Lists
CS106A, Stanford University
Housekeeping

• Assignment #2 due today
• Assignment #3 goes out today (Due on Mon., Apr. 25)
  – Can do Part 1 after today's class
  – Can do Part 2 after this coming Monday's class
Reviewing Parameters and Good Programming Style
# Constant – visible to all functions
NUM_DAYS_IN_WEEK = 7

# Global variable – visible to all functions
balance = 0

def main():
    balance = int(input("Initial balance: "))
    while True:
        amount = int(input("Deposit (0 to quit): "))
        if amount == 0:
            break
        deposit(amount)

def deposit(amount):
    balance += amount

• Also, really BAD style
  – So bad, that Python won't even let you do it unless you basically add a command that says "I want to have bad style"
  – I'm not going to show you that command in Python
  – But, if you know it already, DON'T use it!
  – We're in polite company
Using Parameters: Good Style

Don't want using your toaster to impact your refrigerator!

```python
def main():
    balance = int(input("Initial balance: "))
    while True:
        amount = int(input("Deposit (0 to quit): "))
        if amount == 0:
            break
        balance = deposit(balance, amount)

def deposit(balance, amount):
    balance += amount
    return balance
```

Encapsulation Principle:
Data used by a function should be a parameter or encapsulated in function
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
  – Console prompt looks like doctest indicator
  – Use exit() to leave console
Let’s Take the Console Out For a Spin...
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>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>'x'</td>
<td>'b'</td>
<td>'c'</td>
<td>'d'</td>
<td>'e'</td>
</tr>
<tr>
<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'
```

• Can set individual elements like regular variable:

```python
letters[0] = 'x'
```
Getting Length of a List

• Consider the following list:

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

• Can get length of list with `len` function:

  `len(letters)` is 5

  – Elements of list are indexed from 0 to length – 1

• Example:

  ```python
  for i in range(len(letters)):
      print(i, "->", letters[i])
  ```

  ┌───┬───┬───┬───┬───┐
  │ 0 │ 1 │ 2 │ 3 │ 4 │
  ├───┼───┼───┼───┼───┤
  │ a │ b │ c │ d │ 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?!

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

• Bring me the strangeness!

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

    letters[-1] is same as letters[\text{len}(\text{letters}) - 1]

• How about this?

    letters[6]

    IndexError: list index out of range
Building Up Lists

• 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

![Diagram of list][10, 20, 30, 40]
• Can add elements to end of list with `.append`

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

```plaintext
[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 = []
```

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

```
new_list  → empty list
[[]]
alist  → [10, 20, 30, 40, 50]
```
Building Up Lists

• Can add elements to end of list with \texttt{.append}

\begin{verbatim}
alist = [10, 20, 30]
alist.append(40)
alist.append(50)
new_list = []
new_list.append('a')
\end{verbatim}

\begin{center}
\begin{tabular}{c}
\hline
alist & new_list \\
10 & 'a' \\
20 & ['a'] \\
30 & [10, 20, 30, 40, 50] \\
40 & \\
50 & \\
\hline
\end{tabular}
\end{center}
• 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)
```

```
new_list  ['a', 4.3]

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]
```
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()  
```

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

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

```
[10, 20, 30]
```
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  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()
```

![Diagram showing the removal of elements from a list](image)
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()  # What is we did one more?
x = alist.pop()
```
More Fun With Lists

• Can I get a couple new lists, please?
  
  ```python
  num_list = [1, 2, 3, 4]
  str_list = ['Ruth', 'John', 'Sonia']
  ```

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

• 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')
  ```
• Can I get a couple new lists, please?
  ```python
  num_list = [1, 2, 3, 4]
  str_list = ['Ruth', 'John', 'Sonia']
  ```

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

• General form of test (evaluates to a Boolean):
  ```python
  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 element at specified index
  >>> 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
  >>> 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
List Function Extravaganza (part 2)!

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

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

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

```
>>> 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', 'rey', 'obiwan']
    >>> jedi.insert(1, 'mehran')
    >>> jedi
    ['luke', 'mehran', 'rey', 'obiwan']
    - Don't give up on your dreams...
• Function: `list.copy()`
  
  Returns a copy of the list

```python
>>> actual_jedi = ['luke', 'rey', 'obiwan']
>>> fantasy = actual_jedi.copy()
>>> fantasy
['luke', 'rey', 'obiwan']
>>> fantasy.insert(1, 'mehran')
>>> fantasy
['luke', 'mehran', 'rey', 'obiwan']
>>> actual_jedi
['luke', 'rey', '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
Str_list = ['Ruth', 'John', 'Sonia']

• For loop using range:

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

• We can use a new kind of loop called a "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)
For-Each Loop Over Lists

str_list = ['Ruth', 'John', 'Sonia']

for elem in str_list:
    # Body of loop
    # Do something with elem

    • Like variable i in for loop using range(),
      elem is a variable that gets updated with each
      loop iteration.

    • elem gets assigned to each element in the list
      in turn.

This code gets repeated once for each element in list.
Looping Through List Elements

• General form of for-each loop:
  
  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:
    
    for elem in str_list:
      print(elem)
  – Lists are collections
  – We'll see other kinds of collections later in course
We’ll continue with lists next class!