JavaScript問題集

第７版

for Web Designers

SAMPLE

プログラミングを諦めるのはまだ早い
まずは3つのことをしっかり覚えよう

武舎広幸 著

はじめに

2014年のこと。それまで、（機械翻訳などのソフトウェア開発をしつつ）プログラミングなどの本の翻訳をやっていたのですが、ある事情があって翻訳の仕事が途切れてしまいました。別の翻訳の仕事を開拓するという選択肢もあったのでしょうが、プログラミング講座を始めてみることにしました。というのは、その2年ほど前に、ある通信教育の会社からの依頼で3巻からなるDVD版のJavaScript講座を作成したことがあり、「もう少し突き詰めれば、自分独自の面白いものができるかも」と思っていたのです。

丸一日（約5時間）を使って、プログラミングとは何かを体感していただき、JavaScriptの基本を学びながら、プログラミング技術の習得に必要な「感覚」を身につけていただくために筆者がゼロから開発した講座です。小手先の技術だけではなく、プログラミングの基本がしっかり身に付くように、重要なポイントを解説しており、2014年10月以来、ストアカ、Doorkeeper、connpassなどを通して2,000人以上の方に受講いただきました。

この本はその講座の問題集として作成したものです。受講生の方々の反応を見て、「こんなところで躓（つまず）くのか。たしかに、初心者にはわかりにくいか〜」「こうしたほうがわかりやすいかな？」と試行錯誤を繰り返して「改良」を重ねました。

なんとなく、「プログラミングをマスターするには、覚えなくてはいけないことがたくさんある」と思い込んでいませんか。まずは**3つのことをしっかり覚えれば大丈夫**。すべてのプログラムは、この3つを組み合わせることでできているのです。

その3つのことをしっかり覚えていただいて、プログラミングの基礎を身に着けていただくのがこの問題集の目的です。単純な問題から、ある程度高度な問題まで、70問以上の練習問題を解くことで、JavaScriptのみならず、プログラミングの基本が身につくようになっています。

この本にはウェブ上にサポートページ（https://musha.com/sc/jsbes）があり、下記の資料等をご覧いただけます。

- 問題を解くために利用するサンプルプログラムやデータ
- 問題の解答例
- 前提となる知識をを解説した動画（https://musha.com/ytvj。初心者の方は、こちらの動画をご覧になってからこの問題集に取り組むことをおすすめします）

疑問点などがありましたらまず上記サポートページの資料をご覧ください。動画や資料を参考になさった上で、ご質問等がある場合はサポートページからご連絡ください。この問題集に関するご質問は無料でお受けしております。

なお、次の書籍や講座もおすすめです。

- 書籍『実践 JavaScript! —— プログラミングを楽しみながら しっかり身につける』（武舎広幸著、オーム社刊。https://musha.com/scjs）
  講座の内容をより詳しく解説し、さらに練習問題を約30問追加したプログラミング入門書です。この問題集とともにお読みになると理解がさらに深まるはずです。全国の書店、オンラインショップでご購入いただけます
- ストアカの講座『実習編 JavaScriptで学ぶ プログラミング入門 丸１日コース』（https://musha.com/sc/jsj）
  この問題集を解いていただくための講座です。一人で解くのは大変（無理）だと思う方はご参加ください。講師（この本の著者）がていねいにご説明します

この問題集は、ストアカ、Doorkeeper、connpassで公開した講座『JavaScriptで学ぶ プログラミング入門 丸１日コース』（実習編を含む）にご参加いただいた2,000人を超える皆様のフィードバックにより、長年に渡り改良されてきました。ご参加いただいた皆様およびサイト運営者の皆様に感謝いたします。なお、この本は、第６版までは講座のタイトルを冠していましたが、内容をわかりやすく表現するために、第７版からタイトルを変更しました。

アプリとデータのダウンロード

問題を解くために、下に示すデータやアプリをダウンロード（インストール）しておいてください。

- 練習問題などで利用するサンプルデータ —— https://musha.com/sc/jsbes
  - Windowsでは、ダウンロードが済んだら、エクスプローラで右クリックして、［すべて展開...］を選んで「デスクトップ」などの下に`student`フォルダを展開しておいてください
  - macOS（Macintosh）では、`student`フォルダを「デスクトップ」などに移動しておいてください
- Google Chrome（ウェブブラウザ。略して**ブラウザ**） —— https://musha.com/scjs?ap=chr
- Visual Studio Code（テキストエディタ。略して**エディタ**。 —— https://musha.com/scjs?ap=vsc
  Windowsでインストールする際には、セットアップの途中の画面で表示される「追加タスクの選択」で、すべての項目にチェックを入れてインストールしてください（macOSではこの操作は必要ありません）。なお、Visual Studio Code（略してVS Code）以外のエディタでも、もちろん結構です

VS Codeの日本語化

VS Codeをインストールしたら、VS Codeのメニューなどを日本語化するために機能拡張（extension）の「Japanese Language Pack」をインストールします。

macOSでは「表示言語を日本語に変更するには言語パックをインストールします。」という小さなウィンドウが表示されます（図1）。この場合は［インストールして再起動］のボタンをクリックするとインストールされます（表示されなかった場合は、下の手順に従ってください）。

図1: macOSで表示される日本語の言語パックのインストールを促すメッセージ

Windowsでは図1のウィンドウは表示されませんので、次の手順で日本語用の「言語パック」（Japanese Language Pack）をインストールしてください（図2）。

1. ウィンドウ左側の「アクティビティバー」の機能拡張（Extensions）のアイコンを選択します（①）
2. 右隣の「サイドバー」に「拡張機能」が表示されるので検索欄に「japanese」と入力します（②）
3. 「Japanese Language ...」の右下に［Install］のボタンが表示されるのでこれを選択してインストールします（③）
4. しばらくすると、右下に［Change Language and Restart］（言語を変更して再起動）というボタン（図3）が表示されるので、これを選択してVS Codeを再起動してください

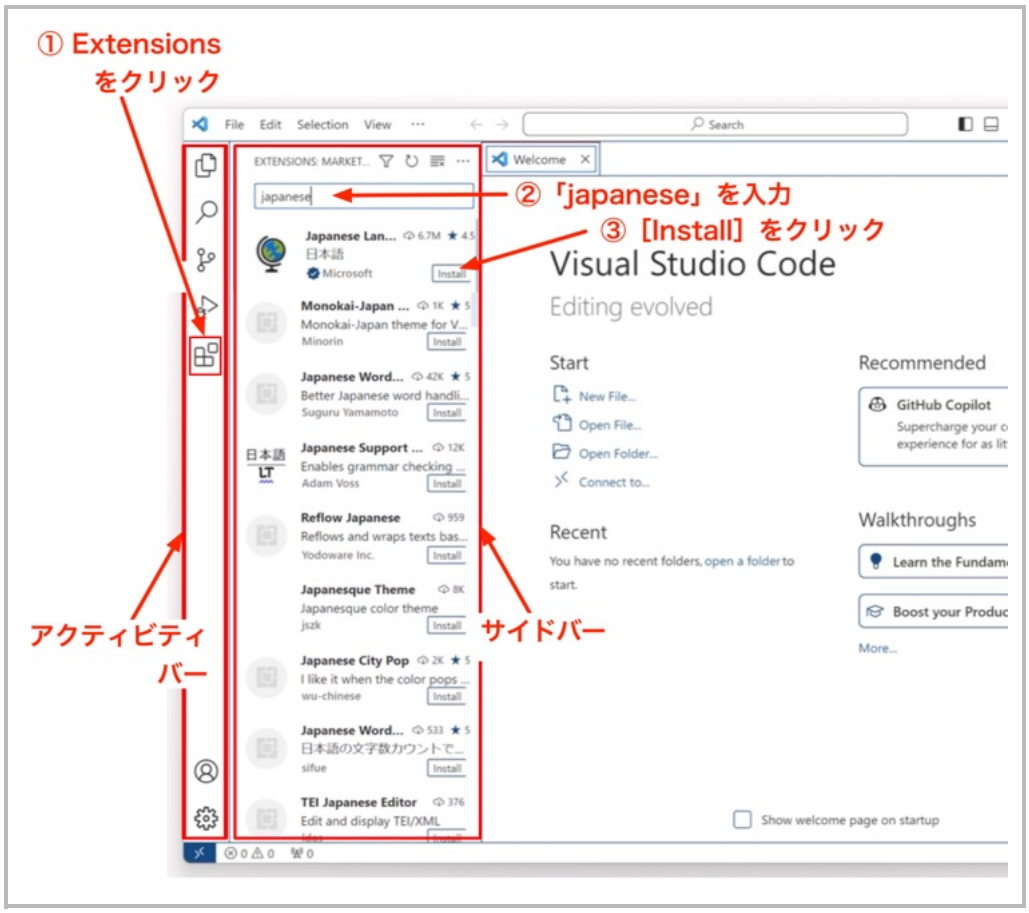


図2: Japanese Langauge Packのインストール

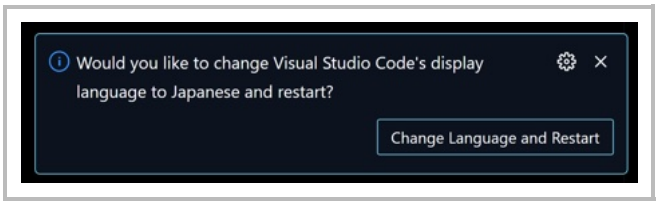


図3: ［Change Language and Restart］をクリックして再起動するとメニューなどが日本語に変わる

VS Codeの環境設定

VS Codeにはさまざまな設定項目があり、色使いや文字の大きさなどを変更できます。たとえば次のような設定が可能です。

- 配色の選択 —— アクティビティバーの一番下にある歯車のアイコンを選択して、［テーマ］→［配色テーマ］で配色を選択できます。標準の設定は［ハイ コントラスト ダーク テーマ］になっていますが、この本では「ライト モダン」にして、紙の本で読みやすくしています
- 画面の各要素の拡大・縮小 —— VS Codeの［表示］→［外観］→［拡大］でアクティビティバーやサイドバーも含め、文字やアイコンを拡大できます。［縮小］を選べば縮小されますし、［ズームのリセット］で元どおりの大きさに戻ります
- よく使用するもの —— 歯車のアイコンを選択して、［設定］を選択すると「よく使用するもの」が表示されます。たとえば「Editor: Font Size」の数字を変えると、編集中のファイルの文字サイズだけを変更できます

