# Guida Pratica all'Utilizzo delle API di OpenAI in Python

Benvenuti a questo corso introduttivo sull'utilizzo delle API di OpenAI tramite Python! In questo percorso esploreremo passo dopo passo come integrare le potenti funzionalità di intelligenza artificiale di OpenAI nelle vostre applicazioni Python.

Impareremo a creare semplici richieste di completamento testuale, gestire le risposte, e costruire applicazioni funzionali anche senza conoscenze avanzate. Iniziamo questo viaggio nel mondo dell'AI generativa!

# Importazione del Pacchetto OpenAI

### Installazione del pacchetto

Prima di importare, installa il pacchetto usando pip:

```
pip install openai
```

### Importazione base

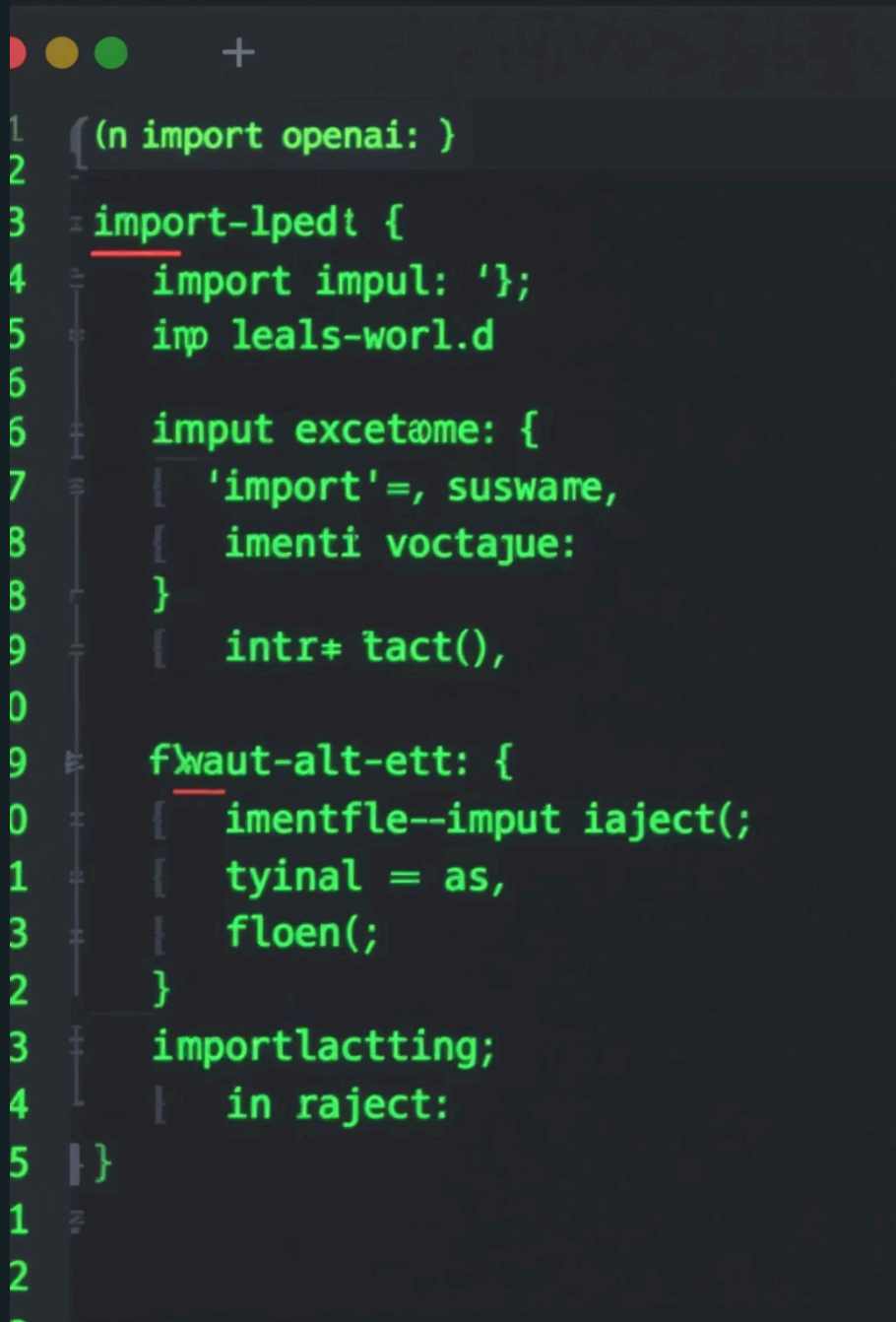
L'importazione più semplice include l'intero modulo:

```
import openai
```

