Introduction to Python 3.8:Numerical manipulations

❖ Get integers from 0 to n-1: range(n)

```
print(list(range(5)); # produces a list of 5 integers from 0 to 4
```

❖ Get integers from p to q-1: range(p,q,step)

```
print(list(range(0,11,2))); # produces the following list of even numbers [0,2,4,6,8,10]
```

❖ Generate a set of integers using the **range()** command

```
print(list(range(m,n,k))) # generates integers from m to n-1 in intervals of k
print(list(range (2,10,2))) # generates even numbers between 2 to 9
print(list(range (10,2,-2)) # generates even numbers from 10 to 4 in reverse order
print(list(range (-10,-1,2)) # generates [-10, -8, -6, -4, -2]
print(list(range (2*3)) # generates the list of 6 integers from 0 to 5
```

Introduction to Python 3.8: Lists

- A list can contain words as well as numbers. A list is *mutable* i.e. it can be altered by adding or removing elements to and from the list

```
mylist = [ ‘Hello’, 777 , 2.27, ‘Virat', 70.2, 85, ‘Kohli’ ]
```

- nth item in list: list[n-1] # Examples below

```
print (mylist[0]) # prints first element in the list i.e. hello
```

- m^{th} to n^{th} item in the list: mylist[m-1:n]

```
print (mylist[0:3]) # prints the first 3 items; list index numbering starts from 0
```

- m^{th} to n^{th} items in the list in steps of p: mylist[m-1:n:p]

```
print (mylist[0:7:3]) # prints [‘Hello’, ‘Virat’, ‘Kohli’]
print (mylist[::3]) # has the same effect as the above statement
```

- List items starting from the (m+1) th element: mylist[m:]

```
print (mylist[2:]) # prints [2.27, 'Virat', 70.2, 85.7, 'Kohli']
```

Introduction to Python 3.8: Strings

❖ String is a list: str = "Hello!“ # Each letter in the string is an element of a matrix

❖nth item in list: str[n-1] # Examples

print (str[0]) #prints the first character of the string i.e. H

print (str[5]) #prints the sixth character of the string i.e. !

print (str[6]) #confuses the computer ☹ resulting in the following output

IndexError: string index out of range

❖m^{th} to n^{th} items in list: str[m-1:n] # Note: smallest value of m is 1

print (str[0:4]) # Hell breaks loose

print (str[:4]) # has same effect as above command; default value before : is set to # first element position i.e. 0

❖Last item (first in reverse direction): str[-1] # prints !

❖nth item from the end: str[-n] # Example: print str[-5] prints e

❖mth to nth items from the end: str[-m:-(n+1):-1] # Example: print str[-2:-6:-1] prints olle

Introduction to Python 3.8: Strings and Lists

- Arithmetic operations with a string

  print (str*2) # prints “Hello!Hello!”

- Strings can be joined together (concatenated)

  print (str) + “ 20MS-batch” # prints Hello! 20 MS-batch

- Algebra with Lists

  X=[1,2,3,4]

  Y=[‘a’, ‘b’]

  2*X+3*Y # gives the following extended list

  [1,2,3,4,1,2,3,4,’a’,’b’,’a’,’b’,’a’,’b’]

- To get the length (number of items) in a list: len(str) # returns 6

- Comparison, Identity and membership

  (==, not, in, <, >, != (➔not equal to), <=, >=, is, is not)

Mathematical Operations on a List of numbers

- *Sum and Average*

```
numlist = [3, 2.5, 4.0, -1, 5, 0]
print(len(numlist)) # gives the number of elements in the list; 6 in this case
print(sum(numlist)) # gives the sum of all the elements in the list
mean=sum(numlist)/len(numlist) # calculates the average, 2.25 in this example
```

- Slicing of a list i.e. removing elements from a list to generate a smaller list

```
numlist[m,n] # creates a list starting from the (m+1)'th to n'th element
numlist[1:4]  # generates the list [2.5,4.0,-1]
numlist[0:5]  # generates the list [3,2.5,4.0,-1,5]
numlist[:4]  # generates the list [3,2.5,4.0,-1]
```

