ECS 32A - Lists

Aaron Kaloti

UC Davis - Summer Session #1 2020

UC DAVIS
COMPUTER SCIENCE

Lists

- As the name implies, a **list** is a list of data.

```
>>> my_first_list = ["Aaron","Ryan","Monica","Bobby","Hillary"]
>>> print(my_first_list)
['Aaron', 'Ryan', 'Monica', 'Bobby', 'Hillary']
>>> my_second_list = [8,-15,28,0]
>>> print(my_second_list)
[8, -15, 28, 0]
```

- Lists can have a mix of data types.

```
>>> my_third_list = ["abc", "xyz", 8, "8", 8.5]
>>> print(my_third_list)
['abc', 'xyz', 8, '8', 8.5]
```

- A list can contain only characters as well.

```
>>> chars = ["a","b","c","d","e"]
>>> print(chars)
['a', 'b', 'c', 'd', 'e']
>>> chars == "abcde"
False
```

`len()` and the Empty List

- As with strings, `len()` can be used to get the length of a list.

```
>>> print(my_first_list)
['Aaron', 'Ryan', 'Monica', 'Bobby', 'Hillary']
>>> print(len(my_first_list))
5
>>> print(len([]))  # the empty list
0
```

Indexing

- Indexing with a list follows the same rules as indexing with a string.

```
>>> stuff = [5, 5.8, "abc"]
>>> stuff[0]
5
>>> stuff[1]
5.8
>>> stuff[2]
'abc'
>>> stuff[3]
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    stuff[3]
IndexError: list index out of range
>>> stuff[-1]
'abc'
>>> stuff[-2]
5.8
>>> stuff[-3]
5
>>> stuff[-4]
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    stuff[-4]
IndexError: list index out of range
```

Slicing

- Slicing also follows the same rules.

```
>>> stuff2 = ["def", "hello", 8, "2 + 2"]
>>> stuff2[1:3]
['hello', 8]
>>> stuff2[1:]
['hello', 8, '2 + 2']
>>> stuff2[:2]
['def', 'hello']
>>> stuff2[0:3:2]
['def', 8]
>>> stuff2[::2]
['def', 8]
>>> stuff2[::-1]
['2 + 2', 8, 'hello', 'def']
>>> stuff2[2:0:-1]
[8, 'hello']
```

Practice

- What is the output of each of the following statements?

```
>>> fruits = ["apple","banana","orange","avocado"]
>>> print(fruits[0])
...
>>> print(fruits[3])
...
>>> print(fruits[4])
...
>>> print(fruits[-1])
...
>>> print(len(fruits))
...
>>> print(fruits[1:4:2])
...
>>> print(fruits[2:0:-1])
...
```

Practice

- What is the output of each of the following statements?

```
>>> fruits = ["apple","banana","orange","avocado"]
>>> print(fruits[0])
apple
>>> print(fruits[3])
...
>>> print(fruits[4])
...
>>> print(fruits[-1])
...
>>> print(len(fruits))
...
>>> print(fruits[1:4:2])
...
>>> print(fruits[2:0:-1])
...
```

Practice

- What is the output of each of the following statements?

```
>>> fruits = ["apple","banana","orange","avocado"]
>>> print(fruits[0])
apple
>>> print(fruits[3])
avocado
>>> print(fruits[4])
...
>>> print(fruits[-1])
...
>>> print(len(fruits))
...
>>> print(fruits[1:4:2])
...
>>> print(fruits[2:0:-1])
...
```

Practice

- What is the output of each of the following statements?

```
>>> fruits = ["apple","banana","orange","avocado"]
>>> print(fruits[0])
apple
>>> print(fruits[3])
avocado
>>> print(fruits[4])
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(fruits[4])
IndexError: list index out of range
>>> print(fruits[-1])
...
>>> print(len(fruits))
...
>>> print(fruits[1:4:2])
...
>>> print(fruits[2:0:-1])
...
```

Practice

- What is the output of each of the following statements?

```
>>> fruits = ["apple","banana","orange","avocado"]
>>> print(fruits[0])
apple
>>> print(fruits[3])
avocado
>>> print(fruits[4])
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(fruits[4])
IndexError: list index out of range
>>> print(fruits[-1])
avocado
>>> print(len(fruits))
...
>>> print(fruits[1:4:2])
...
>>> print(fruits[2:0:-1])
...
```

