FutureMakers Workshop **Day 2**

Artificial intelligence (AI) and machine learning (ML)

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.

Part C
Making a real interactive web app

In this part of the workshop, we'll build an interactive website and put it on the Web so that you - and anyone else - can access it. We'll use three tools to do this:

- **The command line**: an old-fashioned, but still very useful, way of asking your computer to do things
- **Heroku**: a company which takes care of running a web app for you. All you have to do is send Heroku the code for your web app, and it will make it available online using a web server.
- **Git**: a useful piece of software which looks after sending your code to other computers. Git is often used by programmers to work together on software. We will use it to send our code to Heroku.

What does it mean? A web server can mean a computer whose job it is to run websites. Every website needs a web server; this is the computer that your browser talks to when you open a website. Big websites, like Youtube, need hundreds of thousands of servers to be able to deal with all their visitors. If you don't have enough servers to run the website for all your visitors, your website will be slow, or even break.

Exercise: getting used to the command line

The command line is how we used to talk to computers before windows, files and folders. It works with text only. To tell the computer to do something, you write a command in the command line, and press return. The computer will then write some lines in the command window so that you can see whether your command worked.

To open the command line, select Anaconda Prompt from the Start menu:

Now let's try out some commands. If you're using a Windows laptop, use the commands from the Windows column; if you're using a Mac or a Linux computer, use the commands from the OS X/Linux/Unix column.

Here is a list of some common commands. Don't type these out yet.

Task	**Command**	**Windows example**	**OS X/Linux example**
Show the current folder (just like in Windows, a command line window is always *in* a particular folder)	`pwd` *or* `cd`	`pwd`	`cd`
List the files in the current folder	`dir` *or* `ls`	`dir`	`ls`
Change the current folder	`cd`	`cd /Users/alice`	`cd /Users/bob`
Make a new folder	`mkdir`	`mkdir my_new_folder`	`mkdir my_new_folder`

Let's try out some commands.

If you are using a Windows computer:

- Open an Anaconda Prompt window.
- Use cd to find out which folder you're currently in.
- Use dir to show the files in the current folder.
- Use cd TODO to move to the FutureMakers folder.
- Use dir to show the files in the FutureMakers folder.
- Use cd TODO to move to the my-web-app folder in the FutureMakers folder
- Use dir to show the files in the my-web-app folder

If you are using a Mac or Linux computer:

- Open an Anaconda Prompt window.
- Use pwd to find out which folder you're currently in.
- Use ls to show the files in the current folder.
- Use cd TODO to move to the FutureMakers folder.
- Use ls to show the files in the FutureMakers folder.
- Use cd TODO to move to the my-web-app folder in the FutureMakers folder
- Use ls to show the files in the my-web-app folder

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

Exercise: saving a file in Sublime Text

The Sublime Text icon

We won't be using the Jupyter Notebook for this part of the workshop: we will be editing our files in Sublime Text, which is a fast and useful program for editing text files.

Tip: you can't program in Word, or in any editor which supports *rich text* (fonts, colours, italics, bold, etc). For programming, we always use a *plain text* editor (no fonts, you can't change the colours, no italic or bold). There are plenty of plain text editors available; Sublime Text is one of the best.

To get started:

- Open Sublime Text from the Start menu
- Select Project -> Add Folder to Workspace.
  You will now be able to see everything in the FutureMakers folder in the folder bar on the left of Sublime Text.
- Make a new file and save it as test.txt (the .txt extension means that this file just contains text, not program code).

Exercise: running a Python file from the command line

So far, we've been running Python code either on CodeSkulptor or using the Jupyter Notebook. These are both quite modern methods of running code. For the last 50 years, programmers mostly used the command line, and that's what we'll be using to run our web server.

Let's first learn how to run a simple Python file from the command line.

If you're using a Windows computer:

- Run the cd command to check with folder you're in.
- Run cd TODO to move into the python folder inside the FutureMakers folder.
- Run the cd command to check you're in the python folder.
- Run the dir command to list the files in the python folder.
- Run the command python hello.py to run the hello.py file.
  This is exactly the same as executing a cell in the Jupyter Notebook, or pressing the run button on CodeSkulptor: it asks the computer to run the code.

- You should see a "Hello, FutureMakers!" message.

If you're using a Mac or Linux computer:

Congratulations, you just ran a Python file using the command line.

Exercise: editing code in Sublime Text and running it from the command line

In the rest of the workshop, we will be editing code in Sublime Text and either running it on your laptop using the command line, or sending it to Heroku so that your website is accessible to the world.