### Importazione avanzata

Per applicazioni più complesse, puoi importare componenti specifici:

```
from openai import OpenAI
```



# Impostazione della API Key

## Ottenere una API Key

Registrati su OpenAI e genera la tua API key dal pannello di controllo personale. Ricorda che questa chiave è come una password: non condividerla e tienila al sicuro.

## Impostazione tramite variabile

Imposta la chiave come variabile nel tuo codice:

```
openai.api_key = "sk-tuaChiaveAPI123456789"
```

## Variabile d'ambiente (consigliato)

Per maggiore sicurezza, usa una variabile d'ambiente:

```
import os
openai.api_key = os.environ["OPENAI_API_KEY"]
```

# Creazione della Prima Richiesta

### Inizializzazione

Dopo aver importato il modulo e configurato l'API key, sei pronto per creare la tua prima richiesta.

### Struttura base

La funzione openai.Completion.create() è il cuore delle richieste di completamento testuale.

### Esecuzione

La chiamata restituirà un oggetto di risposta contenente il testo generato e metadati aggiuntivi.

### Esempio completo

```
response = openai.chat.completions.create(
    model="GPT-4 Turbo",
    prompt="Ciao, mondo!"
)
```

# Specificare il Modello

### GPT-4

Modello più potente e versatile, ideale per compiti complessi ma più costoso.

### DALL-E 3

Specializzato nella generazione di immagini, ideale per trasformare testo in contenuti visivi creativi.

### GPT-4 Turbo

Ottimizzato per velocità e prestazioni, offre un buon equilibrio tra capacità e efficienza.

### GPT-3.5 Turbo

Più veloce ed economico, indicato per la maggior parte delle applicazioni standard.


# Definire il Prompt Base

### Chiarezza

Formula richieste chiare e specifiche. Il prompt è l'istruzione che dai al modello.

### Contesto

Fornisci sufficiente contesto per ottenere risposte pertinenti.

### Esempi

Include esempi di input/output quando possibile per guidare il modello.

### Formato

Specifica il formato di risposta desiderato per risultati più prevedibili.

```
prompt = "Traduci il seguente testo in francese: 'Ciao, come stai?'"
```

# Impostare il Parametro max_tokens

### Definizione

I token sono unità di testo (circa 4 caratteri in inglese). Il parametro max_tokens limita la lunghezza della risposta generata.

