Technical Documentation

Date - 20 APRIL 2015

IITG Hospital System

Version - 1.1

Group - 5

Prepared for

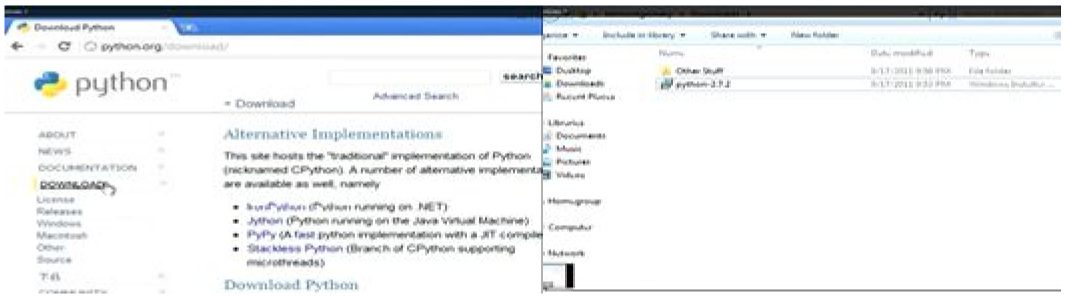
CS 243—Software Engineering

Instructor: Dr. Pradip.K. Das

Spring 2015

Python installation procedure:

At first download python from its official website www.python .org/download/.(we are using python 2.7) ,now open the installer and run it.



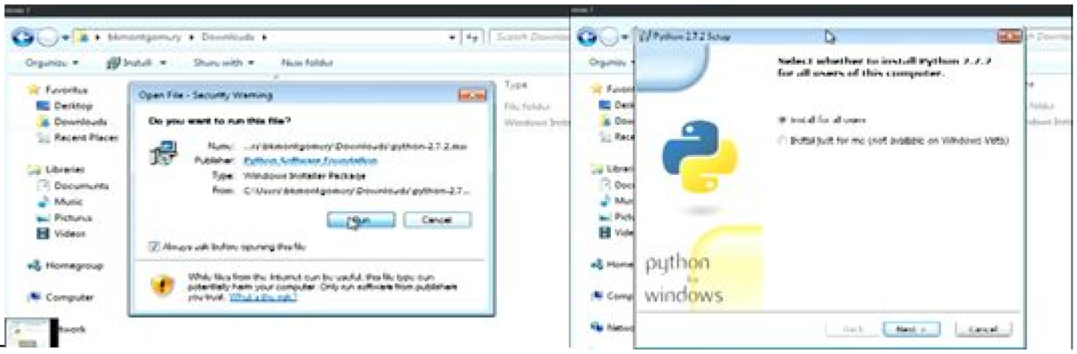
Alternative Implementations

This site hosts the "traditional" implementation of Python (nicknamed CPython). A number of alternative implementations are available as well, namely

- IronPython (Python running on .NET)
- Jython (Python running on the Java Virtual Machine)
- PyPy (A fast python implementation with a JIT compiler)
- Stackless Python (Branch of CPython supporting microthreads)

Download Python

Then click on run button, after this check 'INSTALL FOR ALL USER 'option and click on next.



Do you want to run this file?

Name: ...rs\bkmontgomery\Downloads\python-2.7.2.msi
Publisher: Python Software Foundation
Type: Windows Installer Package
From: C:\Users\bkmontgomery\Downloads\python-2.7...

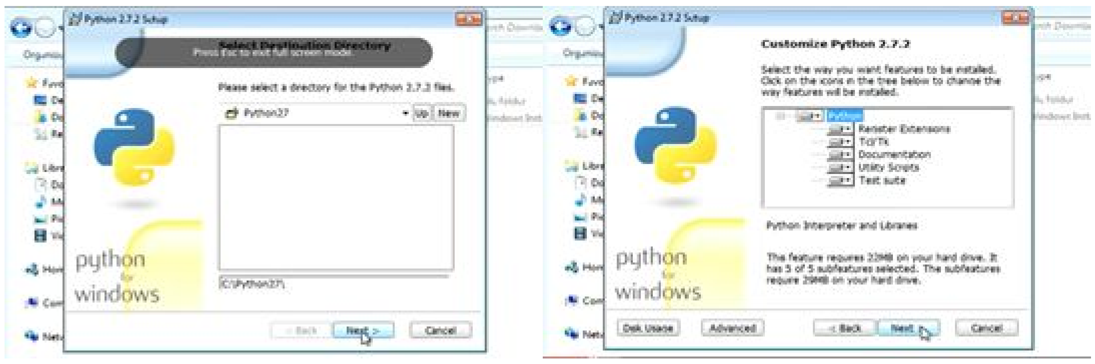
Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Select whether to install Python 2.7.2 for all users of this computer.

- Install for all users
- Install just for me (not available on Windows Vista)

Select a directory for python files and click on 'next' Then again click on 'next'



Please select a directory for the Python 2.7.2 files.

Python27

C:\Python27\

Customize Python 2.7.2

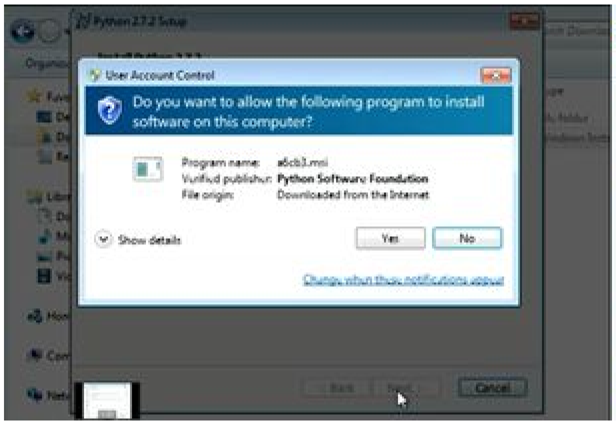
Select the way you want features to be installed. Click on the icons in the tree below to change the way features will be installed.

Python
- Register Extensions
- Tcl/Tk
- Documentation
- Utility Scripts
- Test suite

Python Interpreter and Libraries

This feature requires 22MB on your hard drive. It has 5 of 5 subfeatures selected. The subfeatures require 29MB on your hard drive.

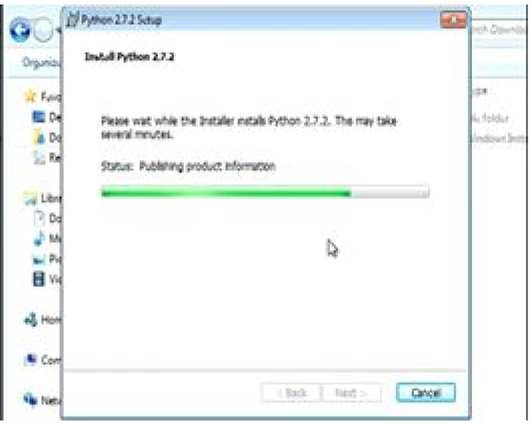
Give permission to user account control


Python 2.7.2 Setup

User Account Control

Do you want to allow the following program to install software on this computer?

Program name: a6cb3.msi
Verified publisher: Python Software Foundation
File origin: Downloaded from the Internet

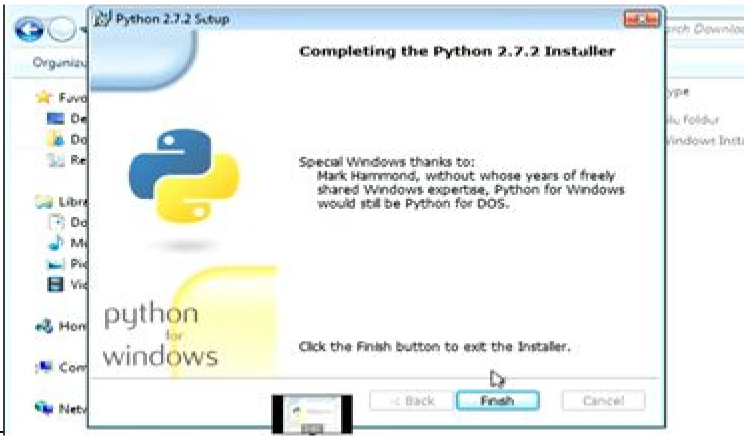

Python 2.7.2 Setup

Install Python 2.7.2

Please wait while the Installer installs Python 2.7.2. This may take several minutes.

