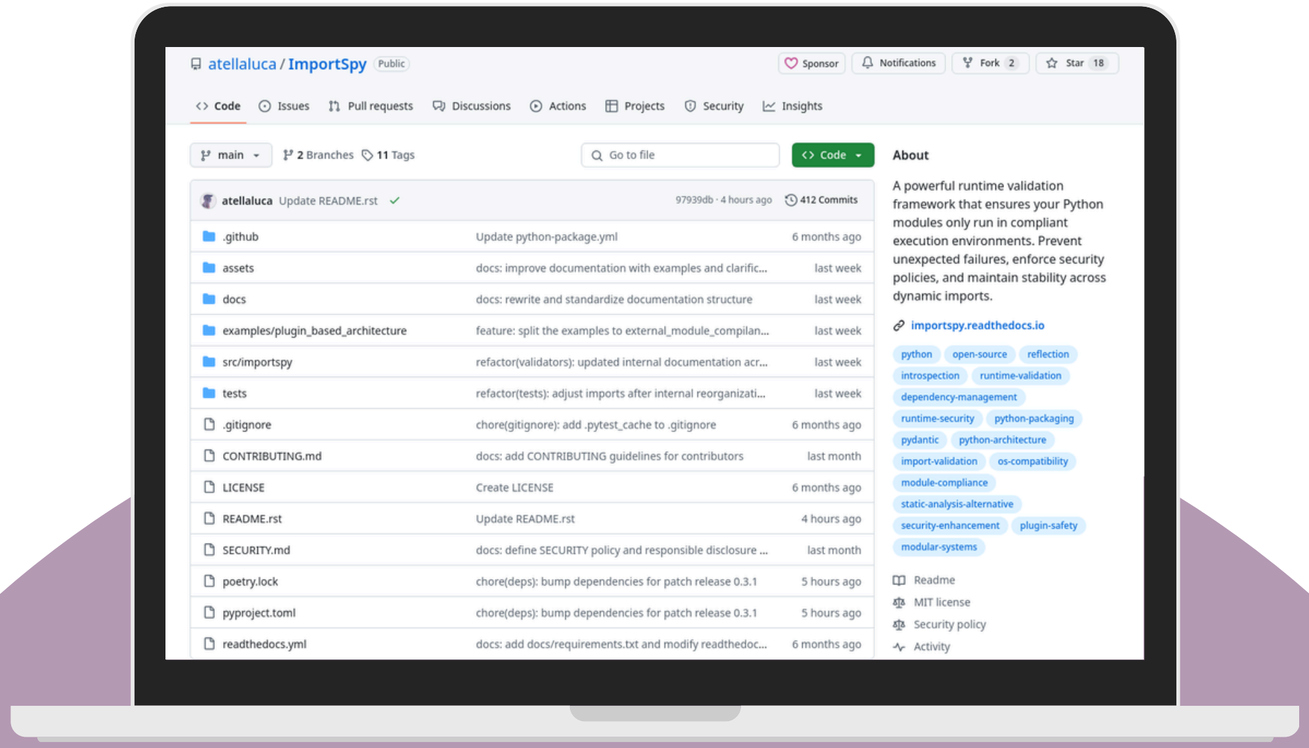
20 Maggio 2025
Flash Talks Arena
GDG Basilicata

ImportSpy

Quando i moduli Python sanno dire “no”

Luca Atella




```
pip install importspy
```

Il problema

ImportSpy

Nelle architetture modulari, plugin-based o dipendenti da un framework si introducono una serie di **vincoli sia nella struttura del codice sorgente che nell'ambiente di runtime**.

- Architettura della CPU (arm64, amd64, ...)
- Sistema operativo utilizzato (Linux based, Windows, Darwin)
- Variabili di ambiente
- Struttura dei moduli (variabili, funzioni, classi, oggetti, metodi, ...)
- Ambiente di runtime (JRE, versione dell'interprete, implementazione di Python...)

Spesso diamo per scontata la presenza di certi vincoli — anche per ragioni legittime, come il rispetto dei principi DRY o SOLID. Il problema è che così facendo il codice perde la capacità di verificare davvero se l'ambiente in cui viene eseguito rispetta quelle condizioni. E quando il contesto non è conforme, gli errori arrivano tardi o in modo poco chiaro.

