

## Android Based Appointment Scheduler and Location Helper using file operation

Kujur Preeti Nancy, Pooja

ESL, Chandannagar Lab, West Bengal, India

---

**Abstract:** With the development of Android OS, an open source operating system has created an android world and enhance the software development skills, where developers are creating new application for Android devices. Application are typically developed using Java programming language using the Android Software Development Kit but another languages such as Python being dynamically typed is more preferable nowadays compared to Java. Here we created an Offline Android Application-Appointment Scheduler and Location Helper using File operation. Moreover in this application user can schedule his appointment and the user can get to know the path and direction of any nearby places by Location Helper.

---

### I. Introduction: Python:

Python is a dynamic typed, multi-paradigm programming language created by Guido Van Rossum. It almost run on every operating system such as Win, Linux/Unix, Mac, OS/2 etc. Python has a large standard Library which is one of the greatest strength of this high-level language as it provide tools for achieving various task in a flexible manner.

#### Python over Java:

Python is easy to learn, use and read compared to Java. Significantly, Python programs are shorter compared to Java. Due to the dictionary feature and list's flexibility of python it has been easy to achieve task with less complicity. Python also allows run-time typing.

#### Android Operating System:

Android is open source based Linux kernel operating system mostly designed for touchscreen for mobile devices such as mobile phones and tablet computers. Android is known to be the best operating system and the fastest emerging one, with user-friendly GUI, open to all with lots of application and amazing features.

#### Scripting Layer for Android:

Scripting Layer for Android (sl4a) is a library, designed for developers, for creating and running of scripts written in various scripting languages directly on Android devices. Scripts can access Application Programming Interface (API) which is library, usually a set of classes with set of function assigned for performing different works, also specifying the interaction between software components.

#### Motivation:

As the development of Android is at rapid pace with various online application which uses database for storing relevant information of users and comes with a problem that once in a while user may not have an internet access which makes an application idle with no use and henceforth android users search for Offline applications. Taking this problem into consideration, we intent to design an Android Offline Application-“Appointment Scheduler” and “Location” using File Operation instead of database. Moreover, we have work in various features on the application using files to store users' relevant information and we have used our algorithm for fetching information from a file using various types of file operation.

### II. Methodology:

#### Appointment Request:

**This will send a message asking for an appointment to the sender.**

Step1: Create an Alert dialog asking “Do you want to have an appointment”.

Step2: If user press positive button then

2.1: Ask for the number with whom he want to have an appointment and the time of appointment.

2.2: Send the message to the number with text in the format of –“firstname lastname time”.

Step3: No work is performed when user press negative button.

Step4: End of Algorithm

### Deleting Appointment:

After the appointment time is over, this algorithm will automatically delete the appointment fetching data from file.

Step 1: Open a file (appo.txt) in a read mode and fetch the file as a dictionary say app= {}.

Step2: Fetching the system time and split on the basis of blank spaces (" ") and present time is selected.

Step3: Check for each keys in app dictionary-

If ending time is smaller than present time then delete that key from the dictionary.

Step 4: Again open a file (appo.txt) in a write mode and write the altered dictionary (app) into the same file.

Step5: End of Algorithm.

### III. Appointment Scheduler:

One of the most important feature, where the user will reject or accept the appointment request depending upon the time schedule and priority assigned to people by user.

Step 1: Using smsGetMessage() function, fetch the unread request(messages) from inbox.

Step 2: Copy the body of message to text. And split it on the basis of blank spaces (" ") and is stroed as List, tem=[].

Step 3: Append the temp item in the another list, name=[] such as name[0] is firstname, name[1] is lastname and name[2] is the starting time of appointment. Fetch name[2] and store as tm\_start.

Step4: Pop a dialog to get an input- "how long you want to have appointmnet" and store the result as tm\_span.

Step 5: Add the tm\_start to tm\_span to get tm\_end.(tm\_end=tm\_start+tm\_span).

