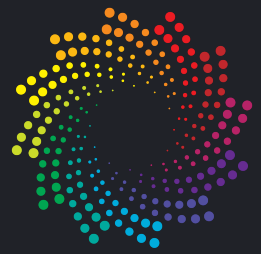


Flask: Introduzione e routes

Andrea Pescetti
andrea@datamira.com

Luglio 2024

Scopo del modulo

Presentare
i concetti base di Flask
e la gestione delle routes

Argomenti

- Il framework Flask
- Creare una webapp
- Gestire le routes

Introduzione a Flask

Introduzione a Flask

- Flask è un framework (un *micro-framework*, come lo si chiama per evidenziarne la leggerezza) per applicazioni web in Python.
- È leggero e semplice da usare, progettato per rendere facile e veloce lo sviluppo di applicazioni web.
- Flask è noto per la sua flessibilità e modularità e permette agli sviluppatori di scegliere i componenti che desiderano integrare nella loro applicazione.

Caratteristiche principali di Flask

- **Leggero**: Flask non include funzionalità di default non necessarie, permettendo di aggiungere solo ciò che è necessario.
- **Modulare**: È possibile aggiungere estensioni per funzionalità come database, autenticazione e così via; le estensioni tipicamente sono disponibili su PyPI (installazione via `pip`).
- **Flessibile**: Consente di personalizzare e configurare ogni aspetto dell'applicazione.
- **Facile da imparare**: Ha una curva di apprendimento dolce, che consente di essere produttivi da subito.

Versioni di Flask

- Flask è stato rilasciato per la prima volta (versione 0.1) nel 2010 ma solo in epoca relativamente recente ha raggiunto la maturità.
- Flask 1.0 è stato rilasciato nel 2019 con molte nuove funzionalità rispetto alle versioni 0.x, miglioramenti alla stabilità e maggiore supporto per le moderne pratiche di sviluppo web.
- Flask 2.0 è stato rilasciato nel 2021 con supporto per Python >= 3.6, miglioramenti a livello di chiamate asincrone e alla gestione delle sessioni.
- **Flask 3.x** è la nostra versione di riferimento. La versione 3.0 è stata rilasciata a settembre 2023 con importanti ristrutturazioni e la rimozione di funzioni deprecate.

Alternative a Flask

- **WSGI**, acronimo di *Web Server Gateway Interface*, è uno standard Python per le interfacce tra i server web e le applicazioni web. Tuttavia ne è consigliato l'uso diretto solo per applicazioni molto semplici o di test. Flask comunque è compatibile con WSGI: un'applicazione Flask in concreto è un'applicazione WSGI; in altre parole, Flask implementa il protocollo WSGI per comunicare con i server web.
- **Django** è un framework Python molto completo, sempre basato su WSGI, pensato per applicazioni complesse. Rispetto a Flask è più rigido (*opinionated*) e supporta applicazioni più strutturate; è anche più difficile da imparare.

Installare Flask: venv

- Prima di installare Flask è bene creare un ambiente apposito con il modulo `venv`; in questo esempio lo chiamiamo `progetto` e il comando creerà una cartella di nome `progetto`.

```
python -m venv progetto
```

- Poi attivare l'ambiente `progetto` con

```
progetto\Scripts\activate       # Su Windows
source progetto/bin/activate    # Su MacOS o Linux
```

  (al momento opportuno lo si potrà disattivare con `deactivate`)
- Impostarlo anche in VSCode scegliendo quel progetto nell'elenco degli interpreti Python in basso a destra.

Installare Flask: pip

- Una volta creato l'ambiente si può installare Flask, insieme alle sue numerose dipendenze, tramite `pip` .

```
pip install flask
# Varianti: python -m pip install flask
# oppure:   py -m pip install flask
```

- Per verificare l'installazione basta eseguire un semplice file Python che importa Flask e mostra la versione:

```
import flask
import importlib
print(importlib.metadata.version('flask'))
```

Applicazione Flask minimale

```
# File: ciao.py

# Importiamo la classe Flask.
from flask import Flask

# Costruiamo un oggetto, che è la nostra applicazione.
# Ci serve un nome: usiamo __name__ cioè il nome del file.
app = Flask(__name__)

# Questo decorator indica a quale URL risponderemo.
@app.route("/")
# Il nome della funzione è arbitrario.
def output():
  # Dobbiamo restituire una stringa in HTML.
  return "<h1>Ciao!</h1>"
```

