

Free Flask Tutorial: Initialization of Flask, Implement Plotly and Publish on Hosting System

By Antonio Blago

Subscribe to my newsletter: www.antoniblago.com

email: info@antoniblago.com

2022-10-03

Originally published on medium.com

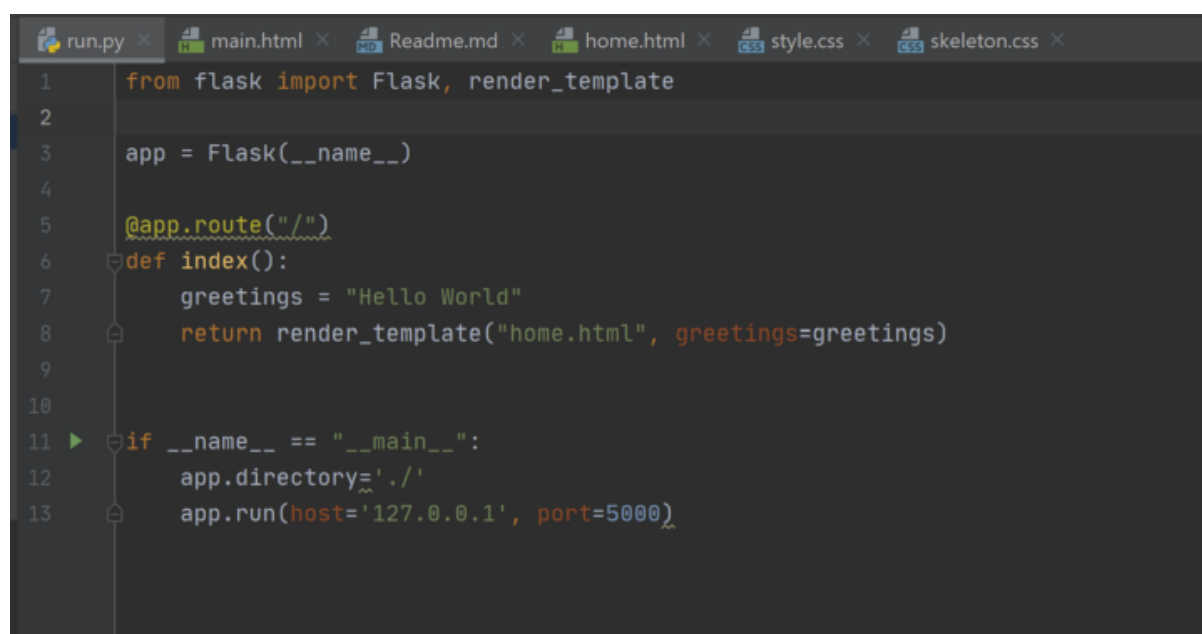
Content

1. Build Your Own Mobile-Friendly Website With Flask	3
a. Introduction to Flask.....	3
b. Requirements	4
c. Development Framework.....	4
d. Initialize your first project	5
e. Write your first python file.....	6
f. Prepare your folder structure.....	9
g. Insert your CSS.....	9
h. Create your first HTML file.....	10
i. Create a responsive navigation bar	12
2. Develop your own financial dashboard with Flask and Plotly.....	20
a. Introduction.....	20
b. Implementation in Flask.....	25
3. Flask Tutorial: How to Deploy & Publish an App on pythonanywhere.....	33
a. Overview	33
b. Create a beginner account.....	33
c. Initialize a web app.....	35
d. Available Plans	38
e. Setup Backend	40
f. Web Configuration	41

1. Build Your Own Mobile-Friendly Website With Flask

TLDR: With Flask you can build fast and easy web apps and websites. I recommend PyCharm or vs code for development. On pythonanywhere.com* you can deploy and host your own apps. On Github, you can fork or download this template ([Repository](#)). Please give it a star. Thank you!

Date: May 23, 2021

A screenshot of a code editor with a dark theme. The editor has several tabs at the top: 'run.py', 'main.html', 'Readme.md', 'home.html', 'style.css', and 'skeleton.css'. The 'run.py' tab is active, showing Python code for a Flask application. The code includes imports for Flask and render_template, the creation of a Flask app, a route for the root path that returns a 'Hello World' greeting, and a main block that runs the app on localhost:5000.

```
1 from flask import Flask, render_template
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def index():
7     greetings = "Hello World"
8     return render_template("home.html", greetings=greetings)
9
10
11 if __name__ == "__main__":
12     app.directory='.'
13     app.run(host='127.0.0.1', port=5000)
```

a. Introduction to Flask

As a regular python user, I dreamed about building my own webpage with python, because I knew it is possible. Even though there are a lot of tutorials of how to build a website with Flask, there are very only few stackoverflow snippets about building a mobile friendly web page.

Integrating a responsive navigation bar is the first important step for that. Nowadays it is really necessary to have that, because around **60%** of the people are surfing with their smartphones (Link to recent statistics

2021: [Here](#)). So, if you want an attractive web page, it should be mobile friendly!

In this tutorial I show you how to develop and build the basics for a mobile friendly website. Everything I show you here, I have gathered in the internet, so it is just a collection and step-by-step guide for you.

In a next step and different tutorial I will show you how to deploy your own web page on pythonanywhere.com*, where I also host my website. For this tutorial you should know a little about Python, HTML and CSS.

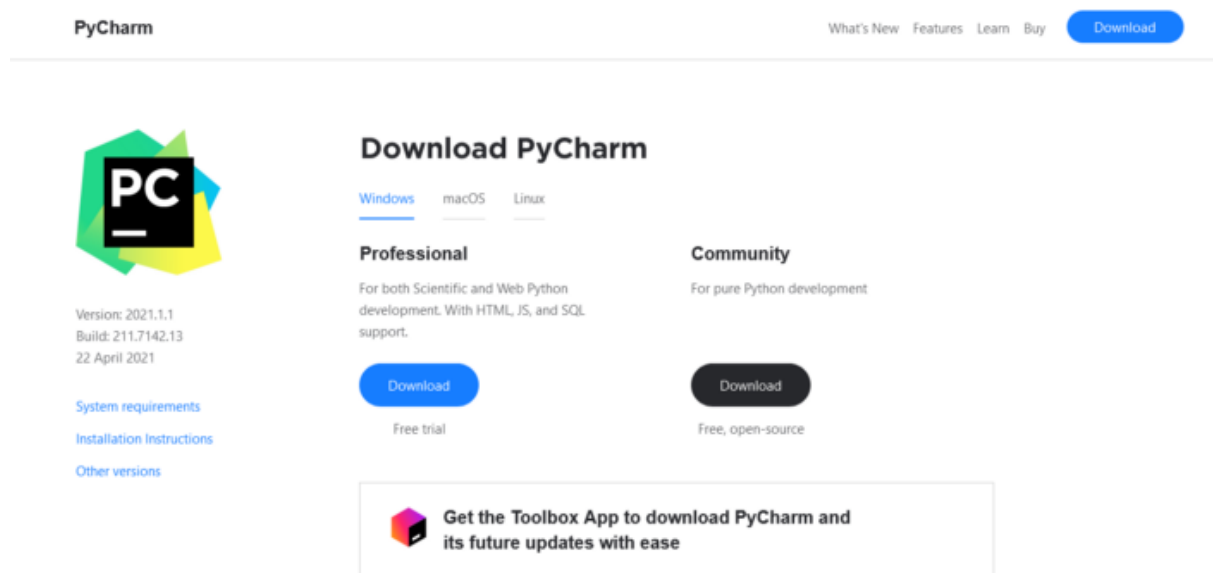
b. Requirements

- Python 3.8+ or Anaconda
- Integrated Development Environment (IDE)

c. Development Framework

To develop your webpage from scratch, you need an IDE (Integrated Development Environment). For the IDE I personally use and recommend [PyCharm](#). It is easy to use, handy with Version Control (Git), installing and updating libraries and using virtual environments. Of course, you can also use any other IDE like Visual Studio Code or Atom. If you have already PyCharm or a similar IDE, you can skip this part.

