CS 17700
Loops and Lists

Week 5
Announcements

- Register the iClicker on Blackboard
- Project 2 will be due 11:59pm Oct 1st
- Midterm will be on Wed Oct 3rd 8:00pm - 9:30pm in EE129
- Textbook part2 is available at Amazon
ANY QUESTIONS?
Table of Contents

- range() function
- While Loops
- For Loops
- While vs. For Loops
- Lists
- Lists Operations
Ranges

- Python allows us to specify a range of values

\[ \text{range}(n) \quad 0, 1, \ldots, n-1 \]

Example:

\[ \text{list(range}(10)) \]

\[ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] \]
Ranges Continued

- But what if we don’t want to start from 0?

- `range(n)` is short hand for `range(0, n)`

Example:

```python
list(range(-4, 4))
[-4, -3, -2, -1, 0, 1, 2, 3]
```
Ranges Continued

- But what if I don’t want to count up by 1
- Python allows us to “step” by a given integer
  - range(start, end, step)

Example:

```python
list(range(0, 10, 2))
[0, 2, 4, 6, 8]
```
Ranges ... again

- But what if I want to count down?
  - Python allows us to “step” by a given integer
  - `range(start, end, step)`

Let's try:

```
list(range(0,10, -1))
[]
```

Example that works:

```
list(range(10, 0, -1))
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```
Python **while** statement

- Command name: **while**

- Boolean condition
  - determines the termination of loop

- A colon (**:`**)

- And a *block* (the indented lines of code)
Interactive loop

- Using while loop, we can write interactive loops

```python
count = 0
str = "Yes"
while str == "Yes":
    print("The count is:",count)
    count = count + 1
    str = input("continue? Yes or No:")
print("while loop ended")
```
Post-test Loops

- When the condition test comes after the body of the while loop it’s called a *post-test loop*.
  - Use `break` to exit a loop
  - Application: input error checking

```python
while True:
    x = eval(input("Enter a positive number: "))
    if x >= 0:
        x = math.sqrt(x)
        print("The square root of this number is: ", x)
    else:
        print("The number you entered was not positive!")
        break  # Exit loop if number is invalid
```
Python `for` statement

- Command name: `for`

- `i` in `range(n)`

- A colon (":")

- And a `block` (the indented lines of code)
For Loops

• One needs to know the number of iterations in advance

```python
n = eval(input("How many numbers?"))
sum = 0.0
for i in range(n):
    x = eval(input("Enter a number >> "))
    sum = sum + x
print("The average is", sum / n)
```
Nested Loops

- We can also nest the while and for loop. Pay attention to the indentation!

```python
x = eval(input("Enter a number (negative to quit) >> "))
while x >= 0:
    f = 1
    for i in range(1,x+1)
        f = f * i
    print("\nThe factorial number of x is", f)
    x = eval(input("Enter a number (negative to quit) >> "))
```
“How long do I have to shower?” You have two possible replies:

- a) 10 minutes
- b) Until you are clean

a) When programming, the first answer would be portrayed as a for-loop because we know exactly how long the shower will continue:

```python
for minutes in range (0,10):
    shower
```

b) The second answer would be portrayed as a while-loop because the length of the shower is undetermined; we instead focus on the condition of cleanliness:

```python
while you are not clean:
    shower
```
For vs While Loop

```python
for count in range(0,10):
    print("The count is:", count)
print("for loop ended")
```

```python
count = 0
while count < 10:
    print("The count is:", count)
    count = count + 1
print("while loop ended")
```
Lists

- **Syntax:**
  
  \[elm0, elm1, elm2, \ldots, elmn]\n
- **Indexes:** provide us a quick mechanism for accessing a *given* element that is contained within a list
  - The index starts from 0, NOT from 1
[] Notation

- \( a[i] \) : gives a name to the \( i \)th element of a list

- \( a = \text{“Sally”} \)
  - \( a[i] \) is the \( i+1 \) character of Sally
  - In the example above, \( a[2] \) is the character ‘l’

- \( a = \text{list(range(0, 10))} \)
  - \( a[i] \) is the \( i+1 \) number in the range of 0 to 9
  - In the list above, \( a[2] \) is 2
Lists: Examples

- `a = list(range(0, 10))`

- `print(a)`  
  `[0,1,2,3,4,5,6,7,8,9]`

- `print(a[3])`  
  `3`
a = 10
b = range(0,5)
c = “Sally”

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>b[0]</td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>b[1]</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>b[2]</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>b[3]</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>b[4]</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>c[0]</td>
<td>c</td>
<td>S</td>
</tr>
<tr>
<td>c[1]</td>
<td></td>
<td>a</td>
</tr>
</tbody>
</table>
Negative Indexes

- What happens if we use a negative index? Do we get an error?