Status: Publishing product information

As the installation completed click on 'FINISH' button.


Python 2.7.2 Setup

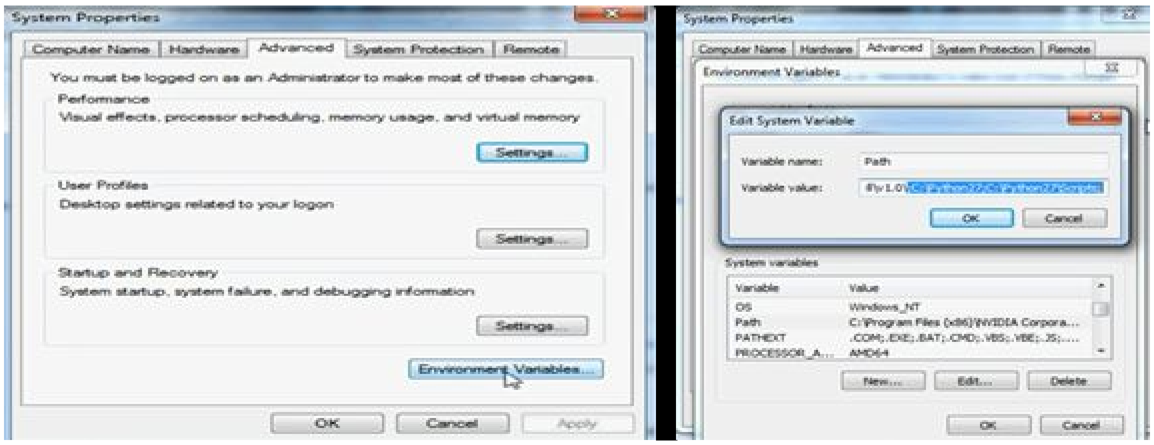
Completing the Python 2.7.2 Installer

Special Windows thanks to:
Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

Click the Finish button to exit the Installer.

Now you can open python shell. There may be some time where you will want to use python from command prompt. For this you have to set path.

To set path go to control panel >> system and security >> system >> advance system settings. Then a window named 'system properties' will open. Click on 'environment variables'. After that edit path. For this go to the very end of the path variable and type ';c:/python27/'.then press ok and save that. Now its done.



System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

Environment Variables

Edit System Variable

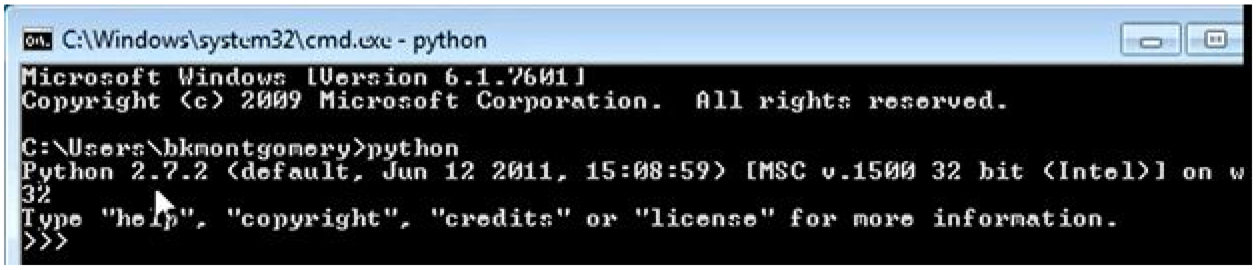
Variable name: Path

Variable value: ifly 1.0\C:\Python27;C:\Python27\Scripts

System variables

Variable	Value
OS	Windows_NT
Path	C:\Program Files (x86)\NVIDIA Corpora...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

Now open the command prompt and and type ‘python’ and after then type ‘import python’. now you can use it in command prompt.

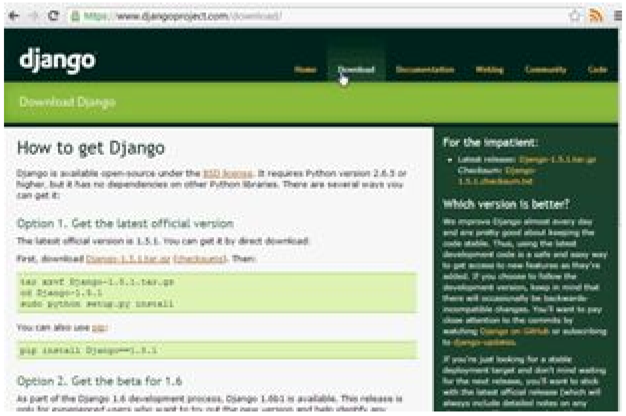


C:\Windows\system32\cmd.exe - python

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\bkmontgomery>python
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on w
32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

DJANGO INSTALLATION PROCEDURE:-

First open www.djangoproject.com , go to download and download it.



https://www.djangoproject.com/download/

How to get Django

Django is available open-source under the BSD license. It requires Python version 2.6.5 or higher, but it has no dependencies on other Python libraries. There are several ways you can get it:

Option 1. Get the latest official version

The latest official version is 1.5.1. You can get it by direct download:

First, download Django-1.5.1.tar.gz (checksums). Then:

```
tar xzvf Django-1.5.1.tar.gz
cd Django-1.5.1
sudo python setup.py install
```

You can also use pip:

```
pip install Django==1.5.1
```

Option 2. Get the beta for 1.6

As part of the Django 1.6 development process, Django 1.6b1 is available. This release is only for experienced users who want to try out the new version and help identify any

For the impatient:

- Latest release: Django-1.5.1.tar.gz Checksum: Django-1.5.1.checksum.txt

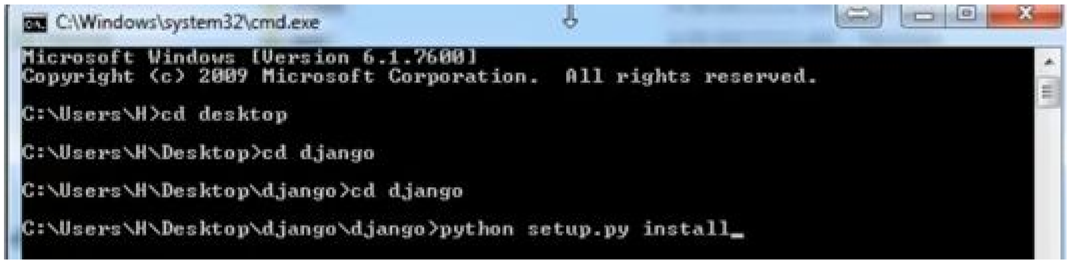
Which version is better?

We improve Django almost every day and are pretty good about keeping the code stable. Thus, using the latest development code is a safe and easy way to get access to new features as they're added. If you choose to follow the development version, keep in mind that there will occasionally be backwards-incompatible changes. You'll want to pay close attention to the commits by watching Django on GitHub or subscribing to django-updates.

