Introduction to Computation and Programming

Miscellaneous topics: files, exceptions, plotting, randomness, and Monte Carlo simulation

Reading: [Guttag, Sections 4.6, 7.1, 7.3, 11.1,16.4]

Slides prepared for EECE 230C, Fall 2018-19, MSFEA, AUB

Material on plotting added during the offering of EECE 230, Spring 2018-19, MSFEA, AUB

Material in these slides is based on [Guttag, Chapters 4, 7, 11, and 16], [MIT OpenCourseWare, 6.0001, Lecture 7, Fall 2016], and matplotlib tutorial

Outline

I. Files:
   - Handling files in Pyhon
   - Reading, writing, and appending

II. Exceptions and assertions:
   - try-except statement to handle exceptions
   - assert statement

III. Plotting in Python

IV. Generating random numbers in Python

V. Monte Carlo Simulation: approximating π

I. Files

I. Handling files in python

General structure:

```
nameHandle = open(fileName, mode)
#process the file and when done close it:
nameHandle.close()
```

- **fileName** : string containing the name of the file, e.g., “File3.txt” or “D:/HOME/…./File3.txt”
- **mode**: ‘r’ for reading, ‘w’ for writing , and ‘a’ for appending. There are also other modes we will not work with
- **nameHandle**: file handle returned by the open function

I. Reading a file in a one shot: read method

```
nameHandle = open(fileName, 'r')
# The file pointer is now in the beginning of the file
# Read the whole file into a single string s:
s = nameHandle.read()
# Now the pointer is at the end of the file
nameHandle.close()
# process the string s
```

- Example: write a function to display file, given fileName

I. Reading files in a one shot: read method (Continued)

```
 9 def  displayFile(fileName):
10     nameHandle = open(fileName, 'r')
11     s = nameHandle.read()
12     nameHandle.close()
13     print(s) # displays the whole file
14
15
16
17 fileName = input("Enter file name:")
18 displayFile(fileName)
```

I. Reading files line by line

```
nameHandle = open(fileName, 'r')
# Python can view the file as sequence of lines, each line is a string
 for line in nameHandle:
    # Process line, which is of type string
nameHandle.close()
```

Example: write a function to display file **with line numbers**, given fileName

I. Reading a file line by line (Continued)

```
20 def  displayFileWithLineNumbers(fileName):
21     nameHandle = open(fileName, 'r')
22     i = 1
23     for line in nameHandle:
24         print("Line",i,":",line, end ='')
25         i+=1
26     nameHandle.close()
27
28 fileName = input("Enter file name:")
29 displayFileWithLineNumbers(fileName)
```

To avoid double new lines: as a string, line ends with '\n' (except possibly for the last line)

I. Writing on a file: write method

```
nameHandle = open(fileName, 'w')
# A new file is created and file pointer is at  the beginning of the file
nameHandle.write( input argument s of type string)
# Now the file consists of s and pointer moved
# Use write method again to write another string and so on
# When done close (otherwise, some writes may not be saved)
nameHandle.close()
```

I. Writing on a file: write method (Continued)

```
nameHandle = open('testingWrite.txt', 'w')
nameHandle.write("abc")
nameHandle.write("de\n")
nameHandle.write("f g")
nameHandle.close()
```

Content of testingWrite.txt:

I. Writing on a file: write method (Continued)

```
nameHandle = open('testingWrite.txt', 'w')
nameHandle.write("abc")
nameHandle.write("de\n")
nameHandle.write("f g")
nameHandle.close()
```

Content of testingWrite.txt:

abcde

f g

I. Append mode

- What if the file you want to write on already exists and instead of overwriting you want to append new strings?

- Instead of

  nameHandle = open(**fileName**, ‘w’)

  use

  nameHandle = open(**fileName**, ‘a’)

- It will create a new file only if the file doesn’t exist

II. Exceptions

II. Back to read mode

- What if the file we are trying to read doesn't exist?
- Python script will crash/terminate with a:

  FileNotFoundError: [Errno 2] No such file or directory
- This is a good thing as we don't program to continue in abnormal situations
- This is an unhandled exception raised by Python.
- Exception: "something that does not conform to the norm"
- We can handle exceptions

II. Other common exceptions

- Include:
  - TypeError: e.g., 1/”abc”
  - IndexError: e.g., **L**=[“a”,”b”] and then try to access **L**[2]
  - ValueError: e.g., int("abc") (int(“12") works fine)
  - ZeroDivisionError: e.g., 1/0

II. Handling exceptions: try-except statement: basic structure

```
try:
    Code Block A
except:
    Code Block B
    will be executed if an
    exception was raised in
    Code Block A, instead of
    program crashing
```

```
try:
    Code Block A
except Error_1:
    Code Block B_1
….
except Error_k:
    Code Block B_k: will execute only if
    Error_k was raised   in Block A
…
except:
    Code Block B: will execute if an exception
    other than all the above was raised in
    Block A
```

II. Handling Exceptions: Example 1: FileNotFoundError

Let's say that instead of program crashing if file not found, you want to handle the situation as follows:

Write function readFile(**fileName**), which given the name of file, tries to read it single shot using the read method and returns the tuple (**s**,**fileFound**), where:

- If the file is found, **fileFound** should be True, and **s** a string consisting of the file's content
- If the file is not found, **fileFound** should be False, and **s** is the empty string "". The function should also display message "Cannot open file! "

II. Handling Exceptions: Example 1: IOError: file name not found (Continued)

```
11 def readFile(fileName):
12     try:
13         nameHandle = open(fileName, 'r')
14         s = nameHandle.read()
15         nameHandle.close()
16         fileFound= True
17     except FileNotFoundError:
18         print("Cannot open file!")
19         s = ""
20         fileFound = False
21     return (s,fileFound)
22 
23 fileName = input("Enter file name:")
24 (s,found)  = readFile(fileName)
25 print(s)
```

II. Handling Exceptions: Example 2: ValueError and ZeroDivisionError

- Consider:

```
a = int(input("Enter integer a:"))
b = int(input("Enter a nonzero integer b:"))
x = a/b
print(x)
```

- Two possible exceptions:
  - ValueError: if user enters non-integer values
  - ZeroDivisionError: if **b** is zero
- Le't say that instead of program crashing, you want to handle those exceptions by insisting that the user enters good values

