Poor man’s introduction to Python

Raymond Atta-Fynn (University of Texas, Arlington)

NSF Summer School on Disordered Materials Modeling

Summer 2019

attafynn@uta.edu

Outline

- Python Overview
- Python Language Basics
- Application to simple problems

Python

- Easy to learn & use
- Ubiquitous, Great for quick prototyping
- Supports various programming styles
  - Procedural, Functional etc
- Dynamically typed; interpretive language
  - A variable can be assigned to any data type
- Strongly typed
  - Once assigned, it remains as that type
- Documentation: https://docs.python.org

Python Basics – Program Organization

- Indentations matter
  - Code block boundaries (e.g. for loop, while loop, if-else block, function block, etc.)
- Modularization
  - A module in python: file containing a set of functions you want to include in your application.
  - Enables code reuse, importing functionality from other code
- Packages
  - Modules can be combined to make packages
  - Packages => Module A => Module B => Module C
  - Can import entire package or specific module from a package

Python Basics – Built-in Datatypes

- Basic Types
  - int, float, long, complex, boolean
- Strings
- Collections
  - List
    - Collection of objects (int, floats, any python object)
  - Tuple
    - Like lists, but unmutable
  - Dictionary
    - *Key:Value* pairs
- Everything is an object
  - Will have its own methods to manipulate it

Outline

It’s now time to fire up Jupyter

[Expert Python coders can operate from the terminal]

Python Basics – Example (1)

```
In [21]: # Importing a module
         import sys

         # accessing a modules parameters
         sys.path

         # This is a single line comment
         """
         This is a
         multiline comment
         """

Out[21]: '\nThis is a\nmultiline comment\n'

In [22]: # Initialising a Variable
         # Integer
         x = 10
         # String
         y = " hello "

In [ ]:
```

Python Basics – Example (2)

```
In [25]: # Collections: List, Tuple, Dictionary
         # List
         example_list = [ "a", "b", "c" ]
         # List : access element
         example_list[2]

Out[25]: 'c'

In [26]: # List : append element
         example_list.append("d")

In [29]: print(example_list)

         ['a', 'b', 'c', 'd']

In [30]: print (len(example_list))

         4

In [31]: # Initialize empty list
         empty_list = list()

In [32]: print(empty_list)

         []
```

Python Basics – Example (3)

```
In [33]: # Tuple
         example_tuple = (1, 2, 3)
         # Tuple : access element
         example_tuple[1]

Out[33]: 2
```

```
In [34]: # Tuple : no append method i.e immutable
         # Get length of tuple
         print(len(example_tuple))

         3
```

Python Basics – Example (4)

In [35]:
```
# Dictionary {Key:Value, Key:Value....}
example_dictionary = {"a":1, "b":3, "c":3}
# Dictionary : access element by key
example_dictionary["c"]
```

Out[35]: 3

In [36]:
```
# Add new element
example_dictionary["d"]=4
print(example_dictionary)
```

```
{'a': 1, 'c': 3, 'b': 3, 'd': 4}
```

In [37]:
```
# Get length of dictionary
print(len(example_dictionary))
```

```
4
```

Python Basics – Control Flow (1)

- **while loop:**
  - while expression/boolean:
  - statements ….
  - ….
- **for loop:**
  - for x in ‘some iterable collection’:
  - statements …
- **break**
  - Terminates loop
- **continue**
  - Next iteration of loop
- **pass**

Python Basics – Control Flow (2)

- if elif else
  - If condition:
  - statements….
  - elif:
  - statements
  - else:
  - statements….
- functions
  - def function_name(arguments):
  - statements …

Python Basics – Control Flow Example (1)

recreate the list example_list

In [38]:

```
# Example control flow
# if-elif-else : Number of elements even or odd or empty list
if len(example_list)==0:
    print("empty list")
elif len(example_list)%2==0:
    print ("Even number of elements")
else:
    print ("Odd number of elements")
```

```
Even number of elements
```

Python Basics – Control Flow Example (2)

In [39]:
```
# For loop : Print even index elements of the list
for element in example_list:
    if example_list.index(element)%2==0:
        print(element)
    else:
        # pass does nothing
        pass
```

```
a
c
```

In [40]:
```
print(example_list)
```

```
['a', 'b', 'c', 'd']
```

Python Basics – Control Flow Example (3)

```
In [44]: example_list.append("x")

In [45]: example_list

Out[45]: ['a', 'b', 'c', 'd', 'x']

In [46]: # While loop : Search for "x" in list
         index=0
         while index<len(example_list):
             if example_list[index] == "x":
                 print ("Found element x")
                 break
             else:
                 index+=1
                 continue

         Found element x
```