Step 6: Open a file (appo.txt) in read mode where the approved appointment is stored (as 'phone no': ['firstname','lastname','tm\_start','tm\_end']) and retrieve the data as a dictionary, temp= {}.

Step 7: For each keys 'i' in dictionary (temp.keys ())

7.1: If the tm\_start and tm\_end lies within a single approved appointment

Then:

Increment the the count and set the flag.

Store the value of 'i' ,i.e. key in 'x' variable.

Step 8: Check if flag is set 'AND' (&-operator) count is equal to 1

Then:

8.1: Create dictionary, priority={ }.

8.2: Open a file (cntct.txt) in a read mode where contact details along with priority of approved appointment is stored (as firstname\tlastame\tphone no\tpriority\n").

8.3: Split on the basis of new line("\n") and store it in list, s=[]

8.4: for each item 'i' in length of list s[]

i. Split item, s[i] depending on tab space (" ") and is stored as list,p=[] ii. Store the phone no. as Keys and priority as value of key in a priority= {}.

8.5: If the appointment requester is not present in priority.keys()

Then:

i.Append as "\n"+firstname+"\t"+lastname+"\t"+phone no+"\t"+"1" where 1=Default priority to previously read file of contact details.

ii. Open a file (cntct.txt) in a write mode and write the altered file.

8.6: If the value of x (phone no of the already approved appointment) is equal to the appointment requester (x=a) (here 'a' is appointment requester)

Then:

i.Update an appointment(temp[a]=[name[0],name[1],tm\_start,tm\_end])

ii.Send a message to requester- "Appointment has been Updated".

Else if priority of a (requester) is greater than priority of x (approved)

i.Update an appointment for recent requester (temp[a]=[name[0],name[1],tm\_start,tm\_end])

ii. Send a message to recent requester-"Appointmnet Approved".

iii. Call the function timeslots().

Go to step 9.

iii.Fetch the return value of function and Send a message to previously approved-"Appointment Cancelled and empty time slots".

Else: //recent requester not approved.

i.Call the function timeslots()

Go to step 9.

ii. Fetch the return value of function and Send a message to previously approved-"Appointment Cancelled and empty time slots".

Step 9: timeslots() //function

9.1: For each key 'i' in dictionary temp (temp is dictionary stores approved appointment)

i. Store the tm\_start, tm\_end of each key in an list, arr=[]

9.2: For each item 'k' in list, arr=[]

i. Split arr[k][0] (starting time) on basis of (":") and is stored in an list, m=[] where m[0] is hours and m[1] is minutes.

ii. Convert the tm\_start into minutes. (d1=m[0]\*60+m[1])

iii. Split arr[k-1][1] (ending time of previous) on the basis of (":") and is stored in list, n=[]

iv. Convert the tm\_end into minutes. (d2=n[0]\*60+n[1])

v. Difference between the starting time of k and ending time of previous (k-1) will give the time of empty time slots.

vi. If the difference is greater than equal to 30min then:

return difference, arr[k][0], arr[k-1][1] //string conversion of these time slots.

Step 10: If flag is not set

Then Check if 'a' (appointment requester) is not present in dictionary, temp.

Then: Make a new entry to the dictionary as temp[a]=[name[0], name[1], tm\_start, tm\_end]

Send the message to requester - "Appointment Approved".