If you're just looking for a stable deployment target and don't mind waiting for the next release, you'll want to stick with the latest official release (which will always include detailed notes on any

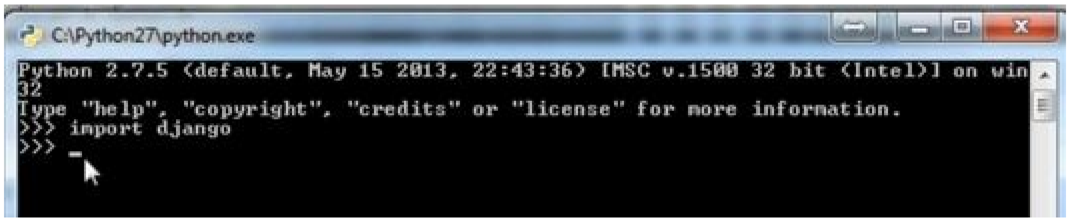
Extract the zipped folder. For convenience just drag the extracted folder to desktop and rename it as ’django’. Also rename the inside folder as ‘django’ also. This folder contains a set up file named ‘setup.py’.

First of all open command prompt and change directory to 'desktop' (since our extracted folder is on desktop). again change directory to 'django'. Again change directory to inner folder (which is 'django' also, for our convenience)
And now type 'python setup.py install' to install django.



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\H>cd desktop

C:\Users\H\Desktop>cd django

C:\Users\H\Desktop\django>cd django

C:\Users\H\Desktop\django\django>python setup.py install_
```

As the installation completed just type 'import django' in command prompt.



```
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> _
```

After this if command prompt doesn't show any error then its done.

Statistical Analysis Requirements:

· Libraries used in this module are nltk for python, highcharts.js for JavaScript . Nltk was used to tokenize sentences in description of prescription. High charts was used to display interactive charts using java script.

Installation procedure for nltk (3.0.2):

1. Download nltk from https://pypi.python.org/pypi/nltk.
2. Installation will start after executing the downloaded file(.exe file).
3. For Mac/Unix users:
   - Install pip: run: `sudo easy_install pip`
   - Install Numpy (optional) : `sudo pip install -U numpy`
   - Install NLTK :run `sudo pip install -U nltk`
4. After installation, now we need to download nltk data.

5. Run the Python interpreter and type the commands:

```
>>> import nltk
>>> nltk.download()
```

6. If your web connection uses a proxy server, you should specify the proxy address as follows. Type the following commands in Python interpreter:

```
>>> nltk.set_proxy('http://proxy.example.com:3128', 'USERNAME', 'PASSWORD')
>>> nltk.download()
```

7. For any other queries please visit http://www.nltk.org/data.html

- **Installation for highcharts.js :**
    1. Highcharts requires two files to run, **highcharts.js** and jQuery.
    2. All these files are stored in static folder(../templates/system/static) .

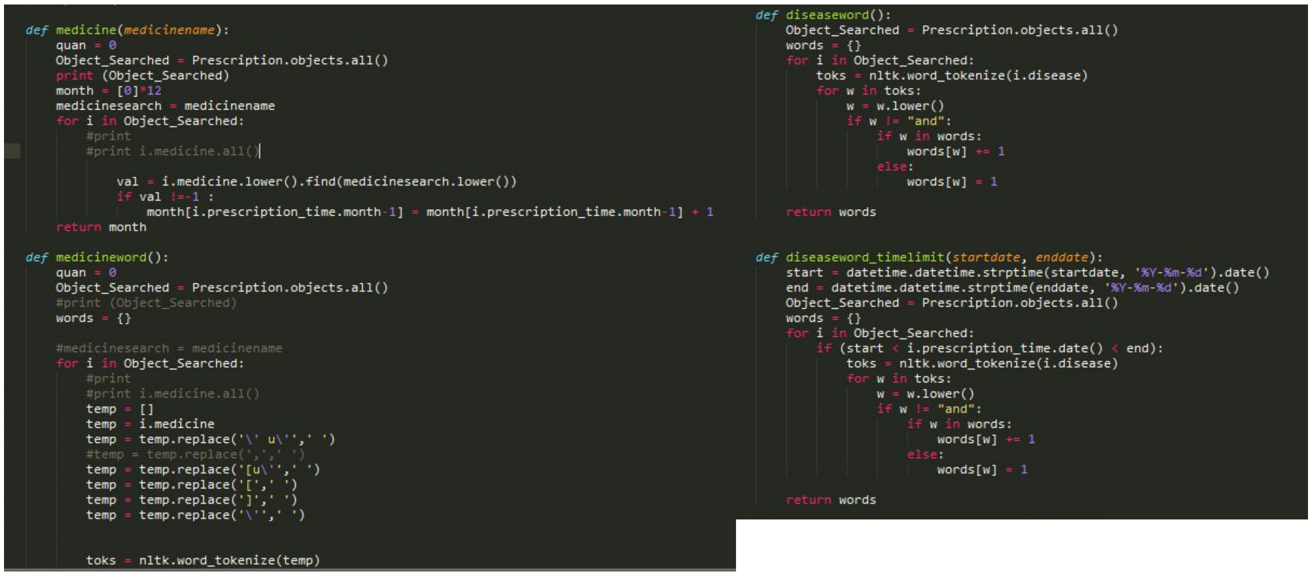
All the graphs were made using highcharts.js . Their input were taken from corresponding tables in database.
Word Cloud is generated using wordcloud.js . All the statements are first tokenized using NLTK and then word cloud is generated.

WORKING OF STATISTICAL ANALYSIS MODULE :-

We have created a system named app inside our software which deals with the analysis and generation of graph part. The python file named analysis.py contains the calculation and fetching data from the database while the file views.py contains function which takes the output of analysis.py and renders it to the html pages.

From the front end whenever admin logins, he/she gets an option of statistical analysis as soon as you click there you get the grahindex.html page which takes all the input. we have distributed the form into three section as “Generate Graph for diseases”, “Generate Graph for Medicine”, “Generate Graph for follow-up”. Each module has option of filling in the dates. If the user fills the date then, he sees the graph for that range of dates only and if he leaves it blank then he sees the complete graph of that particular disease. Apart from this we generate Top5 disease graph and Word Cloud. For generation of graph we first take the prescription database and then we take its disease column or medicine column depending on our need. After that if it is a disease column then we search the particular disease name in it and as soon as the first name is found the value is the index of that disease name else it is -1. Whenever the value is not -1, the month array is increased depending upon the date of prescription.

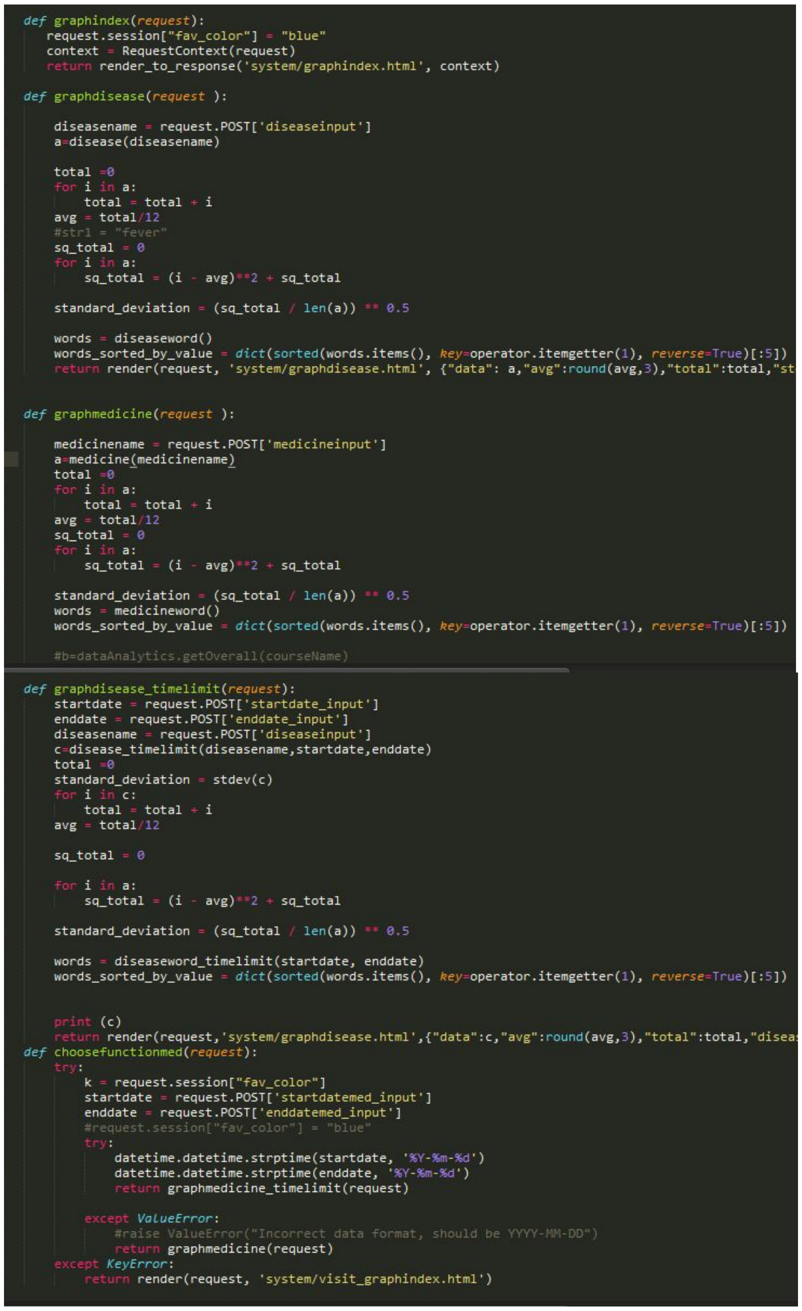
For the word cloud we are using NLTK. and its site-package ’punkt’. We take the input json string of the medicine and remove the redundant things which are present in json string apart from the medicine name. Then we create an dictionary words{} which increases the medicine name quantity by 1 as soon as it finds it . After that this words dictionary is passed to the front-end along with its frequency then using jQCloud, we generate the word cloud in the front end. The three different analysis is displayed in the three different section i.e. graphmedicine.html, graphdisease.html and graphfollowup.html.



