



HTML, PYTHON E PROGETTAZIONE DI UN SITO WEB

UTILIZZO DI FLASK COME SUPPORTO DI GESTIONE ALLE ROUTE DI ACCESSO ALLE
PAGINE

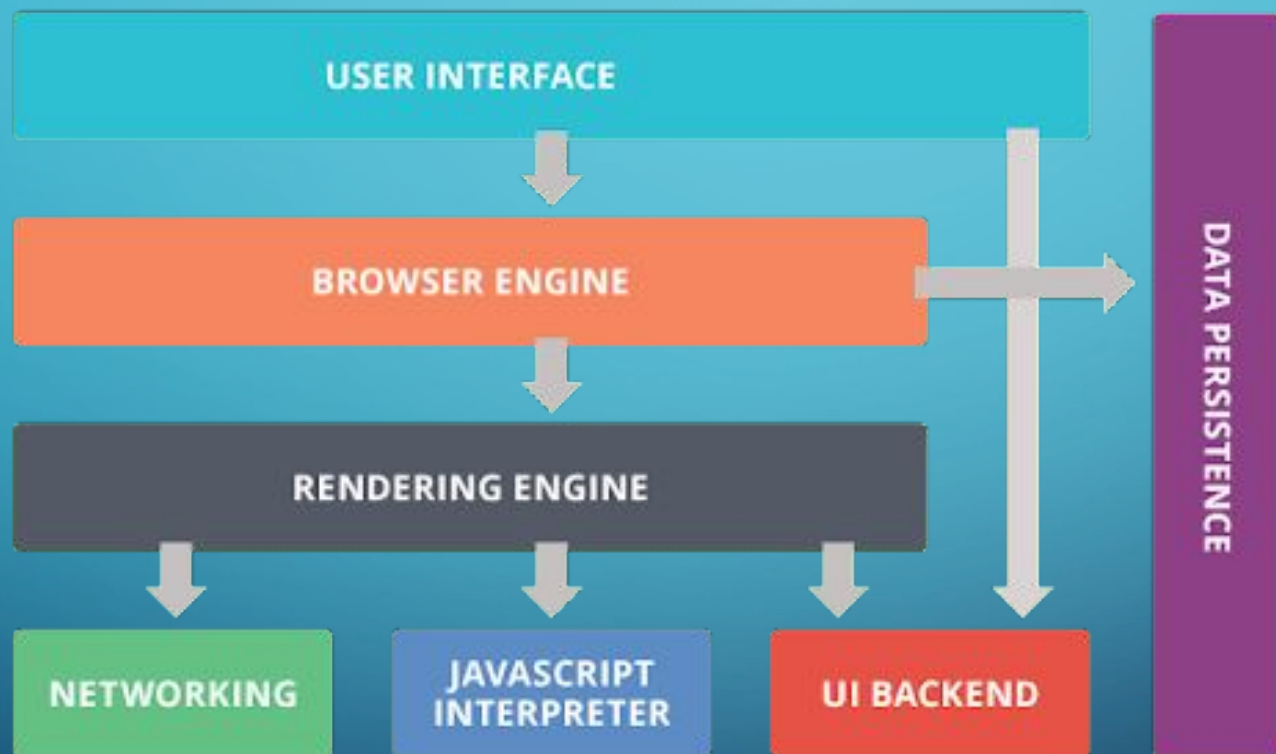
HTML (HYPER TEXT MARKUP LANGUAGE)

- Linguaggio standard per la creazione di pagine web
- Descrive la struttura delle pagine
- Serie di elementi strutturati
- Interpretazione necessita di un browser

