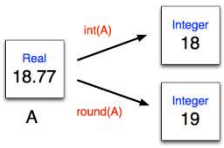
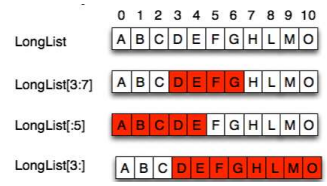
PYTHON NOTES:

Type Conversion:

A = 18.77
int(A)
round(A)
string(A)



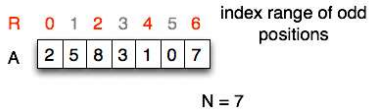
List Slicing:



Traversing a List:

```
# multiply a list by two
A = [2, 5, 8, 3, 1, 0, 7]
N = len(A)
R = range(0,N)

for i in R:
    A[i] = A[i] * 2
print(A)
```

```
# index range of odd positions
A = [2, 5, 8, 3, 1, 0, 7]
N = len(A)
R = range(0,N,2)
# double very element in odd position
for i in R:
    print(i)
    A[i] = A[i] * 2
print(A)
```




```
# traversing a list backwards
A = [2, 5, 8, 3, 1, 0, 7]
N = len(A)
R = range(N-1,-1,-1)
for i in R:
    print(A[i])
```

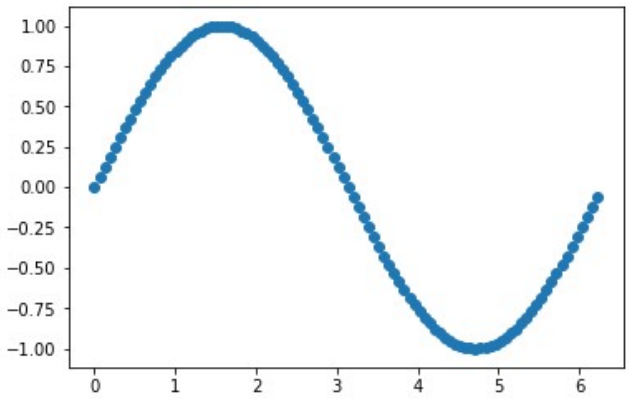
Libraries:

```
# maths functions
import math

e = math.exp(1)
print(e)

Radius = math.sqrt(100)
Area = math.pi * Radius **2
print(Area)

math.sin(3.14)

math.log(e)
```

Plotting Functions:

e.g. set a list x with N = 100 points between 0 and 2*pi. Determine y = sin(x). Plot y vs x.

```
# import modules
import math as mt
import matplotlib.pyplot as pl

# find the step of x values
N = 100
step = 2*mt.pi / N

# determine the x axis
# start with the first point

x = [0]
R = range(1,N)

# for all the points in the range, determine x and append it to the existing list
for i in R:
    x = x + [x[i-1] + step]

# determine y
y = []
for val in x:
    y = y + [mt.sin(val)]
# plot the data
pl.scatter(x,y)
```

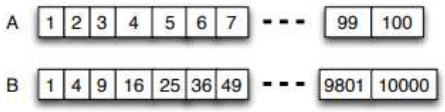


Appending Two Lists:

```
a = [1, 2, 3]
b = [3, 4, 5]
c = a + b
print(c)
```

[1, 2, 3, 3, 4, 5, 6]

Summing Two Lists:



e.g. Generate list A. Generate list B. Sum up two lists to form C.

```
# generate A
N = 100
R = range(1,N+1)
A = []
for i in R:
    A = A + [i]

# generate B
R = range(0,N)
B = []
for i in R:
    B = B + [A[i]**2]
print(B)

# sum A and B to form C
C = []
for i in R:
    C = C + [A[i] + B[i]]
```

Inputting Values from Keyboard:

```
# always given as string
# convert type to integer for number
a = input('Gimme me a number ')
a = int(a)
print(a)
```

Using Random Function to Simulate a Dice:

```
import random
# we multiply it by 6, hence mapping into the range 0 and 6.0, we add 1, hence mapping the range 1 to 7.0 (excluding 7.0)
dice = int(random.random()*6 + 1)
print(dice)
```

Counted Loops:

 # used if we wish to repeat set of statements
 # count from

```
start = 10

# count up to
end = 100
R = range(start,end+1)

for count in R:
    dice = int(random.random()*6 + 1)
    print(dice)
```

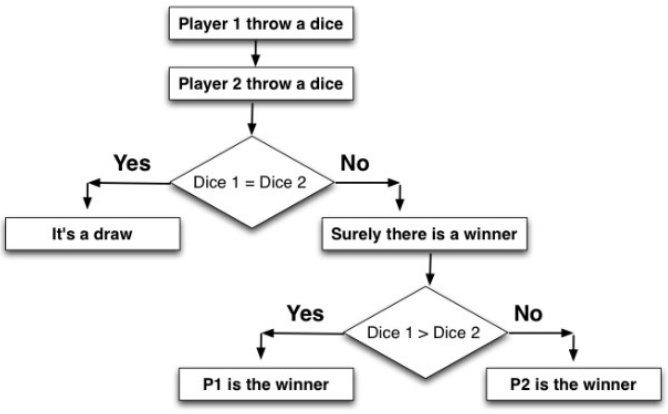
Boolean Statements:

```
(a==b)   a equals b

(a!=b)   a not equal to b
(a<b)    a less than b
(a>b)    a greater than b
(a<=b)   a less or equal to b
(a>=b)   a greater or equal to b

(Condition1 and Condition2)    AND

(Condition1 or Condition2)    OR
```

