DAILY ASSESSMENT

Date:	**27/05/2020**	**Name:**	**CHANDANA.R**
Course:	DSP	**USN:**	**4AL16EC017**
Topic:	FFT, FIR, IIR filters CWT & DWT, welchs method and windowing, ECG signals analysis using MATLAB	**Semester & Section:**	**8(A)**
Github Repository:	**Chandana-shaiva**		

FORENOON SESSION DETAILS

Report:-

- Fourier Transforms:

The Fourier transform of a function f is traditionally denoted , by adding a circumflex to the symbol of the function. There are several common conventions for defining the Fourier transform of an integrable function,

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i x \xi}\,dx,$$

The statement that f can be reconstructed from is known as the Fourier inversion theorem, and was first introduced in Fourier's although what would be considered a proof by modern standards was not given until much later. The functions f and f^ often are referred to as a *Fourier integral pair* or *Fourier transform pair*

- Fast Fourier Transform:

Let $x_0, \ldots, x_{N-1}$ be complex numbers. The DFT is defined by the formula

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k n / N} \qquad k = 0, \ldots, N-1,$$

where $e^{i2\pi/N}$ is a primitive Nth root of 1.

$$X_0 = x_0 e^{-j2\pi(0)(0)/4} + x_1 e^{-j2\pi(1)(0)/4} + x_2 e^{-j2\pi(2)(0)/4} + x_3 e^{-j2\pi(3)(0)/4}$$

$$X_0 = x_0 + x_1 + x_2 + x_3$$

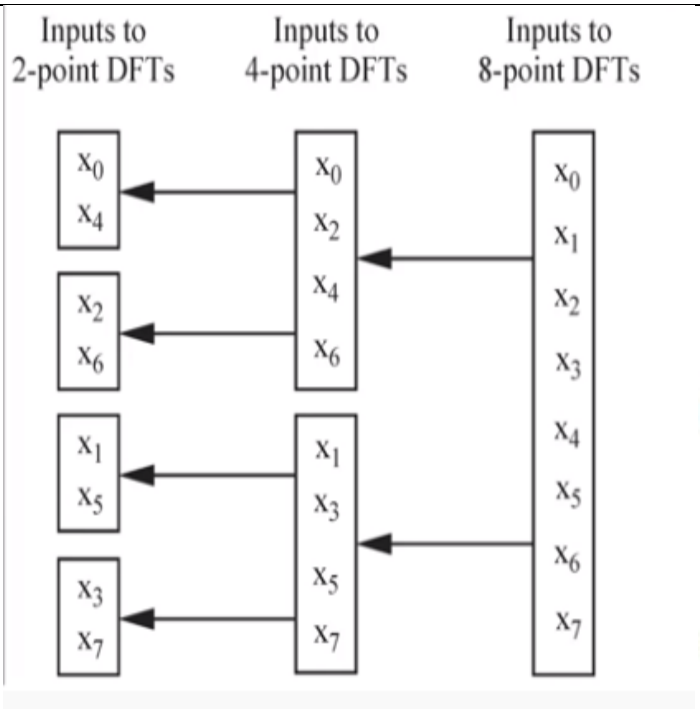
$$X_2 = x_0 - x_1 + x_2 - x_3$$

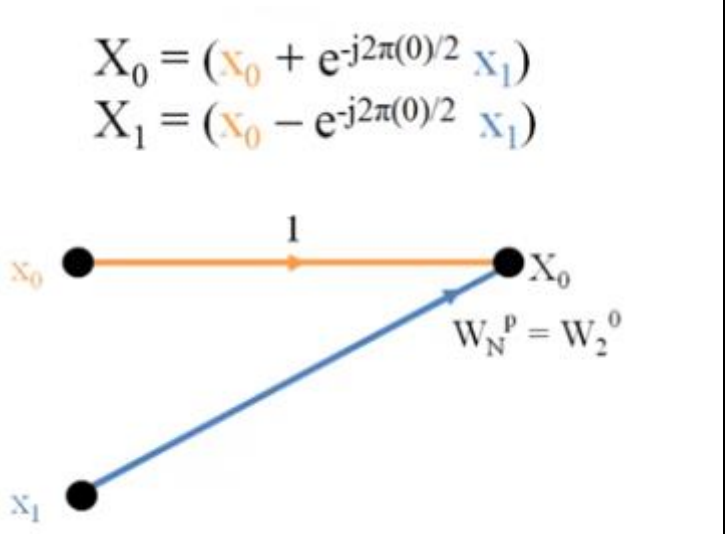
$$X_3 = x_0 + x_1 e^{-j2\pi 3/4} - x_2 - x_3 e^{-j2\pi 3/4}$$

