Insegnamento di Laboratorio di Basi di Dati

Lezione 8: Introduzione alle applicazioni web che usano basi di dati

Roberto Posenato

ver. 1.4, 22/03/2018

Sommario

1. Scopo di questa lezione è dare un'introduzione allo sviluppo di applicazioni web basate su base di dati.
2. Data la vastità di tecnologie disponibili e la loro continua evoluzione, questa introduzione si limiterà a introdurre le applicazioni web secondo l'architettura a 3 strati (3-tier architecture). Si cercherà di chiarire anche l'ambiguità con il pattern di sviluppo Model-View-Controller (MVC).

3. Quindi, come esempio di sviluppo, si introdurrà:
   1. la piattaforma Flask per Python che permette di sviluppare applicazioni web secondo il modello MVC in modo relativamente rapido.
   2. una semplice applicazione web che permette di interrogare la base di dati 'did2014'.

Applicazione client-server

Inizia lo scambio da client a server

- Un'applicazione web (web app) è un'applicazione di tipo client-server fruibile via Web per mezzo di un network (intranet/internet).
- Funzionamento classico di una web app:

Invio documenti (audio,video)

Formato html

1. un client (web browser) invia una richiesta di servizio al server (dove è presente la web app) attraverso il protocollo http.
2. Il server realizza il servizio inviando al client uno o più documenti in formato html visualizzabili dal browser. Per visualizzare il contenuto all'utente

- Un'approssimazione valida dell'architettura per la maggioranza delle web app è l'architettura a tre strati (three-tier architecture):

Divide molto bene le competenze che devono essere ottenute

1. logica di presentazione (presentation layer): presentazione a favore dell'utente (front-end) attraverso un browser. Solitamente fatta di pagine HTML e CSS, JSP/JSF, Javascript o utilizzando un framework). Questo strato NON comprende il browser!

Parte della app. Che si preoccupa di far vedere le pagine all'utente
Assume la presenza che ci sia già un browsers

Core dell'app. Applicativo ch sta sul server scritto in un linguaggio di programmazione raccoglie info e si interfaccia al livello layer e si preoccupa di interpre

2. logica di business (business logic): applicativo (back-end) o logica applicativa tipicamente su un application server. Solitamente fatto di codice PHP(!!)/ASP/servlet/JavaAction in Java/Flask, ecc.; riceve, elabora e soddisfa le richieste del client. Questo strato è ciò che differenzia una web app da un normale sito web.

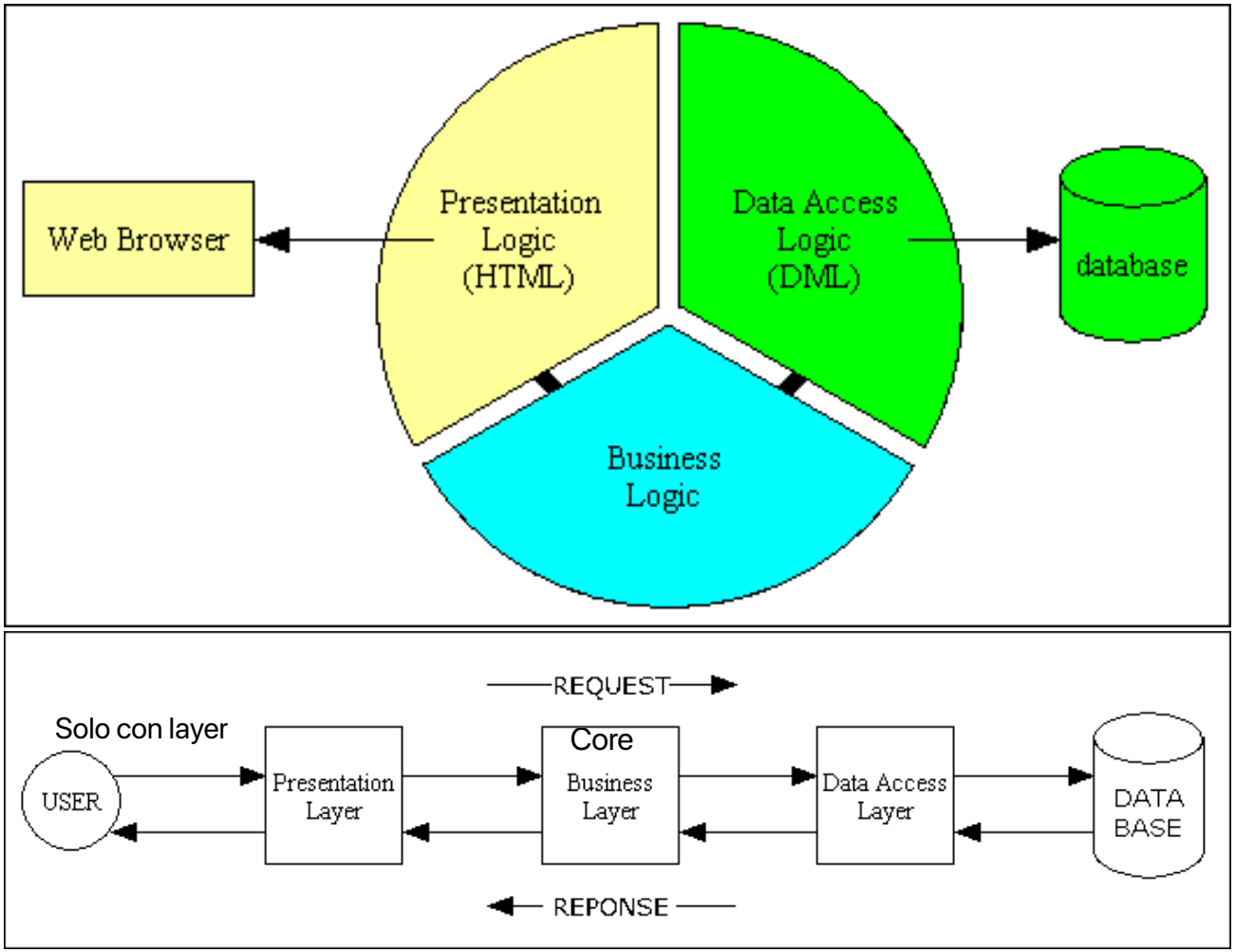
Non ha le info già memorizzate e in base alle richieste mette insieme le info e le invia al livello layer per poterle interpretare

