

Welcome
21MS Batch

General Instructions about Labs

Group Divisions:

Group A: Roll numbers 1 to 60

Group B: Roll numbers 61 to 120

Group C: Roll numbers 121 to 180

Group D: Roll numbers 181 to 248

CS1101 (Monday)

Faculty Instructor and TA list

A: Dr. Supratim Sengupta

TA's: TBA

B: Dr. Dwaipayan Roy

TA's: TBA

C: Dr. Koel Das

TA's:TBA

D: Dr. Susmita Roy/Dr. Swastika Chatterjee

TA's:TBA

General Instructions about Labs

Feb.21-Feb.26: NO lab classes due to theory mid-sem exam

No. of Weeks remaining for Lab Classes ~ 9 (leaving out first week for Intro)

Total no. of assignments ~ 8-9 (approx. 1 assignment per week)

CS1101-Introduction

Instructors: Dr. Dwaipayan Roy, Dr. Koel Das, Dr. Supratim Sengupta, Dr. Susmita Roy/Dr. Swastika Chatterjee

Tentative Grading Scheme

Worksheets: ~8-9;	Total Marks ~ 60
Class tests: 1-2;	Total Marks ~ 10
End-sem Exam:	Total Marks ~ 30 (Format will be decided later)
Total: 100	

CS1101-Introduction

Schedule

Group A: Roll No. 1-60

Group B: Roll No. 61-120

Group C: Roll No. 121-180

Group D: Roll No. 181-248

1 faculty instructors; 5 TA's assigned to each group

Original Structure

- ❖ Basics of Linux Operating System (~ 2-3 Lectures; 2 weeks)
- ❖ Basics of Plotting using Gnuplot (~ 2 Lectures; 2 weeks)
- ❖ Python 3.8 (~ 8 weeks)

This Year's Structure

- ❖ Python 3.8 (~ 8-9 weeks)

Starting Up

- ❖ Each of you can use your account ID and password to log onto any PC in the computer centre.
- ❖ The *Linux* operating System is installed on all PC's



Advantages

- ❖ Open-source
- ❖ As easy to use as Windows.
- ❖ Its *Free*! 😊

Worksheets

- ❖ You will start getting worksheets after 1 week
- ❖ Each worksheet will have to be completed during lab hours and uploaded on WeLearn from your respective accounts.

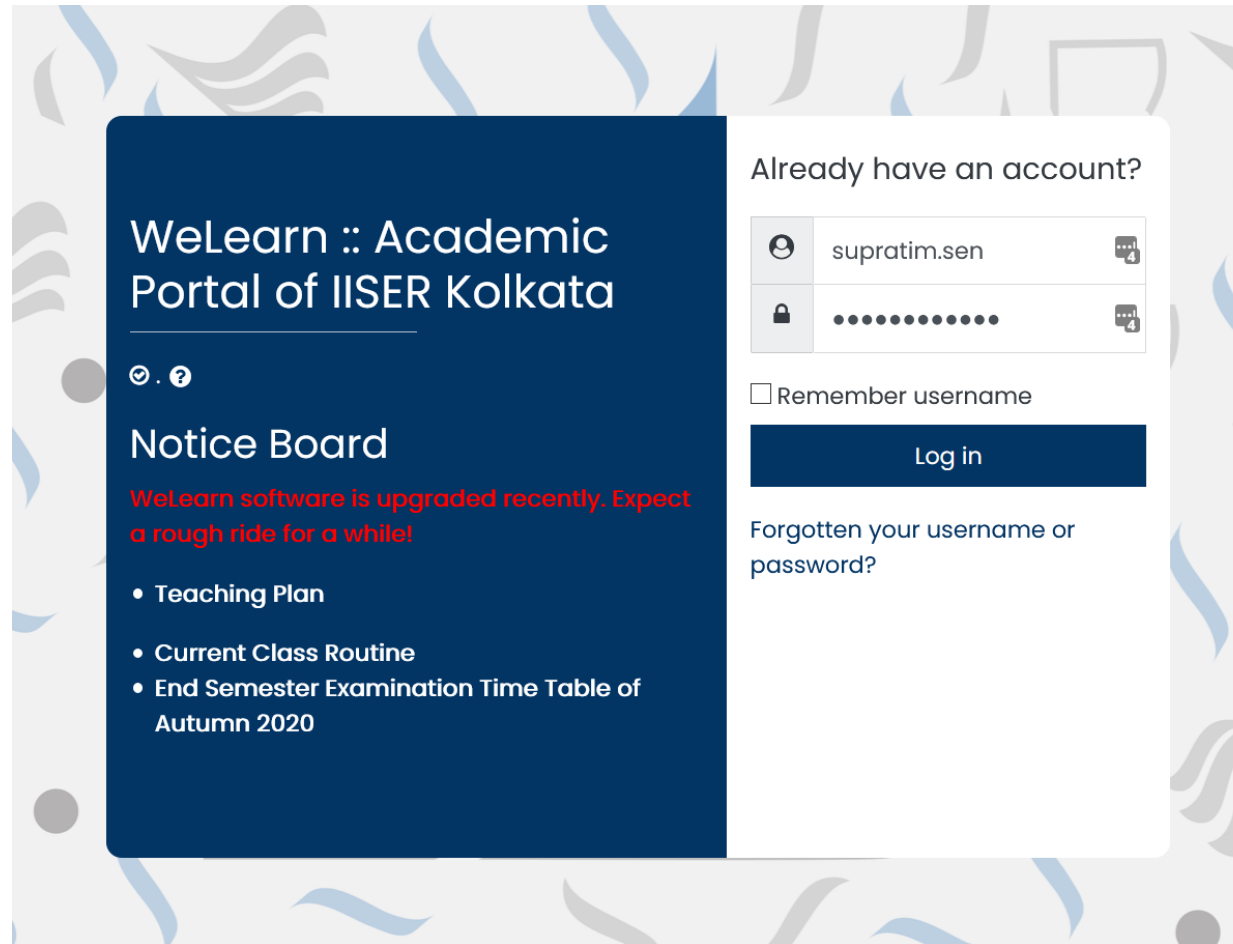
Don't Panic

I hear and I forget. I see and I remember. I do and I understand.

Confucius

Course Material Access

<https://welearn.iiserkol.ac.in>



The screenshot shows the WeLearn Academic Portal of IISER Kolkata. The left sidebar is dark blue with white text. The main content area is white. The login form is on the right.

WeLearn :: Academic Portal of IISER Kolkata

☑ . ?

Notice Board

WeLearn software is upgraded recently. Expect a rough ride for a while!

- Teaching Plan
- Current Class Routine
- End Semester Examination Time Table of Autumn 2020

Already have an account?

☐ Remember username

Log in

Forgotten your username or password?

Log on to WeLearn portal to get a list of *all* the courses you are enrolled in

Click on link to [CS1101](#) course to get access to course material that will be uploaded there

WeLearn Survey



 Course sections < Participants

 Grades

 Dashboard

 Site home

 My courses <

CS1101: Introduction to Computer Programming



Dashboard > My courses > CS1101

Turn editing on

 Discussion forum

 Are you familiar with (having past experience of using) computers

 Was there a curriculum on Computer Programming anytime in your past academic journey?

 Have you ever written computer programs in any language before?

 Which programming languages do you know?

Choose as many languages that you know, or **Not applicable** if you have not been exposed to any programming languages before.

 What type of device will you be using for writing the programs in this course?

Choose **none** if you do not own either a laptop/desktop or a smartphone. In case you own both, please select only that medium that you will be using for writing the programs in this course.

Introduction to Python 3.8: To do list for CS1101

- ❖ Define variables as integer, float, etc.
- ❖ Perform basic mathematical operations using the defined variables
- ❖ Create a list which can be a mixed list or a list of numbers only or a list of words only.
- ❖ Manipulate lists and strings
- ❖ Extract elements, remove elements, find elements, reverse order of elements etc.
- ❖ Perform mathematical operations on lists
- ❖ Loops, Control structures, Functions, Dictionaries, Class and Object in Python
 - ❖ Perform repeated operations (**for** and **while** loop)
 - ❖ Perform conditional operations (**if...elif...else**)
 - ❖ Define **functions** to automate repeated computing tasks
 - ❖ Define **class** and **object** and extract information from them
 - ❖ Importance of algorithm for specifying the logical sequence of operations
 - ❖ Write simple python programs using the above defined structures

Natural Language

Alphabet

+

Words constructed out of the alphabet

+

Sentences constructed using specific rules of grammar

Results

Tagore songs, Bhagavad Gita,
Dostoyevsky's stories,
Shakespeare's plays etc.

Programming Language

Syntax

+

Rules for using syntax

+

Lines of code

Results

Self-driven cars, facial
recognition software,
Voyager, Chandrayan
missions etc.

Programming

❖ Every programming language has their own ***syntax*** :

Form in which numbers, variables, functions, lists, rules of addition, search etc. are represented

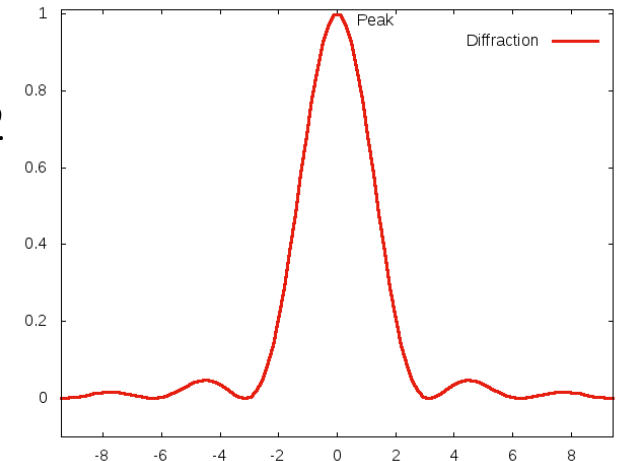
❖ **Algorithm**

Logical sequence of steps for solving a specific computational problem

❖ Syntax + **Algorithm** = Program (Code) that solves a specific computing problem

- Set x & y range
- Plot graph of function with red line of width 3
- Set legend title to “Diffraction”
- Label the peak of the graph as “Peak”

```
plot [-3*pi:3*pi][-0.1:1.01]sin(x)**2/x**2  
lc 7 lw 3 title "Diffraction"  
set key at graph 0.99,0.95  
set label "Peak" at 0.7,0.98  
replot
```



Algorithm

Code

Solution

Python 3.8 Resources

[Documentation at python.org](#)

For additional resources related to numerical and scientific computing with Python

[numpy.org](#)

[scipy.org](#)

❖ Basics of Python

[Python 3 tutorial](#)

[Python@geeksforgeeks](#)

[Python@W3schools](#)

❖ Module for writing, debugging and running python programs

[Anaconda](#)

❖ For general help with all kinds of computing problems (not just on Python)

[stackoverflow](#)

[StackExchange](#)

In-house Python Resources by Dr. Ananda Dasgupta

[History of Python](#)

A nice video lecture that talks about the history of Python

[Python from Scratch](#)

A series of video lectures that provides a gentle introduction to Python

For those of you who know Python programming

Give a man a fish, and you feed him for a day.