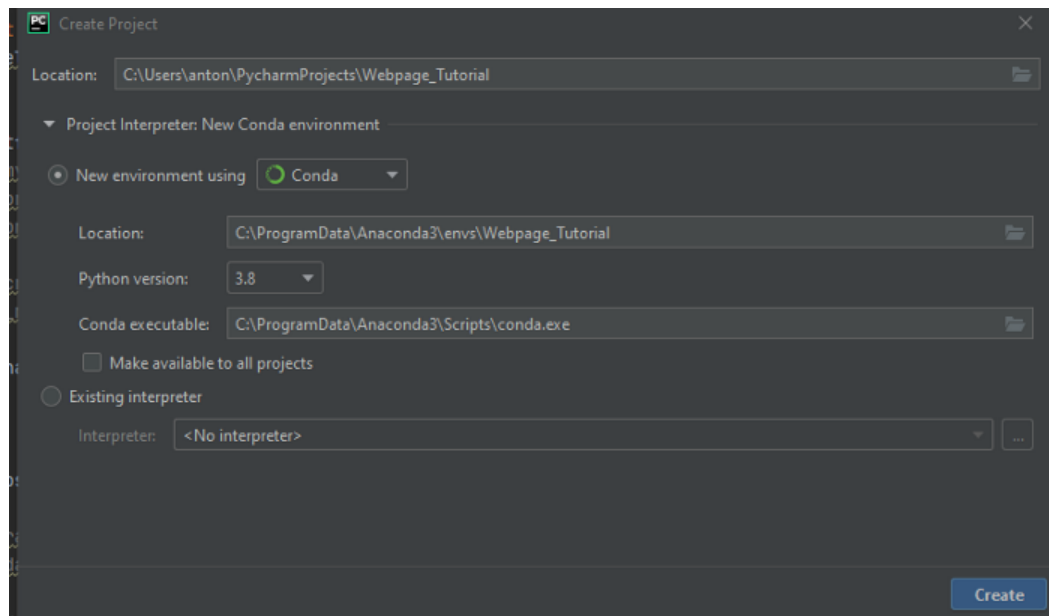
Download and install the program from PyCharm for your dedicated system (Windows, macOS or Linux). The community version is free and enough for this use case. I am using Windows in this tutorial.



d. Initialize your first project

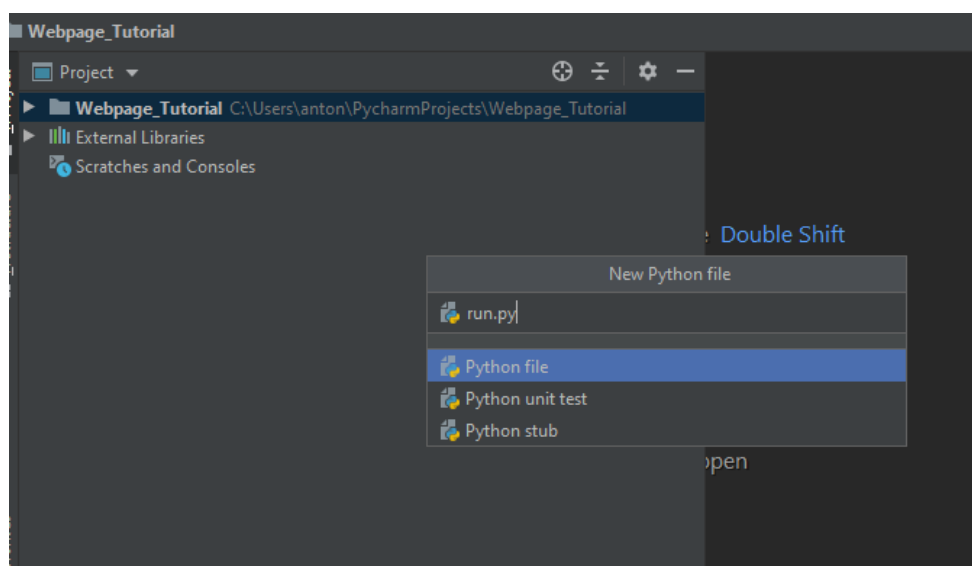
When successfully installed, start PyCharm and go to *File > New Project*. A window pops up and you can name your project and specify the directory on your computer. I named it “Webpage Tutorial”. Now you can choose if you want a virtual environment or use an existing interpreter. I recommend setting up a new virtual environment with “conda” (Anaconda) or any other virtual environment, because it is much simpler to handle your project with your libraries, scaling it up or deploying it. If you do not have Anaconda, you can also simply download and install it (<https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html>).

When you click the “Create” button, PyCharm starts to initialize your project and installs the necessary libraries.

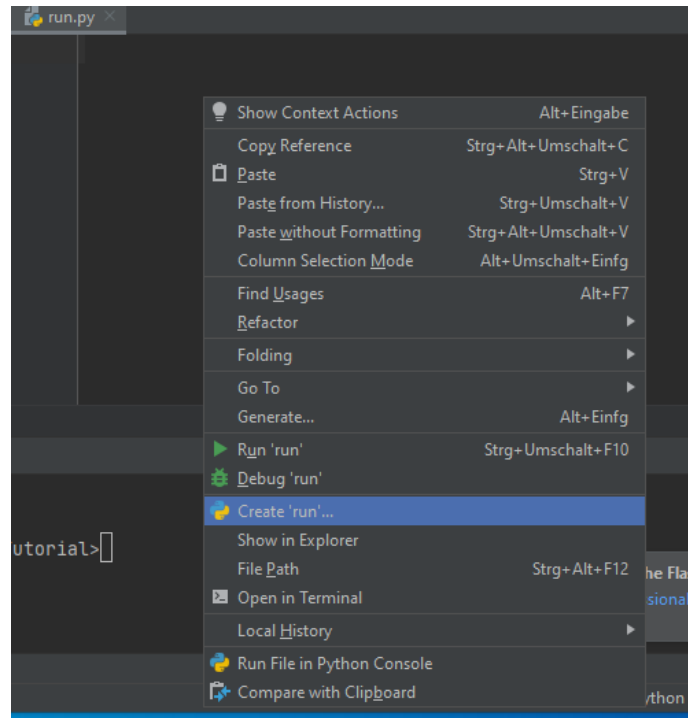


e. Write your first python file

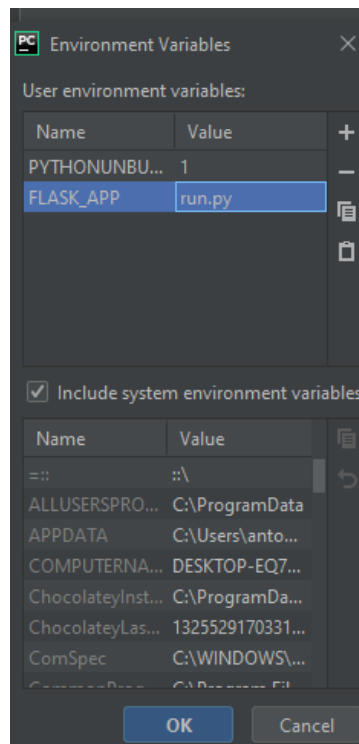
First you will see an empty root directory in PyCharm and you can create your first python file by right clicking on *Webpage_Tutorial* folder > *New > Python File*. Name the python file “run.py”, that is your core file, which will handle all your other files like html and css files. It is your “Backend”, what the user will not see.



Afterwards right click on the empty file and click *Create 'run'* to configure your python file.



Add to the Enviroment variables `FLASK_APP=run.py` and click *OK* and save the configuration.



Open your Terminal in the bottom left corner and install Flask.

```
pip install flask == 1.1.2
```

Write your first browser app in the run.py:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello World!"

if __name__ == "__main__":
    app.directory='./'
    app.run(host='127.0.0.1', port=5000)
```


Now you are able to click on the right hand side in the top corner, the green run button. In the bottom, it opens the RUN Terminal. You can now click on the link 127.0.0.1:5000 which is your localhost. The browser should open and you should see “Hello World”.



```
C:\ProgramData\Anaconda3\envs\Webpage_Tutorial\python.exe C:/Users/anton/PycharmProjects/Webpage_Tutorial/run.py
* Serving Flask app "run" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/May/2021 08:19:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/May/2021 08:19:10] "GET /favicon.ico HTTP/1.1" 404 -
```

f. Prepare your folder structure

Insert your main root directory, the folders for your static files like your css and your templates folders, where the html files will be located.

```
Webpage_Tutorial/
|
├── static
│   └── css
├── templates
│   └── layouts
└── run.py
```

g. Insert your CSS

Download from <http://getskeleton.com/> the CSS template, extract the zip file and copy-paste the two files normalize.css and skeleton.css into your CSS folder.


```
        {% block content%}{% endblock %}

    </div>

</div>
</body>
</html>
```

Add also your start view, which you can call “home.html”. Insert in your HTML file a replace holder where you can insert variables dynamically via your python backend: {{greetings}}.