Usano sistemi esterni (sistemi esterni) per raccogliere le info richieste

3. strato dati (data layer): componente (Data Access Object (DAO)) che gestisce il recupero e la rappresentazione dei dati presenti in un DBMS attraverso interfacce semplici come JDBC/DB-API2 o interfacce più sofisticate (Object Relation Mapping) come Hibernate/Ibatis, ecc. Questo strato NON comprende il DBMS!

Web application comprende anche una base di dati ma non comprende i servizi che necessitano per ottenere informazioni

Referenza: `www.tonymarston.co.uk/php-mysql/3-tier-architecture.html`

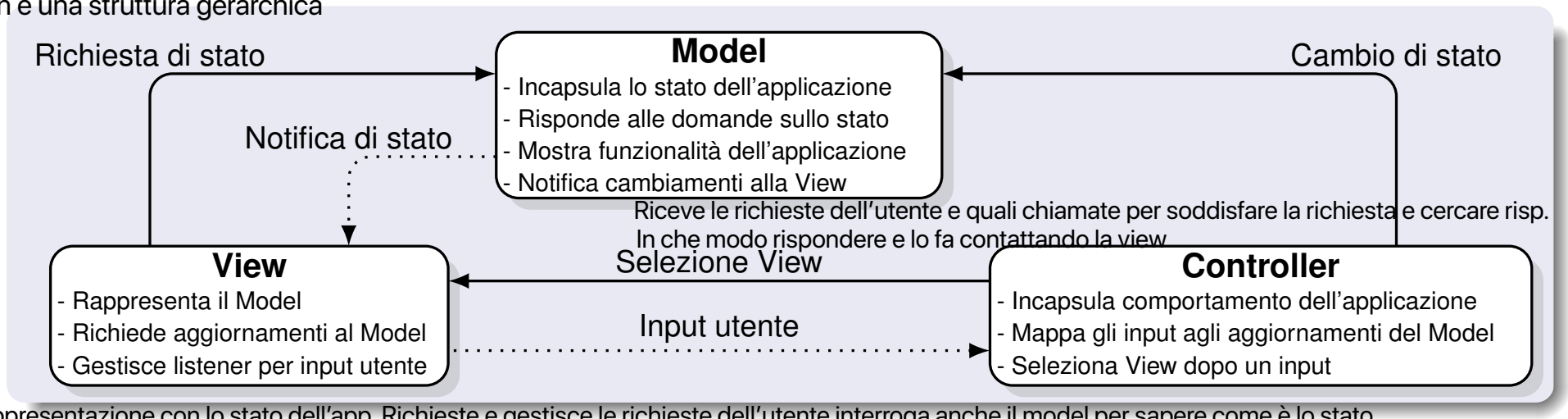

Web Browser
Presentation Logic (HTML)
Data Access Logic (DML)
database
Business Logic
REQUEST
Solo con layer
Core
USER
Presentation Layer
Business Layer
Data Access Layer
DATA BASE
REPONSE

MVC pattern per applicazioni (MA. NON PER QUELLE WEB)

- Pattern architetturale per la progettazione e strutturazione modulare di applicazioni software interattive introdotto nel 1979.
- Separa il modello dei dati (model) e la logica applicativa (controller) dalle modalità di visualizzazione.

