Lab 4 Feedback

- Good overall
  - Problems will keep getting more difficult
  - More and more possibilities as we combine all our newly acquired skills

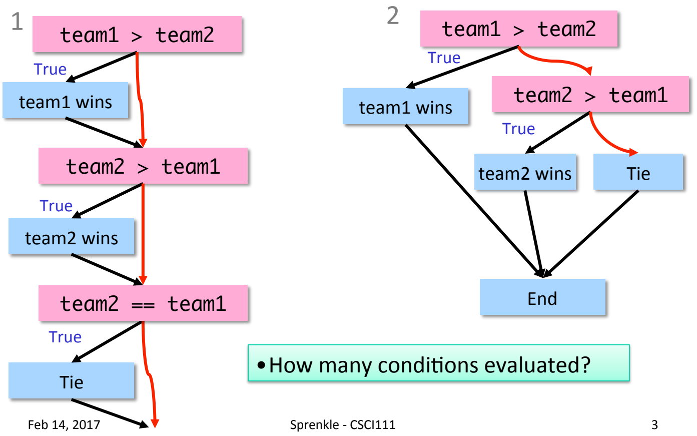
Suggestion: Practice new skills in between labs (after every class)
- Think about when you would want to use that skill
- Use online resources

Common Issue: Inefficiency

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

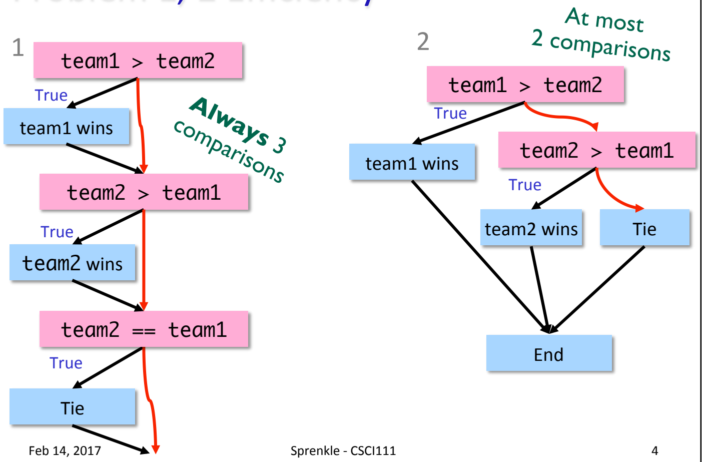
Extra if statement, not necessary
Know when hit second else that the only possibility is a tie

Problem 1, 2 Efficiency



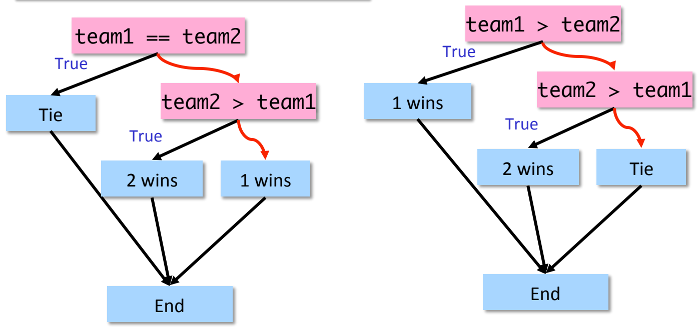
- How many conditions evaluated?

Problem 1, 2 Efficiency



Problem 2 (& 3) Efficiency

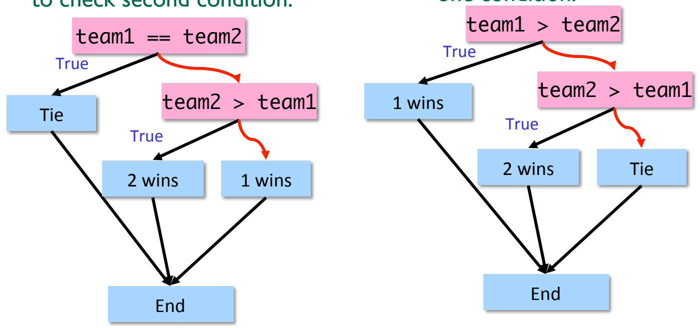
Which tends to be more efficient?
How many conditions to evaluate?



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?

REVIEW: STRINGS

Review

- How can we combine strings?
- How can we find out how long a string is?
- 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 Operations

Operand	Syntax	Meaning
`+`	`str1 + str2`	Concatenate two strings into one string
`*`	`str * num`	Concatenate string `num` times

- Examples:
  - `"I feel " + "sleepy"`
    - Evaluates to `"I feel sleepy"`
  - `"Oops! " * 3`
    - Evaluates to `"Oops! Oops! Oops! "`

