Consulta API

Comunicação com as APIs do ChatGPT:
1.1. Estude a API do ChatGPT e monte um exemplo simples em Python.
https://platform.openai.com/docs/quickstart

PYTHON

```python
from openai import OpenAI
import os


client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY",""))


completion = client.chat.completions.create(

model='gpt-4',

messages=[

{"role": "system", "content": "Você é um professor de
história muito ruim em matemática"},
{"role": "user", "content": "qual a tabuada do 5?"}])

print(completion.choices[0].message)
```

Para consulta logo acima, utilizei da biblioteca openai, através da mesma pude criar um ponto de comunicação direta, através obviamente de uma key de request.

Para meu segundo código, busquei optimizar a consulta em questão, logo, ao invés de utilizar uma variável recebendo a key, busquei utilizar um .env para que esta variavel de ambiente fosse consultada deste arquivo:

PYTHON

```python
import openai
from dotenv import load_dotenv
import os

load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")
openai.api_key = api_key


sistema = input('como eu sou?')
pergunta = input('O que devo responder?')



completion = openai.chat.completions.create(
model='gpt-4',
messages=[
{"role": "system", "content": f"{sistema}"},
{"role": "user", "content": f"{pergunta}"}])



texto = completion.choices[0].message.content



print(texto)
```

Isto foi realizado utilizando biblioteca dotenv com a classe load_dotenv
Esta irá carregar o .env na pasta e utilizar da variavel atribuida nele. No caso em questão foi utilizada a partir da variavel api_key.
openai.api_key recebe essa key e realiza consulta.

Também fiz questão de atribuir um input básico para o encaminhamento tanto de como a IA será quanto o que ela deverá responder.

Atributos

Temperature:

> O atributo de temperature define o quão criativo a resposta através de um prompt será:

```
**Baixo (`0.0` a `0.3`)**: Respostas mais previsíveis e
focadas. O modelo tende a escolher as respostas mais prováveis
e consistentes.
**Médio (`0.4` a `0.7`)**: Um equilíbrio entre previsibilidade
e criatividade. Oferece um mix de respostas previsíveis e
criativas.
**Alto (`0.8` a `1.0`)**: Respostas mais criativas e variadas.
O modelo é mais propenso a gerar respostas únicas e
imprevisíveis.
```

Max_tokens

> Já o atributo de max_tokens define a quantidade máxima de letras que serão retornadas a partir do prompt em questão:

```
**Baixo (`50` a `100`)**: Respostas curtas e diretas.
**Médio (`200` a `500`)**: Respostas mais detalhadas e
extensas.
**Alto (`1000` ou mais)**: Respostas muito longas, abrangendo
muitos detalhes.
```

Top_p

> O top_p por sua vez trabalha com a provabilidade acomunada das palavras utilizadas.
> Um numero próximo a 1 estaria indicando que o numero de palavras utilizadas pode até variar, porém sempre garantirá que opções mais prováveis sejam mantidas até atingir o limite.

Top_k

> O top_k seleciona k palavras independente das suas provabilidades acomunadas e ajusta dinamicamente o numero de palavras com base em um limiar de probabilidade comutativa

Já para task 2.3 busquei formatar os promps do seguinte modo:

PYTHON

```
import openai
from dotenv import load_dotenv
import os

load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")
openai.api_key = api_key



completion = openai.chat.completions.create(
model='gpt-4',
messages=[

{"role": "system", "content":"CREATE TABLE Taxi (Placa
VARCHAR(7) NOT NULL, Marca VARCHAR(30) NOT NULL, Modelo
VARCHAR(30) NOT NULL, AnoFab INTEGER, Licenca VARCHAR(9),
PRIMARY KEY(Placa)); CREATE TABLE Cliente (CliId BIGSERIAL
NOT NULL PRIMARY KEY, Documento VARCHAR(18) NOT NULL UNIQUE);
CREATE TABLE ClienteEmpresa (CliId int references
Cliente(cliid), Nome VARCHAR(80) NOT NULL, CNPJ VARCHAR(18)
NOT NULL UNIQUE); CREATE TABLE ClienteFisico (CliId int
references Cliente(cliid), Nome VARCHAR(80) NOT NULL, CPF
VARCHAR(14) NOT NULL unique); CREATE OR REPLACE FUNCTION
insert_client() RETURNS TRIGGER AS $$ declare x INT; BEGIN IF
(TG_TABLE_NAME = 'clienteempresa') THEN INSERT INTO
Cliente(Documento) VALUES (NEW.CNPJ); select cliid into x
from cliente where documento = (new.cnpj); update
clienteempresa set Cliid = x where cnpj = (new.cnpj); ELSIF
(TG_TABLE_NAME = 'clientefisico') THEN INSERT INTO
Cliente(Documento) values (NEW.CPF); select cliid into x from
cliente where documento = (new.cpf); update clientefisico set
Cliid = x where cpf = (new.cpf); END IF; RETURN NEW; END; $$
LANGUAGE plpgsql; CREATE OR REPLACE TRIGGER insert_user AFTER
INSERT ON ClienteEmpresa FOR EACH ROW EXECUTE FUNCTION
insert_client(); CREATE OR REPLACE TRIGGER insert_user AFTER
INSERT ON ClienteFisico FOR EACH ROW EXECUTE FUNCTION
insert_client(); create TABLE Corrida (idCorrida serial
unique NOT NULL, Placa VARCHAR(7) NOT NULL, DataPedido DATE
NOT NULL, cliidCliente int not null, KMTotal numeric(5) not
null, PRIMARY KEY(idCorrida, Placa, DataPedido,
cliidCliente), FOREIGN KEY(cliidCliente) REFERENCES
```

```
cliente(cliid) ON DELETE NO ACTION ON UPDATE NO ACTION,
FOREIGN KEY(Placa) REFERENCES Taxi(Placa) ON DELETE NO ACTION
ON UPDATE NO ACTION); CREATE TABLE Motorista (CNH VARCHAR(11)
NOT NULL, Nome VARCHAR(80) NOT NULL, CNHValid INTEGER, Placa
VARCHAR(7) NOT NULL, PRIMARY KEY(CNH), FOREIGN KEY(Placa)
REFERENCES Taxi(Placa) ON DELETE NO ACTION ON UPDATE NO
ACTION); CREATE TABLE Zona (id SERIAL PRIMARY KEY, Zona
VARCHAR(40) NOT NULL); CREATE TABLE Fila (Zona INTEGER NOT
NULL, CNH VARCHAR(11) NOT NULL, DataHoraIn TIMESTAMP,
DataHoraOut TIMESTAMP, KmIn INTEGER, FOREIGN KEY(Zona)
REFERENCES Zona(id) ON DELETE NO ACTION ON UPDATE NO ACTION,
FOREIGN KEY(CNH) REFERENCES Motorista(CNH) ON DELETE NO
ACTION ON UPDATE NO ACTION)};"},

{"role": "user", "content": "dado create acima, qual select
devo fazer para retornar todos os motoristas que tiveram
maior tempo em fila"}])




texto = completion.choices[0].message.content




print(texto)
```

Os inputs não mudam conforme as tasks anteriores, o que muda é a passagem da role que encaminha agora toda criação do banco de dados.
Uma alternativa para caso seria a utilização de um input de pergunta, possibilitando assim usuário preencher um checkbox e receber a resposta na sequência
Realizei isto do seguinte modo:

PYTHON

```
import openai

from dotenv import load_dotenv

import os




load_dotenv()

api_key = os.getenv("OPENAI_API_KEY")

openai.api_key = api_key



pergunta = input("Qual sua pergunta acerca deste banco de
dados?")



completion = openai.chat.completions.create(

model='gpt-4',

messages=[

{"role": "system", "content":"CREATE TABLE Taxi (Placa
VARCHAR(7) NOT NULL, Marca VARCHAR(30) NOT NULL, Modelo
VARCHAR(30) NOT NULL, AnoFab INTEGER, Licenca VARCHAR(9),
PRIMARY KEY(Placa)); CREATE TABLE Cliente (CliId BIGSERIAL
NOT NULL PRIMARY KEY, Documento VARCHAR(18) NOT NULL UNIQUE);
CREATE TABLE ClienteEmpresa (CliId int references
Cliente(cliid), Nome VARCHAR(80) NOT NULL, CNPJ VARCHAR(18)
NOT NULL UNIQUE); CREATE TABLE ClienteFisico (CliId int
references Cliente(cliid), Nome VARCHAR(80) NOT NULL, CPF
VARCHAR(14) NOT NULL unique); CREATE OR REPLACE FUNCTION
insert_client() RETURNS TRIGGER AS $$ declare x INT; BEGIN IF
(TG_TABLE_NAME = 'clienteempresa') THEN INSERT INTO
```

```
Cliente(Documento) VALUES (NEW.CNPJ); select cliid into x
from cliente where documento = (new.cnpj); update
clienteempresa set Cliid = x where cnpj = (new.cnpj); ELSIF
(TG_TABLE_NAME = 'clientefisico') THEN INSERT INTO
Cliente(Documento) values (NEW.CPF); select cliid into x from
cliente where documento = (new.cpf); update clientefisico set
Cliid = x where cpf = (new.cpf); END IF; RETURN NEW; END; $$
LANGUAGE plpgsql; CREATE OR REPLACE TRIGGER insert_user AFTER
INSERT ON ClienteEmpresa FOR EACH ROW EXECUTE FUNCTION
insert_client(); CREATE OR REPLACE TRIGGER insert_user AFTER
INSERT ON ClienteFisico FOR EACH ROW EXECUTE FUNCTION
insert_client(); create TABLE Corrida (idCorrida serial
unique NOT NULL, Placa VARCHAR(7) NOT NULL, DataPedido DATE
NOT NULL, cliidCliente int not null, KMTotal numeric(5) not
null, PRIMARY KEY(idCorrida, Placa, DataPedido,
cliidCliente), FOREIGN KEY(cliidCliente) REFERENCES
cliente(cliid) ON DELETE NO ACTION ON UPDATE NO ACTION,
FOREIGN KEY(Placa) REFERENCES Taxi(Placa) ON DELETE NO ACTION
ON UPDATE NO ACTION); CREATE TABLE Motorista (CNH VARCHAR(11)
NOT NULL, Nome VARCHAR(80) NOT NULL, CNHValid INTEGER, Placa
VARCHAR(7) NOT NULL, PRIMARY KEY(CNH), FOREIGN KEY(Placa)
REFERENCES Taxi(Placa) ON DELETE NO ACTION ON UPDATE NO
ACTION); CREATE TABLE Zona (id SERIAL PRIMARY KEY, Zona
VARCHAR(40) NOT NULL); CREATE TABLE Fila (Zona INTEGER NOT
NULL, CNH VARCHAR(11) NOT NULL, DataHoraIn TIMESTAMP,
DataHoraOut TIMESTAMP, KmIn INTEGER, FOREIGN KEY(Zona)
REFERENCES Zona(id) ON DELETE NO ACTION ON UPDATE NO ACTION,
FOREIGN KEY(CNH) REFERENCES Motorista(CNH) ON DELETE NO
ACTION ON UPDATE NO ACTION)};"},

{"role": "user", "content": f"{pergunta}"}]

,temperature = 0.7,
max_tokens=200,
top_p=1
)
texto = completion.choices[0].message.content



print(texto)
```

```
(tf-gpu) oem@Leonardo:~/pythonGPU/PythonMaximiza/task1$ python3 apiGPT.py
Qual sua pergunta acerca deste banco de dados?Gostaria de saber como posso extrair um relatório que traga todos os motoristas
 que tiveram maior tempo em fila durante o ano de 2014
Para extrair o relatório que você mencionou, é preciso calcular a diferença entre as datas de entrada e saída de cada motoris
ta na fila. Depois, agrupar esse tempo por motorista e ordenar os resultados do maior para o menor tempo acumulado. Aqui está
 uma consulta SQL que pode fazer isso:

```sql
SELECT
  Motorista.CNH,
  Motorista.Nome,
  SUM(EXTRACT(EPOCH FROM (Fila.DataHoraOut - Fila.DataHoraIn))) as TotalSecondsInQueue
FROM
  Fila
JOIN
  Motorista
ON
  Fila.CNH = Motorista.CNH
WHERE
  EXTRACT(YEAR FROM Fila.DataHoraIn) = 2014 OR EXTRACT(YEAR FROM Fila.DataHoraOut) = 2014
GROUP BY
  Motorista.CNH,
  Motorista.Nome
ORDER BY
  TotalSecondsInQueue DESC;
```
```

Realizei alguns testes utilizando emojis
Digite um texto e retorne os emojis:

PYTHON

```python
import openai
from dotenv import load_dotenv
import os



load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")
openai.api_key = api_key




guest = True
while(guest):
pergunta = input("O que deseja traduzir para emojis?\n")
completion = openai.chat.completions.create(
model='gpt-4',
messages=[

{"role": "system", "content":"Irei lhe encaminhar um texto e
deverá traduzir este texto para emoji. Não utilize texto
regular. Faça seu melhor utilizando emojis)};"},

{"role": "user", "content": f"{pergunta}"}]

,temperature = 0.9,

max_tokens=64,

top_p=1

)

texto = completion.choices[0].message.content

print(texto)

guest = input ("deseja continuar perguntando? se sim digite 1
se não digite 0 \n")
```

Digite os emojis e retorne texto:

PYTHON

```python
import openai

from dotenv import load_dotenv

import os



load_dotenv()

api_key = os.getenv("OPENAI_API_KEY")

openai.api_key = api_key




guest = True

while(guest):

pergunta = input("Informe os emojis?\n")

completion = openai.chat.completions.create(

model='gpt-4',

messages=[

{"role": "system",

"content":"Irei lhe mandar uma série de emojis, deverá
responder esta mensagem com texto em portugues)};"},

{"role": "user",

"content": f"{pergunta}"}]

,temperature = 0.9,

max_tokens=64,
```

```
    top_p=1

    )

    texto = completion.choices[0].message.content

    print(texto)

    guest = input ("deseja continuar perguntando? se sim digite 1
    se não digite 0 \n")
```

Infelizmente ao utilizar emojis na pergunta os retornos não foram corretos, considero que ainda não haja algo que consiga fazer isto.