慣れてきたら、自分の好みや開発手法に合わせて、いろいろ設定してみてください。

各アプリとデータの確認

これで準備完了です。以下のフォルダやアプリが揃ったことを確認してください。見つからないものがある場合は、上に戻ってもう一度ダウンロード（インストール）してください。

Windowsの場合

「デスクトップ」に次の3つのアイコンがあることを確認してください

- `student`というフォルダ
- `Google Chrome`のアイコン
- `Visual Studio Code`のアイコン

macOSの場合

次のフォルダやアプリがあることを確認してください。

- 「**デスクトップ**」に`student`というフォルダ
- 「**アプリケーション**」フォルダに `Visual Studio Code` と `Google Chrome`
  なお、この2つのアプリを頻繁に使うことになりますので、（まだの場合は）**それぞれのアイコンを「Dock」にドラッグして、Dockから使えるようにしておいてください。**標準設定では、画面の下のほうにアプリのアイコンがいくつか並んでいます。適当な位置まで「ドラッグ」して「ドロップ」すると、その位置にアイコンが固定されます（ほかのアプリのアイコンの上ではなく、アイコンとアイコンの間に入れてください）。

VS Codeの基本

第1章から、VS Code（とブラウザ）を使ってプログラムを作っていきますが、初心者の方に基本的な手順を理解していただくために、次の図のような「ダイアログボックス」を表示する例を使って、ひと通りの操作と予備知識を紹介しておきます。

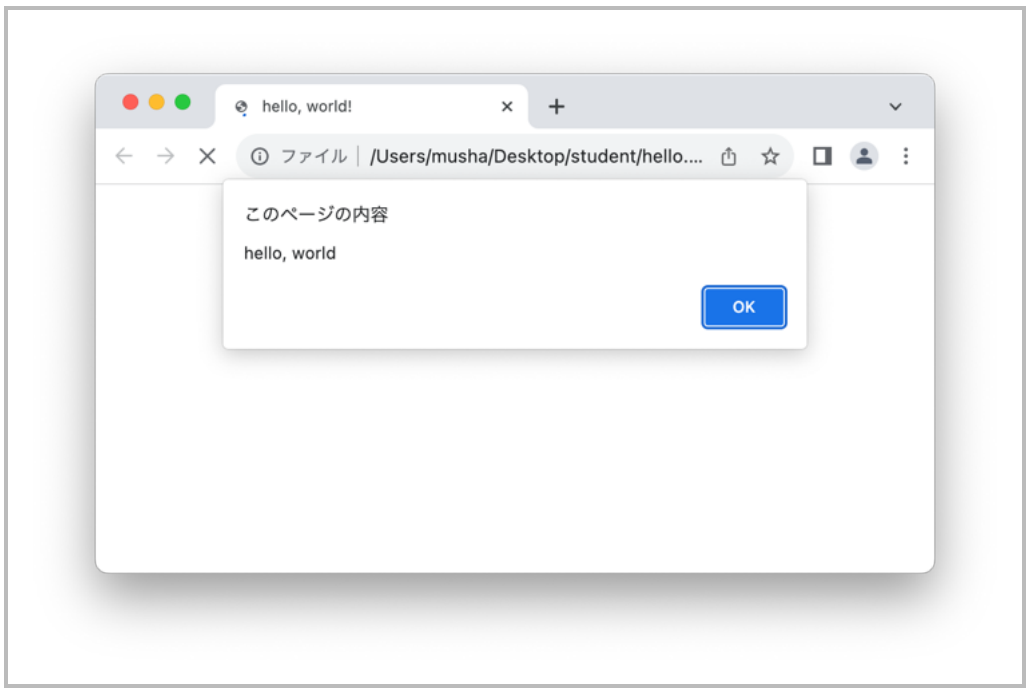


図4: ダイアログボックスの表示

まず次の手順に従って、例題が入っているフォルダをVS Codeで開いてください。

1. VS Codeを起動していない場合は、起動してください（**デスクトップ** の `Visual Studio Code` のアイコンをダブルクリック［ mac Dockのアイコンをクリック。これ以降、WindowsとmacOSで操作が異な

るところでは、まずWindows用の操作を示し、その後に mac のマークを付けてmacOS用の操作を示します）

2. VS Codeの［ファイル］メニューから［フォルダーを開く...］を選択して、**デスクトップ**にある`student`を選択して開きます。
   「このフォルダー内のファイルの作成者を信頼しますか？」というダイアログボックスが表示されたら、［はい、作成者を信頼します］を選択します
3. サイドバーに「エクスプローラー」が表示されるので、`STUDENT`→`hello.html`と選択します

すると図5のように**HTML**の「ソースコード」が右側の「エディタペイン」に表示されます。

■Note

図5に、VS Codeのウィンドウを構成する部分の名前も示しておきます。マイクロソフトによる公式の名称は「エディタペイン」が「エディタ」、「パネルペイン」が「パネル」のようですが、「エディタ」だけでは「テキストエディタ」と区別がつきにくいですし、「パネル」だけでも意味がよくわかりませんので、この本では「エディタペイン」「パネルペイン」と呼びます。

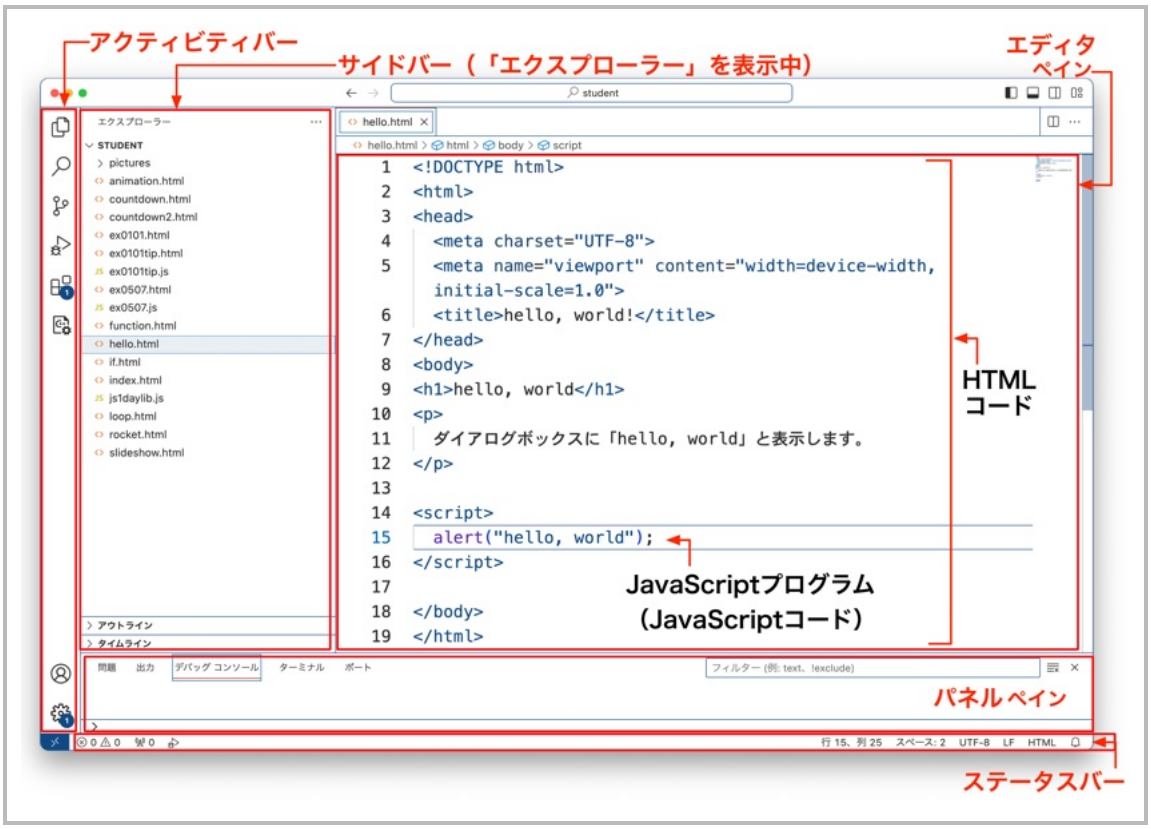


図5: HTMLファイルをVS Codeで開く

VS Codeのエディタペインに表示されているのが`hello.html`というファイルの内容です。HTMLという形式で書かれているので**HTMLファイル**と呼ばれます。エディタペインの左側にある1〜19の数字は、何行目かを示しています。自動的に表示されるのでプログラマーが入力する必要はありません。

図4のようにブラウザ（Chrome）に表示したものと、図5のようにエディタ（VS Code）に表示したものは同じHTMLファイルなのですが、「ブラウザで開くか」「エディタで開くか」によって表示のされ方が異なります。

- ブラウザに表示されるもの（図4）はHTMLコードをルールに従って解釈した結果 —— 一般ユーザーは通常ブラウザを使ってこちらを見ています。ブラウザで内容を編集することはできません。内容を変えたい場合はエディタを使って開く必要があります

- エディタに表示されるもの（図5）は素（す）のままのHTML —— HTMLで書かれた内容を表示して、編集（変更）できます。書かれている内容はHTMLの「ソースコード」あるいは単に「コード」と呼ばれます。HTMLファイルに書かれているのが「HTMLコード」です

HTMLファイルは、HTMLという規格に沿って書かれたファイルですが、HTMLファイルの中にJavaScriptのプログラムを「埋め込む」ことができ、このファイルもそうなっています。

