COMP 141

Lists I

Rhodes College

Announcements

Reminders:

Midterm 2 on Wednesday, April 4th.

Practice from Last Time

- **Solutions in Box.com folder – string3Practice.py**

Introduction to Lists

- **List:** an object that contains multiple data items
  - Element: An item in a list
  - Format: `list = [item1, item2, etc.]`
  - Can hold items of different types
- **`print` function can be used to display an entire list**
- **`list()` function can convert certain types of objects to lists**

Introduction to Lists

A list of integers

```
even_numbers = [2, 4, 6, 8, 10]
```

A list of strings:

```
names = [‘Molly’, ‘Steven’, ‘Will’, ‘Alicia’]
```

A list holding different types:

```
info = [‘Alicia’, 27, 1550.87]
```

5

Example Using Lists

```
def main():
    # Create a list with some items.
    food = ['Pizza', 'Burgers', 'Chips']

    # Display the list.
    print('Here are the items in the food list:')
    print(food)


# Call the main function.
main()
```

Program Output

```
Here are the items in the food list:
['Pizza', 'Burgers', 'Chips']
```

6

Why use lists?

- Lists exist so programmers can store multiple related variables together.
- Useful when we don't know ahead of time how many items we are going to store.
  - Lists solve this problem because a single list can hold from zero to practically any number of items in it.

Basic list operations

- Lists are created using square brackets around items separated by commas.

```
mylist = [1, 2, 3]
numbers = [-9.1, 4.77, 3.14]
fred = ["happy", "fun", "joy"]
```

- Lists are accessed using indices/positions just like strings.
- Most (but not all) string functions also exist for lists.

Strings	Lists
string_var = "abc123"	list_var = [item1, item2, ...]
string_var = ""	list_var = [ ]
len("abc123") len(string_var)	len([3, 5, 7, 9]) len(list_var)
string_var[p] string_var[p:q]	list_var[p] list_var[p:q]
str3 = str1 + str2 str3 = "abc" + "def"	list3 = list1 + list2 list3 = [1, 2, 3] + [4, 5, 6]
"i" in "team" -> False	7 in [2, 4, 6, 8] -> False

One important difference

Strings are *immutable*

- **You can't change a string without making a copy of it.**

```
s = "abc"
s[0] = "A"       # illegal!
s = "A" + s[1:] # legal
```

Lists are *mutable*

- **Can be changed "in-place" (without explicit copying)**

```
L = [2, 4, 6, 8, 10]
L[0] = 15        # legal
L.append(26)     # legal
```

10

Compare Immutable and Mutable

- **How can we switch the first and last letter in a string?**
- **How can we switch the first and last items in a list?**

11

Three common ways to make a list

- Make a list that already has stuff in it:
  `lst = [4, 7, 3, 8]`
- Make a list of a certain length that has the same element in all positions:
  `lst = [0] * 4  #makes the list [0,0,0,0]`
  - Common when you need a list of a certain length ahead of time.
  - Uses the repetition operator, similarly to strings
- Make an empty list:
  `lst = []`
  - Common when you're going to put things in the list coming from the user or a file.

Simple list problems

- How would we write a function to convert a number from 1-12 into the corresponding month of the year as a string?

```
def getmonth(month):
```

Examples of Concatenation

```
a = [1,2,3]
b = [4,5,6]
c = a + b
print(c)    # prints [1, 2, 3, 4, 5, 6]

mylist = ['a','b','c']
other = ['d','e','f']
print(mylist + other)  #['a', 'b', 'c', 'd', 'e', 'f']
```

14

Simple list problems

- What does this code do?

```
lst = [2] * 3
lst2 = [4] * 2
lst3 = lst + lst2
for x in range(0, len(lst3), 2):
    lst3[x] = -1
```