What does it mean? When you **deploy** code, you send it somewhere else so that it can run. When we make an update to a website, or ship a new computer game, we are deploying our code so that it's accessible to users.

Let's try editing some code in Sublime Text and running it.

- In Sublime, use the left bar to find the python folder inside the FutureMakers folder, and to find the hello.py file inside it.
- Use File > Save As to make a copy of hello.py. Give it a new name, but make sure it ends in .py (for example, you could call it goodbye.py).
- Change the new file so that it says something different to the user. Make sure to save it afterwards.
- Go back to the command line. Use cd to make sure you are still in the python folder inside the futuremakers folder.
- Run your new file using the python command (for example, python goodbye.py) and check that it works properly.

Congratulations, you just created a new Python code file and ran it using the command line.

What does it mean?

- A web server is a program which runs constantly to deliver websites to users.
- A web browser communicates with the server by making a **request**. (These are also known as Hypertext Transfer Protocol (HTTP) requests.)
- Servers can also communicate with each other by making requests.
- There are two kinds of request, the GET request (used for asking for a web page) and the POST request (used to send some data to the server and then get a web page back).
- Flask is a simple web server program which is written in Python. We'll be using it to run our website.

Tip: we can either run our web server on our laptop (in which case you will be able to access the website on your laptop, but it won't be accessible from anywhere else) or on Heroku (in which case it will be visible to the world). It would be possible to configure your laptop to serve the website to the world, but this would be a bad idea as the website would stop being available if your laptop was turned off.

Exercise: running the Flask sample web server project

Let's have a look at the Flask sample app. This app is a good starting point for the rest of your project.

- In Sublime Text, you should have the FutureMakers folder displayed in the bar on the left. If it's not there, add it with Project > Add Folder to Project.
- In the left bar in Sublime Text, look inside the FutureMakers folder and find the my_web_app folder.
- Have a quick look at the Python files inside this folder. Read the comments - they tell you what the code does.

Now let's launch the app.

- Open a new command line window by selecting Anaconda Prompt from the Start menu.
- In the command line, let's move to the my_web_app folder: use the command cd TODO.
- Check that you're in the right folder: use the command cd
- To make sure, list the files in this folder with the command ls.
  You should see the same files you saw in the my_web_app folder in Sublime Text.
- We need to install some libraries before we start the app. Run the command
  pip install -r requirements.txt
  You should see some output telling you that libraries have been installed. If you see a warning or error message, ask an instructor
- Once you're in the right folder, use the following command to run the app:
  ./run
  Tip: the command starts with a dot and then a forward slash.

You should see a message like the following:

```
* Serving Flask app "server.py" (lazy loading)
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 277-273-612
```

If you see this message, it means the command worked, and the Flask web server is now running on your laptop, serving the web app. It's not reachable from the entire Internet, but you can visit it on your laptop!

Tip: the command line window is no longer listening for your commands (you can't type anything in it), because it's busy running the Flask web server. If you want to shut it down, press ctrl and C together. To start it again, you can use the ./run command.

How do we visit the web app? When we started Flask, it told us the address it was running on:

```
http://127.0.0.1:5000/
```

Tip: this is a normal web address or URL. 127.0.0.1 is the IP address your laptop gives itself; 5000 is the port on which we are accessing the app.

What does it mean? a URL or uniform resource locator is just a Web address. URLs can either be words (like google.com) or numbers (like 127.0.0.1).

To visit the app, open a Web browser (Firefox or Chrome), enter http://127.0.0.1:5000/ in the address bar, and hit return. If everything is working properly, you should see the sample app and the message "Welcome to the FutureMakers sample app!" If you have any problems or questions, ask a demonstrator.

Tip: Rather than entering the full URL, you can just enter localhost:5000 in your browser's address bar. localhost is the word equivalent to 127.0.0.1.

Exercise: start modifying the sample app

Let's change the welcome message on the sample app.

In Sublime, locate the FutureMakers folder in the left bar, then the python folder, then the my_web_app folder, then the templates folder, then the file index.html.

In other words, use Sublime to open the file FutureMakers/python/my_web_app/templates/index.html

What does it mean? A **template** is a file which helps the Flask server build a web page. Flask assembles different templates into the final web page, and also inserts custom data like the user's name, a customised welcome message, or the current time. Templates, like most other pages, are written in HTML (hypertext markup language), a special language for defining web pages. *Hypertext* is text that you can click on (in other words, a link). A markup language is a language meant for describing documents rather than for programming.

