

# JavaScript con Google Apps Script en soluciones IoT

## Objetivo

Se busca aprender lo básico del IDE (Integrated Development Environment) de Google Apps Script (GAS) y de programación JavaScript en el contexto de soluciones IoT. Gracias al GAS, los estudiantes podrán crear soluciones web con cómputo y almacenamiento en la nube sin tener la necesidad de crear un servidor lo cual implicaría muchos preparativos, pero también costos. Lo que Google permite es imaginar que nuestro servidor ya existe, ya está listo, ya tiene todas las herramientas, ya tiene todas las herramientas para brindar seguridad, confiabilidad, etc, solo debemos crear las soluciones que para lo que buscamos, pueden resultar gratuitas. Eso es lo que se conoce como Serverless. Las soluciones más sencillas de cómputo y almacenamiento en la nube son los web services. Un conjunto de web services o webservices más sofisticados y documentados pueden ser considerados como una API. Trabajar con GAS no nos hará dependientes de Google por varias razones:

- Igual que las API, los web services pueden estar escritos con cualquier lenguaje de programación, los puede consumir cualquier aplicación alrededor del mundo. Desde este punto de vista, la aplicación que requiera consumir nuestro web services podría ni siquiera se enterará que ella está hecha con GAS.
- Los entornos más avanzados para crear web services y en general soluciones en la nube con cómputo son: Django, Flask, Node.js. Los dos primeros usan el lenguaje de programación python, el último usa javascript. Pues bien, Google Apps Script también usa javascript, de manera que trabajar en GAS representa un notable avance para programar en Node. La diferencia es que si quieres trabajar en Django, Flask o Node.js necesitarás implementar un servidor propio con todo lo que esto implica, lo cual haremos en prácticas futuras, pero con menores costos gracias a la experiencia ganada.

Es importante recalcar que en esta guía no se crean aún web services ni APIs, nada de eso, solo se adquieren las bases de programación y de uso del IDE. Se aprende también a crear scripts que corren en la nube lo cual es esencial para más adelante poder crear web services. En conclusión, la meta en un mayor plazo son los webservices, las API, pero por ahora lograremos conquistar el IDE de GAS.

## Competencias que esta guía logra:

- El estudiante sabrá consultar a Chat GPT por el código para un script de manera que realmente obtenga lo que busca.
- El estudiante será capaz de usar el IDE de Google Apps Script para escribir scripts y para depurar código
- El estudiante sabrá que existen las “ejecuciones” y cómo se consultan.

# Conocimientos previos

## IDE:

Significa "Entorno de Desarrollo Integrado" en español o "Integrated Development Environment" en inglés. Es una herramienta de software que proporciona un entorno completo y unificado para escribir, desarrollar, depurar y administrar aplicaciones de software que ese entorno permite. Un IDE generalmente incluye varias características y herramientas que facilitan el proceso de desarrollo de software, como la edición de código, la depuración, la gestión de proyectos, la compilación y la integración con otras herramientas y servicios. Un IDE incluye usualmente:

- Editor de código: Donde los desarrolladores escriben y editan código.
- Depurador: Herramientas para encontrar y corregir errores en el código.
- Compilador/Intérprete: Que traduce el código fuente a un formato ejecutable.
- Gestión de proyectos: Maneja archivos, dependencias, y configuraciones.
- Integración con sistemas de control de versiones: Como Git.

Una confusión común es que muchas plataformas de desarrollo tienen el mismo nombre que el IDE que involucran. Por ejemplo, Matlab es un lenguaje de programación, pero para usar ese lenguaje se requiere el IDE de Matlab. Lo mismo podría decirse de Arduino. Para diferenciarlos podríamos decir por ejemplo que existe Arduino y existe Arduino IDE. Hay otros casos en que la diferencia es clara, por ejemplo GRC es el IDE de GNU Radio. Pero igual hay IDE que soportan múltiples lenguajes de programación, por ejemplo Visual Studio Code.

## Ejemplos de IDE:

### Visual Studio Code (VS Code):

Un editor de código que se ha convertido en un IDE completo mediante extensiones. Soporta depuración, control de versiones, integración con múltiples lenguajes, y más.

### PyCharm:

Un IDE especializado en Python, con herramientas avanzadas para depuración, análisis de código, gestión de proyectos, y soporte para frameworks.

### Eclipse:

Un IDE de código abierto que es altamente extensible. Es conocido principalmente por su soporte para Java, pero puede ser utilizado con otros lenguajes mediante plugins.

### Arduino IDE:

Un IDE simplificado diseñado específicamente para la programación de microcontroladores Arduino, ideal para proyectos de electrónica y prototipos de hardware.

Otros ejemplos que son plataformas con IDE que llevan el mismo nombre:

### Google Colab:

Es más una herramienta de desarrollo en la nube para Python, enfocada en ciencia de datos y machine learning, con capacidades de edición de código, pero no es un IDE completo.

PythonAnywhere: Es una plataforma de desarrollo en la nube que proporciona un editor de código y un entorno para ejecutar y alojar aplicaciones Python, pero no es un IDE completo.

Node-RED: Es una herramienta de desarrollo visual para la programación de flujos, especialmente en proyectos de IoT, pero no es un IDE tradicional.

## Script

Los scripts son secuencias de comandos que se ejecutan de principio a fin y pueden ser utilizados para una amplia variedad de propósitos, desde automatizar tareas simples hasta realizar procesos complejos. Los scripts están escritos en un lenguaje de programación específico. Estos lenguajes pueden variar ampliamente y van desde JavaScript y Python hasta Bash, PowerShell y muchos otros. La principal función de un script es automatizar tareas. Pueden realizar acciones como procesar datos, interactuar con archivos, realizar cálculos, gestionar sistemas y mucho más sin la intervención manual continua del usuario.

## JavaScript

JavaScript es un lenguaje de programación de alto nivel ampliamente utilizado en el desarrollo web y más allá. Fue inventado por Brendan Eich, cofundador del proyecto Mozilla, Mozilla Foundation y la Corporación Mozilla. Como Python, es un lenguaje interpretado. JavaScript puede ser usado para realizar cualquier tarea de programación, pero quizá no cuenta con tantas librerías como Python. Por el contrario, cuando se trata de soluciones de programación del lado del cliente (navegadores web) para agregar interactividad, inteligencia y dinamismo a las páginas web, JavaScript es el que tiene la mayor aceptación mundial. Cuando se usa del lado del cliente, actúa como un lenguaje de scripting permitiendo a los desarrolladores crear experiencias web interactivas al manipular el Document Object Model (DOM) de una página. Esto incluye tareas como validar formularios en tiempo real, realizar animaciones, cargar datos dinámicamente y modificar el contenido de una página en función de las acciones del usuario. Es posible crear juegos, animaciones 2D y gráficos 3D, aplicaciones integradas basadas en bases de datos ¡y mucho más!.

