CISAR

Escuela: E.E.S.T N°7
T.R.Q.
Curso: 7mo 2da Av

Explicación Programa
Categoría: Project Idea

2021

Informe descriptivo de la programación principal en Raspberry:

Empezamos importando las librerías que necesitamos:

```
import logging
import os
from telegram import Update, ForceReply, ReplyKeyboardMarkup, ReplyKeyboardRemove, ParseMode
from telegram.ext import Updater, CommandHandler, CallbackContext, MessageHandler, ConversationHandler, Filters
from decouple import config
from time import sleep
from mfrc522 import SimpleMFRC522
import RPi.GPIO as GPIO
import sqlite3
from sqlite3 import Error
from datetime import date
import requests
import time

```

Creamos las variables y listas que necesitamos:

```
fechahoy = date.today()
fecha_hora = time.ctime()
edad = check = 0
nombre_apellido = fechanac = curso = division = especialidad = dni = numero_tarjeta_rfid = numero_tarjeta_rfid2 = rando = ""
GPIO.setwarnings(False)

#--------------------------IMPORTANTE---------------------------------
contador = -1000
#--------------------------IMPORTANTE---------------------------------

datatuple = []
replies = {
    "curso" : ["1ero", "2do", "3ero", "4to", "5to", "6to", "7mo"],
    "division" : ["1era", "2da", "3era", "4ta", "5ta"],
    "especialidad" : ["Avionica", "Aeronautica"],
    "procedimiento": ["SI", "Reingresar datos"]
}
```

Se ejecuta el **main**, donde se inicializa el **updater**, dejando el programa a la escucha de recibir un mensaje:

```
347  def main() -> None:
348      TOKEN = config('TOKEN')
349      updater = Updater(TOKEN)
350      dispatcher = updater.dispatcher
351      dispatcher.add_handler(CommandHandler("start", start))
352      dispatcher.add_handler(CommandHandler("Ayuda", help_command))
353      dispatcher.add_handler(CommandHandler("Voy", voy_command))
354      dispatcher.add_handler(CommandHandler("Registro", registrar))
355      dispatcher.add_handler(CommandHandler("Ingresar", chequearUsuarios))
356      dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, echo))
357
358  #     crearBasedeDatos()
359
360      updater.start_polling()
361      updater.idle()
362
363  if __name__ == '__main__':
364      main()
```

El usuario para darle inicio al bot debiese mandar “**start**”, lo cual hace que se nos ejecute la función “**start**”:

```
347  def main() -> None:
348      TOKEN = config('TOKEN')
349      updater = Updater(TOKEN)
350      dispatcher = updater.dispatcher
351      dispatcher.add_handler(CommandHandler("start", start))
352      dispatcher.add_handler(CommandHandler("Ayuda", help_command))
353      dispatcher.add_handler(CommandHandler("Voy", voy_command))
354      dispatcher.add_handler(CommandHandler("Registro", registrar))
355      dispatcher.add_handler(CommandHandler("Ingresar", chequearUsuarios))
356      dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, echo))
```

Función “**Start**”:

```
86  def start(update: Update, context: CallbackContext) -> None:
87      user = update.effective_user
88      emoji = "\U0001F601"
89      update.message.reply_markdown_v2(
90          fr'''Buenas {user.mention_markdown_v2()}\! {emoji} El bot de **Cisar** te saluda, en que puedo ayudarte? Para desplegar mi lista de comandos haz clic en /ayuda''',
91          reply_markup=ForceReply(selective=True))
```

Si el usuario necesita ver la lista de comandos presionara en /ayuda dando llamando a el comando **/ayuda** en la función main, el cual activa a la función “**help_command**”:

```
347  def main() -> None:
348      TOKEN = config('TOKEN')
349      updater = Updater(TOKEN)
350      dispatcher = updater.dispatcher
351      dispatcher.add_handler(CommandHandler("start", start))
352      dispatcher.add_handler(CommandHandler("Ayuda", help_command))
353      dispatcher.add_handler(CommandHandler("Voy", voy_command))
354      dispatcher.add_handler(CommandHandler("Registro", registrar))
355      dispatcher.add_handler(CommandHandler("Ingresar", chequearUsuarios))
356      dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, echo))
```

```
98  def help_command(update: Update, context: CallbackContext) -> None:
99      htext = "Mi lista de comandos: \n\n --> \t/Registro (Le permite registarse como alumno)\n\n --> \t/Ingresar (busca iniciar el protocolo de ingreso)\n\n --> \t/voy (usted se ocupa del sospechoso)"
100     update.message.reply_text(htext)
```

