FutureMakers Workshop

Artificial intelligence (AI) and machine learning (ML)

Part B

Using artificial intelligence (AI) and machine learning (ML) services

Google Colab version (online)

Version 1.0.1

This guide will help you complete the FutureMakers workshop.
Follow along at your own pace.

Your name: _____

*Don't forget: programming can be difficult, but it's fun — **you can do it**!*

*Don't be afraid to **make mistakes** - mistakes are how we learn.*

*Don't be afraid to **ask questions** - the instructors want to help you.*

Introduction

Welcome to the FutureMakers workshop. We'll go through a series of activities designed to introduce you to programming, machine learning, and artificial intelligence. Don't worry if you don't have much experience with coding, or even if you've never ever programmed before. This workshop will show you how to code in Python (Part A), how to use exciting tools like face detectors or automatic language translation (Part B), and how to build your own interactive website (Part C).

Throughout this workshop, remember that programming can be difficult - but it's also very fun. Don't be afraid to make mistakes and ask questions - making mistakes is how we learn, and even the most professional programmers frequently search on the Web for things they don't understand. Programming isn't about knowing all the answers - it's about knowing how to look for them, often by doing a Web search or asking a friend or colleague.

Chatbots and AI with the Google Colab notebook

Google Colab (short for Colaboratory) is an online programming toolbox. It lets you use online workspaces called **notebooks**.

Notebooks save your work automatically. You can log on from another computer at a different time, and all the code you wrote will still be there.

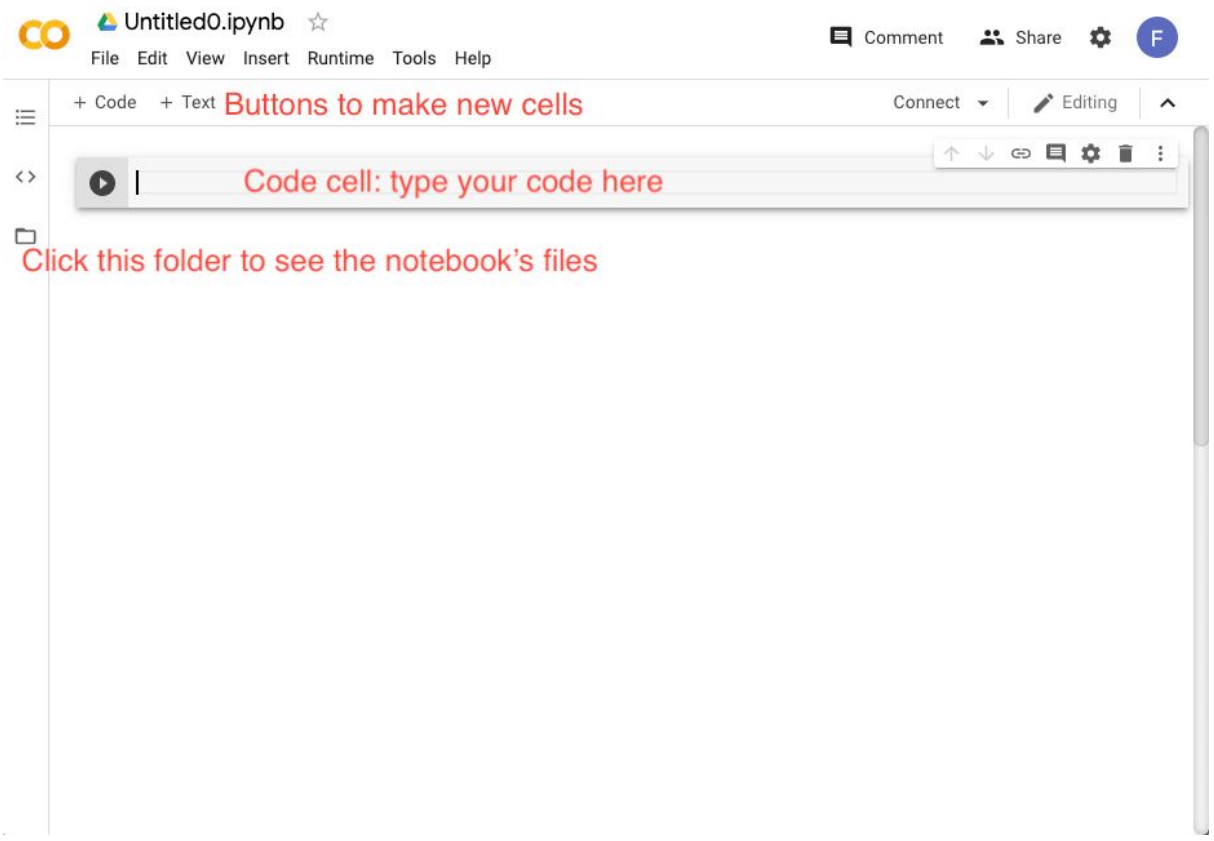
More than one person can work on a notebook at the same time! This makes working together on coding projects easier.

