Django

Install python 3.7 for windows

Install Visual Studio Code or pyCharm.

## ****Installing Pip****

Once you’ve confirmed that Python is correctly installed, you can proceed with installing Pip.

1. Download [get-pip.py](https://bootstrap.pypa.io/get-pip.py) to a folder on your computer.
2. Open a command prompt and navigate to the folder containing get-pip.py.
3. Run the following command:python get-pip.py
4. Pip is now installed!

Create GIT account

# Create a PythonAnywhere account

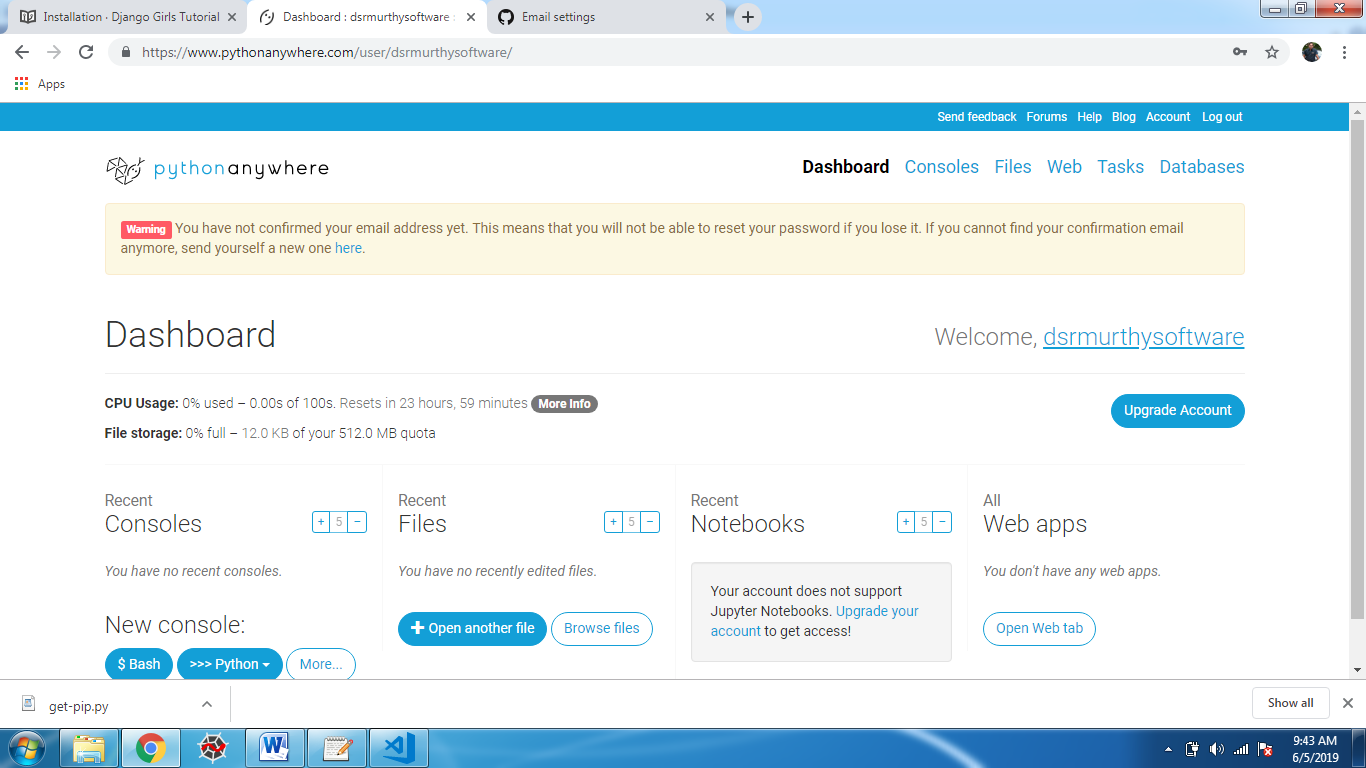
PythonAnywhere is a service for running Python code on servers "in the cloud". We'll use it for hosting our site, live and on the Internet.