Python Basics – Control Flow Example (4)

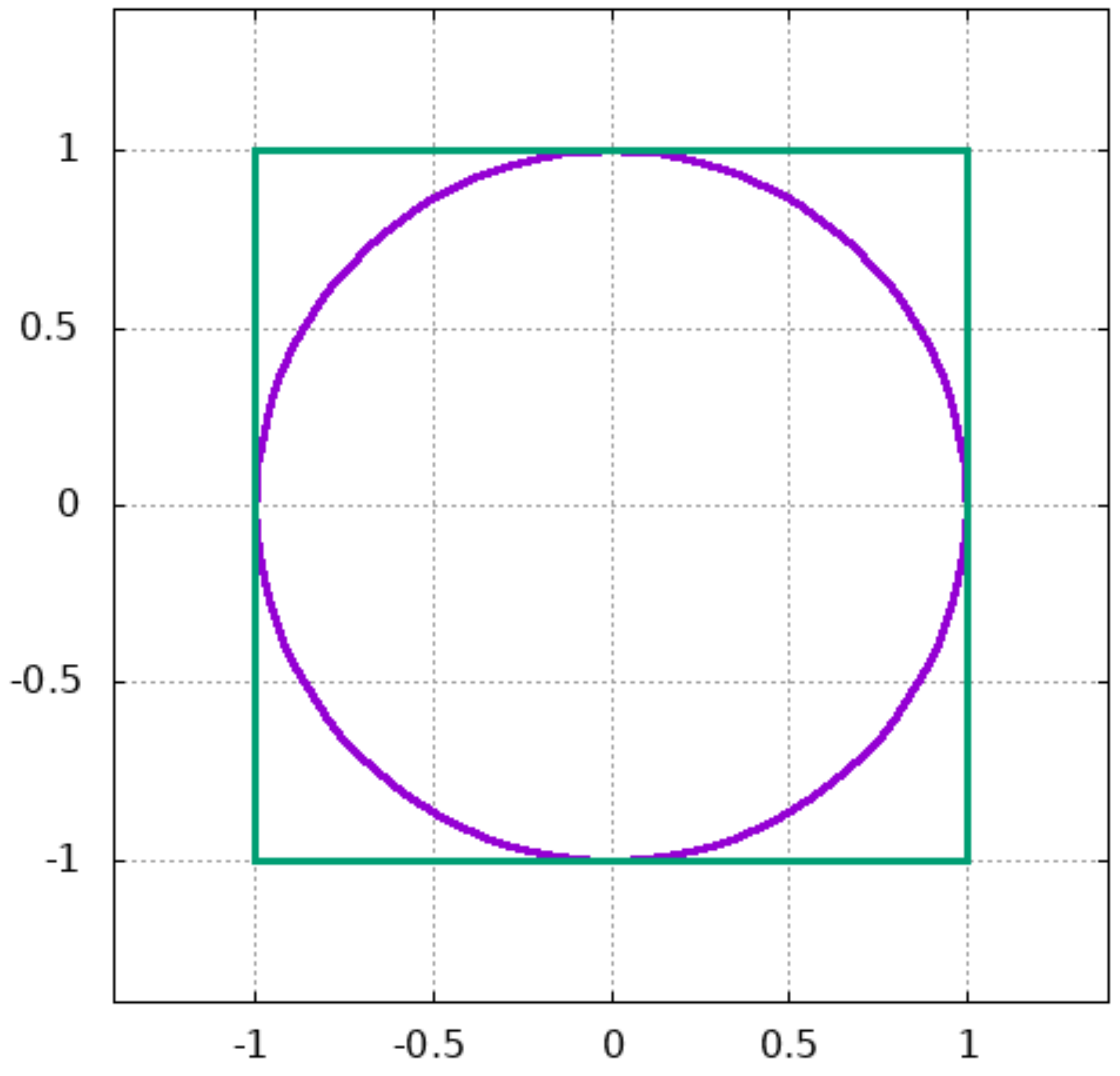
In [47]:

```python
# define function to search a list for an element
# and return its index if found and -1 if not found
def search_list_for_element(element, search_list):
    index=0
    while index < len(search_list):
        if search_list[index] == element:
            break
        else:
            index+=1
            continue
    #If index is less than length, element is found
    if index < len(search_list):
        return index
    else:
        return -1
```

Python Basics – Control Flow Example (5)

```
In [48]: search_list_for_element("x", example_list)

Out[48]: 4

In [49]: example_list

Out[49]: ['a', 'b', 'c', 'd', 'x']
```

