Lab 7

- Feedback on Labs 5 and 6
- Review for Lab 7

Lab Musings

- As we learn more computer science, we're moving toward a **much higher ratio of *thinking* to *coding***
  - Give yourself the time and room to think
  - Discuss, reinforce your understanding
- Going beyond simply correctness in solutions
  - Looking for understanding of good coding practices
    - Testing, readability, usability, documentation, organization, efficiency
      - (not necessarily in that order)

Lab Musings

- Lab benefit: access to lab assistants and instructor to help
- Lab limitation: may not be the best environment
  - Seems to cause a competitive atmosphere, increased anxiety for some students
  - You have until Friday to complete the lab
  - Work at your pace, **think clearly** and **deeply**

LAB 5 FEEDBACK

Common Issue: Inefficiency

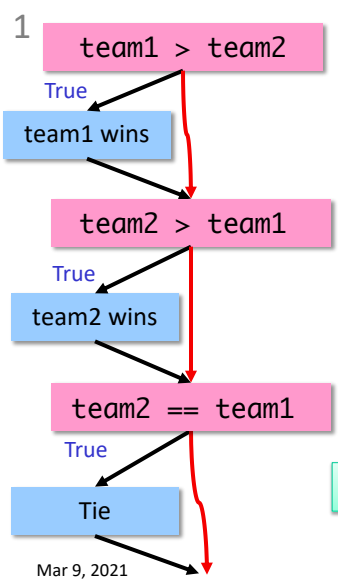
```
if team1Score > team2Score:
    print("Team 1 wins!")
else:
    if team2Score > team1Score:
        print("Team 2 wins!")
    else:
        if team1Score == team2Score:
            print("They tied! We're going to overtime!")
```

