CS 61A SU26

Lecture 07: Sequences and Containers

July 1, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

Potpourri

5 min

- Announcements
- Computing in the News

Announcements

- **Quiz 1 grades by Fri** since staff needs to finish looking into some things
  - Come to OH to ask about your quiz only ***after*** grades are out
- **Quiz 2 logistics Ed post released**
- Reminder: Only email cs61a@berkeley.edu for logistics or DSP questions
  - For extensions fill out Flextensions form
  - For questions about quiz content, we will only respond in OH
  - For questions about quiz logistics, ask in Ed or email us if it's an extenuating circumstance
- **No post-lecture questions today** because I have go to somewhere directly after
- **Askademia** is back online
- **Student Learning Center (SLC) Strategic Learning Program Ed post**

Computing in the News

- Late last night: Anthropic announced the US govt is lifting the export control ban on their latest Claude Mythos 5 and Fable 5 models, with access being restored today
- Today: EECS chair Prof. Jelani Nelson takes leave from UC Berkeley to join Anthropic

Sequences and Containers

30 min

- Definitions
- List Basics
- For Loops and Ranges
- Membership (`in` operator)
- Slicing
- Nested Lists
- Identity vs. Equality (`is` operator)
- List Comprehension
- List Methods
- Truthy/Falsy-ness
- Misc. Functions with Lists

Definitions

A **container** is a collection of values.

A **sequence** is an ordered container.

Examples of sequences in Python: `str`, `list`, `range`

A lot of the syntax we'll cover today for lists also applies to strings!

List Basics

```
>>> lst = [1, 'abc', True] # can mix and match data types
>>> len(lst)
3
>>> lst[0] # zero-indexed
1
>>> from operator import getitem
>>> getitem(lst, 2) # equivalent to square bracket notation
True
>>> lst[1] = 'def' # mutation: reassignment
>>> lst
[1, 'def', True]
```

For Loops and Ranges (1 of 4)

For loops:

```
for <var> in <iterable>:
    <suite>
```

Ranges:

```
range(end)
range(start, end)
range(start, end, step)
```

IMPORTANT: `end` is exclusive → length of range is `end - start`

For Loops and Ranges (2 of 4)

```
>>> lst = ['a', 'b', 'c']
>>> for elem in lst:
...     print(elem)
...
a
b
c
>>> for i in range(len(lst)):
...     print(i)
...
0
1
2
```

For Loops and Ranges (3 of 4)

```
>>> lst = ['a', 'b', 'c']
>>> for i, elem in enumerate(lst): # sequence unpacking
...     print(i, elem)
...
0 a
1 b
2 c
>>> lst = [[1, 2], [3, 4], [5, 6]]
>>> for x, y in lst: # sequence unpacking (more explicit)
...     print(x, y)
...
1 2
3 4
5 6
```

For Loops and Ranges (4 of 4)

```
>>> for _ in range(3): # _ is convention for an unused variable
...         print('Go bears!')
...
Go bears!
Go bears!
Go bears!
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

What would Python display? (one per line)

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

What are the contents of the list after this code has executed? (same as before)

Membership

```
>>> lst = [1, 2, [3, 4]]
>>> 2 in lst
True
>>> 3 in lst
False
>>> [3, 4] in lst
True
>>> len(lst)
3
```

Slicing (1 of 2)

```
>>> lst = ['a', 'b', 'c', 'd', 'e']
>>> lst[1:3]  # end is exclusive
['b', 'c']
>>> lst[1:]  # default end is len(lst)
['b', 'c', 'd', 'e']
>>> lst[:2]  # default start is 0
['a', 'b']
>>> lst[::2]  # default step is 1
['a', 'c', 'e']
```

Slicing (2 of 2)

```
>>> lst[1:3] = ['go', 'bears']  # slicing assignment mutates
>>> lst
['a', 'go', 'bears', 'd', 'e']
>>> temp = lst[1:3]  # slicing creates a (shallow) copy
>>> temp[1] = 'stanford'
>>> lst
['a', 'go', 'bears', 'd', 'e']
>>> temp
['go', 'stanford']
```

Tip 1: `lst[:]` creates a (shallow) copy of the entire list

Tip 2: `lst[::-1]` creates a reversed (shallow) copy of the entire list

Nested Lists

