Python Biella Group

Flask Course

Docente

Andrea Guzzo

https://andreaguzzo.com

Anno 2020

Agenda

Incontri e lezioni

01 Introduzione a Flask e Jinja2 base

02 Jinja avanzato, Bootstrap, Forms

03 Flask con Database

04 Review con Andrea

05 Review con Mario

06 Grandi applicazioni con Flask

07 REST Backend e concetti avanzati

Python Biella Group

JOIN US!

- GitHub: *https://github.com/PythonGroupBiella*
- Telegram: *https://t.me/joinchat/AAAAAFGSWcxhSln_SRhseQ*

Tutto questo è stato reso possibile grazie a:

- Tutta la community di P.B.G.
- Maria Teresa Panunzio: https://www.linkedin.com/in/maria-teresa-panunzio-27ba3815/
- Mario Nardi: https://www.linkedin.com/in/mario-nardi-017705100/

Obiettivo del corso

Realizzare una piccola applicazione

- Concetti base di Flask per usarlo in differenti contesti
- Costruire piccole applicazioni web o backend
- Riuscire a "mettere in produzione" il proprio codice

Altri Framework:
Django, FastAPI, Pyramid, Bottle, ...

Cos'è Flask?

Micro-web framework

- Core semplice altamente estendibile e modulare
- Utilizzato per creare siti web e API
- Server side
- Leggero e performante
- Dipendenze:
  - Werkzeug: routing, debugger, WSGI support
  - Jinja2: templating

Features

- Request dispatcher
- Template engine
- Secure cookies
- User sessions
- Unit testing
- In-browser debugger e reloader

Moduli

- Administration
- Email
- Databases
- Caching
- User auth
- ...

Prima di iniziare...

Visual Studio Code: Estensioni

- Python
- GitLens
- GitHistory
- Jinja
- Jinja2 Snippets
- Kite Autocomplete for Python (?)
- Python Docstring
- Python Indent
- vscode-icons
- Visual Studio IntelliCode

Getting Started

- Python: 3.7
- *Repository Github:*

  https://github.com/PythonGroupBiella/MaterialeLezioni

```
#Install Python virtualenv
> pip install virtualenv

#Linux, MAC
> virtualenv venv
> source venv/bin/activate
> pip install -r requirements.txt

#Windows
> virtualenv venv
> venv\scripts\activate
> pip install -r requirements.txt
```

Nella precedente lezione...

Abbiamo visto:

- Impostazione dell'ambiente e strumenti
- Creazione di una semplice app
- Jinja Base
- Esempio di progetto

Per recuperare la lezione precedente:

- Video Youtube: https://www.youtube.com/watch?v=FPI5-oGKiVI
- *Repository Github:* https://github.com/PythonGroupBiella/MaterialeLezioni

Condividete le vostre soluzioni!

Python Biella Group

Lezione 2

Oggi vedremo...

- Costruzione di Forms
- Jinja Avanzato
- Bootstrap

Librerie e requisiti:

Repository Github

https://github.com/PythonGroupBiella/MaterialeLezioni

Nuove librerie

Flask Form

WTForms

Condividete le vostre soluzioni e fate gli esercizi

HTTP Methods

GET

Leggi (o recupera) una rappresentazione di una sorgente (pagina)

POST

Inserisci o crea una nuova risorsa

PUT (PATCH)