This compositional viewpoint immediately provides the simplest and most common multidimensional DFT algorithm, known as the **row-column** algorithm (after the two-dimensional case, below). That is, one simply performs a sequence of *d* one-dimensional FFTs (by any of the above algorithms): first you transform along the n_1 dimension, then along the n_2 dimension, and so on (or actually, any ordering works). This method is easily shown to have the usual O(*N* log *N*) complexity, where N= N1, N2,……Nd is the total number of data points transformed. In particular, there are N/N_1 transforms of size N_1, etcetera, so the complexity of the sequence of FFTs is:

$$X_0 = (x_0 + e^{-j2\pi(0)/2} x_2) + e^{-j2\pi(0)/4} (x_1 + e^{-j2\pi(0)/2} x_3)$$
$$X_1 = (x_0 - e^{-j2\pi(0)/2} x_2) + e^{-j2\pi(1)/4} (x_1 - e^{-j2\pi(0)/2} x_3)$$
$$X_2 = (x_0 + e^{-j2\pi(0)/2} x_2) - e^{-j2\pi(0)/4} (x_1 + e^{-j2\pi(0)/2} x_3)$$
$$X_3 = (x_0 - e^{-j2\pi(0)/2} x_2) - e^{-j2\pi(1)/4} (x_1 - e^{-j2\pi(0)/2} x_3)$$

A **fast Fourier transform** (**FFT**) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies.[1] This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical.





- **Simple and Easy Tutorial on FFT Fast Fourier Transform Matlab**

```
Fs = 1000;
T = 1/Fs;
L = 1500;
t = (0:L-1)*T;
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
X = S + 2*randn(size(t));

plot(1000*t(1:50),X(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('t (milliseconds)')
ylabel('X(t)')

Y = fft(X);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(L/2))/L;

plot(f,P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')

Y = fft(S);
```

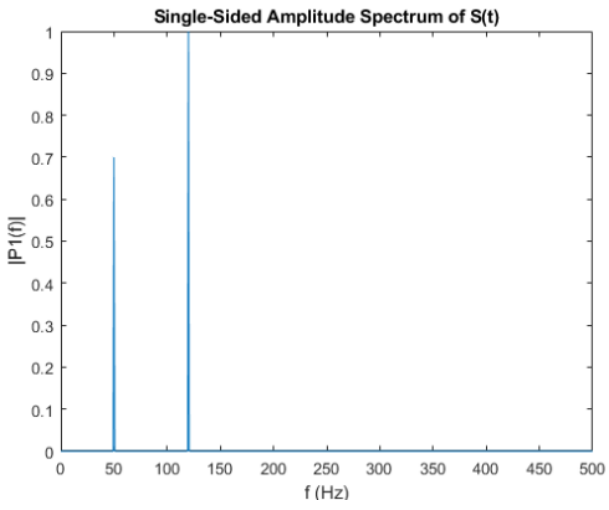
```
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);

plot(f,P1)
title('Single-Sided Amplitude Spectrum of S(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')

Fs = 100;
t = -0.5:1/Fs:0.5;
 L = length(t);

X = 1/(4*sqrt(2*pi*0.01))*(exp(-t.^2/(2*0.01)));
plot(t,X)
title('Gaussian Pulse in Time Domain')
xlabel('Time (t)')
ylabel('X(t)')
n = 2^nextpow2(L);
Y = fft(X,n);
f = Fs*(0:(n/2))/n;
P = abs(Y/n);

plot(f,P(1:n/2+1))
title('Gaussian Pulse in Frequency Domain')
xlabel('Frequency (f)')
ylabel('|P(f)|')
```



➢ Easy Introduction to Wavelets

Wavelets, in general, are constructed by taking the dilations and translations of a single function with sufficient decay in both the time and frequency domains. The definition adopted here for "sufficient" decay is that a function Y(x) and its Fourier transform, denoted by Y(f) , both decay faster than I x I-1 and I f I - r, respectively;

