

JavaScriptで創る 豊かなウェブサイト

アトリエ未来 [著]



第三版

写真や動画、地図に

お問い合わせ、PWA、モーダルウィンドウまで
豊かなウェブサイトを創りたい方にお薦めです

JavaScript で創る 豊かなウェブサイト

[著] アトリエ未来

技術書典 13 新刊
令和四年九月廿三日 ver 3.0

■免責

本書は情報の提供のみを目的としています。

本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。

また本書では、TM、[®]、[©]などのマークは省略しています。

始めに

楽しいプログラミングの世界へようこそ。

情報技術に囲まれた生活を送るわたくしたち。多くの先人が築いた歴史の上に今日があります。

ウェブサイトを作つてみたい方へ向けて、いろいろ記してみました。

ファビコン / アップルタッチアイコン / 検索エンジン用の説明文 / リンクの無効化 / 写真のポップアップやスライド / ヒーローローイメージ / 光りながら出現する文字 / 装飾的なボタン / 見出しの装飾 / トップに戻るボタン / ナビゲーションメニュー / タブ機能 / 続きを読むボタン / 表を互い違いに色を塗る / 電話番号にリンクを設定 / ルビをふる / 地図を載せる / 紹介動画を掲載する / モーダルウィンドウ / PWA(Progressive Web Apps) などなど

巻末には、参考書籍、参考リンク集、HTML / CSS / JavaScript の簡易まとめを付けました。

現代の魔法、それがプログラミングです。自由自在にコンピュータを操つて、幸せな未来へと大きく羽ばたいていってください。

♣ 対象読者

簡単なウェブサイトの作成ができる、初心者の方を想定しています。豊かなウェブサイトを作つてみたい方にご参考になれば幸いです。 *1

♣ プログラムのダウンロード

この本で作成するウェブサイト *2 は、巻末の参考書籍「CSS グリッドで作る HTML5 & CSS3 レッスンブック」などを元にしています。ソースコード *3 も後悔しておりますので、ご活用いただければ幸いです。

♣ 謝辞

Re:VIEW Starter^aを用いて、快適に執筆することができました。作者の kauplan さんに厚く御礼申し上げます。

^a <https://kauplan.org/reviewstarter/>



*1 コンピュータとして Mac をお使いの方を対象に執筆しておりますので、Linux や Windows をお使いの方は、キーボード操作等、一部異なる箇所がございます。適宜読み替えていただければ幸いです。

*2 <https://wave-improve.netlify.app>

*3 <https://github.com/Atelier-Mirai/wave-improve>

表紙絵の女の子は、千葉県松戸市在住のフリーランス SD イラストレーター 早瀬ひろむ^aさんの作品です。素敵なイラストを描いてくださいり、ありがとうございます。

また、背景の桜と青空の絵は、かるら^bさんの作品です。桜咲く青空で心も明るくなります。感謝です。

^a <https://hiromu-hayase.tumblr.com>

^b <http://karura.org>



早瀬ひろむ さん

♣ 著者紹介



卓越した技能を有する者として認められる国家資格「応用情報技術者」を保持。平成30年より「アトリエ未来^a」を創業。HTML講座やRuby講座などプログラミングの個人指導や、ITパスポート講座等の資格講座の開催、ウェブサイト作成等を受注している。

趣味の将棋は、日本将棋連盟より三段の免状を允許。日本の美しい自然や豊かな精神性を宿す熊野古道を歩くことや、花に囲まれた日々を愛している。

^a <https://atelier-mirai.net/>

【コラム】金の延棒クイズ

二進数の不思議を感じる、ちょっと豊かな気分に成れるクイズです。

(正解はこの本のどこかにあるよ。最後まで読んでね。)