El usuario selecciona que comandó quiere utilizar:

- En el caso de seleccionar **/Registro**:
  se llamará al comando /Registro dentro del main, donde este a su vez llamará a la función "**Registrar**":

```
347  def main() -> None:
348      TOKEN = config('TOKEN')
349      updater = Updater(TOKEN)
350      dispatcher = updater.dispatcher
351      dispatcher.add_handler(CommandHandler("start", start))
352      dispatcher.add_handler(CommandHandler("Ayuda", help_command))
353      dispatcher.add_handler(CommandHandler("Voy", voy_command))
354      dispatcher.add_handler(CommandHandler("Registro", registrar))
355      dispatcher.add_handler(CommandHandler("Ingresar", chequearUsuarios))
356      dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, echo))
```

```
203 def registrar(update: Update, context: CallbackContext) -> None:
204     """1er función de la etapa de registro"""
205     global contador
206     contador = 0  #seteamos el contador en 0 para asegurarnos que no tome el nombre del usuario en cualquier string que le mandemos
207     user = update.effective_user
208     emoji = "\N{grinning face with smiling eyes}"
209     update.message.reply_markdown_v2(
210         fr'Hola {user.mention_markdown_v2()}\! {emoji} Yo soy el CisarBot encargado del registro\.'+ ' \n\nLe solicito me mande su nombre y apellido completo por escrito, por favor\. ',
211         reply_markup=ForceReply(selective=True),
212     )
```

Nos solicitará que enviemos nuestro nombre y apellido completo, para lo cual mandaremos esto en formato de texto, dando como resultado el llamado a la función "**echo**" desde el main:

```
347  def main() -> None:
348      TOKEN = config('TOKEN')
349      updater = Updater(TOKEN)
350      dispatcher = updater.dispatcher
351      dispatcher.add_handler(CommandHandler("start", start))
352      dispatcher.add_handler(CommandHandler("Ayuda", help_command))
353      dispatcher.add_handler(CommandHandler("Voy", voy_command))
354      dispatcher.add_handler(CommandHandler("Registro", registrar))
355      dispatcher.add_handler(CommandHandler("Ingresar", chequearUsuarios))
356      dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, echo))
```

Previamente en la función "**registrar**" seteamos el contador en 0, de manera que cuando mandemos texto luego de dicha función, el programa entienda que es el nombre y apellido. En caso de mandar un nombre con números se seteara otra vez el contador en cero y se le solicitará al usuario su reingreso de este dato.
Finalmente una vez que el usuario cargue un nombre válido se llamará a la función "**registro_uno**":

```python
def echo(update: Update, context: CallbackContext) -> None:
    'cada vez que mandemos un string al bot el contador = contador + 1'
    global division
    global contador
    global especialidad
    global dni
    contador = contador + 1 #el contador está global por lo que despues de todo el proceso lo voy a tener que resetear

    if contador == 1:
        global nombre_apellido
        nombre_temp = (update.message.text)
        for i in nombre_temp:
            if i == " ":
                i = ""
                nombre_apellido += i
            else:
                nombre_apellido += i

        nombre_temp = nombre_temp.title()

        if nombre_apellido.isalpha() == True:
            print("nombre y apellido: " + nombre_apellido)
            registro_uno(update, context)
        else:
            contador = 0
            emoji = "\U0001F6AB"
            update.message.reply_text("Hola como estas")
            update.message.reply_text(f"Por favor ingrese un nombre valido {emoji}")
            update.message.reply_text(f"Ingrese su nombre nuevamente")

```

En “**registro_uno**” se le mostrará la lista de cursos al usuario y el tendrá que seleccionar cuál es su curso. Una vez seleccionado se llamará a la función “**echo**” por haber mandado string.

```python
def registro_uno(update: Update, context: CallbackContext) -> int:
    reply_keyboard = [replies["curso"]]

    update.message.reply_text(
        ' --> Indique su curso',
        reply_markup=ReplyKeyboardMarkup(
            reply_keyboard, one_time_keyboard=True, input_field_placeholder='Elegi bien'
        ),
    )
```

