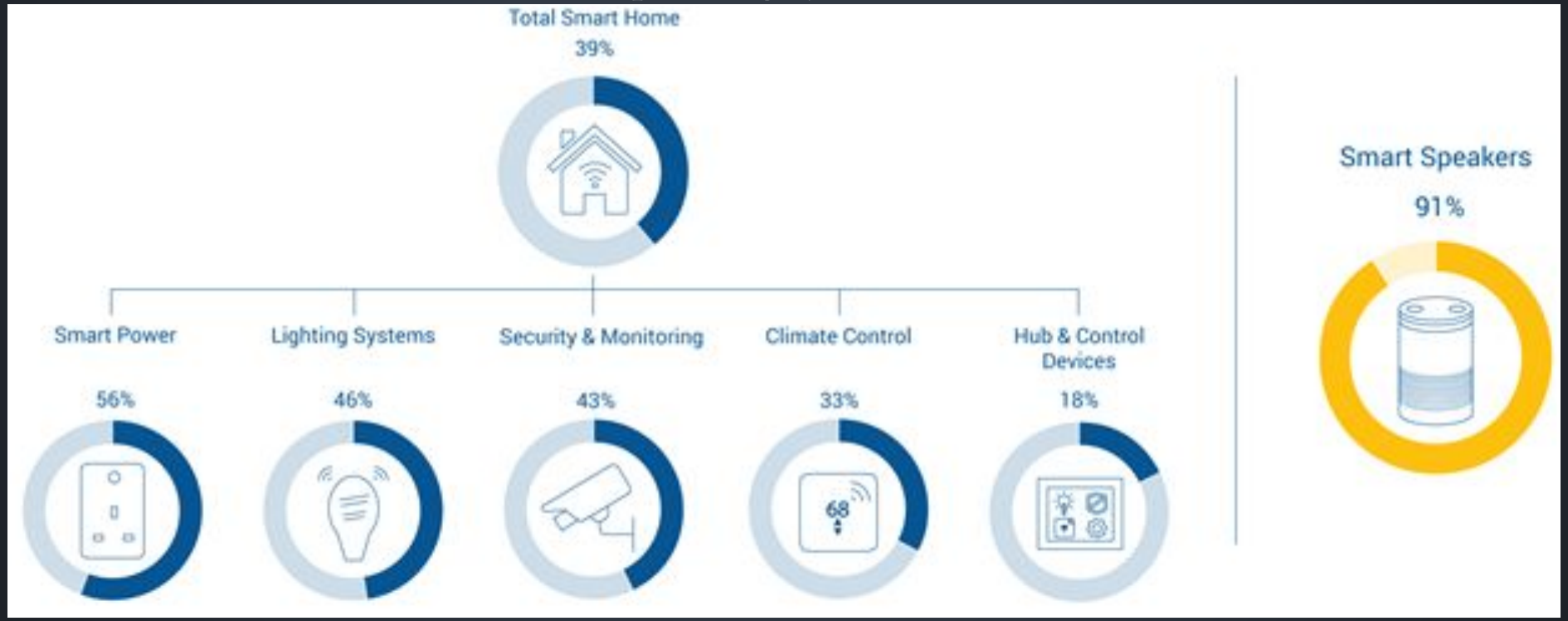
Smart Home Dashboard

•••

Dr. Dmitry Nikolayev,
eSportLab / CDISE / Skoltech
<<Introduction to IoT>>
October 2019