Determination of pi using Monte Carlo



If a dart thrown has an equal probability of landing anywhere inside the green square box:

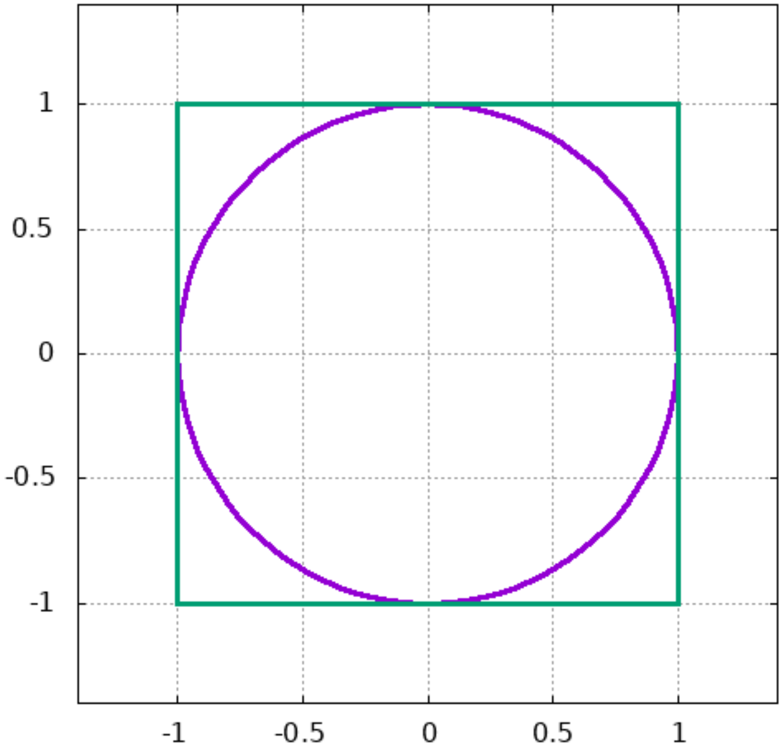
$$\frac{Area\ of\ cirle\ of\ radius\ 1}{Area\ of\ square\ of\ side\ 2} = \frac{N_C}{N}$$

where N_C is the number darts which land inside the circle and N is the total number which land in the square.

This implies that

$$\frac{\pi(1)^2}{4} = \frac{N_C}{N} \Rightarrow \pi = \frac{4N_C}{N}$$

Determ

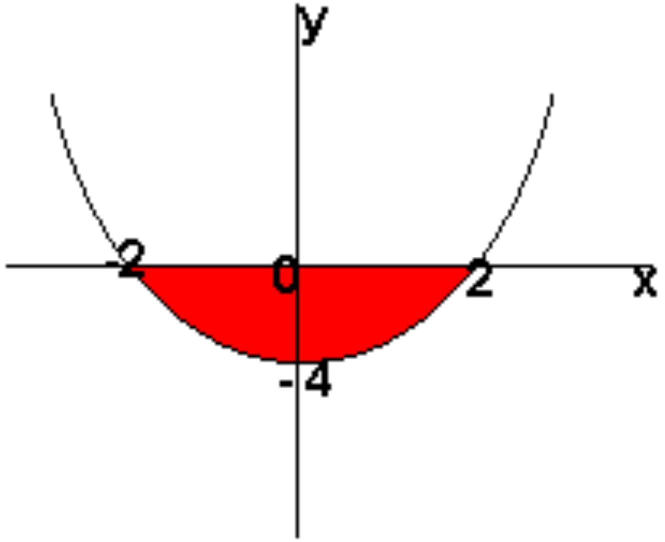


In [7]:

```
import math
import random


n_attempts = 1000000
n_accept   = 0


for i in range(n_attempts):
    x = random.uniform(-1,1)
    y = random.uniform(-1,1)

    r=math.sqrt(x**2 + y**2)
    if r <= 1:
        n_accept += 1


monte_carlo_pi = 4*(n_accept/n_attempts)
exact_pi = math.pi

print("The simulated value of pi is %.6f" %monte_carlo_pi)
print("The exact value of pi is %.6f" %exact_pi)
```

Determination of area under a curve Monte Carlo

Example: What is the area between the curve $y = x^2 - 4$ and the x axis?



The shaded area is the area that we want.

We can easily work out that the curve crosses the x axis when $x = -2$ and $x = 2$. To find the area, therefore, we integrate the function between -2 and 2.

