詳解! Google Apps Script完全入門 第2版と第3版の差分PDF

2020年12月に「詳解! Google Apps Script完全入門 [第2版]」が発売されました。しかし、その前後に Google Apps Script や Google Workspace に関していくつかのアップデートがあり、実際の環境および操作方法と、書籍のものとの間に、いくつかの差分が生まれてしまいました。

すぐに、出版社さまと相談させていただき、書籍については2021年7月に「詳解! Google Apps Script完全入門 [第3版]」を出版することで、その差分を埋めております。

しかし、「詳解! Google Apps Script完全入門 [第2版]」をお買い求めのみなさまにには、引き続き不都合をおかけしてしまう形になってしまいます。微力ではありますが、その不都合を補うべく、第3版で大きく書き直したパートについてPDFとして配布させていただくこととしました。

本PDFは、[第3版] の2章、20章、23章についてまとめたものです。

なお、ご利用の際には、以下についてご了承くださいませ。

- [第2版] とともにご活用いただくことを前提としております。
- [第2版] については2章、20章、23章以外の章についても、現行といくつかの差分がございますが、それについてすべて網羅するものではないことをご了承ください。
- 出版社さまのご了承を得て、著者が独自に作成しているものとなりますため、出版社さまには本PDFについてのお問い合わせはしないようにお願いします。

また、サンプルコードは以下出版社さまのページの「サポート」にてダウンロードが可能になる予定です。

```
https://www.shuwasystem.co.jp/support/7980html/6474.html
```

それでは、どうぞご活用いただければと思います。

2. スクリプトエディタとダッシュボード

2.1 はじめての GAS

◆スプレッドシートからスクリプトエディタを開く

スクリプトエディタは、GAS のスクリプトの編集、実行、デバッグなどを行うための専用のエディタで、メニューやツールバーをはじめ、開発を支援するための便利な機能が搭載されています。スクリプトエディタがあるおかげで、私たちは面倒な準備をすることなく、すぐに GAS をはじめることができます。

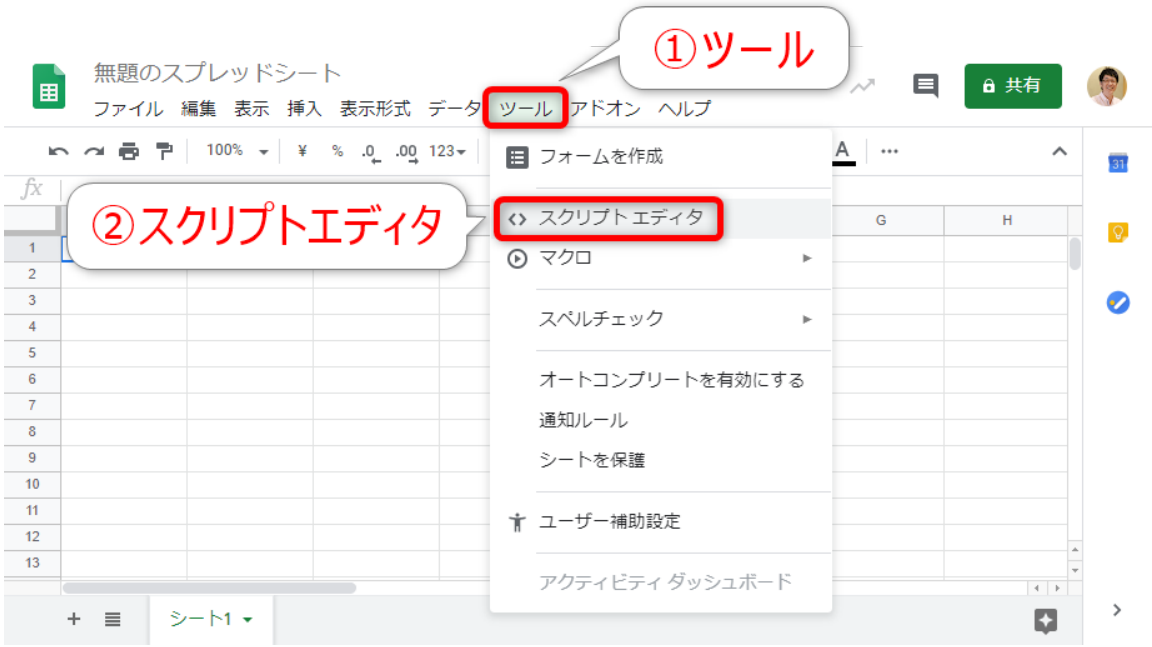
では、実際にスクリプトエディタを起動して、GAS のスクリプトの作成や実行など、GAS 開発の一連の流れを確認してみましょう。

まず、Google ドライブから新規のスプレッドシートを作成します。Google ドライブでフォルダを開き、右クリックメニューから「Google スプレッドシート」を選択します（図 2-1-1）。左上の「＋新規」ボタン→「Google スプレッドシート」でも同様のことが行なえます。

▶図 2-1-1　Google ドライブから新規スプレッドシートを作成

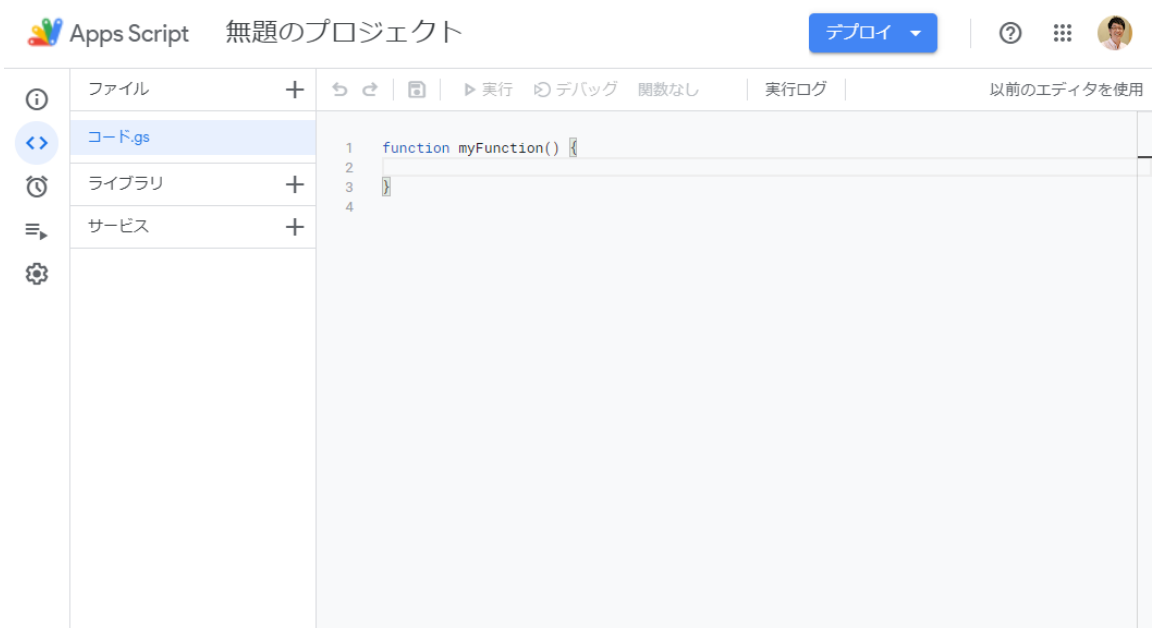
ブラウザの新規タブで、作成した「無題のスプレッドシート」が開きます。次に、スプレッドシートのメニューから、「ツール」→「スクリプトエディタ」を選択します（図 2-1-2）。

▶図 2-1-2　スプレッドシートからスクリプトエディタを開く



すると、ブラウザの新規タブでスクリプトエディタが起動し、「無題のプロジェクト」が開きます (図 2-1-3)。この画面でスクリプトの編集や実行、デバッグを行っていきます。

▶図 2-1-3 スクリプトエディタ



◆スクリプトを編集する

　では、続けてスクリプトの編集をしていきましょう。現時点では、スクリプトエディタに以下コードが記述されているはずです。

▶サンプル 2-1-1 無題のプロジェクトのコード

```
function myFunction() {

}
```

　「function myFunction() {」の行と、「}」の行の間に 1 行空いていますね、そこにコードを入力して、以下のサンプル 2-1-2 を完成させていきます。

▶サンプル 2-1-2 スクリプトの入力

```
function myFunction() {
  Browser.msgBox('Hello');
}
```

　コードの入力は難しいように思われるかも知れませんが、スクリプトエディタの機能を用いれば簡単に入力を進めることができますので、一緒に進めていきましょう。

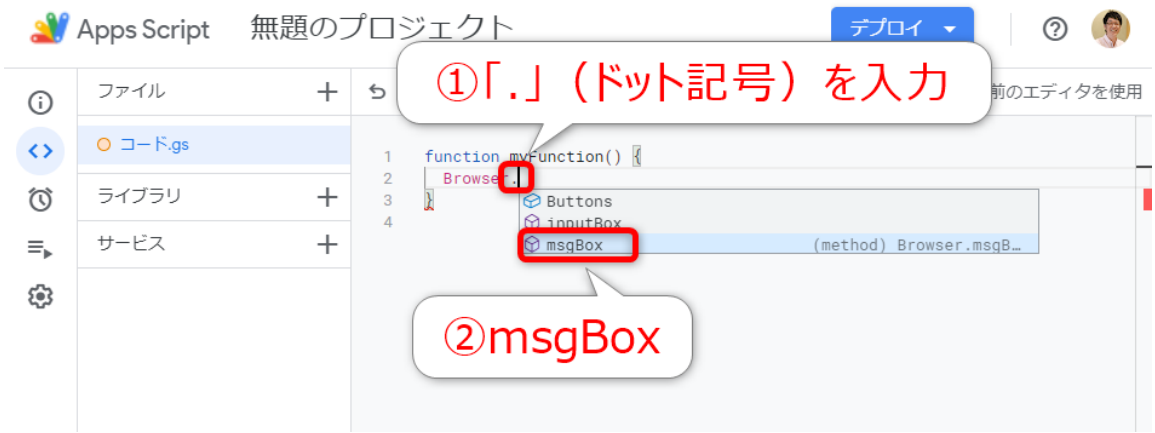
　まず、2 行目のいずれかの箇所をクリックすると、自動的に半角スペース 2 文字分だけ字下げした位置にカーソルが置かれるのに気づくはずです。この字下げは「インデント」といい、コードの読みやすさにとって重要な役割を果たします。そのまま字下げした位置からコードを書き始めてください。

　では、半角アルファベットの「br」と入力してみましょう。すると、図 2-1-4 のように、入力候補が一覧表示されます。[↑][↓] キーで選択項目を移動できますので、「Browser」にあわせて [Tab] キーを押して確定しましょう。

▶図 2-1-4 「Browser」の入力

続いて、ドット「.」を入力すると、再度その後の候補が一覧表示されます。「msgBox」を選択して確定させましょう。

▶図 2-1-5 「.msgBox」の入力



その後、開く丸括弧「(」や、シングルクォーテーション「'」を入力すると、対になる記号が自動で入力されたりしますので活用しましょう。また、「msgBox」を説明するボックスが自動表示されますが、[Esc] キーで消すことができます。

最終的に、サンプル 2-1-2 と同じになるように進めていきましょう。なお、行最後のセミコロン「;」忘れや、記号の打ち間違いに注意してください。

◆スクリプトを保存する

スクリプトに変更を加えると、「コード .gs」の前に黄色の丸アイコンが付与されることにお気づきでしょうか。

スプレッドシートやドキュメントなど多くの Google アプリケーションでは、クラウド上に自動保存されることが多いのですが、**スクリプトエディタのスクリプトは自動保存がなされません。**

ですから、スクリプトに変更を加えたのであれば、都度手動で保存をする必要があります。

保存の方法は、図 2-1-6 に示すツールバーの「プロジェクトを保存」アイコンをクリックするか、またはショートカットキー [Ctrl] + [S] ／ [⌘] + [S] を使用します。

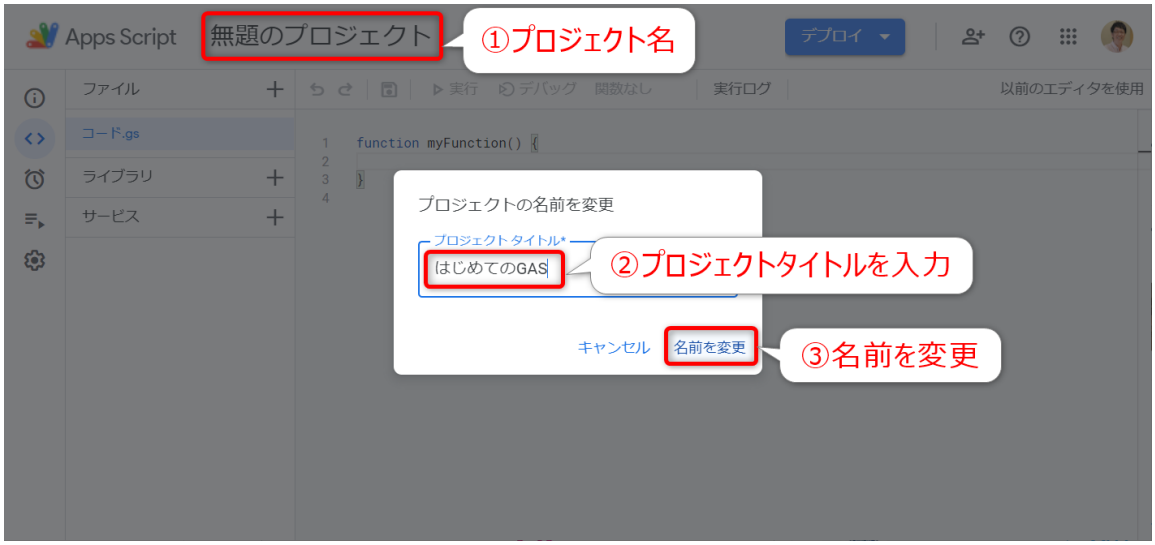
▶図 2-1-6 スクリプトの保存

すると、黄色の丸アイコンが消えます。つまり、保存されたということです。

さて、新規作成したプロジェクトはすべて「無題のプロジェクト」という名称になってしまうので、プロジェクト名を変更しておいたほうがよいでしょう。

プロジェクト名をクリックすることで、プロジェクトの名前を変更するダイアログが開きますので、プロジェクトタイトルを入力して、「名前を変更」をクリックします（図 2-1-7）。