Examples of List Slices

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

numbers[2: ]   #[3, 4, 5, 6, 7, 8, 9, 10]

numbers[:-2]   #[1, 2, 3, 4, 5, 6, 7, 8]

numbers[1:8:2] #[2, 4, 6, 8]

numbers[5::-1] #[6, 5, 4, 3, 2, 1]

numbers[::-1]  #[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

16

Program 8-8 (total_list.py)

```
1  # This program calculates the total of the values
2  # in a list.
3
4  def main():
5      # Create a list.
6      numbers = [2, 4, 6, 8, 10]
7
8      # Create a variable to use as an accumulator.
9      total = 0
10
11     # Calculate the total of the list elements.
12     for value in numbers:
13         total += value
14
15     # Display the total of the list elements.
16     print('The total of the elements is', total)
17
18 # Call the main function.
19 main()
```

Can iterate by item in the list

Program Output

```
The total of the elements is 30
```



```
# The NUM_DAYS constant holds the number of
# days that we will gather sales data for.
NUM_DAYS = 5

def main():
    # Create a list to hold the sales
    # for each day.
    sales = [0] * NUM_DAYS

    # Create a variable to hold an index.
    index = 0

    print('Enter the sales for each day.')

    # Get the sales for each day.
    while index < NUM_DAYS:
        print('Day #', index + 1, ': ', sep='', end='')
        sales[index] = float(input())
        index += 1

    # Display the values entered.
    print('Here are the values you entered:')
    for value in sales:
        print(value)

# Call the main function.
main()
```

Using the repetition operator to initialize list

Program Output (with input shown in bold)

```
Enter the sales for each day.
Day #1: 1000 Enter
Day #2: 2000 Enter
Day #3: 3000 Enter
Day #4: 4000 Enter
Day #5: 5000 Enter

Here are the values you entered:
1000.0
2000.0
3000.0
4000.0
5000.0
```

Class Practice

Get the file March28.py from my Box.com code directory. It has the main function written for you and stubs for 2 other functions that you will need to write.

findAverage(numbers) – will return the average of all the numbers in the list

countNumbers(numbers, average) - will return 2 values; it counts the number of above average and below average numbers in a list

Finding Items in Lists with the `in` Operator

- You can use the **`in`** operator to determine whether an item is contained in a list
  - General format: *`item in list`*
  - Returns `True` if the item is in the list, or `False` if it is not in the list
- Similarly you can use the **`not in`** operator to determine whether an item is not in a list

Rhodes College

Example Using `in` Operator

```
# This program demonstrates the in operator
# used with a list.

def main():
    # Create a list of product numbers.
    prod_nums = ['V475', 'F987', 'Q143', 'R688']

    # Get a product number to search for.
    search = input('Enter a product number: ')

    # Determine whether the product number is in the list.
    if search in prod_nums:
        print(search, 'was found in the list.')
    else:
        print(search, 'was not found in the list.')

# Call the main function.
main()
```

21

List Methods and Useful Built-in Functions

- **`append(item)`**: used to add items to a list – *`item`* is appended to the end of the existing list
- **`index(item)`**: used to determine where an item is located in a list
  - Returns the index of the first element in the list containing `item`
  - Raises `ValueError` exception if *`item`* not in the list

22

find() doesn't exist for lists

- list_var.index(item)
- Searches left to right, returns position where found, but crashes if not found.
- Let's build an algorithm that replicates find(), but works for lists (returns -1 if not found).

Example Using Append

```
def main():

    infile = open("randomNums.txt", 'r')
    numbers = []
    for line in infile:
        numbers.append(int(line))
    print(numbers)


main()
```

Output

[62, 57, 35, 27, 45, 44, 46, 68, 86, 27, 88, 33, 11, 61, 64, 45, 56, 9, 33, 32, 56, 63, 24, 26, 100, 95, 62, 10, 87, 58, 69, 54, 75, 41, 22, 93, 82, 16, 92, 49, 6, 71, 85, 59, 56, 22, 3, 50, 1, 20, 54, 18, 27, 78, 17, 7, 41, 83, 92, 38, 5, 64, 60, 92, 15, 26, 57, 39, 80, 41, 67, 56, 24, 77, 28, 90, 24, 72, 2, 46, 75, 53, 58, 47, 50, 18, 40, 65, 24, 58, 4, 58, 81, 40, 6, 77, 85, 86, 68, 63]

24

List Methods and Useful Built-in Functions (cont'd.)

