# Programming Course and Project

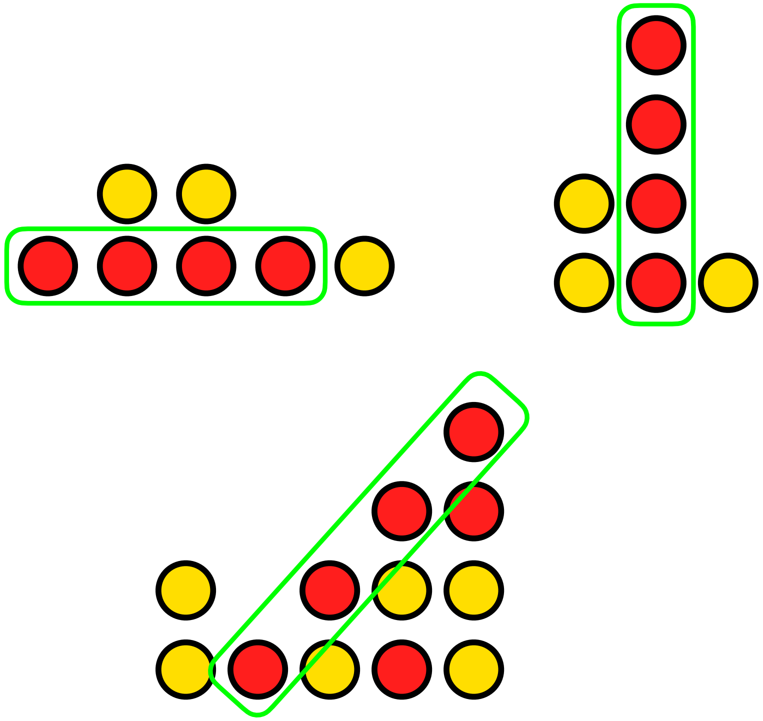
## Summer Term 2025

## Tutorial 1 - Installation, Python Project Setup (& Python/Numpy Primer)

**Felix Lundt - April 14, 2025**

# General Overview

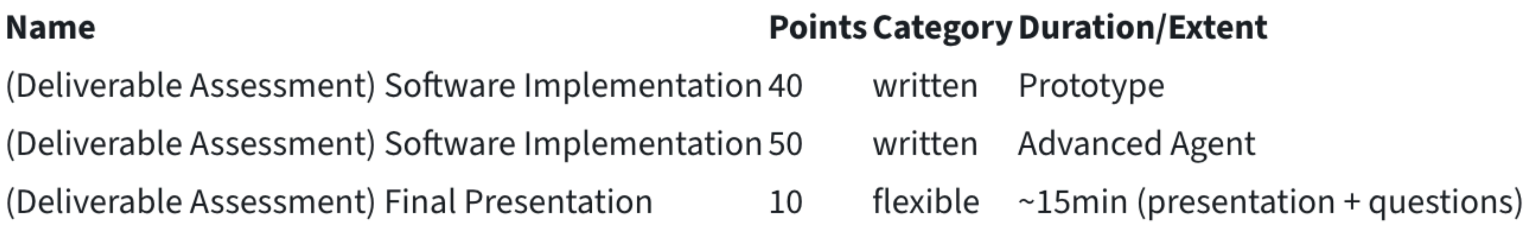
- Topic: Implement agent(s) for the game Connect 4
  - 2 players, turn-based, perfect information
  - Phase 1 (Course): Minimax/MCTS agent (roughly by end of week 5) - individual
  - Phase 2 (Project): (Almost) free choice of more sophisticated agent - groups



# General Overview

- Topic: Implement agent(s) for the game Connect 4
  - 2 players, turn-based, perfect information
  - Phase 1 (Course): Minimax/MCTS agent (roughly by end of week 5) - individual
  - Phase 2 (Project): (Almost) free choice of more sophisticated agent - groups
- Course content: Software carpentry/project management
  - Version control using Git
  - Test-driven development (TDD)
  - Documentation
  - Debugging
  - Profiling & Optimization
  - Other topics optional & on-demand
- 2 software submissions (prototype after ~5 weeks, end of semester) and final presentation
- Pass/fail

