Introduzione alle API REST in Python

Benvenuti a questo corso introduttivo sulle API REST e il formato JSON in Python! Durante le prossime lezioni, esploreremo come le API REST rappresentino uno standard fondamentale per lo sviluppo di applicazioni web moderne.

Impareremo i concetti base, i metodi HTTP, i codici di stato e come Python ci permette di interagire facilmente con queste interfacce. Questo percorso vi fornirà le competenze essenziali per integrare servizi esterni nelle vostre applicazioni e creare i vostri endpoint REST.

Cos'è un'API REST?

Interfaccia Programmabile

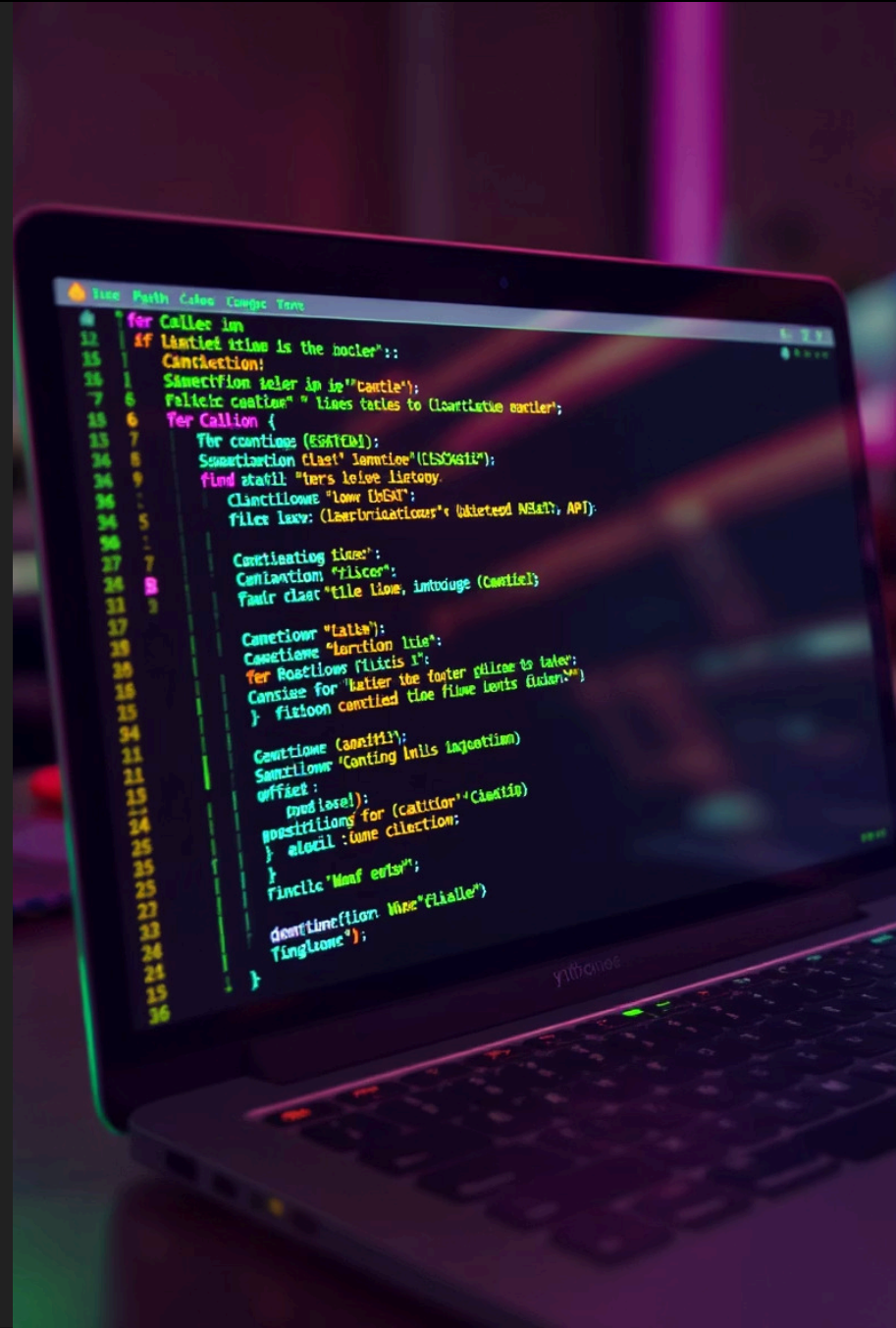
Un'API (Application Programming Interface) è un insieme di regole che permettono a diversi software di comunicare tra loro.

