AMS Bootcamp 2025: Bash Shell, GitHub, and Python

Outline

- BASH
- Integrated Development Environment (IDE)
- Version control
- Jupyter Notebook
- Environment manager
- Formatter/linter
- Testing
- AI coding tools

Resources

- AMS Bootcamp GitHub Repository: https://github.com/bknutson0/ams-coding-bootcamp
- Michael's bootcamp repo (https://github.com/mivanit/bash-git-bootcamp) and python projects template repo (https://github.com/mivanit/python-project-makefile-template)
- Course for scientific computing in python: https://opensourcecourse.dev/osc_intro/intro.html
- Microsoft introduction to Python: https://vscodeedu.com/courses/intro-to-python
- Mines HPC guide: https://rc-docs.mines.edu/pages/user_guides/new_user_guide.html

Part 1 - Learning Objectives

- **Bash**: Basic Command line usage
- **Integrated Development Environment (IDE)**: Work in a unified workspace
- **Version control**: Track changes to code
- **Jupyter Notebook**: Interactive computing and data visualization

BASH (Bourne-Again SHell)

Shell is an interface between the user and the operating system, allowing users to interact with the system through commands.

- Can be a:
  - Graphical User Interface (GUI) or a
  - **Command-Line Interface (CLI)**, examples include:
    - **BASH** (Linux/MacOS)
    - PowerShell, Command Prompt (Windows)
    - Zsh (Z Shell), Fish (Friendly Interactive Shell), Csh (C Shell), etc.

BASH (Bourne-Again SHell)

BASH: a CLI available on most Linux and macOS operating systems. Can be installed on Windows (e.g., via Git Bash (https://git-scm.com/))

- **Key Uses**:
  - Navigating and managing the file system
  - Running commands
  - Automating tasks with scripts
  - Using version control (e.g., `git`)
  - Remote access (e.g., `ssh`, `scp`)
  - Redirection and piping

Example of Bash commands

- `ls` - list files in the current directory
- `cd` - change directory
- `pwd` - print working directory
- `mkdir` - make a new directory
- `touch` - create a new file
- `rm` - remove a file or directory
- `cp` / `mv` - copy/move files or directories
- `cat` - concatenate and display file contents
- other commands...

Activity: Bash Commands

1. Open your terminal (Bash shell).
2. Print your current working directory.
3. Create a new directory called `ams_bootcamp`.
4. Change into the `ams_bootcamp` directory.
5. Create a new file called `hello_world.txt`.
6. Open `hello_world.txt` in a text editor (e.g., vim, nano).
7. Write "Hello, AMS Bootcamp!" in the file.
8. Save and exit the text editor.
9. List contents of `ams_bootcamp` directory to verify file was created.

Demo: Automating tasks with scripts

Run `bash_intro.py` in the terminal

Run `bash_intro.sh` to illustrate how to automate tasks with scripts.

Integrated Development Environment (IDE)

IDEs provide a unified workspace for coding, debugging, and testing.

- **Examples**: RStudio, MATLAB, PyCharm, Spyder, **VSCode**.
- **Key VSCode features**:
  - Code editor with syntax highlighting
  - Integrated terminal
  - Version control integration
  - Support for multiple programming languages
  - Installation of extensions for package management, linting, formatting, code completion, debugging, etc.

Activity: Install VSCode

1. Download and install **Visual Studio Code** from https://code.visualstudio.com/.

2. Create a new Python file (e.g., `hello.py`) and write a simple print statement: `print("Hello, AMS Bootcamp!")`

3. Open the integrated terminal in VSCode (View > Terminal).

4. Run the Python file using the integrated terminal:

```
python hello.py
```

Version Control

Version control: tracking changes to files over time.

- **Benefits**:
    - Recall specific versions later
    - Never lose code
    - Implement collaborative workflows
- A **version control system (VCS)** is a tool that helps us do this.

Git

- **Git** is a popular version control system
- **Key Git commands**:
    - `git init`: Initialize a new Git repository
    - `git add <file>`: Stage changes for commit
    - `git commit -m "message"`: Commit staged changes
    - `git branch`: List, create, or delete branches

Version Tracking with Git

Using global env height:cm

Branching and Merging

Using global env height:cm

Using global env width:cm

Activity: Git

1. Open your terminal (Bash shell).
2. Create a new directory called `ams_git_demo`.
3. Change into the `ams_git_demo` directory.
4. Initialize a new Git repository in this directory using `git init`.
5. Create a new file called `README.md` and add some content to it.
6. Stage the file using `git add README.md`.
7. Commit the changes with a message using `git commit -m "Initial commit"`.
8. Check the status of your Git repository using `git status`.