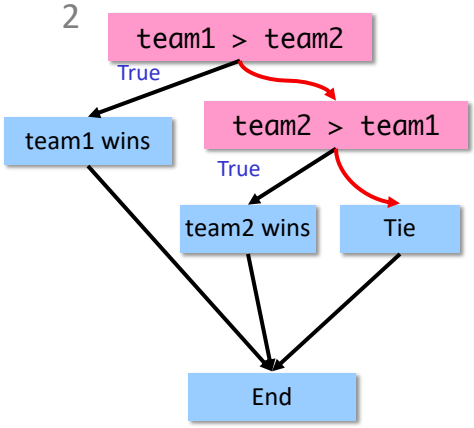
Last if statement is not necessary

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!")
```

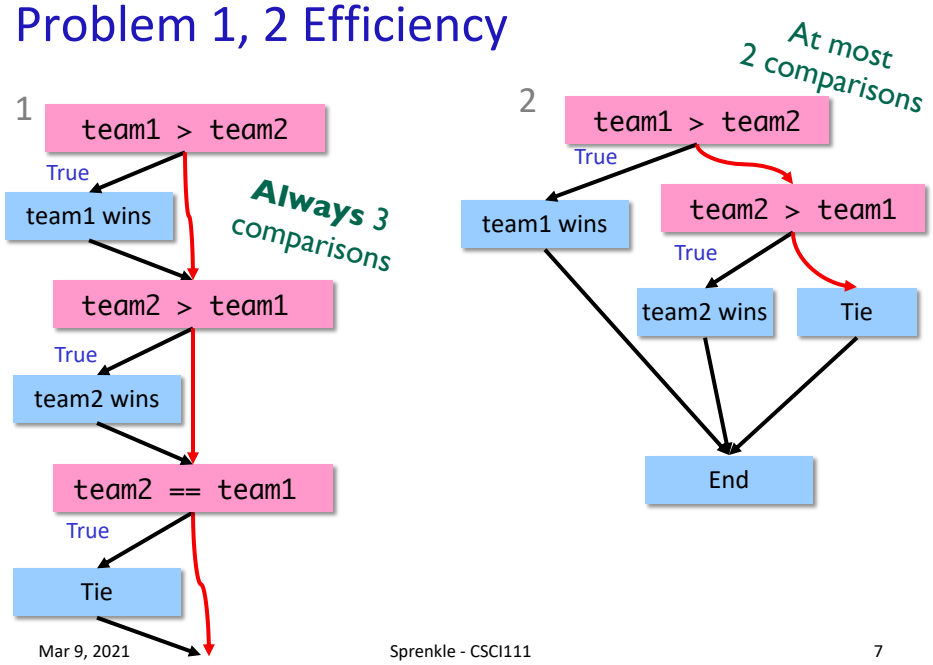

Problem 1, 2 Efficiency





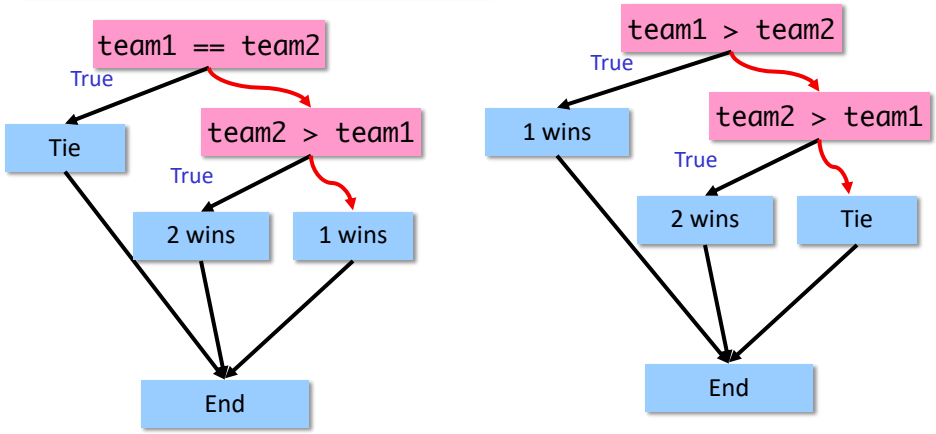
- How many conditions evaluated?

Problem 1, 2 Efficiency



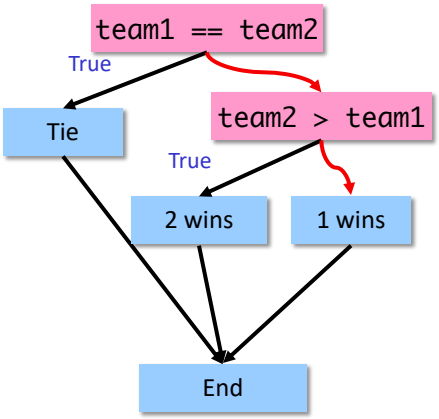
Problem 2 (& 3) Efficiency

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

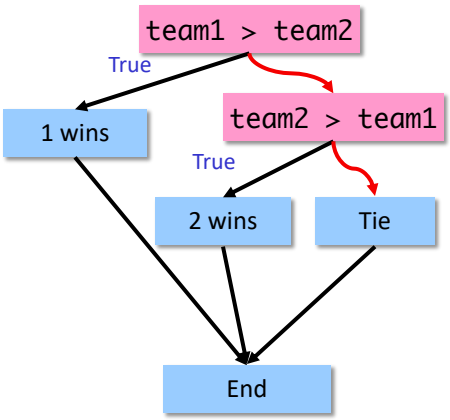


Problem 2 (& 3) Efficiency

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



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



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")
```


Adding to Development Process

- After your program works, consider
  - Is it efficient?
  - Is it readable?
  - Can I simplify?
- Modify, test again

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 simulate a World Series that plays games until a team reaches four wins. How?

LAB 6 FEEDBACK

Checking if a str contains a substring

Instead of using a method, could use `in` operator because didn't care where in the string it was:

```
if "r" in phrase:
```

Over `string`

- Why do you ***not*** need to use `str` in the following code segment?

```
origString = str( input("What is your string? ") )
```


Over `string`

- Why do you ***not*** need to use `str` in the following code segment?

```
origString = str( input("What is your string? ") )
```

  - Because `input` returns a string; no need to cast
- Preferred:

```
origString = input("What is your string? ")
```

Goal: Simplify/reduce code
→ Less code → easier to understand, less error-prone

When to Compute

- Don't do computation until it is needed

```
fileName = input("What is the name of your file? ")

lastPeriod = fileName.rfind(".")

if lastPeriod != -1 and lastPeriod != len(fileName) - 1 :

    extension = fileName[lastPeriod+1:]
    print(fileName, "is a(n)", extension, "file.")

else:
    print(fileName, "does not have an extension")
```

Find extension here

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?
  - What are the two main ways to structure indefinite loops to solve a problem?
- Which is more powerful: a `for` loop or an indefinite loop?