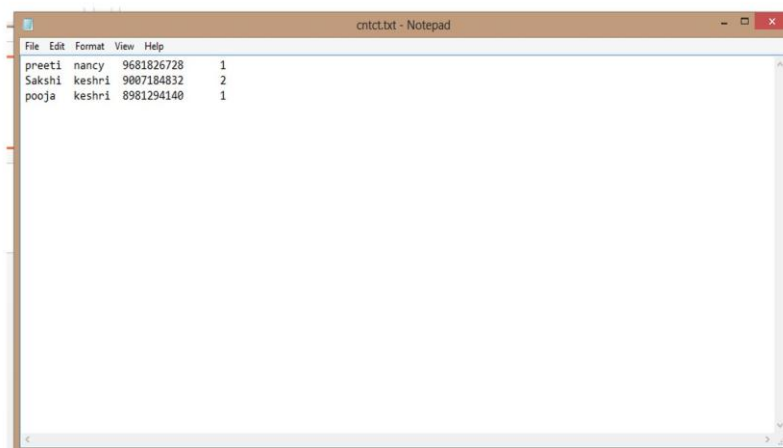
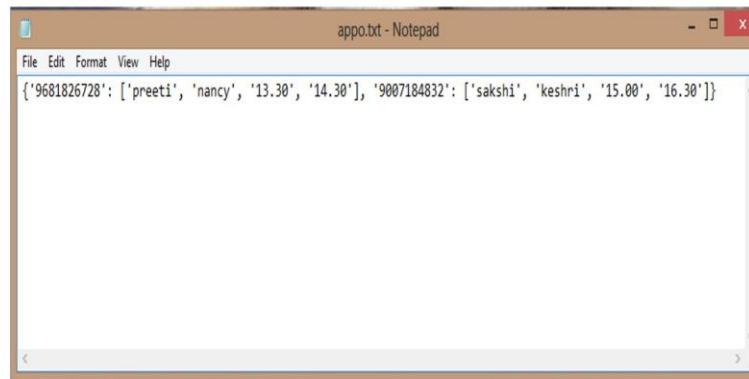
Else if a is present in dictionary, temp

Then: Update an appointment as temp[a]=[name[0], name[1], tm\_start, tm\_end]

Send the message - "Appointment Updated".

Step 11: Open a file (appo.txt) in a write mode and write the altered file into the same file.

Step 12: End of Algorithm.



### Alarm:

Here. User will be alarm few minutes before the starting time of appointment. And user also has "Snooze" option.

Step1: Get an input from user to "Enter the time for snooze", say threshold.

Step2: Open a file (appo.txt) in a read mode and storing it as a dictionary, say temp= {}.

Step3: Start an infinite loop

3.1: Fetch the system time and store it as a list after splitting it on the basis of blank spaces (“ ”).

3.2: Convert the time into minutes (hours\*60+minutes) and store as say, present time

3.3: for each keys in dictionary (temp)

3.3.1: Fetch the starting time of the key whose values is stored in the form of a list.

3.3.2: Splitting the time depending on (“.”) and storing it in list.

3.3.3: Converting the splitted time into minutes by fetching each item from list and converting in an integer and applying “Hours\*60+remaining minutes”. Store as app time.

3.3.4: Calculate the difference between the present time and app time, say diff.

3.3.5: If threshold is equal to diff

Then

- Pop-up a dialog box as an alert asking for “Snoozing” (Positive button) and “Dismissing” (Negative Button).
- If user press Snooze button then
  - Ask for entering the time of snooze.
  - Change the minutes into seconds (sec=min\*60).
  - Calling sleep () function of time module which has the snooze seconds as argument.

Else

- Display-“Alarm over”.
- Call a break keyword to exit from all the loops.

Step4: End of Algorithm.

#### **Admin Part:**

**This is feature each appointment is displayed to user asking for whether he want to accept or drop the appointment. According to the user’ desire file is altered.**

Step1: Open a file where appointment is stored in read mode and store it as dictionary, say ap= { }

Step2: For each keys in dictionary (ap)

2.1: Create an alert dialog asking for Final Approvement of Appointment stored in a file with positive (ACCEPT) and negative buttons (DECLINE).

2.2: If user response to Negative Button then

2.2.1: Send a message-“Appointment is cancelled” to that key.

2.2.2: Delete that key from the Dictionary.

2.3: Else response to Positive Button then

2.3.1: Display “Confirmed”.

Step3: Open a file (appo.txt) in write mode and write the final dictionary (ap) to the file.

Step4: End of Algorithm.

#### **Path & Location Helper:**

**This features helps the user to track all the nearby places after entering the source point and the destination (place) he want to search. In this the places along with their distance is stored in an file as to avoid complexity of database.**

Step 1: Get Input for Source point (where you are?)