七日の給料の支払いとして金の延べ棒が一本あります。これを使って仕事をしてくれる方への日当を支払いたいと思います。

六回鉄を入れ七等分すれば日払いできますが、金の延べ棒を六回も切り取るのは大変です。

必要最小限の二回切り取ることをしたいですが、どことどこを切れば良いでしょうか？

目次

始めに	i
第1章 機能拡張のご紹介	1
1.1 ウェブサイト作成の為の三つの言語	1
1.2 トップページの機能拡張例	3
1.3 記事についての機能拡張（1）	6
1.4 記事についての機能拡張（2）	7
1.5 サイトについての機能拡張例	8
1.6 お問い合わせページの機能拡張例	9
1.7 その他いろいろ詰め合わせ	10
第2章 トップページの機能拡張	13
2.1 ファビコンとアップルタッチアイコン	13
2.2 検索エンジン用の説明文	14
2.3 リンクの無効化	14
2.4 写真をポップアップウィンドウで表示する	15
2.5 写真をスライドさせる	27
2.6 全画面のヒーローイメージ	30
2.7 スクロールを促す為の表示	32
[コラム]position: absolute と position: relativeについて	33
2.8 スクロールすると出現するメニュー	35
2.9 光りながら出現する文字	40
2.10 見出しの装飾	43
2.11 装飾的なボタンを作る	44
2.12 トップに戻るボタン	46
第3章 様々なフッターの装飾	49
3.1 文字のグラデーション	50
3.2 可変の文字サイズ	50
3.3 グラデーションで背景画像を彩る	53
3.4 菱形のナビゲーションメニュー	54
3.5 スムーズスクロール	56
第4章 記事ページの機能拡張	57
4.1 写真をセピア調にする / 枠を付ける	57
4.2 タブ機能	59
4.3 続きを読む機能	61
第5章 サイトについてページの機能拡張	63
5.1 表を互い違いに色を塗る	63

5.2	見出しの均等割付	63
5.3	電話番号、メールアドレスにリンクを設定する	64
5.4	ルビを振る	65
5.5	地図を掲載する	66
5.6	紹介動画を掲載する	71
第 6 章 お問い合わせページの機能拡張		73
6.1	ネットリファイ Netlify	73
6.2	お問い合わせフォームの機能拡張	73
第 7 章 モーダルウィンドウ		85
7.1	モーダルウィンドウ	85
7.2	基本準備	86
7.3	インライン	87
7.4	フルスクリーン	89
7.5	画像	89
7.6	画像ギャラリー（複数枚）	91
7.7	動画	92
7.8	iframe で他のウェブサイトの内容を開く	95
7.9	確認画面	96
第 8 章 文字の演出効果		99
8.1	シャッフルテキスト	99
第 9 章 桜吹雪		107
9.1	HTML	107
9.2	CSS	108
9.3	JavaScript	110
9.4	春夏秋冬	116
第 10 章 スクロールしたら要素を表示させる		117
10.1	HTML	117
10.2	CSS	118
10.3	JavaScript	119
第 11 章 固定フッター		121
11.1	HTML	121
11.2	CSS	123
11.3	JavaScript	125
第 12 章 追隨メニュー		127
12.1	HTML	127
12.2	CSS	129
12.3	縦書きにする	130

第 13 章 モバイルメニュー	133
13.1 HTML	133
13.2 CSS	135
13.3 JavaScript	137
第 14 章 画像の絞り込み	139
14.1 Muuri	139
14.2 HTML	141
14.3 CSS	144
14.4 JavaScript	144
第 15 章 グラフを描画する	147
15.1 Chart.js	147
15.2 HTML	149
15.3 JavaScript	151
15.4 ★による評価	152
第 16 章 プログレッシブウェブアプリ (PWA)	155
16.1 プログレッシブウェブアプリ とは	155
16.2 PWA の利点	156
16.3 実装方法	158
付録 A 珠玉の名著のご紹介	161
A.1 HTML / CSS の学習に	161
A.2 JavaScript の学習に	162
付録 B 参考リンク集	163
B.1 タイピング練習	163
B.2 プログラミング学習に	163
B.3 写真・イラスト素材	164
B.4 洗練されたサイト集	164
付録 C HTML / CSS / JavaScript 簡易まとめ	165
C.1 HTML 簡易まとめ	165
C.2 CSS 簡易まとめ	169
C.3 JavaScript 簡易まとめ	173
終わりに	177

