Deploy!

> **Note** The following chapter can be sometimes a bit hard to get through. Persist and finish it; deployment is an important part of the website development process. This chapter is placed in the middle of the tutorial so that your mentor can help with the slightly trickier process of getting your website online. This means you can still finish the tutorial on your own if you run out of time.

Until now, your website was only available on your computer. Now you will learn how to deploy it! Deploying is the process of publishing your application on the Internet so people can finally go and see your app. :)

As you learned, a website has to be located on a server. There are a lot of server providers available on the internet, we're going to use PythonAnywhere (https://www.pythonanywhere.com/). PythonAnywhere is free for small applications that don't have too many visitors so it'll definitely be enough for you now.

The other external service we'll be using is GitHub (https://www.github.com), which is a code hosting service. There are others out there, but almost all programmers have a GitHub account these days, and now so will you!

These three places will be important to you. Your local computer will be the place where you do development and testing. When you're happy with the changes, you will place a copy of your program on GitHub. Your website will be on PythonAnywhere and you will update it by getting a new copy of your code from GitHub.

Git

> **Note** If you already did the installation steps (../installation/), there's no need to do this again – you can skip to the next section and start creating your Git repository.

Git is a "version control system" used by a lot of programmers. This software can track changes to files over time so that you can recall specific versions later. A bit like the "track changes" feature in word processor programs (e.g., Microsoft Word or LibreOffice Writer), but much more powerful.

Installing Git

Installing Git: Windows

Installing Git: OS X

Installing Git: Debian or Ubuntu

Installing Git: Fedora

Installing Git: openSUSE

Starting our Git repository

Git tracks changes to a particular set of files in what's called a code repository (or "repo" for short). Let's start one for our project. Open up your console and run these commands, in the `djangogirls` directory:

> **Note** Check your current working directory with a `pwd` (Mac OS X/Linux) or `cd` (Windows) command before initializing the repository. You should be in the `djangogirls` folder.

command-line

```
$ git init
Initialized empty Git repository in ~/djangogirls/.git/
$ git config --global user.name "Your Name"
$ git config --global user.email you@example.com
```

Initializing the git repository is something we need to do only once per project (and you won't have to re-enter the username and email ever again).

Adjusting your branch name

If the version of Git that you are using is older than **2.28**, you will need to change the name of your branch to "main". To determine the version of Git, please enter the following command:

command-line

```
$ git --version
git version 2.xx...
```

Only if the second number of the version, shown as "xx" above, is less than 28, will you need to enter the following command to rename your branch. If it is 28 or higher, please continue to "Ignoring files". As in "Initializing", this is something we need to do only once per project, as well as only when your version of Git is less than 2.28:

command-line

```
$ git branch -M main
```

Ignoring files

Git will track changes to all the files and folders in this directory, but there are some files we want it to ignore. We do this by creating a file called `.gitignore` in the base directory. Open up your editor and create a new file with the following contents:

.gitignore

```
# Python
*.pyc
*~
__pycache__

# Env
.env
myvenv/
venv/

# Database
db.sqlite3

# Static folder at project root
/static/

# macOS
._*
.DS_Store
.fseventsd
.Spotlight-V100

# Windows
Thumbs.db*
ehthumbs*.db
[Dd]esktop.ini
$RECYCLE.BIN/

# Visual Studio
.vscode/
.history/
*.code-workspace
```

And save it as `.gitignore` in the "djangogirls" folder.

> **Note** The dot at the beginning of the file name is important! If you're having any difficulty creating it (Macs don't like you to create files that begin with a dot via the Finder, for example), then use the "Save As" feature in your editor; it's bulletproof. And be sure not to add `.txt`, `.py`, or any other extension to the file name -- it will only be recognized by Git if the name is just `.gitignore`. Linux and MacOS treat files with a name that starts with `.` (such as `.gitignore`) as hidden and the normal `ls` command won't show these files. Instead use `ls -a` to see the `.gitignore` file.
>
> **Note** One of the files you specified in your `.gitignore` file is `db.sqlite3`. That file is your local database, where all of your users and posts are stored. We'll follow standard web programming practice, meaning that we'll use separate databases for your local testing site and your live website on PythonAnywhere. The PythonAnywhere database could be SQLite, like your development machine, but usually you will use one called MySQL which can deal with a lot more site visitors than SQLite. Either way, by ignoring your SQLite database for the GitHub copy, it means that all of the posts and superuser you created so far are going to only be available locally, and you'll have to create new ones on production. You should think of your local database as a good playground where you can test different things and not be afraid that you're going to delete your real posts from your blog.