Teach a man to fish, and you feed him for a lifetime.

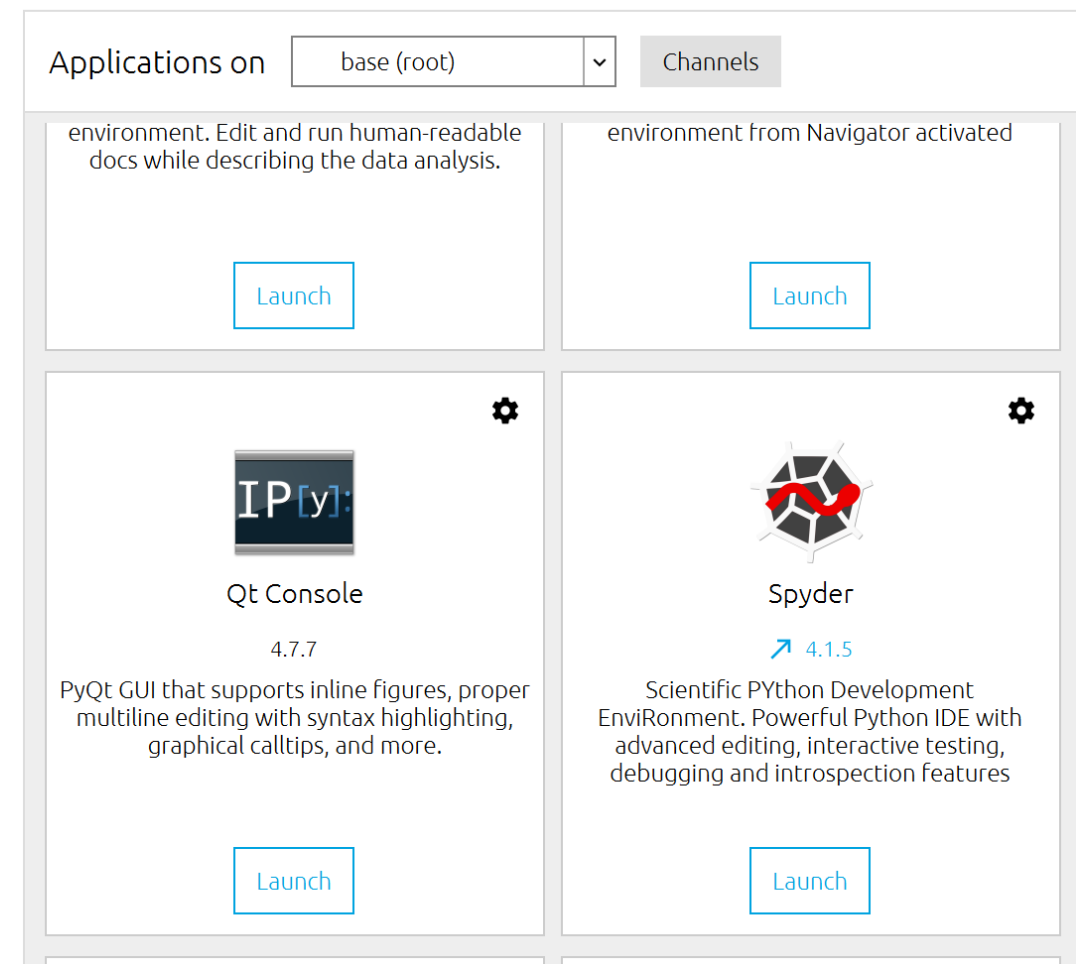
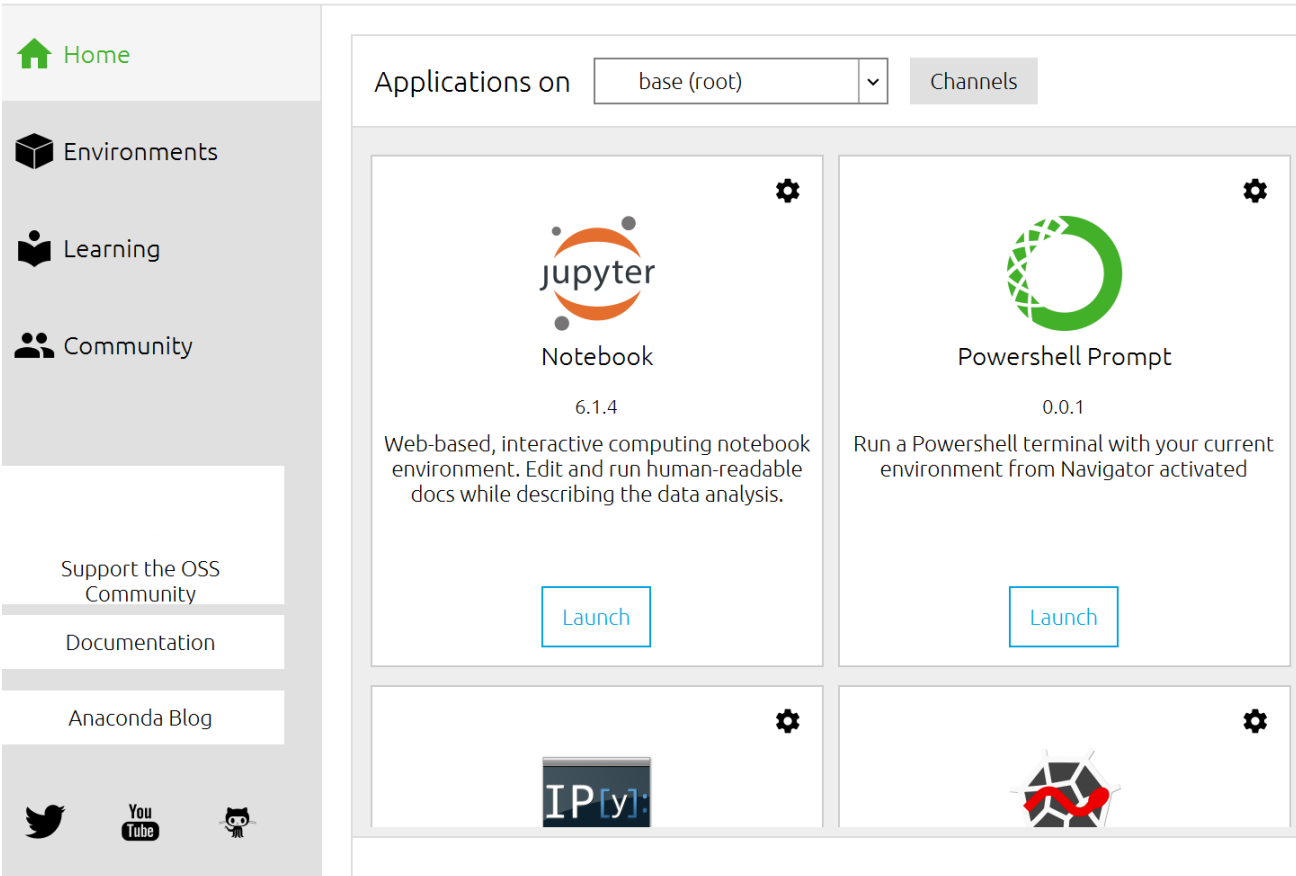
Ancient proverb

Tell your friend the answer to a problem, and you help him once.

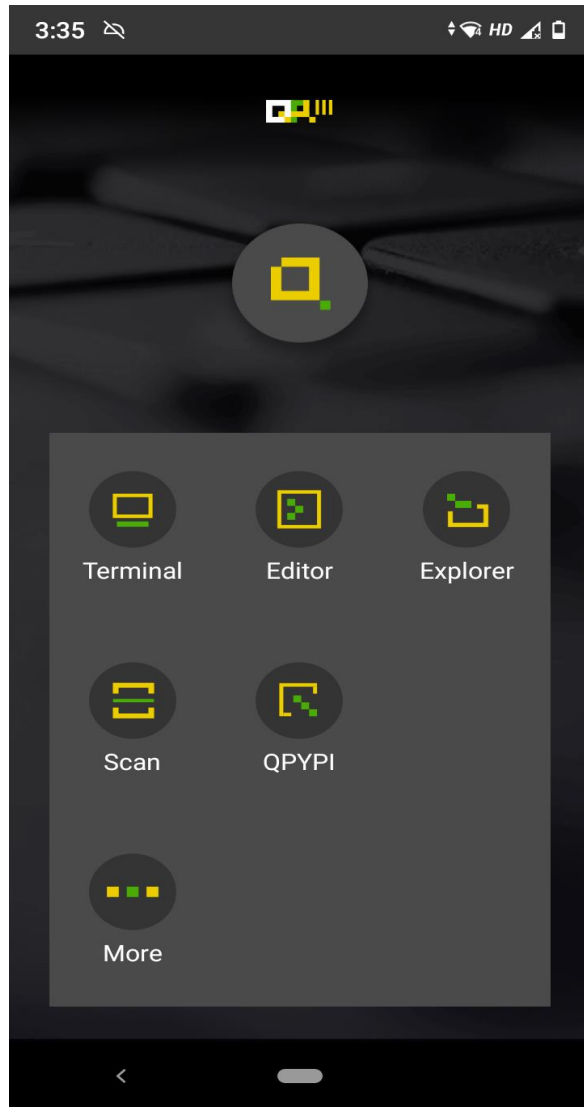
Teach your friend *how to* tackle a problem, and you help him for a lifetime.

