「面白い！」と思ったあなたは
プログラミングに向いている！

JavaScriptで学ぶ
プログラミング入門
丸一日コース
問題集

第５版

SAMPLE

武舎広幸 著

はじめに

この本は『JavaScriptで学ぶ プログラミング入門 丸1日コース』（ストアカ等で公開している対面およびオンラインの講座、およびYouTubeで公開している動画）用の問題集として作成しました。この本のサポートページ（https://musha.com/sc/jsbes）には、上記の対面講座に関する詳細、YouTubeの動画や参考資料などが公開されていますのでご覧ください。

詳細はサポートページや動画に譲りますが、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、connpassなどのサイトを通して2,000人以上の方に受講いただいています。

この問題集は上記講座の実習で使われてきた問題を1冊にまとめたものです。非常に単純な問題から、ある程度高度な問題まで、80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。

> kindle版などの電子書籍をご購入いただいた方には、この問題集のPDF版をダウンロードしていただけます（コードなどのコピー・ペーストが可能です）。解答例もウェブ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。

対象読者

この問題集の対象読者は次の方々です。

- **対面あるいはオンラインの講座をお申し込みになった方**
  この問題集で実習をしていただきます
- **上記の動画を視聴中、または視聴なさった方**
  この問題集で実習をしていただくことができます。独力で解ける方もいらっしゃるでしょうが、プログラミングをゼロから単独で身につけるのは大変です。まず自力でやってみて「難しくて無理そう」と感じた場合は、対面あるいはオンラインの講座にご参加なさることをおすすめします（すぐ下の「関連講座」をご覧ください）
- **ほかの言語をご存じでJavaScriptを学びたい方**
  動画をご覧にならずに独力で解いていただくことも可能かと思われます。必要に応じて動画や資料をご覧ください

疑問点などがありましたらまずサポートページ（https://musha.com/sc/jsbes）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いただくか、講座にご参加ください[*1]。

関連講座

対面およびオンラインの講座については次のページをご覧ください。

- 『JavaScriptで学ぶ プログラミング入門 丸1日コース Version 2』（対面およびオンライン） —— https://musha.com/sc/js2
  この問題集を解いていただくための講座です。ご質問等がある方、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

[*1] 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。また、この問題集以外のプログラミング等に関するご質問も、時間がかかりそうなものでな限りお受けいたします（将来的には変更する可能性がありますので、あしからずご了承ください）。

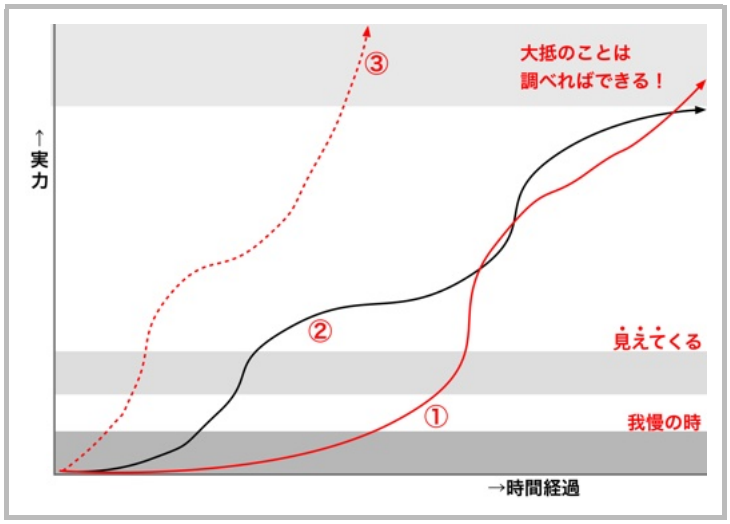
機器と心の準備

この本でプログラミングの学習をするにはパソコンが必要です。

Windows（Win）でもMacintosh（Mac）でも構いません。2017年以降に発売されたパソコンなら問題ないでしょう（これより古くても、メーカーがサポートしているOSが動作するならば、動作は多少遅くなるかもしれませんが多分大丈夫です）。また、Chrome BookやLinuxなどでも可能ですが、こうした環境についての詳しい説明はしませんので、必要に応じてご自分でお調べください