Eseguire un'applicazione Flask

- Le applicazioni Flask si avviano dal terminale; è possibile, ma considerato *legacy*, anche avviare l'applicazione all'interno del file Python.
- Per eseguire `ciao.py`, nella cartella in cui si trova il file:

```
$ flask --app ciao run
# Oppure: python -m flask --app ciao run
# oppure: py -m flask --app ciao run
* Serving Flask app 'ciao' [...]
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

- Si può evitare di specificare ogni volta `--app ciao` in vari modi:

```
export set FLASK_APP=ciao       # Linux o MacOS
$env:FLASK_APP=ciao             # Windows PowerShell
set FLASK_APP=ciao              # Windows Prompt comandi
```

Accedere ad un'applicazione Flask

- Per accedere ad un'applicazione Flask basta aprire nel browser l'URL comunicato da Flask all'avvio dell'applicazione, tipicamente http://localhost:5000 o l'equivalente http://127.0.0.1:5000
- Se per caso la porta 5000 fosse già occupata si può specificare un'altra porta usando `--port` dopo `run`:

```
flask --app ciao run --port=5555
```

- L'applicazione per ragioni di sicurezza è accessibile solo dal sistema locale. Per lasciarla aperta a tutti (insicuro):

```
flask --app ciao run --host=0.0.0.0
```

Le routes (rotte) in Flask

Le routes in Flask

- Le routes (in italiano *rotte*) in Flask definiscono come l'applicazione web risponde a diverse richieste, per la precisione a diversi URL.
- Ogni route è associata a una funzione che viene eseguita quando viene richiesto un URL specifico.
- Dal punto di vista di Flask è molto più descrittiva la route rispetto al nome della funzione ad essa associata, che può essere cambiato senza particolari impatti sull'applicazione.

Definizione di una route

- L'aspetto essenziale nella definizione di una route è il *decorator* `@app.route` che precede la definizione della funzione.
- L'argomento di questo decorator è l'URL a cui risponderemo con la funzione definita immediatamente dopo il decorator.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return '<h1>Homepage</h1>'
```

Routes con parametri

- Le routes possono anche accettare parametri che vengono passati alle funzioni.

```
@app.route('/user/<username>')
def mostra_profilo(username):
  return f'<h1>Utente: {username}</h1>'
```

- In questo esempio, qualsiasi URL che rispetta il pattern `/user/<username>` (ad esempio `/user/andrea`) passerà il valore di `username` alla funzione `mostra_profilo`.

Routes con parametri e tipi

- Anche per motivi di sicurezza è bene imporre il tipo di un parametro quando lo si può conoscere in anticipo.

```
@app.route('/article/<int:article_id>')
def mostra_articolo(article_id):
  return f'<h1>Articolo numero: {article_id}</h1>'
```

- I tipi supportati sono: `string` (non `str`), `int`, `float` e altri.
- Se si tenta di accedere specificando un valore di tipo diverso, ad esempio `/article/abc`, Flask emetterà un errore di pagina non trovata (404).

Esercizio

- Creare una funzione `catalogo()` corrispondente a `/catalogo` che presenta come lista HTML un elenco di prodotti caratterizzati da codice, nome e prezzo.
- I prodotti sono (lista di liste o file CSV):

```
A111, Cappotto, 900
B222, Abito, 650
C333, Borsa, 1200
```

- Fornire anche un titolo e un sottotitolo per la pagina.

Routes e metodi HTTP

- Per impostazione predefinita, le routes rispondono al metodo GET.
- Tuttavia è possibile specificare altri metodi come POST, PUT, DELETE.

```
from Flask import request
@app.route('/login', methods=['GET', 'POST'])
def login():
  if request.method == 'POST':
    return '<em>Autenticazione in corso...</em>'
  else:
    return '<h1>Pagina di login</h1>'
```

- Qui `request` è un oggetto "globale" disponibile a ogni route; contiene ad esempio `request.method`, `request.args`, `request.data`, `request.form`, `request.headers`.

Passaggio di parametri

- Esistono due modi per passare i parametri ad una applicazione via web nel browser, e cioè i classici GET e POST. In questo momento ci occuperemo del caso GET.
- Entrambe le tipologie sono mappate da Flask nell'oggetto `request`, disponibile con `from flask import request`.
- L'oggetto predefinito `request` contiene tutte le informazioni recuperate dalla richiesta dell'utente, compresi eventuali parametri e dati inviati.