Ancient proverb adapted to CS1101 course

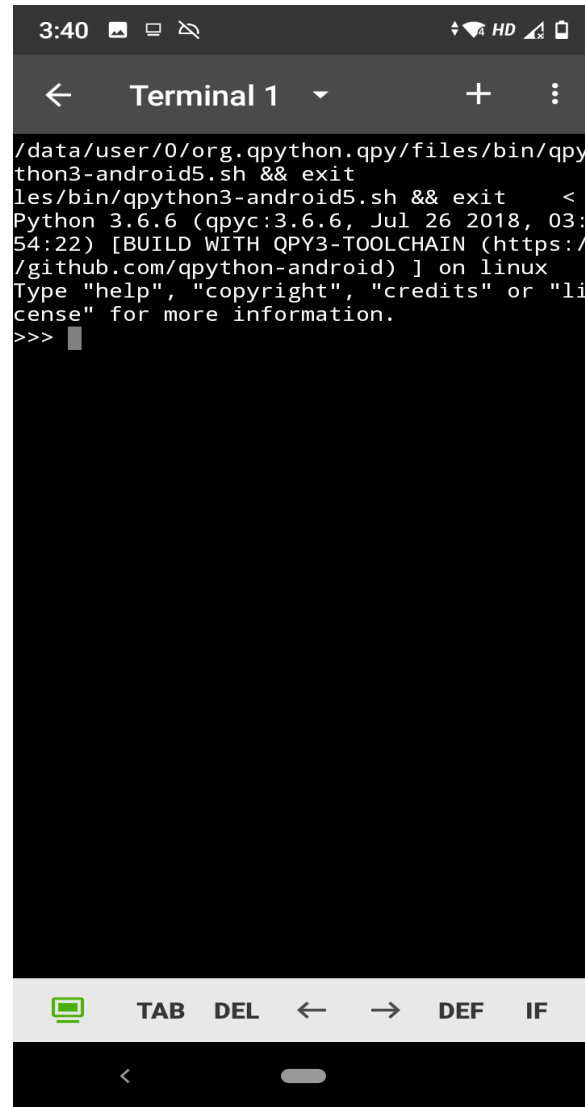
Python 3.8 on Desktop/Laptop Computer



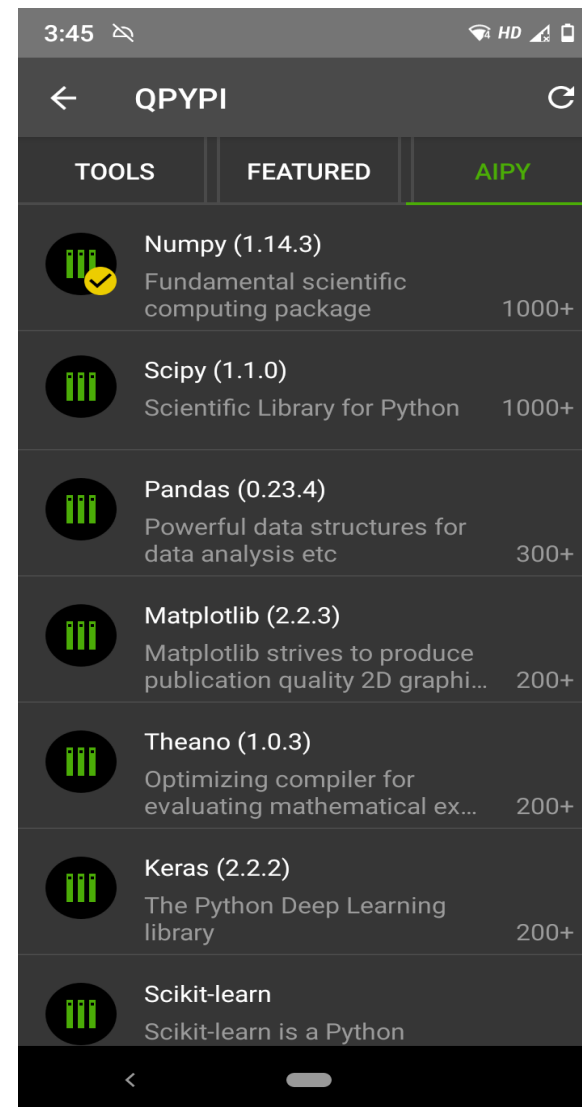
Python 3 on a smartphone: QPython for Android



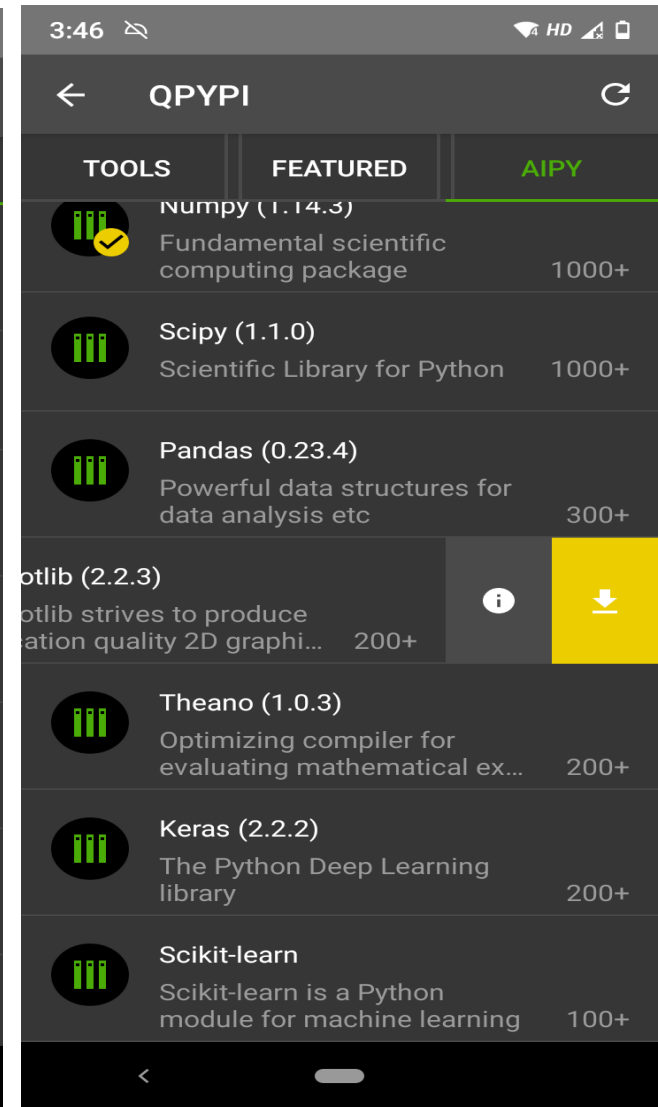
Welcome Screen



Terminal allows you to enter Python commands, load programs etc.



AIPY tab under QPYPI allows you to install additional packages easily



Touching a specific package reveals a yellow download and install link

The End
Of
The Beginning

Structure of a Computer Program



Processing:

Series of logical steps

Introduction to Python 3.8

I hear and I forget. I see and I remember. I do and I understand.

Confucius

❖ To print something: `print ("Hello! 20MS-batch")` # Anything following # is a comment

❖ To clear the screen: Ctrl-L

❖ Data Types: Integers: 4, 7, -8

Floats: numbers such as 12.8, 4.0, 5.3459

Strings: characters enclosed by single or double quotes

❑ Note: Everything is input as string, even numbers

❑ If you want to use the user input as a number, you need to convert it to an integer or a float data type.

Introduction to Python 3.8: Arithmetic Operations

- ❖ Store a number: `x = 1`
- ❖ Add two stored numbers: `x + y`
- ❖ Multiply two numbers: `x * y`
- ❖ Get nth power of x : `x**n`
- ❖ Convert an integer to float: `float(x)` # Example: `float(9)=9.0`
- ❖ Get integer division of integers p and q : `p/q` # Example: `9/4` gives 2.25
- ❖ Get fractional division of integers p and q : `float(p)/q` # Example: `9.0/4` gives 2.25

Introduction to Python 3.8

❖ Input: Information required to solve a computational problem.

❖ Form of the input statement: `input()` # Takes input given on screen by the user

```
st = input()           # expects an input which will be stored as a string in st
print('The input string is:', st)  # Prints the string input by the user
```

#Can be a confusing for the user since the user is not explicitly told that an input is expected

```
st = input('Enter a string')      # tells the user that an input is expected
print('The input string is:', st)  # Prints the string input by the user
```

❖ Example: Multiplication of 2 numbers

```
a=float(input('first number:'))
b=float(input('second number:'))
print("The product is:",a*b)
```

The curious case of the = sign

❖ In the following piece of code, can you guess the values of x and y that will be output by the print statements?

```
x=31  
y=47  
x=y  
print('x=',x)  
y=98  
y=x  
print('y=',y)
```


Introduction to Python 3.8: Arithmetic Operations

a = 21

b = 10

c = 0

c = a + b # ➡ Value of c is 31

c += a # ➡ c=c+a ➡ Value of c is 52

c *= a # ➡ c=c*a ➡ Value of c is 1092

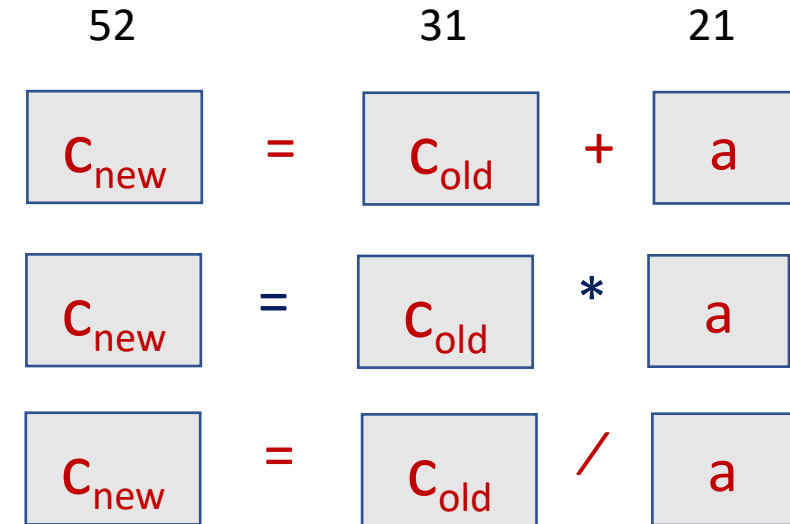
c /= a # ➡ c=c/a ➡ Value of c is 52

c = 200

c %= a # ➡ c=c%a ➡ Value of c is 11 (Remainder when c is divided by a)

a=3; c **= a # ➡ c=c**a ➡ Value of c is $11^3 = 1331$

c //= a # ➡ c=c//a ➡ Value of c is 443 (Quotient when c is divided by a)



Introduction to Python 3.8: Numerical manipulations

❖ Convert number to string: `test=12345`

`z=str(test)` # converts the variable test into a col. Matrix where each digit corresponds to an element in the column matrix. `z[0]=1; z[4]=5` etc

NOTE: `z[i]` are no longer considered to be integers but characters. `z[0]+z[1]≠3`; `z[0]+z[1]=12`

❖ Convert to integer: `int(1.2)` # returns the integer 1

❖ Round off a float: `round(1.3456,2)` # returns the number correct to 2 decimal places i.e. 1.35

