CMSC201
Computer Science I for Majors

Lecture 07 – Lists

Last Class We Covered

- Using **`while`** loops
  - Syntax of a **`while`** loop
  - Interactive loops
  - Infinite loops and other problems
- Practice with **`while`** loops

Any Questions from Last Time?

Today's Objectives

- To learn about lists and what they are used for
  - To be able to create and update lists
  - To learn different ways to mutate a list
    - `append()`
    - `remove()`
- To get more practice with `while` loops
  - Sentinel values

Announcement: BRAID Survey

- Based on your enrollment in this course, you will be receiving an email inviting you to take a short survey about your experiences in computer science. I strongly encourage you to participate in this survey, since your participation is vital to understanding the diverse experiences of computer science students.
- The first 400 respondents to complete the survey across the BRAID campuses will receive a $15 Amazon gift card as compensation and all students who complete the survey will be entered in a drawing to win one of two $125 Amazon gift cards.
- Please watch for an email with a link to complete the survey.

BRAID | Building Recruiting And Inclusion for Diversity

Introduction to Lists

Exercise: Average Three Numbers

- Read in three numbers and average them

```
num1 = int(input("Please enter a number: "))
num2 = int(input("Please enter a number: "))
num3 = int(input("Please enter a number: "))
print((num1 + num2 + num3) / 3)
```

- Easy! But what if we want to do 100 numbers? Or 1000 numbers?
- Do we want to make 1000 variables?

Using Lists

- We need an easy way to hold individual data items without needing to make lots of variables
  - Making **`num1`**, **`num2`**, **`...`**, **`num99`**, **`num100`** is time-consuming and impractical

- Instead, we can use a ***list*** to hold our data
  - A list is a ***data structure***: something that holds multiple pieces of data in one structure

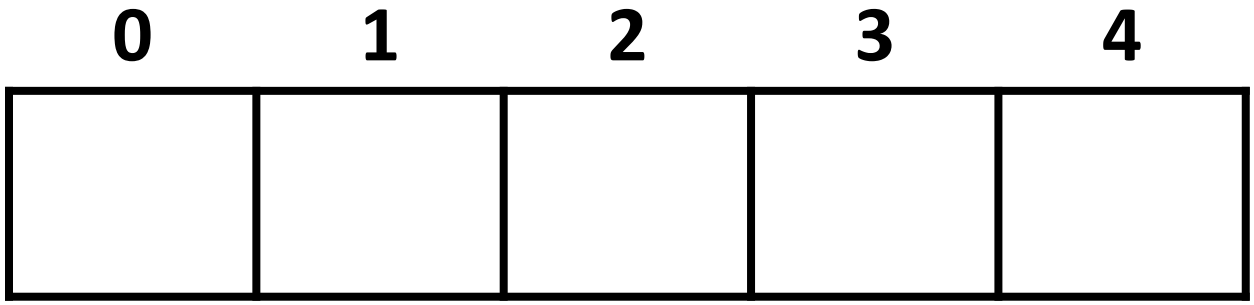
Using Lists: Individual Variables

- We need an easy way to refer to each individual variable in our list
- What are some possibilities?
  - Math uses subscripts (x_1, x_2, x_3, etc.)
  - Instructions use numbers ("Step 1: Combine…")
- Programming languages use a different syntax
  - `x[1]`, `x[0]`, `instructions[1]`, `point[i]`

Accessing Individual Elements

- We can access the individual elements in a list through ***indexing***

- List don't start counting from 1
  - They start counting from 0!



List Syntax

- Use `[]` to assign initial values (*initialization*)

```
myList = [1, 3, 5]
words  = ["Hello", "to", "you"]
```

- And to refer to individual elements of a list

```
>>> print(words[0])
Hello
>>> myList[0] = 2
```

Properties of a List

- Heterogeneous (multiple data types!)
- Contiguous (all together in memory)
- Ordered (numbered from 0 to n-1)

- Have instant ("random") access to any element
- Are "mutable sequences of arbitrary objects"

List Example: Grocery List

- You are getting ready to head to the grocery store to get some much needed food

- In order to organize your trip and to reduce the number of impulse buys, you decide to make a grocery list