```python
def medicine(medicinename):
    quan = 0
    Object_Searched = Prescription.objects.all()
    print (Object_Searched)
    month = [0]*12
    medicinesearch = medicinename
    for i in Object_Searched:
        #print
        #print i.medicine.all()

            val = i.medicine.lower().find(medicinesearch.lower())
            if val !=-1 :
                month[i.prescription_time.month-1] = month[i.prescription_time.month-1] + 1
    return month

def medicineword():
    quan = 0
    Object_Searched = Prescription.objects.all()
    #print (Object_Searched)
    words = {}

    #medicinesearch = medicinename
    for i in Object_Searched:
        #print
        #print i.medicine.all()
        temp = []
        temp = i.medicine
        temp = temp.replace('\' u\'',' ')
        #temp = temp.replace(',',' ')
        temp = temp.replace('[u\'',' ')
        temp = temp.replace('[',' ')
        temp = temp.replace(']',' ')
        temp = temp.replace('\'',' ')


        toks = nltk.word_tokenize(temp)
```

```python
def diseaseword():
    Object_Searched = Prescription.objects.all()
    words = {}
    for i in Object_Searched:
        toks = nltk.word_tokenize(i.disease)
        for w in toks:
            w = w.lower()
            if w != "and":
                if w in words:
                    words[w] += 1
                else:
                    words[w] = 1

    return words


def diseaseword_timelimit(startdate, enddate):
    start = datetime.datetime.strptime(startdate, '%Y-%m-%d').date()
    end = datetime.datetime.strptime(enddate, '%Y-%m-%d').date()
    Object_Searched = Prescription.objects.all()
    words = {}
    for i in Object_Searched:
        if (start < i.prescription_time.date() < end):
            toks = nltk.word_tokenize(i.disease)
            for w in toks:
                w = w.lower()
                if w != "and":
                    if w in words:
                        words[w] += 1
                    else:
                        words[w] = 1

    return words
```

Analysis.py (code snippet)



```python
def graphindex(request):
   request.session["fav_color"] = "blue"
   context = RequestContext(request)
   return render_to_response('system/graphindex.html', context)

def graphdisease(request ):

    diseasename = request.POST['diseaseinput']
    a=disease(diseasename)

    total =0
    for i in a:
        total = total + i
    avg = total/12
    #str1 = "fever"
    sq_total = 0
    for i in a:
        sq_total = (i - avg)**2 + sq_total

    standard_deviation = (sq_total / len(a)) ** 0.5

    words = diseaseword()
    words_sorted_by_value = dict(sorted(words.items(), key=operator.itemgetter(1), reverse=True)[:5])
    return render(request, 'system/graphdisease.html', {"data": a,"avg":round(avg,3),"total":total,"st


def graphmedicine(request ):

    medicinename = request.POST['medicineinput']
    a=medicine(medicinename)
    total =0
    for i in a:
        total = total + i
    avg = total/12
    sq_total = 0
    for i in a:
        sq_total = (i - avg)**2 + sq_total

    standard_deviation = (sq_total / len(a)) ** 0.5
    words = medicineword()
    words_sorted_by_value = dict(sorted(words.items(), key=operator.itemgetter(1), reverse=True)[:5])

    #b=dataAnalytics.getOverall(courseName)
```

```python
def graphdisease_timelimit(request):
    startdate = request.POST['startdate_input']
    enddate = request.POST['enddate_input']
    diseasename = request.POST['diseaseinput']
    c=disease_timelimit(diseasename,startdate,enddate)
    total =0
    standard_deviation = stdev(c)
    for i in c:
        total = total + i
    avg = total/12

    sq_total = 0

    for i in a:
        sq_total = (i - avg)**2 + sq_total

    standard_deviation = (sq_total / len(a)) ** 0.5

    words = diseaseword_timelimit(startdate, enddate)
    words_sorted_by_value = dict(sorted(words.items(), key=operator.itemgetter(1), reverse=True)[:5])


    print (c)
    return render(request,'system/graphdisease.html',{"data":c,"avg":round(avg,3),"total":total,"diseas
def choosefunctionmed(request):
    try:
        k = request.session["fav_color"]
        startdate = request.POST['startdatemed_input']
        enddate = request.POST['enddatemed_input']
        #request.session["fav_color"] = "blue"
        try:
            datetime.datetime.strptime(startdate, '%Y-%m-%d')
            datetime.datetime.strptime(enddate, '%Y-%m-%d')
            return graphmedicine_timelimit(request)

