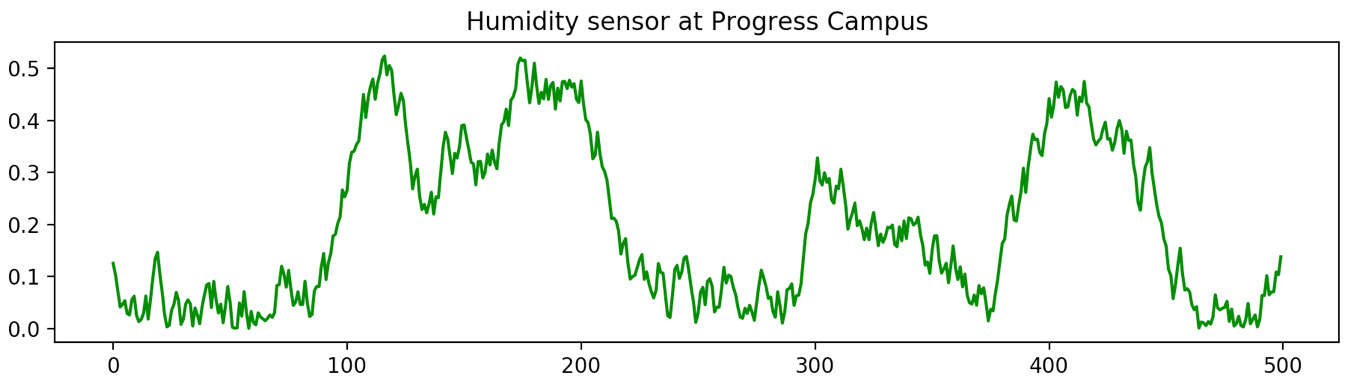
Networking for Software Developers

This is a group lab that will contribute towards your final project. One person will submit this work.

# Lab 8 – Data Generator

For this lab and all other labs in this course, you are not permitted to use any external libraries unless covered in class.

Most application depends on data. In IoT data is collected/generate by sensors. Since, we do not have access to sensors, we will have to generate data via code. The plotting a sufficient number of these data points (500 in this case) should get the following diagram or something very similar:

You will pick a quantity that you would like to simulate (such as temperature, humidity, barometric pressure, customers arriving at a mall, or just with an alternate descriptor) and decide what shape you will emulate. This will guide you in selecting sensible value base lines for your data values. e.g., If you pick inside temperature then normal range will be 18-21oC.

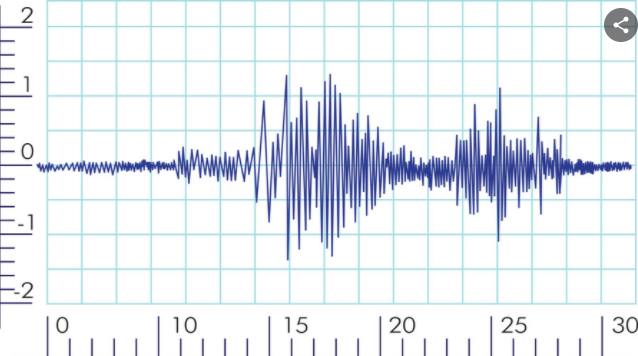
Design and build a class that will model your sensor reasonably well. Notice how the peaks do not occur at regular interval, nor are they the same height. Even the squiggles are the same shapes.

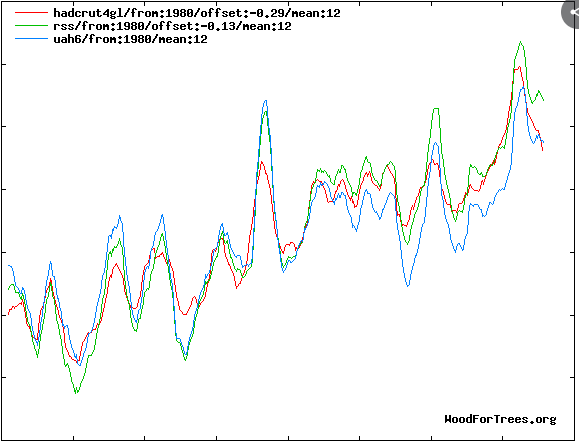
You will create a class with a single public member (property) that give you a “random” value in your intended range and properties. You should create a private method that actually generates a normalized value [a normalized value will be in the range 0 to 1]. And then transform this value by scaling and or transposing to the range that you are interested in.

Your class must have a fair amount of customization but at the same time should be easy to use, so provide a constructor with lots of default values. Make it so that you can generate your data by repeated accessing a property of the class instance. There must not be a limit to the number of values that you can get from your generator.

*All you your subsequent labs MUST use this class when working with random values.*

See some sample over page.





## Due:

See the course calendar.