`while` Loops Comparison

```
# condition says when loop
# will continue
x=eval(input("Enter number:"))
while x % 2 != 0 :
   print("Error!")
   x = eval(input("Enter
       number: "))
print(x, "is an even number.")
```

Loop condition says when to keep going

```
# have to look inside loop to
# know when it stops
while True :
   x = eval(input("Enter number:"))
   if x % 2 == 0 :
      break
   print("Error!")
print(x, "is an even number.")
```

"breaks" out of a loop

Internal condition says when to stop

Using break statements: Best when loop has to execute at least once.

str Review

Review on your own

- 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?

Review

- How do you format strings?
  - What does a format specifier look like?
  - What questions should you ask when formatting strings/creating the format specifier?
- How can we find the ASCII value for a character?
- How can we find the character associated with an ASCII value?

Review: String Formatting

- Use the format method
  - `"templatestring".format(replacementvalues)`
- Format specifiers syntax:
  `{[flags][width][.precision][code]}`
- When determining format specifiers, consider
  - Data type of the replacement value
    - If float, how many decimal places desired
  - Desired width
  - Justification, other flags

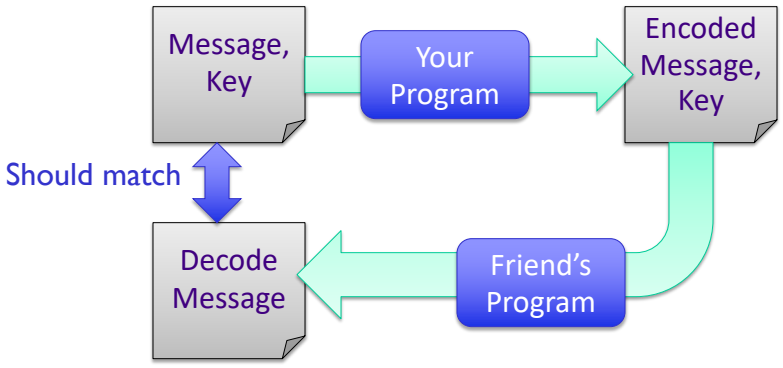
Review: Translating to/from ASCII

- Translate a character into its ASCII numeric code using **built-in function** `ord`
  - `ord('a') ==> 97`
- Translate an ASCII numeric code into its character using **built-in function** `chr`
  - `chr(97) ==> 'a'`

`ascii_table.py`
`ascii.py`

Caesar Cipher

- Write an encoding/decoding program
  - Encode a message
  - Give to a friend to decode



What is the algorithm for encoding a letter?

- Assuming a lowercase letter

- Test Cases:
  - `test.testEqual(encodeLetter('a', 1), 'b' )`
  - `test.testEqual(encodeLetter('y', 1), 'z' )`
  - `test.testEqual(encodeLetter('z', 5), 'e' )`
  - `test.testEqual(encodeLetter('b', -4), 'x' )`

What is the algorithm for encoding a letter?

(Assuming a lowercase letter)

1. Convert the character to its ASCII value
2. Add the key to that value
3. Make sure that the new value is a "valid" ASCII value, i.e., that that new value is in the range of lowercase letter ASCII values
   1. If not, "wrap around" to adjust that value so that it's in the valid range
4. Convert the ASCII value into a character

What is the algorithm for encoding a message?

- Assuming message only made of up lowercase letters and spaces

- Examples:
  - `test.testEqual(encodeMessage('cat', 1), 'dbu' )`
  - `test.testEqual(encodeMessage('w and l', 5), 'b fsi q' )`

Encode a Message

- Accumulate a new encoded message
- For each character in the message
  - Check if the character is a space; if it is, it stays a space
    - Add space to the encoded message
  - Otherwise
    - Encode letter
    - Add encoded letter to the encoded message

We need to accumulate the encoded message in a new string rather than change the message because strings are ***immutable***

Lab 7

- Indefinite Loops
- Caesar Cipher
- Strings
  - Escape sequences
  - Formatting