CSCI-UA-0002

Intro to Computer Programming (No Prior Experience)

Module 8: Lists

Professor Emily Zhao

Section 008
T/R 12:30-1:45PM

Section 012
T/R 4:55-6:10PM

Agenda

- Review Module 8
- Practice Problems

Your Questions

→ Can you go over `try/except`?

→ What is the difference between `del` and `remove()`?

→ What data types do `min()` and `max()` work on?

Data types

```
x = 5                # integer
y = 5.0              # float
z = 1j               # complex
```

Numeric Types

```
q = True             # bool
```

Boolean Type

```
a = “hello”          # str
b = [1, 2, 3]        # list
```

Sequence Types

What’s the key difference between a string and a list (which are both sequence types)?

What’s the key difference between a string and a list (which are both sequence types)?

→ Strings are immutable whereas lists can be changed!

Sequence Operations

Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

Indexing
Length
Min/Max
Concatenation
Repetition
Slicing
Occurrences

`min()` and `max()`

```
# min() and max() operate on any data type
# that supports comparison operations

# String:
name = "Emily"
print(min(name)) # E

# List of Ints:
ints = [2, 4, 3]
print(max(ints)) # 4

# List of Floats:
floats = [1.4, 10.2, 3.5]
print(min(floats)) # 1.2

# List of Ints and Floats:
nums = [3.14, 5, 2]
print(min(nums)) # 2

# List of Strings:
names = ["Min", "Mabel", "Peter"]
print(min(names)) # Mabel

# Mixed List:
mixed_list = ["Mabel", 3.14, True, 42]
print(max(mixed_list)) # TypeError: '>' not supported between instances of 'float' and 'str'
```

String methods

`.isalnum()`	True if all characters are alphanumeric
`.isalpha()`	True if all characters are alphabetic
`.islower()`	True is all alpha characters are lower
`.isspace()`	True if all characters are “whitespace”
`.isupper()`	True if all alpha characters are upper
`.isdigit()`	True if all characters in a string are digits from 0 to 9
`.isnumeric()`	True if all the characters in a string are numeric, not just digits (0-9), for example "½" or "١٢٣"

String modifications

`.lower()`	Returns a lowercase version of the string
`.upper()`	Returns an uppercase version of the string
`.rstrip()`	Removes whitespace at end of string
`.lstrip()`	Removes leading whitespace characters
`.capitalize()`	Returns a copy of the string with the first character capitalized
`.title()`	Returns a copy of the string with the first character of each word capitalized
`.swapcase()`	Returns a copy of the string where case is swapped among all alpha characters

List Methods

`.append(item)`	Appends an item to the end of the list
`.insert(index, item)`	Inserts an item at a specific position in the list.
`.remove(item)`	Removes the first occurrence of an item from the list
`.pop(index)`	Removes and returns an item at the specified index
`.index(item)`	Returns the index of the first occurrence of an item
`.sort()`	Sorts the list in place
`.reverse()`	Reverses the order of elements in the list
`.count(item)`	Counts the number of occurrences of an item in the list

Concatenation

You can use the concatenation operation ("+") to ask Python to combine lists, much like how you would combine strings. Example:

```
greeting = "hello" + " " + "Emily!"
my_list = [1, 2, 3] + [99, 100, 101]

print (greeting)
print (my_list)
```

```
hello Emily!
[1, 2, 3, 99, 100, 101]
```

Concatenation

```
names = ["Emily", "Michael", "Yankee"]
print(names)

names += "Mary"
print(names)
```

```
['Emily', 'Michael', 'Yankee']
['Emily', 'Michael', 'Yankee', 'M', 'a', 'r', 'y']
```

Concatenation

```
mylist = ["hello"]
mylist += ["world!"]
mylist += ["goodbye"]
print (mylist)
```

```
['hello', 'world!', 'goodbye']
```

Appending

```
names = ["Emily", "Michael", "Yankee"]
print(names)

names.append("Mary")
print(names)
```

```
['Emily', 'Michael', 'Yankee']
['Emily', 'Michael', 'Yankee', 'Mary']
```

Removing Items from a List

- You can remove an item from a list by using the “remove” method. Here’s an example:

```
prices = [3.99, 2.99, 1.99]
prices.remove(2.99)
print (prices)
```

```
[3.99, 1.99]
```

- Note that you will raise an exception if you try and remove something that is not in the list. Make sure to test to see if it is in the list first using the “in” operator (or use a try / except block to catch any errors you might raise).