El uso de JavaScript se ha expandido a las soluciones en la nube permitiendo el desarrollo de aplicaciones móviles híbridas y servidores a través de plataformas como Node.js. A pesar de tener menos bibliotecas en comparación con Python, la comunidad de JavaScript es robusta, y existen numerosos frameworks y bibliotecas que permiten a los desarrolladores crear aplicaciones web modernas y dinámicas de manera eficiente."

JavaScript por sí solo es bastante compacto aunque muy flexible, y los desarrolladores han escrito gran cantidad de herramientas encima del núcleo del lenguaje JavaScript, desbloqueando una gran cantidad de funcionalidad adicional con un mínimo esfuerzo. Esto incluye:

- Interfaces de Programación de Aplicaciones del Navegador ([APIs](#)) — APIs construidas dentro de los navegadores que ofrecen funcionalidades como crear dinámicamente contenido HTML y establecer estilos CSS, hasta capturar y manipular un vídeo desde la cámara web del usuario, o generar gráficos 3D y muestras de sonido.
- APIs de terceros, que permiten a los desarrolladores incorporar funcionalidades en sus sitios de otros proveedores de contenidos como Twitter o Facebook.
- Marcos de trabajo y librerías de terceros que puedes aplicar a tu HTML para que puedas construir y publicar rápidamente sitios y aplicaciones.

## Google Apps Script (GAS)

Google Apps Script es una plataforma de desarrollo que permite a los usuarios crear aplicaciones y automatizaciones personalizadas para los productos y servicios en la Web, como web services, Web Apps, API. Se distingue de otras por:

- Ser serverless
- La facilidad para manipular recursos de Google, como Google Sheets, Google Docs, Google Drive y otros.
- Usa el lenguaje de programación que usa es JavaScript tanto para programación del lado del cliente (front end) como del lado del servidor (back end)
- Tiene facilidades para automatizar tareas, por ejemplo que se ejecute una acción cuando llegue un dato o cuando llegue una alarma del Google Calendar

## Objeto de datos/Diccionario

Para una persona que ha trabajado con diccionarios en Python, basta con decirle que en JavaScript también se trabaja con diccionarios, pero que en JavaScript se usa un término diferente para referirse a lo mismo, ese término es “Objeto de datos”. Veamos el siguiente ejemplo:

Ejemplo 1 en javascript. Un objeto de datos sencillo

```
let datos = {
  encargo: "Lista de mercado",
  cliente: true,
  solicitante: "Maria",
  productos: "huevos",
  cantidad: 10
};
```

Ejemplo 2 en Python. El mismo ejemplo pero en Python. Lo curioso es que así también se puede escribir en javascript. Lo bueno de este método es que las claves pueden ser varias palabras, en cambio en el método anterior, una clave es solo una palabra

```
let datos = {
  "encargo": "Lista de mercado",
  "cliente": true,
  "solicitante": "Maria",
```

```
"productos": "huevos",  
"cantidad": 10  
};
```

Ejemplo 3. Un objeto que tiene otro objeto embebido

```
datos = {  
  "encargo": "Lista de mercado",  
  "cliente": True,  
  "solicitante": "Maria",  
  "productos": "huevos",  
  "cantidad": 10,  
  "direccion": {  
    "calle": "Calle 4",  
    "numero": "34-02",  
    "apartamento": "1905"  
  }  
}
```

Trabajar con objetos de datos es demasiado útil, es como el equivalente a una tabla o lista, pero expresada en forma diferente.

## JSON

Para comprender por qué existen los JSON veamos estas consideraciones:

- Los programadores aman expresar los datos en forma de objetos de datos.
- Cuando un usuario necesita transmitir datos desde un lugar remoto a la nube, usando protocolos como MQTT o HTTP, los datos deben ser convertidos previamente a texto, ya que lo que viaja con esos protocolos es texto.
- Una vez los datos llegan al destino en forma de texto, deben ser convertidos nuevamente a datos.

De manera que JSON es lo mismo que un objeto de datos, pero traducido a texto.

Veamos el ejemplo del JSON que corresponde al objeto presentado en el último ejemplo:

```
{ "encargo": "Lista de mercado", "cliente": true, "solicitante": "Maria", "productos":  
"huevos", "cantidad": 10, "direccion": { "calle": "Calle 4", "numero": "34-02", "apartamento":  
"1905" } }
```

Reflexión. El JSON aparentemente es igual al objeto de datos. Para ver la diferencia, veamos una variable llamada data\_JSON en javascript. Podemos ver que la gran diferencia es que todo el contenido del JSON está entre paréntesis, porque en realidad es un texto:

```
let data_JSON = {  
  "encargo": "Lista de mercado",  
  "cliente": true,  
  "solicitante": "Maria",  
  "productos": "huevos",  
  "cantidad": 10,  
  "direccion": {
```

```
"calle": "Calle 4",  
"numero": "34-02",  
"apartamento": "1905"
```

```
}  
};
```

## JSDoc

JSDoc es un conjunto de reglas que se usan para documentar las funciones de JavaScript. Se usa una clave, un valor y un comentario por cada idea que se transmite. La clave es una etiqueta predefinida en JSDoc, el valor es un nombre de variable o parámetro también predefinida en JSDoc y el comentario es lo que el programador quiera explicar. Hay dos grandes motivaciones para aprender a usar JSDoc:

- que toda función que tu crees quede profesionalmente bien explicada
- que el IDE entienda lo que tu explicas sobre una función y use métodos automatizados de ayudas para quienes quieran usar lo que tú has creado.
- Porque si le pasas a Chat GPT el JSDoc de una función que no existe, pero que tu necesitas, Chat GPT comprenderá de manera más certera lo que tu buscas y habrán pocas probabilidades de que se equivoque.

Aquí tienes un listado con viñetas de algunas etiquetas comunes en JSDoc:

- `@function`: usada para nombres de funciones
- `@note`: sirve para agregar notas
- `@param`: Describe un parámetro de una función.
- `@returns` o `@return`: Describe el valor de retorno de una función.
- `@description`: Proporciona una descripción detallada del propósito de la función, clase o método.
- `@example`: Proporciona ejemplos de cómo se usa la función, clase o método.
- `@name`: Proporciona el nombre de una función o método.
- `@type`: Define el tipo de datos de un parámetro, valor de retorno o miembro de una clase.
- `@typedef`: Define un nuevo tipo de datos.
- `@class`: Describe una clase.
- `@constructor`: Describe el constructor de una clase.
- `@inheritdoc`: Indica que una descripción debería heredarse de otra ubicación.
- `@deprecated`: Marca una función, método o clase como obsoleta y sugiere su reemplazo.