Contesto tecnico e umano

Lavorando in team **ogni problema ha almeno due volti**: uno tecnico e l'altro umano.

Difficilmente si risolve un aspetto tecnico se non curi anche quello umano.

Lavorando come Software Architect in una PMI locale su un'architettura plugin-based molto modulare, mi sono accorto che più mi rendevo disponibile a supportare il team, più mi arrivavano problemi "strani" da gestire che erano quasi sempre causati da moduli che non rispettavano i vincoli del framework da cui dipendevano o della struttura dei plugin.

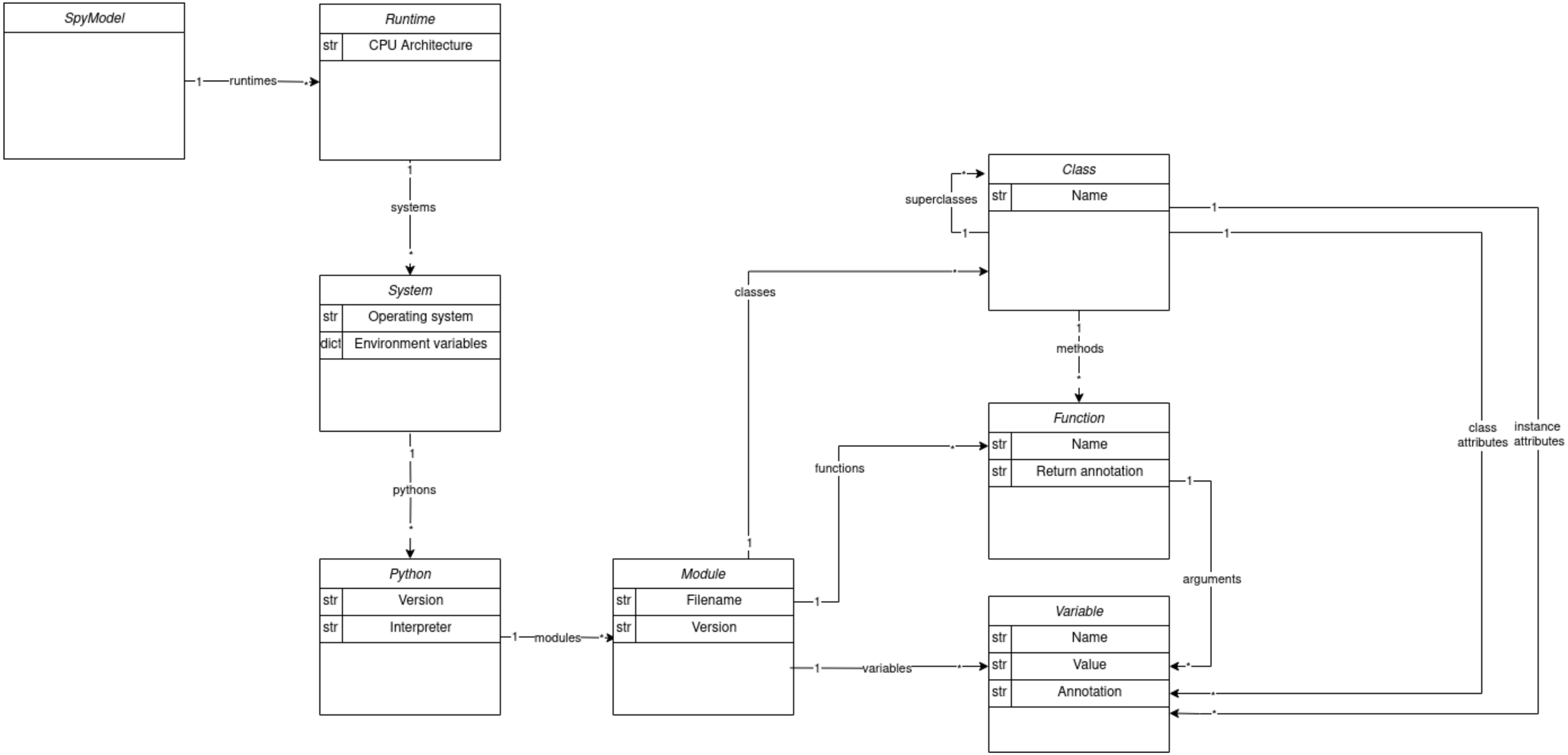
All'apparenza sembravano bug complessi, ma in realtà erano solo ben nascosti.
E col tempo, risolverli era diventato più noioso che stimolante...