第 1 章

機能拡張のご紹介

巻末の参考書籍「CSS グリッドで作る HTML&CSS レッスンブック」をもとに学習すると、CSS グリッドを使って、素敵なウェブサイトが完成いたします。

これをもとに、サイトの動きを司る JavaScript の使用など、様々な機能を追加してより魅力的に仕上げていきます。

1.1 ウェブサイト作成の為の三つの言語

ウェブサイトを作成する為に用いられる言語には、大きく次の三つがあります。

- **HTML(HyperText Markup Language)**

HyperText Markup Language を日本語で表すなら、「ハイパーテキストに目印をつける言語」くらいの意味になります。

目印をつける (Markup) というのは、文書の各部分（見出し・段落・表・リストなど）が果たしている役割が分かるようにすることです。そのために専用の「タグ」を使います。

コンピュータに理解できるよう文書構造を定義することが、HTML の最も重要な役割です。

- **CSS(Cascading Style Sheet)**

ウェブサイトの見た目を飾り付け、彩る為の言語です。文字の大きさや色の指定から画像の配置先など、様々な装飾を行うことができます。

HTML のタグが親子関係（包含関係）にある場合、多くの設定値は親要素に指定されたものが子要素、孫要素に引き継がれていきます。このように設定値が上から下へ伝播していく様子を、階段状の瀧を意味する cascade になぞらえて、命名されました。

- **JavaScript**

ウェブサイトに、「双方向性・相互作用性（インタラクティブ）」を齎すために用いられるプログラミング言語です。

例えば、閲覧者の操作に反応して表示が書き換わったり、ページが表示される際に写真などの要素に動きや効果を加えたり、サーバと通信してデータを取得したりするなど、現在のウェブサイトには欠かせないプログラミング言語となっています。

それぞれの詳細を知悉^{ちしつ}することは大変ですが、全てを知らずとも良く使う基本的なコードのみでもウェブサイトの作成は充分可能ですので、学んでいきましょう。

♣ 学習の為の参考サイトや書籍のご紹介

HTML / CSS を学ぶ為に

ウェブサイト作成のために、様々な参考書籍があり、またネット上でも有益なサイトが多数あります
が、始めて学ぶ方向けには、(準) 公式サイトである、MDNより提供されている

- ウェブ入門^{*1}
- HTML の基本^{*2}
- CSS の基本^{*3}

が役に立つことでしょう。

また、どのような仕組みでウェブサイトが閲覧できるのか、少し技術的な背景についても知見がある
と、(専門家を目指す方はもちろんですが) 知的好奇心を満たす点からも楽しいものです。

- ウェブのしくみ^{*4}

MDNでの学習を終えた方には、巻末に挙げた参考書籍の中から次の二冊をお勧めします。

- CSS グリッドで作る HTML5&CSS3 レッスンブック
- 作って学ぶ HTML & CSS モダンコーディング

前書は、初心者向けに基礎的なウェブサイトの作成を簡単な技術的な背景も含めて解説されている良書です。本書の執筆に関しても、多くの部分を負っており、貢献に感謝いたします。後書は、前書の学習を終え、基礎的な HTML/CSS が書けるようになった方が、より進んだサイト作成の技術を学ぶために最適な一冊となっています。

JavaScript を学ぶ為に

JavaScript の学習に当たっては、以下のMDNの説明が概要を掴むには良いでしょう。

- JavaScript の基本^{*5}

簡潔に説明されてはいますが、始めてプログラミングに触れる方には少し難しいと感じるかもしれません。そういう方へは、巻末の参考書籍

- スラスラ読める JavaScript ふりがなプログラミング

がお勧めです。一語一語、漢文に倣った読み下し文でコードの意味が書かれており、短いコードの一文一文を確かめながら実行することで、理解を深めていくことができるようになっています。