HTML uses special words called **tags** to describe different parts of web pages. For example, some bold text is surrounded by the <b> tag:

<b>This text is bold</b>	**This text is bold**
This is the first line.
 This is the second line	This is the first line. This is the second line.
This text is normal, but <i>this is italic</i> and <b>this is bold</b>	This text is normal, but *this is italic* and **this is bold**

This is normal text <h1>and this is a heading</h1>	This is normal text and this is a heading

Tip: the first, most important page of a website is traditionally called index.html.

Let's look at index.html, which you should already have open in Sublime Text, and find the part of page which says "Welcome to the FutureMakers sample app!"

Here it is; we can see that it's surrounded by <h1> tags, meaning that it will show as a heading:

```
 <h1>Welcome to the FutureMakers sample app!<h1>
```

- Change this to something more interesting.
- Now save the file (control - S, or File > Save).
- Now go back to your browser and refresh the web page.

Once you refresh, you should see that the web page has changed to show your message.

Tip: whenever we make a change to a website we're working on, we must always save our code file in the text editor, then move to the browser and do a refresh. Web programmers do this very often indeed.

Exercise: deploying the sample app to Heroku

At the moment, your web app is only usable on your laptop: it can't be accessed from anywhere else. Let's change that by deploying it to Heroku so that it's visible to the entire world.

We'll be using two tools for this: the Heroku toolbelt, which is a program installed on your laptop used for communicating with Heroku, and Git, which is a general-purpose program for tracking, looking after, sending, and receiving code files like the ones which make up your web app.

The first thing we need to do is set up Heroku. You have been given a Heroku account with its own username and password. Open a new command line window (Anaconda Prompt under the Start menu), run the command **heroku login**, and enter your username and password. Heroku will tell you if you have logged in correctly.

Next, we need to set up Git. To do this,

- Open a new command line window (Anaconda Prompt under the Start menu) and navigate to the FutureMakers/python/my-web-app folder using cd TODO.
- Check you are in the right folder by running **cd**
- Run **git init** to ask Git to start tracking the my-web-app folder
- Run **heroku create** to make a new Heroku app - this will be the online home of our app.

We're nearly ready to deploy our web app. Let's first open the Heroku logs so we can see, live, what happens when we deploy - and if anything goes wrong!

- Keep your existing command line window open on the left of the screen.
- Open another command line window on the right of the screen, and move to the FutureMakers/python/my-web-app folder using cd TODO.
- Run cd to check you're in the right folder.
- Run the command **heroku logs -t**
  This asks Heroku to tell us exactly what it is doing by sending its log files (which keep a diary of everything that happens to our Heroku app) to us.

You should now have two command line windows: one on the left which is ready to accept commands, and one on the right which is showing you the Heroku logs. Make sure you can see all of both windows (you might need to make them smaller or move them around).

Now we're ready to deploy! Just run the command **./deploy**
(don't forget the dot and the forward slash). Deploying should take about a minute, perhaps two minutes: you can look at the logs to see progress.

If the deploy worked properly, you will see a message (in the left-hand window) like

```
https://polar-wildwood-98641.herokuapp.com/ deployed to Heroku
```

This message tells you the Web address of your app. In this case, you would copy and paste https://polar-wildwood-98641.herokuapp.com/ into the address bar of your favourite browser.

Visit the address shown in the deploy message, and check that your web app is working properly. Ask a demonstrator if there are any problems.

You can also run the command `heroku open` to visit the app directly.

Congratulations; you just deployed a web app to Heroku!

Tip: your Heroku app, for example https://polar-wildwood-98641.herokuapp.com/, is now accessible to the entire world. If you wanted a more convenient URL, you could buy a domain name like www.my-new-app.com, and configure it to point to your Heroku URL. Your app will remain available indefinitely, although under Heroku's free plan, it may take a few seconds for your app to respond if no one has used it for a while.

Exercise: add a new page to the Flask app

Let's add another page to the Flask app. In Flask, a page is made up of two parts:

1. The **HTML template**, which controls how the page will look. For example, a very simple HTML template could look like

```
<h1>This is my title!</h1>

<p>And this is my first and only paragraph.</p>
```

You can see this in my-web-app/templates/test_template.html

2. The **route**, which is some Python code which tells Flask what address your page should have and what to do when someone requests that page, including which template to show. For example, this route will show the template test_template.html when the user visits localhost:5000/test

```
@app.route('/test')
def test_function():
    return render_template('test_template.html')
```

This would be the equivalent, if our app was running on www.myapp.com, of visiting www.myapp.com/test

One route is special: it's the one we get when we don't request a particular page (by visiting www.myapp.com on the Web, or localhost:5000 on your laptop). This route starts with @app.route('/'), like this:

```
@app.route('/')
def hello_world():
    return render_template('index.html')
```

Look at these two routes (test_function and hello_world) in my-web-app/server.py, and make sure you understand them. Ask a demonstrator if you have any questions.