GitHub is a web-based platform for hosting Git repositories, enabling collaboration and sharing of code.

- **Common Remote Repository Commands**:
  - `git clone <repo_url>` : Clone a remote repository
  - `git push`: Push local commits to the remote repository
  - `git pull`: Fetch and merge changes from the remote repository

Using global env width:cm

Activity: Clone a GitHub Repository

- Clone the AMS Bootcamp repository (https://github.com/bknutson0/ams-coding-bootcamp) from GitHub to your local machine

- You can make changes but you cannot push them back to the original repository
- If you want to make changes, you can fork the repository to your own GitHub account and then clone that forked repository

Extra Activity: GitHub

1. Create a GitHub account if you don't have one.
2. Create a new GitHub repository (e.g., `ams-bootcamp`).
3. Clone the repository to your local machine.
4. Create a new file in the cloned repository (e.g., `README.md`) and add some content.
5. Stage and commit your changes.
6. Push your changes to the remote repository on GitHub.
7. Verify that the changes appear in your GitHub repository.

Jupyter Notebooks/Google Colab

- Interactive environments for data analysis and visualization
- Support for rich media output (e.g., plots, images)
- Easy sharing and collaboration

Activity: Open a Jupyter Notebook or Google Colab and copy and paste the content of the `bootcamp_example.py` file into various cells.

Summary

- **BASH** provides a text-based interface to interact with the operating system
- IDEs like **VSCode** offer a unified workspace for coding, debugging, and testing
- Version control with **Git** and **GitHub** allows tracking changes to code and collaboration
- **Jupyter Notebook/Google Colab** enables prototyping, interactive computing, and data visualization

Break

Take a 5 min break and stretch your legs

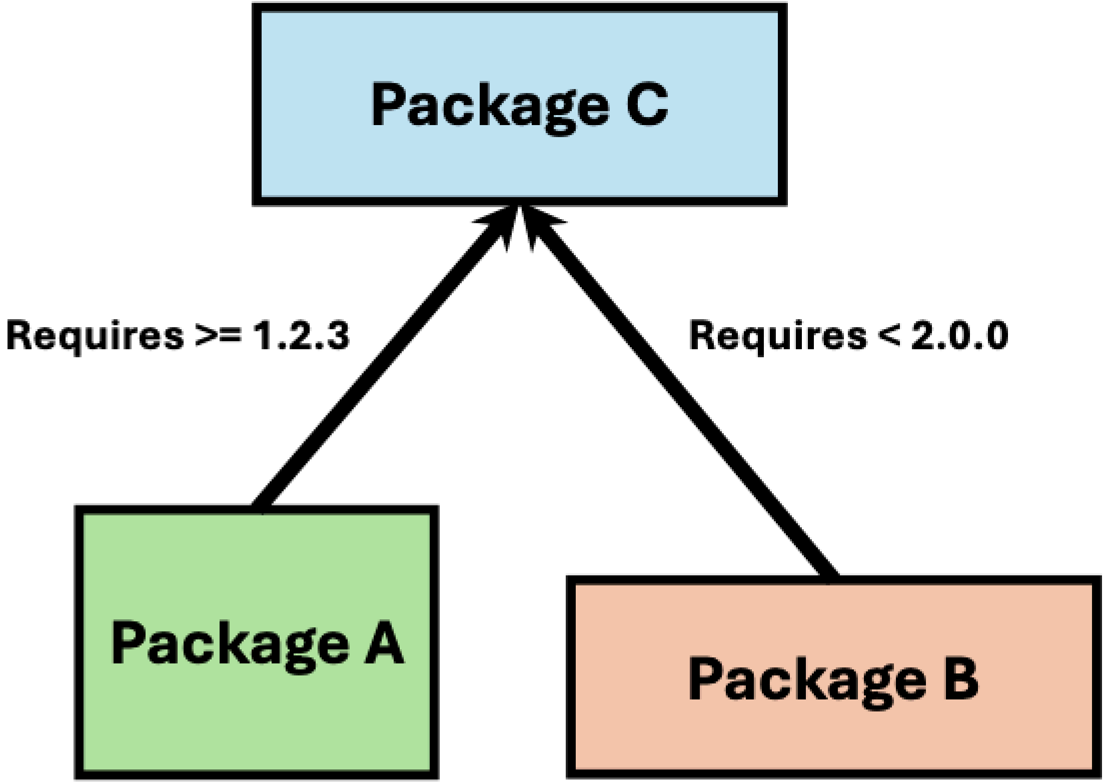
Part II

- **Environment management**: make code reproducible
- **Formatting & linting**: make code clean and consistent
- **Testing**: ensure code works as intended
- **AI coding agents**: pair code with AI