II. Handling Exceptions: Example 2: ValueError and ZeroDivisionError: Solution 1

```
35 while True:
36     try:
37         a = int(input("Enter integer a:"))
38         b = int(input("Enter a nonzero integer b:"))
39         x = a/b
40         break
41     except:
42         print("Bad input!")
43 print(x)
```

II. Handling Exceptions: Example 2: ValueError and ZeroDivisionError: Solution 2

```
46 while True:
47     try:
48         a = int(input("Enter integer a:"))
49         b = int(input("Enter a nonzero integer b:"))
50         x = a/b
51         break
52     except ValueError:
53         print("Please enter integers!")
54     except ZeroDivisionError:
55         print("b should be nonzero!")
56 print(x)
```

II. Assertions

- Raising exceptions: *raise* statement
- Will focus on assert statement, which specifically raises an **AssertionError**
- Syntax:

  assert Boolean expression, "error message"

- If Boolean expression evaluates to False, program terminates with :

  AssertionError: error message

- Useful to confirm that the arguments to a function are of appropriate types and/or satisfy certain conditions

II. Assertions: example

Consider:

```
def f(L):
    """ Function assumes that L is a nonempty list """

    L[0] =1
    ## ....

f(1) # or f([])
```

II. Assertions: example (Continued)

Add assertion to terminate program if conditions not met:

```
def f(L):
    """ Function assumes list L is nonempty """
    assert type(L)==list and len(L)!=0, "L is not a nonempty list"
    L[0] =1
    ## ....

f(1) # or f([])
```

Assert won't cause an error if L is not a list due to **short circuit evaluation of and operator**: If first operand is False, the second won't be evaluated

III. Plotting

III. Plotting graphs in Python

- First you need to import the plotting module:

```
import matplotlib.pyplot as plt
```

- To make sue that the figures do not appear inside the Python console:
  - Go to *Tools> Preferences>IPython console > Graphics > Graphics backend*, and set *Backend* to *Automatic*.
  - You need to restart the kernel for this change to take effect.

III. Plotting graphs in Python (Continued)

- Let **X** and **Y** be lists of numbers of the same length.
- To plot **Y** as a function of **X**, use

  plt.plot(**X**,**Y**,**color**)

  where **color** is a string taking values such as "k" (for blac**k**), "r" (for **r**ed), “b” for **b**lue, etc.. See the documentation of plot for other colors.
- The plot function plots the points (X[i], Y [i]), for i = 0,…,len(X), connected by lines of colors **color**.
- To include labels on the x-axis and the y-axis, use

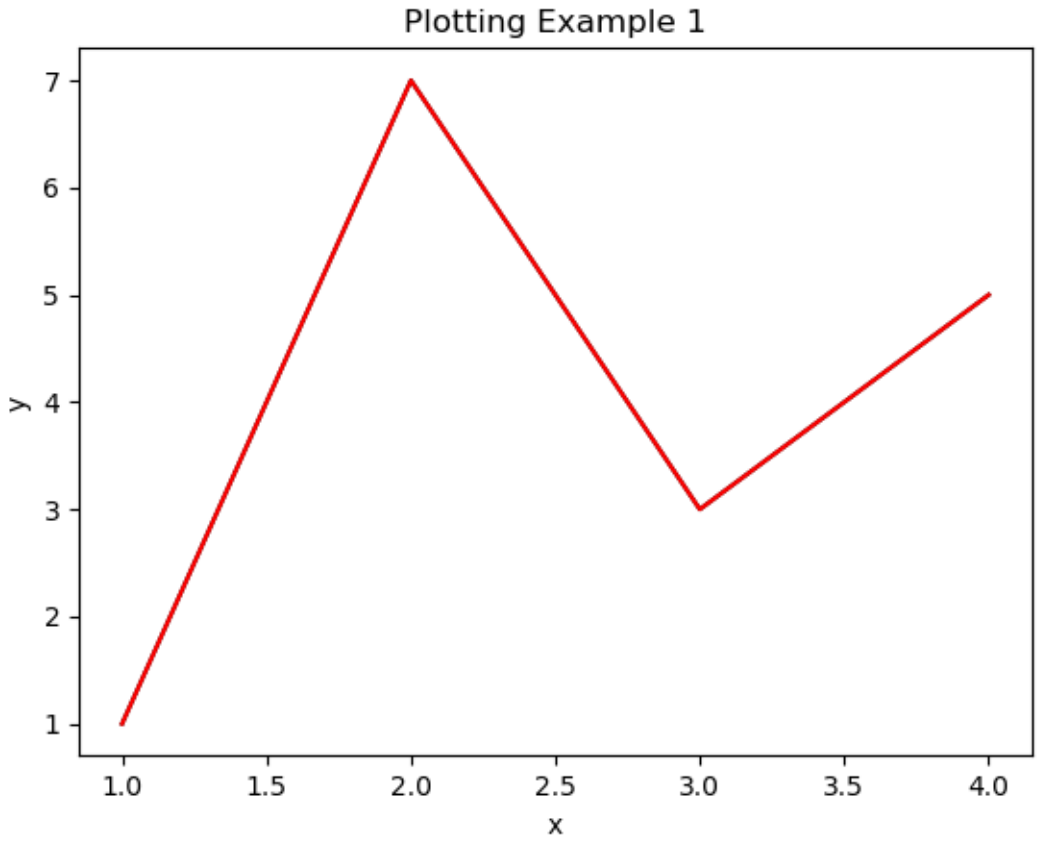
  plt.xlabel(“x label text")

  plt.ylabel(“y label text")
- To include a title, use

  plt.title(“title text")

