Python – Part5 – List Comprehension, Iterators, Maps

List comprehension allows us to create new data in a list from scratch, or by modifying an existing list (or an iterable). List comprehension has three main parts to it:

dlist = [output expression possible set filter]

The possible set usually involves a loop over some data e.g., for x in range(1,10), or for x in dlist. The output expression can be something like x+5, or x**2, or call to a function that returns a value e.g., f1(x). The output expression can be an if then else to allow us to do substitutions on the data items selected from the possible set and the filter. Note that the if statement has a different layout when used in the output expression in a list comprehension. It appears as:

yy if condition else zz

This basically means, output yy if the condition is true, otherwise output zz for each data item selected from the possible set after applying the filter to it.

The filter is a simple if statement without an else. If the if statement returns true, that data is kept in the list from the possible set, otherwise, it is removed. Note that filter is optional.

To practice some of the concepts with list comprehension, create a Python application project called "ListComprehension". Add a class to the project called Employee with the following code in it.

```
class Employee(object):
    def __init__(self,fnm,lnm,id,hrsw,pr):  # constructor - purpose is to initialize
data
        self.firstname = fnm
        self.lastname = lnm
        self.id = id
        self.hours_worked = hrsw
        self.pay_rate = pr

    def compute_pay(self, overtime_rate):
        if self.hours_worked <= 40:
            pay = self.hours_worked * self.pay_rate
        else:
            pay =  40*self.pay_rate + \
                (self.hours_worked - 40)*self.pay_rate*overtime_rate
        return pay

    def __repr__(self):
        return '{' + self.firstname + ', ' + self.lastname + ', ' +
str(self.pay_rate) + '}'
```

In the ListComprehension.pay, type the following code upto each print statement, and run it to see if the output makes sense according to the code written.

```
import sys
import random
from Employee import Employee

def compute_pay(emp):
    if emp.hours_worked <= 40:
        pay = emp.hours_worked * emp.pay_rate
    else:
        pay =  40*emp.pay_rate + (emp.hours_worked - 40)*emp.pay_rate*1.5
    return pay

def main():
    fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
    # create a list of fruits with character a in them
    # without list comprehension - normal loop approach
    second_list = []
    for x in fruits:
      if "a" in x:
        second_list.append(x)
    print(second_list)

    # ----list comprehension approach----------------
    lista = [x for x in fruits if "a" in x]
    print(lista)

    # convert fruits to upper case
    listu = [x.upper() for x in fruits ]
    print(listu)

    listh = [x for x in fruits if "a" in x]
    print(listh)

    # remove apple from the list
    list2 = [x for x in fruits if x != "apple"]
    print(list2)

    # copy fruits list to another list via list comprehension
    list3 = [x for x in fruits]
    print(list3)

    # create a list of numbers from 1 to 4
    list4 = [x for x in range(5)]
    print(list4)

    # create a list of numbers from 0 to 4 by filtering from 0-9
    list5 = [x for x in range(10) if x < 5]
    print(list5)

    list6 = ['yyy' for x in fruits]
    print(list6)

    # replace banana with orange in the fruits list
    list7 = [x if x != "banana" else "orange" for x in fruits]
    print(list7)
```

```
# replace banana with yyy in fruits list
list8 = ['yyy' if x == 'banana' else x for x in fruits]
print(list8)

fruits.sort()
print(fruits)

# create list of 10 random numbers from 0-100
list9 = [int(random.random()*100) for x in range(10)]
list9.sort(reverse=True)  # descending sort
print(list9)

list9.reverse()
print(list9)

list9a = [x for x in range(0,10)]

list9a_square_odd = [x**2 for x in list9a if x%2 ==0]
print(list9a_square_odd)

# if number is < 5, take the square, otherwise take the cube of odd numbers
list9b = [x**2 if x < 5 else x**3 for x in list9a if x%2 ==1]
print(list9b)

#-----------copy lists-------------
list10 = list9#.copy()  # test with and without copy
list9[2] = 'yyy'
print(list10)
listA=[5,10,15]
listB = [50,500,60]
listC = listA + listB  # creates a new list, where as extend does in place
print(listC)

# create a lst of random numbers between 10 and 30
rlist2 = [random.random()*20+10 for x in range(10)]
print(rlist2)

e1 = Employee('Bill','Baker',1234,45,20)
e2 = Employee('Sally','Simpson',1234,50,30)
e3 = Employee('John','Jacobs',12345,60,15)
elist =[e1,e2,e3]
paylist = [compute_pay(x) for x in elist]
print(paylist)

# pay by calling the compute_pay in the Employee class
paylist2 = [x.compute_pay(1.5) for x in elist]
print(paylist2)

# sort list by lastname
elist.sort(key=lambda x: x.lastname)
print(elist)

# sort list by pay_rate in reverse order
elist.sort(key=lambda x: x.pay_rate, reverse=True)
print(elist)
```

```
if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Iterators:

In Python, the looping capability is provided by the “iterator” concept. The iterator supports requires the class representing the data to provide two special functions – one is __iter__ and the other is __next__. The __iter__ function resets the position to point to the start of the data, while the __next__ returns the data at the current position, and moves the position to point to the next data item. If a class supports the iterator, then a simple for loop can be written as:

