

## WildBytes – puntata 2. Calcolatore di Tokens

Ciao Youtubers e bentrovati su WildBytes! Nella puntata di oggi scopriremo come salvare i dati delle nostre variabili, così da poterli riutilizzare anche quando riavviamo il nostro programma.

La capacità di salvare le risposte e, soprattutto, i valori dei tokens usati è fondamentale. Ci permette di monitorare costantemente quanto ci rimane rispetto al budget stabilito. E come direbbe l'androide Roy Batty, interpretato da Rutger Hauer in Blade Runner: "Non vogliamo che questi dati si perdano come lacrime nella pioggia".

Per ottimizzare il nostro lavoro, è utile avvalersi di un "IDE" (Integrated Development Environment) un po' più sofisticato. Un IDE è un termine che identifica i software di supporto alla programmazione che offre vantaggi come il controllo degli errori, una colorazione sintattica che evidenzia le diverse parti del codice, e molte altre funzionalità che esploreremo più avanti.

In questo tutorial uso **PyCharm** perchè è disponibile una versione completamente gratuita del software e per averlo basta andare sul sito internet, fare il download e installarlo.

Una volta aperta PyCharm, creiamo un nuovo progetto e scegliamo dove salvarlo, lo chiameremo “**wildBytes**” e come possiamo vedere **PyCharm** ci ha creato un file: `main.py` con un hint per il codice. Che se facciamo girare stamperà sullo schermo la sempre più celebre frase: “hello world”.

PyCharm ha un sacco di funzioni che al momento non ci interessano, ci basta sapere che il salvataggio è automatico, che se scriviamo `prunt` invece di `print` ci avvisa dell'errore, ma soprattutto ci permette di creare un ambiente virtuale, chiamato Venv dove possiamo gestire e scaricare le librerie.

Quindi creiamo il nostro interprete virtuale andando nelle settings, scegliendo python 3.11 e andiamo a cercare la libreria di “**openAi**”.

A questo punto possiamo creare un file che chiameremo `chatGptAPIWrapper.py` e dentro possiamo fare copia incolla del nostro testo e fare alcune modifiche.

```
import openai
import secretKeys
```

Ovviamente avremo bisogno di copiare il nostro file `secretKeys.py` e per farlo possiamo trascinarcelo dentro.

```
class ChatGptAPIWrapper:
    promptTokens = 0 # tokens utilizzati per il prompt 0.0015 dollari
    completionTokens = 0 # tokens utilizzati per la completion 0.002
    dollari
    totalTokens = 0
    budget = 0.05 # 5 centesimi di dollaro
    models = {"gpt-4", "gpt-3.5-turbo", "babbage-002", "davinci-002",
              "text-davinci-003", "text-davinci-002", "davinci", "curie",
              "babbage", "ada"}
    model = "gpt-3.5-turbo"
```

Qui è dove questo software risulta “**handy**” come dicono gli americani perchè basta selezionare la porzione del codice che voglio indentare, si dice così in python e premere il tasto tab, oppure shift tab per spostarlo indietro.

Un'altra utility che fa comodo, molto comodo, è questa: se aprite le virgolette, lui le chiude automaticamente e la stessa cosa la potete fare anche selezionando una porzione di testo e aprendo le virgolette, lui la include in modo così... “easy”!

Ogni classe, diciamo che ha delle funzioni speciali e quella per fargli fare qualcosa ogni volta che viene chiamata la classe è l' `__init__`:

```
def __init__(self):
    openai.api_key = secretKeys.openAi
```

Ovviamente vi ricordate? Avete sempre la possibilità di andare sulla pagina di openAi -> chatGpt e chiedergli: perchè dobbiamo mettere “self” all'interno delle parentesi quando scriviamo una definizione all'interno di una classe?

E' una convenzione, ma se volete approfondire siete liberi di farlo, anzi, fatelo!

A questo punto posso copiare la nostra funzione `remainingTokens` e scrivere:

```
def remainingTokens(self):
```

```

"""
Calcola i tokens rimanenti partendo dal costo per 1000 tokens.
:return:
"""
# Costo per 1000 tokens
costPer_1000_tokens = 0.002
costPerToken = costPer_1000_tokens / 1000
self.budget = self.budget - (costPerToken * self.totalTokens)
remaining_tokens = float(self.budget / costPerToken)
return f"Ti rimangono ${self.budget:.4f}. Puoi ancora
utilizzare {int(remaining_tokens)} tokens."