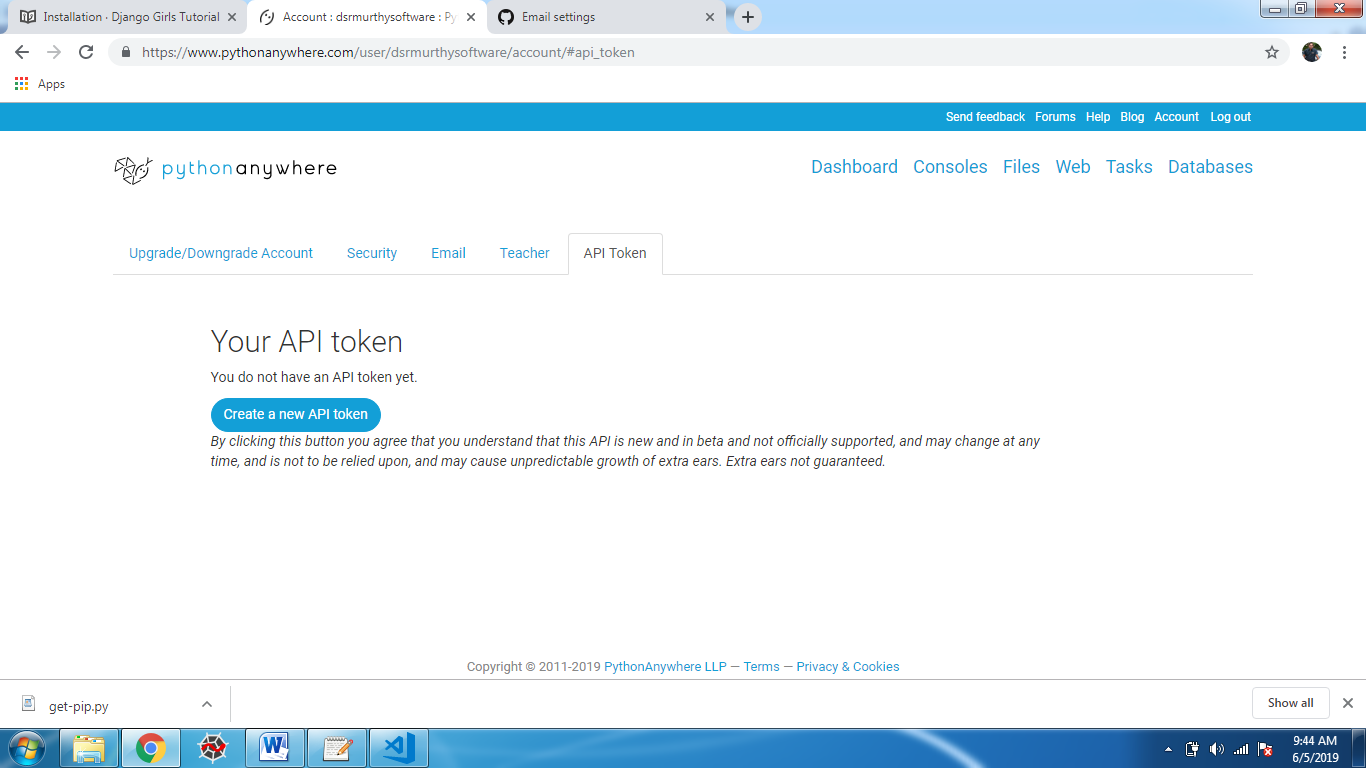
We will be hosting the blog we're building on PythonAnywhere. Sign up for a "Beginner" account on PythonAnywhere (the free tier is fine, you don't need a credit card).