It's a good idea to use a `git status` command before `git add` or whenever you find yourself unsure of what has changed. This will help prevent any surprises from happening, such as wrong files being added or committed. The `git status` command returns information about any untracked/modified/staged files, the branch status, and much more. The output should be similar to the following:

command-line

```
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
        blog/
        manage.py
        mysite/
        requirements.txt

nothing added to commit but untracked files present (use "git add" to track)
```

And finally we save our changes. Go to your console and run these commands:

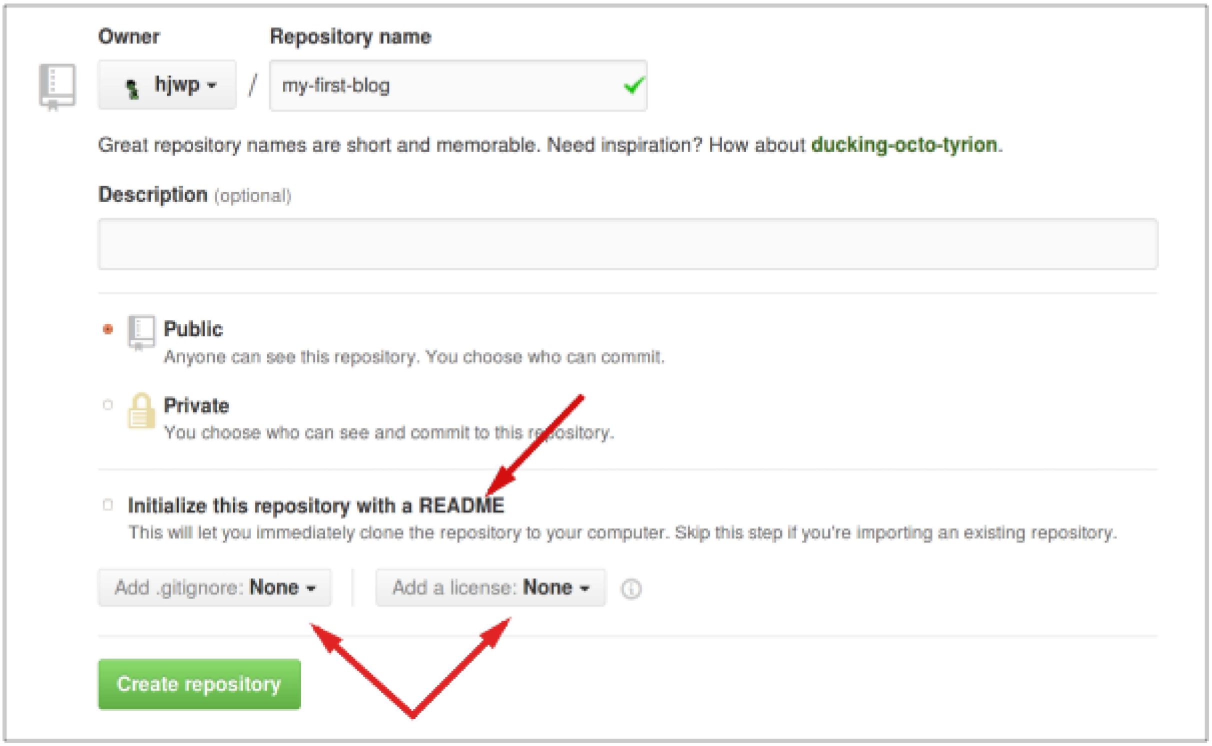
command-line

```
$ git add .
$ git commit -m "My Django Girls app, first commit"
 [...]
 13 files changed, 200 insertions(+)
 create mode 100644 .gitignore
 [...]
 create mode 100644 mysite/wsgi.py
```

Pushing your code to GitHub

Go to GitHub.com (https://www.github.com) and sign up for a new, free user account. (If you already did that in the workshop prep, that is great!) Be sure to remember your password (add it to your password manager, if you use one).

Then, create a new repository, giving it the name "my-first-blog". Leave the "initialize with a README" checkbox unchecked, leave the .gitignore option blank (we've done that manually) and leave the License as None.


Owner / Repository name
hjwp / my-first-blog

Great repository names are short and memorable. Need inspiration? How about ducking-octo-tyrion.