```
{% extends "layouts/main.html" %}

{% block content%}

</body>

<h5>{{greetings}}</h5>

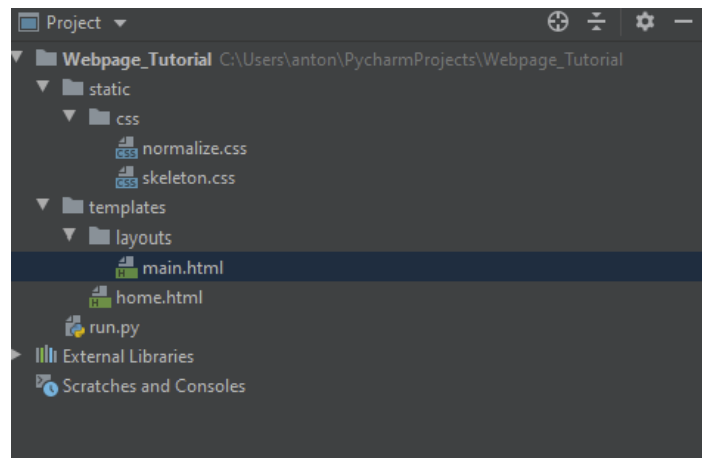
</body>

{% endblock %}
```

Change your run.py file, that you render your new home file.

```
@app.route("/")
def index():
    greetings = "Hello World"
    return render_template("home.html",
greetings=greetings)
```

Your folder directory should look like this:



Now you can run again your run.py to see if the stylesheets and your greetings are displayed correctly.

i. Create a responsive navigation bar

Now it will be more complicated with CSS and JavaScript. Create an additional CSS with the name “style.css” in your CSS path and insert following code:

```
/* Dropdown content (hidden by default) */
.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  z-index: 1;
}
/* Links inside the dropdown */
.dropdown-content a {
  float: none;
  color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
  text-align: left;
}
```

```
/* Add a grey background color to dropdown links on hover */
.dropdown-content a:hover {
    background-color: #ddd;
}

/* Show the dropdown menu on hover */
.dropdown:hover .dropdown-content {
    display: block;
}

/* Add a black background color to the top navigation */
.topnav {
    background-color: #333;
    overflow: hidden;
}

/* Style the links inside the navigation bar */
.topnav a {
    float: left;
    display: block;
    color: #f2f2f2;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
    font-size: 17px;
}

/* Add an active class to highlight the current page */
.active {
    background-color: grey;
    color: white;
}

/* Hide the link that should open and close the topnav on
small screens */
.topnav .icon {
    display: none;
}

/* Dropdown container - needed to position the dropdown
content */
.dropdown {
    float: left;
    overflow: hidden;
}
```

```
/* Style the dropdown button to fit inside the topnav */
.dropdown .dropbtn {
  font-size: 17px;
  border: none;
  outline: none;
  color: white;
  padding: 14px 16px;
  background-color: inherit;
  font-family: inherit;
  line-height: 30px;
}

/* Style the dropdown content (hidden by default) */
.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  z-index: 1;
}

/* Style the links inside the dropdown */
.dropdown-content a {
  float: none;
  color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
  text-align: left;
}

/* Add a dark background on topnav links and the dropdown
button on hover */
.topnav a:hover, .dropdown:hover .dropbtn {
  background-color: #555;
  color: white;
}

/* Add a grey background to dropdown links on hover */
.dropdown-content a:hover {
  background-color: #ddd;
  color: black;
}
```

```
/* Show the dropdown menu when the user moves the mouse
over the dropdown button */
.dropdown:hover .dropdown-content {
    display: block;
}

/* When the screen is less than 600 pixels wide, hide all
links, except for the first one ("Home"). Show the link
that contains should open and close the topnav (.icon) */
@media screen and (max-width: 600px) {
    .topnav a:not(:first-child), .dropdown .dropbtn {
        display: none;
    }
    .topnav a.icon {
        float: right;
        display: block;
    }
}

@media screen and (max-width: 600px) {
    .topnav.responsive {position: relative;}
    .topnav.responsive a.icon {
        position: absolute;
        right: 0;
        top: 0;
    }
    .topnav.responsive a {
        float: none;
        display: block;
        text-align: left;
    }
    .topnav.responsive .dropdown {float: none;}
    .topnav.responsive .dropdown-content {position:
relative;}
    .topnav.responsive .dropdown .dropbtn {
        display: block;
        width: 100%;
        text-align: left;
    }
}

</style>
```

Now change your *main.html* to make the additional CSS file effective.

You need for the small icons also a source from cloudflare:

```
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
```

Also add a dropdown field for additional HTML pages, but this is only a placeholder. I am calling it “Examples” and the dropdown “Example 1”.

```
<div class="dropdown">
    <button class="dropbtn">Examples
        <i class="fa fa-caret-down"></i>
    </button>
    <div class="dropdown-content">
        <a href="/Example1">Example 1</a>
    </div>
</div>
```

Insert also the javascript snippet to control the responsiveness navigation bar.

```
<script>
function myFunction() {
    var x = document.getElementById("myTopnav");
    if (x.className === "topnav") {
        x.className += " responsive";
    } else {
        x.className = "topnav";
    }
}
</script>
```

All together it will look like that:


```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <title>This is a tutorial </title>
    <meta name="description" content="I present to you
a free tutorial for developing a mobile friendly app"/>
    <meta name="keywords" content="Tutorial, Mobile,
Free"/>
    <meta name="author" content="<author>" />
    <meta name="copyright" content="<author>" />
    <meta name="robots" content="index"/>
    <meta name="robots" content="follow"/>
    <meta name="msapplication-TileColor"
content="#da532c">
    <meta name="theme-color" content="#ffffff">

    <link rel="stylesheet"
href="./static/css/normalize.css"/>
    <link rel="stylesheet"
href="./static/css/skeleton.css"/>
    <link rel="stylesheet"
href="./static/css/style.css"/>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">

  </head>
  <body>
    <div class="container">
      <div class="topnav" id="myTopnav">
        <a href="/" class="active">Home</a>
        <div class="dropdown">
          <button class="dropbtn">Examples
            <i class="fa fa-caret-down"></i>
          </button>
          <div class="dropdown-content">
            <a href="/Example1">Example 1 </a>
          </div>
        </div>
        <a href="javascript:void(0);" class="icon">
```

```
onclick="myFunction()">
    <i class="fa fa-bars"></i>
  </a>
</div>

  <div style="clear: both">
    &nbsp;

    {% block content%}{% endblock %}

  </div>

<script>
function myFunction() {
  var x = document.getElementById("myTopnav");
  if (x.className === "topnav") {
    x.className += " responsive";
  } else {
    x.className = "topnav";
  }
}
</script>

