The purpose of this guide is to present you with a series of problems for each unit that increase in difficulty as you go through them. Attempt the problems in order. If you cannot solve a problem, comment out the code and seek help and advice to solve it.

This guide is not guaranteed to cover every possible problem or feature that you will encounter on a practicum. In addition, the problems you will be given on your exam will not be exactly the same, and probably will not be slight variations of the problems herein.

Please note that whenever you are asked to ***print*** something, it refers to displaying a message in standard output (the terminal). Whenever you are asked to ***return*** something from a function, you should use a `return` statement to return the correct value.

There often is no one right answer to each question, but using advanced features of the Python language that have not been covered in class may solve the problem, but will not likely earn you credit on the exam. You should make an effort to use only those features that have been covered in class and homework assignments.

There are many resources available to you:

- Your instructor's office hours. See MyCourses for a schedule.
- Virtual Mentoring hours on the course Discord server. See the #waiting-room channel for the pinned schedule.
- The GCCIS Tutoring Center
- The SSE Mentoring Hours
- The WiC Tutors

1. Define a new function called "`make_array`" that declares parameters for a `prototype` and `length`. Return the array created using the parameters. Test your function by calling it from `main` and printing the array that is returned.

2. Define a new function called "`string_to_array`" that declares a parameter for `a_string` and uses a loop to return an array containing all of the individual characters in the string. For example, calling the function with the string `"abcd"` should return the array `["a", "b", "c", "d"]`. Test your function by calling it from `main` and printing the array.

3. Define a function named "`threes_and_fives`" that declares a parameter for `an_array` and returns a copy of the array containing only the values that are divisible by 3 and/or 5. For example, calling the function with an array containing the integers from 0 to 25 it will return the array [0, 3, 5, 6, 9, 10, 12, 15, 18, 20, 21, 24, 25].

4. Define a recursive function that implements the following mathematical formula:

   $F(1) = 1$

*F(N) = N + F(N+1) if N is odd*
*F(N) = N + F(N/2) if N is even*
*F(N) where N < 1 is undefined*

Test your function by calling it from `main` with at least the following values for `n`: 0 (`None`), 1 (1), 10 (31), 15 (46), 100 (250), and 101 (381).

5. A linear search searches an array index by index. If the target value is found, the index is returned. Otherwise the function returns `None`. Define a function named "`linear_search_rec`" that declares parameters for `an_array` and the target value. Your function should implement a linear search using *recursion*. You may add additional parameters as needed. Test your function from `main` by creating a random array with 20 values and searching for all values from 0 to 20.

6. Define a function named "`shuffle`" that declares a parameter for `an_array`. Implement the Fisher and Yates shuffling algorithm. Test your function from `main` by setting the random seed to 1 and then calling `shuffle` with an array containing the integers from 1 to 10. After shuffling the array should be [7, 9, 10, 8, 6, 4, 1, 5, 2, 3].

7. Define a function named "`random_search`" that declares a parameter for `an_array` and a target value. Search for the target by checking the values at random indices until you find the target (return the index) or run out of indexes to check (return `None`). You should not check the same index twice (hint: use your shuffle function from the last question to create a randomized array of valid indexes). Test your function from `main` by creating a random array with 20 values and searching for all values from 0 to 20. What is the time complexity of this function?

8. SwapSort is an algorithm that works as follows:
    a. Iterate over every index i such that `1 ≤ i < length`. If the value at index `i` is less than the value at index `i-1`, swap them.
    b. Repeat the first step until a complete iteration through the array results in no swaps.

    Create a new Python module in a file named "`study3.py`" and define a function named "`swap_sort`" that declares a parameter for `an_array`. Implement the SwapSort function as described. Test your function by calling it from `main` with a random array.

9. Define a function named "`make_tuple`" that declares three parameters `a`, `b`, and `c` and returns a tuple containing all three parameters. Define a `main` function and use it to call your `make_tuple` function at least twice with different arguments and print the tuples that are returned.

10. Define a function named "`reverse_tuple`" that declares a parameter for a sequence and returns a tuple containing the values in the sequence in reverse order. For example,

if you call the function with the list `[1, 2, 3, 4]` it will return the tuple `(4, 3, 2, 1)`. Your function should work with any kind of sequence (strings, ranges, lists, tuples, etc) with any number of elements. Call `reverse_tuple` from `main` with at least two different kinds of sequences and print the tuple that is returned.

11. A Magic: The Gathering trading card has a name, mana value, and power/toughness values. Define a function named "`make_trading_card`" that declares a parameter for a `name, mana_value, power,` and `toughness`. Your function should return a tuple containing that information. Test your function in main with at least the following data.
    a. Borborygmos is a creature with power 6 and toughness 7. It's mana value is "3RRGG"
    b. Shivan Dragon is a creature with power and toughness 5. It's mana value is "4RR"
    c. Create at least 3 other Magic cards on your own. You can find other examples of creature cards on www.scryfall.com.

12. Define a function named "`mana_key`" that declares a parameter for a `trading_card` tuple. The function should return an integer value that guarantees that a list of cards will be sorted in order of total mana value, least to greatest. Mana value for Magic cards works as follows:
    a. Any number in a mana value would add that number to the total value You only need to worry about the integers 1-9.
    b. Any letter representing a colored mana represents one mana each, regardless of which letter it is. Ie. Borborygmos with mana value "3RRGG" would have a total mana value of 7. (3 + (R + R) + (G + G))

    Be sure to test your function in `main` with a list of the card tuples you wrote in the previous example.