Description (optional)

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

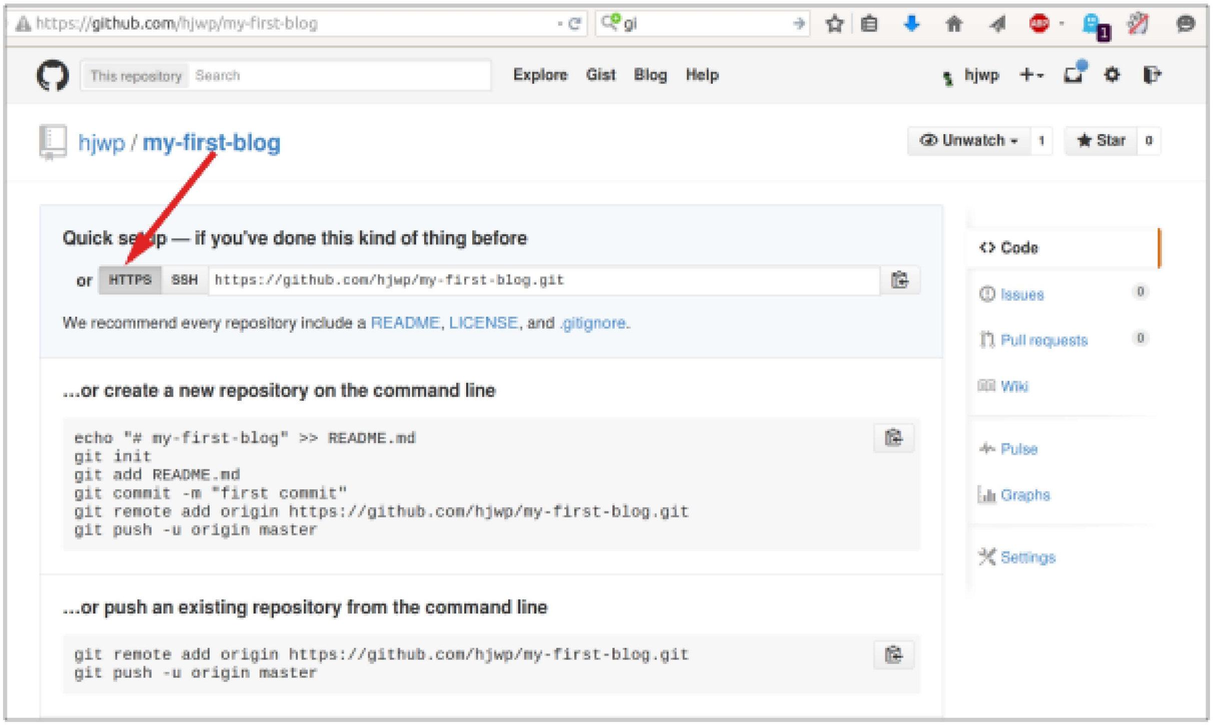
Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None
Add a license: None

Create repository

Note The name `my-first-blog` is important – you could choose something else, but it's going to occur lots of times in the instructions below, and you'd have to substitute it each time. It's probably easier to stick with the name `my-first-blog` .

On the next screen, you'll be shown your repo's clone URL, which you will use in some of the commands that follow:


https://github.com/hjwp/my-first-blog

hjwp / my-first-blog

Quick setup — if you've done this kind of thing before

or HTTPS SSH https://github.com/hjwp/my-first-blog.git

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# my-first-blog" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/hjwp/my-first-blog.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/hjwp/my-first-blog.git
git push -u origin master
```

Now we need to hook up the Git repository on your computer to the one up on GitHub.

Type the following into your console (replace `<your-github-username>` with the username you entered when you created your GitHub account, but without the angle-brackets -- the URL should match the clone URL you just saw).

command-line

```
$ git remote add origin https://github.com/<your-github-username>/my-first-blog.git
$ git push -u origin main
```

When you push to GitHub, you'll be asked for your GitHub username and password (either right there in the command-line window or in a pop-up window), and after entering credentials you should see something like this:

command-line

