Assignment Part 1 for Week 3 (January 27)

Part 2 will be posted by Friday, January 30

Final turn-in date: Friday, February 6 by 110pm in BU417

Assignment points: 50 (20 in Part 1)

"Extra effort" points: 5 (None in Part 1)

Topics list ..... `Python syntax` & notation

- Web page design ..... **URLs and HTML**
- Why computers count from zero ..... **binary numbers**
- Using Python libraries ..... `import`, **libraries**
- Finding a short string in a long one ..... `string.find()`
- Taking strings apart ..... `[:]`, **slicing**
- Putting strings together ..... `+`, **concatenation**
- Input from the web ..... `get()`
- Comments ..... #

How the topics are related

This week's concept maps ***(coming!)*** show the basic relationships between web pages and Python, how Python libraries ("invisible" pre-written code) are used, and how the way that computers count is related to the way we can take strings apart and put them together. You can find Python libraries on the web, and also how to use them, but we will just scratch the surface as we "keep it simple."

Some key points about web pages, HTML and Python:

- A web page is written in a special language called HTML (HyperText Mark-up Language) that your web browser uses to display the page.
- The HTML description of a web page has special commands called *tags* that are enclosed in angle brackets "<>" that specify exactly how the display is created on your screen.
- You can find out what the tags do by searching the web for "HTML tutorial" or "HTML code".
- HTML is *not* Python, and cannot do some important things Python *can* do, like looping. You can see what HTML looks like by choosing "View Source" or a similar command on your browser's menu.

Sample assignment

The sample assignment is to count the number of list items on an example web page. List tags, `"<li>"`, are counted since they correspond to the number of items listed on the page (although in an extra effort assignment you be asked to figure out why this is not always true).

How to do the sample assignment

This is a more complex assignment that the first one, so I broke it down into three pieces. Knowing how to divide a problem into separate parts is a skill that takes time to acquire, but can be learned. It is like knowing that you are going to cook a meal for your family, say a holiday dinner. That means you want to have a menu before you shop for the groceries. If it is for someone's birthday, you might want to have a main dish (maybe baked filet of fish), a vegetable (maybe a green bean casserole with onions on top) and a birthday cake. That is one way to subdivide "Birthday Dinner" into manageable parts.

I subdivided this assignment into three parts:

1. Find a short string in a long one, which meant I also needed to know how to take strings apart and put them together,
2. Get the HTML source code from a web page to have a string to use, and
3. Combine these two parts with a loop so I could count the shorter strings (the list tags, `"<li>"`) in the long one (the HTML source code).

1. Finding a shorter string in a long one

I began by using a Python reference of the web to get the library code to retrieve a web page, and also checked the Python library that had code to find a smaller string in a larger string (the smaller string is called a *substring*) as well as remove the smaller string once I had found it (so I wouldn't count it again). To use the code that retrieves web pages I had to learn a new Python keyword called `import`.

Just as we import, or bring food or other goods into this country from elsewhere (like fresh green grapes from Chile), a Python program can bring new keywords from other code elsewhere (a *library*) into it. Once a library is imported, these keywords will perform actions in your program—and you don't have to write Python code for them, you just have to know their names and how they work.

What usually leads people to ask questions is that some libraries are built into Python already. To find out which ones are there, you can search the web with keywords like "standard Python library". However, Python has ***many*** actions it can perform using the library routines. Finding the one you want can be frustrating and confusing. Here is the entry that defines what the `text.find()`

routine does that we will use in this assignment, where `text` is a variable we create that contains a string.

`string.`**`find`**(*s*, *sub*[, *start*[, *end*]])

Return the lowest index in *s* where the substring *sub* is found such that *sub* is wholly contained in `s[start:end]`. Return `-1` on failure. Defaults for *start* and *end* and interpretation of negative values is the same as for slices.

You do ***not*** have to understand all of this right now. We will only use part of it:

`string.`**`find`**(*sub*)

…and write it this way in an expression:

```
index = text.find("<li>")
```

…which puts a value into a variable named `index` that specifies the place in the string where the list tag—`"<li>"`— begins. Since the list tag is a string, we enclose it in quotes.