```
for  x in classobect:
  # use x
```

The fundamental data types of list, tuple, set, and dictionary already have the iterator support built into them. If you create your own data structure, and you wanted to provide a looping capability over the data, then you should implement the __iter__ and the _next__ methods.

To clarify the iterator concept, create a project called IteratorTest. We will create a class called company which will contain a list of products. The idea is to provide looping capability over the company to access the products.

Add a class called Product to the project with the following code in it.

```
class Product(object):
    def __init__(self, pid, pname, price):
        self.pid = pid
        self.pname = pname
        self.price = price

    def __str__(self):
        return str(self.pid) + " " + self.pname + " " + str(self.price)
```

Add a class called Company to it with the following code:

```
class Company(object):
    # will contain a list of products
    # we will provide an iterator for the products
    # To create an object/class as an iterator you have to implement the methods
    #  __iter__() and __next__() in your class.

    def __init__(self):
        self.company_name = ""
        self.plist = []
        self.index = 0

    def add_product(self, prod):
        self.plist.append(prod)
```

```
    def remove_product(self,pid):
        index = -1
        for i in range(0,len(self.plist)):
            if self.plist[i].pid == pid:
                index = i
                break
        if index >= 0:
            self.plist.pop(index)

    def __iter__(self):
        self.index = 0
        return self

    def __next__(self):
        if self.index < len(self.plist):
            pr = self.plist[self.index]
            self.index = self.index + 1
        else:
            raise StopIteration # try commenting it
            pass
        return pr
```

In the IteratorTest file, type the following code (test upto each print by running the program before typing the next section):

```
import sys
from Company import Company
from Product import Product

def main():
    mytuple = ("apple", "banana", "cherry")
    myit = iter(mytuple)
    print(next(myit))
    print(next(myit))
    print(next(myit))

    # strings are sequence type, so also rovide an iterator
    mystr = "hello"
    myit = iter(mystr)
    print(next(myit))
    print(next(myit))
    print(next(myit))
    print(next(myit))
    print(next(myit))

    # for loop with ietrator supported types
    mytuple = ("apple", "banana", "cherry")

    for x in mytuple:
      print(x)

    class MyNumbers:
      def __iter__(self):
        self.a = 1
```

```
        return self

      def __next__(self):
        if self.a <= 20:
          x = self.a
          self.a += 1
          return x
        else:
          raise StopIteration

    myclass = MyNumbers()
    myiter = iter(myclass)

    for x in myiter:
      print(x)

