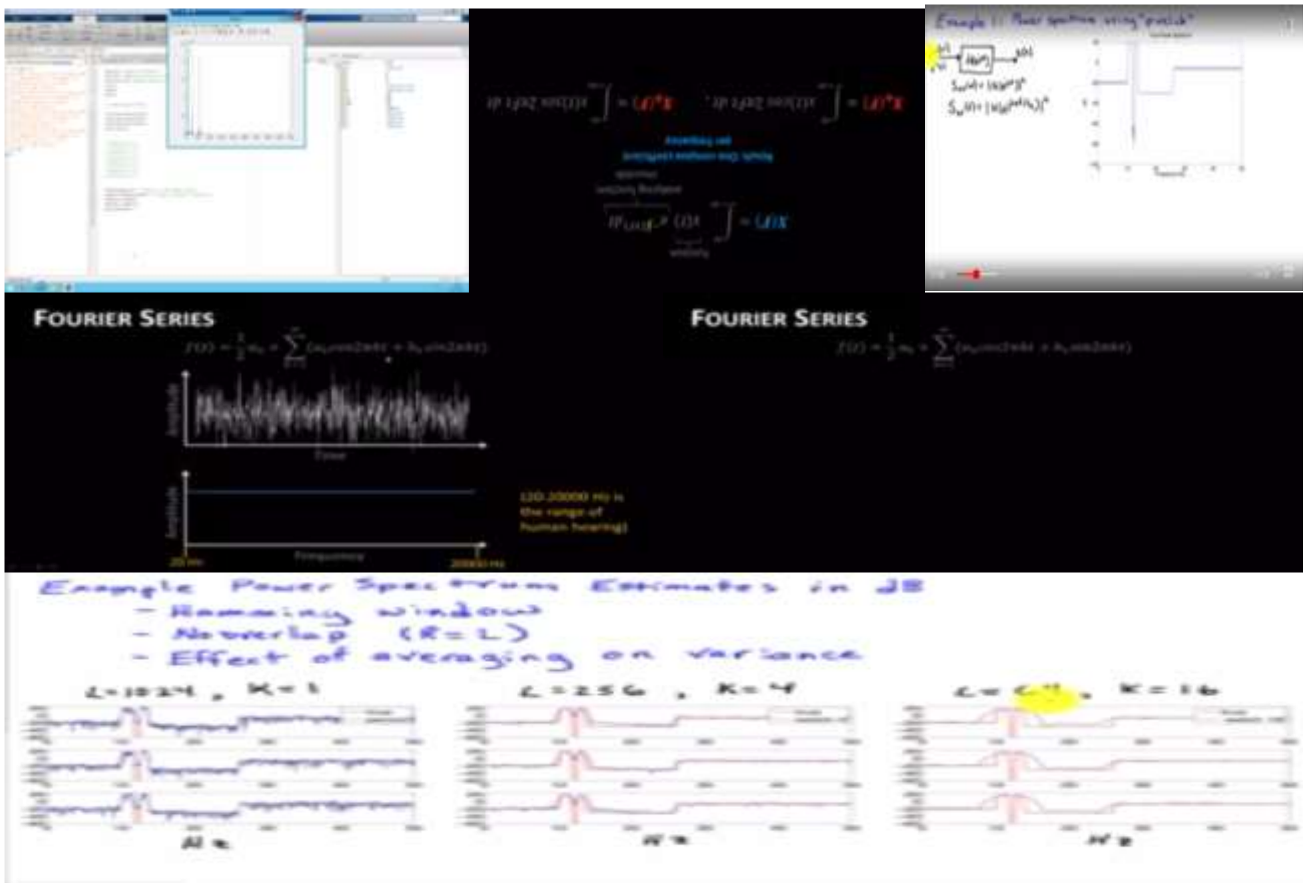
DAILY ASSESSMENT FORMAT

Date:	**27-05-2020**	**Name:**	**PRINCIA MELITA DSOUZA**
Course:		**USN:**	**4AL17EC075**
Topic:	**FFT,FIR,IIR,CWT & DWT**	**Semester & Section:**	**6TH B**
Github Repository:	**MELITA-1999**		

FORENOON SESSION DETAILS

Image of session


Example 1: Power spectrum using "pwelch"
FOURIER SERIES
FOURIER SERIES
(20-20000 Hz is the range of human hearing)
Example Power Spectrum Estimates in dB
- Hamming window
- No overlap (R=L)
- Effect of averaging on variance
L=1024, K=1
L=256, K=4
L=64, K=16

Report – Report can be typed or hand written for up to two pages.