Nested Conditionals:

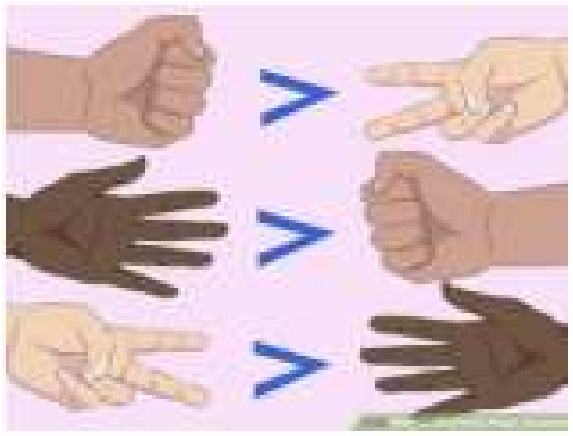
e.g. Dice game

```
dice1 = 1
dice2 = 3
if dice1 == dice2:
    print('The match is a draw')
else:
    print('Surely there is a winner')
    if dice1 > dice2:
        print('The winner is P1')
    else:
        print('The winner is P2')
```



Rock Paper Scissors Nested Conditionals Example:

```
import random

# Ask the user to input his/her throw, i.e. either R, P or S
P = input('Your throw (1,2,3): ')
P = int(P)

# machine plays
M = int(random.random()*3+1)

# check if it is a draw
if M == P:

    # same throw: it's a draw
    outcome = 'It is a draw'
else:

    # compound Boolean condition for the player to win over the machine
    if P==1 and M==3 or P==2 and M==1 or P==3 and M==2:
        outcome = 'Players wins'
    else:
        outcome = 'Machine wins'

# declare the winner
print(P,M)
print(outcome)
```



Generating Prime Numbers List:

```
# % is the modulus operator (checks for remainders)

N = input('Gimme N: ')

# value is read as String. It needs to be converted into a number

N = int(N)

# set the range of the counter
R = range(2,N+1)

# initialise the list of primes
Plist = []

# traverse all the numbers from 1 to N
for i in R:
```

```
    # examine if i is a prime
    # assume that i is a prime
    prime = True

    # find if there is a number between 2 and i-1 that can divide i
    # generate a range from 2 to i-1
    Ri = range(2,i)
    for j in Ri:

        # check if i is divisible by j, by looking at the reminder
        if i % j == 0:

            # i is divisible by j, hence it is not a prime. Our initial assumption goes banana
            prime = False

    # if i is a prime add it to the list
    if prime:
        Plist = Plist + [i]
print(Plist)
```

Finding Pi (Convoluted Method):

1. The value of π can be determined numerically by using a technique based on random numbers.
   - Consider a circle, of diameter 1, inscribed into a square.
   - The area of the circle is $\pi/4$.
   - The area of the enveloping square is 1.
   - Consequently, the ratio of the area of the circle to the area of the square will be $\pi/4$.

   The area of the square can be represented with a set of *N* random spatial points generated within the enveloping square. Some of these points, *Nc*, will reside into the circle too, and would therefore represent the area of the circle. Write a script to estimate the value of π with a number *N* of points. Run the script for various *N* = 1, 10, 100, 1000, 10k, 100k, 1M 10M 100M, and observe how the precision of the computed value for π varies with *N*.

```
import random

# set the number of points
N = 10000

# set the range of the counter
R = range(1,N+1)

# initialise the number of points within the circle
Nc = 0
for i in R:

    # generates x and y coordinates for a point within the square
    x = random.random()
    y = random.random()

    # establish if this point is within the circle
    if (x-0.5)**2 + (y-0.5)**2 < 0.5**2:

        # count it
        Nc = Nc + 1

# find the area
A = 4 * Nc / N
print(A)
```

Writing Data into Files:

e.g. if we want to store values into a list, each on a separate line, for the file

```
# operator \n generates a new line
a = [1,23,45,0,5,8]

# open the file
f = open('NewFile.txt','w')

# write a list into it, separating each element into a new line
for item in a:
    f.write(str(item)+'\n')

# close the file
f.close()
```

Reading Data from a File:

e.g. reading all lines (of existing file) to go into a list

```
# open the file
f = open('Numbers.txt','r')
a = f.readlines()
f.close()
an = []
for item in a:
    an = an + [int(item)]
print(an)
```

