CS65: Introduction to Computer Science

Sequence: List (continued)
Sequence: Tuple

Md Alimoor Reza
Assistant Professor of Computer Science

Announcement

Assignment #1 has been released yesterday and it will be due by October 31st Friday for all sections

Lab #5 has been released on October 21st and it will be due by October 25th Saturday for all sections

Text Reading

https://www.brianheinold.net/python/python_book.html#chapter_lists

Lists

Quick Review: Last Lecture

- Sequence
  - String
  - List
    - List indexing
    - List updating
    - Adding item to a list
  - Tuple

Review: Accessing items in a list

You can access individual items in a list using their **index**, which is their *position* in the list

indices start at 0

```
1  # T. Urness
2  # intro to lists
3
4  beatles = ["John", "Paul", "Ringo", "George"]
5  print(beatles[2])
6
7  rainfall_amounts = [0.0, 0.3, 0.71, 0.0, 0.32, 1.1, 0.4]
8  print(rainfall_amounts[1])
```

Review: Lists *index* notation

You can treat an item in a list just like any other variable:

```
rainfall_amounts = [0.1, 0.3, 0.71, 0.0, 0.32, 1.1, 0.4]
total_weekend_rainfall = rainfall_amounts[0] + rainfall_amounts[6]
print(total_weekend_rainfall)
```

Lists3.py

Review: Updating items in a list

You can print out the entire contents of a list by using `print`

You can also use list notation to change values in a list

```
rainfall_amounts = [0.0, 0.3, 0.71, 0.0, 0.32, 1.1, 0.4]
rainfall_amounts[2] = 0.65
print(rainfall_amounts)
```

Lists5.py

```
>>> %Run Lists5.py
  [0.0, 0.3, 0.65, 0.0, 0.32, 1.1, 0.4]
```

Review: Loops and Lists

What will this code do?

Lists6.py

```
glucose_levels = [140, 119, 153, 135, 160]
for i in range(len(glucose_levels)):
    print(glucose_levels[i])
```

Review: Inserting Items in a List

- Appending elements in an empty list with **.append()** method

```
# building list with append() function
num_list = []

num_list.append(2)
print("num_list: ", num_list)

num_list.append(4)
print("num_list: ", num_list)

num_list.append(6)
print("num_list: ", num_list)
```

```
>>> %Run lec11.py

  num_list:  [2]
  num_list:  [2, 4]
  num_list:  [2, 4, 6]
```

Review: Inserting Specific Items in a List Iteratively

- Appending odd numbers (from 1 to 20) in an empty list with **.append()** method iteratively using **for loop**

```
# building list with append function using loop
# insert only odd numbers
num_list = []

for num in range(1, 20):

    if (num % 2 != 0):

        num_list.append(num)

print("num_list: ", num_list)
```

```
>>> %Run lec11.py

  num_list:  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple

Exercise#1 (Warm-up)

- Appending **multiples of 5** in an empty list with **append()** method iteratively using **for loop**
  - Prompt the user to enter two numbers
    - first number is the lower limit, eg, **1**
    - second number is the upper limit, eg, **16**
  - If user enters **1** and **16**, you should append only **5, 10, 15**

```
# building list with append function using loop
# insert only odd numbers
num_list = []

for num in range(1, 20):

    if (num % 2 != 0):

        num_list.append(num)

print("num_list: ", num_list)
```

```
>>> %Run lec11.py
  num_list:  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Exercise#1: My Solution

- Appending **multiples of 5** in an empty list with **append()** function iteratively using **for loop**

```
# building list with append function using loop
# insert only multiples of 5
lower_limit = int(input("enter the lower limit: "))
upper_limit = int(input("enter the upper limit: "))

num_list = []



for num in range(lower_limit, upper_limit):

    if (num % 5 == 0):

        num_list.append(num)

print("num_list: ", num_list)
```

```
>>> %Run lec11.py

  enter the lower limit: 3
  enter the upper limit: 22
  num_list:  [5, 10, 15, 20]

>>> %Run lec11.py

  enter the lower limit: 5
  enter the upper limit: 49
  num_list:  [5, 10, 15, 20, 25, 30, 35, 40, 45]
```