FOURIER TRANSFORM

A Fourier transform (FT) is a mathematical transform which decomposes a function (often a function of time, or a signal) into its constituent frequencies, such as the expression of a musical chord in terms of the volumes and frequencies of its constituent notes. The term Fourier transform refers to both the frequency domain representation and the mathematical operation that associates the frequency domain representation to a function of time.

The Fourier transform of a function of time is a complex-valued function of frequency, whose magnitude represents the amount of that frequency present in the original function, and whose argument is the phase offset of the basic sinusoid in that frequency. The Fourier transform is not limited to functions of time, but the domain of the original function is commonly referred to as the time domain. There is also an inverse Fourier transform that mathematically synthesizes the original function from its frequency domain representation.

FAST FOURIER TRANSFORM

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.[This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, it manages to reduce the complexity of computing the DFT from which arises if one simply applies the definition of DFT, to where is the data size. The difference in speed can be enormous, especially for long data sets where N may be in the thousands or millions. In the presence of round-off error, many FFT algorithms are much more accurate than evaluating the DFT definition directly or indirectly. There are many different FFT algorithms based on a wide range of published theories, from simple complex-number arithmetic to group theory and number theory.

Fast Fourier transforms are widely used for applications in engineering, music, science, and mathematics. The basic ideas were popularized in 1965, but some algorithms had been derived as early as 1805. In 1994, Gilbert Strang described the FFT as "the most important numerical algorithm of our lifetime", and it was included in Top 10 Algorithms of 20th Century by the IEEE magazine Computing in Science & Engineering.

The best-known FFT algorithms depend upon the factorization of N, but there are FFTs with O complexity for all N, even for prime N. Many FFT algorithms only depend on the fact that is an N-th primitive root of unity, and thus can be applied to analogous transforms over any finite field, such as number-theoretic transforms. Since the inverse DFT is the same as the DFT, but with the opposite sign in the exponent and a 1/N factor, any FFT algorithm can easily be adapted for it.

FIR AND IIR

Two classes of digital filters are Finite Impulse Response (FIR) and Infinite Impulse Response (IIR). The term 'Impulse Response' refers to the appearance of the filter in the time domain. ... The mathematical difference between the IIR and FIR implementation is that the IIR filter uses some of the filter output as input.

INTRODUCTION ON WT

Introduction. What is a C++ library for developing web applications. ... While some developers choose to implement the application in JavaScript and use a client-server protocol to access server-side resources, this has a number of inherent downsides. You need to let go of static typing (or add another layer, such as GWT

CWT & DWT

This topic describes the major differences between the continuous wavelet transform (CWT) and the discrete wavelet transform (DWT) – both decimated and nondecimated versions. cwt is a discretized version of the CWT so that it can be implemented in a computational environment. This discussion focuses on the 1-D case, but is applicable to higher dimensions. The cwt wavelet transform compares a signal with shifted and scaled (stretched or shrunk) copies of a basic wavelet. If is a wavelet centered at t=0 with time support on [-T/2, T/2], then is centered at t = u with time support [-sT/2+u, sT/2+u]. The cwt function uses L1 normalization so that all frequency amplitudes are normalized to the same value. If 0<s<1, the wavelet is contracted (shrunk) and if s>1, the wavelet is stretched. The mathematical term for this is dilation. See Continuous Wavelet Transform and Scale-Based Analysis for examples of how this operation extracts features in the signal by matching it against dilated and translated wavelets. The major difference between the CWT and discrete wavelet transforms, such as the dwt and modwt, is how the scale parameter is discretized. The CWT discretizes scale more finely than the discrete wavelet transform. In the CWT, you typically fix some base which is a fractional power of two, for example,where v is an integer greater than 1. The v parameter is often referred to as the number of "voices per octave". Different scales are obtained by raising this base scale to positive integer powers, for example. The translation parameter in the CWT is discretized to integer values, denoted here by m. The resulting discretized wavelets for the CWT are The reason v is referred to as the number of voices per octave is because increasing the scale by an octave requires v intermediate scales. Take for example and then increase the numerator in the exponent until you reach 4, the next octave. You move from to. There are v intermediate steps. Common values for v are 10,12,14,16, and 32. The larger the value of v, the finer the discretization of the scale parameter, s. However, this also increases the amount of computation required because the CWT must be computed for every scale. The difference between scales on a log2 scale is 1/v. See CWT-Based Time-Frequency Analysis and Continuous Wavelet Analysis of Modulated Signals for examples of scale vectors with the CWT. In the discrete wavelet transform, the scale parameter is always discretized to integer powers of 2, 2j, j=1,2,3,..., so that the number of voices per octave is always 1. The difference between scales on a log2 scale is always 1 for discrete wavelet transforms. Note that this is a much coarser sampling of the scale parameter, s, than is the case with the CWT. Further, in the decimated (downsampled) discrete wavelet transform (DWT), the translation parameter is always proportional to the scale. This means that at scale, 2j, you always translate by 2jm where m is a nonnegative integer. In nondecimated discrete wavelet transforms like modwt and swt, the scale parameter is restricted to powers of two, but the translation parameter is an integer like in the CWT. The discretized wavelet for the DWT takes the following form The discretized wavelet for the nondecimated discrete wavelet transform, such as the MODWT.

