Introduction to Computer Science I - New SME Assignment Solution

Niranjana Srinivasa Ragavan

March 2025

In this document, I have written the explanations for the questions and solutions provided in the given assignment document. I have used PythonTeX package to execute the python code written within the LaTeX. My explanations are based on the assumption of prerequisite knowledge specified in the assignment.

KP1. Understanding Nested List Comprehensions

G1.1 (E) Predicting the Output of a Nested List Comprehension

Question

Consider the following code snippet

```
nested_list = [[(i + j) ** 2 for j in range(1, 4)] for i in range(1, 4)]
print(nested_list)
```

What will be printed?

Solution

```
Output: [[4, 9, 16], [9, 16, 25], [16, 25, 36]]
```

Explanation

Recall that nested list comprehension consists of list comprehensions within a list comprehension. Expressions inside `[]` are called list comprehensions.

- In this code, the outer list comprehension `for i in range(1, 4)` generates values $i = 1, 2, 3$.
- For each value of `i`, the inner list comprehension `for j in range(1, 4)` generates values $j = 1, 2, 3$.
- For each pair of `i` and `j`, the expression `(i + j) ** 2` is computed.
- For each iteration of `i`, the computed result is stored as a sublist and the final result is the list of all these sublists.
- **Step-by-step evaluation**

  i | j | (i+j)**2 |
  1 | 1 | 4 |
  1 | 2 | 9 |
  1 | 3 | 16 |
  Sublist for i = 1: [4, 9, 16]
  i | j | (i+j)**2 |
  2 | 1 | 9 |
  2 | 2 | 16 |
  2 | 3 | 25 |
  Sublist for i = 2: [9, 16, 25]
  i | j | (i+j)**2 |
  3 | 1 | 16 |
  3 | 2 | 25 |
  3 | 3 | 36 |
  Sublist for i = 3: [16, 25, 36]

- **Scope of the variables:**
  - In python, the inner loop in the nested `for` loop shares the same scope as the outer enclosing scope. Thus, outer loop variables defined before the inner loop can be directly referenced by the inner loop.

- In nested list comprehension, each list comprehension creates its own local scope.
- Still, the inner list comprehension can access the variables from the outer comprehension, due to a property called **Closures**. The inner list comprehension can access the variables from the enclosing scope.
- In this code, the non-local variable `i`, which is in the scope of outer comprehension, can be accessed as a free variable by inner list comprehension. This means, `i` is not bound to the inner comprehension, but is stored in a closure. This repeats for each iteration of `i`.
- This can be evident from this line of byte-level code `LOAD_CLOSURE 1 (i)`.

G1.2 (M) Converting a `for` Loop to a Nested List Comprehension

Question

Rewrite the following `for` loop as a nested list comprehension:

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
new_matrix = []
for row in matrix:
    new_row = []
    for num in row:
        new_row.append(num + 2)
    new_matrix.append(new_row)
```

Solution

```
new_matrix = [[num + 2 for num in row] for row in matrix]
```

Explanation

- **Explanation of nested `for` loops:** The outer `for` loop iterates through each row in the matrix. The inner loop iterates through each number in the row and creates a new row by adding 2 to each number in the original row. The new rows are collected to create the new matrix.
- **Creating list comprehension:** Replace the outer loop with `[..for row in matrix]`. In the inner loop, for each `num` in row, `num + 2` expression should be executed. Replace the inner loop with `[num + 2 for num in row]`.
- Finally, all the modified rows are collected to create `new_matrix`. The list comprehension implicitly builds the list and there is no need for `append` function.

KP2. Using Nested List Comprehensions with Strings

G2.1 (E) Predicting Output for Uppercasing Nested Words

Question

What will be printed when the following code is run?