エディタペインの14行目の`<script>`と16行目の`</script>`に囲まれた、15行目がJavaScriptのプログラムです。「JavaScriptコード」とも呼ばれます。「HTMLコード」の中に「JavaScriptコード」が埋め込まれているわけです。

VS Codeから実行

VS CodeでHTMLファイルを表示できましたので、今度は上でインストールしたブラウザChromeに、同じHTMLファイルを図4のように表示してみましょう。

1. VS Codeのエディタペインに`hello.html`が表示されている状態で、メニューから［実行］→［デバッグの開始］の順に選択します（あるいはF5。Windowsでは［実行］などのメニューが、メニューバーの［…］の下に隠れていることがあります。［…］を選択すると下に表示されますので［実行］を選択してください）
2. 図6のように選択肢が表示されるので［Web アプリ（Chrome）］を選択します

図6: Chromeを選択して実行

これでChromeが起動し、VS Codeの手前にウィンドウが開き、中にダイアログボックスが表示されます（図7）。

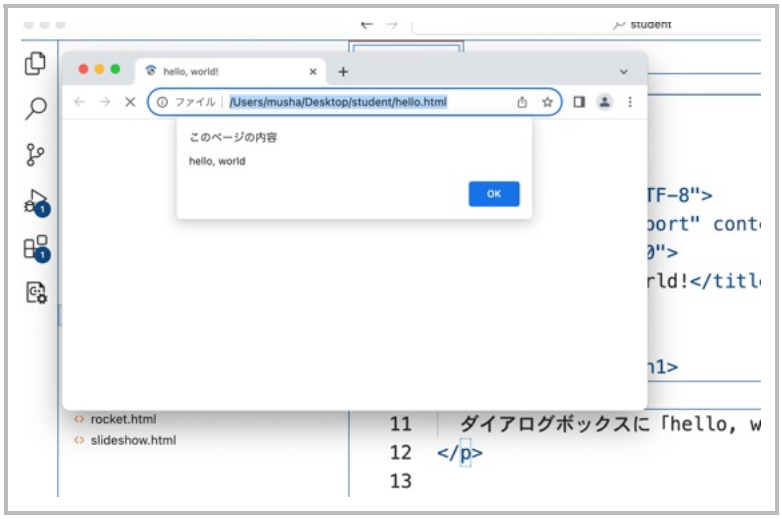


図7: HTMLファイルをChromeで表示

［OK］をクリックして、ダイアログボックスを閉じると、今度は「ドキュメント領域」に「hello, world」などの文字が表示されます。

それでは、一旦Chromeを終了して、HTMLファイルの内容を詳しく見ることにしましょう。Chromeを終了するには、Chrome側で［終了］メニューを選んでもよいのですが、VS Code側から図8に示す「停止」ボタンをクリックして、Chromeを終了することもできます（**終了する前に**、その左のボタンをクリックすれば再起動もできます。再起動ボタンのさらに左側に並んでいるボタンは「デバッグ」用のものです。第7章で説明します）。

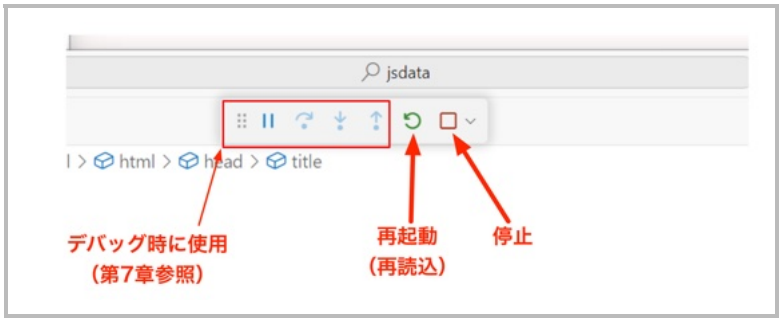


図8: VS Code側からChromeを終了することもできる

HTMLの概要とJavaScriptプログラムとの関係

`hello.html`の内容を見ながら、HTMLの概要を説明します。

まず、コードに説明を加えた図9を見てください。

```
1  <!DOCTYPE html>  ←DOCTYPE宣言（この文書がHTMLで書かれていることを示す
2  <html> ←htmlタグ（開始タグ）
3  <head> ←headタグ                                  ←ヘッド部（ページ全体に関する情報）
4    <meta charset="UTF-8"> ←metaタグ
5    <meta name="viewport" content="width=device-width, initial-scale=1.0"> ←metaタグ
6    <title>Hello, world!</title> ←titleタグ
7  </head>
8  <body> ←bodyタグ                                                  ←ボディ部
9  <h1>Hello, world!</h1>  ←h1タグ  ←見出し（<h1>から<h5>までのレベルあり）  （コンテンツ）
10 <p> ←pタグ
11   ダイアログボックスに「Hello, world」と表示します。  ←パラグラフ（段落）
12 </p>
13
14 <script> ←scriptタグ
15   alert("hello, world");  ←JavaScriptのプログラム
16 </script>                 <script>…</script>で囲む
17
18 </body>
19 </html> ←htmlの終了タグ（閉じタグ）
```

図9: HTMLファイルの構造。DOCTYPE宣言のあと、<html>…</html>で囲む

少し詳しく説明します。

- DOCTYPE宣言と`html`タグ —— HTMLファイルの先頭にはDOCTYPE宣言と呼ばれる文字列`<!DOCTYPE html>`を必ず入れます。この宣言につづいて、HTML文書を`<html>…</html>`で囲んで書きます。`<html>`や`<head>`、`<meta ...>`などの「<」と「>」で囲まれた文字列を**タグ**と呼びます。
- **ヘッド部**（head部。「ヘッダ」とも呼ばれる） —— `<html>`につづいて、`<head>…</head>`で囲まれたヘッド部を書きます。ここには、「`meta`タグ」を用いて、ファイルに保存する際の文字コード（UTF-8）や、特にスマートフォンでの表示を制御する指定（viewport（ビューポート））、そしてこの文書（ページ）のタイトルなどを書きます。
  当面、新しいページを作るときには、（このファイルなど）前に作ったファイルをコピーして`<title>…</`

title>に囲まれたタイトル文字列を変更して、ボディ部を置き換えていくようにするとよいでしょう
- **ボディ部**（body部） —— ページの内容（コンテンツ）を書く部分で、<body>…</body>で囲みます。「ドキュメント領域」に表示される内容は基本的にここに書かれます
- ファイルの最後にHTMLファイルが終わることを示す</html>を書きます。

ページの内容（コンテンツ）はボディ部に書かれます。その際に、タグを用いて、各部分の役割を指定します。

「tag（タグ）」は「荷札」「付箋（ふせん）」などの意味をもつ単語です。荷物に荷札を付けることで、その荷物の行き先や種類などを示しますが、HTMLファイルでは「タグ」によって、対象文字列の「文章中の役割」などを示します。

この例のボディ部で使われているタグは次の3種類です。

- h1タグ —— <h1>…</h1>。見出し（一番レベルの高い見出し）を表す（headerの**h**）。見出しには、h1からh5までの5段階のレベルがある
- pタグ —— … 。一般的な「段落（パラグラフ）」を表す（paragraphの**p**）
- scriptタグ —— <script>…</script>。この本の主題であるJavaScriptのプログラムを表す（JavaScriptのプログラムは「スクリプト（script）」とも呼ばれるため、scriptという名前がついている）。scriptタグは（したがって、JavaScriptのプログラムは）、ヘッド部に書くこともボディ部に書くこともできるが、上の例ではボディ部の最後に書かれている（特定の場所に書かなければいけないときもある）

ほとんどのタグは、上の3種類のタグのように<xx>...</xx>の形式でペアで用いられ、囲まれた部分（対象の文字列）が「その文書においてどのような役割するのか」「どのような形式で書かれるのか」といったことを示します。対象の文字列を開始する「<xx>」のほうを**開始タグ**、対象の終了を示す</xx>のほうを**終了タグ**あるいは**閉じタグ**と呼びます。

<meta>のように終了タグがないものもあります。開始タグと終了タグで囲まれた部分を対象とするわけではなく、単独で機能が果たせるためです。

■Note

厳密に言うと、「タグ」という言葉で指す範囲が微妙に異なる場合があります。たとえば、「pタグ」という言葉は、基本的には「 」を指しますが、「 」と「 」を合わせて「pタグ」と呼ぶ場合があります。

また、少しラフな場面では、「 」と「 」で囲まれた文字列まで含めて「pタグ」あるいは「pタグの部分」などという場合もあります。

JavaScriptの「コード」

では、この本の主題であるJavaScript（以下では「JS」と省略する場合があります）のコードを見ていきましょう。先頭に書いてある数字は、ファイルhello.htmlの「行番号」を表すものです（<script>タグは、hello.htmlの14行目から始まっています）。

```
14  <script>
15    alert("hello, world");
16  </script>
```

JavaScriptのコードは上の例のように<script>...</script>で囲まれます。JavaScriptのコードだけを別のファイルに書いて、そのファイルをHTMLファイルから読み込んで利用することもできるのですが、しばらくはこの例のようにHTMLファイルの中に直接書いていくことにしましょう。このほうが話が単純になりますし、それでいて利用できる機能は変わりません。

先に説明したように、14行目と16行目の<script>と</script>は、このペアで囲まれた部分がJavaScriptのプログラムであることを示すscript（スクリプト）タグです。

. . .