summarize:

The CWT and the discrete wavelet transforms differ in how they discretize the scale parameter. The CWT typically uses exponential scales with a base smaller than 2, for example 21/12 . The discrete wavelet transform always uses exponential scales with the base equal to 2. The scales in the discrete wavelet transform are powers of 2. Keep in mind that the physical interpretation of scales for both the CWT and discrete wavelet transforms requires the inclusion of the signal's sampling interval if it is not equal to one. For example, assume you are using the CWT and you set your base to . To attach physical significance to that scale, you must multiply by the sampling interval , so a scale vector covering approximately four octaves with the sampling interval taken into account is. Note that the sampling interval multiplies the scales, it is not in the exponent. For discrete wavelet transforms the base scale is always .

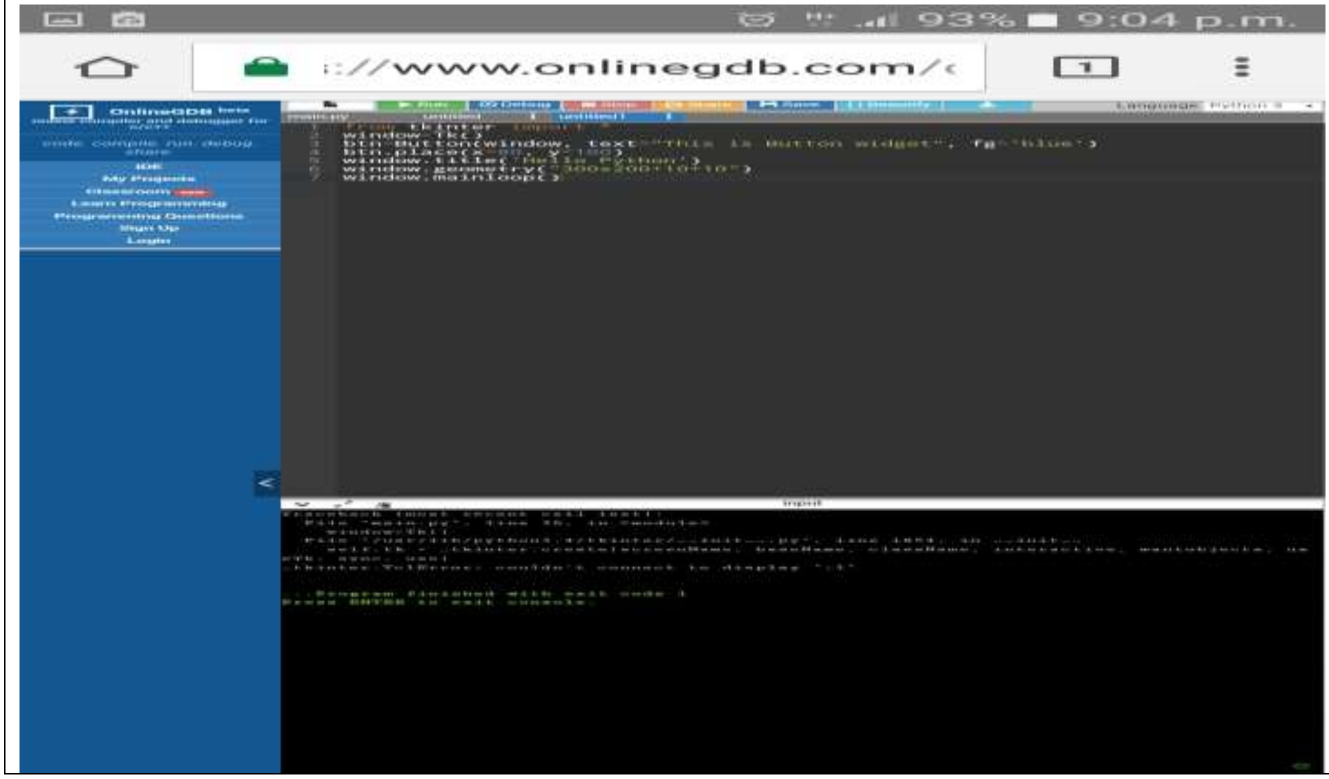
The decimated and nondecimated discrete wavelet transforms differ in how they discretize the translation parameter. The decimated discrete wavelet transform (DWT), always translates by an integer multiple of the scale, 2jm . The nondecimated discrete wavelet transform translates by integer shifts.These differences in how scale and translation are discretized result in advantages and disadvantages for the two classes of wavelet transforms. These differences also determine use cases where one wavelet transform is likely to provide superior results. Some important consequences of the discretization of the scale and translation parameter are:
The DWT provides a sparse representation for many natural signals. In other words, the important features of many natural signals are captured by a subset of DWT coefficients that is typically much smaller than the original signal. This "compresses" the signal. With the DWT, you always end up with the same number of coefficients as the original signal, but many of the coefficients may be close to zero in value. As a result, you can often throw away those coefficients and still maintain a high-quality signal approximation. With the CWT, you go from N samples for an N-length signal to a M-by-N matrix of coefficients with M equal to the number of scales. The CWT is a highly redundant transform. There is significant overlap between wavelets at each scale and between scales. The computational resources required to compute the CWT and store the coefficients is much larger than the DWT. The nondecimated discrete wavelet transform is also redundant but the redundancy factor is usually significantly less than the CWT, because the scale parameter is not discretized so finely. For the nondecimated discrete wavelet transform, you go from N samples to an L+1-by-N matrix of coefficients where L is the level of the transform.