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple

Things you can do with lists: `len`

`len` is a *function* that will return the number of elements in a list

```
3
4  friend_list = ["Max", "Fred", "Nancy", "Waldo"]
5  print(len(friend_list))
6
7  grades = [97.4, 99.0, 87.2, 67.2, 88.8]
8  print(len(grades))
9
10 record = ["W", "W", "W", "L", "W","L"]
11 print(len(record))
```

ListsLen.py

Things you can do with lists: `max`

`max` is a *function* that will return the maximum number within a list (numeric lists only)

```
2  # list functions: max
3
4  grades = [97.4, 99.0, 87.2, 67.2, 88.8]
5  print(max(grades))
6
7  rainfall = [0.0, 0.0, 1.2, 2.4, 0.8]
8  print(max(rainfall))
9
10 daily_steps_taken = [7312, 5074, 5135, 3685, 4405, 7888, 12011]
11 print(max(daily_steps_taken))
12
```

ListsMax.py

Things you can do with lists: `min`

`min` is a *function* that will return the minimum number within a list (numeric lists only)

```
2  # list functions: min
3
4  grades = [97.4, 99.0, 87.2, 67.2, 88.8]
5  print(min(grades))
6
7  rainfall = [0.0, 0.0, 1.2, 2.4, 0.8]
8  print(min(rainfall))
9
10 daily_steps_taken = [7312, 5074, 5135, 3685, 4405, 7888, 12011]
11 print(min(daily_steps_taken))
```

ListsMin.py

Things you can do with lists: `sum`

`sum` is a *function* that will return the total (added) number of all of the elements within a list (numeric lists only)

```
2  # list functions: sum
3
4  grades = [97.4, 99.0, 87.2, 67.2, 88.8]
5  print(sum(grades))
6
7  rainfall = [0.0, 0.0, 1.2, 2.5, 0.8]
8  print(sum(rainfall))
9
10 daily_steps_taken = [7312, 5074, 5135, 3685, 4405, 7888, 12011]
11 print(sum(daily_steps_taken))
```

ListsSum.py

Exercise #2

1. Create an empty list
2. Write a loop that will execute three times
3. Inside the loop, prompt the user to enter a float number
4. Append the float number entered to a list
5. Outside of the loop, print out the min, max, sum, and AVERAGE of the numbers entered (use the functions min, max, sum, len, in your program)

```
>>> %Run Exercise1.py
please enter a number: 2.25
please enter a number: 3.75
please enter a number: 6.9
max is 6.9
min is 2.25
average is 4.3
```

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple

List functions vs List methods

Some list operations use the function notation

```
sum(list)
max(list)
len(list)
```

While some list operation require to use the dot notation.

```
list.reverse()
list.sort()
list.index()
```

Whenever you use a function with dot notation, it is technically called a *method*.

Exercise #3

Create a list filled with numbers. (you can make up your own numbers)

```
my_list = [6.86, 3.19, 2.27, 2.36, 1.63, 1.93, 4.93]
```

Print out the list:

```
print(my_list)
```

Experiment with executing the following lines:

```
my_list.reverse()

my_list.sort()
```

Question: in which order does sort() work (highest to lowest or lowest-to-highest). Can you figure out a way to make it go the opposite way?

Example: *index* method

The `.index()` method will return the first index number where an element occurs

```
2  # index of an element
3
4  rainfall_amounts = [0.0, 0.3, 0.71, 0.0, 0.32, 1.1, 0.4]
5  print(rainfall_amounts.index(0.3)) #which index can I find 0.3 at?
```

ListsIndex.py

Example: *index* method

The `.index()` method will return the first index number where an element occurs

```
2  # index of an element
3
4  rainfall_amounts = [0.0, 0.3, 0.71, 0.0, 0.32, 1.1, 0.4]
5  print(rainfall_amounts.index(0.3)) #which index can I find 0.3 at?
```

```
>>> %Run index.py
 1
```

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple

Testing membership in a list

You can test if a value is in a list using the **`in`** operator.

```
2  # in operator
3
4  grocery_list = ["apples", "bananas", "carrots"]
5  print ("donuts" in grocery_list)
```

```
False
```

Testing membership in a list

You can test if a value is in a list using the **`in`** operator.

