# Playing with Google AIY Voice

## Prerequisites

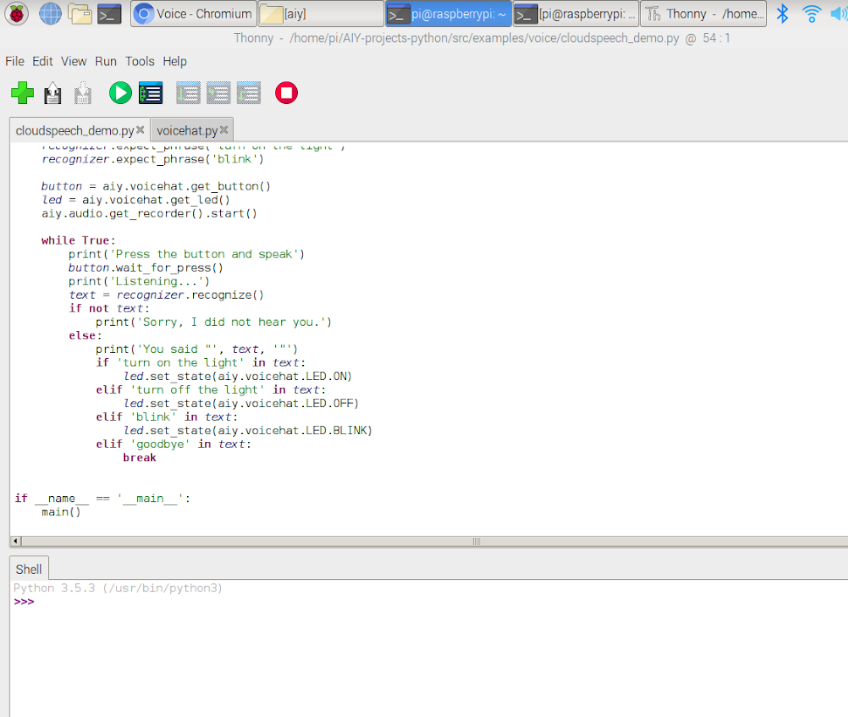
1. Set up the keyboard, as the disk image is for US layout

Raspberry -> preferences -> Mouse and keyboard Settings -> keyboard tab -> layout button. Select “united kingdom” in the left hand “country” pane and click ok.

1. Wifi. Credentials will be on display, click on the networking applet in the top right corner and connect to the indicated network with the supplied password. Once the icon is stable, double click the “check wifi” script on the desktop and follow the prompts to test that everything is working.
2. Audio. Double click on the “check audio” script on the desktop and follow the prompts to confirm that the audio is working correctly.

## Good to Know

* Chromium will use most of your memory. Try not to have more than one or two tabs open.
* Ctrl alt & F1 kills the GUI. Ctrl Alt & F2 takes you to a logon prompt.
* “sudo shutdown now” will shut the pi down gracefully from a terminal.
* Default user is “pi” and the password is “raspberry”.
* All the example files are in /home/pi/AIY-projects-python/src/examples/voice/
* The modules are in /home/pi/AIY-projects-python/src/aiy/
* Downloaded files by default go in /home/pi/downloads/
* Number lock is OFF by default. You might want to turn it on.
* The easiest way to edit the files for windows users is to double click the folder icon in the toolbar at the top, and browse to the file, as if running windows explorer. Double clicking a .py file will give you the option to execute or open it in Thonny. For everyone else, there are no prizes for using vi.

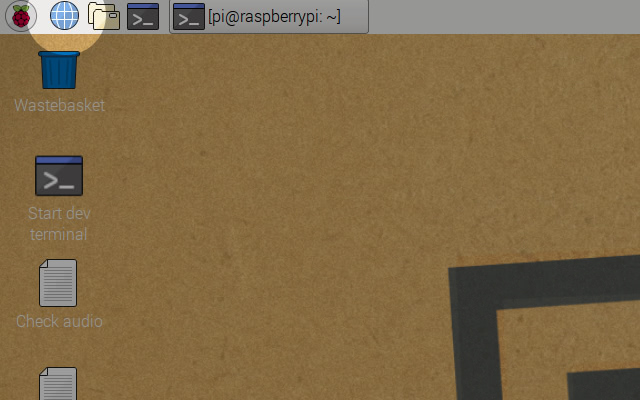


## Lab 1. Setting up your device for the Google Assistant API

### 1.1. CONNECT TO GOOGLE CLOUD PLATFORM

To try the Google Assistant API, you need to first sign into Google Cloud Platform (GCP) and then enable the API.