▶図 2-1-7 プロジェクト名の編集



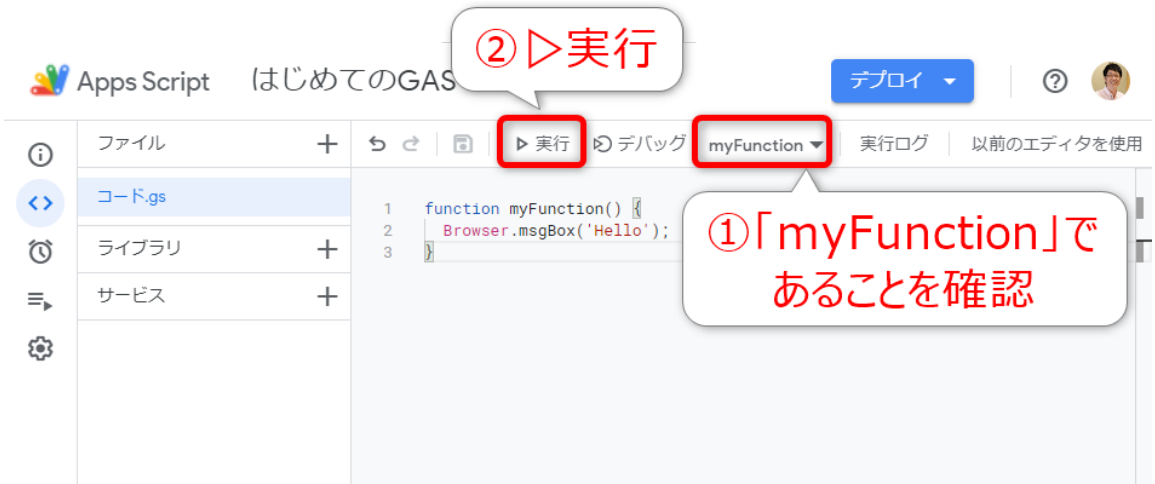
スプレッドシートから作成、保存をしたスクリプトは、再度スプレッドシートのメニュー「ツール」→「スクリプトエディタ」から開くことで、編集を再開できます。

◆スクリプトを実行する

では、完成したスクリプトを実行してみましょう。

ツールバーに「myFunction」が表示されているのを確認した上で、ツールバーの「▷実行」をクリックするか、ショートカットキー [Ctrl] + [R] ／ [⌘] + [R] を押下します（図 2-1-8）。

▶図 2-1-8 スクリプトの実行



【Memo】
「myFunction」は関数名といい、関数はスクリプトの実行単位です。詳しくは第 3 章および第 5 章で解説をしますのでご覧ください。

初回の実行の際は、図 2-1-9 のように「承認が必要です」というダイアログが表示されます。

これは GAS ならではの手順ともいえるものですが、スクリプトに対して、操作対象となるスプレッドシートのアクセスを許可する必要があるのです。

▶図 2-1-9 「承認が必要です」ダイアログ

「承認が必要です」ダイアログで「権限を確認」をクリックし、続く画面でアカウントを選択します（図 2-1-10）。

図 2-1-10 アカウントを選択

◆アクセス許可の手順

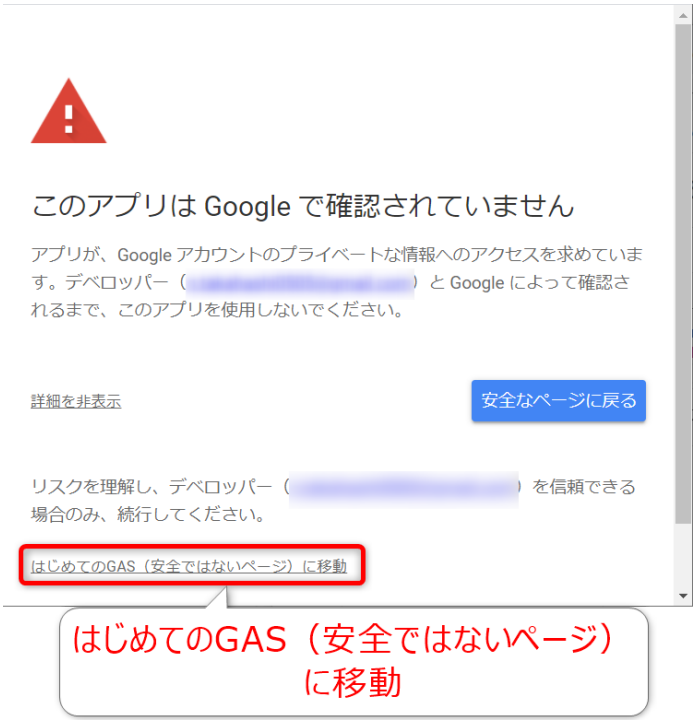
以降の手順は使用しているアカウントが Google Workspace アカウントかどうかで変わってきます。無料の google.com アカウントの場合は、図 2-1-11 に示す「このアプリは確認されていません」という画面が表示されます。Google Workspace アカウントの場合は、この画面はスキップされますので、次の手順として図 2-1-13 へ進んでください。

「このアプリは確認されていません」の画面では「詳細」をクリックしてください。「安全なページに戻る」をクリックすると、アクセス許可されずに戻ってしまいます。

▶図 2-1-11 「このアプリは確認されていません」ダイアログ

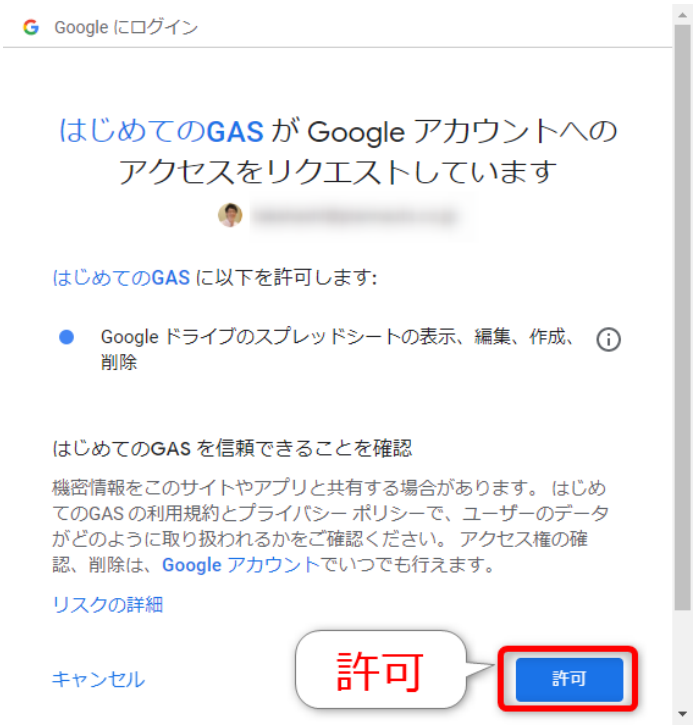
　すると、図 2-1-12 のように画面下部に「はじめての GAS（安全ではないページ）に移動」というリンクが表示されますのでクリックします。

▶図 2-1-12 安全ではないページに移動



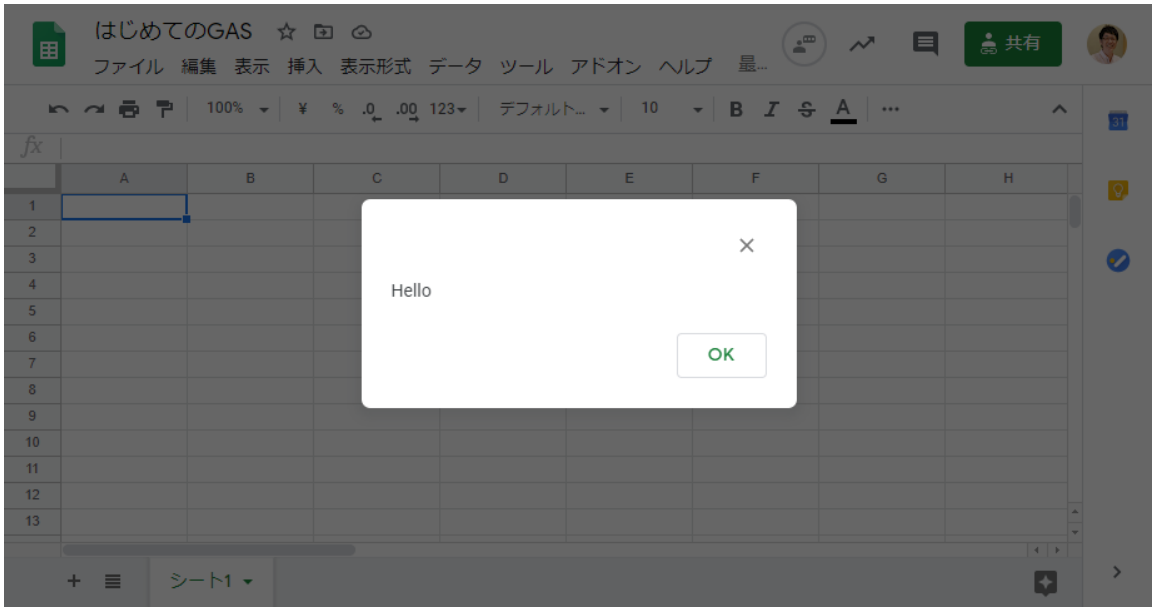
　続いて、図 2-1-13 のように「はじめての GAS が Google アカウントへのアクセスをリクエストしています」という画面に移りますので、「許可」をクリックしてアクセスを許可しましょう。

▶図 2-1-13 スクリプトにアクセスを許可



許可が完了すると、スクリプトの実行がなされます。スクリプトエディタでは何も起きていないように見えますが、実はちゃんと動いています。スプレッドシートに画面を切り替えると、図 2-1-14 のように「Hello」と表示されたメッセージボックスが表示されているはずです。

▶図 2-1-14 メッセージボックスの表示



これでスクリプトエディタの起動、スクリプトの編集、保存、実行と、GAS の一連の流れを確認することができました。しかし、スクリプトエディタには、ここでお伝えした以外にも、GAS の開発や習得に便利な機能がたくさん搭載されています。

以降の節で、実際にそれらスクリプトエディタの機能に触れながら、GAS の仕組みの理解を深めていくことにしましょう。

2.2 プロジェクトとスクリプト

◆プロジェクトとファイル

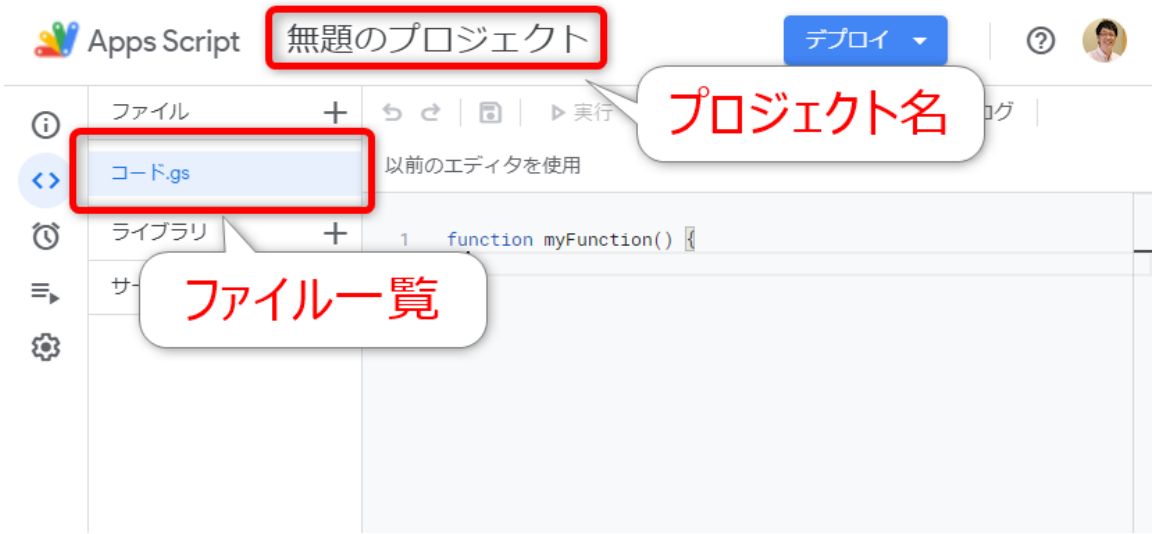
GAS のスクリプトは、**プロジェクト**という単位で作成します。プロジェクトは単一または複数のファイルで構成されており、GAS のスクリプトは拡張子「.gs」のファイルに記述をします (図 2-2-1）。また、プロジェクト内に「.html」ファイルを作成することもできます。

▶図 2-2-1 プロジェクトとファイル

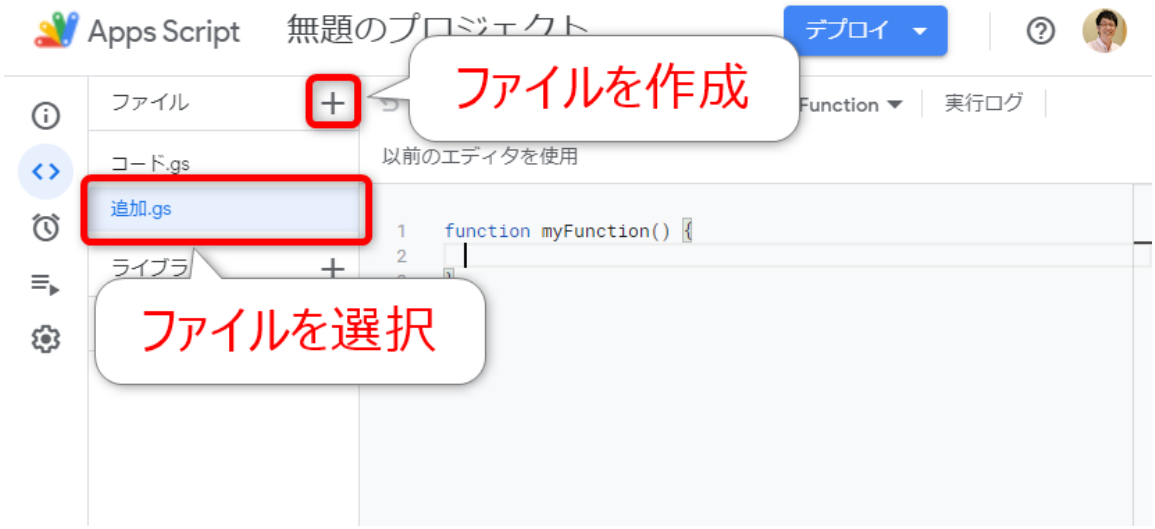
図 2-2-2 は、新規作成した時点のプロジェクトをスクリプトエディタで開いた画面です。「無題のプロジェクト」がプロジェクト名、プロジェクトは「コード .gs」という 1 つのファイルで構成されています。