Quando diventi il punto di riferimento tecnico in un team,
a volte i colleghi smettono di leggere il loro codice e ti chiedono direttamente di interpretarlo.
Così, ho imparato a dire di no.

E a un certo punto ho pensato:
magari posso insegnarlo anche ai moduli.

La mia risposta: ImportSpy

ImportSpy



La mia risposta: ImportSpy

ImportSpy

Un'architettura che **astrae l'ambiente di runtime** tipico di un applicazione Python.

- Architettura della CPU (arm64, amd64, ...)
- Sistema operativo utilizzato (Linux based, Windows, Darwin)
- Variabili di ambiente
- Struttura dei moduli (variabili, funzioni, classi, oggetti, metodi, ...)
- Ambiente di runtime (versione dell'interprete, implementazione di Python...)

Le classi coinvolte nell'astrazione sono costruite intorno a Pydantic un framework pensato per la validazione rigorosa dei dati attraverso la tipizzazione statica introdotta da MyPy e formalizzata nei PEP 484 e successivi.

L'architettura prende forma a runtime - lo SpyModel viene costruito dall'import contract specifico del modulo da validare: un file YAML conforme alla rappresentazione vista.

```python
class Logger:

    def log_event(self, data: dict) -> bool:
        """
        Logs an event to the internal logging system.

        Args:
            data (dict): Event data to log.

        Returns:
            bool: True if the log was successful, False otherwise.
        """
        try:
            # Example logic (this can be replaced with actual logging)
            print(f"[LOG] Event received: {data}")
            return True
        except Exception as e:
            print(f"[ERROR] Failed to log event: {e}")
            return False
```

```yaml
filename: analytics.py
deployments:
  - arch: x86_64
    systems:
      - os: linux
        pythons:
          - version: 3.11
            interpreter: CPython
            modules:
                classes:
                  - name: Logger
                    methods:
                      - name: log_event
                        arguments:
                          - name: self
                          - name: data
                            annotation: dict
                        return_annotation: bool
```

Applicare i contratti

ImportSpy

ImportSpy applica i contratti lungo due dimensioni principali.

La consapevolezza del contesto

Si preoccupa di controllare dove, quando e in quali condizioni viene eseguito il codice.

La consapevolezza strutturale

Si preoccupa di controllare la struttura del codice del modulo: variabili, funzioni, classi, metodi, attributi, superclassi, attributi di classe e attributi di istanza.

Il valore del confronto

ImportSpy costruisce due istanze della classe SpyModel:

- Una per rappresentare il contratto
- Un'altra per rappresentare l'ambiente di runtime attuale

Il confronto avviene poi tramite la logica del sottoinsieme che stabilisce che il modulo è considerato compilance se le informazioni contenute nel contratto rappresentano il limite inferiore del secondo SpyModel.

NOTA: vengono validati staticamente anche l'annotazione sul tipo di ritorno delle funzioni e dei metodi e sulle dichiarazioni di variabili e attributi di classe e di istanza.