**Log into GCP**



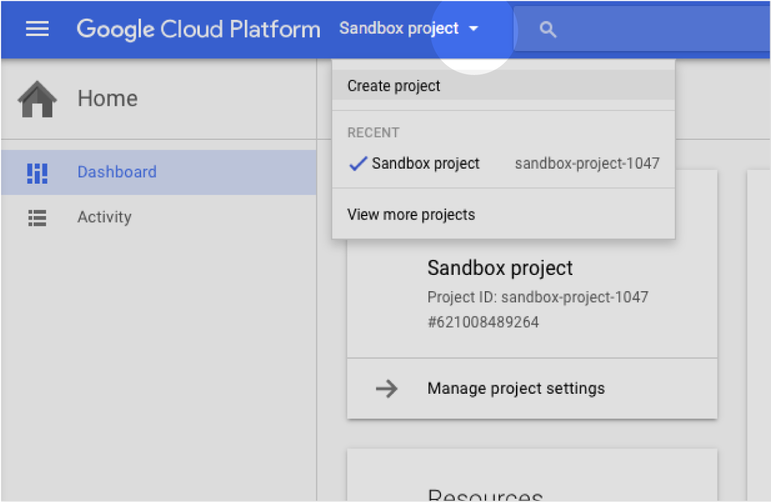
Using your voice recognizer device, open up an internet browser and go to the [Cloud Console](https://console.cloud.google.com/)

**I’VE NEVER USED GOOGLE CLOUD PLATFORM BEFORE**

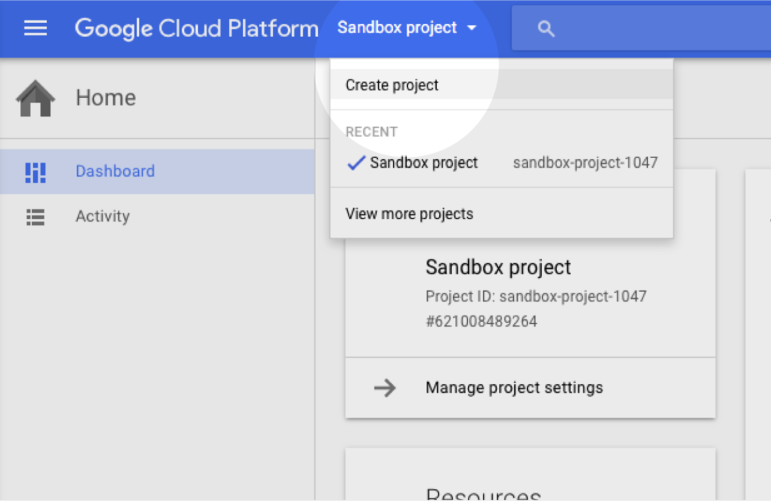
Use your Google account to sign in. If you don’t have one, you’ll need to create one. Trying the Google Assistant API is free to use for personal use.

**Create a project**

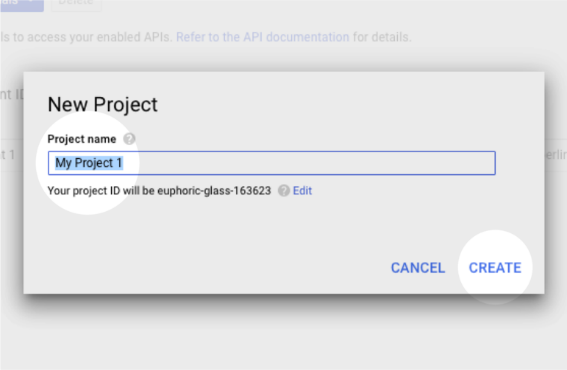
GCP uses projects to organize things. Create one for your voice recognizer box.



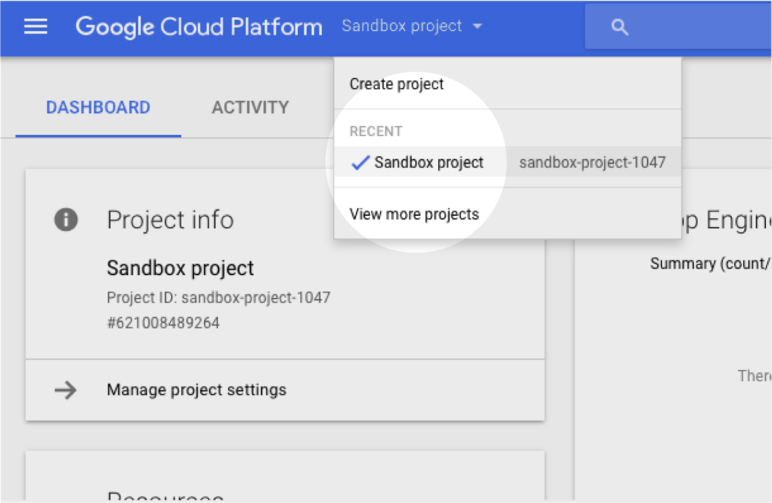
In the Cloud Console, click the drop-down button to the right of the “Google Cloud Platform” logo