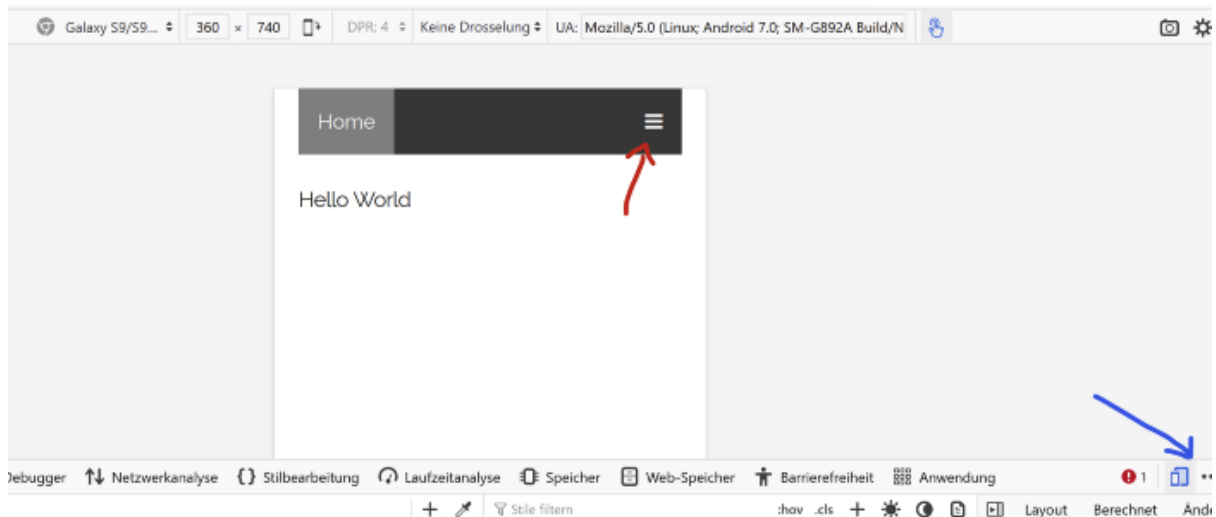
</div>
</body>
</html>
```

When you made the changes and reload your pages, the result should look like that:

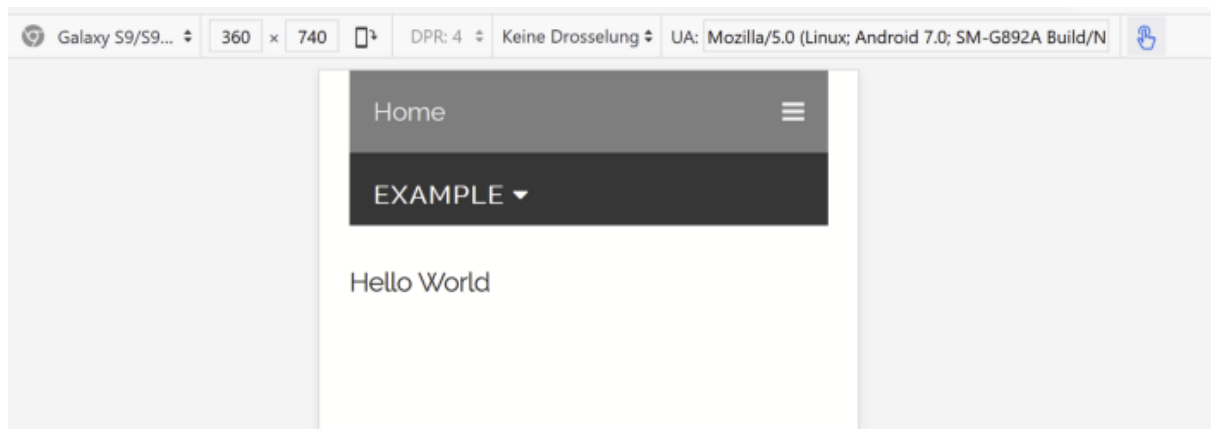


Also, you can test if the page will change its design by a different device like a smartphone. In Firefox or Chrome, you can enter the developer console with F12 and test the smartphone view. In Firefox, it is a small

icon in the right bottom corner (blue arrow). As demonstration your navigation bar is smaller and your subitems can be accessed through the doughnut (red arrow).



If you click the doughnut (red arrow) the navigation item will pop out and show its content:



Congratulations! You have developed a mobile friendly website with flask. Just upload it now to pythonanywhere.com*!

On GitHub you can fork or download this template ([Repository](#)). Please give it a star. Thank you!

2. Develop your own financial dashboard with Flask and Plotly

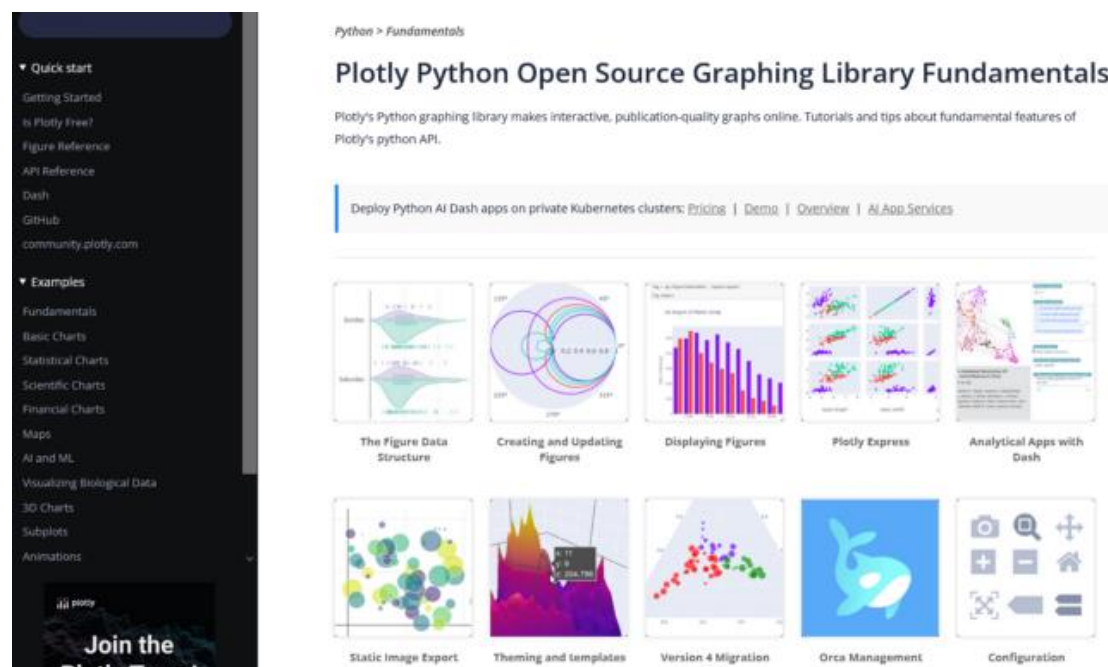
TLDR; Flask is a great tool based on python to build scalable and mobile friendly apps. With Plotly you can create beautiful plots which are automatically fitting to the screen size. I am using pythonanywhere.com* to run my scripts and host my apps. Check it out!

Apr 11 2022



a. Introduction

I created a short video (see below) to show you a quick overview of the dashboard. As you can see Plotly delivers all kind of possibilities to customize your plots. On <https://Plotly.com/python/> is a huge collection of examples and plot types, see the screenshot below.



What I mostly like is the interactivity with the plot. You can zoom in and out, select certain areas or disable elements in the plot. See my dashboard in action: www.tenxassets.com/stock data ([hosted on pythonanywhere.com](http://pythonanywhere.com)*).

Following, I will show you how I created the plots with Plotly. First, install the required packages for yfinance (<https://pypi.org/project/yfinance/>) and Plotly (<https://pypi.org/project/Plotly/>). Afterwards, import them, also in this example I required pandas.

```
import yfinance as yf
import pandas as pd
import plotly.express as px
```

Next, we get the data via the API from yfinance and reformat them to our needs. In this example, I will create a plot for the historical dividends from AAPL.

```
## we create first a ticker object to call the data
ticker = yf.Ticker("AAPL")

## With the ticker oobject we can call the dividends from
## AAPL
dividends = ticker.get_dividends()

## it is a pandas series, so we have to reformat to a
dataframe
dividends = dividends.to_frame()

## reset the index to have the date as a column
dividends = dividends.reset_index()

## we extract the year from the date column to aggregate it
to year
dividends["year"] = dividends["Date"].dt.year
dividends = dividends.groupby("year")["Dividends"].sum()
dividends = dividends.to_frame()
dividends = dividends.reset_index()

## Print the dataframe et voila we have for each year the
total sum of divididens in US Dollar

>> dividends
   year  Dividends
0  1987    0.001786
1  1988    0.003036
2  1989    0.003661
3  1990    0.004018
4  1991    0.004286
5  1992    0.004286
6  1993    0.004286
7  1994    0.004286
8  1995    0.004286
9  2012    0.189286
10 2013    0.421429
11 2014    0.461429
```

Now, we create a function to style the plot, otherwise it will look unprofessional. Also, I recommend you put the code into a function

because it simplifies and cleans up your coding script. Additionally, this will be easier in maintaining or changing the style of your plots.

