

```
1 import azure.functions as func
2 import logging
3
4 app = func.FunctionApp(http_auth_level=func.AuthLevel.ANONYMOUS)
5
6 @app.route(route='firstFunctionAPI')
7 def firstFunctionAPI(req: func.HttpRequest) -> func.HttpResponse:
8     logging.info('Python HTTP trigger function processed a request.')
9
10    name = req.params.get('name')
11    if not name:
12        try:
13            req_body = req.get_json()
14        except ValueError:
15            pass
16
17    return func.HttpResponse('Hello, %s!' % name, status_code=200)
18
19
```

Executed function "firstFunctionAPI". Response: "Hello, Ricardo Calvo. This HTTP triggered function executed successfully." Source: Azure Functions

Aqui podemos ver la implementación de nuestra función de Azure en donde le mandamos como parámetros nuestro nombre y se contesta con un “Hello {name}”

RicardoCalvo  
Aplicación de funciones

Buscar

Examinar Actualizar Detener Reiniciar Intercambiar Obtener perfil de publicación

Introducción

Registro de actividad

Control de acceso (IAM)

Etiquetas

Diagnosticar y solucionar problemas

Microsoft Defender for Cloud

Eventos (versión preliminar)

Mejor juntos (versión preliminar)

Funciones

Implementación

Configuración

Rendimiento

Plan de App Service

API

Supervisión

Automation

Soporte y solución de problemas

Essentials

Grupo de recursos (mover)

SemanaTec

Estado

En ejecución

Ubicación (mover)

West US

Suscripción (mover)

Azure for Students

Id. de suscripción

913e53e5-7e1b-4a9f-a972-87431b4665df

Etiquetas (editar)

Agregar etiquetas

Dominio predeterminado

ricardocalvo.azurewebsites.net

Sistema operativo

Linux

Plan del servicio de aplicaciones

ASP-SemanaTec-97fd (V1:0)

Versión en tiempo de ejecución

4.34.2.2

Funciones Métricas Propiedades Notificaciones (0)

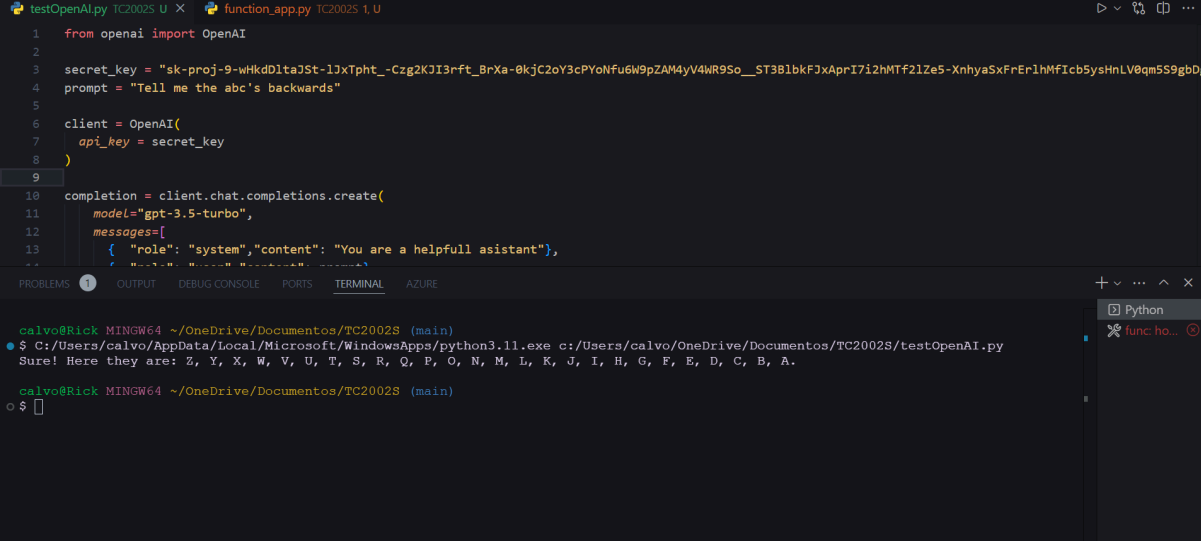
Creación de funciones en el entorno preferido

Crear nueva función

Ver código fuente

Esta es mi aplicación que usaremos para la elaboración de nuestro backend con ayuda de funciones dentro de nuestra aplicación.

## Python file



The image shows a Visual Studio Code editor window with two tabs: `testOpenAI.py` and `function_app.py`. The `testOpenAI.py` tab is active, displaying a Python script that uses the `OpenAI` library to interact with the GPT-3.5-turbo model. The script sets a secret key, a prompt, and a system message, then creates a completion object and prints the result.

```
1 from openai import OpenAI
2
3 secret_key = "sk-proj-9-wHkdDltaJSt-1JxTpht_-Czg2KJI3rft_BrXa-0kjC2oY3cPYoNfu6W9pZAM4yV4WR9So__ST381bkFJxAprI7i2hMTf2lZe5-XnhyaSxFrEr1hMFicb5ysHnLV8qmSS9gbDg"
4 prompt = "Tell me the abc's backwards"
5
6 client = OpenAI(
7     api_key = secret_key
8 )
9
10 completion = client.chat.completions.create(
11     model="gpt-3.5-turbo",
12     messages=[
13         { "role": "system", "content": "You are a helpfull asistant"},
14     ],
15 )
16 print(completion.choices[0].message.content)
```

The terminal window at the bottom shows the execution of the script. The command prompt is `calvo@Rick MINGW64 ~/OneDrive/Documentos/TC2002S (main)`. The command executed is `$ C:/Users/calvo/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/calvo/OneDrive/Documentos/TC2002S/testOpenAI.py`. The output is `Sure! Here they are: Z, Y, X, W, V, U, T, S, R, Q, P, O, N, M, L, K, J, I, H, G, F, E, D, C, B, A.`. The prompt `o $` is visible at the bottom of the terminal.