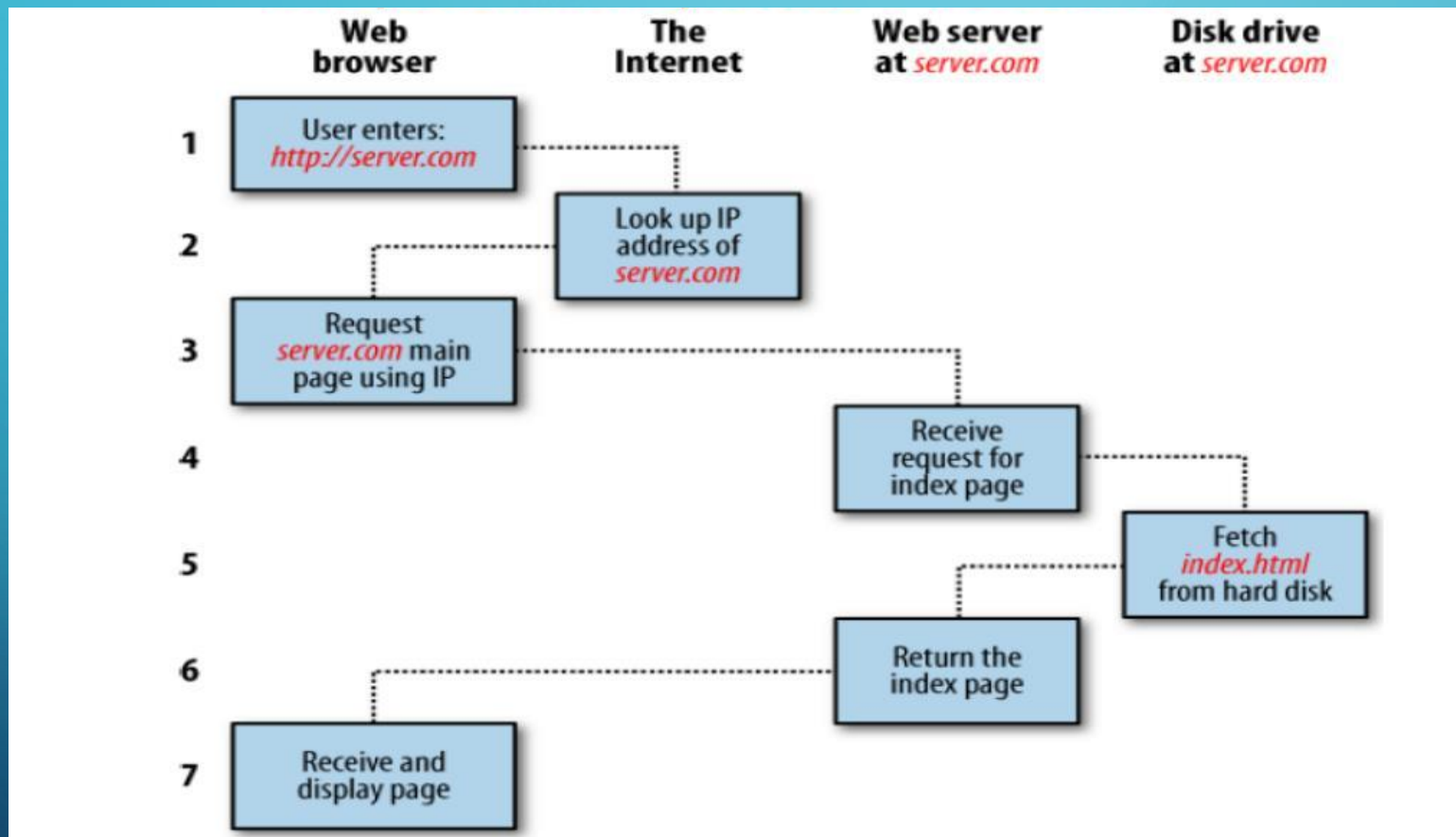
BROWSER

- Programma per visualizzazione di una pagina HTML
 - Esempi di browser possono essere:
 - Brave, Google Chrome, Edge, Mozilla Firefox
- Un browser è composto da diverse componenti:
 - Browser Engine: contiene gli strumenti per la navigazione (barra di ricerca, pulsanti)
 - Rendering Engine: visualizza il contenuto richiesto
 - Javascript interpreter: interpreta il codice javascript contenuto nelle pagine HTML
 - UI backend: mostra grafiche relative a chiamate di funzione di sistema (alert, upload and download file, ...)