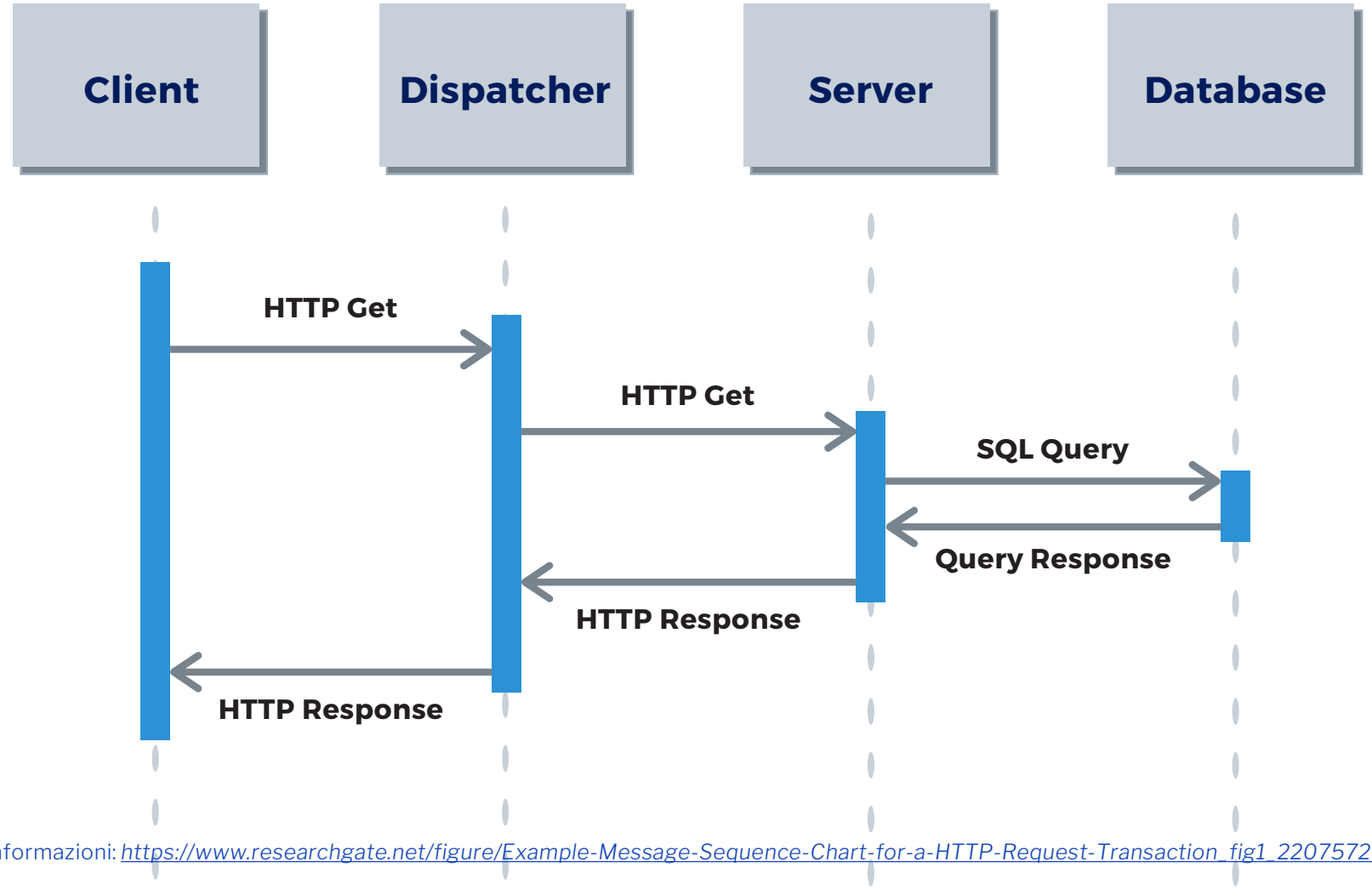
Modifica la rappresentazione corrente con un nuovo aggiornamento (eventualmente crea da zero)

DELETE

Elimina una risorsa esistente

Informazioni: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Sequenza dei messaggi



Informazioni: *https://www.researchgate.net/figure/Example-Message-Sequence-Chart-for-a-HTTP-Request-Transaction_fig1_220757202*

Bootstrap

Front end framework

Componenti

Grafica migliorata

Componenti animati

Grid System

Mobile Ready

Temi

References: *https://getbootstrap.com/*

Flask Bootstrap: *https://pythonhosted.org/Flask-Bootstrap/*

Bootstrap Italia: *https://italia.github.io/bootstrap-italia/*

Bootstrap Themes: *https://bootswatch.com/*

Lezione 3

Oggi vedremo...