How can this be useful?

```
4  grocery_list = ["apples", "bananas", "carrots"]
5  # print ("donuts" in grocery_list)
6
7  item = input("enter an item to add to the list: ")
8  if item in grocery_list:
9      print("you already have", item,"in the list")
10 else:
11     grocery_list.append("item")
12
13 print(grocery_list)
```

ListIn.py

Challenge: there is a mistake in this code. Can you find it?

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple

Removing Items: `.remove()` Method

- When you know the *value* of the item you want to delete from the list, use the ***remove*** method. This will only delete the first instance of the value.

```
# deleting an item from the list

num_list = [10, -1, -2, -3, 20, -4, 30]

print("Initial num_list is ", num_list)

num_list.remove(-1)

print("After removing -1 num_list is ", num_list)

num_list.remove(-2)

print("After removing -2 num_list is ", num_list)
```

```
>>> %Run lec12.py

  Initial num_list is  [10, -1, -2, -3, 20, -4, 30]
  After removing -1 num_list is  [10, -2, -3, 20, -4, 30]
  After removing -2 num_list is  [10, -3, 20, -4, 30]
```

- **Heads up!** the size of the list will change! Be careful when you are removing items using for loop

Removing Items: `.remove()` Method

- When you know the *value* of the item you want to delete from the list, use the ***`remove`*** method. Trying to remove a value that isn't in the list will result in an error.

```
# Trying to remove a value that isn't in the list will result in an error:
rainfall = [0.0, 1.2, 2.5, 0.8, 0.0]
rainfall.remove(-1.0)
```

```
Traceback (most recent call last):
  File "/Users/000617123/Desktop/CS65code/Day08/remove_by_value.py", line 12, in <module>
    rainfall.remove(-1.0)
ValueError: list.remove(x): x not in list
```

Removing Items: `.remove()` Method

- When you know the *value* of the item you want to delete from the list, use the ***remove*** method. Trying to remove a value that isn't in the list will result in an error.

```
num_list = [10, -2, -2, -2, 20, -4, 30]

num_list.remove(-2)
num_list.remove(-2)
num_list.remove(-2)
print("After removing all -2s num_list is ", num_list)
```

```
>>> %Run lec12.py

  After removing all -2s num_list is  [10, 20, -4, 30]
```

- **Heads up!** If the item is not in the list, you will get an error

Removing Items: `.pop()` Method

- When you don't want the *value* of the item you want to delete from the list, use can use index/position of the item using *pop* method. This will only delete the item located at that index.

```
# deleting an item from the list using .pop() method

num_list = [10, -1, -2, -3, 20, -4, 30]

print("Initial num_list is ", num_list)

num_list.pop(0)

print("After removing item at index 0 num_list is ", num_list)

num_list.pop(1)

print("After removing item at index 1 num_list is ", num_list)
```

```
>>> %Run lec12.py

  Initial num_list is  [10, -1, -2, -3, 20, -4, 30]
  After removing item at index 0 num_list is  [-1, -2, -3, 20, -4, 30]
  After removing item at index 1 num_list is  [-1, -3, 20, -4, 30]
```

Removing Items: `del` Keyword

- You can also use index/position of the item using ***`del`*** keyword. This will only delete the item located at that index.

```
# deleting an item from the list

num_list = [10, -1, -2, -3, 20, -4, 30]

print("Initial num_list is ", num_list)

del num_list[0]

print("After removing item at index 0 num_list is ", num_list)

del num_list[1]

print("After removing item at index 1 num_list is ", num_list)
```

```
>>> %Run lec12.py

  Initial num_list is  [10, -1, -2, -3, 20, -4, 30]
  After removing item at index 0 num_list is  [-1, -2, -3, 20, -4, 30]
  After removing item at index 1 num_list is  [-1, -3, 20, -4, 30]
```

Exercise #4

Given a list of numbers, can you think of a way to remove all of the numbers from the list that are == 0?

Exercise #5

A programmer wants to write a program that allows the user to enter a rain amount and then remove that if it is in the list.

However, when they test this code, some of the tests cause the program to crash.

