CSC110 Lecture 12: More With For Loops

Hisbaan Noorani

October 07, 2020

Contents

1 Ex 1: Looping with indexes

1. Consider the following function, which we studied last class.

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

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

    >>> all_fluffy('fffffuy')
    True
    >>> all_fluffy('abcfluffy')
    False
    """
    for letter in s:
        if letter not in 'fluffy':
            return False

    return True
```

In the space below, rewrite the body of the function so that it uses an index-based for loop instead of the element-based for loop.

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

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

    >>> all_fluffy('fffffuy')
    True
    >>> all_fluffy('abcfluffy')
    False
    """
    for i in range(len(s)):
        if s[i] not in 'fluffy':
            return False

    return True
```

2. Implement each of the following functions using index-based for loops.

```python
def is_sorted(lst: List[int]) -> bool:
    """Return whether lst is sorted.

    A list L is sorted when for every pair of *adjacent* elements
    x and y in L, x <= y.

    Lists of length < 2 are always sorted.

    >>> is_sorted([1, 5, 7, 100])
    True
    >>> is_sorted([1, 2, 1, 2, 1])
    False
    """
    for i in range(0, len(lst) - 1):
        if lst[i] > lst[i + 1] :
            return False

    return True
```

```python
def inner_product(nums1: List[float], nums2: List[float]) -> float:
    """Return the inner product of nums1 and nums2.

    The inner product of two lists is the sum of the products of the
    corresponding elements of each list:

        sum([nums1[i] * nums2[i] for i in range(0, len(nums1))])

    Preconditions:
        - len(nums1) == len(nums2)

    >>> inner_product([1.0, 2.0, 3.0], [0.5, 2.5, 0.0])
    5.5
    """
    sum_so_far = 0

    for i in range(len(nums1)):
        sum_so_far = sum_so_far + (nums1[i] * nums2[i])

    return sum_so_far
```

```python
def stretch_string(s: str, stretch_factors: List[int]) -> str:
    """Return a string consisting of the characters in s, each repeated
    a given number of times.


    Each character in s is repeated n times, where n is the int at the
    corresponding index in stretch_factors.
    For example, the first character in s is repeated stretch_factors[0] times.

    Preconditions:
        - len(s) == len(stretch_factors)
        - all({factor >= 0 for factor in stratch_factors})

    >>> stretch_string('David', [2, 4, 3, 1, 1])
    'DDaaaavvvid'
    >>> stretch_string('echo', [0, 0, 1, 5])
    'hooooo
    """
```

2 Ex 2: Nested Loops

1. Implement this function:

```python
def total_mice(dict_of_cats: Dict[str, List[str]]) -> int:
    """Return the number of mice stored in the given cat dictionary.

    dict_of_cats is a dictionary here:
        - Each key is the name of a cat
        - Each corresponding value is a list of items that the cat owns.
          An item is a *mouse* when it contains the string 'mouse'.
          (You can use the "in" operator to check whether one string is
          in another.)

    >>> total_mice({'Romeo': ['mouse 1', 'my fav mouse', 'flower'],
    ...             'Juliet': ['sock', 'mouse for tonight']})
    3
    >>> total_mice({'Asya': ['chocolate', 'toy'], 'Mitzey': []})
    0
    """
    num_of_mice = 0

    for cat_name in dict_of_cats:
        for item in range(len(dict_of_cats[cat_name])):
            if 'mouse' in item:
                num_of_mice = num_of_mice + 1

    return num_of_mice
```

2. Complete the following loop accumulation table to trace the sample function call `total_mice({’Romeo’: [’mouse’, ’my fav mouse’, ’flower’], ’Juliet’: [’sock’, ’dinner mouse’]})`. (We’ve started it for you to save some time.)

Outer Loop Iteration	Outer Loop Variable	Inner Loop Iteration	Inner Loop Variable	Accumulator
0	n/a	n/a	n/a	0
1	'Romeo'	0	n/a	0
1	'Romeo'	1	'mouse'	1
1	'Romeo	2	'my fav mouse'	2
1	'Romeo	3	'flower'	2
2	'Juliet'	0	n/a	2
2	'Juliet	1	'sock'	2
2	'Juliet	2	'dinner mouse'	3

3. Implement this function using a nested loop.

```python
def can_pay_with_two_coins(denoms: Set[int], amount: int) -> bool:
    """Return whether the given amount is the sum of two distinct numbers
    from denoms.

    >>> can_pay_with_two_coins({1, 5, 10, 25}, 35)
    True
    >>> can_pay_with_two_coins({1, 5, 10, 25}, 12)
    False
    """

    # check every combination of two coins

    for i in range(len(denoms):
        for j in range(len(denoms)):
            if denoms[i] + denoms[j] and denoms[i] != denoms[j]:
                return True

    return False
```

4. Implement this function using a nested loop.

```python
import math


def max_average(lists_of_numbers: List[List[float]]]) -> float:
    """Return the largest average of the given lists_of_numbers.

    Preconditions:
        - lists_of_nubers != []
        - all({numbers != [] for numbers in lists_of_numbers})

    >>> max_average([[1.0, 3.4], [3.5, 4.0, -2.5]])
    2.2
    """
    # ACCUMULATOR max_so_far: keep track of the maximum average of the lists
    # visited so far. We initialize to negative infinity so that any
    # computed average will be greater than the starting value.
    # (i.e., for all floats x, x > -math.inf)
    max_so_far = -math.inf

    for list in lists_of_nubers:
        average = sum(list) / len(liss)
        if average > max_so_far:
            max_so_far = average

    return max_so_far
```

3 Additional Exercises

1. Write a function that takes a string `s` and returns whether s is a palindrome. A palindrome is a string consists of the same sequence of characters in left-to-right order as right-to-left order. `'davad'` is a palindrome, and `'david'` is not.
2. Write a function that takes two lists of integers, which have the same length and are non-empty, and returns the greatest absolute difference between the numbers at corresponding positions in the lists.
3. Write a new version of `max_average` that does the same thing, except it returns the list with the highest average rather than the highest average.
4. Hint: use two accumulator variables, one to keep track of the highest average itself, and another to keep track of the list with the highest average.
5. Re-implement all of the functions on this worksheet using comprehensions. You might need to define some separate functions as well.