FutureMakers Workshop
Day 1

Artificial intelligence (AI)
and machine learning (ML)

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

Your name: _____

THREE IMPORTANT RULES:

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

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

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

Remember: we learn by making mistakes, and you should ask as many questions as possible!

Part A
Starting to program (in Python)

CodeSkulptor is a website designed for learning to code in Python without having to set anything up on your computer. Open CodeSkulptor: https://**py3**.codeskulptor.org/. Or just search for CodeSkulptor Python 3. **Make sure you are using the Python 3 version, not the Python 2 version.**

CodeSkulptor has two sides:
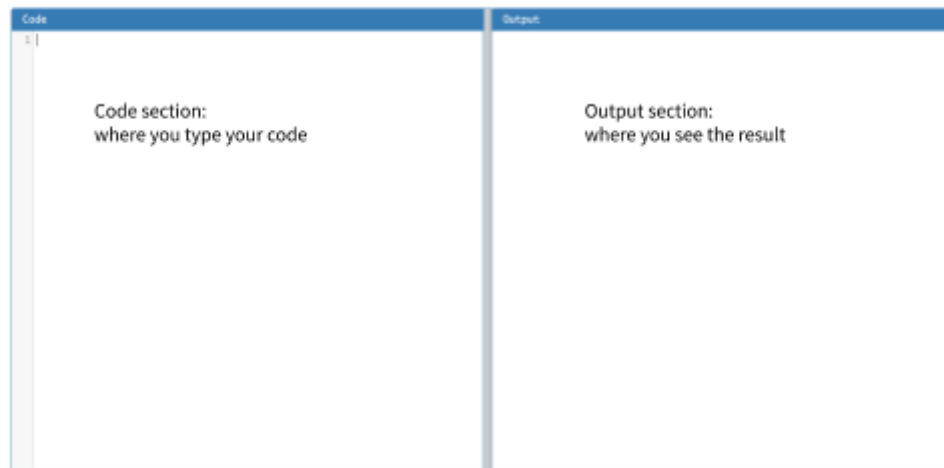
Start by deleting everything from the code side (on the left), so that it's empty.
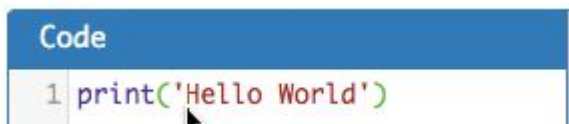
What does it mean?

The programmer is the person who writes the code.
The user is the person who runs the code.

For example, when you are playing a computer game, you are the user.
When you are testing your own program, you are the programmer *and* the user.

There are three steps to coding:

1. Write your code

Here you enter instructions in the Code box.

