Lecture 00: Course Introduction + Python Warmup

EECS 291: Applied Data Structures & Algorithms

Instructor: Marcus M. Darden
Lectures: Tue/Thu 3:00-4:30 pm, 1670 Beyster
Labs: Fri 10:30-12:30 pm, 1303 EECS

Today

- Half 1: Course structure + policies
- Half 2: Python warmup + exercises

How today works

- 80 minutes split in half
- Short break at the midpoint
- One final commit at the end of class

Half 1: Course Structure + Policies

Goal: Know how this course runs and what success looks like.

Course focus (applied, data-first)

- Practical data structures + algorithms in Python
- Emphasis on real datasets and workflows
- Less theory than CS-major track; still rigorous

Prereqs + audience

- **Prereqs:** EECS 280 **or** EECS 402 (EECS 203 not required)
- Intended for **CS minors** and **DS grad students**
- CS majors may take the course, but it does **not** count toward the major

Course structure (four quarters)

Each quarter follows: **2 labs → 1 project → 1 exam**

- Q1: contiguous data structures + hashing intro
- Q2: sorting + priority queues
- Q3: trees + graphs
- Q4: graph algorithms + optimization

Deliverables + grading

- **Exams:** 4 × 10% = 40%
- **Labs:** 8 × 2.5% = 20%
- **Projects:** 4 × 10% = 40%

Minimum competency (must meet all three):

- ≥ 55% average on projects
- ≥ 50% average on exams (after curve)
- ≥ 75% average on labs

Labs (how they work)

- Worksheet-driven; collaboration on concepts OK
- Each student submits their own work
- Late submissions accepted at **50% credit**
- All lab pieces due **11:59 pm the day before each exam**
- No lab drops

Projects (how they work)

- Two-week projects; steady progress expected
- Autograder submissions only
- **Two total late days** for the semester
- Final grading run uses hidden tests; best pre-deadline score counts

Exams (how they work)

- Administered in PrairieLearn at the CBTF
- Closed-book; **one notes sheet** allowed
- Released exam days in Q1–Q3 (no regular lecture)
- Fourth exam during Registrar final exam period (non-cumulative)

Academic integrity + AI policy (summary)

- Honor Code applies to all work
- Collaboration: discuss concepts, **do not share code**
- AI use: **minimal assistance only**, no generated code in submissions

Course tools + environment

- Python 3.13+
- Jupyter notebooks for lecture notes
- PrairieLearn + Canvas for assessments
- Standard library only in the first half of the course

Where to get help

- Office hours and discussion forum
- Email: **eecs291admin@umich.edu** for scheduling conflicts
- If all OH conflict, email to coordinate

Break (3 minutes)

Stretch, refill, and be back at :__

Half 2: Python Warmup (Applied)

Goal: Refresh core Python syntax and workflow for students coming from CS2.

Segment 1: Functions + conditionals + loops

Goal: Read and write basic control flow quickly.

- Whitespace matters!
- Use a colon and indent to begin a block
  - `if value == 5:`
  - `while day < saturday:`
  - `for item in aList:`
  - `for i in range(10):`
  - `def functionName(<parameter list>):`
- Outdent to end a block
- Conditionals **AND** Loops can use `else:`
- Multiple conditionals use `elif <expression>:`

In [ ]:
```python
def clip_points(points, cap):
    total = 0
    for p in points:
        if p > cap:
            total += cap
        else:
            total += p
    return total

print(clip_points([24, 12, 35], 30))
```

Exercise 1: Syntax warmup

Task: Write a function `above_threshold(values, t)` that returns a list of values greater than `t`.

- Use a loop (no list comprehension yet).
- Test with a short list.

In [ ]:
```
# TODO: implement above_threshold
# def above_threshold(values, t):
#     ...

# print(above_threshold([10, 25, 5, 30], 12))
```

Segment 2: Imports + modules

Goal: Use Python modules to keep code organized.

In [ ]:
```
import math

print(math.sqrt(81))

from statistics import mean
print(mean([10, 20, 30]))
```

Exercise 2: Imports

Task: Import `statistics` and compute `median` and `pstdev` for a list.

Multiple syntaxes for `import`

- `import math` access objects with dot (.) like `math.sqrt()`
- `import reallyLongModuleName as rlmn` module aliasing, common shorthands
- `from math import pi, sqrt` no need for dots... `sqrt()` works
- `from math import sqrt as square_root` object aliasing
- `from math import *` a **BAD IDEA**

In [ ]:
```
# TODO: import statistics and compute median/pstdev
# values = [3, 7, 9, 9, 10]
# print(...)
```

Segment 3: Classes

Goal: Define a small class to store data and compute a metric.

- "Dunder" methods use double underscores
- Reserved by system
- `__init__` is a "Constructor"
- `__repr__` and `__str__` are used for display

In [ ]:
```
class PlayerStat:
    def __init__(self, player_id, points):
        self.player_id = player_id
        self.points = points

    def add_points(self, more):
        self.points += more

    def __repr__(self):
        return f"PlayerStat(id={self.player_id}, points={self.points})"

stat = PlayerStat(101, 24)
stat.add_points(12)
print(stat)
```

Exercise 3: Classes

Task: Add a method `impact()` that returns points * 1.2. Print the result.

In [ ]:
```
# TODO: add impact() to PlayerStat and print the result
# print(stat.impact())
```

Segment 4: Module runner

Goal: Understand `if __name__ == "__main__"` for scripts.

- When loaded, a script executes from top to bottom
  - Functions are defined
  - Classes are defined
  - Any global statements are executed
  - No mandatory `main()` like C/C++
- Using a script as both a module and an executable requires control
- When module `impact.py` is imported `__name__ == 'impact'`
- When module `impact.py` is executed `__name__ == 'main'`

In [ ]:
```
def main():
    print("Running analysis...")
```

```
if __name__ == "__main__":
    main()
```

Exercise 4: Module runner

Task: Add a `main()` that calls your `above_threshold` function with sample data.

In [ ]:
```
# TODO: create a main() and call above_threshold
# if __name__ == "__main__":
#     main()
```

Segment 5: Virtual environments (when stdlib is not enough)

Goal: Know how to install third-party libraries safely.

Why venv?

- Keeps project dependencies isolated
- Avoids global package conflicts
- Allows Python version selection (eg. Python 3.11, Python 3.13, etc.)

Basic flow (terminal):

```
python -m venv .venv
source .venv/bin/activate
python -m pip install pytest
```
With the exception of Pytest, we will use stdlib in the first half of the course.

Exercise 5: Venv + imports (concept check)

Task: In one sentence, explain when you would create a venv for this course.

- Write the sentence in your notes cell below.

In [ ]:
```
# TODO: write your sentence here
# answer = "..."
# print(answer)
```

Final commit (end of class)

Commit message: `lecture00-python-warmup`

- Save your notebook with completed exercises
- Push before you leave

Wrap + next steps

- Next time: stdlib data workflow
- Bring questions about setup and tooling
- Lab 00 tomorrow is optional but great for setup and preparation