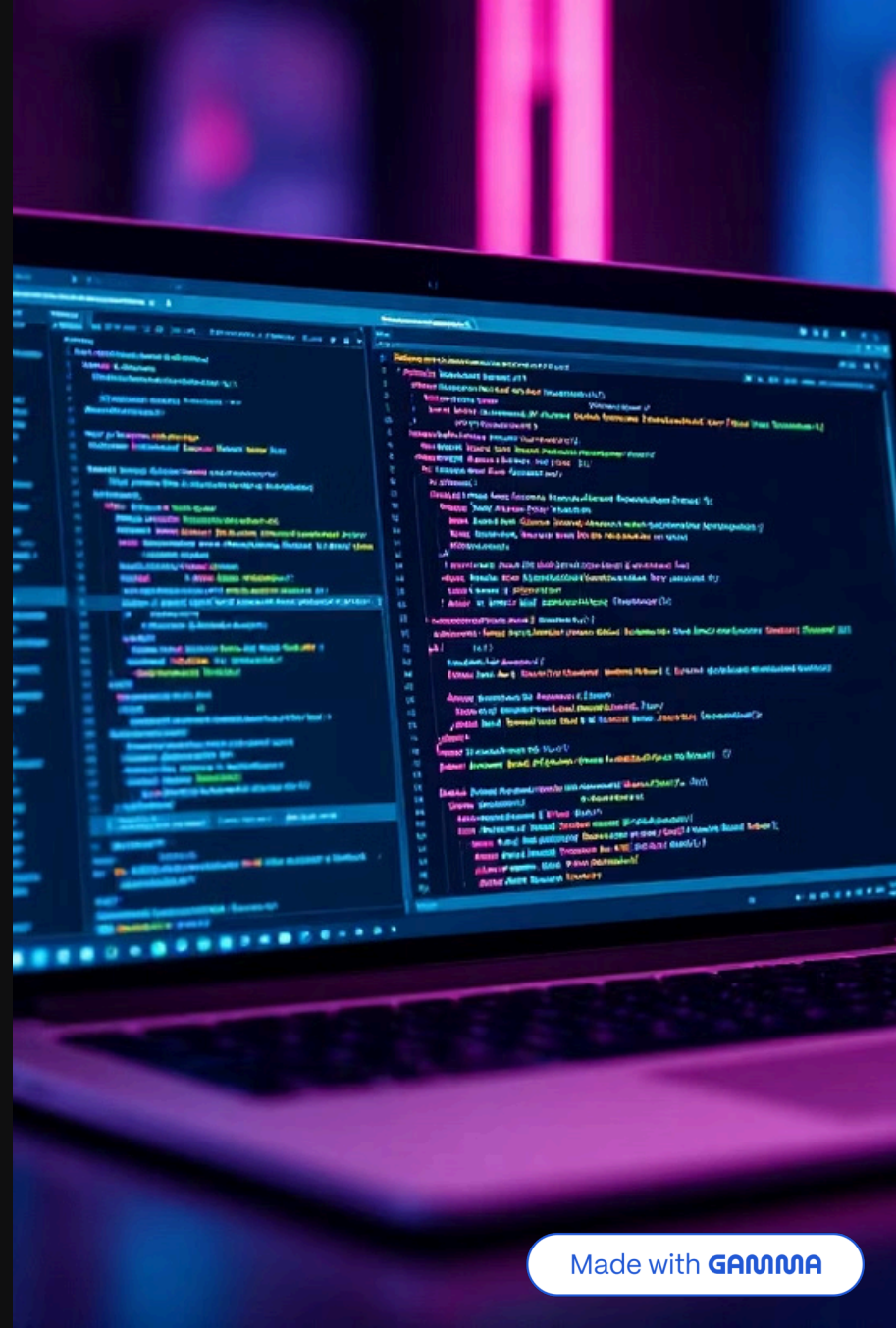
Web Scraping con Selenium: Introduzione

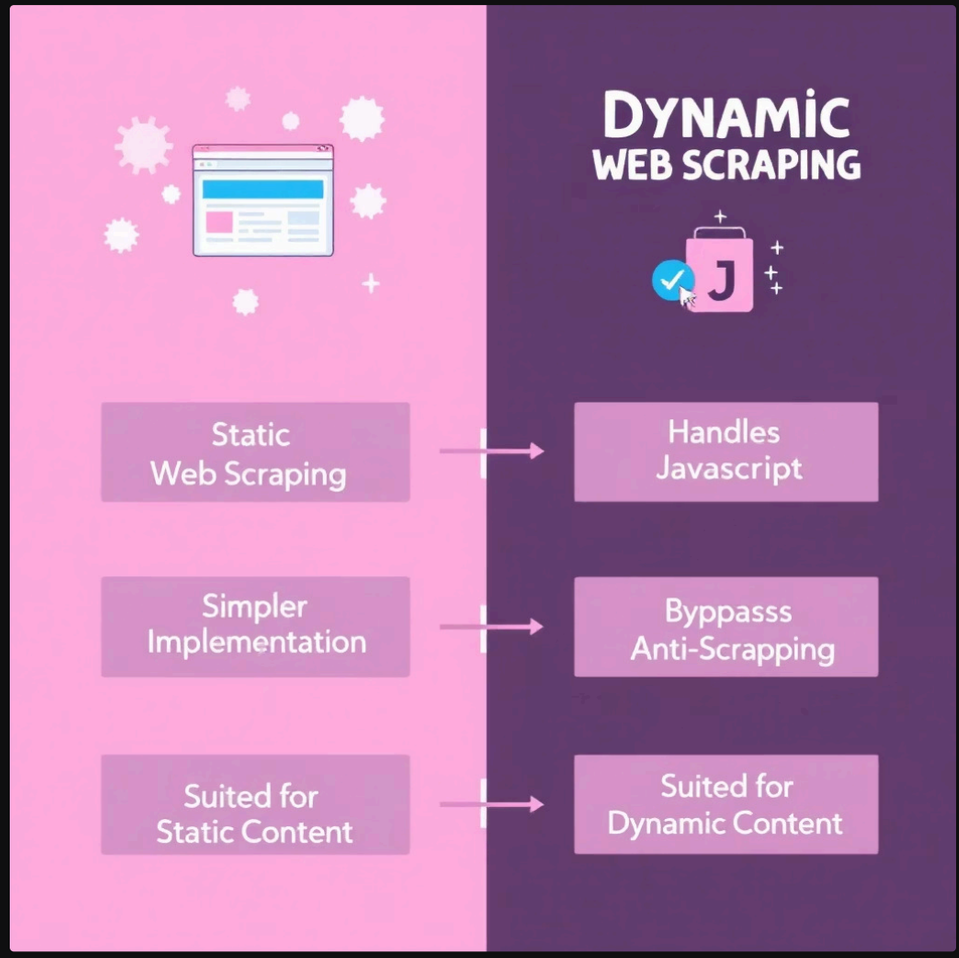
Il web scraping rappresenta una tecnica fondamentale per l'estrazione automatizzata di dati da siti web. Selenium, in particolare, si distingue come strumento open source per l'automazione del browser, essenziale quando si tratta di siti web con contenuti dinamici generati tramite JavaScript o elementi interattivi che richiedono simulazione di comportamento umano.



Perché usare Selenium per il Web Scraping?

Vantaggi Distintivi

- Gestione efficace di contenuti dinamici caricati via JavaScript
- Capacità di simulare interazioni utente complesse (click, scroll, compilazione form)
- Compatibilità con tutti i principali browser (Chrome, Firefox, Edge, Safari)
- Modalità headless per esecuzione invisibile e ottimizzazione risorse



A differenza di strumenti come Beautiful Soup o Requests, Selenium interagisce con il DOM completamente renderizzato, accedendo a contenuti altrimenti irraggiungibili.