# General Overview



| Name | Points | Category | Duration/Extent |
|---|---|---|---|
| (Deliverable Assessment) Software Implementation | 40 | written | Prototype |
| (Deliverable Assessment) Software Implementation | 50 | written | Advanced Agent |
| (Deliverable Assessment) Final Presentation | 10 | flexible | ~15min (presentation + questions) |

- 2 software submissions (prototype after ~5 weeks, end of semester) and final presentation
- Pass/fail
- Exam registration until May 16th (announcement on Moodle later this week)

# How I want to run this course

- Topic: Implement agent(s) for the Game Connect 4
- Course content: Software carpentry/project management - less about programming in Python itself

- **Goal** of the course:
  Prepare you to independently and reliably execute projects (e.g., perform research). You will:
  1. Work on good programming **habits**.
  2. **Practice** executing a larger software development **project**.

- **Consequences:**
  - be aware that developing habits takes time and effort
  - pay a lot of attention to tools, read documentation, material on Moodle page, etc.

- This course is a **learning opportunity** - try to make the best of it by doing the work yourself, and avoid looking for the easy way out.

- My role is to **help** and **support** you along the way (and less to teach). I want to create an environment enabling you to ask the dumb questions, fill knowledge gaps, iron out quirks in workflows, explore, take risks. In turn I ask you: see above.

- Tutorials (Mondays, 10-14):
  - teaching component/opportunity to discuss things of general interest
  - time to simultaneously work on the project, discuss, get help/feedback from me and other students

# Tentative outline for the first phase

| | **Content Software Carpentry** | **Algorithm/ Game Play** | **General** |
|---|---|---|---|
| **Week 1**<br>**April 14** | Project Setup | | Intro,<br>Python & Numpy primer |
| **Week 2**<br>**April 28** | TDD | Code skeleton | |
| **Week 3**<br>**May 5** | Git<br>Modules & Packages | Code to play Random Agent Algorithms | |
| **Week 4**<br>**May 12** | Debugging & Documentation | | Submission Prototype? |
| **Week 5**<br>**May 19** | Profiling | | Submission Prototype? |

# What this course can offer you

Topic:

- Game playing performance as hallmark of AI
- Example: chess
  - Stockfish: Minimax with Neural Network (since 2020)
  - AlphaZero/Leela Chess Zero: Monte Carlo Tree Search with Neural Network (variant of RL)
- Complexity of Connect 4 well-suited for our purposes

Course:

- Sound/well-defined starting point, but flexible & expandable later on
- Room to play around/explore
- Learn to built software independently in controlled environment
- Result is usable (to play against) and adaptable for other games

# What qualities are we looking for in our code?

- Assumption: You will write custom code as means to an end (e.g., scientific research, providing a service, offering a product)
- How should this tool be used, or how should users be able to use it (including yourself)? What will you want to **do** with your code?
  - Reliable results/Correct code
  - Efficient development/Implementation
  - Share/Publish code
  - Maintain/Update/Improve code
  - Easy and intuitive use
- **Critical for me: Flexible/adaptable/expandable code → allows for iterative approach, rather than managing the full complexity at all times**
- Correctness is essential, and so tests are as well

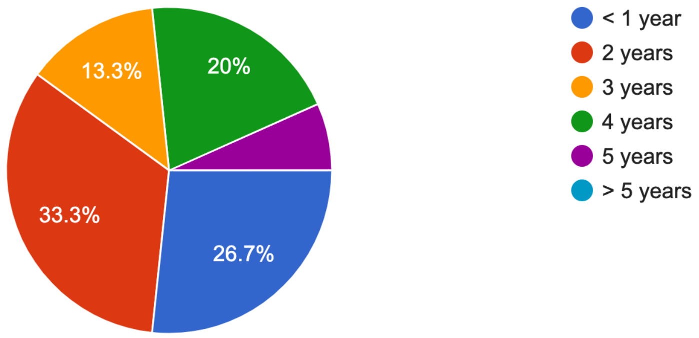
- How do we tell if code is 'good'?
  - It works!
  - Easy to Read
  - (Easily) Testable
  - Efficient
  - Easy to change
  - Simple