From the dropdown, click Create project

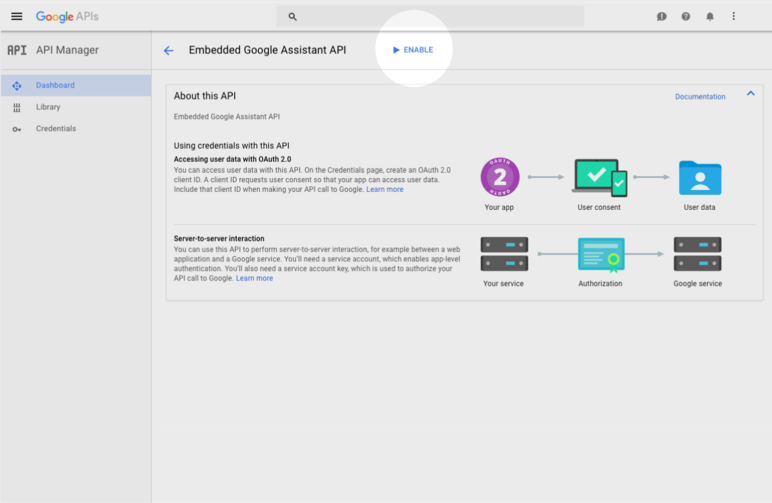


Enter a project name and click Create

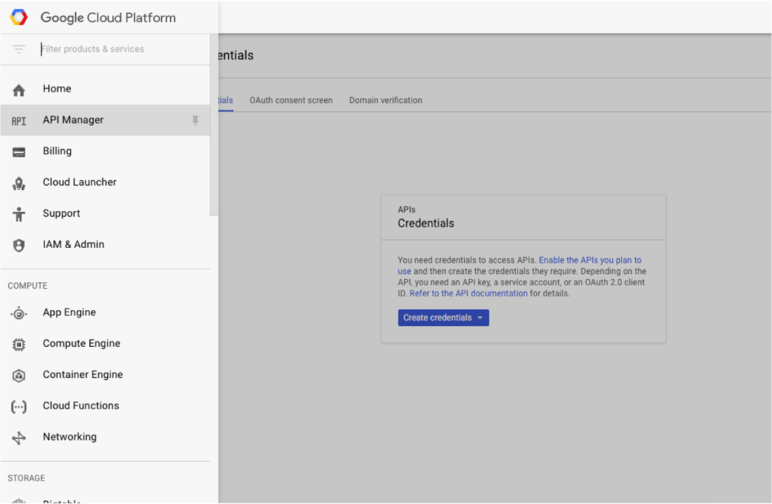


After your project is created, make sure the drop-down has your new project name displayed

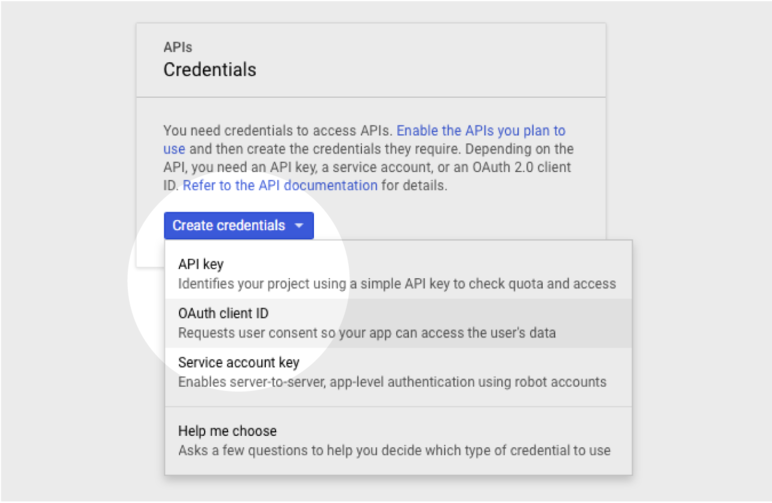
### 1.2. TURN ON THE GOOGLE ASSISTANT API



In the Cloud Console, [enable the "Google Assistant API"](https://console.developers.google.com/apis/api/embeddedassistant.googleapis.com/overview).