Rappresentazione dei Dati

REST (Representational State Transfer) è uno stile architetturale che definisce come le risorse vengono rappresentate e trasferite attraverso il web.

Comunicazione Standard

Le API REST utilizzano HTTP come protocollo di comunicazione, rendendo possibile l'interazione tra sistemi diversi indipendentemente dal linguaggio di programmazione.



Il Concetto di Stateless

Nessuna Memoria di Sessione

Nelle API REST, ogni richiesta dal client al server deve contenere tutte le informazioni necessarie per comprendere e processare la richiesta. Il server non memorizza lo stato delle comunicazioni precedenti.

Vantaggi della Statelessness

Questa caratteristica permette alle API REST di essere altamente scalabili: i server possono gestire richieste da qualsiasi client in qualsiasi momento senza dover conservare informazioni sulle sessioni precedenti.

Implementazione in Python

In Python, framework come Flask e Django REST framework facilitano la creazione di API stateless, gestendo automaticamente l'elaborazione delle richieste in modo indipendente.

Struttura di un Endpoint REST

URL Base

Il dominio principale dell'API, ad esempio: https://api.esempio.com

Percorso della Risorsa

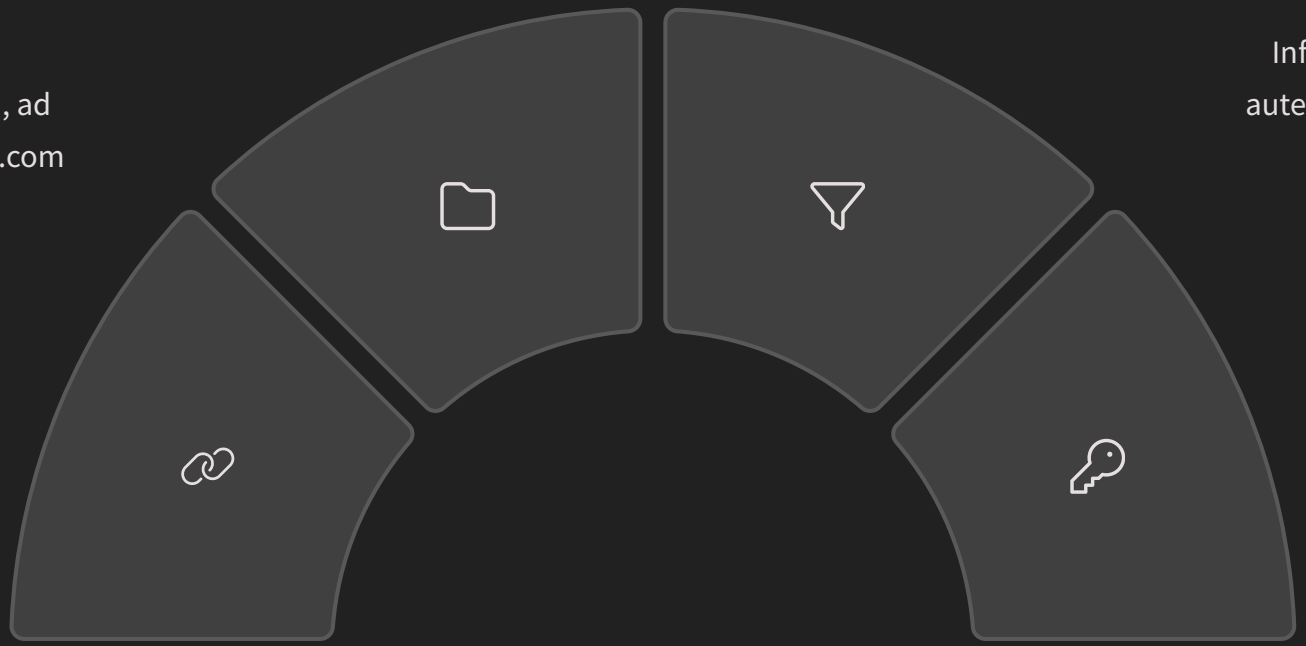
Identifica la collezione o l'oggetto specifico, come /utenti o /prodotti/123

