THE ONLY SHIBBOLETH THE WEST HAS IS SCIENCE. IT IS THE PREMISE OF MODERNITY AND IT DEFINES ITSELF AS A RATIONALITY CAPABLE OF, INDEED REQUIRING SEPARATION FROM POLITICS, RELIGION AND REALLY, SOCIETY. MODERNISATION IS TO WORK TOWARDS THIS.
BRUNO LATOUR

THE BOUNDARY BETWEEN SCIENCE FICTION AND SOCIAL REALITY IS AN OPTICAL ILLUSION.
DONNA HARAWAY

CSEF

INTRODUCTION TO PYTHON

THE STUDENT ACADEMY

Contents

List of Figures

List of Tables

The longest snake ever held captive is Medusa,
a reticulated python (python reticulatus).
On 12 October 2011, she was measured at
7.67 m long.

Note

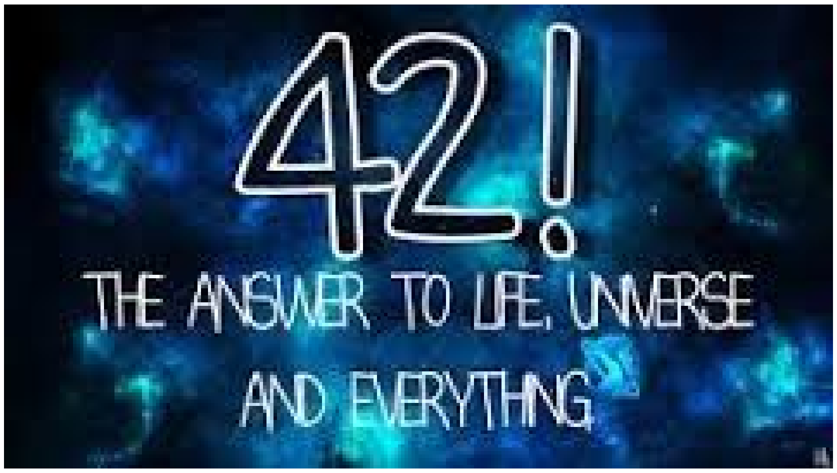
This physics text is an OpenSource academic project developed in abstraction at The Academy. The manuscript is written in $\LaTeX$ and makes use of the `tufte-book` and `tufte-handout` document classes.

http://latex-project.org/ftp.html
https://git-scm.com/downloads

Integers and Floating Point Numbers

What is Integer?

An integer (from the Latin integer meaning "whole") [note 1] is a number that can be written without a fractional component. For example, 21, 4, 0, and âĹŠ2048 are integers, while 9.75, 5Â¡, and âĹŽ2 are not.



[?]

Figure 1: The ultimate answer of the universe is also an Integer!

Floating point!

Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. You can convert an integer to a floating number by your python and vice versa! :

```
Type int(x) to convert x to a plain integer.
```

```
Type: int(4.22222)
Result: 4
```

```
Type float(x) to convert x to a floating-point number.

type: float(1)
Result: 1.0
```

Number Type Conversion

Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you need to coerce a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.

Definition:

Booleans:

In Python we have the following terms (characters and phrases) for determining if something is "True" or "False." Logic on a computer is all about seeing if some combination of these characters and some variables is True at that point in the program.

and
or
not
!= (not equal)
== (equal)
>= (greater-than-equal)
<= (less-than-equal)
True
False

We use these characters to make the truth or not.

NOT:
not False = True
not True = False

OR:
True or False = True
True or True = True
False or True = True
False or False = False

AND:
True and False = False
True and True = True
False and True = False
False and False = False

NOT OR:
not (True or False) = False
not (True or True) = False
not (False or True) = False
not (False or False) = True

NOT AND:
not (True and False) = True
not (True and True) = False
not (False and True) = True
not (False and False) = True

