Day 4:

**List comprehensions:**

List comprehensions provide a concise way to create lists. This type of representation is more readable and fast. They are commonly used in places where each element in the list is a result of some operations performed on certain conditions.

Simple eg:

>>>ls = [x\*\*2 for x in range(1,5)]

>>>print(ls)

Output:

[1,4,9,16]

This code can also be written normally as:

>>>ls = list()

>>>for x in range(1,5):

>>> ls.append(x\*\*2)

Output:

[1,4,9,16]

Those three lines are replaced by a single line in python. This makes it more powerful and readable. You can also include if statements inside list comprehension.

Eg:

>>>ls = [x for x in range(1,5) if x%2 == 0] #only even numbers are stored

Output:

[2,4]

What if you have to do some computation with 2d lists that's also possible

And when are you are storing a tuple as input make sure to give parentheses.

Eg:

>>>arr = [[1,2,3] , [4,5,6] , [7,8,9]]

>>>ls = [col for row in arr for col in row] #traversing through 2d array

Output:

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

>>>arr1 = [1,2,3]

>>>arr2 = [4,5,6]

>>>ls = [ x,y for x in arr1 for y in arr2]

|\_\_\_\_\_*missing ()*

The above code gives a traceback error because we haven't specified the parenthesis present in tuples.

You can also give a method inside list comprehensions. In this example below we convert not proper formatted text to formatted one

Eg:

>>>eng = ["aLphA", "BeTa", "gAMma"]

>>>ls = [x.lower() for x in eng]

We can also have nested list comprehension. Let's make our knowledge more clear with the below example.

>>>matrix = [\

[1,2,3]\

[4,5,6]\

[7,8,9]\

]

>>>transpose = [[row[col] for row in matrix] for col in range(len(matrix[0]))]

The above code can also be represented as:

>>>transpose = []

>>>for col in range(len(matrix[0])):

>>> tmp = []

>>> for row in matrix:

>>> tmp.append(row[col])

>>> transpose.append(tmp)

In nested list comprehension the outer for is executed first and inner is executed after the first for.

You can also include if-else syntax in list comprehensions for example

>>>ls = [1,-2,3,-4,-5,6]

>>>ls = [number if number > 0 else 0 for number in ls] *#if number is less than 0 it is considered 0*

Similar to list comprehensions we also have dictionary comprehensions. The additional requirement is we also need to define keys.

>>>dt = {letter:idx for idx,letter in enumerate("manners")}

Output: {'m': 0, 'a': 1, 'n': 3, 'e': 4, 'r': 5, 's': 6}

Here the letter becomes the key while the indices become the value, also note the curly brackets which are used in dictionary. The enumerate function here returns both index and letter in the string.

**Lambda functions:': 6}**

These are called as small functions in python much like representing a function in a single line. They are easy to use, but they are restricted to a single expression only.

Eg:

>>>dividebytwo = lambda x : x//2

>>>print(dividebytwo(10)) #function call

The same function which we represent normally:

>>>def dividebytwo(x):

>>> return x//2

lambda - keyword specifying it's a lambda function

lambda x - this x specifies the parameter, there can be any number of parameters

: x+2 - this expression after the colon specifies the logic we need *(there can only be one expression not more than that)*

Lambda functions also provide readability since we represent functions in one line. It's not mandatory to use lambda functions but if you encounter this type of syntax you can easily find what it does.

More examples of lambda functions:

>>>formula = lambda a,b: a\*\*2 + b\*\*2 + 2\*a\*b #computes (a+b)^2 formula

>>>print(formula(2,3))

o/p : 25

>>>formatName = lambda firstname,lastname : "Dear "+lastname+" "+firstname+","

>>>print(formatName("Elon","Musk"))

o/p : Dear Musk Elon,

You can also pass a function as parameter to lambda functions

>>>add = lambda x,func : x + func(x)

>>>print(add(2,lambda x : x + 1))

**Map:**

Map basically does some operations on each of the iterable variable. So if you want to perform some computations on list you can use a map. The map function takes in a function as the first variable and the iterable as second and the rest if you want more variables .

Let's look at an example for better understanding

>>>def minimum(a,b,c):

>>> if a < b and a < c:

>>> return "store1: " + str(a)

>>> elif b < a and b < c:

>>> return "store2: " + str(b)

>>> return "store3: " + str(c)

>>>store1 = [10,30,15,60]

>>>store2 = [90,5,4,19]

>>>store3 = [16,25,36,49]

>>>ls = list(map(minimum,store1,store2,store3))

The minimum method takes in three arguments and checks the store with minimum cost. As mentioned notice the first parameter is a function in map, while the other parameters are iterables ( the different lists ).

The parameters after the function parameter in map should match with the parameters in the function (in our case def minimum)

This map function operates like looping the function but in an easier way. It operates till the shortest iterable is exhausted in our case we had three lists with same length (4). So if any one list had a length of three which is smaller than rest it will loop three times (shortest iterable).

**Reduce:**

As the name states reduce is a function in python which reduces a group of numbers to a single value. The first parameter in reduce() always takes in a function with two arguments and the second parameter takes a iterable like list.