Query Parameters

Parametri opzionali per filtrare, ordinare o paginare i risultati: ?page=2&sort=nome

Headers

Informazioni aggiuntive come autenticazione, formato dei dati e altre metainformazioni



Il Metodo HTTP GET

Caratteristiche del GET

Il metodo GET è utilizzato per recuperare informazioni dal server senza modificare lo stato delle risorse. È il metodo HTTP più comune e viene utilizzato ogni volta che visualizziamo una pagina web o richiediamo dati da un'API.

Le richieste GET sono:

- Idempotenti: ripetere la stessa richiesta non modifica il risultato
- Cacheable: i risultati possono essere memorizzati nella cache
- Visibili nell'URL: i parametri sono inclusi nell'URL

Esempio in Python

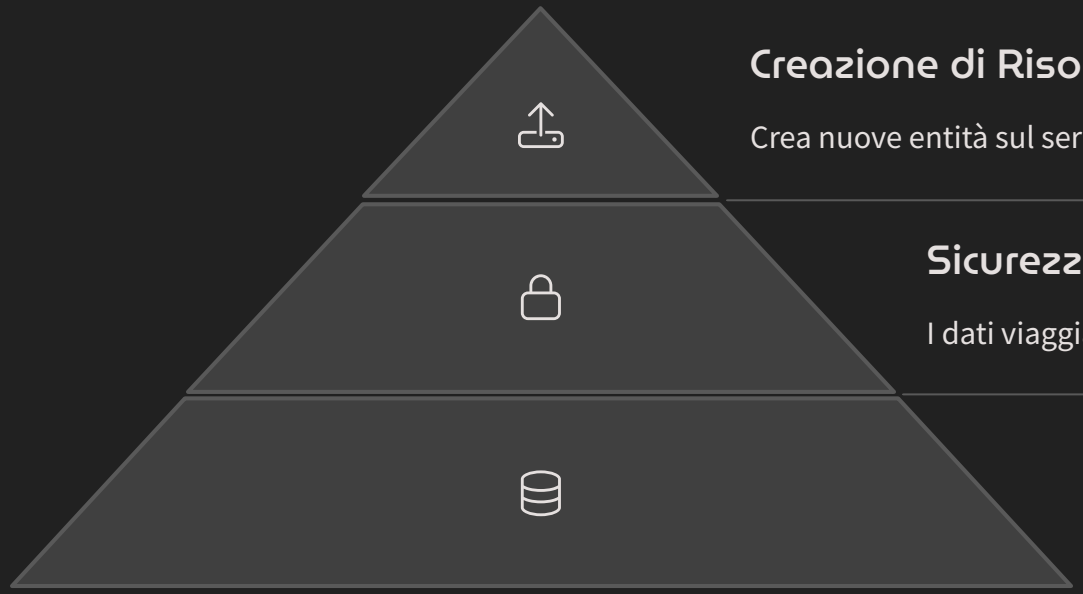
Ecco come effettuare una richiesta GET utilizzando la libreria requests di Python:

```
import requests

# Effettuare una richiesta GET
response = requests.get(
    'https://api.esempio.com/utenti',
    params={'attivi': 'true'}
)

# Verificare lo stato della risposta
if response.status_code == 200:
    utenti = response.json()
    print(utenti)
```

Il Metodo HTTP POST



Creazione di Risorse

Crea nuove entità sul server

Sicurezza

I dati viaggiano nel corpo della richiesta, non nell'URL

Stato del Server

Modifica lo stato delle risorse sul server

Il metodo POST viene utilizzato quando dobbiamo creare una nuova risorsa sul server. A differenza del GET, il POST trasmette i dati nel corpo della richiesta, rendendolo adatto per l'invio di informazioni sensibili o di grandi dimensioni. In Python, possiamo facilmente implementare richieste POST usando la libreria requests, specificando i dati da inviare nel parametro json o data.