Se puede escribir mucho sobre este tema, pero no lo haremos aprovechando que existe el Chat GPT, basta con decir, que si tienes dudas sobre como escribir algo en JSDoc, le puedes pedir ayuda al Chat GPT.

# Reto general para esta práctica

Una tienda digital se dedica a surtir de productos a los hogares que cuentan con neveras inteligentes que reportan a la nube las necesidades que tienen de surtido de los productos. Por ejemplo la falta de huevos, leche, verduras. La oficina de despachos de la tienda nos ha pedido crear un servicio para que ellos puedan en una página web ver todas las solicitudes allegadas de diferentes neveras inteligentes, con todos los detalles, incluyendo el precio en pesos y en dólares al cambio del momento. Para lograrlo usaremos a futuro el concepto de web service y web hook. Pero en este taller solo nos concentraremos en un primer sprint de la solución - en crear un script en GAS capaz de registrar en una hoja de Google Sheet datos IoT simulados. Es decir, por ahora no nos ocuparemos en lograr que lleguen los datos desde un suscriptor remoto real, sino que vamos a simular que ya los tenemos esos datos.

## Tarea 1. Un tour por de Google por GAS

Revise el video que aparece en el siguiente enlace: <https://www.google.com/script/start/> y responda las preguntas de la siguiente tabla:

¿Es posible usar Google Apps Script para resolver un problema matemático?
Si, mediante las funciones que se pueden crear gracias al lenguaje de programación de JavaScript, llegando a ser una gran herramienta para resolver diferentes problemas matemáticos.
¿Es posible usar Google Apps Script para acceder a datos de una base de datos hecha con MySQL?
<p>Si se pueden crear bases de datos MySQL desde 'GAS' usando la clase <b>JDBC (Java Database Connectivity)</b> que es una API que permite a las aplicaciones Java (y en este caso, Google Apps Script) conectarse a bases de datos, como MySQL.</p> <pre>function conectarBaseDeDatos() {   var conn =   Jdbc.getConnection('jdbc:mysql://&lt;host&gt;:&lt;puerto&gt;/&lt;nombre_bd&gt;', 'usuario',   'contraseña');   var stmt = conn.createStatement();   var resultados = stmt.executeQuery('SELECT * FROM tabla');    while (resultados.next()) {     Logger.log(resultados.getString(1)); // Imprime el valor de la     primera columna   }   stmt.close();   conn.close(); }</pre> <p>El anterior código es un ejemplo básico de cómo interactuar con una base de datos MySQL desde Google Apps Script utilizando la clase <b>JDBC</b>. Este código conecta a una base de datos MySQL, ejecuta una consulta para obtener todas las filas de una tabla, y</p>

luego imprime el valor de la primera columna de cada fila en la consola. Cierra la conexión y el statement después de usarlos.

Agregue al menos otros 3 ejemplos de cosas asombrosas que es posible realizar con Google Apps Script

1. **Automatización de tareas en Google Sheets:** Crear scripts que analicen datos, generen informes o automaticen procesos dentro de Google Sheets.
2. **Envío automatizado de correos electrónicos personalizados:** Puedes generar correos electrónicos desde Gmail basados en datos de una hoja de cálculo o cualquier otra fuente.
3. **Creación de aplicaciones web personalizadas:** Google Apps Script permite construir aplicaciones web ligeras con interfaces HTML/CSS/JS que se integran con los diferentes servicios de Google.
4. **Generación automática de calendarios y eventos en Google Calendar:** Puedes crear, modificar o eliminar eventos en Google Calendar a partir de datos de Google Sheets o cualquier otro servicio.

## Tarea 2. Conoce lo básico del IDE de GAS al mismo tiempo que creas un script para simular que ya se tienen a la mano los datos enviados por el suscriptor.

Si logramos simular los datos enviados por el suscriptor, podemos soñar con crear una solución completa aunque aún no se tenga la comunicación establecida entre nuestro script y el suscriptor. Es decir, podemos reducir la complejidad del script para concentrarnos por ahora solo en la solución de nube. Así podremos lograr varios otros objetivos como:

- Conocer qué es un objeto de datos en JavaScript. Si bien Python convierte un JSON en un diccionario, JavaScript lo convierte en un objeto de datos y es importante conquistar ese concepto.
- Conocer comandos JavaScript que jugarán un papel importante en las futuras tareas como: `console.log()`
- Conocer las principales características del IDE de Google Apps Script como:
  - Consultar las ejecuciones que el script ha tenido.
  - Realizar depuración de código.

**Paso 1.** Abrir el IDE de Google Apps Script y crear un nuevo proyecto

te logeas con tu cuenta de gmail en tu navegador > <https://script.google.com/home/start> > Nuevo proyecto > oprime "Proyecto sin título" para darle un nombre al proyecto

Escribe abajo el nombre dado al proyecto:

Practica\_1\_2195533



## Paso 2. Crear el simulador de datos

Simulador de datos que provienen de una nevera. Sin que aún tengamos una conexión real con la nevera, con este simulador podremos afinar el código necesario para lograr registrar los datos de la nevera en un google sheet.

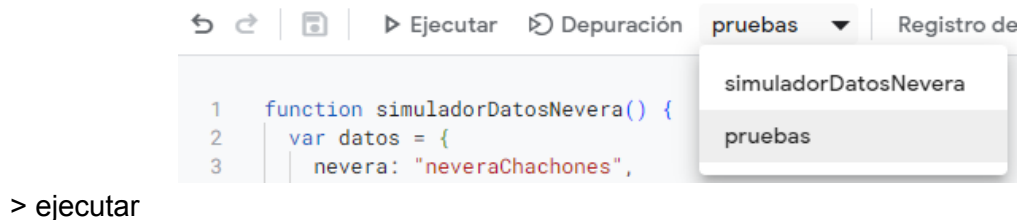
Algunos conocimientos importantes son:

- Más adelante, en un futuro taller, buscaremos que un suscriptor remoto, por ejemplo en Python o en Arduino, pueda hacer una solicitud remota, por internet para enviar datos a nuestro script. Pues bien, para ese viaje, los datos se expresan en forma de texto y lo más conveniente es que el texto sea en formato JSON
- Lo especial es que los datos al entrar al script dejarán de ser un texto JSON y se convertirán en un objeto de datos.
- En este caso, el objeto de datos se llama “datos”
- En la función pruebas() vemos cómo del objeto “datos” se pueden obtener diversos parámetros, como el nombre de la nevera, el número de huevos. En este sentido, JavaScript puede ser ligeramente diferente a Python que no usa objetos tan directos sino diccionarios.