Opening Colab

To get started, search for Google Colab or go to https://colab.research.google.com. Click on New Notebook. (You will need to be logged in with a Google account).

An empty Colab notebook looks like this:

Remember: we learn by making mistakes, and it's a good idea to ask questions!



Colab is a fantastically useful environment for developing and running Python code.

It is very similar to the Jupyter notebook - in fact, it is an online version of the Jupyter notebook. The advantage is that you don't have to install anything.

Code lives in small areas called **cells**. To make a new cell, click the Code button on the top left:

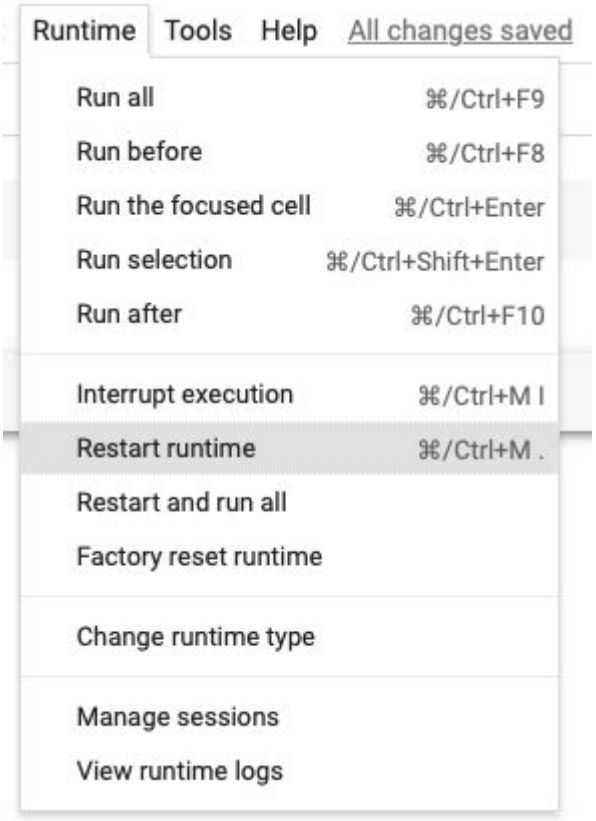
You can make as many cells as you want. Each one can contain a small amount of code, or lots!

To run the code in a cell, press shift and enter at the same time. You can also press the Play button to the left of the cell:

Remember: we learn by making mistakes, and it's a good idea to ask questions!

Restarting the kernel

If the kernel gets stuck, restart it using the menu:



Once you restart the kernel (or open the notebook afresh), all your code and output from the last time will be there, but there will be no variables or functions present in memory; you have to run the code again to get anything done.

It's a good idea to organise your notebook so that the cells can be run naturally in order.

Exercise: running code in the notebook

- Open a new notebook.
- Click in the first cell.
- Click the + Code button to create a new call. Do this a few more times so that you have plenty of cells available.
- Type some code in the first cell; printing something out is a good idea.
  Try `print('Hello!')`

Remember: we learn by making mistakes, and it's a good idea to ask questions!