Dentro de “**echo**” se le suma 1 al contador, dejándonos el contador en 2, por lo cual el programa entiende que el string que le estamos pasando ahora lo debe de almacenar en la variable “**curso**”:

```python
def echo(update: Update, context: CallbackContext) -> None:
    'cada vez que mandemos un string al bot el contador = contador + 1'
    global division
    global contador
    global especialidad
    global dni
    contador = contador + 1 #el contador está global por lo que despues de todo el proceso lo voy a tener que resetear

```

Almacenará el dato en la variable de manera global, imprimirá por pantalla y finalmente llamaremos a la función “**registro_dos**”:

```
149         elif contador == 2:
150             global curso
151             curso=(update.message.text)
152             print("Año: "+ curso)
153             #datatuple.append(curso)
154             registro_dos(update, context)
```

Dentro de la función “**registro_dos**” si el curso es 1ro, 2do o 3ero se mostrará la lista completa de divisiones para escoger a cuál pertenece. En caso de haber seleccionado previamente ser alumno de 4to, 5to, 6to o 7mo se mostrará la lista de divisiones recortando sólo las primeras 2 divisiones.
Una vez que el usuario seleccione su división se llamará a la función “**echo**”

```
224 def registro_dos(update: Update, context: CallbackContext) -> int:
225     if curso =='1ero' or curso =='2do' or curso =='3ero' :
226       reply_keyboard = [replies["division"]]
227     elif curso == '4to' or curso == '5to' or curso =='6to' or curso =='7mo':
228       reply_keyboard = [replies["division"][0:2]]
229 
230     update.message.reply_text(
231         ' --> Seleccione su division',
232         reply_markup=ReplyKeyboardMarkup(
233             reply_keyboard, one_time_keyboard=True, input_field_placeholder='Escribi bien'
234         ),
235     )
```

Al ingresar en “**echo**” aumenta el contador por lo cual tendría un valor de 3, con lo cual ingresará en la condición de que contador es 3, almacenará el string recibido en la variable (previamente declarada como global) “**division**”.
Si el curso seleccionado al momento de registrarse es 1ro, 2do o 3ero, imprimimos la división registrada y en la variable “especialidad” se almacenará un “-” y se llamará directamente a la función “**registro_cuatro**”.
En caso contrario, donde el curso del usuario en proceso de registro es 4to, 5to, 6to o 7mo, imprimimos por pantalla la división registrada y hacemos llamado a la función “**registro_tres**”:

```
156     elif contador == 3:
157         division=(update.message.text)
158         if curso =='1ero' or curso =='2do' or curso =='3ero' :
159             print("Division: "+ division)
160             #datatuple.append(division)
161             especialidad = "-"
162             print("Especialidad: "+ especialidad)
163             registro_cuatro(update, context)
164 
165         elif curso == '4to' or curso == '5to' or curso =='6to' or curso =='7mo':
166             print("Division: "+ division)
167             #datatuple.append(division)
168             registro_tres(update, context)
```

Se ejecuta “**registro_tres**”, donde se enseñará la lista de especialidad:

```
234
235   def registro_tres(update: Update, context: CallbackContext) -> int:
236       """busca obtener a que especialidad pertenece."""
237       reply_keyboard = [replies["especialidad"]]
238
239       update.message.reply_text(
240           ' --> Seleccione su especialidad',
241           reply_markup=ReplyKeyboardMarkup(
242               reply_keyboard, one_time_keyboard=True, input_field_placeholder='Elegi bien'
243           ),
244       )
```

Una vez seleccionada la especialidad, se envía dicho string ingresando en la función "**echo**" aumentando el valor del contador en 1, dando como resultado que contador=4. Con la especialidad previamente definida como global, almacenamos el dato recibido en la variable "**especialidad**", lo imprimimos por pantalla y realizamos el llamado de la función "**registro_cuatro**":

```
170     elif contador == 4:
171         especialidad=(update.message.text)
172         print("Especialidad: "+ especialidad)
173         registro_cuatro(update, context)
```

