Lists

Chris Piech and Mehran Sahami
CS106A, Stanford University
Housekeeping

- Chris and Mehran holding Ask Me Anything (AMA) sessions
  - On class Ed Forum (or maybe Zoom)
  - Chris AMA: May 9 at 9am-10am (PDT)
  - Mehran AMA: May 14 at 4pm-5pm (PDT)

- Assignment #3, Pyramid
  - Blue line at bottom of canvas doesn't show up on Macs
  - You don't need to worry about that
The Python Console

• Can run Python interactively using the "console"
  – In PyCharm click "Python Console" tab at bottom of window
  – In Terminal, run Python (e.g., typing "py" or "python3" or "python", depending on your platform) to get console

• Console has prompt: >>>
  – Can type and execute Python statements (and see results)
  – Example:
    >>> x = 5
    >>> x
    5
  – Easy way to try things out to answer questions you may have
  – Use `exit()` to leave console
Let’s Take the Console Out For a Spin...
And Then There Were None

• The term **None** is used in Python to describe "no value"
  – For example, it is the value you would get from a function that doesn't return anything
  – WHAT?!
  – Example:
    ```python
    >>> x = print("hi")
    >>> print(x)
    None
    ```
  – Comparing anything to **None** (except **None**) is False

• Why does **None** exist?
  – Denotes when the suitcase for a variable has "nothing" in it
Learning Goals

1. Learning about lists in Python
2. Writing code to use lists
3. Understand how lists work as parameters
Lists
What is a List?

• A list is way to keep track of an ordered collection of items
  – Items in the list are called "elements"
  – Ordered: can refer to elements by their position
  – Collection: list can contain multiple items

• The list dynamically adjusts its size as elements are added or removed

• Lists have a lot of built-in functionality to make using them more straightforward
Show Me the Lists!

• Creating lists
  – Lists start/end with brackets. Elements separated by commas.

```python
my_list = [1, 2, 3]
reals = [4.7, -6.0, 0.22, 1.6]
strs = ['lots', 'of', 'strings', 'in', 'list']
mix = [4, 'hello', -3.2, True, 6]
empty_list = []
```

• List with one element is **not** the same as the element
  – Could try this out on the console:

```python
>>> list_one = [1]
>>> one = 1
>>> list_one == one
False
```
Accessing Elements of List

• Consider the following list:

   ```python
   letters = ['a', 'b', 'c', 'd', 'e']
   ```

• Can think of it like a series of variables that are indexed
  – Indexes start from 0

<table>
<thead>
<tr>
<th>letters</th>
<th>'a'</th>
<th>'b'</th>
<th>'c'</th>
<th>'d'</th>
<th>'e'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

• Access individual elements:

   ```python
   letters[0] is 'a'
   letters[4] is 'e'
   ```
Accessing Elements of List

Consider the following list:
```python
letters = ['a', 'b', 'c', 'd', 'e']
```

Can think of it like a series of variables that are indexed:
- Indexes start from 0

<table>
<thead>
<tr>
<th>letters</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>'x'</td>
<td>'b'</td>
<td>'c'</td>
<td>'d'</td>
<td>'e'</td>
<td></td>
</tr>
</tbody>
</table>

Access individual elements:
- `letters[0]` is 'a'
- `letters[4]` is 'e'

Can set individual elements like regular variable:
- `letters[0] = 'x'`
Getting Length of a List

• Consider the following list:
  
  ```python
  letters = ['a', 'b', 'c', 'd', 'e']
  ```

• Can get length of list with `len` function:
  
  ```python
  len(letters) is 5
  ```
  
  – Elements of list are indexed from 0 to length – 1

• Example:
  
  ```python
  for i in range(len(letters)):
      print(str(i) + " -> " + letters[i])
  ```

  0 -> a
  1 -> b
  2 -> c
  3 -> d
  4 -> e
• Recall our old friends:

```python
my_list = [1, 2, 3]
reals = [4.7, -6.0, 0.22, 1.6]
strs = ['lots', 'of', 'strings', 'in', 'list']
mix = [4, 'hello', -3.2, True, 6]
empty_list = []
```