En el IDE de GAS, en el espacio de programación, borra el código que aparece por defecto y copia y pega el siguiente código para el simulador:

```
function simuladorDatosNevera() {  
  var datos = {  
    nevera: "neveraChachones",  
    producto: "huevos",  
    cantidad: 10,  
    pesos: 30000  
  };  
  return datos;  
}  
  
function pruebas() {  
  var a = simuladorDatosNevera();  
  console.log("La nevera se llama: "+a.nevera);  
  console.log("La nevera requiere: " + a.cantidad + ' ' + a.producto);  
}
```

Guarda y Corre la función pruebas() del script y registra el resultado obtenido: Menú superior horizontal > guardar proyecto > en ese mismo menú selecciona la función “pruebas”



#### Registro de ejecución

20:07:25	Aviso	Se ha iniciado la ejecución
20:07:26	Información	La nevera se llama: neveraChachones
20:07:26	Información	La nevera requiere: 10 huevos
20:07:25	Aviso	Se ha completado la ejecución

Tus pruebas. Personaliza un poco los datos para que se diferencien del ejemplo anterior. Puedes cambiar el nombre de la nevera, agregar más productos, imprimir un mensaje que te guste más, sobre todo para demostrar que tu sabes sacar diversos datos de un objeto de JavaScript. Escribe abajo tu nuevo código:

```
function simuladorDatosNevera() {
  var datos = {
    nevera: "nevera_Diego",
    producto_1: {
      nombre_1: "Huevos",
      cantidad_1: 6,
      precio_1: 4200
    },
    producto_2:{
      nombre_2: "Cebolla",
      cantidad_2: 3,
      precio_2: 1500
    },
    producto_3:{
      nombre_3: "Tomate",
      cantidad_3: 3,
      precio_3: 2000
    },
    producto_4:{
      nombre_4: "Mantequilla",
      cantidad_4: 1,
      precio_4: 2000
    }
  };
  return datos;
}

function precioTotal() {
  var total = simuladorDatosNevera();
  precio = total.producto_1.precio_1 + total.producto_2.precio_2 +
total.producto_3.precio_3 + total.producto_4.precio_4;

  return precio;
}

function pruebas() {
  var a = simuladorDatosNevera();
  console.log("El nombre de la nevera es: " + a.nevera);
}
```

```

    console.log("Para hacer huevos pericos se necesita: " +
a.producto_1.cantidad_1 + ' ' + a.producto_1.nombre_1
                + ', ' + a.producto_2.cantidad_2 + ' ' +
a.producto_2.nombre_2 + ', ' + a.producto_3.cantidad_3
                + ' ' + a.producto_3.nombre_3 + " y un poco de " +
a.producto_4.cantidad_4 + " barra de " + a.producto_4.nombre_4 + '.');

    var b = precioTotal();
    console.log("Los productos tuvieron un costo total de: $" + b + "
pesos.");
}

```

## Paso 2. Explora el IDE de GAS. El depurador (debugger)

Con este paso no se busca avanzar en el desarrollo, solo a que practiques con el depurador (debugger) de código del IDE de GAS. Esto va a ser de tremenda ayuda a futuro, cuando tengas que corregir o buscar errores en un código o comprenderlo. Es como una manera de recorrer un código línea por línea de manera similar a lo que se haría en Google Colab con Python.

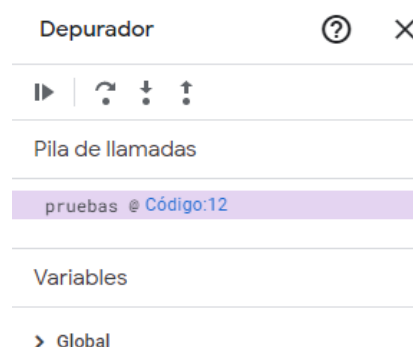
- crea un punto de control en el código: cuando el código se corre usando el depurador, comenzará la ejecución del código, pero esta se detendrá en el punto de control que tu señales, desde el cual comenzará realmente la depuración. Esto se hace así: con el ratón clickea en la parte izquierda (antes de la numeración) la línea de código elegida como punto de control, aparecerá un punto morado, ejemplo:



```


8   | return datos;
9   | }
10  |
11  | function pruebas() {
12  | var a = simuladorDatosNevera();
13  | console.log("La nevera se llama: "+a.nevera);

```

- Menú superior horizontal > selecciona la función que se ha de ejecutar primero, en este caso es "pruebas" > Depuración
- Controla la depuración mediante el Menú que aparece en panel derecho




-  : úsalo para que la ejecución continúe y se detenga en la siguiente línea de código
-  : A veces una línea de código apunta a una función, la cual contiene más código. En este caso podrías desear pasar a la siguiente línea de código, pero también podrías desear penetrar dentro del código de la función. En el segundo caso usarás este ícono.

-  : usalo cuando, estando dentro del código de una función, no te interese continuar dentro de función sino con la siguiente línea del código que la invocó.
- En el ejemplo siguiente, la depuración se detuvo en la función “simuladorDatosNevera()”

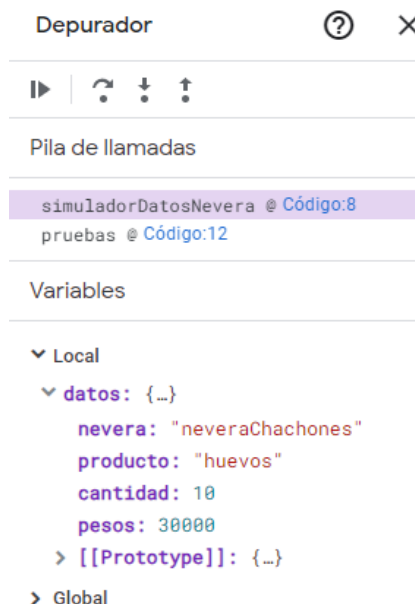
```

10
11 function pruebas() {
12   var a = simuladorDatosNevera();
13   console.log("La nevera se llama: "+a.nevera);
14 }

```

Oprime  para que la ejecución continúe dentro de esa función.

- A medida que el depurador pasa a una nueva línea observa en el menú del panel derecho como cambian las variables. Por ejemplo cuando la ejecución se detiene en la línea 8, en el panel derecho muestra los valores de la variable local “datos”



- En variables de tipo Global, vemos cosas que nosotros podemos haber definido por fuera de las funciones, pero también objetos que GAS incluye por defecto para que este código sea viable. Por ejemplo

```

> Global
> pruebas: function() {...}
> simuladorDatosNevera: function(
  {...}
> Object: function() {...}
> Function: function() {...}

```

Se puede acceder a más detalles de cada cosa clickeando sobre “>”

Demuestra qué has comprendido el concepto. Para ello, introduce un error en el código, que pueda ser detectado usando el depurador. Captura pantalla de las evidencias y pégalas abajo. Por ejemplo, en lugar de a.nevera escribe a.producto. Debes imaginar que tu no conoces donde está el error y debes buscarlo con el depurador. Tan pronto llegues

al lugar donde está el error, captura pantalla y explica lo que esa captura revela.