```

Non ci serve più passare il budget all'interno della funzione perchè avendo una classe posso creare delle variabili globali usando self e aggiornarle direttamente.

Ogni volta che questa funzione verrà chiamata prenderà il valore che ho impostato all'inizio e di volta in volta scalerà quello che rimane in modo da aggiornare il budget iniziale.

Possiamo testarla scrivendo:

```

if __name__ == '__main__':
    ai = ChatGptAPIWrapper()
    ai.totalTokens = 100
    print(ai.remainingTokens())
    ai.totalTokens = 100
    print(ai.remainingTokens())
    ai.totalTokens = 100
    print(ai.remainingTokens())
    ai.totalTokens = 100
    print(ai.remainingTokens())

```

Avremo bisogno di una funzione che prenda la domanda e che ci estraiga la risposta.

```

def getAnswer(self, question):
    """
    Ottiene una risposta da openAI. Ogni risposta costa 0.002 dollari.
    :param question:
    :return:
    """
    messages = [
        {"role": "user", "content": f"{question}"},
    ]
    response = openai.ChatCompletion.create(
        model=self.model,
        messages=messages,

```

```

        max_tokens=50, # limita la lunghezza della risposta
        temperature=0, # 0 = risposta più probabile, 1 = risposta più
creativa
    )
    answer = response['choices'][0]['message']['content']
    self.promptTokens += int(response['usage']['prompt_tokens'])
    self.completionTokens += int(response['usage']['completion_tokens'])
    self.totalTokens += int(response['usage']['total_tokens'])
    return f"Risposta: {answer}\n{self.remainingTokens()}"

```

Per interagire con il nostro programma e fare una domanda, ci basta usare il comando `ai.getAnswer("Questa è la mia domanda")`. Ho impostato un limite di 50 tokens per le risposte, per mantenere un controllo sui costi. Siccome siamo ancora in fase di test, non utilizzeremo questa funzione, quindi non dovremo preoccuparci dei costi!

Per salvare il contenuto delle variabili possiamo fare in vari modi quello che uso io di solito usa due funzioni, la prima crea una struttura dati json esattamente come quella che usa chatGpt. La seconda salva i dati. in questo modo quando carico il programma posso creare una funzione che va a leggere il file di testo e se trova una struttura di tipo Json riempie le variabili con i nuovi valori.

Ho bisogno di una funzione open e di una funzione save.

```

def open(self, fileName):
    """
    Apre un file e ne ritorna il contenuto.
    :param fileName:
    :return:
    """
    with open(fileName, 'r', encoding="utf-8") as file:
        data = file.read()
    return data

def save(self, fileName, data):
    """
    Salva un file.
    :param fileName: il nome del file
    :param data: il contenuto
    """
    with open(fileName, 'w', encoding="utf-8") as file:
        file.write(data)

```

Json è una libreria già integrata con python, ma per poterla utilizzare dobbiamo importarla.

In pyCharm quando una libreria non è stata importata viene sottolineata con una linea tratteggiata in rosso. Provando a posizionare il cursore sulla parola "json", l'IDE ci suggerirà una soluzione per risolvere il problema. Seguendo questo suggerimento, vedremo che all'inizio del nostro codice verrà automaticamente aggiunta la linea `import json`.

La libreria JSON ci fornisce gli strumenti per convertire facilmente i dizionari di Python in stringhe JSON e viceversa. In Python, un dizionario è una raccolta di coppie chiave-valore, molto simile a un dizionario di italiano o di inglese. Per ogni voce (chiave) c'è una definizione (valore).

```
def serialize(self):
    """
    Serializza i dati in un file json.
    :return: i dati in formato json
    """
    data = {
        "promptTokens": self.promptTokens,
        "completionTokens": self.completionTokens,
        "totalTokens": self.totalTokens,
        "budget": self.budget,
    }
    jsonData = json.dumps(data)
    self.save("data.json", str(jsonData))
    return data

def deserialize(self, data=None):
    """
    Deserializza i dati da un file json.
    :return: i dati in formato json
    """
    jsonData = json.loads(data)
    self.promptTokens = jsonData["promptTokens"]
    self.completionTokens = jsonData["completionTokens"]
    self.totalTokens = jsonData["totalTokens"]
    self.budget = jsonData["budget"]
    return jsonData
```