        except ValueError:
            #raise ValueError("Incorrect data format, should be YYYY-MM-DD")
            return graphmedicine(request)
    except KeyError:
        return render(request, 'system/visit_graphindex.html')
```

Views.py (code snippet)

FAQ-

In each Login page a link to the FAQ page is given.
As the user clicks on it url is set to **faq** which calls the view **load_faq.**
The view load_faq redirects to the page **faq.html**
In the **homepage** of each of the login the **target** for the FAQ page is set to **"_blank"** which opens the faq.html in a new tab.

MEDICAL FORMS

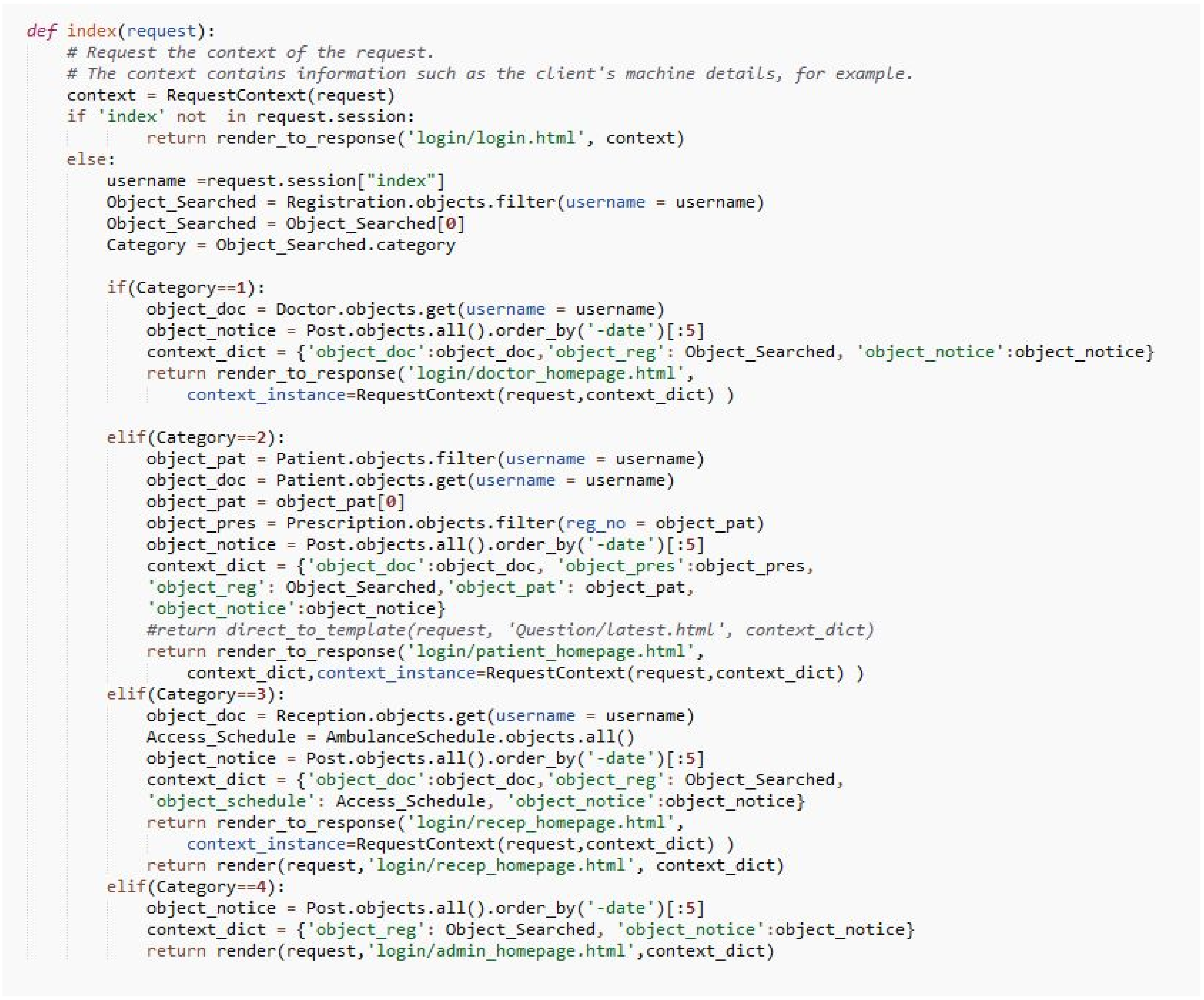
In each Login page a link to the Medical Forms page is given.
As the user clicks on it url is set to **med_forms** which calls the **view med_forms.**
The view **med_forms** redirects to the page **med_forms.html**
In the **homepage** of each of the login the **target** for the **Medical Forms** page is set to **"_blank"** which opens the **med_forms.html** in a new tab.

NOTICE BOARD

From each of the view that redirects to the homepage of different types of users, an object **object_notice** of the last 5 notices from the model Post is passed.
In the html page **DTL** is used to fetch the last 5 notices to the html page by setting the url to **noticeboard/{{obj.id}}.**
The url noticeboard in the login includes the **blog.urls**
So according to the object id the corresponding entry in the Notice Board is fetched and displayed to the Notice Board division in the homepage

LOGIN AND AUTHENTICATION PROCESS

The "index" view in the login app handles the homepage of the website.
If any session exists, appropriate login portal is fetched, otherwise the homepage is fetched.



```python
def index(request):
    # Request the context of the request.
    # The context contains information such as the client's machine details, for example.
    context = RequestContext(request)
    if 'index' not  in request.session:
            return render_to_response('login/login.html', context)
    else:
        username =request.session["index"]
        Object_Searched = Registration.objects.filter(username = username)
        Object_Searched = Object_Searched[0]
        Category = Object_Searched.category

        if(Category==1):
            object_doc = Doctor.objects.get(username = username)
            object_notice = Post.objects.all().order_by('-date')[:5]
            context_dict = {'object_doc':object_doc,'object_reg': Object_Searched, 'object_notice':object_notice}
            return render_to_response('login/doctor_homepage.html',
                context_instance=RequestContext(request,context_dict) )

        elif(Category==2):
            object_pat = Patient.objects.filter(username = username)
            object_doc = Patient.objects.get(username = username)
            object_pat = object_pat[0]
            object_pres = Prescription.objects.filter(reg_no = object_pat)
            object_notice = Post.objects.all().order_by('-date')[:5]
            context_dict = {'object_doc':object_doc, 'object_pres':object_pres,
            'object_reg': Object_Searched,'object_pat': object_pat,
            'object_notice':object_notice}
            #return direct_to_template(request, 'Question/latest.html', context_dict)
            return render_to_response('login/patient_homepage.html',
                context_dict,context_instance=RequestContext(request,context_dict) )
        elif(Category==3):
            object_doc = Reception.objects.get(username = username)
            Access_Schedule = AmbulanceSchedule.objects.all()
            object_notice = Post.objects.all().order_by('-date')[:5]
            context_dict = {'object_doc':object_doc,'object_reg': Object_Searched,
            'object_schedule': Access_Schedule, 'object_notice':object_notice}
            return render_to_response('login/recep_homepage.html',
                context_instance=RequestContext(request,context_dict) )
            return render(request,'login/recep_homepage.html', context_dict)
        elif(Category==4):
            object_notice = Post.objects.all().order_by('-date')[:5]
            context_dict = {'object_reg': Object_Searched, 'object_notice':object_notice}
            return render(request,'login/admin_homepage.html',context_dict)
```

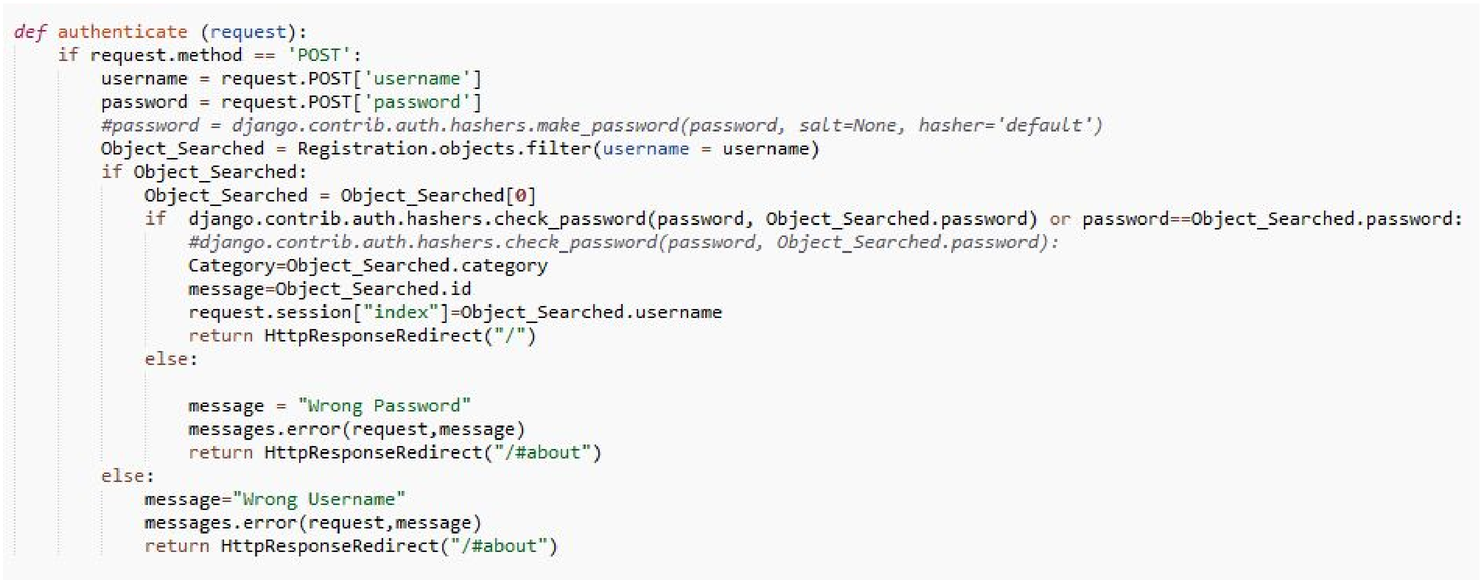
AUTHENTICATION

The login authentication is handled by the “authenticate” view of the login app in the project. It interacts with the Registration Model defined in the Database app.
There is a unique category assigned to different type of users including doctors, patients, admin and the receptionist.