あと、必要なのは皆さんの「忍耐力」です。「コードを直してはブラウザで確認」を何度も何度も繰り返すことになるはずです。慣れてコツがわかってしまえば、さほど苦ではなくなりますが、最初はかなり大変だと思います。

筆者はプログラミングの実力は下の図の赤い線のように向上していく人が多いのではないかと感じています。



パターンは人それぞれです。朱の実線①のように最初はかなり苦労するが、あるところから急に実力がアップする人もいますし、出だしはまずまず順調に進み途中で伸び悩む黒い実線②のような人もいます。最初にくる「我慢の時」を乗り越えられるかどうかが、プログラミングを（ほぼ）不自由なくできるようになるか、楽しめようになるかどうかの分かれ目です。稀（まれ）にですが、朱の破線③のように、ほとんど苦労せずに、見えるようになってしまう方もいらっしゃいます。

「むずかしい」と感じたら、まず、細かいところにあまりこだわらずに、「感じ」を掴（つか）むようにしてみてください。細かい点は本文を読んで練習問題の解答のプログラムを作っていくうちに自然に覚えられます。重要な概念は繰り返し登場するので、強制的に復習させられることになります。ですから、意識的に覚える必要は（あまり）ありません。実習をすることで、徐々に「納得」できるようになります。

ただ、独力で「我慢の時」を乗り越えるのは大変かもしれません。「大変すぎる」と思ったら、どなたか相談できる方を見つけるとよいでしょう。そういう人が周囲に見つからないのなら講座に参加ください。

謝辞

この問題集は、ストアカ、Doorkeeper、connpassで公開した講座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心より感謝いたします。

一部のイラストで「ねこ画伯コハクちゃん」（https://kohacu.com）の作品を使わせていただきました。ありがとうございます。

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指定のURLからダウンロード（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」の下に`student`フォルダを展開しておいてください[*1]
  - macOS（Macintosh）では、`student`フォルダを「デスクトップ」に移動しておいてください[*1]
- 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）

[*1] 「デスクトップ」以外の場所においても構いませんが、「デスクトップ」にあるものとして説明します。それ以外のところに置いた場合は、読み替えてください。

この本では、ブラウザのGoogle Chrome（略してChrome）とエディタのVisual Studio Code（略して「VS Code」）を使ってプログラミングを学んでいきます。ほかのブラウザやエディタなどを使ってもプログラミングはできますが、現在非常に多くのプログラマーが開発に利用していますので、この本でもこの2つを利用しましょう。

> 普段ほかのエディタを使っている人も、ひとまずVS Codeをインストールして使ってみてください。なかなか使いやすいエディタですので。
>
> しばらく使ってみて、「やっぱり『◯◯エディタ』のほうがいい」と思ったら、戻っても結構です。ただし、この本ではVS Codeを仮定して説明していきます。他のエディタを使ってファイルの編集が終わり、ブラウザで実行する際には、［ファイル］メニューから読み込むか、編集したファイルのアイコンをブラウザの「ドキュメント領域」にドロップして開いてください。

