Why isn't there more CS in high schools?

- 24,000 – 30,000 high schools in US
- 2,000 teachers of AP CS, not distributed evenly
- Why?
  1. Curricula are hard to change
  2. Isn't part of "Common Core" (often doesn't count towards graduation)
  3. Feedback loop (no reason to offer CS because students don't want, students don't want because don't have access)
  4. CS often classified as "business" or "vocational," not math or science
- Also problem in UK

http://cacm.acm.org/blogs/blog-cacm/156531-why-isnt-there-more-computer-science-in-us-high-schools/fulltext

- Better example for __name__
- Upcoming project
  - About 3 times as "big" as a lab
  - Encouraged to work together
  - Open-ended
    - Data file manipulation, graphic presentation
    - Simple video game
    - ???

What's been most useful to you

- Labs: 12
- Matt coding in Python during class: 6
- In-class exercises

What was the least useful to you?

- "Class Time. It's not you I just think the material is kind of dry"

What could students do to improve the class?

- More class participation: 5
- Read the textbook more: 5
- More practice outside class
- Bring in major-specific topics
- Try to learn minor details outside of class on own
- Bring in laptop to class
- Practice writing on paper

What could Matt do to improve the class?

- More exercises assigned (not graded): 6
- Provide answers to (assigned) questions: 4
- Review Labs & provide answers

- More hands-on in lecture: 5

- Make labs clearer: 4
- Don't make corrections to questions during exam

What could Matt do to improve the class? Part 2

- Show off project's Matt's done to raise class interest
- More interdisciplinary stuff
- More assignments with graphics

- More examples of python
- More general review
- Go a bit slower
- Could be faster

- Add more color to slides
- Have slides up by lab
- Make sure python shell is in top 3/4 of screen

How many hours per week do you spend on the course *outside* of class/lab

Median: 2.5

Mean / std dev: 2.3 +/- 1.5

Min: 0

Max: 6.5

Multi-Way Decisions

```
if <condition1>:
   <case1 statements>
elif <condition2>:
   <case2 statements>
elif <condition3>:
   <case3 statements>
…
else:
   <default statements>
```

Diagramming multi-way decisions

- All geeks should be CS majors. Non-geeks should be CS majors unless they are awful at programming. But even if they're awful at programming, as long as they're good at math, they should still be CS majors.

- Diagram?
- Python code?

Diagram for a Leap Year?

- A year is a leap year if it is divisible by 4, unless it is a century year that is not divisible by 400. (1800 and 1900 are not leap years while 1600 and 2000 are.)

Study in Design: Max of Three

```
def main():
    x1, x2, x3 = eval(input("Please enter three values: "))

    # missing code sets max to the value of the largest

    print("The largest value is", max)
```

Strategy 1: Compare Each to All

- Looks like a three-way decision, where we need to execute *one* of the following:

```
max = x1
max = x2
max = x3
```

- All we need to do now is preface each one of these with the right condition

Strategy 1: Compare Each to All

- x1 is the largest:
- ```
  if x1 >= x2 >= x3:
      max = x1
  ```
- Is this syntactically correct?
  - Many languages would not allow this *compound condition*
  - Python does allow it, though: equivalent to x1 ≥ x2 ≥ x3
  ```

Strategy 1: Compare Each to All

- Whenever you write a decision, there are two crucial questions:
  - When condition is true, is executing the body of decision the right action to take?
    - x1 is at least as large as x2 and x3, so assigning max to x1 is OK
    - Always pay attention to borderline values

Strategy 1: Compare Each to All

- Secondly, ask the converse of the first question, namely, are we certain that this condition is never false when x1 is the max?
  - Suppose the values are 5, 2, and 4.
  - x1 is largest, but does x1 ≥ x2 ≥ x3 doesn't hold?
  - Don't really care about relative ordering of x2 and x3
  - Make two separate tests: x1 >= x2 *and* x1 >= x3

Strategy 1: Compare Each to All

- We can separate these conditions with and

```
if ???:
    max = x1
elif ???:
    max = x2
else:
    ???
```

- Compare each possible value against all the others to determine largest

Strategy 1: Compare Each to All

- We can separate these conditions with and