Removing Items from a List

- You can also remove an item from a list based on its index position. We can do this using the 'del' keyword, like this:

```
prices = [3.99, 2.99, 1.99]
del prices[0] # remove whatever is at slot 0
print (prices)
```

```
[2.99, 1.99]
```

Mutability

```
name = "Bobby"
name[4] = "i"
print(name)

roster = ["Anne", "Peter", "Stacy", "Justin", "Bobby"]
roster[4] = "Bobbi"
print(roster)
```

Mutability

```
name = "Bobby"
name[4] = "i"
print(name)
```

```
    name[4] = "i"
TypeError: 'str' object does not support item assignment
```

```
roster = ["Anne", "Peter", "Stacy", "Justin", "Bobby"]
roster[4] = "Bobbi"
print(roster)
```

```
['Anne', 'Peter', 'Stacy', 'Justin', 'Bobbi']
```

Mutability

```
name = "Bobby"

for c in name:
    c = "B"

print(name)
```

```
roster = ["Anne", "Peter", "Stacy"]

for name in roster:
    name = "Bobby"

print(roster)
```

Mutability

```
name = "Bobby"

for c in name:
    c = "B"

print(name)
```

```
Bobby
```

```
roster = ["Anne", "Peter", "Stacy"]

for name in roster:
    name = "Bobby"

print(roster)
```

```
['Anne', 'Peter', 'Stacy']
```

Mutability

```
roster = ["Anne", "Peter", "Stacy"]

for name in roster:
    name = "Bobby"

print(roster)
```

```
['Anne', 'Peter', 'Stacy']
```

```
roster = ["Anne", "Peter", "Stacy"]

for i in range(0, len(roster)):
    roster[i] = "Bobby"

print(roster)
```

```
['Bobby', 'Bobby', 'Bobby']
```

→ You have to change lists with indexes

Mutability

```
name = "Bobby"

print(name.lower(), end="")
print(name.upper(), end="")
print(name)
```

#1: bobbyBOBBYBOBBY
#2: bobbyBOBBYBobby
#3: bobbyBobbyBobby
#4: BobbyBobbyBobby

Mutability

```
name = "Bobby"

print(name.lower(), end="")
print(name.upper(), end="")
print(name)
```

#1: bobbyBOBBYBOBBY

#2: bobbyBOBBYBobby

#3: bobbyBobbyBobby

#4: BobbyBobbyBobby

bobbyBOBBYBobby

Programming Challenge: Sales Tax

Given the following list of prices, write a program that modifies the list to include 7% sales tax

```
prices = [1.99, 2.99, 3.99, 4.99, 5.99, 6.99]
```

```
prices = [1.99, 2.99, 3.99, 4.99, 5.99, 6.99]

for i in range(len(prices)):
    prices[i] *= 1.07

print(prices)
```

Programming Challenge: Class Curve

Apply a class "curve" to each score in the grades list.

```
grades = [90, 100, 70, 45, 76, 84, 93, 21, 36, 99, 100]
```

The class curve is as follows:

93 or above: no curve
80 to 90: +2 points
70 to 80: +5 points
Lower than 70: +8 points

```
grades = [90, 100, 70, 45, 76, 84, 92, 21, 36, 99, 100]

for i in range(len(grades)):
    if grades[i] >=  93:
        continue
    elif grades[i] > 80:
        grades[i] += 2
    elif grades[i] > 70:
        grades[i] += 5
    else:
        grades[i] += 8

print(grades)
```