Removing newline trail (\n) from file data:

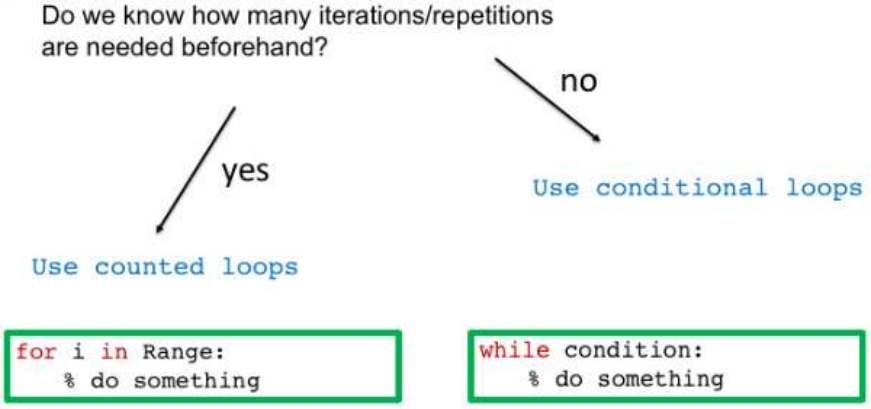
```
marks = ['45\n','80\n','64\n']
print(marks)

# print stripped
for item in marks:
    mk = item.rstrip()
    print(mk)
```

Conditional Loops:

When to use?

Counted Loops or Conditional Loops

Whenever we need to iterate a set of statement and in need of a loop, it is good practice to ask us this question:



Search Algorithm Bingo Example:

```
Bingo = [13,24,5,8,33,44,10,45,2,25]

find = 33
# number to be sought
found = False
count = 0
while (not found):
    if Bingo[count] == find:
        found = True
    else:
        count = count + 1
print(found)

find = 18
# this number is not in the list
found = False
count = 0

# ensures the search algorithm doesn't run forever (as 18 isn't in list)
while (not found) and count < len(Bingo):
    if Bingo[count] == find:
        found = True
    else:
        count = count + 1
print(found)
```

Series Expansion Example:

Task C: Series expansion

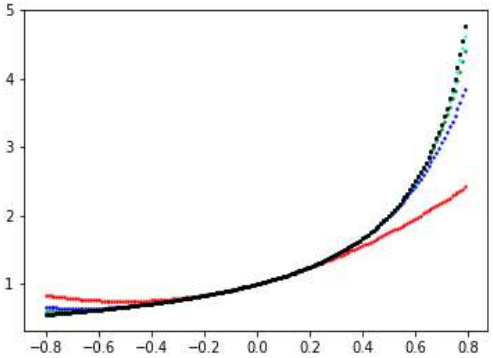
1. The function $y(x) = \frac{1}{1-x}$ can be represented by the series expansion:

$$y(x) = \frac{1}{1-x} = \sum_{i=0}^{N\to\infty} x^i = 1 + x + x^2 + x^3 + x^4 + \cdots$$

in the interval $-1 < x < 1$ only.
Write a script to evaluate the function *y(x)* in the range *x = [-0.8 0.8]* with step *0.01*, for values of *N = 2, 6, 10, 14*.
Plot, on the same graph, *y(x)* vs *x* in the specified range *x = [-0.8 0.8]*, for each value of N.

```
import matplotlib.pyplot as pl
```

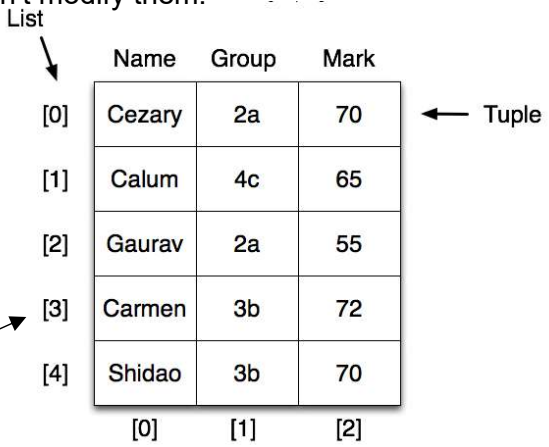
```
# set a list of colours, to be used when plotting different curves on the same graph
Col = ['Red','Blue','Green','Cyan']

# set the dx step
dx = 0.01

# define the range of N
RN = range(2,15,4)

# iterate for every N, i.e. N = 2, 6, 10, 14
for N in RN:

    # set the range of i, from 0 to N (terms to be added in the series)
    Ri = range(0,N+1)

    # initialise the list of x and y values
    x = []
    y = []

    # traverse the range of x starting from the lower boundary
    xc = -0.8
    while xc <= 0.8:


        # iterate along the range of x, until reaching the upper boundary
        # evaluate y(x) for the current x
        # initialise the sum
        yc = 0

        # traverse all the terms i of the series
        for i in Ri:

            # add the current term x**i
            yc = yc + xc**i

        # store the current evaluated x and y in the lists
        x = x + [xc]
        y = y + [yc]

        # update the value of x
        xc = xc + dx

    # plot y(x) vs x, for the current N, using every time a different colour
    pl.scatter(x,y,s=2,c=Col[N//4])

# evaluate and plot the analytical y(x)
for val in x:
```



```
    yex = 1 / (1-val)
    pl.scatter(val,yex,s=4,c='Black')
```

