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. The following Python code is intended to print the factorials of numbers starting from 3, incrementing by 1 each time, and continuing until the user decides to stop. However, the code contains **three errors** that prevent it from functioning correctly. Identify and fix these errors.

   - Line 4 raises the error `SyntaxError: invalid syntax`. *(5 points)*
   - The loop should print "`3!=6`" but instead prints "`True`". *(5 points)*
   - After fixing the previous errors, the loop should continue with "`4!=24`", then print "`5!=120`", etc., but instead it prints "`3!=18`", then "`3!=54`"... *(5 points)*

```
1  # Initialize the loop control and factorial variables
2  number = 3
3  factorial = 6
4  continue = True
5
6  while continue:
7      factorial = factorial * number
8      print(f"{number!=factorial}")
9      ask = input("Continue? (yes/no): ")
10     # If the user says "yes", proceed to the next number
11     if ask == "yes":
12         continue = True
13     else:
14         continue = False
```

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.

```
1  group_before = {"Luke", "Han", "Leia"}
2  # Duplicate the original group and add the new members
3  group_now = group_before
4  for new_member in ["Ben", "Arturo"]:
5      group_now.add(new_member)
6
7  # Compare the groups
8  print(group_now - group_before)
9  # Expected output: {"Arturo", "Ben"}
10 # Actual output: set()  <--- Empty set
```

3. The following function should mutliply the elements of an array by 2. However the code shows an unexpected result. Explain the issue and fix the code. *(10 points)*

```
import numpy as np

def double_array(arr: np.array) -> np.array:
    # Multiply each element by 2
    arr_double = arr * 2
    return arr_double

print(double_array([1, 2, 3]))
# Expected output: [2, 4, 6]
# Actual output:   [1, 2, 3, 1, 2, 3]
```

How would `double_array()` work on NumPy arrays of 2 or more dimensions? *(5 points)*

4. The following code is intended to replace all negative numbers in the array "`data`" with their absolute values using a boolean mask. However, the code raises an error `ValueError: NumPy boolean array indexing assignment cannot assign 7 input values to the 3 output values where the mask is true`. Explain the issue and fix the code. *(10 points)*

```
import numpy as np

data = np.array([5, -3, 7, -1, 0, -6, 8])
mask = data < 0
data[mask] = -data
print(data)
# Expected output: [5, 3, 7, 1, 0, 6, 8]
# Actual output: ValueError
```

Additionally, modify the code so that only the negative numbers greater than -5 are replaced with their absolute values. In the example above, the new output should be: `[5, 3, 7, 1, 0, -6, 8]` *(5 points)*.

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. Count Valid Reports (20 points)

You are given a list of reports, each report consisting of a string of single digits separated by spaces, for instance `"1 2 3"`. A report is considered valid if it meets the following conditions:

1. All digits are strictly increasing from start to end.
2. The absolute difference between any two adjacent digits is at least 1 and at most 3.

Write a function `count_valid_reports()` that, given **any** list of reports of **any size**, returns an integer: the number of valid reports. For instance:

```
ls_reports = [
    "1 2 7 8 9",
    "1 3 2 4 5",
    "1 4 4 6 8",
    "1 3 6 7 9"
]

print(count_valid_reports(ls_reports))  # Output: 1
```

The logic for the example above is:

- `"1 2 7 8 9"`: Invalid because `"2 7"` is an increase of 5.
- `"1 3 2 4 5"`: Invalid because `"1 3"` is increasing but `"3 2"` is decreasing.
- `"1 4 4 6 8"`: Invalid because `"4 4"` is not an increase.
- `"1 3 6 7 9"`: Valid because the digits are all increasing by 1, 2, or 3.