Practice

- What is the output of each of the following statements?

```
>>> fruits = ["apple","banana","orange","avocado"]
>>> print(fruits[0])
apple
>>> print(fruits[3])
avocado
>>> print(fruits[4])
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(fruits[4])
IndexError: list index out of range
>>> print(fruits[-1])
avocado
>>> print(len(fruits))
4
>>> print(fruits[1:4:2])
...
>>> print(fruits[2:0:-1])
...
```

Practice

- What is the output of each of the following statements?

```
>>> fruits = ["apple","banana","orange","avocado"]
>>> print(fruits[0])
apple
>>> print(fruits[3])
avocado
>>> print(fruits[4])
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(fruits[4])
IndexError: list index out of range
>>> print(fruits[-1])
avocado
>>> print(len(fruits))
4
>>> print(fruits[1:4:2])
['banana', 'avocado']
>>> print(fruits[2:0:-1])
...
```

Practice

- What is the output of each of the following statements?

```
>>> fruits = ["apple","banana","orange","avocado"]
>>> print(fruits[0])
apple
>>> print(fruits[3])
avocado
>>> print(fruits[4])
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(fruits[4])
IndexError: list index out of range
>>> print(fruits[-1])
avocado
>>> print(len(fruits))
4
>>> print(fruits[1:4:2])
['banana', 'avocado']
>>> print(fruits[2:0:-1])
['orange', 'banana']
```

Lists are Mutable

- Unlike strings, each element of a list *can* be changed.

```
>>> stuff2
['def', 'hello', 8, '2 + 2']
>>> stuff2[2] = "AAA"
>>> stuff2
['def', 'hello', 'AAA', '2 + 2']
```

List Operations

```
>>> a = [5,8] + [2,5,7]
>>> a
[5, 8, 2, 5, 7]
>>> a = a * 2
>>> a
[5, 8, 2, 5, 7, 5, 8, 2, 5, 7]
>>> [3,-2] * 3
[3, -2, 3, -2, 3, -2]
>>> [0] * 4
[0, 0, 0, 0]
```

The `in` Operator

- Like strings, lists also support the `in` operator.

```
>>> staff = ["aaron","matt","nikhil","sanjat","jiarui"]
>>> "matt" in staff
True
>>> "mat" in staff
False
```

- Again, keep in mind that -- in the right scenarios -- proper use of the `in` operator can save you from using a loop.

The `del` Operator

- You can use `del` to delete an item from a list.

```
>>> staff = ["aaron","matt","nikhil","sanjat","jiarui"]
>>> del staff[0]
>>> print(staff)
['matt', 'nikhil', 'sanjat', 'jiarui']
>>> del staff[3]
>>> print(staff)
['matt', 'nikhil', 'sanjat']
>>> del staff[3]
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    del staff[3]
IndexError: list assignment index out of range
```

- Since strings are immutable, they do not support the `del` operation.

```
>>> chars = "abcde"
>>> del chars[2]
...
TypeError: 'str' object doesn't support item deletion
```

Traversing a List

- Traversing a list also works the same as with a string. That is, you can use a `while` loop or a `for` loop, and you can use `range()` if you want to.

```
>>> stuff2 = ['def', 'hello', 'AAA', '2 + 2']
>>> i = 0
>>> while i < len(stuff2):
        print(stuff2[i])
        i += 1
def
hello
AAA
2 + 2
>>> for elem in stuff2:
        print(elem)
def
hello
AAA
2 + 2
>>> for i in range(len(stuff2)):
        print(stuff2[i])
def
hello
AAA
2 + 2
```

Another Example

```
>>> names = ["Aaron", "Richard", "Bobby"]
>>> ages = [22, 40, 18]
>>> names[1]
'Richard'
>>> ages[1]
40
>>> names[2]
'Bobby'
>>> ages[2]
18
```