Step 2: Get Input for Destination point (Enter the place you want to find?)

Step3: Take empty list t4= [].

Step 4: Open a file (loc.txt) in read mode where all the location with the distance between to places is stored in .csv (comma separated values) format.

Step 5: Splitting the file depending on new line (split (“\n”)) in the file and is stored as the list, say temp= [].

Step 6: for each item in list (temp= [])

6.1: Splitting the item depending on comma (“,”) and storing it again in a temporary list, say s= [].

6.2: Check If source is present in the list(s= [])

Then:

6.2.1: If destination is present

Then:

Display the distance of Source from destination to the user.

Else:

1. If source is store at position s [0] (list) then:

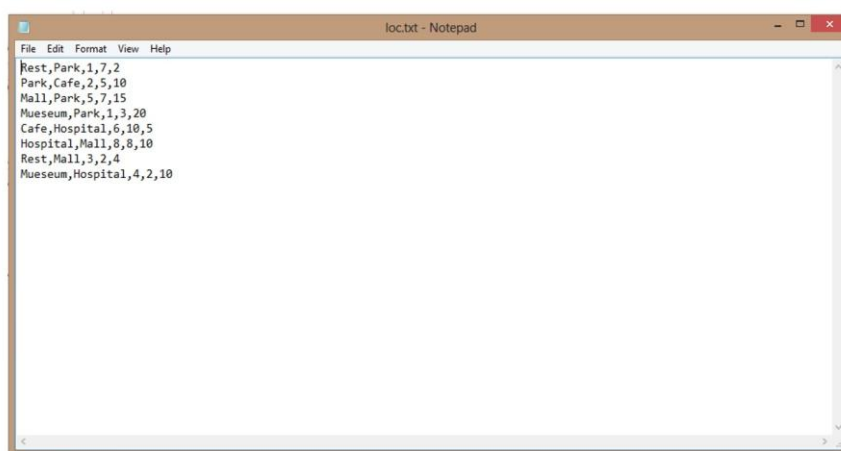
1.1: Consider hop equal to s [1] and call the function by passing value of hop as argument say, check (hop)

1.2: Go to step7.

1.3: For each value’ k’ in list say, t4= []

1.3.1: Cal distance= s [4] + t4 [k] [4]

- 1.3.2: Display the distance from source to destination taking one hop  
2: Else if source is store at s [1] position in a list (s) then:
- 2.1: Consider hop equal to s [0] and call the same function by passing value of hop as argument say, check (hop).  
2.2: Go to step 7.  
2.3: For each value in list say, t4= []  
2.3.1: Cal. distance= s [4] + t4 [k] [4]  
2.3.2: Display the distance from source to destination taking one hop.  
Step 7: Compute function for each hop, check (hop)  
7.1: for each item ('i') in the splitted list temp= [] (on the basis of (“\n”))  
7.1.1: Splitting the item depending on comma (“,”) and storing it again in a temporary list, say top= [].  
7.1.2: Check if the hop present in list, top “AND” (& operator) destination point present in list, top (for same i)  
1. Append top to t4.  
2. Return t4.  
Go to step 6.  
Step 8: End of Algorithm.



```
loc.txt - Notepad
File Edit Format View Help
Rest,Park,1,7,2
Park,Cafe,2,5,10
Mall,Park,5,7,15
Museum,Park,1,3,20
Cafe,Hospital,6,10,5
Hospital,Mall,8,8,10
Rest,Mall,3,2,4
Museum,Hospital,4,2,10
```

#### **IV. Conclusion:**

This Application is designed in order to show the advantages of file operation over database. The algorithm show the flexibility of using file for different operation and also to avoid the complexity of the database as storing a database requires physical memory. I would like to show my sincere gratitude to Sir Avranil Tah, who have motivated and directed me throughout the project.

#### **V. Future Work:**

We will work more on the File operation over database to make it more efficient and will include various features to the application to make it more interactive among user.