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）。

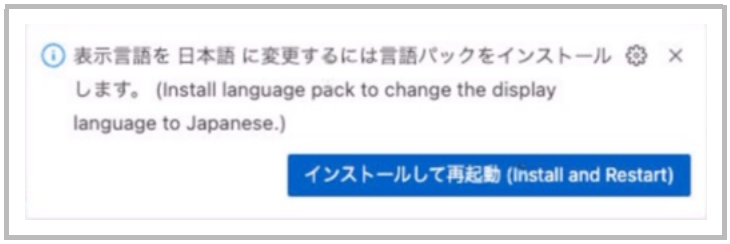


図1: macOSで表示される日本語の言語パックのインストールを促すメッセージ

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

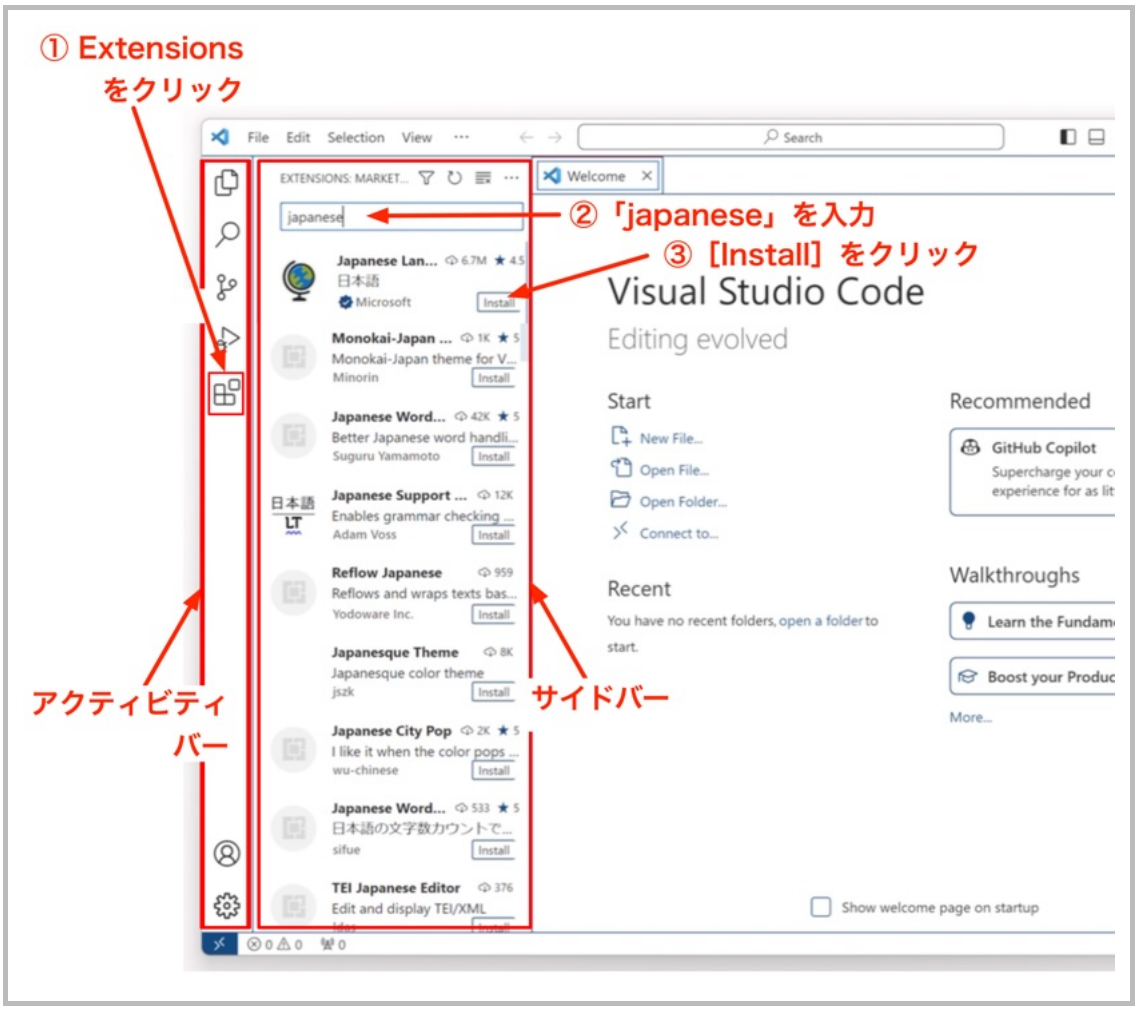


図2: Japanese Langauge Packのインストール

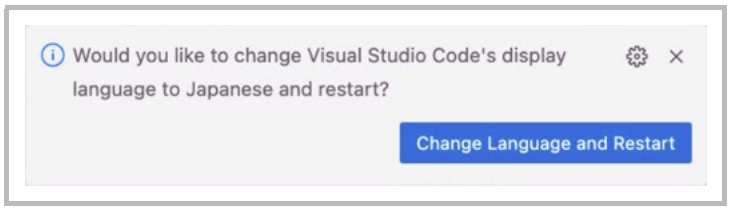


図3: ［Change Language and Restart］をクリックして再起動するとメニューなどが日本語に変わる