```
Counting objects: 6, done.
Writing objects: 100% (6/6), 200 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/ola/my-first-blog.git
 * [new branch]      main -> main
Branch main set up to track remote branch main from origin.
```

Your code is now on GitHub. Go and check it out! You'll find it's in fine company – Django (https://github.com/django/django), the Django Girls Tutorial (https://github.com/DjangoGirls/tutorial), and many other great open source software projects also host their code on GitHub. :)

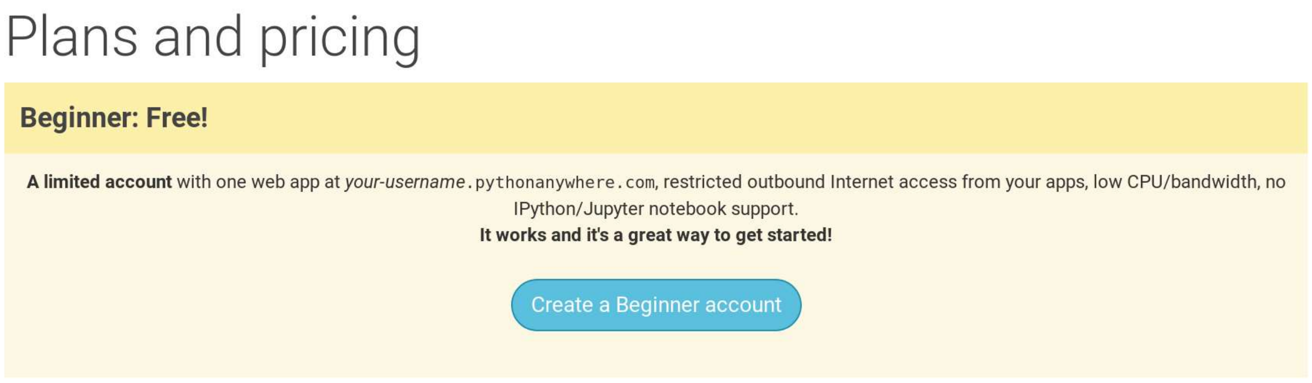
Setting up our blog on PythonAnywhere

Sign up for a PythonAnywhere account

> **Note** You might have already created a PythonAnywhere account earlier during the install steps – if so, no need to do it again. PythonAnywhere is a service for running Python code on servers "in the cloud". We'll use it for hosting our site, live and on the Internet.

We will be hosting the blog we're building on PythonAnywhere. Sign up for a "Beginner" account on PythonAnywhere (the free tier is fine, you don't need a credit card).

- www.pythonanywhere.com (https://www.pythonanywhere.com/)


Plans and pricing

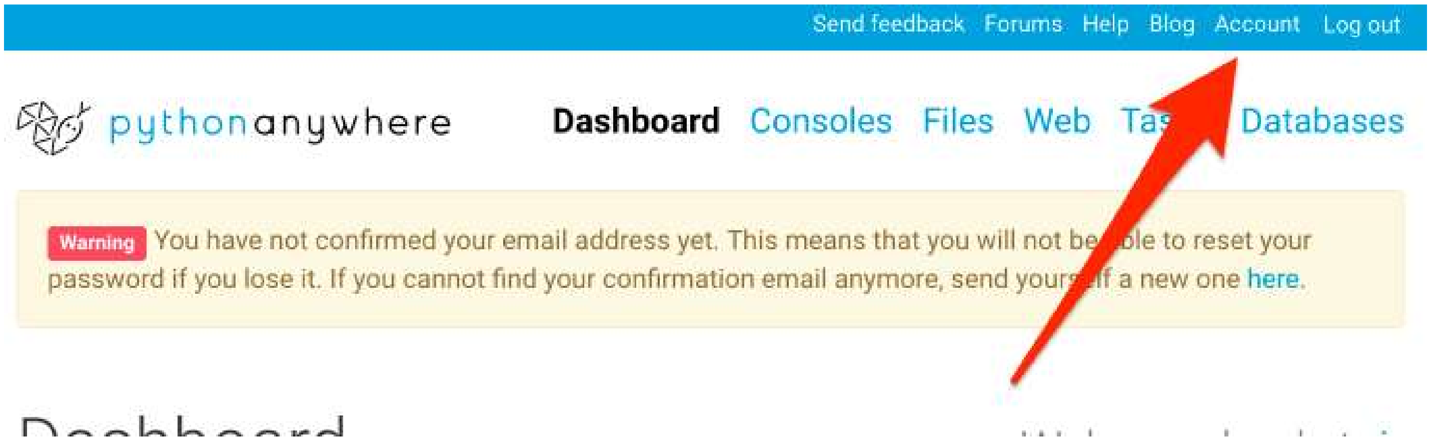
Beginner: Free!

A limited account with one web app at *your-username*.pythonanywhere.com, restricted outbound Internet access from your apps, low CPU/bandwidth, no IPython/Jupyter notebook support.
It works and it's a great way to get started!

> **Note** When choosing your username here, bear in mind that your blog's URL will take the form `yourusername.pythonanywhere.com` , so choose either your own nickname or a name for what your blog is all about. Also, be sure to remember your password (add it to your password manager, if you use one).