2. Run your code

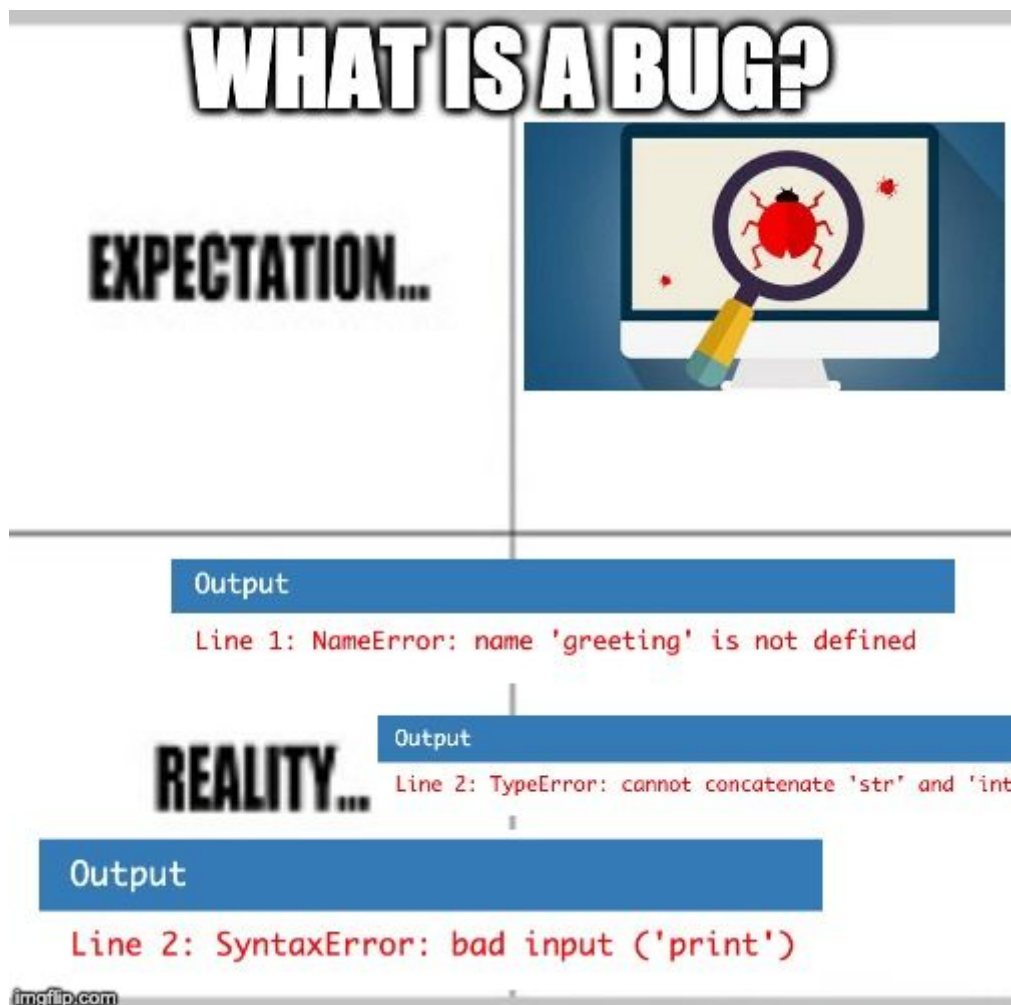*Remember: we learn by making mistakes, and you should ask as many questions as possible!*

You click the Run button:

▶ Run

...and CodeSkulptor will run your Python code and try to follow your instructions.

3. Debug

Did you write the right instructions for the computer to execute? Often, what actually happens is different from what we expect. We might find problems - called bugs - in our code.

Exercise: Running code

Make sure the Code window is empty, then enter the following program:

```
print('Hello World')
```

Remember: we learn by making mistakes, and you should ask as many questions as possible!

Then click Run.

Your turn
Modify the previous program to say something different.

Exercise: Asking the user a question

Enter the following program:

```
name = input('What is your name?')
greeting = "Hello " + name
print(greeting)
```

What does it mean?
A **variable** is a part of your program which can remember things.

Here we have two variables: name (which stores the user's name when they input it, such as 'Alex') and greeting (which stores the greeting, such as 'Hello Alex'). We used the plus (+) to stick the two parts of the greeting ('Hello ' and 'Alex') together. The plus (+) can be used to join words together as well as to add numbers.

Your turn

Change the last program to say Goodbye to the user.

Question: why is there a space after "Hello"? What happens if we take this space out? Take it out and run the program again.

Exercise: doing things many times

Before you start, make sure the code window is empty. If you want to save your code from the previous exercise, you can open a new tab to get a fresh CodeSkulptor window (make sure to use the Python 3 version).
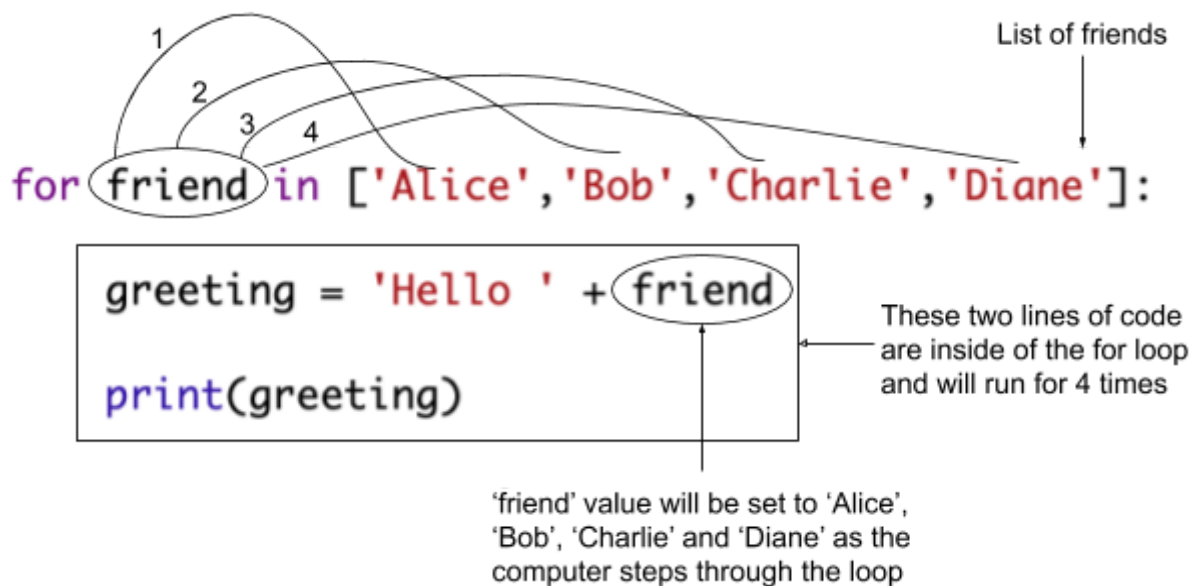
Type in the following program:

```
for friend in ['Alice', 'Bob', 'Charlie', 'Diane']:
    greeting = 'Hello ' + friend
    print(greeting)
```

