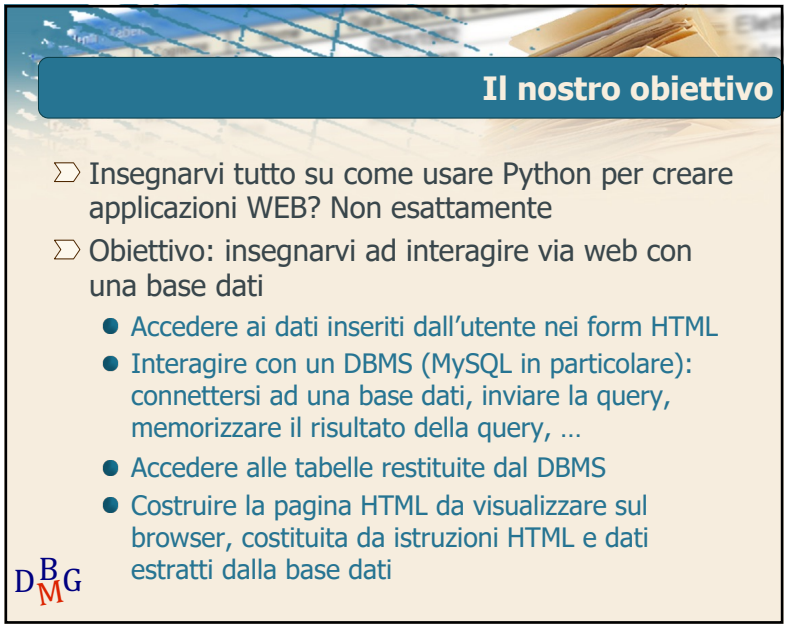

Programmazione Web
Flask
web development, one drop at a time
Il linguaggio Python e Flask
DBMG

1


Il nostro obiettivo
Insegnarvi tutto su come usare Python per creare applicazioni WEB? Non esattamente
Obiettivo: insegnarvi ad interagire via web con una base dati
Accedere ai dati inseriti dall'utente nei form HTML
Interagire con un DBMS (MySQL in particolare): connettersi ad una base dati, inviare la query, memorizzare il risultato della query, …
Accedere alle tabelle restituite dal DBMS
Costruire la pagina HTML da visualizzare sul browser, costituita da istruzioni HTML e dati estratti dalla base dati
DBMG

2

Contenuti

- Panoramica del linguaggio Python
  - Struttura di un programma
  - Template
- Acquisizione dei parametri dai form HTML
  - Validazione dei parametri

DBMG

3

Python frameworks (1)

- Flask
  - microframework
  - Jinja2 templates
  - ORM gestito da altri pacchetti
  - nessuna interfaccia admin
  - nessun sistema di autenticazione
- Django
  - applicazioni complesse
  - Object-Relational Mapping (ORM)
  - Model-Template-View
  - Sistema di autenticazione built-in
  - Interfaccia admin inclusa

Flask

django

DBMG

4

Python frameworks (2)

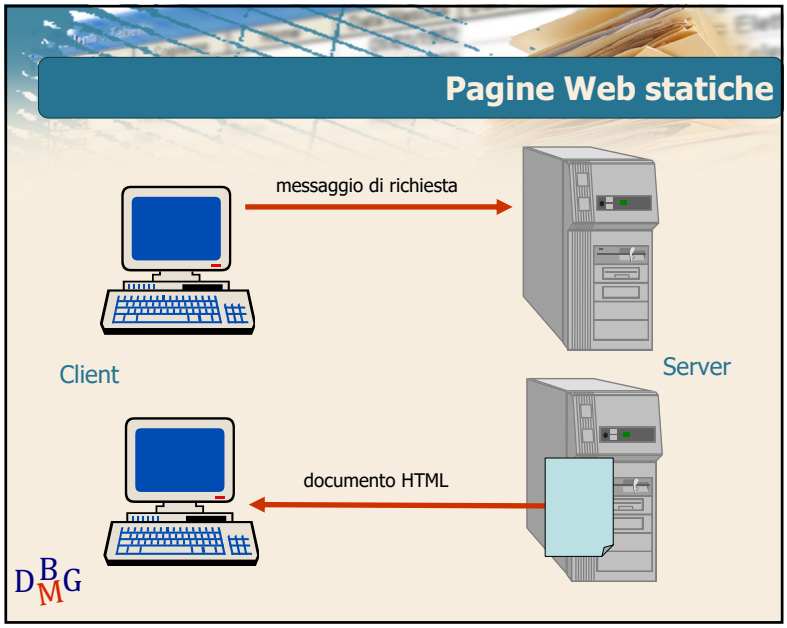
- CherryPy
  - Built-in tools per encoding, sessioni, caching, autenticazione, contenuto statico
  - ORM gestito da altri pacchetti
- Pyramid
  - Object-Relational Mapping (ORM)
  - Mappatura degli URL basata sulla configurazione di Routes
  - Gestione dei templates
  - Sistema di autenticazione flessibile
- Web2Py
  - Supporta l'architettura Model-View-Controller (MVC)
  - Permette di lavorare con database relazionali e NoSQL
- molti altri….