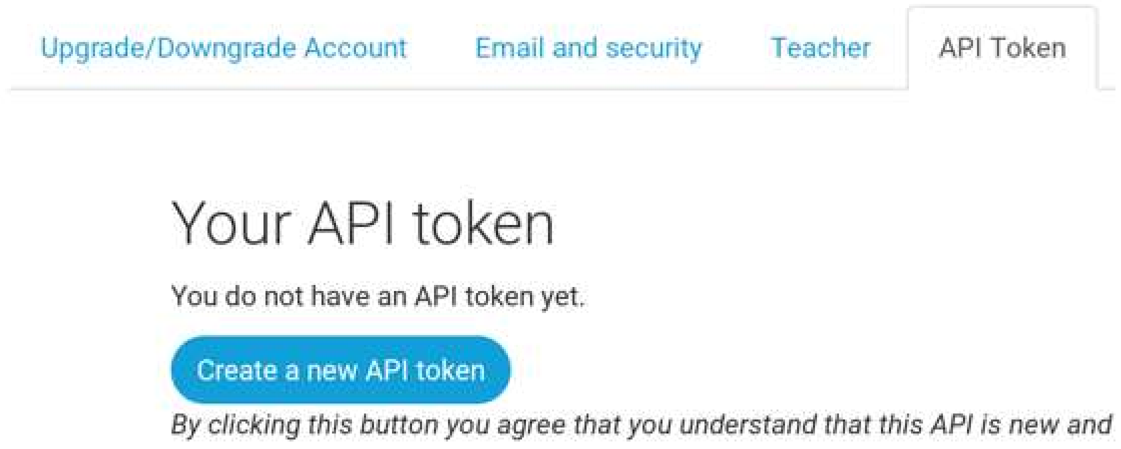
Creating a PythonAnywhere API token

This is something you only need to do once. When you've signed up for PythonAnywhere, you'll be taken to your dashboard. Find the link near the top right to your "Account" page:


Send feedback Forums Help Blog Account Log out
pythonanywhere
Dashboard Consoles Files Web Tasks Databases

Warning You have not confirmed your email address yet. This means that you will not be able to reset your password if you lose it. If you cannot find your confirmation email anymore, send yourself a new one here.

then select the tab named "API token", and hit the button that says "Create new API token".



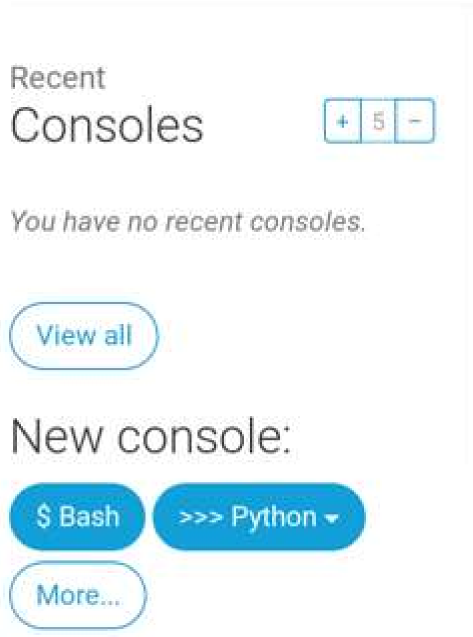
Your API token

You do not have an API token yet.

By clicking this button you agree that you understand that this API is new and

Configuring our site on PythonAnywhere

Go back to the main PythonAnywhere Dashboard (https://www.pythonanywhere.com/) by clicking on the logo, and choose the option to start a "Bash" console – that's the PythonAnywhere version of a command line, just like the one on your computer.



You have no recent consoles.

> **Note** PythonAnywhere is based on Linux, so if you're on Windows, the console will look a little different from the one on your computer. Deploying a web app on PythonAnywhere involves pulling down your code from GitHub, and then configuring PythonAnywhere to recognise it and start serving it as a web application. There are manual ways of doing it, but PythonAnywhere provides a helper tool that will do it all for you. Let's install it first:

PythonAnywhere command-line