    # To create an object/class as an iterator you have to implement the
    # methods __iter__() and __next__() to your object.
    comp = Company()
    comp.company_name = "ABC Corp"
    pr1 = Product(100,"Laptop", 750.50)
    comp.add_product(pr1)
    pr2 = Product(101,"Calculator", 55.50)
    comp.add_product(pr2)
    pr3 = Product(102,"Watch", 50.50)
    comp.add_product(pr3)
    pr4 = Product(102,"Cell Phone", 350.50)
    comp.add_product(pr4)
    comp.remove_product(101) # pid

    iterp = iter(comp)
    print(next(iterp))
    print(next(iterp))
    print(next(iterp))

    for x in comp:
        print(x)


if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Map, Filter, Reduce:

Map allows us to create lists by iterating over another sequence type. In this respect, it is a more powerful version of the list comprehension. The general form of the map is (keep in mind, map is a function).

res = map(function name, or lambda to invoke for each element, iterator1, iterator2,,.. iterator n)

The result of a map call is a map object, that can be converted to a list by invoking the list constructor, as:

dlist = list(map(function, iterator))

To provide, filtering of the data, filter function is provided. The general form of the filter call is:

res = filter(function name or lambda that returns true or false, iterator)

The reduce further allows summarization on the results produced by the filter and map. Note that, we can chain filters and maps i.e., a map can operate on the result of another map, or another filter.

