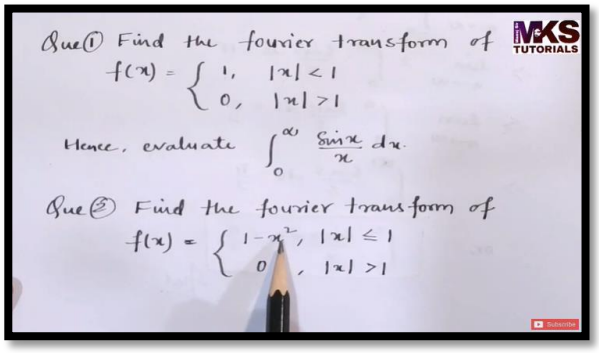
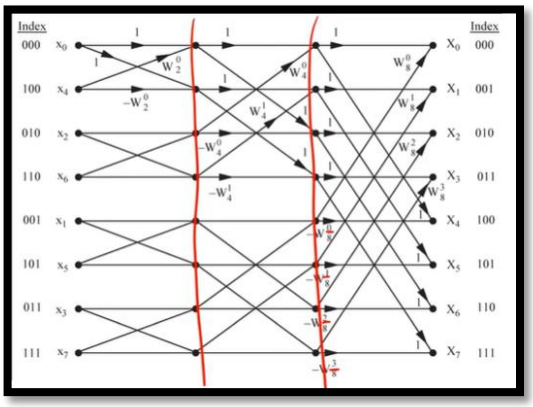
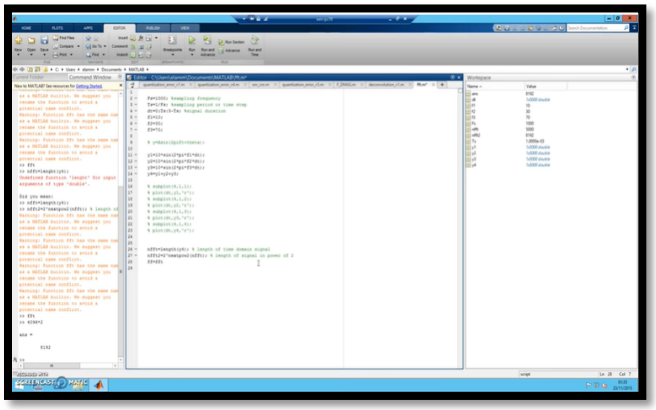
DAILY ASSESSMENT FORMAT

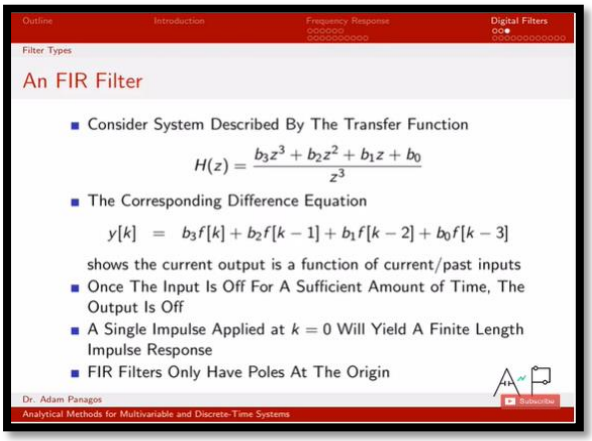
Date:	**27/05/2020**	**Name:**	**Davis S. Patel**
Course:	**Dsp**	**USN:**	**4AL16EC045**
Topic:	**Fourier Transforms FFT FFT Fast Fourier Transform Matlab FIR and IIR Filters Study and analysis FIR and IIR using FDA tool in MatLab Introduction to WT CWT & DWT Implementation of signal Filtering signal using WT in MatLAb Short-time Fourier Transform and the Spectogram Welch's method and windowing ECG Signal Analysis Using MATLAB**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