15行目が本当のJavaScriptプログラムで、ダイアログボックスを表示する**関数**になっています。関数は英語の発音をまねて「ファンクション（function）」と呼ばれることもあります。functionは「機能」「役割」などの意味をもつ単語です。関数はその英語名のとおりに、何か「機能」を提供してくれるものです。

第1章以降で、このようなコードをたくさん書いていただきます！

初心者のためのプログラミング超入門

実際にプログラムを作る前に「プログラミングとは何か」、ザックリと説明しておきましょう。公開中の動画では詳しく説明していますので、是非ご覧ください —— https://musha.com/sc/jsut

3つの側面

プログラミングをマスターするために身に着けなければない基本的な概念としてとして、次の3つがあげられます。

1. フロー制御 —— 計算の手順
2. データ構造 —— 情報の記憶手法
3. 演算子 —— 加減乗除や文字の連結、大小、真偽の判断などの操作

1. フロー制御

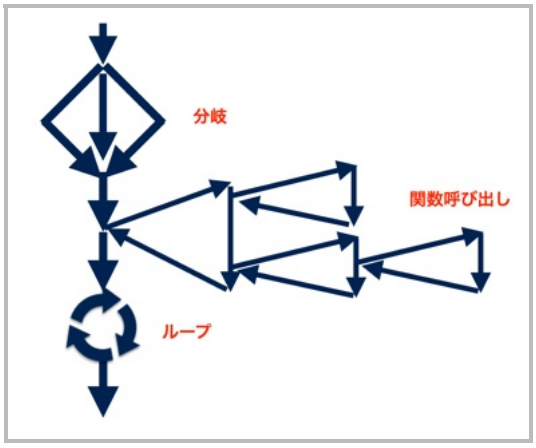
どのような処理をどのような順序で行うかを決めます。これがプログラミング作成の肝となる作業です。

次の3つが基本です。このほかにバリエーションがありますが、まずはこの3つを自由に使えるようになりましょう。

この問題集の前半（第9章まで）には、この3つの概念を理解するための問題が用意されています。

- 関数（第1章および第5章） —— 「レゴブロック」のようなもので、関数を呼び出すことでいろいろな機能を実行します。また、自作の関数を作ることで、一連の処理をまとめることができます。関数を組み合わせてプログラムを作っていきます
- 繰り返し（ループ。第3章および第8章） —— 人間は単純な繰り返しは嫌いですが、プログラムでは何万回でも何億回でも似たような処理を行います
- 分岐（第4章） —— 「場合分け」によって、問題を解決します

これを図にすると次のようなイメージになります。



ただ、上の図は単純化したもので、関数の中で分岐やループが使われることもありますし、分岐の中にループがあったり、逆にループの中に分岐があるといったように、さまざまに組み合わさって使われます。

2. データ構造

パソコンには「メモリ」があり、そこにいろいろな情報を記憶しながら、いろいろな処理を行っていきます。

- **変数**あるいは**定数**（第2章） —— いろいろな値を記憶したり、記憶しておいた値を取り出したりして処理に利用
- **配列**（第10章） —— まとめてたくさんの値を記憶
- **オブジェクト**（第11章） —— 配列よりも複雑な構造を記憶

単純な変数や定数はひとつの値だけを記憶しますが、配列やオブジェクトを変数や定数に記憶することもできます。さらに「オブジェクトの配列」「オブジェクトを要素としてもつオブジェクト」といったものも使います。

フロー制御をきちんと設計することは大切ですが、どのようなデータ構造を使うかも、問題を解く重要な鍵になります。

3. 演算子

いろいろな処理をするのに、数や文字列（文字が並んだもの）などを用いていろいろな「計算」をする必要があります。これに演算子を使います。

- 加減乗除と割算の余り（第7章） —— +、−、*、/、%
- 文字列の連結（第2章） —— +
- 比較（第3章、第4章など） —— <、>、<=、>=、===、!==
- 論理演算（第15章） —— &&（A**かつ**B）や||（A**あるいは**B）など

上で簡単に説明した一つひとつはそれほど難しいものではありませんが、複雑な問題を解決するためのプログラムを作成するには、これらの概念をうまく組み合わせなくてはなりません。

以下の各章で実際に問題を解くことで、各概念のイメージを掴みつつ、組み合わせ方をマスターしていきましょう。

目次

第1章　関数の呼び出し

■注意

エディタを使ったことがない人、HTMLファイルの編集をするのが初めての人は、最初にある「アプリとデータのダウンロード」から「初心者のためのプログラミング超入門」までの内容を読んでから、ここに戻って問題を解いてください。

それでは実際に問題を解いていきましょう。次の点に注意してください。

- 問題中にある**サンプルデータやその他の資料はサポートページ**（https://musha.com/sc/jsbes）**からダウンロードできます**。サンプルデータのZIPファイルを展開すると、studentというフォルダができます。そのフォルダの中に例題用のファイル（たとえば問題1.1で使うex0101.html）が入っています。Windowsの方はstudent.zipを右クリックして、「展開」しておくことをお忘れなく。**展開しておかないと、例題がうまく動きません**
- ほかの言語をご存じの方は、単純な問題はスキップしていただいてかまいません。ただし、新しい事柄の解説が含まれている場合があるので、問題文はお読みください
- コメント（行末の「`// ...`」や「`/* ... */`」の部分）は**入力する必要はありません**。説明です
- コード中の「●●」や「...」の部分は考えて埋めてください（「...」は省略を意味する場合もあります）
- 解答例は「あとがき」に記載されているページにあります（ダウンロードもできます）。まずはご自分でトライしてみてから、解答例をご覧ください
- 【写経】と書いてある問題は、書いてあるコードを書き写すだけで動作します。お坊さんが書き写すことでお経を学ぶように、書き写すことでプログラムを学んでください。ただし、ただ書き写すのではなく、何をしているのか、考えながら入力して実行してください
- 「バリエーション」はその上にあげた問題を少し変化させたものです。「こうしたらどうなるんだろう？」とか「こんなのもできるかな？」と自分なりのバリエーションを考えて、実際に動くものを作ってみると実力が上がるでしょう

■Note

解答例はウェブページで公開されています（ダウンロードも可能です）。詳細は、最後にある「あとがき」をお読みください。

1.1　ダイアログボックスを使って「こんにちは」と表示せよ（ダウンロードしたstudentフォルダにあるex0101.htmlを利用する）【写経】

ダウンロードしたstudentフォルダにあるex0101.htmlを利用する。

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ダイアログボックスの表示</title>
</head>
<body style="margin: 5rem; font-size: xx-large;">
  <p>
    ダイアログボックスに「こんにちは」と表示しました。
  </p>
<script>
  "use strict"; // これがあるとエラーが見つけやすくなる。先頭に書きましょう。詳しくは14章参照
```

```
  alert("こんにちは");
</script>
</body>
</html>
```

■メモ

実践では（製品として公開するシステムをほかの人と共同で作成したりする場合には）通常、HTMLのコードとJavaScriptのコードは別のファイルに書きます。そのほうが、共同作業がしやすいし、（ファイルが大きくなると）管理が楽です。上のコードは、たとえば次のように2つのファイルに書くことができます。

```
// ファイル ex0101tip.html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ダイアログボックスの表示（別ファイルにJSのコードを書く方法）</title>
</head>
<body style="margin: 5rem; font-size: xx-large;">
  <p>
    ダイアログボックスに「こんにちは」と表示しました。
  </p>
<script src="ex0101tip.js"> <!-- src= でJSファイルを指定 -->
</script>
</body>
</html>
```

```
// ファイル ex0101tip.js
"use strict";
alert("こんにちは");
```

この問題集の例は短いものが多いのでJavaScriptのコードもHTMLファイルに書いている場合が多いですが、実践では（この問題集を終える頃には原則として）JavaScriptのコードは別ファイルに書くようにしましょう。

1.2　alertの引数（"こんにちは"）をいろいろ変えてみて、違うメッセージが表示されることを確認せよ

```
alert("●●"); // ●● を適当に変える
```

1.3　alert(...)の代わりにdocument.write(...)を使い、ダイアログボックスではなくブラウザのウィンドウ内（「ドキュメント領域」と呼ぶことにする）に表示されることを確認せよ【写経】


```
document.write("こんにちは"); // alert(...)の行だけを置き換える。他の行は同じ
```

■メモ

実は、実践で`document.write`を用いることは（ほとんど）ありません。この問題集ではしばらく`document.write`を用います。HTMLを知っている人にとって、とてもわかりやすいと思うからです。実践では、`document.write`を使っているところを、後で紹介する関数などに置き換えます。

なぜ実践で`document.write`を使わないかについては、第6章の最後のコラムで説明します。

1.4 HTMLのimgタグを使ってstudentフォルダの中のpicturesフォルダにある画像を表示せよ（ダウンロードしたstudentフォルダにあるimage-tag.htmlを利用する）【写経】

```
<!DOCTYPE html>
...
<body>
  <img src="pictures/picture000.jpg">
</body>
</html>
```

- imgタグはドキュメント領域に画像を表示するためのHTMLタグ
- src=のあとに引用符で囲んでファイル名を指定する
- 画像があることを確認
- うまくいかない場合はHTMLのファイルと画像ファイルとの位置関係を確認すること。studentフォルダの中にHTMLファイルを置けば上のコードで動くはず

■警告

画像がうまく表示されないときは次をチェック。

- ファイル名が間違っていないか、スペルミスはないか
  - フォルダ名は`picture`ではなく`pictures`（複数の画像が入っているので）
  - ファイル名は`picture000.jpg`（ひとつのファイルについて言っているので`pictures`ではない）

■メモ

HTMLのタグについては、登場するたびに説明しますが、**studentフォルダの中にあるtags.htmlにこの講座でよく使うタグ（や属性）に関する説明があります**。

studentフォルダの中にあるtags.htmlファイルのことを`student/tags.html`のように「`/`」（スラッシュ）を使って表現します。「`/`」が「サブフォルダ」を表すわけです。Windowsでは「/」の代わりに、「\」（バックスラッシュ）あるいは「¥」を使う場合もあります。

なお、「フォルダ」は「ディレクトリ」と呼ばれる場合もあります。「ディレクトリ」はmacOSのベースになったUnixに由来する用語です。

バリエーション

- ファイル名を変えて、いろいろな画像が表示されることを確認せよ

1.5 imgタグにstyle属性を指定して、同じ画像を640 px（ピクセル）の幅で表示せよ【写経】