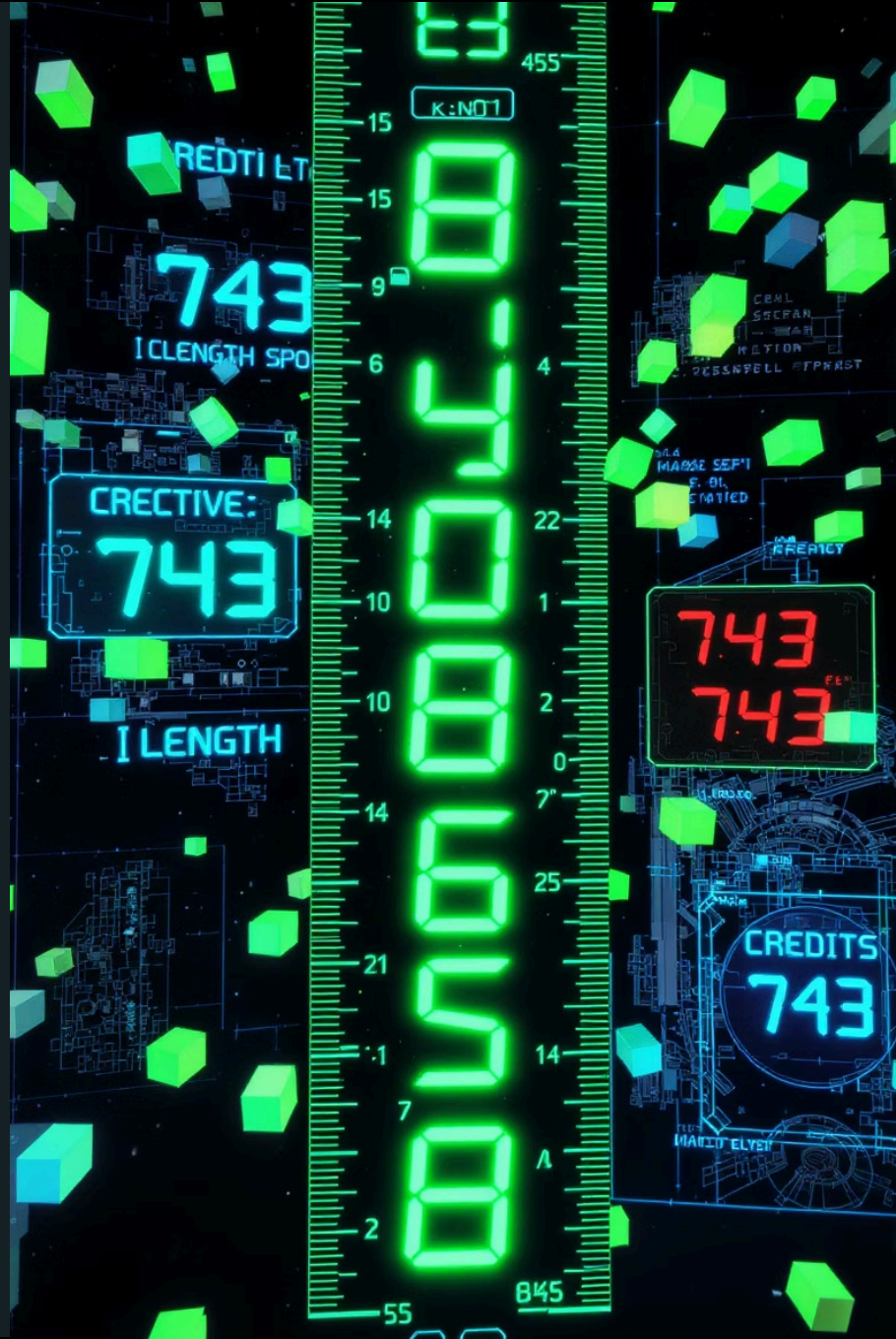
### Bilanciamento

Valori troppo bassi possono troncare le risposte, valori troppo alti possono generare testo superfluo e aumentare i costi.