Image from flickr.com

List Example: Grocery List

- Inputs:
  - 3 items for grocery list
- Process:
  - Store groceries using list data structure
- Output:
  - Final grocery list

Grocery List Code

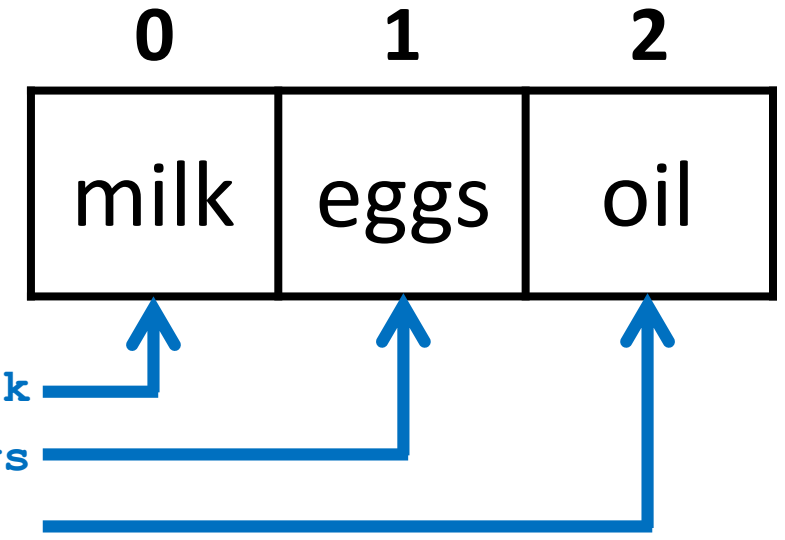
```
def main():
    print("Welcome to the Grocery Manager 1.0")
    grocery_list = ["", "", ""]   # initialize list

    # get grocery items from the user
    grocery_list[0] = input("Please enter your first item:  ")
    grocery_list[1] = input("Please enter your second item: ")
    grocery_list[2] = input("Please enter your third item:  ")

    # print out the items they selected
    print(grocery_list[0])
    print(grocery_list[1])
    print(grocery_list[2])
main()
```

Grocery List Demonstration

- Here's a demonstration of what the code is doing




```
bash-4.1$ python groceries.py
Please enter your first item:  milk
Please enter your second item: eggs
Please enter your third item:  oil
```

```
grocery_list = ["", "", ""]

grocery_list[0] = input("Please enter ...: ")
grocery_list[1] = input("Please enter ...: ")
grocery_list[2] = input("Please enter ...: ")
```

List Example: Grocery List

- What would make this process easier?
- Loops!
  - Instead of asking for each item individually, we could keep adding items to the list until we wanted to stop (or the list was “full”)
- We’ll update our program to use a loop soon
  - For now, let’s talk about lists a bit more

Mutating Lists

Mutating Lists

- Remember that lists are defined as "mutable sequences of arbitrary objects"
  - "Mutable" just means we can change them

- So far, the only thing we've changed has been the contents of the list
  - But we can also change a list's size, by adding and removing elements

List Function: `append()`

- The **`append()`** function lets us add items to the end of a list, increasing its size

  `listName.append(itemToAppend)`

- Useful for creating a list from flexible input
  - Allows the list to expand as the user needs
  - No longer need to initialize lists
    - Can instead start with an empty list **`[]`**

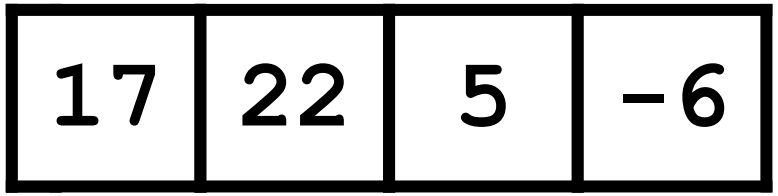
Example of `append()`

- We can use **`append()`** to create a list of numbers (using the loop to control how many)

```
values  = []  # initialize the list to be empty
count   = 0   # count how many numbers added

while count < 10:
    userVal = int(input("Enter a number: "))
    values.append(userVal) # add value to the list
    count += 1
```