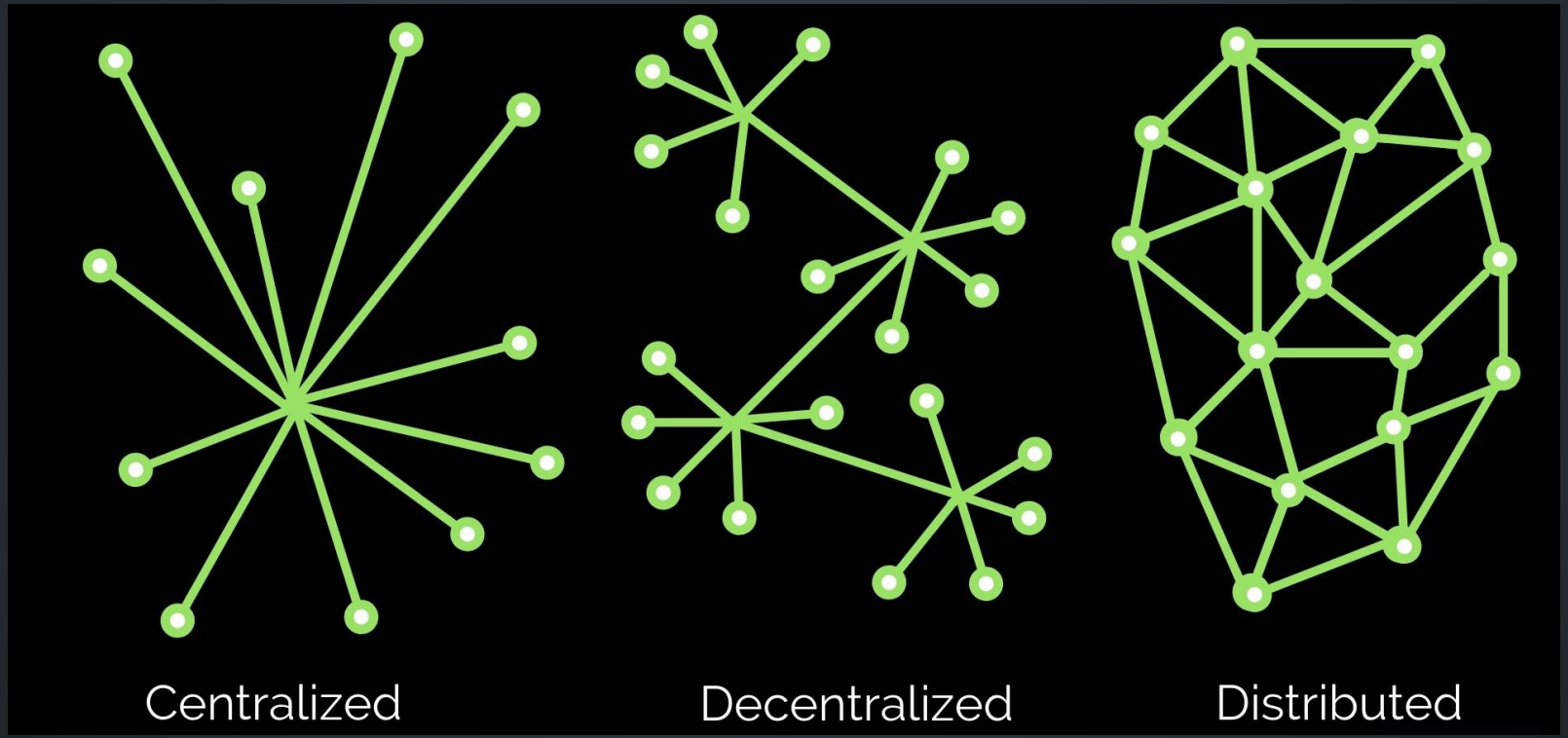
Smart Home Technology Growth (in 2018)

An ***operating system*** is system software that manages computer hardware, software resources, and provides common services for computer programs. What's the ***difference*** between "operating system" and "smart home software"?