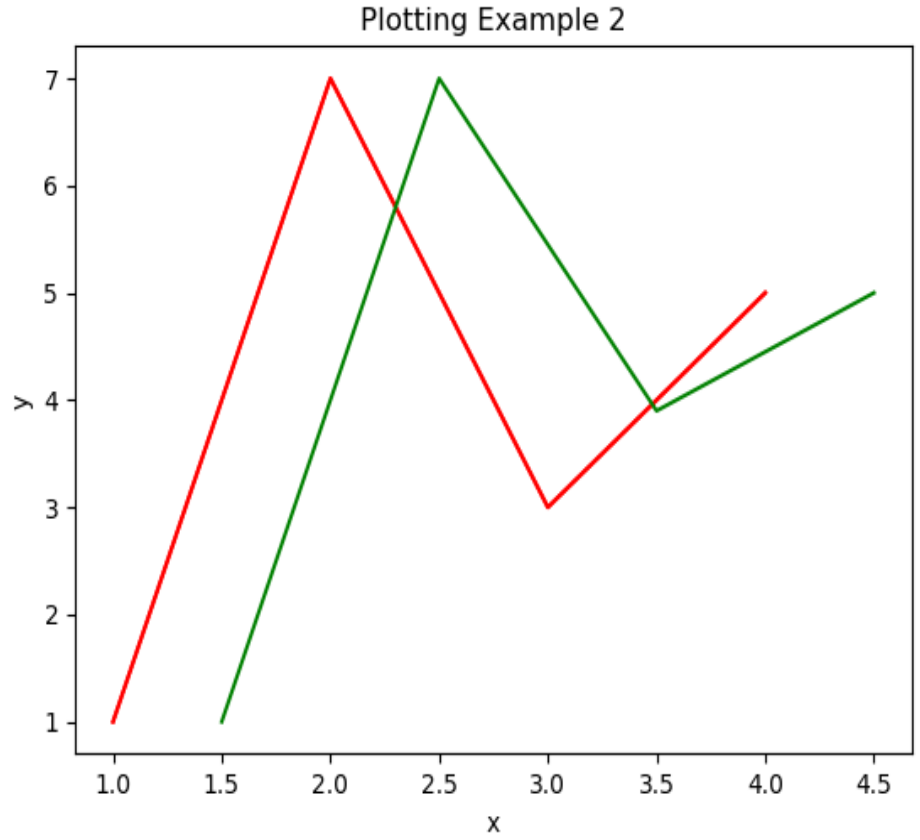
III. Example 1

```
 8 import matplotlib.pyplot as plt
 9 plt.plot([1,2,3,4], [1,7,3,5],"r")
10 plt.xlabel('x')
11 plt.ylabel('y')
12 plt.title('Plotting Example 1')
```



III. Example 2

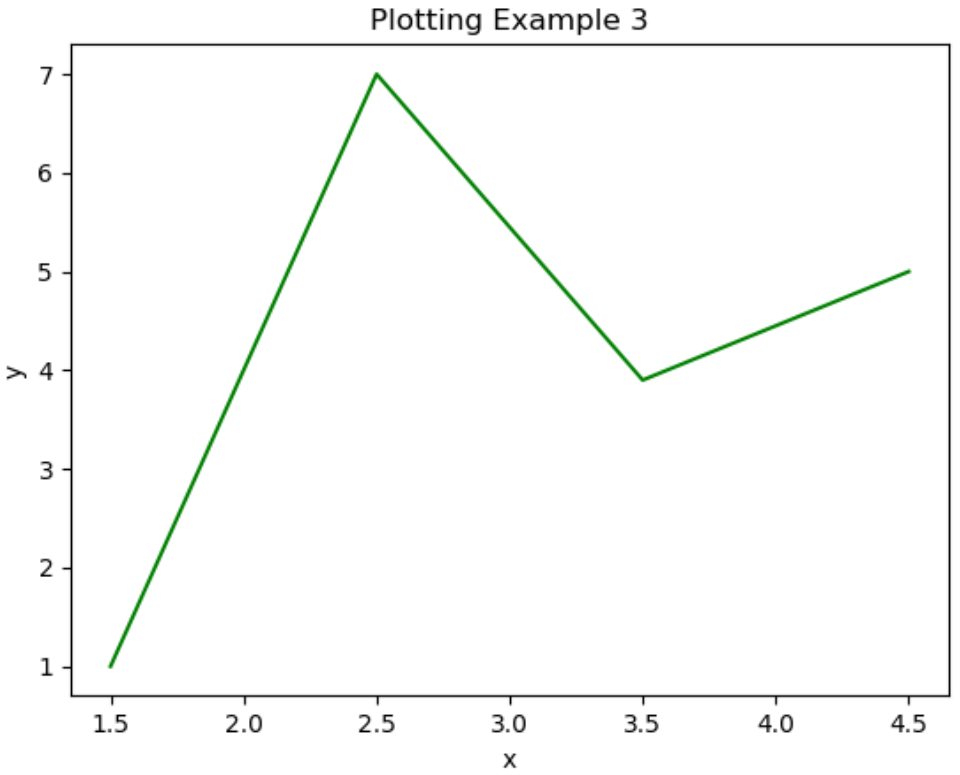
```
15 plt.plot([1,2,3,4], [1,7,3,5],"r")
16 plt.plot([1.5,2.5,3.5,4.5], [1,7,3.9,5],"g")
17 plt.xlabel('x')
18 plt.ylabel('y')
19 plt.title('Plotting Example 2')
```



III. To clear a figure

To clear the figure, use **plt.clf()**, e.g.,

```
23 plt.plot([1,2,3,4], [1,7,3,5],"r")
24 plt.clf()
25 plt.plot([1.5,2.5,3.5,4.5], [1,7,3.9,5],"g")
26 plt.xlabel('x')
27 plt.ylabel('y')
28 plt.title('Plotting Example 3')
```



III. Plotting on multiple figures

- To plot on a new or existing figure whose index is **i**, use

  plt.figure(i)