Dentro de "**registro_cuatro**" declaramos al contador de manera global para posteriormente setearlo en 4 (esto es debido a que si el usuario coloco que pertenece a un curso de ciclo básico, su especialidad quedó definida como "-" y el contador se quedó en 3, debiendo estar en 4 para pasar al registro de DNI y así ingresar correctamente en el apartado de DNI dentro de "**echo**").
Finalmente le solicitamos al usuario que ingrese por teclado su número de DNI:

```
246 def registro_cuatro(update: Update, context: CallbackContext) -> int:
247     global contador
248     contador = 4 #coloco al contador en 4 para que luego cuando entre en echo se saltee la especialidad en caso de ser necesario
249     user = update.effective_user
250     update.message.reply_markdown_v2 (fr''' \-\-\> Ahora {user.mention_markdown_v2()}\! Le voy a solicitar que ingrese el numero de DNI''',
251         reply_markup=ForceReply(selective=True)
252     )
```

Con la variable DNI declarada previamente como global dentro de "**echo**" indicamos que el string recibido debe ser almacenado en la variable "**dni**", además de indicar que si el número introducido tiene más de 8 dígitos, se setea el contador en 4 y se llamará a la función "**registro_seis**", donde se solicitará que reingrese el dni (permitimos que el dni ingresado sea menor a 8 debido a que algunos de estos no llegan a 8 dígitos).
Si el dni ingresado cumple con los requisitos mencionados se llamará a la función "**chequeoDeDatos**":

```
178
179     elif contador == 5:
180         dni = (update.message.text)
181         lenght = len(dni)
182         if dni.isnumeric() == True and lenght <9:
183             print("DNI: " + dni)
184             #datatuple.append(dni)
185             chequeoDeDatos(update, context)
186         else:
187             contador = 4
188             emoji = "\U0001F6AB"
189             update.message.reply_text(f"Por favor ingrese un DNI valido {emoji}")
190             registro_seis(update, context)
```

```
253
254 def registro_seis(update: Update, context: CallbackContext) -> int:
255     global contador
256     user = update.effective_user
257     update.message.reply_markdown_v2 (fr''' \-\-\> {user.mention_markdown_v2()}\! Por favor ingrese el numero de DNI nuevamente''',
258         reply_markup=ForceReply(selective=True)
259     )
```

En “**chequeoDeDatos**” declaramos el contador de manera global, se ejecuta un mensaje pre-definido con los datos cargados por el usuario (almacenados en las variables) donde se pregunta si estos datos son correctos y se visualiza una lista de opciones “**procedimiento**” donde debe escoger entre “**SI**” o “**Reingresar datos**”.

```
261
262 def chequeoDeDatos(update: Update, context: CallbackContext) -> int:
263     global contador
264     #Para confirmar los datos ingresados:
265     procedimiento = ""
266     emoji = "\N{white small square}"
267     reply_keyboard = [replies["procedimiento"]]
268
269     update.message.reply_text(
270     f"Su datos ingresados son: \n\n {emoji} Nombre y apellido: " + nombre_apellido + f"\n\n {emoji} Año: " + curso +
271     f"\n\n {emoji} Division: " + division + f"\n\n {emoji} Especialidad: " + especialidad + f"\n\n {emoji} DNI: " + dni + f"\n\nSon datos sus correctos? ",
272     reply_markup=ReplyKeyboardMarkup(
273         reply_keyboard, one_time_keyboard=True, input_field_placeholder='Elegi bien'
274         ),
275     )
```

```
27  replies = {
28      "curso" : ["1ero", "2do", "3ero", "4to", "5to", "6to", "7mo"],
29      "division" : ["1era", "2da", "3era", "4ta", "5ta"],
30      "especialidad" : ["Avionica", "Aeronautica"],
31      "procedimiento": ["SI", "Reingresar datos"]
32  }
```

Una vez seleccionada la confirmación, ingresará en “**echo**” dando como resultado que contador = 6, donde se almacena el dato recibido en la variable local “**procedimiento**” y si seleccionó “**Reingresar datos**” el contador se seteará en -1000 (para evitar bugs) y se hará llamado a la función “registrar” para iniciar el proceso de registro desde 0.
En caso de haber seleccionado “**SI**” se llama a la función “**registro_cinco**”:

```
192     elif contador == 6:
193         procedimiento = (update.message.text)
194         if procedimiento =='SI' :
195             registro_cinco(update, context)
196 
197         elif procedimiento == "Reingresar datos":
198             contador == -1000 #seteamos el contador porque tiene que volver a empezar de cero el proceso
199             registrar(update, context)
```