```
sentence = "I am happy"
words = [[char.upper() for char in word] for word in sentence.split()]
print(words)
```

Solution

```
Output: [['I'], ['A', 'M'], ['H', 'A', 'P', 'P', 'Y']]
```

Explanation

- `sentence.split()` method splits the sentence and creates a **list** of individual words.
- `char.upper()` method turns lowercase character into an uppercase character.
- **Outer comprehension:** It iterates through each `word` in the list of individual words created by `.split()` method.
- **Inner comprehension:** It iterates through each `char` in the `word` and converts the character into uppercase using `.upper()` method. If the `char` is already in uppercase, it does nothing. For each word, the list of its uppercase characters is created.
- The final list is the collection of sublists of uppercase letters of individual words.
- **Step by step evaluation**

Words List: ['I', 'am', 'happy']
word | char | upper |
I | I | I |
Sublist 'I': ['I']
word | char | upper |
am | a | A |
am | m | M |
Sublist 'am': ['A', 'M']
word | char | upper |
happy | h | H |
happy | a | A |
happy | p | P |
happy | p | P |
happy | y | Y |
Sublist 'happy': ['H', 'A', 'P', 'P', 'Y']

KP3. Using Nested List Comprehensions with Conditions

G3.1 (E) Creating a Nested List of Filtered Words

Question

What is the output of the following code snippet?

```
words = [["cat", "elephant"], ["dog", "tiger"], ["fox", "giraffe"]]
long_words = [[word for word in row if len(word) > 3] for row in words]
print(long_words)
```

Solution

```
Output: [['elephant'], ['tiger'], ['giraffe']]
```

Explanation

- `len(word)` method finds the length of the word.
- **Outer comprehension:** It iterates through each sublist (`row`) in the list `words`.
- **Inner comprehension:** It iterates through each `word` in the `row` and selects the word if the length of the word is greater than 3 `len(word)>3`. The selected words form the sublist.
- The final list (`long_words`) is the collection of sublists of the selected words.
- **Step by step evaluation**

  row | word |
  ['cat', 'elephant'] | elephant |
  Sublist:['elephant']
  row | word |
  ['dog', 'tiger'] | tiger |
  Sublist:['tiger']
  row | word |
  ['fox', 'giraffe'] | giraffe |
  Sublist:['giraffe']

- *Note:* In this case, the elements are filtered based on a condition, thus the `if` condition comes after the `for` loop. If each element in the list is modified based on the conditions `(if-else)`, then the conditions should come before the `for` loop

KP4. Writing Code Using Nested List Comprehensions

G4.1 Writing Code for Creating Given Nested List

Question

Write a Python program using a nested list comprehension to create a 3 x 4 grid filled with zeros and print it.

Solution

```
grid = [[0 for _ in range(4)] for _ in range(3)]
print(grid)
```

Explanation

- Recall that Nested List Comprehension consists of list comprehensions within a list comprehension.
- Here, it has been asked to create $3 * 4$ grid, which means there should be 3 rows and 4 columns.
- Remember that matrix is a list of rows, each row is a sublist which contains column values.
- In general, Outer loop is used to create sublists and Inner loop is used to add or modify the values in each sublist.
- Outer comprehension should create 3 rows `[..  for _ in range(3)]`.
- Inner comprehension should insert value 0 to the 4 columns in each row. `[0 for _ in range(4)]`
- *Note:* `_` is just a regular variable, we can add any meaningful variable in that place. It is used to indicate that the loop variable is not relevant. In this case, the loop variable is not used inside the expression, thus `_` is used as a loop variable
- **Step by step evaluation**

row index | column index | row |
0 | 0 | [0] |
0 | 1 | [0, 0] |
0 | 2 | [0, 0, 0] |
0 | 3 | [0, 0, 0, 0] |
Row:[0, 0, 0, 0]
row index | column index | row |
1 | 0 | [0] |
1 | 1 | [0, 0] |
1 | 2 | [0, 0, 0] |
1 | 3 | [0, 0, 0, 0] |
Row:[0, 0, 0, 0]
row index | column index | row |
2 | 0 | [0] |
2 | 1 | [0, 0] |
2 | 2 | [0, 0, 0] |
2 | 3 | [0, 0, 0, 0] |
Row:[0, 0, 0, 0]

G4.2 Writing Code for Creating Given Nested List from Strings

Question

Write a nested list comprehension that extracts only vowels from each word in a sentence, storing them in nested lists. For `sentence = "Python Is Amazing"`, the output must be `[['o'], ['I'], ['A', 'a', 'i']]`

Solution

