Text Processing

Although we have covered a lot of material, so far, we've been working with a relatively small number of programming constructs:

- For loop
- Assignment
- If statements
- Creating functions

We've been applying these constructs to images and sound. The designers of JES have included a number of features that make it easier to work with media, like setMediaPath() or pickAFile(). Let's start to look under the hood and see how these things are implemented. We'll start by exploring text processing in more detail.

Strings

We've already seen that text is stored in a string. A string is generally enclosed in double quotes. We used it to refer to filenames and output it with the print statement:

```
>>> print "Hello"
```

Would output "Hello" to the screen. "Hello" is the textual string that is being output.

We have also seen that we can output strings in the program area within a function:

```
def outputString()
    print "Hello"
```

The string is really storing 5 consecutive codes that corresponds to the letters 'H', 'e', 'l', 'l', and 'o.'.

Two strings can be combined to form a new string consisting of the strings joined together. The joining operation is called **concatenation** and is represented by a the plus sign (+).

For example, the following outputs "hello world":

```
s1 = "hello"
s2 = "world"
print s1 + s2
```

This outputs: helloworld

Note that Jython will not automatically adds a space between the words for us!

However there is an alternate concatenation operator, comma, which will add a space for us. You can take your pick as to which you prefer to use.

```
s1 = "hello"
s2 = "world"
print s1 , s2
```

This outputs: hello world

Why does Jython complain about the following?

```
>>> print "Dan Quayle said, "I love California; I practically grew up in Phoenix.""
```

Jython has a problem here because it thinks that the double quote before “I” matches up with the beginning of the quotation. It then tries to interpret “I love California…” as actual python code, which it is not.

One way to fix this is that python allows you to mix single quotes and double quotes. The other quotation can be in single quotes, and the inner quotation in double quotes:

```
>>> print 'Dan Quayle said, "I love California; I practically grew up in Phoenix."'
```

Another way to fix this is to use the proper escape character. If the backslash character is encountered, then python will interpret the next character as a code. For example:

\"	- Include the actual "
\t	- Tab
\n	- Newline

For example:

```
>>> print "Dan Quayle said, \"I love California;\nI practically grew up in Phoenix.\""
```

Will output:

```
Dan Quayle said, "I love California;
I practically grew up in Phoenix.”
```

Each character is actually encoded using a two-byte value using a format called Unicode. We can directly output a specific Unicode value using the escape sequence:

\uXXXX	- Unicode character, where XXXX is a code and each X can be 0-9 or A-F. http://www.unicode.org/charts/

We must preface the string with the letter u for Unicode. For example:

```
>>> print u"\u004A"
J
>>> print u "\u0439"
й
>>> print u"\uBEEF"
□
```

This is how you could output mathematical symbols or characters in other languages.

Strings as Arrays of Characters

Strings are arrays of characters. If we want to get the length of a string we can use the len function:

```
s = "Hello"
print len(s)            # Outputs 5
```

Given the string “Hello” it is stored in an array as follows:

H	**e**	**l**	**l**	**o**
0	**1**	**2**	**3**	**4**

Each character in the string is given an index. **This index starts at 0! It does not start at 1**, like most of the functions that you have been using. There are actually some good reasons to start counting at zero, but they mostly have to do with the architecture of the CPU.

We can tell python which character we want by giving the name of the string, then square brackets followed by the index. For example:

```
s = "Hello"
print s[1]              # Outputs e
print s[0]              # Outputs h
letter = s[4]
print letter
print s[5]              # ??
```

string[n:m] returns a substring of the string starting at index n, and up to but not including index m.

```
print s[1:4]        # Outputs ell
print s[0:5]        # Outputs Hello
print s[3:4]        # Outputs l
```

You can optionally leave out n or m. If n is missing it's assumed to be zero (the start of the string). If m is missing, it's assumed to be the end of the string:

```
print s[:4]         # Same as s[0:4] or "Hell"
print s[3:]         # Same as s[3:5] or "lo"
```

If you're not confused yet, then you can use negative numbers as indices, which will trim that many characters from the opposite end:

```
print s[:-1]        # "Hell" or trim rightleft "o" off
```

Looping through Strings

A string is actually a sequence, similar to the range function. This means that you can use the for loop on it. For example:

```
s = "hello"
for c in s:
    print c
```

This outputs:

```
h
e
l
l
o
```

Here is a sample that prints a message if a string contains the % sign:

```
def containsPercent(str):
    for c in str:
        if (c == "%"):
            print "Found percent!"
            return 1
    print "Did not find percent."
    return 0
```

```
>>>  containsPercent("foo%bar")
>>> containsPercent("foobar")
```

String Functions and Objects

It turns out a string in Python is a “object”. In fact, everything is in python is an object. What’s an object?

An object is simply something that encapsulates two things:

- data (like numbers or letters)
- methods or functions that operate on that data

To access the methods that operate on data we use the “dot notation”.   We use the name of the object followed by a dot followed by the method to invoke on that object.

For example, the capitalize method will return a new string with the first letter capitalized:

```
s = "hello"
print s.capitalize()            # Outputs "Hello"
print s                         # s is still "hello"
```

Here are some sample methods with strings:

upper()	- Translate string to uppercase
lower()	- Translate string to lowercase
find(targetstring)	- Returns the index where targetstring is found in the string -1 if targetstring is not found
startswith(prefix)	- Returns true (1) if the string starts with prefix, false (0) otherwise
endswith(prefix)	- Returns true (1) if the string ends with prefix, false (0) otherwise
replace(target,replace)-	Searches for the target string and replaces it with the replace string.