The strict discretization of scale and translation in the DWT ensures that the DWT is an orthonormal transform (when using an orthogonal wavelet). There are many benefits of orthonormal transforms in signal analysis. Many signal models consist of some deterministic signal plus white Gaussian noise. An orthonormal transform takes this kind of signal and outputs the transform applied to the signal plus white noise. In other words, an orthonormal transform takes in white Gaussian noise and outputs white Gaussian noise. The noise is uncorrelated at the input and output. This is important in many statistical signal processing settings. In the case of the DWT, the signal of interest is typically captured by a few large-magnitude DWT coefficients, while the noise results in many small DWT coefficients that you can throw away. If you have studied linear algebra, you have no doubt learned many advantages to using orthonormal bases in the analysis and representation of vectors. The wavelets in the DWT are like orthonormal vectors. Neither the CWT nor the nondecimated discrete wavelet transform are orthonormal transforms. The wavelets in the CWT and nondecimated discrete wavelet transform are technically called frames, they are linearly-dependent sets.
The DWT is not shift-invariant. Because the DWT downsamples, a shift in the input signal does not manifest itself as a simple equivalent shift in the DWT coefficients at all levels. A simple shift in a signal can cause a significant realignment of signal energy in the DWT coefficients by scale. The CWT and nondecimated discrete wavelet transform are shift-invariant. There are some modifications of the DWT such as the dual-tree complex discrete wavelet transform that mitigate the lack of shift invariance in the DWT, see Critically Sampled and Oversampled Wavelet Filter Banks for some conceptual material on this topic and Dual-Tree Complex Wavelet Transforms for an example.
The discrete wavelet transforms are equivalent to discrete filter banks. Specifically, they are tree-structured discrete filter banks where the signal is first filtered by a lowpass and a highpass filter to yield lowpass and highpass subbands. Subsequently, the lowpass subband is iteratively filtered by the same scheme to yield narrower octave-band lowpass and highpass subbands. In the DWT, the filter outputs are downsampled at each successive stage. In the nondecimated discrete wavelet transform, the outputs are not downsampled. The filters that define the discrete wavelet transforms typically only have a small number of coefficients so the transform can be implemented very efficiently. For both the DWT and nondecimated discrete wavelet transform, you do not actually require an expression for the wavelet. The filters are sufficient. This is not the case with the CWT. The most

