

# A simple web database project

## Part 1 Setting up a Flask application

This project uses Flask. Flask is a web application framework written in Python that allows you to develop web applications relatively easily.

To follow the project you need core skills in:

- Python
- HTML (and some understanding of CSS)
- SQL
- Database design

You need to set up an account on Python Anywhere to follow this tutorial. Python Anywhere is free to use (for a basic account) and provides you space to host your files on a web server with Flask pre-installed as well as access to various database servers, including a MySQL server (which you will use for this project).

Your account will allow you to run a single web application at a subdomain on the domain pythonanywhere.com. There are restrictions in terms of storage space, CPU, bandwidth and outbound Internet access from your apps. However, a free account should be adequate to host a student project.

Python Anywhere also has a free education beta feature to support a classroom learning environment.

### Getting started with Python Anywhere.

1. Create a free account on <https://www.pythonanywhere.com/>
2. You need to specify a username and should bear in mind that this name will be used as the subdomain for your website. For example, if you specify the username mickeymouse, your web site will be accessed by typing:

`https://mickeymouse.pythonanywhere.com/`

in a browser.

3. Once you've signed up, you'll be taken to your dashboard and will be given the option of taking a brief tour. This does not take long and is worth doing so that you are introduced to the basic layout of the site. You do not need to do any of the advanced tours.
4. Return to the PythonAnywhere dashboard. You will be prompted to check your email and confirm your email address. This is essential if you are likely to forget (and need to reset) your password at any stage.

## Creating your website

1. From the dashboard, click on the "Web" link near the top right, and you'll be taken to a page where you can create your website. Remember that free accounts can have just one website.
2. Click on the "Add a new web app" button. This will open a configuration wizard that allows you to set up your site.
  - Accept the default (free) domain name
  - Select **Flask** as the web framework you want to use.
  - Select the version of Python you wish to use. PythonAnywhere has various versions of Python installed, and each version has its associated version of Flask. This tutorial uses **Python 3.9**
  - If you are a new user, you can accept the **default settings for the file path**. This will be:  
`/home/username/mysite/flask_app.py`  
If you have an existing account, set another path (instead of mysite) or any files you have may be overwritten.
3. Now PythonAnywhere will set up your website and the configuration page for the site will be displayed. You will find there is a lot of information and there are settings that can be changed. However, there is one really important thing on this page that you need to be

aware of - **the site is given an automatic expiry date in three months time**. If you are likely to need the site to persist for more than 3 months, you must periodically return to this page and extend the expiry date. It is worth getting into the habit of doing this every time you access your account.

4. A very basic Flask web page has been created for you. Right click on the domain name that appears at the top of the page following the words **configuration for** and open the page in a new tab. You should see a page with the greeting “Hello from Flask”. Keep this tab open as you will want to check your progress regularly as you work on your site.

## flask\_app.py

1. Click on **files** on the main menu bar. You will see the directories list on the left and the contents of the current directory on the right.
2. Click on **mysite** (unless you changed the default path in a previous step) and you will see the file flask\_app.py. Click on the file name to open the file.
3. The python file is opened in an editor window:

```
1
2 # A very simple Flask Hello World app
3
4 from flask import Flask
5
6 app = Flask(__name__)
7
8 @app.route('/')
9 def hello_world():
10     return 'Hello from Flask!'
```

The core flask module is imported (line 4) and a Flask app is created (line 6).

The code of line 8 uses a Python decorator. You don't need to understand decorators to complete this tutorial but, fundamentally, it specifies that if a user accesses your website without specifying a particular page (i.e. the app route is "/"), the function `hello_world` will be run. This function returns the words that will be displayed as a basic web page.

4. Change the return value to display something else, for example "Hello World". Save the file (Green Save button) and then click the icon (shown below) to reload the site:



This is an important step as it puts a new copy of your Flask application onto the live server.

5. Now, in the tab you left open which is displaying your site, click the refresh button. The new message should be displayed. If the message hasn't changed you may need to wait a few seconds and try again. If it still doesn't update, make sure you have definitely saved the file and reloaded the site.