BROWSER



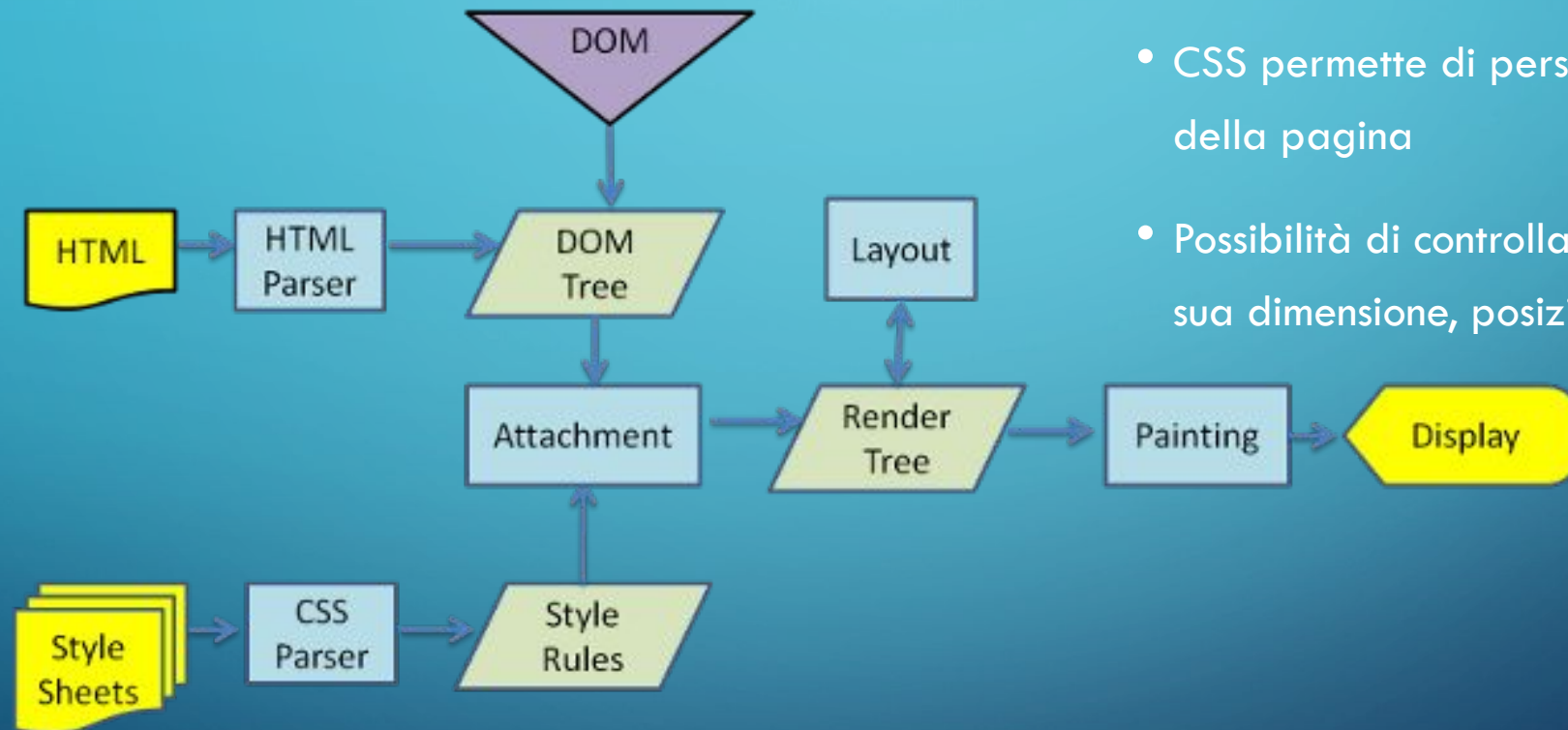
FLUSSO DI RICHIESTA



- Client (browser) invia una richiesta al server utilizzando il suo indirizzo IP
- L'indirizzo IP può essere associato ad un dominio, sarà compito del DNS fornire l'IP associato al dominio (<http://server.com>)
- Il server riceve la richiesta e risponde cercando la pagina richiesta all'interno della propria memoria e restituirà la pagina HTML in caso affermativo oppure con un errore in caso di assenza della pagina

<https://image1.slideserve.com/2934809/introduction-to-dynamic-web-content-request-response-procedure-1.jpg>

WEB PAGE RENDERING



- Personalizzazione tramite CSS (Cascading Style Sheets)
- CSS permette di personalizzare lo stile della pagina
- Possibilità di controllare colore, carattere e sua dimensione, posizione degli elementi

[The rendering process of a web page. | by Gabriel Neutzing | Medium](#)

HTML - ESEMPIO

```
<html>
  <head>
    <title>Esempio href</title>
  </head>
  <body>
    <h1>Esempio attributo href</h1>
    <p> <a href="https://www.google.com">Clicca qui per
aprire la pagina Google.com</p>
  </body>
</html>
```

TAG

HEAD contiene le informazioni della pagina. Titolo, descrizione, tag

ATTRIBUTO chiave=«valore»

BODY definisce la struttura della pagine (paragrafi, div, immagini, tabelle)

Empty tag (
, <hr>, non necessitano la chiusura del tag «<h1></h1>»

HTML - ESEMPIO

`<h1> Intestazione 1 </h1>`

`<h2> Intestazione 2 </h2>`