スクリプトエディタメニューの「ファイル」の「＋」アイコンから新たなスクリプトファイル（.gs）および HTML ファイル（.html）を作成できます。スクリプトファイルを新規作成すると、ファイル一覧にファイルが追加されます。ファイル一覧でファイルをクリックすると、その内容が入力エリアに表示され、編集ができるようになります（図 2-2-3）。

▶図 2-2-2 新規作成した時点のスクリプト



▶図 2-2-3 複数のファイルで構成されるプロジェクト



◆スタンドアロンスクリプトとコンテナバインドスクリプト

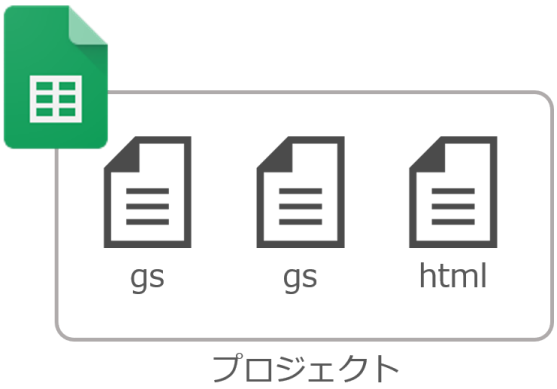
GAS のスクリプトには 2 種類あり、どちらの種類を使うかで作成および保存の方法が異な ります。

1 つは、**スタンドアロンスクリプト**といい、Google ドライブにプロジェクトファイル自体を直接保存します（図 2-2-4）。こちらは、プロジェクトファイル 1 つに対し、1 つのプロジェクトしか持つことができません。

▶図 2-2-4 スタンドアロンスクリプト

もう 1 つはスプレッドシート、ドキュメント、フォームおよびスライドといった、親ファイル（**コンテナ**といいます）に紐づく形で保存するコンテナバインドスクリプトです（図2-2-5）。なお、2-1 で作成および実行をしたスクリプトは、スプレッドシートに紐づいている**コンテナバインドスクリプト**です。

▶図 2-2-5 コンテナバインドスクリプト



【Memo】
コンテナバインドスクリプトは、スプレッドシート、ドキュメント、フォームおよびスライドで作成できます。

両者の比較について、図 2-2-6 にまとめています。ここに記載した内容以外の点では、どちらも同じ動作をします。

▶図 2-2-6 スタンドアロンとコンテナバインド

	スタンドアロンスクリプト	コンテナバインドスクリプト
スクリプトの保存	単体のスクリプトプロジェクトファイルとして保存	スプレッドシート、ドキュメント、フォーム、スライドに紐づいて保存
スクリプトの作成	Apps Sciprtタッシュボードまたは Googleドライブから作成	スプレッドシート、ドキュメント、フォーム、スライドのメニューから作成
スクリプトのオープン	Apps Scriptダッシュボードまたは Googleドライブからプロジェクトファイルを開く	スプレッドシート、ドキュメント、フォーム、スライドのメニューから開く
メソッド	—	紐づいたファイルに対するいくつかのメソッドが利用可能
トリガー	時間主導型、HTTPリクエスト、カレンダーからのトリガーを設置可能	時間主導型、HTTPリクエスト、カレンダーからのトリガーに加えてコンテナに関するトリガーを設置可能
UI	ー	コンテナのUIのカスタマイズが可能
スプレッドシートのカスタム関数	ー	作成可能
ライブラリの作成	作成可能	ー

コンテナバインドスクリプトでは、スプレッドシートのカスタム関数の作成ができる、また紐づいたファイルを参照する getActiveSpreadsheet、getActiveDocument、getUi などの、いくつかの便利なメソッドを使用することができるといったメリットがあります。

【Memo】
基本的に、GAS によるツールを作成するとき、コンテナバインドスクリプトをメインに使用していくという方針でよいでしょう。しかし、ライブラリの作成など、コンテナバインドスクリプトでは都合がよくない場合もありますので、その際にスタンドアロンを使用するケースが出てきます。

◆Apps Script ダッシュボードとは

コンテナバインドスクリプトはコンテナに紐付いて保存されていますが、Google ドライブの表面上からはスクリプトが存在してるかどうか、またそのプロジェクト名は何かとい

ったことを把握することができません。

そこで、登場するのが **Apps Script ダッシュボード**です。Apps Script ダッシュボードは、GAS のプロジェクトを管理およびモニタリングするためのツールで、以下のようなことを行うことができます。

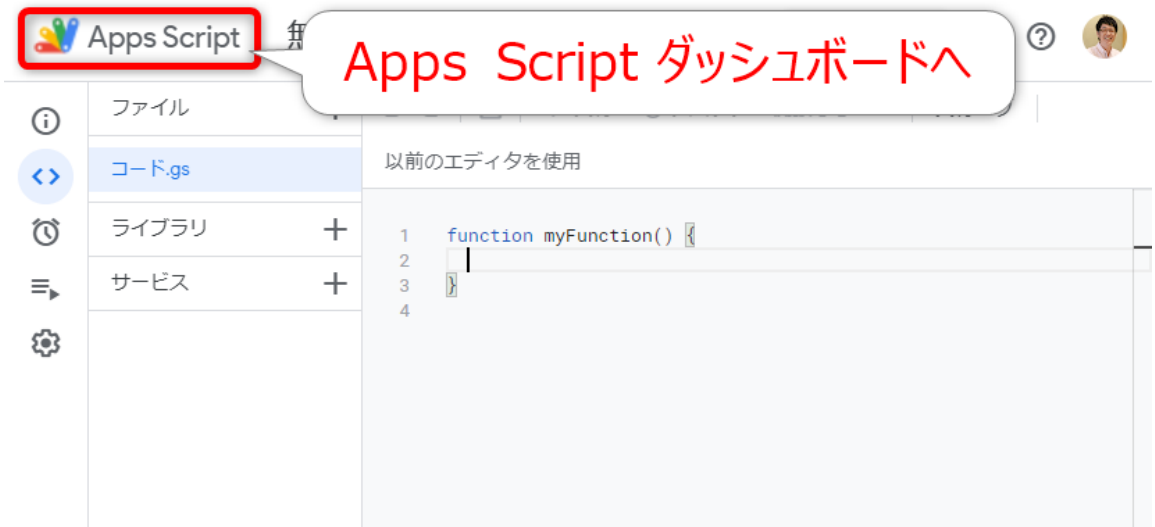
- プロジェクトの閲覧および検索、操作
- スクリプトの使用状況のモニタリング
- インストーラブルトリガーの作成、管理
- スタンドアロンスクリプトの作成
- ログの表示と保管

> 【Memo】
> 上記のうち、個別のプロジェクトに対する使用状況のモニタリング、ログの表示と保管、インストーラブルトリガーの作成、管理であれば、スクリプトエディタからも行うことができます。

コンテナバインドスクリプトもスタンドアロンスクリプトもどちらも合わせて管理できますので、直接的にプロジェクトを探して開いたり、確認したりするときに便利です。

Apps Script ダッシュボードは、図 2-2-7 のようにスクリプトエディタの左上のアイコンからアクセスできます。

▶図 2-2-7 Apps Script ダッシュボードを開く



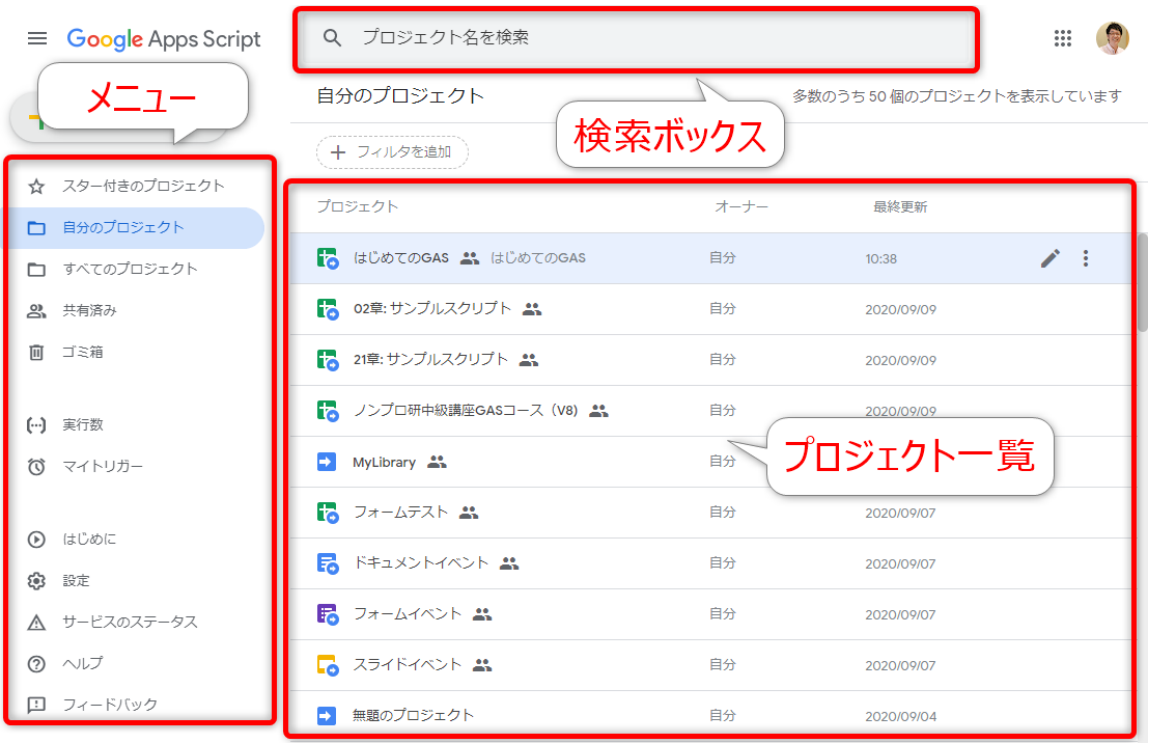
または、以下の URL で直接開くこともできますので、ブックマークしておくとよいでしょう。

```
https://script.google.com/
```

図 2-2-8 が Apps Script ダッシュボードの画面です。

左側のメニューで表示を切り替えたり、上部の検索ボックスでプロジェクトの検索を行うことができます。中央のプロジェクトのリストにはプロジェクトの一覧が表示されていて、編集アイコンでプロジェクトを開いたり、三点リーダーアイコンのメニューからスターを付ける、削除するといった操作を行うことができます。

▶図 2-2-8 Apps Script ダッシュボードの画面構成



プロジェクトをクリックすると、図 2-2-9 のプロジェクトの詳細画面が表示されます。ここで、スクリプトの過去 7 日間の実行状況を確認できます。右側の「プロジェクトの詳細」ではプロジェクトの情報を確認するとともに、プロジェクトやコンテナを開くことも可能です。

▶図 2-2-9 プロジェクトの詳細

このように、Apps Script ダッシュボードは GAS のプロジェクトを管理する上で、強力なツールになりますので、ぜひご活用ください。

◆スタンドアロンスクリプトの作成と開き方

スタンドアロンスクリプトを作成する場合は、Apps Script ダッシュボードから作成するのが便利です。Apps Script ダッシュボードの左上の「+ 新しいプロジェクト」ボタンをクリックします（図 2-2-10）。

▶図 2-2-10 新しいプロジェクトを作成する

これだけでスタンドアロンスクリプトを作成できます。別のタブで、図 2-2-11 のようにスクリプトエディタが開きますので、すぐに編集を開始できます。

▶図 2-2-11 作成したスタンドアロンスクリプト

作成したプロジェクトは、図 2-2-12 のように Apps Script ダッシュボードの一覧に追加されますので、クリックすることで再度開くことができます。

▶図 2-2-12 スタンドアロンスクリプトを開く

なお、このように作成したスタンドアロンスクリプトは、Google ドライブ上ではマイドライブに作成されます。必要に応じて、適切なフォルダに移動してください。

2.3. スクリプトエディタの編集機能

◆候補の表示

GAS で使用するオブジェクトやメソッドには、「つづり」の長いワードも多く存在しています。それらのワードを手打ちで入力することはタイプミスにつながりますし、そもそも、ワードのつづりを覚えることが難しいでしょう。

また、GAS ではアルファベットの大文字と小文字が区別されますので、「Browser.msgBox('Hello')」を入力すべきときに、「browser」や「msgbox」などとするだけでエラーとなります。

【Point】
GAS では、大文字と小文字を明確に区別するので注意が必要です。

しかし、そのような GAS での入力を支援するために、スクリプトエディタには**候補の表示**という機能があります。この機能は、2-1 ですでに体験いただいていますが、再度おさらいをしておきましょう。