```
## the function takes the figure object from Plotly and can
update it according to the layout. Here i suggested some
inputs like title for x- and y-axis, but you can of course
design it your own needs.

def fig_layout(fig, ytitle, ytickformat, xtitle, ticker,
legendtitle, type_of_plot, yaxis_tickprefix=None):
    fig.update_layout(
        yaxis={
            "title": ytitle,
            "tickformat": ytickformat,

        },
        yaxis_tickprefix = yaxis_tickprefix,
        paper_bgcolor="#FFFFFF",

    ## background of the plot
        plot_bgcolor="#FFFFFF",
        autosize=True,

    ## fits the plot according to its box
        legend=dict(
            title=legendtitle,
            yanchor="top",
            y=0.99,
            xanchor="left",
            x=0.01
        ),
        title={
            'text': '{} - {}
<br><sup>tenxassets.com</sup>'.format(type_of_plot,ticker),
            'y': 0.85,
            'x': 0.5,
            'xanchor': 'center',
            'yanchor': 'top'},
        titlefont=dict(
            size=12,
            color="black"),

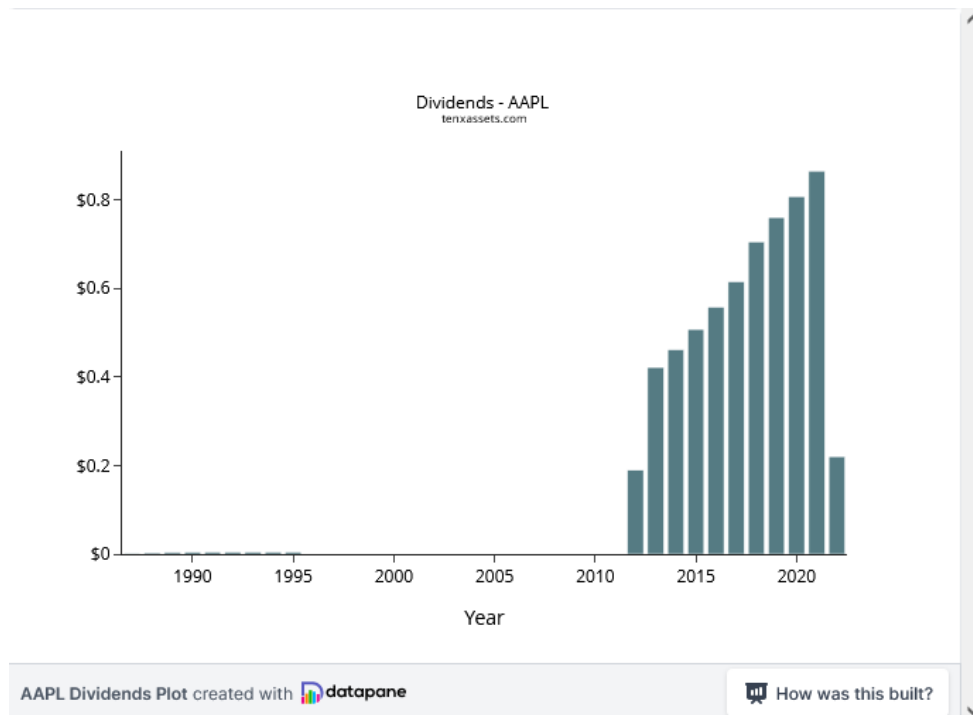
        template="simple_white",
        xaxis=dict(
```

```
        title=xtitle,  
        showticklabels=True),  
    showlegend=True,  
    font=dict(  
        family="Courier New, monospace",  
        size=12,  
        color="black"  
    ),  
)  
return fig
```

As next step, we must create a color palette for the plot. This is also important for a professional and clean look. There are a lot of websites to create different types of color palettes. I chose this one:

<https://colorhunt.co/palette/557b8339aea9a2d5abe5efc1>.

```
## Copy and paste the hex codes into a list  
color_palette = ["#557B83", "#39AEA9", "#A2D5AB", "#E5EFC1"]  
  
## define the plot type, in this case it is a bar plot  
fig4 = px.bar(dividends,  
              x="year",  
              y="Dividends",  
              title="Dividends",  
              barmode='group',  
              color_discrete_sequence =color_palette[:1])  
  
fig4 = fig_layout(fig4, ytitle= "", ytickformat = None,  
                  xtitle= "Year", ticker= "AAPL", legendtitle = "Debt and  
Liabilites", type_of_plot = "Dividends",  
                  yaxis_tickprefix='$',)  
  
## display the plot in your browser  
fig4.show()
```

b. Implementation in Flask

Now, we want to implement the Plotly graph in Flask. You may remember my last tutorial on how to “Build your own mobile friendly website with Flask”. Here is the [link](#) to the article. See below a simple example of a flask website.

```
from flask import Flask, render_template

app = Flask(__name__)

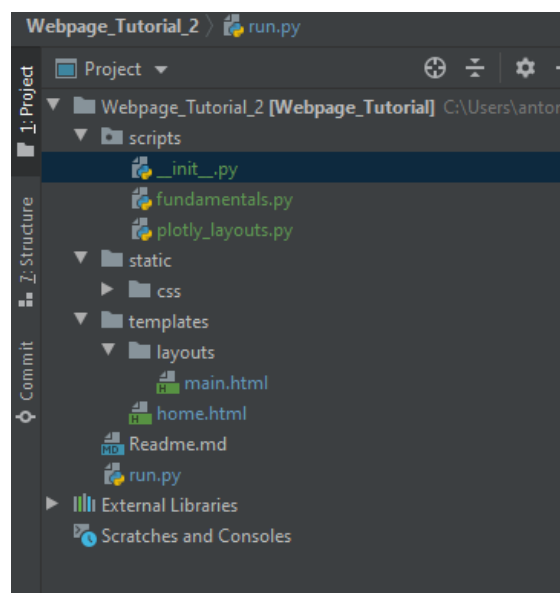
@app.route("/")
def index():
    greetings = "Hello World"
    return render_template("home.html",
greetings=greetings)

if __name__ == "__main__":
    app.directory='./'
    app.run(host='127.0.0.1', port=5000)
```

Output from the code above:



Now we will extend the project with the functions above. Therefore, we create a new folder named **scripts** and put there three new python scripts. The `fundamentals.py` is to get the data from yfinance and put it in the right format to hand it over to the `plotly_layout.py`. This script is creating the Plotly layout of the plot. The `__init__.py` is to call the functions from the `run.py`. It is an empty script.



fundamentals.py

```
import yfinance as yf
import pandas as pd
import plotly_layouts as ply
import datetime as dt

def get_dividends(ticker):

    ticker = yf.Ticker(ticker)

    ## Dividends
    dividends = ticker.get_dividends()
    dividends = dividends.to_frame()
    dividends = dividends.reset_index()
    dividends["year"] = dividends["Date"].dt.year
    dividends =
dividends.groupby("year")["Dividends"].sum()
    dividends = dividends.to_frame()
    dividends = dividends.reset_index()

    return ply.create_plotly(dividends)

if __name__ == '__main__':
    get_dividends("AAPL")
```

plotly_layout.py

```
import plotly.express as px

def create_plotly(data):

    color_palette =
["#557B83", "#39AEA9", "#A2D5AB", "#E5EFC1"]

    fig = px.bar(data,
                  x="year",
                  y="Dividends",
                  title="Dividends",
                  barmode='group',
```

```

        color_discrete_sequence
=color_palette[:1])
    fig = fig_layout(fig, ytitle= "", ytickformat = None,
xtitle= "Year", ticker= "AAPL",
                    legendtitle = "Debt and Liabilites",
type_of_plot = "Dividends", yaxis_tickprefix='$',)
    fig

    return fig

def fig_layout(fig, ytitle, ytickformat, xtitle,ticker,
legendtitle, type_of_plot, yaxis_tickprefix=None):
    fig.update_layout(
        yaxis={
            "title": ytitle,
            "tickformat": ytickformat,

        },
        yaxis_tickprefix = yaxis_tickprefix,
        paper_bgcolor="#FFFFFF", # rgba(0,0,0,0)',
        plot_bgcolor="#FFFFFF", # 'rgba(0,0,0,0)',
        # autosize=True,
        legend=dict(
            title=legendtitle,
            yanchor="top",
            y=0.99,
            xanchor="left",
            x=0.01
        ),
        title={
            'text': '{} - {}
<br><sup>tenxassets.com</sup>'.format(type_of_plot,ticker),
            'y': 0.85,
            'x': 0.5,
            'xanchor': 'center',
            'yanchor': 'top'},
        titlefont=dict(
            size=12,
            color="black"),

        template="simple_white",
        xaxis=dict(
            title=xtitle,
            showticklabels=True),
        showlegend=True,

```

```
        font=dict(
            # family="Courier New, monospace",
            size=12,
            color="black"
        ),
    )
    return fig
```

Now, we want to call the fundamentals script in flask to create the plot in our website. We modify our run.py with new import for plotly and fundamentals.