Il Metodo HTTP PUT

Aggiornamento Completo

PUT sostituisce interamente una risorsa esistente con i nuovi dati forniti

Idempotenza

Eseguire più volte la stessa richiesta PUT produce lo stesso risultato

Implementazione in Python

Utilizzo di requests.put() con l'URL della risorsa specifica e i dati completi nel corpo

Quando utilizziamo il metodo PUT, dobbiamo sempre fornire tutti i campi della risorsa, anche quelli che non vogliamo modificare. Se omettessimo alcuni campi, questi verrebbero rimossi o impostati ai valori predefiniti. Questo comportamento differisce significativamente dal metodo PATCH, che vedremo dopo.

Il Metodo HTTP DELETE

Eliminazione di Risorse

DELETE rimuove una risorsa specifica dal server, identificata tramite URL.

Idempotenza

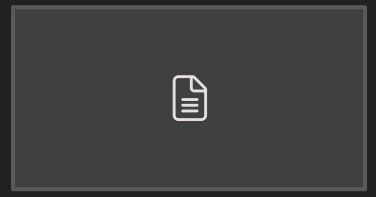
Ripetere la stessa richiesta DELETE non cambia il risultato: dopo la prima esecuzione, la risorsa non esiste più.

Sicurezza

Richiede solitamente autenticazione per prevenire eliminazioni non autorizzate.

In Python, implementare una richiesta DELETE è semplice utilizzando la libreria requests. L'URL deve identificare precisamente la risorsa da eliminare, tipicamente includendo il suo ID. Il server risponderà con un codice di stato appropriato, come 204 No Content in caso di successo.

Differenza tra PUT e PATCH

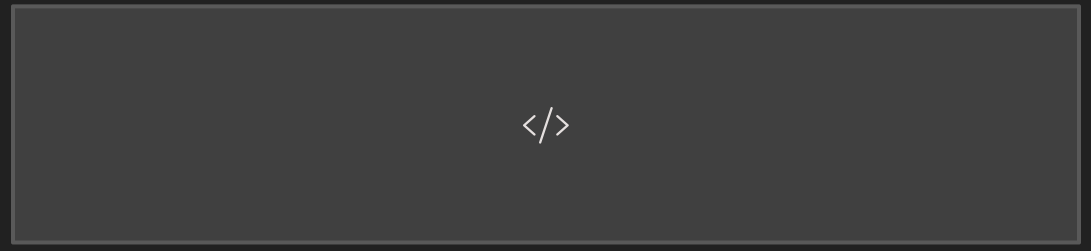


PUT: Sostituzione Completa

Sostituisce l'intera risorsa con una nuova versione

PATCH: Aggiornamento Parziale

Modifica solo i campi specificati nella richiesta



Implementazione Python

requests.patch() per modifiche selettive, requests.put() per aggiornamenti completi

La scelta tra PUT e PATCH dipende dalle esigenze dell'applicazione. PATCH è più efficiente quando si vogliono aggiornare solo pochi campi di una risorsa complessa, mentre PUT è preferibile quando si desidera garantire uno stato coerente e completo della risorsa dopo l'aggiornamento. Molte API moderne supportano entrambi i metodi per offrire maggiore flessibilità agli sviluppatori.

I Codici di Stato HTTP

2XX: Successo

200 OK: richiesta completata con successo

201 Created: risorsa creata correttamente

3XX: Reindirizzamento

301 Moved Permanently: risorsa spostata definitivamente

4XX: Errori Client

400 Bad Request: richiesta malformata

404 Not Found: risorsa non trovata

5XX: Errori Server

500 Internal Server Error: errore generico del server

In Python, è fondamentale verificare sempre i codici di stato nelle risposte HTTP per gestire correttamente i vari scenari. La libreria requests semplifica questo processo, permettendo di accedere al codice di stato tramite l'attributo status_code dell'oggetto response.

Autenticazione nelle API REST

API Key

Un token semplice inviato come parametro URL o header. È il metodo più basilare ma anche meno sicuro.

```
# Autenticazione con API Key
response = requests.get(
    'https://api.esempio.com/dati',
    headers={'X-API-Key':
'chiave_segreta'}
)
```

Basic Auth

Credenziali username:password codificate in Base64. Semplice ma deve essere usato sempre con HTTPS.