Network Software Architectures

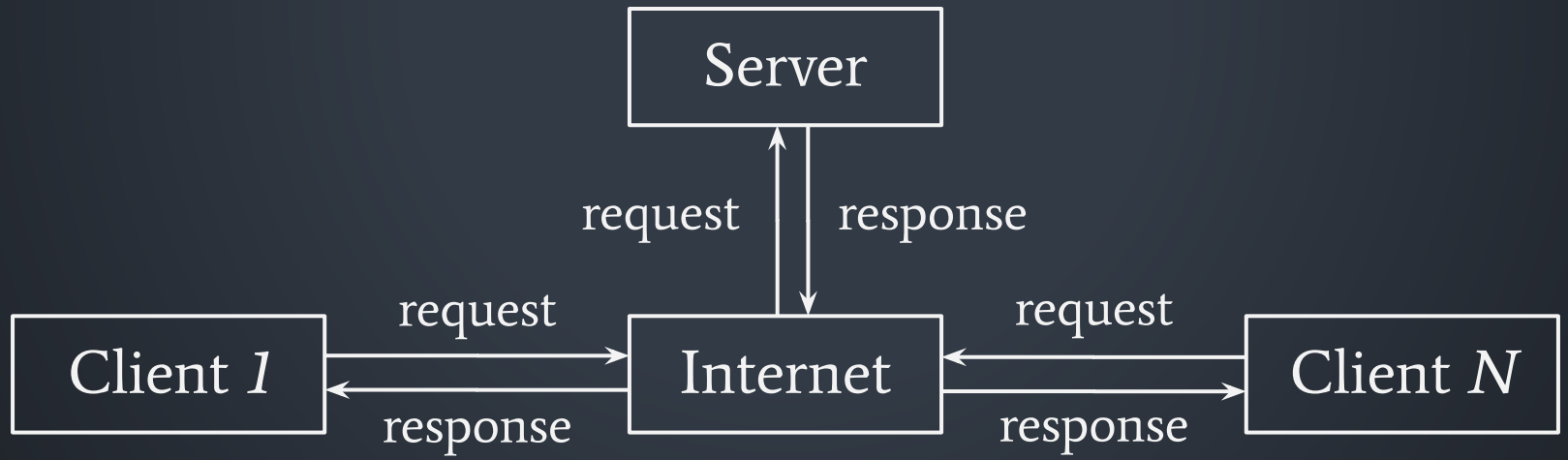
What network software architectures are applied for smart homes and how?



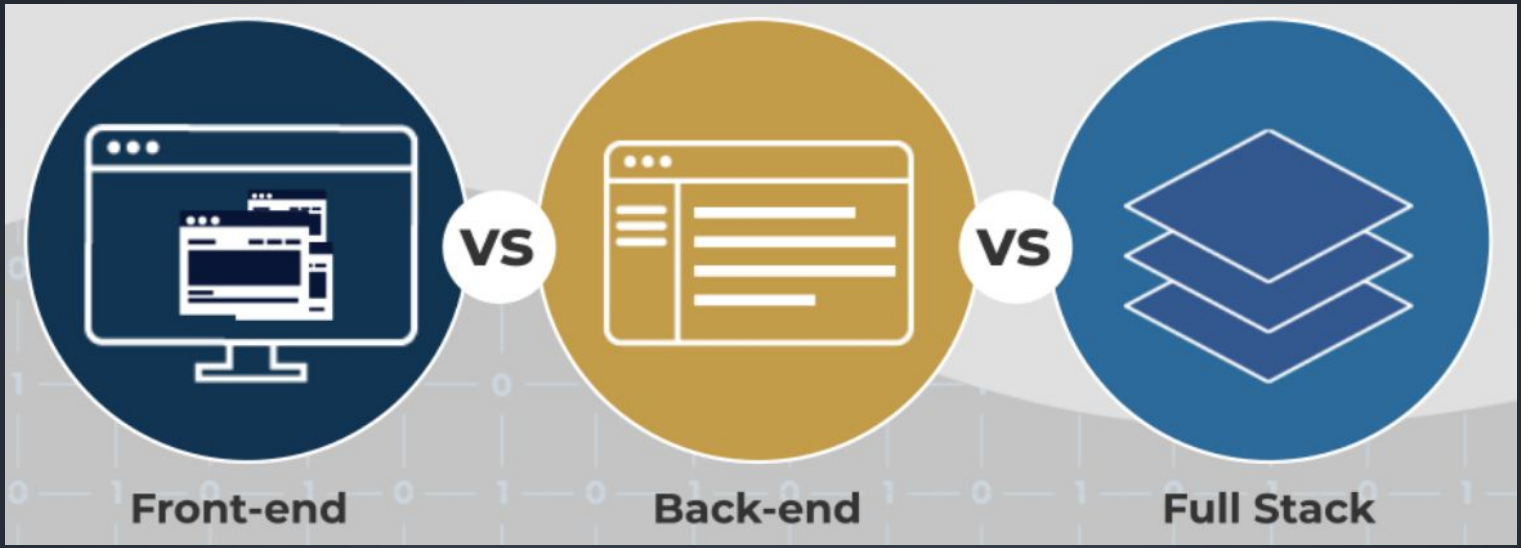
Centralized Architecture

is a distributed application architecture that partitions tasks between the providers of a resource or service (servers) and service requesters (clients).