```
if x1 >= x2 and x1 >= x3:
    max = x1
elif x2 >= x1 and x2 >= x3:
    max = x2
else:
    max = x3
```

- Compare each possible value against all the others to determine largest

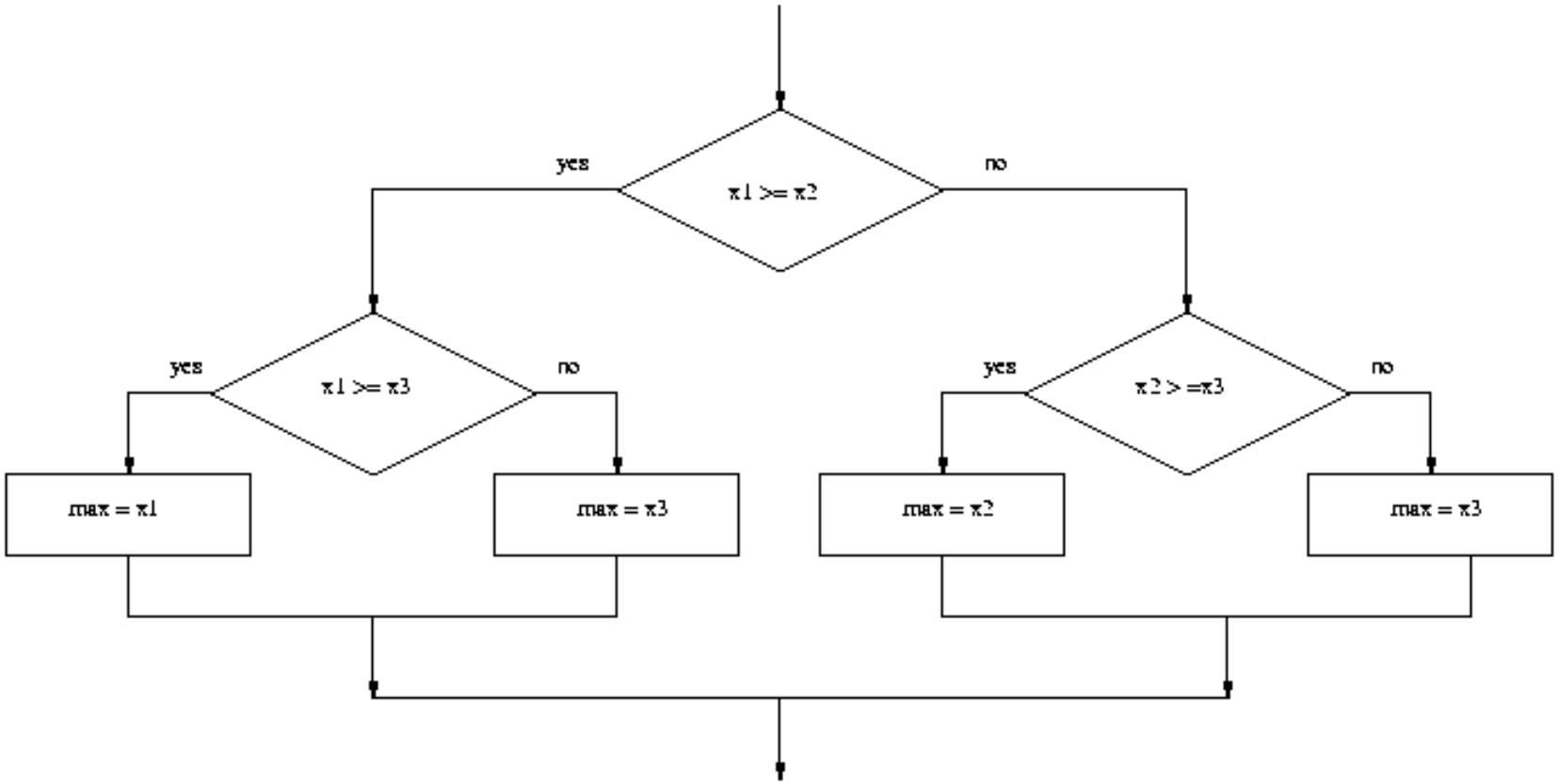
Strategy 1: Compare Each to All

- What would happen if we were trying to find the max of five values?
- We would need four Boolean expressions, each consisting of four conditions *and*ed together
- Sounds like a mess….

Strategy 2: Decision Tree

- Avoid the redundant tests of the previous algorithm using a *decision tree* approach
- Start with `x1 >= x2`
  - Either `x1` or `x2` is eliminated
- If condition is true, need to see which is larger, `x1` or `x3`

Strategy 2: Decision Tree



Strategy 2: Decision Tree