```
// img タグのみを示す。ほかの部分は省略
<img src="pictures/picture000.jpg" style="width: 640px;">
```

- styleを指定することで、画像の大きさなどを指定できる。ほかのタグでstyleを指定すると、文字の大きさや色、背景の色、位置揃えなどさまざまな「属性」を指定できる（student/tags.htmlを参照）
- 画像の高さ（height）を指定することもできるが、widthだけを指定した場合、同じ割合で高さも調節される

1.6　imgタグにstyle属性を指定して、同じ画像をブラウザの横幅の25%の大きさで表示せよ

- 上の問題の640pxのところを25%に変えればよい

バリエーション

- pxや%の前の数値を変えて、画像をいろいろな大きさで表示してみよ

1.7　JavaScriptのdocument.writeを使ってstudentフォルダの中のpicturesフォルダにある画像を表示せよ【写経】

- document.writeを使うと、HTMLコードをJavaScriptを使って出力できる。この問題ではimgタグを出力する

```
document.write(`<img src="pictures/picture000.jpg">`); //「`」の代わりに「'」でもOK
//  「`」はバッククオート。多くのキーボードでは「Shift+@」。英語キーボードでは左上にある
```

次のようにしてもよいが、引用符は必ずペアで用いることに注意。

```
document.write("<img src='pictures/picture000.jpg'>");
```

次のように書くのは、引用符の対応が取れていないので、どれも間違い（エラーになってしまう）。

```
document.write("<img src="pictures/picture000.jpg'>'); //「src=」までで文字列が終わってしまう
document.write(`<img src="pictures/picture000.jpg`>");
document.write(`<img src="pictures/picture000.jpg'>`); //  「"」に対応する「"」がない
```

ファイル名などを属性として指定するタグを書くときは、一番外を`...`にすれば、内側はHTML単独のときと同じように書けるので、次のいずれかがおすすめ。

```
document.write(`<img src="pictures/picture000.jpg">`); // ファイル名を "..." で指定
document.write(`<img src='pictures/picture000.jpg'>`); // ファイル名を '...' で指定
```

バリエーション

- 連続して書くことで、複数の画像が表示されることを確認せよ。スタイルも指定してみよ

```
document.write(`<img src="pictures/picture000.jpg" style="width: 25%;">`);
document.write(`<img src="pictures/picture001.jpg" style="width: 25%;">`);
document.write(`<img src="pictures/picture002.jpg" style="width: 25%;">`);
```

第2章　文字列の連結とconsole.log

■メモ

あとで復習できるように、問題ごとに別のファイル名で保存するとよいでしょう。たとえば、次の問題はex0201.htmlなどといったファイルに保存しておきましょう。

前の問題と同じようなコードを使うのならば、前の問題から同じ（ような）部分をコピー・ペースト（「コピペ」）しましょう。**入力すると間違えるので、できるだけ「コピペ」しましょう**。プログラムは1文字でも間違えば動かなくなります！

2.1　次のコードを読み、何が出力されるか予想してから、入力して実行せよ

```
1  let a = "武田"; // '武田' でも `武田` でもOK
2  let b = "信玄";
3  let c = a + b; // aとbに記憶されている文字列が、連結されてcに代入される
4  document.write(c) // cの値がドキュメント領域に書かれる
```

1行目は**変数**の**宣言**と変数への**代入**を行う**文**（**代入文**）である。

この文によって、コンピュータの本体に備わっている「メモリ」（情報の記憶場所）の特定の領域（**変数**）に「a」という名前を付け、そこに「武田」という文字の並びを記憶する。この操作を「代入（assignment）」と呼ぶ。

文字列を「連結」するには「+」を使う（「+」は普通の足し算にも使われる）。上のコードでは「武田信玄」がドキュメント領域に書かれることになる。

■メモ

「a = "武田"」の「=」は数学で使う「イコール」とは意味が違います。数学で使う「=」は左側と右側が等しいという**状態**を表します。これに対して、ほとんどのプログラミング言語で「=」は、aという名前がつけられた記憶域（メモリ）に「武田」という値を記憶するという動作を表します。結果として、記憶域内の状態の変化を引き起こします。次のように書いたほうがわかりやすいでしょう。

```
a ← "武田"
```

キーボードから「←」をワンタッチで入力できないので、代わりに「=」が用いられていると考えましょう。

2.2　次の文字列処理の結果xの値がどうなるかを予想し、結果を出力して予想と比較せよ

```
let x = "JavaScript";  // let x = 'JavaScript';    でもOK
x += "入門"; // xの後ろに"入門"が追加される
let y = "実習編もどうぞ -- ";
x = y + x;
x = "「" + x + "」"; //数字と文字列、文字列と文字列をつなげるには「+」
// なお、この下の問題にある「バッククオート（`...`）」も使える
document.write(x);
```

2.3　上の2.1の各文の後ろにconsole.log()（あるいはalert()）を入れて、途中経過を表示しながら、変数の値を追ってみよ

```
let a = "武田";
let b = "信玄";
let c = a + b;
document.write(c)
```

↓

```
let a = "武田";
console.log(a);
```

```
let b = "信玄";
console.log(b);
let c = a + b;
console.log(c);
document.write(c)
```

- 実行するとVS Codeの下のほうの「パネルペイン」の「デバッグコンソール」のタブに結果が表示される

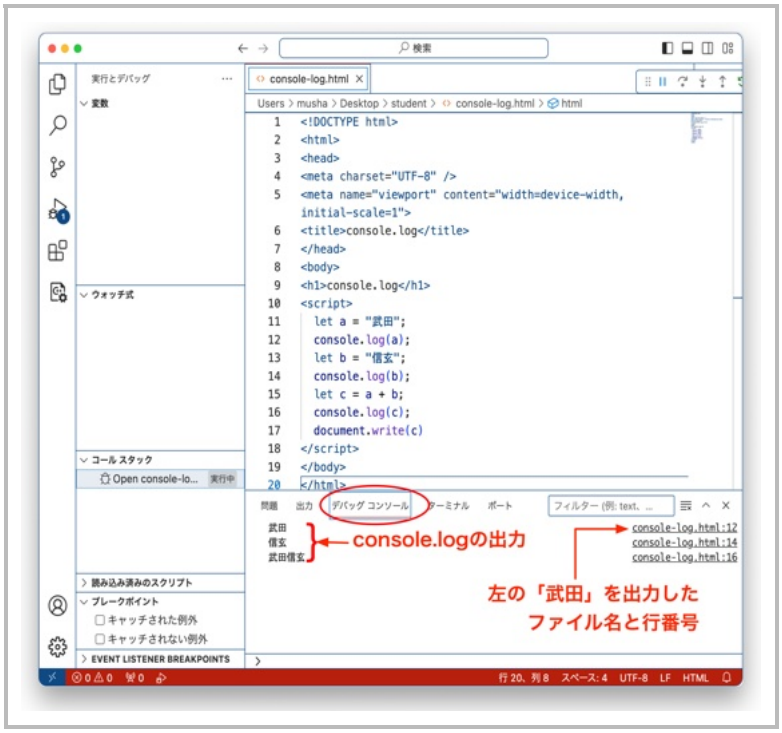


図2.1: console.logの出力

- Chromeでも「コンソール」を表示するすることで同じ出力を見ることができる（次のいずれかを実行。他のブラウザにも同じような項目がある）
  - ドキュメント領域で右クリック→［検証］→［コンソール］【mac 「右クリック」を設定してない場合はcontrol+クリックしてから→［検証］→［コンソール】
  - ショートカットキー —— Ctrl+Shift+J【mac ⌘+option+J】
  - メニューから［表示］→［開発/管理］→［JavaScript コンソール］

console

2.4 次のバッククオート（`）を使った文字列処理の結果xの値がどうなるかを予想し、結果を出力して比較せよ

```
1: let x = "JavaScript";
2: let y = "入門";
3: x += y;
4: x = `『${x}』`; //  「`」はバッククオート（多くのキーボードでは「shift+@」で入力）
5: //   `...`の中では、${...}で囲むと変数を使ったり、計算式を使ったりできる
6: document.write(x);
```

上のリストの4行目のバッククオートで囲まれた文字列（`『${x}』`）の中には変数が含まれている。このような表現を**テンプレートリテラル**という。**${...}**の「...」の部分にある変数はその値が計算されて文字列に置き換わる。テンプレートリテラルは「可変部分付きの文字列」である。これに対して、普通の文字列は変化しない、いつも決まったものになる（「リテラル」と呼ばれる。この場合は文字列のリテラルなので「文字列リテラル」）。

2.5　次の計算の結果がどうなるかを予想し、実行して予想と比較せよ

```
const a = 2; // 「let a = 2」でもよいが、値が変わらない場合はconst（定数）を使うほうが、なおよい
const b = 3;
document.write(`a*b=${a*b}`);  // `...` の中で計算をしている
// constはletとよく似ているが、それ以後に変更しない場合に用いる
// letを用いても結果は同じだが、この値は変わらないことを読む人（将来の自分も含む）に意識してもらえる
```

constはletとほとんど同じ働きをするが、constで宣言しておくと、その変数の値が変わらないことが明確になる。

なお、後半のほうで登場するオブジェクトをconstで宣言した場合、そのオブジェクトの構成要素（プロパティ）の値は変更できてしまうので、必ずしもこれが当てはまらない。このため、constは単純な変数について使う場合が多い。

2.6　次のコードを実行して画像が（300 px の幅で）表示されることを確認せよ【写経】

```
const ファイル名 = "pictures/picture000.jpg"; // 変わらない値はletよりconstを使うほうがよい
const 画像タグ = `<img src="${ファイル名}" style="width: 300px;">`;
document.write(画像タグ);
```

■メモ

変数、定数、関数などの名前（まとめて「**識別子**（しきべつし）」と呼びます）には日本語も使えます。たとえば、上のコードの「**ファイル名**」や「**画像タグ**」などがその例です。

ネットで全世界に公開するような場合や、プロジェクトに日本語が理解できない人がいる場合は、識別子には英語を使う必要があります。また、「日本語の識別子は使わないほうがよい」と考える開発者も少なくありません。

この問題集では「わかりやすさ」を優先して日本語の識別子を積極的に用います（自分でコードを書く際には、もちろん英語を使ってもOK）。

バリエーション

- 上のコードのwidthの値をいろいろの変更し、画像の大きさを変えてみよ。たとえば、200px, 50%, 25%, 20rem, ...など（remは文字と同じ幅を表す。20remは20文字分）

第3章　ループ（その１）

3.1　「私は○○が好きです。」と24回出力するプログラムをforループを使って作成せよ。○○は適当に変え、「。」のあとには改行を入れること【写経】

