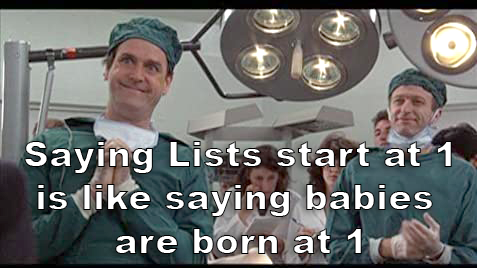
| Unit 7 Resources & Problems Python [Learning Plan Index - Python](https://docs.google.com/document/d/1Ys_VACt7p2P5NUuhDetAiOVoUwDGXNA_h2P2aBKoZ5Y/edit?usp=sharing)    *Unit 07 of Python Programming - Unit 7 Problems Python* | |
| --- | --- |
| Learning Targets  This unit we will…  Explore lists and the different ways they can be used as well as some of the different types of sorts involved with lists.  I can…   * Create a list with a set group of values. * Create a list of a set length with a group of random values * Insert sort, merge sort, and bubble sort a list. * Create a list of boolean values. * Find and use list indexes to manipulate and output list data. * Create a method to shuffle the contents of a list. * Read and understand existing code so that I can modify it to perform a different function. * Solve logic problems   Vocabulary: list, sort, bubble sort, merge sort, insert sort, random values, list indexes, shuffle, logic | |
| Learn About It!  *You can explore some, or all of these resources. If you want to see a resource again, go for it!*  [Learning Plan Index - Python](https://docs.google.com/document/d/1Ys_VACt7p2P5NUuhDetAiOVoUwDGXNA_h2P2aBKoZ5Y/edit?usp=sharing) *These Collab documents review the concepts of each unit with code you can run and modify.* | |
| Evidence of Learning  *Complete the following programming exercises.*  [Grading Rubric](https://docs.google.com/document/d/1shjqolaw_5tSX9T5OJ2FZuBeon7K3hDrYEJ5m1ltSEw/edit?usp=sharing) | |
| Unit Programs  Review:   1. [Colab - Lists](https://colab.research.google.com/drive/1skLw_vqZ2_4iBeAjc61MqEElbXjqyJA7)   Once you have reviewed the Colab document complete the problems below. There are tips, sample code, and links to sample code that you will use within the Colab documents, you also may want to refer back to early colabs. There will be two sets of problems to do, the first group can be done in a single file and the turtle program should be done in a separate file. There are pictures of what your output should look like below. Name the files **Unit07\_YourLastName.py and Unit07Turtle\_YourLastName.py**, if you do this set of problems in [repl.it](https://repl.it/) name the repl.it Unit07\_YourLastName and Unit07Turtle\_YourLastName and turn the share links into the classroom.  **YOU BETTER USE LISTS AND/OR SEQUENCES TO SOLVE THESE PROBLEMS!!**  In the problems in this chapter you are going to need to create a bunch of lists with random numbers for you to manipulate in your program. Below is sample code that is creating a random list of 1000 numbers from 0-9. Everytime you need a list for a problem you can change the number at the end of the name of the list numbers1, numbers2, etc. and then generate however many random numbers you need. For creating a list of boolean values check the colab document for some sample code.  import random numbers1 = [] #create empty list for x in range(1000): #for loop from 0-999  numbers1.append(random.randint(0,9)) #fill list with random numbers 0-9  **Unit07\_YourLastName**  **Sample output is shown below**   1. Count of Digits - (12 points) -  Create a list with 1000 random numbers from 0-9 in it. Now write a program that counts how many 0’s, 1’s, 2’s, 3’s, 4’s, 5’s, 6’s, 7’s, 8’s, and 9’s are in the list of 1000 numbers and output those results, you probably want to use a list since this unit is all about lists. Sample output is below. 2. Smallest Index - (12 points) -  Required function heading - **def indexOfSmallestElement(lst): returns minIndex** Write a function that takes a list as a parameter. The function will search the list and return the index number of the smallest number in the list. Create a list with 15 random numbers 0-100 in it. Output the random list and then the value of the smallest number and then the value of that numbers index, by using the function you created. Sample output is below. 3. List Shuffle - (12 points) -  Required function heading - **def shuffle(lst): returns shuffled list** While there is already a function that exists that you can use to [shuffle lists](https://www.w3schools.com/python/ref_random_shuffle.asp) I want you to make your own function instead of using an existing one. First generate a random list of 10 numbers from 0-100. Print the unsorted list, then [sort the list](https://www.w3schools.com/python/ref_list_sort.asp) and output the sorted version, then pass your list into your function to shuffle it and output the shuffled results. 4. Revise Selection Sort - (14 points) -  Required function heading - **def selectionSortsm(lst):** Required function heading - **def selectionSortlg(lst):** This [repl.it has the code for the selectionSortsm](https://replit.com/@MrReynolds/07PProblem4SelectionSort#main.py), the repl.it also has a simple list at the bottom that is sorted by the function, all you need from this repl.it is the code for the function. If you run the function in the repl.it you will notice that in the output it tells you how many swaps were made in order to sort the simple list.  There are a number of different sorting algorithms and we probably could do a whole unit just on sorting. This [video (2:07)](https://www.youtube.com/watch?v=BeoCbJPuvSE) shows 18 different sorting algorithms working side by side, you’ll notice that insertion and bubble sort are much slower then the merge sort. This [video (10:37)](https://www.youtube.com/watch?v=qk7b4-iyCJ4) briefly summarizes the different types of sorts and all the sorts we are doing in this lesson are covered in the first four minutes, the merge sort problem I am giving you is not fully a merge sort since it is only sorting two sorted lists into one sorted list. For this problem copy the selectionSortsm that I have shared with you into your code. Make a list of 10 random numbers 0-100 and then make a [copy()](https://www.geeksforgeeks.org/python-list-copy-method/) of that list so you have two copies of the same list. Make a copy of selectionSortsm and rename it to selectionSortlg so you now have two copies of the function in your code.   The selectionSortsm function starts at the beginning of the list and searches the whole list for the smallest element and if it finds one that is smaller then where it started it swaps the elements so that the smallest element is at the start of the list. It then moves to the next element in the list and searches the list again for something smaller and again it will do a swap if it finds a smaller element later in the list. This process continues until the list is sorted from smallest to biggest by methodically moving the smallest values to the beginning of the list in sorted order.   