Due modalità operative

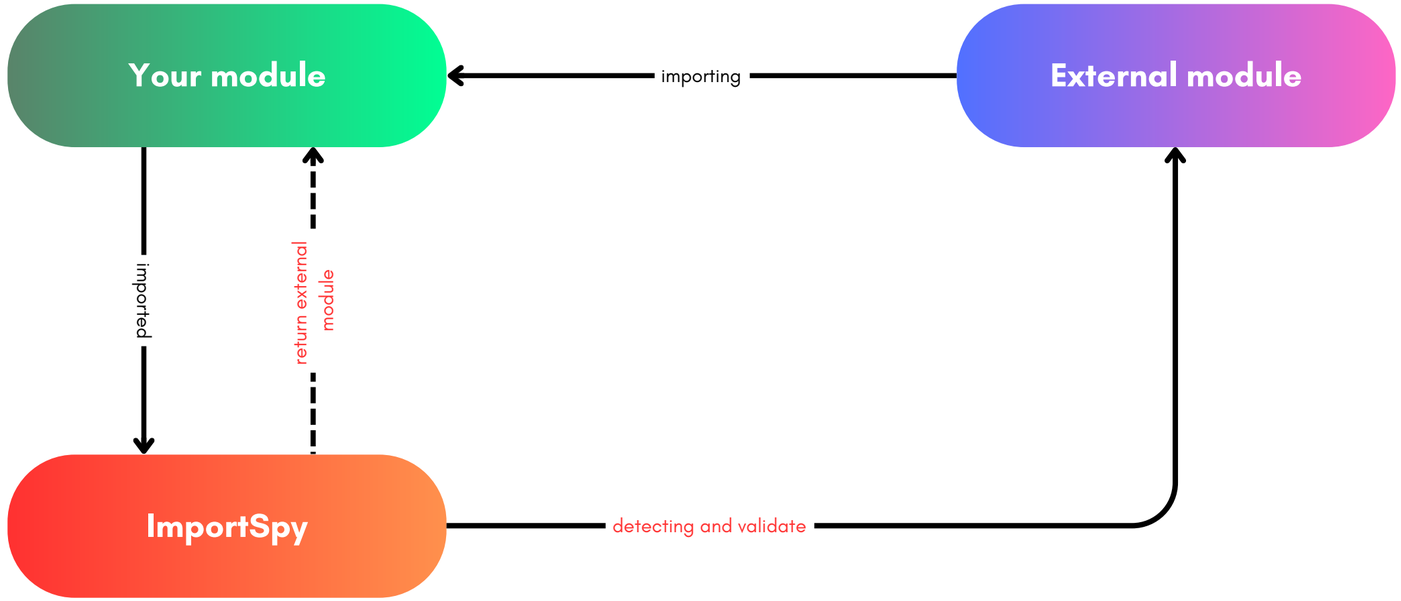
ImportSpy

Embedded mode

Il modulo protegge se stesso:
valida il contesto di esecuzione e la struttura di chi lo importa e blocca l'import con un messaggio di errore se il contratto non è rispettato.

```python
from importspy import Spy

Spy().importspy(filepath="contract.yml")
```