Tuples:

Lists are mutable. Tuples are immutable; therefore, we can't modify them.




```
# this produces an error
a = ('a','b','c','d','e','f','g','h')
print(a[4])
a[4] = 'XX'

# can append tuples
a = ('a','b','c','d','e','f','g','h')
a = a[0:4] + ('XX',) + a[5:]
print(a)
```

('a', 'b', 'c', 'd', 'XX', 'f', 'g', 'h')

List of Tuples:

```
ME1 = [('Cezary','2a',70),('Calum','4c',65),('Gaurav','2a',55),('Carmen','3b',72),('Shidao','3b',7
0)]

# picking best mark
MarkofBest = ME1[3][2]
print(MarkofBest)
```

72

```
# adding a student
ME1 = ME1 + [('Orace','2a',63)]
```

Finding the Average Marks:

```
# initialise the sum
Sum = 0

# traverse all the students in ME1
for student in ME1:

    # at every iteration, the variable student will take the value of an element of ME1,
hence it will be a tuple.
    # add the mark for this student:
    # the mark is in position 2 of the tuple
    Sum = Sum + student[2]
average = Sum / len(ME1)
print(average)
```

PT	Num	Tutees
Fred	4	Paul, Orsina, Isabel, Charlie

Tuples Containing Lists:

PersonalTutor = ('Fred Marquis',4,['Paul','Orsina','Isabel','Charlie'])

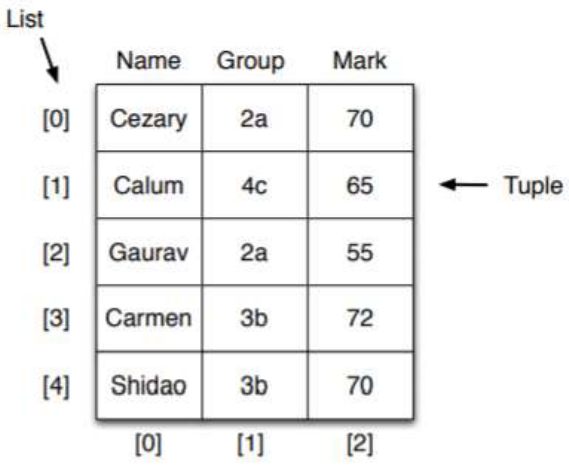
Sorting Algorithm:

e.g. into ascending order

```
A = [5,8,1,5,7,6,9,3,4,2]
N = len(A)

# Set the range for k, and in it repeat the same lines of of codes as before
Rk = range(0,N)
for k in Rk:
    Rm = range(k+1,N)
    for m in Rm:
        if A[m] < A[k]:

            # swap and put A[m] at position k (=0)
            # I am swapping using tuples
            (A[m],A[k]) = (A[k],A[m])
print(A)
```

Opening Files, Sorting Tuples:

e.g. 1. Download the files *Names.txt*, *Groups.txt* and *Marks.txt* from Blackboard. Write a script to form a list of tuples, associating every line content of the three files into a tuple.



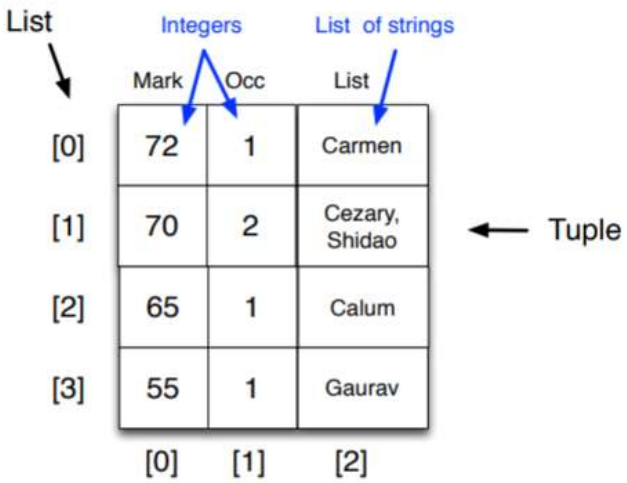

```
    # read files
```

```
f = open('Names.txt','r')
Names = f.readlines()
g = open('Groups.txt','r')
Group = g.readlines()
h = open('Marks.txt','r')
Marks = h.readlines()
f.close()
g.close()
h.close()

# set the range to traverse the files
R = range(0,len(Names))

# initialise the list
Students = []
for i in R:

    # add to the list a tuple made of a line from each of the three files
    Students = Students + [(Names[i].rstrip(),Group[i].rstrip(),int(Marks[i]))]

# print out the list
for student in Students:
    print(student)
```

Count Occurrences:

e.g. 1. Count the occurrences of every mark and form a list of tuples with (see figure below):
   a) the numerical mark,
   b) the number of occurrences of that mark,
   c) the list of students who achieved that mark.