The screenshot shows the Google Apps Script editor interface. The main editor displays the following code:

```
1 function simuladorDatosNevera() {  
2   var datos = {  
3     nevera: "neveraChachones",  
4     producto: huevos,  
5     cantidad: 10,  
6     pesos: 30000  
7   };  
8   return datos;  
9 }  
10  
11 function pruebas() {  
12   var a = simuladorDatosNevera();  
13   console.log("La nevera se llama: "+a.nevera);  
14   console.log("La nevera requiere: " + a.cantidad + ' ' + a.producto);  
15 }  
16
```

The execution is paused at line 8. The right sidebar shows the 'Depurador' (Debugger) panel with the 'Pila de llamadas' (Call Stack) showing 'simuladorDatosNevera @ Código:4' and 'pruebas @ Código:12'. The 'Variables' panel shows 'Local' variables: 'datos: undefined'.

En este caso, al detener la ejecución en la línea 8, ya no se observan los datos de la variable local `datos`, ya que muestra "undefined" en lugar de los respectivos datos de esa variable, al continuar con la depuración se comprueba que si hay un error:

The screenshot shows the Google Apps Script editor interface with the 'Registro de ejecución' (Execution Log) panel open. The log shows the following messages:

Time	Level	Message
21:27:55	Aviso	Se ha iniciado la ejecución
21:27:56	Error	ReferenceError: huevos is not defined simuladorDatosNevera @ Código.gs:4 pruebas @ Código.gs:12

Este error se debe a que `huevos` no está definida como una cadena de texto, es decir no está dentro de comillas.


The screenshot shows the Google Apps Script editor interface with the 'Registro de ejecución' (Execution Log) panel open. The log shows the following messages:

Time	Level	Message
21:39:21	Aviso	Se ha iniciado la ejecución

The main editor shows the same code as before, but the execution is now paused at line 8 of 'simuladorDatosNevera'. The right sidebar shows the 'Depurador' (Debugger) panel with the 'Pila de llamadas' (Call Stack) showing 'simuladorDatosNevera @ Código:8' and 'pruebas @ Código:12'. The 'Variables' panel shows 'Local' variables: 'datos: {...}' with properties: 'nevera: "neveraChachones"', 'producto: "huevos"', 'cantidad: 10', 'pesos: 30000'.

Al corregir el error de las comillas, se puede observar que en la variable local datos, ya muestra lo que contiene dicha variable.

### Paso 3. Explora el IDE de GAS. La ejecuciones

Piensa en lo siguiente: por ahora tu corres el script y sabes lo que ocurre usando comandos como `console.log()` o el debugger. El problema es que más adelante, buscaremos que ese script se convierta en una solución web, tendrá una URL que podrán usar miles de usuarios, igual que ocurre con una página web. Entonces, cómo sabrías tú, como desarrollador, qué ha ocurrido con las ejecuciones que han lanzado miles de usuarios?. La respuesta está en que el IDE ofrece una herramienta para revisar esas ejecuciones. Pruebala así: Ejecuta tu código (funcion pruebas) de manera normal > pasa luego a Menú vertical izquierdo >  > observa que hay datos del momento de la ejecución, su duración, su estado. En términos de cómputo en la nube la duración es muy importante porque implica el costo computacional. El estado indica si se ejecutó, si se canceló o si se presentó un error.

Entra a la última ejecución, observa que se guarda información de comandos de salida como `console.log()` o `Logger.log()`. Esto tomará mucha relevancia más adelante.

Abajo proporciona una captura de pantalla que muestre las ejecuciones realizadas



Implementación	Función	Tipo	Hora de Inicio	Duración	Estado
Principal	pruebas	Editor	19 ago 2024, 21:42:25	0.387 s	Completada

Registros de Cloud

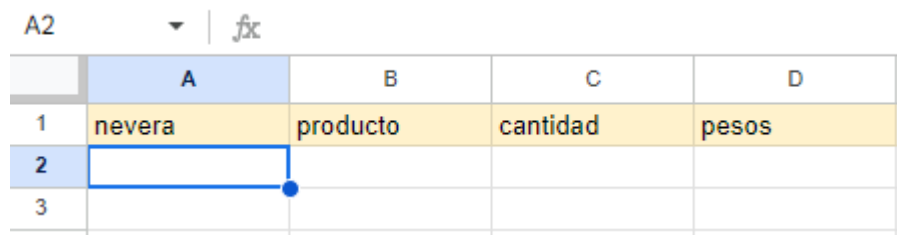
19 ago 2024, 21:42:26	Depuración	La nevera se llama: neveraChachones
19 ago 2024, 21:42:26	Depuración	La nevera requiere: 10 huevos

## Tarea 3. Completa el script para que sea capaz de registrar los datos de la nevera en un archivo de Google sheet.

El propósito no es solo escribir el script, sino también adquirir habilidades en el uso de Chat GPT para obtener código funcional.

**Paso 1.** Preparar el archivo de Google Sheet que servirá como base de datos.

- Ve a Google Drive y crea el archivo, asígnale un nombre.
- Crea una hoja en el archivo y dale un nombre, por ejemplo “RegistrosIoT”
- En esa hoja crea los encabezados para los datos que deseas que se registren. Nota: los encabezados deben tener nombres que coincidan con las claves de nuestros datos. Agrega dos columnas adicionales al inicio para ‘numero’ y ‘fecha’ ya que es importante que cada registro tenga un identificador único. Para nuestro caso la hoja quedaría como en la figura siguiente:



	A	B	C	D
1	nevera	producto	cantidad	pesos
2				
3				

Escribe abajo los datos siguientes

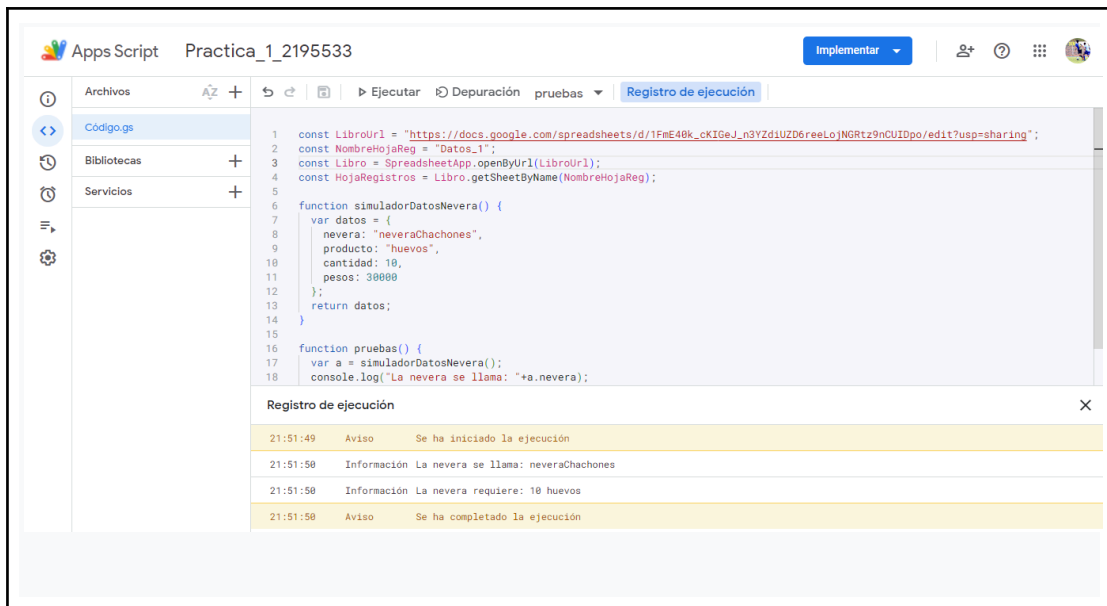
- La URL de tu archivo de google sheet:  
[https://docs.google.com/spreadsheets/d/1FmE40k\\_cKIGeJ\\_n3YZdiUZD6reeLojNGRt\\_z9nCUIDpo/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1FmE40k_cKIGeJ_n3YZdiUZD6reeLojNGRt_z9nCUIDpo/edit?usp=sharing)
- Nombre de la Hoja en el Google Sheet donde se registran los datos: Datos\_1

**Paso 2.** En nuestro script registrar la información que corresponde al archivo y hoja del paso anterior. Se usa la clase SpreadsheetApp

Agrega a tu script, al inicio del código, como variables globales, el siguiente código. Cambia “tu url aqui, entre comillas” por la URL de tu archivo, **entre comillas**; cambia “nombre de tu hoja aqui, entre comillas” por el nombre **entre comillas** de la hoja donde llegarán los datos.

```
const LibroUrl = tu url aqui, entre comillas;
const NombreHojaReg=nombre de tu hoja aqui, entre comillas;
const Libro=SpreadsheetApp.openByUrl(LibroUrl);
const HojaRegistros = Libro.getSheetByName(NombreHojaReg);
```

Comprueba que el código sigue corriendo sin errores. Al correr el código por primera vez, deberás aceptar los permisos que te pide. Registra aquí todo el código resultante:



**Paso 3.** Crea una función para registrar datos en tu Google Sheet. La función a crear tiene un valor especial - puede ser re usada en muchas aplicaciones, de manera que convendría guardarla por ahora en un lugar especial, por ejemplo en una carpeta llamada "FuturaBiblioteca".

Creamos un nuevo archivo que se llame "FuturaBiblioteca". Allí escribiremos la nueva función ya que es una función que tiene un valor especial, que conviene ser conservada porque puede ser re usada una y otra vez en múltiples futuras aplicaciones. Este es el proceso para crear el nuevo archivo

archivo así: Menú vertical panel izquierdo > Archivos > **+**.

En el ejemplo de abajo, el archivo para estas funciones se llama "FuturaBiblioteca" > pasa la función anterior al archivo "FuturaBiblioteca"

En el archivo "FuturaBiblioteca" escribimos el comentario sobre lo que hará la nueva función. Se debe seguir este formato

```
/** Aquí va la descripción */
```

Nota 1: si no se sigue ese formato para la descripción, el sistema de ayudas no podrá brindar ayudas sobre esa función.

Nota 2: la descripción debe ser muy buena, porque más abajo la entregaremos a Chat GPT para que nos dé el código

La siguiente es un ejemplo de una buena descripción:

```

/**
 * La función registroObjetoEnHoja(laHoja,losDatos) tiene las siguientes entradas:
 * laHoja: es el objeto de una hoja de Google SpreadSheetApp
 * losDatos: es un objeto de datos compuesto de pares clave:valor
 * La función se encarga de registrar cada "clave:valor" de la siguiente manera:
 * se registra el valor en una nueva fila de manera que coincida con la columna que
 * tiene como nombre, en la fila 1, la clave.
 */