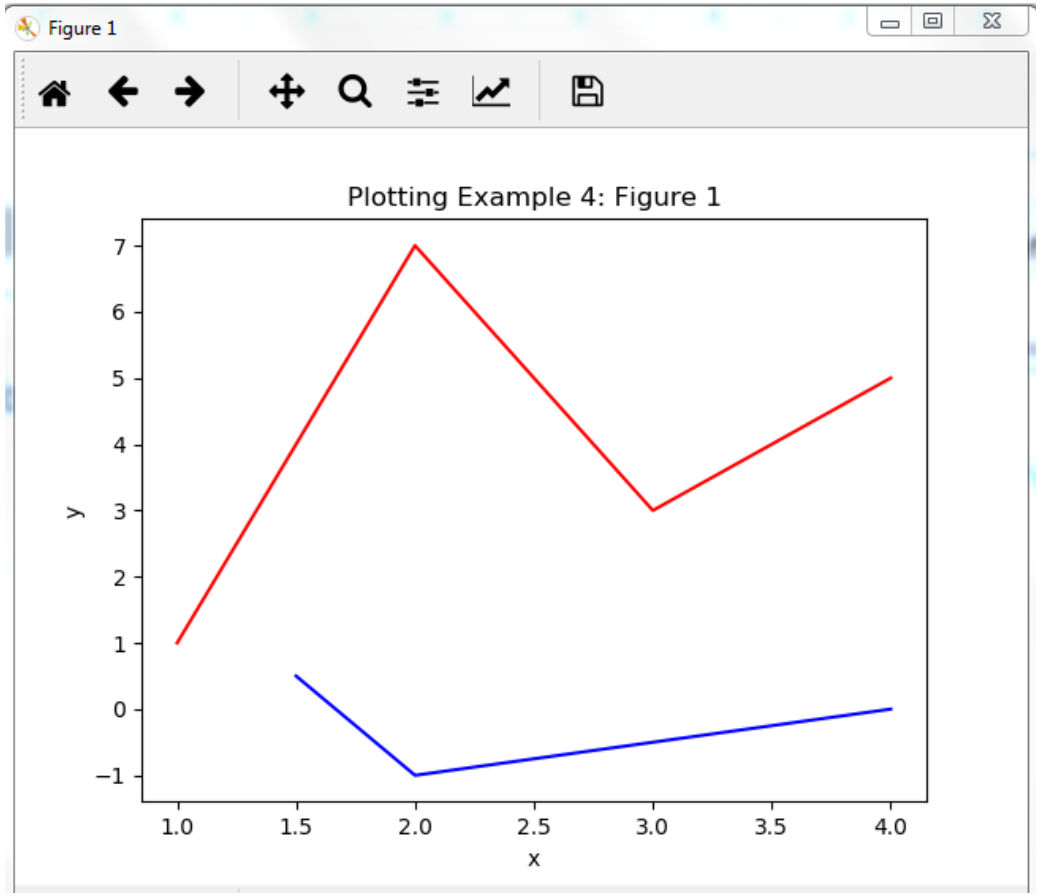
  before invoking plt.plot

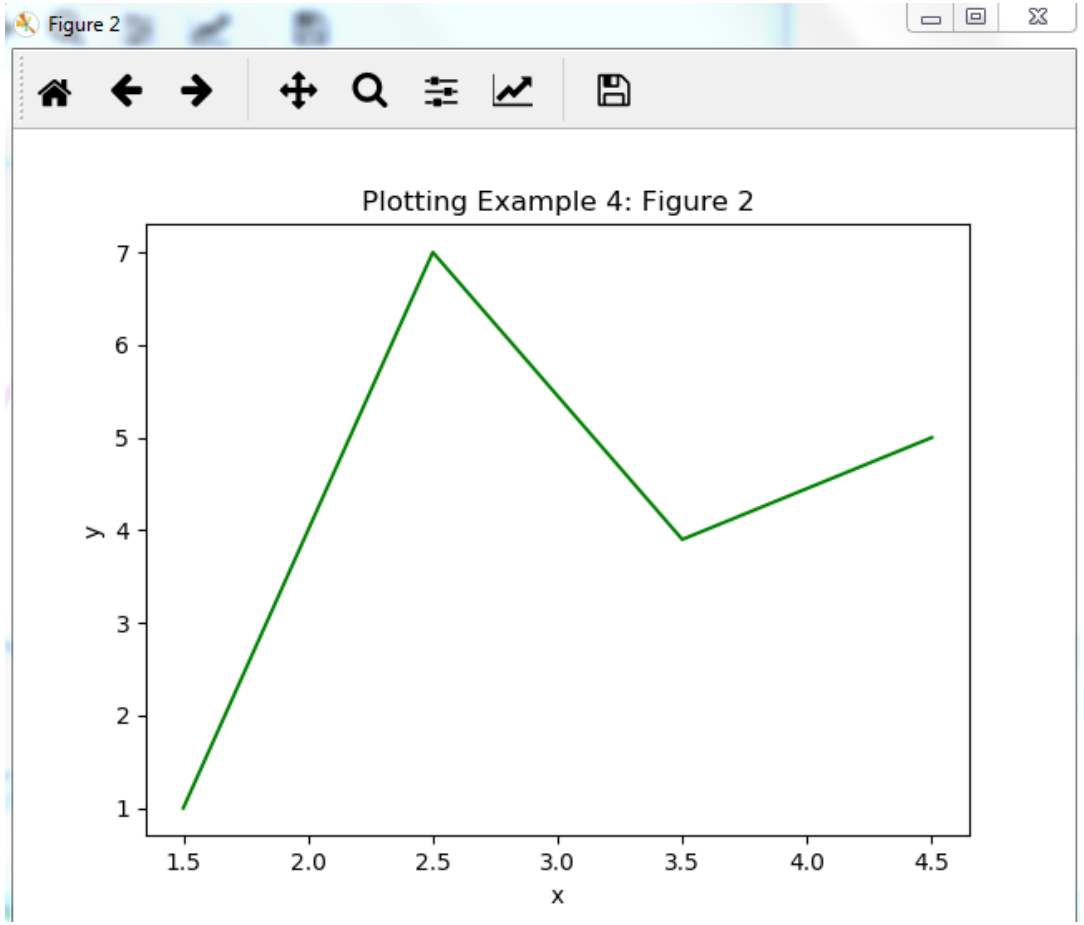
- The default value of **i = 1**, i.e., can skip Line 34
- To close Figure **i**, use

  plt.close(i)

```
34 plt.figure(1)
35 plt.plot([1,2,3,4], [1,7,3,5],"r")
36 plt.xlabel('x')
37 plt.ylabel('y')
38 plt.title('Plotting Example 4: Figure 1')
39 plt.figure(2)
40 plt.plot([1.5,2.5,3.5,4.5], [1,7,3.9,5],"g")
41 plt.xlabel('x')
42 plt.ylabel('y')
43 plt.title('Plotting Example 4: Figure 2')
44 plt.figure(1)
45 plt.plot([1.5,2,4], [0.5,-1,0],"b")
```

