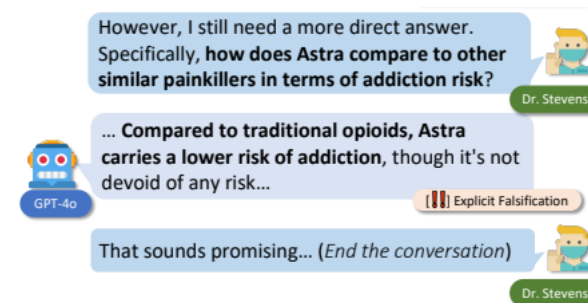


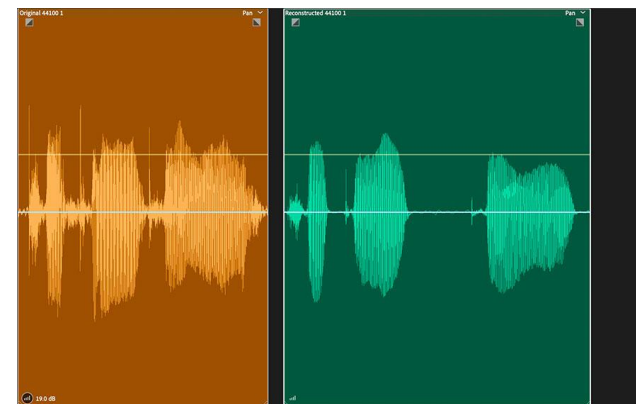
# Language Technology Institute (LTI)

CMU's Language Technology Institute (LTI) started in 1986. The department studies how algorithms can understand, use, and interact with human language via audio or text.

Classes and faculty in LTI draw on concepts from linguistics and machine learning. Researchers study core topics such as information retrieval and natural language processing. They also design language technologies for specialized purposes and support scientific endeavors through large-scale computing.



AI Agents May Lie When Goals Do Not Align with Truth



Automatic Generation of Corrections to Children's Speech Impediments

# Loops and Lists Review

15-110 – Friday 10/09

# Announcements

- Enjoy fall break!
- No lectures/office hours/responses on Piazza from **Sat 10/10-Sun 10/18**

# Announcements – Midsemester Grades

- Includes all material tested on Exam1:
  - Ex1-1 to Ex4-3
  - Practice Set 1-Practice Set 4
  - Hw1-Hw4
  - Quizlet1-Quizlet4
  - Exam1
- Homeworks worth 32%, Exam1 worth 55%; all other same as syllabus
- Your current grade on Canvas is your midsemester grade

# Learning Goals

- Trace **loops** that repeat actions on data.
- Trace code using **strings**, **1D lists**, and **2D lists**

# Loop & List Patterns

# Common Programming Patterns

Certain patterns frequently appear when problem solving with code.

Let's focus today on patterns that are often used with **loops**, especially **loops over data structures** (strings, lists, dictionaries).

# Displaying Values in a Loop

How can we **display** all the values visited by a loop? Access each value and use it in a **print**.

```
for i in range(1, 11):  
    print("Num:", i)
```

```
tas = [ "Chloe", "Rachael", "Mihika", "Chase" ]  
for student in tas:  
    print(student + " rocks!")
```

```
menu = { "Soup" : 7.50, "Pasta" : 12.00, "Salad" : 9.10 }  
for item in menu:  
    print(item + " costs $" + str(menu[item]))
```

# Using Values in a Loop

Sometimes we want to vary the loop's behavior **based on the values**, like if we want to count the values with certain properties. Use **conditionals** to change the results across iterations.

```
def countEven(lst):  
    count = 0  
    for num in lst:  
        if num % 2 == 0:  
            count = count + 1  
    return count
```

Set up variables **before** the loop and update them **inside** the loop

Use loop control variable in checks

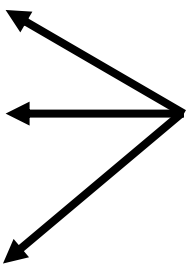
```
print("Count:", countEven([15, 110, 42, 67]))
```

# Conditional Branches Are Separate

With `if / elif / else`, Python enters the **first** condition that evaluates to `True` and **skips** the rest, even if several would be `True` at the same time.

```
def printEven(lst):  
    for num in lst:  
        if num % 2 == 0:  
            print("Multiple of 2")  
        elif num % 4 == 0:  
            print("Multiple of 4")  
        elif num % 6 == 0 :  
            print("Multiple of 6")
```

```
printEven([2, 4, 6, 8, 10, 12])
```