5

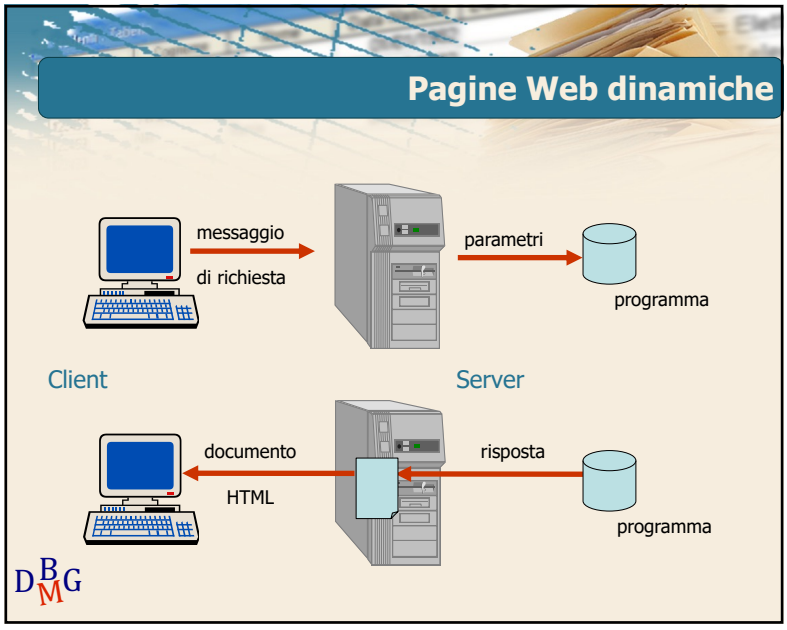
Che cos'è il Flask

- Nasce nel 2010
- Flask è un *micro-framework* sviluppato in Python per la creazione di applicazioni web
  - Offre le *funzionalità di base* per lo sviluppo di applicazioni web
  - Non richiede librerie aggiuntive per le sue funzionalità
  - Estendibile con diverse librerie Python per esternedere le funzionalità
- Moltissime risorse utili, e.g.
  - https://flask.palletsprojects.com/en/2.0.x/

6


Pagine Web statiche
messaggio di richiesta
Client
Server
documento HTML
DBMG

7


Pagine Web dinamiche
messaggio
di richiesta
parametri
programma
Client
Server
documento
HTML
risposta
programma
DBMG

8

Obiettivo principale

- Flask ha come obiettivo principale sviluppare pagine web dinamiche in HTML
  - In particolare, produrre codice HTML "condizionato" ai risultati di una elaborazione, che dipendono dall'input degli utenti, dai dati contenuti in un database, …
  - Il codice Flask viene eseguito sul server Web e il risultato (HTML e risultato dello script) viene inviato al browser

DBMG

9

Perché usare Python?

- È disponibile per molte piattaforme, diverse per
  - Hardware (Intel, Spark, Mac, ecc....)
  - Sistema operativo (Linux,Unix, Windows, ecc...)
- Il codice Python è "altamente portabile"
- L'interprete Python è Open Source
  - Gratuito, ampia disponibilità di strumenti, supporto, sviluppatori, comunità di utenti
- Flask è semplice da imparare conoscendo il Python
- In grado di interagire con vari Database Management Systems (MySql, Postgres, Oracle, ...)

DBMG

10

Primo esempio

- File di testo con estensione .py

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<h1>Hello, World! </h1>"
```

Hello, World!

11

Primo esempio

- Se visualizzo il sorgente sul browser…

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<h1>Hello, World! </h1>"
```

- Perché?
  - Il browser NON visualizza il risultato dell'esecuzione del file Python, ma il file Python
  - Per visualizzare il risultato serve qualcosa esegua il codice

12

Un altro esempio

- Visualizzare la data corrente
- In modo statico
  - E domani?

```
<html>
  <body>
    Today is 09/12/2011
  </body>
</html>
```

- In modo dinamico
  - Si aggiorna in tempo reale

```
from datetime import datetime
from flask import Flask

app = Flask(__name__)

@app.route("/")
def current_date():

    #Get current date in format: DD-MM-YYYY
    today = datetime. today().strftime('%d-%m-%Y')

    return today
```

13

Analisi del codice 1/2

- Import librerie:
  - from datetime import datetime, from flask import Flask
- Creare un oggetto della **classe** Flask:
  - Flask (__name__)
    - __name__ è il nome del modulo
- Creare **decoratori** per legare un URL ad una funzione usata per creare HTML
  - route("/")
- Definire la funzione che deve creare il codice HTML
  - def current_date()

```
from datetime import datetime
from flask import Flask

app = Flask(__name__)

@app.route("/")
def current_date():

    #Get current date in format: DD-MM-YYYY
    today = datetime. today().strftime('%d-%m-%Y')

    return today
```