   Plot graphically Occurrences vs Marks.



```
import matplotlib.pyplot as pl

# after having read the files (see Task A)
# set the range to traverse the files
Ns = len(Students)

# initialise the list that will contain the statistics
Stat = []

# examine all the marks, from 100 to 1, in reverse order
for mark in range(100,0,-1):
    # find which student had got this mark
    # initialise the counting
    count = 0

    # initialise the list of students with this mark
    LS = []

    # examine all the students in list Students
    for i in range(0,Ns):

        # check the mark of this student
        if Students[i][2] == mark:

            # this student has the mark examined
            # increment the counting
            count = count + 1

            # add his name to the list
            LS = LS + [Students[i][0]]

    # if this mark occurred add it to the list Stat
    # an element of the list is made of a tuple: (mark, occurrence, list of students with this mark)
    if count > 0:
        Stat = Stat + [(mark,count,LS)]

# print out the statistics
for item in Stat:
    print('Mark '+str(item[0])+' occurs '+str(item[1])+' times. List of students: ')
    for student in item[2]:
        print(student)
    print('\n')

# plot the statistics in a bar chart
for item in Stat:
    pl.bar(item[0],item[1])
```

Functions Basics:

```
# defining a function
def func1():
    print('Coffee')

# invoking a function
func1()
```

Arguments of a function:

```
def Power(base,exponent):

    # the variables base and exponent are input arguments to the function Power
    p = base ** exponent
    print(p)
Power (2, 3)
```

8

```
# the variable base was private to Power. It doesn't exist outside of the function
# the following will cause an error
Print (base)
```

Output Arguments of a Function:

```
# the following provides the caller with the variable p to be used in the main script
(before, a function would compute an answer that could no longer be used)
def Power(base,exponent):

    # the variables base and exponent are input arguments to the function Power
    p = base ** exponent
    return p

# main script
base = 2
exponent = 4
p = Power(base,exponent)
print(p)
```

Lists as Arguments:

```
def add10(inp):
    out = []
    for i in range(0,len(inp)):
        out += [inp[i] + 10]
    return out
```

```
# main
a = [1,2,3,4,5,6,7,8,9,10]
print(a)
b = add10(a)
print(b)
print(a)
```

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Function Examples:

Factorial Function:

```
def Factorial(n):

  # this function computes the factorial of n
    Rf = range(1,n+1)
    f = 1
    for i in Rf:
        f = f * i
    return f

# test the function
n = int(input('Gimme a number: '))
res = Factorial(n)
print(res)
```

Series Expansion (for exp): $y(x) = \sum_{i=0}^{N} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \cdots$

```
def ExpSeries(x,N):

    # this function computes the exponential of x
    Ri = range(0,N+1)
    y = []
    for xp in x:

        # computes exp(xp) for this xp
        yp = 0
        for i in Ri:

            # add terms up to N (using previous factorial function)
            yp += xp**i/Factorial(i)

        # append this yp
        y += [yp]
    return y
```

Sort Array in Ascending Order:

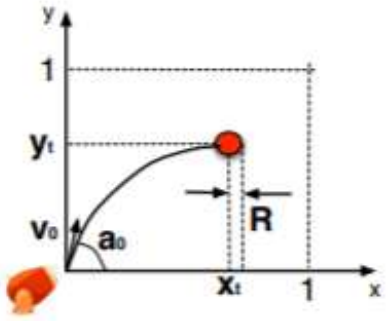
```
def SortAscending(x):

    # this function sorts the values of x
    # Note that the output is the same list, as the input list, but sorted
    Nx = len(x)
    Ri = range(0,Nx)

    #  I am reusing the sorting algorithm we used in Session 5 Task B, with minor modifications
    for i in Ri:
        Rright = range(i+1,Nx)
        for j in Rright:
            if x[j] < x[i]:

                # swap
                (x[j],x[i]) = (x[i],x[j])
```

Warfare Shooting Game Example:

Task D: Warfare: the shooting videogame

Scope of the game is to fire a bullet from the ground and hit a circular target suspended in air (in a 2-dimensional space). The target is a circle of radius R, positioned at a random place, *(xt, yt)*, in the spatial unity quadrant. The bullet is a single sizeless point.

The user needs to guess and input the shooting angle, a_0, and the initial velocity, v_0, of the bullet, in order to hit the target.



Write a function, *Shootbullet*, that receives the shooting angle, the initial velocity and the information about the target, i.e. *xt*, *yt* and *R*. The function must return the consequent shooting trajectory, i.e. two lists, *x* and *y*, coordinates of the trajectory, as described by:

$$y = xtan(a_0) - \frac{1}{2}\frac{x^2 g}{v_0^2}\frac{1}{cos^2(a_0)}$$

where $g = 9.81\ m/s^2$ is the gravitational constant.