common implementation of the CWT requires you have the wavelet explicitly defined. Even though the nondecimated discrete wavelet transform does not downsample the signal, the filter bank implementation still allows for good computational performance, but not as good as the DWT. The discrete wavelet transforms provide perfect reconstruction of the signal upon inversion. This means that you can take the discrete wavelet transform of a signal and then use the coefficients to synthesize an exact reproduction of the signal to within numerical precision. You can implement an inverse CWT, but it is often the case that the reconstruction is not perfect. Reconstructing a signal from the CWT coefficients is a much less stable numerical operation. The finer sampling of scales in the CWT typically results in a higher-fidelity signal analysis. You can localize transients in your signal, or characterize oscillatory behavior better with the CWT than with the discrete.

Date:	**27-05-2020**	**Name:**	**PRINCIA MELITA DSOUZA**
Course:	**PYTHON**	**USN:**	**4AL17EC075**
Topic:	**GRAPHICAL USER INTERFACES WITH TKINTER INTERACTING WITH DATABASES**	**Semester & Section:**	**6TH B**

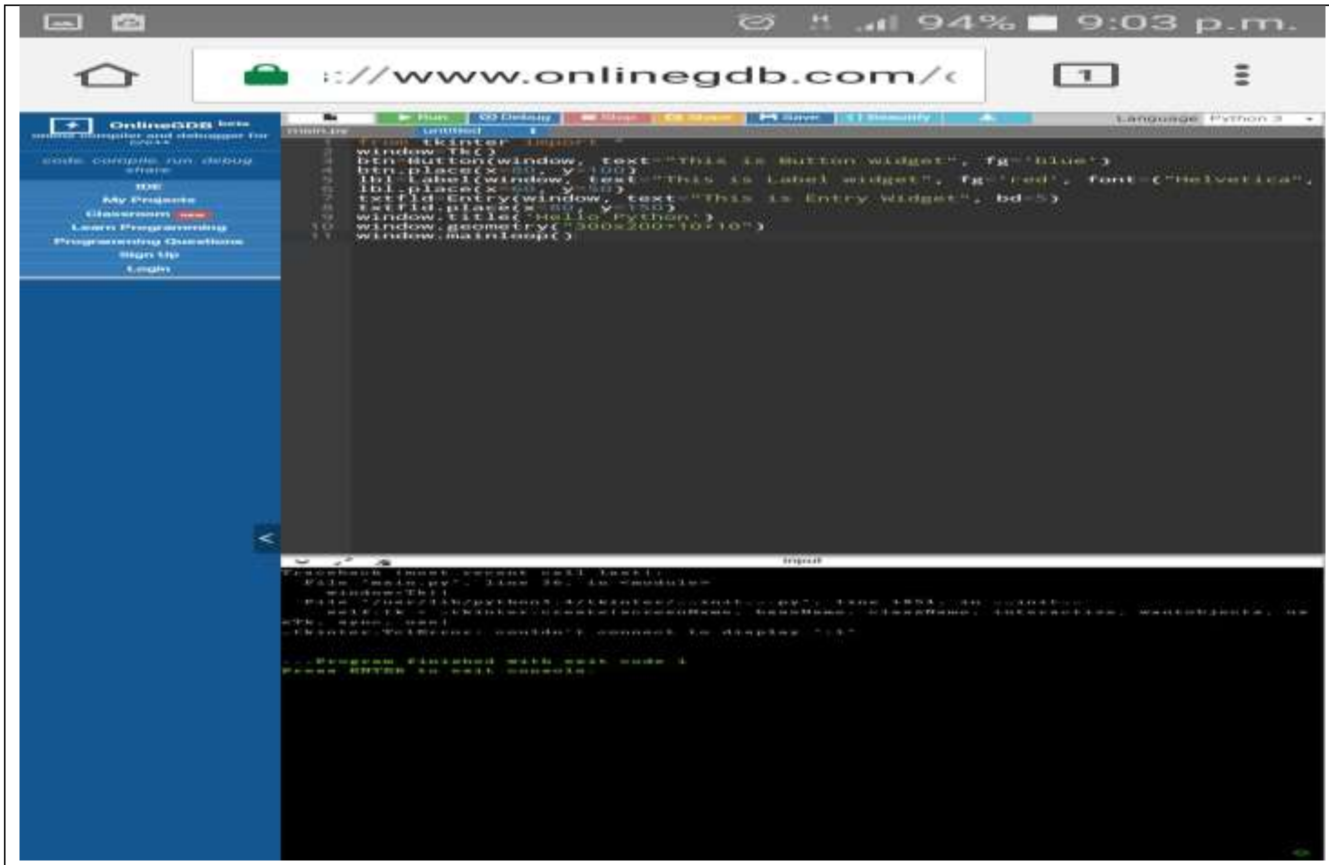
AFTERNOON SESSION DETAILS

Image of session



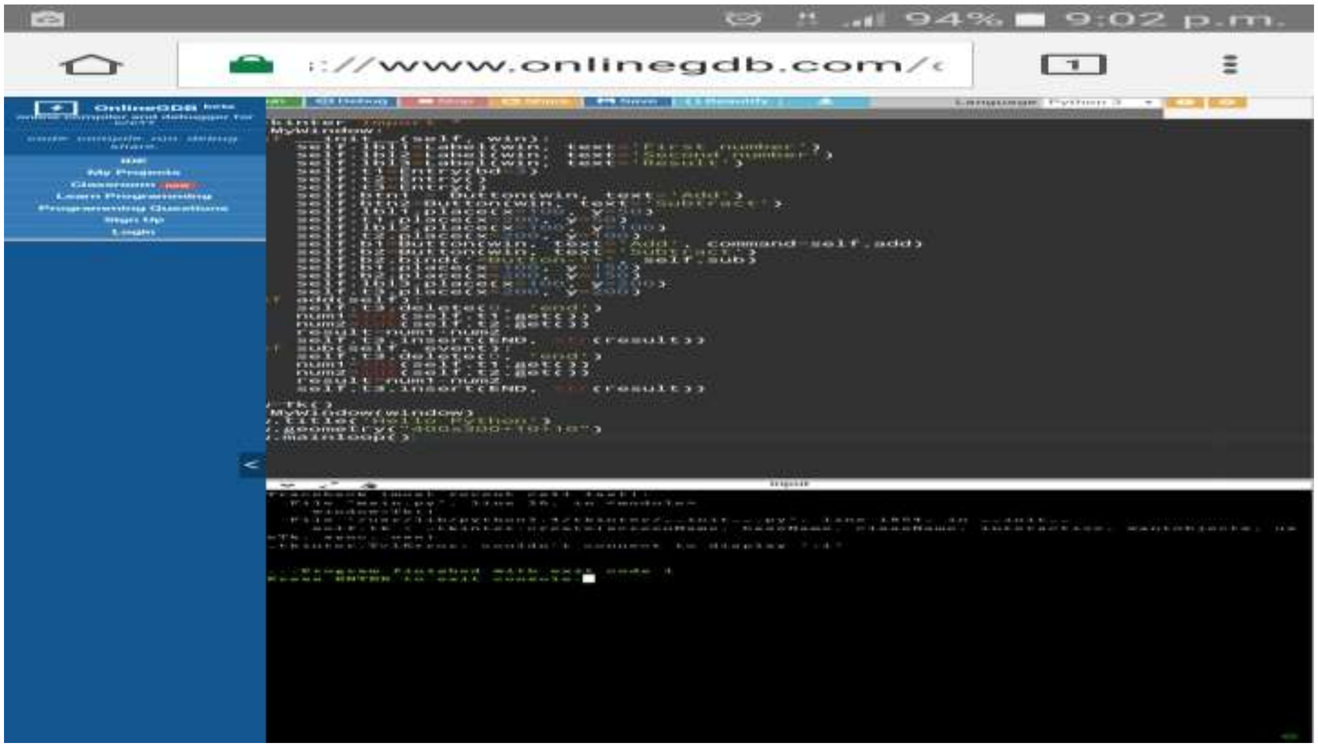
```
from tkinter import *
window=Tk()
btn=Button(window, text="This is Button widget", fg='blue')
btn.place(x=80, y=100)
window.title('Hello Python')
window.geometry("300x200+10+10")
window.mainloop()
```

```
Traceback (most recent call last):
  File "main.py", line [illegible], in <module>
    window=Tk()
  File "/usr/lib/python3.4/tkinter/__init__.py", line [illegible], in __init__
    self.tk = _tkinter.create(screenName, baseName, className, interactive, wantobjects, use
Tk, sync, use)
_tkinter.TclError: couldn't connect to display ":0"

...Program finished with exit code 1
Press ENTER to exit console.
```



```
from tkinter import *