En **“registro_cinco**” declaramos contador de manera global y lo seteamos en -1000. Indicamos al usuario mediante un mensaje que debe apoyar la tarjeta RFID sobre el sensor y se llama a la función “**leerRfid**” (encargada de lectura de RFID). Luego llamamos a la función “**subirBasedeDatos**” encargada de subir los datos al browser.

```
277 def registro_cinco(update: Update, context: CallbackContext) -> int:
278     """busca registrar la tarjeta/llavero RFID."""
279     global contador
280     emoji = ""
281     contador = -1000 #reseteo el contador para que la proxima persona que quiera registrase lo haga
282     update.message.reply_text("Apoye la tarjeta sobre el sensor: ")
283 
284     leerRfid()
285 
286     #rando = numero_tarjeta_rfid[0]
287     #numero_tarjeta_rfid = rando(str)
288     #datatuple.append(numero_tarjeta_rfid)
289     subirBasedeDatos(update, context)
```

```
75  def leerRfid():
76      try:
77          global numero_tarjeta_rfid
78          reader = SimpleMFRC522()
79          id, text = reader.read()
80          print("Numero de identificacion:", id)
81      finally:
82          GPIO.cleanup()
83          numero_tarjeta_rfid = id
84      return
```

```
65  def subirBasedeDatos(update, context):
66      sqliteConnection = sqlite3.connect('/home/pi/Desktop/Principal/BASE_DE_DATOS_CISAR.db')
67      cursor = sqliteConnection.cursor()
68      cursor.execute("INSERT INTO usuarios VALUES (?,?,?,?,?,?)", (nombre_apellido, curso, division, especialidad, dni, numero_tarjeta_rfid))
69      emoji = "\N{flexed biceps}"
70      update.message.reply_text(f"Datos cargados satisfactoriamente! {emoji} ")
71 
72      sqliteConnection.commit()
73      sqliteConnection.close()
```

- En el caso de seleccionar “**/Ingresar**” se llamará a dicho comando dentro del main, donde este a su vez llamará a la función “**chequearUsuarios**”:

```python
def main() -> None:
    TOKEN = config('TOKEN')
    updater = Updater(TOKEN)
    dispatcher = updater.dispatcher
    dispatcher.add_handler(CommandHandler("start", start))
    dispatcher.add_handler(CommandHandler("Ayuda", help_command))
    dispatcher.add_handler(CommandHandler("Voy", voy_command))
    dispatcher.add_handler(CommandHandler("Registro", registrar))
    dispatcher.add_handler(CommandHandler("Ingresar", chequearUsuarios))
    dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, echo))
```

Dentro de la función “**chequearUsuarios**” se le enviará un mensaje al usuario indicando que debe apoyar la tarjeta sobre el sensor. Se llamará a la función encargada de la lectura del RFID “**leerRfid**” y se establecerá una conexión con el browser de la base de datos en búsqueda del número de identificación de la tarjeta asociado a un nombre, apellido, curso, división, especialidad y dni.
se declara la variable local “**check**” y se la setea en 0; de manera que el for se encargue de leer cada fila del registro tomado de la base de datos, una vez que encuentra al número de la tarjeta RFID almacenado en la base de datos toma el nombre, apellido, curso, división, especialidad y dni asociados a este número de tarjeta se le suma un 1 a “**check**” :

```python
def chequearUsuarios(update, context):
    update.message.reply_text("Apoye la tarjeta sobre el sensor ")

    leerRfid()

    sqliteConnection = sqlite3.connect('/home/pi/Desktop/Principal/BASE_DE_DATOS_CISAR.db')
    sqlite_select_query = """SELECT numero_tarjeta_rfid, nombre, curso, division, especialidad, dni from Usuarios"""
    cursor = sqliteConnection.cursor()
    cursor.execute(sqlite_select_query)
    records = cursor.fetchall()
    check = 0

    for row in records:
        print(row)
        sleep(.5)

        if row[0] == numero_tarjeta_rfid:
            nombreuser = row[1]
            cursouser =  row[2]
            divisionuser = row[3]
            especialidaduser = row[4]
            dniuser = row[5]
            check =+ 1
            break

```

```python
def leerRfid():
    try:
        global numero_tarjeta_rfid
        reader = SimpleMFRC522()
        id, text = reader.read()
        print("Numero de identificacion:", id)
    finally:
        GPIO.cleanup()
        numero_tarjeta_rfid = id
    return
```

Por lo que evaluamos si “**check**” es mayor a 0, si es así declaramos a “**pul**” como variable global y enviamos un mensaje indicando al usuario que se encuentra en la base de datos, declaramos la variable local “**pulsador**” y le almacenamos el pin 40 de la Raspberry.
En cambio si “**check**” es igual a 0 se le envía un mensaje al usuario indicando que no se encuentra registrado en la base de datos y se ejecuta la función “**ingreso_no_exitoso**”.