Could the clients send requests to each other?
Could the clients share resources with each other?



Dashboard Development Steps



Version 1.0 Single-Page WebApp with RESTfull API
Version 2.0 Multi-Page WebApp with Authorization
Version 3.0 Multi-Page WebApp with Interactive Chart

HyperText Transfer Protocol

1. The browser requests an HTML page. The server returns an HTML file.
2. The browser requests a style sheet. The server returns a CSS file.
3. The browser requests a JPG image. The server returns a JPG file.
4. The browser requests JavaScript code. The server returns a JS file
5. The browser requests data. The server returns data (in XML or JSON).

Resource /device	GET Read device details	POST Create device	PUT Update device details
	GET Read device list	DELETE Delete device	

What is ***the main disadvantage*** of this protocol?

Websockets Transfer Protocol

is a computer communications protocol, providing full-duplex communication channels over a single TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011.

The WebSocket protocol enables interaction between a web browser (or other client application) and a web server with lower overhead than half-duplex alternatives such as HTTP polling, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the client without being first requested by the client, and allowing messages to be passed back and forth while keeping the connection open.

What is ***the main advantage*** of this protocol?

REpresentational State Transfer (RESTfull) API

-- a software architectural style that defines a set of constraints to be used for creating Web services using HTTP requests to GET, PUT, POST and DELETE data. RESTful Web services allow to use a uniform and predefined set of stateless operations sharing the design properties: *performance, scalability, simplicity, modifiability, visibility, portability, reliability.*

POST	/user	Create user
GET	/user/login	Logs user into the system
GET	/user/logout	Logs out current logged in user session
GET	/user/{username}	Get user by user name
PUT	/user/{username}	Updated user
DELETE	/user/{username}	Delete user

Scientific Data Visualization Software

1980

Symbolic Computing

Matlab, Maple, Matematica, Latex

1990

Proprietary Software

Microsoft Office, Oracle and their tooling

2000

Open Source Software

Matplotlib, Numpy, Scipy, Scikit-Learn, IPython

2010

Web-based Software (javascript)

D3js, Chart.js, etc.and their ***bindings*** to Python (python-bokeh)

2015

Web-based Software (notebooks)

Jupiter notebooks for interactive calculations

2018

Web-based Software (python)

Plotly Dash for Interactive web-dashboards written on a pure python.