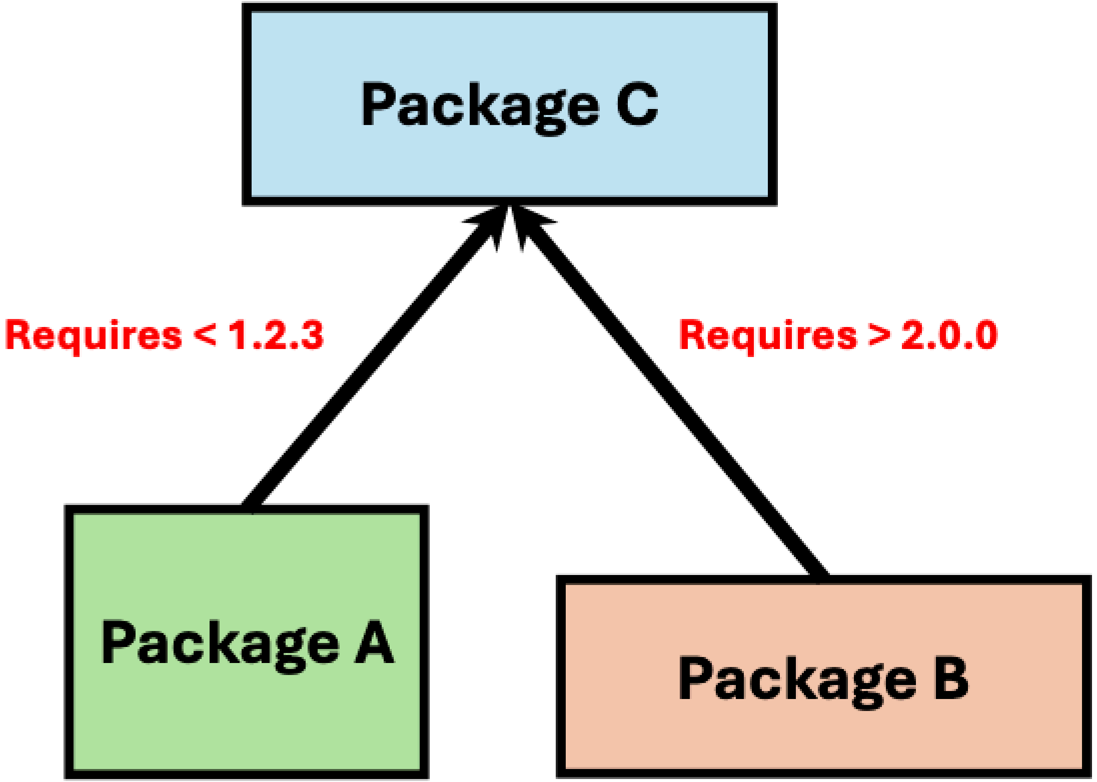
Environment management

- **Motivation:** Make code reproducible by identifying and solving dependencies
- **Examples:** `pip venv`, `conda`, `uv`

Environment management



Environment management



Environment management with `uv`

- Install `uv` [here]

- **Basic commands**:
  `uv init`: create a new project
  `uv add <package>`: add a dependency to the project
  `uv sync`: install all dependencies
  `uv run <script.py>`: run a script
  `uv help`: display help information for uv commands

- Control all `uv` settings and dependencies in `pyproject.toml`

Environment management with `uv`

Let's practice using `uv` and `pyproject.toml` to manage our environment.