Example: Names and Ages

Prompt

- Write a function called `print_names_ages` that takes two lists -- one list corresponding to names and another corresponding to ages -- and prints each name with its corresponding age. For example, if `names` is `["Aaron","Richard"]` and `ages` is `[22, 40]`[1], then the output should be:

```
Aaron is 22.
Richard is 40.
```

Solution

```
# Assumes @names and @ages are same size.
def print_names_ages(names, ages):
    for i in range(len(names)):
        print("{} is {}.".format(names[i], ages[i]))
```

Traversing a List

- As we saw with strings, `for` loops of the style `for elem in stuff2` don't provide you the indices while iterating, unlike a `while` loop or a `range()`-based `for` loop. Moreover, with lists -- which are *mutable* - using the latter two options also allows you to modify an individual element of a list.
- As with strings, the below will fail to change the list.

```
items = [8,5.3,"abc"]
for x in items:
    if x == 5.3:
        x = "AAA"
print(items)
```

change-list-bad.py

```
[8, 5.3, 'abc']
```

- This, however, will succeed in changing the list, because lists are mutable.

```
items = [8,5.3,"abc"]
for i in range(len(items)):
    if items[i] == 5.3:
        items[i] = "AAA"
print(items)
```

change-list.py

```
[8, 'AAA', 'abc']
```

Practice

- What do each of the following function calls return?

```
def find(items, target):
    for i in range(len(items)):
        if items[i] == target:
            return i
    return -1
```

practice1.py

```
>>> find(["a","b","c","d"],"ab")
...
>>> find("abcd","ab")
...
>>> find("a","b","c","d","ab")
...
>>> find(["ab","bc","cd"],"ab")
...
```

Practice

- What do each of the following function calls return?

```
def find(items, target):
    for i in range(len(items)):
        if items[i] == target:
            return i
    return -1
```

practice1.py

```
>>> find(["a","b","c","d"],"ab")
-1
>>> find("abcd","ab")
...
>>> find("a","b","c","d","ab")
...
>>> find(["ab","bc","cd"],"ab")
...
```

Practice

- What do each of the following function calls return?

```
def find(items, target):
    for i in range(len(items)):
        if items[i] == target:
            return i
    return -1
```

practice1.py

```
>>> find(["a","b","c","d"],"ab")
-1
>>> find("abcd","ab")
-1
>>> find("a","b","c","d","ab")
...
>>> find(["ab","bc","cd"],"ab")
...
```

Practice

- What do each of the following function calls return?

```
def find(items, target):
    for i in range(len(items)):
        if items[i] == target:
            return i
    return -1
```

practice1.py

```
>>> find(["a","b","c","d"],"ab")
-1
>>> find("abcd","ab")
-1
>>> find("a","b","c","d","ab")
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    find("a","b","c","d","ab")
TypeError: find() takes 2 positional arguments but 5 were given
>>> find(["ab","bc","cd"],"ab")
...
```

Practice

- What do each of the following function calls return?

```
def find(items, target):
    for i in range(len(items)):
        if items[i] == target:
            return i
    return -1
```

practice1.py

```
>>> find(["a","b","c","d"],"ab")
-1
>>> find("abcd","ab")
-1
>>> find("a","b","c","d","ab")
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    find("a","b","c","d","ab")
TypeError: find() takes 2 positional arguments but 5 were given
>>> find(["ab","bc","cd"],"ab")
0
```

Example: Prompting for List Elements

Prompt

- Write a function called `build_list` that keeps prompting the user for input until the user enters "end". The function should then return a list of all of the inputs that the user entered, excluding the "end".
  - For example, if the user enters "squidward", "spongebob", and "end", then the function should return the list `["squidward","spongebob"]`.
- Place the function in a file called `exercises.py`.

