Decisions, decisions

Announcements

- Quiz 1 on Friday
  - Study guide on course website
  - Covers everything up to basic for loops
  - No accumulators, indexing, or slicing
  - Ninja session Weds. 7-10pm for quiz review
- Lab 2 due Saturday before midnight
  - Does cover accumulators, indexing, slicing

Today's plan

- Academic Integrity
- Go over `roster.py`
    - Review for loops, indexing, slicing, string formatting
- Programs that "make decisions"
- Boolean type
- Comparison operators
- `if` statements, `if-else` statements

Academic Integrity

cs.swarthmore.edu/~mauskop/cs21/s17/index.php#integrity

Don’t cheat!

- “Because plagiarism is considered to be so serious a transgression, it is the opinion of the faculty that for the first offense, failure in the course and, as appropriate, suspension for a semester or deprivation of the degree in that year is suitable; for a second offense, the penalty should normally be expulsion.”
- TL;DR - First offense: fail the course, second offense: expulsion

Don’t cheat!

- Cheating in Intro CS courses is common here and around the world.
- Why?
  - We encourage collaboration, but the line between collaboration and cheating can be fuzzy
  - It’s easy to detect cheating

We encourage collaboration

- …But there’s a line
- Do talk to your classmates about concepts
- Do go over code from textbook or lecture together
- Do not look at or allow someone to look at lab code
- Same goes for Google and people not in the class

How to avoid cheating

- Start labs early so you have time to get help
- Go over non-lab programs with your classmates
- Turn, step, or walk away from computers when working with classmates