* [www.pythonanywhere.com](https://www.pythonanywhere.com/)

Signup / Create Beginner account

[dsrmurthysoftware/welcome@786/dsrmurthysoftware@gmail.com](mailto:dsrmurthysoftware/welcome@786/dsrmurthysoftware@gmail.com)





dfe533b4166e34ac3ea0408aa3f25fe15952c3b9

Use this token for our API by setting a request header called Authorization, followed by Token <token>, eg:

import requests

my\_domain = 'dsrmurthysoftware.pythonanywhere.com'

username = 'dsrmurthysoftware'

token = 'dfe533b4166e34ac3ea0408aa3f25fe15952c3b9'

response = requests.post(

'https://www.pythonanywhere.com/api/v0/user/{username}/webapps/{domain}/reload/'.format(

username=username, domain=my\_domain

),

headers={'Authorization': 'Token {token}'.format(token=token)}

)

if response.status\_code == 200:

print('All OK')

else:

print('Got unexpected status code {}: {!r}'.format(response.status\_code, response.content))

 Your API token is also available from pythonanywhere consoles, tasks and webapps, as an Environment Variable, $API\_TOKEN

Virtual Environment:

Virtualenv will isolate your Python/Django setup on a per-project basis. This means that any changes you make to one website won't affect any others you're also developing.

find a directory in which you want to create the virtualenv; your home directory, for example. On Windows, it might look like C:\Users\Name\

$ mkdir jangodemo

$ cd jangodemo

$ python -m venv myvenv

The command above will create a directory called myvenv

Upgrade pip:

$ python -m pip install --upgrade pip

**Activate your environment:**

**C:\>cd jangodemo**

**C:\jangodemo>myvenv\scripts\activate**

**<myvenv> c:\jangodemo**

**Installing packages with requirements**

A requirements file keeps a list of dependencies to be installed using pip install:

First create a requirements.txt file inside of the djangodemo/ folder.

Opening a new file in the code editor and then saving it as requirements.txt

Add below line in this file.