- Database con Flask
- Migrazioni
- CRUD

Librerie e requisiti:

Repository Github

https://github.com/PythonGroupBiella/MaterialeLezioni

Nuove librerie

Flask SQLAlchemy

Flask Migrate

Condividete le vostre soluzioni e fate gli esercizi

SQL Light con Flask

Basic applications (100,000 hits /day)

ORM - Object Relational Mapper

CRUD Operations: Create, Read, Update, Delete

Migrate - sincronizzazione automatica

```python
#MacOS/Linux
export FLASK_APP=myapp.py

#Windows
set FLASK_APP=myapp.py
```

Python

Ci sono 4 comandi principali che si possono usare da command line:

```python
flask db init #set up the migrations directory
flask db migrate -m "some message" #set up the migration file (is always usefull ins
flask db upgrade #update the database with the migration
```

Python

SQL Alchemy: https://docs.sqlalchemy.org/en/13/

Flask SQLAlchemy: https://flask-sqlalchemy.palletsprojects.com/en/2.x/

Flask Migrate: https://flask-migrate.readthedocs.io/en/latest/

Python Biella Group

Lezione 6

Oggi vedremo...

- Blueprints - riorganizzazione per “Large applications”
- Gestire configurazioni
- Unittest
- Idee per le prossime evoluzioni

Repository Github

https://github.com/PythonGroupBiella/MaterialeLezioni/tree/master/Flask/Lezione6

Nuove librerie

Unittest

Condividete le vostre soluzioni e fate gli esercizi

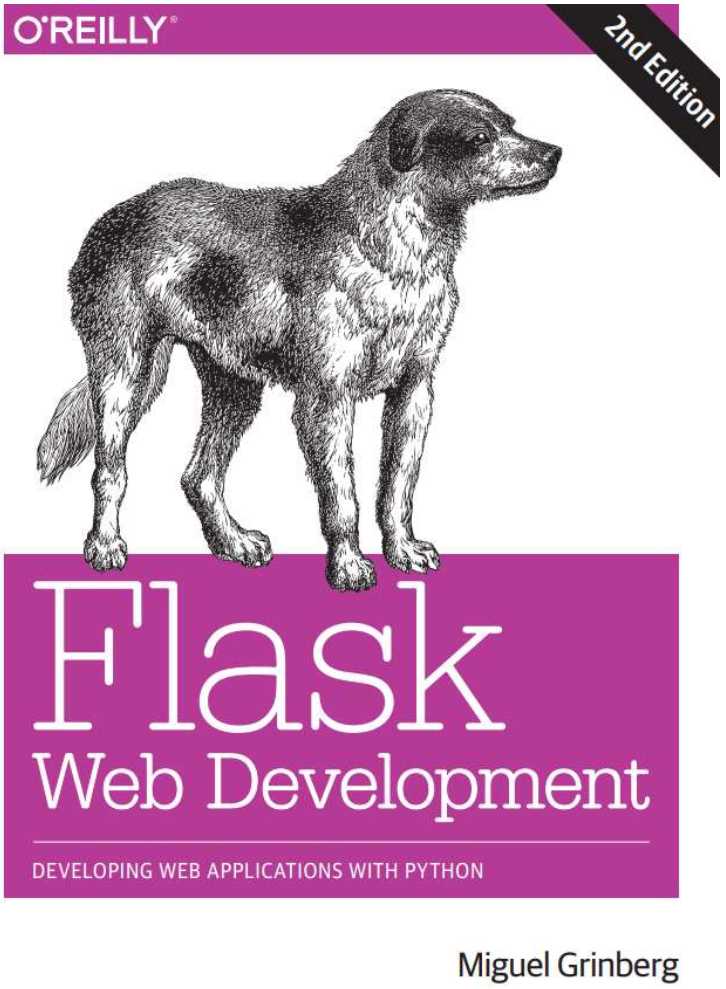
Libro di riferimento

Flask Web Development

Progetto di esempio

Flasky

https://github.com/miguelgrinberg/flasky



Riorganizzare per “Large applications”

Blueprints

In Flask, a blueprint is a method of extending an existing Flask app.

They provide a way of combining groups of views with common functionality and allow developers to break their app down into different components.

A blueprint is similar to an application in that it can also define routes and error handlers.

The difference is that when these are defined in a blueprint they are in a dormant state until the blueprint is registered with an application, at which point they become part of it.

Using a blueprint defined in the global scope, the routes and error handlers of the application can be defined in almost the same way as in the single-script application.

Like applications, blueprints can be defined all in a single file or can be created in a more structured way with multiple modules inside a package.

To allow for the greatest flexibility, a subpackage inside the application package can be created.

