1. A particular cell phone plan includes 50 minutes of air time and 50 text messages for $15.00 a month. Each additional minute of air time costs $0.25, while additional text messages cost $0.15 each. All cell phone bills include an additional charge of $0.44 to support 911 call centers. The entire bill (including 911 charge) is subject to 9% sales tax.

Write a program that reads the number of minutes and text messages used in a month from the user. Display the base charge, additional minutes charge (if any), additional text message change (if any), the 911 fee, tax, and total bill amount. Only display the additional minute and text message charges if the user incurred costs in these categories. Ensure that all of the charges are displayed using 2 decimal places.

1. Most years have 365 days. However, the time required for the Earth to orbit the Sun is actually slightly more than that. As a result, an extra day, February 29, is included in some years to correct for this difference. Such years are referred to as leap years. The rules for determining whether or not a year is a leap year follow:

* Any year that is divisible by 400 is a leap year.
* Of the remaining years, any year that is divisible by 100 is NOT a leap year.
* Of the remaining years, any year that is divisible by 4 is a leap year.
* All other years are NOT leap years.

This program reads a year from the user and displays a message indicating whether or not it is a leap year.

1. One of the first known examples of encryption was used by Julius Caesar. Caesar needed to provide written instructions to his generals, but he didn't want his enemies to learn his plans if the message slipped into their hands. As a result, he developed what later became known as the Caesar Cipher.

The idea behind this cipher is simple (and as a result, it provides no protection against modern code breaking techniques). Each letter in the original message is shifted by 3 (or n) places. As a result, A becomes D, B becomes E, etc. The last three letters in the alphabet are wrapped around to the beginning: X becomes A, Y becomes B, and Z becomes C. Non-letter characters are not modified by the cipher.

Write a program that implements a Caesar cipher. Allow the user to supply the message and the shift amount, and then display the shifted message. Ensure that your program encodes both uppercase and lowercase letters. Your program should also support negative shift values so that it can be used both to encode messages and decode messages.

1. In this exercise, your will first write a function that determines whether or not a password is good. We will define a good password to be a one that is at least 8 characters long and contains at least one uppercase letter, at least one lowercase letter, and at least one number. Your function should return True if the password passed to it as its only parameter is good. Otherwise it should return False.

Then use the random password generator from Example 4-4 (copy that function and paste it here) and the function you defined above, write a program that generates a random good password and displays it. Count and display the number of attempts that were needed before a good password was generated. Call the two functions mentioned above from a function named main in the file that you create for this exercise. Ensure that your main program only runs when your solution has not been imported into another file.

1. Create a program that identifies all of the words in a string entered by the user. Begin by writing a function that takes a string of text as its only parameter. Your function should return a list of the words in the string with the punctuation marks at the edge of the words removed. Do not remove punctuation marks that appear in the middle of a words, such as the apostrophes used to form a contraction. For example, if your function is provided with the string "Examples of contractions include: don't, isn't, and wouldn't." Your function should return the list ["Examples", "of", "contractions", "include", "don't", "isn't", "and", "wouldn't"].

Write a main program that demonstrates your function. It should read a string from the user and display all of the words in the string with the punctuation marks removed. As a result, you should ensure that your main program only runs when your file has not been imported into another program.

1. Simulate 10,000 rolls of two dice. Begin by writing a function that simulates rolling a pair of six-sided dice. Your function will not take any parameters. It will return the total that was rolled on two dices as its only result.

Write a main program that uses your function to simulate rolling two six-sided dice 10,000 times. As your program runs, it should count the number of times that each total occurs. Then it should display a table that summarizes this data. Express the frequency for each total as a percentage of the total number of rolls. Your program should also display the percentage expected by probability theory for each total. Sample output is shown below. (Exe6-1)

Total Simulated Expected

Percent Percent

2 2.90 2.78

3 6.90 5.56

4 9.40 8.33

......

12 2.60 2.78

1. There is at least one word in the English language that contains each of the vowels (case insensitive) a, e, i, o, u and y exactly once and in order. Write a program that searches program-7-data.txt and displays all of the words that meet this constraint. (Exe7-6)
2. Create a square root function that takes two parameters. The first parameter, n, will be the number for which the square root is being computed. The second parameter, guess, will be the current guess for the square root. The guess parameter should have a default value of 1.0. Do not provide a default value for the first parameter. Your square root function will be recursive. The base case occurs when guess^2 is within 10^-12 of n. In this case, your function should return guess because it is close enough to the square root of n. Otherwise, your function should return the result of calling itself recursively with n as the first parameter and (guess + n/guess)/2 as the second parameter.