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

これで準備が完了しました。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。

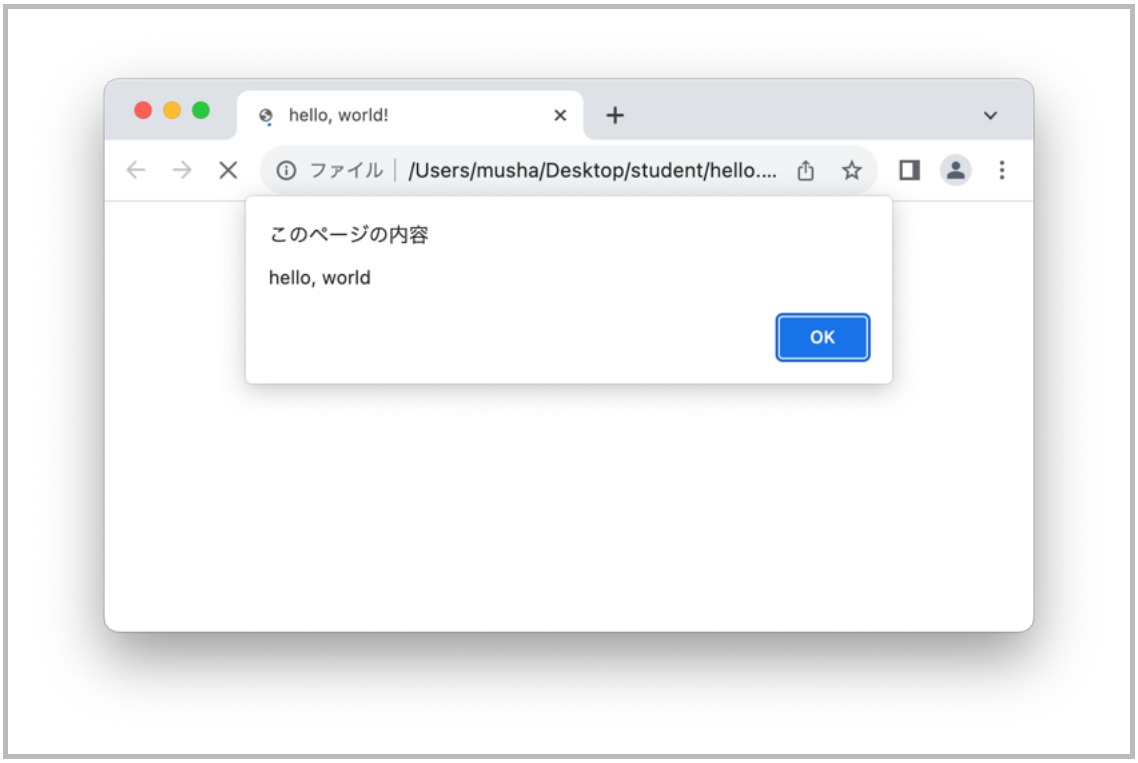


図4: ダイアログボックスの表示

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が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[*2]］）
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

