List Comprehension, Nested Lists

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Lists

- Lists contain a list of ordered values (of any type)
- Assignment: L_var = [i1, i2, …]
- Lists are indexed - can use the list or an item (by indexing) or part of the list (by slicing)
- Work on list: Functions (e.g. len(), sum(),...); ops (like +, *); operations on list objects e.g. append(), insert(), remove(), pop(), index(), reverse(), count(), sort() …
- Lists are mutable - i.e. a list object can be changed; so when a list item is changed current list is changed (new list is not created)
- L1 = L2 - the pointer to the list object in L2 is assigned to var L1
- To copy the list, we can use copy() operation

List Comprehension

- In maths you have learned how to construct a set by specifying conditions on the values. E.g.

  S = {x: x=n*(n+1) where 0<n<6} # from CBSE book

  Ans: S = {2, 6, 12, 20, 30}

- Let us form this as a list (can also do set). The statement is:

  ```
  S = [n*(n+1) for n in range(1,6)]
  ```

- This is list comprehension: a natural way to form a list; general syntax:

  ```
  newLst = [ expr for item in iterable] # i.e. list or range()
  newLst = [ expr for elt in iterable if condition ]
  ```

List Comprehension vs for loop

- What list comprehension can do, a for loop can also do, e.g

```
        lst = [n for n in range(10) if n%2==0] # list of even nos
        lst = [elt for elt in given_lst if elt%2==0]
```

- The for loop for this is:

```
        lst = []
        for n in range(10):
            if n%2 == 0:
                lst.append(n)
```

- List comprehension is compact and elegant
- It is also computationally more efficient - takes less computer time

Quiz : Multi Correct

Which of the following is the correct expansion of the following list comprehension:

list_1 = [expr(i) for i in list_0 if func(i)]

A)
```
list_1=[]
for i in range(len(list_0)):
        if func(list_0[i]):
                list_1.append(list_0[i])
```

B)
```
list_1=[]
for i in list_0:
        if func(list_0[i]):
                list_1.append(expr(list_0[i]))
```

C)
```
list_1=[]
for i in range(len(list_0)):
        if func(list_0[i]):
                list_1.append(expr(list_0[i]))
```

D)
```
list_1=[]
for i in list_0:
        if func(i):
                list_1.append(expr(i))
```

Quiz : Multi Correct

Which of the following is the correct expansion of the following list comprehension:

list_1 = [expr(i) for i in list_0 if func(i)]

A)
```
list_1=[]
for i in range(len(list_0)):
        if func(list_0[i]):
                list_1.append(list_0[i])
```

B)
```
list_1=[]
for i in list_0:
        if func(list_0[i]):
                list_1.append(expr(list_0[i]))
```

C)
```
list_1=[]
for i in range(len(list_0)):
        if func(list_0[i]):
                list_1.append(expr(list_0[i]))
```

D)
```
list_1=[]
for i in list_0:
        if func(i):
                list_1.append(expr(i))
```

Solution: Option C and D are correct

List Comprehension Examples

- List of even numbers till a number n

```
even_lst = [i for i in range(n) if i%2 == 0]
```

- List of squares of a list of numbers

```
lst = [5, 7, 18, ...]
sq_lst = [x*x for x in lst]
```

- List of squares of even numbers in a list

```
squares = [x**2 for x in lst if x%2==0]
```

- List of multiples of items in a list

```
c = 5
m = [x*c for x in lst]
```

More List Comprehension Examples

Create a list that contains the elements of the given input list, excluding a given element.

Input: l1 = [2, 3, 3, 5, 7, 3, 4, 3]

Element = 3

Output: res = [2, 5, 7, 4]

```
l1 = [2, 3, 3, 5, 7, 3, 4, 3]


def remove_all(lst, x):
    return [i for i in lst if i != x]


res = remove_all(l1, 3)
print(res)
```

Quiz : Single Correct

What will be output of the following code :

```
list1 = [i % 3 for i in range(0, 10, 2) if i % 3 != 0]
print(list1)
```

A. [1,2,2]
B. [1,2,1]
C. [2,1,2]
D. [2,2,1]

Quiz : List Comprehension

What will be output of the following code :

```
list1 = [i % 3 for i in range(0, 10, 2) if i % 3 != 0]
print(list1)
```

A. [1,2,2]
B. [1,2,1]
C. [2,1,2]
D. [2,2,1]

Explanation : range(0,10,2) generates values 0,2,4,6,8 ; 2,4 and 8 are not divisible by 3 and leave remainders 2,1 and 2 respectively

Examples

- Comprehension also useful for operation on 2 lists also
- E.g. multiply corresponding elements of 2 lists

```
l1 = [1, 2, 3, 4]
l2 = [5, 6, 7, 8]
[l1[i]*l2[i] for i in range(len(l1))]
```

- Common items in two lists l1 and l2
- [elt for elt in l1 if elt in l2]
- Note first in is part of for loop, second is checking membership
- Expression is a regular expression in python - it can use any values accessible at this statement; can call functions in it also

Quiz

- For multipling corresponding elements of 2 lists, we have

      [expr for elt in l1]

- Q: what is expr? (If items in l1 are unique and no duplication)

Quiz

- For multipling corresponding elements of 2 lists, we have

  [expr for elt in l1]
- Ans: elt*l2[l1.index(elt)]

Nested Lists

- We can have list of lists - creates a 2-D list, like a matrix
- l1 = [l11, l12, l13, …] # each of the li is a list of integers
- Then li[i] will return a list
- To access an item: li[i][j] # jth item in the ith list
- This is a matrix
- Higher dimensions are also possible - will not discuss them

2-D Lists

- A 2-D list is a list of lists, i.e. a list, whose items are also lists
- So, a 2D list is just a list - all list operations can be done
- Can create one like this:

```
M = [[1,2,3], [4,5,6], [7,8,9]]
```

- M[0] is the 1st list ([1,2,3]), M[1] is the 2nd etc
- To access individual item, we can access the item of the item
- so for accessing in the ith item, its j th item, we can do: M[i][j]
- M[1][2] is therefore 3rd item of the 2nd list, i.e. 6
- We can replace items, or items of items, just as in list
-
- Can perform functions/ops like len, +, * (how about sum?)