```
>>> lst = [1, 2, [3, 4]]
>>> copy = lst[:]  # careful, this is a shallow copy!
>>> lst[1] = 'go'
>>> lst
[1, 'go', [3, 4]]
>>> copy
[1, 2, [3, 4]]
>>> lst[2][0] = 'bears'
>>> lst
[1, 'go', ['bears', 4]]
>>> copy
[1, 2, ['bears', 4]]
```

Identity vs. Equality

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a == b
True
>>> a is b
False
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

True or false: Code snippet (a) is equivalent to code snippet (b).

List Comprehension (1 of 2)

Basic Syntax: `[<map_expression> for <var> in <iterable>]`

```
>>> [x for x in range(5)]
[0, 1, 2, 3, 4]
>>> [x ** 2 for x in range(5)]
[0, 1, 4, 9, 16]
>>> [x ** 2 for x in range(5) if x % 2 == 0]
[0, 4, 16]
>>> [x ** 2 if x % 2 == 0 else x + 1 for x in range(5)]
[0, 2, 4, 4, 16]
```

List Comprehension (2 of 2)

- Fun fact: You can do comprehensions on most containers (e.g. dictionaries, tuples, etc. which we'll discuss tomorrow!)
- When to use vs. when not to use: Readability vs conciseness
  - If you really must have a complex list comprehension, break it up into multiple lines for better readability

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

What list would be produced by the following? [x - y for x in range(3) for y in range(3)]

List Methods: Adding elements

```
>>> lst = [1, 2, 3]
>>> lst.append(7)
>>> lst
[1, 2, 3, 7]
>>> lst.extend([4, 5])
>>> lst
[1, 2, 3, 7, 4, 5]
>>> lst.insert(1, 'a')
>>> lst
[1, 'a', 2, 3, 7, 4, 5]
>>> [1, 2, 3] + [4, 5]
[1, 2, 3, 4, 5]
```

List Methods: Deleting elements

```
>>> lst = [1, 'a', 2, 3, 7, 4, 5]
>>> lst.remove('b')
Error
>>> lst.remove('a')
>>> lst
[1, 2, 3, 7, 4, 5]
>>> lst.pop()
5
>>> lst
[1, 2, 3, 7, 4]
>>> lst.pop(2)
3
>>> lst
[1, 2, 7, 4]
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

List methods: What would Python display?

Truthy/Falsy-ness

```
>>> not []  # empty list is falsy
True
>>> not [1]
False
```

Misc. Functions with Lists (1 of 4)

```
>>> list('abc')  # convert things into lists
['a', 'b', 'c']
>>> sum([1, 2, 3])
6
>>> sum([1, 2, 3], 10)  # optional start
16
```

Misc. Functions with Lists (2 of 4)

```
>>> is_even = [x % 2 == 0 for x in range(5)]
>>> is_even
[True, False, True, False, True]
>>> all(is_even)
False
>>> all([])
True
>>> any(is_even)
True
>>> any([])
False
```

Misc. Functions with Lists (3 of 4)

```
>>> max([1, 2, 3])
3
>>> min([1, 2, 3])
1
>>> max(['apple', 'pear', 'banana'])  # alphabetically highest
'pear'
>>> max(['apple', 'pear', 'banana'], key=lambda s: len(s))
'banana'
```

Misc. Functions with Lists (4 of 4)

```
>>> map(lambda x: x * x, [1, 2, 3])
<map object ...>
>>> list(map(lambda x: x * x, [1, 2, 3]))
[1, 4, 9]
>>> filter(lambda x: x > 2, [1, 2, 3, 4, 5])
<filter object ...>
>>> list(filter(lambda x: x > 2, [1, 2, 3, 4, 5]))
[3, 4, 5]
>>> from functools import reduce
>>> reduce(lambda x, y: x + y, [1, 2, 3, 4, 5])
15
>>> reduce(lambda x, y: x + y, [1, 2, 3, 4, 5], 10) # optional start
25
```

5 min break

Practice

(reminder to self to use high contrast theme)

20 min

- Sum List
- Decode and Encode

Practice: Sum List

- Implement `sum_list_iter` and `sum_list_rec`
- Download starter code `07.py` from the course website under Lecture 7
- Run doctests with `python3 -m doctest 07.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?

Practice: Decode and Encode

- Implement `decode` and `encode`. Hints:
  - For `decode`: Python supports negative indexing, e.g. `lst[-2]` will get the second to last element
  - For `encode`: How can you use `%` to solve this problem?
  - The `index` method of a string returns the index of the first occurrence of the input. For example, `"abc".index("b")` returns `1`.
  - The `join` method of a string takes in an iterable and uses the string as a separator when joining the elements of the iterable. For example, `','.join(['a', 'b', 'c'])` returns `'a,b,c'`.
- Download starter code `07.py` from the course website under Lecture 7
- Run doctests with `python3 -m doctest 07.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?