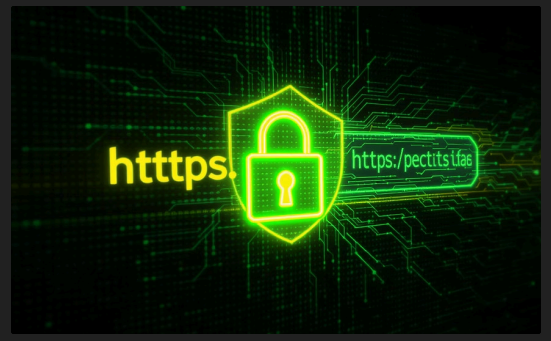
```
# Basic Authentication
from requests.auth import
HTTPBasicAuth
response = requests.get(
    'https://api.esempio.com/dati',
    auth=HTTPBasicAuth('utente',
'password')
)
```

OAuth 2.0

Standard avanzato che permette autorizzazioni granulari e token temporanei. Più complesso ma molto più sicuro.

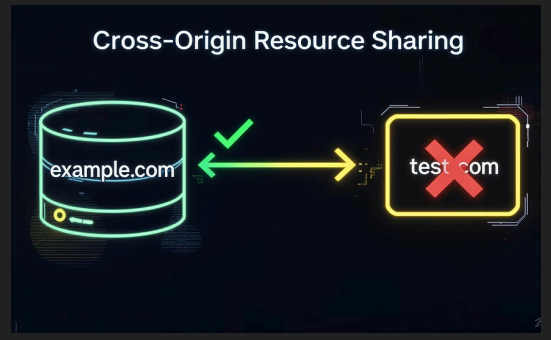
```
# Esempio con token OAuth
headers = {
    'Authorization': 'Bearer
token_accesso'
}
response = requests.get(
    'https://api.esempio.com/dati',
    headers=headers
)
```

Sicurezza nelle API REST



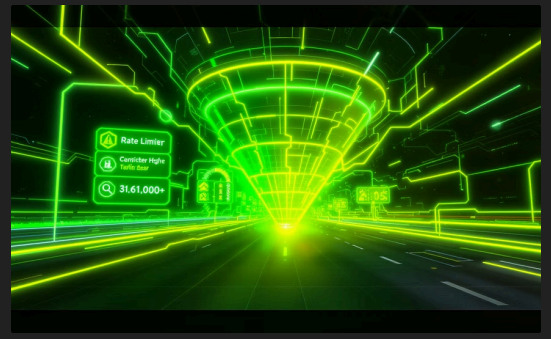
HTTPS

Utilizzare sempre HTTPS per crittografare il traffico tra client e server. In Python, non è necessaria alcuna configurazione speciale: la libreria requests gestisce automaticamente le connessioni HTTPS, verificando i certificati SSL/TLS. È tuttavia possibile personalizzare questo comportamento quando necessario.



CORS

Cross-Origin Resource Sharing (CORS) è un meccanismo che consente o blocca le richieste da domini diversi. Dal lato client Python, non c'è nulla da configurare, ma è importante comprendere questo meccanismo quando si sviluppano API o si interagisce con esse tramite JavaScript in un browser.



Rate Limiting

Limitare il numero di richieste che un client può effettuare in un determinato periodo di tempo. Come utenti di API, dobbiamo rispettare questi limiti e implementare logiche di retry appropriate quando riceviamo errori 429 (Too Many Requests).

Formato JSON: Struttura Base

Oggetti JSON

Un oggetto JSON è una collezione non ordinata di coppie chiave/valore racchiuse tra parentesi graffe {}. Ogni chiave è una stringa, seguita da : e dal valore corrispondente.

```
{
  "nome": "Mario",
  "età": 30,
  "iscritto": true
}
```

Array JSON

Un array JSON è una sequenza ordinata di valori racchiusi tra parentesi quadre []. Gli array possono contenere qualsiasi tipo di valore: stringhe, numeri, booleani, null, oggetti o altri array.