- Click Play or Press shift+enter to run the first cell:

Google Colab notebooks connect to a server which actually runs your Python code and sends you back the results. This Python processor is called the kernel.

The notebook does not connect to the kernel until you run your first cell. When you do, you will see the status message in the top left changing:

- First it will say Allocating
- Then it will say Connecting
- Then it will say Initialising
- Finally, it will say Connected. You are now connected to the kernel and your code is being run.

Once your cell has run, you will be able to see its output just below it:

You can run any of the examples from Part A on Colab, because Colab is essentially a more powerful version of CodeSkulptor. The programs are written in the same language, Python.

Exercise: starting and stopping the kernel

What does it mean? The Python kernel is simply Python itself: the program which reads your Python code and runs it.

Type the following code into a notebook cell:

```
while True:
      print("I'm still running!")
```

This code will run forever and won't stop on its own.
Run the cell (shift+enter, or click Play).

Observe that Python is busy (you can see the Stop button rather than the Play button), and that lots of text is being printed out:

Now stop the kernel, either by using the Stop button or by using the menus: Runtime > Interrupt Execution. Now you know how to stop the kernel if you need to later.

Exercise: defining some variables

Make sure you have four empty cells before starting this exercise.
Start by restarting the kernel (Runtime > Restart runtime). This makes sure that Python is ready and that there are no variables left over from previous code.

Now fill out the cells like this:

Type this in the first cell	`alices_dog = 'Rover'`	Run this cell now
Type this in the second cell	`bobs_dog = 'Fido'`	*Don't run this cell*
Type this in the third cell	`print(alices_dog)`	Run this cell now
Type this in the fourth cell	`print(bobs_dog)`	Run this cell now

You will notice that `print(alices_dog)` works and prints out 'Rover'.
However `print(bobs_dog)` doesn't work: there's a bug. This line causes an error:

NameError: name 'lol' is not defined

What happened here? Python knows about the variable `alices_dog`, *because we ran that cell.* But Python doesn't know about the variable `bobs_dog`, *because we didn't run that cell.*

You have to run a cell for the code in it to work. Otherwise, the code will have no effect.
You can run a cell more than once, and you can run it again whenever you want to.

What does it mean? Functions

In programming, we often want to do the same thing again and again. For this, we use functions. A function is like a little machine that takes an input, does something to it, and returns an output.

Some examples of functions could be:

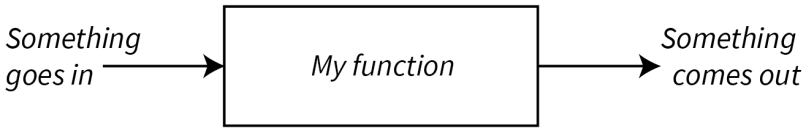
- A function to return TRUE if a number is less than 10, and FALSE otherwise;
- A function to add two to a number;

Remember: we learn by making mistakes, and it's a good idea to ask questions!

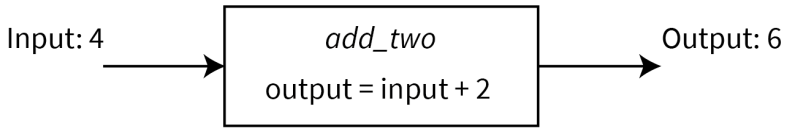
- A function to capitalise a word;
- A function to check whether a date is a Wednesday.

You can imagine functions as re-usable boxes which can process inputs and turn them into outputs, like this:

1. What a function looks like generally



2. Example function which adds two to a number



Exercise: making a function

Start this exercise by restarting the kernel (Kernel menu > Restart).

Let's make a function - a piece of code that we can re-use multiple times by giving it a name.

Type the following in a cell, but don't run the cell yet.

```
def add_100(number):
      return number + 100
```

```
[7]  def add_100(number):
           return number + 100
```

Tip: don't forget to indent the line which is inside the function.

A function is like a machine that does something useful for us. Something goes in, and something goes out. This code defines a function that takes a number, adds 7 to it, and gives us back the result.

