Mock Exam, Week 3

Name: _____

Directions

- **This exam contains 16 questions (+1 optional question) and will last 90 minutes.**
- Use your time wisely. If you are having too much trouble on a question, skip it and return to it later. **Avoid getting stuck.**
- In the answer options, the ↵ symbol indicates a new line. The ↵ symbol will only be used to separate lines of output and will not appear at the end of the final line.
- For questions with *circular bubbles*, you should select *exactly one* choice.
  - ○ You must choose either this option
  - ○ Or this one, but not both!
- For questions with *square checkboxes*, you may select *multiple* choices.
  - □ You could select this choice.
  - □ You could select this one too!

Staff use only.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Q12	Q13	Q14	Q15	Q16	Q17					Total

1. For each of the following lines, fill in the blank (_____) with **one of the following types**: `int`, `list`, `str`, `bool`, `float`.

```
print(type("Jam Coders"))    # Printed: ___________________
print(type(62.5))            # Printed: ___________________
print(type([1, 2, 4, 8]))    # Printed: ___________________
print(type(5 > 1))           # Printed: ___________________
print(type(5 // 2))          # Printed: ___________________
```

2. For which value(s) of `a` and `b` will the following code print `Ackee and Saltfish`?

```
if a and b:
    print("Patty")
else:
    print("Ackee and Saltfish")
```

Fill in the boxes next to all answers that print `Ackee and Saltfish`.

- ☐ `a = True, b = True`
- ☐ `a = True, b = False`
- ☐ `a = False, b = True`
- ☐ `a = False, b = False`

3. What would the code output when run?

```
L = [1, 1, 3, 2, 1, 2, 2]
final = 0
for i in L:
    final += i
print(final)
```

- ○ `0`
- ○ `1132122`
- ○ `12`
- ○ `final`

4. Consider the following code:

```
nums = [2, 4, 3]
i = 0
while i < len(nums):
    print(nums[i])
    if nums[i] % 2 == 0:
        i += 2
    else:
        i -= 1
```

When it is run, what is printed?

- ○ 2 ↵ 3 ↵ 4
- ○ 2 ↵ 4 ↵ 3
- ○ Nothing is printed.
- ○ An error occurs.

5. The following function searches a non-empty list of **positive integers** to find and return the **maximum value**. Fill in the blanks to complete the implementation.

```
def max_in_list(lst, item):
    max_value = 0
    for i in range(len(lst)):
        if __________ > max_value:    # YOUR CODE HERE
            ____________________      # YOUR CODE HERE
    return max_value
```

6. Consider the following lines of code:

```
def piyush():
    return "papaya"

def manolis():
    print("hmm")
    return "mango"

manolis()
print(piyush())
piyush()
print(manolis())
```

What is the printed output of the code above?

- ○ hmm ↵ mango ↵ None ↵ papaya
- ○ hmm ↵ None ↵ papaya ↵ hmm ↵ papaya
- ○ hmm ↵ mango ↵ papaya ↵ hmm ↵ mango
- ○ hmm ↵ papaya ↵ hmm ↵ mango

7. What is printed when you run the following code?

```
count = 0
while count == 0:
       if count == 5 or count > 2:
            break
       print(count)
```

- ○ 0 ↵ 1 ↵ 2 ↵ 3 ↵ 4
- ○ 0 ↵ 0 ↵ 0 ↵ 0 ↵ 0
- ○ An error occurs, and nothing is printed.
- ○ 0 ↵ 0 ↵ 0 ↵ 0 ↵ 0 ↵ 0 ↵ 0 ↵ 0 . . . (An infinite loop that keeps printing 0 ↵)

8. What is the output printed by each of the following code snippets? If a snippet results in an **infinite loop**, write `"Infinite"`. If it produces an **error**, write `"Error"` and briefly explain the reason. **Write your answer in the empty box after each snippet**.

(a)

```
x = [6, 2, 1, 2]
y = [1, 2, 2, 4]
z = 0

for i in range(len(x)):
    z = z + (x[i] * y[i])

print(z)
```

(b)

```
num1 = 5
num2 = 12

def max_value(num1, num2):
    if num1 > num2:
        return num1
    return num2

print(max_value(45, 32))
```

(c)

```
def foo(num):
  if num == 1:
    return 1
  return num * foo(num - 1)

print(foo(5))
```

(d)

```
def bar(lst):
  if len(lst) == 1:
    print(lst[0])
  bar(lst)

print(bar(["Hello", "world!"]))
```

9. This question has two parts.

Lydia, Bruno, and Sam all love fruits—but each of them has their own favorite! Ms. B uses the function `pour_juice` to serve them their favorite fruit.