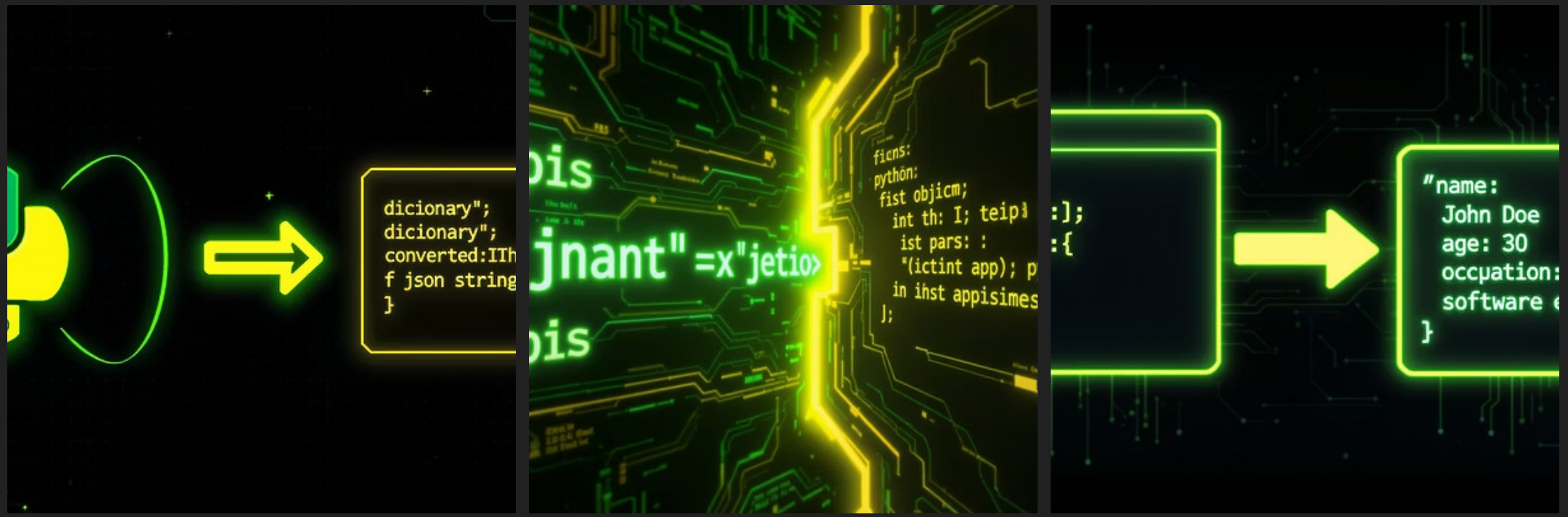
```
[
  "Rosso",
  "Verde",
  "Blu"
]
```

Strutture Nidificate

La vera potenza del JSON sta nella possibilità di annidare oggetti e array, creando strutture dati complesse ma facilmente leggibili.

```
{
  "utente": {
    "nome": "Mario",
    "hobby": ["calcio", "musica"]
  }
}
```

Serializzazione e Deserializzazione JSON



In Python, la serializzazione (convertire oggetti Python in JSON) e la deserializzazione (convertire JSON in oggetti Python) sono operazioni molto semplici grazie al modulo json della libreria standard.

```
import json

# Serializzazione (Python -> JSON)
dati_python = {"nome": "Anna", "età": 28, "corsi": ["Python", "SQL"]}
json_string = json.dumps(dati_python, indent=2)

# Deserializzazione (JSON -> Python)
json_ricevuto = '{"prodotto": "Laptop", "prezzo": 999.99}'
dati_python = json.loads(json_ricevuto)
```

Perché REST è Così Diffuso?

75%

Quota di Mercato

Percentuale approssimativa delle API pubbliche che utilizzano REST

2000

Anno di Nascita

Roy Fielding introduce REST nella sua tesi di dottorato

71%

Sviluppatori

Percentuale di sviluppatori che preferiscono REST rispetto ad altre architetture

REST si è affermato come standard de facto per le API web grazie alla sua semplicità e flessibilità. L'utilizzo di HTTP come protocollo di trasporto rende REST accessibile da qualsiasi piattaforma e linguaggio di programmazione. In Python, framework come Flask e Django REST framework hanno ulteriormente semplificato lo sviluppo di API REST.

La natura stateless delle API REST le rende altamente scalabili, mentre l'adozione di JSON come formato di scambio dati offre un'alternativa più leggera e facile da utilizzare rispetto a XML. Queste caratteristiche hanno reso REST la scelta preferita per l'integrazione di servizi e lo sviluppo di applicazioni moderne.