## Making a new web page

A Flask web application is made up of two types of file:

- source code, which is written in Python
- templates, which are written in HTML with Flask extensions

The first page you will create for the site will be the home page. In this tutorial, the page will be very simple. If you have good design skills (backed up by great HTML and CSS skills), you can create something far more sophisticated!

1. Click on `mysite` in the breadcrumb trail (at the top left of the editor page), to return to your working directory.
2. Create a new subdirectory (of `mysite`) named `templates`. Type “templates” into the text box that is displaying “Enter new directory name” and then click on the “New directory” button. This will create the new directory and switch into it.
3. Create a new template file here named `index.html`. Type the name “index.html” into the text box that is displaying “Enter new file name” and then click the “New file” button. This will open the editor.
4. Type the following HTML into the editor:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>News</title>
  </head>

  <body>
    <h1>Sustainable Development</h1>

    <p>
      A set of <a
      href="https://www.undp.org/sustainable-development-goals">
        Global Goals</a>, also called the Sustainable Development
        Goals, were adopted by the United Nations in 2015 as a
        call to action to end poverty, protect the planet, and to
        ensure that by 2030 everyone can enjoy peace and
        prosperity.
    </p>

  </body>
</html>
```

5. Save the file and return to the `mysite` directory and open your source code file

`flask_app.py` for editing. You need to change the code so that it displays the new template file (rather than a basic message):

- Add another module to the import statement:

```
from flask import Flask, render_template
```

- Change the action for the default path. Delete the existing function `hello_world` function and replace it with a new function named `index`, as follows:

```
@app.route("/")
```

```
def index():  
    return render_template("index.html")
```

The name of the function here is not important although naming it `index` will make it easy to identify when you have more code. The main thing is that you always use **meaningful identifiers** and a **consistent naming style**.

6. Now save the python file and reload the site. Return to your browser tab that is displaying the web page and refresh the page. Your new home page will be displayed.

## Extension task(s)

1. Add more text information to your web page (you will learn how to add styles and images in the next steps). Research the 17 sustainability targets and add a summary of each to the page.

## Trouble-shooting

1. If your new page is not displaying, make sure that you have:

- saved the html file with the name `index.html` in a folder named `templates`
- checked that the `templates` folder is a subdirectory of your `mysite` folder
- saved the file `flask_app.py` and reloaded the site
- refreshed the web page in the browser

2. In the editor, you will sometimes get warnings if there is a problem with your code. You can also “run” the code from the run button on the menu bar. This will open a terminal window and display regular Python error messages to help you debug your code.

```
1
2 # A very simple Flask Hello World app for you to get started with...
3
4 from flask import Flask, render_template
5
6 app = Flask(__name__)
7
8 @app.route('/')
9 def index()
10     return render_template("index.html")
11
12
13
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/bin/pythonanywhere_runner.py", line 26, in _pa_run
    code = compile(f.read(), filename.encode("utf8"), "exec")
  File "/home/germio/mysite/flask_app.py", line 9
    def index()
        ^
SyntaxError: expected ':'
```

3. You can also view various logs in PythonAnywhere. Return to the website configuration page by selecting Web from the menu bar. You will see that there are 3 logs:

## Log files:

---

The first place to look if something goes wrong.

Access log: [germio.pythonanywhere.com.access.log](#)

Error log: [germio.pythonanywhere.com.error.log](#)

Server log: [germio.pythonanywhere.com.server.log](#)

Log files are periodically rotated. You can find old logs here: [/var/log](#)

The middle one (error log) is usually the most useful for debugging.

## Attaching a style sheet

If you have done some web development previously you will know that HTML is used to mark up the content of a web page and CSS is used to style the page. This tutorial does not cover CSS but you should be able to find help on the excellent [W3 schools site](#).

The CSS file will be a **static file** - its contents do **not** need to be processed by the Flask application. You will save the file in a suitably named subdirectory and tell Flask that the file is static.

1. Select the `files` tab from the menu bar and make sure you are in the `mysite` directory.

Create a new subdirectory named `assets` and then a subdirectory (within `assets`) named `css`. Keep an eye on the breadcrumb trail so you know which directory you are working in.

2. Now create a very basic css file - named `sustain.css`. Something like this:

```
body {  
    background-color: #e8f4ea;  
    font-family: Arial, Helvetica, sans-serif;  
}
```


3. Save the file inside the new `css` subdirectory.



- Now you need to update the web site configuration with details of where to find the static files. Navigate to the `web` tab from the menu bar and, in the Static Files section, set up the URL and path:

Static files:

Files that aren't dynamically generated by your code, like CSS, JavaScript or uploaded files, can be served much faster straight off the disk if you specify them here. You need to **Reload your web app** to activate any changes you make to the mappings below.

URL	Directory	Delete
<a href="#">/static/</a>	<a href="#">/home/germio/mysite/assets</a>	
<a href="#">Enter URL</a>	<a href="#">Enter path</a>	

- The final step is to attach the stylesheet to your web page. Open the `index.html` file and inside the `<head>` section at the top of the file, add the following link statement:

```
<link rel="stylesheet" href="/static/css/sustain.css">
```

Notice that `static` is used within the URL and will resolve to the correct path as specified in step 4.

## Trouble-shooting

- If your styles have not been applied, make sure that you have:
  - Saved the style sheet in a directory whose path (starting at `mysite`) is `mysite/assets/css`
  - Added the link to the `index.html` file in the head section with the correct syntax for the href (all slashes are correct) and you have used the word `static` rather than `assets`
  - Checked that all of your directories and filenames are specified in lower case names and include no spaces.
  - reloaded the site
  - refreshed the web page in the browser

- Checked that the mappings on the site configuration page are correct. The settings are case sensitive. Check carefully that the URL setting has a slash on either side of the word `static` and that the directory setting starts (but not ends) with a slash.

You can copy the `index.html` file and the `sustain.css` file to a local folder on your computer to see if the files work locally. You will need to change the link statement in the html file to `<link rel="stylesheet" href="sustain.css">` and make sure that both files are saved in the same folder.

## Adding an image to a page

Images are also static files and can be saved within the same assets folder that you have already mapped as static.

1. Create a new subdirectory within the assets folder named `images`.
2. Find a suitable royalty free image to use on your site. [Pixabay](#) is a good source.
3. Make sure that the image is suitably prepared:
  - Give the image a sensible name, e.g. `turbine.jpg`
  - Use a photo editing tool to crop and resize the image to a suitable size for the web (a general guideline is no more than 1000 pixels wide and height kept in proportion).
  - Compress the image further if it is still too large. A reasonable size is around 200KB. You may be able to do this in a photo editor (often as an option when saving) or there are on-line tools that will do the job.
4. Upload the image to your new `images` subdirectory.
5. Add an image link in an appropriate place in the body section of your web page:

```

```

## Part 2. Linking to a database

By now, you should be familiar with navigating your way around the PythonAnywhere site and be able to troubleshoot common problems.

Now you will add a second page to your site. This page will be used to collect and display sustainability pledges from people who engage with your site. The pledges will be saved in a simple database.

### Adding a second page

1. Create a new HTML web page in the templates subdirectory. Name the file

`pledges.html`. The easiest way to do this is to copy the `index.html` file and save the copy with the new name. Then tweak the page content as required so that you have a page that looks something like this:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <link rel="stylesheet" href="/static/css/sustain.css">
6     <title>Pledges</title>
7   </head>
8
9   <body>
10    <h1>Make a pledge</h1>
11
12    <p>
13      Make your pledge to help meet the global sustainability goals.
14    </p>
15
16  </body>
17 </html>
```

2. Now edit your main application file `flask_app.py` to add a route to the new page.

Underneath your path to the index page add the following lines of code:

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

3. Save the python file and reload your site. Now test the path by typing the following URL in the browser address bar:

<https://xxxx.pythonanywhere.com/pledges>

(where xxxx is your own username). The new page should be displayed.

## Extension task(s)

- Add a navigation bar to both pages so that you can move easily between them

## Adding a form to collect a pledge

The next step is to add a simple form to your new web page.