Eg:

>>>def add(x,y):

>>> return x+y

>>>tot = reduce(add,store1)

As said the add function has two parameters , this code just sums up the entire list and gives the output.

A simple factorial program using reduce:

#n! -> n\*n-1\*n-2\*...\*1

>>>def factorial(x,y):

>>> return x\*y

>>>n = int(input())

>>>ls = range(1,n+1) #generating list of 1..n

>>>fact = reduce(factorial,ls)

**Filter:**

The filter function is used filter the elements inside the list. This also has a syntax similar to reduce. The first parameter of filter parameter is a function and the second is a iterable generally a list.

The filter function generally returns an object, you can store it in another list to view the filtered elements.

Eg:

>>>ls = range(1,11)

>>>odd\_elements = list(filter(lambda x : x%2,ls))

The first parameter here is a lambda function and second is a list(iterable)

This code returns the odd elements from 1-10. Function can also be a normal function ,it is not mandatory to be lambda.

You can infer that filter reduces the list size due to some filtering been done with the elements inside the list.

**Big O notations:**

The measure of how good and efficient your program is done by means of big O notations. You might think ,well I may have a good computer and the other person might have an old computer which is probably slower than this , so my program is faster . Well No! it's doesn't work like that.

Let's get more concrete by seeing the simple example:

Consider a string of length n, now you want to find the number of characters in the given string, we'll do something like this,

>>>cnt=0 #counter for counting the number of characters

>>>for letter in word:

>>> cnt+=1

We can say that the above program has a time complexity of O(n) . Why? Because we need to traverse each element in the string which has length of n so we represent it as O(n).

O(n) - is called Linear time complexity i.e completely depends on size of string so n can vary from 0 <= n <= 10^6, so our program tends to get slower as n increases.

Earlier we saw constant time access in lists and dictionaries well that means the time complexity doesn't care about the input size , no matter what it is we can always obtain elements in constant time giving us the notation of O(1).

If we represent O() (times) then it means the worst case possible, while Ω represents the best case scenario.

Next we'll cover O(n^2) , so this definitely has a bad run time when compared that we discussed since it is n \* n. For example let's take bubble sort

>>>ls = [ random.randrange(1,100) for i in range(10)] #generating random numbers in list

>>>for i in range(len(ls)-1,-1,-1):

>>> for j in range(i):

>>> if ls[j] > ls[j+1]:

>>> ls[j] , ls[j+1] = ls[j+1],ls[j]

The first for loop takes O(n) time and the inner loop also takes O(n) time so as whole we take about O(n^2). O(n^2) might be faster for smaller inputs and difference between O(n^2) and O(n) are miniscule for smaller inputs , but as inputs scale larger O(n^2) becomes pretty bad.

Now we'll look at a much more efficient time complexity than those we saw above which is O(log n). What does log n mean? Here n denotes the size of our inputs and we apply log to n.

Eg:

if n=5 then math.log2(n) = 2.321

if n is doubled to 10 then math.log2(10) = 3.321

You can see that even if we double the amount there isn't much of a difference due to the presence of log operation.

A simple program which has O(log n) complexity is binary search.

>>>ls = [2,3,4,5,6,1,8,9,10]

>>>ls.sort() #notice the sorting operation which is reqd for binary search

>>>element = int(input("Enter element to find: "))

>>>low,high = 0,len(ls)-1

>>>while low<=high:

>>> mid = low + high // 2

>>> if ls[mid] == element:

>>> print("Element found at: ",mid)

>>> break

>>> if element < ls[mid]:

>>> high = mid - 1

>>> elif element > ls[mid]:

>>> low = mid + 1

In the above while loop you can see that at every iteration we halve the input this accounts to log n complexity. This is considered as one of the efficient ways to find/search an element inside a list.

If you want a cheat sheet refer this : [https://www.bigocheatsheet.com](https://www.bigocheatsheet.com/)

Additional video resource for better understanding: <https://youtu.be/v4cd1O4zkGw>

**Assignments:**

Given a 2d array/list , write a python program using list comprehension to sort each row inside the 2d array/list.

i/p : [[3,1,2] , [5,4,6] , [9,8,7]] o/p : [[1,2,3],[4,5,6],[7,8,9]]

Given a list , write a python program using filter to extract words which are not conjunctions. For simplicity words that have length less than or equal to 3 are assumed as conjunctions.

i/p: ["bought","to","wash","a","an","school","at","and","but","master"]

o/p: ["bought","wash","school","master"]

Write a lambda function which creates a default signature format and takes two inputs -> salutation and name

i/p: o/p: *(use single lambda function)*

Best Regards Please do not reply to this mail

Harry Best Regards

Harry

Write a lambda function which returns the derivative of this expression 1/χ with respect to χ. Value of χ will be given.

i/p: 12 o/p: -0.006944444

Given a list of strings having some names write a python program using map to return a list of encrypted strings . Here for encryption we use "#" symbol.

i/p: ["Kratos","Odin","Thor"] o/p: ['K#r#a#t#o#s','O#d#i#n','T#h#o#r']

Given this program find the time complexity of it.

>>>for i range(x):

>>> for j in range(y):

>>> print(i,j)