Announcements

- Project 2
  - Is now posted
  - Will be due on Oct. 1

- Week 7
  - Midterm on Wed. Oct. 3 in EE 129
    - Come early, EE is a complicated building
  - Review session on Oct. 2

- Text book part 2 out at Amazon
Lists Reviewed

- Syntax:
  \[
  [ \text{<elt 0>}, \text{<elt 1>}, \ldots, \text{<elt m>} ]
  \]

- Example:
  \[
  [3, 9, \text{“hi”}, 2.9, \text{False}]
  \]

- Lists can be nested:
  \[
  [[1,2], 3, [4, [5,6]], 7]
  \]

- Lists can be empty:
  \[
  [ ]
  \]
CQ

- Is this list empty?

  [[]]

A: This list is empty

B: This list is not empty
Operations

- Let \( L = [[1,2], 3, [4, [5,6]], 7] \)
- Length function: \( \text{len}(L) \) is 4
- Element selection:
  - \( L[0] \) is \([1,2]\)
  - \( L[1] \) is 3
  - \( L[2] \) is \([4, [5,6]]\)
  - \( L[3] \) is 7
  - \( L[\text{len}(L)] \) evaluates to \( L[4] \) and throws an error
- Nesting:
  - \( L[2][0] \) is 4
  - \( L[2][1][0] \) is 5
Appending etc.

- L.append(x) makes list L longer by one element, at the end, which is x:
  >>> L = [[1,2], 3, [4, [5,6]], 7]
  >>> x = 8
  >>> L.append(x)
  >>> L
  [[1, 2], 3, [4, [5, 6]], 7, 8]

- L1 + L2 concatenates two lists
  >>> L1 = [1, 2, 3]
  >>> L2 = [4, 5]
  >>> L1 + L2
  [1, 2, 3, 4, 5]
Python Boot Camp

- Introduction to Ranges
  - Sequences

- Definite Loops
  - For Loops

- Comparison of For and While loops
Our First For Loop

# File: chaos.py
# A simple program illustrating chaotic behavior

def main():
    print("This program illustrates a chaotic function")
    x = eval(input("Enter a number between 0 and 1:"))
    for i in range(10):
        x = 3.9 * x * (1 - x)
    print(x)

main()
>>> This program illustrates a chaotic function
Enter a number between 0 and 1: .5
0.975
0.0950625
0.335499922266
0.869464925259
0.442633109113
0.962165255337
0.141972779362
0.4750843862
0.972578927537
0.104009713267
>>>
Definite Loops

- In chaos.py, what did `range(10)` do?
  ```python
  >>> list(range(10))
  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  ```

- `range` is a built-in Python function that generates a sequence of numbers, starting with 0.

- `list` is a built-in Python function that turns the sequence into an explicit list (sequence)

- The body of the loop executes 10 times.
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:

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

Example that works:
```python
list(range(10, 0, -1))
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```
Definite Loops

A *definite* loop executes a definite number of times, i.e., at the time Python starts the loop it knows exactly how many *iterations* to do.

for `<var>` in `<sequence>`:
   `<body>`

The beginning and end of the body are indicated by indentation.

Sequences: ranges, lists, strings
Definite Loops

for <var> in <sequence>:  
  <body>

- The variable after the *for* is called the *loop index*. It takes on each successive value in *sequence*.

- You can reassign the index in the body, but this won’t change the loop execution:

```python
for k in range(5):
    print(k)
    k = k+3
    print("* ",k)
```
For (Definite) Loops

- These loops have a more controlled structure and cannot run infinitely.

- This type of loop is commonly used with the `range(x,y)` function.

```python
for x in range(0,10):
    print (x)

for x in list(range(0,10)):
    print (x)
```
Definite Loops

```python
>>> for i in [0,1,2,3]:
    print (i)

0
1
2
3
```
Definite Loops

```python
>>> for i in range(0,4):
    print (i)

0
1
2
3
```
Definite Loops

>>> for i in list(range(0,4)):
    print (i)

0
1
2
3
Definite Loops

```python
>>> for odd in [1, 3, 5, 7]:
    print(odd*odd)

1
9
25
49
```
CQ: Are these programs equivalent?

