Problem Set (Python)

Complete as much as you want, when you want. No rules on what you can lookup/google, though I do recommend spending more time googling how to do a specific operation you want then the problem itself, particularly for easier problems.

Problems here are specified as function headers which take input and with an expected output, though some problems may involve multiple functions if there's class design or similar involved.

Generally problems are intended to loosely get harder for later numbers, though obviously it depends on your comfort level with various concepts. I generally make the last problem or two quite a bit harder for fun though...

Problems

max_even_table

```
def max_even_table(lst : list[list[int]]) -> list[int | None]:
  """
  Returns the maximum even number in each row of `lst`, or `None` if none exists
  Note that negatives will be negative when the associated positive is

  Examples:
    max_even_table([[1, 2, 3]]) -> [2]
    max_even([[-4, 3, -1, -6], [1, 3, 5]) -> [-4, None]
    max_even_table([[], [1, 4, 6, 2], [-1, -2]]) -> [None, 6, -2]
  """
```

expansion

```
def expansion(s : str) -> list[str]:
  """
  Returns the list of each letter of `s` expanded in sequence

  Examples:
    expansion('test') -> ['t', 'te', 'tes', 'test']
    expansion('') -> []
    expansion('abccba') -> ['a', 'ab', 'abc', 'abcc', 'abccb', 'abccba']
  """
```

row_sum

```
def row_sum(table : list[list[int]]) -> list[list[int]]:
  """
  Returns the sum up to each element of the given table

  Examples:
    row_sum([[1, 2, 3], [4, 5, 6]]) -> [[1, 3, 6], [4, 9, 15]]
    row_sum([[], [4], [], [5]]) -> [[], [4], [], [5]]
    row_sum([[-1, -5, 6], [7, 2], [10, 10, 10]]) -> [[-1, -6, 0], [7, 9], [10, 20,
30]]
  """
```

max_prime

```
def max_prime(table : list[list[int]]) -> int:
  """
  Returns the maximum prime in the given table, or -1 if none exists
  negative numbers, 0, and 1 are not prime numbers

  Examples:
    max_prime([[1, 4, 8], [0, -5]) -> -1
    max_prime([[3, 7, 1, 4], [8, 4, 12, 221]]) -> 7
    max_prime([[4, 5, 6], [1, 2, 3], [17, 9, 6]]) -> 17
    max_prime([[]]) -> -1
  """
```

staggered_sum

```
def staggered_sum(lst : list[int | list[int]]) -> int:
  """
  Returns the sum of elements in `lst`, which is a list of integers _or_ lists

  Examples:
    staggered_sum([1, 2, 3]) -> 6
    staggered_sum([[], [], []]) -> 0
    staggered_sum([[1, 2], 3, [4, 5]]) -> 15
    staggered_sum([-4, [-3, 2, 1], 6, 8, [-1]]) -> 9
  """
```

double

```
def double(table : list[list[int]]):
  """
  Returns nothing
  Modifies table in place to double each number
```

```
  Examples:
    double([[1, 2], [3, 4, 5]]) changes lst to [[2, 4], [6, 8, 10]]
    double([[-1], [-2], [0, 0]]) changes lst to [[-1], [-2], [0, 0]]
    double([[]]) does "nothing"
  """
```

column_sum

```
def column_sum(mat : list[list[int]]) -> list[int]:
  """
  Returns the sum of each column of the _rectangular_ matrix `mat`
  (rectangular means each row is the same length)

  Examples:
    column_sum([[1, 2, 3], [4, 5, 6]]) -> [5, 7, 9] # yes, I did it right this
time
    column_sum([[], []]) -> [0]
    column_sum([[2, 2], [-1, -2], [3, 9]]) -> [4, 9]
  """
```

Count letters

Required reading for the next two problems:
https://docs.python.org/3/tutorial/datastructures.html#dictionaries. Lmk if you have questions

```
def count_letters(s : str) -> dict[str, int]:
  """
  Returns the count of each letter in `s` as a dictionary

  Examples (dictionary print order doesn't matter):
    count_letters("") -> {}
    count_letters("abc") -> {'a' : 1, 'b' : 1, 'c' : 1}
    count_letters("aaaa") -> {'a' : 4}
    count_letters("ababbcca) -> {'a' : 3, 'b' : 3, 'c' : 2}
  """
```

Ordered string

Pretty tricky problem, all considering

```
def ordered_string(d : dict[str, int]) -> str:
  """
  Returns the alphabetically-ordered string from a dictionary of letters and
counts
  Note that each key of the dictionary must be a single lowercase letter
```

```
  Examples:
    ordered_string({}) -> ""
    ordered_string({'a' : 3, 'b' : 2}) -> "aaabb"
    ordered_string({'d' : 1, 'a' : 2', 'c' : 3}) -> "aacccd"
    ordered_string({'z' : 5, 'd' : 2, 'a' : 3, 'x' : 2} -> "aaaddxxzzzzz"
  """
```

matrix_multiply

Super famously hard problem (this was my "challenge problem" for my course) Do not try to do LU-decomposition unless you're feeling very brave XD

```
def matrix_multiply(mat1 : list[list[int]], mat2 : list[list[int]]) ->
list[list[int]]:
  """
  Returns the result of multiplying mat1 by mat2
  The dimensions of the input matrices must be valid
  https://en.wikipedia.org/wiki/Matrix_multiplication

  Examples:
    matrix_multiply([[1, 2], [3, 4]], [[1, 2], [3, 4]]) -> [[7, 10], [15, 22]]
    matrix_multiply([[1, 2]], [[5], [6]]) -> [[17]]
    matrix_multiply([[1], [2]], [[5, 6]]) -> [[5, 10], [6, 12]]
    https://matrix.reshish.com/multiplication.php <-- working reference
  """
```