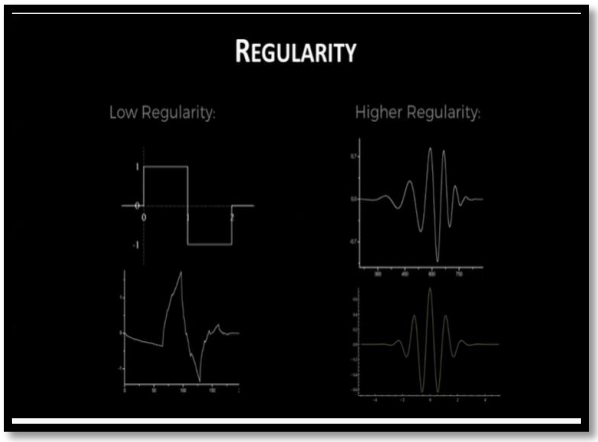
FORENOON SESSION DETAILS
Image of session

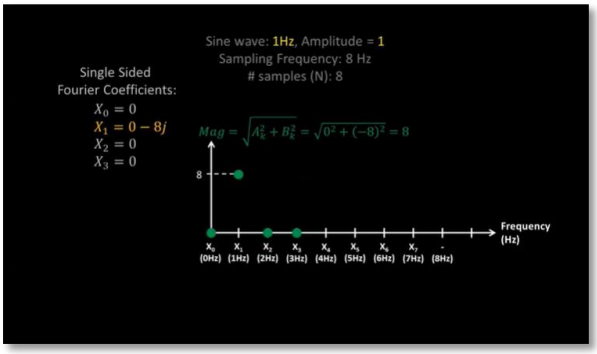


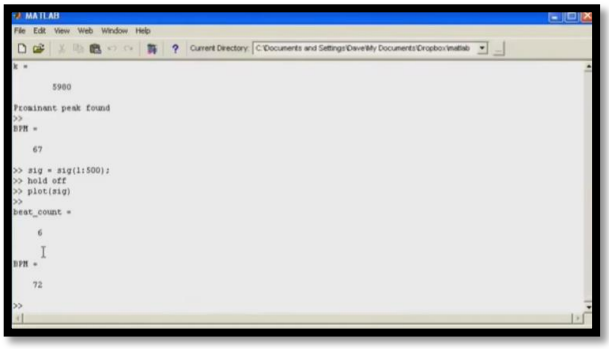

Index
000 x0
100 x4
010 x2
110 x6
001 x1
101 x5
011 x3
111 x7
Index
X0 000
X1 001
X2 010
X3 011
X4 100
X5 101
X6 110
X7 111




Outline
Introduction
Frequency Response
Digital Filters
Filter Types
An FIR Filter
Consider System Described By The Transfer Function
$H(z) = \frac{b_3 z^3 + b_2 z^2 + b_1 z + b_0}{z^3}$
The Corresponding Difference Equation
$y[k] = b_3 f[k] + b_2 f[k-1] + b_1 f[k-2] + b_0 f[k-3]$
shows the current output is a function of current/past inputs
Once The Input Is Off For A Sufficient Amount of Time, The Output Is Off
A Single Impulse Applied at $k = 0$ Will Yield A Finite Length Impulse Response
FIR Filters Only Have Poles At The Origin
Dr. Adam Panagos
Analytical Methods for Multivariable and Discrete-Time Systems


REGULARITY
Low Regularity:
Higher Regularity:


Single Sided
Fourier Coefficients:
$X_0 = 0$
$X_1 = 0 - 8j$
$X_2 = 0$
$X_3 = 0$
Sine wave: 1Hz, Amplitude = 1
Sampling Frequency: 8 Hz
samples (N): 8
$Mag = \sqrt{A_k^2 + B_k^2} = \sqrt{0^2 + (-8)^2} = 8$
8
Frequency (Hz)
X0 (0Hz)
X1 (1Hz)
X2 (2Hz)
X3 (3Hz)
X4 (4Hz)
X5 (5Hz)
X6 (6Hz)
X7 (7Hz)
- (8Hz)


MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\Daveilly Documents\Dropbox\matlab
N =
5900
Prominant peak found
>>
BPM =
67
>> sig = sig(1:500);
>> hold off
>> plot(sig)
>>
beat_count =
6
BPM =
72
>>

Report –

Fast Fourier Transform

The fast Fourier transform (FFT) is a discrete Fourier transform algorithm which reduces the number of computations needed for N points from 2N^2 to 2NlgN, where lg is the base-2 logarithm. Fast Fourier transform algorithms generally fall into two classes: decimation in time, and decimation in frequency. The Cooley-Tukey FFT algorithm first rearranges the input elements in bit-reversed order, then builds the output transform (decimation in time). The basic idea is to break up a transform of length N into two transforms of length N/2 using the identity –

$$\sum_{n=0}^{N-1} a_n e^{-2\pi i n k/N} = \sum_{n=0}^{N/2-1} a_{2n} e^{-2\pi i (2n) k/N} + \sum_{n=0}^{N/2-1} a_{2n+1} e^{-2\pi i (2n+1) k/N}$$

$$= \sum_{n=0}^{N/2-1} a_n^{\text{even}} e^{-2\pi i n k/(N/2)} + e^{-2\pi i k/N} \sum_{n=0}^{N/2-1} a_n^{\text{odd}} e^{-2\pi i n k/(N/2)},$$

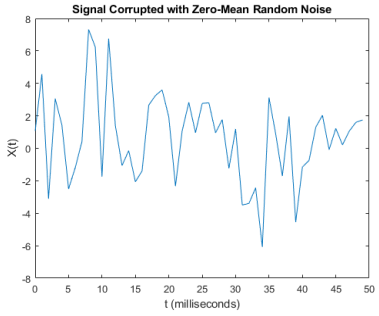
FFT Fast Fourier Transform Matlab

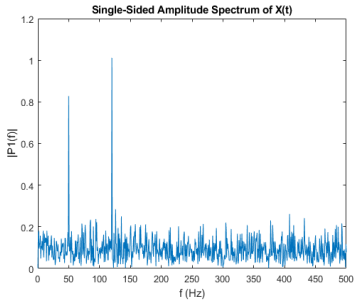
Fourier transforms to find the frequency components of a signal buried in noise.

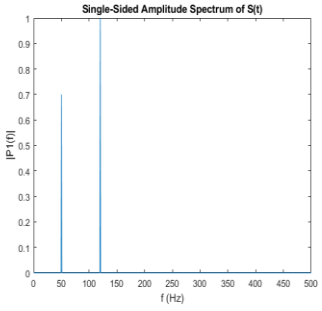
```
Fs = 1000;            % Sampling frequency
T = 1/Fs;             % Sampling period
L = 1500;             % Length of signal
t = (0:L-1)*T;        % Time vector
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
X = S + 2*randn(size(t));
plot(1000*t(1:50),X(1:50))
title('Signal Corrupted with Zero-Mean Random 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)|')
```

Output -







FIR & IIR Filtering

Infinite impulse response (IIR) filters

The infinite impulse response (IIR) filter is a recursive filter in that the output from the filter is computed by using the current and previous inputs and previous outputs.

Because the filter uses previous values of the output, there is feedback of the output in the filter structure. The design of the IIR filter is based on identifying the pulse transfer function G (z) that satisfies the requirements of the filter specification. This can be undertaken either by developing an analogue prototype and then transforming it to the pulse transfer function, or by designing directly in digital.

Finite impulse response (FIR) filters

FIR filters are one of two primary types of digital filters used in Digital Signal Processing (DSP) applications, the other type being IIR. "FIR" means "Finite Impulse Response." If you put in an impulse, that is, a single "1" sample followed by many "0" samples, zeroes will come out after the "1" sample has made its way through the delay line of the filter.

