

Smart Home Assistant

In this project, you'll make a "smart" personal assistant to help out around some tasks around the house, such as turning lights on and off, or turning fans on and off.

You'll give your assistant a command in the same way you might talk to a person, and see how well it responds to your request.

What you'll need to know:



How to use the Python shell



How to write Python scripts

Part 1: What can our assistant do?

Instead of using real light bulbs and appliances, we're going to build a **prototype** that will **1)** wait for a command, **2)** decide which task to perform, and **3)** tell you what it did.



A **prototype** is a simpler version of your project that lets you test out an idea

1. Create a new python script, and save it as `smart_home.py`
2. Add the following code. Before running it, **predict** what you think it will do!

```
1 command = input("Enter your command: ")
2
3 if command == "Turn on the lights":
4     print("Ok, turning the lights on.")
5
6 elif command == "Turn off the lights":
7     print("Ok, turning the lights off.")
8
9 else:
10    print("Sorry, I don't understand!")
0
```

Can you guess what this **algorithm** will do when you run it?

Try reading it aloud, line-by-line, and discuss your predictions w/ your partner.

What do you think the **if**, **elif**, and **else** statements are for?

3. Run your script. In the **shell**, you should see: Enter your command:
4. Type in a command such as "Turn on the lights" and press *Enter*.
5. Keep trying different commands (you'll need to run the script each time)



Part 2: Adding more conditions

Right now, our assistant can only control lights. What if we also want to control a fan? We'll need to program our assistant to handle more **conditions**.

6. Add another condition to the if-statement (start after [line 8](#)):

```
1  command = input("Enter your command: ")
2
3  if command == "Turn on the lights":
4      print("Ok, turning the lights on.")
5
6  elif command == "Turn off the lights":
7      print("Ok, turning the lights off.")
8
9  elif command == "Turn on the fan":
10     print("Ok, turning the fan on.")
11
12  elif command == "Turn off the fan":
13     print("Ok, turning the fan off.")
14
15  else:
16     print("Sorry, I don't understand!")
```

Tip: it's ok to copy-and-paste! Copy a line, paste it below, and change whatever you need.

□ Add new conditions here for "Turn on the fan" and "Turn off the fan"

7. Run the script and test it out! Make sure all 4 conditions work.

How smart is our assistant, really?

What happens when you say "Turn the lights on" instead of "Turn on the lights"?

8. Run the script, and enter both commands to see what happens:

```
>>> %Run smart_home.py
Enter your command: Turn on the lights
Ok, turning on the lights.

>>> %Run smart_home.py
Enter your command: Turn the lights on
Sorry, I don't understand!
```



Reflect: why do you think this happens? What might you do to fix it?

Part 3: How to train your robot

As you may have realized, there are *many* different ways you could ask someone to turn on and off the lights. Adding all the possible commands in our code would take forever!

Next, we'll try a better approach: teaching the computer to recognize commands for itself. This technology is called **Machine Learning**



Create a new project and add training data

1. Learn how to create a new project in the [Machine Learning Reference Guide](#). Refer to the **Adding training data** and **Recognizing text** sections in the following steps.
2. Create a new project for **recognizing text**, and add training data for each possible outcome: `lights_on`, `lights_off`, `fan_on`, & `fan_off`.
3. Try to think of a bunch of the different ways you might tell someone to turn on the lights. Be creative!
4. Click on *Add example* to add at least 6 to 8 example commands for each of your labels:



5. Click the *Back to Project* link, then click *Learn & Test*.
6. Scroll down to the bottom, and click "*Train new machine learning model*". Wait for the training to complete (this might take a minute or two).
7. Once the training has completed, you will have a **machine learning model**. Try typing some test commands in the text box below to test out your model!

A **machine learning model** is the result of training.



You can interact with a model it by giving it input (such as text, images, or numbers), and it will give you some answer based on what it has learned.

Update your Python code

8. Highlight the **API key** text on your project page, and press *Ctrl+C* to copy it.
9. Go back to your Python script, and change your code to use the new **model**:

<pre>1 import ml4k 2 3 API_KEY = "PASTE-API-KEY-HERE" 4 model = ml4k.Model(API_KEY) 5 6 command = input("Enter your command: ") 7 result = model.classify(command) 8 label = result["class_name"] 9 10 if label == "lights_on": 11 print("Ok, turning the lights on.") 12 13 elif label == "lights_off": 14 print("Ok, turning the lights off.") 15 16 elif label == "fan_on": 17 print("Ok, turning the lights off.") 18 19 elif label == "fan_off": 20 print("Ok, turning the lights off.")</pre>	<ol style="list-style-type: none">1. Add <code>import ml4k</code> at the top2. Paste your API key using <i>Ctrl+V</i>3. Use the <code>classify</code> function to send the command to your model, and store the result in a variable called <code>result</code>.4. Create a variable called <code>label</code> and set it to <code>result["class_name"]</code>. This tells you which of the four "buckets" the model chooses.5. Replace <code>command</code> with <code>label</code>, and use the label names you created in your training data.6. Remove the <code>else</code> section. The model will <i>only</i> return one of the four options.
---	--

10. Run your script! Try entering commands that you **didn't** enter during training. How well did it do? If you got an unexpected answer, try adding more training data.

Advanced: How confident is our assistant?

What happens if you enter the command "Turn up the music! "? You'll notice the model will still give you one of the four options. Sometimes, the model will give you an answer, but it won't be very confident about it. We need a way to tell our user when we don't know what to do with a certain command.

Use `result["confidence"]` to get a number between `0` and `100`, and store it in a **variable** called `confidence` that will tell you how confident the model is in its answer. See if you can add an **if statement** that *first* checks the confidence level and display an appropriate response. For help, refer to the **Variables**, **Boolean Expressions** and **If Statements** chapters in your [Python Reference Guide](#).

Extra: Show an image instead of text!

1. Find some images online of a light bulb and fan on and off.
2. Add `import webbrowser` at the top of your script
3. Instead of `print()`, use `webbrowser.open("url-goes-here")` to show the image!