Formatting

- **Motivation:** Enforce consistent style (spacing, line lengths, etc.)
- **Examples:** `isort`, `black`, `ruff`

Linting

- **Motivation:** Identify and fix potential bugs and remove logically unnecesssary code
- **Examples:** `pylint`, `flake8`, `ruff`

Formatting & linting with `ruff`

- **Format command:**

```
uv run ruff format file_to_be_formatted.py
```

- **Lint command:**

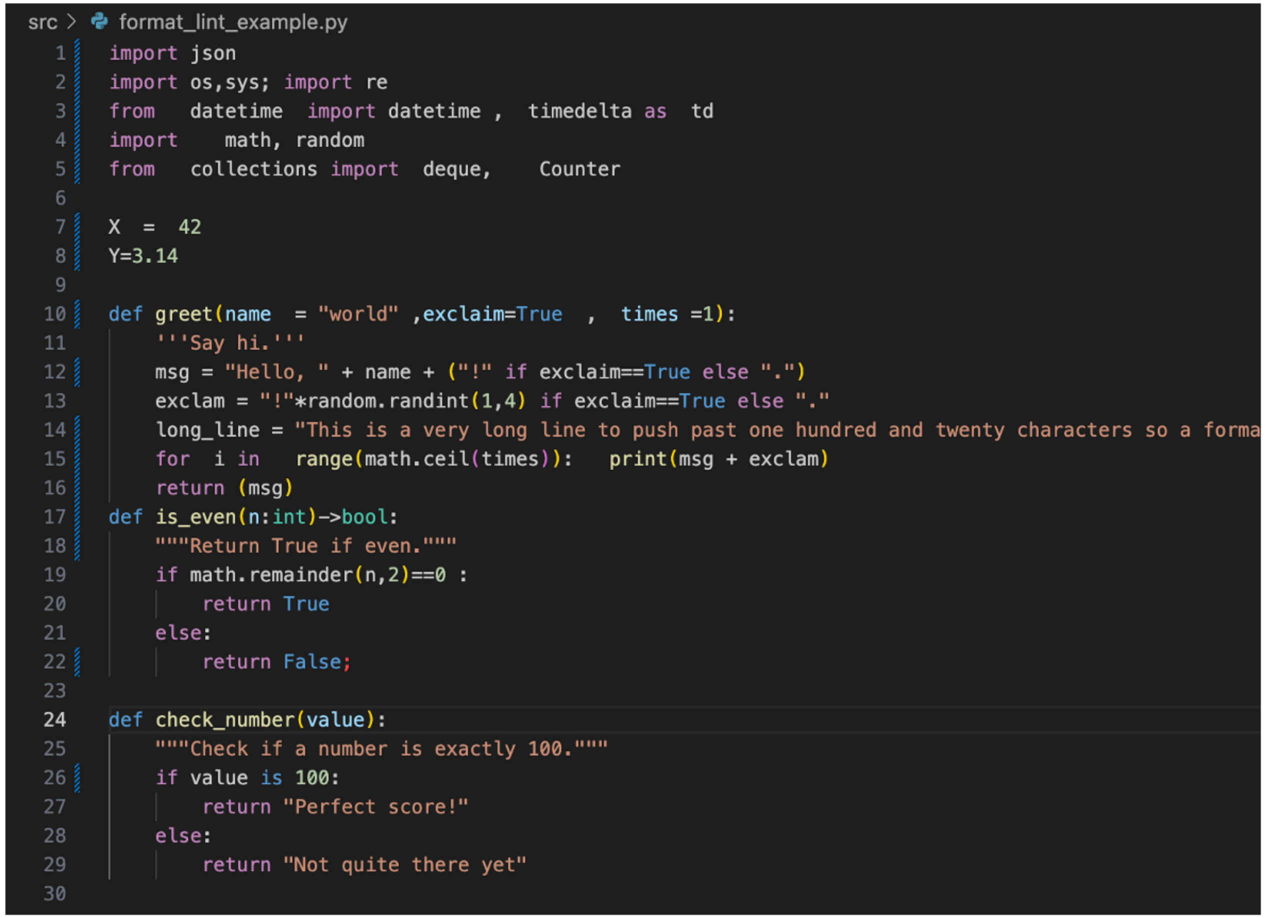
```
uv run ruff check --fix file_to_be_linted.py
```

- Or, simply use VSCode's `ruff` extension and format-on-save.
- Control all `ruff` settings (i.e. rules) in `pyproject.toml`

Formatting & linting with `ruff`

Let's practice using `ruff` and `pyproject.toml` to format and lint `src/format_lint_example.py`.