They can easily be designed to be "linear phase" (and usually are). Put simply, linear-phase filters delay the input signal but don't distort its phase. They are simple to implement. On most DSP microprocessors, the FIR calculation can be done by looping a single instruction.

They are suited to multi-rate applications. By multi-rate, we mean either "decimation" (reducing the sampling rate), "interpolation" (increasing the sampling rate), or both. Whether decimating or interpolating, the use of FIR filters allows some of the calculations to be omitted, thus providing an important computational efficiency. In contrast, if IIR filters are used, each output must be individually calculated, even if it that output will discarded (so the feedback will be incorporated into the filter).

Wavelet Transform

The wavelet transform is similar to the Fourier transform (or much more to the windowed Fourier transform) with a completely different merit function. The main difference is this: Fourier transform decomposes the signal into sines and cosines, i.e. the functions localized in Fourier space; in contrary the wavelet transform uses functions that are localized in both the real and Fourier space. Generally, the wavelet transform can be expressed by the following equation:

$$F(a,b) = \int_{-\infty}^{\infty} f(x)\, \psi^*_{(a,b)}(x)\, \mathrm{d}x$$

where the * is the complex conjugate symbol and function ψ is some function.

Discrete Wavelet Transform

The discrete wavelet transform (DWT) is an implementation of the wavelet transform using a discrete set of the wavelet scales and translations obeying some defined rules. In other words, this transform decomposes the signal into mutually orthogonal set of wavelets, which is the main difference from the continuous wavelet transform (CWT), or its implementation for the discrete time series sometimes called discrete-time continuous wavelet transform (DT-CWT).The wavelet can be constructed from a scaling function which describes its scaling properties.

Continuous Wavelet Transform

Continuous wavelet transform (CWT) is an implementation of the wavelet transform using arbitrary scales and almost arbitrary wavelets. The wavelets used are not orthogonal and the data obtained by this transform are highly correlated. For the discrete time series we can use this transform as well, with the limitation that the smallest wavelet translations must be equal to the data sampling. This is sometimes called Discrete Time Continuous Wavelet Transform (DT-CWT) and it is the most used way of computing CWT in real applications.

In principle the continuous wavelet transform works by using directly the definition of the wavelet transform, i.e. we are computing a convolution of the signal with the scaled wavelet.

For each scale we obtain by this way an array of the same length N as the signal has. By using M arbitrarily chosen scales we obtain a field $N \times M$ that represents the time-frequency plane directly. The algorithm used for this computation can be based on a direct convolution or on a convolution by means of multiplication in Fourier space, this is sometimes called Fast Wavelet Transform.

Short-time Fourier transform (STFT)

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.

Welch's method

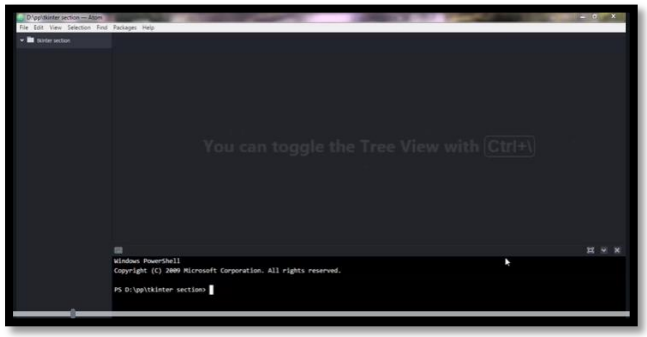
Welch's method, named after Peter D. Welch, is an approach for spectral density estimation. It is for estimating the power of a signal at different frequencies. The method is based on the concept of using periodogram spectrum estimates, which are the result of converting a signal from the time domain 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.