Parametri via GET con query

- Passaggio: via URL, cioè http://localhost:5000/somma?a=10&b=20
- I parametri saranno registrati nell'oggetto `args` di `request`
- Definizione route di esempio, per fare la somma di due numeri passati come `a` e `b` e forzati ad `int` dal nostro codice:

```
@app.route('/somma')
def somma_get_query():
  print(request.args)
  a = int(request.args.get('a'))
  b = int(request.args.get('b'))
  return f'Somma di {a} e {b}: {a+b}'
```

Parametri via GET con URL

- Passaggio: via URL, cioè http://localhost:5000/somma/10/20
- I parametri in questo caso non saranno registrati nell'oggetto `args` di `request`
- Definizione route di esempio, che in questo caso diventa una ordinaria route con parametri e tipi::

```
@app.route('/somma/<int:a>/<int:b>')
def somma_get_url(a, b):  # Devono essere specificati
  print(request.args)     # Non contiene a e b
  return f'Somma di {a} e {b}: {a+b}"
```

Esercizio

- Estendere la pagina catalogo in modo che ogni elemento punti ad una pagina che contiene tutti i dettagli di un articolo.
- Avremo due routes, una per il catalogo e una (parametrica, con il codice come parametro) per la pagina specifica di ciascun articolo.

Route statiche

- In Flask è possibile fornire direttamente dei file (tipicamente immagini) creando routes direttamente collegate al filesystem.
- Sono dette *routes statiche* e sfruttano la funzione predefinita `send_from_directory`:

```
from flask import send_from_directory
@app.route('/images/<path:path>')
def send_images(path):
  return send_from_directory('images', path)
```

- Quindi ad esempio un'immagine `logo.png` presente nella cartella `images` sarà disponibile all'URL `/images/logo.png`.

L'oggetto globale "g"

- A volte vogliamo che una route acceda a dati che non sono parte della richiesta. Flask fornisce un oggetto globale `g` a tale scopo.
- Per popolare `g` e per la precisione `g.prova` :

```
from flask import g
def get_prova():
  if 'prova' not in g:
    g.prova = 'Questa è una prova'
  return g.prova

@app.route('/valore-prova')
def show_prova():
  return f'<h1>Prova vale: {get_prova()}</h1>'
```

Esercizio

- Estendere la pagina catalogo in modo che nella pagina di dettaglio di ciascun articolo ci sia spazio per una foto e per la quantità residua.
- Nella pagina di dettaglio inserire un link a `/compra/<codice>` che permette di fare scendere di 1 la quantità disponibile e compare solo se ci sono ancora articoli in stock.

Routes multiple

- A volte vogliamo che la stessa funzione sia usata per URL diversi.
- In questo caso basta specificare più decorators:

```
@app.route("/")
@app.route("/home")
@app.route("/index")
def homepage():
    return "<h1>Homepage</h1>"
```

Redirect ed errori

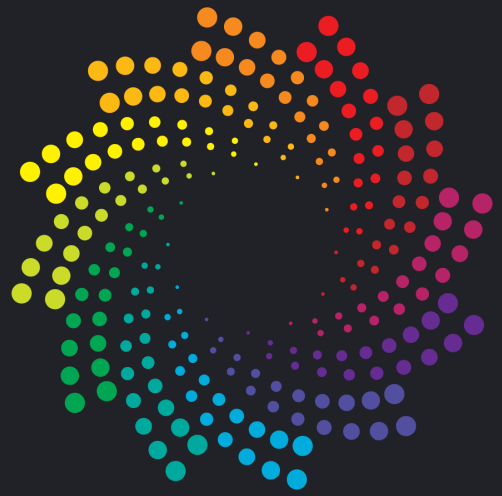
- La funzione `redirect` si usa per portare l'utente su un'altra pagina.
- La funzione `abort` invece invia un codice di errore.

```
from flask import redirect, abort
...
@app.route('/index')
def index():
  return redirect('/')
@app.route('/privato')
def privato():
  abort(401)
@app.errorhandler(401)
# Gli handler hanno un argomento implicito.
def unauthorized_error(error):
  return '<h1>401 Non autorizzato</h1>'
```

La funzione `url_for`

- La funzione `url_for()` costruisce un URL (assoluto) sulla base dei parametri forniti, e in particolare del primo parametro che è il nome della funzione.
- È utile per pagine di indice e per redirect, per assicurare maggiore stabilità.
- Esempi:

```
from flask import url_for
@app.route('/info')
def info():
  a = url_for('index')
  b = url_for('mostra_articolo', article_id=123)
  return f'{a} {b}'
```



DATAMIRA

Andrea Pescetti

andrea@datamira.com

(o via LinkedIn)