## Requirements:

1. Create a **class** that does the following:
2. Create a **private method** that will generate random values in the range 0-1. Look at the examples at the end of this document for ideas on approaching this.  
   This method may take argument that can be passed by the property below.
3. Create a **public** **property** that will use the above member to return a value in your preferred range. You may use a simple transformation of the form:
4. You will use the Matplotlib library to display your data values. The output must look professional. You will label your **output axes** and **title** appropriately.
5. Code to drive (run your program)

See the appendix of this document for some code sample and possible directions to explore. You will need some combination of the last three examples.

* Use generator\_4() will give peaks and valleys. You may also use a sin function or some other function that will give you the base shape.
* Use generator\_3() to change the length (or frequency) of the peaks.
* Use generator\_2() and to get the squiggles.

DO NOT USE THE CODE AS IS! Look at the intension behind the code.

#### Submission

1. You must use only the libraries that are available in the standard python distribution.
2. Your code file will be named group\_«your\_group\_number»\_data\_generator.py e.g. group\_1\_data\_generator.py
3. Must be uploaded to course dropbox before the deadline.
4. See schedule for due date.

## Sample Code:

The following example illustrate the various possibilities in generating data. You can also play with the jupyter notebook at:

https://colab.research.google.com/drive/1QsxHRlUVITd-aQ0CMf\_ubVw7qIdzXq\_V?usp=sharing

#### Constant value

The first example gives you a constant value regardless of how many times you call it.

import matplotlib.pyplot as plt

def generator\_1() -> int:

'''

This is the greatest generator.

It returns Narendra's favourite number

'''

return 10

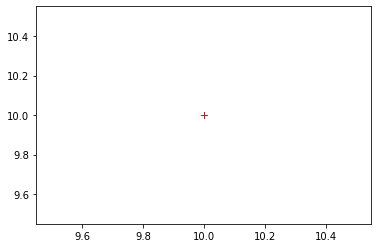
number\_of\_values = 200

y = [generator\_1() for \_ in range(number\_of\_values)]

x = [generator\_1() for \_ in range(number\_of\_values)]

plt.plot(x, y, 'r+')

plt.show()



#### Uniform values

The second example gives you a uniformly random value. It uses the randint() method of the random class that returns a value in the interval [a, b]. Uniform distributed values occur frequently in everyday situations such as the odds of getting a particular value on the toss of an un-biased die.

import matplotlib.pyplot as plt

import random

def generator\_2() -> int:

'''

This generator gives you a uniform random number in a 0 to 20

'''

return random.randint(0, 20)

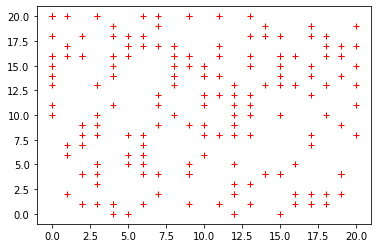
number\_of\_values = 200

y = [generator\_2() for \_ in range(number\_of\_values)]

x = [generator\_2() for \_ in range(number\_of\_values)]

plt.plot(x, y, 'r+')

plt.show()



#### Standard values

The third example gives you a normal random value. It uses the gauss() method of the random class that returns a value based on a mean and a standard deviation. Normal/standard distributed values also occur frequently in everyday situations such as the number of students in a queue waiting for the TTC bus are the Progress terminal. The is different because the number of students in the line quickly builds up to a maximum when the bus has arrived and a minimum when there is no bus.

import matplotlib.pyplot as plt

import random

def generator\_3() -> int:

return random.gauss(10, 1.0)

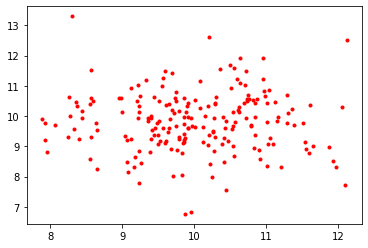
number\_of\_values = 200

y = [generator\_3() for \_ in range(number\_of\_values)]

x = [generator\_3() for \_ in range(number\_of\_values)]

plt.plot(x, y, 'r.')

plt.show()



If you increase the number of points, you will see there is a cluster at the center of the grid.

#### Pattern values

The fourth example gives you a value that follows a predictable pattern. It uses gauss method of the random class that returns a value based on a mean and a standard deviation. Normal/standard distributed values also occur frequently in everyday situations such as the number of students in a queue waiting for the TTC bus are the Progress terminal. The is different because the number of students in the line quickly builds up to a maximum when the bus has arrived and a minimum when there is no bus.

value = {'base':10, 'delta': 0.15}

def generator\_4(increment = True) -> float:

if increment:

value['base'] += value['delta']

else:

value['base'] -= value['delta']

return value['base']

number\_of\_values = 200

y = [generator\_4((x % 50) > 24) for x in range(number\_of\_values)]

plt.plot(y, 'g')

plt.show()