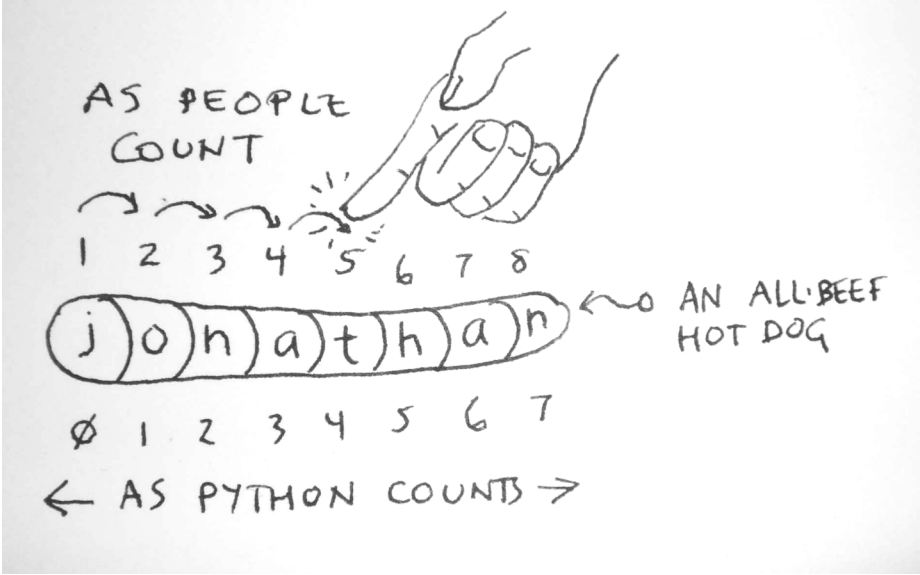
This finds a string, but doesn't help me when I want to use what I found. I need to take the shorter one out of the long one, and put it back together again, or just use part of it. **So now I have to focus on even simpler tasks. That means I need to find out…**

How to take a string apart and put it back together

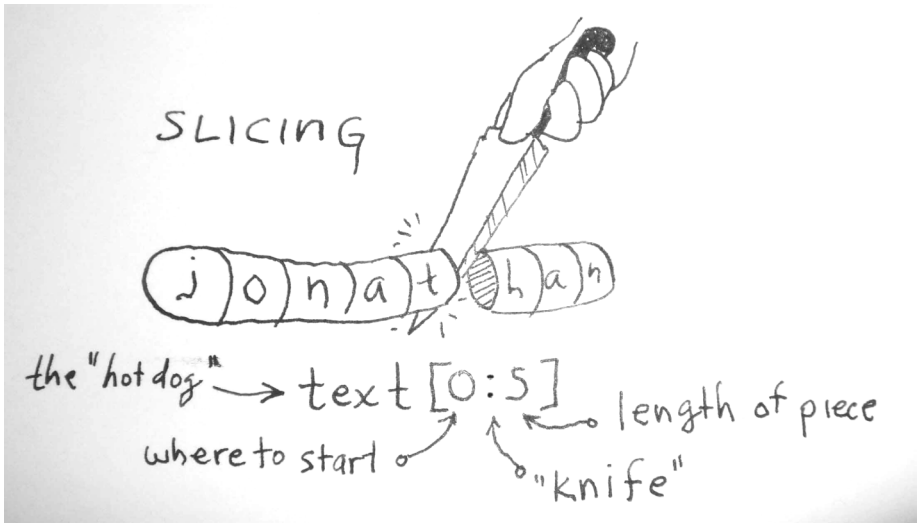
This is still a piece of Part 1, but a tinier piece. To understand how it works, I need to know why computers count the way they do. It is because computers count with binary numbers that are made up of zeroes and ones. Since the designers of a computer language such as Python do not want to waste *any* part of a computer's memory, that means that they count in binary (base 2) this way:

0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110 = 6
0111 = 7
1000 = 8
…and so on up to…
1111 = 15
if we are counting with only four bits. (Can you fill in the other numbers?)

Because computers can count starting from zero ***or*** starting from one, it is necessary to know whether I am counting *the position of something* (like where a character is in a string, where I start with zero), or the *number of things* (like **how many** characters are in a string, where I start counting with one). Here is a diagram that explains the difference between the ways to count:



In Python, taking a string apart is called *slicing* using the notation "[:]".Think of the string as a hot dog, which is taken apart is by "cutting" it at some point *between the letters*. For Python to do that, it has to know the place to cut. The only way to tell the computer where to cut is by giving it the numeric position of the two letters between which the "slice" will be made. The diagram below explains how "slicing" works. You can see that it uses both kinds of counting:



Putting two short strings back together again is also useful, but is easy. I just "add" them together with a plus operator ("+"). This kind of "adding" is more like gluing the two short strings together (the technical term is *concatenation*).