```
for (let i=1; i<=24; i++) {
  document.write("私は○○が好きです。<br>"); // HTMLファイルでは改行するのに<br>が必要
  // この場合は "..." でも '...' でも`...`でも同じ
}
```

■メモ

上のiのように、ループを制御する変数のことを「ループ変数」と呼びます。ループ変数の値は整数（integer）になることが多いので、i（とかそれに続くj、kなど）がよく使われます。

3.2　前の問題の出力の、各行の前に、1から順番に番号を振れ【写経】

```
for (let i=1; i<=24; i++) {
  document.write(`${i}. 私は○○が好きです。<br>`);
  // 文字列中で変数を使いたい場合は`...`（バッククオートの組）で囲む（テンプレートリテラル）
}
```

バリエーション

- 上の問題を1から80まで、80行出力するように変更せよ
- 上の問題を1から1万まで、1万行出力するように変更せよ

3.3　前の問題で、出力するHTMLコードをすべて変数に保存しておいて、最後に一度だけdocument.writeを呼ぶように変更せよ【写経】

```
let x = ""; //  「空文字列」に「初期化」しておく
for (let i=1; i<=24; i++) {
  x += `${i}. 私は○○が好きです。<br>`; // できた文字列を x の後ろに次々と追加して行く
}
document.write(x); // 最後に一度だけ書き出す。実はこうした方が効率がよい（実行が少し速い）
```

3.4　前の問題で、途中で作られるHTMLコードをconsole.logで確認しながら実行するようにしてみよ【写経】

```
 1: let x = ""; //  「空文字列」に「初期化」しておく
 2: for (let i=1; i<=24; i++) {
 3:   x += `${i}. 私は○○が好きです。<br>\n`; // できた文字列を x の後ろに次々と追加して行く。
 4:   console.log(`--- ${i}回目のループ ---`); // 各回の出力冒頭の「見出し」
 5:   console.log(x);  // xの途中経過を表示
 6: }
 7: document.write(x);
```

- コード3行目の「\n」は改行を表す。consoleに表示するときは、各行の最後に改行を付けておいたほうが見やすい
- 「\」はWindowsでは「¥」と表示されることが多い。Macのエディタでは「option+¥」で入力できることが多い。入力方法がわからない場合は、サポートページからダウンロードできるstudentフォルダのindex.htmlにあるので、それをコピー・ペースト
- consoleの出力を最初に戻って追ってみる。ループするごとに、出力が1行ずつ増えていることを確認

■メモ

このように、デバッグの際に log（ログ）（途中経過）を表示する方法は、どのような環境でも、どのような言語を使っているときでも使えるものです。

なお、デバッガの際には**デバッガ**を使うのもおすすめです。デバッガについては第7章の最後の問題で説明します。

第4章　分岐

4.1　大吉か大凶が同じ割合で出るおみくじのプログラムを作れ（studentフォルダのif.htmlにあり）【写経】

```
 1: let x = Math.random();  // 0以上1未満の小数を返す
 2: if (0.5 <= x) {
 3:   document.write("大吉");
 4: }
 5: else {
 6:   document.write("大凶");
 7: }
```

なお、次のように変数に一旦運勢の文字列を記憶しておいて出力したほうが、変更に強くなる。

```
let x = Math.random();  // 0以上1未満の小数を返す
if (0.5 <= x) {
  x = "大吉";
}
else {
  x = "大凶";
}
document.write(x);
```

たとえば、「今日のあなたの運勢は○○です」と表示したくなったら、すぐ上のコードならば最後の1文を下のコードのように変えればすむが、最初のプログラムだと3行目と7行目のdocument.writeを書き換える必要がある（次の問題も参照）。

```
  document.write(`今日のあなたの運勢は${x}です。`);
```

4.2　大吉、小吉、末吉、凶が同じ割合で出るおみくじのプログラムを作れ

```
let x = Math.random();  // 0以上1未満の小数を返す
if (0.75 <= x) { // 4つを同じ割合で出すようにするなら、たとえば
  x = "大吉"; // おみくじの文字列をxに記憶しておく
}
else if (0.5 <= x) {
  x = "小吉";
}
else if (●●) {   // ●●を変える
  x = "末吉";
}
else {
  x = "凶";
}
document.write(x); // 上でおみくじの文字列を作って、ここで書き出す
```

4.3　今日の運勢（大吉、中吉、小吉、末吉、凶、大凶のいずれか）を次の割合で表示するプログラムを作成せよ

- 「大凶」のでる確率10% -- Math.random()の値が0.1以下の時
- 「凶」のでる確率20% -- Math.random()の値が0.3以下の時
- 「末吉」のでる確率20% -- Math.random()の値が0.5以下の時
- 「小吉」のでる確率20% -- Math.random()の値が0.7以下の時
- 「中吉」のでる確率20% -- Math.random()の値が0.9以下の時
- 「大吉」のでる確率10% -- その他の場合

4.4　前の問題と同じ割合で今日の運勢（大吉、中吉、小吉、末吉、凶、大凶のいずれか）を表示するプログラムを作成せよ。picturesフォルダにある該当する画像（omikuji_chuukichi.pngなど）を（も）表示せよ

第5章　自作の関数とタイマー

システムが提供してくれる「組み込み関数」のほかに、「自作の関数」（「**ユーザー定義関数**」のほうが一般的です）を作ることもできます。

5.1　引数xに対してxの2乗を返す関数「自乗を計算」を作り、これを呼び出して10, 4, 9, 12, 128の2乗を計算せよ【写経】

```
function 自乗を計算(x) {
  return x * x;
}

document.write(自乗を計算(10) + "<br>");
document.write(自乗を計算(4) + "<br>");
document.write(自乗を計算(9) + "<br>");
...
```

5.2　2つの引数を与えられて、その平均を返す関数「平均を求める」を作り、これを呼び出して次の組の平均を求めよ ―― 10と6、4と4、130と80、5.2と4.6

```
function 平均を求める(a, b) {
  return ●●  // ←●●を変える（考えてください！）
}

document.write(平均を求める(10, 6) + "<br>"); // console.log(...); でも確認できる
document.write(平均を求める(4, 4) + "<br>");
document.write(平均を求める(130, 80) + "<br>");
document.write(平均を求める(5.2, 4.6) + "<br>");
```

5.3　ページを読み込んでから3秒後に、「3秒経過しました」というダイアログボックスを一度だけ表示せよ

```
"use strict";
// 関数func1()を3秒ごとに呼び出す（この場合は1回で止める）
const タイマーID = setInterval(func1, 1000*3);

function func1() {
  alert("●●") // ダイアログボックスを表示
  clearInterval(タイマーID); // タイマーを止める
}
// 長くすれば「休憩アラーム」にできる
```

- setIntervalは、第1引数に指定された関数を、第1引数に指定された時間間隔（1/1000秒単位）で繰り返す
- clearIntervalはタイマーを止める。ひとつのプログラムで複数のタイマーを使えるので、どのタイマーを止めるかを「タイマーID」で指定する。タイマーIDはsetIntervalの戻り値として返される値

5.4　上の問題のfunc1を無名関数に変更せよ

その場で関数を作りたい時、名前はいらない時は**無名関数**（**匿名関数**）を使うと便利。上のfunc1を「展開」してその場に書く。

```
"use strict";
const タイマーID = setInterval(
  function() {  // ここから *END* までが関数の定義（第1引数）
    alert("●●");
    clearInterval(タイマーID);
  },  // *END* 「}」までが関数の定義
  1000*3);  // 1000*3 は第2引数
```

5.5　上の問題の無名関数をアロー関数を使って書け

無名関数の「function(【引数】)」を「(【引数】) =>」の形に省略できる。この形式の関数を**アロー関数**という。なお、アローを使わない無名関数とアロー関数には少し違いがある。詳細は次のページなどを参

照 —— https://qiita.com/suin/items/a44825d253d023e31e4d

```
"use strict";
const タイマーID = setInterval(
  () => {  // この行から *END* までが関数の定義（第1引数）
    alert("●●");
    clearInterval(タイマーID);
  },  // *END* 「}」までが関数の定義
  1000*3);  // 1000*3 は第2引数
```

5.6　ページを読み込んでから3秒後に、「3秒経過しました」というダイアログボックスを表示し、以降3秒経過するたびに同じダイアログボックスを表示せよ。ただし3回でやめるものとする

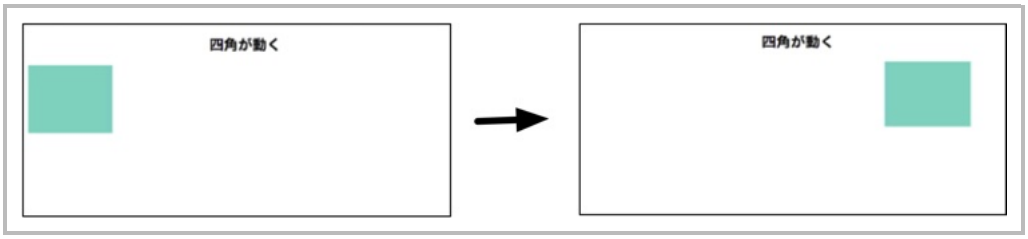
```
"use strict";
let カウンタ = 0;
const タイマーID = setInterval(ダイアログボックスを表示, 1000*3);

function ダイアログボックスを表示() {
  alert("3秒経過しました")
  カウンタ++;
  if (3 <= カウンタ) { // 3回表示したら
    ●● // ←★★要変更★★  clearInterval()を呼んでタイマーを止める。上の問題を参考に
  }
}
```

バリエーション

- 無名関数やアロー関数を使って書いてみよ

第6章　アニメーション

6.1　画面上に長方形を表示し、その長方形を左から右に動かすプログラムを作れ



例（方法はいろいろあるが一例。animation.htmlにあり。「●●」を要変更）

- CSSで背景色を使って四角を書く
- マージンをJavaScriptで変化させる

コード