- Location of an element in the list

```
numlist.index(2.5) # gives the location of the element 2.5 in the list. Output is 1
                   #NOTE: Counting of the location starts from 0; location of the first
                    # element is 0, second element is 1 and so on
```

Mathematical Operations on a List of numbers

- Maximum *and* Minimum

```
numlist = [3, 2.5, 4.0, -1, 5, 0]
max(numlist) # gives the element with the maximum value; 5 in this case
min(numlist) # gives the element with the minimum value; -1 in this case

numlist.index(max(numlist)) # gives the location of  5 in the list. Output is 4
```

- Reversing the order of elements in the list

```
numlist.reverse()
numlist  # generates the reversed list [0,5,-1,4.0,2.5,3]
```

Caution: using reverse() to reverse the order of elements in numlist changes numlist

```
numlist.reverse()  # reversing the reversed numlist
numlist  # regenerates the original list [3,2.5,4.0,-1,5,0]
```

Mathematical Operations on a List of numbers

- Sorting a list by using the *sort()* command

```
numlist.sort() # sorts numlist in ascending order generating the list [-1, 0, 2.5, 3, 4.0, 5]
                    # but changes numlist itself
numlist # numlist becomes the sorted list [-1, 0, 2.5, 3, 4.0, 5]

numlist.sort(reverse=True) # sorts numlist in descending order generating the list
                                # [5, 4.0, 3, 2.5, 0, -1]
numlist # numlist becomes the sorted list in descending order [5, 4.0, 3, 2.5, 0, -1]
```

- Sorting a list *without* changing the list by using the *sorted* command

```
sorted(numlist) # sorts numlist in ascending order generating the list [-1, 0, 2.5, 3, 4.0, 5]
                    # but does not change numlist
numlist # generates the original numlist [3, 2.5, 4.0, -1, 5, 0]  that remains unchanged

sorted(numlist, reverse=True) # sorts numlist in descending order generating the list
                                # [5, 4.0, 3, 2.5, 0, -1]
```

Sorting a mixed List of numbers *and* strings

- Sorting a list of strings

```
strlist=['Virat', 'Kohli']

sorted(strlist) # sorts strings in alphabetical order with
                # generating the list ['Kohli', 'Virat']
                # does not change strlist

print(strlist) # generates the original list ['Virat', 'Kohli']  that remains unchanged
```

Adding items to a List

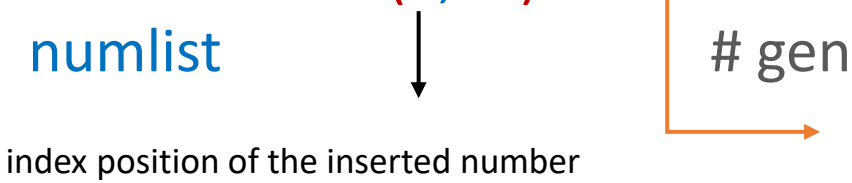
- Adding items to a list from the right using append

```
numlist = [3, 2.5, 4.0, -1, 5, 0]

numlist.append(10) # adds 10 to the end i.e. from the right. Also works for a mixed list
numlist            # generates the new list [3, 2.5, 4.0, -1, 5, 0, 10]
```

- Adding items to a list at a given position using insert

```
numlist.insert(2,10)   # inserts the number 10 at index position 2
numlist                # generates the list [3, 2.5, 10, 4.0, -1, 5, 0]
```




```
numlist.insert(0,7.25)   # inserts the number 7.25 at index position 0
numlist                  # generates the list [7.25, 3, 2.5, 10, 4.0, -1, 5, 0]
```