Check Out My Git Repositories

https://github.com/esportslab

manuals Star

Updated 3 minutes ago

smarthome3 Star

HTML Updated 31 minutes ago

smarthome2 Star

HTML Updated 32 minutes ago

smarthome1 Star

HTML Updated 33 minutes ago

Register a Domain on Pythonanywhere.com

pythonanywhere

Plans and pricing

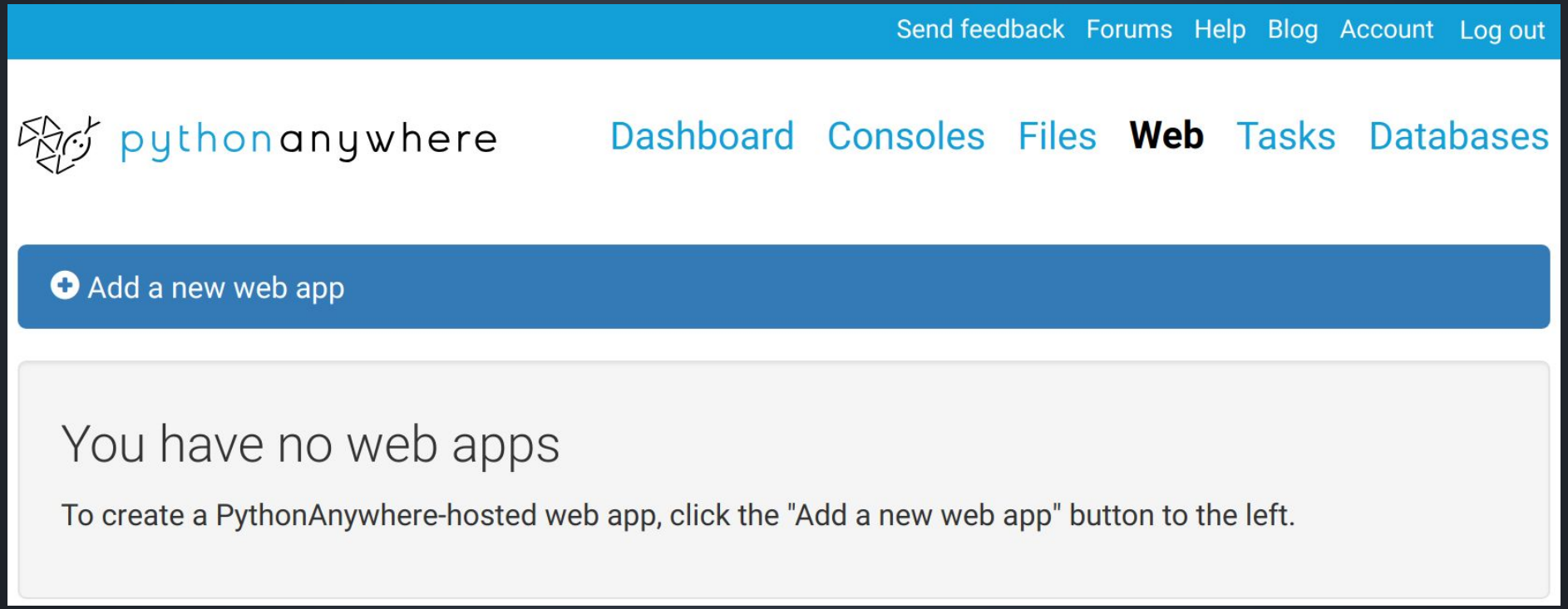
Beginner: Free!

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

Add New Web Application



Choose Flask-powered BackEnd on Python 3.7

Create new web app

Select a Python Web framework

...or select "Manual configuration" if you want detailed control.

- » **Django**
- » **web2py**
- » **Flask** ← Choose this one
- » **Bottle**
- » **Manual configuration** (including virtualenvs)

What other frameworks should we have here? Send us some feedback using the link at the top of the page!