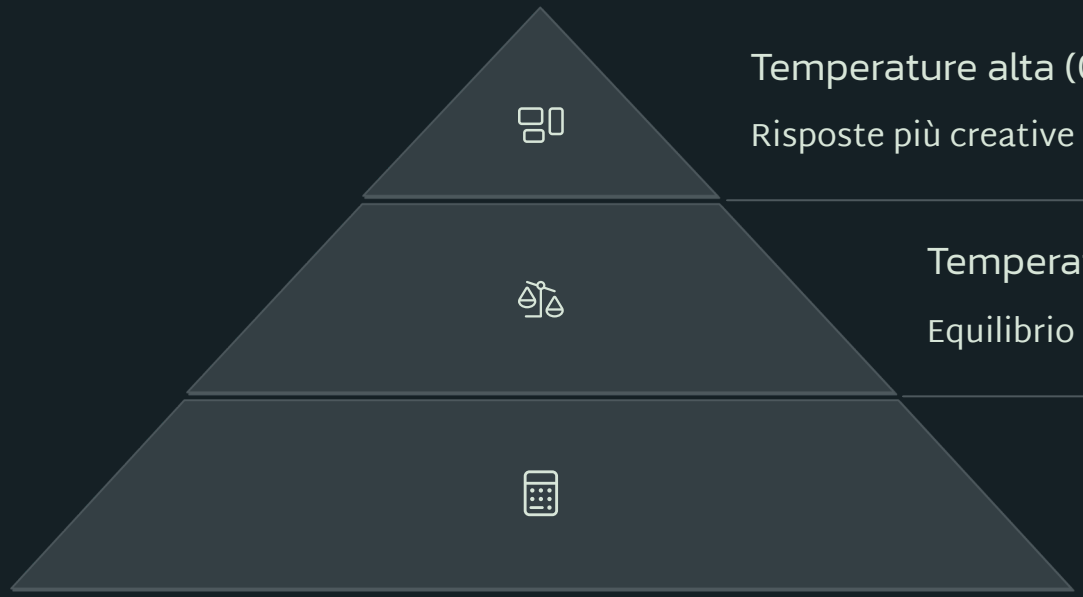
### Costi

Ricorda che vieni addebitato per il numero totale di token (prompt + risposta), quindi ottimizza questo valore.

```
response = openai.chat.completions.create(
    model="gpt-3.5-turbo",
    prompt="Scrivi una breve poesia sul mare",
    max_tokens=150
)
```



# Controllare il Parametro temperature



### Temperature alta (0.8–1.0)

Risposte più creative e variegate

### Temperature media (0.3–0.7)

Equilibrio tra creatività e coerenza

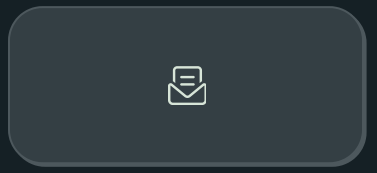
### Temperature bassa (0.0–0.2)

Risposte deterministiche e prevedibili

Il parametro temperature controlla la casualità delle risposte. Utilizzalo in base al tipo di applicazione: per generare contenuti creativi come poesie o storie, usa valori più alti; per risposte fattuali o tecniche, usa valori più bassi per garantire maggiore precisione e consistenza.

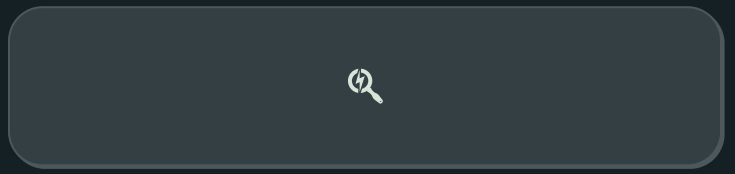
```
temperature=0.7  # Buon equilibrio per la maggior parte degli usi
```

# Ricevere e Stampare la Risposta



### Struttura della risposta

La risposta è un oggetto JSON con diversi campi



### Estrazione del testo

Accedi al testo generato nella struttura annidata

### Visualizzazione

Stampa la risposta elaborata

La risposta dell'API è un oggetto JSON strutturato che contiene vari metadati oltre al testo generato. Per accedere al contenuto testuale effettivo, dovrai navigare questa struttura annidata.

```
testo_risposta = risposta.choices[0].text
print(testo_risposta)
```

Puoi anche stampare l'intera struttura della risposta per esplorarla:

```
print(risposta)
```

# Gestire Errori Comuni delle API

### Authentication Error

Si verifica quando la API key non è valida o è scaduta. Controlla che la chiave sia corretta e che non ci siano spazi o caratteri aggiuntivi.