[*2] 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。

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。

> 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。

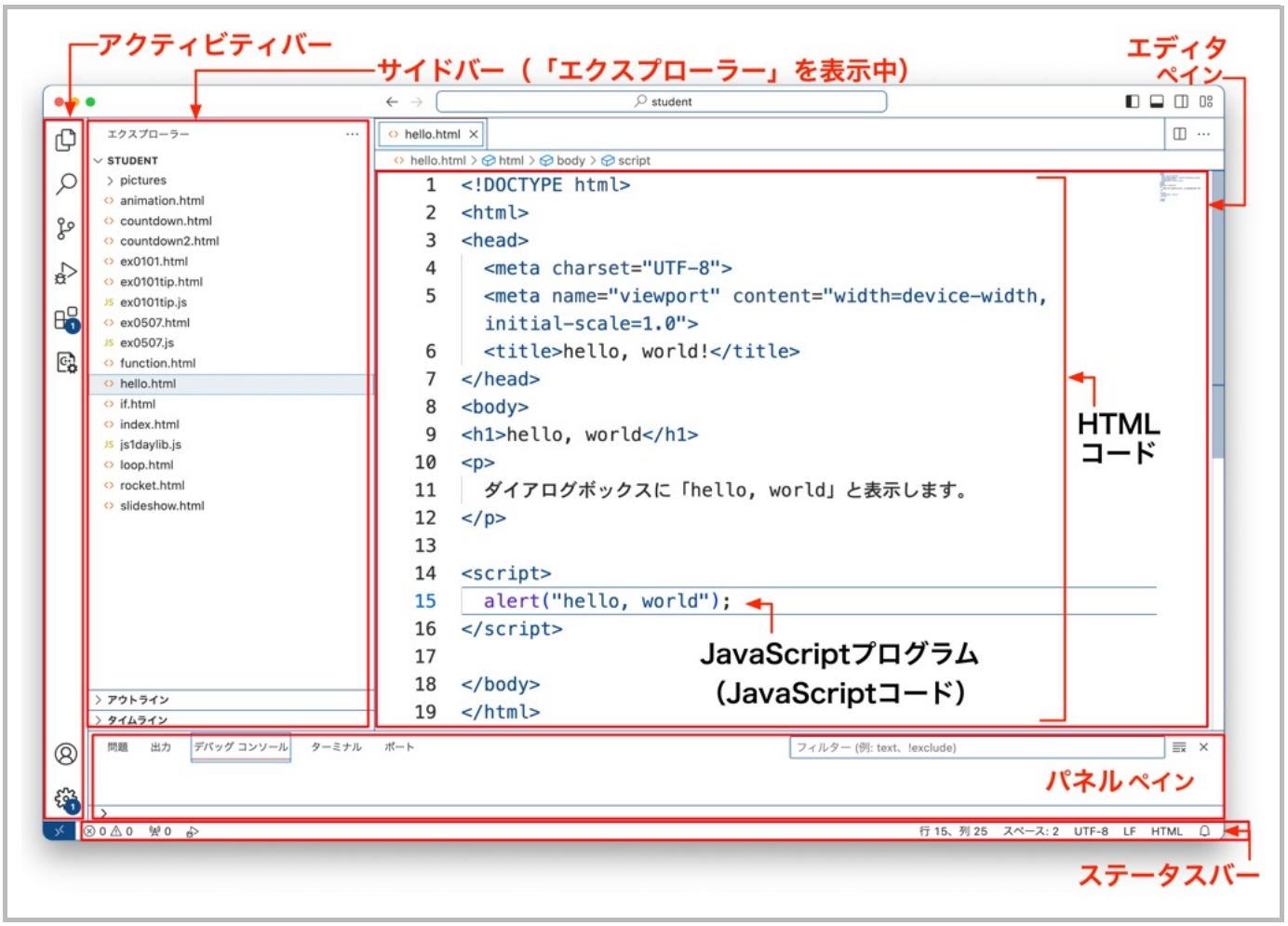


図5: HTMLファイルをVS Codeで開く

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）。

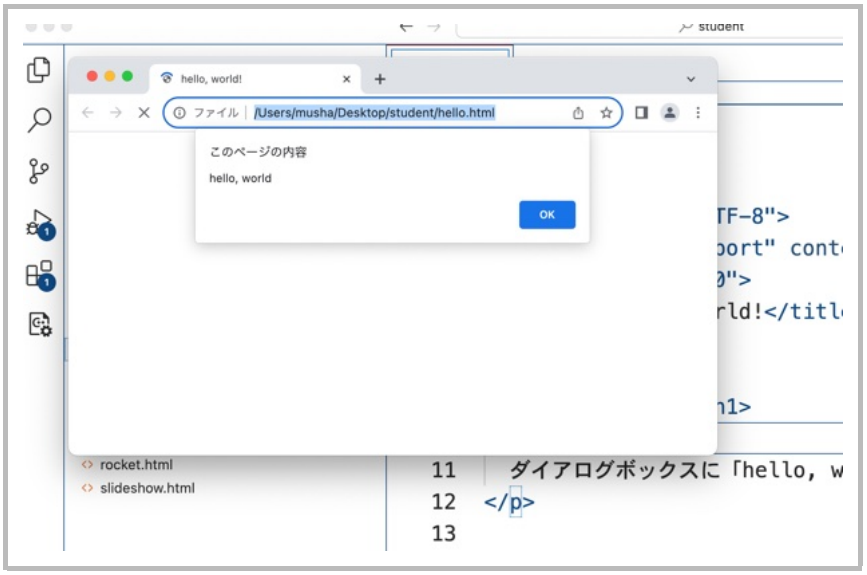


図7: HTMLファイルをChromeで表示

［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5章で説明します）。

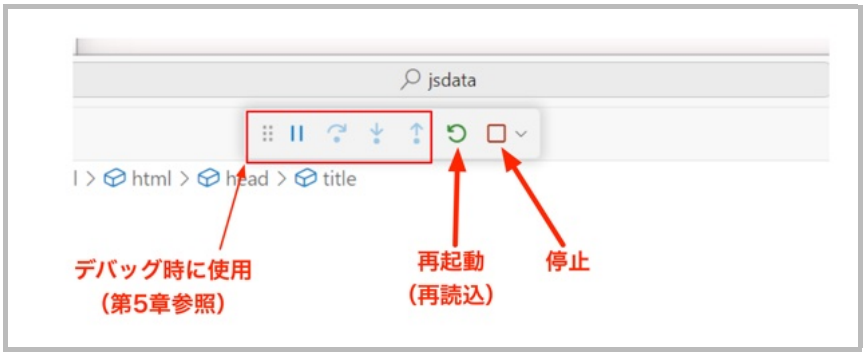


図8: VS Code側からChromeを終了することもできる

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。

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 —— `<p>…</p>`。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。

> 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。

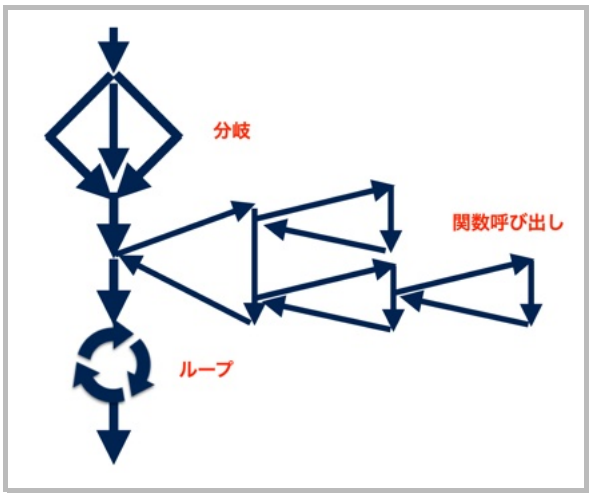
1. フロー制御 —— 計算の手順
2. データ構造 —— 情報の記憶手法
3. 演算子 —— 加減乗除や文字の連結、大小、真偽の判断などの操作

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。次の3つが基本です。この問題集の前半（第6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。

- 関数呼び出し（第1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。関数（の呼び出し）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5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6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**（第2章）あるいは**定数**（第4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。