Set the step of the x-axis to $dx = 0.01$.

The function must also return a Boolean variable indicating whether the bullet has hit the target or reached the ground after missing the target.

Now that you are ready to play, write a script that creates the target and asks the user to play until the target is hit. For every attempt, plot the war scenery (i.e. the trajectory and the target).

```
import random as rn
import math as mt
import matplotlib.pyplot as pl

def shootball(a0,v0,xt,yt,R):
    g = 9.81 # gravity
    dx = 0.01 # step
    shot = False

    # set lists x and y
    x = []
    y = []

    # set the initial point
    xp = 0
    yp = 0

    # keep moving the bullet until the target is hit, or the bullet reaches the ground
    while (not shot) and yp >= 0:

        # next x step
        xp += dx
        yp = xp*mt.tan(a0)-0.5*xp**2*g/(v0**2*mt.cos(a0)**2)

        # append new values to the lists
        x += [xp]
        y += [yp]

        # check if the new point has hit the target
        if ( (xp-xt)**2 + (yp-yt)**2 ) <= R**2:
            shot = True
    return (x,y,shot)

###############
# main script
R = 0.01 # radius of the target
# generate the target at random position
xt = rn.random()
yt = rn.random()

#xt = 0.9
#yt = 0.7
# generate the coordinates of the target
```

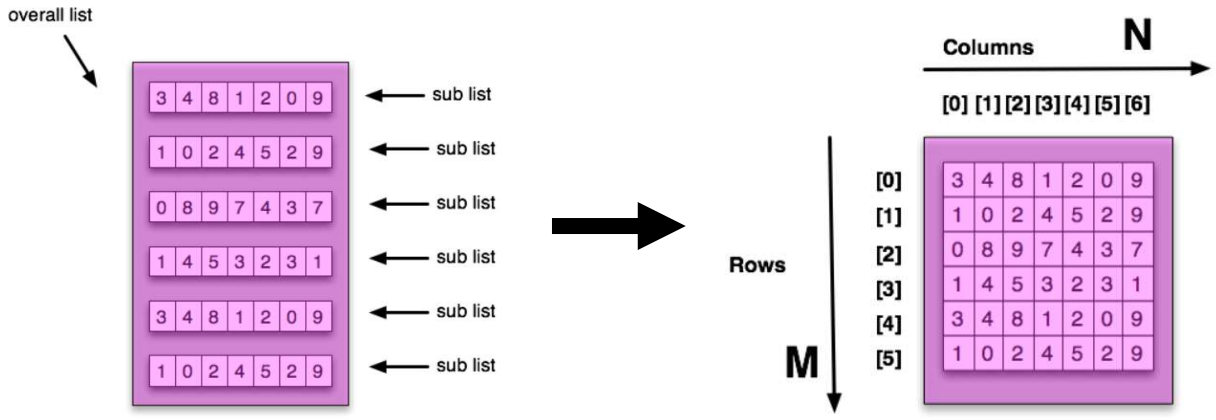
```
Rtheta = range(0,360)
xtarget = []
ytarget = []
for theta in Rtheta:
    thetar = theta * mt.pi / 180
    xtarget += [R*mt.cos(thetar)+xt]
    ytarget += [R*mt.sin(thetar)+yt]

# plot teh target
pl.scatter(xtarget,ytarget,s=2,c='r')

# set the axis to the unity quadrant
pl.axis([0, 1, 0, 1])
pl.show()
print('Shoot and hit me, if you dare!')

shot = False

# play the game, until target is shot
while not shot:
    # insert shooting angle and initial velocity
    theta0 = int(input('Shooting angle (in degree):'))
    theta0 = mt.pi/180*theta0
    v0 = input('Initial speed:')
    v0 = float(v0)

    # call the function
    (x,y,shot) = shootball(theta0,v0,xt,yt,R)
    # plot the war scenery
    #pl.plot(x,y,'b',xtarget,ytarget,'r')
    pl.scatter(x,y,xtarget,ytarget)
    pl.axis([0, 1, 0, 1])
    pl.show()
    if not shot:
        print('You missed the target lol. Try again')

print('Well-done: target centred. Enrol to the Army')
```

Recursive Functions:

e.g. an iterative countdown (a recursive function calls upon itself)

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

        # if last number, print the blastoff
        print('Blastoff')
    else:
        # not the last number: count the current number and invoke a counting for all the number below the current
        print(n)
        Countdown(n-1)

# main
n = int(input('Gimme a number: '))
Countdown(n)
```

Matrices:



A = [[3,4,8,1,2,0,9], [1,0,2,4,5,2,9], [0,8,9,7,4,3,7], [1,4,5,3,2,3,1], [3,4,8,1,2,0,9], [3,4,8,1,2,0,9], [1,0,2,4,5,2,9]]