List Mechanics

- List variables are considered “references”
- This means that they “reference” or “point” to a specific region of your computer’s memory. This behavior can cause some interesting side effects. For example, the following two list variables refer to the same list in memory.

```
mylist1 = [1,2,3]
mylist2 = mylist1

print (mylist1)
print (mylist2)
```

```
[1, 2, 3]
[1, 2, 3]
```

List Mechanics

- This means that you can change one of the lists and the change will be reflected in the other.

```
mylist1 = [1,2,3]
mylist2 = mylist1

mylist1[0] = 999

print (mylist1)
print (mylist2)
```

```
[999, 2, 3]
[999, 2, 3]
```

Copying a List

- Python will only create new lists when you use [] syntax to define a list for the first time
- You can take advantage of this behavior to create true copies of your list objects. For example:

```
mylist1 = [1,2,3]
mylist2 = [] + mylist1

mylist1[0] = 999

print (mylist1)
print (mylist2)
```

```
[999, 2, 3]
[1, 2, 3]
```

Creating Lists

- You can create an empty list with no elements using the following syntax:

```
mylist = []
```

- Sometimes you want to create a list that contains a certain number of “pre-set” elements. For example, to create a list with 10 elements that are all set to zero you could do the following:

```
mylist = [0] * 10
```

Creating Lists

- You can also create lists using the range() function. For example, to create a list of all even numbers between 0 and 100 you can do the following:

```
even_numbers = list(range(0,100,2))
```

Programming Challenge

Given these two lists:

```
a = [1,2,3,4,5,1]
b = [2,3,10,11,12,1]
```

Write a program that finds all elements that exist in both lists (i.e. the integer 2 exists in both lists). Store your result in a list and print it out to the user.

```
a = [1,2,3,4,5]
b = [2,3,10,11,12,1]
c = [] # list to hold shared numbers

for num in a:
    # check to see if it was already added
    if num in c:
        continue
    elif num in b:
        c.append(num)

print(c)
```

Sorting list items

- You can have Python sort items in a list using the sort() method. Here's an example:

```
my_list = ['pie', 'cake', 'pizza']
my_list.append('apple')
print (my_list)

my_list.sort()
print (my_list)
```

```
>> ['pie', 'cake', 'pizza', 'apple']
>> ['apple', 'cake', 'pie', 'pizza']
```

Programming Challenge: Alphabetizing Names

- Write a program that continually prompts a user to enter in a series of first names.
- Store these first names in a list.
- Print out the list sorted in alphabetical order by first name at the end of your program

```
names = []
while True:
    name = input("Enter a first name (type 'end' to stop): ")
    if name == "end":
        break
    else:
        names.append(name)

print(names)
names.sort()
print(names)
```

Programming Challenge: Price Lookup

Given that the following lists match up with one another (i.e. the product at the first position of the products list matches the price at the first position in the prices list), write a product price lookup program.

```
products = ['peanut butter', 'jelly', 'bread']
prices = [3.99, 2.99, 1.99]
```

```
Enter a product name: jelly
Price for jelly: 2.99
```

```
products = ['peanut butter', 'jelly', 'bread']
prices = [3.99, 2.99, 1.99]

while True:
    prod = input("Enter a product name (or 'end'): ")

    if prod == "end":
        break

    # figure out the index of the item
    if prod in products:
        index = products.index(prod)
        print("Price for", prod + ":", prices[index])
    else:
        print(prod, "not found!")
```

```
products = ['peanut butter', 'jelly', 'bread']
prices = [3.99, 2.99, 1.99]

while True:
    prod = input("Enter a product name ('end' to end): ")
    if prod == "end":
        break
    if prod in products:
        index = products.index(prod)
        print("Price for", prod + ":", prices[index])
    else:
        print("Product not found!")
```

Homework

- Assignment #7 (due Thursday)
- Self-Paced Learning Module #9 (due next Tues)
- Quiz #9 (due next Tues)