14

Analisi del codice 2/2

- Commenti:
  - `#… monolinea`
  - `''' multilinea '''`
- Variabili: `today`
- Funzioni: `datetime.today()`
- Operatori e costrutti del linguaggio: `return`

```
from datetime import datetime
from flask import Flask

app = Flask(__name__)

@app.route("/")
def current_date():

    #Get current date in format: DD-MM-YYYY
    today = datetime. today().strftime('%d-%m-%Y')

    return today
```

DBMG

15

Decoratori

- I decoratori sono usati per aggiungere funzionalità ad una funzione
- route(): decoratore legato all'oggetto Flask
  - Legare una funzione ad un URL
  - Indicare ad una funzione quali metodi HTTP gestire (GET/POST)

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<h1>Hello, World! </h1>"
```

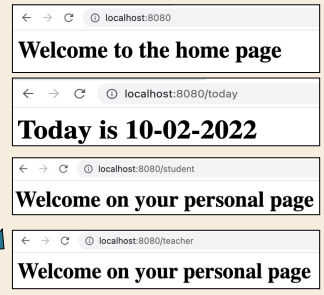
DBMG

16

Decoratori

- Un Applicazione WEB può gestire
  - più funzioni e più decoratori
  - la stessa funzione può gestire più decoratori

```
from datetime import datetime
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home_page():

    return "<h1>Welcome to the home page </h1>"

@app.route("/today")
def current_date():

    today = datetime. today().strftime('%d-%m-%Y')

    return "<h1> Today is " + today + " </h1>"

@app.route("/student")
@app.route("/teacher")
def personal_page():

    #Place dynamic code here

    return "<h1>Welcome on your personal page </h1>"
```



17

Server WEB con Flask

- Flask, oltre ad essere una libreria per sviluppare applicazione Web comprende un **web server**
- Il web server permette di far funzionare **localmente** gli script Python senza connettersi ad un server esterno
  - Il PC diventa client e server
- Il web server crea automaticamente un dominio virtuale (in locale) all'indirizzo di
  - localhost (http://127.0.0.1, http://localhost)
  - Alla porta 5000
  - Non è necessario essere connessi ad Internet per utilizzare il server WEB

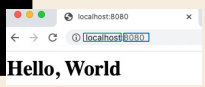
18

FLASK : server Web locale

- Flask può avviare il server web da qualsiasi cartella nel computer in cui è installato

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<h1>Hello, World! </h1>"
```

```
dgiordan@smartdata-danilo:~/web_examples$ export FLASK_APP=1_hello.py
dgiordan@smartdata-danilo:~/web_examples$ flask run -p 8080
 * Serving Flask app '1_hello.py' (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://127.0.0.1:8080/ (Press CTRL+C to quit)
127.0.0.1 - - [10/Feb/2022 12:02:47] "GET / HTTP/1.1" 200 -
```



- Quando avviato la pagina web è accessibile tramite l'indirizzo http://127.0.0.1/ o http://localhost
  - **-p 8080** per specificare la porta dove è visibile la pagina web
    - La porta viene specificata alla fine dell'indirizzo alla pagina
    - Porta di Default 5000 - **WARING MACOS**: La porta è già usata

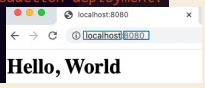
19

FLASK : server Web locale

- Flask può avviare il server web da qualsiasi cartella nel computer in cui è installato

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<h1>Hello, World! </h1>"

app.run(port=8080)
```

```
dgiordan@smartdata-danilo:~/web_examples$ python3 1_hello_run.py
 * Serving Flask app '1_hello_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://127.0.0.1:8080/ (Press CTRL+C to quit)
127.0.0.1 - - [10/Feb/2022 17:58:25] "GET / HTTP/1.1" 200 -
```



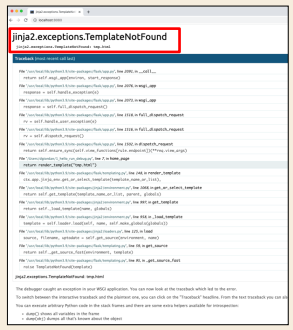
- Quando avviato la pagina web è accessibile tramite l'indirizzo http://127.0.0.1/ o http://localhost
  - **-p 8080** per specificare la porta dove è visibile la pagina web
    - La porta viene specificata alla fine dell'indirizzo alla pagina
    - Porta di Default 5000 - **WARING MACOS**: La porta è già usata

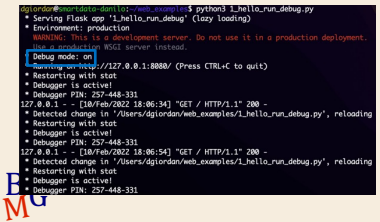
20

FLASK : server Web locale

- Flask può avviare il server web in modalità **debug**
  - Utile per modificare la pagina web e vedere le modifiche senza riavviare il server
  - Identificare possibili **errori** di programmazione

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<h1>Hello, World! </h1>"

app.run(port=8080, debug=True)
```