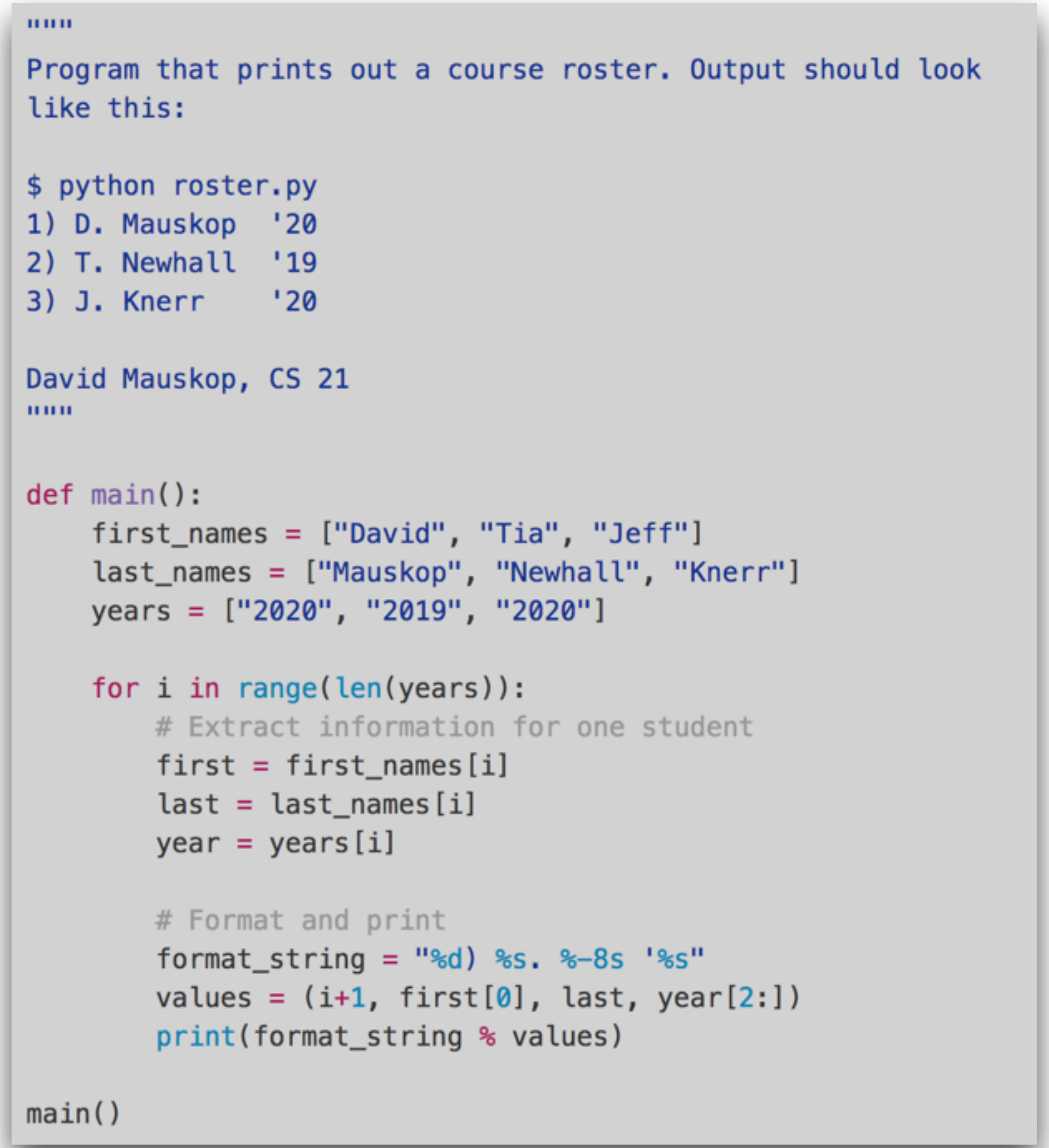
It’s easy to catch cheaters

- We run a program that does it for us
- Changing variable names doesn’t fool the program
- Even if you could fool the program, you shouldn’t cheat—that’s the ‘integrity’ part.

My challenge to you

- Prioritize learning and personal growth over getting a good grade. The good grade will come naturally.
- Besides, cheating will likely hurt your quiz and exam grades.

Back to roster.py



```python
"""
Program that prints out a course roster. Output should look
like this:

$ python roster.py
1) D. Mauskop  '20
2) T. Newhall  '19
3) J. Knerr    '20

David Mauskop, CS 21
"""

def main():
    first_names = ["David", "Tia", "Jeff"]
    last_names = ["Mauskop", "Newhall", "Knerr"]
    years = ["2020", "2019", "2020"]

    for i in range(len(years)):
        # Extract information for one student
        first = first_names[i]
        last = last_names[i]
        year = years[i]

        # Format and print
        format_string = "%d) %s. %-8s '%s"
        values = (i+1, first[0], last, year[2:])
        print(format_string % values)

main()
```

Programs that “make decisions”

- Or are imbued by programmers with the appearance of decision-making abilities
- Amazon: decide what book to recommend
- Google translate: decide on a translation of ‘mi casa es su casa’
- Digital camera: decide whether that’s a face
- Chess AI: decide on next move

Basic decision making

- In programs, we start with decisions based on yes-no questions
- if-then decisions:
  - “Is it raining? If yes, then carry an umbrella.”
  - “Did the user click the ‘like’ button? If yes, register a ‘like’.”
- if-then-otherwise decisions:
  - ‘Is it hot? If yes, wear shorts. If no, wear pants.’
  - ‘The user pressed ‘j’. Are we in normal mode? If yes, move the cursor up. If no, type the letter ‘j’.

Boolean: True/False

- Represents the answer to a yes-no question
- Data type with only two possible values:
    - `True` / ‘Yes’
    - `False` / ‘No’
- Named after logician George Boole
- It’s a bit like a bit

Comparison Operators

- Equal, not equal: `==`, `!=`
- Inequalities: `<`, `>`, `<=`, `>=`
  - Compare ints and floats based on numeric order
  - Compare strings based on alphabetic order (almost)
- These are like yes-no questions, evaluate to a Boolean

if statement

- Syntax:

```
if <boolean expression>:

    <block A>
```

- Semantics: If the boolean expression evaluates to `True`, then perform the instructions in `<block A>` before continuing to next unindented instruction. If the boolean expression evaluates to `False`, continue immediately to the next unindented instruction.

if-else statement

- Syntax:

```
if <boolean expression>:
   <block A>
else:
   <block B>
```

- Semantics: if the boolean expression evaluates to `True`, perform the instructions in `<block A>`. If not, perform instructions in `<block B>`. Then continue with the next unindented instruction.

Example

Expanded notion of accumulator

- Before: *Combine the values in a sequence into a single value.*

- Now: *Keep track of some quantity that may be updated multiple times within a program. Usually the update happens within a loop.*

- Note: for loops can be used for many things besides accumulation.

Next time: more logic