```
...
<body>
<div id="rectangle" style="background-color: #6DBBA1;
         width: 200px; height: 180px;">
</div>

<script>
"use strict";
let カウンタ = 0; // 左端からの距離（px（ピクセル）単位）
const タイマーID = setInterval(四角を動かす, 10); // 時間は適当な長さに。1/1000秒単位

// setIntervalで、連続して「四角を動かす」を呼び出す
function 四角を動かす() {
  カウンタ++;
  const 四角 = document.getElementById("rectangle");
  // <div id="rectangle">...</div>の下にJavaScriptのコードを書くこと！
  // "rectangle"のIDをもつ部分が、この上にないとそのオブジェクトを取得できない
  四角.style.marginLeft = `${カウンタ}px`;
  // CSSのmargin-left（左端から四角までの距離）を指定
  if (●● < カウンタ) { // 終了させる　←★★要変更★★ 数値を入れる
    clearInterval(タイマーID);
  }
}
</script>
</body>
</html>
```

6.2　前の問題の長方形を5倍の速度で動かせ。

- たとえば、マージンの指定を5倍にする

6.3　画面上に長方形を表示し、その長方形を左から右、右から左と交互に動かすプログラムを作れ

方法（一例）

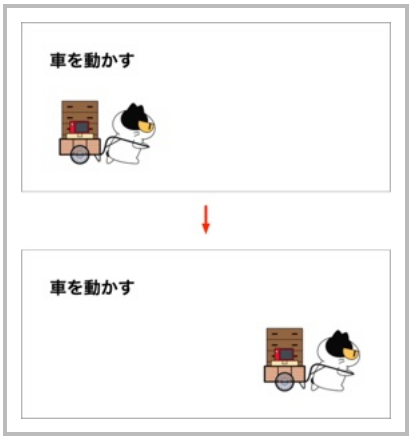
- 右側に来たら、方向を変える → marginの値を減らしていく

コード例

```
"use strict";
const 倍率 = 5; // 一度に動かすpx数
const 右端 = 800 / 倍率; // px単位
const 四角 = document.getElementById("rectangle");
let 座標 = 0;  // 現在の座標（左端から距離。px単位）
let 方向 = 1;  // 現在の方向。右が左か。1が右方向 -1が左方向
const タイマーID = setInterval(四角を動かす, 10);

function 四角を動かす() {
  if (0 < 方向) {
    座標++;
```

```
    四角.style.marginLeft = `${座標*倍率}px`;
    if (右端 <= 座標) { // 右端に到達した
      方向 = -1; // 次からは右から左に戻る
    }
  }
  else {  // 左に戻っている時
    座標--;
    四角.style.marginLeft = `${座標*倍率}px`;
    if (座標 <= 0) { // スタート地点に戻った
      方向 = 1; // 次からは左から右に進む
    }
  }
}
```

6.4　画面上に車の画像（pictures/car1.png）を表示し、その車を左から右に動かすプログラムを作れ



- 背景ではなく、画像のマージンを変化させる
- たとえば、次のようなタグを用いて画像を表示して、idがcarの ... のマージンを増やしたり減らしたりすればよい（このようにすれば、四角を動かす問題と同じ手法が使える）

```
<div id="car">
  <img src="pictures/car1.png">
</div>
```

6.5　車を左から右、右から左と交互に動かすプログラムを作れ（pictures/car1.pngとpictures/car2.pngの画像を使う）

- 方向を変えるところで画像も変える（問題6.3参照）

6.6　上の問題で、ウィンドウの右端より少し左で折り返すようにせよ

- window.innerWidth でウィンドウの幅（ピクセル単位）がわかる
- これがピクセル単位なので、移動する単位もピクセルにしたほうがよい

6.7　上の問題で、ウィンドウの大きさを変えたときにも右端より少し左で折り返すようにせよ

- ウィンドウの大きさを変えるとresizeイベントが発生するので、resizeイベントを処理すればよい

```
window.addEventListener("resize", () => {
  // ウィンドウのサイズが変わったときの処理をここに書く
});
```

6.8　画面上に長方形を表示し、その長方形を左から右に動かすプログラムを作れ。長方形が移動するとき、色がランダムに変わるものとする（★ちょっと難問★）

```
<script src="js1daylib.js"> <!-- JSのコードを読み込む -->
</script>
<script>
"use strict";
let カウンタ = 0;
const タイマーID = setInterval(四角を動かす, 10);
let 右端 = 600;

function 四角を動かす() {
  カウンタ++;
  const 四角 = document.getElementById("rectangle");
  四角.style.margin = "0 0 0 " + カウンタ + "px";

  if (カウンタ % 20 === 0) { // 20回に1回色を変える（カウンタが20で割り切れるときに実行）
    四角.style.backgroundColor = ランダムな色を取得();
    // "#AB0033"のような文字列で色を指定する。「#」のあとに、2桁ずつ16進数で指定
    //  R（red）G（green）B（blue）の成分を指定する
  }

  if (右端 < カウンタ) {
    clearInterval(タイマーID);
  }
}

// "#AB0033"のような文字列を返す
function ランダムな色を取得() {
  let 色 = "#";
  for (let i=0; i<6; i++) { // 6回繰り返す
    let x = ランダムな整数を生成(0,15); // 上で読み込んだjs1daylib.jsで定義
    x = x.toString(16);
    色 += x;
  }
  // console.log(色);
  return 色;
}
</script>
```

document.writeについて

1. 実行速度

入出力操作（画面表示、ファイル読み込みなど）は実行に時間がかかることが多いので、document.writeも何度も呼び出すのは効率が悪い。その都度呼び出すのではなく、表示したいものをまとめておいて一度に出力するほうが効率がよい。

この講座では初心者にもわかりやすくするために、特に最初のほうではその都度document.writeを呼ぶ例を示したが、ある程度以上の規模のプログラムを作るときは実行速度も考慮する必要がある。たとえば次のように書き出すコードを変数に追加する形で記憶しておいて、最後にまとめて出力するようにしたほうが実行が速くなる。

```
let x = "....";
x += "...";
...
x += "...";
document.write(x);
```

2. ブラウザによる非同期処理の挙動の違い

document.writeの非同期の処理（タイマーやネットワーク経由の処理など即座に実行されない処理）の動作は（残念ながら）ブラウザによって異なる。このため非同期の処理の場合document.writeは使わないほうがよい。

非同期の処理などを行う場合は、たとえば次のように ... とdocument.getElementByIdの組み合わせなどを使って、その部分のinnerHTMLに代入してしまえばdocument.writeを使わずに済む。

```
【htmlファイル】
...
<div id="yyy">
  <!-- ここに書き込む -->
</div>
...
<script src="xxx.js"></script>

【JSファイル】
"use strict";
let x = ...;
x += ...;
..
obj = document.getElementById("yyy"); // ID "yyy"のオブジェクトを得る
obj.innerHTML = x; // 「ここに書き込む」のところに書き込む
```

3. document.writeの評判が悪い

次のようなスクリプトを書くと、document.write によって外部のスクリプトを動的に挿入することができるが、このような処理をするとスクリプトを読み込む間、表示用の処理ができないので、コンテンツの表示がかなり遅れる。

```
document.write('<script src="https://www.xxx.com/yyy.js"></script>');
```

このような「動的なスクリプトの挿入」など遅延の原因になる処理を行わなければdocument.writeを使っても明らかな不具合は生じないと思われる。ただし、一概にdocument.writeを「使うな」あるいは「非推奨」とする傾向がある。たとえば、Google Chromeでは「使うな（Avoid using document.write)」といったメッセージが表示されるようになった（おそらくこうしたほうがChrome側の処理が簡単だからだと思われる。詳細はhttps://developers.google.com/web/updates/2016/08/removing-document-write）。

上の2.にあげたような方法を使えばdocument.writeを使わなくても済むので、将来、ウェブに公開するようなスクリプトを書く場合にはdocument.writeは使わないようにしたほうがよい。練習に用いたり、自分用のアプリを作るのには用いてもかまわない。

いろいろ理由をあげたが、「プロなら本番環境ではdocument.writeは非効率だから使うな。ほかに方法がある」ということだ。if文やfor文の機能を覚えたりするために練習用に使うのはかまわない（ブラウザでサポートされなくなれば話は別だが）。HTMLを知っている人にとって、こんなにわかりやすい関数はないのだから。

第7章　演算子（加減乗除）

7.1　次の計算の結果を予想し、結果を出力して比較せよ

```
let x = 320; // xという名前の変数（情報の記憶場所）に320を記憶する
             //  「=」の意味は下記「メモ」参照。
             //  「;」は「文」の区切り。行の最後ならば省略してもよい
x = x - 20;  // xに記憶されている値からから20を引いて、再度xに記憶する
x += 33; //  「x = x+33;」と同じ。 xに33を足してそれをx記憶する
x -= 20; //  「x = x-20;」と同じ。 xから20を引いて、それをx記憶する
x++; //  「x += 1」でも「x = x+1」でも同じ。xに1を足してそれをxに記憶する
     // 1を足す操作はよく使われるので特別な演算子(オペレータ)++が用意されている
     // 通常は、「x++」を使う
x--; //  「x -= 1」でも「x = x-1」でも同じ。xから1を引いてそれをxに記憶する
document.write(x); // ドキュメント領域にxの値を出力する
</script>
</body>
</html>
```

■メモ

第2章でも説明しましたが、「=」は数学で使う「イコール」とは意味が違います。次のようなイメージで、xという名前がつけられた記憶域（メモリ）に値を記憶するというアクション（記憶域内の状態の変化）を引き起こします。

