Handout 6

Lists and Tuples.

List is a built-in Python type, e.g. `['a', 'b', 'c'], [ 3, 5, 6].`
List is a *sequence*, i.e. it is used to represents finite ordered sets.

Sequence functions len(), max(), min(), sum() are applicable to lists.

Ordering starts at 0.
List can contain arbitrary Python objects, e.g.

```
['a', 23, 5.6], [['cs799', 'Python'], foo, 456 ]
```

Lists are *mutable*, i.e. can be changed through operations, functions and methods.

Tuples are unchangeable lists, they use () instead of []; Tuples are created using , (comma) operator

CONSTRUCTING A LIST

- Using [] notation:
  - empty list: `[]`
  - `['a', 23, 5.6]` – a 3-item list
  - `[a, b, c]`– list containing *variables* a, b, c

```
>>> a = 5; b = 8; c = -1
>>> [a, b, c]
[5, 8, -1]
```

  - `['a', 'b', 'c']` – list of strings
  - `[['a', 'b', 'c'] ['d', 'e', 'f']]` – two-dimensional list, i.e. a matrix
- Using the type **constructor**: `list()`

```
list(), list("green"), list((2, 3, 4))
```

SEQUENCE FUNCTIONS :

len(), min(), max(), sum()

LIST OPERATIONS: [::],+ , *, IN

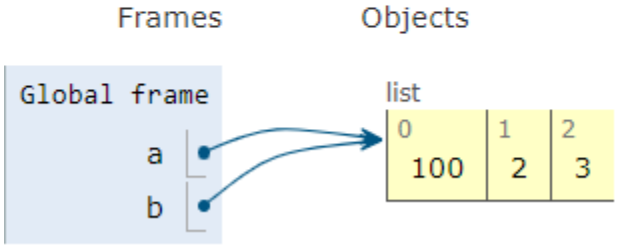
```
>>> lst = ['a', 23, 5.6, 45]
>>> lst[0]
'a'
>>> lst [-1]
45
>>> lst [1:3]
[23, 5.6]
>>>lst [0:4:2]  #start at 0, end at 4, move to next by adding 2
 ['a', 5.6]
```

* List variables store references

```
a = [1,2,3]
b = a
b[0] = 100
print(a)
print(b)
```

Print output (drag lower right corner to resize)

```
[100, 2, 3]
[100, 2, 3]
```



LIST METHODS

In the summary below, x refers to a Python object of any type, index is an int, and **`alist`**, lst denote a list.

`alist.append(x)` add an item x to the end of **`alist`.** **Does not return anything.**

`alist.insert(index, x)` insert an item x at a given index in **`alist`.**.

`alist.remove(x)` remove the first occurrence of the item x from **`alist`.**

`alist.extend(lst)` append all the individual items in lst to **`alist`.**. **Does not return anything**.

`alist.index`(x) return the index of the item x in **`alist`**, ERROR, if x does not appear in **`alist`**

PRACTICE PROBLEMS

1. Create a list of punctuation symbols in a given string using a list
2. User will enter strings showing day and number of cookies sold on that day, terminated by -1, as shown:

```
Mon 3.5
Tue 5
Mon 10
Fri 7
Sat 4.5
-1
```

Output

1) a count of total values on each day of the week
2) a * for those days when the number of cookies sold > average

as shown:

```
Mon   Tue   Wed   Thu   Fri   Sat   Sun
13.5  5     0     0     7     4.5   0
*     *                 *
```

COMPARISON: ==, <, <=, >, >=

Two lists are equal if all of their their individual elements are equal, e.g.

```
>>> a = ['a', 'b', 'c']
>>> b = ['A'.lower(), 'b', 'c']
>>> a == b
True
```

MORE LIST METHODS

In the summary below, x refers to a Python object of any type, index is an int, and **`alist`** denotes a list.

