IE-509
Week 9: Advanced Python Functions

Agenda: 1. List comprehension
2. Split/strip stings
3. Csv file handling
4. Recursive functions

Chapter 1: List comprehension

List comprehension is a compact way of defining lists, in a manner similar to how mathematicians do it. We define sets as

$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$S = \{x^2 : x \in X\}$

$P = \{2^i : i \in -6, -5, \cdots, 0, 1, \cdots, 5, 6\}$

$T = \{x : x \in S, \ is\ even\}$

We can define the above sets as lists in the following way.

```
X = [1,2,3,4,5,6,7,8,9]
S = []
for x in X:
 S.append(x**2)
print(S)
```

Using **list comprehension**, we can define both the above lists in a concise form:

```
Xnew = [x for x in range(1,10)]
Snew = [s**2 for s in Xnew]
print(Xnew)
print(Snew)
```

Let us define $P$and Tusing list comprehension.

```
P = [2**x for x in range(-6,7)]
T = [x for x in Snew if x%2 == 0]
print(P)
```

```
print(T)
```

Printing multiples of 3 in range 3 to 100.

```
[y for y in range(3, 100, 3)]
```

We can do complicated examples using nested lists. Let us generate the list of prime numbers between 2 and 50 using two steps.

1. Build a list containing numbers that are multiples of 2, multiples of 3, multiples of 5, multiples of 6 and multiples of 7.
2. List the numbers that are not in the above list which will give us prime numbers between 2 and 50.

```
nonprimes = [x for y in range(2,8) for x in range(2*y,
50, y)]
primes=[i for i in range(2, 50) if i not in nonprimes]
print(primes)
```

We can use list comprehension on lists too.

```
s1 = 'abc'
s2 = 'xyz'
[(x,y) for x in s1 for y in s2]
```

And on lists of strings.

```
str1 = ['Mumbai', 'Pune', 'Delhi', 'Indore']
str2 = [x for x in str1 if len(x)>4]
print(str2)
```

Chapter 2: String operations: split() and strip()

Let us take the string

```
mystr = 'The quick brown fox jumps over the lazy dog'
```

We can do the following to create a list of words, where each list-element is one word.

```
strlist=mystr.split()
print(strlist)
stuff = [[w, w.upper(), len(w)] for w in strlist]
for j in stuff:
 print(j)
```

The split() function can be used to separate a string using a given separator. A few examples.

```
mystr='line1ABC\n line2MNOPQR\n line3XYZ\n'
print(mystr.split())
print(mystr.split(' '))
print(mystr.split('\n'))
print(mystr.split('line'))
print(mystr.split('n'))
print(mystr.split('j'))
```

In the above examples, a list of strings is returned where the components are the result of what is obtained by separating the original string using the given argument. In the last example, the character ‘j’ is not present in the original string, so a list with one element is returned and this element is the original string itself.

The strip() function is used to remove ceratin characters from a string. A few examples.

```
mystr='   82828line1 ABC28282   '
print(mystr.strip())  # Removes all spaces from the beginning and the end
print(mystr.lstrip()) # Removes all spaces in the beginning
print(mystr.rstrip()) # Removes all spaces at the end
print('88888444332228888'.strip('8')) # Removes 8 from both ends
```

Chapter 3: CSV file handling

The file scores.csv has the marks of n students in a quiz. We want to read this file in Python and compute the mean, max and min of each quiz and write them back at the end of the files.

CSV stands for comma-separated values where the values are separated by a comma. It is a popular way of saving data.

```
from numpy import mean
fin = open('scores.csv', 'r+')
fin.readline() # read the header line
lines = fin.readlines() # read the rest of the file
exam1 = []
exam2 = []
for line in lines:
 mylist = line.split(',')
 exam1.append(float(mylist[1]))
 exam2.append(float(mylist[2]))
fin.write('Min, '+ str(min(exam1)) + ', '+ str(min(exam2))+ '\n')
fin.write('Max, '+ str(max(exam1)) + ', '+ str(max(exam2))+ '\n')
fin.write('Mean, '+ str(mean(exam1)) + ', '+ str(mean(exam2))+ '\n')
fin.close()
```

Chapter 4: Recursive functions

Recursion is a method of programming or coding a problem, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition satisfies the condition of recursion, we call this function a recursive function.