Parte di applicazione (1 o piu classi) rappresenta lo stato dell'applicazione e rende chiaro le funzioni dell'applicazione che usa

Non è una struttura gerarchica



Rappresentazione con lo stato dell'app. Richieste e gestisce le richieste dell'utente interroga anche il model per sapere come è lo stato

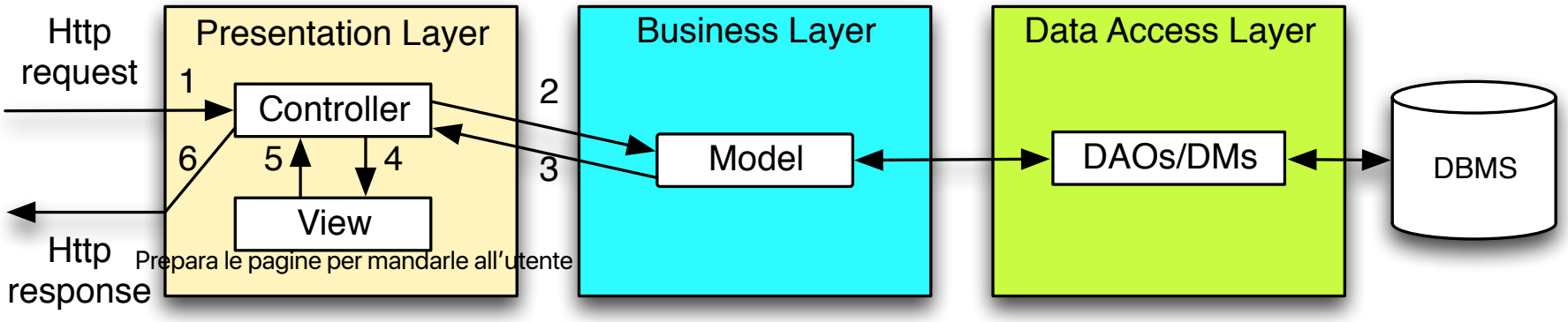
Nota!

Il modello MVC NON è una rappresentazione alternativa dell'architettura a 3 strati:

- in un'architettura a 3 strati le 3 componenti sono organizzare in una gerarchia lineare stretta;
- nel MVC le 3 componenti sono organizzate in una struttura triangolare!

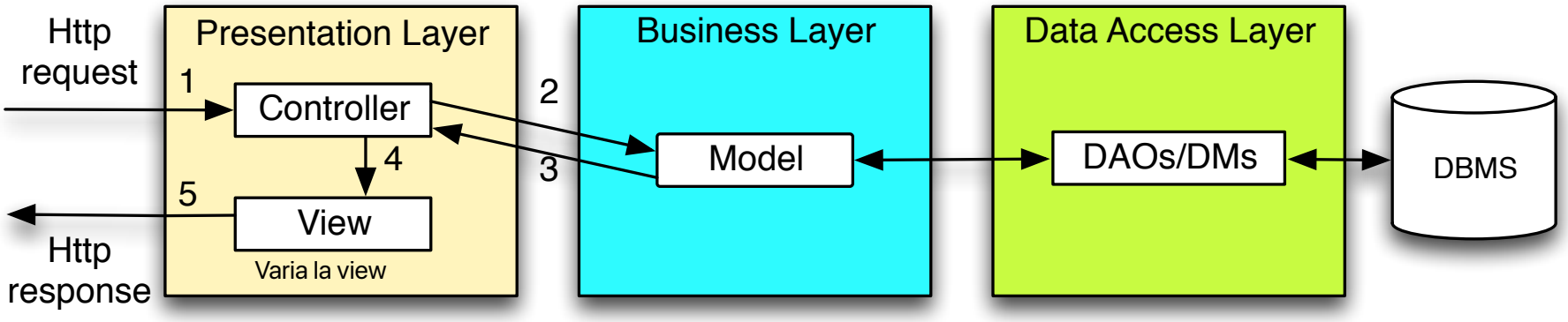
- Il pattern MVC originale NON è adatto per sviluppare web app con architettura a 3 strati in modo corretto!
- Il pattern MVC è comunque un buon pattern per individuare le componenti di una web app.
- Si può conciliare il MVC con l'architettura a 3 strati, adattando il MVC come in figura (due versioni).