!=:
1 != 0 = True
1 != 1 = False
0 != 1 = True

0 != 0 = False
==:
1 == 0 = False
1 == 1 = True
0 == 1 = False
0 == 0 = True

While Loops

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

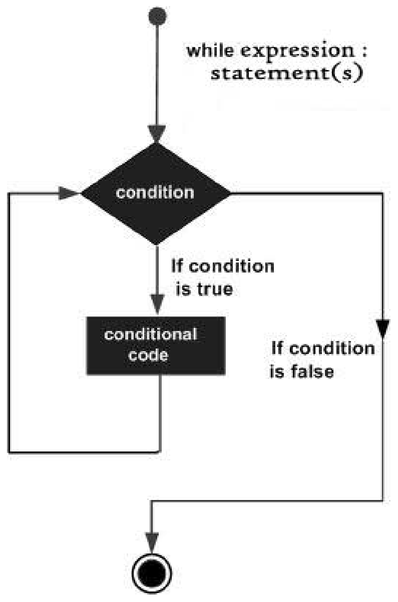


Figure 2: Flow diagram about how the while loop works

The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true.When the condition becomes false, program control passes to the line immediately following the loop.In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

Python code example

```
count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```

Output

The code will produce the following output.

```
>>>
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
>>>
```

Infinite Loop

A loop becomes infinite loop if a condition never becomes **FALSE**. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

```
var = 1
while var == 1:
   num = raw_input("Enter a number  :")
   print "You entered: ", num

print "Good bye!"
```

This python code is an example of how infinite loop can be created.

```
Enter a number  :X
You entered:  x
Enter a number  :Y
You entered:  Y
Enter a number  :Z
You entered:  Z
Enter a number between :
```

This code creates an infinite loop where it will need your input of any number.Once you input any number, it will output it like if you input "X" it will show back "X".

To break the loop you will either need to add the "break" command in your code OR press **CTRL+C** to exit the program.

Using else statements with while loops

Python supports to have an else statement associated with a loop statement.

If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

The following code illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise else statement gets executed.

```
count = 0
while count < 5:
   print count, " is  less than 5"
   count = count + 1
else:
   print count, " is not less than 5"
```

This is the output in python when the code above is executed.

```
>>>
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
>>>
```

While Loops

A while loop is a function, which needs a boolean statement to run, in order to prit out a list of results. The while loop will print out as many resultts as possible, until the boolean statement stops being true. For example:

```
i=0
while (i<5):
    print i
    i=i+1
```

Now, the boolean statement inserted here is "i<5", meaning that "i" must be less than 5. The next function now commands the system to print "i". The result would be:

```
0
1
2
3
4
```

This is because after the computer was demanded to print out "i", the function "i==i+1", was entered, meaning that it should print out all the numbers that make the bolean statement rue, until the number is five, making it false.

There are also many ways you could manipulate this code. A break may be inserted as follows:

```
i=0
while (i<5):
    print i
    i=i+
    if i==4:
        break
```

The result would be:

```
0
1
2
3
```

Excluding "4", because as a result of the break, the computer is now told not to print anything from the number 4.

However, if the break came before "i=i+1" like:

```
i=0
while (i<5):
    print i
    if i==4:
        break
    i=i+1
```

The result would be:

```
0
1
2
3
4
```

Array 1-D, 2-D, 3-D

Intro

Array is like a storage, it can fill with string or integer. In 1-D, 2-D it can also represents the x-axis and y axis.

Creating arrays

Arrays is created buy blanket.

Example:

```
a=[ ]        *a is the array name that you want.
```

The things in the [ ] and be store and when you want to access it you will need its position in the array and type like a[0]

Example:

```
a=["apple","orange","banana"]
```

If you want to print banana form the array, you may want to type

```
print a[2]
```

Filling arrays

Everything can be store in the array, strings, integers, arrays. When you create an array you can fill things in it as the default things that the array have.

Example:

```
    a=["Billy","Bud",90,60,50]
    b=["Anne","Chow",90,95,100]
    c=["Jen","Bo",60,80,90]
```

If you want to add things into the array that u create already, you can use

```
array_name=array_name+[Things you want to add]
```

Example:

```
a=[apple]
```

and now I want to add orange into it, so we add

```
a=a+[orange]
```

To create 2-D or more array we need to create array in the nested for-loop.

Example:

```
a=[]
for i in range(N): *N how long you want the array to be
    b=[]    *This is a temporary array to generate every array inside the main array.
    for j in range(N):
        *Things you want to put in the array by b=b+[ ]
    a=a+[b]     *Here put the temporary array back to the main array.
```

Traversing array

Traversing array is visiting each element in the array and do something. In 1-D we can do it with for loop to identify things in array.
Example:

```
a=[1,2,3]
for i in range(len(a))    *len(a) =  Numbers of elements in the array
    *Things put here can edit the specific element a[i]
```

In 2-D we start using nested for-loop to identify the x-axis and y-axis. So we use nested for-loop to traversing it too.

Example:

