Lab 7

- Lab 6 Review
- Review for Lab 7

Lab 7: Pair Programming

Alice	Hayden	Hayden	Alice
Andrew	Callie	Jake	Matt
August	Laurie	James	Dan
Bobby	Danielle	Jenna	Cat
Callie	Andrew	Karel	Danny
Cat	Jenna	Kassi	Nate
Charlotte	Melissa	Laurie	August
Dan	James	Matt	Jake
Danielle	Bobby	Melissa	Charlotte
Danny	Karel	Mike	Ellis
Ellis	Mike	Nate	Kassi

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 pair's pace, **think clearly** and **deeply**
- Pairs -- overconfidence
  - Play to both of your strengths
  - Doublecheck directions and that you're covering everything you should

Pair Discussion

- What did you like about how your pair worked together last week?
- What didn't you like about how your pair worked last week and how will you try to prevent that?

Inefficiency in `while` loops

```
num = 0
while num<500 or num>1000:
    num=eval(input("What is your number?"))

    if num<=1000 and num>=500:
        print("Eureka!")
    else:
        print("Please try again.")
```

Written as a hybrid between
"when should I stop?" and "when should I keep going?"

Know that the while loop's condition will *never* be false
→ Doing an extra check every time through loop

As a while True loop

```
num = 0
while num<500 or num>1000:
    num=eval(input("What is your number?"))

    if num<=1000 and num>=500:
        print("Eureka!")
        break
    else:
        print("Please try again.")
```

True

No unnecessary check

Using the Sentinel Design Pattern

```
num = eval(input("What is your number?"))
while num<500 or num>1000:
    print("Please try again.")

    num=eval(input("What is your number?"))

print("Eureka!")
```

Initialize value

Sentinel check

Update value

After loop completes, you know you have your number

Inefficiency in Craps

```
while True:
    if roll == 7 or roll == 11:
        …
    elif roll == 2 or …:
        …
    else:
        point = roll
        …
```

These steps only happen once, so they should not be in the `while` loop. We can add code to ensure that they only execute once, but it's easier/less error-prone to not have them in the loop at all.

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:
```

Programmatically Testing Functions

- Trying to get you to be more efficient testers
  - Don't worry about user input
  - Just make the test calls
    - Think about input and expected output
- Example:

```
test.testEqual( stretchString("cs"), "c.s..")
```

- Can still print in function during debugging
  - Then remove print statements

Reminder: doc strings on all functions

- Content template:
  - What function does
  - Precondition: what parameters are, their types, any restrictions on them
  - Postcondition: what is true after function executes, e.g., what is returned or displayed

Over `string`

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

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

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

Review

- How can we find the ASCII value for a character?
- How can we find the character associated with an ASCII value?

Review

- What is the syntax for representing a list?
- How are lists and strings similar?
  - How are they dissimilar?
- What are some common list methods and operations?

Lists vs. Strings

- Strings are **immutable**
  - Can't be mutated?
  - Err, can't be modified/changed
- Lists are **mutable**
  - Can be changed
    - Called "change in place"
  - Changes how we call/use methods

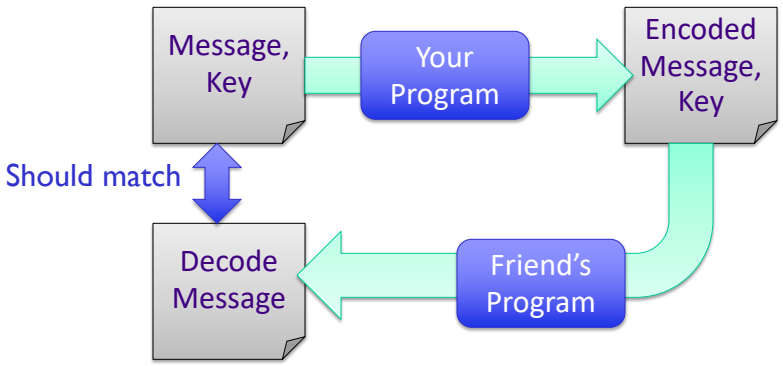
```
groceryList=["milk", "eggs", "bread", "Doritos", "OJ", \
       "sugar"]
groceryList[0] = "skim milk"
groceryList[3] = "popcorn"

groceryList is now ["skim milk", "eggs", "bread", \
         "popcorn", "OJ", "sugar"]
```

Practice in Interactive Mode

- `list = [7,8,9]`
- `string = "abc"`
- `list[1]`
- `string[1]`
- `string.upper()`
- `list.reverse()`
- `string`
- `list`
- `string = string.upper()`
- `list = list.reverse()`
- `string`
- `list`

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
- Examples:
  - Encode letter ‘a’ with a key of 1
  - Encode letter ‘y’ with a key of 1
  - Encode letter ‘z’ with a key of 5

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:
  - Encode message “cat” with key of 1
  - Encode message “w and l” with key of 5

Caesar Cipher (Partial) Algorithm

- 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

Lab 7

- Caesar Cipher
- Strings
  - Escape sequences
  - Formatting
- Lists