III. Plotting on multiple figures (Continued)





III. Subplots

To plot on the same figure multiple graphs with tiled axes, use

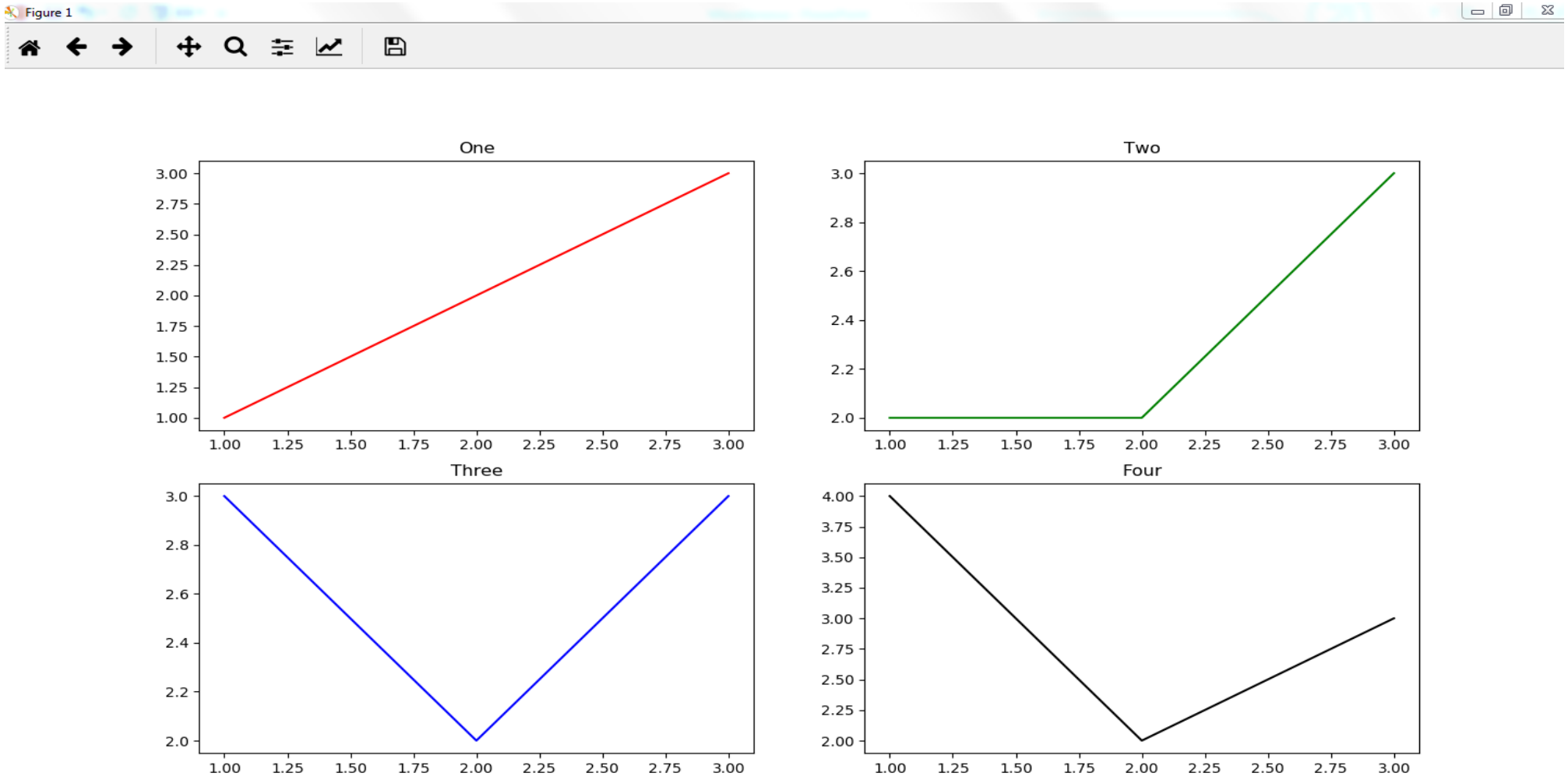
plt.subplot(m,n,i)

before invoking plt.plot, where:

- **m** is the desired number of rows
- **n** number of columns
- **i** the graph index: **i=1** (upper left), …, **m** ×**n** (lower right)

```
49 plt.figure(1)
50 plt.subplot(2,2,1)
51 plt.plot([1,2,3], [1,2,3],"r")
52 plt.title('One')
53 plt.subplot(2,2,2)
54 plt.plot([1,2,3], [2,2,3],"g")
55 plt.title('Two')
56 plt.subplot(2,2,3)
57 plt.plot([1,2,3], [3,2,3],"b")
58 plt.title('Three')
59 plt.subplot(2,2,4)
60 plt.plot([1,2,3], [4,2,3],"k")
61 plt.title('Four')
```