Specify the Folder Name and the Entry Point

Create new web app

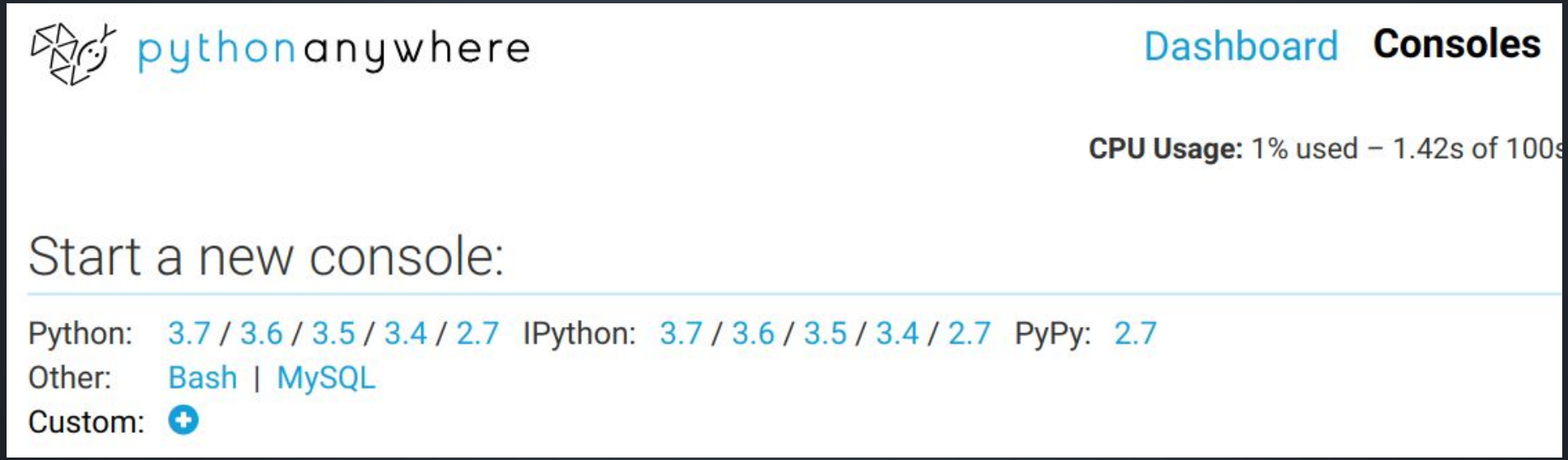
Quickstart new Flask project

Enter a path for a Python file you wish to use to hold your Flask app. **If this file already exists, its contents will be overwritten with the new app.**

Path

/home/skoltech/smarthouse/main.py

Run Bash Console on Hosting



Ensure that the Folder Exists

```
08:37 ~ $ pwd | cowsay
 _________________
| /home/skoltech |
 -----------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
08:37 ~ $ ls
README.txt  smarthouse
08:38 ~ $ cd smarthouse/
08:38 ~/smarthouse $ cat main.py

# A very simple Flask Hello World app for you to get started with...

from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello from Flask!'
```

Ensure that the Site Works

Bash console 13536320

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

Not secure skoltech.pythonanywhere.com

Hello from Flask!

Decorator (Wrapper) Design Pattern

is a design pattern that allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class. In Python it is implemented as a high order function -- a function that receives function and returns function wrapping the previous one.

```
def makebold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

def makeitalic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

@makebold
@makeitalic
def hello():
    return "hello world"

print hello()  ## returns "<b><i>hello world</i></b>"
```

Change the Message and Reload the Website

Bash console 13536320

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def hello_world():
7     return 'Hello from Skoltech!'
```

Dashboard Consoles Files **Web**

Configuration for
skoltech.pythonanywhere.com

Reload:

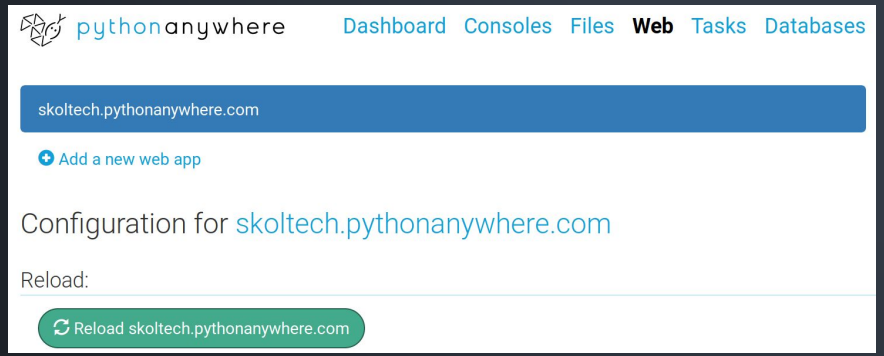
Reload skoltech.pythonanywhere.com

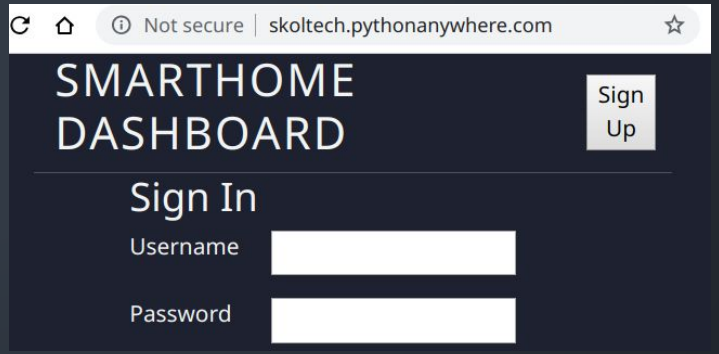
Not secure skoltech.pythonanywhere.com

Hello from Skoltech!

Download and Deploy SmartHome 1.0 from GitHub