Formatting & linting with `ruff`



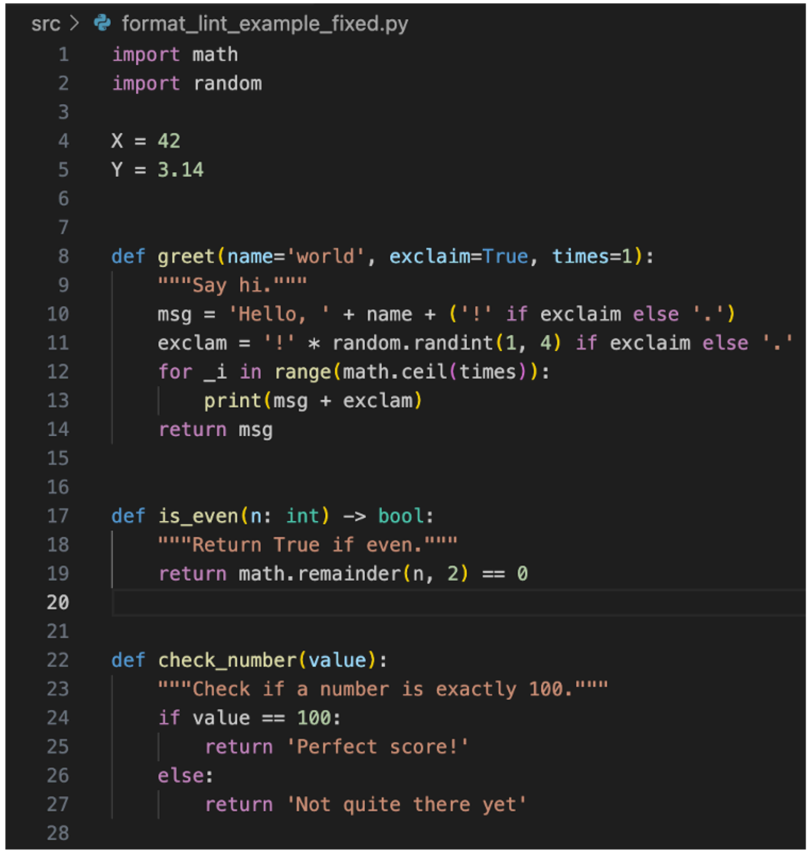
src > format_lint_example.py

```python
import json
import os,sys; import re
from   datetime  import datetime ,  timedelta as  td
import    math, random
from   collections import  deque,    Counter

X  =  42
Y=3.14

def greet(name  = "world" ,exclaim=True  ,  times =1):
    '''Say hi.'''
    msg = "Hello, " + name + ("!" if exclaim==True else ".")
    exclam = "!"*random.randint(1,4) if exclaim==True else "."
    long_line = "This is a very long line to push past one hundred and twenty characters so a forma
    for  i in   range(math.ceil(times)):   print(msg + exclam)
    return (msg)
def is_even(n:int)->bool:
    """Return True if even."""
    if math.remainder(n,2)==0 :
        return True
    else:
        return False;

def check_number(value):
    """Check if a number is exactly 100."""
    if value is 100:
        return "Perfect score!"
    else:
        return "Not quite there yet"
```



src > format_lint_example_fixed.py

```python
import math
import random

X = 42
Y = 3.14


def greet(name='world', exclaim=True, times=1):
    """Say hi."""
    msg = 'Hello, ' + name + ('!' if exclaim else '.')
    exclam = '!' * random.randint(1, 4) if exclaim else '.'
    for _i in range(math.ceil(times)):
        print(msg + exclam)
    return msg


def is_even(n: int) -> bool:
    """Return True if even."""
    return math.remainder(n, 2) == 0


def check_number(value):
    """Check if a number is exactly 100."""
    if value == 100:
        return 'Perfect score!'
    else:
        return 'Not quite there yet'
```

Testing

- **Motivation:** Ensure code is working as intended
- **Examples:** … `pytest`
- **To run tests:** save tests as `tests/test_<name>.py` then run the command `uv run pytest`

Testing

- You can create a GitHub Action workflow so that GitHub automatically runs your tests on every push and pull request
- To do so, add `.github/workflows/<name>.yml` to your repo and enable Actions in your GitHub repository settings
- See the example in this repo

AI agent

- **Motivation:** Pair coding with an agent is the way of the future
- **Examples:** OpenAI Codex, Claude Code, GitHub Copilot
- AI agents can read your repo and suggest changes based on conversation

AI agent

- GitHub Copilot is a VS Code extension with a built-in interface that lets you select different models
- As a student, you can get GitHub Copilot Pro **for free** via GitHub's Student Developer Pack, which gives you access to more advanced models

AI agent

Let's use GitHub Copilot to add a test for the `is_even()` function.

Thank you!

If you enjoyed this presentation, please consider giving a star to our GitHub repository