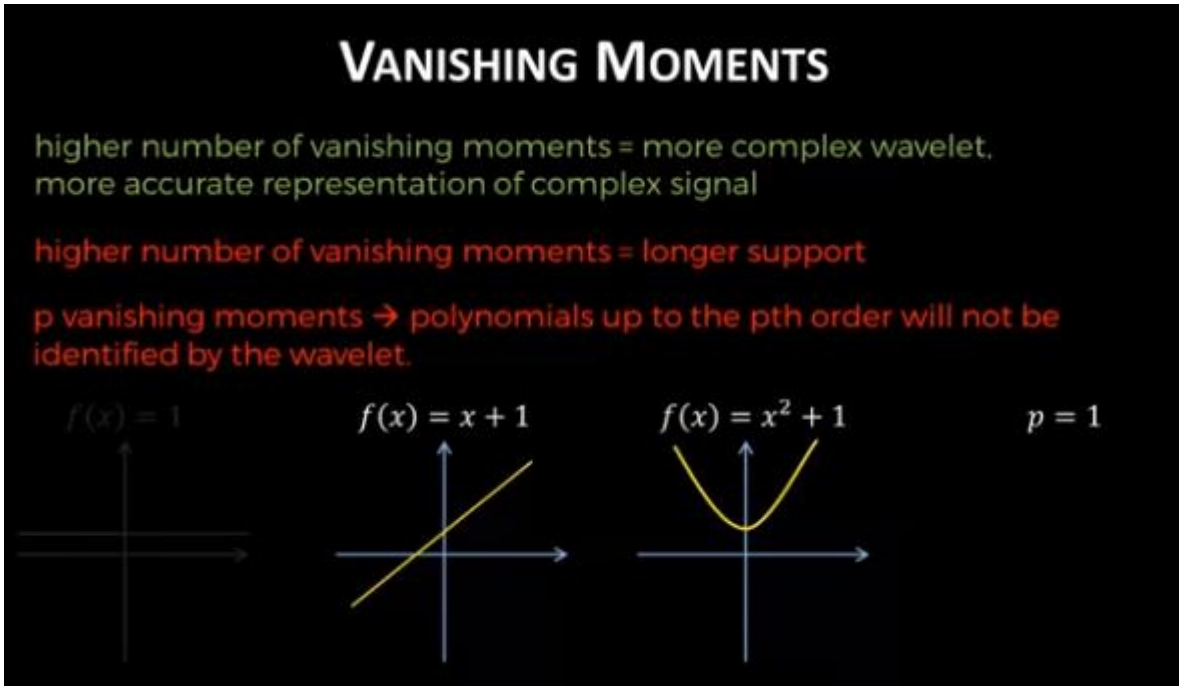
$$\int_{-\infty}^{\infty} |x|^{-1} |\Psi(x)|^2 \, dx < \infty,$$

and

$$\int_{-\infty}^{\infty} |f|^{-1} |\hat{\Psi}(f)|^2 \, df < \infty.$$

In most situations it is useful to restrict ψ to be a continuous function with a higher number *M* of vanishing moments, i.e. for all integer $m < M$