III. Subplots (Continued)

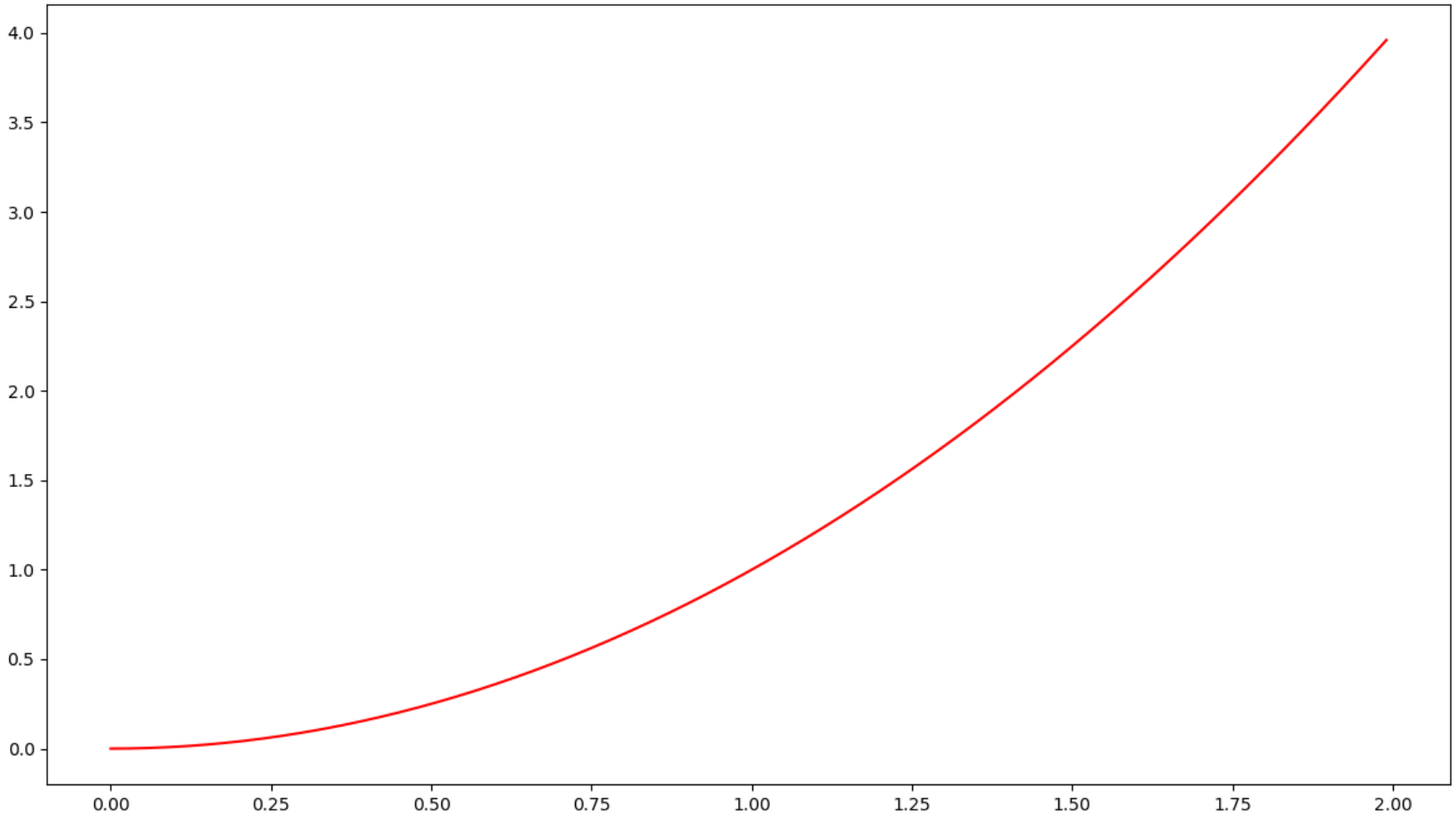


III. Plot the graph $y = x^2$ for $0 \leq x < 2$

III. Plot the graph $y = x^2$ for $0 \leq x < 2$

```
65 X = [0.01*i for i in range(0,200)] # [0.00, 0.01,...,1.99]
66 Y = [x*x for x in X]
67 plt.plot(X,Y,'r')
```

III. Plot the graph $y = x^2$ for $0 \leq x < 2$



III. Plotting data from txt file

- Say that we are given text file data.txt with x and y measurements each on a line , e.g.,

```
 1.3 2.13
2 3.16
5  7.20
8.3 -6.0
11.1  10.4
```

- Plot y versus x

III. Plotting data from txt file

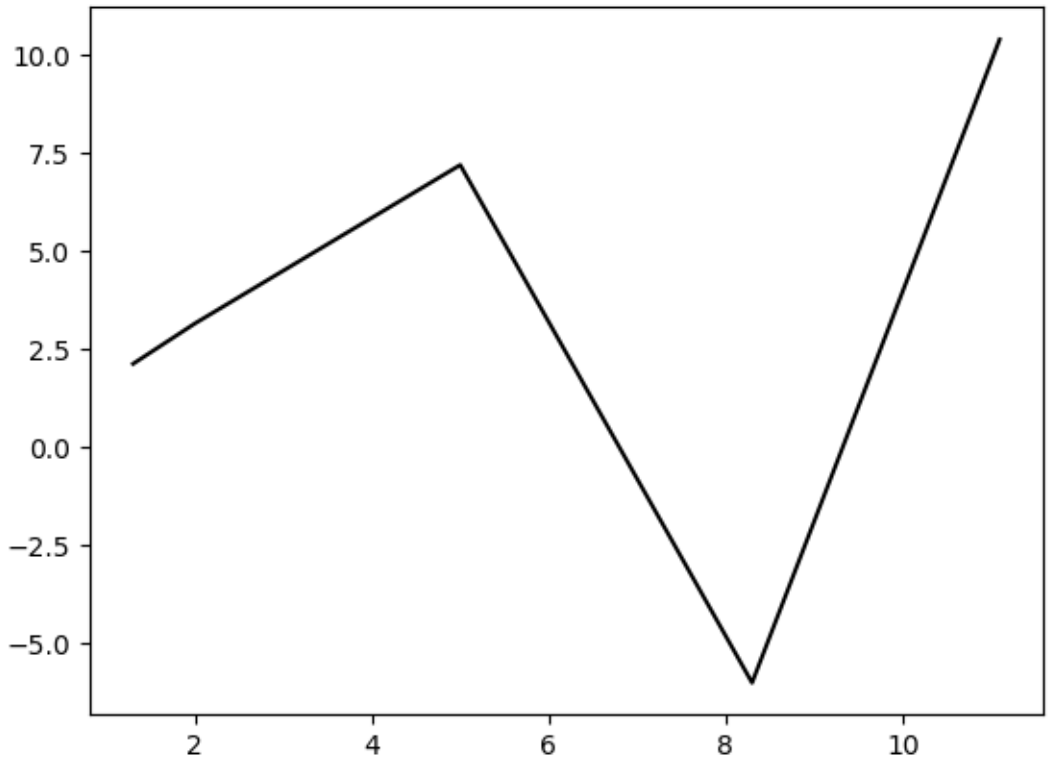
- Say that we are given text file data.txt with x and y measurements each on a line , e.g.,

```
 1.3 2.13
2 3.16
5  7.20
8.3 -6.0
11.1  10.4
```

- Plot y versus x

```
75 ##### Ploting data from file ##
76 nameHandle = open("data.txt","r")
77 X = []
78 Y = []
79 for line in nameHandle:
80     L = line.split()
81     if len(L)==2:
82         # in case  there are  empty lines
83         X.append(float(L[0]))
84         Y.append(float(L[1]))
85 nameHandle.close()
86 plt.plot(X,Y,'k')
```