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2章） —— +、−、*、/、%
- 文字列の連結（第3章） —— +
- 比較（第6章、第8章） —— <、>、<=、>=、===、!==
- 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

■メモ

対面あるいはオンラインで、1回目の講座に出席中の方は、全部解いていると時間が足りなので、次の順番で問題を解くことをおすすめします（他の問題は後で解いてみてください）。スキップする問題には「★講座ではスキップ★」と書いてあります。

- PART Ⅰ（午前）
  - 第1章（関数の呼び出し）
  - 第2章の2.1と2.5（console.logの使い方）
    （第2章の残りは計算問題なので後でやってください）
  - 第3章（文字列）
  - 第4章（テンプレートリテラル）
  - 第5章の5.1〜5.4まで（ループ）
  - 第6章（分岐）
  - （時間があれば）第5章の残り（ループの少し複雑な問題）
- PART Ⅱ（午後）
  - 第7章（タイマー）
  - 第8章（アニメーション）

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">
  <title>1-1 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
  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特に）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">
  <title>1-1 ダイアログボックスの表示（別ファイル）</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="ex0101-tip.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（ほとんどの場合）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8章の最後のコラムで説明します。

1.4　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(`<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
```

- 画像があることを確認
- うまくいかない場合はHTML（JavaScript）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）

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