13. Define key functions for all other attributes of a trading card tuple besides name, and be sure to test them in `main`.

14. Define a function named "`make_list`" that declares parameters `a`, `b`, and `c` and returns a list containing all three parameters. Call your `make_list` function from `main` at least twice with different arguments and print the lists that are returned.

15. Define a function named "`nth_list`" that declares parameters for a sequence and an integer n. Return a list that has every n$^{th}$ value from the sequence. For example, if you call the function with `sequence=range(1, 11)` and `n=2` it will return the list with the values `[2, 4, 6, 8, 10]`. Your function should work with any kind of sequence (strings, ranges, lists, tuples, etc.). Call your function from `main` with at least two different sequences and print the lists that are returned.

16. Define a function named "`splice`" that declares parameters for lists `a` and `b`. The function should permanently add all of the elements from list `b` to list `a`. ***Do not*** return a value. For example, if you call the function with `a=[1, 2]` and `b=['a', 'b']` then `a` will be modified to be `[1, 2, 'a', 'b']`. List `a` should be persistently modified by the function. Test your function by calling it from `main`. Print list `a` before and after the function is called.

17. Define a function named "`scramble`" that declares a parameter for `a_string`. Using slicing, your function should append all characters in the string into a list, then using slicing, print the word out backwards. Be sure your function is able to handle if the word starts or ends with a space.

18. Test your previous function in main with a sentence of your choice. Then use the result that was returned by the previous function as an argument. Did you get the same string you started with?

19. Define a function named "`odds_before_evens`" that declares a parameter for `a_list` of integers. Rearrange the elements in the list so that all of the odd values come before the even values. Otherwise, the values should remain in the same relative order. ***Do not*** return a value - modify the list in place. For example, if you call the function with the list `[3, 4, 1, 2, 5]` the values will be rearranged into the configuration `[3, 1, 5, 4, 2]`. Test your function by calling it from `main`.

20. Define a function named "`bisect`" that declares a parameter for `a_list` and returns the list bisected into two halves. For example, if you call the function with `[3, 4, 1, 2]` you will return both halves of the list: `[3, 4]` and `[1, 2]`. If the list has an odd number of elements, *do not* include the middle element in either half. For example, if you call the function with `[1, 2, 3, 4, 5]` you would return the halves `[1, 2]` and `[4, 5]`. Test your function by calling it from `main` and printing the halves that are returned.

21. Define a function named "`equivalent`" that declares parameters for sequences `a` and `b`. Return `True` if the sequences contain all of the same values, even if they are in a different order, and `False` otherwise. For example, if you called the function with `a=(2, 1, 3)` and `b=range(1, 4)` the function would return `True`. because all of the elements in `a` are also in `b` and vice versa. A naive solution runs in O($n^2$) time. Can you implement a faster version? Can you implement an O(n) version? Test your function by calling from `main` with several different kinds of sequences with the same and different elements.

22. Define a function named "`list_range`" that declares parameters for `m` and `n` and uses list comprehension to return the list containing all of the values from `m` to `n-1`. In other words, calling your function with `m=5`, and `n=10` would return the list `[5, 6, 7, 8, 9]`. Test your function by calling it from `main`.

23. Define a function named "`fizz_buzz_list`" that declares no parameters. Using list comprehension, your function should create a list from all integer values 1-100 that are divisible by 3, 5, or both.

24. Define a function named "`multiples`" that declares a parameter for a `sequence` and an integer `n`. Use list comprehension to create a list that has all of the values from the sequence that are evenly divisible by n. For example, if called with `sequence=range(10)` and `n=4` it should return `[0, 4, 8]`. Test your function by calling it from `main`.

25. Define a function named "`only_vowels`" that declares a parameter for `a_string`. Use list comprehension to return a list containing only the vowels in the string. Your function should run in O(n) where n is the length of the string. For example, calling the function with the string "`Humpty Dumpty sat on a wall eating his curds and whey`" should return the `['u', 'u', 'a', 'o', 'a', 'a', 'e', 'a', 'i', 'i', 'u', 'a', 'e']`. Test your function by calling it from `main`.

<u>For the next set of questions, refer to the problem statement below:</u>

Congratulations! You've just been hired for your first co-op at a company called Jaeger Lumber! You've been tasked with working with the company database to create various reports for salespeople and upper management (It's a real job, I promise). The file `jaeger.csv` acts as the database for this problem.

26. Define a function named "`jaeger_analysis`" and use it to answer the following questions:
    a. Who is the highest grossing salesperson? Who is the lowest?
    b. Which customer has purchased the highest **quantity** of items?
    c. What month of the year is the busiest in terms of total sales?
    d. Which product group has the highest number of transactions associated with it?

27. Define a function named "`average_price`" that declares a parameter for a `product_name`. Using the Jaeger "database", find the average cost per unit of the specified product. For this problem, you do not need to worry about sales tax or any other extraneous costs involving sales transactions. Note that salespeople often negotiate on prices for wholesale customers, so items will often be bought and sold at different prices depending on the customer.

28. Define a function named "`employee_scores`". Phil, the head salesperson, loves to fire people. This function should analyze all of the sales data of a given salesperson and

help determine whether or not Phil has cause to fire them. Your function should return every salesperson and a score, determined by the average number of sales they have per year. Return a data structure containing each person, their score based on the above formula, and whether or not Phil should fire them based on if the score is below 7 sales/year. (Phil should have 7.65 sales/year).