$$\int_{-\infty}^{\infty} t^m \, \psi(t) \, dt = 0.$$

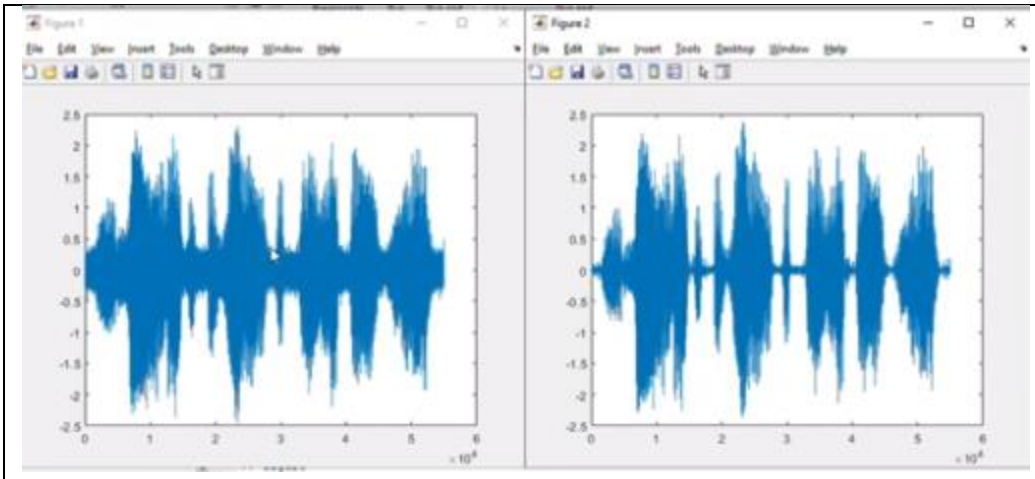


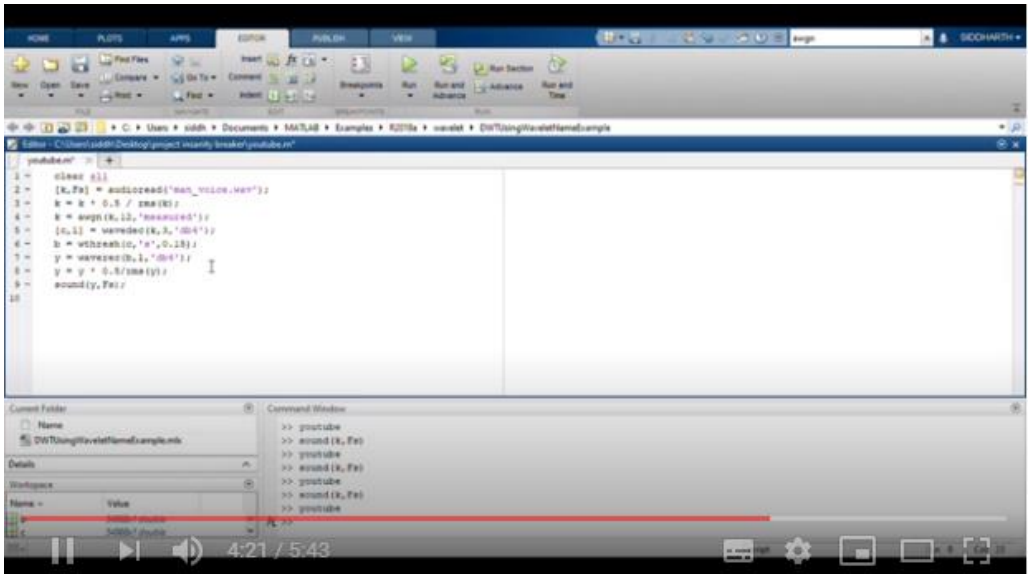
Simple audio denoising using wavelet decomposition and thresholding, wavelet denoising [MATLAB]

```
rng default;
[X,XN] = wnoise('bumps',10,sqrt(6));
subplot(211)
plot(X); title('Original Signal');
AX = gca;
AX.YLim = [0 12];
subplot(212)
plot(XN); title('Noisy Signal');
AX = gca;
AX.YLim = [0 12];


xd = wdenoise(XN,4);
figure;
plot(X,'r')
hold on;
plot(xd)
legend('Original Signal','Denoised Signal','Location','NorthEastOutside')
axis tight;
hold off;


xdMODWT = wden(XN,'modwtsqtwolog','s','mln',4,'sym4');
figure;
plot(X,'r')
hold on;
plot(xdMODWT)
legend('Original Signal','Denoised Signal','Location','NorthEastOutside')
axis tight;
hold off;
```



