Lab 6

- Pair Programming
- Review Lab 5
- Review indefinite loops, strings
- Lab 6

Lab 6: Pair Programming

Every lab, pairs will change

Alice	Callie	Hayden	Andrew
Andrew	Hayden	James	Matt
August	Danny	Karel	Cat
Callie	Alice	Kassi	Mike
Cat	Karel	Matt	James
Dan	Ellis	Melissa	Nate
Danny	August	Mike	Kassi
Ellis	Dan	Natalie	Giovanni
Giovanni	Natalie	Nate	Melissa

Sit with your teammate.

You can each log in, but one computer will be the machine you [both] program on.

Pair Programming

- **Two people work together at a single computer**
- **Driver** and **Navigator** work together on one task
- Roles change often
- Collective responsibility for outcome
- One approach used in “real world”

Pair Programming Tradeoffs

Pros

- Bring together multiple perspectives, experiences, abilities, and expertise
- Higher quality code
  - Catch bugs earlier
- Knowledge transfer
- Enhanced learning, communication
- Requires 100% engagement

Cons

- Slows down lines per minute (~15%)
- Loss of autonomy
- Scheduling
- Overconfidence
- Concentration
- Requires 100% engagement

Pair Programming Roles

Driver

- (Like the role I play when we write programs in class)
- Uses keyboard and mouse to execute all actions on the computer
- Ask questions wherever there is a lack of clarity
- Offer alternative solutions if you disagree with the navigator
  - When there is disagreement, defer to the navigator. If idea fails, get to failure quickly and move on
- Make sure code is "clean"
- Explains actions taken
- Brainstorms

Navigator

- (Like the role you play when we write programs in class)
- Directs driver's actions
  - Dictates the code that is to be written - the "what"
  - Clearly communicates what code to write
- Explains *why* chose particular solution to this problem
- Checks for errors and typos
- Plans the problem solving or debugging actions
- Asks questions

Your team will create your own workflow, within these guidelines

Expectations

- Take collective ownership of the code you and your partner are writing
  - No "my part" and "your part."
- Be an active, engaged, respectful team player
  - Goal: 50/50 division of labor (brainstorming, typing, testing, problem-solving, debugging, …)
  - Speak up when you don't understand, think there is an error, or wonder if there is a better way
  - Don't be too proud to admit a mistake
  - Apologize if you hurt your partner's feelings

Expectations

- Be open-minded
  - Pair programming is an opportunity to learn
  - One of the most important predictors of success in pair programming is buy-in: if you are determined to make the practice fail, it will.
- Coordinate breaks (e.g., for bathroom)
- Seek advice when you need it
  - We're still here to help
  - We'll ask even more questions to guide your approach

Expectations

- Don't be bound to the keyboard/mouse/monitor
  - Draw pictures
  - Refer to handouts
- Break down the problem into manageable pieces
  - Not always in order of problem
- Comments – include both authors

Submissions

See lab for more information

- At the end of the lab period, run `pairturnin.sh` to copy your files to a shared location
- If you finish the lab in class, `pairturnin.sh` also serves as your electronic submission
- Otherwise (if you didn't finish),
  - If you want to continue working together after lab, you can, **BUT** you must always work together in pair
    - **No** working partly on your own/partly together
    - Run `pairturnin.sh` when you complete the lab
    - Use `indiv_startup.sh` to copy the shared code into your own lab directory
  - If you want to finish up on your own
    - Run `indiv_startup.sh` to copy the shared code into your own lab directory
    - Submit using `turnin.sh` as usual

Advice from Previous Students

- I would advise them to really study the material and make sure they **understand the vocabulary** so that they can have **useful conversations** with their partners and the **navigator can actually navigate**.
- I would advise them to **prepare for the labs beforehand** so the person and their partner would be on **even footing** before the lab.
- **Try talking through** the syntax, semantic, or EOF **error** before asking for help. By doing so, if the pair does not figure out the error on their own, they are able to **explain their error better** after talking it through as a pair.

Advice from Previous Students

- If your partner seems to be taking a lead and you are **falling behind**, **voice** that to them and ask if they can **help you to understand** the material better. Try to be conscious of your role and if you are taking too much of a lead.
- I would also recommend, even though the labs are finished in pairs, to make sure **practice is done on your own before taking the tests**.
- It often felt as if one partner was far more skilled than the other and thus that only one should have a say. As we began to **practice this more**, I think that the class as a whole became far more relaxed and were able to **begin to truly collaborate effectively**.