A function in python will look like this template:

```
def my_python_function(input_variable_1, input_variable_2):
      output = input_variable_1, input_variable_2
      return output
```

This template just adds two input variables together, but you can do whatever you like in a function. Functions can have any number of input variables, or none. The keyword **def** is short for *define* - it shows that we're defining a new function. The keyword **return** is what we use to send output back from the function.

Now, still without running the first cell, let's try to use our function in another cell. Find or make an empty cell and type the following code:

```
print(add_100(7))
```

This line is asking Python to take 7, add 100 to it, and print the result.

Now try to run the cell. There's a bug - we get another error:

```
NameError: name 'add_100' is not defined
```

Why is this happening? As before, it's because we haven't run the cell with the function in it. So go back and run the cell in which you typed the function. Then run the cell which contains print(add_100(7)). This time, it will work, and print out 107.

Your turn

Define another function which does something interesting.
Tip: a function doesn't have to take an input value, and it doesn't have to return anything. For example:

Code for the function	**Notes**	**How you use the function**	**What you see on the screen when you use the function**
`def say_hello():` `  print('Hello!')`	Doesn't take any input Doesn't return anything (just prints Hello)	`say_hello()`	`Hello!`
`def say_hello_to(name):` `  print('Hello, '+ name + '!')`	Takes input Doesn't return anything (just prints a greeting)	`say_hello_to('Alice')`	`Hello Alice!`

`def my_favourite_singer():` `  return 'Justin Bieber'`	Doesn't take any input Returns a name	`print(my_favourite_singer)`	`Justin Bieber`
`def double(number):` `  return 2 * number`	Takes input Also returns something (twice the original number)	`print(double(7))`	`14`

Using Amazon artificial intelligence (AI) services with the Jupyter Notebook

Let's do something exciting - using state-of-the-art artificial intelligence services to translate and speak text and to recognise people and objects. Most of the tasks we are about to accomplish were impossible five years ago!

Exercise: importing the FutureMakers library

Start a new notebook - it's good to have a clear, fresh working space.

One of the best things about Python is that it's easy to use other people's code.

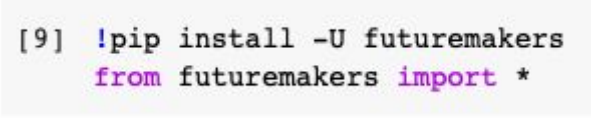
What does it mean?
A **library** is a program which is meant to be used by others. For example, there are Python libraries for displaying images on the screen, for finding out the weather anywhere in the world, or for making a series of images into an animation such as a GIF. Python makes it very easy to use the thousands of libraries written by other Python users. Here we will use the **futuremakers** library, which was made by the FutureMakers instructors to help you use AI and machine learning.

Let's start by importing the **futuremakers** library. This is a file called **futuremakers.py** which should be in your **python** directory inside the FutureMakers folder.

Once you've opened a new notebook, type the following line in a cell, and run the cell.

```
!pip install -U futuremakers
from futuremakers import *
```

The cell should look like this:



You should see some output that looks like this:

If you don't, ask for some assistance.

What does it mean? *
The star symbol * is often used in programming to mean "everything". So this line means *From the futuremakers library, import everything*.

Next, we need to set up our username and password for the Amazon Web Services account we will be using to access machine learning services.

The demonstrators will let you know the username (key) and password (secret key). Enter them like this, then run the cell:

```
[12]  set_access_key('567578675645645576')
      set_secret_key('897685675797890789786546543676565 6')
```

Now let's test that our import worked. Type out this line in a new cell, and run it:

`test_futuremakers()`

The cell should look like this:

```
[16] test_futuremakers()

     The FutureMakers library is imported and ready to use.
```

This asks Python to find the test_futuremakers function (which is inside the futuremakers module) and run it. You should see the message *The FutureMakers library is imported and ready to use.* This means everything is ready to proceed with the workshop.

If you see an error, ask an instructor for help.