As an example, the function call:

```
pour_juice([["Lydia", "Lychee"],
            ["Bruno", "Banana"],
            ["Sam", "Strawberry"]],
            "Sam")
```

Will print:

```
Sam LOVES Strawberry
```

a) **Fill in the blanks** to complete the function so that it behaves correctly.

```
def pour_juice(name_fruit_lst, name):
    for name_fruit_pair in name_fruit_lst:
        if name == ____________________:
            print(name_fruit_pair[0] + " LOVES " + ___________________)
            return
    print("Who's that?")
```

b) Consider the following statement

```
pour_juice([["Lydia", "Lychee"],
            ["Bruno", "Banana"],
            ["Sam", "Strawberry"]],
            "Frank")
```

What will be printed? Write your answer here: _____

10. The following code is intended to print numbers 50 to 0 at the intervals of 10. So it should print:

```
50
40
30
20
10
0
```

Fill in the blanks to make the code work as intended.

```
for num in range(50, _____ , _____):     # YOUR CODE HERE
    print(num)
```

11. Fill in the blanks in the code below so that the following is printed:

```
xxxxxxxxxxxxxxxxx
x               x
x               x
x               x
x               x
x               x
xxxxxxxxxxxxxxxxx
```

```
print("x"*17)
for row in range(5):
    print(____________________)       # YOUR CODE HERE
print(____________________)           # YOUR CODE HERE
```

12. A **palindrome** is a string that reads the same forwards and backwards. For example, `"bob"` and `"racecar"` are palindromes, while `"apple"` and `"race"` are not.

Note that the empty string and any single-character string are also considered palindromes.

Below is a **recursive** function, `is_palindrome`, which takes a lowercase string `s` as input and returns `True` if `s` is a palindrome, and `False` otherwise.

Fill in the blanks in the code below to complete the implementation:

```
def is_palindrome (s):
    if len(s) < 2:
        return ____________________     # YOUR CODE HERE
    if ____________________:            # YOUR CODE HERE
        return is_palindrome(s[1:-1])
    return ____________________         # YOUR CODE HERE
```

13. The function `sum_of_list(lst)` takes a list of integers as input and returns the sum of its elements. Note that the sum is computed **recursively** in this function.

Fill in the blanks in the code below to complete the implementation:

```
def sum_of_list (lst):
    if len(lst) == 0:
        return ____________________                    # YOUR CODE HERE
    return lst[0] + sum_of_list(____________________)   # YOUR CODE HERE
```

14. The function `intersection(lst1, lst2)` takes two lists as input and returns a new list containing all the elements that are present in **both** lists.

Fill in the blanks in the code below to complete the implementation:

```
def intersection (lst1, lst2):
    answer = []
    for elem in lst1:
        if ____________________:     # YOUR CODE HERE
            ____________________     # YOUR CODE HERE
    return answer
```

15. Write a **recursive** function `func(n)` that takes a **positive integer** `n` and follows these rules:

- If `n` is less than or equal to 3, the function returns `1`.
- Otherwise, it returns the sum of `func(n - 1)` and `func(n - 3)`.

Write your code below. You may not need to use all the lines.

16. Write a function called `collect_numbers()` that follows these rules:

- Create an empty list to store numbers.
- **Repeatedly** prompt the user to enter a number between 1 and 100 (inclusive).
- If the user enters a number less than 1 or greater than 100, print `"Invalid number"` and ask again.
- If the number is valid and has not been entered before, add it to the list and continue asking.
- If the number has already been entered before, stop asking for inputs.
- Finally, print the list of collected numbers.

For example, if the user enters the following inputs:

```
32
5
91
10
200
6
5
```

the output should be:

```
Invalid number
[32, 5, 91, 10, 6]
```

Note that the `"Invalid number"` was printed due to 200 and thus it was not included in the list. Write your code below. **You may not need to use all the lines**.

17. Optional Challenge Question. Write a function `challenge(lst)` that takes as input a list `lst`, where each element is a **non-empty list of integers**. The function should return a new list where each element is `True` if all the integers in the corresponding inner list are the same, and `False` otherwise. For example,

```
challenge([[1,1,1], [4,5], [6,6,8,6], [5]])
```

will return:

```
[True, False, False, True]
```

Write your code below. You may not need to use all of the lines.