Modellato come una gerarchia a 3 strati
Versione 1



- Il pattern MVC originale NON è adatto per sviluppare web app con architettura a 3 strati in modo corretto!
- Il pattern MVC è comunque un buon pattern per individuare le componenti di una web app.
- Si può conciliare il MVC con l'architettura a 3 strati, adattando il MVC come in figura (due versioni).

Versione 2



Dare significato a qualche parte di testo per marcarli

- Linguaggio a marcatori solitamente usato per la formattazione e impaginazione di documenti ipertestuali disponibili nel World Wide Web sotto forma di pagine web.

Lettura non semplicemente sequenziale ma poter saltare da un punto ad un altro

Arricchisco il testo

- Anche se HTML permette l'inserimento di script e oggetti esterni quali immagini o filmati, non è un linguaggio di programmazione.
- Scopo principale del linguaggio è gestire i contenuti (sia strutturalmente, sia graficamente) all'interno della pagina web mediante l'uso di marcatori (tag).
- Ogni tag (ad esempio `<h1>` o `<p>`) specifica un diverso ruolo dei contenuti che esso contrassegna.
- La formattazione consiste nell'inserimento nel testo di tag.
- I browser che leggono il codice mostrano all'utente formattazioni predefinite per ogni tag che incontrano.
- Quando un documento ipertestuale scritto in HTML è memorizzato in un file la sua estensione è tipicamente `.html`.

7 versioni differenti

- La struttura generale di un file HTML5 è: Standard W3C con tutti i browsers
  <!DOCTYPE html>
  <html>
  intestazione + corpo
  </html>
- *Intestazione*: <head>...</head> contiene informazioni sul documento come, ad esempio, il titolo (<title>...</title>)
- *Corpo*: <body>...</body> contiene il testo del documento e i tag per la presentazione.

```
<!DOCTYPE html> Dichiaro che il testo è HTML
<html>
<head>
   <title>Programma lab. di basi
     di dati</title>
</head>
<body>
   <h1>Registro del modulo di
     Laboratorio di Basi di
     dati</h1>
   <h3>Obiettivi formativi</h3>
   <p>Questo modulo...</p>
...
</body>
</html>
```

- Fin dall'inizio (1990), HTML ha avuto un'evoluzione non sempre lineare che ha portato a diverse versioni del linguaggio con diverse e contrastanti approcci: 3.2, 4.01, XHTML 1.0, HTML5.
- Non esiste una versione recente supportata completamente dai browser più diffusi (Chrome, Firefox, Internet Explorer, Safari, etc)
- La versione HTML5, più recente, sembra essere il punto di partenza di un nuovo approccio condiviso. Alcune novità:
  - Semplificazione della notazione dei tag.
  - Nuovi tag per rendere più semplice integrazione di audio e video.
  - Nuovi tag per migliorare la strutturazione logica dei documenti.
  - Rinuncia ai tag di formattazione estetica (come <FONT>, <B>, ecc) in favore dei fogli di stile (Cascade Style Sheet (CSS).

- Si rimanda allo studente l'apprendimento del linguaggio nella sua versione 5.
- Si consiglia di considerare fonti autorevoli quali:
  - Specifica ufficiale del W3C: `https://www.w3.org/TR/html5`
  - Introduzione più concreta del gruppo Mozilla: `https://developer.mozilla.org/it/docs/Web/HTML/HTML5`
  - Introduzione più semplice (ma con qualche imprecisione): `http://www.html.it/guide/guida-html5/`

Nota!

Per il resto del corso, si assume che lo studente sia in grado scrivere semplici documenti HTML5.

In particolare si assume la conoscenza dei tag (e loro attributi): `<link>`, `<style>`, `<article>`, `<section>`, `<p>`, `<ul>`, `<ol>`, `<table>`, `<form>` `<input>` (importante!), `<br>`, `<a>`, `<span>`.

Liste non ordinate
Liste ordinate
Link
Sotto

- Le prime versione di HTML prevedevano tag o attributi anche per personalizzare aspetti grafici: <font>, , , <table border=1>, ecc.
  Con html 5 tolti
- Dal 1994 si è iniziato a proporre di separare la specifica degli aspetti grafici dei tag dai tag stessi.
  File che descrive per ogni tag descrive le caratteristiche grafiche
- Il Cascade Style Sheet (CSS) è un linguaggio per definire la formattazione di documenti HTML, XHTML e XML.
- CSS permette di definire gli aspetti visivi/sonori di ciascun tipo di tag, di classi di tag o di specifici tag.
- Data la natura gerarchica dei documenti HTML, un tag può ereditare più specifiche: esse vengono applicate in modo gerarchico (cascade).
- Un foglio di stile si associa a un documento HTML usando o il tag `<style>` o il tag `<link>` nello `<head>`.