`<h3> Intestazione 3 </h3>`

Differiscono per grandezza nel rendering

HELLO WORD CON HTML

- Flask per la creazione e gestione del server
- Creazione di una pagina HTML «index.html»
- Creazione di uno stile CSS per gestire colori e posizione

FLASK

Framework web per Python, leggero e flessibile. È un *microframework* che fornisce le basi per creare applicazioni web, lasciando libertà di aggiungere funzionalità tramite estensioni. È ideale per progetti semplici e prototipi, ma può essere esteso per applicazioni più complesse.

Le sue caratteristiche principali sono:

- **Semplicità** e facilità di configurazione
- **Routing** per definire percorsi URL
- **Templating** con Jinja2 per creare pagine dinamiche
- **Estensioni** per funzionalità aggiuntive (esempio: forms, database)

FLASK – HELLO WORLD

```
app.py
from flask import Flask, render_template
```

render_template : funzione che permette il render di una pagina HTML (precedentemente creata)

```
app = Flask(__name__)
```

```
@app.route('/')
def hello():
    return render_template('index.html')
```

IMPORTANTE: Inserire le pagine HTML all'interno di una cartella templates

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

```
* Serving Flask app 'temp'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
```

Hello, World!

FLASK – HELLO WORLD

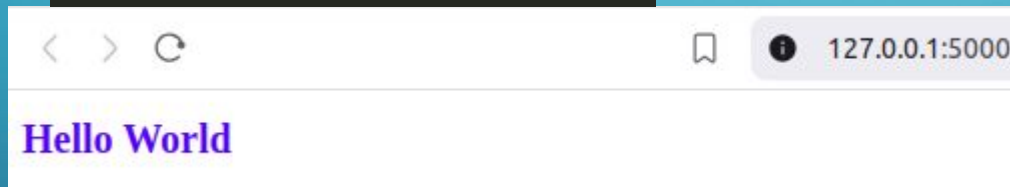
Creazione stile CSS

personalizzazione H1 con colore blu e dimensione 20 px

modifica HTML e app.py.