Then click Run.

This is called a **loop**. We know it's a loop because it starts with the word **for**. Wherever you see **for**, you know something is going to happen several times.

Remember: we learn by making mistakes, and you should ask as many questions as possible!

Question: Why is there some blank space before some of the lines?
This blank space is called an indent. Every line with an indent is inside a loop.
Sometimes, CodeSkulptor will helpfully insert an indent automatically for you. If you want to make one yourself, press the tab key.

What does it mean?
A **bug** is a problem with your program. Sometimes bugs happen because you made a mistake. Sometimes they happen because another programmer made a mistake.

Bug: what happens if we forget the indent?

Delete the indents so that your program looks like this:

```
for friend in ['Alice', 'Bob', 'Charlie', 'Diane']:
greeting = 'Hello ' + friend
print(greeting)
```

Now run the program. In the output, you will see

```
Line 2: SyntaxError: bad input ('greeting')
```

This means that your program has a bug. Here, the bug is that there is no indent. Because there is a loop, Python is expecting an indented line; it doesn't find one, so it doesn't know what to put in the loop and shows us an error.

Your turn
Let's add some friends to the list.

This part of the program is a list of friends:

```
['Alice', 'Bob', 'Charlie', 'Diane']
```

Remember: we learn by making mistakes, and you should ask as many questions as possible!

You can see that each friend's name is in quotes, and they are separated by commas. Add some more names to the list. Don't forget the quotes and the commas. For example:

```
['Alice', 'Bob', 'Charlie', 'Diane', 'Jeremy']
```

Then run the program again.

Tip
Words which you want to print on the screen (like names) always need to be in quotes, like 'London'.

```
print('London')
```

However, variable names (such as greeting) do not need to be in quotes.

Exercise: doing something once, then some things repeatedly, then something once

Enter the following code:

```
name = input('What is your name?')

for thing_to_say in ['Hello', 'Have a nice day', 'Goodbye']:
    print(thing_to_say + name)

print('Have fun!')
```

Tip: if you have typed the loop line (the line starting with **for**) properly by ending it with a colon, CodeSkulptor will automatically indent the next line. If it doesn't, you can put the indent yourself by pressing the tab key.

What's happening here? The first line, which gets the user's name, is being run once. Then the line inside the loop, which says something to the user, is being run several times. Finally, the last line, which says 'Have fun!' is being run once.

Tip: a line which is indented might be in a loop, so might be run several times.
A line which is *not* indented is not in a loop, so will only be run once.

Your turn

Make your own loop and run it. It can do anything you want! You can get input from the user first if you want to, but you don't have to. You can even get input from the user *inside* the loop - this means you will get input several times.

Exercise: doing some maths

Before you start, make sure the code window is empty. If you want to save your code from the previous exercise, you can open a new tab to get a fresh CodeSkulptor window (make sure to use the Python 3 version).

Remember: we learn by making mistakes, and you should ask as many questions as possible!

There are many things we can do with Python. One of the simplest is to use it as a calculator.