- La struttura generale di un foglio CSS è un elenco di blocchi di dichiarazione CSS:
  selettore [, ...] { *proprietà* : *valore* [; ...] }
- *selettore*: è un tag, o un identificatore di classe (inizia con '.'), o un identificatore di un tag specifico (inizia con '#').
- *proprietà*: è il nome di un aspetto grafico da controllare. Esempio: border, color, margin, ecc.
- *valore*: è il valore che si vuole assegnare.

```
/*Font e dimensione per i tag
      html, body e table*/
html, body, table {
    font-family: Arial,sans-serif;
    font-size: small
}

/*Bordo grigio per tutte le
      celle di tabelle, per tutti
      i tag di classe 'conBordo'
      e per un elemento con id
      'box14'*/
td, .conBordo, #box14 {
    border-width: 1px;
    border-style: solid;
    border-color: gray
}
```

Tag

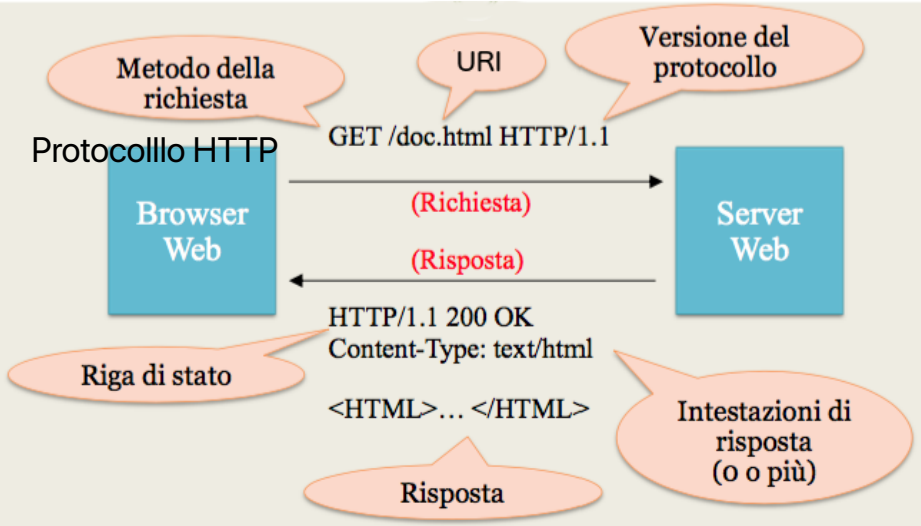
Cella tabella

- La versione attuale di CSS è la 3, anche se solo la 2 è quella maggiormente supportata.
- Si rimanda allo studente l'apprendimento del linguaggio nella sua versione 2.
- Si consiglia di considerare fonti autorevoli quali:
  - Specifica ufficiale del W3C: `https://www.w3.org/Style/CSS/learning`
  - Introduzione più concreta e completa del gruppo Mozilla: `https://developer.mozilla.org/it/docs/Web/CSS`
  - Introduzione più semplice: `http://it.html.net/tutorials/css/`

Nota!

Per il resto del corso, si assume che lo studente sia in grado scrivere semplici fogli di stile e che mai inserisca in documenti HTML tag deprecati per la modifica di aspetti di visualizzazione come <font>, <b>, <i>, ecc.

- HTTP è un protocollo senza stato a livello applicativo per la scambio di messaggi (e invio di documenti HTML), nel Web, tra browser (client) e web server.
- HTTP prescrive le regole mediante le quali i browser effettuano le richieste e i server forniscono le relative risposte.
- In generale, il funzionamento prevede che un client inizi una comunicazione inviando una richiesta secondo un formato specifico e il server restituisce la risposta chiudendo poi la comunicazione.



- Uniform Resource Identifier (URI): identificatore di una risorsa generica come un un documento, un'immagine, ...

  Esempio: urn:example:mammal:monotreme:echidna

Località di una risorsa

- Uniform Resource Locator (URL): è un URI che indica anche dove la risorsa è disponibile ("location") in una rete e il meccanismo di accesso primario. Un URL è semplicemente un URI che indica una risorsa fisica nella rete:

  Servizio dove si trova

  `scheme://[user:password]host[:port]][/]path[?query][#fragment]`

  Protocollo

Esempio di URL

Location

`http://www.di.univr.it/?ent=oi&cs=4&discr=&discrCd=&id=40079`

Se c'è un unico servizio prima del ? Non scrivo nulla perchè sarà solo quello

Path -> percorso e eventuale ?querystring

Quale servizio che vogliamo accedere e ? Indica quali parametri per il servizio

QueryString Parametri che il servizio deve avere per preparare la richiesta

- scheme = `http`
- host = `www.di.univr.it`
- path = *vuoto*
- query = `ent=oi&cs=4&discr=&discrCd=&id=40079`
- fragment = *vuoto*

Se il valore 'port' in un URL con schema='http' non è specificato, allora è il valore standard 80.