21

Limitazioni di usare solo Flask

- Flask non è pensato per generare l'intero contenuto di una pagina web ma principalmente per la parte dinamica

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<html>\
                <head>\
                   <title>=Home Page </title>\
                </head>\
                <body>\
                   <h1> Welcome to the home page! </h1>\
                </body>\
               <html>"

app.run(port=8080, debug=True)
```

22

Template

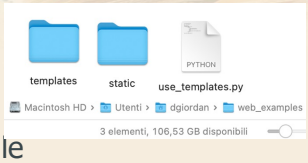
- Oltre che generare codice HTML per generare intere pagine WEB, Flask permette di interagire con dei **template**
  - I template permettono di avere una pagina di base statica, scritta in HTML ed inserire SOLO la parte dinamica con Flask
    - Semplifica la creazione delle pagine web in quando **solo** gli elementi dinamici saranno gestiti da Flask
    - Basati sul linguaggio e motore per template **Jinja2**

DBMG

23

Template

- Per utilizzare template è necessario creare la cartella
  - Template
- È possibile creare altre cartelle per aggiungere risorse statiche
  - CSS, Immagini, etc.



use_template.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def hello_world():
    return render_template("home.html")
app.run(port=8080, debug=True)
```

+

home.html

```
<html>
<head>
    <title> Home Page </title>
</head>
<body>
    <h1> Welcome to the home page </h1>
</body>
</html>
```

=

DBMG

24

Jinja2

- Per rendere i template dinamici si usa **Jinja2**
- Il codice **Jinja2** si inserisce nell'HTML usando dei **delimitatori**
  - **{% ... %} per inserire istruzioni e struttura di controllo**
  - **{{ ... }} per gestire variabili**

home.html

```
<html>
<head>
    <title> Home Page </title>
</head>
<body>
    {% if page_name %}
        <h1> Welcome to the {{page_name}} </h1>
    {% endif %}
</body>
</html>
```

25

Jinja2: Passaggio delle variabili

- Flask passa le **variabili** indicandole come parametri della funzione render_template
- Le variabili possono essere di qualsiasi tipo
  - **Stringhe, numeri, vettori, dizionari**

use_template.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def hello_world():
    return render_template("home.html", page_name="custom name")

app.run(port=8080, debug=True)
```

+

home.html

```
<html>
<head>
    <title> Home Page </title>
</head>
<body>
    {% if page_name %}
        <h1> Welcome to the {{page_name}} </h1>
    {% endif %}
</body>
</html>
```

=

localhost:8080

Welcome to the custom name!

26

Jinja2: Variabili

- Oltre alla visualizzazione delle variabili passate da Flask è possibile definire delle variabili in Jinja2
  - Utile per fare operazioni all'interno del template
  - Si usa l'istruzione **set**

use_template.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def hello_world():
    return render_template("home.html", number = 2)

app.run(port=8080, debug=True)
```

+

home.html

```
<body>
    {% set ris = 5* number%}
        5 * {{number}} = {{ris}} <br>
    {% endif %}
</body>
```

=

localhost:8080

5 *2 = 10

27

Jinja2: Stringhe

- Per concatenare le stringhe è sufficiente inserire le variabili stringa direttamente all'interno del codice HTML

use_template.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/fullnames")
def programs():
    return render_template("fullnames.html", name = "Alpha", surname = "Beta")

app.run(port=8080, debug=True)
```

+

fullnames.html

```
<body>
    <h3> Your name is {{name}}, your surname is {{surname}} </h3>
</body>
```

=

localhost:8080/fullnames

Your name is Alpha, your surname is Beta

28

Principali funzioni per gli array e dizionari

- array | length, dict | length : ritorna il numero di elementi nell'array o nel dizionario
- array | sort: ordina l'array. Altri parametri opzionali specificano come ordinare.
- dict | dictsort: ordina il dizionario. Altri parametri opzionali specificano come ordinare.
- key in array, key in dict: verifica che un elemento ***key***, esista nell'array ***array*** *o nel dizionario* ***dict***

29

Le strutture di controllo

- Le istruzioni e le strutture di controllo permettono l'esecuzione condizionale di parti di programma
- Permettono l'esecuzione iterativa di parti di programma
- Valutano determinate condizioni
- Strutture di controllo in Jinja2
  - if, if..else, if..elseif
  - for

30

Le condizioni

- Una condizione è un'espressione che genera un valore booleano (vero o falso)
  - Utilizzano gli operatori di confronto e gli operatori booleani
- Sono equivalenti a falso (false)
  - Il valore booleano false
  - Il numero intero 0 e il numero reale 0.0
  - La stringa vuota ("") e la stringa "0"
  - Un array vuoto
- Ogni altro valore è considerato vero (true)

31

Jinja2: Stringhe

- Per concatenare le stringhe è sufficiente inserire le variabili stringa direttamente all'interno del codice HTML

use_template.py

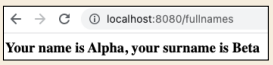
```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/fullnames")
def programs():
    return render_template("fullnames.html", name = "Alpha", surname = "Beta")

