Python for Technologies #3

Standard Library e funzioni

Alvaro Gaiotti — alvaro.gaiotti@randstad.it

Indice

La Standard Library

Le funzioni *built-in* di Python sono molte, ma non coprono tutti i casi d'uso.

Per questo Python viene fornito con la propria Standard Library, una vasto numero di moduli che forniscono funzioni aggiuntive, che aiutano a gestire le normali attività che un programma deve svolgere come:

- Date e tempo
- Matematica avanzata
- Numeri casuali

NOTA

Python è organizzato in moduli: ogni file è un modulo, che possiamo importare con l'istruzione `import`:

```python
import this
from datetime import date
```

Ogni modulo contiene funzioni, variabili e classi che offrono funzionalità aggiuntive e ci aiutano a svolgere un compito specifico in maniera più comoda.

Quanto manca alle ferie?

Python

```python
from datetime import date # Importa tipo "date"

oggi = date.today() # L'operatore `.` 'accede' al modulo
natale = date(2024, 12, 25) # Formato: aaaa, mm, gg

print("Oggi: ", oggi)
# Possiamo sottrarre due date per ottenere una durata
print("Giorni a Natale: ", natale - oggi)
```

L’istruzione import

L’istruzione `import` permette di aggiungere modulo al vocabolario di Python (*global namespace*). Questa istruzione esegue il codice all’interno del modulo indicato, in modo da permetterci di utilizzarlo.

> **Suggerimento**
>
> Utilizzate la funzione `dir()` prima e dopo un `import` per vedere cosa cambia nell’elenco dei nomi definiti nel *global namespace*.

Ci sono varie modalità di import che possiamo usare.

Modalità di import

```python
import module
from module import function, class, variable, CONSTANT

# Wildcard import: aggiungi direttamente il contenuto del
# modulo al global namespace e non il modulo in sé.
from module import *
```

Quando utilizziamo la prima modalità, per accedere ai contenuti del modulo (ovvero al suo *namespace privato*) utilizziamo l'operatore `.`: `module.function()`, `module.class()`, `module.CONSTANT`

Funzioni

Python ci permette di definire nuove funzioni con l'istruzione `def`. L'idea è quella di dare un nome ad una porzione di codice, in modo da poterla riutilizzare comodamente più volte.

Una funzione riceve, generalmente, degli imput, esegue del codice utilizzando questi input e, infine, restituisce un output.

```python
def somma(a, b):
    print("Eseguo la funzione somma...")
    return a + b
```

La definizione di una funzione si compone di due elementi:

- Intestazione
  - Istruzione `def`
  - Nome della funzione
  - Lista degli argomenti o parametri tra parentesi
- Corpo della funzione con opzionale istruzione `return`

Ogni volta che una funzione è *chiamata* (es. `print("Hello")`), viene eseguito il corpo della funzione fino al primo `return` incontrato, o fino al proprio termine, nel qual caso il valore di ritorno è `None`.

La nostra prima libreria

Esercizio

Create un file nominato `primo.py`, dove definire una funzione chiamata `is_prime` che riceva un numero e ritorni `True` se esso è primo e `False` se non lo è.

Un numero primo è un numero maggiore di 2 divisibile solo per 1 e per sé stesso, quindi la vostra funzione dovrà provare ad individuare un divisore e, nel caso in cui fallisca, ritornare `True`

La nostra prima libreria

Esercizio

Create il file `gemelli.py`: un programma per trovare tutti i primi gemelli inferiori a `100`.

Due primi p e q sono gemelli se e solo se $p - q = 2$.
es. (5, 3), (13, 11).

Importate il modulo `primo` (senza estensione!) e utilizzata la sua funzione `is_prime` per aiutarvi.

Classi, proprietà e metodi (di nuovo!)