```
a=[[0,1],[0,0],[0,1]]
for i in range(len(a)):
    for j in range(len(a)):
        *Things put here can edit the specific element a[i][j]
```

In 3-D we use more for-loop to identify the more dimension.

Example:

```
a=[[[0,0],[0,0]],[[0,0],[0,0]]]
for x in range(len(a)):
    for y in range(len(a)):
        for z in range(len(a)):
            ∗Things put here can edit the specific element a[x][y][z]
```

For Loops, Nested For Loops

For in loop has the ability to iterate over the items of any sequence, such as a list or a string.

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating var. Next, the statements block is executed. Each item in the list is assigned to iterating var, and the statement(s) block is executed until the entire sequence is exhausted.

PYTHON CODE

Syntax:

```
for var in sequence:
   statements
```

Example 1

```
for i in ["hello", "hey", "yo"]:
    print i
```

The goal is to print all the elements in the array

Output 1

```
>>> Hello
    hey
    yo
```

Example 2

```
for i in range(5):
    print i
```

The goal is to print all the number in range 5. Note, that Python starts counting from 0

Output 2

```
>>> 1
    2
    3
    4
    5
```

Nested for loops

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax

```
for iterating_var in sequence:
   for iterating_var in sequence:
      statements(s)
   statements(s)
```

Example 1

```
for i in range(5): #loop everything indented 5 times
    for j in range(3): #loop everything indented 3 times
        print i #print output of i
```

The goal is to print every number from 0 to 5 three times in a row.

Output 1

```
>>> 0
    0
    0
    1
    1
    1
    2
    2
    2
    3
    3
    3
    4
    4
    4
```

Example 2

```
import random #importing "random" library that will take random numbers for our program
x = [] #creating initial, empty array that will be fulfilled with other arrays later
for i in range(5): #loop everything indented 5 times
    r=[] #creates 5 more arrays
    for j in range(5): #loop everything indented 5 times
        if random.uniform(0,10)<3: #compare if randomly chosen number is smaller than 3
            r=r+[1] #if it is, add value of "1" inside the secondary array - r
        else: #or
            r=r+[0] #if it is bigger, add value of "0" inside the secondary array - r
    x.append(r) #add all of these secondary arrays to our main - x array.
    #Basically we create 2D array.
print x #print our 2D array to see the output.
```

The goal is to create a 2D arrays that would be fulfilled with values of "1" or "0" depending on which number was randomly chosen by the computer

Output 2

```
>>> [[0, 0, 0, 1, 1], [1, 0, 0, 0, 0], [0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0], [0, 0, 0, 0, 0]]
```

However, the numbers that you see here isn't the only output you can get. We used random uniforms and each time it will give different values.

Libraries

Introduction

Libraries in Python are extensions to the basic Pyhton coding. Python comes with some libraries of its own. But it is also possible to write your own libraries. But before you can use libraries you have to import the libraries that you want to use in your script. There are several ways how you can import libraries. Libraries are always imported at the beginning of a scrip.

Importing Python Libraries

The easiest way to import libraries is to use the import function. For these examples we will use the random library.

With this you have to put the libraries name in front of the function of the library

```
import random
print random.uniform(1,10)
```

If you want to rename a library before you are using it you can do the following

```
import random as rndm
print rndm.uniform(1,10)
```

If you dont want to have to write a library name in front of it at all you can do

```
from random import *
print uniform(1,10)
```

There is one other option how you can import libraries. If you use all of what we learned before we can use the following.

With this you can import a single function of a libary and you name the function.

```
from random import uniform as makeRandom
print makeRandom(1,10)
```

Importing custom libraries

If you want to use libraries you or someone else has written in python you can do that also. First you have to make sure that the script you want to import is in the same folder as the scripit you want to import it into. Lets asume we have to following script we want to use as a libary.

```
def Bla():
    print "Bla"

def MyFunction(A)
    print A+A
```

Lets assume the scripts name is lib. Now if we want to use this in our main script we can do eaither of our ways. Use the file name as the libraries name.