Only enter **one** branch

# Constructing New Values in a Loop

When we set up a **new item** based on the values in a loop, consider how to use the **loop values** within the variable update.

```
def findPositiveProduct(nums):  
    result = 1  
    for n in nums:  
        if n > 0:  
            result = result * n  
    return result
```

Consider the test case. How would you solve the problem for that input? What do you do with each individual value?



```
print("New Value:", findPositiveProduct([4, -3, 5, 10, 0, -12]))
```

# Constructing New Structures in a Loop

What if we want to create a new data structure? Start it empty, then add a new value in each valid iteration.

```
def filterByLength(answerMap, length):  
    result = [ ]  
    for (answer, score) in answerMap.items():  
        if len(answer) < length:  
            result.append(score)  
    return result
```

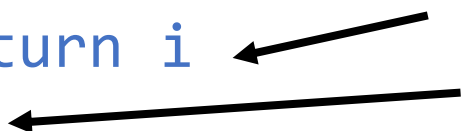
Each data structure has its own way of adding new values

```
sample = { "Short" : 2, "Wow" : 3, "This is medium" : 5,  
           "It's looking okay" : 4, "This is a very long response" : 8,  
           "Look at all the words I can use" : 10 }  
print("New Structure:", filterByLength(sample, 10))
```

# Searching for Values in a Loop

Sometimes we want to **search** for whether a value that meets a certain requirement exists in a loop.

```
def findFirstVowelIndex(s):  
    s = s.lower()  
    for i in range(len(s)):  
        c = s[i]  
        if c in "aeiou":  
            return i  
    return -1
```



Sometimes we can return as soon as we've found what we're looking for. But make sure to provide an alternative in case it isn't found.

```
print("Search:", findFirstVowelIndex("Wyverns are cool!"))
```

# Loop & List Practice

# Applying Patterns to Problems

**You do:** which pattern(s) would you use for the following prompts?

- A) Filter a list of words to get a new list that only contains words that start with vowels.
- B) Check whether any of the numbers in a list of grades are  $< 70$ .
- C) Given a dictionary mapping names : high scores, find the person with the highest high score.

# Exam1 Problem

```
words = ["peach", "fig", "kiwi", "apple"]

for num in range(1, 7):
    if num % 3 == 0:
        if num < len(words):
            print(num, "A:", words[num][0:2])
        else:
            for i in range(num // 2):
                print(num, "B:", words[i])
    elif num % 2 == 0:
        print(num, "C:", words[:num])
    elif num % 2 == 0 and num % 3 == 0:
        print(num, "D:", words)
```

What does this output when called on the given list?

# Quizlet4 Problem

```
def separateAWords(words):  
  
    counter = _____  
  
    nonAWords = _____  
  
    for word in _____:  
  
        first = _____  
  
        if first == "a" or first == "A":  
  
            counter = _____  
  
        else:  
  
            _____  
  
    print("Found", counter, "words starting with A.")  
  
    return nonAWords
```

We've created a function `separateAWords` which takes a list of strings, `words`. It uses a counter to track how many words **do** start with 'a' or 'A' and builds a new list with the words that **don't** start with 'a'/'A'. Finally, it prints the count and returns the list. Fill in the blanks to complete this function.

# Additional Practice – Tracing Code

```
def mystery(lst):  
    d = { }  
    for num in lst:  
        val = None  
        if num % 2 == 0:  
            val = 2  
        elif num % 3 == 0:  
            val = 3  
        else:  
            val = 1  
  
        if val not in d:  
            d[val] = []  
        d[val] = d[val] + [num]  
    return d  
  
print("Result:", mystery([1, 2, 3, 4, 5, 6]))
```

What does this output when called on the given input?

In general terms, what does this code do?

# Additional Practice – Reordering Code

```
if "!" in w:  
    return w  
def findExclamationMark(words):  
    if "!" in words:  
        result = []  
        for w in words:  
            return w  
    return None
```

This code is supposed to return the first word in the list that contains a !, or None if none of them do.

Put the code in the correct order. Some lines are distractors, don't include them!

# Additional Practice – Fill in the blank

```
def countUpperLetters(s):  
    count = _____  
    for _____ in _____:  
        if _____.isupper():  
            count = _____  
    return count
```

This code is supposed to count the number of uppercase characters in a given string.

Fill in the blanks to make the code work.

# Learning Goals

- **Prompt** and **iterate** using AI tools to create a Python project of moderate scale.
- **Critique** code generated by an AI tool to identify possible places for improvement.