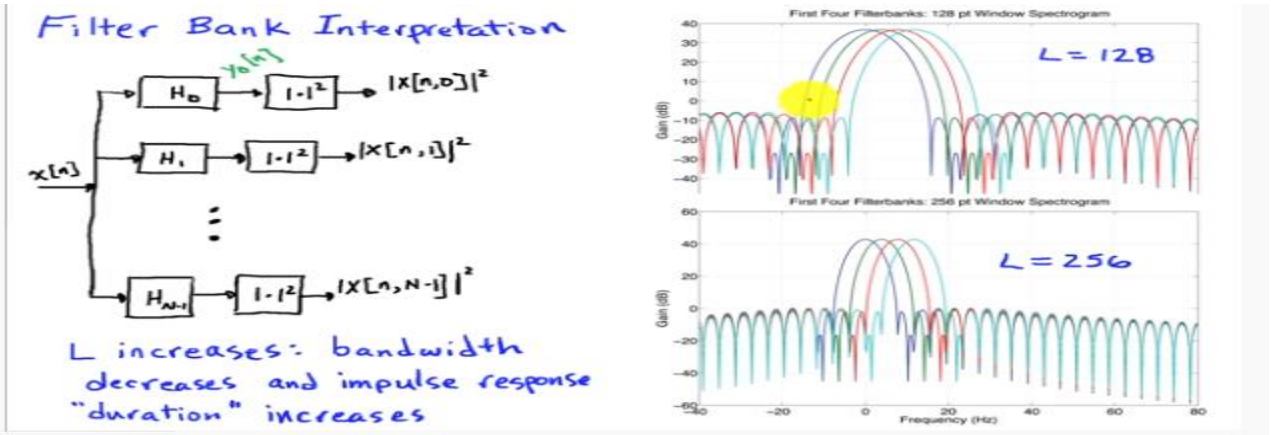


➢ Short-time Fourier Transform and the Spectogram

The **Short-time Fourier transform** (**STFT**), is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time.[1] In practice, the procedure for computing STFTs is to divide a longer time signal into shorter segments of equal length and then compute the Fourier transform separately on each

shorter segment. This reveals the Fourier spectrum on each shorter segment. One then usually plots the changing spectra as a function of time, known as a spectrogram or waterfall plot.

If the DFT coefficients of each frame are placed into a separate column of a matrix, the STFT can be represented as a matrix of coefficients, where the column index represents time and the row index is associated with the frequency of the respective DFT coefficient. If the magnitude of each coefficient is computed, the resulting matrix can be treated as an image and, as a result, it can be visualized. This image is known as the spectrogram of the signal and presents the evolution of the signal in the time-frequency domain. To generate the spectrogram, we can use the magnitude or the squared magnitude of the STFT coefficients on a linear or logarithmic scale (dB). In MATLAB, the spectrogram of a signal is implemented in the spectrogram () function, which can plot the spectrogram and return the matrix of STFT coefficients, along with the respective time and frequency axes. In this book, we will mainly use the spectrogram as a visualization tool. The STFT coefficients will be extracted, when required, by means of a more general function that we have developed for short-term processing purposes.



Short-time Fourier Transform and the Spectogram

- **Power Spectrum Estimation Examples: Welch's Method**

Welch's method, named after Peter D. Welch, is an approach for spectral density estimation. It is used in physics, engineering, and applied mathematics for estimating the power of a signal at different frequencies. The method is based on the concept of using periodogram spectrum estimates, which are the result of converting a signal from the time domain to the frequency domain. Welch's method is an improvement on the standard periodogram spectrum estimating method and on Bartlett's method, in that it reduces noise in the estimated power spectra in exchange for reducing the frequency resolution. Due to the noise caused by imperfect and finite data, the noise reduction from Welch's method is often desired.

- ECG Signal Analysis Using MATLAB

A real-time QRS detection algorithm, which references [1, lab one], [3] and [4], is developed in Simulink with the assumption that the sampling frequency of the input ECG signal is always 200 Hz (or 200 samples/s). However, the recorded real ECG data may have different sampling frequencies ranging from 200 Hz to 1000 Hz, e.g., 360 Hz in this example. To bridge the different sampling frequencies, a sample rate converter block is used to convert the sample rate to 200 Hz. A buffer block is inserted to ensure the length of the input ECG signal is a multiple of the calculated decimation factor of the sample-rate converter block.