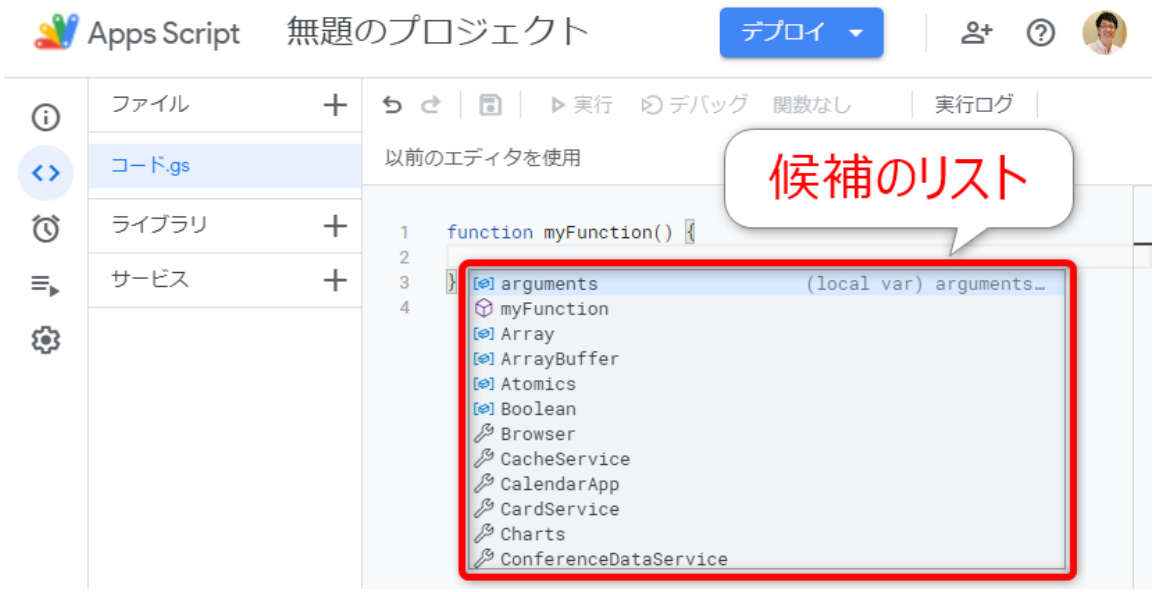
スクリプトエディタを開き、空白行で [Ctrl] + [Space] キーを入力すると、図 2-3-1 のように候補のリストが表示されますね。

【Memo】
Mac や Chromebook をお使いの場合は [Ctrl] + [Space] キーが他に割り当てられていて候補の表示が使えないときがあります。そのような場合は、[Ctrl] + [Space] キーを長押ししたり、以下のように [fn] や [option] および [Alt] などと組み合わせると機能することが多いので試してみてください。

- Mac: [fn] + [control] + [space] ／ [control] + [⌘] + [space] ／ [fn] + [control] + [⌘] + [space]
- Chromebook: [Ctrl] + [Alt] + [Space]

ここに表示されている候補は、トップレベルオブジェクトや予約語などとよばれるものです。GAS のコードはこれらのキーワードのいずれかを使って書き始めることになります。[↑][↓] キーで選択して、 [Tab] キーまたは [Enter] キーで入力を確定できます。もちろん、アルファベットの大文字、小文字も正しく入力がされます。

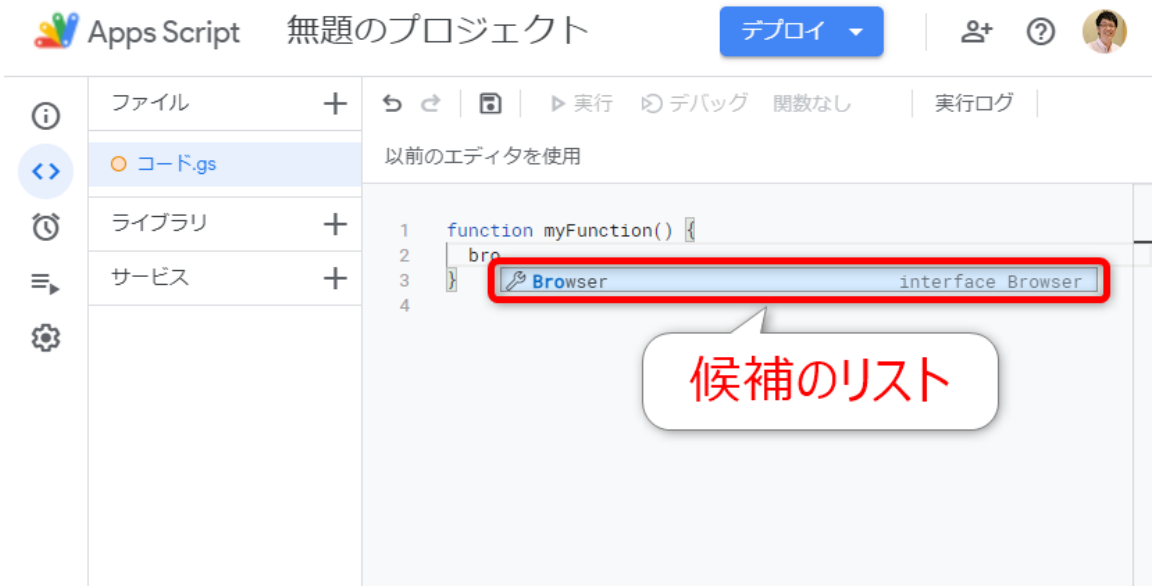
▶図 2-3-1 候補の表示



【Memo】
これを機に、トップレベルオブジェクトや予約語にひととおり目を通しておくのもよいでしょう。GAS の全体像を垣間見ることができます。

2-1 で体験いただいたとおり、候補は文字を入力するだけで自動表示されます。さらに、文字入力を進めていくと、図 2-3-2 のように候補が絞り込まれていきます。

▶図 2-3-2 候補の絞り込み



また、すでに定義されている変数・定数名や関数名も候補として表示されます。たとえば、図 2-3-3 の状態で「n」と入力すると、その候補として「numA」や「numB」が含まれていることが確認できます。

▶図 2-3-3 変数名の候補表示

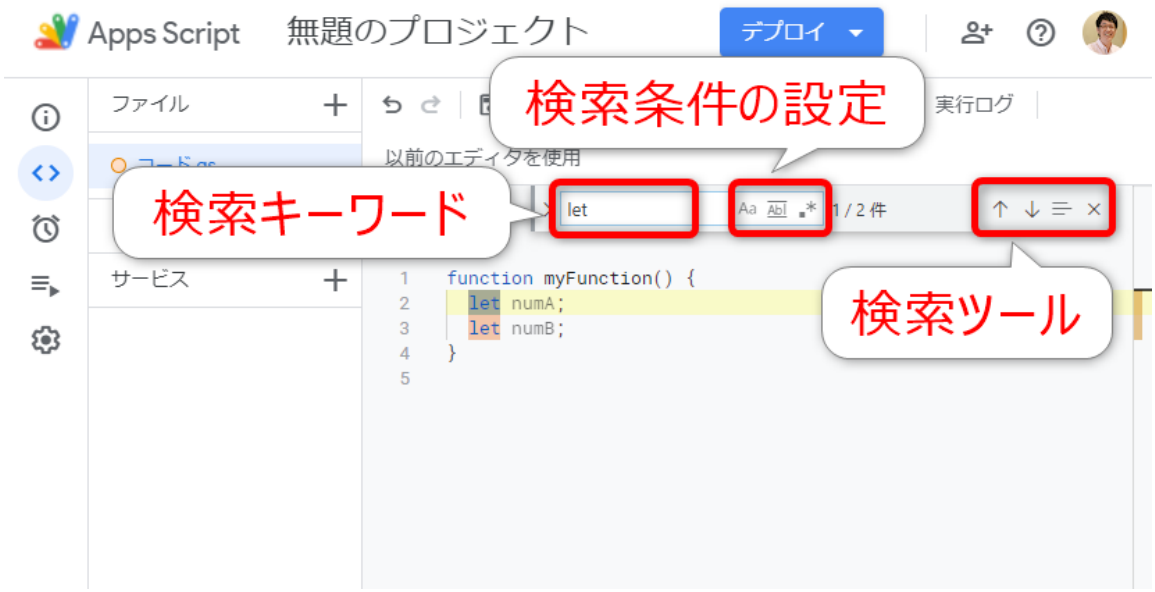
【Memo】
変数、定数については第 3 章で、関数については第 5 章で紹介します。

【Point】
候補の表示は、GAS の入力作業の強い味方になりますので、確実に使えるようにしておきましょう。

◆検索と置換

スクリプトの編集時に、[Ctrl] + [F] ／ [⌘] + [F] キーで、検索バーを表示できます。この検索バーの入力欄にキーワードを入力することで検索を行うことができます（図 2-3-4）。検索にマッチした箇所がオレンジ色に、その行が黄色にハイライトされます。複数の検索結果がある場合は、[Enter] キーで次の検索結果に、 [Shift] + [Enter] キーで前の検索結果に移動できます。

▶図 2-3-4 検索とその機能



検索バーの右側に配置する 4 つのアイコンは、左から順番に、検索に関する以下のような操作を行うものです。

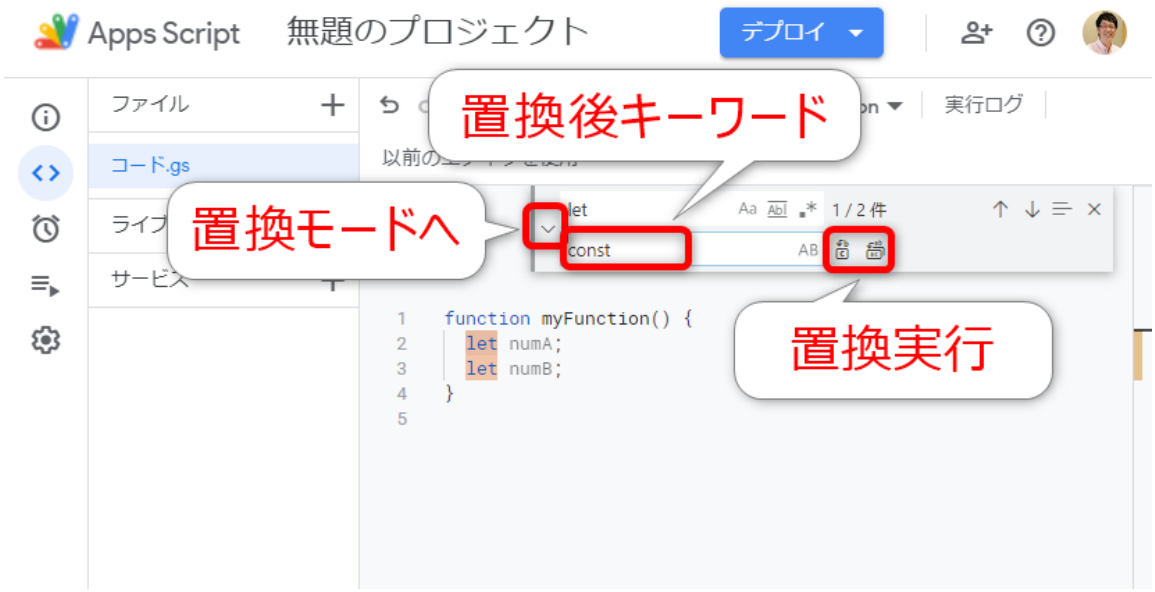
- 前の検索結果
- 次の検索結果
- 選択範囲を検索
- 閉じる

また、キーワード入力欄内の右側の 3 つのアイコンは、検索条件を設定するもので、左から順番に以下のような設定の切り替えを行います。

- 大文字と小文字を区別する
- 単語単位で検索する
- 正規表現を使用する

検索バーの表示中に「>」アイコンをクリックすると、検索バーが置換モードに切り替わります（図 2-3-5）。または、検索バーの非表示時も含めて [Ctrl] + [H] ／ [⌘] + [H] キーで置換の機能を呼び出すことができます。

▶図 2-3-5 置換とその機能



下の入力欄に置換後のキーワードを入力して、[Enter] キーで現在の検索結果を置換、[Ctrl]+ [Alt] + [Enter] ／ [⌘] + [Enter] キーですべての検索結果を置換します。置換の操作は、置換後キーワードの入力欄の右側のアイコンでも実行可能です。

◆「元に戻す」と「やり直す」

スクリプトエディタでは、たとえ直前の動作の後にファイルの保存をしてしまったとしても、直前の操作を取り消し、元の状態に戻すことができます。その方法は、ツールバーの「**元に戻す**」ボタン、またはショートカットキー [Ctrl] + [Z] ／ [⌘] + [Z] です。

また、ツールバーの「**やり直す**」ボタン、またはショートカットキー [Ctrl] + [Y] ／ [⌘] + [Y] で、元に戻した操作をやり直すこともできます。それぞれのアイコンの位置は、図 2-3-6 をご覧ください。

▶図 2-3-6 「元に戻す」と「やり直す」

◆スクリプトエディタのショートカットキー

スクリプトエディタには、これまで紹介したものも含めて便利なショートカットキーが用意されていますので、図 2-3-7 にまとめています。いずれも GAS の開発効率を上げる有用なものばかりですので、使いこなせるようにしておくとよいでしょう。

▶図 2-3-7 スクリプトエディタのショートカットキー

メニュー	操作	Windows	Mac
ファイル	ファイルを保存	[Ctrl] + [S]	[⌘] + [S]
編集	元に戻す	[Ctrl] + [Z]	[⌘] + [Z]
編集	やり直す	[Ctrl] + [Y]	[⌘] + [Y]
編集	検索	[Ctrl] + [F]	[⌘] + [F]
編集	置換	[Ctrl] + [H]	[⌘] + [H]
編集	すべての出現箇所を変更	[Ctrl] + [F2]	[shift] + [⌘] + [L]
実行	選択している関数を実行	[Ctrl] + [R]	[⌘] + [R]
移動・削除・コピー	単語単位で移動	[Ctrl] + [←][→]	[option] + [←][→]

メニュー	操作	Windows	Mac
移動・削除・コピー	定義へ移動	[Ctrl] + [F12]	[⌘] + [F12]
移動・削除・コピー	行の移動	[Alt] + [↑][↓]	[option] + [↑][↓]
移動・削除・コピー	行を削除	[Ctrl] + [Shift] + [K]	[shift] + [⌘] + [K]
移動・削除・コピー	行のコピー	[Shift] + [Alt] + [↑][↓]	[shift] + [option] + [↑][↓]
表示	候補をトリガー	[Ctrl] + [Space]	[control] + [space]
表示	コマンドパレットの表示	[F1]	[F1]
表示	コンテキストメニューの表示	[Shift] + [F10]	[shift] + [F10]
整形・コメント	インデントをする	[Tab]	[tab]
整形・コメント	ドキュメントのフォーマット	[Shift] + [Alt] + [F]	[shift] + [option] + [F]
整形・コメント	行コメントの切り替え	[Ctrl] + [/]	[⌘] + [/]
整形・コメント	ブロックコメントの切り替え	[Shift] + [Alt] + [A]	[shift] + [option] + 「A]