If you're interested in seeing the Python code for the FutureMakers library, just visit the project Github, where the code is stored: https://github.com/fusionlove/futuremakers. You will need to search for the futuremakers.py file. There's a lot of code - about 100 lines -

and most of it deals with using the Internet to make requests to Amazon's servers, which do most of the work.

Exercise: translating text

Now we're ready to use some of the other functions in the FutureMakers library. Let's translate some text.

Type out this line in a new cell, and run it:

```
translate_english_to_spanish('I have a very good dog.')
```

You should see the output

```
'Tengo un perro muy bueno.'
```

What happened here?
The FutureMakers library used the Internet to make a request to Amazon's servers (either in London or somewhere in the USA). Just like when you access a web page, the library sent your sentence to the Amazon servers and asked them to translate it into Spanish. The Amazon servers sent back the response, and the library returned it to you. Finally, the Jupyter notebook displayed it as text.

Your turn
Try translating some different text from English to Spanish.
What happens if you enter some text that is already in Spanish? What happens if you enter some text that isn't in English *or* Spanish?

Exercise: translating different languages

To translate between different languages, we need to tell Amazon Translate which language our text is currently in, and which language we want to translate to. We do this using language codes, such as **en** for English, **fr** for French, and **es** for Spanish (*español*).

For example, we can translate from English to French:

```
translate('en', 'fr', 'I have a good dog')
```

```
"J'ai un bon chien."
```

...then from French to Spanish:

```
translate('fr', 'es', "J'ai un bon chien.")
```

```
'Tengo un buen perro.'
```

...then from Spanish back to English:

```
translate('es', 'en', 'Tengo un buen perro.')
```

```
'I have a good dog.'
```

Try this out for yourself. If you use a more complex sentence, Amazon might get it wrong!

Your turn
Translate a sentence from a language you know into a language you don't know as well, then translate it back. Use the table below to look up language codes. If a language doesn't appear in this table, it means Amazon Translate can't use it yet.

Arabic ar	Italian it
Chinese (Simplified) zh	Japanese ja
Chinese (Traditional) zh-TW	Korean ko
Czech cs	Malay ms
Danish da	Norwegian no
Dutch nl	Persian fa
English en	Polish pl
Finnish fi	Portuguese pt
French fr	Russian ru
German de	Spanish es
Hebrew he	Swedish sv
Hindi hi	Turkish tr
Indonesian id	

Exercise: speaking text out loud (synthesising speech)

We'll now use another artificial intelligence service related to language: speech synthesis. We will ask Amazon to generate an audio file for us from whatever text we want; then we'll play the file, just as if the Amazon service could actually speak. This uses an Amazon service called **Polly**.

Check your volume is turned up, then ype this into a new cell, and run it:

```
speak('Hello, FutureMakers!')
```

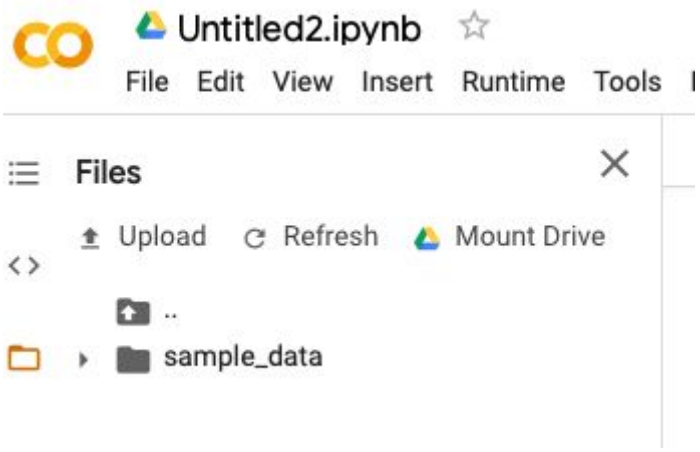
After a short delay, an audio player will open and play your synthesised speech. This audio was just generated by powerful machine learning systems inside one of Amazon's buildings.