Solution

```
def build_list():
    s = ""
    l = []
    while True:
        s = input("Enter: ")
        if s != "end":
            # l += [s]
            # l.append(s)
            l.extend([s])
        else:
            break
    return l
```

Example: Summation of Items

Prompt

- Write a function called `sum_elems` that takes a list of numbers and returns the sum of all of the numbers in that list.
  - For example, if the user passes the list `[8,5,2]`, then the function should return 15.
- Add the function to your `exercises.py` file.
- Do not use the built-in `sum` function, shown below:

```
>>> sum([2,3,8])
13
```

Solution

```
def sum_elems(vals):
    s = 0
    i = 0
    while i < len(vals):
        s += vals[i]
        i += 1
    return s
```

Example: Target Character

Prompt

- Write a function called `has_character` that takes two arguments -- a list of strings and a target character -- and returns `True` if *any* string within the given list contains the target character and `False` otherwise.
  - Examples:
    - `has_character(["abc","def","ghi"],"e")` should return `True`.
    - `has_character(["abc","xyz"],"w")` should return `False`.
- Add the function to your `exercises.py` file.

Solution

```
def has_character(strings, target):
    for string in strings:
        # Check if @string contains @target.
        if target in string:
            return True
    return False
```

Another String Method: `split()`

```
>>> chars = "abcde"
>>> chars.split('b')
['a', 'cde']
>>> chars
'abcde'
>>> chars.split('d')
['abc', 'e']
>>> alternating = "axbxc"
>>> alternating.split('x')
['a', 'b', 'c']
```

- Here are more examples of `split()` from during the lecture:

```
>>> string = "adjklwejdq"
>>> string.split("d")
['a', 'jklwej', 'q']
>>> string = "abcdeedbcba"
>>> string.split("b")
['a', 'cdeed', 'c', 'a']
>>> string.split("e")
['abcd', '', 'dbcba']
```

List Methods

```
>>> coins = ["penny","nickel","dime"]
>>> coins.append("quarter")
>>> coins
['penny', 'nickel', 'dime', 'quarter']
>>> coins.extend(["half dollar","dollar"])
>>> coins
['penny', 'nickel', 'dime', 'quarter', 'half dollar', 'dollar']
>>> coins.sort()
>>> coins
['dime', 'dollar', 'half dollar', 'nickel', 'penny', 'quarter']
```

- You are allowed to use -- and must know how to use -- the `append()` and `extend()` methods on both homework assignments and exams. However, all other list methods are prohibited.

Example: Rewrite `build_list()`

- Rewrite `build_list()` to use approved methods (i.e. `append()` or `extend()`). Recall that this function should keep prompting the user for input until the user enters "end". The function should then return a list of all of the inputs that the user entered, excluding the "end".
- When we originally wrote `build_list()` earlier, we already did these versions that use these methods.

Strings as Function Arguments

- If a function takes a string as an argument, the function cannot change the original string (which is immutable).

```
def foo(string):
    string = "xyz"

chars = "abcde"
foo(chars)
print(chars)
```

string-untouched.py

Output:

```
abcde
```

Lists as Function Arguments

- However, if a function takes a list as an argument, you *can* change the original list (which is mutable), *if* you use indexing to modify the list (i.e. to "mutate" it).

```
def foo(items):
    items[2] = "xyz"

names = ["Aaron","Ryan","Bob","Jake"]
foo(names)
print(names)
```

list-arg.py

```
['Aaron', 'Ryan', 'xyz', 'Jake']
```

- If you *reassign* `items`, however, then the original list will be unharmed.

```
def foo(items):
    items = ["hello","world"]

names = ["Aaron","Ryan","Bob","Jake"]
foo(names)
print(names)
```

list-arg-bad.py

```
['Aaron', 'Ryan', 'Bob', 'Jake']
```

Practice

- What is the output of the following function?

```
def foo(items):
    i = 0
    while i < len(items):
        if i == 3:
            items[i] = "UPDATED"
        i += 1

names = ["Matt","Rick","Jessica","Peter","Tommy"]
foo(names)
print(names)
```

practice2.py

Practice

- What is the output of the following function?

```
def foo(items):
    i = 0
    while i < len(items):
        if i == 3:
            items[i] = "UPDATED"
        i += 1

names = ["Matt","Rick","Jessica","Peter","Tommy"]
foo(names)
print(names)
```

practice2.py

- Output:

```
['Matt', 'Rick', 'Jessica', 'UPDATED', 'Tommy']
```

Practice

- What is the output of the following function?

```
def foo(items):
    i = 0
    while i < len(items):
        if i == 3:
            del items[i]
        i += 1

names = ["Matt","Rick","Jessica","Peter","Tommy"]
foo(names)
print(names)
```