Here is an example: in pseudocode (which is almost Python):

Add the string "first" to "last"

Put it into a variable named whole_string

Print whole_string

The result to do the "adding" and put the two strings put back together, as written in Python, would be the code:

```
whole_string="first"+"last"
```

If we printed whole_string the result would be the string firstlast. Now it is time to test our knowledge of counting, "slicing" and "adding" strings. There is not much pseudocode to write to describe this program:

Cut a string into two pieces using several different ways of "cutting" and put it together again.

I have used a lot of *comments*—lines that begin with a "#" that Python ignores—to explain how each variation of slicing works. The program itself, called cut, expects you to give it a string—"something in quotes like this"—and a number where you want the cut to be made. See how cut prints out the results of each action.

```
def cut(text, at):
    # len(text) finds out how long the string is counting from 1
    # the string "YourName" contains eight letters
    length=len(text)
    print "Length is ", length

    # Slicing "[:]" takes the string apart but starts counting from 0
    # "YourName"
    #  01234567
    #
    # The LOCATION of a letter in the string is found by starting the
    #count at zero, NOT one. This is important! Watch what follows...
    #
    # Slicing also does something else. It adjusts the value of the
    # place where you take the string apart. If the value of the place
    # to take the string apart is 4, then [0:at] is [0:4-1] or [0:3]
    front=text[0:at]
    print "Front is ", front

    # Slicing also is smart enough NOT to adjust the value of at if
    # it is where you want to start the cut. [at:len(text)] will
```

```
    # equal [4:8-1] or [4:7] to give you the back of the string
    back=text[at:len(text)]
    print "Back is ", back

    # Slicing also can get the whole string in two ways, both giving
    # the same result but one shorter to write. [:] is the same as
    # [0:len(text)]
    print "Whole string by [:] is ", text[:]
    print "Whole string by [0:len(text)] is ", text[0:len(text)]

    # You can slice strings with negative numbers. The cutting will
    # start at the END of the string
    print "Cut from the end is ", text[0:-at]

    # You can cut out nothing at all! Simply write [at:at]
    print "Cutting out nothing at all is ", text[at:at]

    # And finally, you can put strings back together with the "+"
    # operator which "adds" strings together
    print "Front+Back is ", front+back
```

```
>>> cut("YourName", 5)
Length is  8
Front is  YourN
Back is  ame
Whole string by [:] is  YourName
Whole string by [0:len(text)] is  YourName
Cut from the end is  You
Cutting out nothing at all is
Front+Back is YourName
```

Finding a shorter string in a long one

Now we are ready to find a shorter string in a long one. I will use a made-up example with my name to show how to find the shorter name "jo" in "jonathan" and the longer name "nathan" in "jonathan." We will also see what happens when we look for "sally" in the string "jonathan," since it is not there. The pseudocode is short:

Put the string "jonathan" in a variable called my_name

Find out how long the string in my_name is

Find "jo" in my_name and print out the place it begins

Find "nathan" in my_name and print out the place it begins

Find "sally" in my_name and print out the place it begins

Here is the Python program and its output:

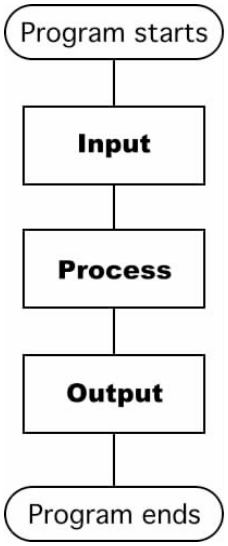
```
>>> def findname(text):
    print len(text)
    print text.find("jo")
    print text.find("nathan")
    print text.find("sally")

>>> findname("jonathan")
8
0
2
-1
```

This program uses both ways of counting. The ***length*** of the string "jonathan" is eight letters long. For length, Python started counting at one. But the ***place*** where the name "jo" is found is zero. For position, Python started counting at zero, and our experiment confirmed that. When we searched for "nathan", which we found at position 2—the third character, counting from 0, is at position 2 (0,1,2). But when we tried to find "sally" in the string "jonathan", it was not there. So Python let us know by giving a negative one (-1) to show that the end of the string "jonathan" was reached, but it did not find "sally".