2.4. ログとデバッグ

◆ログの出力

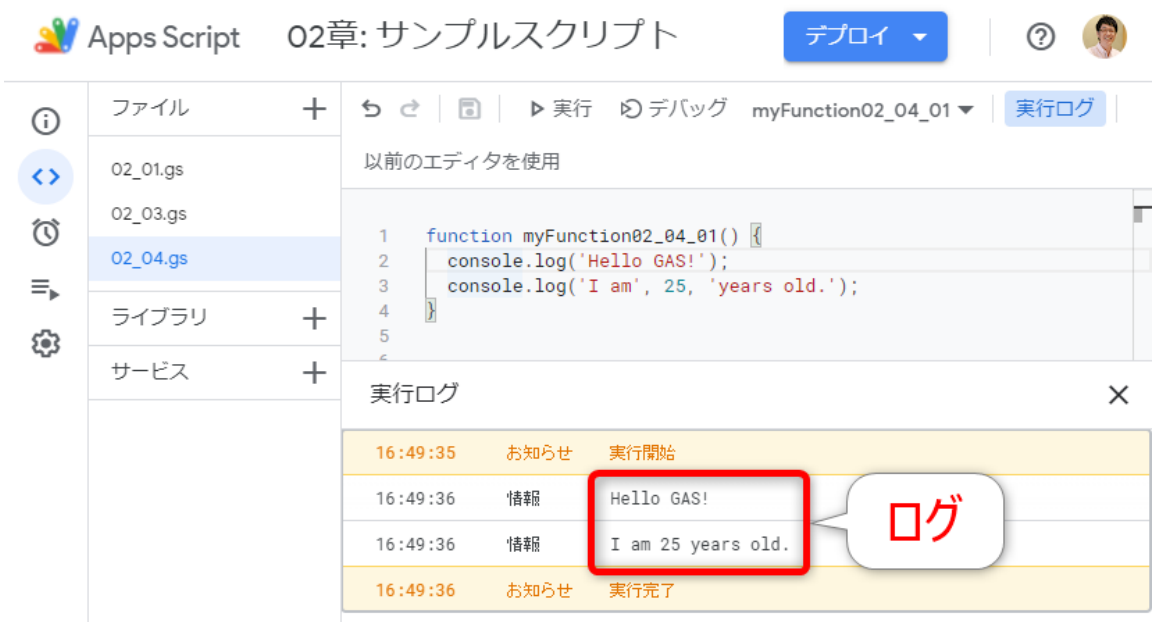
GAS では、スクリプトの動作を確認するため**ログ**を出力する機能が用意されています。以下のサンプル 2-4-1 を入力して、実行をしてみてください。

▶サンプル 2-4-1 ログ出力するスクリプト [sample02-04.gs]

```
function myFunction02_04_01() {
  console.log('Hello GAS!');
  console.log('I am', 25, 'years old.');
}
```

実行すると、図 2-4-1 のように「実行ログ」という画面が表示され、その一部に「Hello GAS!」「I am 25 years old.」という表示が確認できます。

▶図 2-4-1 実行ログ



ここで登場する **「console.log」は、GAS でログ出力をするための命令**です。以下のように、括弧内に複数の値をカンマ区切りで記述することにより、それらの値をログに出力できます。

▶構文

```
console.log(値1[, 値2, …])
```

なお、構文内の角括弧は省略可能という意味です。以降の構文でもこの表記法が登場します ので、覚えておきましょう。

```
【Memo】
console クラスは GAS の Base サービスで提供されているクラスです。
```

実行ログでは、最新の実行についてのログのみを確認できますが、左側にあるメニューの「**実行数**」から、ログを表示したい実行をクリックすることで、過去のログを確認できます（図2-4-2）。

▶図 2-4-2 実行数

「実行数」の画面では、過去 7 日分の実行結果とそのログが蓄積されています。また、トリガーなどにより、スクリプトエディタを開かずに実行したものも確認できます。

【Memo】
ログとその出力については、第 16 章でも詳しく紹介していますので、そちらもご覧ください。

◆デバッグ機能

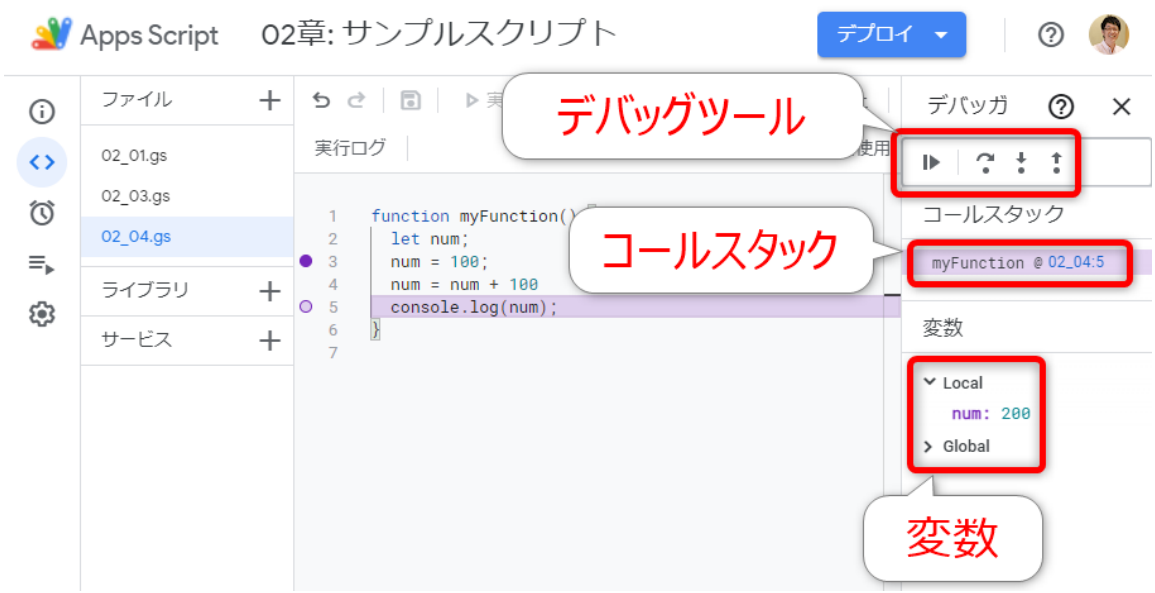
スクリプトエディタには、**デバッグ機能**が搭載されています。ツールバーの「デバッグ」をクリックすると、選択されている関数について**デバッグ実行**を行います。

図 2-4-3 のように、スクリプトの行番号の左側をクリックすることで、**ブレークポイント**を設置できます。この状態でデバッグ実行を行うと、各ブレークポイントの位置で一時停止します。

▶図 2-4-3 ブレークポイントの設置とデバッグ実行

一時停止中は、画面右側に表示される**デバッガ**を使って、停止している時点の情報を確認したり、デバッグ実行の操作を行うことができます（図 2-4-4）。

▶図 2-4-4 デバッガ



デバッグツールを使って、デバッグ実行の方法をコントロールできます。それぞれのアイコンは、左側から順に以下の機能となります。

- 再開： デバッグ実行を再開する
- ステップオーバー： 1 行ずつ実行、呼び出した関数では一時停止しない
- ステップイン： 1 行ずつ実行、呼び出した関数でも一時停止する
- ステップアウト： 現在の関数の最後まで実行する

デバッガでの「変数」には、停止時点に使用されている変数とその内容が表示されます。また、コールスタックでは、停止位置のファイル名と行数とともに、関数の呼び出し履歴を確認できます。

これら、デバッグ実行とデバッガの機能は、スクリプトの動作確認や、不具合（バグといいます）を解決するときに大いに役立ちますので、ぜひ使いこなしていきましょう。

2.5. 権限と許可

◆スクリプトの共有と権限

GAS はクラウドにスクリプトが存在していますから、**どのユーザーが該当のスクリプトの編集や実行についての権限を持っているのかが明確に定められています**。図2-5-1に示す通り、スクリプトの権限のレベルは 3 段階に分かれています。

▶図 2-5-1 スクリプトの権限

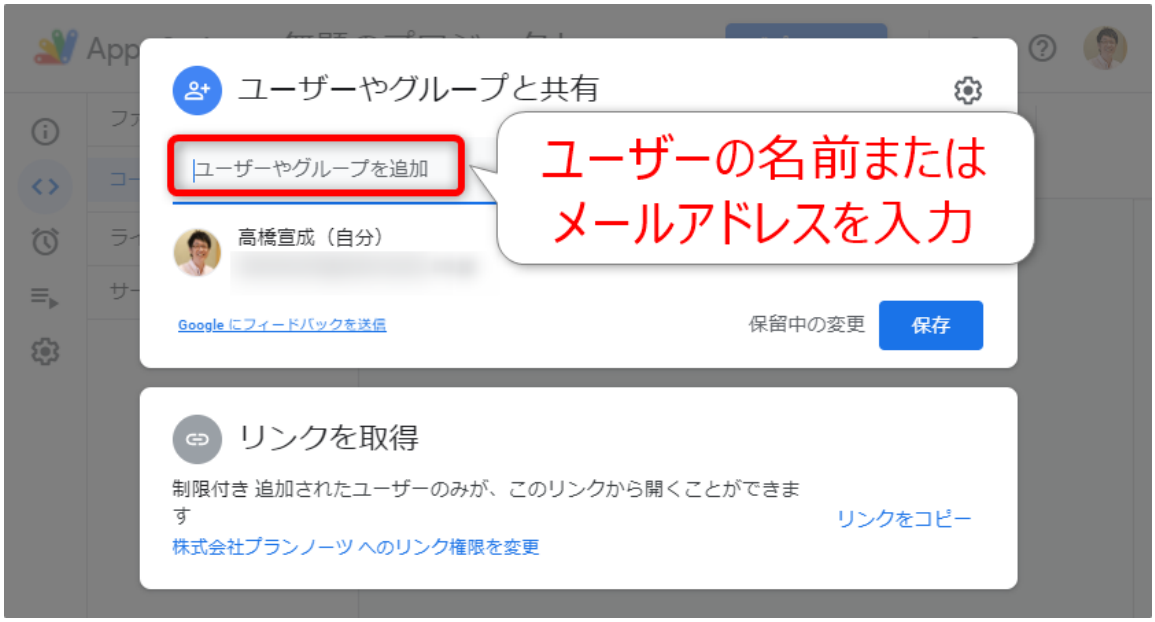
操作	オーナー	編集者	閲覧者
スクリプトの閲覧	○	○	○
実行・デバッグ	○	○	○
ログ表示	○	○	○
トリガー設定	○	○	○
スクリプトの編集	○	○	
他のユーザーの権限設定	○	○	
プロジェクトの公開	○		
オーナー権限の譲渡	○		

まず、スタンドアロンスクリプトの場合は、スクリプトの作成者がオーナーとなります。他のユーザーと共有するには、図 2-5-2 のスクリプトエディタの右上の「他のユーザーとこのプロジェクトを共有」アイコンをクリックして、「ユーザーやグループと共有」ダイアログを開きます。

▶図 2-5-2 スタンドアロンスクリプトの共有

続いて、入力欄に共有したいユーザーの名前かメールアドレスを入力します（図 2-5-3）。

▶図 2-5-3 ユーザーやグループと共有



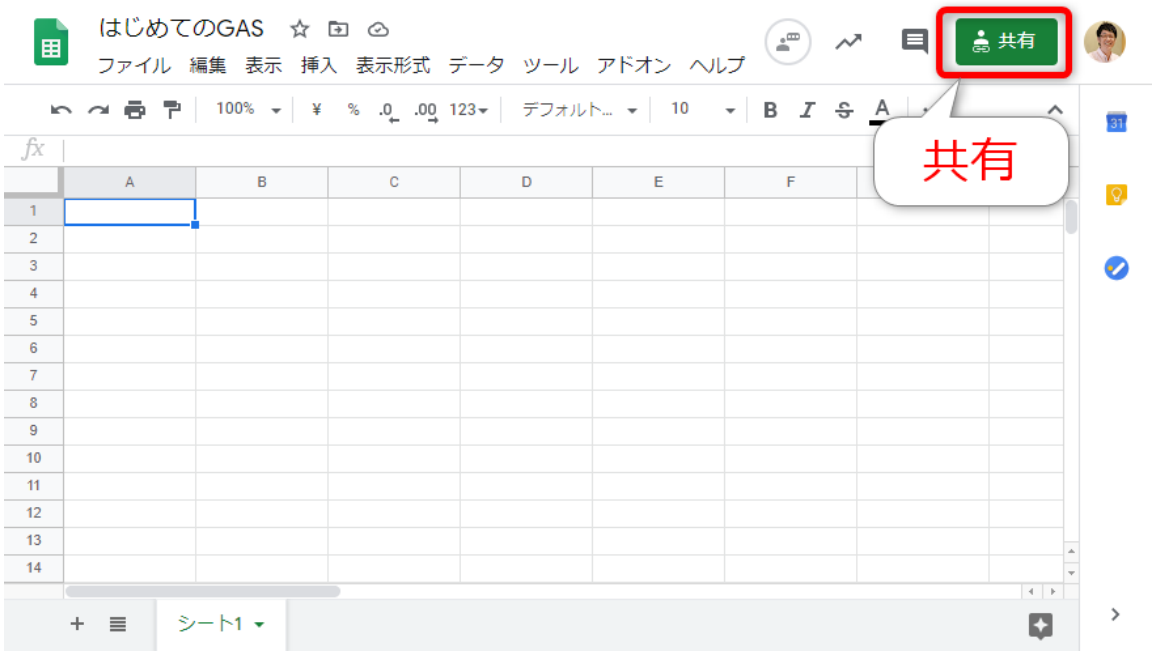
すると、図 2-5-4 の画面となります。ここで、共有するユーザーの権限をプルダウンから選択し、「送信」をクリックすることで共有が完了です。「通知」にチェックが入っている場合は、共有先のユーザーにメールで通知されますので、必要に応じてチェックおよびメッセージを入力するとよいでしょう。

▶図 2-5-4 ユーザーの権限の選択と送信

コンテナバインドスクリプトの権限は、親ファイルとなるコンテナの権限と紐づいています。

ですから、コンテナの作成者がそのままスクリプトのオーナーとなります。他のユーザーと共有する場合は、図 2-5-5 のようにコンテナの「共有」から共有の操作をすることで、スクリプトの共有も同時になされます。以降の共有の手順はスタンドアロンスクリプトの手順と同様です。

▶図 2-5-5 コンテナバインドスクリプトの共有



【Memo】
コンテナの権限が「閲覧者」または「閲覧者（コメント可）」である場合、コンテナのメニューで「ツール」→「スクリプトエディタ」が表示されません。しかし、Apps Script ダッシュボードから開くことで閲覧可能です。