Now let's add a new page. We'll call it goodbye.

Let's make the template first. Make a new file called my-web-app/templates/goodbye.html, and save the following inside it:

```
<h1>Goodbye!</h1>

<p>We hope you visit again soon.</p>
```

Now let's make the route. It should look like this:

```
@app.route('/goodbye')
def goodbye():
    return render_template(goodbye.html')
```

In other words, this is a function called goodbye, which listens for a user typing a URL ending with goodbye, which then renders a template called goodbye.html. These names don't all need to be the same - we've chosen them to be the same here for simplicity.

Now let's check everything works - visit localhost:5000/goodbye. You should see your new Web page. If anything goes wrong, ask a demonstrator.

Congratulations, you just made a brand new web page!

Your turn: add a new page of your own

Follow the steps in the last exercise to first make your own template, then make your own route function, then visit it in the browser and test it.

Tip: make sure that neither your function, your route or your template have the same name as any pre-existing ones!

Exercise: deploy your app again

Exercise: add some visual styles

CSS (Cascading Style Sheets)

Style sheets are used to apply consistent visual styles (fonts, colours and spacing) to all pages.

At this point you may notice that you're making changes to the CSS and they're not reflected in the browser. This may be because your browser is not refreshing the latest version of the CSS file; it's using a cached version which it's storing itself (this makes refreshes much faster, but causes problems if the CSS has just changed).
You can either try another browser (Firefox or Chrome) or, to make sure everything is refreshed, do a "hard refresh":

- In most Windows and Linux browsers: Hold down Ctrl and press F5.
- In Apple Safari: Hold down ⇧ Shift and click the Reload toolbar button.
- In Chrome and Firefox for Mac: Hold down both ⌘ Cmd+⇧ Shift and press R.

Your turn: add some visual styles yourself

Exercise: designing an AI service to solve a problem

- Work in **teams**. There's an option to continue working on the project you started at Day 1 of our workshop. Just make sure your teammates are happy to go with this option too or you can start from scratch.
- If you already know the **Sustainable Development Goal** you'd like to tackle, you can narrow your focus on one of the subcategories called "targets" to be more specific: https://www.globalgoals.org. Just open the goal page and look for targets you would like to address.
- Do a bit of **research** to better understand the problem you are trying to solve, especially to figure out who are the people that suffer from this problem the most and what are their pressing needs.
- **Brainstorm** on how AI can help you solve this problem. Think of how, where and in what form AI can be applied to make these people's lives better or save our planet.

- Apply your **superpowers**: creativity, empathy, emotional intelligence, judgement, reasoning, communication, teamwork, and the AI skills you learned today.
- Think of any existing **alternatives** to your idea and why your idea is better. You can also think of ways to improve the existing solutions to this problem or augment human capabilities with AI.
- Use **Human Centered Design** framework to come up with a meaningful solution that will address the actual needs of these people. Make sure your solution is not only technically feasible, but also accessible to all the disadvantaged groups.
- Use **the Ethics of Code** framework: https://joinfuturemakers.com/ethics-of-code/ to make sure your solution doesn't harm or affect people in a negative way or increases inequality. Try to anticipate and plan for any unintended consequences.
- Think of **sustainable** ways to make your idea a real thing, a product that people around the world will use. How can you make it affordable for the disadvantaged groups? How can you scale it from one country/region to many?
- Think of the **next steps**. What are you going to do after the workshop is over?

Exercise: presenting your work

- **Introduction**: Please start by introducing yourself
- **Problem**: What problem are you trying to solve? So what / why do you think it matters?
- **User persona**: Who are the people that suffer from this problem the most? What does a day of their life look like now?
- **Impact**: How big is the impact? How would it change these people's lives?
- **SDG's**: Is this problem part of the UN Sustainable Development Goals?
- **Alternatives**: What are the current solutions and why are they not effective enough?
- **Your idea**: What is your solution and how does it solve the problem? How is it different from existing solutions?
- **Human Centred design**: What do people actually need? What is financially viable for them? What is technically and logistically feasible?
- **The Ethics of Code:** What ethical risks have you identified and how did you address them?
- **Sustainability**: What is the long-term ambition for your AI solution? Is it compatible with any of the existing approaches? Can it be scaled across different regions / spheres?
- **Next steps**: What do you plan to do with your idea next?