```


Pero, en las descripciones conviene respetar las normas JSDoc (Javascript

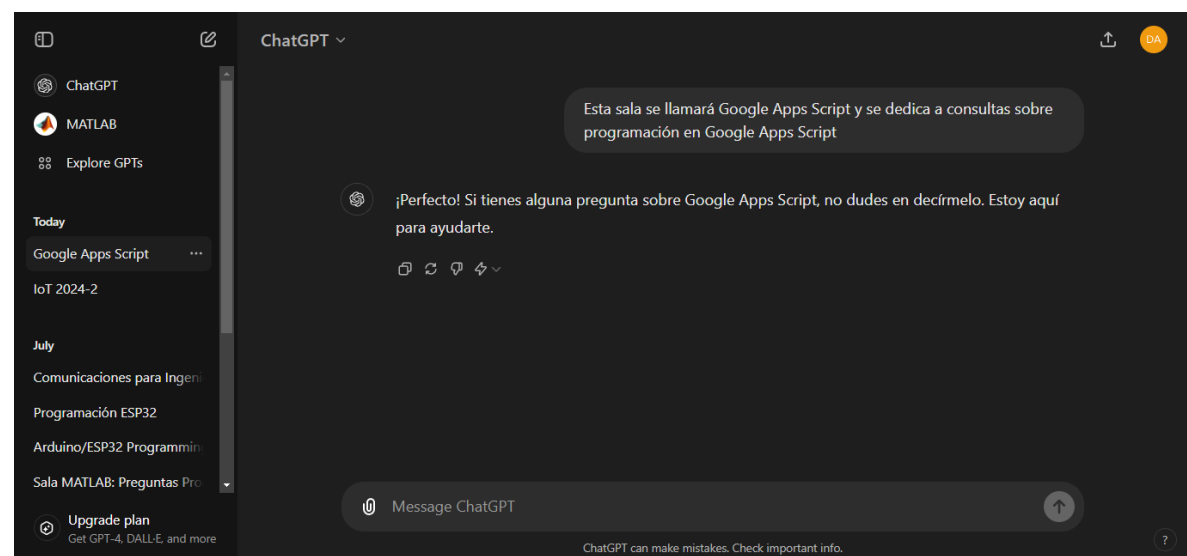


Documentation) para que el IDE pueda usar esa descripción cuando requiera brindar ayudas a un programador que quiera usar esa función. También es útil para que Chat GPT comprenda claramente nuestra intencionalidad para que nos brinde el código más apropiado. La siguiente es la descripción respetando esas normas.

```
/**
 * @function:registroObjetoEnHoja(laHoja,losDatos)
 * @description: Registra los datos en una hoja de Google Spreadsheet, colocando cada valor
 * en la columna que corresponde a su clave, teniendo en cuenta que la fila 1 está dedicada
 * a los encabezados, los cuales ya están registrados y son las mismas claves.
 * @param {GoogleAppsScript.Spreadsheet.Sheet} laHoja - El objeto de una hoja de Google
 * Spreadsheet.
 * @param {Object} losDatos - Un objeto de datos compuesto de pares clave:valor.
 */
```

Crea una sala de Chat GPT y asígnale el nombre “Google Apps Script”. Explica a Chat GPT que esa sala la usará única y exclusivamente para consultas relacionadas con desarrollo de código en Google Apps Script. Nota: debes cumplir esa promesa, no debes usar esa sala para otras cosas. El proceso para crear una sala es así: abres chat GPT >

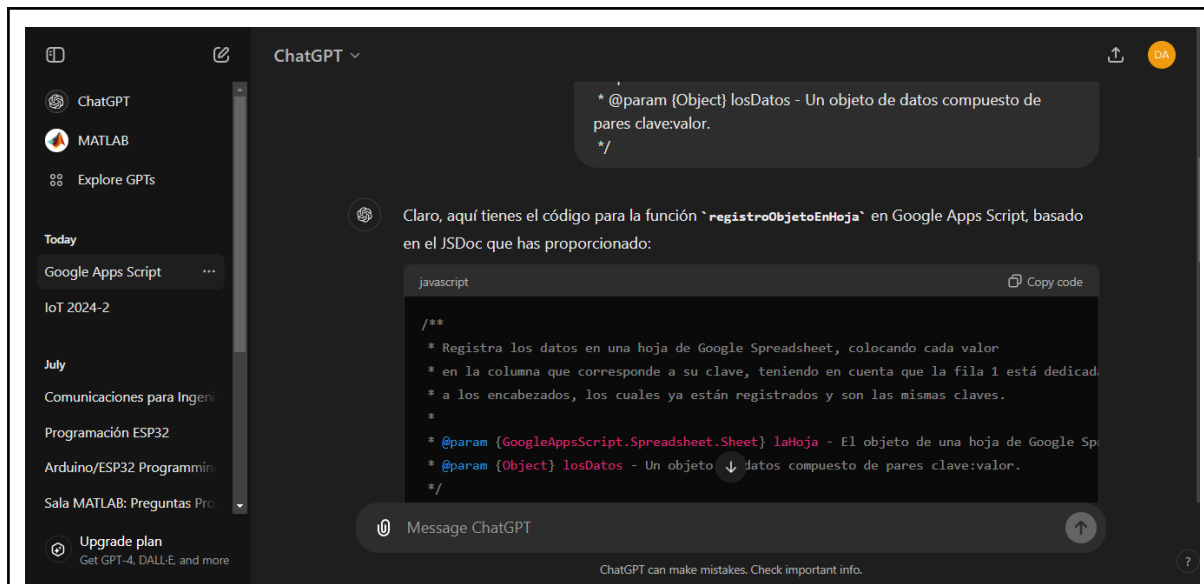
oprimes  ChatGPT > en el chat escribes “esta sala se llamará Google Apps Script y se dedica a consultas sobre programación sobre Google Apps Script”. Agrega abajo una captura de pantalla del resultado



Escribe la pregunta a Chat GPT siguiendo estos pasos:

- 1) Si tienes miedo de no haber escrito correctamente el JSDoc, puedes pedirle a Chat GPT que te lo corrija. Debes revisar bien que el Chat GPT no te desvíe de tus intenciones.
- 2) escribe la pregunta así: “Necesito el código para la función dada por el siguiente JSDoc:”
- 3) Pasa a otro renglón para que Chat GPT pueda diferenciar entre la pregunta y los valores. Para pasar a otro renglón oprime Shift+Enter
- 4) pega el contenido del JSDoc
- 5) oprime Enter

Agrega abajo una captura de pantalla del resultado



El código que ofrece Chat GPT lo pegamos en el archivo “FuturaBiblioteca”

Realiza las pruebas para comprobar que la función `registroObjetoEnHoja(laHoja, losDatos)` cumple lo que promete. Para ello debes completar el código creando una función `pruebas()` para que consuma la nueva función. Por ejemplo, el código completo en el archivo “Código.gs” puede ser este:

```
const LibroUrl = La url de tu google sheet;
const NombreHojaReg = nombre de la hoja para los registros;
const Libro = SpreadsheetApp.openByUrl(LibroUrl);
const HojaRegistros = Libro.getSheetByName(NombreHojaReg);

function simuladorDatosNevera() {
  var datos = {
    nevera: "neveraChachones",
    producto: "huevos",
    cantidad: 10,
    pesos: 30000
  };
  return datos;
}

function pruebas() {
  var a = simuladorDatosNevera();
  console.log("La nevera se llama: " + a.nevera);
  registroObjetoEnHoja(HojaRegistros, a);
}
```

Corre la función `pruebas()` y revisa tu archivo de google sheet para corroborar que los datos se registran correctamente. Registra aquí una captura de pantalla que evidencie que tuviste éxito en esta tarea.

Registros\_lot\_2195533 ☆ Guardado en Drive

Archivo Editar Ver Insertar Formato Datos Herramientas Extensiones Ayuda

100% 123 Predet... 10 B I

	A	B	C	D	E	F	G
1	numero	fecha	nevera	producto	cantidad	pesos	
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							

Así se observa sin que se registren los datos en el archivo de Google Sheets.

Apps Script Practica\_1\_2195533 Implementar

Archivos Código.gs Futura\_Biblioteca.gs Bibliotecas Servicios

```

1 const LibroUrl = "https://docs.google.com/spreadsheets/d/1FmE48k_cKIGeJ_n3Yzd1UZD6reeLoJNGRtz9nCUiDpo/edit?usp=sharing";
2 const NombreHojaReg = "Datos_1";
3 const Libro = SpreadsheetApp.openByUrl(LibroUrl);
4 const HojaRegistros = Libro.getSheetByName(NombreHojaReg);
5
6 function simuladorDatosNevera() {
7   var datos = {
8     nevera: "neveraChachones",
9     producto: "huevos",

```

Registro de ejecución

22:28:30	Aviso	Se ha iniciado la ejecución
22:28:31	Información	La nevera se llama: neveraChachones
22:28:31	Aviso	Se ha completado la ejecución

Registros\_lot\_2195533

Archivo Editar Ver Insertar Formato ...