### Rate Limit Error

Appare quando superi il numero di richieste consentite. Implementa un sistema di backoff esponenziale e riprova dopo un breve intervallo.

### Timeout Error

Quando la richiesta richiede troppo tempo. Considera l'aumento del timeout nelle impostazioni della richiesta o la suddivisione in richieste più piccole.

```
try:
    response = openai.chat.completions.create(...)
except AuthenticationError:
    print("[!] Errore di autenticazione. Controlla la tua API Key.")
    break
except RateLimitError:
    print("[!] Rate limit raggiunto. Attendo 10 secondi e riprovo...")
    time.sleep(10)
except TimeoutError as e:
    print(f"[!] Timeout nella richiesta: {e}")
    logging.error(f"Timeout nella richiesta: {e}")
    break
```

# Creare una Funzione Riutilizzabile

### Validazione

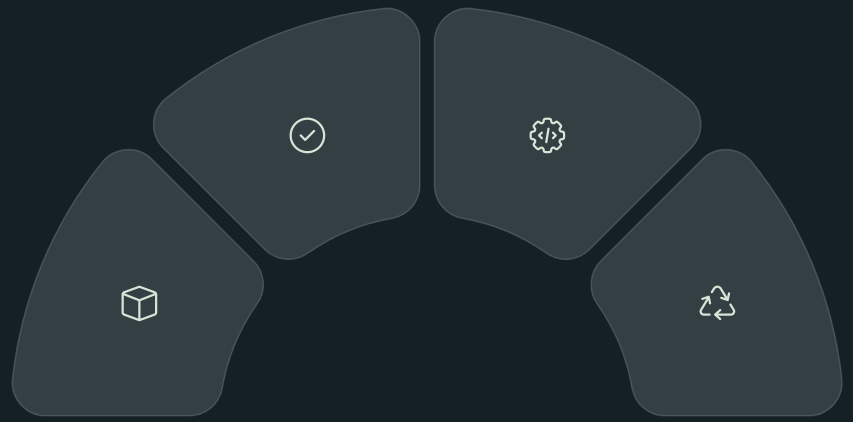
Verifica i parametri di input prima dell'invio all'API

### Gestione Errori

Implementa try/except per catturare e gestire le eccezioni

### Parametrizzazione

Rendi la funzione flessibile con parametri personalizzabili



### Riutilizzabilità

Progetta la funzione per diversi casi d'uso

```
def genera_testo(prompt, max_tokens=100, temperature=0.7, model="text-davinci-003"):
    """Genera testo usando l'API di OpenAI."""
    try:
        response = openai.chat.completions.create(
            model=model,
            messages=[{"role": "user", "content": prompt}],
            max_tokens=max_tokens,
            temperature=temperature
        )
        text = response.choices[0].message.content.strip()
        logging.info(f"Prompt: {prompt}\nRisposta: {text}")
        return text
    except Exception as e:
        return f"Errore: {str(e)}"
```

# Logging delle Risposte

### Configurazione del Logger

Imposta un sistema di logging per monitorare le richieste e le risposte. Il modulo logging di Python è perfetto per questo scopo e permette di registrare diverse tipologie di eventi con livelli di gravità differenti.

### Registrazione delle Richieste

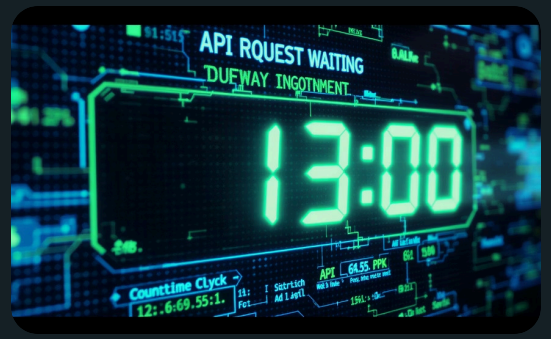
Registra i parametri di ogni richiesta prima dell'invio. Questo facilita il debug e l'ottimizzazione dei prompt. Ricorda di mascherare le informazioni sensibili nei log.

