CSC110 Lecture 12: For Loops

Exercise 1: Practice with for loops

1. Consider the following function.

```
def sum_of_squares(numbers: list[int]) -> int:
    """Return the sum of the squares of the given numbers.

    >>> sum_of_squares([4, -2, 1])  # 4 ** 2 + (-2) ** 2 + 1 ** 2
    21
    """
    sum_so_far = 0

    for number in numbers:
        sum_so_far = sum_so_far + number ** 2

    return sum_so_far
```

   a. What is the loop variable?

   b. What is the accumulator?

   c. Fill in the loop accumulation table for the call to function `sum_of_squares([4, -2, 1])`.

Iteration	Loop variable (`number`)	Loop accumulator (`sum_so_far`)
0	N/A	0
1		
2		
3		

2. Implement the following function.

```python
def long_greeting(names: list[str]) -> str:
    """Return a greeting message that greets every person in names.

    Each greeting should have the form "Hello <name>! " (note the space at
        the end).
    The returned string should be the concatenation of all the greetings.

    >>> long_greeting(['David', 'Mario'])  # Note the "extra" space at the
        end
    'Hello David! Hello Mario! '
    """
```

Exercise 2: Marriage license data revisited

In Lecture 9, we saw how to query marriage license data using a nested list (i.e., `list[list]`). In Lecture 10, we saw how to use data classes to store the marriage license data using a list of `MarriageData` (i.e., `list[MarriageData]`):

```python
@dataclass
class MarriageData:
    """..."""
    id: int
    civic_centre: str
    num_licenses: int
    month: datetime.date
```

Each of the following sets of functions takes marriage license data and performs some aggregation of those values. The first function in each set is implemented for you and uses a nested list (as we did at the end of last week). Your task is to implement each of the other two functions in the set in two ways: first with a comprehension, and second with a for loop.

1.
```python
def total_licenses_for_centre_v1(data: list[list], civic_centre: str) ->
        int:
    """Return how many marriage licenses were issued in the given civic
        centre."""
    return sum([row[2] for row in data if row[1] == civic_centre])
```

```python
def total_licenses_for_centre_v2(data: list[MarriageData], civic_centre:
        str) -> int:
    """Return how many marriage licenses were issued in the given civic
        centre."""
```

```
def total_licenses_for_centre_v3(data: list[MarriageData], civic_centre:
        str) -> int:
    """Return how many marriage licenses were issued in the given civic
        centre."""
```

2.

```
def civic_centre_meets_threshold_v1(data: list[list], civic_centre: str,
        num: int) -> bool:
    """Return whether civic_centre issued at least num marriage licences
        every month.

    You only need to worry about the rows that appear in data; don't worry
        about "missing" months.


    Preconditions:
        - num > 0
        - civic_centre in {'TO', 'NY', 'ET', 'SC'}
        - data satisfies all of the properties described in Worksheet 9,
        Exercise 2
    """
    licenses_issued = [row[2] for row in data if row[1] == civic_centre]
    return all(num_issued >= num for num_issued in licenses_issued)
```

```
def civic_centre_meets_threshold_v2(data: list[MarriageData], civic_centre:
        str, num: int) -> bool:
    """Return whether civic_centre issued at least num marriage licences
        every month.

    You only need to worry about the rows that appear in data; don't worry
        about "missing" months.


    Preconditions:
        - num > 0
        - civic_centre in {'TO', 'NY', 'ET', 'SC'}
    """
```

```
def civic_centre_meets_threshold_v3(data: list[MarriageData], civic_centre:
        str, num: int) -> bool:
    """Return whether civic_centre issued at least num marriage licences
        every month.

    You only need to worry about the rows that appear in data; don't worry
        about "missing" months.

    Preconditions:
        - num > 0
        - civic_centre in {'TO', 'NY', 'ET', 'SC'}
    """
```

Exercise 3: Looping over other data types

For each function, add at least one example to the docstring and complete the function body.

1.
```
def count_uppercase(s: str) -> int:
    """Return the number of uppercase letters in s.









    """
```

2.
```
def all_fluffy(s: str) -> bool:
    """Return whether every character in s is fluffy.

    Fluffy characters are those that appear in the word 'fluffy'.









    """
```

3.
```
def sum_davids(scores: dict[str, int]) -> int:
    """Return the sum of all values in scores that correspond to a key that
        contains 'David'.









    """
```

4.
```
def david_vs_mario(scores: dict[str, int]) -> str:
    """Return the name of the person with the highest total score in scores.

    David's score is the sum of all values in scores that correspond
    to a key that contains the string 'David'.

    Mario's score is the sum of all values in scores that correspond
    to a key that contains the string 'Mario'.









    """
```

Additional Exercises

Implement the function below in two ways: first using comprehensions, and second using a for loop.

```
def count_anomalies(data: list[MarriageData]) -> int:
    """Return the number of months where there is at least one
    civic centre differing by at least 100 from the average number
    of marriage licenses.
    """
```