Esempio funzionamento 1/2

1. In un browser si inserisce URL: http://www.di.univr.it/?ent=oi&codiceCs=S24&codins=12700&cs=420 e si preme invio.
2. Il browser apre una connessione sulla porta 80 del server web www.di.univr.it.
3. Il browser invia il messaggio (versione minima!) composto dalla riga
   GET ?ent=oi&codiceCs=S24&codins=12700&cs=420 HTTP/1.1
   seguita da una riga con solo il carattere di fine riga \n Indica fine messaggio
4. Il server web restituisce un messaggio (testo) che inizia con:

```
HTTP/1.1 200 OK
Date: Tue, 03 May 2016 09:33:36 GMT
Server: Apache/2.2.3 (Red Hat)...
```

Riga vuota

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 ...
<html><head>
<title>Insegnamenti-Dep.Computer Science-University of
     Verona</title>
...
```

Esempio funzionamento 2/2

1. Il browser analizza intestazione messaggio ricevuto: se è presente HTTP/1.1 200 OK, significa che la richiesta è stata soddisfatta (documento presente dopo riga vuota).
2. Il browser legge quindi il codice HTML e visualizza il risultato.
3. Se il codice HTML contiene riferimenti ad altre risorse che devono essere caricate con il documento, allora il browser invia una richiesta per ogni risorsa necessaria.

Possibili richieste via HTTP/1.1

HTTP prevede 8 possibili richieste, anche se solo 2 sono le più usate:

- GET: serve a un client per recuperare una risorsa dal server (come la richiesta di una pagina web). Eventuali parametri da inviare al server sono specificati nella query string dell'URL.
- POST: serve a un client per trasmettere informazioni ai server. La maggior parte dei browser web usa comunemente tale metodo per inviare i dati delle form ai server. I dati sono specificati nel corpo della richiesta.
- PUT: metodo diverso per eseguire un POST. Solitamente serve a un client per inviare file.
- HEAD, CONNECT, DELETE, OPTIONS e TRACE: altri metodi (vedere referenze per dettagli).

- Le risposte hanno sempre un codice (esito).
- I codici più importanti sono:
  - Serie 2xx (successo): azione è stata ricevuta con successo, compresa ed accettata. Esempio: 200 OK.
  - Serie 3xx (redirezione): client deve inoltrare ulteriori richieste. Esempio: 301 Moved Permanently, 302 Temporarily Moved o 303 See Other.

Client problemi

  - Serie 4xx Client Error: richiesta sintatticamente scorretta o non può essere soddisfatta. Esempio: 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 405 Method Not Allowed, 408 Request Timeout, ecc.

Server problemi

  - Serie 5xx Server Error: server non in grado di rispondere. Esempio: 500 Internal Server Error, 501 Not Implemented, 503 Service Unavailable, ecc.

- La versione attuale di HTTP è la 1.1.
- Si rimanda allo studente l'approfondimento del protocollo, soprattutto per la modalità GET e POST.
- Si consiglia di considerare fonti quali:
  - Riferimento del W3C: https://www.w3.org/Protocols/
  - Introduzione più discorsiva e abbastanza completa: http://tweb.ing.unipi.it/tia/http.pdf

Nota!

Per il resto del corso, si assume che lo studente sia in grado di capire le differenze d'uso tra GET e POST quando si usa il tag FORM di HTML.

- Web (application) framework è una collezione di librerie/moduli che permettono lo sviluppo di applicazioni web senza dover gestire gli aspetti di basso livello come i protocolli, thread, ecc.
- Web Server Gateway Interface (WSGI) è l'interfaccia standard per lo sviluppo di applicazioni web in Python: specifica come un server web e un'applicazione web devono interagire.
- Werkzeug (`http://werkzeug.pocoo.org`) è una libreria WSGI che rende disponibili le richieste/risposte HTTP come oggetti e altre funzioni per usare il protocollo HTTP. Rappresenta una possibile base per un web application framework.
- Jinja2 (`http://jinja.pocoo.org`) è un motore per modelli di presentazione (template engine) per Python. Un sistema di modelli web (template) permette di costruire documenti HTML o simili inserendo dati dentro a dei modelli (pagine web dinamiche).

- Flask (http://flask.pocoo.org) è un (micro) framework per web application in Python basato su Werkzeug e Jinja2.
- Flask permette lo sviluppo (abbastanza veloce) e l'esecuzione diretta di applicazioni web.
- Principali caratteristiche:
  - debugger e server di sviluppo integrato.
  - supporta i cookies sicuri.
  - conforme 100% a WSGI 1.0.
  - basato su Unicode.
  - facilmente estensibile (ci sono molte estensioni open source).

Nota!

In questa lezione si presentano solo le funzionalità principali di Flask: quelle necessarie per sviluppare una semplice applicazione web che accede a una base di dati in PostgreSQL.

Installazione Flask su Ubuntu 14.04 Desktop

- Flask può essere installato via pip3:

```
pip3 install --user flask
Collecting flask
  Downloading Flask-0.10.1.tar.gz (544kB)
Collecting Werkzeug>=0.7 (from flask)
  Downloading Werkzeug-0.11.9-py2.py3-none-any.whl (306kB)
Collecting Jinja2>=2.4 (from flask)
  Downloading Jinja2-2.8-py2.py3-none-any.whl (263kB)
Collecting itsdangerous>=0.21 (from flask)
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting MarkupSafe (from Jinja2>=2.4->flask)
  Downloading MarkupSafe-0.23.tar.gz
Installing collected packages: Werkzeug, MarkupSafe, Jinja2,
     itsdangerous, flask
...
Successfully installed Jinja2-2.8 MarkupSafe-0.23 Werkzeug-0.11.9
     flask-0.10.1 itsdangerous-0.24
```

- Flask rende disponibile la classe `Flask`.
- Un'oggetto di `Flask` rappresenta un'applicazione web (che non fa nulla).
- Un'applicazione web può essere personalizzata aggiungendo dei metodi mediante il meccanismo dei decorator.
- Flask mette anche a disposizione un web server interno che può essere attivato direttamente da un'applicazione web.

Nota!

Un decorator in Python è una funzione che altera la funzionalità di un'altra funzione/metodo/classe senza la necessità di creare/usare una sottoclasse.
In Python l'applicazione dei decorator è semplificata dalla notazione @.

Esempio di decorator applicato a un metodo

```
def myHtml(cap): #Decorator
    '''Racchiude il valore di cap tra <p></p>. Input è il metodo \
       da 'modificare'. Output è un metodo che rappresenta il \
       metodo modificato.'''
    def capMod(name):
        return "<p>{:s}</p>".format(cap(name))
    return capMod
```

Quando viene chiamato cap(nome) viene chiamato myHtml che contiene cap(nome) e dentro la funzione myHtml uso l'altra funzione capMod(name)
E ritorno a cap(capMod) che avrà il nome tra <p>

```
@myHtml  #cap viene decorata da html
def cap(nome):
    '''Ritorna nome con la prima lettera maiuscola'''
    return str(nome).capitalize()

print(cap('roberto'))
```

Nota!

Un decorator in Python è una funzione che altera la funzionalità di un'altra funzione/metodo/classe senza la necessità di creare/usare una sottoclasse.
In Python l'applicazione dei decorator è semplificata dalla notazione @.

Esempio di decorator applicato a un metodo

Output:

```
<p>Roberto</p>
```

Esempio di una semplice applicazione web: file Controller.py