Write a main program that demonstrate your square root function by computing the square root of several different values. When you call your square root function from the main program you should only pass one parameter to it so that the default value for guess is used. (Exe8-2)

1. Read the stock list from the file program-9-data.txt. (The file is in the folder “Part 2: Data Collection and Preparation” on blackboard). For each stock on the list, retrieve the minute stock price data for an entire day. Create a new txt file for each stock, named after the stock symbol itself. In this file, save all minute stock price, one on each line. Keep two decimal places for stock prices.
2. Data structure is critical for the term project. Consider the following data structure for a job posted by Google. (https://www.google.com/about/careers/search?src=Online/Job+Board/indeed&utm\_source=indeed&utm\_medium=jobaggr&utm\_campaign=freeaggr#!t=jo&jid=149335001)

Consider using a list to capture the key information we are interested in. Here is the data structure I come up with.

[‘business intelligent analyst, ‘Google’, ‘Mountain View, CA’, ‘NA’, ‘full time’, ‘BS/BA’, ‘4’, [‘business intelligence’, ‘data reporting’, ‘database query’, ‘java’, ‘python’, ‘SQL’, ‘Data Warehousing’, ‘Visualization’, ‘statistical and quantitative modeling’, ‘analytical problem solving skills’, ‘business judgment’, ‘presentation skills’]]

Please work with all your teammates and collect information on at least twenty jobs. Convert job information using the above data structure.

Note that same or similar qualification/skill may be phrased differently. Try to categorize those skills and qualifications into a few skillsets. For example, here Google calls it ‘business judgment’. Some other companies may call it ‘business acumen’ or ‘business sense’. They may be included in the same category or skillset. For another example, ‘statistical and quantitative modeling’ and ‘analytical problem solving skills” are also similar.

Although you share the data with your teammates, you shall independently write a program to count the relative frequency of each skillset. Hint: you may need to create a separate python dictionary for each skillset.

1. In the file “program-11-AAPL-NFLX.txt”, there are minute-level stock prices of AAPL and NFLX, each occupying one line. Please plot a line chart with time as your x-axis and stock price being your y-axis. The line chart must show the stock prices of both stocks. Please use different color and line styles for different stocks. Please label each of the two stocks: AAPL or NFLX and provide a legend for your figure. In the end, your x-axis should be labeled as “Time in Minutes” and your y-axis should be labeled as “Stock Price”.
2. In the file “program-11-AAPL-NFLX.txt”, there are minute-level stock prices of AAPL and NFLX, each occupying one line. Please plot a scatter with AAPL’s price on your x-axis and NFLX’s price on your y-axis. Your x-axis should be labeled as “AAPL” and your y-axis should be labeled as “NFLX”.
3. The file “program-12.txt” contains 10,000 data points, one on each line. Please plot a figure containing two normalized histograms based on the data in this file with number of bins = 50 and 100, respectively. Please label the x-axis as “Score” and y-axis as “Probability”. Display the figure title as “Test Score Distribution”. Please make sure you display the number of bins associated with each of the two histograms.
4. In class, we discussed how to use gradient descent algorithm to come up with the best values of and to fit a simple linear regression model, which has only one input variable (feature). In this assignment, you will be asked to develop a gradient descent algorithm that can handle multiple linear regression models, which have more than one features. Suppose this MLR model has k features. Then your gradient descent algorithm should be able to find the best values for , , …, .

To that end, your script needs to define three functions: one main function; one function that calculates the value for the cost function; and one function that computes the values for gradient descent.

Your code will read data from a csv. Each row of this file is one observation or one data point. Suppose this csv file has k+1 columns. The first k columns carry the values for x0, x1, …, xk. And the last column carries the value for y. (Be careful, this format is different from the one used in class)

The csv file provided for this assignment has 4 columns. These data are for estimating the price of a used mini-cooper. The first column is the mileage of mini-cooper in 1000 miles; the second column is the model year; the third column is a binary variable, indicating whether this mini-cooper has T-top (yes = 1; no = 0); and the fourth and the last column is the actual price of mini-cooper (y in $). Based on this data set, we’d like to develop a multiple linear regression model, with x1 being mileage, x2 being model year, and x3 being T-top, to predict the price of a used mini-cooper. The csv file has 13 rows, indicating that we’ve collected historical data on 13 mini-coopers. That is, the sample size or the size of the training set is 13.

Make sure that your code will be able to handle general MLR models, not just the one described above. That is, your program should be able to generate correct values for all ’s if the program is fed with another csv file of same format, but with different number of rows and columns.

Please make sure your program will print out result of each iteration. Please print result of each iteration in the same row, including iteration number, all values of this iteration, value of the cost function of this iteration, the different between the cost function value of this iteration and previous iteration.