• Pop quiz!

```python
len(my_list) = 3
len(reals) = 4
len(strs) = 5
len(mix) = 5
len(empty_list) = 0
```
The Strangeness of Indexing

• Can use negative index to work back from end of list
  – What?!
    
    \[
    \text{letters} = [\text{'a'}, \text{'b'}, \text{'c'}, \text{'d'}, \text{'e'}] 
    \]

• Bring me the strangeness!
  
  \[
  \text{letters}[-1] \text{ is 'e'}
  \]
  \[
  \text{letters}[-2] \text{ is 'd'}
  \]
  \[
  \text{letters}[-5] \text{ is 'a'}
  \]
  
  – For indexes, think of \(-x\) as same as \(\text{len(list)}-x\)
  
  \[
  \text{letters}[-1] \text{ is same as letters[\text{len(letters)}-1]}
  \]

• How about this?
  
  \[
  \text{letters}[6]
  \]

  \text{IndexError: list index out of range}
• Can add elements to end of list with `append`

```python
alist = [10, 20, 30]
```
Building Up Lists

- Can add elements to end of list with `.append`

```python
alist = [10, 20, 30]
alist.append(40)
```

alist \rightarrow [10, 20, 30, 40]
• Can add elements to end of list with `.append`

```python
alist = [10, 20, 30]
alist.append(40)
alist.append(50)
```

```
alist
[10, 20, 30, 40, 50]
```
Building Up Lists

• Can add elements to end of list with `.append`

```python
alist = [10, 20, 30]
alist.append(40)
alist.append(50)
new_list = []
```

new_list  ➔  empty list

alist  ➔  

```
[10, 20, 30, 40, 50]
```
Building Up Lists

• Can add elements to end of list with `.append`

```python
alist = [10, 20, 30]
alist.append(40)
alist.append(50)
new_list = []
new_list.append('a')
```

```
new_list  ➔ ['a']
[ 'a' ]
alist  ➔ [10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
```
Building Up Lists

- Can add elements to end of list with `.append`

```
alist = [10, 20, 30]
alist.append(40)
alist.append(50)
new_list = []
new_list.append('a')
new_list.append(4.3)
```

```
new_list  →  ['a', 4.3]
[ 'a', 4.3 ]
alist  →  [10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
```
Removing Elements from Lists

- Can remove elements from end of list with `.pop`
  - Removes the last element of the list and returns it

```py
alist = [10, 20, 30, 40, 50]
```
Removing Elements from Lists

• Can remove elements from end of list with `.pop`
  – Removes the last element of the list and returns it

```python
alist = [10, 20, 30, 40, 50]
x = alist.pop()
```

```
x = 50
alist = [10, 20, 30, 40]
```

Piech + Sahami, CS106A, Stanford University
Removing Elements from Lists

- Can remove elements from end of list with `.pop`
  - Removes the last element of the list and returns it

```python
alist = [10, 20, 30, 40, 50]
x = alist.pop()
x = alist.pop()
```

- Diagram showing the removal of elements from the list: `alist` initially contains `[10, 20, 30]`, after two pop operations, it becomes `[10, 20]`.

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="diagram.png" alt="Diagram of list" /></td>
<td><img src="diagram.png" alt="Diagram of list" /></td>
<td><img src="diagram.png" alt="Diagram of list" /></td>
</tr>
</tbody>
</table>
Removing Elements from Lists

- Can remove elements from end of list with `.pop`
  - Removes the last element of the list and returns it

```python
alist = [10, 20, 30, 40, 50]
x = alist.pop()
x = alist.pop()
x = alist.pop()
```

```
30
30

alist
10 20
[10, 20]
```
Removing Elements from Lists