```
if category==1:
        doctor login.
```

```
elif category==2:
        patient login.
elif category==3:
        reception login
else:
        admin login.
```

The authenticate view identifies the user on the basis of his category. The username is set as a request.session, corresponding objects are created and the required login template is fetched.



```python
def authenticate (request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        #password = django.contrib.auth.hashers.make_password(password, salt=None, hasher='default')
        Object_Searched = Registration.objects.filter(username = username)
        if Object_Searched:
            Object_Searched = Object_Searched[0]
            if  django.contrib.auth.hashers.check_password(password, Object_Searched.password) or password==Object_Searched.password:
                #django.contrib.auth.hashers.check_password(password, Object_Searched.password):
                Category=Object_Searched.category
                message=Object_Searched.id
                request.session["index"]=Object_Searched.username
                return HttpResponseRedirect("/")
            else:

                message = "Wrong Password"
                messages.error(request,message)
                return HttpResponseRedirect("/#about")
        else:
            message="Wrong Username"
            messages.error(request,message)
            return HttpResponseRedirect("/#about")
```

PATIENT LOGIN

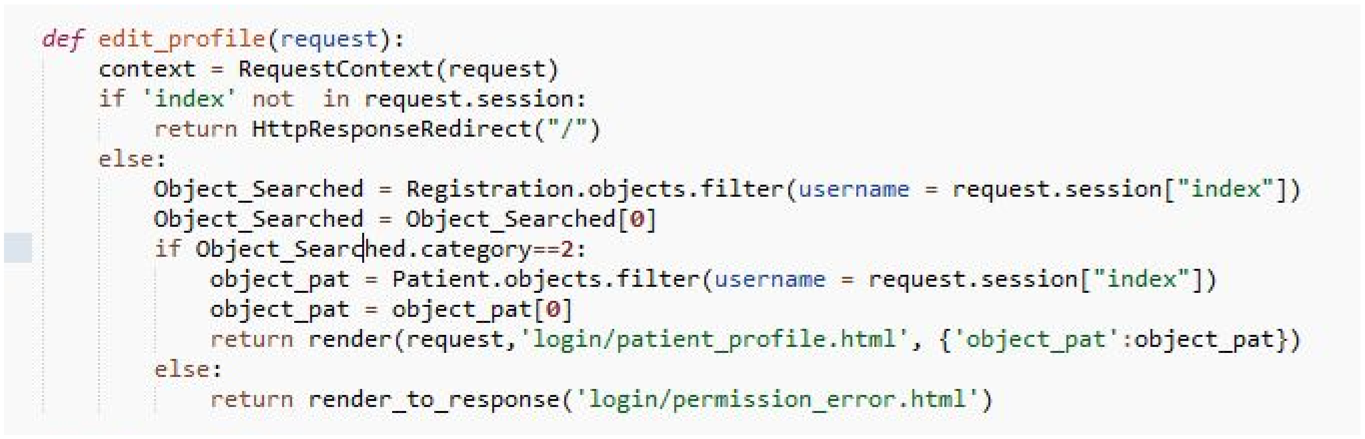
Models, Views and Templates for different functionality-

Homepage-

Model Used- “patient” to fetch the profile information of the patient.
Template fetched - “patient_homepage.html”
The blog app handles the notice board which is explained separately.

Edit Profile-

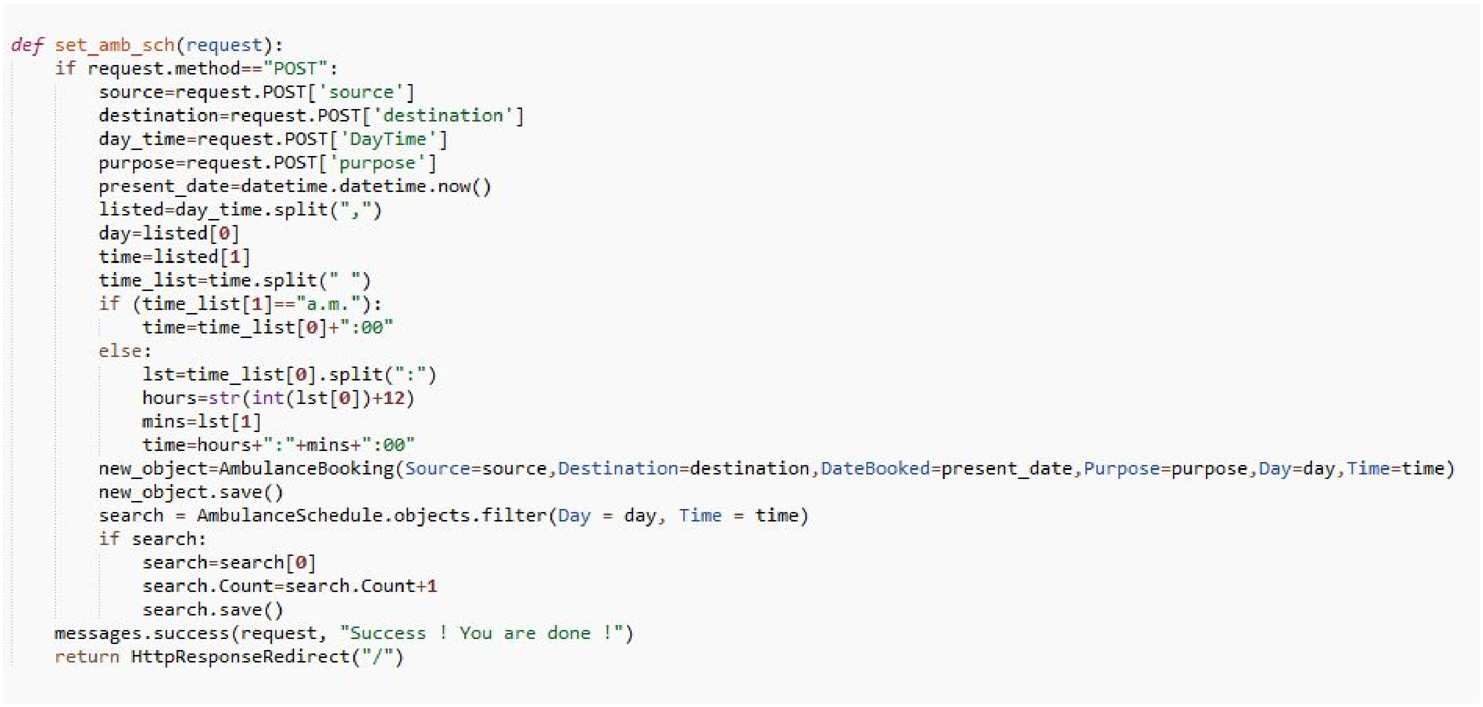
“edit_profile” view helps to render the “patient_profile.html” template which extends the “patient_homepage.html” template.
“pat_prof_sub” view handles the submission of the form for edit profile. If it is a success, it redirects to the homepage with a success message.



