Lab 5: Strings - Slice and Dice

CSE/IT 107

Introduction

The purpose of this lab is to introduce strings and reading files.

Reading

Chapter 8 in Think Python: How to Think Like a Computer Scientist (Think)

Coding Conventions

Follow PEP-8 recommendations for your code.

Problems

Make sure your source code files are appropriately named. Make sure your code has a main function; use `boilerplate.py` you created in Lab 1.

Think Python is available from `http://www.greenteapress.com/thinkpython/`. Page numbers and exercise numbers refer to those found in the PDF file available at the website.

1. Exercise 8.1 *Think Python*, page 72. Use a while loop. Name your source code `think_8-1.py`

2. Rewrite the function in Exercise 8.1 using negative indexing (see section 8.2 in Think). Use a while loop. Add the function to `think_8-1.py`.

3. Exercise 8.2 *Think Python*, page 73. Write the code as a function. Name your source code `think_8-2.py`. The function should return a list of duckling names. The `in` operator can be used to check whether a value is contained inside another object such as a string or list. The `in` operator returns True or False. For example:

```
letters = 'abcd'
i = 'a'
if i in letters:
    print i + ' is in letters'
else:
    print i + ' not in letters'
```

Use `in` your function to test to see in the letter is a 'O' or 'Q'. Do not test using equality.

4. Given

```
s = 'spam'
t = 'ni!'
```

With an interpreter, evaluate the following expressions:

- 'The Knights who say, ' + t
- 3 * s + 4 * t
- s[1]
- s[1:3]
- s[2] + s[:2]
- s[2] + t[:2]
- s[:] + t[:]
- s + t[-1]
- s.upper()
- t.upper().ljust(6) * 3
- t.upper().rjust(6) * 3

More information on slicing can be found here:

`http://docs.python.org/2/tutorial/introduction.html#strings`

Now, given `s` and `t`, write a program using string operations that produces the following output:

```
'NI'
'ni!spamni!'
'Spam Ni!  Spam Ni!  Spam Ni!  '  # two spaces between ! and S
'spam'
['sp', 'm']
'spm'
'spniam'
```

You can place everything in the `main()` function. You cannot change `s` or `t` or add any other variables. Name your source code `spamni.py`

5. Exercise 8.4 *Think Python*, page 74. Name your source code `think_8-4.py`
6. Exercise 8.5 *Think Python*, page 75. Name your source code `think_8-5.py`
7. Exercise 8.6 *Think Python*, page 75. Name your source code `think_8-6.py`
8. Exercise 8.7 *Think Python*, page 76. Name your source code `think_8-7.py`
9. Exercise 8.8 *Think Python*, page 76. Nothing to turn in here, but familiarize yourself with string methods. The url is:

   `http://docs.python.org/2/library/stdtypes.html#string-methods`

Caesar Cipher

10. Write a program that uses a Caesar cipher (see `http://en.wikipedia.org/wiki/Caesar_cipher` to encrypt and decrypt a string. Name your program `caesar.py`.

    You will pass in three command line arguments: one for the string to cipher, one for the direction to shift ('right' or 'left'), and one for the distance to shift.

    Make sure to enclose the string to encrypt in quotes, otherwise it will be stored as a list. Convert the string to lower case. Spaces are left as spaces. Write a function that checks to make sure the string consists of only characters a - z or a space. Use the Python provided string methods `isalpha()` and `isspace()` to determine if the string is valid or not.

    Once you have a valid string, then you can encrypt it using a Caesar cipher. Pass in as parameters the string, the direction you want to shift ('left' or 'right'), and the number of positions you want to shift. As you can't perform modular arithmetic on characters (a string of length one), you will first have to convert the string to its ASCII equivalent and back to a character. Python provides two builtin functions `chr()` and `ord()` that will help in this task. `chr(i)`, returns a string of one character with i an integer, 0 ¡= i ¡ 256. `ord(c)` returns the integer value of the one character string c. Since all numbers are lower case, you will find it easier if you subtract off `ord('a')` from each character before you take the modulus 26. Subtracting off `ord('a')` will make the letter 'a' zero.Of course, you have to add `ord('a')` to convert the number to the correct character.

    To decrypt the encrypted text, you perform the inverse of the encryption. If you called the Caesar function with a left shift of 3 to encrypt the string, you call the function with a right shift of 3 to decrypt it.