- Here's a demonstration of what the code is doing




```
bash-4.1$ python numberList.py
Enter a number: 17
Enter a number: 22
Enter a number: 5
Enter a number: -6
```

```
values  = []  # initialize empty list
count   = 0
while count < 10:
    userVal = int(input("Enter a number: "))
    values.append(userVal) # add value to the list
    count += 1
```

List Function: `remove()`

- The `remove()` function lets us remove an item from the list – specifically, it finds and removes the first instance of a given value

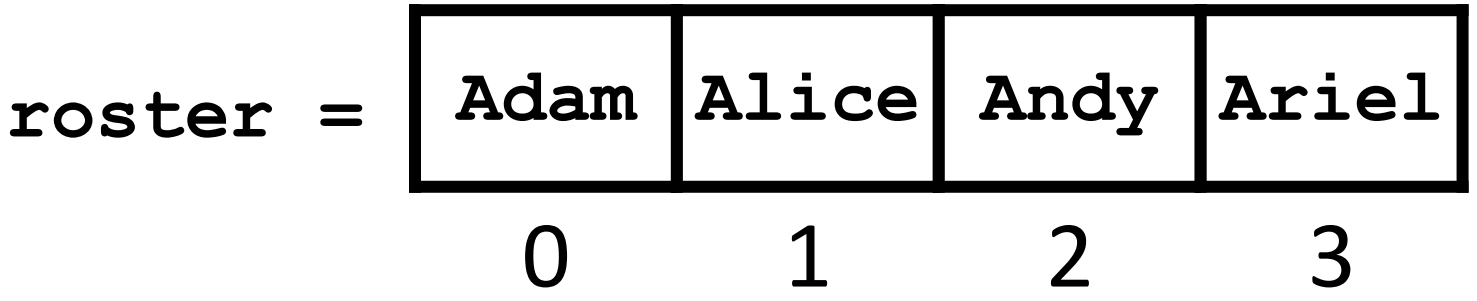
```
listName.remove( valueToRemove )
```

- Useful for deleting things we don't need
  - For example, removing students who have dropped the class from the class roster
  - Instead of the list having "empty" elements

Example of `remove()`

- We can use `remove()` to remove students who have dropped the class from the roster

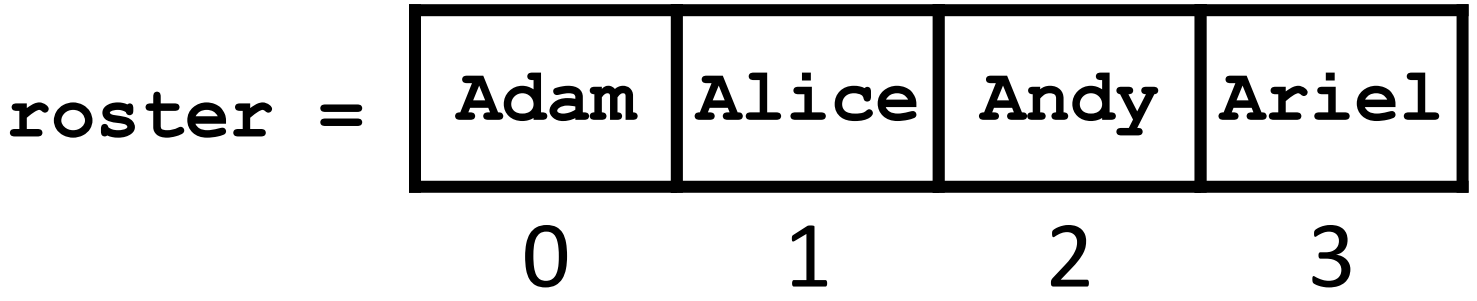
```
roster = ["Adam", "Alice", "Andy", "Ariel"]
```



Example of `remove()`

- We can use `remove()` to remove students who have dropped the class from the roster