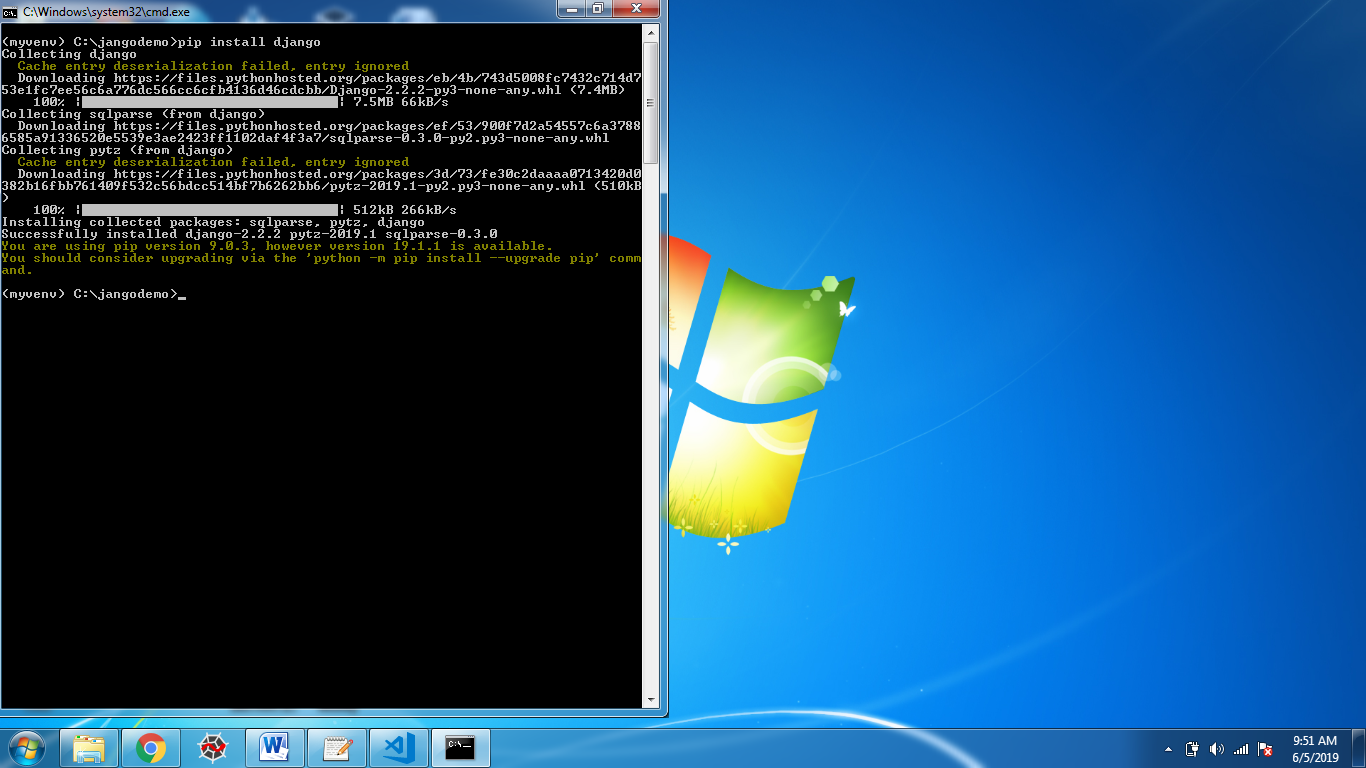
Django~=2.2.2

run :

<myvenv> c:\jangodemo\>pip install -r requirements.txt

to install Django

or <myvenv> c:\jangodemo> pip install Django==2.2.2



Create a project of Django

<myvenv> c:\jangodemo\>django-admin startproject murthysite .

django-admin.py is a script that will create the directories and files for you.

djangodemo

├───manage.py

├───murthysite

│ settings.py

│ urls.py

│ wsgi.py

│ \_\_init\_\_.py

└───requirements.txt

In settings.py, find the line that contains TIME\_ZONE and modify it to choose your own timezone. For example:

mysite/settings.py

TIME\_ZONE = 'Europe/Berlin'

When DEBUG is True and ALLOWED\_HOSTS is empty, the host is validated against ['localhost', '127.0.0.1', '[::1]']. This won't match our hostname on PythonAnywhere once we deploy our application so we will change the following setting:

mysite/settings.py

ALLOWED\_HOSTS = ['127.0.0.1', '.pythonanywhere.com']

Setup database

mysite/settings.py

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

To create a database for our blog, let's run the following in the console:

(myvenv) c:\jangodemo> python manage.py migrate

## Starting the web server

(myvenv) c:\jangodemo\> python manage.py runserver

Open browser and type :

<http://127.0.0.1:8000/>

Create blog application:

(myvenv) c:\jangodemo\> python manage.py startapp blog

jangodemo

├── blog

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── migrations

│ │ └── \_\_init\_\_.py

│ ├── models.py

│ ├── tests.py

| ├── urls.py

│ └── views.py

├── db.sqlite3

├── manage.py

├── murthysite

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

└── requirements.txt

After creating an application, we also need to tell Django that it should use it.

We do that in the file mysite/settings.py -- open it in code editor.

We need to find INSTALLED\_APPS and add a line containing 'blog', just above ]. So the final product should look like this:

mysite/settings.py

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'blog',

]

### Creating a blog post model

In the blog/models.py file we define all objects called Models – this is a place in which we will define our blog post.

Let's open blog/models.py in the code editor, remove everything from it, and write code like this:

blog/models.py

from django.conf import settings

from django.db import models

from django.utils import timezone

class Post(models.Model):

author = models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)

title = models.CharField(max\_length=200)

text = models.TextField()

created\_date = models.DateTimeField(default=timezone.now)

published\_date = models.DateTimeField(blank=True, null=True)

def publish(self):

self.published\_date = timezone.now()

self.save()

def \_\_str\_\_(self):

return self.title

### Create tables for models in your database

(myvenv) c:\jangodemo> python manage.py makemigrations blog

Migrations for 'blog':

blog/migrations/0001\_initial.py:

- Create model Post

Now apply migrations to create database.