```
1  '''Applicazione web iniziale'''
2
3  from flask import Flask
4
5  app = Flask(__name__) #app è un'applicazione web
6
7  @app.route('/') #helloWorld è associato all'url '/'
8  def helloWorld():
9      return 'Hello World!' #è testo, no html!
10
11 @app.route("/it") #ciaoMondo è associato all'url '/it'
12 def ciaoMondo():
13     return '''<html><head></head>
14 <body><p>Ciao <strong>mondo</strong>!</p>
15 </body></html>'''
16
17 if __name__ == '__main__':# se il modulo è invocato direttamente
18     app.run(debug=True) #attiva il web server con questa app
```

Variabile predefinita = nome file

Variabile name può assumere il valore __main__ oppure nome file dove la variabile è usata

Nome file quando è importato dentro un altro file

Direttamente eseguito dall'interprete se è __main__

Capisco se il file è eseguito oppure importato

App rappresenta la mia applicazione definita all'inizio

Analisi della semplice applicazione web

- L'istruzione `app = Flask(__name__)` crea l'applicazione (vuota) e chiede sempre un nome. `__name__` è una variabile definita dall'interprete. È il nome del modulo se il file è importato, o la stringa `'__main__'` se il file è direttamente eseguito.
- Il decorator `route(rule, options)` della classe Flask modifica l'applicazione in modo all'URL 'rule' venga associato il metodo subito sotto la specifica del decorator. 'Options' è una lista di parametri opzionali che permettono di specificare altre condizioni per l'associazione.
- Il metodo `run(host, port, debug, options)` di Flask esegue l'applicazione nel web server locale. I parametri sono opzionali. I valori di default sono: host=127.0.0.1 (0.0.0.0 permette di far raggiungere il server da Internet), port=5000, debug=false, options=None.

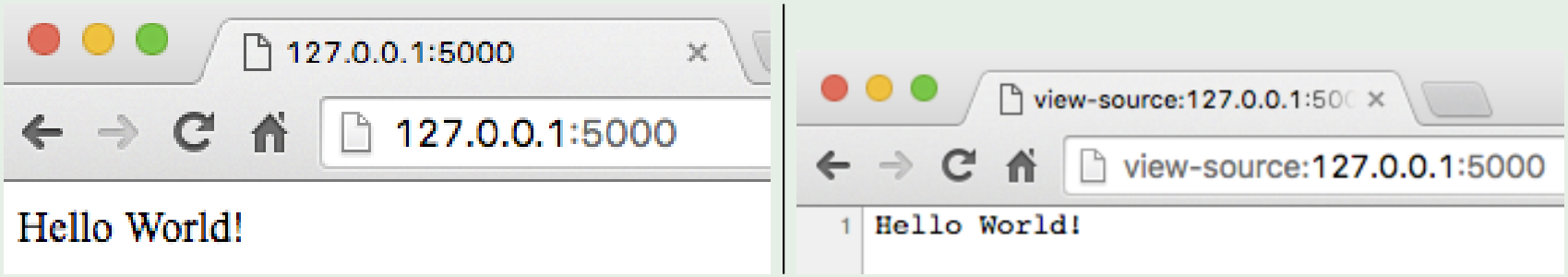
__ variabile privata della classe o modulo

Esecuzione della semplice applicazione web

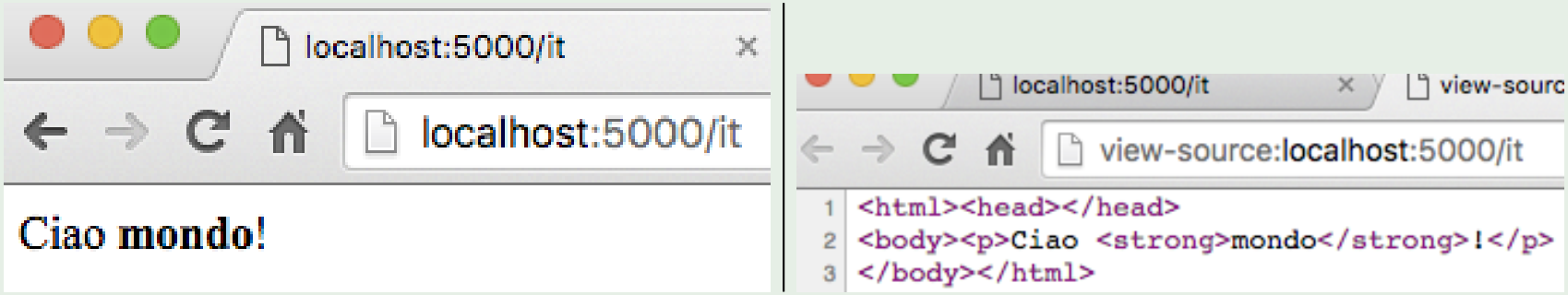
Il server web + app si attiva lanciando il comando

```
python3 Controller.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Debugger is active! ...
```

Si interagisce con l'applicazione aprendo un browser e inserendo URL dell'applicazione. Risultato e sorgente per URL '/':



Risultato e sorgente per URL '/it'



Esercitazione

- Installare Flask sul proprio pc di laboratorio
- Realizzare un'applicazione web in Flask che realizzi il seguente gioco:
  1. Durante l'inizializzazione, la applicazione sceglie una sequenza binaria casuale di lunghezza 3. 011
  2. Presenta all'utente una pagina con due bottoni, uno bianco (associato a True) e uno nero (associato a False) e le regole de gioco: l'utente deve indovinare la sequenza casuale premendo i bottoni avendo a disposizione 10 mosse.
  3. Ad ogni bottone premuto, l'applicazione restituisce un feedback: **'OK, continua'** o **'Sbagliato, Ritenta!'**
  4. Ad ogni errore, l'applicazione ri-inizializza la partita modificando la sequenza.
  5. Se l'utente indovina la sequenza entro 10 mosse, vince, altrimenti vince l'applicazione.