Advice from Previous Students

- **Trust your partner** more than you think you should; be willing to take an **objective step back** and start over; **test often** and **test well**; and **experiment to prove to *yourself* how something works**.
- I recommend that they learn to listen to their partner. By actually hearing out their ideas, you can learn a lot about how **problems can be solved in different fashions** and be able to **apply these skills later on in their lives in and out of programming**.
- My best advice for pair programming and programming is ***to not be afraid to fail***. It is okay to mess up, your partner will not judge you for a mistake, but you have to ***not be afraid to fail in order to succeed in computer science***

LAB 5 REVIEW

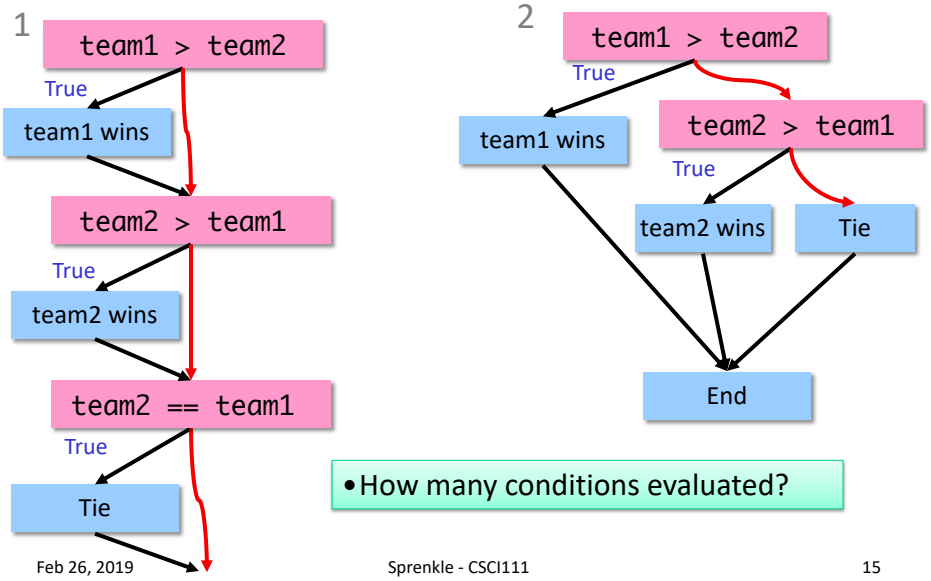
Common Issue: Inefficiency

```
if team1Score > team2Score:
    print("Team 1 wins!")
else:
    if team2Score < team1Score:
        print("Team 2 wins!")
    else:
        if team1Score == team2Score:
            print("They tied! We're going to overtime!")
```

Extra if statement, not necessary
Know when hit second else that the only possibility is a tie

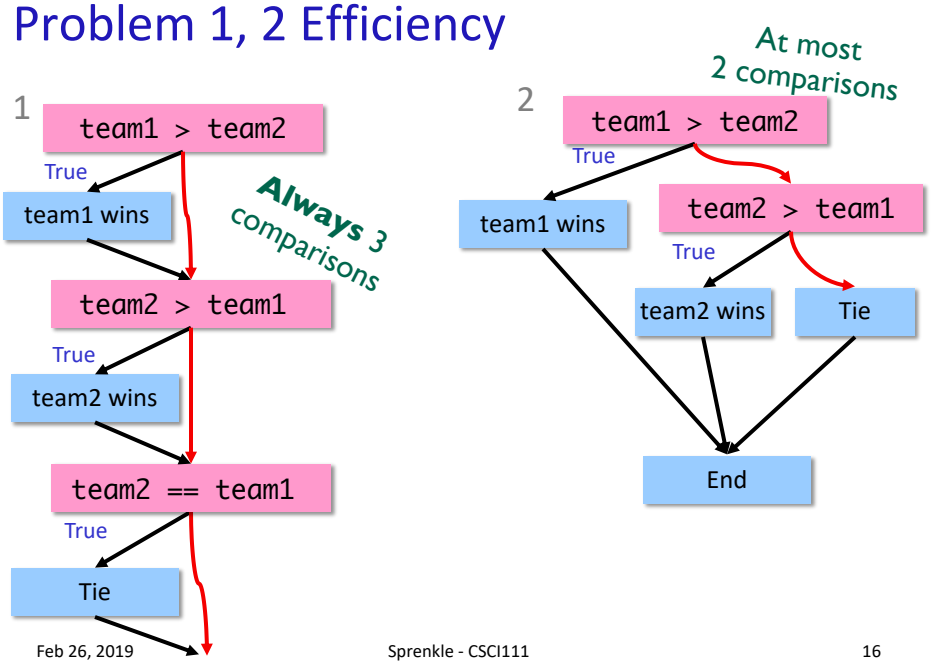
```
if team1Score > team2Score:
    print("Team 1 wins!")
else:
    if team2Score < team1Score:
        print("Team 2 wins!")
    if team1Score == team2Score:
        print("They tied! We're going to overtime!")
```

Problem 1, 2 Efficiency



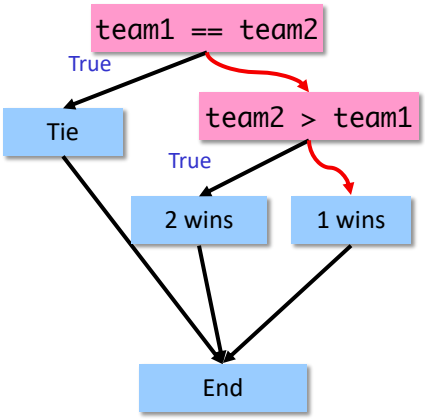
- How many conditions evaluated?