app.run(port=8080, debug=True)
```

+

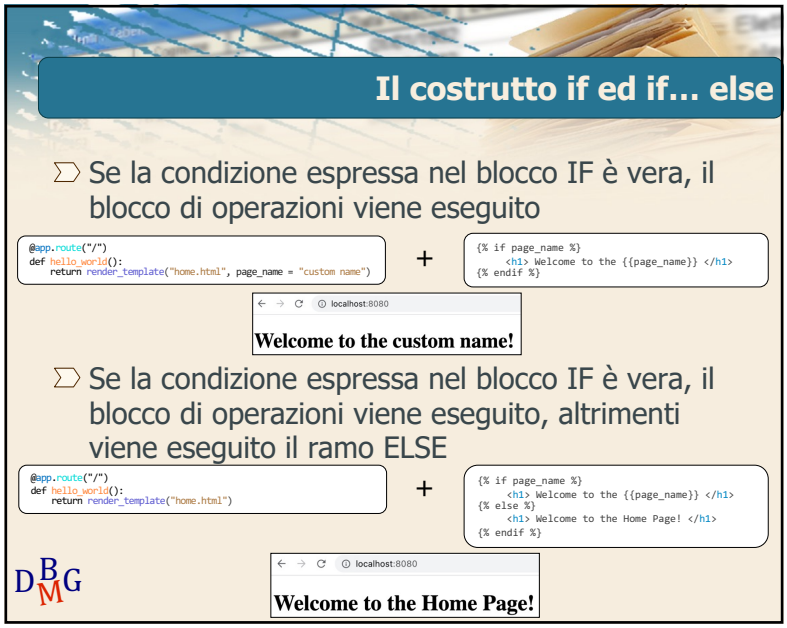
fullnames.html

```
<body>
    <h3> Your name is {{name}}, your surname is {{surname}} </h3>
</body>
```

=



32


Il costrutto if ed if... else
Se la condizione espressa nel blocco IF è vera, il blocco di operazioni viene eseguito
@app.route("/")
def hello_world():
return render_template("home.html", page_name = "custom name")
+
{% if page_name %}
<h1> Welcome to the {{page_name}} </h1>
{% endif %}
localhost:8080
Welcome to the custom name!
Se la condizione espressa nel blocco IF è vera, il blocco di operazioni viene eseguito, altrimenti viene eseguito il ramo ELSE
@app.route("/")
def hello_world():
return render_template("home.html")
+
{% if page_name %}
<h1> Welcome to the {{page_name}} </h1>
{% else %}
<h1> Welcome to the Home Page! </h1>
{% endif %}
DBMG
localhost:8080
Welcome to the Home Page!

33


Il costrutto if .. elseif
Consente di scegliere fra più opzioni
@app.route("/")
def hello_world():
return render_template("home.html", page_name = "about")
+
</body>
{% if page_name=="home" %}
<h1 style="color:blue"> Welcome to the Welcome to the Home Page! </h1>
{% elif page_name== " about" %}
<h1 style="color:red"> Welcome to the About Page! </h1>
{% else %}
<h1> Welcome to the {{page_name}}! </h1>
{% endif %}
</body>
=
localhost:8080
Welcome to the About Page!
DBMG

34

Il ciclo for

- Consente di ripetere un blocco di istruzioni definendo direttamente
  - Le istruzioni di inizializzazione, eseguite una sola volta all'ingresso del ciclo
  - La condizione, che deve essere vera per eseguire il blocco di istruzioni
  - L'aggiornamento, eseguito al termine di ogni iterazione

```
<body>
    {% for i in range(1,11) %}
        {% set ris=5*i %}
        5 *{{i}} = {{ris}} <br>
    {% endfor %}
</body>
```

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

DBMG

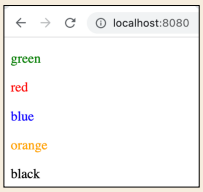
35

Il ciclo for

- Come per il Python anche in Jinja2 è possibile iterare su
  - **vettori**
  - dizionari

```
@app.route("/")
def hello_world():
    colors = ["green", "red", "blue", "orange", "black"]
    return render_template("home.html", colors = colors)
```