Removing items from a List

- Removing items from a list from the right using pop()

```
numlist = [3, 2.5, 4.0, -1, 5, 0]

numlist.pop()       # gives the last element 0 from the end. numlist is shortened
numlist             # generates the new list [3, 2.5, 4.0, -1, 5]
numlist.pop()       # gives the last element 5 from the end.
numlist             # generates the new list [3, 2.5, 4.0, -1]
```

- Removing a *specific* item from a list using remove

```
numlist.remove(3)       # removes the number 3 at position 2
numlist                 # generates the list [2.5, 4.0, -1]
```

- Removing items from a list at specific positions using del

```
numlist = [3, 2.5, 4.0, -1, 5, 0]
del numlist[1:4]        # deletes elements starting from index position 1 to position 3
numlist                 # generates the shortened list [3, 5, 0]
```

More about Strings and Lists: Conversion from String *to* List

- *String to List conversion*

```
numstr='1234' # number defined as a string.
numlst =list(numstr) # creates a list out of each character making up the string
print (numlst) # generates the list ['1', '2', '3', '4']

numstr2='56'  # NOTE: This is not a number; 5 & 6 are treated as characters
numstr+numstr2 # concatenation of two strings gives  a larger string '123456'
```

- A number defined as a string can be treated as number using the *eval* command

```
eval(numstr) # converts  the string 1234 into a number
```

- Evaluation and concatenation operations *do not* commute

```
eval(numstr+numstr2) # gives 123456 i.e. concatenation precedes evaluation
eval(numstr)+eval(numstr2) # gives 1290 = 1234+56
```

- ***Caution***: You can convert a string to a number using the *eval* command only if the characters making up the string are numbers

```
eval(str) # gives an error if for example str='Hello!'
```

More about Strings and Lists: Conversion from List *to* String

- *List to String conversion*

```
newlst = ['P', 'y', 't', 'h', 'o', 'n', '3.8']
newstr = ''.join(newlst) # join each element in the list to the next without any space
print (newstr) # gives a single string 'Python3.8'
```

- There are many ways to *join* the elements of a list

```
newstr2='-'.join(newlst) # join each element in the list to the next with a dash
print (newstr2) # gives a single string 'P-y-t-h-o-n-3.8'
```

More String Operations: Splitting, Manipulating and Joining

❖Splitting a string by blank spaces

```
text = 'IISER Kolkata is the best IISER'
brokentext = text.split()  # brokentext becomes a list of strings where each element is a word of the text

print (brokentext) # prints it in the form of a list
print (brokentext[1]) # extracts and prints the second element (i.e. Kolkata) from the list
```

❖Manipulating the split string

```
rev=brokentext[-1:-7:-1] # reverses the order of elements in the string and stores it in a new string rev

print (rev) # to verify that the order has indeed been reversed
```

❖Rejoining the split string

```
jbs=" "  # indicates that the separate words in the string should be joined with a blank space between them

joined=jbs.join(rev)

print (joined) # Prints the original string  called text but with the words in reverse order
```

More String Operations: Splitting, Manipulating and Joining

❖ Splitting a string can be done at any specified separator such as : or , or ;

```
text = 'IISER Kolkata,is the best,IISER'
```

➢ Splits at ',’

```
print (text.split(', ‘)) # prints ['IISER Kolkata', 'is the best', 'IISER']
```

```
text = 'IISER:Kolkata:is:the:best:IISER'
```

➢ Splitting at ':’

```
print (text.split(‘:’)) # prints ['IISER’, ‘Kolkata', ‘is’, ‘the’, ‘best', 'IISER']
```

Introduction to Python 3.8: Comparison Operations

```
a = 21
b = 10
c = 0

if ( a == b ):        # : is essential at the end of
    print "Line 1 is True“ # indentation is important in python
else:
    print "Line 1 is False“ # statement following if and else statements must aligned differently

    if ( a != b ):
            print ("True“)
    else:
            print (“False“)

a = 5; b = 20;
if ( a <= b ):
        print "a is either less than or equal to b"
else:
        print "a is neither less than nor equal to b"
```