# Goals for today

- Get everyone ready
  - Set up IDE
  - Set up project
  - Write/run scripts
  - Go over intro material

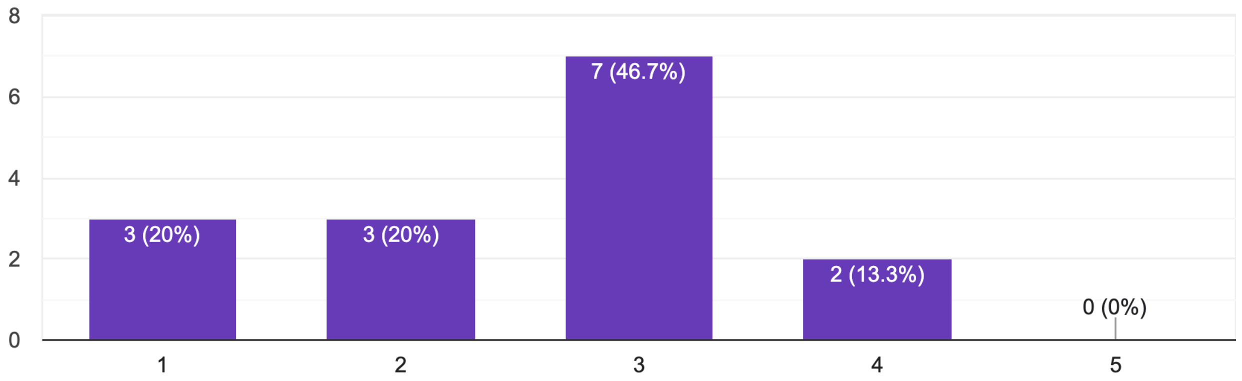
How many years of programming experience do you have?

15 responses



How familiar are you with Python?

15 responses



# Project Setup

- Good organization can set you up for success
- Helps with maintainability, reproducibility and collaboration
  - Project structure
  - Virtual environments
  - Dependency management
  - Development tools (IDE, linting, formatting, ..)
  - Testing
  - Documentation
  - Version Control

# Project structure

Publishable package

```
project_name/
├── .venv/                  # Virtual environment (not in version control)
├── src/                    # Source code
│   └── project_name/      # Main package directory
├── tests/                 # Test files
├── docs/                  # Documentation
├── pyproject.toml        # Project metadata and build configuration
├── README.md             # Project documentation
└── .gitignore           # Git ignore file
```

App development

```
app_name/
├── .venv/                # Virtual environment
├── app/                  # Application code
│   ├── __init__.py
│   ├── main.py          # Entry point
│   ├── config.py        # Configuration
│   ├── models/          # Data models
│   ├── views/           # UI components
│   └── utils/           # Utility functions
├── tests/               # Test files
├── static/              # Static assets
├── templates/           # Template files
├── data/               # Application data
├── requirements.txt    # Dependencies
├── README.md          # Documentation
└── .gitignore        # Git ignore file
```

# Project Structure

Scientific Project

```
project_name/
├── .venv/              # Virtual environment
├── notebooks/          # Jupyter notebooks
├── src/               # Source code
│   ├── data/         # Data processing scripts
│   ├── features/     # Feature engineering
│   ├── models/       # Model definitions
│   └── visualization/# Visualization code
├── data/             # Data directory
│   ├── raw/         # Original data
│   ├── processed/   # Processed data
│   └── external/    # External data sources
├── models/          # Trained models
├── reports/         # Generated reports
│   ├── figures/    # Generated figures
│   └── results/    # Analysis results
├── tests/           # Test files
├── requirements.txt # Dependencies
├── README.md       # Documentation
└── .gitignore     # Git ignore file
```

