Computer Science 1 — CSci 1100
Lecture 10 — For Loops

Overview

- Review the basic idea of `for` loops, and discuss other potential used of `for` loops
- Part 1: Ranges and their use in `for` loops
- Part 2: `for` loops that do not use all of the indices
- Part 3: Nested `for` loops and images

Reading: *Practical Programming*, Section 7.1 and the start of 7.2.

Current Knowledge of For Loops

- So far we have used for loops to access each entry in a list or input file in succession and apply one or more operations to each.
- Example 1 (from Lecture 7):

```
co2_levels = [ 320.03, 322.16, 328.07, 333.91, 341.47, 348.92,
 357.29, 363.77, 371.51, 382.47, 392.95 ]

avg = sum(co2_levels) / len(co2_levels)
for value in co2_levels:
      diff = value - avg
      print 'Difference from average is %.2f' %diff, 'ppm'
```

- Example 2 (from HW 3):

```
# Access the file containing the valid words
words_file = open('words.txt')

# Read each word, remove the white space and the \n and append it to the list
words_list = []
for w in words_file:
    w = w.strip().strip('\n')
    words_list.append(w)
```

- Example 3 (also from Lecture 1 and from HW 3):

```
def three_double(s):
    for i in range(0, len(s)-5):
        if s[i] == s[i+1] and s[i+2] == s[i+3] and s[i+4] == s[i+5]:
            return True
    return False
```

What Else Might We Want to Do?

- Output the year associated with each CO2 level. This requires knowing the position of each value the list
- Look for trends in the data: compare value for current year with value for a previous year.
- Access data from multiple lists in the same loop.
- Look at combinations of values.
- Loop over two-dimensional (image) data.

Part 1: Ranges

We already started using ranges in HW 3, but we will go over them in detail here.

- A range is a function to generate a list of integers. For example,

```
 >>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Notice this is up through and **not including** the last value specified!
- If we want to start with something other than 0, we provide the starting values

```
>>> range(3,8)
[3, 4, 5, 6, 7]
```

- We can create increments. For example,

```
>>> range(4,20,3)
[4, 7, 10, 13, 16, 19]
```

  starts at 4, increments by 3, stops when 20 is reached or surpassed.

- We can create backwards increments

```
>>> range(-1, -10, -1)
[-1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Using Ranges in For Loops

- We can use the `range` to generate the list of values in a for loop. Our first example is printing the contents of the `planets` list

```
planets = [ 'Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter',
    'Saturn', 'Uranus', 'Neptune', 'Pluto' ]
for i in range(len(planets)):
    print planets[i]
```

- The variable `i` is variously known as the "index" or the "loop index variable" or the "subscript".
- We will modify the loop in class to do the following:
  - Print the indices of the planets (starting at 1!)
  - Print the planets backward.
  - Print every other planet.

Part 1 Exercises

1. Generate a range for the positive integers less than 100. Use this to calculate the sum of these values, with and without a for loop.
2. Use a range and a for loop to print the even numbers less than a given integer `n`.
3. The years for the CO2 levels were in parts per million every 5 years starting in 1960. Generate a range based on this and on the length of `co2_levels`. Assign the resulting list to the variable `year`.
4. Suppose we want a list of the squares of the digits 0..9. The following does NOT work

   ```
   squares = range(10)
   for s in squares:
       s = s*s
   ```

   Why not? Write a different for loop that uses indexing into the `squares` list to accomplish our goal.

Part 2: More Sophisticated Uses of Indexing in for Loops

- Accessing multiple lists in one for loop
- Stopping before the end of a list
- Comparing adjacent values in a list

These types of operations are used over and over again in different programming problems.

Accessing Two Lists in One Loop: Printing Years and CO2 Levels

- We can use our generated list, `years`, from the third exercise just above, to solve the problem of printing both the year and the CO2 level.

  ```
  for i in range(len(co2_levels)):
      print "Year %d: CO2 level %.2f" %(years[i], co2_levels[i])
  ```

  Notice that we created a second range of indices and used these to index **both** `years` and `co2_levels`.