- When we are programming, we use the plus (+) symbol to add numbers and the minus (-) symbol to subtract numbers.
- However, we don't use the times (×) symbol to multiply numbers, we use the star (*) instead.
- And we don't use the division (÷) symbol to divide numbers, we use the slash (/) instead.

Enter and run the following code:

```
print(14 * 21)
```

You can see that the answer is printed.

Your turn: enter two really big numbers (with 6 or 7 digits each) and multiply them. Python is much, much more powerful than your calculator!

Exercise: maths with variable names

We can use variables to store numbers with a particular name. Enter and run the following code:

```
number = 5
other_number = 10

total = number + other_number
print(total)
```

Bug: what happens if we try to glue a word to a number?

Enter the following code:

```
number = 5
other_number = 10

total = number + other_number
print('The total is ' + total)
```

Now try and run it. You will see an error:

```
Line 5: TypeError: cannot concatenate 'str' and 'int' objects
```

What does it mean? "**Concatenate**" just means "glue" or "stick together". Python is confused because we are asking it to glue a string (word - **str** is short for string) to an integer (a whole number).

Remember: we learn by making mistakes, and you should ask as many questions as possible!

Python is quite clever, but sometimes it doesn't know how to do obvious things. To fix this error, we just need to convert our integer variable into a string first.

Exercise: convert an integer to a string

Before you start, make sure the code window is empty. If you want to save your code from the previous exercise, you can open a new tab to get a fresh CodeSkulptor window (make sure to use the Python 3 version).

Enter and run the following code:

```python
my_integer = 7
my_string = '7'

print('This is the integer:')
print(my_integer)

print('This is the string:')
print(my_string)

print('This is the integer converted to a string:')
print(str(my_integer))

print('They all look the same!')
```
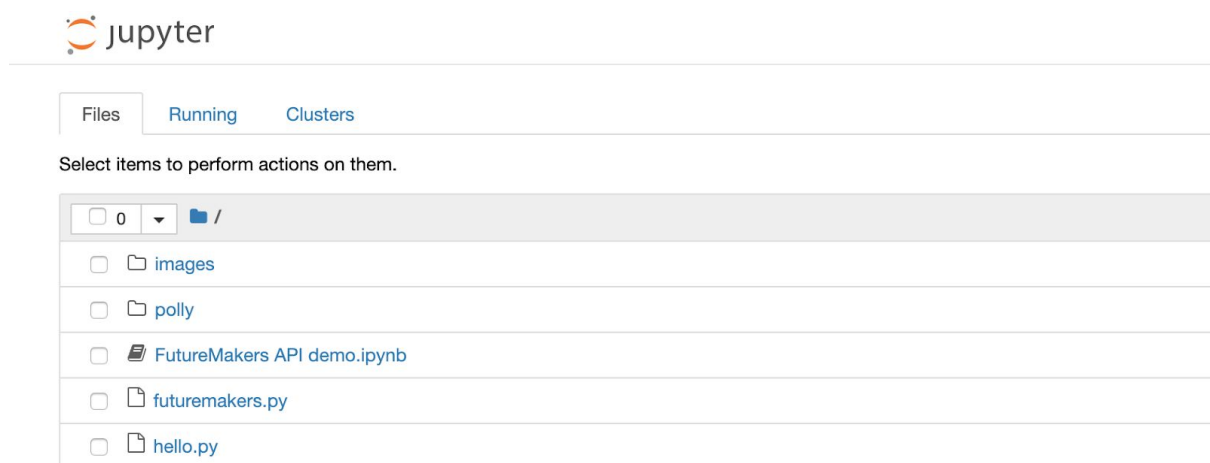
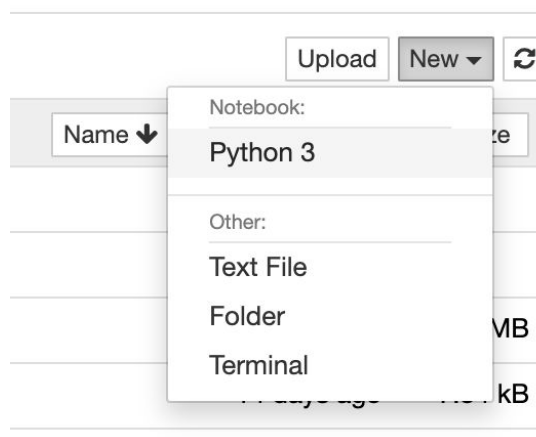*Remember: we learn by making mistakes, and you should ask as many questions as possible!*