Examples to try:

```
s = "hello"
print s.upper()
print s.lower()
print s.startswith("hell")
print s.startswith("ho")
print s.endswith("lo")
print s.find("el")
print s.replace("ell", "MOO")
print s.replace("l", "ZZZ")
```

Remember that none of these functions change the original string, they just return back a value or a copy of the string with the appropriate modifications.

Exercise: Your task in this problem is to write a pig latin translator. One of the rules for pig latin is as follows:

For words that begin with consonants, move the leading consonant to the end of the word and add "ay." Thus, "ball" becomes "allbay"; "button" becomes "uttonbay"; "star" becomes "tarsay", and so forth.

Let's make a function that converts an input string into pig latin using this rule, assuming the word starts with a consonant:

```
def pigLatin(word):
  firstLetter = word[0:1]
  rest = word[1:]
  pig = rest + firstLetter + "ay"
  return pig
```

Now, let's add a second rule. For wards that do not begin with consonants, simply add "way" to the end. Thus, "eat" becomes "eatway" and "air" becomes "airway". To do this we need a function that can determine whether or not the first letter starts with a consonant. This function returns true (1) if so, and false (0) if it starts with a vowel.

```
def startsWithConsonant(word):
  firstLetter = word[0:1]
        if (firstLetter == "a") or (firstLetter == "e") or (firstLetter == "i") or
         (firstLetter == "o") or (firstLetter == "u"):  # All on same line
          return 0
  return 1
```

Now we can use this with an if else statement. The block that follows the else is only executed if the Boolean condition in the if statement is false:

```
def pigLatin(word):
  if (startsWithConsonant(word)):
    firstLetter = word[0:1]
    rest = word[1:]
    pig = rest + firstLetter + "ay"
  else:
    pig = word + "way"
  return pig
```

Input and Output

We have already seen how to print strings out. How about a way to type strings in from the keyboard? There is a way to do it, but it's a bit unwieldy. The solution is through "pop-up" windows that really use a Java function. We'll say more about this later, but for now you'd add something like this to your program:

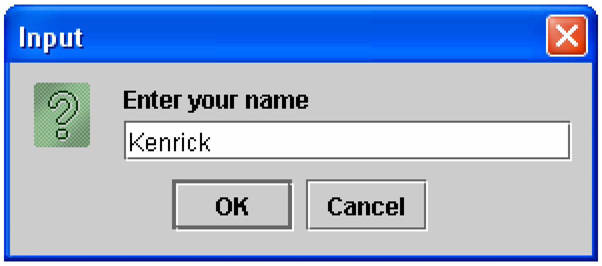
```
import javax.swing as swing

def getInput(message):
  return swing.JOptionPane.showInputDialog(message)
```

This imports the "Swing" library from Java and invokes the "showInputDialog" function when the getMessage subroutine is called. Now, whenever you want to pop up a dialog box that prompts the user for a string and returns it, you would use:

```
stringVariable = getInput("Prompt")
```

If the user presses cancel, then the string returned is empty. For example:

```
>>> s = getInput("Enter your name")
```



```
>>> print s
Kenrick
```

Note that the value entered by the user is always a string. If you want to treat it like a number, for example as an integer or double, then typecast it:

```
num = int(getInput("Enter an integer"))
print num

flt = float(getInput("Enter a floating point value"))
print flt
```

In-class Exercise:

It is recommended that you maintain your training heart rate during an aerobic workout. Your training heart rate is computed as:

0.7(220-a)+(0.3*r)

where **a** is your age in years and **r** is your resting heart rate. Write a program that inputs a and r and then outputs your training heart rate.

Example:

You are running a marathon (26.2 miles) and would like to know what your finishing time will be if you run a particular pace. Most runners calculate pace in terms of minutes per mile. So for example, let's say you can run at 7 minutes and 30 seconds per mile. Write a program that calculates the finishing time and outputs the answer in hours, minutes, and seconds.

Input:

```
        Distance : 26.2
        PaceMinutes:  7
        PaceSeconds: 30
```

Output:

```
        3 hours, 16 minutes, 30 seconds
```

Here is one algorithm to solve this problem:

1. Express pace in terms of seconds per mile, call this SecsPerMile
2. Multiply SecsPerMile * 26.2 to get the total number of seconds to finish. Call this result TotalSeconds and make it an integer.
3. There are 60 seconds per minute and 60 minutes per hour, for a total of 60*60 = 3600 seconds per hour. If we divide TotalSeconds by 3600 and throw away the remainder, this is how many hours it takes to finish.
4. TotalSeconds mod 3600 gives us the number of seconds leftover after the hours have been accounted for. If we divide this value by 60, it gives us the number of minutes, i.e. (TotalSeconds mod 3600) / 60
5. TotalSeconds mod 3600 gives us the number of seconds leftover after the hours have been accounted for. If we mod this value by 60, it gives us the number of seconds leftover. (We could also divide by 60, but that doesn't change the result), i.e. (TotalSeconds mod 3600) mod 60
6. Output the values we calculated!

In-Class Exercise: Write the code to implement the algorithm given above.