```
def edit_profile(request):
    context = RequestContext(request)
    if 'index' not  in request.session:
        return HttpResponseRedirect("/")
    else:
        Object_Searched = Registration.objects.filter(username = request.session["index"])
        Object_Searched = Object_Searched[0]
        if Object_Searched.category==2:
            object_pat = Patient.objects.filter(username = request.session["index"])
            object_pat = object_pat[0]
            return render(request,'login/patient_profile.html', {'object_pat':object_pat})
        else:
            return render_to_response('login/permission_error.html')
```

Appointment Form
Model Used- “CRequest”- All the appointments are stored in the “CRequest” model of database app.
“call_appoint” view calls the appointment form child template “appointment.html” which again extends the “patient_homepage.html” template.
“fix_appoint” view handles the submission of appointment form.

Ambulance booking
The patient books the available ambulances as per his requirement and all are stored in the AmbulanceBooking Model.
It takes the available ambulances from the Ambulance Schedule model for the convenience of the patient.
The views “book_amb” and “set_amb_sub” handle the calling and the submission of the Ambulance Booking form.



```python
def set_amb_sch(request):
    if request.method=="POST":
        source=request.POST['source']
        destination=request.POST['destination']
        day_time=request.POST['DayTime']
        purpose=request.POST['purpose']
        present_date=datetime.datetime.now()
        listed=day_time.split(",")
        day=listed[0]
        time=listed[1]
        time_list=time.split(" ")
        if (time_list[1]=="a.m."):
            time=time_list[0]+":00"
        else:
            lst=time_list[0].split(":")
            hours=str(int(lst[0])+12)
            mins=lst[1]
            time=hours+":"+mins+":00"
        new_object=AmbulanceBooking(Source=source,Destination=destination,DateBooked=present_date,Purpose=purpose,Day=day,Time=time)
        new_object.save()
        search = AmbulanceSchedule.objects.filter(Day = day, Time = time)
        if search:
            search=search[0]
            search.Count=search.Count+1
            search.save()
    messages.success(request, "Success ! You are done !")
    return HttpResponseRedirect("/")
```

FAQ and Medical Form- The views ”load_faq” and “med_forms” call the utilities respectively in a new tab.

RECEPTIONIST LOGIN

MODELS USED-
Registration, AmbulanceSchedule, AmbulanceBooking, Post, Doctor

VIEWS-
load_faq, med_forms, set_amb_sch, call_recep_schedule, end_recep_schedule, new_notice, notice_submit, call_addpatient, end_addpatient

DIFFERENT FEATURES AND DATA FLOWS-

AMBULANCE SCHEDULES

As the user clicks on this button url is set to **recep_schedule**.
This url calls the view **call_recep_schedule**
This view checks if the user is logged in or not.
In case the user is not logged in it redirects to the login page
If the user is logged in it checks if the **category is 3** then only redirects to the **recep_schedule.html** page, otherwise gives an access forbidden page **permission_error.html**

In the recep_schedule.html page two forms are provided to the receptionist-

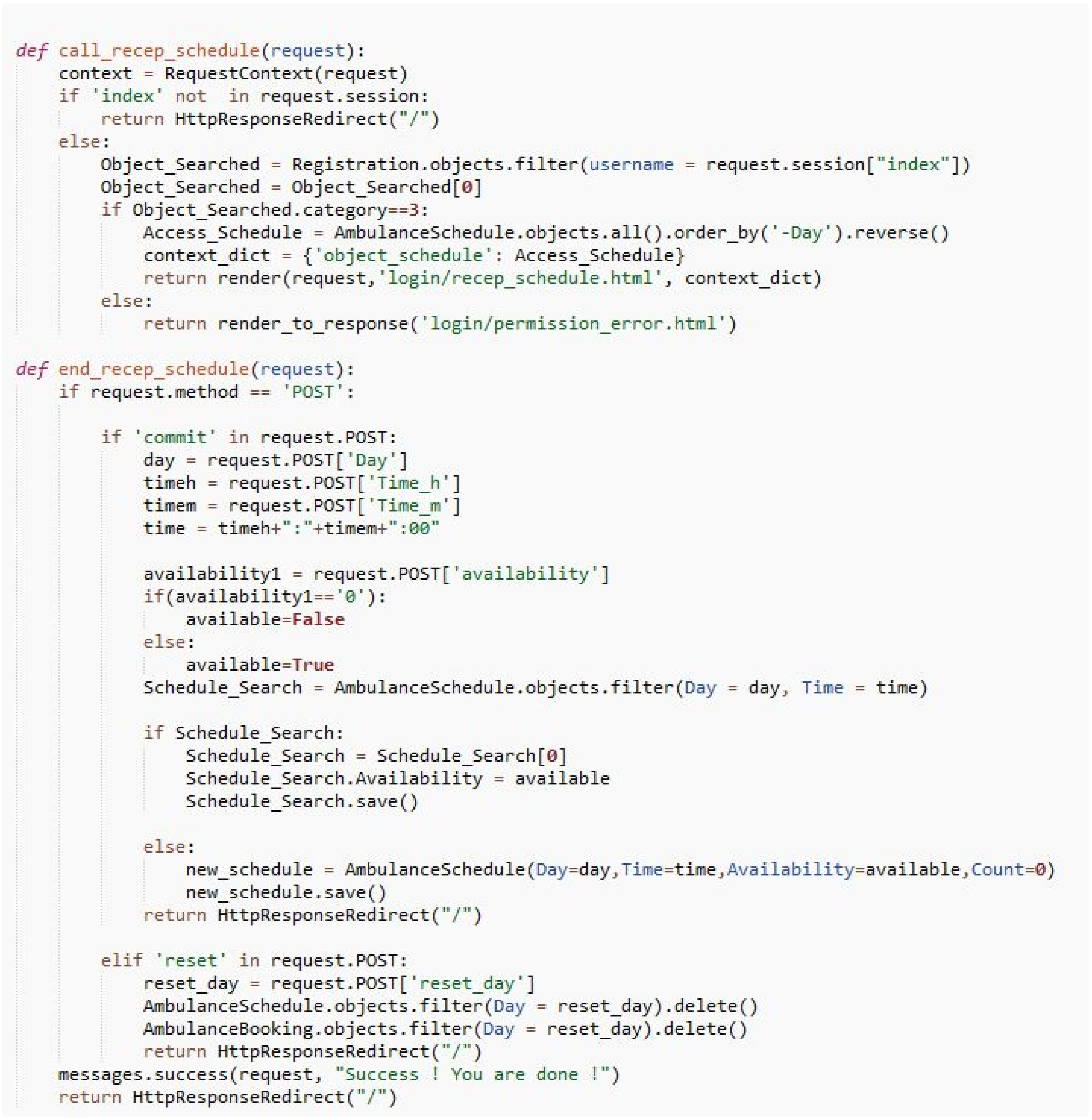
1) UPDATE AMBULANCE SCHEDULE
2) RESET AMBULANCE SCHEDULE

One of the forms is submitted with the required details.
The url is set to **recep_submit** when either of the forms is submitted which calls the view **end_recep_schedule.**

The view end_recep_schedule checks which form was submitted using the value of **request.POST**
If it is **commit** it means the Update Form is submitted
If it is **submit** it means the Reset form is submitted

In case of Update Form the corresponding schedules are pushed in the model **Ambulance_Schedule**
If a pre existing Schedule is cancelled the availability is set to 0 and the entries in the model **Ambulance_Booking** is deleted

In Reset form all the entries in the database for the given timing are deleted