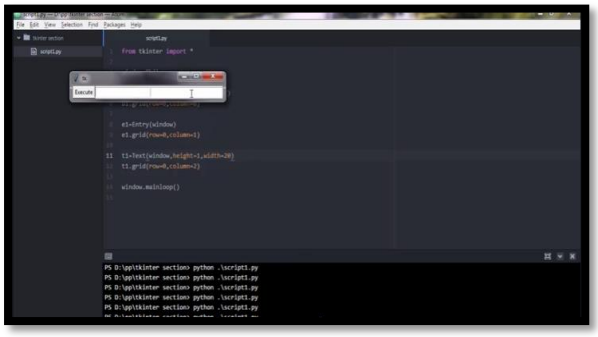
DAILY ASSESSMENT FORMAT

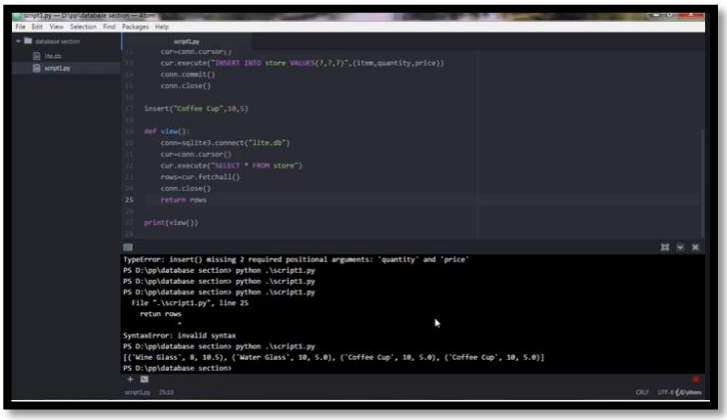
Date:	**27/05/2020**	**Name:**	**Davis S. Patel**
Course:	**Python Course**	**USN:**	**4AL16EC045**
Topic:	**Graphical User Interfaces with Tkinter** **Interacting with Databases**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

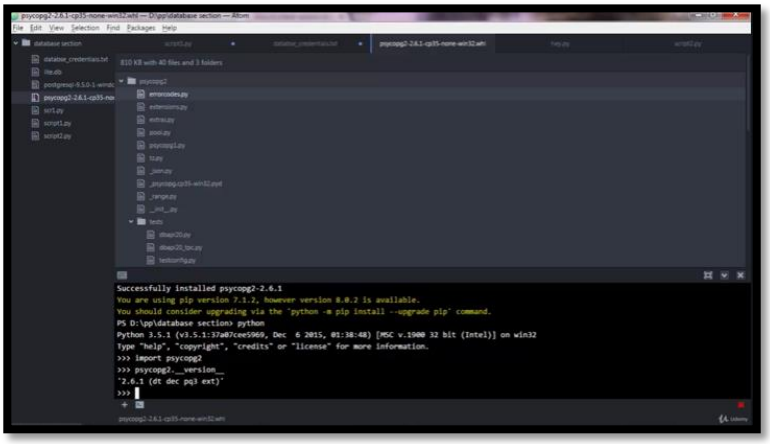
AFTERNOON SESSION DETAILS

Image of Session









Report –

Graphical User Interfaces with Tkinter

The Tkinter module ("Tk interface") is the standard Python interface to the Tk GUI toolkit from Scriptics (formerly developed by Sun Labs).Both Tk and Tkinter are available on most UNIX platforms, as well as on Windows and Macintosh systems. Starting with the 8.0 release, Tk offers native look and feel on all platforms.

Tkinter consists of a number of modules. The Tk interface is provided by a binary extension module named _tkinter. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

The public interface is provided through a number of Python modules. The most important interface module is the Tkinter module itself. To use Tkinter, all you need to do is to import the Tkinter module:

```
import Tkinter
```

Or, more often:

```
from Tkinter import *
```

The Tkinter module only exports widget classes and associated constants, so you can safely use the from-in form in most cases. If you prefer not to, but still want to save some typing, you can use import-as:

```
Import Tkinter as Tk
```

Create a Multi-widget GUI

Create a Python program that expects a kilogram input value and converts that value to grams, pounds, and ounces when the user pushes the *Convert* button.