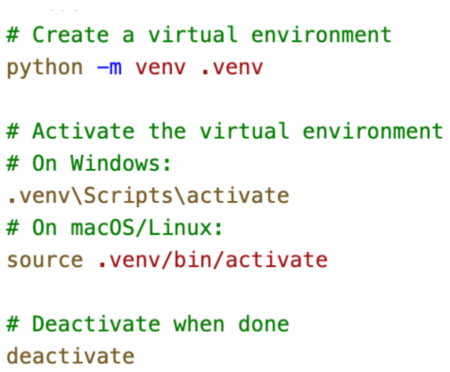
Key Principles:

- Separation of Concerns
- Configuration Management
- Data Management
- Testing Organization
- Documentation

# Virtual Environments

- Virtual environments are isolated Python environments that help with managing project-specific dependencies (Python version, packages)

- Venv

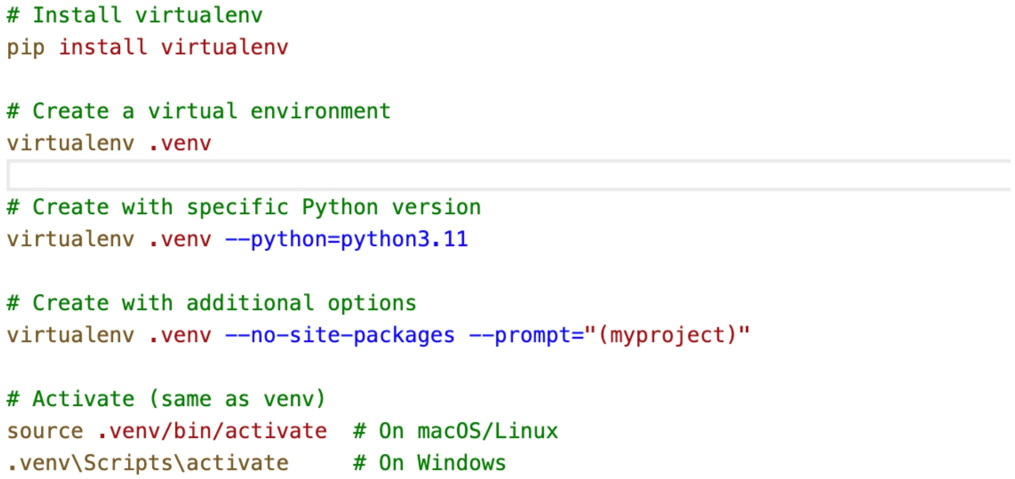


```
# Create a virtual environment
python -m venv .venv

# Activate the virtual environment
# On Windows:
.venv\Scripts\activate
# On macOS/Linux:
source .venv/bin/activate

# Deactivate when done
deactivate
```

- Virtualenv



```
# Install virtualenv
pip install virtualenv

# Create a virtual environment
virtualenv .venv

# Create with specific Python version
virtualenv .venv --python=python3.11

# Create with additional options
virtualenv .venv --no-site-packages --prompt="(myproject)"

# Activate (same as venv)
source .venv/bin/activate  # On macOS/Linux
.venv\Scripts\activate     # On Windows
```

- Other option: Containers

# Dependency management

- Simplest option: by hand

```
# Install dependencies
pip install -r requirements.txt

# Install development dependencies
pip install -r requirements-dev.txt

# Generate requirements file
pip freeze > requirements.txt
```

- Uv (integration with virtual environments)

```
# Install uv
pip install uv

# Create a virtual environment and install dependencies
uv venv
uv pip install -r requirements.txt

# Install a package
uv pip install requests
```

- Conda (integration with virtual environments)

```
# Create a new conda environment
conda create --name myenv python=3.11

# Activate the environment
conda activate myenv

# Install packages
conda install numpy pandas scipy

# Export environment
conda env export > environment.yml

# Create environment from file
conda env create -f environment.yml
```

**Another popular option: Poetry**