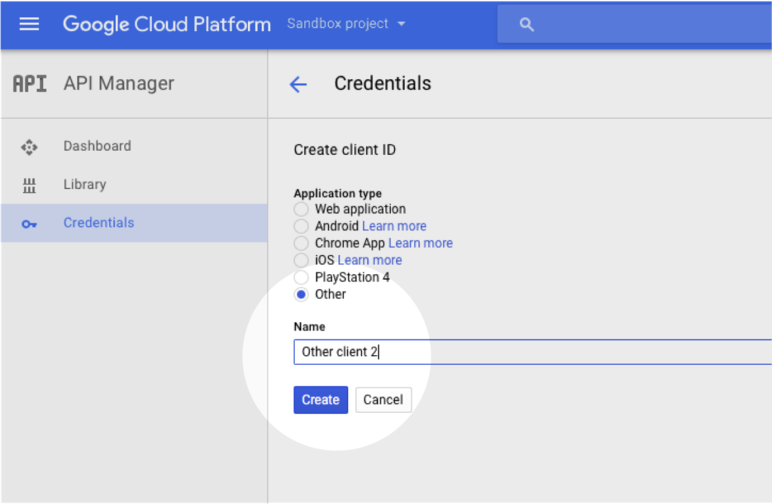


In the Cloud Console, [create an OAuth 2.0 client](https://console.cloud.google.com/apis/credentials/oauthclient) by going to API Manager > Credentials

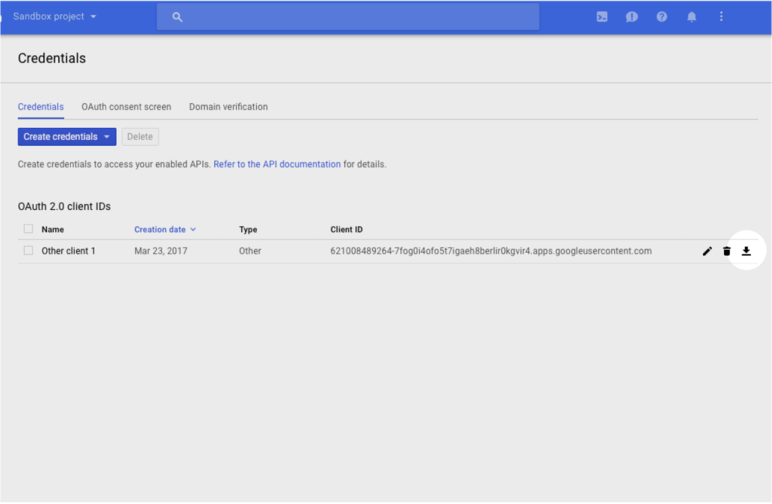


Click Create credentials and select OAuth client ID

* If this is your first time creating a client ID, you’ll need to configure your consent screen by clicking Configure consent screen. You’ll need to name your app (this name will appear in the authorization step)

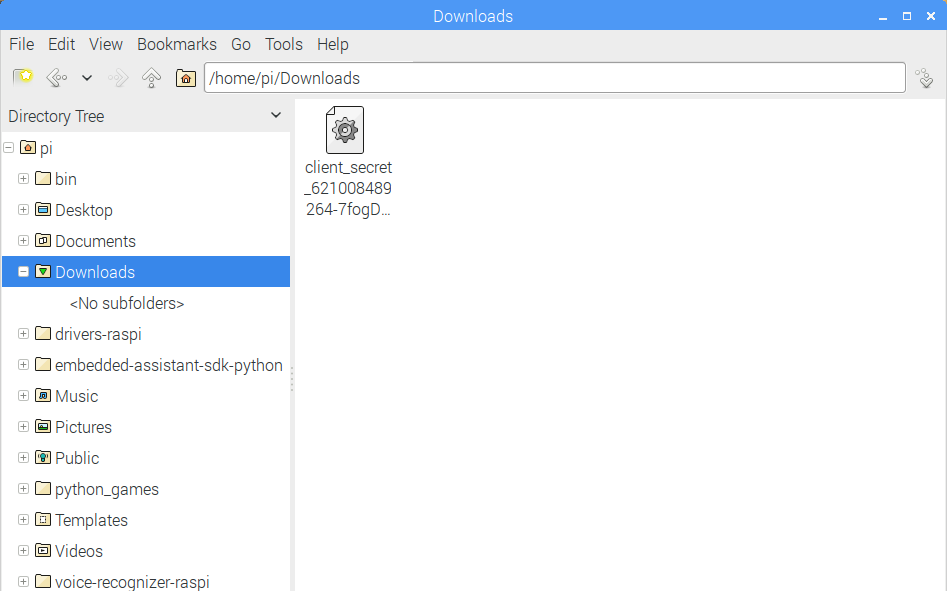


Select Other, enter a name to help you remember your credentials, then click Create.