Indentation is essential

Control structure: if ……elif……. else in Python 3.8

❖ Examples:

```
k = input("Enter a number: ") # asks for user input from the terminal
if (not k==1):                          # be careful to give the :
        print ("Not one“)
else:
        print ("One“)


a=input(‘Enter a: \n’) # \n indicates line break and takes the input from the next line
b=input(‘Enter b: \n’)
c=input(‘Enter c: \n’)


x = b**2-4*a*c
if (x<0):
        print (‘x is negative’)
elif (x>0):
        print (‘x is positive’)
else:
        print (‘x is zero’)
```

Repeated operations: for loops

❖ Code blocks and indent structure of a for loop

```
for x in range(beginning value, end value, step-size):
        statement 1
        statement 2
print ('end of loop') # end of indentation implies end of loop
```

Indentation is essential

Body of loop

❖ Example of a for loop in action: generate integers & their squares from 1 to n=10

```
for x in range(1,11):
        print (x, x**2)
```

Repeated operations: while loops

❖ General structure of the while loop

initialize counter

while specify condition: # Don't forget the : after the condition

perform operations within the loop as long as above condition is satisfied

increment counter by 1; return to the beginning of the loop to check condition

repeat operation if condition is satisfied, otherwise exit loop

❖ Example of a while loop in action

```
ko = 0                          # initialize counter
while ko**2<101:                # check value of counter to decide whether to enter the loop
    print ("hello“)             # perform operations within the loop:
    j = ko**2                       # generate square of integers
    print (ko)                      # print integer whose square is calculated
    ko += 1                         # increment counter by 1
```

Indentation is essential

NOTE: The operations within the loop continues as long as condition following the while statement is not satisfied. The position (denoted by presence of absence of indentation) determine whether the statements are with or outside the loop

The break command

How to get out of a loop without waiting for it to end?

while loop example: Sum of integers with break in loop

Algorithm

- While the number < 11
- Add first number to 0
- Store result in memory
- Increment count by 1
- Repeat steps until count is 10 and you have added the last number to generate the final total

Algorithm

- Initialize count to 0
- Initialize memory storage counter to 0
- While the number < 11
- Add first number to 0
- Store result in memory
- If count is 10 break out of the while loop even if sum desired is of first n>10 integers.
- Otherwise increment count by 1 and repeat until the first 10 integers have been added to generate the total

List Comprehension

- Creating a new list from an existing list

```
L1=[1,2,3,4,5]
L2=[i**2 for i in L1] # loop within a list to generate a new list of squares of numbers in L1

from math import *                  #  imports all functions in the python math module
theta=[0, pi/2, pi, 3*pi/2, 2*pi]   #  lists
L3=[sin(x) for x in theta]          #  generates list of sin values for each entry in the list theta
```

- Logical structures inside lists.

```
L4=[n**2 for n in L1 if n<=4]      # generates squares of numbers in L1 upto the specified cutoff
print (L4)                                  # returns the list [1,4,9,16]
```

- Creating a list of pairs of numbers from two lists

```
L5=[1,2,3]
L6=[4,5,6]
LP=[(i,j) for i in L5 for j in L6]      # generates the list of pairs [(1,4), (1,5),(1,6),….(2,5),….(3,6)]
```

Tuples

- A tuple is an *immutable* list that ***cannot*** be changed ➔ elements cannot be inserted or deleted from a tuple.

```
t1=(0,1,2,3,4) # defines a tuple
t2=(9,8,7,6,5)
subjects=('CS', 'phys', 'chem', 'math')
```

NOTE: The round bracket () used to define a tuple as opposed to a square bracket [] used to specify a list.