Going further after the workshop

Learning more about AI and machine learning

Setting up your development environment at home

To get your Anaconda notebooks and Flask server working, install the following:

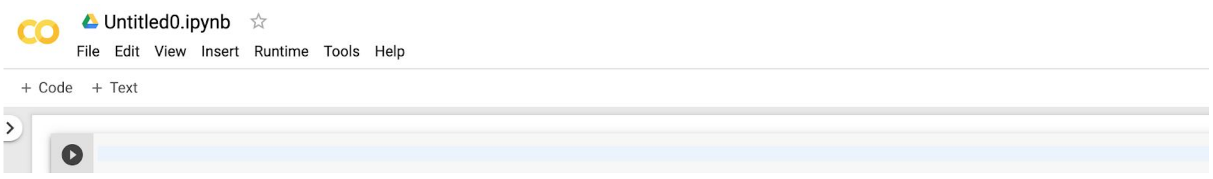
- Anaconda Python environment
  Make sure you install the Python 3.7 version
  https://www.anaconda.com/download/
- Git
  Windows: https://git-scm.com/download/win
  OS X: see here
  https://gist.github.com/derhuerst/1b15ff4652a867391f03
- Heroku command line interface (CLI)
  https://devcenter.heroku.com/articles/heroku-cli
- Make sure both Firefox and Chrome are installed
  (so that you can run your web app in one, and do Google searches or look up documentation in the other)
- Sublime Text
  https://www.sublimetext.com/

Using the Google Jupyter Notebook

The Google Jupyter Notebook allows you to use a Jupyter Notebook online without needing to install anything on your computer.

Visit https://colab.research.google.com/notebooks/welcome.ipynb. Create a new notebook under the 'File' menu.

Google Jupyter Notebook has slightly different commands to the jupyter notebook you have been using. The interface looks like this:



To run the cell click the run button to the left of a cell:

You can create anew code cell by clicking on the button to the top left:

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

+ Code

Noted that running this notebook requires an internet connection, so itmight be slower compared to running notebook on your own computer. This is a great tool if you want to share your code with your friends and work in the same notebook together!

Getting Heroku working

Once you've installed the Heroku CLI:

- In Anaconda Prompt, navigate to the server folder. Make sure you're inside the server folder and you can see **server.py** when you do dir.
- Run **heroku login**.
- Run **heroku apps:info** to see your apps and their URLs. You can then visit the URLs to see your apps.
- If you've copied your folder over properly from the workshop, you should be able to run
  git add .
  git commit -m 'deploy'
  git push heroku master
  to deploy a new version of your app.

Next steps - web servers

Storing data about the user

Web servers are **stateless**: the server doesn't remember anything between requests. Once it has answered a request, it forgets about everything that happened in that request. This is so that you can maintain a fleet of web servers, any one of which can deal with any request; if one crashes, no data is lost.

So where do we store our data?

All the data a web app needs - from user accounts to profile pages - is usually stored in a separate server, the database server. This is a specialised server which doesn't answer web requests, just allows them to store and read data. The most commonly used database program is called Postgres.

Here, we will cheat by not using a database: we will store our data in the **session**, a small file which is held inside the user's browser in a cookie. So all data the server needs to remember is stored not in the server or the database, but the user's browser.

Database

Remember you can't store any information in the web server (because many servers and ephemeral filesystems). At the moment we are storing information in the user's session; we should instead do this in a database.

The database's job is to store all data that's important to our app. We talk to the database using a language called SQL (Structured Query Language).

You will need two databases: a local one for test, and a production one which the web server can talk to. It's best to use the same type of database for both.

Heroku comes with a free database called Postgres (which is an old open source project from Berkeley, California).

Tutorial:
https://devcenter.heroku.com/categories/heroku-postgres

You can also install Postgres locally. Installation guides are here:
https://wiki.postgresql.org/wiki/Detailed_installation_guides

Flask uses a library called Flask-SQLAlchemy to talk to the database.

Whether local or production, you will need a database URL; you configure this in Flask to ensure Flask is talking to the right database.

Javascript

All the code we have written so far has run on the server. You can install Javascript files in your web app which run on the user's browser; they can do things like create popup windows and animate parts of the page.

The best Javascript framework to start with is called JQuery; look up how to use the document ready function (a function which is called when the page is ready, into which you can place your code).

Tutorial: https://www.tutorialspoint.com/jquery/