❖ To determine the type of a variable `z`: `print(type(z))`

❖ Create list of numbers: `numlist=[1, 2, 4, 2, 5]`

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' ]
```

- ❖ n^{th} 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 $(m+1)^{\text{th}}$ 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

❖ mth to nth 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

More String Operations: Finding

`str="firefox is the name of the mozilla browser"`

❖ **Task:** verify if the word mozilla is in the above string using the find command

`result=str.find('mozilla')` # returns the position of the string after which the word "fox" appears i.e.27. Note that spaces are counted as well and counting starts from 0

❖ **Task:** verify if the word 'wolf' is in the above string using the find command

`result2=str.find('wolf')`

`result2 = -1` since the word wolf is absent from the string

Introduction to Python 3.8: *Recap and* Things to do

- ❖ Define variables as integer, float, etc.
- ❖ Perform basic mathematical operations using the defined variables
- ❖ Create a list which can be a mixed list or a list of numbers only or a list of words only.
- ❖ A list is like a 1-dimensional array/matrix.
- ❖ Extract elements from the list
- ❖ Concatenate two or more lists
- ❖ Perform basic mathematical operations on lists
 - ❖ Identify location of a specific element in the list
 - ❖ Sum the elements in a numerical list
 - ❖ Find the maximum and minimum from a numerical list
 - ❖ Reverse the order of elements in the list
 - ❖ Sort the elements in a list in ascending or descending order
 - ❖ Add or remove elements from a list
 - ❖ Generate nested lists i.e. multi-dimensional arrays/matrices
 - ❖ Convert a string to a list
 - ❖ Convert a list to a string

I hear and I forget. I see and I remember. I do and I understand.

Confucius

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]

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]
```

index position of the inserted number

Number to be inserted

```
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: Nested Lists

❖ *Nested Lists* or multi-dimensional arrays (matrices)

An ordinary list is a one dimensional array. Also possible to represent multi-dimensional arrays by creating a nested list (i.e. a list of lists)

NLst=[[1,2,3],[4,5,6]] is equivalent to the 2x3 matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

`print (NLst[0])` # gives [1,2,3] i.e. elements in the first row of the 2x3 matrix

`print (NLst[0][0])` # gives 1 i.e. the element in the first row and first col.

`print (NLst[1][2])` # gives 6 i.e. the element in the second row and third col

Generic Form: NLst[m][n] gives the element in the (m+1)th row and (n+1)'th col

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: *Recap and Things to do*

- ❖ Define variables as integer, float, etc.
 - ❖ Perform basic mathematical operations using the defined variables
 - ❖ Create a list which can be a mixed list or a list of numbers only or a list of words only.
 - ❖ Manipulate lists and strings
 - ❖ Extract elements, remove elements, find elements, reverse order of elements etc.
 - ❖ Perform mathematical operations on lists
-
- ❖ Loops, Control structures, Functions, Dictionaries, Class and Object in Python
 - ❖ Perform repeated operations (**for** and **while** loop)
 - ❖ Perform conditional operations (**if...elif...else**)
 - ❖ Define **functions** to automate repeated computing tasks
 - ❖ Define **class** and **object** and extract information from them
 - ❖ Importance of algorithm for specifying the logical sequence of operations
 - ❖ Write simple python programs using the above defined structures

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: ifelif..... 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)
```


Sum of integers from 1 to n

$$\begin{array}{r} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ \hline 55 \end{array}$$

$$\sum_{x=1}^{x=n} x = \frac{n(n+1)}{2}$$

Algorithm

- ❖ Add the first 2 numbers
- ❖ Store result in your memory
- ❖ Add third number to result
- ❖ Store new result in your memory
- ❖ Repeat until you have added the last number to generate the final sum

Repeated operations: for loops

Algorithm

- ❖ Add the first 2 numbers
- ❖ Store result in your memory
- ❖ Add third number to result
- ❖ Store new result in your memory
- ❖ Repeat until you have added the last number to generate the final sum



Algorithm

- ❖ Initialize storage counter to 0
- ❖ Add the first 2 numbers
- ❖ Store result in your memory
- ❖ Add third number to result
- ❖ Store new result in your memory
- ❖ Repeat until you have added the last number to generate the final total

- ❖ Example of a for loop in action: Sum of positive integers from 1 to n=10

```
n = int(input("n = ")) # Get integer up to which sum is to be calculated
total=0                # initialize storage counter to store sum
for x in range(1,n+1): # loop over integers starting from 1 and ending at 10
    total+=x           # perform addition
print (total)          # print result
```

Sum of integers *without* the *for* loop

❖ **Task:** To calculate the sum of all integers up to the number specified by the user

```
n = int(input("n = ")) # Get the integer up to which the sum needs to be calculated from user
```

Adding all integers from 1 to n by first generating a list x of integers and then using #the sum function on the list x

```
x=range(1,n+1) # produces a numerical list of integers from 1 to 10  
print (sum(x))
```

#Another alternative way of finding the sum of integers using a *single* line

```
print (sum(range(1,n+1)))
```

❖ Example of *for* (list comprehension)

```
x = [i for i in range(5)] # generates a list of integers starting from 0 to 4
```

Output: x=[0,1,2,3,4]

```
y = [i**2 for i in range(5)] # generates a list of squares of integers starting from 0
```

Output: x=[0,1,4,9,16]

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

while loop example: Sum of integers with **break** in loop

❖ **Task:** To calculate the sum of all integers up to the number specified by the user

```
n = int(input("n = ")) # Get the integer up to which the sum needs to be calculated
```

```
x=1 # Initialize counter for keeping track of summation
```

```
total = 0 # Initialize a variable for keeping the total
```

```
# Next loop iterates over all integers from 1 to N
```

```
while x<n+1:
```

```
    total += x # At each stage, the current integer is added to total
```

```
    print x
```

```
    if (x==10):
```

```
        break # break out of the while loop and terminates the summation after the first 10 integers
```

```
    x+=1 # counter is incremented by 1
```

```
print ("The calculated total is", total # Print the calculated total)
```

```
print ("The summation has been carried out for the first %d integers" % (x))
```

```
print ("The expected total based on the user input is", n*(n+1)/2)
```

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 in 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='11110011001011'
```

```
# 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', 'o', '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
```

The Standard Genetic Code

		Second base					
		U	C	A	G		
First base 5'	U	UUU } Phenyl-alanine UUC } UUA } Leucine UUG }	UCU } UCC } Serine UCA } UCG }	UAU } Tyrosine UAC } UAA } Stop codon UAG } Stop codon	UGU } Cysteine UGC } UGA } Stop codon UGG } Tryptophan	Third base 3'	U C A G
	C	CUU } CUC } Leucine CUA } CUG }	CCU } CCC } Proline CCA } CCG }	CAU } Histidine CAC } CAA } Glutamine CAG }	CGU } CGC } Arginine CGA } CGG }		U C A G
	A	AUU } Isoleucine AUC } AUA } AUG } Methionine start codon	ACU } ACC } Threonine ACA } ACG }	AAU } Asparagine AAC } AAA } Lysine AAG }	AGU } Serine AGC } AGA } Arginine AGG }		U C A G
	G	GUU } GUC } Valine GUA } GUG }	GCU } GCC } Alanine GCA } GCG }	GAU } Aspartic acid GAC } GAA } Glutamic acid GAG }	GGU } GGC } Glycine GGA } GGG }		U C A G

❖ Creating a Dictionary using the dict() function

```
sgc=[('UUU','F'), ('UUC','F'), ('UUA','L'), ('UUG','L'), ('CUU','L'), ('CUC','L'), ('CUA','L'), ('CUG','L'), ('AUU','I'), ('AUC','I'), ('AUA','I'), ('AUG','M'), ('GUU','V'), ('GUC','V'), ('GUA','V'), ('GUG','V')]
```

```
print (dict(sgc))
```

generates the following dictionary : as specified by a mapping between codons and amino acids

```
{'AUA': 'I', 'GUC': 'V', 'CUU': 'L', 'GUG': 'V', 'GUU': 'V', 'AUG': 'M', 'UUU': 'F', 'GUA': 'V', 'AUC': 'I', 'CUG': 'L', 'CUC': 'L', 'CUA': 'L', 'AUU': 'I', 'UUG': 'L', 'UUA': 'L', 'UUC': 'F'}
```

NOTE about syntax: If the values of the keys are also characters or strings, they should be enclosed within ''

Dictionaries: More examples

```
X=[('a',2), ('b', 4), ('c', 6)]    # also a list of pairs that can be converted to a dictionary
print (dict(X))                  # generates the dictionary {'a': 2, 'c': 6, 'b': 4}
```

❖ Using dictionary to generate the values of $\sin(\theta)$ for a given set of θ values; key: θ , value: $\sin(\theta)$