```
document.write(`<img src='pictures/picture000.jpg'>`);
document.write(`<img src='pictures/picture001.jpg'>`);
document.write(`<img src='pictures/picture002.jpg'>`);
```

第2章　演算子（加減乗除）

2.1　次の計算の結果を予想し、結果を出力して比較せよ（あとで復習できるように、前の章の例題とは別のファイル名で保存することを推奨。たとえばex0201.html）

```
<!DOCTYPE html>
... 【途中省略。前のファイルを利用して必要な部分を書き換える】
<script>
"use strict";
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
             //  「=」の意味は下記参照。「;」は「文」の区切り。行の最後ならば省略できる
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 -= 21; //  「x = x-21;」と同じ。 xから21を引いて、それをx記憶する
x++; //  「x++」は　「x = x+1」と同じで、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
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

「x = 320」の「=」は数学で使う「イコール」とは意味が違います。どちらかというと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320という値を記憶するという、記憶域内の状態の変化を引き起こします。

```
「x ← 320」
```

2.2　次の計算の結果を予想し、結果を出力して比較せよ（★講座ではスキップ。問題2.5に進んでください★）

```
let x = -3;
x -= 20; //  「x = x-20;」自分（x）から20を引いて、それを自分（x）に記憶する
x += 3;
x--; //「x--」は　「x = x-1」と同じで、xから1を引いて、それををxに記憶する
     // 1を引く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
document.write(x);
```

バリエーション（★講座ではスキップ★）

- +, -, *, /, %（余り）を使っていろいろな計算をしてみよ

2.3　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】（★講座ではスキップ★）

```
// 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);
document.write(x);
```

2.4　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（★講座ではスキップ★）

```
// 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);
```

2.5　上の2-1の各文の後ろにconsole.log()やalert()を入れて、途中経過を表示しながら、xの値の変化を追ってみよ

```
let x = 300;
x += 33;
x -= 21;
x++;
```

↓

```
let x = 300;
console.log(x);
x += 33;
console.log(x);
x -= 21;
console.log(x);
x++;
console.log(x);
```

- 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

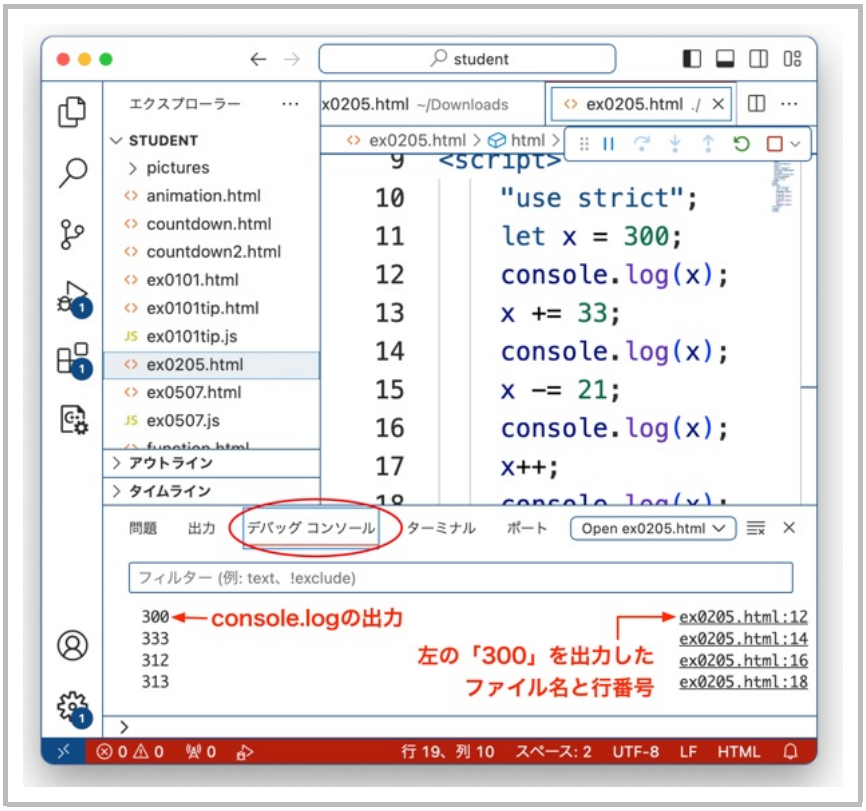


図2.1: console.logの出力

- Chromeでも「コンソール（console）」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］

第3章　文字列の連結

3.1　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

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

文字列の連結にも「+」を使う。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。

3.2　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、第4章でやるバッククオート（`...`）も使える
document.write(x);
```

第4章　テンプレートリテラル

4.1　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「可変部分付き文字列」である。

4.2　次の計算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