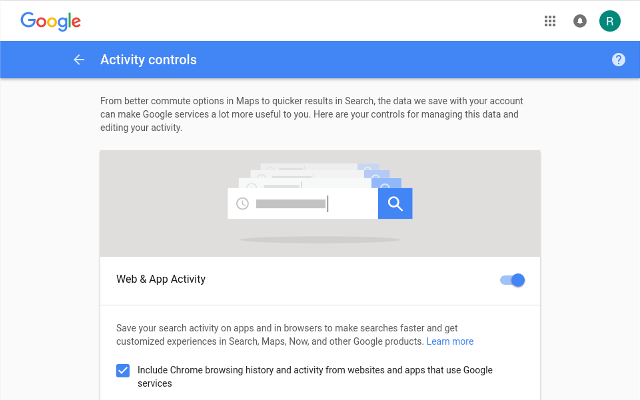


A dialog window will pop up. Click OK. In the Credentials list, find your new credentials and click the download icon (Download icon) on the right.

Note: if you don't see the download icon, try expanding width of your browser window or zooming out.



Find the JSON file you just downloaded (client\_secrets\_XXXX.json) and rename it to assistant.json. Then move it to /home/pi/assistant.json



Go to the [Activity Controls](https://myaccount.google.com/activitycontrols) panel. Make sure to log in with the same Google account as before.

* Turn on the following:
  1. Web and app activity – tick the box to track chrome. You can untick it after we’re done.
  2. Device information
  3. Voice and audio activity

### 1.3 RUN THE PROGRAM

You’re ready to turn it on: follow the manual start instructions below:

For the device to begin acting as your Google Assistant much like Google Home, start the assistant library demo app by double-clicking "Start dev terminal" on the Desktop and entering:

src/examples/voice/assistant\_library\_demo.py

The assistant library app has hotword detection built-in. To start a conversation with the Google Assistant, say "Okay, Google" or "Hey Google". When you are done, press Ctrl-C to end the application.

You can also SSH from another computer. You’ll need to use ssh -X to handle authentication through the browser when starting the example for the first time.

Authorize access to the Google Assistant API, when prompted

Make sure you're following the [manual start instructions](https://aiyprojects.withgoogle.com/voice#users-guide-3-1--start-the-assistant-library-demo-app) the first time - if you run as a service, you won't be prompted for authorization. It may hang at this point. If so, try again. If it repeatedly hangs, then scream for help.

Try an example query like "how many ounces in 2 cups" or "what's on my calendar?"-- and the Assistant should respond!

If the voice recognizer doesn't respond to your button presses or queries, you may need to restart.

If the response is ‘Actually, there are some basic settings that need your permission first...’, check the activity controls again, being sure to use the same account that you used for the authorization step.

## Lab 2. Interact with a button

This code demonstrates two ways of using the google assistant with hardware interaction and a different trigger. Compare the code in assistant\_gprc\_demo.py and assistant\_library\_with\_button\_demo.py. note the use of threading in the second one. This allows the button loop and assistant loops to run simultaneously.

|  |
| --- |
| class MyAssistant(object): |
|  | """An assistant that runs in the background. |
|  |  |
|  | The Google Assistant Library event loop blocks the running thread entirely. |
|  | To support the button trigger, we need to run the event loop in a separate |
|  | thread. Otherwise, the on\_button\_pressed() method will never get a chance to |
|  | be invoked. |
|  | """ |
|  |  |
|  | def \_\_init\_\_(self): |
|  | self.\_task = threading.Thread(target=self.\_run\_task) |
|  | self.\_can\_start\_conversation = False |
|  | self.\_assistant = None |

Double-click "Start dev terminal" on the Desktop and enter:

src/examples/voice/assistant\_library\_with\_button\_demo.py

to see it in action.

## Lab 3. Interact with the pi

This demonstrates the use of the assistant to control the local device. You should be able to get the assistant to tell you both the local ip and the public ip addresses.

|  |
| --- |
| def reboot\_pi(): |
|  | aiy.audio.say('See you in a bit!') |
|  | subprocess.call('sudo reboot', shell=True) |

…

|  |
| --- |
| elif text == 'reboot': |
|  | assistant.stop\_conversation() |
|  | reboot\_pi() |

Double-click "Start dev terminal" on the Desktop and enter:

src/examples/voice/assistant\_library\_with\_local\_commands\_demo.py

the built in commands are ‘power off’, ‘reboot’ and ‘say ip’.

