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 function below is designed to remove all vowels from a given string. However, it doesn't remove all the vowels as expected. Identify the problem and fix the code.

```
1  def remove_vowels(input_str: str) -> str:
2      vowels = ("a", "e", "i", "o", "u")
3      result = ""
4      for char in input_str:
5          if char.lower() in vowels:
6              input_str.replace(char, "")
7      return input_str
8
9  text = "Programming"
10 print(remove_vowels(text))  # Output: "Programming"
```

2. The following function is supposed to merge two dictionaries, without modifying the originals. In case of key conflicts, the value from the second dictionary should overwrite the one from the first. The code raises an error on line 5: "`SyntaxError: cannot assign to function call here`". Identify and fix the issue(s).

```
1  def merge_dictionaries(dict1: dict, dict2: dict) -> dict:
2      merged_dict = dict1
3      for key, value in dict2.items():
4          merged_dict.keys() = value
5      return merged_dict
6
7  dict_a = {"a": 1, "b": 2}
8  dict_b = {"b": 3, "c": 4}
9  print(merge_dictionaries(dict_a, dict_b))
10 # Expected: {"a": 1, "b": 3, "c": 4}
```

3. This code is meant to generate a list of squares of numbers from 1 to 5 using a while loop. However, it results in an infinite loop. Find the mistake and correct it.

```
def generate_squares(limit: int=5) -> list[int]:
    squares = []
    i = 1
    while i <= limit:
        squares.append(i ** 2)
    return squares

limit = -1
print(generate_squares())  # Expected: [1, 4, 9, 16, 25]
```

4. You have created a function that stacks two 1D-arrays, one of top of another. If the arrays have a different length, the function fills that difference with zeros. Your function raises an error on line 6: "`ValueError: could not broadcast input array from shape (3,) into shape (0,)`". Find the mistake(s) and correct it.

```
import numpy as np

def stack_uneven(x: np.array, y: np.array) -> np.array:
    if len(x) > len(y):
        y = np.zeros(len(x))
        y[len(y):] = y
    else:
        x = np.zeros(len(y))
        x[len(x):] = x
    return np.stack((x, y))

x = np.array([1, 2, 3])
y = np.array([4, 5])
print(stack_uneven(x, y))
# Expected:
# [[1, 2, 3],
#  [4, 5, 0]]
# Output:
# [[1., 2., 3.],
#  [0., 0., 0.]]
```

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.

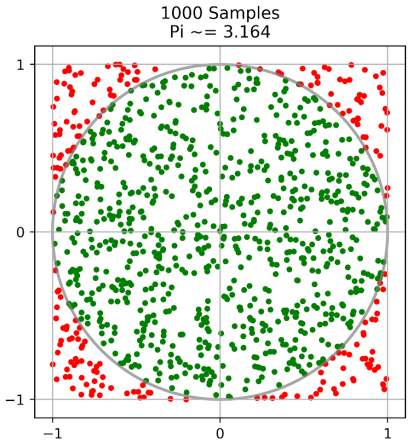
5. Greatest Common Divisor (20 points)

The Greatest Common Divisor (GCD) of two integers is the largest positive integer that divides each of the integers without leaving a remainder. For instance, the GCD of 6 and 15 is 3. Write a Python function named `find_gcd()` that takes two integers as input and returns the GCD of these numbers. *(12 points)*

- Your function should handle cases where either or both of the input numbers could be negative, returning the GCD as a positive integer. *(3 points)*
- If one of the inputs is zero, the GCD should be the absolute value of the non-zero number. For example, the GCD of 0 and 5 is 5. *(3 points)*
- If both inputs are zero, the function should return 0, as GCD is not defined in this case. *(2 points)*

6. Approximating Pi with Monte Carlo (20 points)

Consider a square with a circle inside it. We randomly generate points within this square. The image below illustrates this with green points inside the circle and red points outside.



To estimate the value of π, we use the Monte Carlo method. This involves comparing the number of points inside the circle (green) to the total number of points (green and red). The formula for approximating π is:

$$\pi \approx 4 \times \frac{\text{number of blue points}}{\text{total number of points}}$$

Write the `monte_carlo_pi()` function to approximate π. It should:

- Accept `num_points`, the number of random points to create.
- Generate random points in a square with corners (-1, -1) and (1, 1).
- Count how many points fall inside the unit circle $x^2 + y^2 \leq 1$.
- Calculate π following the equation above.
- Return this estimated value of π.