```python
def call_recep_schedule(request):
    context = RequestContext(request)
    if 'index' not  in request.session:
        return HttpResponseRedirect("/")
    else:
        Object_Searched = Registration.objects.filter(username = request.session["index"])
        Object_Searched = Object_Searched[0]
        if Object_Searched.category==3:
            Access_Schedule = AmbulanceSchedule.objects.all().order_by('-Day').reverse()
            context_dict = {'object_schedule': Access_Schedule}
            return render(request,'login/recep_schedule.html', context_dict)
        else:
            return render_to_response('login/permission_error.html')

def end_recep_schedule(request):
    if request.method == 'POST':

        if 'commit' in request.POST:
            day = request.POST['Day']
            timeh = request.POST['Time_h']
            timem = request.POST['Time_m']
            time = timeh+":"+timem+":00"

            availability1 = request.POST['availability']
            if(availability1=='0'):
                available=False
            else:
                available=True
            Schedule_Search = AmbulanceSchedule.objects.filter(Day = day, Time = time)

            if Schedule_Search:
                Schedule_Search = Schedule_Search[0]
                Schedule_Search.Availability = available
                Schedule_Search.save()

            else:
                new_schedule = AmbulanceSchedule(Day=day,Time=time,Availability=available,Count=0)
                new_schedule.save()
            return HttpResponseRedirect("/")

        elif 'reset' in request.POST:
            reset_day = request.POST['reset_day']
            AmbulanceSchedule.objects.filter(Day = reset_day).delete()
            AmbulanceBooking.objects.filter(Day = reset_day).delete()
            return HttpResponseRedirect("/")
    messages.success(request, "Success ! You are done !")
    return HttpResponseRedirect("/")
```

ADD NEW NOTICE
url is set to **new_notice** which calls the **view new_notice**

In case not logged in redirects to the login page

If the user logged in is receptionist it redirects to the **new_notice.html** page.
In this page the receptionist can fill in the title of the new notice and the details.
Once he/she submits the form the **url** is set to **notice_submit** which calls the **view notice_submit**.
This view update the **model Post** with the fields submitted and the current date and time

DOCTOR SCHEDULE
url is set to **doctor_schedule** which calls the view **call_doctor_schedule**.
All doctors are fetched from the database and passed in the **context_dict**.
It redirects to the page **doctor_schedule.html**
Form asks for the details and when it is submitted the url is set to **doctor_submit** which calls the view **end_doctor_schedule**
This view updates the submitted details in the **model Doctor** altering the **Doctor.schedule** field

ADD NEW PATIENT
url is set to **add_patient** and the view **call_addpatient** is called.
This view redirects to the **createpatient.html** page
As the patient details are entered and the submit button is pressed url is set to **createpatient** which calls the **view create_patient**
This view pushes the corresponding details in the **models Patient** and **Registrations**

FAQ and Medical Form- The views ”load_faq” and “med_forms” call the utilities respectively in a new tab.

ADMIN LOGIN PORTAL

Models, Views and Templates for different functionality-

Homepage-
Template fetched - “admin_homepage.html”
The blog app handles the notice board which is explained separately.

Add new Doctors-
Models- “Doctor” and “Registration”
“create_doctor” view handles the form for creation of new doctor.
Template used- “createdoctor.html”

Add new Receptionist-
Models- “Receptionist” and “Registration”
“create_reception” view handles the form for creation of new doctor.
Template used- “createreception.html”

Add new Admin-
Models- “Registration”
“call_addadmin” view handles the form for creation of new doctor.
Template used- “admin_adduser.html”

View Doctors and View Patient Feature-
The admin can look at the present list of doctors and patient in the hospital.
Models used to fetch data- “Doctor” and “Patient”
Templates Used- “admin_viewdoctors.html” and “admin_viewpatients.html”
Views- The corresponding views are “admin_viewdoctor” and “admin_view_patient”

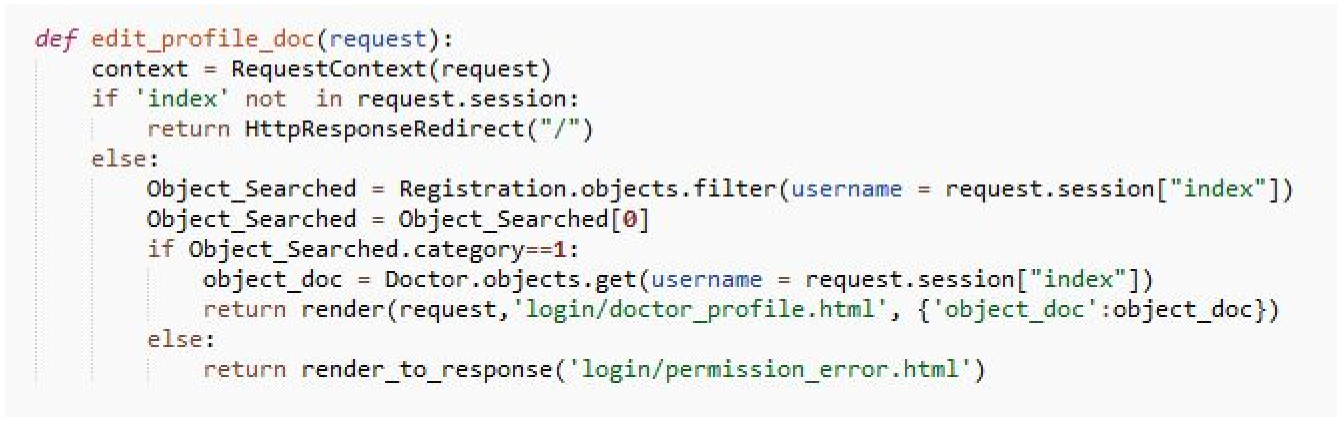
Statistical Analysis-
The “call_stats” view calls the system app which handles the Statistical Analysis module which is explained separately.
FAQ and Medical Form- The views ”load_faq” and “med_forms” call the utilities respectively in a new tab.

DOCTOR LOGIN PORTAL

Models, Views and Templates for different functionality-

Homepage-
Model Used- “Doctor” to fetch the profile information of the patient.
Template fetched - “doctor_homepage.html”
The blog app handles the notice board which is explained separately.

Edit Profile-
“edit_profile_doc” view helps to render the “doctor_profile.html” template which extends the “doctor_homepage.html” template.
“doc_prof_sub” view handles the submission of the form for edit profile. If it is a success, it redirects to the homepage with a success message.



```python
def edit_profile_doc(request):
    context = RequestContext(request)
    if 'index' not  in request.session:
        return HttpResponseRedirect("/")
    else:
        Object_Searched = Registration.objects.filter(username = request.session["index"])
        Object_Searched = Object_Searched[0]
        if Object_Searched.category==1:
            object_doc = Doctor.objects.get(username = request.session["index"])
            return render(request,'login/doctor_profile.html', {'object_doc':object_doc})
        else:
            return render_to_response('login/permission_error.html')
```

Statistical Analysis-
The “call_stats” view calls the system app which handles the Statistical Analysis module which is explained separately.

FAQ and Medical Form- The views ”load_faq” and “med_forms” call the utilities respectively in a new tab.

View Patient and Prescriptions History-
The doctor can have a look at the prescriptions given by him to the patients.
Model- The data is fetched from the “Prescription” model for this purpose.
View- “doc_pres”
Template-”doctor_pres_his.html”