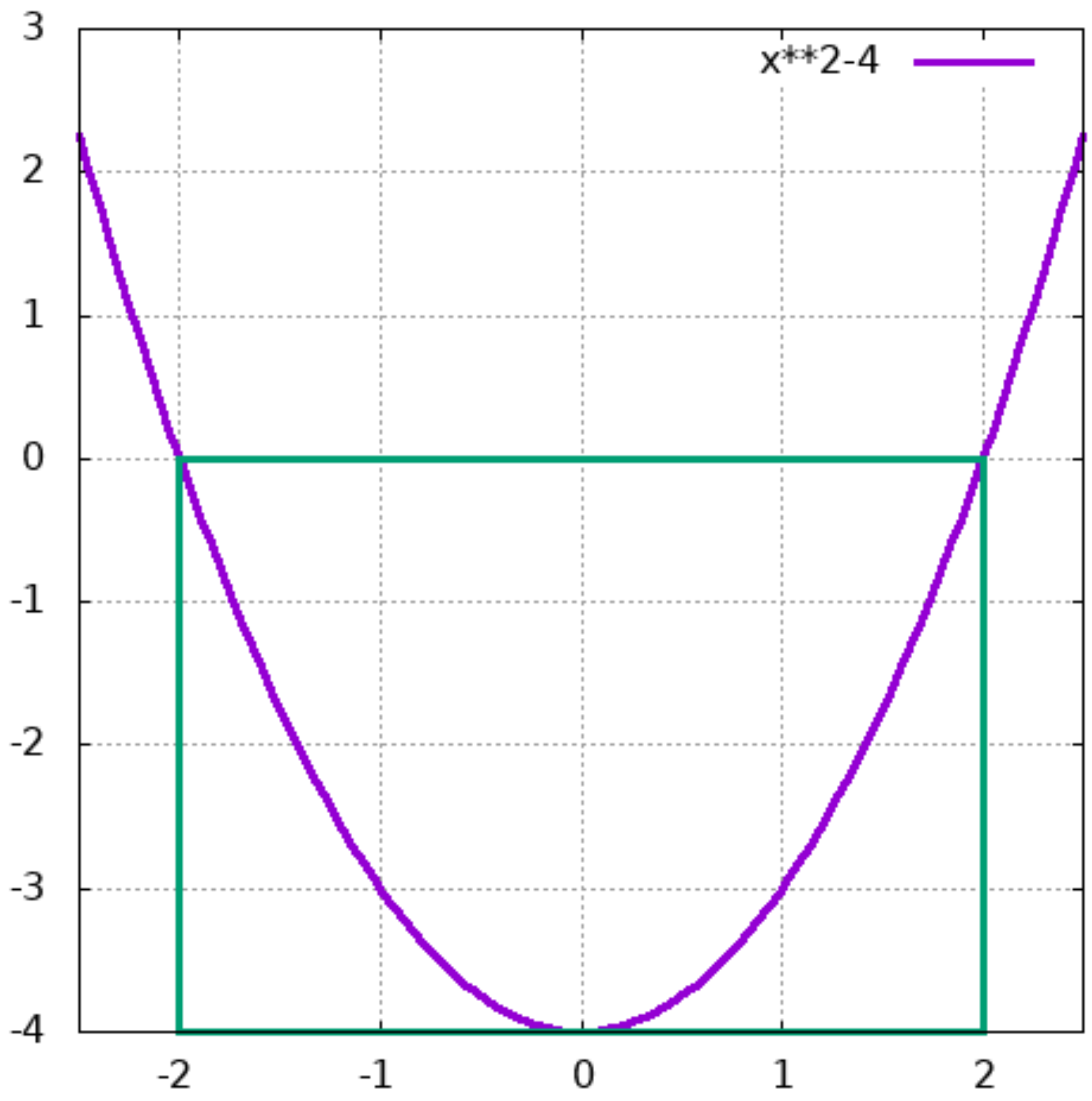
$$\int_{-2}^{2} (x^2 - 4)\, dx = \left[\frac{x^3}{3} - 4x \right]_{-2}^{2} = \left(\frac{8}{3} - 8 \right) - \left(\frac{-8}{3} + 8 \right)$$

$= 16/3 - 16$

$= \underline{-10.67}$ (2d.p.)

Note: the area is negative because it is below the x-axis. Areas above the x-axis, on the other hand, give positive results.