practice3.py

Practice

- What is the output of the following function?

```
def foo(items):
    i = 0
    while i < len(items):
        if i == 3:
            del items[i]
        i += 1

names = ["Matt","Rick","Jessica","Peter","Tommy"]
foo(names)
print(names)
```

practice3.py

- Output:

```
['Matt', 'Rick', 'Jessica', 'Tommy']
```

More Notes on Lists as Function Arguments

- There is a way to change the *entire* original list when it is used as a function argument.

```
def modify_bad(items):
    items = [500,200]

def modify_good(items):
    items[:] = [500,200]
```

change-entire-list.py

```
>>> fruits = ["banana","apple","avocado","orange"]
>>> modify_bad(fruits)
>>> fruits
['banana', 'apple', 'avocado', 'orange']
>>> modify_good(fruits)
>>> fruits
[500, 200]
```

Appendix: Global Mutability of Lists

- Because lists are mutable, we can change them when they are passed as function arguments, as we saw earlier.
- Additionally, in a function, we can change lists that are global variables, even if they are not passed to the function.

```
>>> def foo():
        fruits[:] = [500,200]

>>> fruits = [0]  # list only containing 0
>>> foo()
>>> fruits
[500, 200]
```

Appendex: Lists and Relational Operators

- When comparing two lists with relational operators, each pair of items is compared one at a time. That is, the first elements of both lists are compared, and if those are equal, then the second elements of both lists are compared, and so on.

```
>>> [8,16,5] == [8,16,5]
True
>>> [8,16,5] == [8,16,"a"]
False
>>> [8,16,5] > [7,5]
True
>>> [8,16,5] > [8,30]
False
>>> [15,-20,3] < [35,40]
True
```

Appendix: Lists and Relational Operators

- What is the output of each of the following statements?

```
>>> [200,100,500,100] == [200,100,300,100]
...
>>> [200,100,-1,-5] > [200,100,-5,-1]
...
>>> [18,-2] <= [18,-3]
...
>>> [15,"a"] > [15,"A"]
...
>>> [15,"abc"] < [15,"aaa"]
...
```

Appendix: Lists and Relational Operators

- What is the output of each of the following statements?

```
>>> [200,100,500,100] == [200,100,300,100]
False
>>> [200,100,-1,-5] > [200,100,-5,-1]
...
>>> [18,-2] <= [18,-3]
...
>>> [15,"a"] > [15,"A"]
...
>>> [15,"abc"] < [15,"aaa"]
...
```

Appendix: Lists and Relational Operators

- What is the output of each of the following statements?

```
>>> [200,100,500,100] == [200,100,300,100]
False
>>> [200,100,-1,-5] > [200,100,-5,-1]
True
>>> [18,-2] <= [18,-3]
...
>>> [15,"a"] > [15,"A"]
...
>>> [15,"abc"] < [15,"aaa"]
...
```

Appendix: Lists and Relational Operators

- What is the output of each of the following statements?

```
>>> [200,100,500,100] == [200,100,300,100]
False
>>> [200,100,-1,-5] > [200,100,-5,-1]
True
>>> [18,-2] <= [18,-3]
False
>>> [15,"a"] > [15,"A"]
...
>>> [15,"abc"] < [15,"aaa"]
...
```

Appendix: Lists and Relational Operators

- What is the output of each of the following statements?

```
>>> [200,100,500,100] == [200,100,300,100]
False
>>> [200,100,-1,-5] > [200,100,-5,-1]
True
>>> [18,-2] <= [18,-3]
False
>>> [15,"a"] > [15,"A"]
True
>>> [15,"abc"] < [15,"aaa"]
...
```

Appendix: Lists and Relational Operators

- What is the output of each of the following statements?

```
>>> [200,100,500,100] == [200,100,300,100]
False
>>> [200,100,-1,-5] > [200,100,-5,-1]
True
>>> [18,-2] <= [18,-3]
False
>>> [15,"a"] > [15,"A"]
True
>>> [15,"abc"] < [15,"aaa"]
False
```