

Programming Foundations in Python

Adapted From: CMSC 201 Computer Science I for Majors

Lecture 08 – Lists

Last Class We Covered

- Constants
- More on **while** loops
 - Sentinel loops
 - Boolean flags

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
 - To understand the syntax of lists
- To be able to use the membership “**in**” operator
- To understand how functions and methods differ

Reminder About Loop Evaluations

- The conditional in a **while** loop is not checked until the body of the loop has finished


- How many times will this code print “Hello”?

```
count = 0
```

```
while count < 4:
```

```
    count += 1
```

```
    print("Hello")
```



The loop does NOT stop as soon as count's value is changed to 4

- “Hello” will be printed out four times

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

Lists vs Individual Variables

- Individual variables are like sticky notes
 - Works best when you only need a few
 - Good for storing different “pieces” of info
- Lists are like a checklist written on a single piece of paper
 - Best for storing a lot of related information in one place



Properties of a List

- Heterogeneous (multiple data types!)
- Contiguous (all together in memory)
- Ordered (remain in the order they were set in)
- Have instant (“random”) access to any element
- Are “mutable sequences of arbitrary objects”

Creating and Modifying Lists

Creating an Empty List

- To create an empty list, use square brackets:

```
newList = []
```

- This creates a list variable called **newList**, with no elements in the list

- (Sort of like a new checklist on a blank page)

- Similar to how we create an empty string:

```
newString = ""
```

List Function: `append()`

- The `append()` method lets us add items to the end of a list, increasing its size
- Syntax:
`listName.append(itemToAppend)`
- Useful for creating a list from flexible input
 - Can start with an empty list, and add items as the user requests them

Example of `append()`

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

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

while count < 4:
    userVal = int(input("Enter a number: "))
    # add value to the list
    values.append(userVal)
    count += 1
```

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

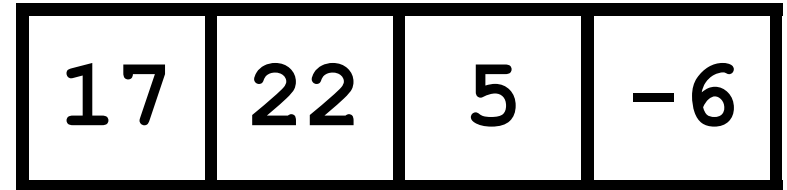
```
bash-4.1$ python numberList.py
```

```
Enter a number: 17
```

```
Enter a number: 22
```

```
Enter a number: 5
```

```
Enter a number: -6
```



```
values = [] # initialize empty list
count = 0
while count < 4:
    userVal = int(input("Enter a number: "))
    values.append(userVal)
    count += 1
```

List Function: `remove()`

- The `remove()` method 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

Example of `remove()`

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

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

```
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
```

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  
roster.remove("Bob")       # Bob is not in the roster
```

roster =

Alice	Andy	Ariel
-------	------	-------

ERROR

Quick Note – Methods vs Functions

- Functions include things like
 - `print()`
 - `input()`
 - `int()`
- Methods are a bit different, and include
 - `.append()`
 - `.remove()`

Quick Note – Methods vs Functions

- All you need to know for now is the difference between how they look when written out

`print("dogs!")`



Functions perform the action on the object inside the parentheses

This function prints out "dogs!"

`names.append("Ed")`



Methods perform the action on the object before the period

This method appends to names (In this example, it appends "Ed")

Editing List Contents

Mutating Lists

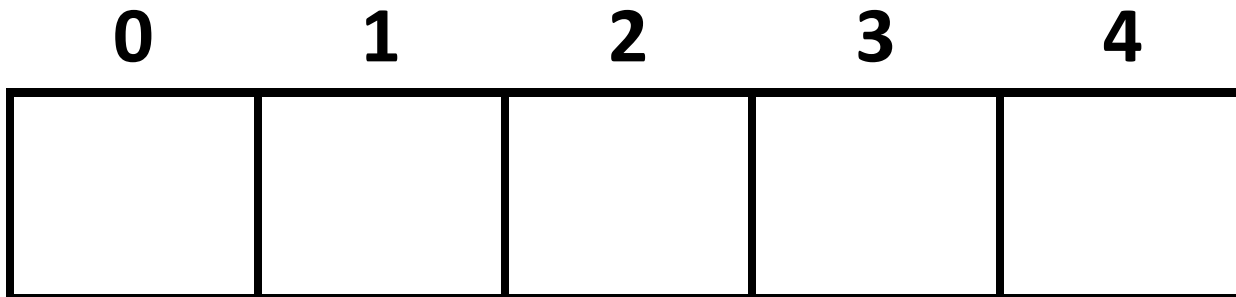
- Remember that lists are defined as “mutable sequences of arbitrary objects”
 - “Mutable” means we can change them
- So far, the only thing we’ve done has been to add or remove items from the list
 - But we can also edit the contents of a list “in place,” without having to add or remove

Using Lists: Individual Variables

- First, 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!