```
</body>
    {% for color in colors %}
        <p style="color:{{color}}"> {{color}} </p>
    {% endfor %}
</body>
```



DBMG

36


Il ciclo for
Come per il Python anche in Jinja2 è possibile iterare su
vettori
dizionari
@app.route("/")
def hello_world():
agenda = [
{"surname": "Rossi",
"name": "Francesca",
"mobile": 3331234567
},
{"surname": "Verdi",
"name": "Mario",
"mobile": 3337654321
}
]
return render_template("home.html", agenda = agenda)
</body>
{% for record in agenda %}
<h3> {{record.surname}} {{record.name}} </h3>
Mobile {{record.mobile}}
{% endfor %}
</body>
localhost:8080
Rossi Franesca
Mobile 3331234567
Verdi Mario
Mobile 3337654321

37


Il ciclo for
Come per il Python anche in Jinja2 è possibile iterare su
vettori
dizionari
@app.route("/")
def hello_world():
agenda = {
"one": {"surname": "Rossi",
"name": "Francesca",
"mobile": 3331234567
},
"two": {"surname": "Verdi",
"name": "Mario",
"mobile": 3337654321
}
}
return render_template("home.html", agenda = agenda)
</body>
{% for record in agenda %}
<h3> {{agenda[record].surname}} {{agenda[record].name}} </h3>
Mobile {{agenda[record].mobile}}
{% endfor %}
</body>
localhost:8080
Rossi Franesca
Mobile 3331234567
Verdi Mario
Mobile 3337654321

38

Estensione Template

- Oltre la creazione di template è possibile **estendere** template già esistenti
  - Si crea uno **template base** contenente lo scheletro della pagina con tutti gli elementi in comune tra le pagine
  - I **template figli** ereditano la base definendo solo un sottonisieme di **blocchi** da modificare

39

Estensione Template: template base

- Il template base definisce le parti dinamiche tramite
  - **{% ... %} per inserire istruzioni e struttura di controllo**
- I blocchi di contenuto definiti nei template figli sono identificati da
  - **{% block <nome> %}**
  - **{% endblock <nome> %}**

layout.html

```
<html>
<head>
{% if title %}
    <title> {{ title }} </title>
{% else %}
    <title> Default Title </title>
{% endif%}
</head>
<body>
    {% block content %}
    {% endblock content %}
</body>
</html>
```

40

Estensione Template: template figli

- I template figli identificano il template base tramite l'istruzione di controllo
  - **{% extends "nome" %}**
- I template figli definiscono quale blocco di dati vogliono andare a modificare
  - **{% block <nome> %}**
  - **{% endblock <nome> %}**

home.html

```
{% extends "laoyout.html" %}
{% block content %}
    <h1> Welcome to the Home Page! </h1>
{% endblock content %}
```

agenda.html

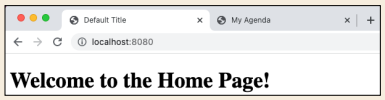
```
{% extends "laoyout.html" %}
{% block content %}
  {% for record in agenda %}
    <h3> {{agenda[record].surname}} {{agenda[record].name}} </h3>
    Mobile {{agenda[record].mobile}}
  {% endfor %}
{% endblock content %}
```

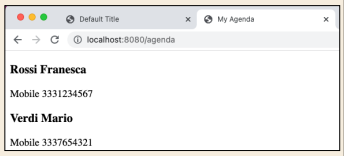
41

Estensione Template: Flask

- L'estensione è trasparente a Flask
  - Vengono richiamati direttamente i template figli
- Ai template figli possono essere passati
  - Un numero diverso di parametri
  - Parametri diversi

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def agenda_page():
    return render_template("home.html")

@app.route("/agenda")
def agenda_page():
agenda = {
                "one": {"surname": "Rossi",
               "name": "Francesca",
               "mobile": 3331234567
              },
                "two": {"surname": "Verdi",
               "name": "Mario",
               "mobile": 3337654321
              }
            }
    return render_template("agenda.html", agenda = agenda,
                           title = "My Agenda")
app.run(port=8080, debug=True)
```





42

Flask e form HTML

```
<form name="UserData" action="processlogin" method="GET">
    Input Elements
</form>
```

- Tag "form" con alcuni attributi
  - Name: nome del form
  - Action: nome del programma che elaborerà i dati del form
  - Method: modalità in cui vengono passati i parametri dal form al programma (può essere "GET" o "POST")
- All'interno del form ci sono più elementi di input

DBMG

43

Accesso ai dati del form

- Per inviare i dati del form alla pagina di risposta (**action**)
  - sia la pagina del form che la pagina di risposta **devono essere gestite** dal web server
- Per accedere ai parametri passati tramite un form è necessario importare l'oggetto **request** di Flask
  - **from** flask **import** request
- Le funzioni specificano il **metodo** che accettano per ricevere i dati all'interno del decoratore
  - route("/answer", **method**=['GET','POST'])
  - Se il metodo non è specificato Flask accetta solo parametri da richieste di tipo GET

DBMG

44

Accesso ai dati del form

- Metodo GET
  - Per accedere ai parametri passati tramite GET nell'URL (?key=value) è possibile usare l'attributo args dell'oggetto **request**
  - **request**.args.get("<parametro>")
- Metodo POST
  - Per accedere ai parametri passati tramite POST è possibile utilizzare direttamente l'oggetto request come vettore associativo
  - **request**.form["<parametro>"], **request**.files["<parametro>"]

DBMG

45

Accesso ai dati del form

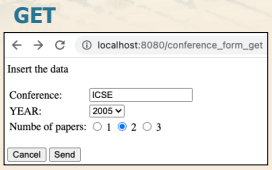
- Se sia il metodo get GET che POST possano essere usati
  - Prima di accedere ai parametri è necessario verificare il metodo utilizzato nell'oggetto request
  - request.method == 'GET'
- L'oggetto request può essere usato per ottenere altre informazioni
  - Cookie, sessioni, etc.