If you want to play the file again, you can just call the **speak** function again. If you want to find the audio file (for example, to send it to a friend), it is in the **polly** folder; every time you run the **speak** function, a new audio file is saved in this folder.

Your turn: try asking Amazon to say something longer. Experiment with full stops and commas to generate pauses in the speech.

Exercise: adding files to Colab

Each Colab notebook has its own area where we can store files. You can see this by opening the Files button on the left:



When we open the filesystem, it already contains some sample data from Google; we will not be using this.

We will need to load some files into Colab to work with - especially images. The best way to do this is by using the Upload button, which you can see in the image above.

Exercise: finding, saving and showing images

For the next few exercises, we'll be working with images. The best way to do this is to find images on Google Images, then save them on your desktop, then upload them to the Colab notebook as shown above.

Let's try saving an image now.

- Open Chrome or Firefox and go to www.google.com (or use the search bar).
- Search for something (an animal or a famous person are good places to start).
- Once you're looking at the search results, click on Images.
- Click on one of the images so that it gets larger than all the others.
- **In Chrome**, right-click on the image and click **Open image in new tab**. The aim is to get the image open in a window all by itself; if this doesn't happen immediately, you might need to right-click on the image again and select **Open image in new tab** again. Then right-click on the image again and do **Save image as**.
- **In Firefox**, right-click on the image and click **View image**. The aim is to get the image open in a window all by itself; if this doesn't happen immediately, you might need to right-click on the image again and select **View image** again. Then right-click on the image again and do **Save image as**.
- When you do **Save image as**, the browser will ask you where you want to save the image. Select your desktop.

When you save the image, give it a nice name like cat.png or cat.jpg (keep the extension), not a difficult-to-use name like 8978_cat_small_96.jpg.
- In Windows (Finder in OS X), double-click on your image on the desktop to check that it's there.
- Finally, upload the image to Google Colab using the Upload button on the filesystem drawer on the left of the screen. Upload the image into the **images** folder in your filesystem. You may need to open this folder first so that you can upload into it.

For the following exercises, it's best to use medium-size images - not so small that they don't have much detail, but not so big that they can easily fill the whole screen.

Getting images ready

The following exercises assume that you have the three following images in your **images** folder:

- `friends.jpg`
- `mike_wilks.jpg`
- `patrick_stewart.jpg`

To get ready, you can either same some images with these names, or get your own images and insert the right names when you do the exercises. Remember to choose short, convenient names for your images.

Exercise: recognising celebrities

We'll now use Amazon services to try to recognise celebrities. Let's try first with an image which is already in the Futuremakers/images folder: friends.jpg.

Let's look at the image first. In a new cell, type this line and run it:

```
show_image('images/friends.jpg')
```

This will show the image just below the cell.

Now let's ask Amazon to recognise the people in this image. In a new cell, type this line and run it:

```
find_celebrities('images/friends.jpg')
```

After a short delay, you should see the following line:

```
Amazon Rekognition found 4 celebrities in images/friends.jpg: David Schwimmer, Matt LeBlanc, Lisa Kudrow, and Courteney Cox.
```

Amazon hasn't done the job perfectly: it's only recognised four people, missing out Jennifer Aniston and Matthew Perry.

Your turn: find an image of a more modern celebrity (or one containing more than one person), save it in the **images** folder, then use find_celebrities to try and detect the famous person or people. You can use show_image to look at the image in the Jupyter notebook, too.

Why not try the (supplied) images cage.jpg and patrick_stewart.jpg? Why does cage.jpg take such a long time to process?

Exercise: detecting faces

Amazon Rekognition can also detect faces in an image - not just celebrities' faces, but any faces - and estimate their ages, their expressions, and (interestingly) whether they have beards.

Let's try first with a sample image. Let's look at it first; Type this line in a new cell and run it:

```
show_image('images/patrick_stewart.jpg')
```

To detect faces, type this line in a new cell and run it:

```
detect_faces('images/patrick_stewart.jpg')
```

You should see the following output:

```
Amazon Rekognition found 1 face in this image.
I think this person is between 60 and 80 years old. I am 100 percent sure that they do
not have a beard. I am 68 percent sure that they are feeling happy.
```

Your turn: choose a new image from Google Images, save it to the FutureMakers/images folder with an easy-to-remember name, view it in the Notebook, then try to detect faces. See if you agree with Amazon Rekognition! You can try images containing more than one person, too.