1. Open the file `pledges.html` and add the following code to the body section of the page

```
<form action="" method="POST">  
    <textarea name="pledge" placeholder="Enter your pledge"  
        rows="5" cols="30"></textarea>  
    <br/>  
    <input type="submit" value="Make pledge">  
</form>
```

2. Save the python file and reload your site. You can test your form but if you press the submit button you will get a “Method not allowed” error message. This is because the site has not been configured to receive data from your application.

When the HTTP request is made for the page, it uses the default GET method. Here, because we want to send data to the web server we want it to respond to the POST method.

3. Open the file `flask_app.py` and change the line:

```
@app.route("/pledges")
```

to:

```
@app.route("/pledges", methods=["GET", "POST"])
```

4. Save the file and reload the site. Refresh the page in the browser and try again to make a pledge. Now you won't get an error message. However, nothing is happening with your pledge because you haven't coded the necessary form action.

## Extension task(s)

- Validate the data collected from the form. Display an error message if any data is missing.

## Creating a database

Now it's time to add a database. You are going to use a MySQL database server, which you can use from a free PythonAnywhere account. You are also going to use a database interface module named SQLAlchemy, which works well with Flask.

1. On the right hand side of the PythonAnywhere window you will see three horizontal bars. This opens a drop down menu. Select the `databases` option.

2. You need to choose a password to access the database. This should be different to your main PythonAnywhere password, because it is likely to appear in plain text in any web applications you write. Make a note of the password you've chosen, and then click the "Initialize MySQL" button. Wait for a while for your database to be set up.
3. When step 2 is complete you will be shown your MySQL settings:
  - At the top, under "Connecting", you'll see the details you'll need to connect to the MySQL server associated with your account.
  - Next is a list of MySQL databases; you will see that one has been created for you called `yourusername$default`. You can use this database for your application.

Note that if your username is longer than 16 characters, the prefix will be truncated to 16 characters.

4. Leave the page open in a browser tab as you will need information from the page in the next steps.

## Connecting the database to your application

1. Open the file `flask_app.py` and add an extra import statement to the start of the program. Add this line of code after the other import statement at the top of your file:

```
from flask_sqlalchemy import SQLAlchemy
```

2. To help keep the `flask_app.py` file organised, add a couple of blank lines below the line

```
app = Flask(__name__)
```

and add the comment:

```
# Database settings
```

3. Add another blank line and then type the following lines of code:

```
SQLALCHEMY_DATABASE_URI =  
"mysql+mysqlconnector://{username}:{password}@{hostname}/{databasename}  
}".format(  
    username="xxx",  
    password="xxx",  
    hostname="xxx",  
    databasename="xxx",  
)  
app.config["SQLALCHEMY_DATABASE_URI"] = SQLALCHEMY_DATABASE_URI  
app.config["SQLALCHEMY_POOL_RECYCLE"] = 299  
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
```

4. You need to replace the "xxx" in the code with the values set for `username`, `hostname` and `databasename` on the Databases configuration page. The value for `password` will be the one you set earlier.
5. The next step is to create a reference to the database. Type this line after the code you just added:

```
db = SQLAlchemy(app)
```

6. At this stage it is sensible to save the file and reload the site. You can test it as you did previously. If all is well, you should get a site that behaves just like it did before. If there's a problem, Flask should tell you what it is.

## Creating a model for a database table

The next step is to add a *model* which is a Python class that defines a table in the database.

1. To help keep the flask\_app.py file organised, add a couple of blank lines and a comment below the database definition statement:

```
# Table models
```

2. Now add a class for the model that defines the table that will store the pledges:

```
class Pledge(db.Model):  
  
    __tablename__ = "tblPledge"  
  
    id = db.Column(db.Integer, primary_key=True)  
    pledge = db.Column(db.String(4096), nullable=False)  
    country = db.Column(db.String(100), nullable=False)  
    screen_name = db.Column(db.String(100), nullable=False)
```

There are four fields in the table:

- `id` (data type integer). This is specified as the primary key and will auto increment by default.
- `pledge` (data type string). This will hold the pledge that the user enters. The field length is set to the maximum value of 4096 characters. The field cannot be left blank.
- `country` (data type string). This will hold the name of the country that the user enters. The field length is set to 100 characters. The field cannot be left blank.
- `screen_name` (data type string). This will hold the name tha the user enters to be displayed alongside their pledge. The field length is set to the maximum value of 100 characters. The field cannot be left blank.

