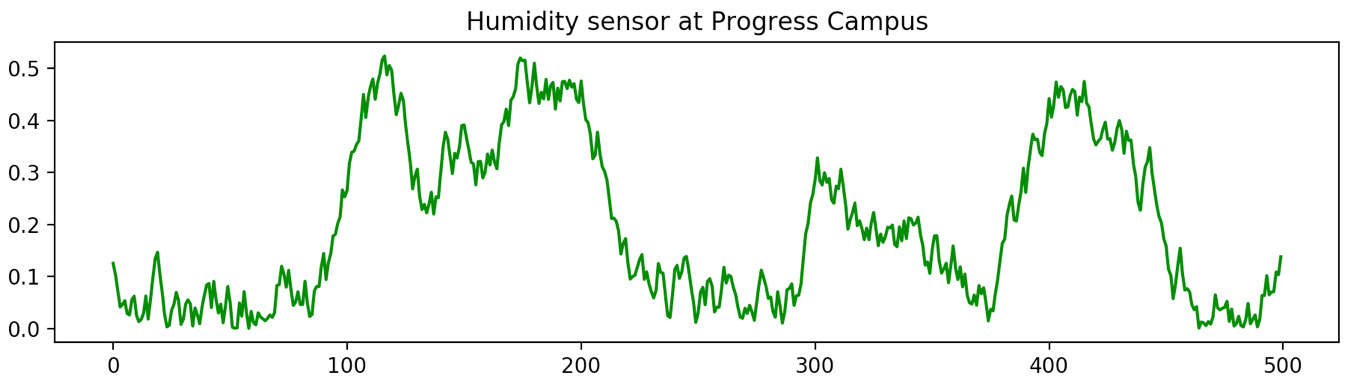
Networking for Software Developers

This is a group lab that will contribute towards your final project. One person will submit this work. I don’t know if there is a group dropbox???

# Lab 8 – Simulator.

Most application depends on data. In IoT data is collected/generate by sensors. We do not have access to sensors, so we will have to generate data via code. You will create a class with a member (property) that give you a “random” value. If you generate a large number of these values (at least 500) and plot it using matplotlib you should get the following diagram or something very similar:

## Due:

Before the start of week 10

## Requirements:

1. You will pick a quantity that your value will mimic (such as temperature, humidity, barometric pressure, customers at a mall, or just with an alternate descriptor). This will guide you in picking sensible value base lines for your data values. If you pick inside temperature then normal range will be 18-21oC.
2. 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.
3. 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.
4. Make it so that you can generate your data by repeated calling a method or accessing a property of the class instance.

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
* 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. Your code file will be named group\_«your\_group\_number»\_data\_generator.py e.g. group\_1\_data\_generator.py
2. Must be uploaded to course dropbox.
3. This is due by the start of week 10

## 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

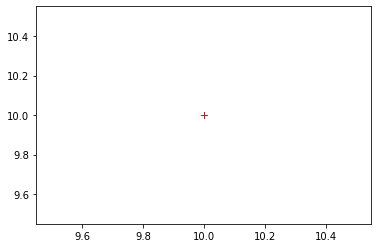
number\_of\_values = 200

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)

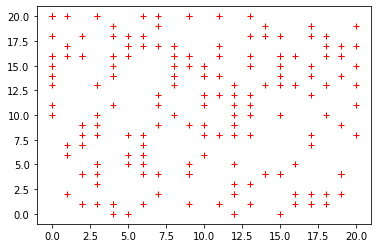
number\_of\_values = 200

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)

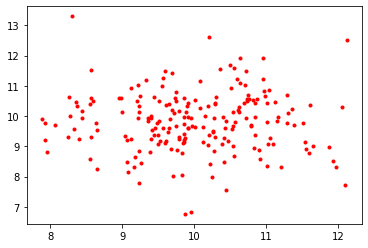
number\_of\_values = 200

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()