```python
x = list(range(10))
print(x[-1])  # this will print 9
print(x[-10])  # this will print 0
print(x[-11])  # Error!
```

```python
>>> print(x[-11])
Traceback (most recent call last):
  File "<pyshell#173>", line 1, in <module>
    print(x[-11])
IndexError: range object index out of range
```

- **Under the hood:**
  - If you pass in a negative index, Python adds the length of the list to the index
a = 10
b = range(0,5)
c = “Sally”

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>b-5</td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>b-4</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>b-3</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>b-2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>b-1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>c-5</td>
<td>c</td>
<td>S</td>
</tr>
<tr>
<td>c-4</td>
<td></td>
<td>a</td>
</tr>
</tbody>
</table>

...
Lists: Examples

- \( a = \text{list(range(0, 10))} \)

- \( \text{print(a)} \) \[0,1,2,3,4,5,6,7,8,9\]

- \( \text{print(a[-3])} \) \[7\]
[] Notation

- The [] can be used to index not only into a list, but also a range, or a string. For example:

```python
i = 0
x = range(0,10)
while i < 10 :
    print (x[i])
i = i + 1
```

Output:
```
0
1
2
3
4
5
6
7
8
9
```

```python
i = 0
x = "This is a string."
while i < 10 :
    print (x[i])
i = i + 1
```

Output:
```
The
i
s
i
s
a
```
More Complex Lists

- $y$ is an example of a list. Each element is a string:

$$y = ['ABCD', 'BCD', 'CD', 'D']$$

- As you can see each element can be of different length.

- The list elements can also be different types:

$$y = ['ABCD', [1,2,3], 'CD', 'D']$$
Indexing into Nested Lists

- Suppose we wanted to extract the value 3

\[
y = \left[ "ABCD", [1,2,3], "CD", "D" \right]
\]
\[
y[1][2]
\]

- The first set of [] get the element at position 1 of y. The second [] selects the element at position 2 of the element y. This is equiv. to:

\[
z = y[1]
\]
\[
z[2]
\]
Typical mistakes

- Undershooting the bounds
  - `a = "hello" a[-6]`

- Overshooting the bounds
  - `a = "hello" a[5]`

- Off by one
  - `a[0] vs a[1]`
  - By convention we use 0-based indexing
    - `a="hello"`
    - `print(a[0])`
    - `print(a[1])`
Assigning to Lists

- The [] syntax not only allows us to retrieve the value of a given element, it also lets us change the content of that memory location.

  Namely, we can assign to that location:
  - `b = list(range(0,5))`
  - `b[2] = 100`
  - `print(b[2])`
  - `print(b[2])`

<table>
<thead>
<tr>
<th>a</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>b[0]</td>
<td></td>
</tr>
<tr>
<td>b[1]</td>
<td>1</td>
</tr>
<tr>
<td>b[2]</td>
<td>2</td>
</tr>
<tr>
<td>b[3]</td>
<td>3</td>
</tr>
<tr>
<td>b[4]</td>
<td>4</td>
</tr>
</tbody>
</table>
Assigning to Lists

- The [] syntax not only allows us to retrieve the value of a given element, it also lets us change the content of that memory location
- Namely, we can assign to that location
  - b=list(range(0,5))
  - b[2] = 100
  - print(b[2])
  - print(b[2])

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>b[0]</td>
<td></td>
</tr>
<tr>
<td>b[1]</td>
<td>1</td>
</tr>
<tr>
<td>b[2]</td>
<td>100</td>
</tr>
<tr>
<td>b[3]</td>
<td>3</td>
</tr>
<tr>
<td>b[4]</td>
<td>4</td>
</tr>
</tbody>
</table>
Assigning to Lists

- The [] syntax not only allows us to retrieve the value of a given element, it also lets us change the content of that memory location.
- Namely, we can assign to that location.
  - b=list(range(0,5))
  - b[2] = 100
  - print(b[2])
  - print(b[2])

<table>
<thead>
<tr>
<th>a</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b[0]</td>
<td>10</td>
</tr>
<tr>
<td>b[1]</td>
<td>1</td>
</tr>
<tr>
<td>b[2]</td>
<td>60</td>
</tr>
<tr>
<td>b[3]</td>
<td>3</td>
</tr>
<tr>
<td>b[4]</td>
<td>4</td>
</tr>
</tbody>
</table>
Operations on Lists

- `len()`: gives you the “length” or number of elements in a list.

  >>> len([0,1,2,3,4,5])
  6

- Recall the example of printing each element of a string:
Operations on Lists

```
i = 0
x = "This is a string."
while i < 10 :
    print (x[i])
i = i + 1
```

Output:
```
T
h
i
s
```

```
i = 0
x = "This is a string."
while i < len(x) :
    print (x[i])
i = i + 1
```

Output:
```
T
h
i
s
```
Operations on Lists

- **append()**: append an element at the end of a given list.

  ```python
  >>> c = [1, 2, 3, 4, 5]
  >>> c.append(6)
  ```

  Results in `c` having an additional element: `[1, 2, 3, 4, 5, 6]`

- Just like we can *concatenate* strings we can concatenate lists
  
  - **print** `([1, 2, 3] + [4, 5, 6])`
  - Will print: `[1, 2, 3, 4, 5, 6]`

- Just like we can *slice* strings we can also slice lists
  
  - `b = [1, 2, 3, 4, 5, 6]`
  - **print** `(b[2:5])`
  - Will print `[3, 4, 5]`
ANY QUESTIONS?