```
from math import pi, sin
```

```
theta = [0, pi/2, pi, 3*pi/2, 2*pi]
```

```
sine = {x:sin(x) for x in theta}
```

```
print (sine[pi])
```

```
# create a list of theta values
```

```
# generates the dictionary by looping over x in theta
```

```
# prints the value associated with the key:pi
```

Don't forget to Use

Square brackets [...] for Lists

Round brackets (...) for Tuples

Curly brackets {...} for Sets & Dictionaries

Defining a Function in Python 3.8

- ❖ Define functions to avoid code repetition and for simplicity
- ❖ Functions should have at least 1 argument and should return a value when called
- ❖ Functions can have multiple arguments
- ❖ Functions can have conditional return statements
- ❖ Functions usually have a return statement
- ❖ Functions without a return statement is executed but returns None

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

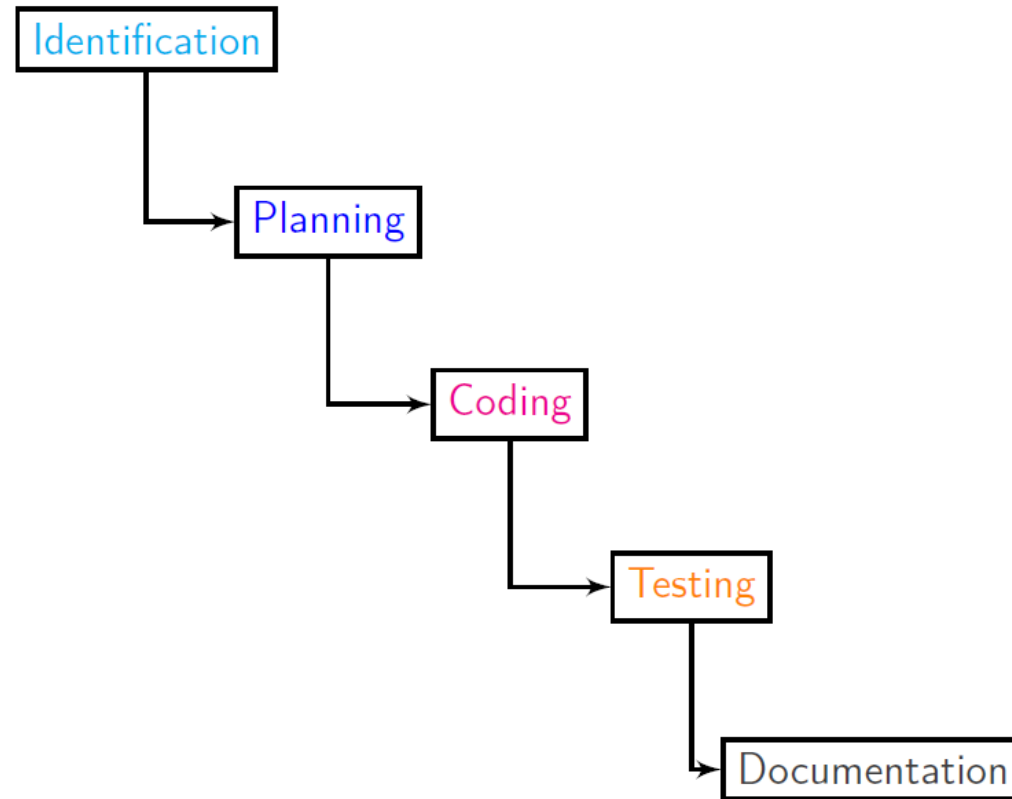
```
def square_add(x,y): # A function with 2 arguments
    z = x**2 + y**2
    return z # returns the sum of squares of the 2 arguments
>>> square_add(3,4) # gives the output 25
```

❖ Functions can return multiple results

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

Program development phases

Identification → Planning → Coding → Testing → Documentation.



Identifying the problem

To determine the requirement of the program.

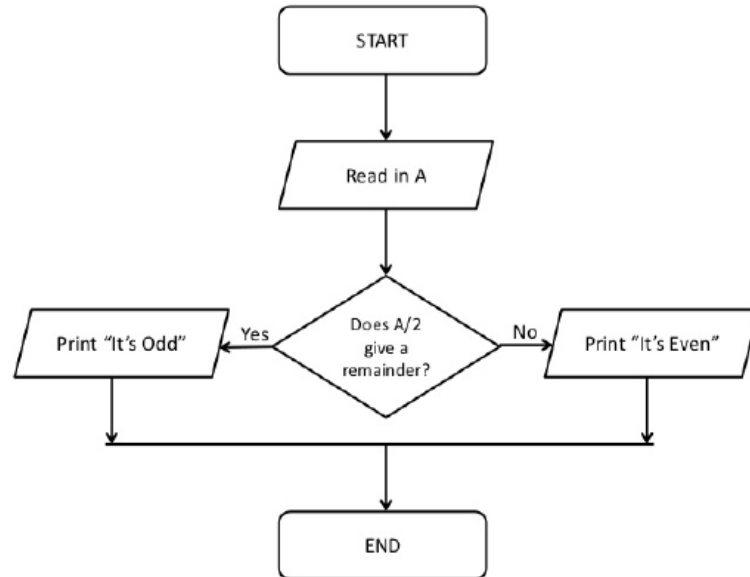
- what would be the input(s)?
- what additional tools would be needed? Eg: do you need to import functions from math module or some other modules?
- what would be the output(s)?

Planning a solution

Two ways of planning the solution to a problem:

Flow chart

a step-by-step instruction.



Pseudo code

listing down set of instructions.

```
procedure odd_even_number(number)

    check if number divisible by 2

    IF divisible
        RETURN "Even"
    ELSE
        RETURN "Odd"
    END IF
```

Coding the program

- Consult the *planning* (flow chart or pseudo code);
- Write set of instructions (**the code**) for performing the task in the desired language following the grammar (syntax / **set of rules**) of that language;
- Fix all grammatical errors (**syntax errors**).

Testing

- ① **desk checking** (**dry run**) - the programmer mentally checks the logic of the program.
- ③ **debugging** - detecting, locating and correcting bugs, either an error or a mistake.
(**most popular method for fixing errors.**)

Documentation

- A detailed description on how the program was created by the programmer himself.
- Contains a brief narrative process undergone by the program.
- ★ Necessary for future use of the code by another programmer who will upgrade the program.

Some additional tips about writing programs

Comment your code!

- ❖ Avoid using single-letter variable names
- ❖ Use variable names that can be searched easily
- ❖ Reduce use of variables for storage as much as possible, be frugal in using memory

```
for i in range(0,11):  
    j=i**2  
    print(i,j)
```

```
for i in range(0,11):  
    print(i,i**2)
```


Searching for Primes

Algorithm

- ❖ Get the number (n) from the user
- ❖ Run loop from 2 to ($n-1$) to check if the number n is divisible by any number other than itself.
- ❖ If divisible by any of the numbers between 2 to ($n-1$), then the number n entered is not a prime; exit the program
- ❖ Else it is a prime number

Code

```
import sys
n = input('Enter number: \n')

for i in range(2,n):
    if n%i==0:
        print (n, 'is not a prime number')
        sys.exit() # exits from the program
print (n, 'is a prime number')
```

`sys` module contains functions that are used to manipulate the python runtime environment

Why won't the 'break' command work instead of `sys.exit()` ?

Hint: Try replacing `sys.exit()` with `break` and see what you get as an output if the number input is *not* a prime

Mean, Variance, Correlation Coefficient

Mean: $\langle x \rangle = \frac{1}{n} \sum_{i=1}^n x_i$

Variance: $\langle x^2 \rangle - \langle x \rangle^2 = \frac{1}{n} \sum_{i=1}^n x^2 - \left(\frac{1}{n} \sum_{i=1}^n x \right)^2$

Code

```
import math
x=[2,3,0,1,-2,5]
y=[i**2 for i in x]
n=len(x)

mean=float(sum(x))/n
var=float(sum(y))/n - mean**2

stdev=math.sqrt(var)
print 'Mean, Variance, Standar Deviation = ', mean,
var, stdev
```

Correlation Coefficient

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \langle x \rangle)(y_i - \langle y \rangle)}{\sqrt{\sum_{i=1}^n (x_i - \langle x \rangle)^2} \sqrt{\sum_{i=1}^n (y_i - \langle y \rangle)^2}}$$

Code

```
import math
x=[2,4,5,6,8]
y=[5,9,11,10,12]
n=len(x)
xav=float(sum(x))/n
yav=float(sum(y))/n
sigx=sqrt(sum([(i-xav)**2 for i in x]))
sigy=sqrt(sum([(i-yav)**2 for i in y]))
covxy= sum([(i-xav)*(j-yav) for i,j in zip(x,y)])
corr=covxy/(sigx*sigy)
print ('Correlation coefficient=', corr)
```

Mean, Variance, Correlation Coefficient

```
import math
x=[2,4,5,6,8]
y=[5,9,11,10,12]
zip(x,y)
# generates the pairs [(2,5), (4,9), (5,11), (6,10), (8,12)]
```

❖ Calculate correlation coefficient using numpy

```
import numpy as np
x=np.array([2,4,5,6,8]) #convert the data into lists
y=np.array([5,9,11,10,12])

r=np.corrcoef(x,y) # corrcoef is a function available in numpy to calculate CC
print ('Correlation coefficient=', corr)
```

Correlation Coefficient

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \langle x \rangle)(y_i - \langle y \rangle)}{\sqrt{\sum_{i=1}^n (x_i - \langle x \rangle)^2} \sqrt{\sum_{i=1}^n (y_i - \langle y \rangle)^2}}$$

To check if a number is a perfect square

Algorithm

- ❖ Get the number (*n*) from the user
- ❖ Extract the integer part the square root of the number. Example:
If number is 27, $\text{nsqrt}=\text{int}(\text{sqrt}(27))=5$
If the number is 36, $\text{nsqrt}=\text{int}(\text{sqrt}(36))=6$
- ❖ Check if the square of nsqrt = number input by the user
If *true*, the number input is a perfect square
If *false*, the number is not a perfect square
- ❖ Example: For 27, $\text{nsqrt}^2=25 \neq 27 \rightarrow$ not a perfect square

For 36, $\text{nsqrt}^2=36 \rightarrow$ a perfect square

Code

```
from math import *

num=int(input("Enter a number: \n"))
nsqrt=int(sqrt(num))
nsq=nsqrt**2

if nsq == num:
    print ('The number is a perfect square')
else:
    print ('NOT a perfect square')
```

Armstrong Numbers

Algorithm

- ❖ Convert the number to a string
- ❖ Initialize (to 0) a counter to store the sum of the cubes of the individual digits
- ❖ Loop over all the characters in the string
- ❖ Extract each character, convert it to a numerical value using the `eval` function.
- ❖ Calculate the cube of the digit and keep adding recursively to the sum
- ❖ Check if the sum of cubes = number input
If `true`, it is an Armstrong number
If `false`, not an Armstrong number

Code

```
#An Armstrong number is a number for which the sum of cubes of the digits
#of the number is the number itself. Eg.: 153=1**3 + 5**3 + 3**3
#Code to determine whether a given number is an Armstrong number

num=input('Enter a number: \n')
sum=0
for i in num:
    k=eval(i) #convert each digit to a number starting with the first digit
    sum+=k**3
if sum==num:
    print (num, 'is an Armstrong number')
else:
    print (num, 'is NOT an Armstrong number')
```

Armstrong Number Generator

Algorithm

- ❖ Initialize (to 0) a counter to store the sum of the cubes of the individual digits
- ❖ Loop over first 100000 integers
- ❖ Rest of the algorithm is same as before
- ❖ If sum of cubes = number encountered within the loop, print that number as it is an Armstrong number

Code

```
#Code to generate Armstrong numbers
#sum=0
print ('Armstrong numbers are:\n')
for num in range(1,100000):
    sum=0
    # convert to a string where each digit is a character
    numstr=str(num)
    for i in numstr:
        k=eval(i) #convert each digit to a number
        sum+=k**3
    if sum==num: print num
```

Palindrome Numbers

Algorithm

- ❖ Specify the lower and upper bound of the range within which you want to identify Palindrome numbers
- ❖ Initialize counter (to 0) to calculate the total number of Palindrome numbers within the range
- ❖ Loop over the range
- ❖ Convert the number into a string where each digit is a character
- ❖ Reverse the order of digits in the string
- ❖ If original string = reversed string, increment counter by 1 and print the Palindrome number
- ❖ After the loop ends, print the Total

Code