## Lab 4. CHANGE TO THE CLOUD SPEECH API

Want to try another API? Follow the instructions below to try the Cloud Speech API, which recognizes your voice speech and converts it into text. The Cloud Speech API supports 80 languages, long audio clips, and the ability to add phrase hints for processing audio.

**Turn on billing**

**WHY DO I NEED TO TURN ON BILLING?**

The voice recognizer cube uses Google’s Cloud Speech API. If you use it for less than 60 minutes a month, it’s free. Beyond that the cost is $0.006 for 15 seconds. Don’t worry: you’ll get a reminder if you go over your free limit.

1. In the Cloud Console, open the **navigation menu** Navigation menu
2. Click **Billing**
3. If you don’t have a billing account, then click **New billing account** and go through the setup
4. Return to the [main billing page](https://console.cloud.google.com/billing), then click the **My projects** tab.
5. Find the name of your new project. Make sure it’s connected to a billing account.
6. To connect or change the billing account, click the **three-dot** button Navigation menu, then select **Change billing account**

**Enable the API**

1. In the [console](https://console.cloud.google.com/), open the **navigation menu** and click **API Manager**
2. Click **ENABLE API**
3. Enter “Cloud Speech API” into the search bar, then click the name
4. Click **ENABLE** to turn on the API

**Create a service account and credentials**

1. Go to the left-hand navigation menu, click **API Manager** and then click **Credentials**
2. Click **Create credentials** and then click **Service account key** from the list
3. From the “Service account” dropdown, click **New service account**
4. Enter a name so that you’ll know this is for your voice recognizer stuff, like “Voice credentials”
5. Select the **Project viewer** role
6. Use the JSON key type
7. Click **Create**
8. Your credentials will download automatically. The file name contains your project name and some numbers: locate it rename it to **cloud\_speech.json**
9. Open your workstation’s terminal. Move your credentials.json file to the /home/pi/ folder and rename it cloud\_speech.json.

**Start the demo app**

On your desktop, double-click the **Start Dev Terminal** icon. Then start the app:

src/examples/voice/cloudspeech\_demo.py

**CHECK THAT IT WORKS CORRECTLY**

On your desktop, double-click the **Check Cloud** icon. Follow along with the script. If everything is working correctly, you’ll see this:

The cloud connection seems to be working

If you see an error message, follow the details and try the Check Cloud script again.

### 4.1 VOICE COMMANDS

To issue a voice command, press the arcade button once to activate the voice recognizer and then speak loudly and clearly.

To see the LED you will need to raise the lid of the AIY box – it’s on the VoiceHAT board.

| **VOICE COMMAND** | **RESPONSE** |
| --- | --- |
| turn on the light | The LED is turned on and is solid |
| turn off the light | The LED is turned off |
| blink | The LED starts blinking |
| goodbye | The app automatically exits |

### 4.2 CREATE A NEW VOICE COMMAND (OR ACTION)

You can create new actions and link them to new voice commands by modifying src/examples/voice/cloudspeech\_demo.py directly.

**Example: repeat after me**

To add a voice command, first make it explicit what command is expected to the recognizer. This improves the recognition rate:

  recognizer = aiy.cloudspeech.get\_recognizer()

recognizer.expect\_phrase('turn off the light')

recognizer.expect\_phrase('turn on the light')

recognizer.expect\_phrase('blink')

recognizer.expect\_phrase('repeat after me')

Then add the code to handle the command. We will use aiy.audio.say to repeat the recognized transcript:

// In the process loop. 'text' contains the transcript of the voice command.

elif 'repeat after me' in text:

// Remove the command from the text.

to\_repeat = text.replace('repeat after me', '', 1)

aiy.audio.say(to\_repeat)

The modified cloudspeech\_demo.py looks like this:

"""A demo of the Google CloudSpeech recognizer."""

import os

import aiy.audio

import aiy.cloudspeech

import aiy.voicehat

def main():

recognizer = aiy.cloudspeech.get\_recognizer()

recognizer.expect\_phrase('turn off the light')

recognizer.expect\_phrase('turn on the light')

recognizer.expect\_phrase('blink')

recognizer.expect\_phrase('repeat after me')

button = aiy.voicehat.get\_button()

led = aiy.voicehat.get\_led()

aiy.audio.get\_recorder().start()

while True:

print('Press the button and speak')

button.wait\_for\_press()

print('Listening...')

text = recognizer.recognize()

if text is None:

print('Sorry, I did not hear you.')