`alist.copy()`	create a copy of the contents of **`alist`**
`alist.sort()`	sort the items in **`alist`**
`alist.count(x)`	return the number of times item x appears in **`alist`**
`alist.reverse()`	reverse the items in **`alist`**

PRACTICE PROBLEMS

For problems 3 - 5, create a function implementing the described functionality:

3. Passed a text-storing string, return a sorted list of words in it that includes each word only once. Deal with punctuation the best way you can.

4. Passed in a text-storing string, print out a count of all letters in the string, i.e., for each letter of the alphabet, calculate how many times it is found in the string. Use a list to store letter counts.

5. Implement the Hangman using a list to represent the displayed template with dashes. Hangman is a two-player game in which one player, in our case, the program, picks a word (you can hardcode it) that the other player (the user) has to guess. The program originally reveals only the first and the last letters, and dashes for every other letter, e.g. `N------K`, if the word is `NOTEBOOK`. At each turn of the game, the user suggests a letter and the program responds by either stating that the word does not have the letter, or revealing the letter in the word, e.g. O in `NO---OOK`.

   The game stops when either
   - user guesses the entire word, and then user is the winner, or
   - when user has proposed 6 incorrect letters (not found in the word). In this case, the program wins.

   Here's a sample run of the program:

```
Let's play a game of Hangman.
I have picked a word that starts with letter T and ends with E.
Here's the template in which each dash denotes a single letter:
T - - - - - - - - E
Now it's your turn to guess a letter.

Please enter your next guess: F
Incorrect. Letter F does not occur in this word.
```

```
Please enter your next guess: E
Correct.
T E - - E - - - - - E

Please enter your next guess: Z
Incorrect. Letter Z does not occur in this word.

Please enter your next guess: P
Correct.
T E - P E - - - - - E

Please enter your next guess: Q
Incorrect. Letter Q does not occur in this word.

Please enter your next guess: O
Incorrect. Letter O does not occur in this word.

Please enter your next guess: L
Incorrect. Letter L does not occur in this word.

Please enter your next guess: I
Incorrect. Letter I does not occur in this word.

You lost this game. The word is TEMPERATURE
```

LIST COMPREHENSIONS

- List comprehensions provide a concise way to create a sequence from another sequence
- A comprehension is computed via a set of looping and filtering instructions.
- A list comprehension consists of **brackets** containing an **expression** followed by a **for** clause, **then zero or more for or if clauses**. The result will be a list resulting from evaluating the expression. Here are some examples:

```
>>> list1 = [x for x in range(0, 5)]
>>> list1
[0, 1, 2, 3, 4]

>>> list2 = [0.5 * x for x in list1]
>>> list2
[0.0, 0.5, 1.0, 1.5, 2.0]

>>> list3 = [a for a in list2 if a < 1.5]
>>> list3
[0.0, 0.5, 1.0]
```

Note that the variables in the **for** clause are **local to the comprehension**, i.e.'do not leak' outside.

```
[expression for x in sequence if condition]
 is equivalent to the following computation

resultLst = [] #start with an empty list
for x in sequence :
      if condition :
            resultLst.append (expression)

resultLst
```

PRACTICE PROBLEMS

1. What is the result of the following list comprehensions?

   - `a = [x for x in 'A little knowledge is a dangerous thing.'.split() if len (x) <= 3]`
   - `b = [ch.upper() for ch in "university" if ch.lower() not in 'iouea']`
   - `c = [ x % 3 for x in range (20)]`

2. A user will enter a sentences. Produce a list of all starting letters using a list comprehension. For example, when user enters
   The goal of this sentence is to practice list comprehensions.
   the output should be ['T', 'G', 'O', 'T', 'S', 'I', 'T', 'P', 'L', 'C']

3. User will enter a string of numbers separated by commas. Compute the **sum, min** and **max** of the numbers. Use list comprehension to create a list of numbers
   For example, for user input 2, 4.5 ,7,3
   the output should be 16.5 2.0 7.0