3. Save your Python file



## Creating the database table

Now that we've defined our model, we need to get SQLAlchemy to create the database table.

This is a one-off operation. Click the blue "Run" button at the top left of the screen. Then use the black console window (in the area below your Python code).

1. The first step is to import the database object that is created in your code. Type the following command in the console window:

```
from flask_app import db
```

2. Now, you can create the table. Type the following command in the console window:

```
db.create_all()
```

3. Once you've done that, the table will have been created in the database from the model which you defined. You can check this by interrogating the database. Access the tab that you have kept open that's showing the Databases configuration. Click on the name of your database (*yourusername\$default*). This will open a new console window, running the MySQL command line prompt: `mysql>`. Here you can run any valid SQL commands against the database.

4. Type the command:

```
show tables; (don't forget the semicolon at the end)
```

and press enter.

This should show you that you have a table called `tblPledge`.

5. You can inspect the table using the command:

```
describe tblPledge;
```

You'll see that you have four columns to match those defined in the model in your Python code!

```
mysql> describe tblPledge;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
pledge	varchar(4096)	YES		NULL	
country	varchar(100)	YES		NULL	
screen_name	varchar(100)	YES		NULL	

```
4 rows in set (0.00 sec)
```

## Using the database - adding records

Now that you have set up the database, we need to write some code to add new pledges to the database and to display pledges from the database.

1. Open the file `flask_app.py` and modify the code for the pledges route as shown below:

```
def pledges():
    if request.method == "GET":
        return render_template("pledges.html")
    else:
        record = Pledge(pledge=request.form["pledge"],
country=request.form["country"],
screen_name=request.form["screen_name"])
        db.session.add(record)
        db.session.commit()
        return redirect(url_for("pledges"))
```

Here the code deals with each of the two ways the page can be requested:

- with the standard GET method
- with the POST method that will be used only if the form has been submitted.

If the GET method is used, the page is displayed as usual.

If the POST method is used, the data from the form is harvested and is passed to the model for the Pledge table to create a record. This record is then added to the database. The change to the database is then committed.

2. You can check that the record has been written by using the MySQL console and running the command:

```
select * from tblPledge;
```

The record you have just added should be displayed.

## Using the database - displaying records

Your final challenge is to display the records from the database (i.e. the pledges) on the web page. You will simply write the data into the space below the form.

1. Open the file `flask_app.py` and modify the code for the pledges route as shown below:

```
def pledges():
    if request.method == "GET":
        data = Pledge.query.all()
        return render_template("pledges.html", pledge_data =
data)

    . . .
```

The command `Pledge.query.all()` will select all of the records from the table (as defined by the model). This data will then be passed to the web page where it can be displayed.

2. Save the file.

3. Now, you need to change the web page template `pledges.html`. You are going to use Flask to display the data on the page. Underneath the form you need to add a loop to display the data. You will put the data into a table so it looks tidy:

```
<table>
  <tr>
    <th>Name</th>
    <th>Country</th>
    <th>Pledge</th>
  </tr>

  {% for r in pledge_data %}

    <tr>
      <td>{{r.screen_name}}</td>
      <td>{{r.country}}</td>
      <td>{{r.pledge}}</td>
    </tr>

  {%endfor%}

</table>
```

This creates a table with three columns and then uses a Flask for loop to iterate through the data that was passed to the page, taking each row (from the table data) in turn. The information is displayed in cells in a new row in the table.

4. Save the file and reload the site. If you have typed everything carefully, the data from the database should display underneath the form. Everytime you add a pledge, a new record will be displayed.

## A final polish!

The table that displays the data from the database looks much better with a bit of styling. You can find the code for the final version of the website [here](#). This includes all of the files including a style sheet with rules to style the table.