```
from flask import Flask, render_template
import plotly
import json
import scripts.fundamentals as funds ## fundamentals.py

app = Flask(__name__)

@app.route("/")
def index():
    greetings = "Hello World, this is my first plotly plot"

    ## This gets the data and create a plot
    plot = funds.get_dividends("AAPL")      ## To display
    the plot as html we have to put into a json      ##
    format.
    plotly_plot = json.dumps(plot,
    cls=plotly.utils.PlotlyJSONEncoder)

    return render_template("home.html",
    greetings=greetings, plotly_plot= plotly_plot)

if __name__ == "__main__":
    app.directory='./'
    app.run(host='127.0.0.1', port=5000)
```

Also, adapt the home.html. You need a java script implementation for plotly. `<script src="//cdn.plot.ly/plotly-latest.min.js"></script>`. As well, define a `<div> </div>` container for the plot.

```
{% extends "layouts/main.html" %}{% block head %}
<script src="//cdn.plot.ly/plotly-latest.min.js"></script>
{% endblock %}

{% block content%}

</body>

<h5>{{greetings}}</h5><!-- Plotly implementation -->
<div>
    <div id="plotly-stock-candle"></div>

        <script>
            //Parse your Json variable here
            var graphs = {{ plotly_plot | safe }};
            Plotly.plot('plotly-stock-candle', graphs,
        {{}});
        </script>
    </div>

</body>

{% endblock %}
```

Next step is to modify your index.html in the `<head>` tag to import the plotly java script: `{% block head %} {% endblock %}`.

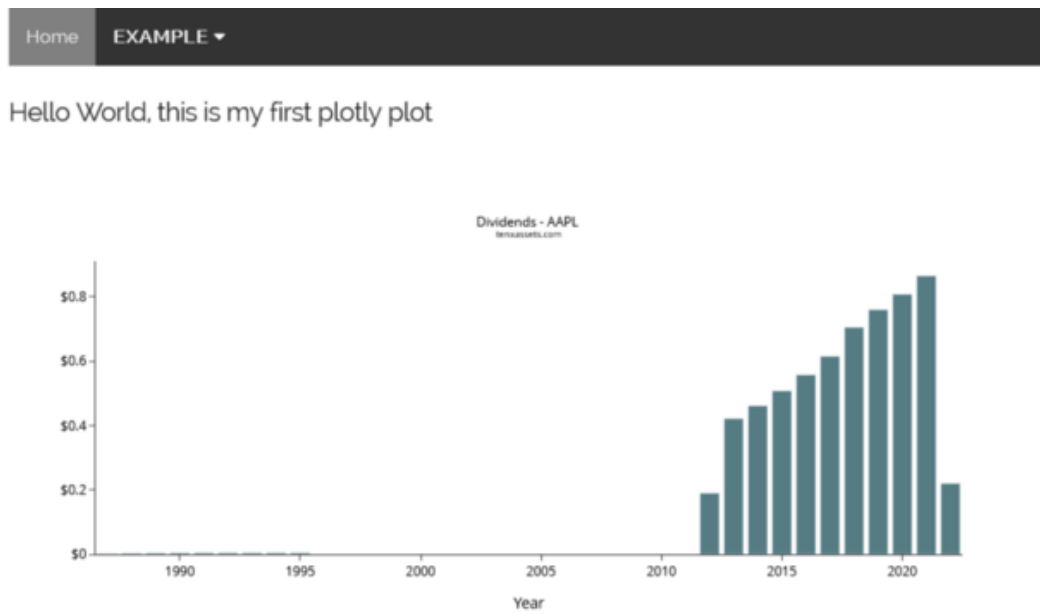
```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <title>This is a tutorial </title>
    <meta name="description" content="I present to you
a free tutorial for developing a mobile friendly app"/>
    <meta name="keywords" content="Tutorial, Mobile,
Free"/>
    <meta name="author" content="<author>" />
    <meta name="copyright" content="<author>" />
    <meta name="robots" content="index"/>
    <meta name="robots" content="follow"/>
    <meta name="msapplication-TileColor"
content="#da532c">
    <meta name="theme-color" content="#ffffff">

    <link rel="stylesheet"
href="./static/css/normalize.css"/>
    <link rel="stylesheet"
href="./static/css/skeleton.css"/>
    <link rel="stylesheet"
href="./static/css/style.css"/>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css"><!-- Plotly
implementation -->
    {% block head %} {% endblock %}
  <!-- end -->
</head>    <body>