JavaScript の文法を理解し、簡単なコードを書けるようになったところで、実際のウェブサイトに組み込んで見ましょう。

- 動く Web デザインアイディア帳 / 動く Web デザインアイディア帳 実践編

*1 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web

*2 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web/HTML_basics

*3 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web/CSS_basics

*4 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web/How_the_Web_works

*5 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web/JavaScript_basics

では、様々な動きを^{もたら}すための、CSS アニメーション や、JavaScript で書かれた各種ライブラリの使い方等が紹介されています。少しコードが汚いのが残念ではあります、二冊合計で 900 ページにも上る豊富な事例が掲載されており、実際のウェブサイト作成に役立つ一冊となっています。

- JavaScript Primer 迷わぬための入門書

は、JavaScript をより深く知りたい方にお勧めの一冊。難易度高目ではありますが、前半だけでも読み通すと、全容を知ることができ、他のプログラミング言語の学習にも生きてくることでしょう。

HTML / CSS / JavaScript に関して、本書でも要所要所で解説を加えていきたいと思いますが、紙幅の限りもございますので、上記の参考書籍やサイト等で、適宜学習いただければ幸いです。

1.2 トップページの機能拡張例

「CSS グリッドで作る HTML&CSS レッスンブック」に基づく [作成例^{*6}](#) に次のような機能拡張を施していきます。

- ファビコン
- アップルタッチアイコン
- 検索エンジン用の説明文
- リンクの無効化
- 写真のポップアップ
- 写真をスライドさせる
- 全画面ヒーローイメージ
- スクロールを促す為の表示
- スクロールすると出現するメニュー
- 光りながら出現する文字
- 見出しの装飾
- 装飾的なボタン
- トップに戻るボタン
- 様々なフッターの装飾