window=Tk()
btn=Button(window, text="This is Button widget", fg='blue')
btn.place(x=80, y=100)
lbl=Label(window, text="This is Label widget", fg='red', font=("Helvetica",
lbl.place(x=60, y=50)
txtfld=Entry(window, text="This is Entry Widget", bd=5)
txtfld.place(x=80, y=150)
window.title('Hello Python')
window.geometry("300x200+10+10")
window.mainloop()
```

```
Traceback (most recent call last):
  File "main.py", line 36, in <module>
    window=Tk()
  File "/usr/lib/python3.4/tkinter/__init__.py", line 1854, in __init__
    self.tk = _tkinter.create(screenName, baseName, className, interactive, wantobjects, us
eTk, sync, use)
_tkinter.TclError: couldn't connect to display ":1"

...Program finished with exit code 1
Press ENTER to exit console.
```



```
tkinter import *
MyWindow:
f   __init__(self, win):
      self.lbl1=Label(win, text='First number')
      self.lbl2=Label(win, text='Second number')
      self.lbl3=Label(win, text='Result')
      self.t1=Entry(bd=3)
      self.t2=Entry()
      self.t3=Entry()
      self.btn1 = Button(win, text='Add')
      self.btn2=Button(win, text='Subtract')
      self.lbl1.place(x=100, y=50)
      self.t1.place(x=200, y=50)
      self.lbl2.place(x=100, y=100)
      self.t2.place(x=200, y=100)
      self.b1=Button(win, text='Add', command=self.add)
      self.b2=Button(win, text='Subtract')
      self.b2.bind('<Button-1>', self.sub)
      self.b1.place(x=100, y=150)
      self.b2.place(x=200, y=150)
      self.lbl3.place(x=100, y=200)
      self.t3.place(x=200, y=200)
f   add(self):
      self.t3.delete(0, 'end')
      num1=int(self.t1.get())
      num2=int(self.t2.get())
      result=num1+num2
      self.t3.insert(END, str(result))
f   sub(self, event):
      self.t3.delete(0, 'end')
      num1=int(self.t1.get())
      num2=int(self.t2.get())
      result=num1-num2
      self.t3.insert(END, str(result))

=Tk()
MyWindow(window)
.title('Hello Python')
.geometry("400x300+10+10")
.mainloop()
```