```
if x1 >= x2:
    if x1 >= x3:
        max = x1
    else:
        max = x3
else:
    if x2 >= x3:
        max = x2
    else
        max = x3
```

Strategy 2: Decision Tree

- Exactly two comparisons, regardless of ordering of the original three variables
- However, this approach is more complicated than the first
- What happens with max of four values?

Strategy 2: Decision Tree

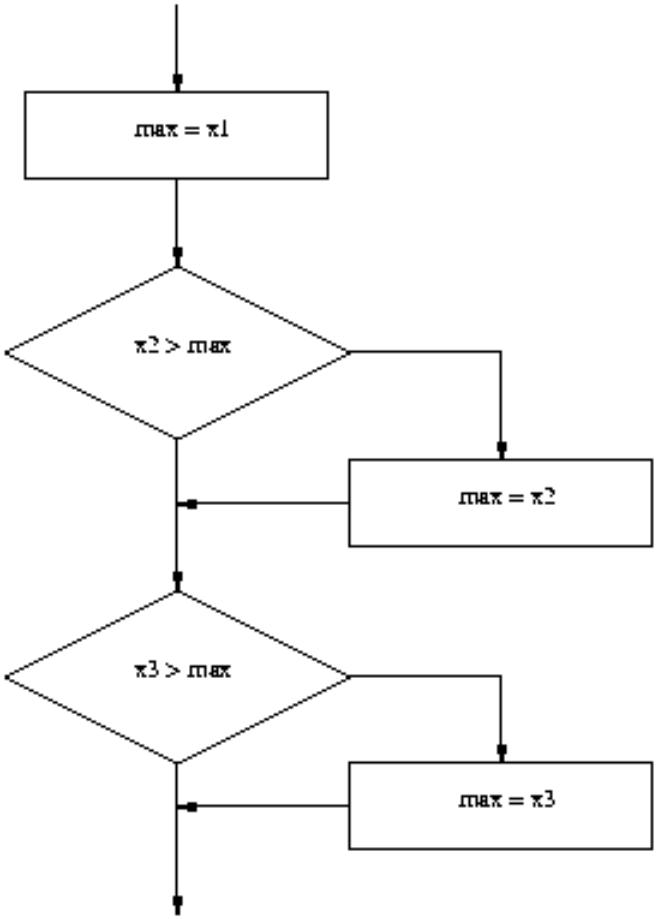
- Exactly two comparisons, regardless of ordering of the original three variables
- However, this approach is more complicated than the first
- To find the max of four values you'd need `if-else`s nested three levels deep with eight assignment statements

Strategy 3: Sequential Processing

- How a human solve this problem?
- With three numbers, this is trivial. But what about a list of 100 numbers?
- One strategy is to
  - Scan through the list looking for a big number
  - When one is found, mark it with your finger, and continue looking
  - If you find a larger value, move your finger and continue looking

- Use this as inspiration to find the max of 3?

Strategy 3: Sequential Processing



Strategy 3: Sequential Processing

- This idea can easily be translated into Python

```
max = x1
if x2 > max:
    max = x2
if x3 > max:
    max = x3
```

Strategy 3: Sequential Programming

- Repetitive process lends itself to looping

- Example: Max of user input numbers
  - Prompt the user for a number
  - Compare it to our current max, if it is larger, we update the max value
  - Repeat

Strategy 3: Sequential Programming

```
# maxn.py
#    Finds the maximum of a series of numbers

def main():
    n = eval(input("How many numbers are there? "))

    # Set max to be the first value
    max = eval(input("Enter a number >> "))

    # Now compare the n-1 successive values
    for i in range(n-1):
        x = eval(input("Enter a number >> "))
        if x > max:
            max = x

    print("The largest value is", max)
```

Strategy 4: Use Python !?

- Python has a built-in function called `max` that returns the largest of its parameters.

```
def main():
    x1, x2, x3 = eval(input("Please enter three values: "))
    print("The largest value is", max(x1, x2, x3))
```

Some Lessons

- Often many ways to solve a problem
  - Don't rush to code first idea that pops into your head. Think about the design and ask if there's a better way to approach the problem
  - Your first task is to find a correct algorithm. After that, strive for clarity, simplicity, efficiency, scalability, and elegance

Some Lessons

- Be the computer
  - Formulate an algorithm by thinking about how you'd solve the problem
  - This straightforward approach is often simple, clear, and efficient (enough)

http://www.tcm.com/mediaroom/video/323607/Caddyshack-Movie-Clip-Be-The-Ball.html

Some Lessons

- Generality is good
  - Consideration of a more general problem can lead to a better solution for a special case
  - If the max of *n* program is just as easy to write as the max of three, write the more general program because it's more likely to be useful in other situations

Some Lessons

- Don't reinvent the wheel
  - If the problem you're trying to solve is one that lots of other people have encountered, find out if there's already a solution for it
  - As you learn to program, designing programs from scratch is a great experience
  - Expert programmers know when to *borrow*
    - Search through Python functions
    - Search the Internet