Setup e Installazione di Selenium

Prerequisiti

Installare Python 3.12+ e verificare il funzionamento di pip con il comando

```
python --version
```

Installazione Pacchetti

Eseguire il comando

```
pip install selenium webdriver-manager
```

per installare Selenium e il gestore di driver

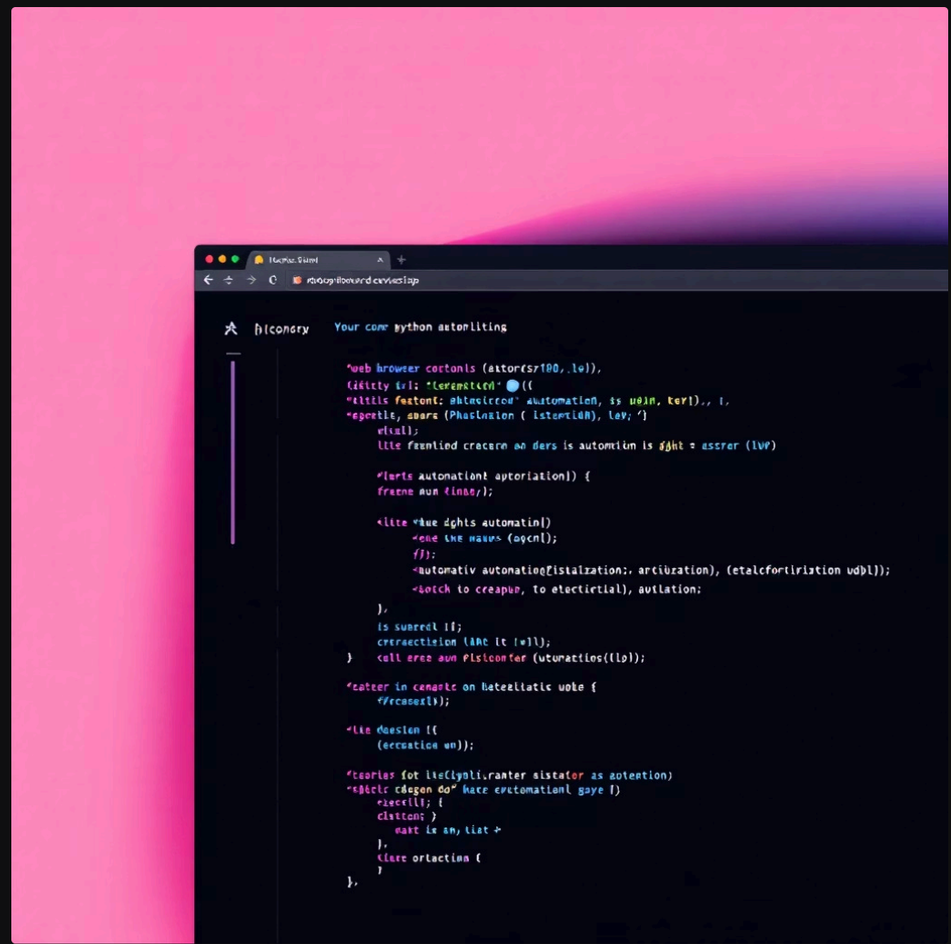
Configurazione WebDriver

Utilizzare webdriver-manager per gestire automaticamente i driver necessari (ChromeDriver, GeckoDriver) senza configurazione manuale

Il modulo webdriver-manager elimina la necessità di scaricare e aggiornare manualmente i driver, garantendo compatibilità con le versioni del browser installate.

Primo Script di Web Scraping con Selenium

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import
ChromeDriverManager

# Inizializza il driver
driver =
webdriver.Chrome(service=Service(ChromeDriverManag
er().install()))

# Visita la pagina
driver.get("https://esempio.it")

# Estrai il contenuto
html = driver.page_source
print(html[:500])  # Mostra i primi 500 caratteri

# Chiudi il browser
driver.quit()
```



Questo script essenziale mostra i passaggi fondamentali: inizializzazione del driver, navigazione della pagina, estrazione del codice HTML e chiusura del browser per rilasciare le risorse.

Localizzare Elementi HTML con Selenium

1 Metodi di Localizzazione

Selenium offre diversi selettori per individuare elementi nella pagina:

- **driver.find_element(By.ID, "id_elemento")** - Più veloce e affidabile
- **driver.find_element(By.CLASS_NAME, "classe_elemento")** - Utile per gruppi
- **driver.find_element(By.XPATH, "//div[@class='articolo']")** - Potente ma più lento
- **driver.find_element(By.CSS_SELECTOR, "div.articolo > h2")** - Versatile e leggibile

2 Esempio Pratico

Per estrarre i titoli di tutti gli articoli su una pagina di notizie:

```
titoli = driver.find_elements(By.CSS_SELECTOR, "article h2.titolo")
for titolo in titoli:
    print(titolo.text)
```

Nota l'uso di **find_elements** (plurale) che restituisce una lista di elementi corrispondenti.

Gli strumenti di ispezione del browser (F12) sono fondamentali per identificare il selettore più appropriato per ciascun elemento.

Gestione Contenuti Dinamici e Attese Esplicite

Problema

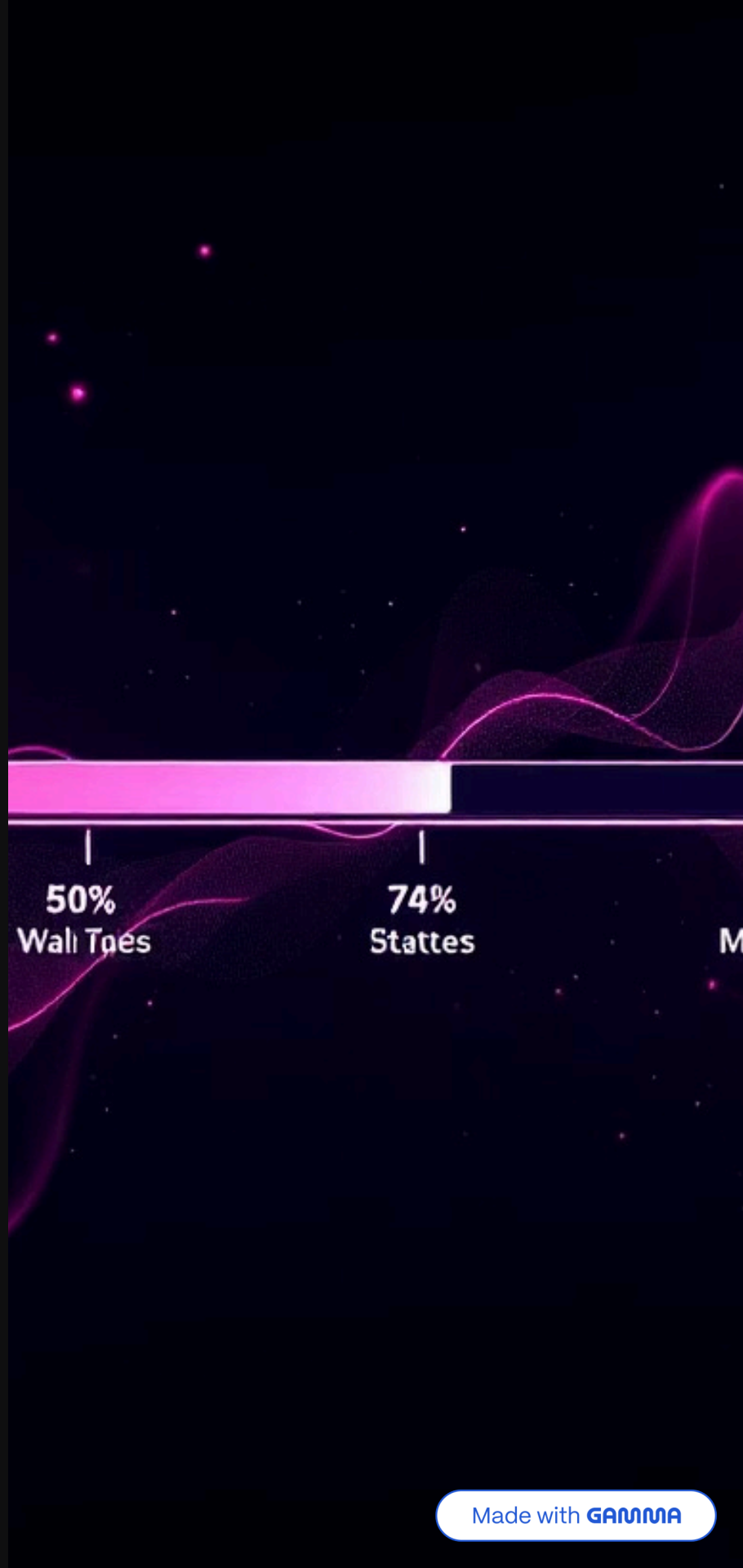
I contenuti dinamici vengono caricati asincroni dopo il caricamento iniziale, causando errori di elemento non trovato

Soluzione

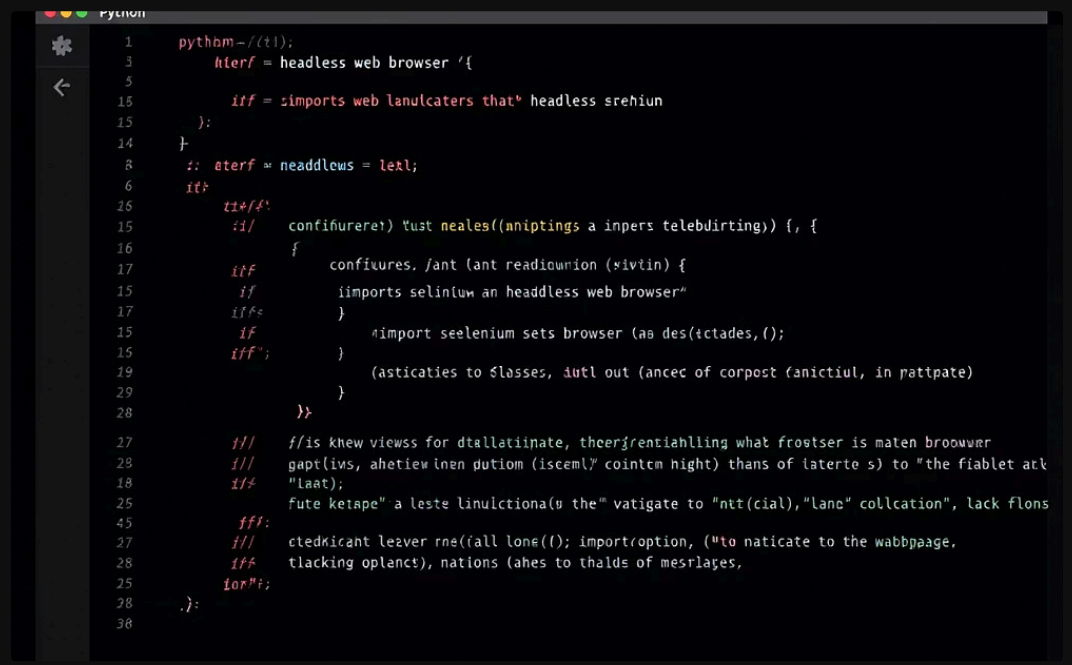
```
from selenium.webdriver.support.ui import
WebDriverWait
from selenium.webdriver.support import
expected_conditions as EC