Python supporta il paradigma ad oggetti utilizzando le keyword `class` e `self`. La prima inizia la dichiarazione di una classe (il famoso template), la seconda è utilizzata per riferirsi all'istanza concreta della classe (*oggetto*) all'interno dei metodi, che sono normali funzioni dichiarate all'interno di una classe.

```python
class Utente():
    def __init__(self, name, password): # Costruttore
        self.name = name # Proprietà
        self.password = password # Proprietà
        self.su = False # Proprietà

    def name(self): # Metodo
        return self.name


```

```
11  class Admin(Utente): # Ereditarietà
12      def __init__(self, name, password):
13          super().__init__(name, password)
14          self.su = True
```

Un po’ di tooling

Facciamo una carrellata con le principali tecnologie afferenti all’ecosistema Python che possiamo incontrare!

Pandas

Pandas è una libreria più utilizzata nel settore Data Analytics/Data Science.

Il suo componente principale è la classe `DataFrame`: una classe simile ad una tabella di un database relazionale e che implementa metodi atti alla manipolazione semplice e performante di righe e colonne, al fine di ottenere dati di qualità sui quali lavorare.

Ideale anche per effettuare analisi statistiche, fornisce un backend per splendide visualizzazioni.

Libreria nodale per ogni attività *data-related* in Python.

NumPy

Numpy è una libreria dedicata al calcolo scentifico (circa).

Con un backend scritto in `C` per massime performance, NumPy semplifica e rende efficienti calcoli e manipolazioni su matrici e array multidimensionali.

Offre, inoltre, una serie di funzioni matematiche di alto livello (trasposizioni, identità, ecc.) fondamentali per operare efficacemente su molteplici dati.

Accelera gran parte dell'ecosistema ML e DL di Python.

Scikit-learn

Scikit-learn è LA libreria per il Machine Learning classico in Python (no DL).

Offre una vasta gamma di modelli per clustering, regressione e classificazione.

Implementa anche tutta una serie di funzioni e classi ideali per la validazione di modelli, riduzione di dimensionalità, preprocessing e tanto altro.

PyTorch

Inizialmente sviluppata da Meta, PyTorch è una libreria focalizzata su algoritmi di Deep Learning.

Offre la possibilità di allenare modelli adatti ad entrare in produzione in modo parallelo, distribuito e rapido.

Risulta particolarmente adatta a task legate al NLP e alla Computer Vision.

Progetto notevole: Autopilot di Tesla.

TensorFlow

Sviluppata da Google, TensorFlow è la libreria rivale di PyTorch.

Le due librerie offrono grossomodo le stesse funzionalità, anche se TensorFlow è spesso usato per task di inferenza.

Google utilizza TensorFlow per mostrarvi risultati di ricerca più rilevanti per voi, inferendo da diverse informazioni (es. che ricerche avete effettuato poco fa o estraendo informazioni da come avete formulato la *query*)

Keras

Keras è una libreria di alto livello per il Deep Learning, che espone *astrazioni* più generali rispetto ad altre librerie.

Può usare diversi backend, tra cui TensorFlow.

Più semplice da utilizzare rispetto alle due precedenti, è utile per *POC* o prototipi, in quanto è molto generale e dichiarativa (cosa fare, non come).

Keras stesso si definisce un'*interfaccia* e non una libreria proprio per questo motivo.

Django

Il più famoso framework Python per lo sviluppo web, Django offre un ecosistema completo agli sviluppatori, permettendo la creazione di API, siti web, webapp, e-commerce e tanto altro.

Integra un ORM di default (non si scrive SQL!) e ha una forte community che sviluppa integrazioni, middleware e altre estensioni che lo rendono completo sotto tutti i punti di vista.

Per la parte frontend utilizza Jinja, un *templating engine* per utilizzare Python in file HTML.

Difetti: è lento (colpa di Python)

FastAPI, Flask, Bottle

Altre librerie legate allo sviluppo web non fullstack: non offrono, infatti, supporto per la parte frontend (vi dovete arrangiare).

Sono utili per la veloce creazione di API o backend su cui innestare frontend esterni.

Semplici da usare e ottime per rapide prototipazioni (potete scrivere un'API in probabilmente 25-30 righe di codice), sono sicuramente meno estese rispetto all'ecosistema Django.