```
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}<br>`);  // `...` 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。

4.3　次のコードを実行して画像が（300ピクセル［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);
```

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）

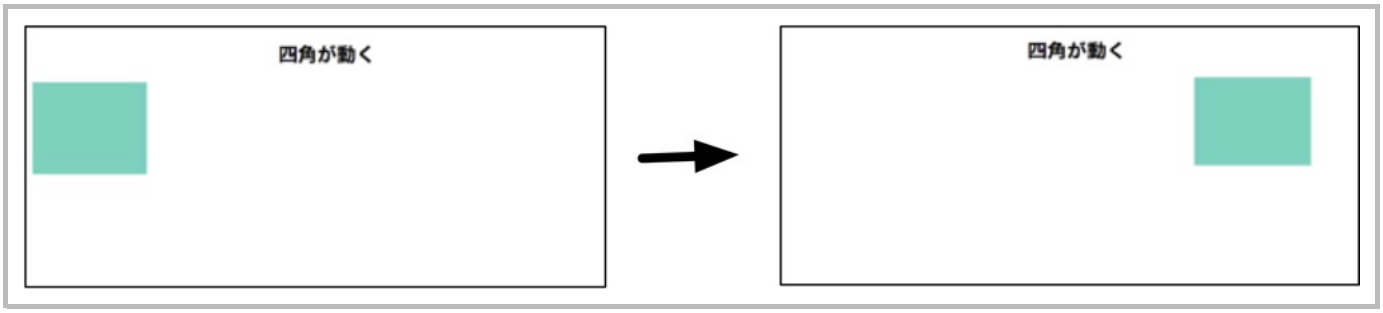
■メモ

変数、定数、関数などの名前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。

ネットで全世界に公開するような場合は英語で書かなければなりませんし、「日本語の変数名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）。

第8章　アニメーション

8.1　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>
```

8.2　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

8.3　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

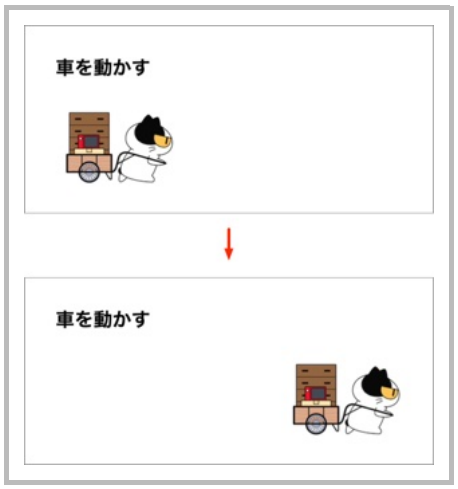
コード例

```
"use strict";
const 倍率 = 5;
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
    }
  }
}
```

8.4　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、 ... 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>
```

8.5　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）

8.6　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

8.7　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
});
```

8.8　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★）

```
/* 以下のコードは参考。一部分のみ */
"use strict";
// 色を変える
let 四角 = document.getElementById("rectangle");
四角.style.backgroundColor = "#BA03EE";

let 色指定 = "#";
for (let i=1; i<=3; i++) {
  const j = ランダムな整数を生成(0, 15); // studentフォルダのjs1daylib.jsの中にある
  色指定 += j.toString(16); //文字列を16進数にして、「色指定に追加」
```

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。

●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、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。訳書に『Symantec C++トレーニングブック』（翔泳社）、『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）、『Goプログラミング実践入門』『Python基礎&実践プログラミング』『全容解説GPT』（以上インプレス）『初めてのJavaScript 第3版』『初めてのGo言語』『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）など多数がある。https://www.musha.com/にウェブページ。

JavaScriptで学ぶプログラミング入門 丸一日コース 問題集 第5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

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 9798863971599