```
sentence = "Python Is Amazing"
vowels = "aeiouAEIOU"
vowel_list = [[char for char in word if char in vowels] for word in sentence.split()]
print(vowel_list)
```

Explanation

- It has been asked to create a list of lists. Each sublist should contain only the vowels from each word of the given sentence.
- Remember that `sentence.split()` method splits the sentence and creates a **list** of individual words.
- Given sentence is `"Python is amazing"`.
- It is necessary to define the list (or any iterable) of vowels for the comparison.
- Here, it is defined as string `vowels = "aeiouAEIOU"`. Note that, in Python, string is an iterable.
- Recall that the outer loop is used to generate sublists, and inner loop is used to add or modify values in the sublist.
- Here, the number of sublists should be equal to the number of words. Thus, outer comprehension should iterate through the list of individual words and creates an row for each word. `[...  for word in sentence.split()]`
- The inner loop should select characters from each word, if the character is present in the `vowel` string and fill the

rows.

- Remember if it is for **filtering** in the list comprehension, then the `if` condition should come after the `for` loop. `[expression for item in iterable if condition]`
- The inner comprehension should be `[char for char in word if char in vowels]`.
- **Step by step evaluation**

Words List: ['Python', 'Is', 'Amazing']
word | char |
Python | o |
Sublist: ['o']
word | char |
Is | I |
Sublist: ['I']
word | char |
Amazing | A |
Amazing | a |
Amazing | i |
Sublist: ['A', 'a', 'i']

G4.3 Writing Code for Creating Given Nested List with Conditionals

Question

Write a nested list comprehension that creates a 5×5 grid, but fills it with 1 if the sum of row and column indices is even, and 0 otherwise, and print it.

Solution

```
grid = [[1 if (i + j) % 2 == 0 else 0 for j in range(5)] for i in range(5)]
print(grid)
```

Explanation

- Recall that Nested List Comprehension consists of list comprehensions within a list comprehension.
- Here, it has been asked to create $5 * 5$ grid, which means there should be 5 rows and 5 columns.
- Remember that matrix is a list of rows, each row is a sublist which contains column values.
- In general, Outer loop is used to create sublists and Inner loop is used to add or modify the values in each sublist.
- Outer comprehension should create 5 rows `[... for i in range(5)]`.
- Inner comprehension should populate the 5 column values in each row. It should insert 1 if the sum of row and column indices is even, else it should insert 0, in the columns in each row.
- The sum $i + j$ is even, if the remainder is 0 when $i + j$ is divided by 2.
- The modulo operator `%` is used to get the remainder of division of two numbers.
- Thus, the expression would be `(i+j)%2`.
- The condition to check whether the sum is even or not: `1 if (i+j)%2==0 else 0`.
- Remember if it is for **modifying** each value in the list comprehension, then the `if-else` condition should come before the `for` loop. `[expression if condition else expression for item in iterable]`
- Inner comprehension would be `[1 if (i+j)%2==0 else 0 for j in range(5)]`.
- **Step by step evaluation**

row index | column index | expression | row
0 | 0 | 0 | [1] |
0 | 1 | 1 | [1, 0] |
0 | 2 | 0 | [1, 0, 1] |
0 | 3 | 1 | [1, 0, 1, 0] |
0 | 4 | 0 | [1, 0, 1, 0, 1] |
Row:[1, 0, 1, 0, 1]
row index | column index | expression | row
1 | 0 | 1 | [0] |
1 | 1 | 0 | [0, 1] |
1 | 2 | 1 | [0, 1, 0] |
1 | 3 | 0 | [0, 1, 0, 1] |
1 | 4 | 1 | [0, 1, 0, 1, 0] |
Row:[0, 1, 0, 1, 0]
row index | column index | expression | row
2 | 0 | 0 | [1] |
2 | 1 | 1 | [1, 0] |
2 | 2 | 0 | [1, 0, 1] |
2 | 3 | 1 | [1, 0, 1, 0] |
2 | 4 | 0 | [1, 0, 1, 0, 1] |
Row:[1, 0, 1, 0, 1]
row index | column index | expression | row
3 | 0 | 1 | [0] |
3 | 1 | 0 | [0, 1] |
3 | 2 | 1 | [0, 1, 0] |
3 | 3 | 0 | [0, 1, 0, 1] |
3 | 4 | 1 | [0, 1, 0, 1, 0] |
Row:[0, 1, 0, 1, 0]
row index | column index | expression | row
4 | 0 | 0 | [1] |
4 | 1 | 1 | [1, 0] |
4 | 2 | 0 | [1, 0, 1] |
4 | 3 | 1 | [1, 0, 1, 0] |
4 | 4 | 0 | [1, 0, 1, 0, 1] |
Row:[1, 0, 1, 0, 1]