# Multiple Choices Questions

## Question 1.1 | Boolean Algebra

In Boolean algebra, all the variables have two possible values. What are they?

- ☐ A: True and False
- ☐ B: Yes and No
- ☐ C: High and Low
- ☐ D: Positive and Negative

## Question 1.2 | Boolean Algebra

What are the basic operations in Boolean algebra?

- ☐ A: Addition, Subtraction, Multiplication
- ☐ B: AND, OR, NOT
- ☐ C: Greater than, Less than, Equal to
- ☐ D: Union, Intersection, Complement

## Question 1.3 | Cache

Consider the following Python code:

```
nums = “123456789”
idx = [8, 4, 1, 5, 0]
s = 0

for i in idx:
   s += int(nums[i])
```

During the execution, when i = 1 happens, which character from nums will be loaded into the cache?

- ☐ A: 7
- ☐ B: 8
- ☐ C: 4

Name:_____ NYU-ID (NetID): _____

☐ D: 2

## Question 1.4 | Spatial Locality

Consider the following Python code:

```
nums = “123456789”
idx = [8, 4, 1, 5, 0]
s = 0

for i in idx:
   s += int(nums[i])
```

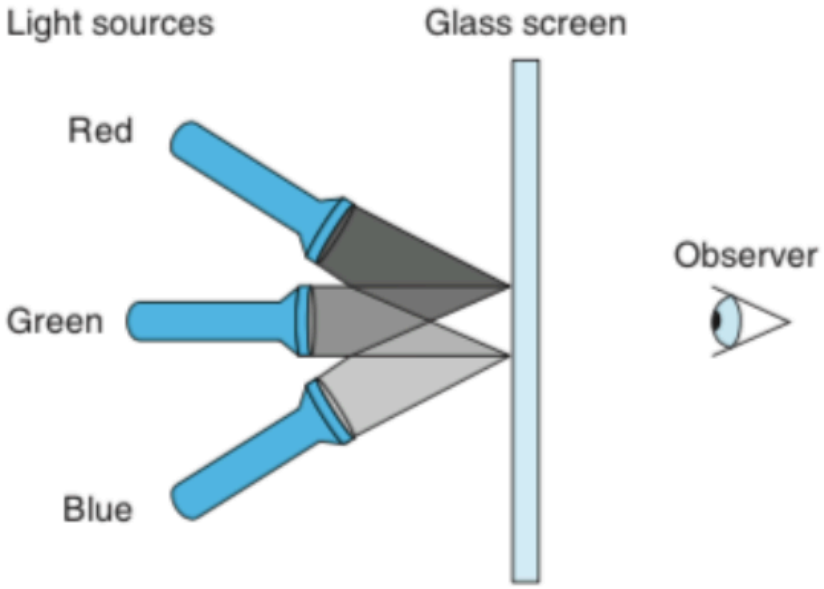
Does the code make good use of the spatial locality?

☐ A: Yes, the code makes good use of spatial locality.

☐ B: No, the code does not make good use of spatial locality.

## Question 1.5 | Color Values

Computers generate color on a video or liquid crystal display by mixing three different colors of lights: red, green, and blue. Imagine a simple scheme with three different lights, each of which can be turned on or off, projecting onto a glass screen:



We can then create eight colors based on the absence(0) or presence(1) of light sources R, G, and B. A color in the table can be a mixture of some other colors.

| R | G | B | Color |
|---|---|---|---|
| 0 | 0 | 0 | Black |
| 0 | 0 | 1 | Blue |
| 0 | 1 | 0 | Green |
| 0 | 1 | 1 | Cyan |
| 1 | 0 | 0 | Red |
| 1 | 0 | 1 | Magenta |
| 1 | 1 | 0 | Yellow |
| 1 | 1 | 1 | White |

What are the colors if we apply Boolean operations on the following color signals:

(Red AND Yellow) XOR Magenta

- ☐ `A: Blue`
- ☐ `B: Black`
- ☐ `C: Cyan`
- ☐ `D: Green`

## Question 1.6 | Boolean Expressions

Assume variables a, b, and c are declared and initialized with integer values. The following Python expression is logically equivalent to which of the provided options? (Hint: Apply De Morgan's Laws)

not (a<b or b>c)

- ☐ `A: a < b and b >= c`

- ☐ B: `a >= b or b <= c`
- ☐ C: `a >= b and b <= c`
- ☐ D: `a > b or b <= c`

## Question 1.7 | Debugging Times 5

Assume the below program is designed to multiply the input number by 5.

```
def multiplied_by_5():
    x = (input("Please give a number: "))
    return x*5.0

##calling the function in the console
>>> multiplied_by_5()
>>> Please give a number: 5
```

Does the program compute the solution correctly?

- ☐ A: `Yes, the program works as expected.`
- ☐ B: `No, the program does not work as expected.`

## Question 1.8 | Debugging Output

Consider the following Python code:

```
def fun2(y):
    y[0] += 2

y = [0]
fun2(y)
print(y)
```

What is the output of this code?

- ☐ A: `[2]`
- ☐ B: `[0]`
- ☐ C: `[0, 2]`
- ☐ D: `The program can not be executed.`

# Programming Questions

## Question 1 | Counting

You are tasked with writing two Python functions: count_letters and count_letter_pairs. Both functions analyze a given string and return a dictionary that maps characters or character sequences to their respective frequencies.

1. **count_letters:** This function takes a single string as input, where the string contains no spaces or punctuation. The function returns a dictionary where each key is a letter from the input string, and the corresponding value represents the number of times that letter appears in the string. For instance, calling count_letters("banana") should return the following dictionary:

   ```
   {'b': 1, 'a': 3, 'n': 2}
   ```

2. **count_letter_pairs:** This function takes two input arguments: a string and an integer representing the length of letter pairs (consecutive letter sequences). It returns a dictionary where the keys are the letter pairs of the specified length from the input string, and the values represent the number of occurrences of each pair. For example, calling *count_letter_pairs("banana", 2)* should return:

   ```
   {'ba': 1, 'an': 2, 'na': 2}
   ```

Both functions provide a method for analyzing letter frequency as individual characters or consecutive letter pairs within a string.

**Example Console Output**

```
>>> result = count_letters("banana")
>>> print(result)
{'b': 1, 'a': 3, 'n': 2}

>>> result = count_letter_pairs("banana", 2)
>>> print(result)
{'ba': 1, 'an': 2, 'na': 2}
```

**Write your code on the next page!**

```
def count_letters(s):
      # Please write your code here
```

```
def count_letter_pairs(s, pair_len):
      # Please write your code here
```