```
Traceback (most recent call last):
  File "main.py", line 36, in <module>
    window=Tk()
  File "/usr/lib/python3.4/tkinter/__init__.py", line 1854, in __init__
    self.tk = _tkinter.create(screenName, baseName, className, interactive, wantobjects, us
eTk, sync, use)
_tkinter.TclError: couldn't connect to display ":1"

...Program finished with exit code 1
Press ENTER to exit console.
```

Report – Report can be typed or hand written for up to two pages.

GRAPHICAL USER INTERFACES WITH TKINTER

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:

Importing the module – tkinter
Create the main window (container)
Add any number of widgets to the main window
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
mainloop(): 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()

```
import tkinter

m = tkinter.Tk()
'''
widgets are added here
'''
m.mainloop()
```
tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows. There are mainly three geometry manager classes class.

pack() method:It organizes the widgets in blocks before placing in the parent widget.
grid() method:It organizes the widgets in grid (table-like structure) before placing in the parent widget.
place() method:It organizes the widgets by placing them on specific positions directed by the programmer.

INTERACTING WITH DATABASES

Given the variety of techniques available to produce protein-protein interaction data and the large number of studies that are published every day, an enormous effort is required to store this information in a way that is both accessible and intelligible to the user. Molecular interaction databases aim to fulfil this need by extracting information from scientific publications or, in some cases, by including protein-protein interaction predictions found using computational method.The storage of interactions in publicly available databases allows access to a large volume of interaction data and subsequent analysis of the interactome