CFG Advanced Python Course - Session 3
Flask & GET/POST requests

Recap

- What is pip?
- How do you write comments in Python?
- What is the difference between **10 / 3** and **10 / 3.0** in Python?
- What is a function?
- Can functions take variables?
- Can functions have return values?
- What does a function without a return value actually return?
- What is a loop? What is a list?

Flask: An Introduction
Flask is a framework for building lightweight web applications. In order to get up and running, we will follow the instructions over at http://flask.pocoo.org/

Windows users, you should already have pip installed from last week. Mac people you can install pip by going to your terminal and typing the following:

sudo easy_install pip

We will install Flask using pip:

sudo pip install Flask

Then we will create a baseline Flask application - call it hello.py (py is the extension given to all Python files) - which will look like this:

```
1  from flask import Flask
2
3  app = Flask("MyApp")
4
5
6  @app.route("/")
7  def hello():
8      return "Hello World"
9
10 app.run()
11
```

Imports
You might have noticed that at the very beginning of the file we are doing **from flask import Flask**; this is Python's way of letting you import different modules and libraries to use.

In this case, we already have Flask installed, so we can safely import it. If you tried to import a module or library that was not installed using pip, then you would get an error.

Flask routes & decorators
You may be wondering what **@app.route** stands for and why we are using it. This is what is known as a **decorator**.

The concept of decorators exists in most programming languages; their purpose is to allow you to inject or modify code in functions. In simpler terms, imagine you would like to do something at the entry and/or exit points of a function. This is what is happening here - @app.route, which is part of Flask, is doing some things internally for us so that we can

accept a request coming from the user's browser into Python. The details of what it actually does is beyond the scope of this course, so don't worry about it for the time being.

A decorator always starts with the @ sign and goes on the line right before a function is defined.

Task:

- Run the simple hello.py application by doing **python hello.py** and hitting enter
- Navigate to localhost:5000 in your browser and confirm that you see the message expected
- Make a change to hello.py - return something other than "Hello World!"; refresh the page in your browser and see what happens
- Try visiting a URL like localhost:5000/i/dont/exist - what happens?

Flask: Taking parameters from the URL

Take a look at the following code:

```
1  from flask import Flask
2
3  app = Flask("MyApp")
4
5
6  @app.route("/")
7  def hello():
8      return "Hello World"
9
10
11 @app.route("/<name>")
12 def hello_someone(name):
13     return "Hello {0}!".format(name.title())
14
15 app.run()
```

The <name> is a URL matcher - It will match / followed by any word, for example /andreas, /rachel etc. Flask will make this value available as a parameter to hello_someone. We then use the name variable to say hello to a particular user.

Task

- Add hello_someone to your hello.py file and check that it works by visiting localhost:5000/andreas or your own name if you prefer
- Try and make it so that visiting localhost:5000/bye/andreas returns 'Goodbye Andreas' or Goodbye for any other name given

Serving your HTML using Flask

Flask lets you serve your HTML - this is possible using the render_template function. All you have to do is provide the name of the template and the variables you want to pass to the template engine as keyword arguments. Here's a simple example of how to render a template:

```
1  from flask import Flask
2  from flask import render_template
3
4  app = Flask("MyApp")
5
6
7  @app.route("/")
8  def hello():
9      return render_template("hello.html")
10
11
12 @app.route("/<name>")
13 def hello_someone(name):
14     return render_template("hello.html", name=name.title())
15
16 app.run()
```

Note: Flask will look for templates in the templates folder.

Flask uses another library called Jinja2, which is a templating language for Python. It's extremely powerful, but it's beyond the scope of this course. If you are interested however, go ahead and read up on it and make use of it.

