Quiz 2 Solutions

Name: _____

Lab: _____

Directions

Congrats on completing Week 2!

- **This quiz contains 11 questions and will last 90 minutes.** There are also two optional bonus problems at the end you can try if you finish all the other questions early.
- You may use any method to solve a problem unless it clearly asks for a recursive solution.
- The questions are not necessarily in order of difficulty.
- Please write your full name and lab number on the first page of the quiz.
- When writing code, make sure it is indented clearly and correctly.
- **When a question asks for the printed output, write only what the program prints — nothing else.** (Do not write words like `print` or `=>`.)
- For questions with *circular bubbles*, you should select *exactly one* choice.
  - ○ You must choose either this option
  - ○ Or this one, but not both!

1. [nested loops and printed output] What is the printed output of the following code?

```
for i in range(1, 4):
    row = ""
    for j in range(i):
        row = row + "*"
    print(row)
```

```
*
**
***
```

2. [dictionaries] Answer the following questions.

a) Write code to create a dictionary named `library` with the following book names and book counts:

Book name	Book count
"Sememen"	3
"Oromay"	2

```
library = {"Sememen": 3, "Oromay": 2}
```

b) Write code to add the book "Efta" with count 1 to `library`.

```
library["Efta"] = 1
```

c) Write code to decrease the count of book "Oromay" by 1.

```
library["Oromay"] = library["Oromay"] - 1
```

d) Write code that prints the total number of **books** in `library` (add up all the values in the dictionary).

```
total = 0
for title in library:
    total = total + library[title]
print(total)
```

e) Write a function `is_available(library, title)` that returns `True` if the count for `title` in `library` is greater than `0`, and `False` otherwise. You can assume that `title` is always a key in `library`.

```
def is_available(library, title):
    return library[title] > 0
```

3. [list aliasing and mutation] Answer the following questions.

a) After the following code runs, what is the final value of `saved`?

```
cart = ["milk", "bread", "eggs"]
saved = cart
cart.append("butter")
```

`saved` → `["milk", "bread", "eggs", "butter"]`

b) After the following code runs, what is the final value of `data`?

```
data = [[1, 2], [3, 4], [5, 6]]
row = data[1]
row[0] = 99
data[2] = data[0]
data[0][1] = 88
```

`data` → `[[1, 88], [99, 4], [1, 88]]`

4. [functions and loops] Answer the following questions.

a) Write a function `remove_stars(word)` that takes a single string `word` and returns a new string with all `*` characters removed.

Examples:

- `remove_stars("a*dd*is")` → `"addis"`
- `remove_stars("**hi**")` → `"hi"`

```
def remove_stars(word):
    result = ""
    for ch in word:
        if ch != "*":
            result = result + ch
    return result
```

b) An *acronym* is a short string made from the first letter of each word. Write a function `make_acronym(words)` that takes a list of words and returns that string. Do **not** change whether letters are uppercase or lowercase.

Examples:

- `make_acronym(["addis", "coder", "program"])` → `"acp"`
- `make_acronym(["file", "transfer", "protocol"])` → `"ftp"`

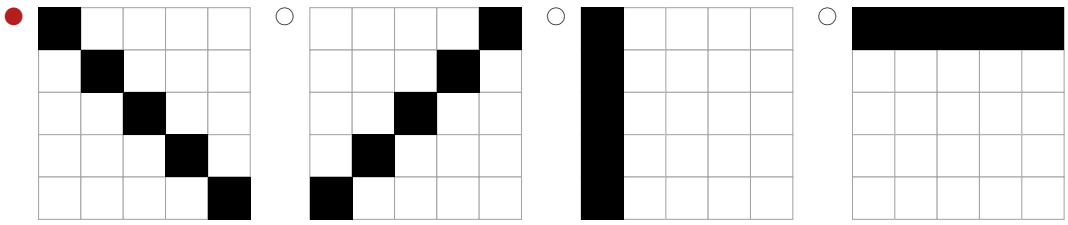
```
def make_acronym(words):
    result = ""
    for word in words:
        result = result + word[0]
    return result
```

5. [SimpleImage coordinates] Consider the following code, which uses the SimpleImage library. It starts with a blank 5×5 white image and then colors some pixels black.

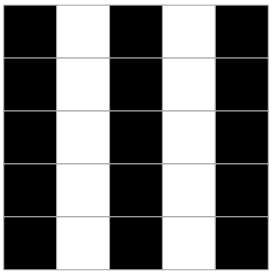
```
from simpleimage import SimpleImage

img = SimpleImage.blank(5, 5)
for i in range(5):
    img.set_rgb(i, i, 0, 0, 0)
```

Which image does this code produce? Select exactly one answer.