1
for a in range(0, 10, 1):
    print(a)

2
for a in range(10):
    print(a)

A: yes
B: no
Definite Loops

- **for** loops alter the flow of program execution, so they are referred to as *control structures*.
More Complex Examples

```python
x = 0
for a in range(10):
    x = x + a
print(x)
```

```python
x = 1
for a in range(1,10):
    x = x * a
print(x)
```
How to “visualize” a loop

- You can use print statements to figure out what is happening inside a loop
  - This works for For loops and While loops

- What should we print?
  - The loop index
    - This will give us information of how the loop index changes as the loop executes

- Any calculations we may perform
More Complex Examples

```python
x = 0
for a in range(10):
    print("a is: ", a)
    x = x + a
    print("x is: ", x)
print(x)
```

```
a is:  0
x is:  0
a is:  1
x is:  1
a is:  2
x is:  3
a is:  3
x is:  6
a is:  4
x is:  10
a is:  5
x is:  15
a is:  6
x is:  21
a is:  7
x is:  28
a is:  8
x is:  36
a is:  9
x is:  45
```
More Complex Examples

```python
x = 1
for a in range(1, 10):
    print("a is: ", a)
    x = x * a
    print("x is: ", x)
print(x)
```

```
a is:  1
x is:  1
a is:  2
x is:  2
a is:  3
x is:  6
a is:  4
x is:  24
a is:  5
x is:  120
a is:  6
x is:  720
a is:  7
x is:  5040
a is:  8
x is:  40320
a is:  9
x is:  362880
```
CQ: Are these programs equivalent?

A: Yes

B: No

1

```
x = 0
y = 0

for k in range(5):
    x = x + k
    y = x + k
print (y)
```

2

```
x = 0
y = 0

for k in range(5):
    x = x + k
    y = x + k
print (y)
```
Loops can be nested

```python
def nested(a, b):
    for x in range(0, a):
        for y in range(0, b):
            print(x*y)
```
Visualizing a Nested Loop

def nested(a, b):
    for x in range(0, a):
        print("x is: ", x)
        for y in range(0, b):
            print("y is:", y)
            print(x*y)
CQ: Do these functions have the same output?

```python
def nested1(a,b):
    for x in range(0, a):
        for y in range (0, b):
            print(x*y)
```

```python
def nested2(a,b):
    for y in range(0,b):
        for x in range (0, a):
            print(x*y)
```

A: yes
B: no
When might they be equivalent?

- What about when \( a = b \)?
  - That is, we call the functions and provide the same values for \( a \) and \( b \)
  - Output of \( \text{nested1}(2,2) \) = output of \( \text{nested2}(2,2) \)
Differences between While and For

- For loops provide a finite and enumerated “range” of values
  - This determines the “length” of the loop
- For loops *explicitly* rebind the *loop index*
  - for X in ...
- While loops express an *execution* condition
  - It executes *until* that condition no longer holds
def main():
    print("This program illustrates a chaotic function")
    x = eval(input("Enter a number between 0 and 1:"))
    for i in range(10):
        x = 3.9 * x * (1 - x)
        print(x)

main()
def main():
    print("This program illustrates a chaotic function")
    x = eval(input("Enter a number between 0 and 1:"))
    a = 0
    while a < 10:
        x = 3.9 * x * (1 - x)
        print(x)
        a = a+1
main()
CQ: Are these programs equivalent?

1

```
a = 0
while(a < 10):
    print(a)
a = a+1
```

2

```
for a in range(10):
    print(a)
```

A: yes
B: no
Homework

- Read Chapter 6 (Loops) in the text book
Announcements

- Work on Project 2
  - Due on Oct. 1, 11:59pm

- Week 7
  - Midterm on Wed. Oct. 3 in EE 129
    - Come early, EE is a complicated building
  - Review session on Oct. 2
  - No class on Oct. 4

- Text book part 2 out at Amazon
  - ToC posted on home page
Python Boot Camp

- How to specify strings
- Special Characters
- ASCII
Strings

- In addition to basic data types such as **Integers** most languages provide **Strings**

- **Strings** are sequences of characters. We have seen a few examples already.
  - “**This is a string**”
  - Quotation marks of open and closing marks are the same

- **Strings** are written using quotation marks
Sequence

- Abstraction of items following one another in linear order

- Python objects that are sequences:
  - range(…)
  - ( item, item, …, item )
  - [ item, item, …, item )
  - “text”

- Analogy: car
  - coupe, convertible, sedan, cross-over, …
Strings and how we specify them

- Strings are defined with quotations
  - "hello"

- There are four ways we can define a string
  - 'hello'
  - "hello"
  - ""hello"" or ""hello"
  - """"hello""""
Why do we have three versions?