The ECG signal is filtered to generate a windowed estimate of the energy in the QRS frequency band. The filtering operation has these steps:

1. FIR Bandpass filter with a pass band from 5 to 26 Hz

2. Taking the derivative of the bandpass filtered signal

3. Taking the absolute value of the signal

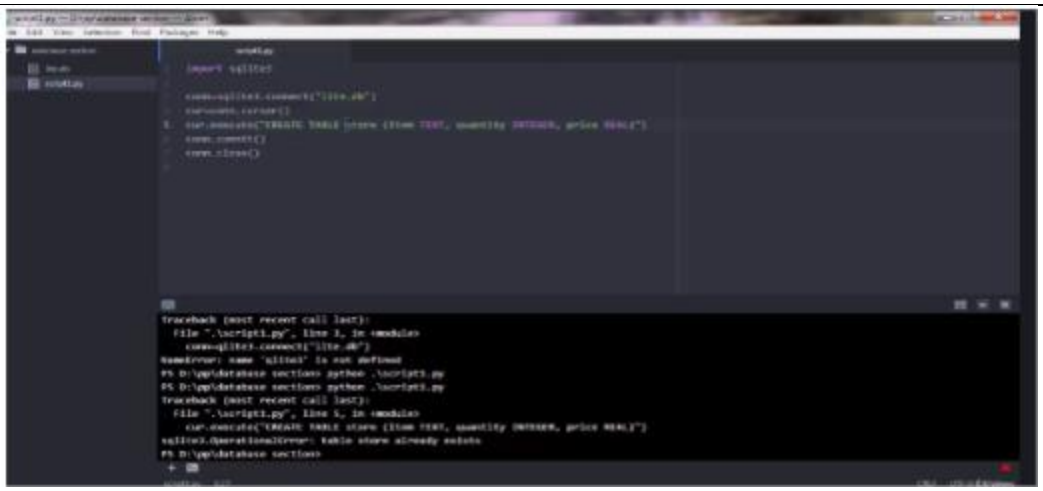
4. Averaging the absolute value over an 80 ms window

DAILY ASSIGEMENTS DETAILS:

Date:	**27/05/2020**	**Name:**	**CHANDANA.R**
Course:	Python	**USN:**	**4AL16EC017**
Topic:	Graphical user interface with tkinter, interacting with database	**Semester & Section:**	**8(A)**
Github Repository:	**Chandana-shaiva**		

AFTERNOON SESSION

Session image:



Report:

➢ MULTI WIDGET GUI:

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

To create a tkinter app:

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

Importing tkinter is same as importing any other module in the Python code. Note that the name of the module in Python 2.x is 'Tkinter' and in Python 3.x it is 'tkinter'.

import tkinter

There are two main methods used which the user needs to remember while creating the

Python application with GUI.

- Tk(screenName=None,

baseName=None,

className='Tk', useTk=1):

To create a main window, tkinter offers a method

'Tk(screenName=None,

baseName=None,

className='Tk', useTk=1)'.

To change the name of the window, you can change the className to the desired one. The basic code used to create the main window of the application is:

m=tkinter.Tk() where m is the name of the main window object

- There is a method known by the name mainloop() is used when your application is ready to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

  m.mainloop()
- From tkinter import *
- # Create an empty Tkinter window
- Window = Tk ()
- Def from_kg ():
- # Get user value from input box and multiply by 1000 to get kilograms
- Gram = float(e2_value.get())*1000
- # Get user value from input box and multiply by 2.20462 to get pounds
- Pound =float(e2_value.get())*2.20462
- # Get user value from input box and multiply by 35.274 to get ounces

- ounce =float(e2_value.get())*35.274

Empty the Text boxes if they had text from the previous use and fill them again

```
t1.delete("1.0",END)

# Deletes the content of the Text box from start to END
t1.insert(END, gram)

# Fill in the text box with the value of gram variable
t2.delete("1.0",END)
    t2.insert(END, pound)
    t3.delete("1.0",END)
    t3.insert(END, ounce)

# Create a Label widget with "Kg" as label
e1=Label(window, text="Kg")
e1.grid(row=0,column=0)

# The Label is placed in position 0, 0 in the window
e2_value=StringVar()

# Create a special StringVar object
e2=Entry (window, Textvariable =e2_value)

# Create an Entry box for users to enter the value
e2.grid(row=0,column=1
```

```
# Create a button widget
# The from_kg() function is called when the button is pushed
b1 = Button(window, text= "Convert", command=from_kg)
b1.grid(Row = 0,Column = 2
# Create three empty text boxes, t1, t2, and t3
t1=Text (window, height=1,width=20)
t1.grid(row=1,column=0)
t2 =Text(window, height=1,width=20)
t2.grid(row=1,column=1)
t3 =Text( Window, height=1,Width = 20)
t3.grid(row=1,column=2)

# This makes sure to keep the main window open
window.Mainloop ()
```