共有について覚えておくべき重要な点として、異なるドメインどうしではコンテナ（そのコンテナバインドスクリプトも含む）およびスタンドアロンスクリプトのオーナー権限の譲渡ができません。したがって、組織で Google Workspace を使用している場合で、外部のパートナーと共同で GAS の開発や運用をする際には注意をする必要があります。

【Point】
スクリプトのオーナー権限は異なるドメイン間では譲渡ができません。

◆スクリプトからのアクセス許可

本章の冒頭で「はじめての GAS」の初回実行時に、「承認が必要です」ダイアログが表示されたのを覚えていますでしょうか？

その際は、スクリプトに対してスプレッドシートへのアクセス許可を与えましたね。GAS のスクリプトの多くは Google アプリケーションにアクセスをしますから、**スクリプトがそ**

れぞれのアプリケーションにアクセスをしてもよいという許可を与える必要があるのです。

GAS ではスクリプトを実行する、またはトリガーを設定する際に、自動的にコードの内容を検査し、必要となるアプリケーションへの許可を求めにいきます。許可はプロジェクト単位でなされ、一度与えられた許可は解除をするまで保持されます。

【Point】
スクリプトから Google アプリケーションを操作するには、それぞれのアプリケーションへのアクセス許可が必要です。

【Memo】
アクセス許可の情報は、スクリプトエディタの左メニュー「概要」を開き、プロジェクトの詳細画面の「プロジェクトの OAuth スコープ」の欄で確認できます。

2.6. サポートメニューとリファレンス

◆サポートメニュー

スクリプトエディタの右上の「？」アイコンは**サポートメニュー**です（図 2-6-1）。

▶図 2-6-1 サポートメニュー

メニューの上から 4 つについては、以下公式ドキュメントへのリンクとなっています。

- ドキュメント： 公式トップ　https://developers.google.com/apps-script/
- トレーニング： Samples　https://developers.google.com/apps-script/articles/
- 更新： Release Notes　https://developers.google.com/apps-script/releases/
- 利用規約： Additional Terms　https://developers.google.com/apps-script/terms

また、「フィードバックの送信」では、スクリプトエディタの不具合の報告や、要望の送信が可能です。

◆リファレンスの活用

スクリプトを作成する上で、GAS で提供されているたくさんのクラスやメソッドについて知る必要があります。スクリプトエディタ上でもそれらの情報を得ることができますが、より詳しく調べたいときには、公式ドキュメントの「Reference」を使うことができます。

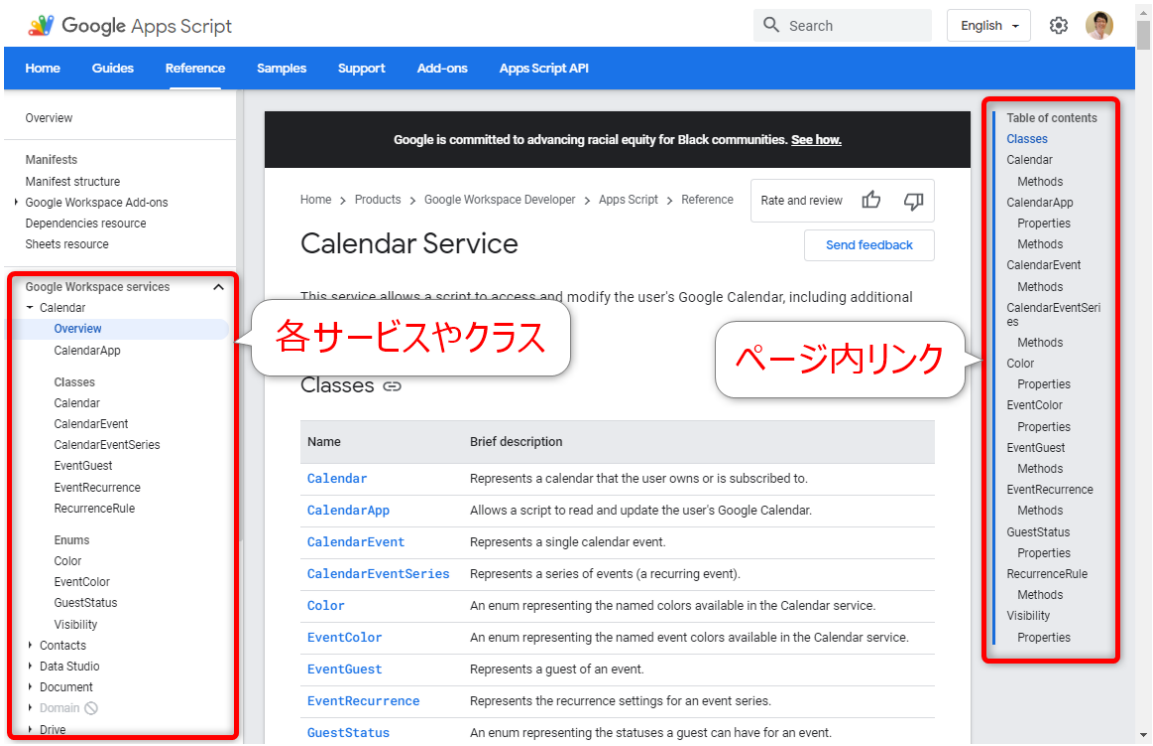
サポートメニューの「ドキュメント」から公式ドキュメントを開き「Reference」タブを選択するか、以下 URL で開くことができます。

```
Reference overview | Apps Script

https://developers.google.com/apps-script/reference
```

リファレンスページの左側の一覧から、調べたいサービスやクラスについてアクセスできます（図 2-6-2）。

▶図 2-6-2 リファレンス：サービス概要



各メソッドの解説は、図 2-6-3 のようになっています。基本構成はすべてのメソッドで統一されていますので、構成さえ理解してしまえば英語が苦手でも読み進めやすくなると思います。

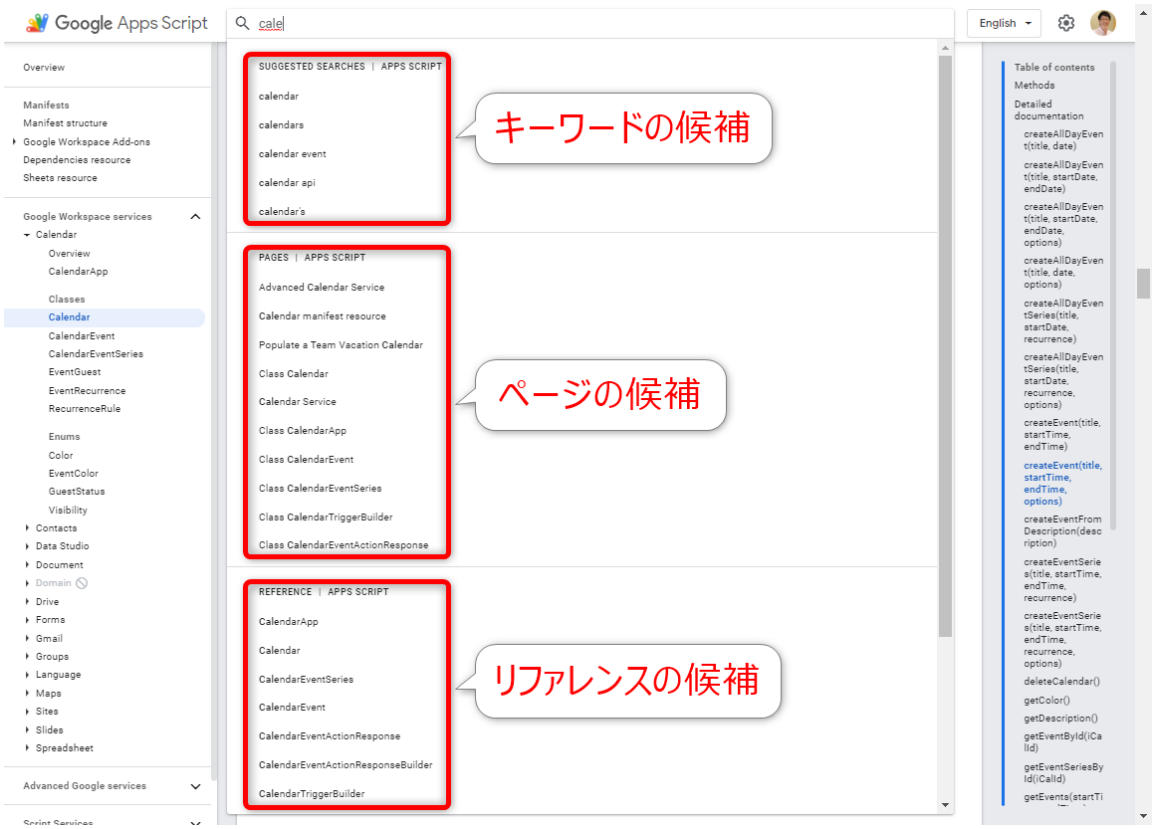
▶図 2-6-3 リファレンス：メソッド

画面上部の検索ボックスを使用すれば、リファレンス内を検索できます。いくつかの文字を入力すると、図 2-6-4 のようにキーワードや、ページまたはリファレンスのサジェストが表示されますので活用しましょう。

GAS で提供されるクラスやメソッドはかなりのボリュームがありますので、すべてを網羅して覚える必要はありません。入力はコンテンツアシストを活用し、使用方法を知りたい場合は都度、本書やリファレンスを参照するとよいでしょう。

【Memo】
クラスとは何か、メソッドとは何かについては、第 6 章で解説をします。

▶図 2-6-4 リファレンス内検索



さて、本章では、GAS の専用エディタである「スクリプトエディタ」とプロジェクトの管理ツールである「Apps Script ダッシュボード」の使い方についてお伝えしてきました。これまで見てきたように、GAS の開発をサポートするさまざまな機能が提供されています。上手に使いこなすことで、習得のスピードも上がることでしょう。

次章から、いよいよ本格的なプログラミングへと入っていきます。GAS のベースとなる JavaScript の基本構文について学んでいくことにしましょう。

20. プロパティサービス

20.1. プロパティストア

◆プロパティストアとは

GAS では、プロジェクトやドキュメントに紐づく形でデータを格納しておくことができます。その格納しておく領域を、**プロパティストア**といいます。プロパティストアには、文字列形式のキーと値のペアを格納することができ、スクリプトから読み書きできます。

たとえば、スクリプト内で使用するファイルの ID や、外部と接続するために必要な情報などは、スクリプトのグローバル領域に記述することもできますが、その場合プロジェクト内のすべての領域から変数名だけでアクセスできてしまいますし、コードをコピーして再利用するときに流出してしまうという心配があります。

そのようなときに、**プロパティストアを使用することで、それらのデータをコードから分離して安全に管理できます**。また、ユーザーごとに別のデータを格納することもできますので、ユーザーによって別のスプレッドシートを参照するといったことも可能になります。

> 【Point】
> プロパティストアは、データをキーと値のペアの形式で格納する領域です。

プロパティストアには、図 20-1-1 のように、プロジェクトに紐づいて格納する**スクリプトプロパティ**、ユーザーごとに格納する**ユーザープロパティ**、ドキュメントに紐づいて格納する**ドキュメントプロパティ**という 3 種類がありますので、用途に応じて使い分けることになります。

▶図 20-1-1 プロパティストアの種類

プロパティストア	プロパティのアクセス	例
スクリプトプロパティ	スクリプト単位で保有するスクリプトプロパティにアクセス	スクリプトで使用するフォルダやファイルのID、DB接続のアカウント情報など
ユーザープロパティ	スクリプトを実行するユーザー自身のユーザープロパティにアクセス	ユーザーごとに作成したファイルIDなど
ドキュメントプロパティ	開いているドキュメントのドキュメントプロパティにアクセス	ドキュメントに使用されているデータのソースURLなど

20.2 Properties サービスとPropertiesService クラス

◆Properties サービスと PropertiesService クラス

Properties サービスはプロパティストアを取り扱うためのサービスで、Script Services に含まれています。Properties サービスには、図 20-2-1 に挙げる 2 つのクラスが用意されいます。

▶図 20-2-1 Properties サービスのクラス

クラス	説明
PropertiesService	Propertiesサービスのグローバルオブジェクト
Properties	プロパティストアを操作する機能を提供する

Properties サービスを使用することで、プロパティストアへのデータの読み書きができますが、第 1 章でお伝えした 割り当て、制限により、1 日あたりのプロパティの読み書きができる回数や、プロパティとして持てるデータサイズが定められています。その範囲で使用するように注意しましょう。

【Point】
プロパティの読み書きの回数や格納する容量に注意しましょう。

PropertiesService クラスでは、図 20-2-2 に示す通り、各プロパティストアをProperties オブジェクトとして取得するメンバーが提供されています。

▶図 20-2-2 PropertiesService クラスのメンバー

メンバー	戻り値	説明
getScriptProperties()	Properties	スクリプトプロパティを取得する
getUserProperties()	Properties	ユーザー自身のユーザープロパティを取得する
getDocumentProperties()	Properties	ドキュメントプロパティを取得する

スクリプトプロパティを取得する **getScriptProperties メソッド**、ユーザープロパティを取得する **getUserProperties メソッド**、ドキュメントプロパティを取得する **getDocumentProperties メソッド**は、以下の書式でそれぞれのプロパティストアの内容を Properties オブジェクトとして返します。

▶構文

```
PropertiesService.getScriptProperties()
```

```
PropertiesService.getUserProperties()
```

```
PropertiesService.getDocumentProperties()
```

実際にプロパティストアの内容を操作する際には、これにより取得した Properties オブジェクトに対してデータの読み書きなどの操作を行うことになります。なお、getUserProperties メソッドでは、現在スクリプトを実行しているユーザーに紐づくユーザープロパティを取得します。

例として、サンプル 20-2-1 を実行してみましょう。この例では、スクリプトプロパティを設定し、またそれを取り出してログ出力するものです。

ここで、setProperties メソッドと getProperties メソッドは、それぞれ Properties オブジェクトのキーと値のペアを設定または取得するものです。

▶サンプル 20-2-1 スクリプトプロパティの設定と取得 [sample20-02.gs]

```
function myFunction20_02_01() {
  const scriptProperties = PropertiesService.getScriptProperties();

  scriptProperties.setProperties({'犬': 'わんわん', '猫': 'にゃあにゃあ',
'狸': 'ぽんぽこ'});
  console.log(scriptProperties.getProperties());
}
```