- Can remove elements from end of list with `.pop`
  - Removes the last element of the list and returns it

```
alist = [10, 20, 30, 40, 50]
x = alist.pop()
x = alist.pop()
x = alist.pop()
x = alist.pop()
```

```
alist → [10]
   ↓
20 → 20
```

```
x → 20
```
Removing Elements from Lists

• Can remove elements from end of list with `.pop`
  – Removes the last element of the list and returns it

```python
alist = [10, 20, 30, 40, 50]
x = alist.pop()
x = alist.pop()
x = alist.pop()
x = alist.pop()
x = alist.pop()
```

alist → `empty list`

```
[ ]
```
Removing Elements from Lists

• Can remove elements from end of list with `.pop`
  – Removes the last element of the list and returns it

```python
alist = [10, 20, 30, 40, 50]
x = alist.pop()
x = alist.pop()
x = alist.pop()
x = alist.pop()
x = alist.pop()
What is we did one more?
x = alist.pop()
IndexError: pop from empty list
```

Don't do it, Mehran! There might be children watching!!
More Fun With Lists

• Can I get a couple new lists, please?
  
  ```python
  num_list = [1, 2, 3, 4]
  str_list = ['Leia', 'Luke', 'Han']
  ```

• Printing lists (here, we show using the console):
  
  ```python
  >>> print(num_list)
  [1, 2, 3, 4]
  >>> print(str_list)
  ['Leia', 'Luke', 'Han']
  ```

• Check to see if list is empty (empty list is like "False")
  
  ```python
  if num_list:
      print('num_list is not empty')
  else:
      print('num_list is empty')
  ```
Even More Fun With Lists

• Can I get a couple new lists, please?
  num_list = [1, 2, 3, 4]
  str_list = ['Leia', 'Luke', 'Han']

• Check to see if a list contains an element:
  x = 1
  if x in num_list:
    # do something

• General form of test (evaluates to a Boolean):
  element in list
  – Returns True if element is a value in list, False otherwise
  – Could use as test in a while loop too
List Function Extravaganza (part 1)!

- **Function:** `list.pop(index)`  # pop can take parameter
  - Removes (and returns) an returns element at specified index
    ```python
    >>> fun_list = ['a', 'b', 'c', 'd']
    >>> fun_list.pop(2)
    'c'
    >>> fun_list
    ['a', 'b', 'd']
    ```

- **Function:** `list.remove(elem)`
  - Removes (and returns) first occurrence of element in list
    ```python
    >>> another_list = ['a', 'b', 'b', 'c']
    >>> another_list.remove('b')
    >>> another_list
    ['a', 'b', 'c']
    ```
  - **ValueError** if you try to remove an element that isn't in list
• Function: `list.extend(other_list)`
  – Adds all element from other list to list that function is called on

  ```python
  >>> list1 = [1, 2, 3]
  >>> list2 = [4, 5]
  >>> list1.extend(list2)
  >>> list1
  [1, 2, 3, 4, 5]
  ```

• `append` is **not** the same as `extend`
  – Append adds a single element, extends merges a list onto another

  ```python
  >>> list1 = [1, 2, 3]
  >>> list2 = [4, 5]
  >>> list1.append(list2)
  >>> list1
  [1, 2, 3, [4, 5]]
  ```
List Function Extravaganza (part 3)!

• Using `+` operator on lists works like `extend`, but creates a new list. Original lists are unchanged.

```python
>>> list1 = [1, 2, 3]
>>> list2 = [4, 5]
>>> list3 = list1 + list2
>>> list3
[1, 2, 3, 4, 5]
```

• Can use `+=` operator just like `extend`

```python
>>> list1 = [1, 2, 3]
>>> list2 = [4, 5]
>>> list1 += list2
>>> list1
[1, 2, 3, 4, 5]
```
List Function Extravaganza (part 4)!