Create a project called MapTest. Type the following code in it. Type up to each print, and then test the output by running the program.

```
import sys
import math
import functools
import operator
import re   # regular expressions

def po2(x):
    return 2**x

def convert_to_float(x):
    try:
        return float(x)
    except ValueError:
        return float("nan")

def remove_punctuation(word):
    return re.sub(r'[\'!?."()-:;]',"",word)

def rotate_char_right(c): # rotates a char by 3
    rotate_by = 3
    c = c.lower()  # convert to lower case
    alphabet = 'abcdefghijklmnopqrstuvwxyz'  # string is a sequence type
    if not c in alphabet:
        return c
    code = ord(c) + rotate_by  # ord returns ASCII code for a char
    if code > ord(alphabet[-1]):
        code = code - len(alphabet)
    return chr(code)

def rotate_char_left(c): # rotates a char left by 3
    rotate_by = 3
    c = c.lower()  # convert to lower case
    alphabet = 'abcdefghijklmnopqrstuvwxyz'  # string is a sequence type
    if not c in alphabet:
        return c
    code = ord(c) - rotate_by  # ord returns ASCII code for a char
    if code < ord(alphabet[0]):
        code = code + len(alphabet)
    return chr(code)

def mysqrt(num):
    try:
        return math.sqrt(num)
    except ValueError:
        return float("nan")
```

```
def c_to_f(c):   # c/5 = (f-32)/9
    f = c * (9/5) + 32
    return f

def f_to_c(f):
    c = ((f-32)/9)*5
    return c

def unique_words(sentence):  # 'hello there how are you, see you soon'
    wlist = sentence.split(' ')
    return set(wlist)

def main():
    s1 = 'hello there how are you see you there soon'
    s2 = unique_words(s1)
    print(s2)

    # take the square of even numbers from 1-10
    d1 = [x for x in range(1,10)]
    print(d1)

    # list comprehension to produce square of even numbers in d1
    d2 = [x**2 for x in d1 if x%2==0]
    print(d2)

    # filter is a global function in Python
    # filter(function or lambda, iterable)
    d3 = list(filter(lambda x:x%2==0,d1))
    print(d3)
    d4 = list(map(lambda x:x**2,d3))
    print(d4)

    d5 = list(map(lambda x:x**2, filter(lambda x:x%2==0, d1)))
    print(d5)

    numlist = [24, 16, 36, -25, 65, 72, -50,  81]
    # goal is to compute the square root on each element
    #sqlist = list(map(math.sqrt, numlist))  # error if list has negative nums
    #print(sqlist)

    sqlist = list(map(mysqrt,numlist))
    print(sqlist)

    sqlist2 = list(map(math.sqrt, filter(lambda x:x>=0, numlist)))
    print(sqlist2)

    clist = [20, 15, 25, 35, 19.5, 32, 100, 0]
    tempf = list(map(lambda x: round(x,2), map(c_to_f, clist)))
    print(tempf)

    # convert tempf to tempc i.e., in centigrades
    tempc = list(map(lambda x: round(x,2),map(f_to_c,tempf)))
    print(tempc)
```

```
# reduce - functools and operator imports are needed
# goal - count words in a list of sequences,
# prepare list of unique words in a list of sentences
slist = ['hello there how are you',
         'good morninig hope you are having a good day',
         'python has good data constructs',
         'how good is python',
         'you will find out']

ulist = list(map(unique_words,slist))
print(ulist)

ulist2 = functools.reduce(operator.or_, map(unique_words,slist))
print(ulist2)
print('count of unique words=',len(ulist2))

# count all words in slist
wcount = functools.reduce(operator.add,map(lambda x:len(x.split()),slist))
print('total words=', wcount)

c1 = 'y'
r1 = rotate_char_right(c1)
print('rotated char=', r1)

s1 = 'hello there how are YOU?'
#enc = list(map(rotate_char_right, s1))
enc = "".join(map(rotate_char_right, s1))  # map is a sequence
print(enc) # "".join can be used to convert sequence to a string

dec = "".join(map(rotate_char_left,enc))
print(dec)

# how do we convert a sequence to a string?
# use join function on a string
d1 = ['apples', 'oranges', 'kiwi', 'plums']
res = "".join(d1)
print(res)

# map(function or lambda, iterable)
# function in a map can be a built-in function, e.g., len, abs, sqrt,..
# or it can be a user defined function

dlist = [5, 3, 6, 4, 8]
res = map(po2, dlist) # map returns a map object, not a list
res2 = list(map(po2, dlist))
print(res2)

# using list comprehension
res3 = [2**x for x in dlist]
print(res3)

slist = ['8', '12', '15', '15', '4', '13']
#dc = [int(x) for x in slist]  runs into an error
#print(dc)

dc2 = list(map(int, slist))
print(dc2)
```

```
    slist2 = ['7.25', '8.1', 'hi', '23.76', '85.67']
    #m2 = list(map(float, slist2))  runs into an error
    m2 = list(map(convert_to_float,slist2))
    print(m2)

    numlist = [25, -8, 32, 65, -56, -87, 20]
    numabs = list(map(abs, numlist))  # abs built-in function
    print(numabs)

    # for string processing, global fuctions  str.lower(), str.capitalize(),
    # str.strip()   to remove leading and trailing spaces
    wlist = ['hello', 'artificial', 'intelligence', 'neural', 'network']

    caplist = list(map(str.capitalize, wlist))
    print(caplist)

    ulist = list(map(str.upper, wlist))
    print(ulist)

    wlist2 = ['hello', '  greeting ', 'hey there    ', ' good morning  ']
    # goal is to remove leading, trailing spaces, and to capitalize
    plist = list(map(lambda x:str.capitalize(str.strip(x)), wlist2))
    print(plist)

    f1 = lambda x,y: math.sqrt(x**2 + y**2)
    res = f1(5,7)
    print('res from lambda=', res)

    para = """
    Artificial Inteligence is becoming very "popular".
    It has many branches including "NLP'", "Reinforcement Learning".
    Can it solve all problems?  Perhaps!
    """
    wordlist = para.split()
    print(wordlist)
    clist = list(map(remove_punctuation,wordlist))
    print(clist)

    dx = [8, 9, 6, 4, 5, 3]
    dy = [12, 15, 20, 30, 40]
    # sqrt(x^2 + y^2)
    vlist = list(map(lambda x,y: math.sqrt(x**2+y**2), dx, dy))
    print(vlist)


if __name__ == "__main__":
    sys.exit(int(main() or 0))
```