DBMG

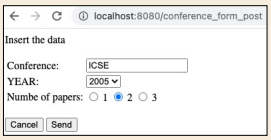
46

Esempio: Form

GET



POST




```
<html>
<head>
   <title>GET form</title>
</head>
<body>
   <p> Insert the data </p>
   <form method="GET" action="conference">
   <table>
      <tr>
          <td> Conference: </td>
          <td> <input type="text" name="conf" size="20"> </td>
      </tr>
      <tr>
          <td> YEAR: </td>
          <td> <select name="year">
                  <option value="2005"> 2005 </option>
                  <option value="2006"> 2006 </option>
               </select> </td>
      </tr>
      <tr>
          <td> Number of papers: </td>
          <td> <input type="radio" name="num" value="1"> 1
               <input type="radio" name="num" value="2" checked> 2
               <input type="radio" name="num" value="3"> 3 </td>
       </tr>
     </table> <br />
     <input type="reset" value="Cancel">
     <input type="submit" value="Send">
</form>

</body>
</html>
```

47

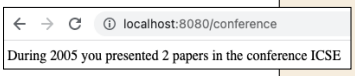
Esempio: Ricezione Parametri FORM

File conferences.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/conference_form_get")
def get_form():
    return render_template("conference_from_get.html")

@app.route("/conference_form_post")
def post_form():
    return render_template("conference_from_post.html")

@app.route("/conference", method=["GET","POST"])
def read_form():

    if(request.method == 'GET'):
        conf = request.args.get('conf')
        year = request.args.get('year')
        num  = request.args.get('num')
    else:
        conf = request.form['conf']
        year = request.form['year']
        num  = request.form['num']

    output_string = "During %s you presented %s papers in the conference %s"%(year,num,conf)

    return render_template("conference.html", output_string = output_string)

app.run(port=8080, debug=True)
```

conference.html

```
<html>
<head>
   <title>View form parameters</title>
</head>
<body>
     <p> {{ output_string}} </p>
</body>
</html>
```



48

Esempio: calcolatrice

Insert the numbers
50 + 20 Cancel Compute

the result is: 70

```
<html>
<head>
   <title>Calculator</title>
</head>
<body>
   <p> Insert the numbers </p>
   <form method="GET" action="result">
      <input type="text" name="val1" size="8" maxlength="8">
      <select name="year">
            <option value="sum"> + </option>
            <option value="sub"> - </option>
            <option value="mul"> * </option>
            <option value="div"> / </option>
      </select>
      <input type="text" name="val2" size="8" maxlength="8">
      <input type="reset" value="Cancel">
      <input type="submit" value="Compute">
</form>

</body>
</html>
```

49

Esempio: calcolatrice

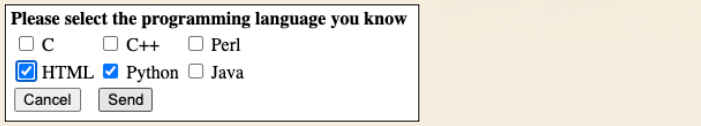
```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/calculator")
def get_form():
   return render_template("calculator.html")

@app.route("/result", method=["GET"])
def read_get_form():
    val1 = request.args.get('val1')
    val2 = request.args.get('val2')
    op   = request.args.get('op')

    if(val1=="" or val2==""):
        return render_template('result.html', error_message = "Missing one or more numbers. Please check the input.")
    if(val1.isnumeric() == False or val2.isnumeric() == False):
        return render_template('result.html', error_message = "Only integer number are supported. Please check the input.")

    val1 = int(val1)
    val2 = int(val2)
    if(op == "div" and val2 == 0):
        return render_template('result.html', error_message = "Error try division by zero.")

    result = 0
    if(op == "sum"):
        result = val1 + val2
    if(op == "sub"):
        result = val1 - val2
    if(op == "mul"):
        result = val1 * val2
    if(op == "div"):
        result = val1 / val2

    return render_template('result.html', result = result)

app.run(port=8080, debug=True)
```

result.html

```
<html>
<head>
   <title>Result</title>
</head>
<body>
   {% if error_message %}
        <h3 style="color:red"> {{ error_message}} </h3>
   {% else %}
        <h3> the result is: {{ result }} </h3>
    {% endif %}
</body>
</html>
```

50

Esempio: scelta multipla



You know 2 programming languages:

- HTML
- Python

51

Esempio: scelta multipla

- Form HTML
  - Utilizza l'array langs invece di 6 variabili distinte