```
14:02 ~ $ rm -r smarthome; ls
README.txt
14:02 ~ $ git clone https://github.com/esportslab/smarthome1 smarthome
Cloning into 'smarthome'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 19 (delta 1), reused 19 (delta 1), pack-reused 0
Unpacking objects: 100% (19/19), done.
Checking connectivity... done.
```





Final Code

```
.
├── assets
│   ├── bootstrap.css
│   └── custom.css
├── main.py
└── templates
    ├── error.html
    └── signin.html

2 directories, 5 files
```

```python
@app.errorhandler(404)
def page_not_found(e):
    return render('error.html')


@app.route('/assets/<path:path>', methods=['GET'])
def send_assets(path):
    return send_from_directory('assets', path)


@app.route('/data/<path:path>', methods=['GET'])
def send_data(path):
    return send_from_directory('data', path)


@app.route('/', methods=['GET'])
@app.route('/signin', methods=['GET'])
def signin():
    return render('signin.html')


if __name__ == '__main__':
    app.run(debug=True, port=8050)
```

Version 2.0. Dynamic WebApp with Authorization

```
smarthome2
├── assets
│   ├── bootstrap.css
│   ├── custom.css
│   └── fonts.css
├── data
├── main.py
└── templates
    ├── error.html
    ├── signin.html
    └── signup.html

3 directories, 7 files
```

```
{
    "_default": {},
    "users": {
        "1": {
            "email": "user@mail.ru",
            "name": "user",
            "password": "u"
        },
        "2": {
            "email": "admin@mail.ru",
            "name": "admin",
            "password": "a"
        }
    }
}
```

```
from tinydb import TinyDB, Query
import os
import pathlib

app = Flask(__name__)

session = {'username': ''}

app_path = str(pathlib.Path(__file__).parent.resolve())
db_path = os.path.join(app_path, os.path.join("data", "db.json"))

db = TinyDB(db_path, sort_keys=True, indent=4, separators=(',', ': '))
usr = db.table('users')

```

User Authorization

```python
@app.route('/signin', methods=['POST'])
def do_signin():
    User = Query()
    users = usr.search(User.name == request.form['username'])
    if not users:
        return render('signin.html', text='Wrong username or password')
    user = users[0]
    if user['password'] != request.form['password']:
        print(user['password'], request.form['password'])
    session['username'] = user['name']
    return redirect('dashboard')


@app.route('/signup', methods=['POST'])
def do_signup():
    User = Query()
    users = db.search(User.name == request.form['username'])
    if len(users) > 0:
        text = 'Such user have already exists'
        return render('signup.html', text=text)
    usr.insert({
        'name' : request.form['username'],
        'email': request.form['email'],
        'password': request.form['password']
    })
    return redirect('/')
```

Version 3.0. Dynamic WebApp with Interactive Chart

```
├── assets
│   ├── bootstrap.css
│   ├── custom.css
│   └── fonts.css
├── dashboard.py
├── data
│   ├── db.json
│   └── smarthome.csv
├── main.py
└── templates
    ├── error.html
    ├── signin.html
    └── signup.html

3 directories, 10 files
```

```python
app_path = str(pathlib.Path(__file__).parent.resolve())
df = pd.read_csv(os.path.join(app_path, os.path.join("data", "smarthome.csv")))

app = dash.Dash(__name__, url_base_pathname='/dashboard/')
server = app.server

theme = {
    'background': '#111111',
    'text': '#7FDBFF'
}

def build_banner():…
def build_graph():…

app.layout = html.Div(
    className='big-app-container',
    children=[
        build_banner(),
        html.Div(
            className='app-container',
            children=[
                build_graph(),
            ]
        )
    ]
)
```

Final Task

1. Download command line tool based on unix environment (with bash, git and etc.)

   https://cmder.net/

2. Develop the improved version of dashboard, add one resource for some kind of sensors with the RESTfull API, add a chart for its visualization. See examples

   https://dash-gallery.plotly.host/dash-manufacture-spc-dashboard/

3. Create your own git repository and load the project on it

   https://github.com/, https://bitbucket.org

4. Deploy your project on the hosting

   https://www.pythonanywhere.com/, https://www.heroku.com/

5. Send me links to your dashboard and git-repository @dmitrynvm (telegram)