L'utilizzo di `{{“variabile”}}` in una pagina HTML consente l'utilizzo di variabile che sono state passate come parametro tramite la funzione `render_template`

```
# style.css > h1
h1 {
    color: blue;
    font-size: 20px;
}
```



```
css_path = "static/style.css"

@app.route('/')
def hello():
    return render_template('index.html', css = css_path)
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{css}}>
    <title>Page Name</title>
</head>
<body>
    <h1>Hello World</h1>
</body>
</html>
```

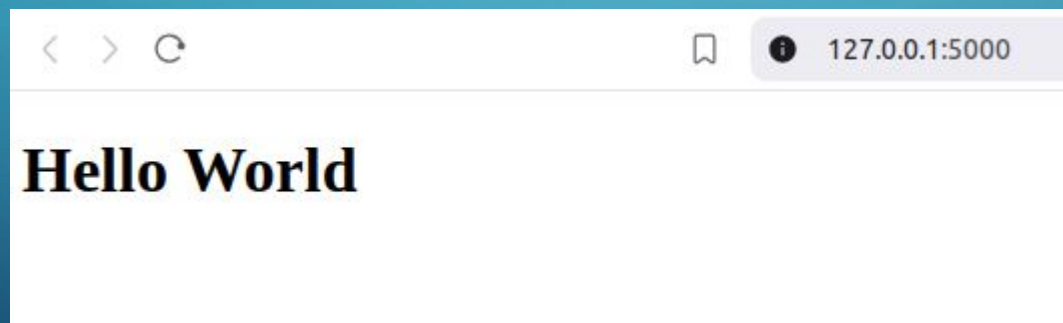
FLASK – HELLO WORLD

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Page Name</title>
</head>
<body>
  <h1>Hello World</h1>
</body>
</html>
```

Creazione pagina HTML

intestazione H1 con testo del
messaggio

titolo della pagina Page Name



Flask - Esercizio

Renderizza una pagina HTML con Hello world stampato a video nelle diverse dimensioni di intestazione, il cui colore dell'intestazione h3 deve essere verde, h2 blu, h1 rosso

Hello World

Hello world

Hello world

Flask - Soluzione

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="{{css}}>
  <title>Page Name</title>
</head>
<body>
  <h1>Hello World</h1>
  <h2> Hello world </h2>
  <h3> Hello world </h3>
</body>
</html>
```

```
style.css / h3
h1 {
  color: blue;
}

h2 {
  color: green;
}

h3 {
  color: red;
}
```

Flask - Esercizio

Renderizza una pagina HTML inserendo Hello world come h1 e un paragrafo con il tuo nome e cognome e una descrizione dei tuoi dati anagrafici.

Suggerimento: il paragrafo si inserisce con tag `<p>`

Hello World

Mi chiamo Paolo Rossi e sono un programmatore

Utilizzo dei form

Permettono di definire un form per inserire dati, utile nei campi di Login

`form.py`

```
from wtforms import StringField, PasswordField, SubmitField, TextAreaField
from wtforms.validators import DataRequired
from flask_wtf import FlaskForm

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')
```

Utilizzo dei form


Integrazione form in HTML
uso {{“variabile”}}
form.attributo.label -> etichetta
form.attributo(size) -> input field

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="{{css}}>
  <title>Page Name</title>
</head>
<body>
  <h1>Login</h1>

  <form method="POST" action="/login">
    {{ form.hidden_tag() }}
    <div class="form-group">
      {{ form.username.label }}<br>
      {{ form.username(size=32) }}<br>
      {% for error in form.username.errors %}
      <span style="color: red;">[{{ error }}]</span>
      {% endfor %}
    </div>
    <div class="form-group">
      {{ form.password.label }}<br>
      {{ form.password(size=32) }}<br>
      {% for error in form.password.errors %}
      <span style="color: red;">[{{ error }}]</span>
      {% endfor %}
    </div>
  </form>
```

