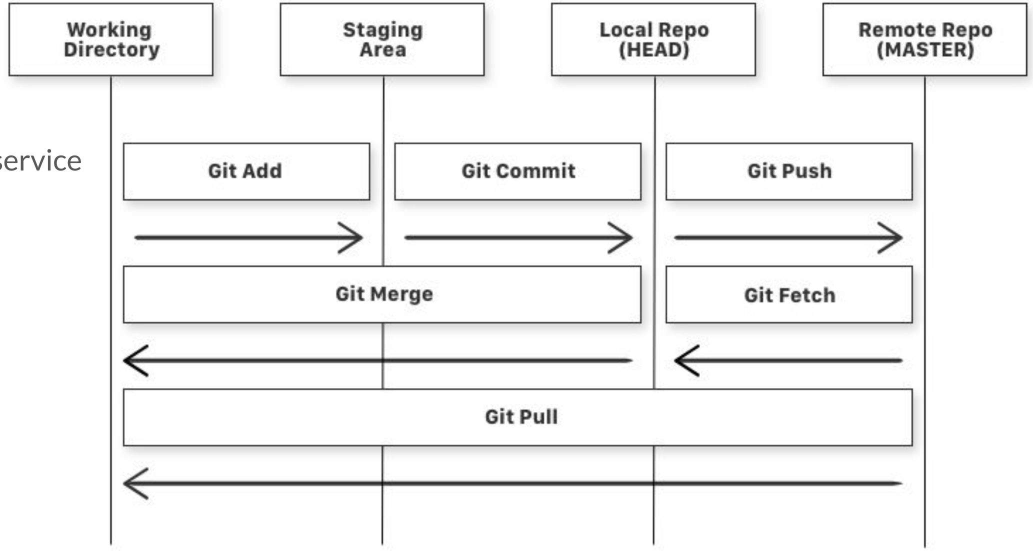


In case of fire

1. git commit
2. git push
3. leave building

Simple Git Workflow

- Git =/= GitHub
  - Git: Version Control System
  - GitHub: Cloud-based hosting service for Git repostiories
- [Git Cheat Sheet]



Summary and Key Takeaways

- Have a proper Development Environment
- Use Virtual Environments for managing project dependencies
  - Have a Virtual Environment for each Python Project
  - Maintain Virtual Environments in a dedicated directory
  - Keep Virtual Environments separate from Python Projects
- Use ***pip*** as a Python module i.e. `***python -m pip***`
- Make use of `***pip freeze***` and always maintain a `***requirements.txt***` file
- Git is essential; understand the Git Workflow

References

- [Real Python] Introduction to ***pip***
- [Real Python] Introduction to **Virtual Environments**
- Setting up Python in Visual Studio Code
- [FreeCodeCamp] Git under 10 minutes