.....
```

Now we run and see if it works. It should look like that:



On Github you can fork or download this template ([Repository](#)). Please give it a star. Thank you!

3. Flask Tutorial: How to Deploy & Publish an App on pythonanywhere

TLDR; A step-by-step guide on how to deploy and publish an app on pythonanywhere. In this tutorial, I describe every step on how to deploy and publish your app on pythonanywhere. I am personally really convinced by the hosting service. If you are creating an account via my affiliate links you support my work and further tutorials in this direction.

a. Overview

- Create a beginner account
- Initialize a web app
- Available Plans
- Setup Backend
- Web Configuration

b. Create a beginner account

Click this link to start: pythonanywhere.com*. You can start with a beginner account. It is totally free for three months and comes with HTTPS encryption and MySQL Database.

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!

Create a Beginner account

Education accounts

Are you a teacher looking for a place your students can code Python? You're not alone. Click through to find out more about our [Education beta](#).

All of our paid plans come with a no-quibble 30-day money-back guarantee — you're billed monthly and you can cancel at any time. The minimum contract length is just one month. You get unrestricted Internet access from your applications, unlimited in-browser Python, Bash and database consoles, and full SSH access to your account. All accounts (including free ones) have screen-sharing with other PythonAnywhere accounts, and free SSL support (though you'll need to get a certificate for your own domains).

Welcome to www.pythonanywhere.com!

It looks like you're in Europe. You're quite welcome to sign up here on our US-hosted system, but if you want complete peace of mind about the location of any data you store on our servers — especially if you're planning to store personal data and need [GDPR](#) compliance — you can try our [EU-hosted site](#) instead. The servers are all in Germany, and so is all of your data. Also, for paid accounts, prices are in euros :-)

Create your account

Username:

Email:

Password:

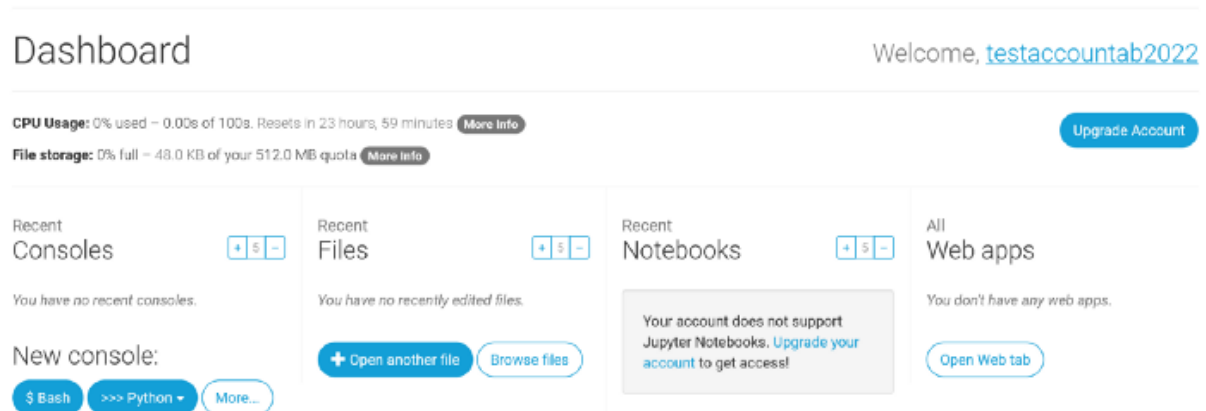
Password (again):

☒ I agree to the [Terms and Conditions](#) and the [Privacy and Cookies Policy](#), and confirm that I am at least 13 years old.

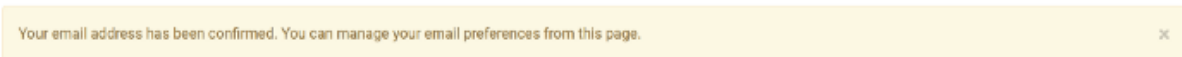
Register

We promise not to spam or pass your details on to anyone else.

You will be redirected and you can start the tour.



Also, do not forget to confirm your email address.



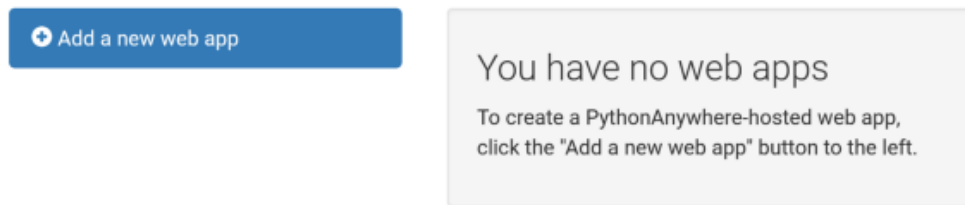
Screenshot from pythonanywhere.com by author

c. Initialize a web app

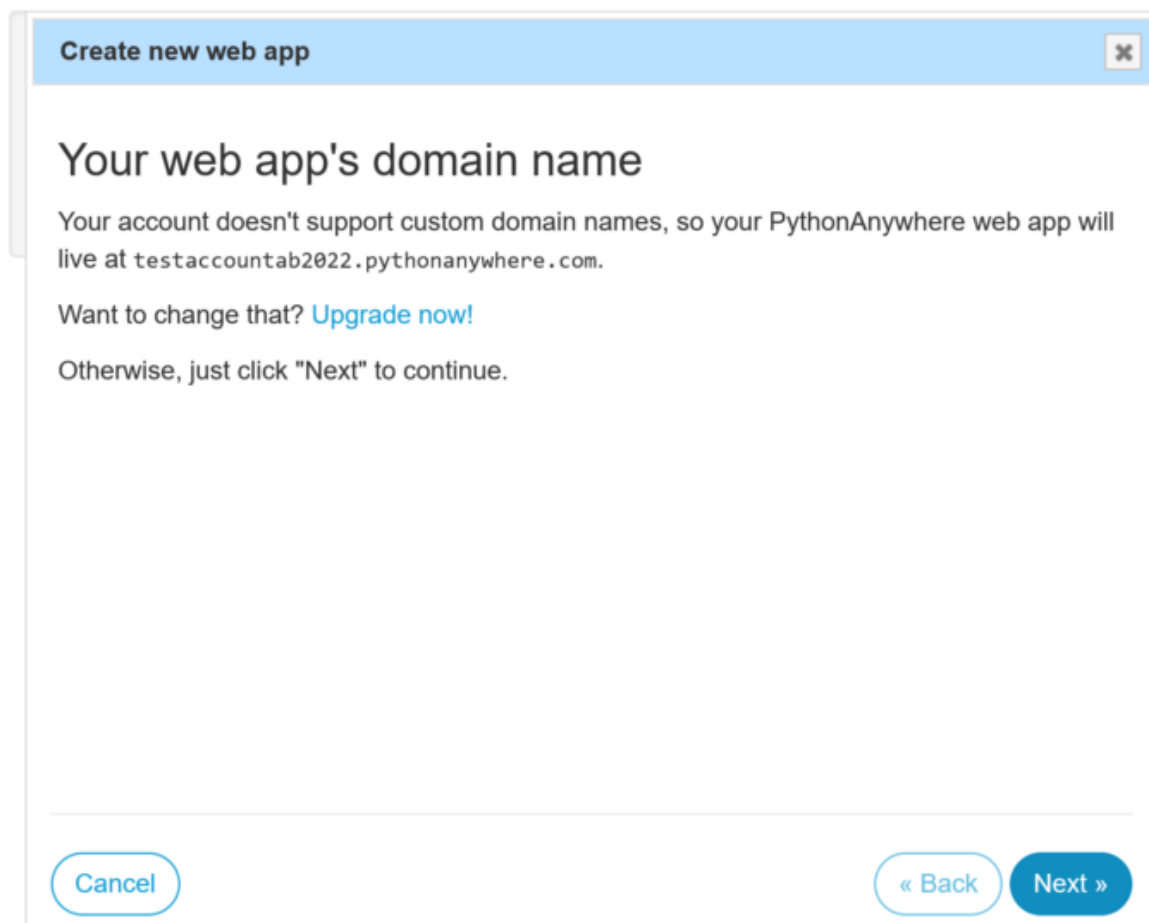
We can start now to add a new web app. It is pretty easy. Go to the Web tab.

[Dashboard](#) [Consoles](#) [Files](#) **Web** [Tasks](#) [Databases](#)

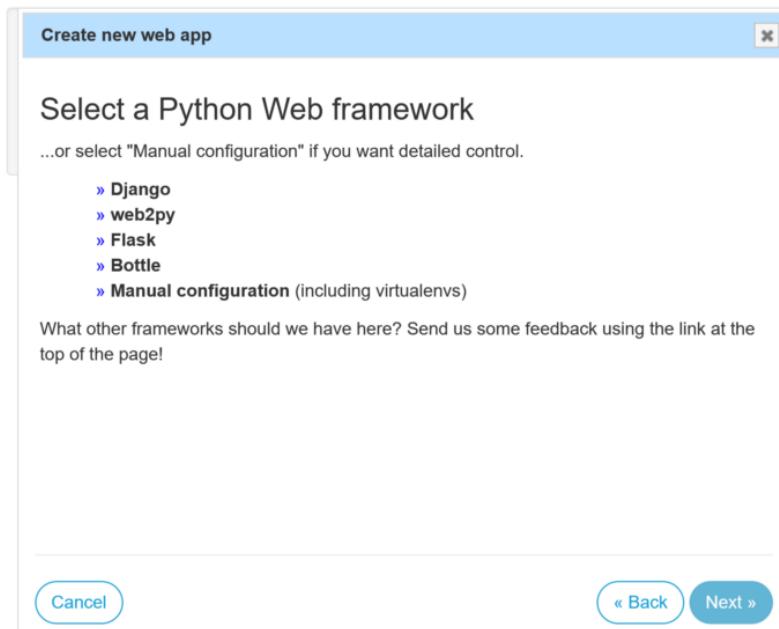
And add a new web app.



It is opening a window, if you have a paid account, you could enter a customized URL. But, it is not relevant for now, you can modify it later.



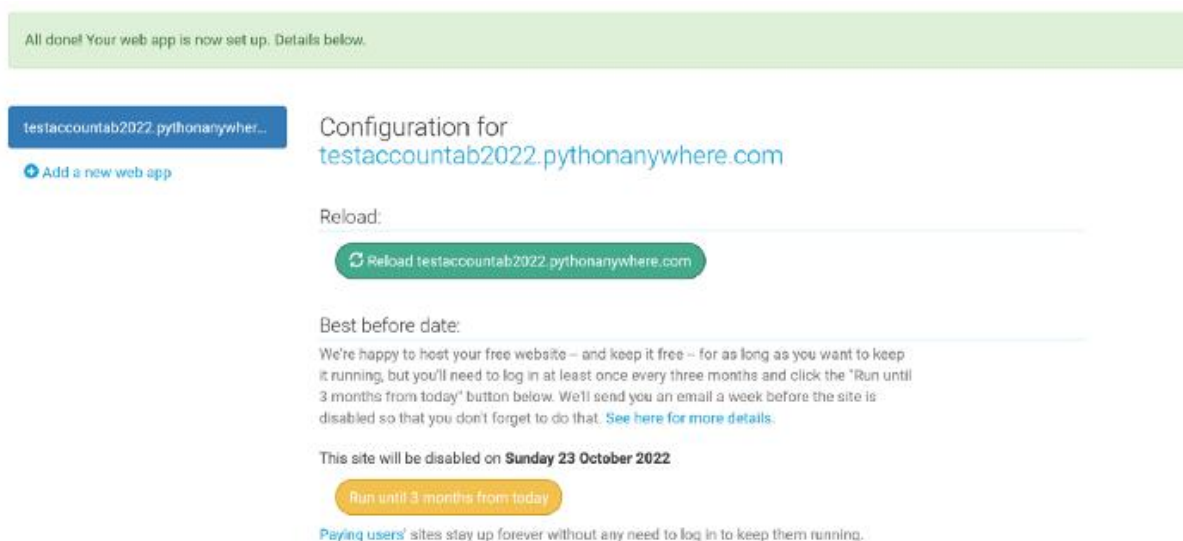
Select a python framework, just choose “*manual configuration*” and select a python version. I chose 3.9.



Click “Next”



Et voilà done!



d. Available Plans

Later, if you feel comfortable, you can do the “*Hacker*” Plan for 5 \$ USD. Is it enough for running a basic website like a portfolio? I had it for almost one year. Then I changed to the “*Web dev*” plan because I implemented another webpage. You can also customize any plan by parameters like CPU, the number of apps, or disk space. Compared to other services like DigitalOcean it is way cheaper because you have so much extra included.

Hacker	\$5/month	Web dev	\$12/month	Startup	\$99/month	Custom	\$5 to \$500/month
Run your Python code in the cloud from one web app and the console		If you want to host small Python-based websites for you or for your clients		Start a business and don't worry about having to scale to handle traffic spikes		Want a combination that's not on the list? Create your own! All custom plans have:	
A Python IDE in your browser with unlimited Python/bash consoles		A Python IDE in your browser with unlimited Python/bash consoles		A Python IDE in your browser with unlimited Python/bash consoles		A Python IDE in your browser with unlimited Python/bash consoles	
One web app on a custom domain or your-username.pythonanywhere.com		Up to 2 web apps on custom domains or your-username.pythonanywhere.com		Up to 3 web apps on custom domains or your-username.pythonanywhere.com		Up to 20 web apps, on custom domains or your-username.pythonanywhere.com	
Enough power to run a typical 100,000 hit/day website. (more info)		Enough power to run a typical 150,000 hit/day website on each web app. (more info)		Enough power to run a typical 1,000,000 hit/day website on each web app. (more info)		As many web workers as you need to scale your site's capacity. (more info)	
2,000 CPU-seconds per day for consoles, scheduled tasks and always-on tasks. (more info)		4,000 CPU-seconds per day for consoles, scheduled tasks and always-on tasks. (more info)		10,000 CPU-seconds per day for consoles, scheduled tasks and always-on tasks. (more info)		Up to 100,000 CPU-seconds per day for consoles, scheduled tasks and always-on tasks. (more info)	
IPython/Jupyter notebook support		IPython/Jupyter notebook support		IPython/Jupyter notebook support		IPython/Jupyter notebook support	
1GB disk space		5GB disk space		50GB disk space		As much disk space as you choose	
Create a Hacker account		Create a Web Developer account		Create a Startup account		Create a Custom account	

	Custom: \$5 to \$500/month	Startup: \$99/month	Web Developer: \$12/month	Hacker: \$5/month	Beginner: \$0/month
Monthly billing, cancel at any time Or get in touch to hear about discounts if you pay for a year up front, or want to pay annually in bitcoins	✓ 30-day no-quibble money-back guarantee	✓ 30-day no-quibble money-back guarantee	✓ 30-day no-quibble money-back guarantee	✓ 30-day no-quibble money-back guarantee	
Python web applications	You choose	3	2	1	1
At your-username.pythonanywhere.com	✓	✓	✓	✓	✓
At your own domain	✓	✓	✓	✓	✗
Web workers per web app	You choose	25	3	2	1
Python, Bash, MySQL consoles	Unlimited	Unlimited	Unlimited	Unlimited	Up to 2
SSH access to your account Via <code>ssh.pythonanywhere.com</code>	✓	✓	✓	✓	✗
Access to external internet sites from your code e.g. <code>urllib</code> or <code>wget</code>	✓	✓	✓	✓	Specific sites via HTTP(S) only
Gold star next to your name in the forums :-)	✓	✓	✓	✓	✗
Share your consoles with other people	✓	✓	✓	✓	✓
Scheduled tasks Like cron, but simpler	20, hourly or daily	20, hourly or daily	20, hourly or daily	20, hourly or daily	1 daily task
Always-on tasks Tasks that are always running	You choose	1	1	1	✗
CPU allowance more info	You choose	10,000 seconds	4,000 seconds	2,000 seconds	100 seconds
Bandwidth more info	Depends on price	High	Medium	Low	Low
Private file storage	You choose	50GB	5GB	1GB	512MB

e. Setup Backend

Go to the Console tab and open a new bash console.