6. [SimpleImage and nested loops] Fill in the code below to produce this image using the SimpleImage library. Finish writing the function `draw_stripes`.



```
from simpleimage import SimpleImage

def draw_stripes(img):
    for x in range(img.width):
        for y in range(img.height):
            if x % 2 == 0:
                img.set_rgb(x, y, 0, 0, 0)
    return img

draw_stripes(SimpleImage.blank(5, 5)).show(resize_width=200)
```

7. [nested lists and helper functions] Consider the following two functions.

```
def fun_multiply(x):
    result = 1
    for i in x:
        result = result * i
    return result

def fun_func(x):
    result = []
    for i in x:
        result.append(fun_multiply(i))
    return result
```

What is the printed output when this line of code is run?

```
print(fun_func([[1, 2, 3], [2, 3, 0], [3, 5, 1]]))
```

```
[6, 0, 15]
```

8. [debugging recursion] Each function below has exactly one incorrect line of code.

a) The function `reverse_list` should return a new list with the elements of `lst` in reverse order. For example, `reverse_list([1, 2, 3])` should return `[3, 2, 1]`. **Circle the incorrect line of code.**

```
def reverse_list(lst):
    if len(lst) == 0:
        return 0
    return reverse_list(lst[1:]) + [lst[0]]
```

b) The function `find_max` should return the largest integer in a non-empty list.
For example: `find_max([3, 9, 2])` should return `9`. **Circle the incorrect line of code.**

```
def find_max(lst):
    if len(lst) == 1:
        return lst[0]
    rest_max = find_max(lst[1:])
    if lst[0] > rest_max:
        return lst[0]
    else:
        return rest_max
```

9. [recursion tracing] Consider the following code.

```
def countdown(n):
    if n == 0:
        print("Done!")
        return
    print(n)
    countdown(n - 1)

countdown(3)
```

a) What is the printed output?

```
3
2
1
Done!
```

b) How many times is the function `countdown` called (including the first call)?

```
4
```

10. [branching recursion tracing] Consider the following code.

```
def greeting(n):
    if n < 1:
        return 1
    elif n % 2 == 1:
        print("Hello")
        return greeting(n - 1) + greeting(n - 2)
    else:
        print("World")
        return greeting(n // 2)

result = greeting(3)
```

a) How many times is "Hello" printed?

3

b) How many times is "World" printed?

1

c) What is the value of the variable `result`?

4

11. [recursive digit product] Write a function `digit_product(n)` that uses recursion to multiply all the digits of the number `n`. You can assume that `n` ≥ 0.

Restrictions:

- You must use recursion.
- You **cannot** use any loops (`for` or `while`).
- It is okay to convert `n` to a string.

Examples:

- `digit_product(473) = 84`, because $4 \times 7 \times 3 = 84$.
- `digit_product(25) = 10`.
- `digit_product(0) = 0`.

Hint:

- `n % 10` gives the last digit. Example: `473 % 10` is `3`.
- `n // 10` removes the last digit. Example: `473 // 10` is `47`.

```
def digit_product(n):
    if n < 10:
        return n
    return (n % 10) * digit_product(n // 10)
```

12. [BONUS — recursion before and after] *Optional bonus problem*: only attempt this if you finish the rest of the quiz and have spare time.

Consider the following recursive function:

```
def mystery(n):
    if n == 0:
        return
    print("A", n)
    mystery(n - 1)
    print("B", n)
```

Write down the exact output, in order, of the call `mystery(3)`.

```
A 3
A 2
A 1
B 1
B 2
B 3
```

13. [BONUS — recursive subsequences] *Optional bonus problem*: only attempt this if you finish the rest of the quiz and have spare time.

A *subsequence* of a string `s` is a new string you make by keeping some letters from `s` (you may keep none, some, or all), but you must keep the same left-to-right order.

For example, if `s = "cat"`, some subsequences are:

- `""` (keep nothing)
- `"cat"` (keep everything)
- `"ct"`
- `"a"`

But `"tc"` is **not** a subsequence, because in `"cat"` the letter `c` comes before `t`, so you cannot put `t` first.

a) **Write `augment`.** Write a function `augment(x, y)`, where `x` is a list of strings and `y` is one character. For every string in `x`, add `y` at the end. Return the list of new strings.

Example: this code should print `["c", "ac", "abc"]`.

```
x = ["", "a", "ab"]
y = "c"
print(augment(x, y))
```

```
def augment(x, y):
    result = []
    for s in x:
        result.append(s + y)
    return result
```

b) **Generate every subsequence.** Now assume `augment(x, y)` already works. Using it, write a function `generate_all_subsequences(s)` that returns a list of every subsequence of the string `s`.

Example: this code should return `["", "a", "b", "ab", "c", "ac", "bc", "abc"]`.

```
s = "abc"
print(generate_all_subsequences(s))
```

```
def generate_all_subsequences(s):
    if len(s) == 0:
        return [""]
    without_last = generate_all_subsequences(s[:-1])
    with_last = augment(without_last, s[-1])
    return without_last + with_last
```

Scratch space (not graded)

Scratch space (not graded)