Square Bracket Syntaxes

- Can use `[]` to assign initial values

```
myList = [1, 3, 5]
```

```
words = ["Hello", "to", "you"]
```

– (Also called *initialization*)

- And to refer to individual elements of a list

```
>>> print(words[0])
```

```
Hello
```

```
>>> myList[0] = 4
```

Length of a List

- To get a list's length, use the function `len()`

```
>>> dogs = ["Lacey", "Kieran", "Al"]  
>>> len(dogs)  
3  
>>> len([2, 0, 1, 8])  
4
```
- Why would we need the length of a list?
 - We'll see in the next few slides!

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



List Example: Grocery List

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

Grocery List Code

```
MAX_GROC = 3
```

```
def main():  
    print("Welcome to the Grocery Manager 1.0")  
    groceryList = []    # initialize empty list  
  
    # get grocery items from the user  
    count = 0  
    while count < MAX_GROC:  
        item = input("Please enter an item: ")  
        groceryList.append(item)  
        count += 1  
  
main()
```

Grocery List Code

```
MAX_GROC = 3
```

```
def main():  
    print("Welcome to the Grocery  
    groceryList = []    # initialize  
  
    # get grocery items from the u  
    count = 0  
    while count < MAX_GROC:  
        item = input("Please enter an item: ")  
        groceryList.append(item)  
        count += 1
```

Is there a way to do this without using **count**? How else could we keep track of how *long* the list is?

```
main()
```

Grocery List Code

```
MAX_GROC = 3
```

```
def main():  
    print("Welcome to the Grocery Manager")  
    groceryList = []    # initialize list  
  
    # get grocery items from the user  
  
    while len(groceryList) < MAX_GROC:  
        item = input("Please enter an item: ")  
        groceryList.append(item)
```

This works just as well as **count**, but we don't need to keep track of any extra variables!

```
main()
```


Iterating Over a List

- Now that we have our grocery list, how do we ***iterate*** over each element of the list and print out its contents?
 - *Hint: Use a **while** loop and the **len()** function!*

```
index = 0
while index < len(groceryList):
    print( groceryList[index] )
    index += 1
```

Membership “**in**” Operator

Types of Operators in Python

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Membership Operators
- Bitwise Operators
- Identity Operators

what we're
covering now

Membership Operator Example

- What do you think this code does?

```
colors = ["orange", "purple", "red", "blue"]
```

```
guess = input("Please enter a color name: ")
```

```
while guess not in colors:
```

```
    print("You guessed wrong!")
```

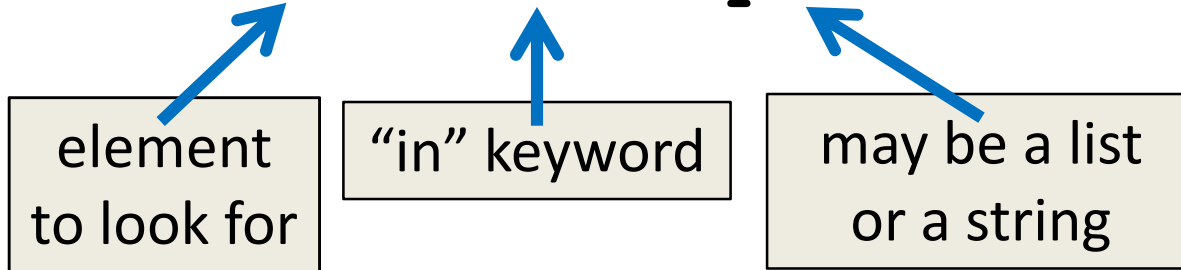
```
    guess = input("Guess again: ")
```

- Runs until the user guesses a color **in** the list
 - The membership operator can be very useful

Membership “in” Operator

- Syntax:

element **in** sequence



- Checks to see if element exists in sequence
 - Evaluates to either **True** or **False**
 - Can use it any time you have a conditional
- Can also use **not in** to test for absence

HW 06: Updated Grocery List

- Let's update our grocery list program to allow as many items 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

Image Sources

- Grocery bag (adapted from):
 - <https://www.flickr.com/photos/77106971@N00/1420127033>
- Sticky note:
 - <https://www.flickr.com/photos/winning-information/2325865367>
- Checklist:
 - <https://pixabay.com/p-1316848/>