```
# plot line by line
for row in A:
    print(row)

# select row
row = A[3]

# select cell
cell = A[3][4]
```

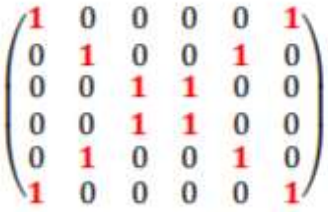
```
# traverse matrix using indices (horizontally first)
RM = range(0,len(A))  # numbers of rows
RN = range(0,len(A[0])) # numbers of columns
for i in RM:  # traverse all the rows
    for j in RN: # in this row, traverse all the columns
        print(A[i][j])
```

```
# traverse matrix using indices (vertically first)
RM = range(0,len(A))  # numbers of rows
RN = range(0,len(A[0])) # numbers of columns
for i in RM:  # traverse all the rows
    for j in RM: # in this row, traverse all the columns
        print(A[i][j])
```

Recursive Fibonacci Sequence:

```
def Fibonacci(n):
    if n == 1 or n == 2:

        # exit condition
        f = 1
    else:

        # recursive formula
        f = Fibonacci(n-1) + Fibonacci(n-2)
    return f

# main
N = int(input('Gimme a number'))
for i in range(1,N+1):
    f = Fibonacci(i)
    print(f)
```

Matrix 'Scottish Flag' Pattern:




```
N = int(input('N:  '))
RN = range(0,N)
A = [] # initialise the matrix as a list
for i in RN:

    # form i-th row
    row = [] # initialise i-th row as a list
    for j in RN:

        # add j-th column into i-th row
        if j == i or j == N-i-1:
            row += [1]
```

```
        else:
            row += [0]

    # add row i-th to the matrix A
    A += [row]

for row in A:
    print(row)
```

Read Matrix Function (for size N x N):

```
def ReadMatrix(filename,N):

    # this function read data from a file called filename and
    # put these in a matrix of size N x N

    # it does assume that the number of data/lines in the file is correct and equal to N*N
    f = open(filename,'r')

    # read all the lines in a temp list
    temp = f.readlines()
    f.close()

    # for the matrix NxN
    RN = range(0,N)
    Mat = [] # initialise the matrix
    c = 0 # count the line scrolling in the temp list
    for i in RN:

        # form row i-th
        row = [] # intialise row i-th
        for j in RN:

            # insert column j in row i
            row += [int(temp[c].rstrip())]
            c += 1 # increment the counter for the lines of temp, as we progress along
        Mat += [row]
    return Mat # IMPORTANT!!!!!!!!!!!!!!!! Return the matrix to the main caller
```

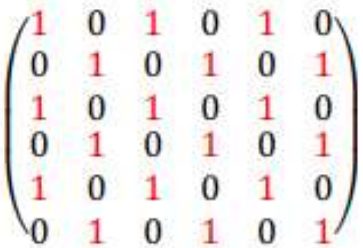
Matrix Vector Multiplication Function:

$$c_i = \sum_{j=1}^{M} A_{ij} b_j$$

```
def MatVec(A,b):

    # this function computes the matrix vector multiplication between A and b
    # c = A . b
   # check if dimensions are compatible
    N = len(A) # number of rows
    M = len(A[0]) # number of columns

    # number of columns of A must equal the length of b
    if M == len(b):

        # dimensions are compatible: do the multiplication
        c = []
        RN = range(0,N)
        RM = range(0,M)
        for i in RN:

            # compute c(i)
            sum = 0
            for j in RM:
                sum += A[i][j] * b[j]
            c += [sum]
    else:

        # dimensions are not compatible, return the value of zero
        c = 0

    return c
```

Transpose of a Matrix Function: $T_{i,j} = A_{j,i}$

```
def Transpose(A):

    # find size of A
    M = len(A) # number of rows
    N = len(A[0]) # number of columns
    RM = range(0,M)
    RN = range(0,N)
    T = []
    for i in RN:
        row = []
        for j in RM:
            row += [A[j][i]]
        T += [row]
    return T
```



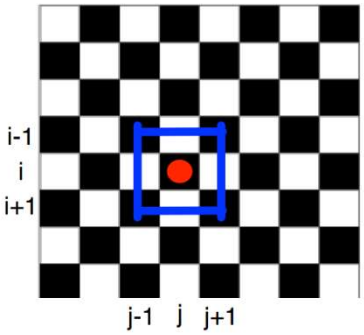
Chess Board Matrix Configuration:

```
# this script generates a square matrix with a chess board configuration
N = 8

RN = range(0,N)
A = []
for i in RN:

    # generate a row
    row = []
    for j in RN:
        if (i+j)%2 == 0:
            row += [1]
        else:
            row += [0]

    # append this row
    A += [row]

for row in A:
    print(row)