### Archiviazione delle Risposte

Salva le risposte ricevute per analisi future. Questo può aiutarti a costruire un dataset di esempi per migliorare le tue applicazioni e identificare pattern utili.

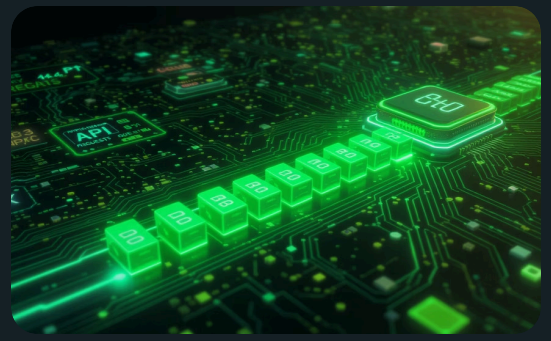
```
import logging
logging.basicConfig(level=logging.INFO, filename="openai_api.log")

def genera_testo_con_log(prompt, **kwargs):
    logging.info(f"Richiesta: {prompt[:50]}...")
    risposta = genera_testo(prompt, **kwargs)
    logging.info(f"Risposta: {risposta[:50]}...")
    return risposta
```

# Gestire Limiti di Quota e Rate Limiting




## Backoff Esponenziale

Implementa un sistema di attesa crescente tra i tentativi falliti. Il tempo di attesa aumenta esponenzialmente ad ogni tentativo, riducendo il carico sul server e aumentando le probabilità di successo.




## Coda di Richieste

Crea una coda per gestire le richieste in modo ordinato, distribuendole nel tempo per evitare picchi di utilizzo. Questo è particolarmente utile per applicazioni che generano molte richieste.


## Monitoraggio Utilizzo

Tieni traccia del consumo di token e richieste per prevenire il superamento delle quote. Implementa avvisi quando ti avvicini ai limiti stabiliti nel tuo piano.

# Prova di Prompt Differenti

1. **Prompt Generico**

   "Parlami dei pianeti."

   *Risposta vaga e generale, può divagare.*

2. **Prompt Specifico**

   "Elenca i pianeti del sistema solare in ordine di distanza dal sole."