◆実行結果

```
{'犬': 'わんわん', '猫': 'にゃあにゃあ', '狸': 'ぽんぽこ'}
```

この例は、あまり意味のあるデータではありませんが、プロパティストアを使ってスクリプト内で使用する他のファイルの ID や、API 接続で使用するトークンなど、定数的に使用するータや流出を避けたいデータなどを管理するとよいでしょう。

【Memo】
以前、スクリプトプロパティはスクリプトエディタのメニューから閲覧および編集をすることが可能でした。しかし、2020 年 12 月に新しい IDE が導入された際に、その機能は取り除かれました。したがって、新しい IDE でプロパティストアを使用する際には、最初の手順として、そのデータを設定するためのスクリプトを作成、実行することが必要になっています。

20.3. Properties クラス - プロパティストアの読み書き

◆Properties クラスとは

Properties クラスは、プロパティストアを操作する機能を提供するクラスです。図 20-3-1のようにプロパティストアのデータを読み書きする、または削除をするメンバーで構成されています。

▶図 20-3-1 Properties クラスのメンバー

メンバー	戻り値	説明
deleteAllProperties()	Properties	プロパティストアのすべてのプロパティを削除する
deleteProperty(key)	Properties	プロパティストアの指定されたkeyを持つプロパティを削除する
getKeys()	String[]	プロパティストアのすべてのキーを取得する
getProperties()	Object	プロパティストアのすべてのキーと値のペアを取得する
getProperty(key)	String	プロパティストアの指定されたkeyに関連付けられた値を取得する
setProperties(properties[, deleteAllOthers])	Properties	プロパティストアにpropertiesで指定したすべてのキーと値のペアを設定する（deleteAllOthersにtrueを設定すると、事前に既存のすべてのプロパティを削除する）
setProperty(key, value)	Properties	プロパティストアに指定されたkeyと値valueのペアを設定する

◆キーと値のセットをまとめて読み書きする

プロパティストアのデータをまとめて読み書きするには、**getProperties メソッド**と**setProperties メソッド**を使います。

書式は以下の通りです。

▶構文

```
Propertiesオブジェクト.getProperties()
```

```
Propertiesオブジェクト.setProperties(オブジェクト[, 削除オプション])
```

getProperties メソッドは、指定したプロパティストアのすべてのキーと値のペアをオブジェクト形式で取得します。setProperties メソッドでは、オブジェクト形式で指定したキーと値の組み合わせすべてについて、キーが存在すれその値を書き換え、キーが存在しなければキーと値を新たに追加します。削除オプションは省略できますが、true を設定すると、一度対象とするプロパティストアのすべてのデータを削除してからの設定となります。

例として、サンプル 20-2-1 の実行後の状態からスクリプトプロパティのデータの書き換えと追加をしてみましょう。

▶サンプル 20-3-1 スクリプトプロパティの書き換えと追加 [sample20-03.gs]

```
function myFunction20_03_01() {
  const scriptProperties = PropertiesService.getScriptProperties();
  scriptProperties.setProperties({'猫': 'にゃーご', '馬': 'ひひーん'});

  const properties = scriptProperties.getProperties();
  for (const key in properties) console.log(key, properties[key]);
}
```

◆実行結果

```
馬 ひひーん
猫 にゃーご
狸 ぽんぽこ
犬 わんわん
```

一方で、プロパティストアにセットされている、すべてのキーと値のペアを削除したいときには、以下の **deleteAllProperties メソッド**を使用できます。

▶構文

```
Propertiesオブジェクト.deleteAllProperties()
```

サンプル 20-3-2 を実行すると、実行ログには空のオブジェクトが出力されますので、確認してみましょう。

▶サンプル 20-3-2 スクリプトプロパティをすべて削除する [sample20-03.gs]

```
function myFunction20_03_02() {
  const scriptProperties = PropertiesService.getScriptProperties();
  scriptProperties.deleteAllProperties();

  console.log(scriptProperties.getProperties()); //{}
}
```

◆特定のキーの値を読み書きする

プロパティストアの操作でもっとも使用する機会が多いのは、特定のキーに対する値を読み書きするというものです。取得には **getProperty メソッド**を使用し、設定には **setProperty メソッドを**使用します。

書式はそれぞれ、以下の通りです。

▶構文

```
Propertiesオブジェクト.getProperty(キー)
```

```
Propertiesオブジェクト.setProperty(キー, 値)
```

getProperty メソッドでは、指定したキーの値を取り出すことができます。キーが存在しない場合は、null が返ります。setProperty メソッドでは、指定したキーに値を設定します。キーが存在しない場合は、新たにキーと値をペアで設定をします。

使用例として、サンプル 20-3-3 をご覧ください。実行するたびに、スクリプトプロパティのキー COUNT に対応する値を 1 ずつカウントアップするものです。

▶サンプル 20-3-3 スクリプトプロパティの特定のキーの値の設定と取得 [sample20-03.gs]

```
function myFunction20_03_03() {
  const scriptProperties = PropertiesService.getScriptProperties();
  let count = Number(scriptProperties.getProperty('COUNT'));

  if (count) {
    count++;
    scriptProperties.setProperty('COUNT', count.toString());
  } else {
    scriptProperties.setProperty('COUNT', '1');
  }

  console.log(scriptProperties.getProperty('COUNT'));
}
```

実行するたびに、実行ログに出力される値が 1 ずつ増えていくことを確認してください。なお、プロパティストアの値は文字列ですから、この例では、取得した値を Number 関数によって数値に変換したり、設定する値を toString メソッドで文字列に変換したりしていることを確認しておきましょう。

さて、プロパティストアにセットされている、特定のキーと値のペアを削除するには、以下の **deleteProperty メソッド**を使用します。

▶構文

```
Propertiesオブジェクト.deleteProperty(キー)
```

例として、サンプル 20-3-3 の実行後の状態で、サンプル 20-3-4 を実行してみましょう。キーと値が削除されたことを確認できます。

▶サンプル 20-3-4 特定のキーと値の削除 [sample20-03.gs]

```
function myFunction20_03_04() {
  const scriptProperties = PropertiesService.getScriptProperties();
  scriptProperties.deleteProperty('COUNT');

  console.log(scriptProperties.getProperties()); //{}
}
```

◆ユーザープロパティの活用

ユーザープロパティは、ユーザーごとに異なるデータを管理できます。たとえば、ユーザープロパティにユーザーごとのスプレッドシート ID を格納するようにすることで、ユーザーごとのスプレッドシートに対して処理をすることができるようになります。

例として、サンプル 20-3-5 を用意しました。ユーザープロパティに「SPREAD_SHEET_ID」の値が存在していなければ、新規でスプレッドシートを作成し、その ID をユーザープロパティに格納するというものです。

▶サンプル 20-3-5 ユーザーごとのスプレッドシートを作成する [sample20-03.gs]

```
function myFunction20_03_05() {
  const userProperties = PropertiesService.getUserProperties();
  const spreadSheetId =
userProperties.getProperty('SPREAD_SHEET_ID');

  if (spreadSheetId) {
    throw('すでにスプレッドシートが存在しています: ' + spreadSheetId);
  } else {
    const ssName = `スタッフ別(${Session.getActiveUser().getEmail()})`;
    const ss = SpreadsheetApp.create(ssName);
    userProperties.setProperty('SPREAD_SHEET_ID', ss.getId());
  }
}
```

ただし、前述の通り、ユーザープロパティは自らのデータにのみアクセスできます。ですから、スクリプトの動作を確認しづらい場合があります。その点、留意して使用するようにしましょう。

【Point】
ユーザープロパティは、自らのデータにのみアクセスできるので注意しましょう。

本章では、コードとは別の領域にデータを格納できるプロパティストアと、その操作をする Properties サービスについて紹介をしました。派手な機能ではないように思われるかも知れませんが、スクリプトの管理のしやすさ、読みやすさや安全性を高めることにつながりますので、上手に活用をしていきましょう。

さて、次章はクラウド上ですべてが動作する GAS ならではの機能である、「イベント」および「トリガー」について解説をしていきます。これらの機能を使用することで、スプレッドシートやドキュメントの起動や編集、もしくは特定の時間帯といったイベントにあわせてスクリプトを動作させることができるようになります。

GAS によるツールやアプリケーションの幅を大きく広げる機能となりますので、ぜひ使いこなせるようにしていきましょう。

23. ライブラリ

23.1. ライブラリを使用する

◆ライブラリとその追加

GAS では、作成した関数を他のプロジェクトから利用できるライブラリという機能を使うことができます。他のユーザーが公開しているライブラリを使用することもできますし、自作をすることもできます。

すでにあるライブラリを活用することで、よく使う機能については自分で組み上げる必要がなくなり、GAS の開発効率を大きく上げることにつながります。ここでは、導入テスト用に用意した「Hello」というライブラリを例として、その導入方法や使い方を見ていきます。

> 【Point】
> ライブラリを使用することで、すでにある関数やクラスを再利用できます。

では、ライブラリの追加の方法を見ていきましょう。まず、ライブラリを追加したいプロジェクトを開き、「ライブラリ」の「＋」アイコンをクリックします（図 23-1-1）。

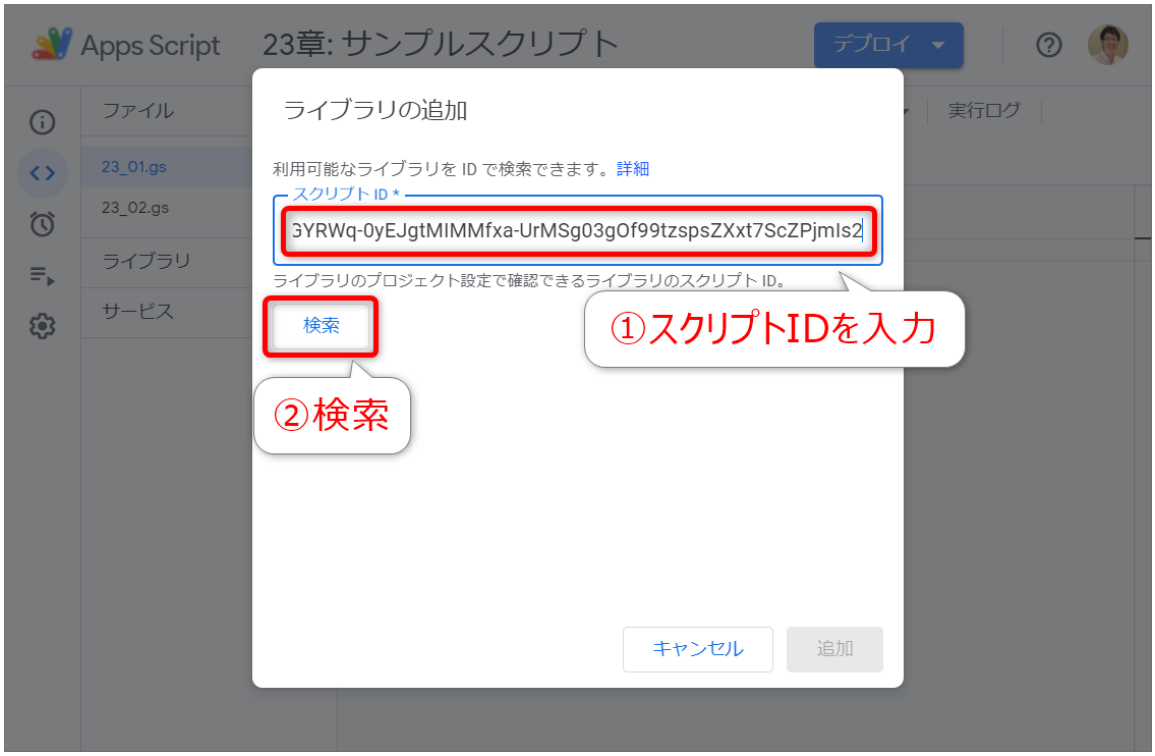
▶図 23-1-1 ライブラリを追加する

図 23-1-2 のように、「ライブラリの追加」ダイアログが開きますので、以下ライブラリ「Hello」の「スクリプト ID」を入力して「検索」をクリックします。

```
スクリプト ID:
1U81fRGYRWq-0yEJgtMIMMfxa-UrMSg03gOf99tzspsZXxt7ScZPjmIs2
```

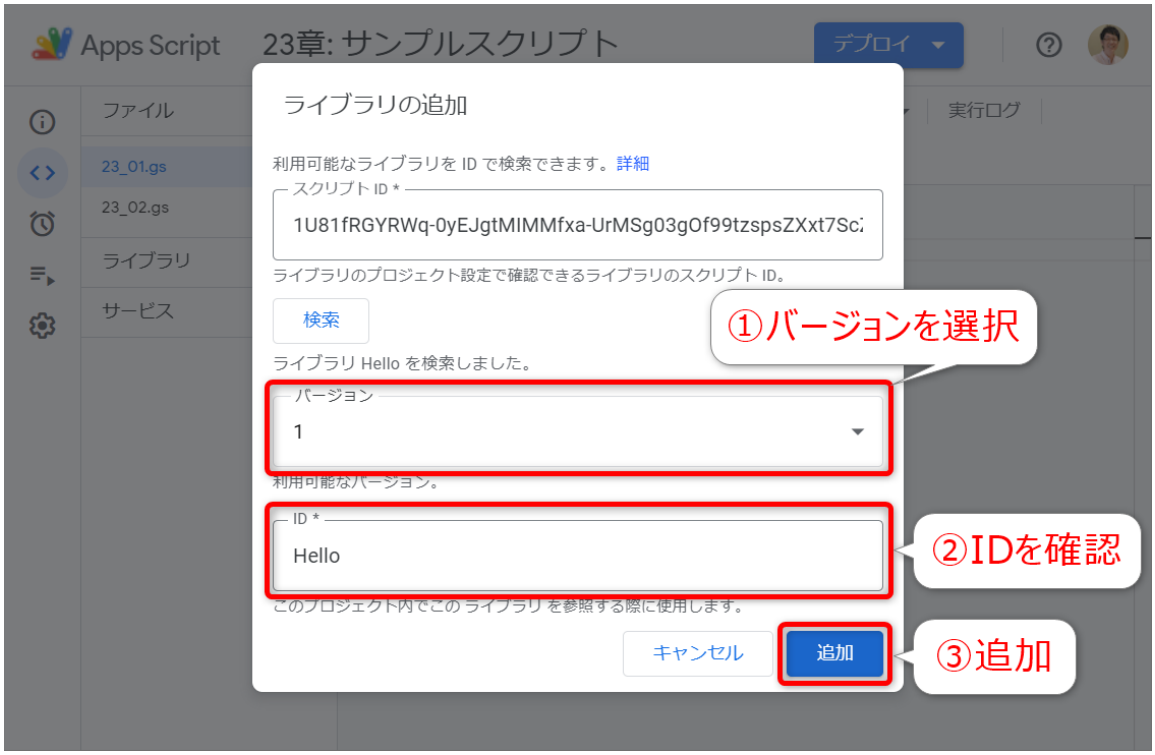
【Memo】
ライブラリ「Hello」のスクリプト ID は、サンプルファイル [sample23_01.gs] 内にも記載していますので、ご活用ください。

