- This lab will review basic python concepts with an emphasis on memory map images and list comprehensions.
- It is assumed that you have reviewed chapters 1 and 2 of the textbook. You may want to refer to the text and your lecture notes during lab as you solve the problems.
- When approaching the problems, think before you code. Doing so is good practice and can help you lay out possible solutions.
- Think of any possible test cases that can potentially cause your solution to fail!
- You must stay for the duration of the lab. If you finish early, you may help other students. If you don't finish by the end of the lab, we recommend you complete it on your own time.
- Your TAs are available to answer questions in lab, during office hours, and on Piazza.

Vitamins (maximum 1 hour)

---

1. For each section below, write the correct output shown after the Python code is run. Explain your answer by **drawing the memory image** for the execution of these lines of code. That is, you should draw the variables as they are organized in the call stack, and the data they each point to.

a.
```
lst = [1, 2, 3]
lst.append(4)

print(lst)
```
____

b.
```
s = "aBc"
s.upper()

print(s)
```
____

c.
```
s = "aBc"
s = s.upper()

print(s)
```
____

d.

```
lst = [1, 2, 3]
def func(lst):
     lst.append(4)
     print("Inside func lst =", lst)

func(lst)
```

____

```
print(lst)
```

____

e.

```
s = "abc"
def func(s):
     s = s.upper()
     print("Inside func s =", s)

func(s)
```

____

```
print(s)
```

____

f.

```
import copy
lst = [1, 2, [3, 4]]
lst_copy = copy.copy(lst)
lst_copy[0] = 10
lst_copy[2][0] = 30


print(lst)
```

____

```
print(lst_copy)
```

____

g.

```
import copy
lst = [1, [2, "abc"], [3, [4]], 7]
lst_deepcopy = copy.deepcopy(lst)
lst[0] = 10
lst[1][1] = "ABC"
lst_deepcopy[2][1][0] = 40

print(lst)
```

____

```
print(lst_deepcopy)
```

____

h.

```
lst = [1, [2, 3], ["a", "b"] ]
lst_slice = lst[:]
lst_assign = lst
lst.append("c")
for i in range(1, 3):
     lst_slice[i][0] *= 2

print(lst)
```

____

```
print(lst_slice)
```

____

```
print(lst_assign)
```

____

2. Use list comprehension to generate the following lists:

a.
```
[1, 2, 4, 8, 16, 32, 64, 128]
```

____

b.
```
[128, 64, 32, 16, 8, 4, 2, 1]
```

____

c.
```
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

____

3. What will the following list comprehensions output?

a.
```python
print([i for i in range(15) if i % 3 == 0])
```

____

b.
```python
print([char for char in "New York City" if char not in "aeiou"])
```

____

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code. This will be good practice for when you write code by hand on the exams.

1. Implement the following function:

```
def add_binary(bin_num1, bin_num2):
    """
    bin_num1 - type: str
    bin_num2 - type: str
    return value - type: str
    """
```

This function is given bin_num1 and bin_num2 which are two binary numbers represented as strings. When called, it should return their sum (also represented as a binary string). Do not use any python bit manipulation functions such as `bin( )`.

For example, calling `add_binary("11", "1")` should return `"100"`.

2. Implement the following function:

```
def can_construct(word , letters):
    """
    word - type: str
    letters - type: str
    return value - type: bool
    """
```

This function is passed in a string containing a word, and another string containing letters in your hand. When called, it will return True if the word can be constructed with the letters provided; otherwise, it will return False.

Notes:

- Each letter provided can only be used one.
- You may assume that the word and letters will only contain lower-case letters.

For example, `can_construct("apples", "aples")` will return `False`,
but `can_construct("apples", "aplespl")` will return `True`.

3.

a. Implement a function:

```
def create_permutation(n)
```

This function is given a positive integer n, and returns a list containing a random permutation of the numbers:
0, 1, 2, … , (n-1).

For example, one call to `create_permutation(6)` could return the list: [3, 2, 5, 4, 0, 1].
Another call to `create_permutation(6)` could return the list: [2, 0, 3, 1, 5, 4].

Implementation requirement:
You may only use the randint function from the random module. Specifically, you are not allowed to use the shuffle function.

b. Implement a function:

```
def scramble_word(word)
```

This function is given a string word, and returns a scrambled version of word, that is new string containing a random reordering of the letters of word.

For example, one call to `scramble_word("pokemon")` could return "okonmpe".
Another call to `scramble_word("pokemon")` could return "mpeoonk".

Implementation requirement:
To determine the new order of the letters, call the function `create_permutation`.
For example, for the word "pokemon", the scrambled word implied by the permutation [1, 4, 5, 2, 3, 0, 6] is "omokepn" (since, the first letter is the letter from index 1, the second letter is the letter from index 4, the third letter is the letter from index 5, and so on).

c. Write a guessing game that takes a word, scrambles it, prints the letters to the user, and allows them three chances to find the unscrambled word.

Have your program interact with the user as demonstrated below:

```
Unscramble the word:   o m o k e p n
Try #1: openkom
Wrong!
Try #2: pokemon
Yay, you got it!
```

Notes:
You should use the functions you implemented in the previous sections.

When printing the letters of the scrambled word, include a space between every two letters.