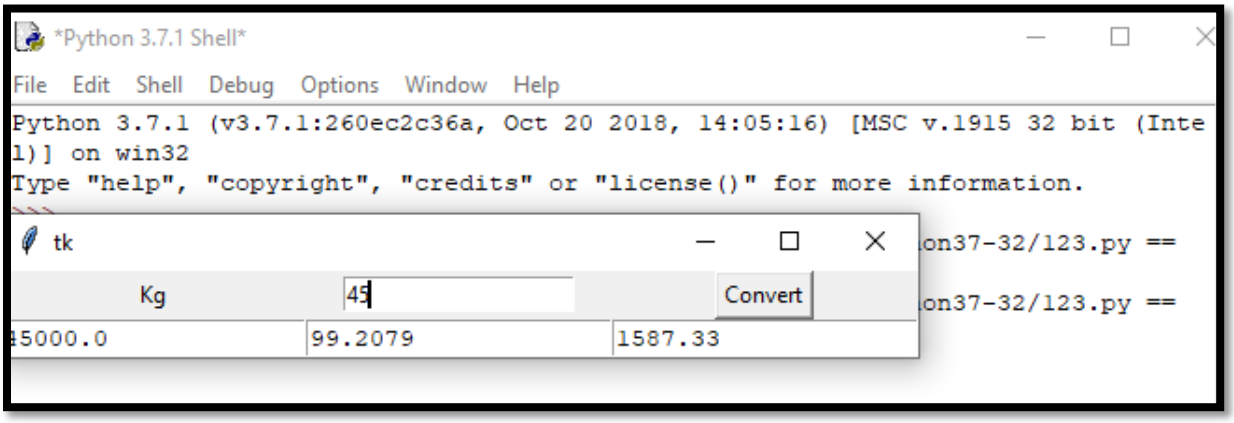
Note - The following code output will in form of conversion tab which will take the value in kg's and covert those values in grams, pounds & ounces respectively.

Code –

```
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()
```

Output –



Interacting with Databases

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

MySQL dB is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and is built on top of the MySQL C API.

A database is an organized collection of data. The data are typically organized to model aspects of reality in a way that supports processes requiring this information. The term "database" can both refer to the data themselves or to the database management system.

The Database management system is a software application for the interaction between users database itself. Users don't have to be human users. They can be other programs and applications as well. The Python standard for database interfaces is the Python DB-API, which is used by Python's database interfaces. The DB-API has been defined as a common interface, which can be used to access relational databases. In other words, the code in Python for communicating with a database should be the same, regardless of the database and the database module used.

SQLite is a simple relational database system, which saves its data in regular data files or even in the internal memory of the computer, i.e. the RAM. It was developed for embedded applications, like Mozilla-Firefox (Bookmarks), Symbian OS or Android. SQLITE is "quite" fast, even though it uses a simple file. It can be used for large databases as well. If you want to use SQLite, you have to import the module sqlite3. To use a database, you have to create first a Connection object. The connection object will represent the database. The argument of connection - in the following example "companys.db" - functions both as the name of the file, where the data will be stored, and as the name of the database. If a file with this name exists, it will be opened. It has to be a SQLite database file.

If you work under a Python 2.x version, the module MySQLdb can be used. It has to be installed.

PostgreSQL database

PostgreSQL is the recommended relational database for working with Python web applications. PostgreSQL's feature set, active development and stability contribute to its usage as the backend for millions of applications live on the Web today.

Connecting to PostgreSQL with Python

To work with relational databases in Python you need to use a database driver, which is also referred to as a database connector. The most common driver library for working with PostgreSQL is psycopg2. There is a list of all drivers on the PostgreSQL wiki, including several libraries that are no longer maintained. If you're working with the asyncio Python stdlib module you should also take a look at the aiopg library which wraps psycopg2's asynchronous features together.

To abstract the connection between tables and objects, many Python developers use an object-relational mapper (ORM) with to turn relational data from PostgreSQL into objects that can be used in their Python application.