▶図 23-1-2 「ライブラリの追加」ダイアログ



すると、図 23-1-3 のように、ライブラリ Hello についての検索結果が表示されます。「バージョン」のプルダウンは、最新のバージョン（もっとも大きな数字）を選択します。また、「ID」はライブラリの識別子を表します。変更することもできますが、とくに理由がない限りはデフォルトのままでよいでしょう。

▶図 23-1-3 ライブラリのバージョンを選択して追加



そして「追加」をクリックすると、図 22-1-4 のようにスクリプトエディタの「ライブラリ」に「Hello」が追加されます。

▶図 23-1-4 追加したライブラリ

◆ライブラリを使用する

追加したライブラリ内のメンバーは、**識別子**を使用して、以下の記述で呼び出すことができます。

▶構文

```
識別子.メンバー名
```

Hello ライブラリが追加されたことを、サンプル 23-1-1 を実行して確認してみましょう。「～さん、こんにちは！」というメッセージがログ出力されていれば、Hello ライブラリは正しく動作しているということになります。

▶サンプル 23-1-1 Hello ライブラリの動作確認 [sample23-01.gs]

```
function myFunction23_01_01() {
  console.log(Hello.getHello('Bob')); //Bob さん、こんにちは！
  console.log(Hello.getHello()); // 名無し さん、こんにちは！
}
```

なお、ライブラリを使用している場合、単一のプロジェクトよりも実行速度が遅くなりますので、スピードを求められるプロジェクトの利用には推奨されていません。

さて、ライブラリを使用している場合、作成者がライブラリのバージョンアップをしたとしても、追加したライブラリのバージョンが自動的に最新になることはありません。

ライブラリのバージョンの確認や、バージョンアップを確認する場合は、「ライブラリ」から該当のライブラリ名をクリックして開く、「ライブラリの設定」ダイアログで行うことができます（図 23-1-5）。

新しいバージョンがあれば、今より大きい数字のバージョンが選べるようになっていますので、そのバージョンに切り替えて「保存」をクリックします。

▶図 23-1-5 「ライブラリの設定」ダイアログ

【Memo】
公開されているライブラリは永続的に使用できるとは限りません。実際に、とても利用されているメジャーなライブラリでも開発がストップし、メンテナンスがされなくなるということも起こりえます。ライブラリは GitHub などで公開されているものも多いので、最近のアップデートがあるかなどを、活用する際の判断材料にしましょう。

また、ライブラリの三点リーダーアイコンをクリックすると、「新しいタブで開く」「削除」の 2 つのメニューが表示されます（図 23-1-6）。

「新しいタブで開く」を選択すると、そのライブラリについてのドキュメントをブラウザの別のタブで開きます。「削除」を選択すると、ライブラリを削除できます。

▶図 23-1-6 ライブラリのその他のメニュー

23.2. ライブラリを作成する

◆ライブラリのスクリプトを準備する

ライブラリを作成する方法を見ていきましょう。ライブラリを作成する手順は、以下の通りとなります。

1. スタンドアロンスクリプトでプロジェクトを作成する
2. スクリプトを作成する
3. 共有設定で公開範囲を設定する
4. 版の管理で新しいバージョンを作成する

まず、新規のプロジェクトを作成します。**ライブラリを作成する場合はスタンドアロンスクリプトである必要があります**ので、コンテナバインドで作成しないようにしましょう。

> 【Point】
> ライブラリはスタンドアロンスクリプトで作成する必要があります。

また、**プロジェクト名がデフォルトの識別子として採用されます**。ですから、プロジェクト名としては、JavaScript や GAS の予約語やグローバルオブジェクトとバッティングすることはできませんし、日本語は避ける必要があります。今回は「MyLibrary」として進めます。

続いて、サンプル 23-2-1 のスクリプトを作成します。

▶サンプル 23-2-1 ライブラリ MyLibrary [sample23-02.gs]

```
/** 人を表すクラス */
class Person {
  /**
   * Person オブジェクトを生成する
   * @param {string} name - 名前
   * @param {number} age - 年齢
   */
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  /**
```

```
    * あいさつ文をログ出力する
    */
   greet(){
     console.log(`Hello! I'm ${this.name}!`);
   }
}

/**
 * Person クラスのインスタンスを生成して返すファクトリ関数
 *
 * @param {String} name - 名前
 *@param {Number} age – 年齢（既定値は 0.1）
 * @return {Person} - 生成した Person オブジェクト
 */
function createPerson(name, age) {
  return new Person(name, age);
}

/**
 * 税込み価格を返す関数
 *
 * @param {Number} price - 価格
 * @param {Number} taxRate - 税率（既定値は 0.1）
 * @return {Number} - 税込価格
 */
function includeTax(price, taxRate = 0.1) {
  return price * (1 + taxRate);
}
```

人を表すクラス Person、Person クラスのインスタンスを作成する関数 createPerson、税込み価格を返す関数 includeTax で構成されています。ライブラリ上のクラスについては、他のプロジェクトから直接的に new 演算子によるインスタンス生成を行うことができませんので、インスタンスを生成するための関数をライブラリ内に定義し、それを呼び出します。そのような、インスタンス生成用の関数を**ファクトリ関数**といいます。

ここで、図 23-2-1 のタグを用いて**ドキュメンテーションコメント**を使うことができます。ドキュメンテーションコメントを記述した関数は、ライブラリを追加したプロジェクトにおいて補完対象にできます。

▶図 23-2-1 ライブラリのドキュメンテーションコメントで使用するタグ

タグ	説明	書式
@param	引数の情報を追加する	{データ型} 仮引数名 - 概要
@return	戻り値の情報を追加する	{データ型} - 概要

◆ライブラリの公開範囲を設定する

次に、ライブラリを公開する範囲を定めるために、スクリプトの共有設定を行います。なお、自分だけが使用するライブラリであれば、共有設定を変更する必要はありません。

まず、スクリプトエディタの「プロジェクトの共有」アイコンをクリック、続いて「ユーザーやグループ」ダイアログの「～へのリンクの権限を変更」または「リンクを知っている全員に変更」をクリックします（図 23-2-2）。

▶図 23-2-2 「ユーザーやグループと共有」ダイアログ

続いて、図 23-2-3 のとおり、公開範囲のプルダウンを開き、組織名または「リンクを知っている全員」を選択して「完了」とします。これで、公開範囲の設定ができました。

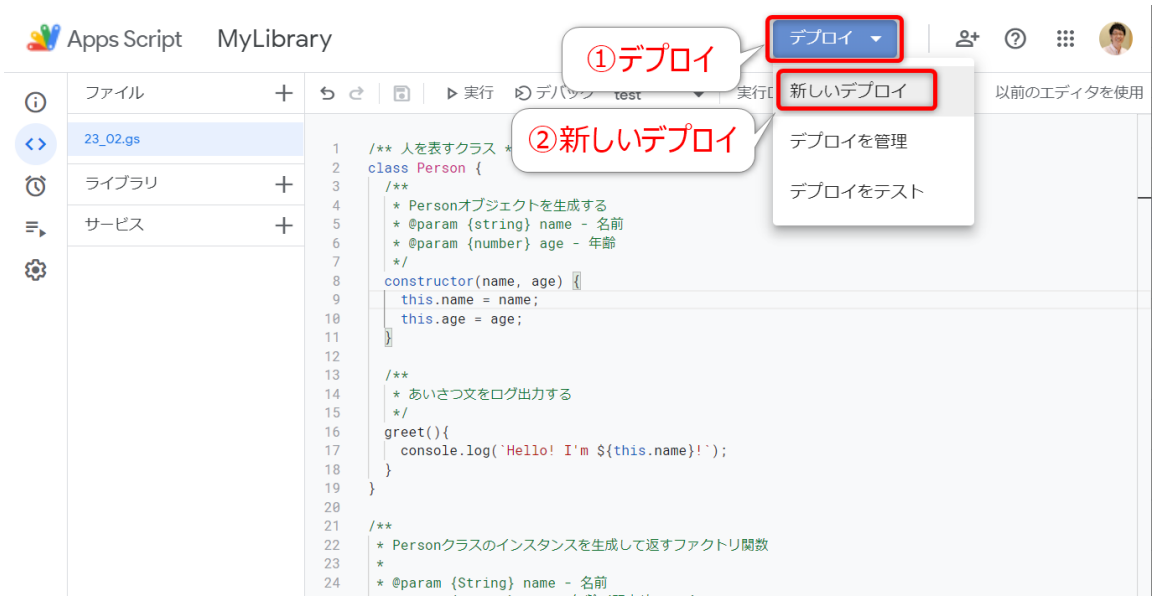
▶図 23-2-3 公開範囲を選択する

さらに、ライブラリとして使用する場合には、「**デプロイ**」という手順を踏む必要があります。

デプロイというのは、公開して使える状態にするということを指します。

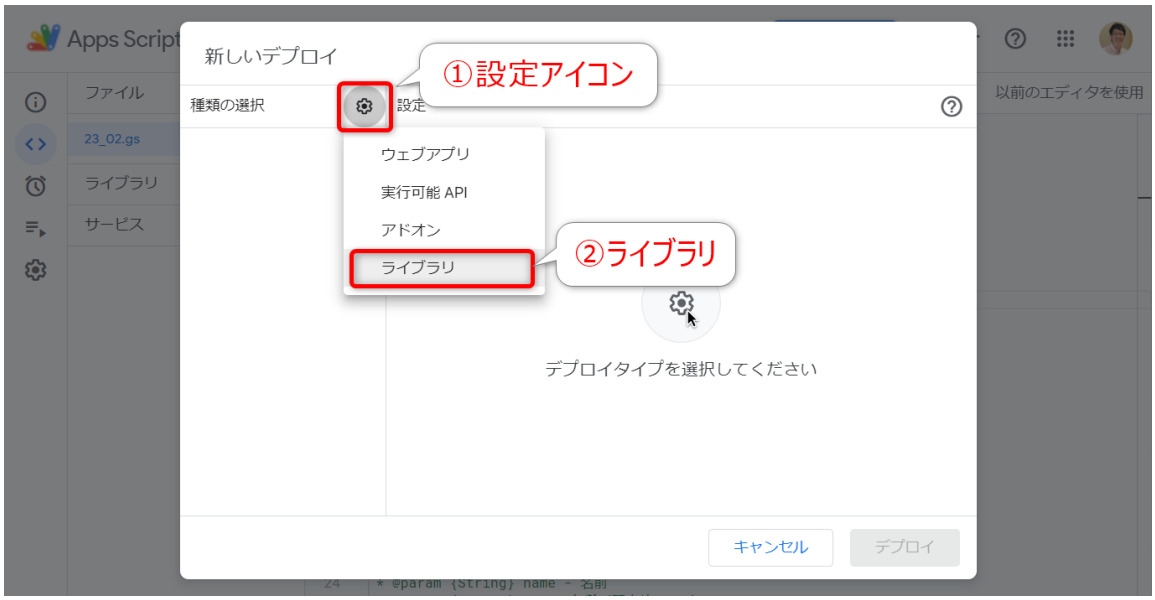
まず、スクリプトエディタの「デプロイ」ボタンから「新しいデプロイ」を選択します（図23-2-4）。

▶図 23-2-4 「新しいデプロイ」を選択



すると、「新しいデプロイ」ダイアログが開きます（図 23-2-5）。ここでは、種類の選択の設定アイコンから「ライブラリ」を選択してください。

▶図 23-2-5 デプロイの種類を選択



続いて、ライブラリの説明を入力します（図 23-2-6）。ここで、説明を日本語で入力をするとライブラリ導入時にそのバージョンが認識されなくなってしまいますので、アルファベットで入力する必要があります。注意してください。

入力が完了したら「デプロイ」をクリックします。

▶図 23-2-6 説明を入力してデプロイ

これで、新しいバージョンがデプロイされ、図 23-2-7 の画面となりますので、「完了」をクリックします。なお、ここでデプロイ ID やライブラリの URL は不要ですので、コピーの必要はありません。

▶図 23-2-7 デプロイの完了

他のプロジェクトでライブラリの追加をするには、スクリプト ID が必要でした。スクリプト ID は「プロジェクトの設定」メニューを開き、「スクリプト ID」のパートに表示されています。「コピー」をクリックして取得しましょう（図 23-2-8）。

▶図 23-2-8 スクリプト ID のコピー

なお、自身が編集可能なライブラリであれば、「ライブラリの追加」ダイアログにおいて「HEAD（開発モード）」というバージョンを選択できます（図 23-2-9）。

▶図 23-2-9 開発モードでライブラリを追加する

このバージョンでライブラリを追加すると、ライブラリの変更が保存された際に、その変更が即時反映されるようになります。ライブラリの開発時などに活用しましょう。

では、このスクリプト ID を用いて他のプロジェクトにライブラリ「MyLibrary」を追加しましょう。その上で、サンプル 23-2-2 を作成して、ライブラリが使用できるか確認してみてください。

▶サンプル 23-2-2 他のプロジェクトから自作のライブラリを使用する [sample23-02.gs]

```
function myFunction23_02_02() {
  const p = MyLibrary.createPerson('Bob', 25);
  console.log(p); //{ name: 'Bob', age: 25 }
  p.greet(); //Hello! I'm Bob!

  console.log(MyLibrary.includeTax(1000)); //1100
  console.log(MyLibrary.includeTax(1000, 0.08)); //1080
}
```

本章では、ライブラリとは何か、およびライブラリの作成の仕方について解説しました。

ライブラリを活用することで、効果的にコードの再利用を行うことができるようになります。ぜひ、お気に入りのライブラリを作成してみてください。