(myvenv) c:\jangodemo> python manage.py migrate blog

Operations to perform:

Apply all migrations: blog

Running migrations:

Applying blog.0001\_initial... OK

# Django admin

To add, edit and delete the posts we've just modeled, we will use Django admin.

Open the blog/admin.py file in the code editor and replace its contents with this:

blog/admin.py

from django.contrib import admin

from .models import Post

admin.site.register(Post)

To login, create superuser:

(myvenv) C:\\jangodemo> python manage.py createsuperuser

Username: murthy , [murthy@gmail.com](mailto:murthy@gmail.com) / welcome@123

run python manage.py runserver in the console to run the web server.

Go to your browser and type the address <http://127.0.0.1:8000/admin/>.

Login with murthy/welcome@123

Click post and add 3 posts.

Now Deploy on pythonanywhere cloud:

Install git in your system.

## Starting our Git repository

Git tracks changes to a particular set of files in what's called a code repository (or "repo" for short).

Open up git console and run these commands, in the jangodemo directory:

command-line

$ git init

Initialized empty Git repository in ~/jangodemo/.git/

$ git config --global user.name "dsrmurthysoftware"

$ git config --global user.email dsrmurthysoftware@gmail.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).

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 editor and create a new file with the following contents:

.gitignore

\*.pyc

\*~

\_\_pycache\_\_

myvenv

db.sqlite3

/static

.DS\_Store

And save it as .gitignore in the "jangodemo" folder.

It's a good idea to use a git status command before git add or whenever you find yourself unsure of what has changed.

The git status command returns information about any untracked/modified/staged files, the branch status, and much more.

$ git status

On branch master

Initial commit

Untracked files:

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

.gitignore

blog/

manage.py

murthysite/

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 --all .

$ git commit -m "My Django Girls app, first commit"

[...]

13 files changed, 200 insertions(+)

create mode 100644 .gitignore

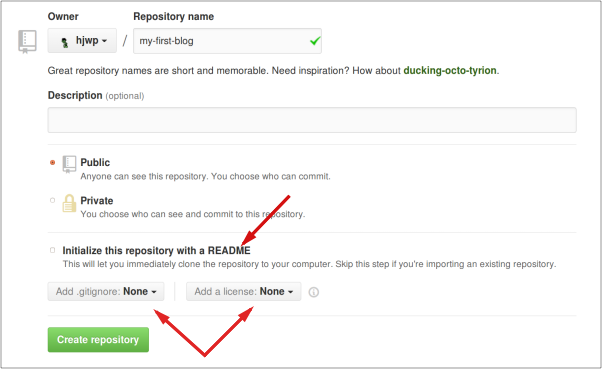
[...]

create mode 100644 mysite/wsgi.py

## Pushing your code to GitHub

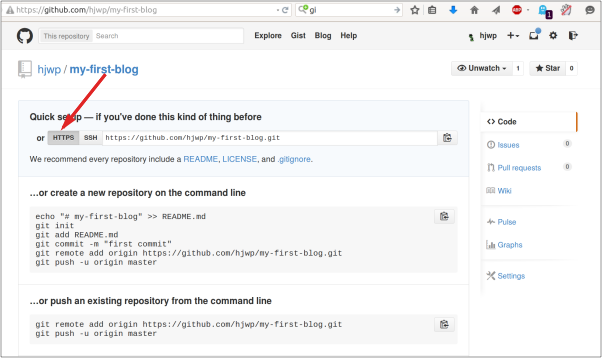
Go to [GitHub.com](https://www.github.com/) and sign up for a new, free user account. (If you already did that in the workshop prep, that is great!) Be sure to remember your password (add it to your password manager, if you use one).

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.