else:

print('You said "', text, '"')

if 'turn on the light' in text:

led.set\_state(aiy.voicehat.LED.ON)

elif 'turn off the light' in text:

led.set\_state(aiy.voicehat.LED.OFF)

elif 'blink' in text:

led.set\_state(aiy.voicehat.LED.BLINK)

elif 'repeat after me' in text:

to\_repeat = text.replace('repeat after me', '', 1) #<- 2 single quotes, not a double

aiy.audio.say(to\_repeat)

elif 'goodbye' in text:

os.\_exit(0)

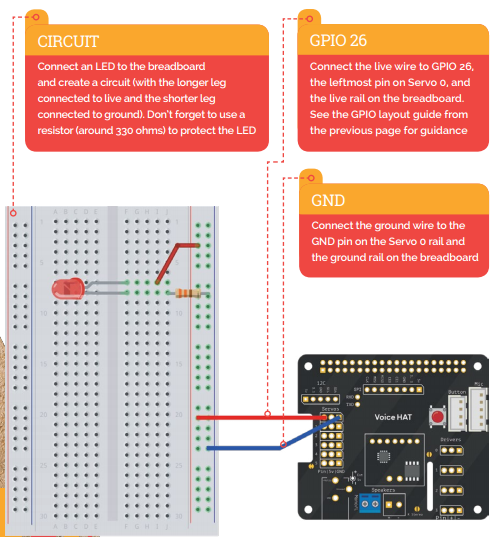
if \_\_name\_\_ == '\_\_main\_\_':

main()

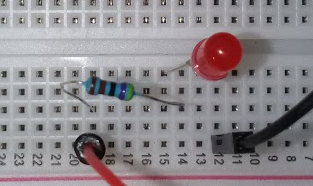
You may add more voice commands. Several ideas include a "time" command to make it speak out the current time or commands to control your smart light bulbs.

## Lab 5. Controlling the external world

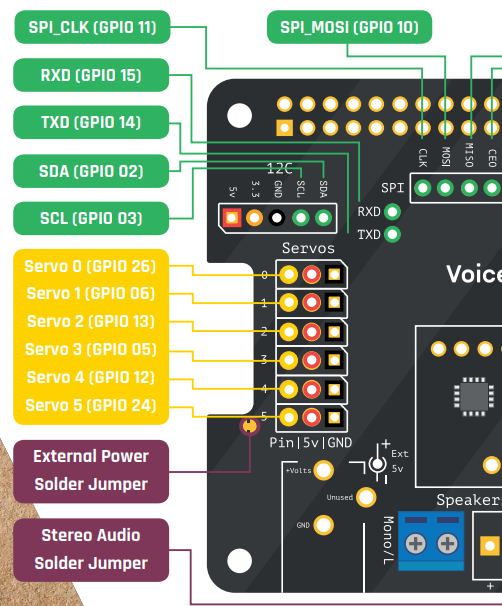
The circuit you will construct is diagrammed below. Don’t try to copy this – your breadboard is different:



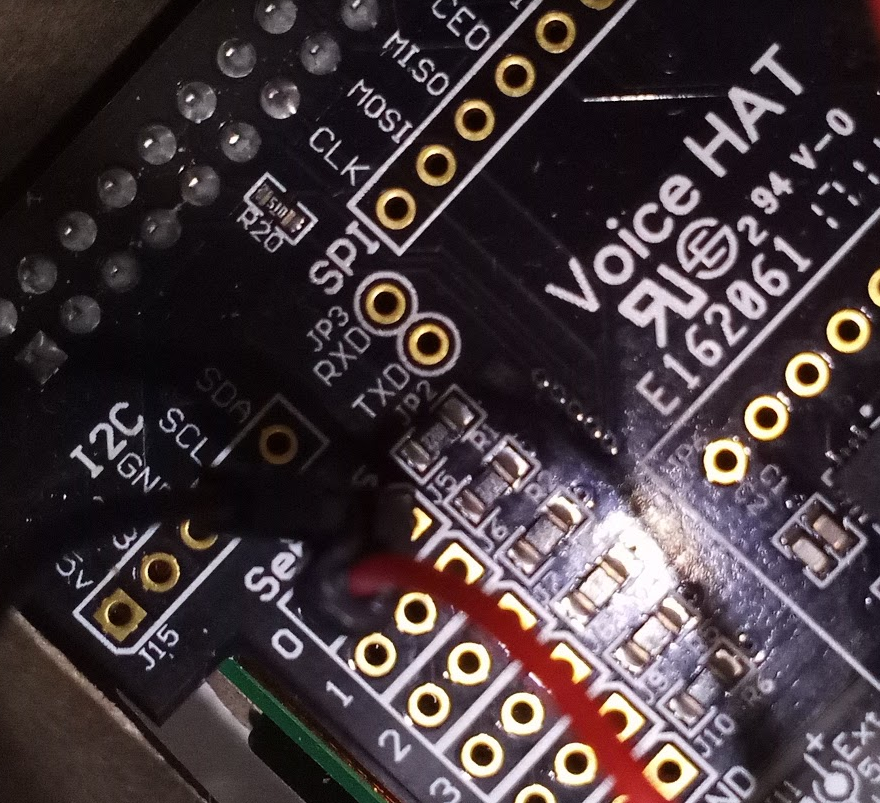
Your breadboard is cheaper and smaller, and doesn’t have the voltage and GND rails as shown (the columns the red and blue connectors are in). You’ll have to connect directly to the strips, as in the photo below:



