# Lab #9

## LAB OBJECTIVES

- Strings, Lists and Tuples (oh, my).
- Function calls with mutable objects.

## CODE EXPERIMENTS

***Note: This section contains some material that is either new or only loosely covered in class. You should review this section first before attempting the activities required for this lab.***

1. What is displayed when the following is entered in the Python Shell:

```
>>> s = " "
>>> myList = ['a','b','c']
>>> s.join(myList)
```

2. Try the following variation:

```
>>> s = "-"
>>> myList = ['a','b','c']
>>> s.join(myList)
```

   Do you understand what is displayed?

3. Suppose myList was defined as:

```
>>> myList =['s','-','c','r','-','t']
```

   What would you do to get a string that contained: "s-cr-t"?

4. Tuples are defined as follows:

```
>>> t = (4,5,6)
```

   But you can also define a tuple as

```
>>> t = 4,5,6
```

   What part of the code actually tells Python that you want a tuple?
   What role do the parentheses fill?

5. What happens when you try the following:

```
>>> t.append(7)
```

   How about other operations that work with Lists and Strings?

6. Consider the function defined as:

```
def parse(line):
    fields = line.split(",")
    return fields[0], fields[1:]
```

What is the result when you type:

```
>>> parse("Mary, 9,5,9,8")
```

How about:

```
>>> name, scores = parse("Mary, 9,5,9,8")
```

## ACTIVITIES

0. We are going to create a function that generates a deck of playing cards. For our purposes, a card is defined by a tuple that is a rank and a suit. The deck is a list of cards. Start with the following code to define the ranks and suits:

```
ranks = ("Ace", "2","3","4","5","6","7","8","9","10","Jack","Queen","King")
suits = ("Spades", "Clubs","Diamonds","Hearts")
```

Your function (Create Deck) should return a list that has all of the combinations of ranks and suits (52 cards in all)

The beginning of a sample run of your program may look like this:

```
>>> deck = createDeck()
>>> deck
[('Ace', 'Spades'), ('Ace', 'Clubs'), ('Ace', 'Diamonds'), ('Ace', 'Hearts'), ('
2', 'Spades'), ('2', 'Clubs'), ('2', 'Diamonds'), ('2', 'Hearts'), ('3', 'Spades
'), ('3', 'Clubs'), ('3', 'Diamonds'), ('3', 'Hearts'), ('4', 'Spades'), ('4', '
Clubs'), ('4', 'Diamonds'), ('4', 'Hearts'), ('5', 'Spades'), ('5', 'Clubs'), ('
5', 'Diamonds'), ('5', 'Hearts'), ('6', 'Spades'), ('6', 'Clubs'), ('6', 'Diamon
ds'), ('6', 'Hearts'), ('7', 'Spades'), ('7', 'Clubs'), ('7', 'Diamonds'), ('7',
```

1. Create a function (shuffle) that takes a deck of cards as a parameter. The result of the function is that the deck is arranged in a random order (Note, I didn't say it "returns" a shuffled deck – it should alter the deck that is passed in).

    a. There are a variety of algorithms for shuffling the array – but all of them involve moving the contents of the list found at a random location.

b. When you are testing your outcome --- watch for a couple of common mistakes. Make sure that you still have 52 cards in your list when you are done. Make sure that you don't have any duplicates cards.

```
>>> deck = createDeck()
>>> shuffleDeck(deck)
>>> deck
[('10', 'Diamonds'), ('5', 'Spades'), ('Jack', 'Clubs'), ('2', 'Spades'), ('2',
'Clubs'), ('7', 'Clubs'), ('6', 'Diamonds'), ('8', 'Clubs'), ('Queen', 'Diamonds
'), ('10', 'Clubs'), ('5', 'Diamonds'), ('4', 'Hearts'), ('3', 'Clubs'), ('10',
'Spades'), ('Ace', 'Clubs'), ('3', 'Hearts'), ('9', 'Hearts'), ('4', 'Spades'),
('10', 'Hearts'), ('Jack', 'Hearts'), ('Queen', 'Hearts'), ('7', 'Diamonds'), ('
```

2. Hangman is a classic game where a player tries to identify a secret word by guessing letters. Let's create a player-vs.-computer version:

   Rules:

   a. The Computer selects a secret word from a file[1] and displays dashes for each letter in word. Note – the dashes should be separated by spaces so that it is readable (e.g. – – – – – –).

   b. The Human Player guesses a letter. Whenever the secret word contains the guessed letter, the computer reveals ***all*** of the instances of that letter in the word (e.g. – e – – e –). When the letter is not in the word, the guess is wrong.

   c. The game ends when all the letters have been revealed, or the guesser has 10 incorrect guesses.

   Your program should:

   a. Provide a brief welcome message and have clear prompts about what to do next.

   b. Display the current state of the word in a readable fashion:
   "– e – – e –" is readable.
   [" - ","e"," - "," - "," - ","e"," - "] is not.

   c. Display an alphabetical list of the letters that have already been guessed

   d. Display the number of guesses remaining. (Remember that only incorrect guesses count against you).

## SUBMIT

Submit your work as attachments to the Moodle Assignment for Lab 9 by Friday evening.
There will be a post-lab assignment for this week.

[1] You will have to create your own file of about 20 words. Make sure there are words of different length.