# save the matrix
f = open('ChessBoard.txt','w')
f.write(str(N)+'\n')
f.write(str(N)+'\n')
for row in A:
    for col in row:
        f.write(str(col)+'\n')
f.close()
```

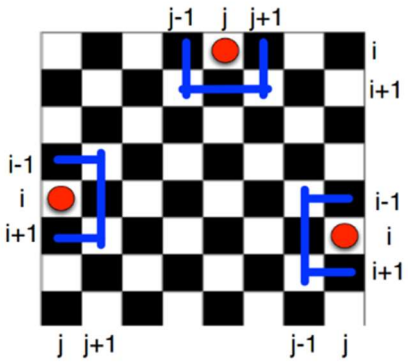
Playing Chess:

Theory:

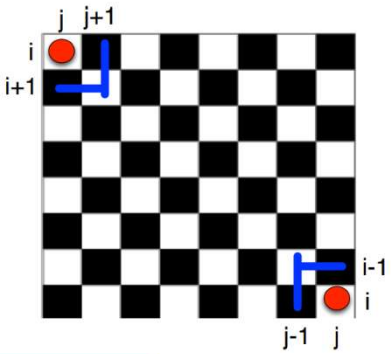
Checking checkmate



Checking checkmate

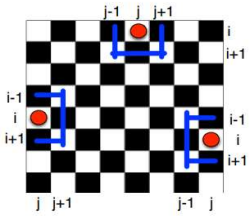


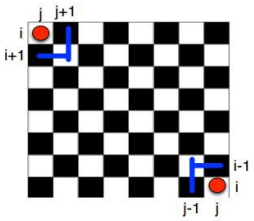
Checking checkmate



Checking checkmate

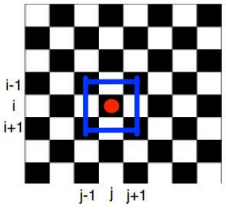
```
# Set the positions of the surrounding cells.

ib = max(i-1,0)  # line before (i,j)
jb = max(j-1,0)  # column before (i,j)
ia = min(i+1,7)  # line after (i,j)
ja = min(j+1,7)  # column after (i,j)
```





Checking checkmate

```
# check cells north of (i,j)
if Game[ib][jb] < 0 or Game[ib][j] < 0 or Game[ib][ja] < 0:
    checkmate = True

# check cells south of (i,j)
if Game[ia][jb] < 0 or Game[ia][j] < 0 or Game[ia][ja] < 0:
    checkmate = True

# check cells east and west of (i,j)
if Game[i][jb] < 0 or Game[i][ja] < 0:
    checkmate = True
```



Playing Chess:

Code:

```
# load a game
f = open('ChessC.txt','r')
temp = f.readlines()
f.close()

# form the chess board matrix from the data in file
N = 8
RN = range(0,N)
Game = []
line = 2 # start from line 2, as lines 0 and 1 contains N = 8
for i in RN:
    row = []
    for j in RN:
        row += [int(temp[line].rstrip())]
        line += 1
    Game += [row]

# Search for the Black King.
# There must be one, otherwise the game would be over.
found = False
i = 0
while (not found) and i < N:
    j = 0
    while (not found) and j < N:
        if Game[i][j] == 1:
            found = True;
        else:
            j += 1
    i += 1

i = i-1

# The Black King is in position i,j.
print('Black King found in: ('+str(i)+','+str(j)+')')

# Check for checkmates in adjacent cells
checkmate = False

# Set the positions of the surrounding cells.
# Consider also if the Black King (i,j) is at the border or
# at one corner of the board
ib = max(i-1,0)  # line before (i,j)
jb = max(j-1,0)  # column before (i,j)
ia = min(i+1,N-1)  # line after (i,j)
ja = min(j+1,N-1)  # column after (i,j)
```

```
# check cells north of (i,j)
if Game[ib][jb] < 0 or Game[ib][j] < 0 or Game[ib][ja] < 0:
    checkmate = True

# check cells south of (i,j)
if Game[ia][jb] < 0 or Game[ia][j] < 0 or Game[ia][ja] < 0:
    checkmate = True

# check cells east and west of (i,j)
if Game[i][jb] < 0 or Game[i][ja] < 0:
    checkmate = True

# print out the alert
if checkmate:
    print('Warning! Checkmate in action')
else:
    print('You are safe')
```

Matrix Multiplication (NEW):

```
# Matrix-Matrix multiplication
def MatMat(A,B):
    M = len(A) # rows of A
    NA = len(A[0]) # columns of A
    NB = len(B) # rows of B
    P = len(B[0]) # columns of B

    # define ranges
    RM = range(0,M)
    RN = range(0,NA)
    RP = range(0,P)

    # check that dimensions are compatible
    if NA == NB:
        # compute the product C = A * B
        C = []
        for i in RM:

            # compute row i of the matrix
            row = []
            for j in RP:

                # compute column j, i.e. element C[i][j]
                Sum = 0
                for k in RN:
                    Sum += A[i][k] * B[k][j]
                row += [Sum] # append this column
            C += [row] # append this row
    else:
        # dimensions not compatible
        C = 0
    return C
```