• Function: `list.index(elem)`
  – Returns index of first element in list that matches parameter elem

```python
>>> alist = ['a', 'b', 'b', 'c']
>>> i = alist.index('b')
>>> i
1
```
  – ValueError if you ask for index of an element that isn't in list

• Function: `list.insert(index, elem)`
  – Inserts `elem` at the given index. Shifts all other elements down.

```python
>>> jedi = ['luke', 'obiwan']
>>> jedi.insert(1, 'mehran')
>>> jedi
['luke', 'mehran', 'obiwan']
```
  – Don't give up on your dreams...
List Function Extravaganza (part 5)!

- Function: `list.copy()`
  - Returns a copy of the list

```python
>>> actual_jedi = ['luke', 'obiwan']
>>> fantasy = actual_jedi.copy()
>>> fantasy
['luke', 'obiwan']
>>> fantasy.insert(1, 'mehran')
>>> fantasy
['luke', 'mehran', 'obiwan']
>>> actual_jedi
['luke', 'obiwan']
```
List Function Extravaganza (part 6)!

```
reals = [3.6, 2.9, 8.0, -3.2, 0.5]

• Function: max(list)
  – Returns maximal value in the list
  >>> max(reals)
  8.0

• Function: min(list)
  – Returns minimal value in the list
  >>> min(reals)
  -3.2

• Function: sum(list)
  – Returns sum of the values in the list
  >>> sum(reals)
  11.8
```
Looping Through List Elements

```
str_list = ['Leia', 'Luke', 'Han']

• For loop using `range`:
  ```python
  for i in range(len(str_list)):
    elem = str_list[i]
    print(elem)
  ```
  ```
  Leia
  Luke
  Han
  ```

• For-each loop:
  ```python
  for elem in str_list:
    print(elem)
  ```

• These loops both iterate over all elements of the list
  – Variable `elem` is set to each value in list (in order)
  – Similar to when you iterated through pixels in images
Looping Through List Elements

- General form of for-each loop:
  ```python
  for element in collection:
    # do something with element
  ```

- `element` can be any variable you want to use to refer to items in the collection
  - On each iteration through the loop, `element` will be set to be the next item (in order) in the collection
  - Recall, example:
    ```python
    for elem in str_list:
      print(elem)
    ```
  - Lists are collections
  - Images are also collections (of pixels)
  - We'll see other kinds of collections later in course
When Passed as Parameters

Variables that act like they are **copied**. (called "immutable")

- integer
- float
- Boolean
- string

These types are called "immutable". You get copies of values for parameters.

Variables that act like their **URL is copied**. (called "mutable")

- canvas
- pixel
- SimpleImage
- list

These types are called "mutable". You get reference (URL) for parameters. They are changed *in place* when you assign.
• When you pass a list as a parameter you are passing a **reference** to the actual list (not a copy)
  – It's like getting a URL to the list (*pass-by-reference*)
  – In function, changes to values in list **persist** after function ends

```python
def add_five(num_list):
    for i in range(len(num_list)):
        num_list[i] += 5

def main():
    values = [5, 6, 7, 8]
    add_five(values)
    print(values)

Output: [10, 11, 12, 13]
```
• But, watch out if you create a **new** list in a function
  – Creating a **new** list means you're no longer dealing with list passed in as parameter
  – It's like the URL you are using is pointing to a different page
  – At that point you are no longer changing parameter passed in

```python
def create_new_list(num_list):
    num_list.append(9)
    num_list = [1, 2, 3]

def main():
    values = [5, 6, 7, 8]
    create_new_list(values)
    print(values)
```

Output: `[5, 6, 7, 8, 9]`
Note on Loops and Lists

• For loop using `range`:
  ```python
  for i in range(len(list)):
      list[i] += 1  # Modifying list in place
  ```

• For-each loop:
  ```python
  for elem in list: # Modifying local variable
      elem += 1     # elem. If elem is primitive
                     # type, not changing list!
  ```

• Often use **for loop with range** when *modifying* elements of list (when elements are *primitive types*)

• Often use **for-each loop** when *not modifying* elements of list or when elements are *not primitive types*
Putting it all together: averagescores.py
Learning Goals

1. Learning about lists in Python
2. Writing code to use lists
3. Understand how lists work as parameters