```
#Code to generate Palindrome numbers
#Sequence of digits from left to right is the same
#as the sequence of digits from right to left
#Example: 49594

print 'Palindrome numbers are:\n'
total=0
Ubound=500 # specify upper bound of range for finding palindromes
Lbound=10 # specify lower bound of range for finding palindromes

for num in range(Lbound,Ubound):
    numstr=str(num) # convert to a string where each digit is a character
    revstr=numstr[::-1] #reverses the order of the digits
    if numstr==revstr:
        total+=1
        print (numstr)

print ('Total number of Palindromes between 10 and', Ubound;) print
('are:',total)
```

Collatz Conjecture and Collatz Sequence

For **any** positive integer n , generate the **next term** according to the following algorithm

$n/2$, if n is even

$3n+1$, if n is odd

Repeat to generate a sequence called the Collatz Sequence

The **Collatz conjecture** is that the sequence of numbers will end in 1 regardless of the initial choice of the positive integer

Example: 5, 16, 8, 4, 2, 1

Code

```
# Code to generate a Collatz sequence
num=int(input('Enter a number: \n'))

print 'The Collatz sequence is:'
print (num) # first number of the sequence

while num>1:    # repeat as long as number >1
    if num%2==0: # if even
        num=num/2
        print (num)
    else:        # if odd
        num=3*num+1
        print (num)
```


Bubble Sort (To sort a list of numbers in ascending order)

Algorithm

- ❖ Outer loop goes over the full set of numbers in the list
- ❖ For each value of the outer loop variable, compare two adjacent numbers and swap if needed.

In the **first** pass, the **largest number** rises to the top.

In the **second** pass (i.e. second value of the outer loop variable), adjacent numbers need to be compared and swapped if needed only till the **second-last** number. **Second largest** number rises to the position just below the largest number.

In the **third** pass, adjacent numbers need to be compared and swapped if needed only till the **third-last** number.

- ➔ Upper bound of the inner loop is $n-i-1$
- ❖ Completion of the process leads to proper ordering of the entire list

First Pass		
[6, 2, 9, 0, 1]	→	[2, 6, 9, 0, 1] Swap
[2, 6, 9, 0, 1]	→	[2, 6, 9, 0, 1] No swap
[2, 6, 9, 0, 1]	→	[2, 6, 0, 9, 1] Swap
[2, 6, 0, 9, 1]	→	[2, 6, 0, 1, 9] Swap
Second Pass		
[2, 6, 0, 1, 9]	→	[2, 6, 0, 1, 9] No swap
[2, 6, 0, 1, 9]	→	[2, 0, 6, 1, 9] Swap
[2, 0, 6, 1, 9]	→	[2, 0, 1, 6, 9] Swap
Third Pass		
[2, 0, 1, 6, 9]	→	[0, 2, 1, 6, 9] Swap
[0, 2, 1, 6, 9]	→	[0, 1, 2, 6, 9] Swap
Fourth Pass		
[0, 1, 2, 6, 9]	→	[0, 1, 2, 6, 9] No Swap

Code

```
L=[6,2,9,0,1]
n=len(L)
for i in range(n):
    for j in range(0,n-i-1):
        if L[j]>L[j+1]:
            L[j], L[j+1] = L[j+1], L[j] # swap numbers
print ('The list sorted in ascending order is', L)
```

Evaluating a Series

$$\cos(x) = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \dots \infty = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

$$T_n = (-1)^n \frac{x^{2n}}{(2n)!} \quad T_{n-1} = (-1)^{n-1} \frac{x^{2n-2}}{(2n-2)!} \quad T_0 = 1 \quad \frac{T_n}{T_{n-1}} = -\frac{x^2}{2n(2n-1)}$$

```
from math import pi, factorial
N=10
# evaluating series expansion of cos(60) for first N terms
theta=60
x=theta*pi/180 # convert degree to radian
print (sum([float((-1)**i*x**(2*i))/factorial(2*i)) for i in range(0,N)])
```

Code: By list comprehension

#Terms	Estimated value
N=1:	cos(60) = 1.0
N=2:	cos(60) = 0.451688644384
N=3:	cos(60) = 0.5017962015
N=4:	cos(60) = 0.499964565329
N=5:	cos(60) = 0.500000433433
N=6:	cos(60) = 0.499999996391

abs(exact value – estimated value)
0.5
0.048311355616
0.0017962015
0.000035434671
0.000000433433
0.000000003609

Evaluating a Series

$$T_n = (-1)^n \frac{x^{2n}}{(2n)!} \quad T_{n-1} = (-1)^{n-1} \frac{x^{2n-2}}{(2n-2)!} \quad T_0 = 1 \quad \frac{T_n}{T_{n-1}} = -\frac{x^2}{2n(2n-1)}$$

Algorithm

- ❖ Input the angle and accuracy up to which the cos(angle) needs to be evaluated
- ❖ Convert angle in degrees to radians
- ❖ Initialize counters, sum (to 0) and first term in the series (to 1)
- ❖ Initialize counter (n) for number of iterations
- ❖ While evaluated value > desired accuracy, keep updating term and add it to the original series (sum)
- ❖ After exiting while loop, print sum (i.e. evaluated series up to desired accuracy) and exact value of the cosine function for angle input

Code: By evaluating accuracy term by term

```
import math
x, tol=input('Angle in degree, tolerance: \n')
x=x*math.pi/180 # convert angle to radians
sum, term=0, 1 # initialize counters
n=1

while abs(term) > tol:
    sum+=term
    term=term*(-x**2/(2*n*(2*n-1)))
    n+=1

print (sum, math.cos(x))
```

Tolerance	Estimated	Exact
tol=0.1:	cos(60) = 0.451688644384	0.5
tol=0.01:	cos(60) = 0.5017962015	0.5
tol=0.001	cos(60) = 0.499964565329	0.5
tol=0.0001:	cos(60) = 0.499964565329	0.5
tol=0.00001:	cos(60) = 0.500000433433	0.5

Tolerance	Estimated value
tol=0.1➡	correct to first decimal place
tol=0.01➡	correct to second decimal place
tol=0.001➡	correct to third decimal place
tol=0.0001➡	correct to fourth decimal place
tol=0.00001➡	correct to fifth decimal place

Formatted Output and Writing to a File

❖ **Task:** To generate data for an exponentially decaying function

`import math` # This imports the math module. Without this module, exponential function will not work

`A0=1050.0`

`tau=0.94` # define parameters for an exponentially decaying function

`outfile=open("exp-decay-generated.dat", "w")`

`for i in range(0,11,1):`

`f=A0*math.exp(-i/(2.0*tau))` # define exponentially decaying function

`# print (float(i)/2, f, int(f))` # prints without formatting

`print ("%3.1f\t %1.1f\t %d" %(i/2, f, int(f)))` # printing 3 cols. of data separated by tabs

`outfile.write("%3.1f\t %9.2f\t %4d\n" %(i/2.0, f, int(f)))` # formatted output to a file
`outfile.close`

Formatted **print** statement and writing to a File

```
print ("%1.1f\t %1.2f" %(i/2.0, f))
```

i/2	f
0.0	1050.00
0.5	616.85
1.0	362.39
1.5	212.90
2.0	125.07
2.5	73.48
3.0	43.17

```
print ("%3.1f\t %1.2f" %(i/2.0, f))
```

i/2	f
0.0	1050.00
0.5	616.85
1.0	362.39
1.5	212.90
2.0	125.07
2.5	73.48
3.0	43.17

```
print ("%6.1f\t %1.2f" %(i/2.0, f))
```

i/2	f
0.0	1050.00
0.5	616.85
1.0	362.39
1.5	212.90
2.0	125.07
2.5	73.48
3.0	43.17

Writing data to a specified output file

```
exp-decay-generated-test - Notepad
```

File	Edit	Format	View	Help
0.0		1050.00		1050
0.5		616.85		616
1.0		362.39		362
1.5		212.90		212
2.0		125.07		125
2.5		73.48		73
3.0		43.17		43
3.5		25.36		25
4.0		14.90		14
4.5		8.75		8
5.0		5.14		5

If total # digits < 6, spaces are added from the left.
Eg.: 3 spaces are added from the left in the first col

Formatting statement

```
outfile.write("%3.1f\t %9.2f\t %4d\n" %(i/2.0, f, int(f)))
```

Anatomy of a formatted write statement

```
outfile=open("exp-decay-generated.dat", "w")
```

name of output file

w → writing to the named file; every time the open statement is encountered, a fresh file with the given name will be created and any existing file with the same name will be overwritten; use "a" instead of "w" to append to an existing file instead of overwriting it

File identifier, can be given any other name like myfile, datafile, thisfile, thatfile etc.

```
outfile.write("%3.1f\t %9.2f\n" %(i/2.0, f))
```

Specify data
type for variable
i as float

Total # digits including decimal points
allocated for the variable f

digits after decimal points

If total # digits < 9, spaces
are added from the left as
required

```
outfile.close
```

More for loop examples: Generate data for parabola plotting

- ❖ **Task:** To generate data for plotting a parabola $y=x^2$ symmetric about the y-axis
- ❖ The data points along the X-axis take on integer values only according to the code below

```
outfile=open("parabola-data2.dat", "w")
```

```
for x in range(-20, 21, 1):  
    outfile.write("%6.2f\t %9.2f\n" %(x, x**2))
```

```
outfile.close
```

NOTE: outfile is the identifier for the file in the program while the name of the file in quotes "parabola-data2.dat" is the file that will be created in the folder where you are running the program.

You can give any name to the file identifier in the program i.e. outfile can be replaced by names such as myfile, datafile, thisfile, thatfile etc. But whatever name you give should be the same as that is used in subsequent write and close statements

Every time the open statement is encountered with "w", a new and empty file called "parabola-data2.dat" will be created in the same folder and subsequently filled with data when outfile.write statement is encountered.

Formatted Output

```
print ("%5.2f %4.2f\n" %(x, x**2))
```

x	x**2
-2.00	4.00
-1.00	1.00
0.00	0.00
1.00	1.00
2.00	4.00

There is just *1 space* between the first and second columns in this formatted output

```
print ("%5.2f %12.2f\n" %(x, x**2))
```

x	x**2
-2.00	4.00
-1.00	1.00
0.00	0.00
1.00	1.00
2.00	4.00

There are *8 more spaces* between the first and second columns in this formatted output.

The total space allocated for the second col. data is 12. Actual space used is 4 → each of the numbers are padded from the left by 8 spaces

Defining a Range Generator

❖ **Task:** To generate data for plotting a parabola $y=x^2$ symmetric about the y-axis

```
start = int(input("starting value = "))
```

```
end = int(input("ending value = "))
```

```
step = float(input("step size = "))
```

```
def myrange(start, end, step): # range generator; creates a list of values to iterate over
```