Nel libro: Cap. 7 - Large Application Structure

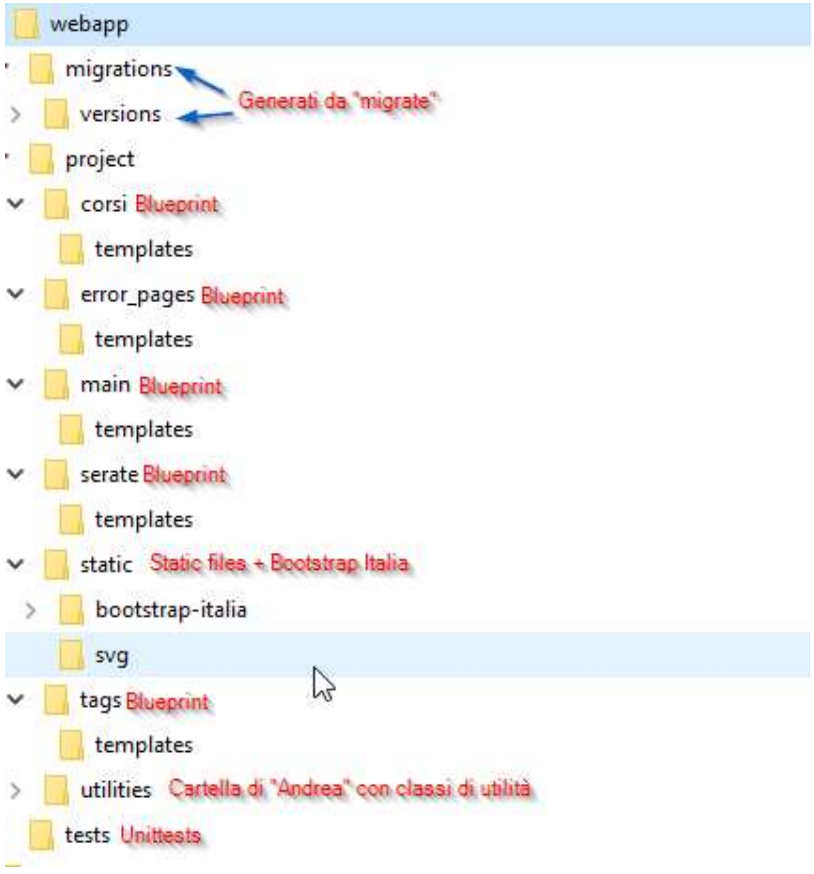
Riorganizzare per “Large applications”

Revisione della struttura

Non ci sono vincoli (al contrario di altri framework come Django)

Quindi non ci sono standard comuni

Nel ns progetto (vedi immagine)



Nel libro: Cap. 7 - Large Application Structure

Python Biella Group

Gestire configurazioni

Config file

create_app()

Application factory –

- Design pattern “creazionale”

- In software engineering, a design pattern is a general repeatable solution to a commonly occuring problem in software design
- It isn't a finished design that can be transformed directly into code, but a **description** or **template** for how to solve a problem that can be used in many different situations

- Design patterns:
  - Provide general solutions, documented in a format that doesn't require specifics tied to a particular problem
  - Can speed up the development process by providing tested, proven development paradigms
  - Help you benefit from the experience of fellow developers
  - Prevent subtle issues that can cause major problems
  - Improve code readability for coders and architects familiar with them