   *Risposta precisa e mirata, più utile.*

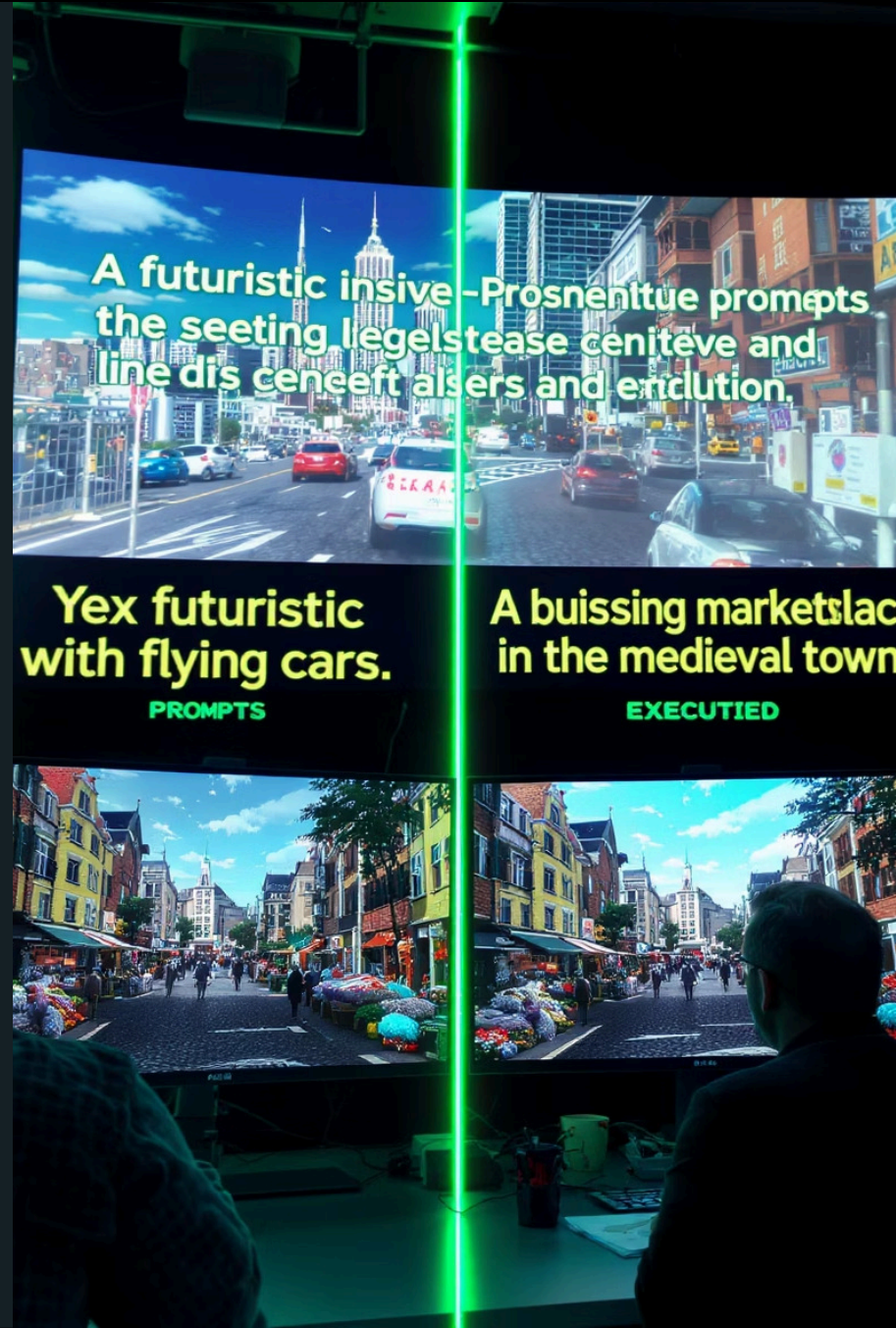
3. **Prompt con Formato**

   "Elenca i pianeti del sistema solare in formato JSON con nome e diametro."

   *Risposta strutturata, ideale per elaborazioni successive.*

La qualità del prompt influenza direttamente la qualità della risposta. Esperimenta con diversi stili di prompt per trovare quello più efficace per il tuo caso d'uso. Considera di creare una libreria di prompt testati per riutilizzarli in futuro.

Ricorda che i modelli più recenti hanno una finestra contestuale più ampia, permettendo prompt più dettagliati con esempi e contesto aggiuntivo.



# Salvare i Risultati e Creare un'Applicazione CLI

1

### Salvataggio su File

Salva i risultati in vari formati (TXT, JSON, CSV) per analisi future o integrazione con altri sistemi.

2

### Argparse

Utilizza la libreria argparse per creare un'interfaccia a riga di comando flessibile e documentata.

3

### Applicazione Completa

Combina tutti gli elementi precedenti in un'applicazione CLI funzionale che accetta prompt dall'utente.

```
import argparse

parser = argparse.ArgumentParser(description='Genera testo con OpenAI')
parser.add_argument('prompt', help='Il prompt da inviare')
parser.add_argument('--output', help='File di output per salvare la risposta')
parser.add_argument('--tokens', type=int, default=100, help='Numero massimo di token')
args = parser.parse_args()

risposta = genera_testo(args.prompt, max_tokens=args.tokens)

if args.output:
    with open(args.output, 'w') as f:
        f.write(risposta)
else:
    print(risposta)
```