```
    while start <= end:
```

```
        yield start
```

```
        start += step
```

```
outfile=open("parabola-data.dat", "w")
```

```
for x in myrange(start, end, step):
```

```
    outfile.write("%6.2f\t %9.2f\n" %(x, x**2))
```

```
outfile.close
```

Random numbers in Python

- ❖ To generate random numbers in a Python code, the `random` module has to be imported using the command `import random`
- ❖ Multiple types of random numbers can be generated using the `random` module. One of the most common is a *uniformly distributed* random number between 0 and 1.
- ❖ The code segment
`import random`
`random.random()` # generates a *uniformly distributed* random number between 0 & 1
- ❖ The code segment prints 100 uniformly distributed random number between 0 & 1 to an output file 'unirand.dat'

```
fran=open("unirand.dat", "w")
```

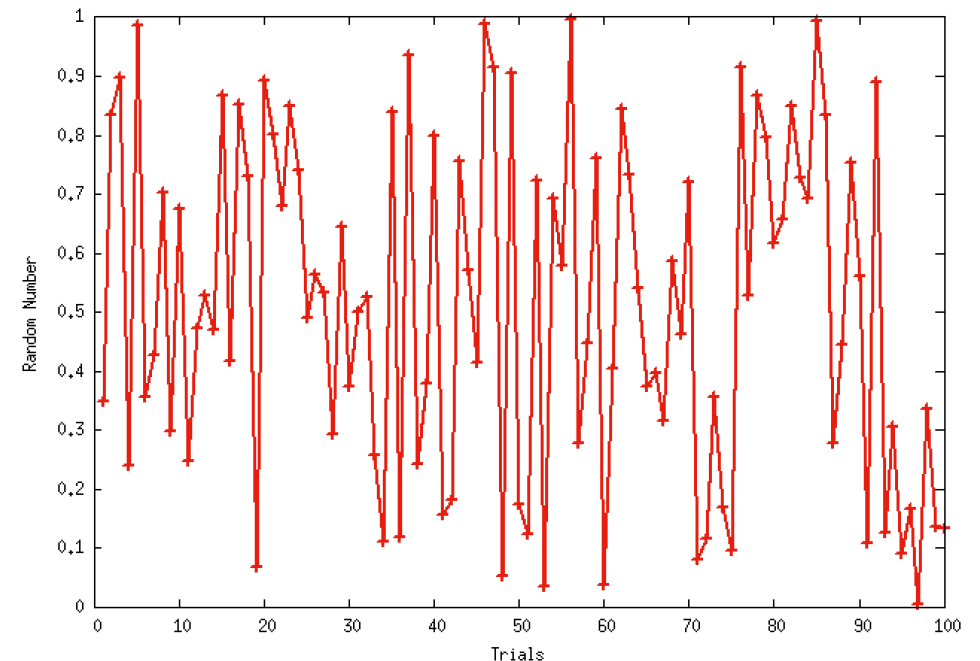
```
import random
```

```
for i in range(1,101):
```

```
    r=random.random()
```

```
    fran.write ("%d\t%f\n" %(i, r))
```

```
fran.close()
```



Random numbers in Python

❖ Some other types of random numbers that can be generated using the `random` module:

`uniform(a,b)` : uniformly distributed random number between `a` & `b`

`randint(a,b)` : uniformly distributed random integers between `a` & `b`, including end points `a,b`

`gauss(μ , σ)` : Gaussian (Normal) distributed random numbers with `Mean`= μ and `Standard dev.` = σ

❖ The code segment prints a list of 50 random integers between 1 & 50 to an output file 'randint.dat'

```
fran=open("randint.dat", "w")
```

```
import random
```

```
for i in range(50):
```

```
    ri=random.randint(1,50)
```

```
    fran.write ("%d\t %d\n" %(i, ri))
```

```
fran.close()
```

Output:

```
[23, 9, 22, 33, 41, 16, 24, 7, 1, 32, 19, 23, 15, 2, 18, 40, 14, 11, 38, 37, 32, 8, 13, 3, 37, 17, 38, 36, 33, 22, 49, 19, 14, 21, 26, 11, 9, 27, 17, 32, 44, 4, 23, 19, 16, 42, 22, 36, 16, 45]
```

Numbers in `red` are repeated and appear multiple times in the list

Random numbers in Python

❖ The code segment prints a list of 50 *unique* random integers between 1 & 50

```
import random
```

```
total=0                # initialize counter to keep track of #unique random integers generated
```

```
num=50                 # upper bound of range
```

```
rintlist=[]           # define an empty list
```

```
while total<num:       # generate 50 random integers between 1 and 50
```

```
    r=random.randint(1,num)
```

```
    if (r in rintlist)==False:    # check if random integer generated is already in list
```

```
        rintlist.append(r)        # append to list if there is no repetition
```

```
        total+=1                 # increment counter
```

```
print ('total=',total)
```

```
print ('rintlist=',rintlist)
```

```
rintlist= [30, 22, 5, 33, 20, 29, 19, 16, 28, 11, 13, 15, 14, 37, 31, 24, 47, 3, 2, 46, 48, 7, 45, 40, 26, 41, 10, 27, 32, 8, 50, 23, 4, 17, 25, 42, 9, 49, 1, 12, 39, 44, 35, 38, 18, 43, 34, 6, 36, 21]
```

What is the probability of getting a *head* when you toss a coin ?

Is the probability of getting a *head* in a coin-toss experiment always $\frac{1}{2}$?

- ❖ Mimic a coin toss experiment by generating a uniformly distributed random number between 0 & 1
- ❖ If the random number generated is ≤ 0.5 , we call it a *HEAD*, otherwise, it is a *TAIL*
- ❖ Generate many random numbers and calculate the fraction (f_H) of times, we get a *HEAD*.
- ❖ How does f_H depend on the number of *trials* (i.e number of times a random number is generated)



$f_H=1$ $f_H=0.5$ $f_H=0.67$ $f_H=0.75$ $f_H=0.8$ $f_H=0.83$ $f_H=0.71$ $f_H=0.625$ $f_H=0.67$ $f_H=0.6$

Algorithm

- ❖ Import random module, open output file for writing, specify number of trials, create an empty list, initialize headcount counter
- ❖ Loop over number of *trials*
 - ❖ Calculate fraction of heads
 - ❖ Generate a *uniformly distributed random number* between 0 & 1
 - ❖ If the random number generated is ≤ 0.5 ,
 - ❖ increment *HEAD* count by 1
 - ❖ update fraction of heads up to that *trial*
 - ❖ Append f_H to a list that keeps track of how f_H changes with number of *trials*
 - ❖ *Write* number of *trials* and fraction of heads to an output file
- ❖ Close output file

Is the probability of getting a head in a coin-toss experiment always $\frac{1}{2}$?

Code

```
import random
hfrac=open('head-frac.dat', 'w')
n=1000                # Specify number of trials
headlist=[]           # a list which lists the changing fraction of heads as #trials increases
head=0                # initialize counter for calculating number of heads in n trials

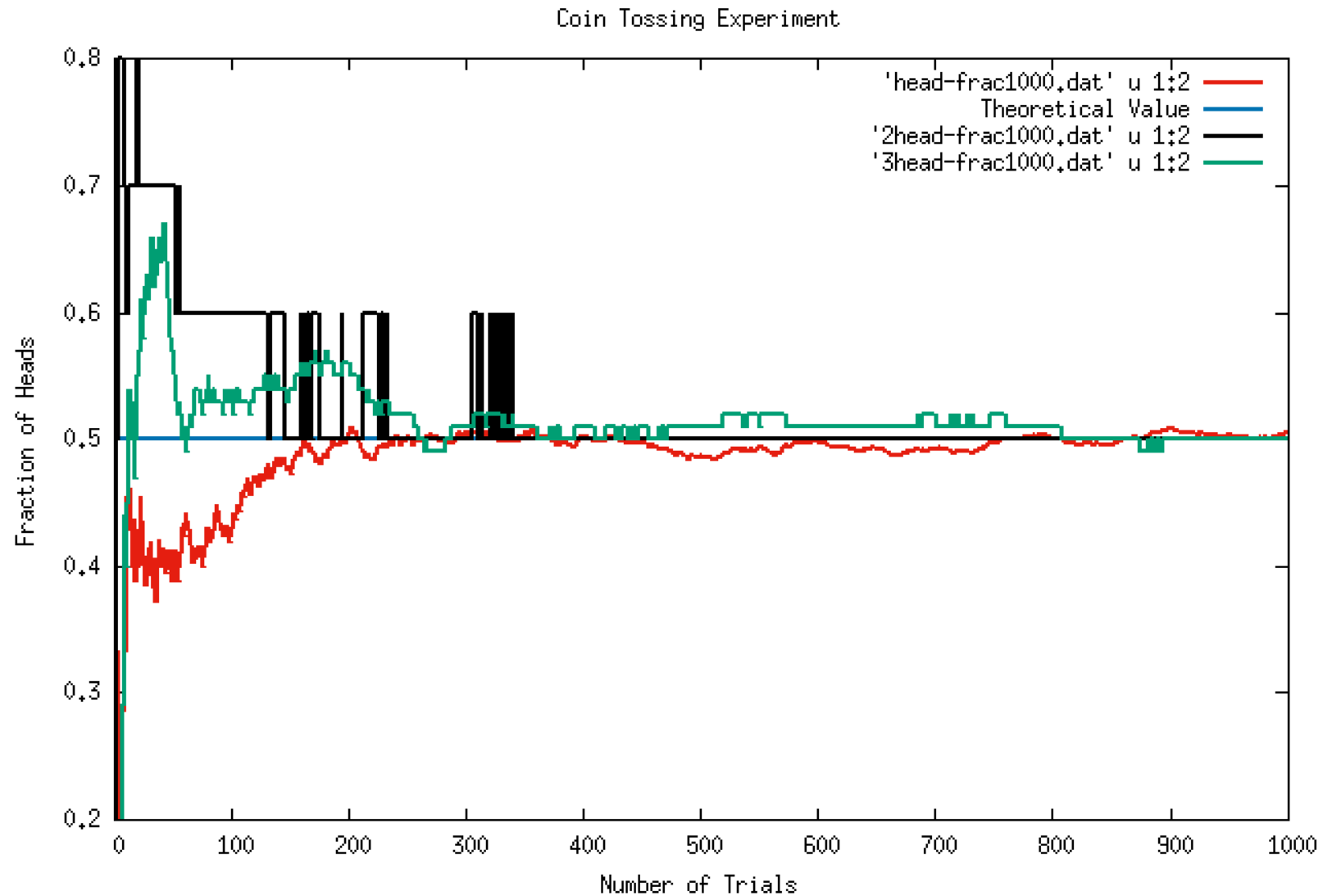
for i in range(1,n+1):
    frachead=float(head)/i

    if random.random()<=0.5:        # generated random number qualifies for classification as head
        head+=1                    # increment head count by 1
        frachead=float(head)/i      # update fraction of heads up to that trial

    headlist.append(frachead)        #append fraction of heads to headlist
    hfrac.write("%d\t%f\n" %(i, frachead)) # write to a file, formatted output