- We can solve the same problem without using the `years` range, as in

  ```
  for i in range(len(co2_levels)):
      print "Year %d: CO2 level %.2f" %(1960 + i*5, co2_levels[i])
  ```

Loops That Do Not Iterate Over All Indices

- Sometimes the loop index should not go over the entire range of indices, and we need to think about where to stop it "early", as the next two examples show.
- Example 1: We'd like to count how many years the CO2 level has risen relative to the previous five years. We'll write a simple loop to do this.
- Example 2: Returning to our example from Lecture 1 and HW 3, we will briefly re-examine our solution to the following problem: Given a string, how can we write a function that decides if it has three consecutive double letters?

```
def has_three_doubles(s):
    for i in range(0, len(s)-5):
        if s[i] == s[i+1] and s[i+2] == s[i+3] and s[i+4] == s[i+5]:
            return True
    return False
```

- In each case, we have to think carefully about where to start our looping and where to stop!

Part 2 Exercises

1. The following code for finding out if a word has two consecutive double letters is wrong. Why? When, specifically, does it fail?

```
def has_two_doubles(s):
    for i in range(0, len(s)-5):
        if s[i] == s[i+1] and s[i+2] == s[i+3]:
            return True
    return False
```

2. A local maximum, or peak, in a list is a value that is larger than the values next to it. For example,

```
L = [ 10, 3, 4, 9, 19, 12, 15, 18, 15, 11, 14 ]
```

has local maxima at indices 4 and 7. Write code to print the index and the value of each local maximum.

Part 3: Nested For Loops

- Some problems require "iterating" over either
  - two dimensions of data, or
  - all pairs of values from a list
- As an example, here is code to print all of the products of digits:

```
digits = range(10)
for i in digits:
    for j in digits:
        print "%d x %d = %d" %(i,j,i*j)
```

- How does this work?
  - for each value of i — the variable in the first, or "outer", loop,
  - Python executes the *entire* second, or "inner", loop
- We will look at finding the two closest points in a list...

Aside: Tuples

- A *tuple* is simply a list that can not be changed, e.g.

```
>>> x = (4, 5, 10)   # note the parentheses rather than [ ]
>>> print x[0]
4
>>> print x[2]
10
```

- A list can be converted to a tuple:

```
>>> L = [ 'a', 6 ]
>>> T = tuple(L)   # T is a copy of L that can not be changed.
>>> print T[0]
a
>>> L[0] = 'b'
>>> print L[0]     # The original list is changed.
b
>>> print T[0]     # The tuple created from the list has not.
a
```

Example: Finding the Two Closest Points

- Suppose we are given a list of point locations in two dimensions, where each point is a tuple. For example,

```
points = [ (1,5,), (3.5, 4), (10, 5), (-5, 2), (6,3) ]
```

- Our problem is to find the two points that are closest to each other.
- The natural idea is to compute the distance between any two points and find the minimum.
  - We can do this with and without using a list of distances.
- We will work through the approach to this and post the result on the Piazza site.

Image Examples Are Good Illustrations of Nested For Loops

- We will start with the problem of converting a color image to gray scale as an example of the use of a double for loop. This is the code `rgb2gray.py` on the course Piazza site.
- We will modify this to do several things:
  - Flip an image upside down
  - Reflect an image left-right
  - Create a cartooned version of the image

Exercise: Nested vs. Sequential Loops

1. The following simple exercise will help you understand loops better. Show the output of each of the following pairs of `for` loops. The first two pairs are nested loops, and the third pair is formed by consecutive, or "sequential", loops.

```
# Version 1
sum = 0
for i in range(10):
    for j in range(10):
        sum += 1
print sum

# Version 2
sum = 0
for i in range(10):
    for j in range(i+1,10):
        sum += 1
print sum

# Version 3
sum = 0
for i in range(10):
    sum += 1
for j in range(10):
    sum += 1
print sum
```

Summary

- `range` is used to generate a list of indices in a `for` loop
- Indices in a `for` loop can be used to access and change the contents of a list.
- The indices in a for loop can be used to access two or more different lists.
- Changing the range of indices allows us to control how we access values in a `for` loop.
- Nested for loops can be used to access two-dimensional data or compute properties of all pairs of values in a list.