- You can perform mathematical operations on tuples just as is done on lists.

```
t1+t2     # generates a new tuple (0,1,2,3,4,9,8,7,6,5) of length 10
len(t1+t2)
```

- You cannot reverse or sort a tuple using t1.reverse() or t1.sort() unlike a list because a tuple cannot be changed.

```
print (sorted(t2)) # works; generates a tuple whose elements are is ascending order
print (sorted(t1+t2,reverse=True)) # sorts the tuple t1+t2 in descending order
```

- You can convert lists to tuples and tuples to lists

```
x = [0, -1, -3, -4, 3, 8, 9, 11, -2]          # A list
tuple(x)                                      # converts the list x to a tuple
list(t1)                                      # converts tuple t1 to a list
```

Sets

- A *set* is an unordered collection of ***unique*** elements defined with curly brackets {}

```
s1={1,2,6,9,6}                  # defines a set
print s1                        # generates the set {1,2,6,9}; ignores duplicate 6 in s1
```

- You can convert sets to tuples and lists and vice versa

```
tuple(s1)                       # converts set s1 to a tuple (1,2,6,9)
list(s1)                        # converts set s1 to a list [1,2,6,9]
bstrn='111100011001011'         # a binary string
set(bstrn)                      # generates the set (['1', '0'])
nstr='Python 3.8'
set(nstr)                       # generates set([' ', 'h', 'o', 'n', 'P', '2', 't', '7', 'y', '.'])
```

- Operations on sets

```
st1=set('Hello20MS')            # generates the unique set {'2', 'l', '0', 'H', 'e', 'M', 'o', 'S'}
st2=set('CS1101')               # generates the unique set {'0', '1', 'C', 'S'}
print (st1|st2)                 # generates a set which is the union of the 2 sets
print (st1&st2)                 # generates a set which is the intersection of the 2 sets
print (st1-st2)                 # gives characters unique to set s1, not found in s2
print (st2-st1)                 # gives characters unique to set s2, not found in s1
```

Dictionaries

- A *dictionary* is an unordered *set* of ***keys*** along with ***values*** associated with the specified ***keys***

```
D={'a':2, 'c':6, 'b':4} # defines a dictionary with keys a,b,c & values 2, 4, 6 associated with them
print (D)                                  # generates the entire Dictionary

D.keys()                   # generates the list of keys defined in the dictionary
D['a']                     # returns the value associated with the key 'a'

if 'c' in D:               # check if the key 'c' is present in the dictionary
        print (D['c'])     # if true, print the value associated with 'c'
else:
        print ('c is absent') # if false, print key is absent

D['d']=8                   # inserts a new key and its associated value

del D['a']                 # deletes an existing key and its associated value

print (sorted(D.keys()))   # sorts dictionary in alphabetical order of keys
```

Defining a Function in Python 3.8

```
def f(x):  # x is the argument of the function.
        return x**2        # returns the square of the argument
```

Note: A function can be *called* multiple times with different arguments

```
>>> f(7)              # gives 49
>>>f(3)+f(4)         # gives 25
```

❖ A function can also be defined inside the main block of the code using the foll. syntax

```
def main():
        x=input('Enter the value of x')
        def f(x):
                return x**2
        print (f(x))
main()
```

❖ A function can also be defined without the def statement by using the 'lambda function'

```
f= lambda x: x**2          # lambda x: ➔ f is a function of x whose functional form is specified after the :
>>>f(2)                   # gives 4
```

Defining a Function in Python 3.8

❖ Define functions to avoid code repetition and for simplicity

```
def last_digit(x):  # Takes a number (as argument) and returns its last digit
        if x>=10:
                x = x%10
        return x
>>>last_digit(17) #gives the output 7
```

Indentation is essential

Algorithm

❖ If number < 10, return number

❖ If number>=10, return remainder when number is divided by 10

❖ A function with 2 arguments: returns the sum of squares of the 2 arguments

```
def square_add(x,y):        # A function with 2 arguments
        z = x**2 + y**2
        return z  # returns the sum of squares of the 2 arguments
>>>square_add(3,4) #gives the output 25
```

❖ Functions can return multiple results

```
def f(x,y):
        x = x + y
        y = x - y
        x = x - y
        return x, y
```