# Attendi che l'elemento sia visibile (timeout 10 secondi)
elemento = WebDriverWait(driver, 10).until(
    EC.visibility_of_element_located((By.ID,
"contenuto_dinamico"))
)
```

Benefici

Evita errori di **NoSuchElementException** e garantisce l'estrazione di dati completi



Modalità Headless e Ottimizzazione Risorse



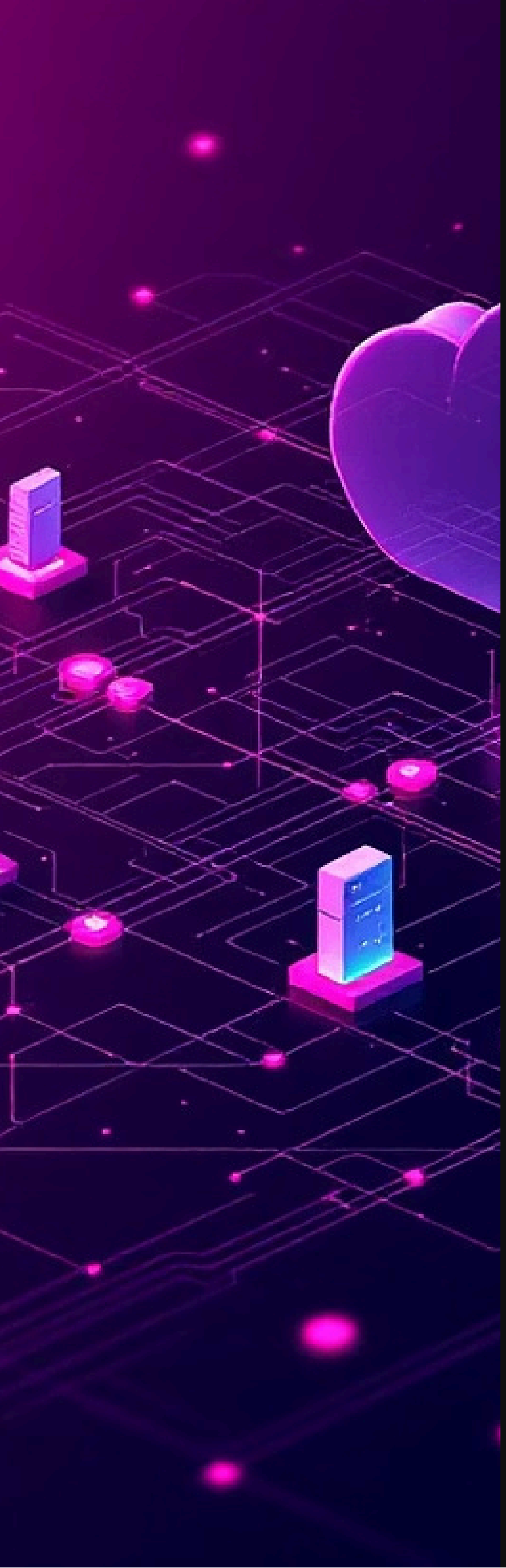
Vantaggi Modalità Headless

- Riduzione consumo memoria fino al 30%
- Velocità di esecuzione superiore
- Perfetta per server e pipeline CI/CD

Configurazione Chrome Headless

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options

options = Options()
options.add_argument("--headless=new")
options.add_argument("--disable-gpu")
options.add_argument("--window-size=1920,1080")

driver = webdriver.Chrome(options=options)
```