The **\*short\*** leg of the LED is on the same track as the black wire – 12. The long leg and the resistor are in track 15. The red wire and the other end of the resistor are in track 19. The connections on the voice hat are to servo 0 on the voice hat. The black wire is to GND, the red wire is to Pin:



Putting the male connectors in the VoiceHAT does not result in a snug fit, but a contact is achieved:



You \*could\* write a program to control the led, like this (but don’t, until you’ve read the entire section):

#!/usr/bin/env python3

import aiy.audio

import aiy.cloudspeech

import aiy.voicehat

import RPi.GPIO as GPIO

def main():

recognizer = aiy.cloudspeech.get\_recognizer()

recognizer.expect\_phrase('turn on the light')

recognizer.expect\_phrase('turn off the light')

button = aiy.voicehat.get\_button()

aiy.audio.get\_recorder().start()

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(26,GPIO.OUT)

while True:

print('Press the button and speak')

button.wait\_for\_press()

print('Listening...')

text = recognizer.recognize()

if text is None:

print('Sorry, I did not hear you.')

else:

print('You said "', text, '"')

if 'turn on the light' in text:

GPIO.output(26,GPIO.HIGH)

elif 'turn off the light' in text:

GPIO.output(26,GPIO.LOW)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**or** you could just change the \_GPIO\_LED pin out value in /src/aiy/voicehat.py to 26:

|  |
| --- |
|  |
| import aiy.\_drivers.\_button |
|  | import aiy.\_drivers.\_led |
|  | import aiy.\_drivers.\_status\_ui |
|  |  |
|  | # GPIO definitions (BCM) |
|  | \_GPIO\_BUTTON = 23 |
|  | \_GPIO\_LED = 25 <-change this to 26 |
|  |  |

Save it and then run the cloudspeech\_demo.py program again.

## OPTIONAL- RUN YOUR APP AUTOMATICALLY

This section is included for interest.

Imagine you have customized an app with your own triggers and the Google Assistant library. It is an AIY-version of a personalized Google Home. Now you want to run the app automatically when your Raspberry Pi starts. All you have to do is make a system service (like the status-led service mentioned in the user's guide) and enable it.

Assuming that your app is src/my\_assistant.py. We would like to make a system service called "my\_assistant". First, it is always a good idea to test your app and makes sure it works to your expectation. Then you need a systemd config file. Open your favorite text editor and save the following content as my\_assistant.service:

The config file is explained below.

| **LINE** | **EXPLANATION** |
| --- | --- |
| Description= | A textual description of the service. |
| ExecStart= | The target executable to run. In this case, it executes the python3 interpreter and runs your my\_assistant.py app. |
| WorkingDirectory= | The directory your app will be working in. By default, we use /home/pi/AIY-projects-python. If you are working as a different user, please update the path accordingly.   Note shortcuts files do not support $HOME, so you have to explicitly use /home/pi/. |
| Restart= | Here we specify that the service should always be restarted should there be an error. |
| User= | The user to run the script. By default we use the "pi" user. If you are working as a different user, please update accordingly. |
| WantedBy= | Part of the dependency specification in systemd configuration. You just need to use this value here. |

For more details on systemd configuration, please consult its [manual page](https://www.freedesktop.org/software/systemd/man/systemd.service.html).

We also need to move the file to the correct location, so systemd can make use of it. To do so, move the file with the following command:

sudo mv my\_assistant.service /lib/systemd/system/

Now your service has been configured! To enable your service, enter:

sudo systemctl enable my\_assistant.service

Note how we are referring to the service by its service name, not the name of the script it runs. To disable your service, enter:

sudo systemctl disable my\_assistant.service

To manually start your service, enter:

sudo service my\_assistant start

To manually stop your service, enter:

sudo service my\_assistant stop

To check the status of your service, enter:

sudo service my\_assistant status