III. Plotting data from txt file



III. Useful tools for plotting functions: **legend**

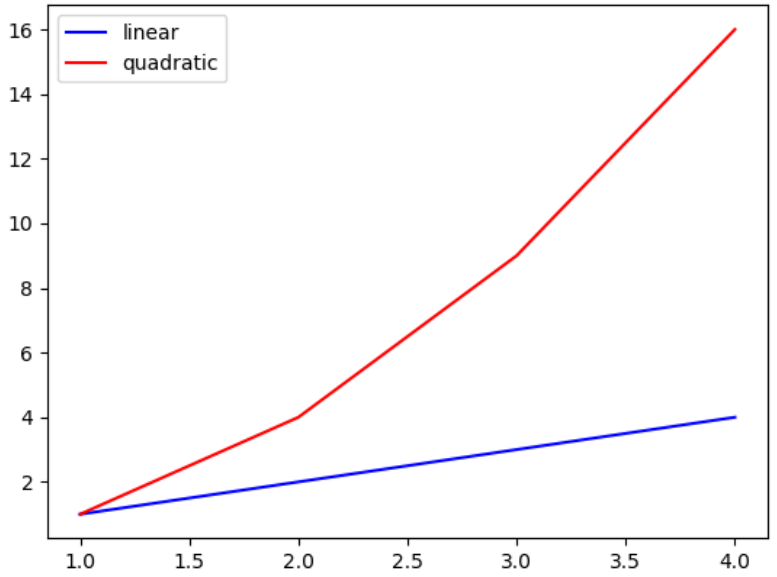
- When plotting multiple functions on the same figure, it helps to include a **legend** to label the functions: when creating a plot, you can add a label

  plt.plot(X,Y,**label="myLabel"),**

  which will appear when your code invokes

  plt.legend().

- Example:

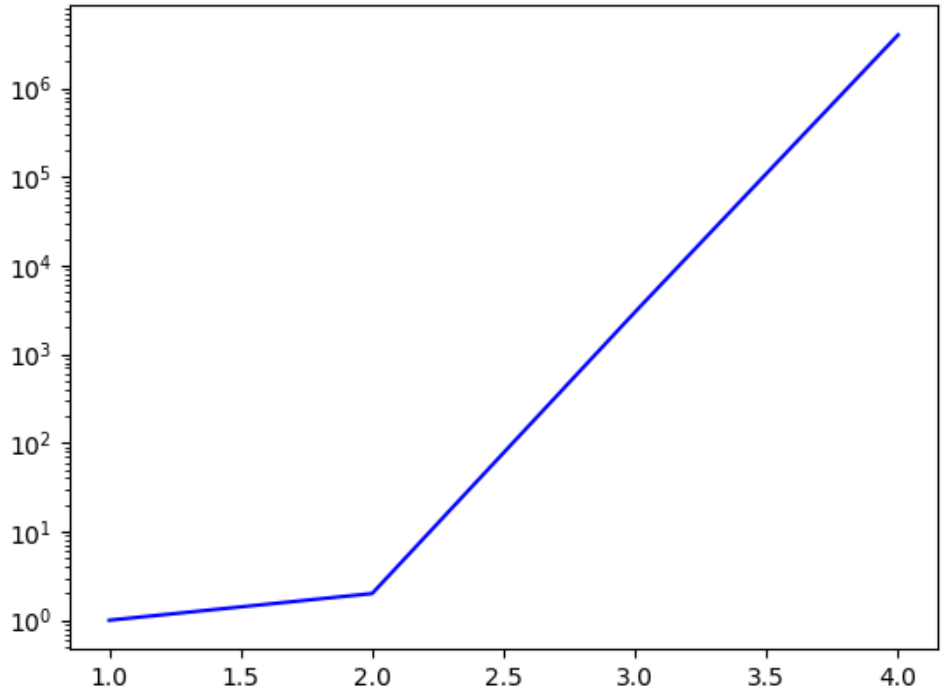



```
92 plt.plot([1,2,3,4], [1,2,3,4],"b",label = 'linear')
93 plt.plot([1,2,3,4], [1,2**2,3**2,4**2],"r", label = "quadratic")
94 plt.legend()
```

III. Useful tools for plotting functions: logscale

- If some y-values are relatively very large, use **log scale on the y-axis: plt.yscale('log')**
- Example:

```
 99 plt.plot([1,2,3,4], [1,2,3000,4000000],"b")
100 plt.yscale('log')
```



III. Matplotlib module

- For more on the matplotlib.pyplot library, check the tutorial https://matplotlib.org/users/pyplot_tutorial.html

- It has many plotting tools. For instance, below are some useful plotting tools which you may want to check (not included in assignments or exams):
  - In plt.plot, you can specify line style and markers in addition to color. You can also skip the color string and rely on python to appropriately choose colors.
  - numpy.linspace function (read about numpy ndarrays, which are like python lists, but with all elements of the same type)

IV. Generating random numbers

IV. Generating random numbers in Python

- Import the numerical python module: **numpy**

  import numpy.random as rand

- Basic random functions:
  - rand.uniform(x,y): generates a uniformly random float in the real interval [x,y] (default x=0 and y=1)
  - rand.randint(a,b+1): generates a uniformly random integer between a and b inclusive

IV. Generating random numbers in Python (Continued)

- Multiple runs give different random numbers:

- Where does the randomness come from?

```
In [10]: rand.uniform(-1,1)
Out[10]: -0.8839333339290703

In [11]: rand.uniform(-1,1)
Out[11]: 0.3341273572370478

In [12]: rand.randint(0,10)
Out[12]: 6

In [13]: rand.randint(0,10)
Out[13]: 1
```

IV. Randomness in computation

- In this course, we will see multiple applications of randomness in computation
- First application: Monte Carlo simulation

V. Monte Carlo simulation

V. Monte Carlo simulation

- Monte Carlo simulation is a technique used to approximate the probability of an event by random sampling multiple times and averaging the results.
- We will see how Monte Carlo simulation can be used to solve a problems that are not inherently stochastic:
  - Approximate π

V. Approximating π

- Consider the unit circle inscribed in the unit square:

