

CS61A Discussion 3: **Data Abstraction and Sequences**

TA: **Jerry Chen**

Email: **jerry.c@berkeley.edu**

TA Website: **jerryjrchen.com/cs61a**

Agenda

1. Week in Review
2. Sequences & Lists
3. Slicing & list comprehension
4. Data abstraction

Week In Review

Hw 2 is due next Tuesday

- How many have **started** hw2?

How was lab 3? (Lambdas/HOFs, Recursion)

Midterm 1 is next Thursday!

- In-lab review survey: tinyurl.com/jerrytm1labreview
- Next discussion will be a special review session
- TA sanctioned review session Friday, 5-7pm in 155 Dwinelle

Free one-one-one tutoring sign-ups available! (Check Piazza)

Data

So far, we've been able to store **singular values**

`x = 10`

Great. What if we want to store a **bunch of things**?

- For example, a group of temperatures over a week

Enter sequences!

Sequences

Sequences have **length** and **element selection**

Lists are a Python data type that groups together many things

```
x = [10, 20, 30] # all numbers
```

```
y = [10, "twenty", [30]] # num, str, lst
```

Lists

Length

Can easily retrieve the length of a list:

```
>>> x = [1, 2, 3]
```

```
>>> len(x)
```

```
3
```

```
>>> y = [x, 4, 5] # Does nesting matter?
```

```
>>> len(y)
```

```
3
```

Lists

Element Selection

Get an item at an index using bracket notation

```
>>> x = [1, 2, 3]
```

```
>>> x[0]
```

```
1
```

```
>>> x[0] = 10
```

```
>>> x
```

```
[10, 2, 3]
```

Slicing

Important tool for generating sublists

Anatomy of a slice:

`lst [2 : 10 : 3]`

Step size

Ending index (exclusive)

Starting index (inclusive)

Excluding any part of the slice invokes the default value:
0 for start, `len(lst) - 1` for end, 1 for step

Lists

Slicing

```
>>> x = [1, 2, 3]
```

```
>>> x[0:2]
```

```
[1, 2]
```

```
>>> x[0:2] == x[:2]
```

```
True
```

```
>>> x[0:2:-1]
```

```
[]
```

```
>>> x[2:0:-1]
```

```
[3, 2]
```

Lists

Odds & Ends

Check membership using `in`

```
>>> x = [1, 2, 3]
```

```
>>> 1 in x
```

```
True
```

```
>>> "bananas" in x
```

```
False
```

Lists

Odds & Ends

`for` can be used to loop through lists

```
>>> x = [1, 2, 3]
```

```
>>> for elem in x: #elem can be any name
```

```
...     print(elem)
```

```
1
```

```
2
```

```
3
```

Lists

Odds & Ends

range is a useful function that returns a sequence

```
>>> x = range(0, 3) # 0, 1, 2
```

```
>>> range(0, 3, 1) == range(3) # Like slicing?
```

```
True
```

```
>>> for n in x:
```

```
...     print(n)
```

```
0
```

```
1
```

```
2
```

Lists

List Comprehension

Quick way of making lists by applying **expressions** to elements in **another sequence**

```
[<map exp> for <name> in <iter> if <filter>]  
# filter is optional  
  
>>> [x for x in range(4)]  
[0, 1, 2, 3]  
  
>>> [x * 2 for x in range(4) if x % 2 == 1]  
[2, 6]
```

Data Abstraction

Focus on **what happens**, not how it happened

- **Abstract data type (ADT)** - represents an object/thing in code. Abstract since we (as the user) don't need to know how it was built and how it works!
- **Constructor** - creates an ADT
- **Selector** - retrieve information from an ADT

Lists Questions

WWPP - Page 2, Q1

```
>>> a = [1, 5, 4, [2, 3], 3]
```

```
>>> print(a[0], a[-1])
```

```
1 3
```

```
>>> len(a)
```

```
5
```

```
>>> 2 in a
```

```
False
```

```
>>> 4 in a
```

```
True
```

```
>>> a[3][0]
```

```
2
```

Lists Questions

WWPP - Page 3, Q1

```
>>> a = [3, 1, 4, 2, 5, 3]
```

```
>>> a[1::2]
```

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

```
>>> a[:]
```

```
[3, 1, 4, 2, 5, 3]
```

```
>>> a[4:2]
```

```
[]
```

```
>>> a[1:-2]
```

```
[1, 4, 2]
```

```
>>> a[::-1]
```

```
[3, 5, 2, 4, 1, 3]
```