Exercise: detecting objects

Amazon Rekognition can also detect objects in an image. Let's try this out with a difficult image containing lots of objects.

To show the image, type this line in a new cell and run it:

```
show_image('images/mike_wilks.jpg')
```

This is an image from a book called The Ultimate Alphabet by Mike Wilks. It contains many objects beginning with the letter S.

Now, to detect objects, type this line in a new cell and run it:

```
detect_objects('images/mike_wilks.jpg')
```

Amazon Rekognition should find about 20 or 30 objects. However, they don't all begin with S! Amazon Rekognition is doing its best to describe the objects in the image.

Your turn: choose a new image from Google Images, save it to the FutureMakers/images folder with an easy-to-remember name, view it in the Notebook, then try to detect objects. See if you agree with Amazon Rekognition! How good is it at telling the difference between similar things, like different kinds of dog?

Exercise: connecting AI services together

We've now tried out each of the artificial intelligence and machine learning services we'll be using:

- Translation
- Speech synthesis
- Celebrity detection
- Face detection and emotion estimation
- Object recognition

Now let's try to combine them together. As an example, we'll make a function which asks Amazon first to translate a sentence to English, then to speak it.

Type this function into a cell, and run it (nothing will happen when you run it - you will have to call the function from another cell):

```
def translate_and_speak(french_sentence):
  english_sentence = translate('fr', 'en', french_sentence)
  speak(english_sentence)
```

Make sure to get the indentation right. Use the tab key to make an indent; both of the lines inside the function should be indented.

This function first translates a sentence from French into English, then speaks it. Let's make a French sentence to try it out:

```
translate('en', 'fr', "I'm cold")
```

Output below the cell will be:

```
"J'ai froid."
```

Tip: there is an apostrophe in the sentence "I'm cold". This means that we have to use double quotes around this sentence, because if we use single quotes, Python will confuse the apostrophe for a single quote.

Now we can copy and paste the French sentence into a call to our new function. This will translate the sentence back to English and then speak it:

```
translate_and_speak("J'ai froid")
```

Your turn: modify the previous example so that it translates from a different language to English. Then try it out.

Exercise: connecting two more services

In this example, we'll ask Amazon to speak the objects it finds in an image.

Let's make a new function first. Type this code in a new cell, and run it. This function first detects objects, then speaks the results.

```
def detect_objects_and_speak(image):
  objects = detect_objects(image)
  speak(objects)
```

Now let's try it out. First, find an image online with only one or two objects - we don't want the description to take too long to speak. Save it in the images folder with an easy name. Then run, for example:

```
detect_objects_and_speak('images/my_new_image.jpg')
```

Your turn
Think of another way in which to combine two services, and put them together. If you're not sure, don't hesitate to ask one of the instructors.

Exercise: ideation

- Work in teams.
- Have a look at the Sustainable Development Goals list: https://www.globalgoals.org.
- Think of a real-world problem you and your teammates would like to solve together.
- Do a bit of research to understand the problem better.
- Try to understand and empathise with the people that suffer from it the most.
- Brainstorm on how AI can help you solve this problem.
- Apply your superpowers: creativity, empathy, emotional intelligence, judgement, reasoning, communication, teamwork, and the AI skills you learned today.
- Think of how, where and in what form AI can be applied to make these people's lives better or save our planet.

Final exercise: presenting your work

- Introduce yourself
- Problem you are trying to solve with AI (SDG)
- What AI would you create to address this problem – your idea

Remember: we learn by making mistakes, and it's a good idea to ask questions!

- How your AI addresses the problem
- Any alternatives to your idea
- Any product design considerations to make your idea more accessible
- Any ethical concerns you have
- What you want to do next