```
13:13 ~ $ █
```

Clone [my repo](#) from GitHub.



```
13:13 ~ $ ls
README.txt
13:16 ~ $ git clone https://github.com/AntonioBlago/webpage_Tutorial_3.git webpage
Cloning into 'webpage'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 26 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (26/26), 12.54 KiB | 37.00 KiB/s, done.
13:16 ~ $ █
```

Create a virtual environment for your web app. Here is a good help on how to do it: <https://help.pythonanywhere.com/pages/Virtualenvs/>.

```
mkvirtualenv webpage_env --python=/usr/bin/python3.9
```



```
13:17 ~/webpage (main)$ mkvirtualenv webpage_env --python=/usr/bin/python3.9
created virtual environment CPython3.9.5.final.0-64 in 10887ms
creator CPython3Posix(dest=/home/testaccountab2022/.virtualenvs/webpage_env, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/testaccountab2022/.local/share/virtualenv)
added seed packages: pip==21.1.2, setuptools==57.0.0, wheel==0.36.2
activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator
virtualenvwrapper.user_scripts creating /home/testaccountab2022/.virtualenvs/webpage_env/bin/predeactivate
virtualenvwrapper.user_scripts creating /home/testaccountab2022/.virtualenvs/webpage_env/bin/postdeactivate
virtualenvwrapper.user_scripts creating /home/testaccountab2022/.virtualenvs/webpage_env/bin/preactivate
virtualenvwrapper.user_scripts creating /home/testaccountab2022/.virtualenvs/webpage_env/bin/postactivate
virtualenvwrapper.user_scripts creating /home/testaccountab2022/.virtualenvs/webpage_env/bin/get_env_details
webpage_env) 13:19 ~/webpage (main)$
```

Now, we have to install the required libraries:

```
pip install -r requirements.txt
```


f. Web Configuration

Next, head to the tab Web on pythonanywhere and set up the last configurations:

Under **Code**, set up your “*source code*” directory, where your app folder is:

Code:

What your site is running.

Source code:	/home/testaccountab2022/webpage	➔Go to directory
Working directory:	/home/testaccountab2022/	➔Go to directory
WSGI configuration file:	/var/www /testaccountab2022_pythonanywhere_com_wsgi.py	
Python version:	3.9 	

You get the information when initializing the virtual environment in your bash console:


```
13:17 ~/webpage (main)$ mkvirtualenv webpage_env --python=/usr/bin/python3.9
created virtual environment CPython3.9.5.Final.0-64 in 10887ms
creator CPython3Posix(dest=/home/testaccountab2022/.virtualenvs/webpage_env, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/testaccountab2022/.local/share/virtualenv)
```

Screenshot from pythonanywhere.com by author

Then, configure the wsgi.py as follows:

Code:

What your site is running.

Source code:	/home/testaccountab2022/webpage	Go to directory
Working directory:	/home/testaccountab2022/	Go to directory
WSGI configuration file:	/var/www /testaccountab2022_pythonanywhere_com_wsgi.py	
Python version:	3.9 	

Below

+++++ FLASK +++++

uncomment the lines for import sys and change your working directory as above. Also, edit your import statement. In this case, our run.py file is the main file.

```
from run import app as application
```

```
91
92
93 # +++++ FLASK +++++
94 # Flask works like any other WSGI-compatible framework, we just need
95 # to import the application. Often Flask apps are called "app" so we
96 # may need to rename it during the import:
97 #
98 #
99 import sys
100 #
101 ## The "/home/testaccountab2022" below specifies your home
102 ## directory -- the rest should be the directory you uploaded your Flask
103 ## code to underneath the home directory. So if you just ran
104 ## "git clone git@github.com:myusername/myproject.git"
105 ## ...or uploaded files to the directory "myproject", then you should
106 ## specify "/home/testaccountab2022/myproject"
107 path = '/home/testaccountab2022/webpage'
108 if path not in sys.path:
109     sys.path.append(path)
110
111 from run import app as application # noqa
112 #
```

Comment any other lines and save your changes.

Also, put the directory for your virtual environment under **Virtualenv**.

Virtualenv:

Use a virtualenv to get different versions of flask, django etc from our default system ones. [More info here](#). You need to **Reload your web app** to activate it; NB - will do nothing if the virtualenv does not exist.

/home/testaccountab2022/.virtualenvs/webpage_env

[Start a console in this virtualenv](#)

Then reload and go to the URL:

<http://testaccountab2022.pythonanywhere.com/>

Configuration for
testaccountab2022.pythonanywhere.com

Reload:

[Reload testaccountab2022.pythonanywhere.com](#)

Don't forget to set:

Enable HTTPS also:

You need to **Reload your web app** to activate any changes made below.

Forcing HTTPS means that anyone who goes to your site using the insecure http URL will immediately be redirected to the secure https one. [More information here](#).

Force HTTPS: ☒ Enabled

Password protection is ideal for sites that are under development when you don't want anyone to see them yet.

Password protection: ☐ Disabled

Username: [Enter a username](#)

Password: [Enter a password](#)

It should display:



Congratulations you are done!