Part B
Chatbots and AI with the Jupyter notebook

Open the Anaconda Navigator from the Start menu, and click on Notebook. This will open the Jupyter Notebook, a fantastically useful environment for developing and running Python code.
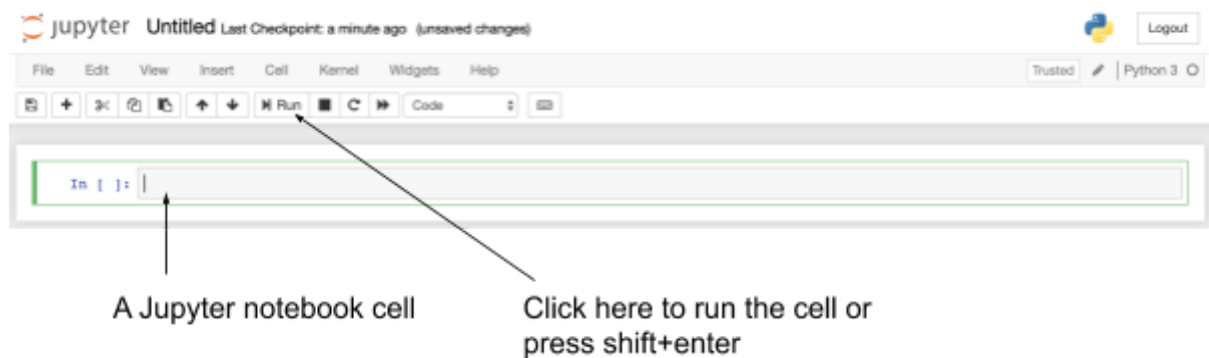
Make sure you navigate to the folder where the file named futuremakers.py is stored like this:

Click on the 'New' button on the upper right and select Python 3 notebook

Remember: we learn by making mistakes, and you should ask as many questions as possible!

A Jupyter notebook cell

Click here to run the cell or press shift+enter

Code lives in small areas called **cells**. Press escape (esc), then b to create a new cell under the current one. Press esc, then a to create a new cell above the current one.

To run the code in a cell, press shift and enter at the same time.

When the Python kernel is ready, the indicator in the top right looks like this:

When the Python kernel is busy, on the other hand, it looks like this:
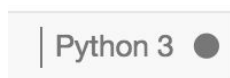
If the kernel gets stuck, restart it using the menu or by pressing ctrl-c. Once you restart the kernel (or open the notebook afresh), all your code and output from the last time will be there, but there will be no variables or functions present in memory; you have to run the code again to get anything 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.
- Press esc, then b to create a new cell after the current one. 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.
- Press shift+enter to run the first cell.

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.

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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).

Observe that Python is busy:

Now stop the kernel, either by using the Stop button or by using the menus: Kernel > Interrupt. 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 (Kernel menu > Restart). 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.

Exercise: making a function

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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
```


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 python_function (required_input):
    #do some operations with required_input
    desired_ouput=required_input + required_input
    return desired_output
```

def is short for *define* - it shows that we're defining a new function.

Now 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 'lol' 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

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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.

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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.

```
from futuremakers import *
```

```
In [1]: from futuremakers import *
```

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*.

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

```
test_futuremakers()
```

```
In [2]: 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 a welcome message is printed under the cell.

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

If you're interested in seeing the Python code for the FutureMakers library, just open futuremakers.py (you can do it from the Jupyter Notebook's folder page) and have a look. 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.

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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.

Your turn

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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**.

Type 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: 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 in the **images** folder inside the FutureMakers folder so that we can work with them easily.

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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. Find the **FutureMakers** folder (which is on your desktop), then find the **images** folder inside it, and save the image inside the **images** folder. 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, double-click on your image in the **images** folder to check that it's there.

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.

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.

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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?

Possible services to add to the list:
- Amazon Rekognition Facial Analysis, Unsafe content detection, text in images
- Amazon transcribe
- Amazon textract
- Text summarisation
- Text generation
- Image captioning

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: "J'ai froid."
```

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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.

Remember: we learn by making mistakes, and you should ask as many questions as possible!

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
- 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