# Lab 7: Classes/Objects, Functions, Search & Sorting

## Due date: March 09, 6:00pm

## Lab Assignment: 50 points

**Lab type:** This is an individual lab. Each student will submit his/her set of deliverables.

**Collaboration:** Students are allowed to consult their peers in completing the lab. Any other collaboration activities will violate the non-collaboration agreement.

**Purpose:** The purpose of this lab is to practice functions, objects/classes, searching and sorting techniques.

## Lab Tasks

Note: Please consult the instructor if any of the steps are unclear or you get stuck in an activity.

## Task 1 (15 points)

Write a function named **cal_area()**. This function takes two arguments: the list **Geometric_Shapes** in the format given above, and a string denoting a specific geometric shape for which the area needs to be calculated. This function must return a value that is the maximum area amongst all the shapes that is specified as the choice mentioned above. For example, the specific shape for which the area is to be calculated is “rectangle”. Then the function **cal_area()** must display “98 is the max area amongst all shapes of type: rectangle “
Your function should not make any assumptions about the lengths of any lists.

```
class Shape:


    def __init__(self, name, dim):

        self.name = name

        self.dim = dim




class Dimension:

    def __init__(self, a,b):


        self.a = a

        self.b = b

 Geometric_Shapes = [

Shape('rectangle', [Dimension(5,6), Dimension(14,7), Dimension(5,2)]),


Shape('square', [ Dimension(5,5), Dimension(4.2,4.2)]),


Shape('ellipse', [Dimension(5,6), Dimension(7,7)]),

]
```

## Task 2 (15 points)

You will now write a program that enables sorting and searching for an employee in an office. For doing this, you will create "employee" objects whose attributes are given below:

| Name | age |
|---|---|
| Michael | 45 |
| Jim | 35 |
| Pam | 30 |
| Dwight | 40 |
| Creed | 60 |

Store the employee objects inside a list. Your first task is to sort these employees according to their age in ascending order. You can use any sorting algorithm (insertion sort or selection sort) that you like; your sorting algorithm should sort the employees in-place (i.e., the original list must be sorted). Display the sorted list.

Once the list is sorted, append a new employee to the list ("Toby", 45). Now sort the list again by using the same sorting technique that you developed earlier.

Finally, once you have a sorted list you can now implement the linear search algorithm to search for employees whose age matches the search key. For instance, you can search for the employee whose age is 40 and you should be able to display the employee name.

Hint: Think about which functions you will have to define/overload in the employee class for searching and sorting in the way that you have been asked to.

## Task 3 (5 points)

Write a module that defines a function named **is_even()** that accepts a number as its only input parameter and returns True and False based on whether this number is even or odd. Now, in another program, import the module that contains the definition of is_even() and prompt the user for a number. In the second program, you have to call is_even() to check if the user input is one of the following: positive and even, negative and even, positive and odd, negative and odd.

Display the category to which the number belongs. Note that you must call the is_even() function defined in the first module from inside the second program to do this task.

## Task 4 (15 points)

Write a function linear_Search_uniq () to search a key from a list of non-unique numbers.

(a)The function should take in two arguments: a list and the search key and return a new list containing unique numbers. For example:

a = [1,3,5,4,5,3,4,3,3]

key = 3

linear_Search_uniq(a, key) = [1, 5, 4, 5, 4]

Write two unit test cases for this function.

(b) Write another function **selection_sort()** that uses the unique list and sorts in ascending order by applying selection sort. Show the number of comparisons in each iteration and the elements that are swapped in each iteration along with the intermediate sorted list in every pass. Write two unit test cases for this function.