Determination of area under a curve Monte Carlo



If a dart thrown has an equal probability of landing anywhere inside the green square box:

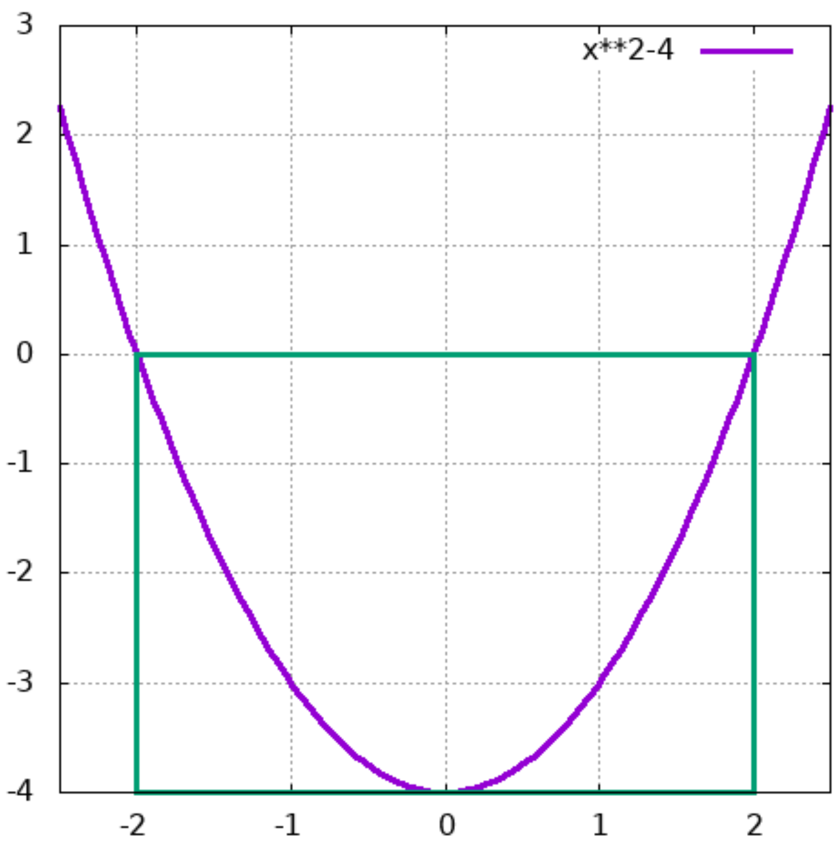
$$\frac{A}{Area\ of\ square\ of\ side\ 4} = \frac{N_C}{N}$$

where *A* is the area between curves and the x-axis, N_C is the number darts which land inside the circle and N is the total number which land in the square.

This implies that

$$\frac{A}{16} = \frac{N_C}{N} \Rightarrow A = \frac{16N_C}{N}$$

Determination of area under a curve Monte Carlo



In [13]:

```
import math
import random
#Hastily written on 6/6 at 3:30 PM
#Raymond Atta-Fynn
n_attempts = 100000
n_accept = 0

for i in range(n_attempts):
    x = random.uniform(-2,2)
    y = random.uniform(-4,0)
    yc=x**2-4


    if yc <= y <= 0:
        n_accept += 1


area = 16*(n_accept/n_attempts)
print(area)
```

Rosenbrock Function Minimization using Steepest Descent

Function minimization

Rosenbrock function: $E(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - {x_1}^2)^2$

The partial derivatives are:

$$\frac{\partial E}{\partial x_1} = -2(1 - x_1) - 400x_1(x_2 - {x_1}^2)$$

$$\frac{\partial E}{\partial x_2} = 200\,(x_2 - {x_1}^2)$$

Exact results of minimization: $(x_1, x_2) = (1,1) \qquad E_{min} = 0$

Rosenbrock function Minimization using Steepest Descent

In [21]:

```
import math
#Hastily written on 6/6 at 3:30 PM
#Raymond Atta-Fynn

def rosenbrock(x1,x2):
    E=(1 - x1)**2 + 100*(x2 - x1**2)**2
    return E

def rosenbrock_gradient(x1,x2):
    grad1= -2*(1-x1) - 400*x1*(x2 - x1**2)
    grad2= 200*(x2 - x1**2)
    return grad1,grad2

x1 = 0
x2 = 2
e = 0.000001
max_iterations = 20000
alpha = 0.1

for i in range(max_iterations):
    gradx,grady=rosenbrock_gradient(x1,x2)
    delta = math.sqrt(gradx**2 + grady**2)
    if delta < e:
        break
    alpha = 2*alpha
    while rosenbrock(x1-alpha*gradx,x2-alpha*grady)>=rosenbrock(x1,x2):
        alpha=alpha/2

    x1=x1-alpha*gradx
    x2=x2-alpha*grady

print('The solution converged')
print(' x1= %.5f ,' %x1,' x2= %.5f ,' %x2,' and the minimum_value of the function is %.5f'%rosenbrock(x1,x2))
```