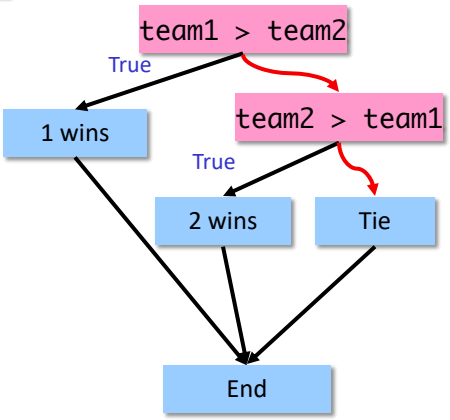
Problem 1, 2 Efficiency



Problem 2 (& 3) Efficiency

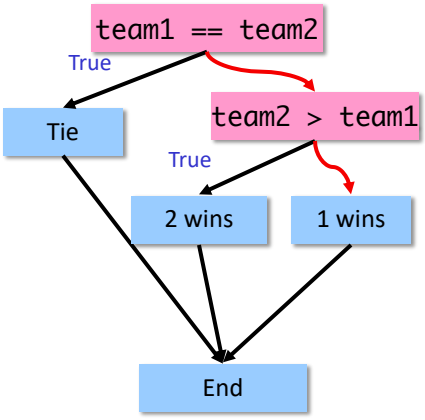
Which tends to be more efficient?
How many conditions to evaluate?



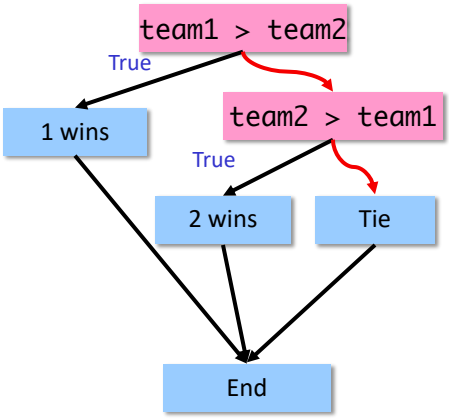


Problem 2 (& 3) Efficiency

Equality is a rare condition; on average, will always need to check second condition.



More common case. May only need to check one condition.



Adding to Development Process

- Last development step:
  - Assess your program again after it works
  - Is it efficient? Is it readable? Can I simplify?

Lab 5 – Greatest Hits: Less-Complicated Approaches for Customized Display

- Correct but more complicated solution to handling customized display

Other, similar examples in submissions

```
if albums == 1 and extraTracks == 0:
    print("Your album requires", albums, "cd")
elif albums == 1 and extraTracks > 0:
    print("Your album requires", albums, "cd")
    print(extraTracks, "tracks will have to wait for
                        the next Greatest Hits album")
elif albums > 1 and extraTracks > 0:
    print("Your album requires", albums, "cds")
    print(extraTracks, "tracks will have to wait for
                        the next Greatest Hits album")
elif albums > 1 and extraTracks == 0:
    print("Your album requires", albums, "cds")
```

Lab 5 – Greatest Hits: Less-Complicated Approaches for Customized Display

- Less complicated solution
  - Simpler logic, conditions
  - Less duplicated code

```
if albums == 1:
    print("Your album requires", albums, "CD.")
else:
    print("Your album requires", albums, "CDs")

if extraTracks > 1:
    print(extraTracks, "tracks will have to wait for
                    the next Greatest Hits album")
elif extraTracks==1:
    print(extraTracks, "track will have to wait for
                    the next Greatest Hits album")
```


Relational Operators

- Reminder: instead of, for example,

```
num < 0 or num > 0
```

can use

```
num != 0
```

Championship Extensions

A lot you could add already;
even more with a little more knowledge

- Simulate scores (rather than the difference)
- Change odds based on home/visiting team
- Dynamically change odds based on who won/lost already in the series

- Today: could stop the series after a team reaches four wins. How?

Review: Conditions and Indefinite Loops

- How do we write a condition that is true
  - Iff two expressions are both true
  - If at least one of those expressions is true
- What is the syntax for an indefinite loop?
- Which is more powerful: a `for` loop or an indefinite loop?

str Review

- How can we combine strings?
- How can we find out how long a string is?
- How can you tell if one string is contained in another string?
- How can we find out the character at a certain position?
- How can we iterate through a string?
- How do you call a method on a string?

String Methods vs. Functions

Functions

- All input comes from arguments/parameters
- Example: `len` is a built-in function
  - Called as `len(strobj)`

Methods

- Input comes from arguments ***and*** the string the method was called on
- Example:
  - `strobj.upper()`

Using the APIs

- Given a problem, break down the problem
  - Can any of the parts of the problem be solved using a method in the API?

Are You Smarter Than a 5th Grader?

- Problem in spelling from the show: How many a's are in abracadabra?
  - Solve using `str` methods
- Silly problem but can generalize to other problems
  - How many a's are in a given word?
  - How many of a certain letter are in a given word?

Lab 6

- Advanced conditions
- Indefinite Loops
- Text-based problems