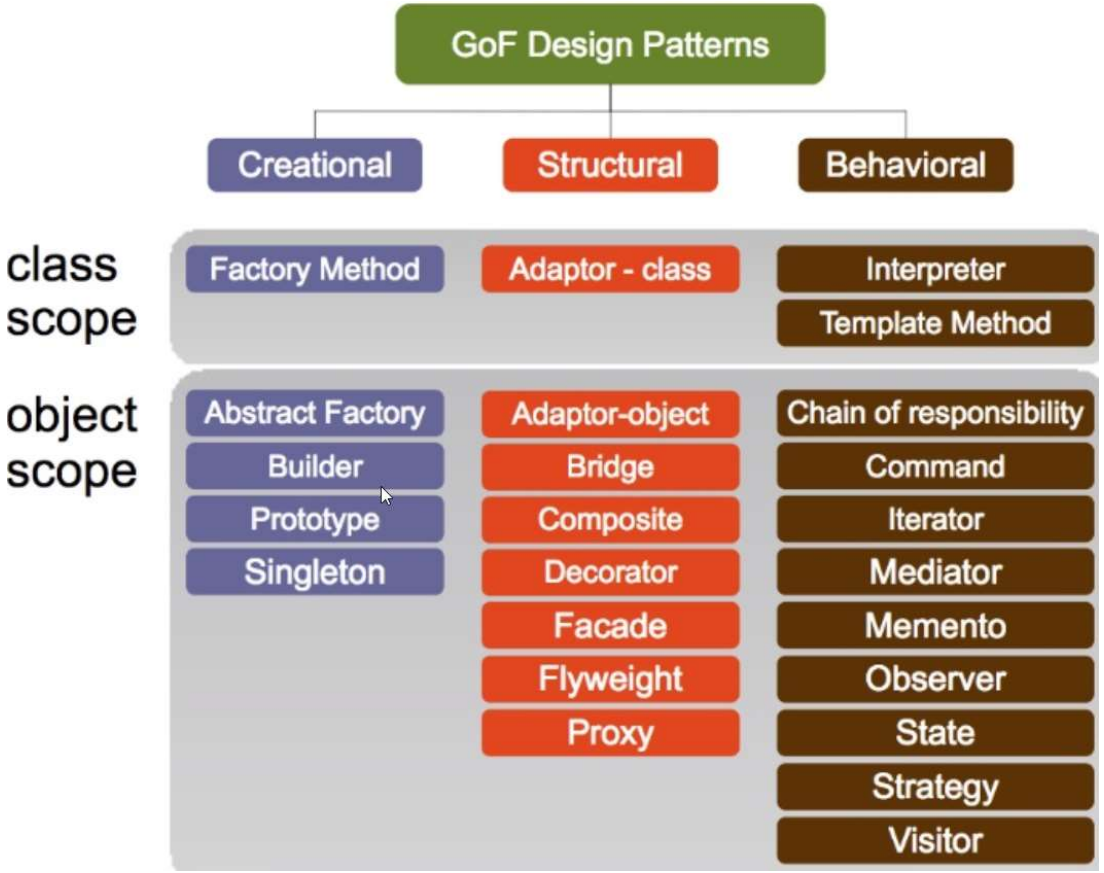
Nel libro: Cap. 7 - Large Application Structure

Design Patterns

GoF Design Patterns

Design Patterns: Definition and Utility

- The **Gang of Four** are the four authors of the book « Design Patterns: Elements of Reusable Object-Oriented Software »
- Defined 23 design patterns for recurrent design issues, called GoF design patterns
- Classified by *purpose*:
  - Structural : Concerns the **composition** of classes and objects
  - Behavioral : Characterizes the **interaction and responsibility** of objects and classes
  - Creational : Concerns the **creation process** of objects and classes
- ... and by *scope:*
  - Class scope: relationship between classes and subclasses, defined statically
  - Object scope: object relationships, dynamic



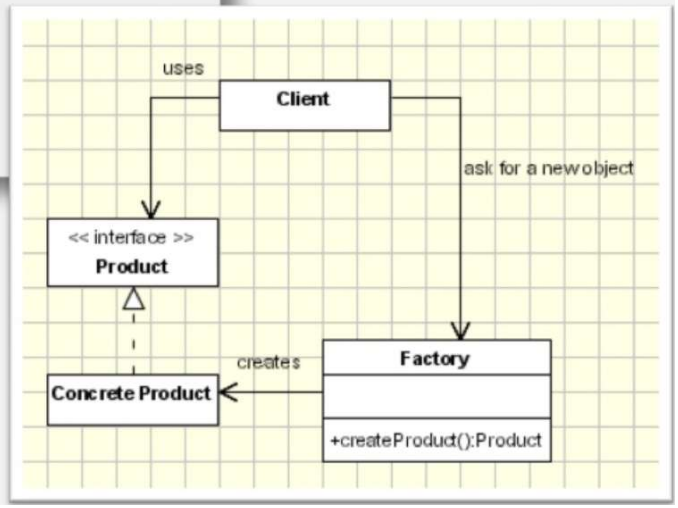
Factory Pattern

Factory

Creational Patterns

- Creates objects without exposing the instantiation logic to the client
- Refers to the newly created object through a common interface.

```
public class ProductFactory{
        public Product createProduct(String ProductID){
                if (id==ID1)
                        return new OneProduct();
                if (id==ID2) return
                        return new AnotherProduct();
                ... // so on for the other Ids

        return null;
    }
    ...
}
```



Factory Pattern - Esempio

Introduction

As per Wikipedia:

"The factory pattern is a creational design pattern used in software development to encapsulate the processes involved in the creation of objects."