- **`insert(index, item)`**: used to insert *item* at position *index* in the list
- **`sort()`**: used to sort the elements of the list in ascending order
- **`remove(item)`**: removes the first occurrence of *item* in the list
- **`reverse()`**: reverses the order of the elements in the list

25

Program 8-5 (insert_list.py)

```
1  # This program demonstrates the insert method.
2
3  def main():
4      # Create a list with some names.
5      names = ['James', 'Kathryn', 'Bill']
6
7      # Display the list.
8      print('The list before the insert:')
9      print(names)
10
11     # Insert a new name at element 0.
12     names.insert(0, 'Joe')
13
14     # Display the list again.
15     print('The list after the insert:')
16     print(names)
17
18 # Call the main function.
19 main()
```

Program Output

```
The list before the insert:
['James', 'Kathryn', 'Bill']
The list after the insert:
['Joe', 'James', 'Kathryn', 'Bill']
```

26

Program 8-6 (remove_item.py)

```
1  # This program demonstrates how to use the remove
2  # method to remove an item from a list.
3
4  def main():
5      # Create a list with some items.
6      food = ['Pizza', 'Burgers', 'Chips']
7
8      # Display the list.
9      print('Here are the items in the food list:')
10     print(food)
11
12     # Get the item to change.
13     item = input('Which item should I remove? ')
14
15     try:
16         # Remove the item.
17         food.remove(item)
18
19         # Display the list.
20         print('Here is the revised list:')
21         print(food)
22
23     except ValueError:
24         print('That item was not found in the list.')
25
26 # Call the main function.
27 main()
```

Program Output (with input shown in bold)

```
Here are the items in the food list:
['Pizza', 'Burgers', 'Chips']
Which item should I remove? Burgers Enter
Here is the revised list:
['Pizza', 'Chips']
```

27

List Methods and Useful Built-in Functions (cont'd.)

- **`del` statement**: removes an element from a specific index in a list
  - General format: `del list[i]`
- **`min` and `max` functions**: built-in functions that returns the item that has the lowest or highest value in a sequence
  - The sequence is passed as an argument
- **`sum` function**: built-in functions that returns the total of all the values in a sequence
  - The sequence is passed as an argument

28

Example Using `del`, `min`, `max`, and `sum` functions

```
my_list = [5, 4, 3, 2, 50, 40, 30]
print("Before Deletion:", my_list)
del my_list[2]
print("After Deletion:", my_list)

print("The lowest value is", min(my_list))
print("The highest value is", max(my_list))
print("The sum of values in my list is", sum(my_list))

alpha_list = ['a','b','c','d']
print("The lowest value is", min(alpha_list))
print("The highest value is", max(alpha_list))
# You cannot take the sum of a list that has strings in it
```

Output

```
Before Deletion: [5, 4, 3, 2, 50, 40, 30]
After Deletion: [5, 4, 2, 50, 40, 30]
The lowest value is 2
The highest value is 50
The sum of values in my list is 131
The lowest value is a
The highest value is d
```

29

Practice

Write a program that randomly generates 20 integers between 1 and 50, and stores them in a list. Print out the **lowest** and the **highest** numbers in your list, as well as the **sum** of all the numbers in the list.

Write a function that prints out sums of adjacent pairs of numbers in the list (don't use sliding window; use indices)

Hint: You don't need the sliding window technique; instead, use math with list indices.

Write a function that takes a list and shifts all the elements in the list one spot to the left, without using slices! (the left-most element disappears)

Example: [1, 2, 3, 4, 5] turns into [2, 3, 4, 5, 5]

30