```
$ pip3.8 install --user pythonanywhere
```

That should print out some things like `Collecting pythonanywhere`, and eventually end with a line saying `Successfully installed (...) pythonanywhere- (...)`.

Now we run the helper to automatically configure our app from GitHub. Type the following into the console on PythonAnywhere (don't forget to use your GitHub username in place of `<your-github-username>`, so that the URL matches the clone URL from GitHub):

PythonAnywhere command-line

```
$ pa_autoconfigure_django.py --python=3.8 https://github.com/<your-github-username>/my-firs
```

As you watch that running, you'll be able to see what it's doing:

- Downloading your code from GitHub
- Creating a virtualenv on PythonAnywhere, just like the one on your own computer
- Updating your settings file with some deployment settings
- Setting up a database on PythonAnywhere using the `manage.py migrate` command
- Setting up your static files (we'll learn about these later)
- And configuring PythonAnywhere to serve your web app via its API

On PythonAnywhere all those steps are automated, but they're the same steps you would have to go through with any other server provider.

The main thing to notice right now is that your database on PythonAnywhere is actually totally separate from your database on your own computer, so it can have different posts and admin accounts. As a result, just as we did on your own computer, we need to initialize the admin account with `createsuperuser`. PythonAnywhere has automatically activated your virtualenv for you, so all you need to do is run:

PythonAnywhere command-line

```
(ola.pythonanywhere.com) $ python manage.py createsuperuser
```

Type in the details for your admin user. Best to use the same ones as you're using on your own computer to avoid any confusion, unless you want to make the password on PythonAnywhere more secure.

Now, if you like, you can also take a look at your code on PythonAnywhere using `ls` :

PythonAnywhere command-line

```
(ola.pythonanywhere.com) $ ls
blog  db.sqlite3  manage.py  mysite requirements.txt static
(ola.pythonanywhere.com) $ ls blog/
__init__.py  __pycache__  admin.py  apps.py  migrations  models.py
tests.py  views.py
```

You can also go to the "Files" page and navigate around using PythonAnywhere's built-in file browser. (From the Console page, you can get to other PythonAnywhere pages from the menu button in the upper right corner. Once you're on one of the pages, there are links to the other ones near the top.)

You are now live!

Your site should now be live on the public Internet! Click through to the PythonAnywhere "Web" page to get a link to it. You can share this with anyone you want. :)

> **Note** This is a beginners' tutorial, and in deploying this site we've taken a few shortcuts which aren't ideal from a security point of view. If and when you decide to build on this project, or start a new project, you should review the Django deployment checklist (https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/) for some tips on securing your site.

Debugging tips

If you see an error while running the `pa_autoconfigure_django.py` script, here are a few common causes:

- Forgetting to create your PythonAnywhere API token.
- Making a mistake in your GitHub URL
- If you see an error saying *"Could not find your settings.py"*, it's probably because you didn't manage to add all your files to Git, and/or you didn't push them up to GitHub successfully. Have another look at the Git section above

- If you previously signed up for a PythonAnywhere account and had an error with collectstatic, you probably have an older version of SQLite (eg 3.8.2) for your account. In that case, sign up for a new account and try the commands in the PythonAnywhere section above.

If you see an error when you try to visit your site, the first place to look for some debugging info is in your **error log**. You'll find a link to this on the PythonAnywhere "Web" page (https://www.pythonanywhere.com/web_app_setup/). See if there are any error messages in there; the most recent ones are at the bottom.

There are also some general debugging tips on the PythonAnywhere help site (http://help.pythonanywhere.com/pages/DebuggingImportError).

And remember, your coach is here to help!

Check out your site!

The default page for your site should say "It worked!", just like it does on your local computer. Try adding `/admin/` to the end of the URL, and you'll be taken to the admin site. Log in with the username and password, and you'll see you can add new Posts on the server -- remember, the posts from your local test database were not sent to your live blog.

Once you have a few posts created, you can go back to your local setup (not PythonAnywhere). From here you should work on your local setup to make changes. This is a common workflow in web development – make changes locally, push those changes to GitHub, and pull your changes down to your live Web server. This allows you to work and experiment without breaking your live Web site. Pretty cool, huh?

Give yourself a *HUGE* pat on the back! Server deployments are one of the trickiest parts of web development and it often takes people several days before they get them working. But you've got your site live, on the real Internet!