```
import lib
import lib as mylib
from lib import Bla as Tell
```

Statistics Library

Introduction

This statistics library includes eight functions that we can use to deal with a set of data.

Tips

- *Additional libraries are needed*
- *Multiple functions are needed for some operations*

Zero

```
def zeros(n):
    a=[]
    for i in range(n):
        a=a+[0]
    return a
```

(1)

This function can be used to create a consecutive and repeating array(all elements are the same).
Example:

```
def zero(n):
    a=[]
    for i in range(n):
        a=a+[0]
    return a

print zero(9)
```

Output: `[0, 0, 0, 0, 0, 0, 0, 0, 0]`

Summing an array

```
def sum_array(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s
```

(2)

This function can help us to sum up all the elements in an array.
Example:

```
a=[1,2,3,4]
def sum_array(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s
print sum_array(a)
```

Output: `10`

Finding means

```
def avg(a):
    return sum_array(a)/len(a)
```

(3)

This function can help us to calculate the mean of all the data in an array, and we need assistance of the sum function.
Example:

```
a=[23,19,25,21]
def sum_array(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s

def avg(a):
    return sum_array(a)/len(a)

print avg(a)
```

Output: `22`

Variance

```
def var(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]**2
    m=avg(a)
    return (s/len(a)-m**2)
```

(4)

This function can help us to find the variance of the data.Also, we need the assistance of the average function.
Example:

```
a=[23,19,25,21]
def sum_array(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s

def avg(a):
    return sum_array(a)/len(a)

print avg(a)

def var(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]**2
    m=avg(a)
    return (s/len(a)-m**2)
print var(a)
```

Output: 5

Construct an array of random numbers

```
def rand_array(n,mini,maxi):
    a=[]
    for i in range(n):
        a=a+[random.uniform(mini,maxi)]
    return a
```

(5)

We can use this function to construct an array filled with random numbers. We need the random library to run the function.Here n represents the number of elements in the array; mini is the minimum value of the elements;maxi is the maximum value

To fill a histogram

*Library "graphics" is needed**

```
def histogram(mini,maxi,bins,a):
    h=zeros(bins)
    w=(maxi-mini)/bins
    for i in range(len(a)):
        for j in range(bins):
            if (a[i]>(mini+j*w))and a[i]<(mini+(j+1)*w):
                h[j]=h[j]+1
    return h
```

(6)

The four arguments in parenthesis are decisive for the histogram. Here mini represents the minimum value in the data; maxi represents the maximum value in the data; bin represents the number of different groups of data; a is the number of all the data.

Find the maximum value

```
def maximum(a):
    m=0
    for i in range(len(a)):
        if a[i]>m:
            m=a[i]
    return m
```

(7)

This funciton can help us to identify the maximum value in the data.

Drawing a histogram

Library "graphics" is needed
To draw a histogram, we first need to define a function that can draw a bargragh:

```
def bar(a,win):
    win.setCoords(-1,-1,len(a)+1,maximum(a)+1)
    bl=[]
    tr=[]
    rec=[]
    for i in range(len(a)):
        bl=bl+[Point(i,0)]
        tr=tr+[Point(i+a,a[i])]
        rec=rec+[Rectangle(bl[i],tr[i])]
        rec[i].draw(win)
```

(8)

Then we combine the "bar" function and the "histogram" function.
Here is just an example:

```
def main():
    win=GraphWin()
    a=zeros(400)
    for i in range(len(a)):
        a[i]=sum_array(rand_array(10,0,1))
    histo=histogram(0.,10.,7,a)
    bar(histo,win)
main()
```

(9)

Graphics Library

The graphics library is a library for making easy graphical objects in python. It was written by John Zelle for use with his book Python Programming: An Introduction to Computer Science. This chapter is going to discuss the topics we covered during class. To use the library you have to import it first. For an explanation on how to import libraries see the section of the book.

Creating a basic window

The graphics library works as a programm of it own. A window gets updated as long as it gets changed or is waiting for a new command. To draw somthing on a window we first have to create a window. A creation follows this pattern:

```
Window = GraphWin(WindowName, Width, Height)
Window.setCoords(StartX,StartY,WidthPixel,HeightPixel)
```

Drawing on windows

Now that we know how to create a window we can start drawing on it. There are various objects we can draw on windows. Because the graphics library uses its own coordinates system we need to use it for creating objects.

```
Point(2,4) #Creates a point at 2,4
```

To actually draw in windows we have to use the draw command.