1. **Bootstrap Resampling for Difference between Two Means**

Imagine we have given some people a placebo and others a drug. The measured improvement (the more positive the better) is:

Placebo: 54 51 58 44 55 52 42 47 58 46 (10 in total)

Drug: 54 73 53 70 73 68 52 65 65 (9 in total)

As you can see, the drug seems more effective on the average (the average measured improvement is nearly 63.7 (63 2/3 to be precise) for the drug and 50.7 for the placebo). But, is this difference in the average real? Formula-based statistics would use a *t*-test which entails certain assumptions about normality and variance; however, we are going to look at just the samples themselves and *shuffle* the labels. The meaning of this can be illustrated in the following table—in which we put all the people— labeling one column ‘Value’ and the other ‘Label’ (P stands for placebo, D for drug).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 54 | 51 | 58 | 44 | 55 | 52 | 42 | 47 | 58 | 46 | 54 | 73 | 53 | 70 | 73 | 68 | 52 | 65 | 65 |
| Label | P | P | P | P | P | P | P | P | P | P | D | D | D | D | D | D | D | D | D |

Shuffling the labels means that we will take the P’s and D’s and randomly distribute them among the patients. (Technically, we do a uniform random permutation of the label column.) Note that after shuffling, you will still have 10 P’s and 9 D’s. This might give:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 54 | 51 | 58 | 44 | 55 | 52 | 42 | 47 | 58 | 46 | 54 | 73 | 53 | 70 | 73 | 68 | 52 | 65 | 65 |
| Label | P | P | D | P | P | D | D | D | D | D | P | P | P | D | P | P | D | P | D |

We can then look at the difference in the average P value vs. the average D value. We get an average of 59.0 for P and 54.4 for D. We repeat this shuffle-then-measure procedure 10,000 times and ask what fraction of time we get a difference between drug and placebo greater than or equal to the measured difference of 63.67 - 50.7 = 12.97.

Write a program based on the numbers provided above. In your program, create a function that randomly shuffles the labels. Then in your main function, repeating the random shuffling 10,000 times. For each shuffle, calculate the difference between the average P value and the average D value. You will then have 10,000 values of difference. Plot a histogram based on these 10,000 values of difference. Then compute the probability that the difference is greater than or equal to 13. In the end, display the following message:

“If the drug is as effective as the placebo, then the probability that the different between the average P value and the average D value is greater than or equal to 12.97 is (fill in your probability here). Thus, we conclude that the drug is (choose ‘effective’ or ‘ineffective’) given a significance level of 1%.”

Pseudocode:

1. Measure the difference between the two group means. The difference in means is measured by (sum(grpA) / len(grpA)) - (sum(grpB) / len(grpB)). In this example the difference between the two group means is 12.97.

2. Set a counter to 0, this will count the number of times we get a difference between the means greater than or equal to 12.97.

3. Do the following 10,000 times:

a. Shuffle the original measurements. To do this:

i. put the values from all the groups into one array but remembering the start and end indexes of each group

ii. shuffle the values in the array, effectively reassigning the values to different groups

b. Measure the difference between the two group means, just as we did in step (1).

c. If the difference from step (3b) is greater than or equal to 12.97, increment our counter from step (2). Note: if our original difference between the means were a negative value we would check for values less than or equal to that value.

4. counter / 10,000 equals the probability of getting our observed difference of two means greater than or equal to 12.97, if there is in fact no significant difference.

1. **Problem 2: Bootstrap Resampling – from Sample to Sampling Distribution**

A restaurant keeps a record of number of reservations it has received over the last 20 weekends. They are:

48, 24, 51, 12, 21, 41, 25, 23, 32, 61, 19, 24, 29, 21, 23, 13, 32, 18, 42, 18

The restaurant has hired you as a consultant to help determine the sampling distribution of the number of reservations over weekend. It is very useful because it helps answer the questions such as: what is the probability that the restaurant will receive at least 40 reservations over the coming weekend.

You decide to use bootstrap resampling to generate the sampling distribution. To that end, you would like to write a Python program. In this program, please define a function called resampling. This function will take a list of numbers and return another list of numbers of equal length from random sampling with replacement.

In your main function, repeat the resampling process 10,000 times. For each iteration, compute and store its mean, which is called a sample mean. After the completion of 10,000 iterations, compute and display the mean and standard deviation of these 10,000 sample means. Then your program will allow the end user to enter a positive integer. You program will be able to compute and display the probability that the number of reservations will be more than the number entered by the end user.

Here is an explanation how random resampling with replacement works.

Consider an original list of 3 numbers: list = [1, 3, 4]. The random resampling process works like this: randomly pick one number at a time with equal probability (in this case, 1/3 probability for each number); repeat the first step until we have len(list) numbers (in this case, 3 numbers) in our new list. Suppose the first number randomly selected from the list is 3. Then we randomly select another number for [1, 3, 4] with equal probability again. Even though 3 was selected in the first round, it seems like another 3 replaces it in the original list when we select again. This is why it is called sampling WITH REPLACEMENT. Suppose the second number selected is 4. And we will randomly select one more number from the same list [1, 3, 4] with equal probability. Let’s say, we have a 3 again. In the end, the first sample we randomly generated from [1, 3, 4] is [3, 4, 3]. Note that the new list and the old list are of equal length, each having 3 numbers. We can repeat this resampling process many times.