```
<html>
<head>
   <title>Programming Language</title>
</head>
<body>
   <p> Please select the programming language you know </p>
   <form method="GET" action="known">
   <table>
      <tr>
          <td> <input type="checkbox" name="langs" value="C"> C </td>
          <td> <input type="checkbox" name="langs" value="C++"> C++ </td>
          <td> <input type="checkbox" name="langs" value="Perl"> 3 </td>
       </tr>
       <tr>
          <td> <input type="checkbox" name="langs" value="HTML"> HTML </td>
          <td> <input type="checkbox" name="langs" value="Python"> Python </td>
          <td> <input type="checkbox" name="langs" value="Java"> Java </td>
       </tr>
       <tr>
           <td> <input type="reset" value="Cancel">
           <td> <input type="submit" value="Send">
        </tr>
   </table>
   </form>
</body>
</html>
```

52

Esempio: scelta multipla

- Script Python
  - L'array languages contiene tutti i valori value selezionati (HTML, Python in questo caso)

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/programs")
def get_form():
   return render_template("programs.html")

@app.route("/known", method=["GET"])
def known():

  languages = request.args.getlist('langs')
  return render_template('known.html', languages = languages)

app.run(port=8080, debug=True)
```

```
<html>
<head>
   <title>Known Programming Language</title>
</head>
<body>
   {% if languages | length == 0 %}
      <h3>You don't know any programming language.</h3>
   {% else %}
      <p> You know {{languages | length }} programming languages </p>
      <ul>
      {% for l in languages %}
          <li> {{l}} </li>
      {% endfor %}
      </ul>
    {% endif %}
</body>
</html>
```

You know 2 programming languages:

- HTML
- Python

53

Controllo dei valori inseriti

- Prima di processare i dati forniti dall'utente conviene sempre **validarli**
  - Evita di processare dati errati
    - E.g., l'inserimento di un indirizzo email non correttamente formattato, o di un valore non previsto
  - Utile per evitare possibili attacchi informatici
    - E.g., l'inserimento di query SQL in un campo per visualizzare il contenuto del DB

54

Validazione dei dati

- Verificare che l'età inserita dall'utente rispetti i vincoli del servizio (di avere almeno 16 anni)

```
from flask import Flask, render_template, request

app = Flask(__name__)

def check_age(age):

    if(age == ""): return False
    if(age.isnumeric() == False): return False
    if(int(age) < 16): return False

    return True

@app.route("/register")
def register_form():
    return render_template("register.html")

@app.route("/registered", method=["GET"])
def process_registration():

surname = request.args.get('surname')
    age     = request.args.get('age')

    if(checkAge(age) == False):
        return render_template('error_page.html', errror = 'Invalid Age')

    return render_template('registered.html')

app.run(port=8080, debug=True)
```

Verifica l'inserimento del campo age

Verifica che age sia un numero

Verifica il vincolo dell'età

55

Validazione dei dati

- Prima di utilizzare i dati è necessario
  - Verificare che l'utente abbia dati inseriti
  - Effettuare una validazione formale per verificare che **il tipo** sia corretto
    - La verifica formale può essere effettuata sfruttando funzioni e librerie offerte dal Python, espressioni regolari, etc.
  - Validare eventuali vincoli
    - E.g., il limite minimo di età

56

Una breve parentesi: XAMPP

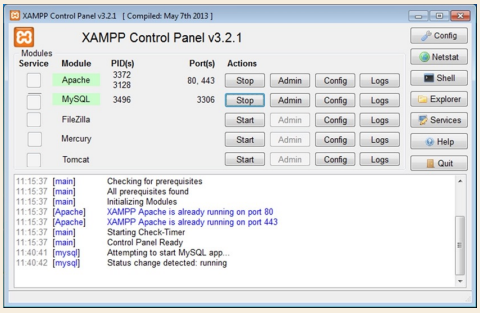
- XAMPP è una piattaforma di sviluppo Web (ambiente di sviluppo web-database) che comprende
  - Un web server (Apache)
  - Un database management system (MySQL)
  - Un interprete di script PHP e PERL
  - **Un amministratore grafico di db MySQL (phpMyAdmin)**
- Permette di far funzionare localmente un server MySQL

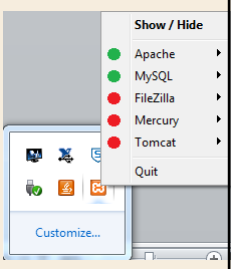
DBMG

57

XAMPP : amministrazione servizi

- Consente di gestire i servizi
  - Interfaccia grafica





DBMG

58


XAMPP : amministrazione DB
Consente di gestire le basi di dati
Interfaccia grafica indirizzo: localhost/phpmyadmin
Se non è specificata una porta diversa nel server Apache
phpMyAdmin
hotel
città
hotel
locazione
stanza
Crea tabella
127.0.0.1
hotel
Struttura
SQL
Cerca
Query da esempio
Tabella
Azione
città
hotel
locazione
stanza
Mostra
Struttura
Cerca
Inserisci
Svuota
4 tabelle
Totali
Seleziona tutti / Deseleziona tutti
Se selezionati:
Visualizza per stampa
Data Dictionary

59