```
<div class="form-group">
  {{ form.submit() }}
</div>
</form>
{% with messages = get_flashed_messages(with_categories=true) %}
{% if messages %}
  <ul>
    {% for category, message in messages %}
    <li class="{{ category }}">[{{ message }}]</li>
    {% endfor %}
  </ul>
{% endif %}
{% endwith %}
</body>
</html>
```

controllo errori e validazione dati in HTML
, pre-check



Utilizzo dei form

Validazione dati inseriti



Login

Username

Password

[Registrati](#)

 Please fill out this field.

The image shows a login form with a title 'Login'. It contains two input fields: 'Username' and 'Password'. Below the 'Password' field is a 'Login' button. At the bottom of the form is a link labeled 'Registrati'. A validation error message is displayed over the 'Password' field, consisting of an orange square icon with a white exclamation mark and the text 'Please fill out this field.'.

Render index con form

importa il login form da form.py

imposta una chiave segreta

```
from flask import Flask, render_template

from form import LoginForm

app = Flask(__name__)
app.config['SECRET_KEY'] = "chiave_segreta"
css_path = "static/style.css"

@app.route('/')
def hello():
    login_form = LoginForm()
    return render_template('index.html', css = css_path, form = login_form)

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

form come parametro

Route

route primaria. Ogni qual volta che si digita indirizzo del server la richiesta verrà indirizzata a questa route e verrà renderizzato index.html

```
from flask import Flask, render_template

from form import LoginForm

app = Flask(__name__)
app.config['SECRET_KEY'] = "chiave_segreta"
css_path = "static/style.css"

@app.route('/')
def hello():
    login_form = LoginForm()
    return render_template('index.html', css = css_path, form = login_form)

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

Route

Form in HTML.

La richiesta con il quale viene processata la navigazione.

Cliccando il pulsante LOGIN in questo caso viene inviata una richiesta di tipo POST al server tramite la route /login.

All'interno della richiesta saranno contenuti i campi username e password inseriti

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="{{css}}>
  <title>Page Name</title>
</head>
<body>
  <h1>Login</h1>

  <form method="POST" action="/login">
    {{ form.hidden_tag() }}
    <div class="form-group">
      {{ form.username.label }}<br>
      {{ form.username(size=32) }}<br>
      {% for error in form.username.errors %}
      <span style="color: red;">[{{ error }}]</span>
      {% endfor %}
    </div>
    <div class="form-group">
      {{ form.password.label }}<br>
      {{ form.password(size=32) }}<br>
      {% for error in form.password.errors %}
      <span style="color: red;">[{{ error }}]</span>
      {% endfor %}
    </div>
  </form>
</body>
</html>
```


Scriviamo la funzione per la route /login

```
np.py > ...
from flask import Flask, render_template, request

from form import LoginForm

app = Flask(__name__)
app.config['SECRET_KEY'] = "chiave_segreta"
css_path = "static/style.css"

@app.route('/')
def hello():
    login_form = LoginForm()
    return render_template('index.html', css = css_path, form = login_form)

@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    return f"Username: {username}, Password: {password}"

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

importa request da flask

per ogni route va gestita
la route tramite una
funzione python

per ogni route va gestita
la route tramite una
funzione python

Scriviamo la funzione per la route /login

```
np.py > ...
from flask import Flask, render_template, request

from form import LoginForm

app = Flask(__name__)
app.config['SECRET_KEY'] = "chiave_segreta"
css_path = "static/style.css"

@app.route('/')
def hello():
    login_form = LoginForm()
    return render_template('index.html', css = css_path, form = login_form)

@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    return f"Username: {username}, Password: {password}"

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

importa request da flask

per ogni route va gestita
la route tramite una
funzione python

per ogni route va gestita
la route tramite una
funzione python.
Funzione avanzata nella
repo