**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:



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 master

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] master -> master

Branch master set up to track remote branch master 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

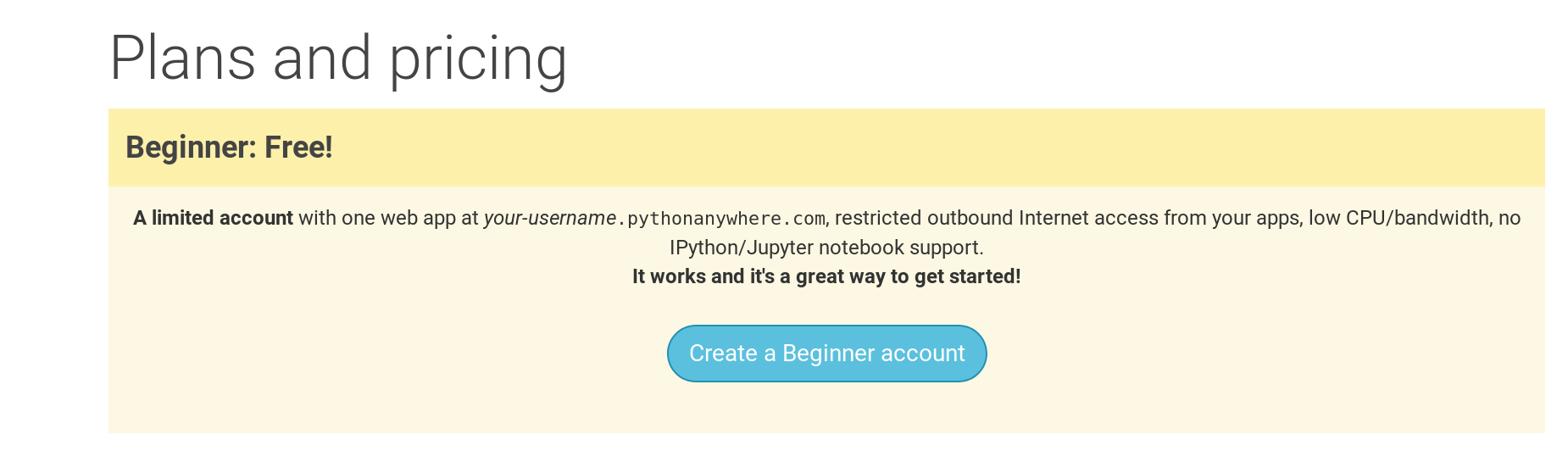
## Sign up for a PythonAnywhere account

**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).

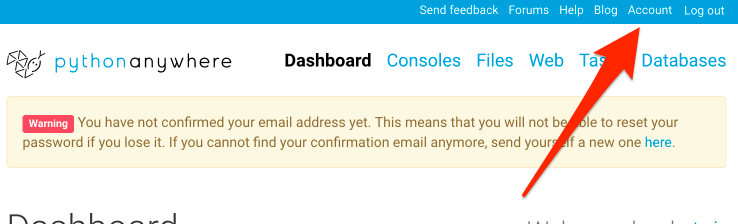
* [www.pythonanywhere.com](https://www.pythonanywhere.com/)



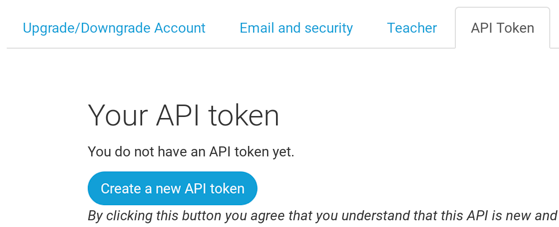
**Note** When choosing your username here, bear in mind that your blog's URL will take the form yourusername.pythonanywhere.com, so choose either your own nickname or a name for what your blog is all about. Also, be sure to remember your password (add it to your password manager, if you use one).

## 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:

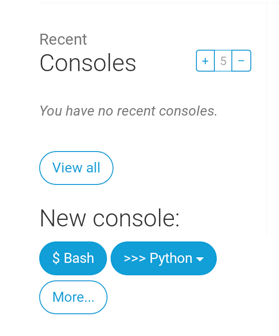


then select the tab named "API token", and hit the button that says "Create new API token".



## 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.



**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

$ pip3.6 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 https://github.com/<your-github-username>/my-first-blog.git

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/2.0/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 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!