Due modalità operative

ImportSpy

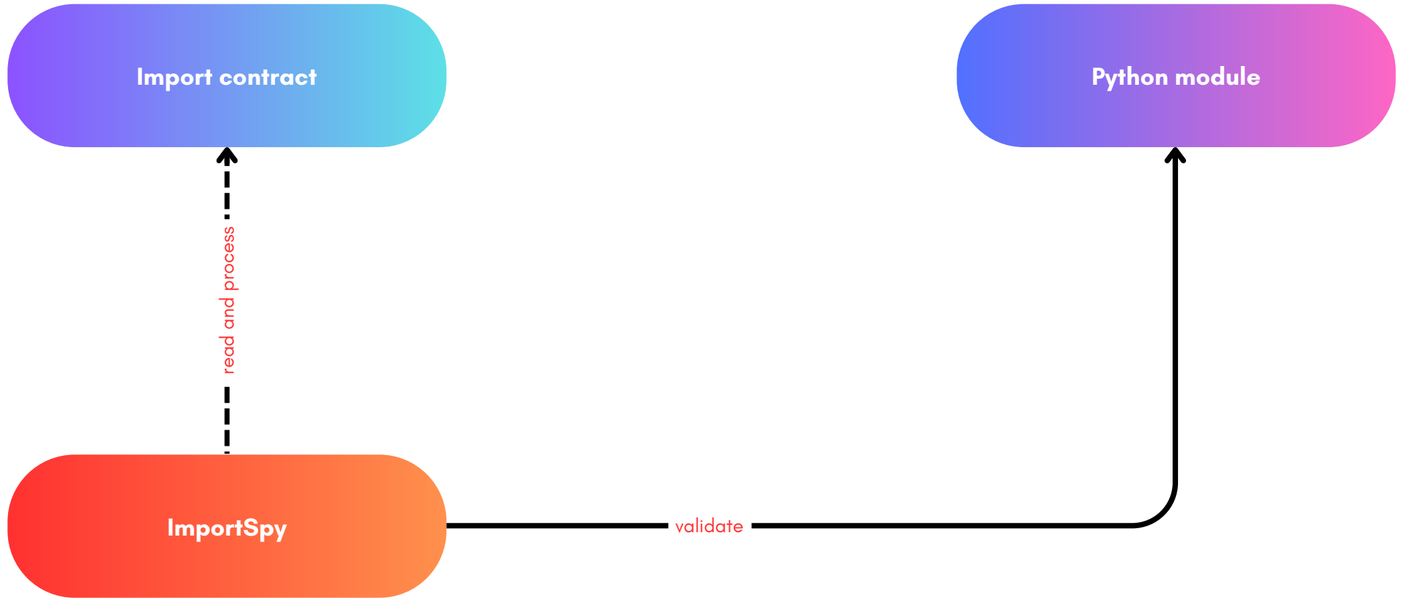
CLI mode

Valida un modulo dall'esterno, senza modificarlo: ideale per flussi CI/CD, adatta per controlli strutturali prima del deploy.

```
Usage: importspy [OPTIONS] [MODULEPATH] COMMAND [ARGS]...

 Arguments
   modulepath      [MODULEPATH]  Path to the Python module to load. [default: <class 'str'>]

 Options
 --version              -v                                  Show the version and exit.
 --spymodel             -s      TEXT                        Path to the SpyModel YAML file.
                                                            [default: spymodel.yml]
 --log-level            -l      [DEBUG|INFO|WARNING|ERROR]  Log level for output verbosity.
                                                            [default: None]
 --install-completion                                       Install completion for the current shell.
 --show-completion                                          Show completion for the current shell, to
                                                            copy it or customize the installation.
 --help                                                     Show this message and exit.
```



I contratti falliscono

ImportSpy

Quando il modulo non rispetta il contratto dichiarato, ImportSpy blocca l'esecuzione e solleva un errore chiaro e specifico che spiega esattamente quale vincolo è stato violato.

Error Type	Description
Missing Elements	A required **function**, **class**, **method**, or **attribute** is not found in the module or structure defined in the import contract.
Type Mismatch	A return annotation, argument type, or class attribute type does **not match** the one declared in the contract.
Value Mismatch	A variable or attribute exists but has a **different value** than expected (e.g., metadata mismatch).
Function Argument Mismatch	A function's arguments do **not match in name, annotation, or default values**.
Function Return Type Mismatch	The return type annotation of a function differs from the contract.
Class Missing	A required class is **absent** from the module.
Class Attribute Missing	One or more declared **class or instance attributes** are missing.
Class Attribute Type Mismatch	A class attribute exists, but its **type or annotation** differs from what is expected.
Superclass Mismatch	A class does not inherit from one or more required **superclasses** as declared.
Variable Missing	A required **top-level variable** (e.g., *plugin_name*) is not defined in the module.
Superclass Mismatch	A class does not inherit from one or more required **superclasses** as declared.
Variable Missing	A required **top-level variable** (e.g., *plugin_name*) is not defined in the module.
Variable Value Mismatch	A variable exists but its value does not match the one declared in the contract.
Filename Mismatch	The actual filename of the module differs from the one declared in *filename*.
Version Mismatch	The module's *__version__* (if defined) differs from the expected version.
Unsupported Operating System	The current OS is **not included** in the allowed platforms (e.g., Linux, Windows, macOS).
Missing Required Runtime	A required **architecture, OS, or interpreter version** is not satisfied.
Unsupported Python Interpreter	The current interpreter (e.g., CPython, PyPy, IronPython) is not supported by the contract.
Missing Environment Variable	A declared environment variable is **not present** in the current context.
Invalid Environment Variable	An environment variable exists but contains an **unexpected value**.

Casi d’uso

ImportSpy

Quando il contesto conta, ImportSpy protegge.

- Plugin system complessi
- Toolchain DevSecOps
- Sistemi soggetti a conformità normativa (finance, healthcare)
- Embedded / IoT
- Formazione e didattica sul design modulare
- Librerie open-source

Il cerchio si chiude.

ImportSpy

Adesso che il codice sa dove può girare e dove non deve
tu puoi regalarmi una stella sul repository di GitHub
e condividere ImportSpy con chi vuoi.

atellaluca

E se ti fa piacere, richiedi la tua card preferita 😀