#print (headlist)                  # List tracks changes in fraction of heads with #trials
print (head,frachead)              # The number & fraction of heads after n trials
hfrac.close()
```

Can you explain the reason for the difference between the 3 graphs ?



The algorithms used to generate the 3 data sets was identical

How different ways of formatting output data can give very different graphs

Sample Output data from each of the 3 different sets

`hfrac.write("%d\t %f\n" %(i, frachead))`

1	0.000000
2	0.500000
3	0.333333
4	0.500000
5	0.600000
6	0.666667
7	0.571429
8	0.625000
9	0.666667
10	0.600000
11	0.545455
12	0.500000
13	0.538462
14	0.571429
15	0.600000
16	0.625000

`hfrac.write("%d\t %1.1f\n" %(i, frachead))`

1	0.0
2	0.5
3	0.7
4	0.8
5	0.8
6	0.8
7	0.9
8	0.8
9	0.7
10	0.7
11	0.6
12	0.7
13	0.7
14	0.7
15	0.7
16	0.7

`hfrac.write("%d\t %1.2f\n" %(i, frachead))`

1	0.00
2	0.50
3	0.33
4	0.50
5	0.40
6	0.33
7	0.29
8	0.38
9	0.44
10	0.50
11	0.45
12	0.50
13	0.54
14	0.50
15	0.47
16	0.50

❖ Fraction of heads is written with a different accuracy in each of the 3 output files

❖ Red: Default accuracy of 6 decimal places, Black: accuracy of 1 decimal place, Green: accuracy of 2 decimal places

➤ This leads to graphs showing very different behaviour even though the algorithm used for generating the output is identical in all 3 cases

Can you explain the reason for the difference between the 2 graphs ?

Both sets are generated with *identical* code and *identical* formatting

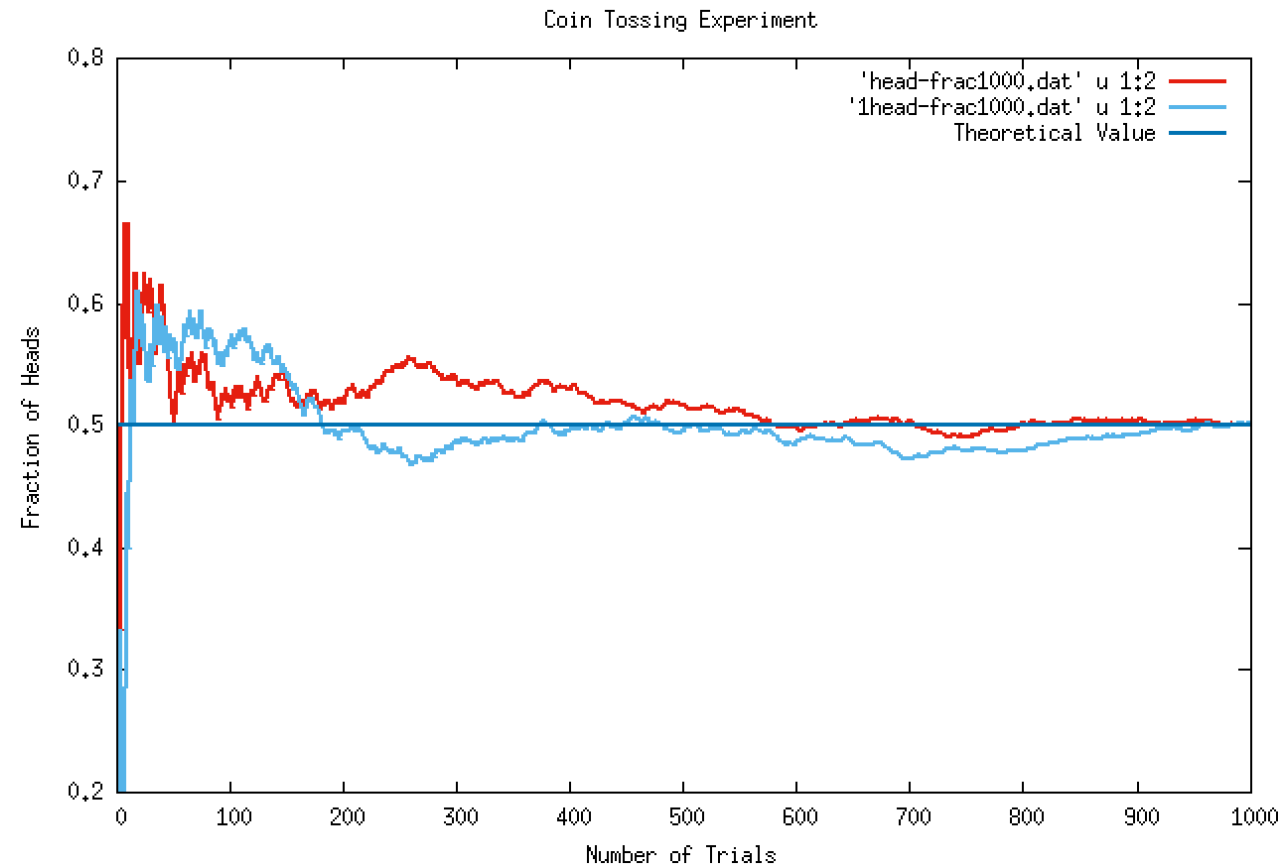
Sample data with identical formatting

Run 1 data

<u>Trial</u>	<u>f_H</u>
1	0.000000
2	0.500000
3	0.333333
4	0.500000
5	0.600000
6	0.666667
7	0.571429

Run 2 data

1	0.000000
2	0.000000
3	0.333333
4	0.250000
5	0.200000
6	0.166667
7	0.285714



- ❖ Every time you run the code, the sequence of random numbers generated changes leading to a change in fraction of heads with number of trials
- Even though both times, the f_H is written with the same (default) accuracy, the plots (red and blue) generated are different because the sequence of heads and tails are different.

Class and Object in Python

❖ **Class:** A blueprint created for an **Object**. Defines a set of attributes that characterizes any **Object**

❖ **Object:** Specific instance of a **Class**

❖ Example of a Class

```
class myclass: # consists of 3 different quantities, a variable (x), a function(y) and a string(z)
```

```
    x=3
```

```
    y=x**2
```

```
    z="Welcome to my class"
```

```
ob=myclass() # creates an object (a specific instance of the class) which, in this case, is the same as the class
```

```
>>> myclass.x
```

```
3
```

```
>>> myclass.y
```

```
9
```

```
>>> myclass.z
```

```
'Welcome to my class'
```

```
>>> ob.x
```

```
3
```

```
>>> ob.y
```

```
9
```

```
>>> ob.z
```

```
'Welcome to my class'
```

Class and Object in Python

❖ The code segment can be saved as a python code called (for example) “myfirstclass.py” and imported as a module in other Python codes

```
import myfirstclass # imports the code myfirstclass.py as a module in a Python code
print (myfirstclass.ob.x) # accesses the specified attribute (x in this case) of the object ob defined as an instance of myfirstclass
print (myfirstclass.ob.y) # prints 9
```

❖ Built-in special functions inside a class: A function that begins with a *double underscore* __

```
class vector:
    def __init__(self, vx, vy, vz): # self-referencing function: the arguments vx, vy, vz are defined through self
        self.x=vx
        self.y=vy
        self.z=vz

V=vector(2,3,4) # V is an Object, i.e. an instance of the class vector
print (vector.x, vector.y, vector.z) # Not allowed! gives AttributeError: class vector has no attribute 'x'
print (V.x, V.y, V.z) # gives 2,3,4
```

❖ It is possible to delete attributes from an object

```
>>> del V.x # removes the x-component from the object V
>>> V.x # gives the following message: AttributeError: vector instance has no attribute 'x'
>>> del V # removes the object V
>>> V.y, V.z # gives the following message: NameError: name 'V' is not defined
```

Class and Object in Python

- ❖ Create a **class** which defines a vector with 3 components as a class and uses the components to define a function that returns the **norm** of the vector.
- ❖ Since the **norm** function is defined within a **class**, it is defined through the *built-in function* which has to be called whenever a function dependent on it is used.

```
import math
```

```
class vector: # a class that contains 2 functions, a vector with 3 components and its norm
```

```
    def __init__(self, vx, vy, vz): # specifies components of a vector which can be accessed through the 1'st argument self
        self.x = vx
        self.y = vy
        self.z = vz
```

Indentation is essential

```
    def norm(self): # specifies the norm of a vector, depends on vx, vy, vz defined in the built-in special function
        return math.sqrt(self.x**2+self.y**2+self.z**2)
```

```
>>> vector.norm() # norm cannot be accessed through a class but only through an object i.e. a specific instance of the class
TypeError: unbound method norm() must be called with vector instance as first argument (got nothing instead)
```

```
>>> vector(2,3,4).norm() # vector(2,3,4) is an object i.e. a specific instance of the class vector
5.385164807134504 # output correct to 15 decimal places by default
```

Creating and Importing modules in python codes

- ❖ Modules are like code libraries in Python. You can write your own python code (module) and put it in the library for everyone to use or you can use existing codes (modules) that are already present in the library

- ❖ *Creating your own module*

```
def greeting(name): # defines a module that can be used to say hello to a specified person by name
    print("Hello " + name)
```

- ❖ The above piece of code should be saved as a python code with .py extension. Lets give the filename “mymodule.py”

- ❖ The module can be imported to any python program by using the statement

```
import mymodule
mymodule.greeting(“Virat”) # generates the output Hello Virat
```

- ❖ The math module is a pre-existing module that can be imported by using the command

```
import math # imports all common mathematical functions to be used in your python program
math.sqrt(2) # generates the square-root of 2.
```

- ❖ NOTE: any math function like sqrt, sin, cos, log, factorial etc. needs to be called in the form math.sin(x) or math.log(x)

- ❖ <https://www.programiz.com/python-programming/modules/math> lists all available functions in the math module

- ❖ Here is a simpler way to import the math module that does not require calling the math functions with the prefix math.

```
from math import * # This imports all the functions in the math module and they can be used directly
sqrt(2) # gives the value of square root of 2
```

- ❖ A nice explanation for the difference between import math and from math import * is given at the following link

<https://discuss.codecademy.com/t/import-math-vs-from-math-import-is-there-a-difference/43314>