Here is what an example html file called hello.html would look like, inside a templates folder, that uses Jinja2:

```
1  <!doctype html>
2  <title>Hello from Flask</title>
3  {% if name %}
4    <h1>Hello {{ name }}!</h1>
5  {% else %}
6    <h1>Hello World!</h1>
7  {% endif %}
```

Task

- Extend the above code so that you can say goodbye to someone if you visit localhost:5000/goodbye/andreas (or feel free to use any other name)
- Try to add some more parameters to the URL and pass those onto your hello.html template; for example localhost:5000/goodbye/andreas/day would say something like “Hello Andreas! Have a good day”, while localhost:5000/goodbye/andreas/night would say “Hello Andreas! Have a good night”.

GET & POST requests

In the world of HTTP, there are a number of types of requests; we will be concentrating on two of them:

- **GET** - when you want to retrieve data
- **POST** - when you want to send data

Everything we have been doing so far has been using a GET request - let's see how we can use a simple POST request. Let's add a form to hello.html:

```
<div id="contact-form">
    <h1>Get In Touch!</h1>
    <form method="post" action="/signup">
        <label for="name">Name: </label>
        <input type="text" id="name" name="name" value="" placeholder="Andreas Savvides" required="
        required" autofocus="autofocus" />
        <label for="email">Email Address: </label>
        <input type="email" id="email" name="email" value="" placeholder="andreas@editd.com" required="
        required" />
        <input type="submit" value="Submit" id="submit-button" />
    </form>
</div>
```

Pay attention to the form element which has an action equal to /signup - that's the URL which the form will be submitted to, relevant to the absolute URL you are at (in this case localhost:5000) - so we need to create an entry in our hello.py file:

```
@app.route("/signup", methods=['POST'])
def sign_up():
    form_data = request.form
    print form_data['name']
    print form_data['email']
    return "All OK"
```

There is one thing we need to do! You might have noticed that we are now accessing a variable called request; where did this come from? This is actually part of Flask and we need to import it - so at the very top of your hello.py file insert:

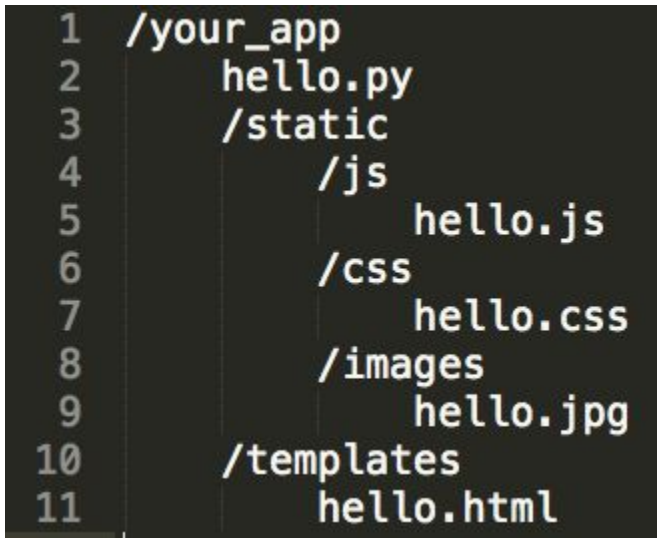
Now if you navigate to localhost:5000/hello/andreas then you should see something like this:

Task

- Try submitting the form without entering any information and see what happens!
- Now fill in the form and submit it
- Go to your terminal or command prompt and have a look to see if you can spot the name and e-mail address you entered and submitted
- Extend the form in hello.html to have another one or two input fields to capture some more information and ensure that this is printed out by hello.py

Flask project structure

This is the recommended structure that your Flask projects/applications should follow:



Whatever is stored in templates can be automatically detected when using render_template, so there isn't anything else you need to do. In order to reference your CSS, JavaScript and images from within your HTML then you need to do something like this:

```
<!doctype>
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/hello.css') }}">
<title>Hello from Flask</title>
{% if name %}
    <h1>Hello {{ name }}!</h1>
{% else %}
    <h1>Hello World!</h1>
{% endif %}
```

Homework

- Head to **Mailgun's website** and create an account; we will be using Mailgun to programmatically send e-mails
- Head to **twitter.com** and create yourself a **Twitter** account if you don't already have one; we will be using Twitter's API to fetch data - you will need to have an account in order to do that