- We can use ‘ ‘ to print strings that contain “
  - Example: print( ’ “ ‘)
  - Will print “
  - Example: print( “ ’ “)
  - Will print ‘

- We can use “” “”” to print strings that span multiple lines. That is, the string contains the newline character.
There are other special Characters

- "\n" is a newline (like pressing the Enter key)
- Example: print ("hello\nworld")
- prints hello
  world
- Also observe

```python
>>> s = """"hello
world"""
>>> print(s)
hello
world
>>> 
```
There are other special Characters

- “\t” is a tab
  - Example: print ("hello\tworld")
  - prints hello world
String Operations

- We already know how to concatenate strings
  - Example: `print ("Hello"+"World")`
  - Will print `HelloWorld`

- We also know how to get the length of a string
  - Example: `print(len("HelloWorld"))`
  - Will print `10`
String Operations

- We also know how to print the ASCII encoding of a character
  - Example: print (ord('b'))
  - Will print 98

- We can use this to print the ASCII encoding of a string:

```python
str = "hello"
for char in str:
    print (ord(char))
```
Advanced String Operations

- But first we need to introduce some new syntax

  - Object.Method(*arguments*)

- We will study Object Oriented programming more deeply later in the course
Objects and Methods

- Methods are similar to functions
  - We can think of a Method as a function or an *action* performed on an Object

- An Object can be anything
  - More concretely, any python value
    - We will revisit this definition
    - For now we will consider strings as Objects
  - Different types support different methods
Advanced String Operations

- **print** (“Hello World”.capitalize())
  - Hello world

- **print** (“hello world”.capitalize())
  - Hello world

- **print** (“Hello World”.endswith(“World”))
  - True

- **print** (“Hello World”.endswith(“Hello”))
  - False
CQ: Are these programs equivalent?

1. capitalize()

2. “1”.capitalize()

A: yes
B: no
Advanced String Operations

- `print` ("HeLIO WoRID".lower())
  - hello world

- `print` ("hello world".lower())
  - hello world

- `print` ("Hello World".upper())
  - HELLO WORLD

- `print` ("hello world".upper())
  - HELLO WORLD
Advanced String Operations

- `print` ("Hello World".find("World"))
  - 6

- `print` ("Hello World".find("or"))
  - 7

- `print` ("Hello World".find("Hello"))
  - 0

- `print` ("Hello World".find("o"))
  - 4
Example

```python
b = "HELLO"
a = b.lower()
print(b)
print(a)
```

```bash
>>> b = "HELLO"
>>> a = b.lower()
>>> print(b)
HELLO
>>> print(a)
hello
```
Variable Storage

In week 2 we discussed memory / variables as boxes. Note that b is a string.

\[
\begin{array}{|c|c|}
\hline
a & 10 \\
\hline
b & S \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
a & a \\
\hline
l & l \\
\hline
l & y \\
\hline
\end{array}
\]

a = 10
b = “Sally”
String Storage

- We can see that strings are really individual characters stored into different locations in memory, in order.
- There is another way to store values in memory in this way.
The Range Function Revisited

Consider this example

\[
\begin{align*}
a &= 10 \\
b &= \text{range}(0, 5)
\end{align*}
\]
String Storage

- Like a string, we have a single variable assigned to the start of multiple values in memory.

<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></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>a</td>
</tr>
</tbody>
</table>

a = 10
b = range(0,5)
c = “Sally”
Plotting

- Requires libraries. There are choices. Popular choice:
  - numpy
  - matplotlib
- Lab machines will have this installed
- Web search finds them for a free download
  - but you need the right version!
  - Choices:
    - do that pert in the labs; we install the libraries
    - venture into the wild and do it yourself
- Handy link:
Basic plotting

- Single graph: xList and yList needed; **must have same length**

Example:

```python
import numpy
import pylab  # part of matplotlib

def graphIt(xList, yList):
    pylab.plot(xList, yList, 'b')
    pylab.show()

x = list(range(10))
y = list()
for k in range(10):
    y.append(k*k)
graphIt(x, y)
```
Multiple curves

Example:

```python
import numpy
import pylab # part of matplotlib

def graphIt2(xList, yList, zList):
    pylab.plot(xList,yList,'b')
    pylab.plot(xList,zList,'r')
    pylab.show()

x = list(range(10))
y = list()
z = list()
for k in range(10):
    y.append(k*k)
    z.append(100-k*k)
graphIt2(x,y,z)
```
def graphIt(yList):
    xList = list(range(len(yList)))
    pylab.plot(xList, yList, 'b')
    pylab.show()

def main():
    print("This program illustrates a chaotic function")
    y = eval(input("Enter a number between 0 and 1:")
    yL = list()
    for i in range(200):
        y = 3.9 * y * (1 - y)
        yL = yL + [y]
    graphIt(yL)
Homework

- Read chapter 7, strings; in part 2
- Work on project 2