Come abbiamo visto trasformiamo data in una struttura json usando dumps e possiamo trasformare un testo in un dizionario usando loads. A questo punto posso provare la nostra classe scrivendo:

```
if __name__ == '__main__':
    ai = ChatGptAPIWrapper()
    ai.totalTokens = 100
    print(ai.remainingTokens())
    ai.serialize()
    ai.totalTokens += 100
```

```

print(ai.remainingTokens())
ai.serialize()
ai.totalTokens += 100
print(ai.remainingTokens())
ai.serialize()
ai.totalTokens += 100
print(ai.remainingTokens())
print(ai.serialize())

```

e come vedete ho ottenuto come risposta:

Ti rimangono \$0.0498. Puoi ancora utilizzare 24900 tokens.  
 Ti rimangono \$0.0494. Puoi ancora utilizzare 24700 tokens.  
 Ti rimangono \$0.0488. Puoi ancora utilizzare 24400 tokens.  
 Ti rimangono \$0.0480. Puoi ancora utilizzare 24000 tokens.  
 {'promptTokens': 0, 'completionTokens': 0, 'totalTokens': 400, 'budget': 0.048}

Andando a controllare nel file che ha salvato troveremo i nostri dati. Uno dei vantaggi di questo sistema è che ci permette rapidamente di salvare in modo veramente semplice tutte le informazioni che ci interessano e il file che viene creato è leggibile. Dobbiamo però fare attenzione a come sono formattati i dati all'interno perchè se facciamo una modifica e magari ci dimentichiamo una parentesi graffa o aggiungiamo una virgola in più rischiamo che non sia più leggibile dalla funzione loads di json.

Per fare in modo che il programma carichi i nostri dati ogni volta che viene aperto ci basterà inserirlo nell'init. Per evitare però che il programma si blocchi perchè c'è un errore durante la lettura dei dati, dobbiamo inserire un try/except che in caso di problemi faccia qualcosa.

```

try:
    data = self.open("data.json")
    self.deserialize(data)
    print(f"data loaded: {data}")
except Exception as e:
    print(f"No data found. {e}")
    answer = input("Vuoi che creo un nuovo file? [y/n]")
    if answer.lower() == "y":
        self.serialize()
    else:
        print("Exiting...")
        exit(1)

```

Facendo girare nuovamente il programma come posso vedere il programma carica il file di dati e aggiorna i valori. Ma, come in ogni scenario reale, potrebbero esserci delle complicazioni. Che succede se, per esempio, eliminiamo accidentalmente il file "data.json"? O se, magari per

una distrazione, alteriamo il suo contenuto rimuovendo una parentesi? Il programma, di fronte a un file corrotto, si interromperà facendoci perdere tutti i nostri dati? E questa è un'ottima occasione per mettersi alla prova e vedere come il nostro codice gestisce questi imprevisti.

Il problema da risolvere ora è, a cosa far accoppiare la chiamata alla funzione di salvataggio. Che è una domanda cruciale, tanto quanto quella del lucertolone del Sudan di Willy signori e vengo da lontano.

Io la metterei all'interno della funzione per calcolare i tokens in modo da ridurre al minimo la possibilità che gli aggiornamenti non vengano salvati.

È fondamentale testare il programma. Solo attraverso test ripetuti potremo assicurarci che i dati vengano aggiornati come previsto. E, come potrete vedere, ogni volta che riavviamo il programma, il nostro budget verrà prontamente aggiornato con le informazioni precedenti.

A questo punto possiamo fare una domanda a chat gpt, se volete, come Willy scrivendo `ai.getAnswer("con chi lo facciamo accoppiare il lucertolone del Sudan?")`.

Siamo giunti al termine di questa puntata, e oltre a ricordarvi è completamente gratuito, o come si dice, Open Source con licenza MIT vi ricordo che nella descrizione di ogni puntata, potete trovare il link alla mia pagina gitHub dove trovare il codice che utilizzeremo durante il video, e il pdf con il riassunto dei punti chiave oltre ai link per gli eventuali approfondimenti. Ma prima di salutarvi vi ricordo di mettere un like e iscrivermi al canale se non lo avete fatto e come sempre fate domande, tante domande... ma soprattutto sperimentate gente, sperimentate...

GoodByte Alla Prossima!