Recall lab 0

String Comparisons

- Same operations as with numbers:
  - ==, !=
  - <, <=
  - >, >=

  } Alphabetical comparison
- Use in conditions in `if` statements

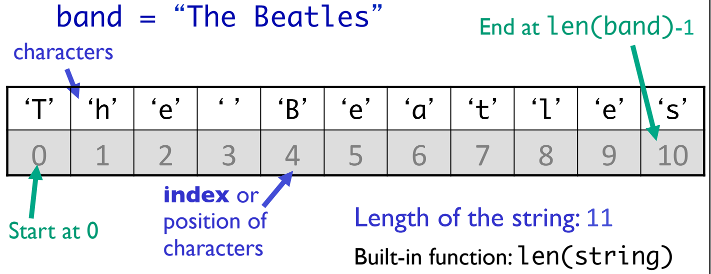
```
if userpick == pick4num:
    print("We have a winner!")
else:
    print("You lose.")
```

`string_compare.py`

Strings

- A *sequence* of characters
  - Example:

  `band = “The Beatles”`



End at len(band)-1
characters
‘T’ ‘h’ ‘e’ ‘ ’ ‘B’ ‘e’ ‘a’ ‘t’ ‘l’ ‘e’ ‘s’
0 1 2 3 4 5 6 7 8 9 10
Start at 0
index or position of characters

Length of the string: 11

Built-in function: `len(string)` to find length of a string

Summary: Iterating Through a String

- For each character in the string

string of length 1

```
for char in mystring:
    print(char)
```

Determines loop's behavior

- For each position in the string

An integer

```
for pos in range(len(mystring)):
    print(mystring[pos])
```

Index into the string

`str` Methods

- Example method: **`find(substring)`**
  - Finds the index where substring is in string
  - Returns -1 if substring isn't found
- To call a method:
  - `<str_obj>.methodname([arguments])`
  - Example: `filename.find(".py")`

Executed on this string

Common `str` Methods

Method	Operation
`center(width)`	Returns a copy of string centered within the given number of columns
`count(sub[, start [, end]])`	Return # of non-overlapping occurrences of substring `sub` in the string.
`endswith(sub), startswith(sub)`	Return `True` iff string ends with/starts with `sub`
`find(sub[, start [, end]])`	Return first index where substring `sub` is found
`isalpha(), isdigit(), isspace()`	Returns `True` iff string contains letters/digits/whitespace only
`lower(), upper()`	Return a copy of string converted to lowercase/lowercase

`string_methods.py`

Common `str` Methods

Method	Operation
`replace(old, new[, count])`	Returns a copy of string with all occurrences of substring `old` replaced by substring **`new`.** If `count` given, only replaces first `count` instances.
`split([sep])`	Return a list of the words in the string, using `sep` as the delimiter string. If `sep` is not specified or is None, any whitespace string is a separator.
`strip()`	Return a copy of the string with the leading and trailing whitespace removed
`join(<sequence>)`	Return a string which is the concatenation of the strings in the sequence with the string this is called on as the separator
`swapcase()`	Return a copy of the string with uppercase characters converted to lowercase and vice versa.

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?

Escape Sequences

- Escape character: `\`
- Escape sequences
  - newline character (carriage return) → `\n`
  - tab → `\t`
  - quote → `\"` or `\'`
  - backslash → `\\`
- Example:
  - `print("To print a \\, you must use \"\\\\\"")`
    - What does this display?

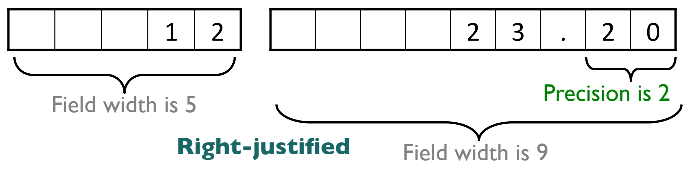
demo_str.py

FORMATTING STRINGS

Example Format Specifiers

`"{:5d}".format(12)` → `"   12"`

`"{:9.2f}".format(23.1999)` → `"    23.20"`

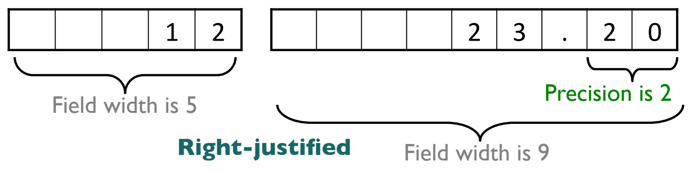


- What if precision is bigger than the decimal places?
- What if field width is smaller than the length of the value?

Any guesses? Try out in interpreter.

Example Format Specifiers

```
"{:5d}".format(12)   "{:9.2f}".format(23.1999)
  → "   12"              → "    23.20"
```



- What if precision is bigger than the decimal places?
  - Fills decimal with 0s
- What if field width is smaller than the length of the value?
  - String contains entire value

Formatting Practice

- `x = 10`
- `y = 3.5`
- `z = "apple"`
- `"{:6d}".format(x)`
- `"{:6.2f}".format(x)`
- `"{:06.2f}".format(y)`
- `"{:6.2f}".format(y)`
- `"{:^10s}".format(z)`
- `"{:5d} {:<7.3f}".format(x,y)`

Example: Printing Out Tables

- A table of temperature conversions

```
Temp F      Temp C      Temp K
------      ------      ------
-459.7      -273.1         0.0
   0.0       -17.8       255.2
  32.0         0.0       273.1
```

- If we want to print data in rows, what is the template for what a row looks like?
  - How do we make the column labels line up?

`temp_table.py`