Now let's turn our attention to getting a really ***huge*** string to work with—the HTML source code for a web page!

2. Reading the HTML source code for a web page

Here is a quick review of a program's structure, which corresponds to reading and printing the HTML source code for a web page. Our web-page-reading program looks like this flowchart:



Input is obtained from the web page.

The input is **processed** by putting the HTML source code into a variable. The HTML is one huge string.

Finally, **output** is produced by printing the string.

Part of the processing is new. It would take too long for use to write the Python code that reads a string from a web page. We *could* do it, but it would easily take us all semester. We have better things to do. So, we will use someone else's Python code, which is in a *library* of routines that deal with getting things from the web. ***You do not have to understand how to write this function in Python, but you do need to know how to use it in your program.***

First, to use these routines you must get the "invisible code" for them into your Python program. That is done with the `import` command, which brings them in from a library. But *which* library? You can find out by searching the web, but to keep it simple I will speed things up by telling you that the library is called `urllib` (URL is the jargon term for the address of a web page). Here is how you use it with the import command:

```
import urllib
```

That is it. After we write the pseudocode that outlines what we want to do, I will give you the Python commands from the `urllib` to get the HTML source code. You already know how to display it with Python's `print` keyword.

Pseudocode for the HTML reader and printer

The three basic actions we need to perform to get the HTML source code are:

1. Open the web page,
2. Get the HTML source code, and
3. Close the web page so we can let other people use it.

Here is the pseudocode:

```
Start the program
Get the address of the web page as a parameter from the keyboard
Import the web library (called urllib)
Open the web page
Read the HTML source code as a huge string
Close the web page
Print the string of HTML source code (it will be long)
End the program
```

Compare this pseudocode to the general form of the programs. This time it is straightforward. The initialization is to import the web library. Processing is reading the HTML source code from the web page into a variable that holds a string (many characters). The output is just to print the string. Later we will do something more interesting with it.

Python program to read and print HTML

Here is the sample program with only *some* of the output shown. As many of you found, printing everything wastes paper!

```
>>> def get(url):
    # The simple program to read a web page and print it
    # You will see a bunch of stuff that matches the HTML
    # for the web page you selected
    import urllib
    connection = urllib.FancyURLopener({}).open(url)
    text = connection.read()
    connection.close()
    print text

>>> get("http://www.cs.indiana.edu/svp/listall")

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
    <title>Sudoc Virtualization Project</title>

…

</div>
      <!--/PageFootMenuFmt-->
         </div>
</body>
</html>

>>>
```

First part of your assignment (more is coming)

5 points. Enter the Python program that cuts the "yourname" string apart. Experiment with cutting it at different places and save the output to turn in. Now run it using your full name, such as "Margaret Delia Johannson". Count how many characters are in your name, including spaces. Count starting at zero and list the starting position of each of your names (first, middle, and last, or as many as you have). Write these counts down. Now cut your name at different places , like at the space. Write down what you find. Did the Python program count spaces as a part of your name? Write down what you find.

5 points. Enter the Python program that finds a shorter string in a longer one. Use your full name instead of "jonathan". Change the program to find at least two different names in your name (or if your name is something like "wendolyn" pick strings like "wen" and "dolyn"). Also pick at least one string that is *not* in your name. Using the example of "wendolyn" you could pick "sam". Run the program and save the output to turn in. Write down on your lab page what happened, explaining it in your own words.

10 points. Find two or three web pages that have a list of items on them. Look at the page with your browser. Look at the HTML source with your browser. Enter the Python program that reads and prints the HTML source and run it. Write on your lab page what you see, describing ***briefly*** the differences. Look for any <li> tags and write down the URL for that page if you find any. Save the URL to use in the next part of this assignment.

Lab sheet (attach more sheets of paper if needed)

Lab for Week#: Date: Page 1

Problem as you see it:

Solution in pseudocode and diagrams:

Draft of Python code (show to AI to start typing on computer):

Lab sheet (attach more sheets of paper if needed)

Lab for Week#: Date: Page 2

Debugging (what was wrong, how you found it and how you fixed it):

Bug 1.

Bug 2.

Bug 3.

Bug 4.

Bug 5. (add more sheets if needed ☺)

Final Python program (attach printed version to this sheet)

Summary: