Name: _____

1. Recursive Code Tracing

Carefully trace the execution of the following Python function. What is printed to the console when the function call `mystery(4)` is executed?

```
def mystery(n):
    if n <= 0:
        return
    print(n)
    mystery(n - 2)
    print(n + 1)
```

Write the complete, exact printed output below.

2. List Slicing Evaluation

Consider the following Python list definition.

```
data = [10, 20, 30, 40, 50, 60, 70, 80]
```

Determine exactly what each print statement displays.

a) `print(data[2:6])`

b) `print(data[::-2])`

c) `print(data[5:1:-1])`

3. Writing Slicing Expressions

Given the initial list:

```
values = [5, 12, 19, 26, 33, 40, 47, 54, 61, 68]
```

Write the single Python slicing expression (e.g., `values[...]`) that produces each desired target output list.

a) Target output: `[19, 26, 33, 40]`

```
values[                    ]
```

b) Target output: `[68, 54, 40, 26, 12]`

```
values[                    ]
```

c) Target output: `[33, 47, 61]`

```
values[                    ]
```

4. Counting Vowels

Write a recursive function called `count_vowels` that takes a lowercase string `s` and returns the total number of vowels (`"a"`, `"e"`, `"i"`, `"o"`, `"u"`) present in the string. You may **not** use loops.

Step 1: Ideation & Testing.

Before writing code, write down exactly three input-output test cases. Think carefully about the smallest possible inputs (edge cases).

1. ***Case 1 (Edge Case):*** `count_vowels("")` → _____
2. ***Case 2:*** `count_vowels("xyz")` → _____
3. ***Case 3:*** `count_vowels("jamaica")` → _____

Step 2: Implementation.

Fill in the blanks for the `count_vowels` function.

```
def count_vowels(s):

    # BASE CASE

    if _______________________________:

        return ____________________________________

    # Check the first character

    if _________________ in _________________:

        # RECURSIVE CASE #1

        return ____________________________________

    # RECURSIVE CASE #2

    return ____________________________________
```

5. N-th Fibonacci Number

The Fibonacci sequence begins with `0` and `1`. Each subsequent number is found by adding the two previous numbers together: `0, 1, 1, 2, 3, 5, 8, 13, 21, ...`

Write a recursive function called `fibonacci` that takes a non-negative integer `n` and returns the n-th Fibonacci number (where `n = 0` returns `0`, `n = 1` returns `1`, and so on). You may **not** use loops.

Step 1: Ideation & Testing.

Write down three representative test cases covering the sequence boundaries and a standard case.

1. ***Case 1 (Edge Case):*** `fibonacci(0)` $\rightarrow$ _____
2. ***Case 2:*** `fibonacci(1)` $\rightarrow$ _____
3. ***Case 3:*** `fibonacci(5)` $\rightarrow$ _____

Step 2: Implementation.

Fill in the blanks for the `fibonacci` function.

```
def fibonacci(n):

    # BASE CASE #1

    if ______________________________:
        return 0

    # BASE CASE #2

    if ______________________________:
        return 1

    # RECURSIVE STEP

    return ________________________________________________
```

6. *Challenge Recursion*: Grid Paths

You are placed at the top-left corner of an $M \times N$ grid. You can only move either ***down*** or ***right*** at any point in time.

Write a recursive function `count_paths(m, n)` that returns the total number of unique paths to reach the bottom-right corner of the grid.

Hint: If you are on an edge (e.g., only 1 row or 1 column left), there is only 1 path straight to the destination.

Your function ***must use recursion*** and must not use math formulas (like factorials) or loops.

```
def count_paths(m, n):
    # Write your base cases and recursive steps below:
```

7. *Challenge Recursion*: Making Change

Write a recursive function `count_change(amount, coins)` that counts the number of ways to make change for a target `amount` of money using a list of available coin denominations.

Hint: The total ways to make change using a set of coins is the sum of:

1. *The ways to make change using the first coin denomination (keeping the same list of coins but reducing the amount).*
2. *The ways to make change without using the first coin denomination at all.*

Your function ***must use recursion*** and must not use loops.

```
def count_change(amount, coins):
    # Write your base cases and recursive steps below:
```