```
x ← 320
```

7.2　上の問題と同様、次の計算の結果を予想し、結果を出力して比較せよ

```
let x = -3;
x -= 20;
x += 3;
x--;
document.write(x);
```

バリエーション

- +, -, *, /, %（余り）を使っていろいろな計算をしてみよ

7.3　今日のラッキーナンバー(0〜9までのいずれか）を表示するプログラムを作成せよ。0〜9までの数字はランダムに出現するものとする【写経】

```
// Math.random() -- 0以上1未満の小数をランダムに返す
// Math.floor(x) -- xを整数に切り下げる
// document.write("xxx") -- xxx をHTMLコードとして出力
//
let x = Math.random();  // 0 ≦ x < 1
x *= 10; //  「x = x*10」 と同じ。自分を10倍したものを、自分（x）に記憶。 0 ≦ 10x < 10 になる
x = Math.floor(x); // 小数点以下を切り下げる → 整数（小数点の付かない値）になる
document.write(x);
```

7.4　今日のラッキーナンバー(1〜10までのいずれか）を表示するプログラムを作成せよ。1〜10までの数字はランダムに出現するものとする

```
// まず0から9を出し
let x = Math.random();// 0 ≦ x < 1
x *= 10;              // 0 ≦ 10x < 10
x = Math.floor(x);    // 0以上の整数 0 ≦ 10x ≦ 9（切り下げて整数になる）
【ここで1を追加する操作を行う】// ←考えてください
document.write(x);
```

7.5　上のプログラムの各ステップに、console.logを入れて、途中経過を確認せよ（console.logについては問題2.3参照）

```
let x = Math.random();// 0 ≦ x < 1
console.log(`x（1番目）= ${x}`);
x *= 10;              // 0 ≦ 10x < 10
console.log(`x（2番目）= ${x}`);
x = Math.floor(x);    // 0以上の整数 0 ≦ 10x ≦ 9（切り下げて整数になる）
console.log(`x（3番目）= ${x}`);
【ここで1を追加する操作を行う】// ←考えてください
console.log(`x（4番目）= ${x}`);
document.write(x);
```

7.6　上のプログラムをデバッガを使いながら実行し、変数xの値の変化を確認し、コンソールに表示される値と比較せよ

■メモ

デバッガでは途中で実行を止めて、変数の途中の値を見たりできます。

VS Codeで途中で止めたい行番号の上にマウスポインタをもっていく（「ホーバー」する）と赤い丸が表示されるので、番号（たとえば12）の左側をクリックします（図7.1）。

すると赤い丸が濃くなりマウスを移動してもそこに赤い丸が表示されたままになります。これで12行目に**ブレークポイント**（一時停止する場所）が置かれたことになります。

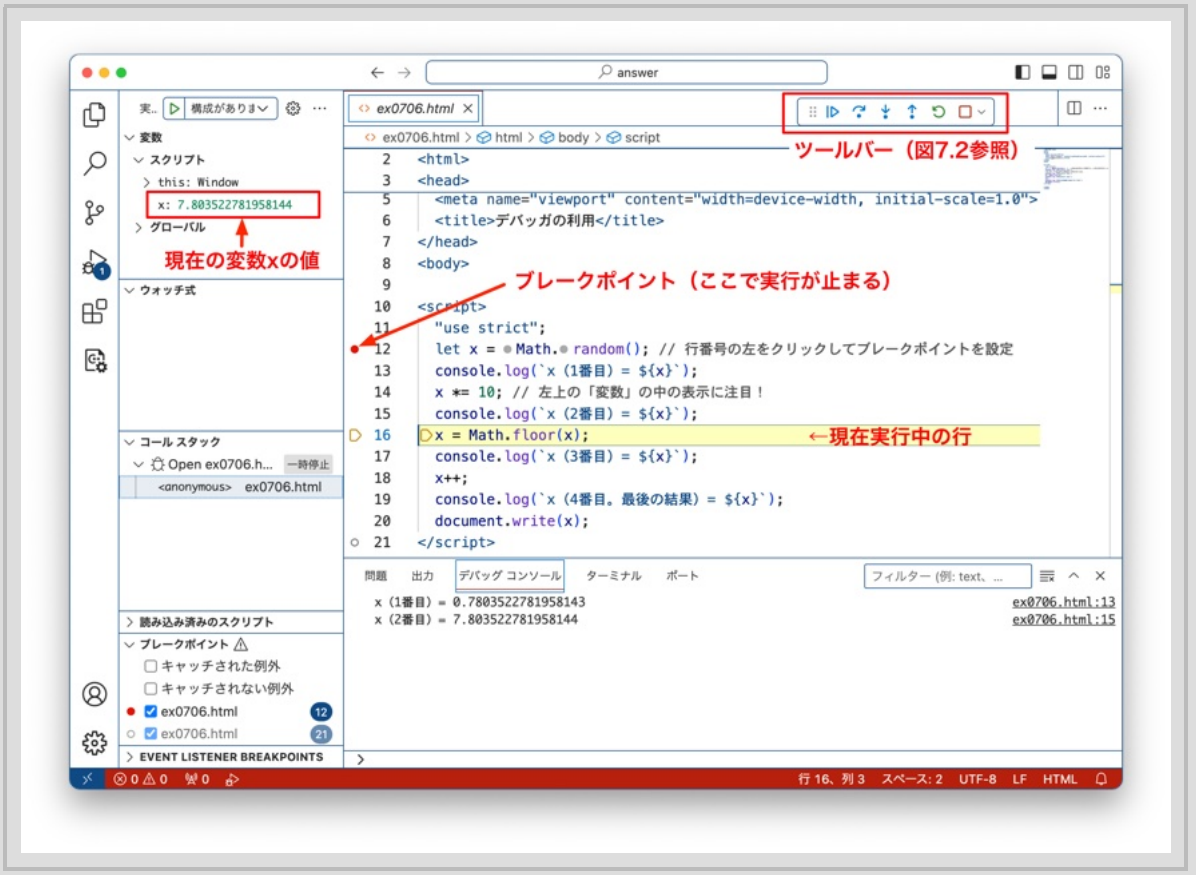


図7.1: デバッガでブレークポイントを設定

右上にあるツールバーのボタンを使って、さまざまな方法で実行することができます。

ここでは、②のステップオーバーのボタンを使って、1ステップずつ実行し、左上に表示される変数xの値の変化を追ってみてください。

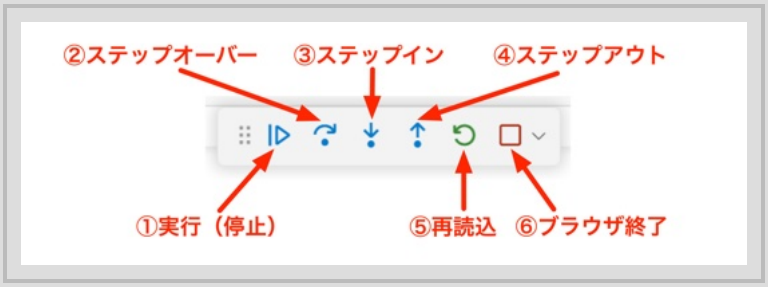


図7.2: ツールバーのボタンの機能

①から④の各ボタンの意味は次とおりです。

①実行（停止）—— ブレークポイントまで実行。実行中は停止ボタンに変わる
②ステップオーバー —— 関数の中にもぐらずに 1 ステップ実行
③ステップイン —— 関数の中にもぐって1ステップ実行
④ステップアウト —— 関数を抜け出して上のレベルに移動

このほかにも、デバッガにはさまざまな機能がありますが、直感的でわかりやすいツールだと思いますので、デバッグの際に使ってみてください。なお、『実践 JavaScript！』（https://musha.com/scjs）の第8章に詳しい説明があります。また、この本の付録Aに生成AIを使ったデバッグの説明がありますので、参考にしてください。

●著者紹介

武舎 広幸 （むしゃ ひろゆき）

国際基督教大学、山梨大学大学院、リソースシェアリング株式会社、オハイオ州立大学大学院、カーネギーメロン大学機械翻訳センター客員研究員等を経て、東京工業大学大学院博士後期課程修了。マーリンアームズ株式会社（https://www.marlin-arms.co.jp/）代表取締役。主に自然言語処理関連ソフトウェアの開発（独立行政法人情報処理推進機構［IPA］の「未踏ソフトウェア創造事業」に2年連続して採択）、コンピュータや自然科学関連の翻訳、プログラミング講座や辞書サイト（https://www.dictjuggler.net/）の運営などを手がける。
著書に『実践 JavaScript！』（オーム社）、『BeOSプログラミング入門』（プレンティスホール出版。共著）、『プログラミングは難しくない！』（チューリング）など、訳書に『初めてのGo言語』『初めてのJavaScript 第3版』『インタフェースデザインの心理学 第2版』（以上オライリー・ジャパン）、『全容解説GPT』『プロフェッショナルPython』『Goプログラミング実践入門』（以上インプレス）、『HTML入門 第2版』『Java言語入門』『Perl入門 第2版』（以上プレンティスホール）、『マッキントッシュ物語』（翔泳社）など多数がある。
個人ウェブサイト：https://www.musha.com/

JavaScript問題集 for Web Designers 第7版

2022年12月15日　初版第1刷発行
2023年 1月15日　第2版第1刷発行
2023年 2月24日　第3版第1刷発行
2023年 4月10日　第4版第1刷発行
2023年12月17日　第5版第1刷発行
2024年 4月18日　第6版第1刷発行
2025年 2月24日　第7版（改題）第1刷発行

著　者　武舎（むしゃ） 広幸（ひろゆき）
発行人　武舎 広幸
発行所　マーリンアームズ株式会社
https://www.marlin-arms.co.jp

ISBN 9798311429795

本書は著作権法上の保護を受けています。本書の一部あるいは全部について（ソフトウェアおよびプログラムを含む）、マーリンアームズ株式会社からの文書による許諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。本書に登場する会社名、製品名は、各社の登録商標または商標です。本文では、®やTMマークは明記しておりません。