Termination condition: A recursive function has to fulfill an important condition to be used in a program: it has to terminate. A recursive function terminates, if with every recursive call the solution of the problem is downsized and moves towards a base case. A base case is a case, where the problem can be solved without further recursion. A recursion can end up in an infinite loop if the base case is not met in the calls.

Example:
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1

Replacing the calculated values gives us the following expression
4! = 4 * 3 * 2 * 1
In other words, recursion in computer science is a method where the solution to a problem is based on solving smaller instances of the same problem.

Now we come to implement the factorial in Python. It's as easy and elegant as the mathematical definition.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

We can track how the function works by adding two print() functions to the previous function definition:

```
def factorial(n):
    print("factorial has been called with n = " + str(n))
    if n == 1:
        return 1
    else:
        res = n * factorial(n-1)
        print("intermediate result for ", n, " * factorial("
,n-1, "): ",res)
        return res
print(factorial(5))
```

It is common practice to extend the factorial function for 0 as an argument. It makes sense to define 0! to be 1 because there is exactly one permutation of zero objects, i.e. if nothing is to permute, "everything" is left in place. Another reason is that the number of ways to choose n elements among a set of n is calculated as n! divided by the product of n! and 0!.

All we have to do to implement this is to change the condition of the if statement:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

We can use recursion to generate Fibonacci sequences in the following way.

```
def fib(n):
    if n == 0:
```

```
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
print(fib(20))
```

Recursive functions use something called “the call stack.” When a program calls a function, that function goes on top of the call stack. This is similar to a stack of books. You add things one at a time. Then, when you are ready to take something off, you always take off the top item. If the termination criteria is not properly specified or the call stack runs out of memory, an error will be displayed. We demonstrate this using the following simple function.

```
def overflow_function(n):
 return overflow_function(n+1)
overflow_function(1)
```

This results in the following error.

```
      in overflow_function(n)
            1 def overflow_function(n):
      ----> 2   return overflow_function(n+1)
            3 overflow_function(1)

      RecursionError: maximum recursion depth exceeded
```

Submission Exercises

Exercises E1, E2, E3, E4 and E5 and their respective subparts are for submission. Create one Python notebook for submission. Try to answer each exercise in a single separate cell. The output csv files in E3 and E4 should not be submitted. However, the code should be able to generate the csv file as asked when executed.

E1. Using list comprehension, define the following lists:

- Set of all odd numbers between 1 and 100
- First 13 multiples of 7
- Consider 50 equally spaced values in the given interval. Numpy may be used.
- The number of occurrences of eight events is 8, 7, 11, 23, 45, 6, 12, 17. Find the probability of each event.

E2. Write a function `get_month(date_list)` that takes a list of dates `date_list` as an input argument in dd-mm-yyyy format. The function should return the list of months for each date in the list. For example `get_month(['06-08-2003', '09-12-2015'])` should return `['August', 'December']`

E3. Read scores2.csv which includes marks of 4 tests. The weightage of Test1, Test2, Test3 and Test4 are 15%, 15%, 30% and 40% respectively. Write a program to compute

- The weighted score of each student
- Summary states min, max, mean for each exam

Write these results along with the given marks in scores2.csv to a new file score2_new.csv

E4. Write a program using list comprehension that calculates and prints the values according to the given formula $Q = \sqrt{\frac{2CD}{H}}$, where $C = 50$ and

$H = 30$ are fixed values. $D$is the variable whose values should be the input of your program in a csv file. Using these values compute Q and write the output in a file q_values.csv
Ensure that the values written in the file are rounded off to 2 digits.

E5. A pascal triangle with 6 rows is given as follows.

```
                    1

                  1   1

                1   2   1

              1   3   3   1

            1   4   6   4   1

          1   5   10  10  5   1
```

Write a function `pascal(n)` which returns as a list row number `n` of a Pascal triangle where `n` is a natural number. For example, `pascal(5)` should return `[1, 4, 6, 4, 1]`. Use this function to generate a pascal triangle with 4 rows,