^{*6} <https://wave-example.netlify.app>

WAVE

TOP サイトについて お問い合わせ



最新記事



スケッチが楽しくなるノート



緑のアクセントならこれ



小物と飾り棚の組み合わせ



居心地のいい部屋にあるもの



カラフルなタイル

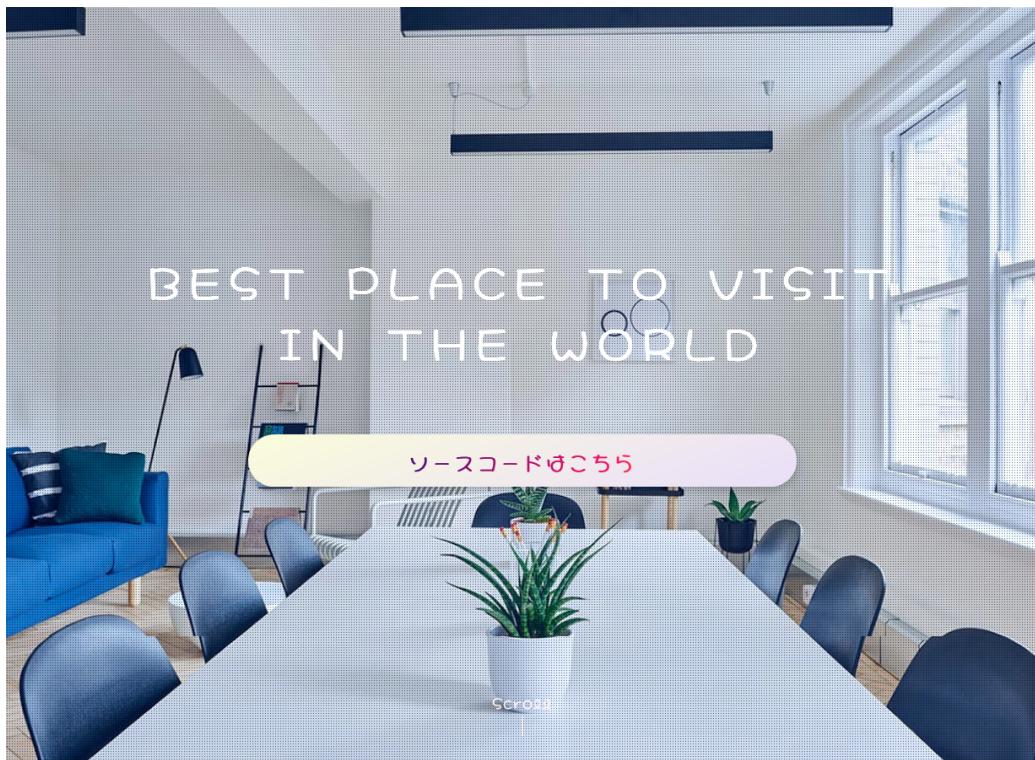


美味しいオムレツのレシピ

© WAVE

機能拡張後の例^{*7}です。

*7 <https://wave-improve.netlify.app>



最新記事

▼スクッチが楽しくなるノート▲

▼緑のアクセントさらこれ▲

▼小物と飾り棚の組み合わせ▲

▼居心地のいい部屋にあると△

▼カラフルなタイル▲

▼美味しいオムレツのレシピ▲

▼青空と△

▼モーダルウィンドウ▲

手軽にテキストシャッフル演出ができるJavaScriptライブラリ'shuffle-text'を公開

▼シャッフルテキスト▲

スクロールアニメーション

文學堂
文學紹介

文學堂は日本で最も長い歴史を持つ文庫による書評を掲載しています。

ボーラーの広場 実践実治

多くの人に楽しんで、リーチするための実践的な知識を提供します。

ボーラーの広場 実践実治

多くの人に楽しんで、リーチするための実践的な知識を提供します。

固定フッター△

追跡メニュー△

モバイルメニュー△

画像の並び替え△

さざざぎグラフ表示△

MUURI

Chart.js

動物園の思い出

動物園に行ってきました。(いろいろな動物に会いました。)



海の思い出

海に行ってきました。波の音はとっても心落ち着きます。



Copyright © 令和四年 WAVE All rights reserved.

1.3 記事についての機能拡張（1）

- 写真をセピア調にする
 - 枠を付ける

1.4 記事についての機能拡張（2）

- タブ機能
- 続きを読むボタン



The screenshot shows a blog post titled "緑のアクセントならこれ" (Green Accents). The post features a large image of a cactus in an orange pot. Below the image is a text block about the cactus, mentioning it's from the Cactaceae family and its popularity as a houseplant. A blue button labeled "続きを読む" (Read More) is visible at the bottom of the text block.

PROFILE



部屋の掃除は好きだけれど、お皿洗いは苦手。毎日そこら辺をお散歩しています。

PICKUP



▼スケッチが楽しくなるノート▲



▼緑のアクセントならこれ▲



▼小物と飾り棚の組み合わせ▲



▼居心地のいい部屋にあるもの▲



▼カラフルなタイル▲

1.5 サイトについての機能拡張例

- 表を互い違いに色を塗る
- 見出しの均等割付
- 電話番号、メールアドレスにリンクを設定する
- ルビをふる
- 地図を載せる
- 紹介動画を掲載する

 [TOP](#) [ABOUT](#) [CONTACT](#)

サイトについて

綺麗なインテリアや可愛い文房具など、日常の中で見つけたお気に入りのものを 繰り
ているサイトです。

サイト名	WAVE
開設日	四月一日
住所	名古屋市桜区桜町一丁目一番地 桜ビル一階
電話番号	名古屋 勝さん 052-373-4649
メールアドレス	contact@example.com
運営方針	とさとさよたり更新中。最新記事の一覧はトップページに掲載しています。



会社案内
地下鉄桜線 桜駅1番出口より徒歩三分、交通至便の地にあります。

PROFILE


部屋の掃除は好きだけれど、お皿洗いは苦手。毎日そこら辺をお散歩していきます。

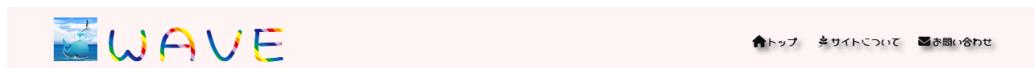
PICKUP


 ▼スケッチが楽しくなるノート▲
 ▼緑のアクセントなうごれ▲
 ▼小物と飾り棚の組み合わせ▲
 ▼居心地のいい部屋にあるもの▲
 ▼カラフルなタイル▲


一日社長 ノンちゃん
 「一日社長 ノンちゃん」で
 す。のんびり寝ていています。

1.6 お問い合わせページの機能拡張例

- 記入例(プレースホルダ)の表示
- スパム対策
- 必須項目設定
- 連絡先自動入力
- ページ離脱時のアラート表示



お問い合わせ

ご意見、ご感想などがございましたら、以下の欄にご記入の上、送信してください。記事に関するご質問などもお気軽にご寄せてください。

お問い合わせ内容 (必須)