100% 123 Predet... 10 B I

	A	B	C	D	E	F
1	numero	fecha	nevera	producto	cantidad	pesos
2			neveraChachones	huevos	10	30000
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						

Datos\_1

Finalmente se muestra el código ejecutado correctamente junto con el archivo en Google Sheets con los datos exitosamente registrados.

## Tarea 4. Usar lo aprendido para una solución más profesional de bases de datos.

El problema a resolver es que las bases de datos deben incluir una columna dedicada a un identificador único para cada registro. También podemos incluir otros parámetros como el momento en que se hizo el registro, como se muestra en la siguiente tabla

A2						
	A	B	C	D	E	F
1	numero	fecha	nevera	producto	cantidad	pesos
2						
3						

En este caso, al ID le hemos llamado “número”. El reto es adaptar la solución que se obtuvo en la tarea anterior para cumplir estos nuevos requerimientos. Eso implica:

- cambiar la tabla o crear una nueva con los nuevos encabezados
- complementar el JSON con los nuevos parámetros requeridos de ID y fecha. Eso implica pensar cómo obtener un ID que sea único para el registro

La estrategia que seguiremos en este caso puede no ser la mejor, pero ayuda a aprender nuevas cosas. Vamos a modificar la función `registroObjetoEnHoja(laHoja, losDatos)` para que sea ella la encargada de obtener tanto la ID como la fecha, pero también de insertar esos nuevos parámetros dentro del objeto de datos.

### Paso 1:

Prepara tu Google Sheet para que contenga las columnas “numero” y “fecha” como en la figura anterior

### Paso 2:

Se requiere una nueva función capaz de obtener la fecha y hora, así como el número consecutivo que le corresponde a un nuevo registro. Para ello deberá leer la hoja de registros para conocer cuál es el último número registrado. La siguiente puede ser la descripción para la nueva función.

Le diremos a Chat GPT lo siguiente:

Quiero mejorar el código de `registroObjetoEnHoja(laHoja, losDatos)` para atender las siguientes consideraciones:

- 1) debes crear una función `generarCodigoUnico()` capaz de devolver un número único apropiado para ser asociado un nuevo registro en una base de datos;
- 2) a “losDatos” se le insertará una nueva clave/valor de manera que la clave se llamará “numero” y el valor sea el que entrega la función `generarCodigoUnico()`
- 3) a “losDatos” se le insertará una segunda clave/valor, de manera que la clave se llamará “fecha” y el valor será la fecha del momento.
- 4) quiero que incluyas también en JSDoc para las funciones, considerando los cambios realizados

Copia y pega el código obtenido en el IDE de Google Apps Script y también en el siguiente espacio:

```
/**
 * Genera un código único basado en la marca de tiempo actual.
 *
 * @returns {number} - Un número único basado en la fecha y hora actual.
 */
function generarCodigoUnico() {
  return new Date().getTime(); // Usa la marca de tiempo actual como código único
}

/**
 * Registra los datos en una hoja de Google Spreadsheet, colocando cada valor
```

```

    * en la columna que corresponde a su clave, teniendo en cuenta que la fila 1
    está dedicada
    * a los encabezados, los cuales ya están registrados y son las mismas claves.
    * Además, agrega dos claves adicionales: "numero" con un valor único generado
    * y "fecha" con la fecha y hora actuales.
    *
    * @param {GoogleAppsScript.Spreadsheet.Sheet} laHoja - El objeto de una hoja de
    Google Spreadsheet.
    * @param {Object} losDatos - Un objeto de datos compuesto de pares clave:valor.
    */
function registroObjetoEnHoja(laHoja, losDatos) {
    // Genera un código único y la fecha actual
    const codigoUnico = generarCodigoUnico();
    const fechaActual = new Date().toISOString(); // Formato ISO para la fecha

    // Agrega los nuevos datos a losDatos
    losDatos['numero'] = codigoUnico;
    losDatos['fecha'] = fechaActual;

    // Obtiene el rango de la fila de encabezados
    const rangoEncabezados = laHoja.getRange(1, 1, 1, laHoja.getLastColumn());
    const encabezados = rangoEncabezados.getValues()[0];

    // Crea un array para almacenar los valores a insertar
    const valores = new Array(encabezados.length).fill('');

    // Itera sobre las claves de losDatos para ubicar los valores en las columnas
    correctas
    for (const clave in losDatos) {
        const valor = losDatos[clave];
        const indiceColumna = encabezados.indexOf(clave);

        // Solo si la clave está en los encabezados, coloca el valor en la columna
        correcta
        if (indiceColumna !== -1) {
            valores[indiceColumna] = valor;
        }
    }

    // Añade los valores a la siguiente fila vacía
    const ultimaFila = laHoja.getLastRow();
    laHoja.getRange(ultimaFila + 1, 1, 1, valores.length).setValues([valores]);
}

```

Ahora deberemos realizar pruebas para saber si la función anterior cumple lo prometido. Muestra abajo una captura de pantalla con los resultados obtenidos

Apps Script

Practica\_1\_2195533

Implementar

Archivos

Código.gs

Futura\_Biblioteca.gs

Bibliotecas

Servicios

Ejecutar

Depuración

pruebas

Registro de ejecución

```
6 function simuladorDatosNevera() {
7   var datos = {
8     nevera: "neveraChachones",
9     producto: "huevos",
10    cantidad: 10,
11    pesos: 30000
12  };
13  return datos;
14 }
15
16 function pruebas() {
17   var a = simuladorDatosNevera();
18   console.log("La nevera se llama: " + a.
19   nevera);
20   registroObjetoEnHoja(HojaRegistros, a);
21 }
```

Registro de ejecución

22:50:45

Aviso

Se ha iniciado la ejecución

22:50:46

Información

La nevera se llama: neveraChachones

22:50:46

Aviso

Se ha completado la ejecución

Registros\_lot\_2195533

Archivo

Editar

Ver

Insertar

Formato ...

100%

\$

%

0.00

123

Predet...

	A	B	C	D	
1	numero	fecha	nevera	producto	c
2			neveraChachones	huevos	
3	1724125620439		neveraChachones	huevos	
4	1724125846568	2024-08-20T03:50:46.568Z	neveraChachones	huevos	
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

Datos\_1