Name: _____ **Surname:** _____ **Date:** _____

Debugging (15 points each)

Instructions

The functions below contain one or more errors. You have to find these errors, explain why they happen, and suggest a fix to them.

Questions

1. This code is supposed to count down from 5 to 1 and print "Launch!" when the countdown ends. But it never stops.

   - Identify the issue and fix it. *(10 points)*
   - Ensure the count stops at 1. *(5 points)*

```python
def countdown(start: int = 5):
    while start >= 0:
        print(start)
        start + 1
    print("Launch!")

countdown()
```

2. The following code is intended to create two sets: one containing the original members of a group, and another containing both the original and new members. It then displays the difference between these sets. However, the output is not as expected. Explain the issue and fix the code.

```python
group_before = {"Apple", "Banana", "Carrot"}

# Duplicate the original group and add the new members
group_now = group_before
for new_member in ["Eggplant", "Date"]:
    group_now.add(new_member)

# Compare the groups
print(group_now - group_before)
# Expected output: {"Date", "Eggplant"}
# Actual output: set()  <--- Empty set
```

3. This function is intended to increment the first element of an array by 1, the second by 2, the third by 3, etc. However, it doesn't return the expected result.

   - Ensure the function outputs an array, instead of `None`. *(5 points)*
   - Make the output array be the expected output. *(5 points)*
   - After you run this function, what will be the value of `x`? *(5 points)*

```python
import numpy as np

def increment_array(arr: np.array):
    length = len(arr)
    for idx in range(length):
        arr[idx] = arr[idx] + idx

x = np.array([1, 0, 1])
y = increment_array(x)

print(y)
# Expected Output: [2, 2, 4]
# Actual Output: None

print(x)  # What will be its value now?
```

4. The following code aims to take a subslice of the array "x". See what the user expects to get, and what is the actual output.

   - Explain briefly what is happening. *(5 points)*
   - Update the code to get the expected output. *(10 points)*

```python
import numpy as np

x = np.array(
    [[10, 20, 30],
     [40, 50, 60],
     [70, 80, 90]])

idx_rows = [0, 2]
idx_cols = [1, 2]

print(x[idx_rows, idx_col])
# Expected output: [[20, 30], [80, 90]]
# Actual output:  [20, 90]
```

Pseudo-Code

Instructions

In the following exercises, you will be asked to write Python functions by hand. Please follow these guidelines:

- Any build-in function you need is in the cheat-sheet provided.
- Clearly indicate each level of indentation using a right arrow (⇒).
- Ensure your code works correctly for any possible example.
- If your code has errors, it will be evaluated based on the provided docstring, comments, and explanations.

Full points will be awarded for code that performs as expected under all conditions.

5. Random Walk (20 points)

You are given a list of player names and a fixed number of steps X. Each player begins at position 0 and performs a random walk of X steps. Write a function `random_walks()` that simulates this process. At each step, the player should either move $+1$ or -1, chosen uniformly at random. After completing the walk for each player, return a dictionary where each key is a player's name, and the corresponding value is that player's final position *(15 points)*.
Your function should accept the following arguments:

- `players` (positional): A list or tuple of player names.
- `num_steps` (keyword): The number of steps for each player's random walk (same number for all of them), defaulting to 10.
- `random_seed` (keyword): If provided, set the random seed to ensure reproducible results.

Additionally, if a player's name appears multiple times in the input list, append a "-" to subsequent occurrences to ensure that all dictionary keys are unique. For example, if "Alex" appears twice, the keys should be "Alex" and "Alex−". If "Bob" appears three times, the keys should be "Bob", "Bob−", and "Bob−−". *(5 points)*
See the code below for an example of input and output:

```
players = ["Alex", "Alex", "Bob", "Bob", "Bob", "Chia"]
x = 10

print(random_walks(players, num_steps=x))
# Example output (actual results will vary due to randomness):
# {"Alex": 2, "Alex-": 0, "Bob": -1, "Bob-": -3, "Bob--": 0, "Chia": 4}
```

6. Training Alignment at IE University (20 points)

Each department at IE University offers a list of training modules. Some team members cannot attend certain modules because they have already completed them. Your task is to find which modules can be attended by everyone in each department.
Create a function `modules_for_team()`. It takes two inputs:

- `modules_by_dept`: a dictionary where keys are department names and values are lists of module titles.
- `employee_opt_outs`: a list of tuples with employee names and the modules they cannot attend.

The function `modules_for_team()` returns a dictionary with each department's module list filtered so that only universally acceptable modules remain. *(15 points)*
If a department ends up with no options for everyone, it is not included in the output. *(3 points)*
Also, ensure the original inputs remain unchanged after running your function. *(2 points)*
See the code below for an example of input and output:

```
modules_by_dept = {
    "HR": ["Harmony", "Inclusion", "HybridOps"],
    "Tech": ["Coding", "CloudTech", "Scrum"],
    "Admissions": ["CRM", "Recruitment", "Outreach"]
}

employee_opt_outs = [
    ("Sonia", ["Inclusion", "Scrum"]),
    ("Jose", ["Harmony", "HybridOps"]),
    ("Angel", ["CloudTech"]),
    ("Yuriko", ["Recruitment", "CloudTech"]),
    ("Sara", ["Inclusion", "CRM"]),
]

modules_viable = modules_for_team(modules_by_dept, employee_opt_outs)
print(modules_viable)
# Expected output:
# {
# "Tech": ["Coding"],
# "Admissions": ["Outreach"]
# }
```