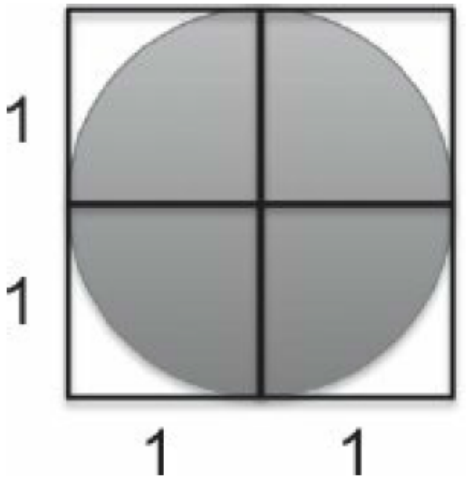


Figure 16.5 Unit circle inscribed in a square

Figure 16.5 in [Guttag, page 356]

V. Approximating π (Continued)

- Key observation:

$$\frac{\pi}{4} = \frac{\text{area of unit circle}}{\text{area of unit square}}$$

= probability $\boldsymbol{p}$ that a random point of the unit square belongs to the unit circle

- Thus $\pi = 4 \times \boldsymbol{p}$
- Hence, to approximate π:

V. Approximating π (Continued)

- Key observation:

$$\frac{\pi}{4} = \frac{\text{area of unit circle}}{\text{area of unit square}}$$

= probability $\boldsymbol{p}$ that a random point of the unit square belongs to the unit circle

- Thus $\pi = 4 \times \boldsymbol{p}$
- Hence, to approximate π:
  - Choose n points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ in the unit square: choose $-1 \leq x_i \leq 1$ and $-1 \leq y_i \leq 1$ uniformly at random for $i = 1, \ldots, n$
  - Find number m of points in the unit circle
  - Return $4m/n$ (as m/n is an approximation of $\boldsymbol{p}$)
- For large n, get an approximation of π

V. Approximating π (Continued)

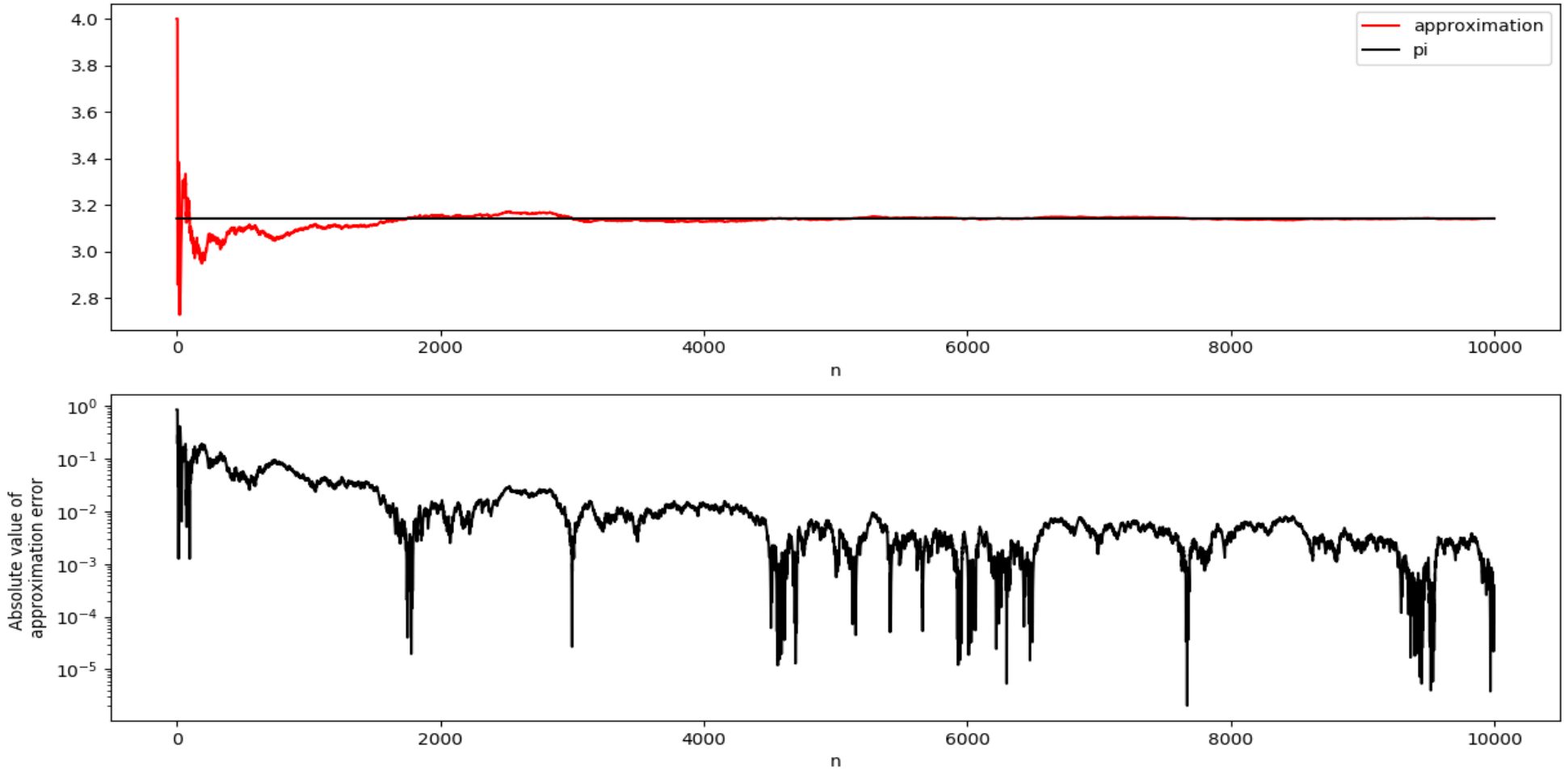
```
 8 import numpy.random as rand
 9 from math import pi
10 def approximatePi(n):
11     m = 0
12     for i in range(n):
13         x = rand.uniform(-1,1)
14         y = rand.uniform(-1,1)
15         if x**2+y**2<=1:
16             m+=1
17     return 4*m/n
18 piHat=approximatePi(10000)
19 print("Approximate pi:", piHat)
20 print("Absolute value of error:", abs(pi-piHat))
```

Output:

Approximate pi: 3.1452

Absolute value of error:
0.003607346410206879 7

V. Approximate π as a function of n



- You will generate plots like the above (not the same!) in the next solving session