Practice Exam 3

Name:

....

- This is a digital exam consisting of 6 assignments where you will write short Python programs.
- You will complete the exam using the online editor linked on the course website.
- Use a single file for all solutions called mcs_practice3.py. This file is already open in the online exam editor.
- You may use the course website (mcs.proglab.nl) as a recource. Keep in mind that:
  - you cannot use any other website,
  - you cannot use rely on any existing code you've written before this exam,
  - you cannot get assistance with the programming during the exam.
- You will be **evaluated solely on the correctness** of your solutions. Code design, comments, and style are not important, so you do not need to worry about them.
- Do not use external modules such as numpy, csv, or others unless the assignment specifically says you can.

- **Submitting the exam**. When you have completed all assignments:
  - use the **submit button** on the website to ensure your final edits are committed,
  - **go to the teacher** with your **student card or ID** and this exam sheet,
  - have the teacher mark your work as completed,
  - hand in this exam sheet.

Assignment 1: Odd

Write a function named `sumodd(numbers)` that accepts a list of integers and returns the sum of all odd numbers in the list.

Example Usage:

```
numbers = [1, 2, 3, 1, 2, 3, 3, 4]
result = sumodd(numbers)
print(result)
```

Expected Output:

```
11
```

Tip: you can use `number % 2 == 1` to test if a number is odd.

Assignment 2: Reverse

Write a function named `reverse_list(lst)` that accepts a list and returns a new list with the elements in reverse order.

Example Usage:

```
lst = ["he", "l", "lo"]
result = reverse_list(lst)
print(result)
```

Expected Output:

```
['lo', 'l', 'he']
```

Assignment 3: Reps

Write a function named `longest_repetition(lst)` that accepts a list and returns the length of the longest consecutive repetition of the same element in the list.

For example, in the list `[1, 2, 2, 5, 5, 5, "a", "a"]`, the element `5` appears three times in a row, which is the longest consecutive repetition.

Example Usage:

```
lst = [1, 2, 2, 5, 5, 5, "a", "a"]
result = longest_repetition(lst)
print(result)
```

Expected Output:

```
3
```

Assignment 4: Combine

Write a function named `combine_dicts(dict1, dict2)` that accepts two dictionaries and returns a new dictionary that combines them.

- If a key is present in only one of the dictionaries, include it in the new dictionary with its corresponding value.
- If a key is present in both dictionaries, include it in the new dictionary with a list of both values.

For example, given `dict1 = {"a": 9, "b": 22, "c": 8}` and `dict2 = {"x": 2, "b": 18, "y": 3}`, the key `"b"` is present in both dictionaries. In the combined dictionary, `"b"` will have the value `[22, 18]`.

Example Usage:

```
dict1 = {"a": 9, "b": 22, "c": 8}
dict2 = {"x": 2, "b": 18, "y": 3}
result = combine_dicts(dict1, dict2)
print(result)
```

Expected Output:

```
{'a': 9, 'b': [22, 18], 'c': 8, 'x': 2, 'y': 3}
```

Assignment 5: Newton

Write a function named `newtons_method_square_root_t(n, threshold)` that computes the square root of a number `n` using Newton's method. The function should repeatedly improve the estimate of the square root until the absolute difference between `root ** 2` and `n` is less than the given `threshold`.

Newton's method is used to find approximations of the square root of a number. To find the square root of a number `n`, we start with an initial guess (`n` itself) and iteratively refine the root using the formula:

```
root = 0.5 * (root + n / root)
```

In each iteration, this formula produces a better approximation of the square root. The process continues until the difference between `root ** 2` and `n` is less than the specified `threshold`.

Example Usage:

```
result1 = newtons_method_square_root_t(9, 1)
result2 = newtons_method_square_root_t(9, 0.01)
result3 = newtons_method_square_root_t(16, 0.1)
print(result1)
print(result2)
print(result3)
```

Expected Output:

```
3.023529411764706
3.00009155413138
4.002257524798522
```

Assignment 6: Grid

Write a function named `generate_1x1_grid(step)` that generates all coordinate points within the 1x1 grid (from (0,0) to (1,1)) using the given `step` value. The function should return a list of tuples representing the grid points.

Example Usage:

```
result = generate_1x1_grid(0.5)
print(result)
```

Expected Output:

```
[(0.0, 0.0), (0.0, 0.5), (0.0, 1.0), (0.5, 0.0), (0.5, 0.5), (0.5, 1.0), (1.0, 0.0),
(1.0, 0.5), (1.0, 1.0)]
```