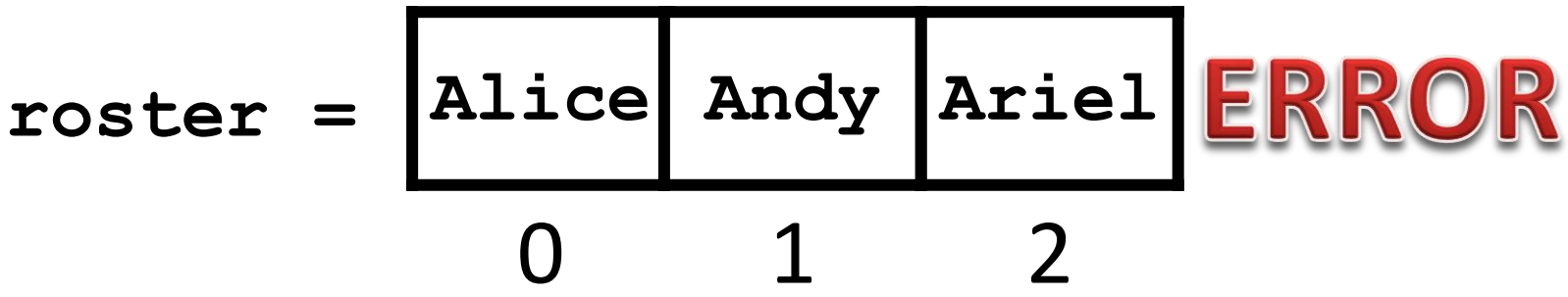
```
roster = ["Adam", "Alice", "Andy", "Ariel"]
roster.remove("Adam")    # Adam has dropped the class
```



Example of `remove()`

- We can use `remove()` to remove students who have dropped the class from the roster

```
roster = ["Adam", "Alice", "Andy", "Ariel"]
roster.remove("Adam")    # Adam has dropped the class
roster.remove("Bob")     # Bob is not in the roster
```



Sentinel Values and `while` Loops

When to Use `while` Loops

- `while` loops are very helpful when you:
  - Want to get input from the user that meets certain specific conditions
    - Positive number
    - A non-empty string
  - Want to keep getting input until some "end"
    - User inputs a value that means they're finished

what we're covering now

Sentinel Values

- ***Sentinel values*** "guard" the end of your input
- They are used:
  - When you don't know the number of entries
  - In **`while`** loops to control data entry
  - To let the user indicate an "end" to the data
- Common sentinel values include:
  - **`STOP`**, **`-1`**, **`0`**, **`QUIT`**, and **`EXIT`**

Image from publicdomainpictures.net

Sentinel Loop Example

- Here's an example, where we ask the user to enter student names:

```
students = []
name = input("Please enter a student, or 'QUIT' to stop: ")


while name != "QUIT":
    students.append(name)
    name = input("Please enter a student, or 'QUIT' to stop: ")
```

Sentinel Loop Example

- Here's an example, where we ask the user to enter student names:

initialize the loop variable with user input

```
students = []
name = input("Please enter a student, or 'QUIT' to stop: ")
```

check for the termination condition

```
while name != "QUIT":
    students.append(name)
    name = input("Please enter a student, or 'QUIT' to stop: ")
```

get a new value for the loop variable

Sentinel Loop Example

- Here's an example, where we ask the user to enter student names:

make sure to tell the user how to stop entering data

```
students = []
name = input("Please enter a student, or 'QUIT' to stop: ")


while name != "QUIT":
    students.append(name)
    name = input("Please enter a student, or 'QUIT' to stop: ")
```

make sure to save the value before asking for the next one

Priming Reads

- This loop example uses a ***priming read***
  - We "prime" the loop by reading in information before the loop runs the first time

- We duplicate the line of code asking for input
  - Once before the loop
  - And then inside the loop

Time for…

LIVECODING!!!

Livecoding: Updated Grocery List

- Let's update our grocery list program to be as long as the user wants, using a while loop and a sentinel value of "STOP"
  - Print out the grocery list (item by item) at the end
- You will need to use:
  - At least one while loop (a sentinel loop)
  - Conditionals
  - A single list

Other List Operations

Length of a List

- To get the length of a list, use **`len()`**

```
>>> dogs = ["Lacey", "Kieran", "Ed"]
>>> len(dogs)
3
>>> len([2, 0, 1, 7])
4
```

- Why would we need the length of a list?

Announcements

- HW 3 is out on Blackboard now
  - Complete the Academic Integrity Quiz to see it
  - Due by Friday (Feb 24th) at 8:59:59 PM

- Pre Lab 5 Quiz will come out Friday @ 10 AM
  - Must be completed by 9 AM Monday morning