```
Window = GraphWin(WindowName, Width, Height)
Window.setCoords(StartX,StartY,WidthPixel,HeightPixel)
Obj = OurObject
Obj.draw(Window)
```

In this example we are drawing Obj to our window.

Rectangles

In python rectangles are drawn from the down left point to the top right point.

```
Rect = Rectangle(Point(X1,Y1),Point(X2,Y2))
```

This creates a rectangle calles Rect at the position X1 and Y1 with the width X2 and height Y2. After you have created your basic reactangle you can also color it.

```
Rect = Rectangle(Point(X1,Y1),Point(X2,Y2))
Rect.setFill(color_rgb(255,0,0)) #Set the color to red
```

Monte Carlo

Monte Carlo simulations are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other mathematical methods. Monte Carlo methods are mainly used in three distinct problem classes:[1] optimization, numerical integration, and generating draws from a probability distribution. In Class we have used the Monte Carlo simulation for the probability function.

We have estimated the value of pi

We start the familiar example of finding the area of a circle. Figure 1 below shows a circle with radius r=1 inscribed within a square. The area of the circle is $Pi * r^2 = Pi * 1 = Pi$ and the area of the square is 4 The ratio of the area of the circle to the area of the square is $[Graphics : Images/MonteCarloPiMod_g r_4.gif]$

$[Graphics : Images/MonteCarloPiMod_g r_5.gif]$ $[Graphics : Images/MonteCarloPiMod_g r_1 1.gif]$

```
import random
x=random.uniform(-1,1)
y=random.uniform(-1,1)
n=0.
#n is the number of random points in the circle
p=999999
#p is the number of random prints
for i in range(p):
    x=random.uniform(-1,1)
    y=random.uniform(-1,1)
    if (x*x+y*y)<1:
     n=n+1
pi=4*(n/p)
print pi
```

As you increase P the estimation will be more accurate. for p=999999

```
>>>
3.14152314152
>>>
```

Whereas for p=100

```
>>>
3.24
>>>
```

Game of Life

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970. The "game" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves.

Game of Life Rules

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

First
: Any live cell with fewer than two live neighbours dies, as if caused by under-population.

Second
: Any live cell with two or three live neighbours lives on to the next generation.

Third
: Any live cell with more than three live neighbours dies, as if by over-population.

Fourth
: Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Here is the code we wrote in class:

```
[frame=single]

import random
from graphics import *
```

```
#this function creates an NxN array filled with zeros
def empty(N):
    a=[]
    for i in range(N):
        b=[]
        for j in range(N):
            b=b+[0]
        a=a+[b]
    return a

#this function fills the array a with a portion p of live cells
def fill(a,p):
    N=len(a)
    for i in range(N):
        for j in range(N):
            if random.uniform(0,1)<p:
                a[i][j]=1

def update(A,B):
    N=len(A)
    for i in range(N):
        for j in range(N):
            neigh=A[(i-1)%N][(j-1)%N]+A[(i-1)%N][j]+A[(i-1)%N][(j+1)%N]+A[i][(j-1)%N]+A[i][(j+1)%N]+
            A[(i+1)%N][(j-1)%N]+A[(i+1)%N][j]+A[(i+1)%N][(j+1)%N]
            if A[i][j]==0:
                if neigh==3:
                    B[i][j]=1
                else:
                    B[i][j]=0
            else:
                if neigh==2 or neigh==3:
                    B[i][j]=1
                else:
                    B[i][j]=0


def gen2Dgraphic(N):
    a=[]
    for i in range(N):
        b=[]
        for j in range(N):
            b=b+[Circle(Point(i,j),.49)]
        a=a+[b]
    return a

def push(B,A):
    N=len(A)
    for i in range(N):
        for j in range(N):
            A[i][j]=B[i][j]

def drawArray(A,a,window):
#A is the array of 0,1 values representing the state of the game
#a is an array of Circle objects
#window is the GraphWin in which we will draw the circles
    N=len(A)
    for i in range(N):
        for j in range(N):
            if A[i][j]==1:
                a[i][j].undraw()
                a[i][j].draw(window)
            if A[i][j]==0:
                a[i][j].undraw()

N=10            #N is the number of live cells you start with
win = GraphWin("title",400,400)
win.setCoords(-1,-1,N+1,N+1)
grid=empty(N)
grid2=empty(N)
circles=gen2Dgraphic(N)
fill(grid,0.3)

while True:
    drawArray(grid,circles,win)
    update(grid,grid2)
    push(grid2,grid)
```