Explain the problem and write the code that will fix it (Hint: you might need to add an if statement).

```
rainfall_amounts = [0.0, 0.3, 0.71, 0.0, 0.32,
1.1, 0.4]
val_to_remove = float(input("Enter a value to
remove: "))
rainfall_amounts.remove(val_to_remove)
print(rainfall_amounts)
```

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple

List Slicing

- Slice is a span of items that are taken from a list (or any sequence)
  - Span is a list containing copies of elements from **start** up to, but not including, **end**
  - Format: *`list_variable_name`***`[start : end]`**
    - **`start :`** starting index — if not specified 0 is used
    - **`end :`** end index — if not specified ***len****(list)* is used

List Slicing

- Format: *list_variable_name*[**start : end**]
  - **start :** starting index — if not specified 0 is used
  - **end :** end index — if not specified ***len****(list)* is used

```
num_list = [10, 11, 12, 13, 14, 15]

print("num_list ", num_list)
print("num_list[0] ", num_list[0])
print("num_list[:1] ", num_list[:1])
print("num_list[1:4] ", num_list[1:4])
print("num_list[1:] ", num_list[1:])
```

```
>>> %Run lec12.py

num_list  [10, 11, 12, 13, 14, 15]
num_list[0]  10
num_list[:1]  [10]
num_list[1:4]  [11, 12, 13]
num_list[1:]  [11, 12, 13, 14, 15]
```

Extra: Consider “parallel” lists

Parallel lists are two or more list variables that contain contain related data – the elements of each list are associated with each other based on their position in the list.

ParallelLists.py

```
# parallel lists

my_list = ["apples", "grapes", "bananas"]
quantity = [ 100, 23, 84]

# print out the value and the quantity
for i in range(len(my_list)):
    print(my_list[i], quantity[i])
```

```
apples 100
grapes 23
bananas 84
```

Extra: Lists and random values

What will this code do?

RandomLists.py

```
2  # random lists
3
4  import random
5
6  random.seed(0)
7  my_list = []
8  for i in range(10):
9      my_list.append(random.randint(1,100))
10
11 print(my_list)
12
```

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple

Tuple: another type of a sequence

- We **cannot change/modify** items in a tuple after its creation
  - This property is called **immutability**

- Items are accessed by index (similar to List or String)

Sequence	Example	Syntax	Accessing
String	my_str = “My name is walle”	within enclosing quotation marks, ie, “ ” or ‘ ’	my_str[0] my_str[1]
List	my_list = [1, 2, “a”, “abs”]	within enclosing brackets [ ] and separated by commas	my_list[0] my_list[1]
Tuple	my_tuple = (1, 2, “a”, “abs”)	Within enclosing parenthesis ( ) and separated by commas	my_tuple[0] my_tuple[1]

Mutable Property of List

```
# ---------- mutability of List ------------

my_list = [1, 2, "a", "abs"]

for i in range(len(my_list)):
    print(my_list[i])


# trying to update a location with a new value

my_list[1] = 3

print("modified value of list ", my_list[1])
```

```
>>> %Run lec14demo.py

  1
  2
  a
  abs
  modified value of list  3
```

Immutable Property of Tuple

```
# ---------- immutability of Tuple -------------
my_tuple = (1, 2, "a", "abs")

for i in range(len(my_tuple)):
    print(my_tuple[i])


# trying to update a location with a new value

my_tuple[1] = 3

print("modified value of tuple ", my_tuple[1])
```

```
1
2
a
abs
Traceback (most recent call last):
  File "/Users/reza/Class_and_Research/drake_teaching/CS65/c
>
    my_tuple[1] = 3
TypeError: 'tuple' object does not support item assignment
```

Tuple

- **Tuple examples**

```
# tuple examples

tup1 = ()

tup2 = (1,)          # one-tuple needs a comma in Python

tup3 = ("Georg Cantor", "Bertrand Russell", "Kurt Godel")

tup4 = (True, False, True, False)

tup5 = ([1, 2, 3], [4, 5, 6])

tup6 = ((1, 2, 3), (4, 5, 6))
```

- Exercise: Try the examples above in Thonny. Find the items you can modify and which the ones you cannot

Work on Lab #5 which is due on Saturday, October 25th

Text Reading

https://analytics.drake.edu/~reza/teaching/cs65_fall25/readings/Chapter9.html