Loop over 2D list

```
# Printing each sub list
M = [[1,2,3], [4,5,6], [7,8,9]]
for row_elt in M:
        print(elt)

for row in range(len(M)):
        print(M[row])
```

```
# Printing each item

M = [[1,2,3], [4,5,6], [7,8,9]]
for row_elt in M:
    for elt in row_elt:
        print(elt, end= " ")
    print("")

for row in range(len(M)):
    for col in range(len(M[row])):
        print(M[row][col], end=" ")
    print("")
```

Creating an empty 2–list

N, M = 3, 4

[var]*N # creates a list by copying object ref N times

[0]*N # creates a 0 list, N long

[[0]*N]*M

# copies 0 N times to create one list

# the outside * will copy the ref of the inner list and replicate it M times; so each row is the same list object

gives an impression of a 2D list; but just gives pointers to the first row, as * copies refs

Lets see in pythontutor

Creating with list comprehension

[0 for i in range(N)] # 1-D

Creating 2-D:

[[0 for i in range(N)] for j in range(M)] #a true 2-D list

Lets see this in pythontutor

Creating 2-D lists of 0s

```
# list comprehension
N, M = 3, 4
lst2 = []
for i in range(N):
    lst2.append([0 for i in
range(M)])
print(lst2)
```

```
lst2 = []
for i in range(N):
    row = []
    for j in range(M):
        row.append(0)
    lst2.append(row)
print(lst2)
```

Inserting/Deleting in 2-D

- Adding a row is easy - just append a row
- Inserting a row in middle also easy: lst2.insert(i,[row])
- Adding/appending a column is a bit harder

```
for i in range(N):
    lst2[i].append(1)
print(lst2)
```

- Deleting a row is easy : del(lst2[i]), lst2.pop(i)
- Deleting a column: Will have to loop and del/pop from each row

An example of working with 2D

A program to create a matrix of size nxn such that diagonals are 0, right of the diagonal is 1 and left (below) the diagonal is -1

Steps:

1. Create M, an empty 2-D list (can use list comprehension)
2. Loop with i over no of rows, and j over no of columns
3. Set value depending on i and j
4. Have print2d function to print matrices in rows and columns

Code

```
def print2d (m):
    for row in m:
        print(row)
    return

# Initializing - of 0s
m = [[0 for i in range(4)] for i in
range(4)]

print2d(m)
```

```
for i in range(len(m)):
    for j in range(len(m[0])):
        if i==j:
            m[i][j] = 0
        elif i < j:
            m[i][j] = 1
        else:
            m[i][j] = -1
print2d(m)
```

Quiz

Q: What is the output of this

```
m = [[1,2,3], [4,5,6],[7,8,9]]
s = 0
for i in range(len(m)):
    for j in range(len(m[0])):
        s = s + m[i][j]
print(s)
```

Quiz

```
m = [[1,2,3], [4,5,6],[7,8,9]]
s = 0
for i in range(len(m)):            # len(m) is the no of rows
    for j in range(len(m[0])):     #len(m[0]) is the no of columns
        s = s + m[i][j]
print(s)
```

Ans: 45 (sum of all elements)

Summary

- List comprehension - elegant and efficient way to create lists
- They are like looping over a list and checking for condition, but in one line - using concept from math/sets
- Use list comprehension wherever you can

- 2-D lists are lists of lists - they represent matrices
- All operations of lists work on the 2D list, and the items on the list
- List comprehension can be used for 2D lists also

- List comprehension requires practice - then you will love it

Announcements

- Next lecture will be online
    - Labs will be available for those who need it (bring earphones)
    - You can also come to LH with your laptop and earphones…, do quizzes together…
- Next lecture we will discuss strings

- Next week is mid-sem exams - so no lectures
- Mid-sem syllabus: everything till next lecture

Practice For You

- Play around with lists on terminal/online
- Create lists - try different operations
- Write the programs given in the lecture (after closing the slide)
- Play around with list comprehension - look for problems on the net (plenty) and then try them without looking at the code
- Work with 2D matrices - create some of different values, change some values based on some row,column property
- Try matrix multiplication of M1 (n1xn2) and M2 (n2xn3) to produce a matrix of size n1xn3. This will require three loops - as for each

Some practice problems for list comprehension

1. Create a list of natural numbers less than 50 that are divisible by 2 and 3
2. Given a list, create a list of all even elements
3. Given two lists of same length, create a list of sum of corresponding elements
4. Given a list of lists, create a list of lists with each list reversed (recall that lst[::-1] returns lst in reverse)
5. Given two lists, create a list of common items in the two lists
6. From a list of lists, form a list of items which have fewer than 3 elts
7. Create a 3x3 list using list comprehension which is [[0,1,2],[3,4,5],[6,7,8]]
8. Transpose a matrix using list comprehension, take m as above