For the selectionSortlg function I want you to modify it so it does the same thing, but it starts at the end of the list looking for the largest value and moves the largest items to the end of the list in sorted order. The sorted results from this function should be the same as the sorted results you got from selectionSortsm.  When you are done, output your unsorted list1 and then output list1 again after it has been sorted by selectionSortsm. Now do the same thing with selectionSortlg, output list2 unsorted, sort it with selectionSortlg and then output the sorted results. The number of swaps required to sort the list will vary depending on how shuffled the original list was. The number of swaps that each function makes can vary, a question to think about is why would the number of swaps be different for sorting the same list into the same order? 5. Bubble Sort - (12 points) -  Required function heading - **def bubbleSort(lst):** For this problem you will write the code for a bubble sort. Create a list of 10 random numbers from 0-100. Your function will need two variables, a swap variable to count the number of swaps and a boolean trigger to stop the sorting. You will need an indefinite loop controlled by your trigger because you don’t know how many times you will have to loop over the list. Start at the beginning of the list and compare the first item and the second item and if the first item is larger you need to swap them. When you have a swap increment swap. Then move to the next item in the list and compare it to the one after it and swap if they are out of order, continue this process all the way to the end of the loop. Keep in mind that your loop should stop one place short of the end because if you try to compare the last item with what comes after you will have an out of bounds error. Your loop will not stop until it has gone all the way through the list one time and made no swaps. At the end of the function print the number of swaps. You do not need to return the list from the function because changes that happen to the list inside of the function, happen to the actual list. Output your unsorted list, pass the list into your bubble sort and then output the sorted list. Notice that this sort method will have more swaps then the selection sort for the same sized list. If you want you can compare these two types of sorts on two copies of the same list to see the difference in swaps. 6. Merge Sort - (14 points) -  Required function heading - **def merge(lst1, lst2): return resultinglist** For this program you will create a function that takes two sorted lists as parameters and then combines them, in a sorted fashion, into a single sorted list. You may not combine the two lists and then sort the new list. You must copy each element from each list into a new list in a sorted fashion. So starting at the first element compare the lists, element by element, and add the smaller of the two to the new list. You will need to create a new empty list in the function to sort the two parameter lists into. At the end of the function return the new list that contains the sorted results of the two lists. Create two lists of 5 random numbers 0-100. Output the two unsorted lists, [sort()](https://www.w3schools.com/python/ref_list_sort.asp) each list and output the two sorted lists. Output the results of the function with the single sorted list. 7. Locker Problem - (12 points) -  A school has 100 lockers and 100 students. All lockers are closed on the first day of school. As the students enter, the first student, denoted S1, opens every locker. Then the second student, S2, begins with the second locker, denoted L2, and closes every other locker. Student S3 begins with the third locker and changes every third locker (closes it if it was open, and opens it if it was closed). Student S4 begins with locker L4 and changes every fourth locker. Student S5 starts with L5 and changes every fifth locker, and so on, until student S100 changes L100. After all the students have passed through the building and changed the lockers, which lockers are open? Write a program to find your answer. (Hint: Use a list of 100 Boolean elements, each of which indicates whether a locker is open (True) or closed (False). Initially, all lockers are closed.) The colab document has an example of how to create a boolean list with either all true or all false values. This is going to be a nested loop problem. 8. Turtle Hangman - (12 points) -  Required function heading - **def drawHangman(num):** For this final program you will make the turtle drawings for [this hangman game](https://replit.com/@MrReynolds/07TurtleHangman#main.py). Either fork the repl.it or copy the code into a file in pycharm, in pycharm you may need to add the exitonclick() command. You can watch this [video (2:54)](https://drive.google.com/file/d/1qROfPuFqTUrubh8RWHZvsf32BkDQTMh4/view) of the game being played to see how it should work. This [video (2:15)](https://drive.google.com/file/d/1yZol3LRSE4KLof3ulMXn-wtSjMrmcKHP/view) will give you a tip about how to develop the drawing part quicker without having to play the hangman game again and again. You already have a working hangman game. You need to create a function that takes a number and based on that number it draws part of the hangman. There should be at least six different drawings, head, arm1, arm2, body, leg1, leg2. After leg2 the game is over. For six drawings you will have to set the game up so that you get six wrong guesses before you lose. The game you have now does not have a lose feature so you will have to add that. In the game whenever the user is wrong you will need to call your draw function and pass the correct drawing number to it. After six wrong guesses you will need the program to exit the loop. If the user chooses to play again you will need to call your hangman function again to reset the drawing screen.   Make sure you have a comment block at the top of your program with your name, the date and a list of the programs that are being run in the program. Also make sure to comment your variables, control structures, and each problem. Also use white space between the problems.  ############################################################  # Name : Date: #  # Unit 7 Problems #  # Count Digits, Smallest index of list, List shuffle #  # Revise selection sort, bubble sort, merge sort #  # Locker Problem, Turtle hangman #  ############################################################  When your code works and is commented, turn it into the classroom. | |



Lists start at 0, just like babies!!