Lists

Chris Gregg
Based on slides by Chris Piech and Mehran Sahami
CS106A, Stanford University
Housekeeping

• Chris is holding Ask Me Anything (AMA) sessions
  - On Zoom
  - July 15th from 11:30am-12:30pm (PDT) (right after class)
    • Link will be the same as lecture – I’ll just stay on lecture that day

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

  \[
  \text{my\_list} = [1, 2, 3]
  \]

  \[
  \text{reals} = [4.7, -6.0, 0.22, 1.6]
  \]

  \[
  \text{strs} = ['lots', 'of', 'strings', 'in', 'list']
  \]

  \[
  \text{mix} = [4, 'hello', -3.2, True, 6]
  \]

  \[
  \text{empty\_list} = []
  \]

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

  \[
  >>> \text{list\_one} = [1]
  \]

  \[
  >>> \text{one} = 1
  \]

  \[
  >>> \text{list\_one} == \text{one}
  \]

  False
• Consider the following list:

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

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

• Access individual elements:

    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

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

• Can set individual elements like regular variable:
  ```python
  letters[0] = 'x'
  ```
• 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(f"{i} -> {letters[i]}")
```

```
0 -> a
1 -> b
2 -> c
3 -> d
4 -> e
```
List Length: The Advanced Course

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

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

• How about this?
  ```python
  letters[6]
  ```
  ```python
  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 = [10, 20, 30, 40]
Building Up Lists

- 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]
[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'

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.append(4.3)
```

```
alist    new_list
[10, 20, 30, 40, 50] ['a', 4.3]
```
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]
```

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10, 20, 30, 40, 50]</td>
<td></td>
<td></td>
<td></td>
<td></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

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

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

```
• 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()
```

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>
```

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>40</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

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

- `x` contains the last element removed from `alist`.
- After the first call to `pop()`, `alist` becomes `[10, 20]`.
- After the second call to `pop()`, `alist` becomes `[10]`.
- After the third call to `pop()`, `alist` becomes `[]`.

```
x  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

```python
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()

x  # 10
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

```
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()
```

```python
IndexError: pop from empty list
```

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

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

• Printing lists (here, we show using the console):
  
  >>> 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")
  
  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:** `mylist.pop(index)`  # pop can take parameter
  - Removes (and returns) an element at specified index
  
  ```python
  >>> fun_list = ['a', 'b', 'c', 'd']
  >>> fun_list.pop(2)
  'c'
  >>> fun_list
  ['a', 'b', 'd']
  ```

- **Function:** `mylist.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:** `mylist.extend(other_list)`
  - Adds all element from other list to list that function is called on
    ```
    >>> 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
    ```
    >>> list1 = [1, 2, 3]
    >>> list2 = [4, 5]
    >>> list1.append(list2)
    >>> list1
    [1, 2, 3, [4, 5]]
    ```
• 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: `mylist.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, 'rey')
>>> jedi
['luke', 'rey', 'obiwan']

  – Don't give up on your dreams...
List Function Extravaganza (part 5)!

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

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

\[ \text{reals} = [3.6, 2.9, 8.0, -3.2, 0.5] \]

- **Function: max(mylist)**
  - Returns maximal value in the list
  >>> max(reals)
  8.0

- **Function: min(mylist)**
  - Returns minimal value in the list
  >>> min(reals)
  -3.2

- **Function: sum(mylist)**
  - Returns sum of the values in the list
  >>> sum(reals)
  11.8
str_list = ['Leia', 'Luke', 'Han']

- For loop using `range`:

```python
for i in range(len(str_list)):
    elem = str_list[i]
    print(elem)
```

- 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

<table>
<thead>
<tr>
<th>Variables that act like they are <strong>copied</strong>. (called &quot;immutable&quot;)</th>
<th>Variables that act like their <strong>URL is copied</strong>. (called &quot;mutable&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>canvas</td>
</tr>
<tr>
<td>float</td>
<td>pixel</td>
</tr>
<tr>
<td>Boolean</td>
<td>SimpleImage</td>
</tr>
<tr>
<td>string</td>
<td>list</td>
</tr>
</tbody>
</table>

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

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(mylist)):
      mylist[i] += 1  # Modifying list in place
  ```

- For-each loop:
  ```python
  for elem in mylist:
      # 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