Factory pattern involves creating a super class which provides an abstract interface to create objects of a particular type, but instead of taking a decision on which objects get created it defers this creation decision to its subclasses. To support this there is a creation class hierarchy for the objects which the factory class attempts create and return.

Factory pattern is used in cases when based on a "type" got as an input at run-time, the corresponding object has to be created. In such situations, implementing code based on Factory pattern can result in scalable and maintainable code i.e. to add a new type, one need not modify existing classes; it involves just addition of new subclasses that correspond to this new type.

In short, use Factory pattern when:

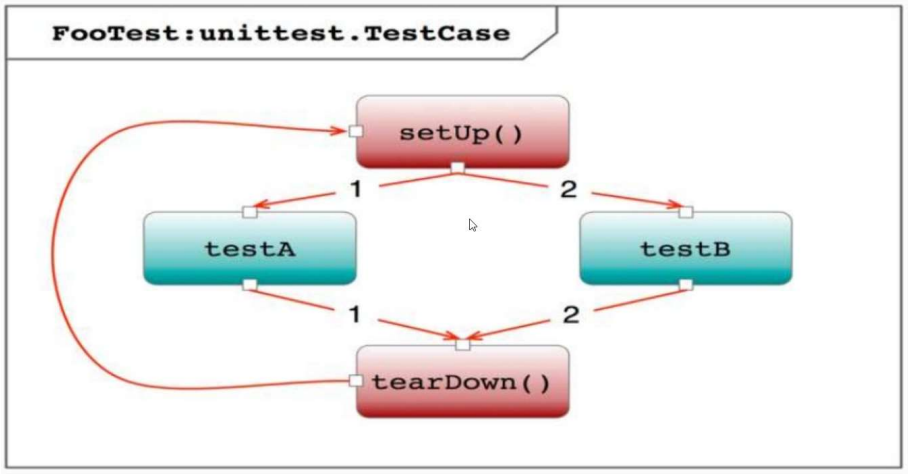
- A class does not know what kind of object it must create on a user's request
- You want to build an extensible association between this creater class and classes corresponding to objects that it is supposed to create.

```
class Person:
        def __init__(self):
                self.name = None
                self.gender = None

        def getName(self):
                return self.name

        def getGender(self):
                return self.gender

class Male(Person):
        def __init__(self, name):
                print "Hello Mr." + name

class Female(Person):
        def __init__(self, name):
                print "Hello Miss." + name

class Factory:
        def getPerson(self, name, gender):
                if gender == 'M':
                        return Male(name)
                if gender == 'F':
                        return Female(name)
```

Unit Test / 1

Some Important Points

- Every test class must be sub class of **`unittest.TestCase`**
- Every test function should start with **test** name.
- to check for an expected result use **assert** functions.
- The **setUp()** method define instructions that will be executed before test case.
- The **tearDown()** method define instructions that will be executed after test case.
- Run Test with **python -m unittest -v test_module**
- Only test single part of code



Unit Test / 2

Assert functions

- assertEqual(a, b)
- assertNotEqual(a, b)
- assertTrue(x)
- assertFalse(x)
- assertIs(a, b)
- https://docs.python.org/2/library/unittest.html#test-cases

Unit Test / 3

Integrazione con Flask

Decorator in app.py -> flask test

Package test

Nel libro: Cap. 7 - Large Application Structure

Prossime evoluzioni dell'app

Autenticazione e modulo login (nuovo blueprint)

Utenti non autenticati:

- Prossime serate
- Lista corsi
- Blog in lettura

Utenti autenticati

- Blog in scrittura

Utenti amministratori

- Gestione tag, serate, corsi

Gestione blog (nuovo blueprint)

INIZIAMO!

- GitHub: https://github.com/PythonGroupBiella/MaterialeLezioni