    Test with the string 'the quick brown fox jumps over the lazy dog'. Encrypt the string with a left shift of 3. Print the encrypted string. Decrypt the string and print to stdin. For example,

```
run caesar.py 'the quick brown fox jumps over the lazy dog' 'left' 3
encrypt: qeb nrfzh yoltk clu grjmp lsbo qeb ixwv ald
decrypt: the quick brown fox jumps over the lazy dog
```

ROT13

Once you have your Caesar cipher working, you have ROT13 encryption/decryption as well. For information on ROT13 see `http://en.wikipedia.org/wiki/ROT13`. Try a few of the words in the table next to the section entitled "Letter Games" to make sure your encryption/decryption scheme is working correctly.

Word Count

Splitting Strings

Python has a nice builtin function for strings called `split()`, which splits a string into substrings. You will find many uses for split. Try the following examples in an interpreter to get a feel how split works:

```
import string
for w in string.split("to be or not to be"):
    print w

for w in string.split("Mississippi", "i"):
    print w
```

In the second example, the `i` is the delimiter in which to split the string on. The default delimiter is any whitespace character. Delimiters can be longer than a single character. Split Mississippi on ss. Notice how the delimiter is removed from the substrings.

Reading a File

It is very easy to open a file in Python. The following program opens a file for reading, echoes the contents to the terminal, closes the file, and exits.

```
#!/usr/bin/env python

from sys import argv


def main():
    infile = open(argv[1], 'r')
```

```
    data = infile.read()
    print data
    infile.close()


if __name__ == '__main__':
    main()
```

To run, assuming the name of the program is `echo.py` and the file `getty.txt` is available on Moodle:

```
run echo.py getty.txt
```

Word Count

On *nix systems, there is a nice little utility called `wc` which analyzes a file and determines the number of lines, words, and characters in the file.

11. Write a program named `wc.py` that reads in a file and prints out the counts of lines, words, and characters.

    Write a function `line_count` which counts the number of new lines (`\n`) in the file. Use Python's `find` method.

    Write a function `word_count` which counts the number of words in the file. For purposes here, we will consider any text separated by whitespace a ‘‘word”. This means you will count `--` as a word.

    Write a function char_count which counts the number of characters in the file. Count all characters.

    Your output should match that of the `wc` command. That is

```
run wc.py getty.txt
5 278 1476 getty.txt

$ wc getty.txt
   5  278 1476 getty.txt
```

Anagrams

You are going to write two programs to determine if two words are anagrams. Two strings are anagrams if they are simply rearrangements of each other. For example, `earth` and `heart` are anagrams. While `googl` and `apple` are not. If the words differ in length they obviously are not anagrams.

Each program will have a function `is_anagram(s, t)` that returns true if the strings are anagrams or false if not. `is_anagram()` implements the given algorithm. Your `main()` function processes the input and prints whether the words are anagrams or not.

Enter the two words via the command line. You can simply use `argv[1]` and `argv[2]` to get the words. Your programs should check to see that the strings are of the same length. Otherwise it should print an error.

12. *Check Off Algorithm*

    Implement a solution that checks to see that each character in the first string occurs in the second string. If each character in the first string occurs in the second string, then the strings are anagrams. The algorithm can be implement by checking if the first character in the first word is in the second word, the second character in the first word is in the second word, etc. Assume words have unique characters.

    The `break` statement is useful here. You can use a boolean to see if the character in the first word is in the second word. If it isn't you can break out of the loop. Section 7.4 of Think Python discusses the `break` statement.

    Name your source code `check_off.py`.

13. *Sort and Compare Algorithm*

    Implement a sort and compare algorithm to find if two words are anagrams. The words are only anagrams if they consist of exactly the same characters. So if you sort each string and then check to make sure they consist of the same characters, you can determine if the words are anagrams.

    Since strings are immutable, you cannot sort strings directly. You first need to convert them to a list and then you can use python's builtin `sort()` function. To convert a string to a list and sort it, it is as simple as doing the following:

    ```
    >>> s = 'hello'
    >>> s = list(s)
    >>> s.sort()
    >>> print s
    ```

    Name your source code `sort.py`.

Submission

Create a tarball of your *.py files.

```
tar czvf cse107_firstname_lastname_lab5.tar.gz *.py
```

To check the contents of your tarball, run the following command:

```
tar tf cse107_firstname_lastname_lab5.tar.gz *.py
```

You should see a list of your Python source code files.
Upload your tarball in Moodle before the start of you next lab.