Durante lo sviluppo è consigliabile utilizzare la modalità visibile per facilitare il debug, passando alla modalità headless solo in produzione.



Best Practice e Consigli Avanzati

1 Eludere i Sistemi Anti-Bot

Implementare randomizzazione dei tempi di attesa e rotazione degli user-agent:

```
import random
import time

# Attesa randomizzata
time.sleep(random.uniform(1.0, 5.0))

# Rotazione user-agent
user_agents = [
   "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36...",
   "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit..."
]
options.add_argument(f"user-agent=
{random.choice(user_agents)}")
```

2 Struttura Modulare del Codice

Organizzare il codice in classi e funzioni riutilizzabili:

```
class AmazonScraper:
 def __init__(self, headless=True):
self.options = Options()
if headless:
self.options.add_argument("--headless=new")
self.driver = webdriver.Chrome(options=self.options)

def search_product(self, keyword):
# Logica di ricerca

def extract_product_details(self, product_url):
# Logica di estrazione

def close(self):
self.driver.quit()
```

Conclusioni e Risorse Utili

Riepilogo Principali Punti

- Selenium è indispensabile per scraping di siti dinamici complessi
- La combinazione con webdriver-manager semplifica la gestione dei driver
- Le attese esplicite sono fondamentali per contenuti asincroni
- La modalità headless ottimizza le risorse in produzione
- Rispettare sempre i termini di servizio dei siti web (robots.txt)

Prossimi Passi

Approfondire: WebDriverManager, gestione cookie, rotazione proxy, scraping etico e legale

1 Risorse per Approfondire

- **Documentazione ufficiale:** selenium.dev/documentation
- **Tutorial avanzati:** Real Python, Towards Data Science
- **Librerie complementari:** Beautiful Soup (parsing), Pandas (dati)
- **Framework:** Scrapy-Selenium per progetti strutturati