Si recibimos un 1 (5V) desde la Arduino en el puerto de sensado de temperatura se ejecutará un mensaje pre-definido con las variables obtenidas de la base de datos previamente con destino el grupo de los directivos de la institución, a manera de alerta de sospechoso y se ejecutará la función “**ingreso_sospechoso**”
Por otro lado si no recibimos un 1 (5V) desde la Arduino en el puerto de sensado de temperatura se ejecutará la función “**ingreso_exitoso**”:

```python

if check > 0:
    global pul
    print(f"Se encuentra en la base de datos: " , row)
    GPIO.setmode(GPIO.BOARD)
    pulsador = 40
    GPIO.setup(pulsador,GPIO.IN)

    time.sleep(3)
    dniuser = str(dniuser)
    if (GPIO.input(pulsador) == 1):
        r = requests.get("https://api.telegram.org/bot1611398547:AAG9YCiIxoW1SrGpsSHzDj1vSXMlqLf5kEY/sendMessage?chat_id=-1001507958281&text=El%20sujeto%20" + nombreuser
        with open("index.html", "wb") as f:
            f.write(r.content)
            r.close()
        print ("Pulsador encendido")
        print ("$$$$$$$$$$$$$$$$$$$")
        time.sleep(1)
        ingreso_sospechoso(update, context)

    else:
        print ("Pulsador apagado")
        print ("$$$$$$$$$$$$$$$$$$$")
        time.sleep(1)
        ingreso_exitoso(update, context, nombreuser)
    GPIO.cleanup()

else:
    print("No se encuentra en la base de datos")
    ingreso_no_exitoso(update, context)

```

Dentro de la función “**ingreso_exitoso**”: Se le envía un mensaje al usuario indicando que puede ingresar a la cabina y deja un registro de fecha y hora del ingreso.

Dentro de la función “**ingreso_sospechoso**”: Se le envía un mensaje al usuario indicando que debe retirarse de la institución.

Dentro de la función “**ingreso_no_exitoso**”: Se le envía un mensaje al usuario indicando que no se encuentra registrado y se le enseña al usuario el comando para registrarse.

```
39  def ingreso_exitoso(update: Update, context: CallbackContext, nombreuser) -> None:
40      htext = '''Bienvenido/a ''' + nombreuser + '''\n\nPuede ingresar a la cabina\n\n'''
41      Htext = f"Fecha y Hora de ingreso: \n\n{fecha_hora}"
42      update.message.reply_text(htext + Htext)
43
44  def ingreso_sospechoso(update: Update, context: CallbackContext) -> None:
45      htext = ''' Volvete a tu casa lagarto'''
46      update.message.reply_text(htext)
47
48  def ingreso_no_exitoso(update: Update, context: CallbackContext) -> None:
49      htext = ''' Usted no se encuentra registrado, \n /Registro para registarse'''
50      update.message.reply_text(htext)
```

- En el caso de seleccionar /Voy, se llamará al comando /Voy dentro del main, donde este a su vez llamará a la función “**voy_command**”:

```
347  def main() -> None:
348      TOKEN = config('TOKEN')
349      updater = Updater(TOKEN)
350      dispatcher = updater.dispatcher
351      dispatcher.add_handler(CommandHandler("start", start))
352      dispatcher.add_handler(CommandHandler("Ayuda", help_command))
353      dispatcher.add_handler(CommandHandler("Voy", voy_command))
354      dispatcher.add_handler(CommandHandler("Registro", registrar))
355      dispatcher.add_handler(CommandHandler("Ingresar", chequearUsuarios))
356      dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, echo))
```

Se ejecuta la función “**voy_command**”, la cual envía le responde seleccionando el mensaje del usuario dentro del grupo de administración.

```
94  def voy_command(update: Update, context: CallbackContext) -> None:
95      htext = '''Usted se está comprometiendo con proceder con el posible sospechoso, tenga cuidado.'''
96      update.message.reply_text(htext)
```