お名前 (必須)

メールアドレス (必須)

電話番号

送信

PROFILE



部屋の掃除は好きだけれど、お皿洗いは苦手。毎日そこら辺をお散歩している。

PICKUP



スクラップが楽しく
あるノート▲



緑のアクセントな
うこれ▲



小物と飾り棚の組
み合わせ▲



居心地のいい部屋
にあるもの▲



カラフルなタイル
△

- 問い合わせ完了時の表示



お問い合わせを承りました

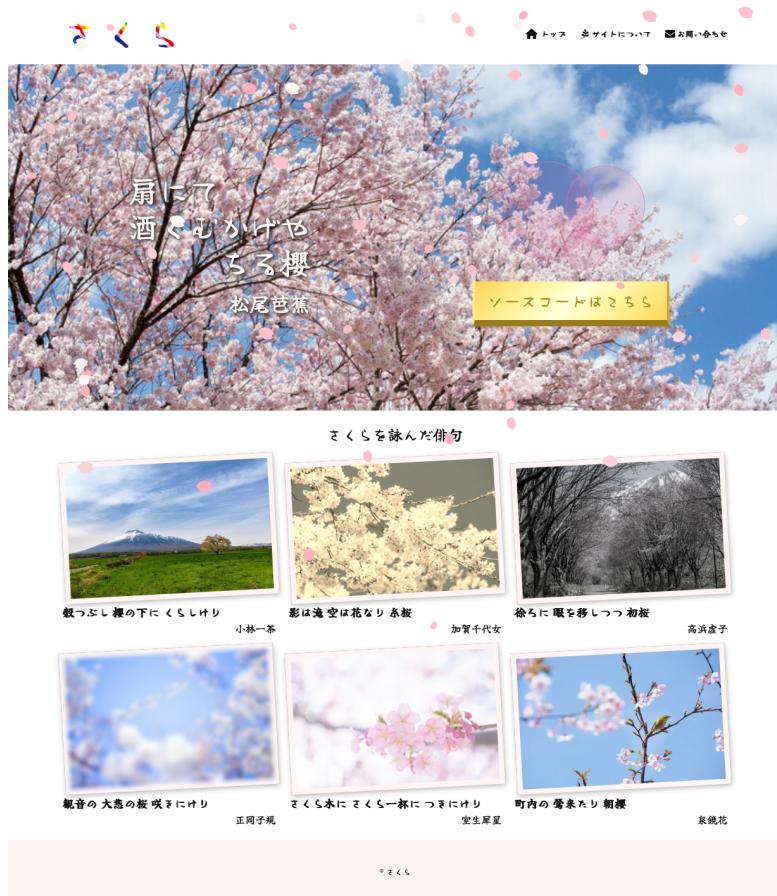
担当者よりご連絡差し上げますので、しばらくお待ちください。

PROFILE



1.7 その他いろいろ詰め合わせ

♣ 桜吹雪を散らせる



♣ 固定フッターと追随メニュー

文學堂

文學紹介

文學堂では古今の様々な作家による名作を紹介しています。

ポーラーの広場 宮沢賢治

そのころわたくしは、モリオ市の博物局に勤めて居りました。

十八等官でしたから役所のなかでも、ずうっと下の方でしたし俸給もほんのわずかでしたが、受持ちが機本の収集や整理で生れ付き好きなことでしたから、くっちは毎日ずいぶん愉快にはたらきました。殊にそのころ、モリオ市一高浜虚子 構物園に堀え直すというので、その景色のいいまわりにアカシアを植えたり、小地図が、切符売場や信号所の建物のついたまま、わたくしどもの役所の方へまわって

▼固定フッター▲

文學堂

文學紹介

文學堂では古今の様々な作家による名作を紹介しています。

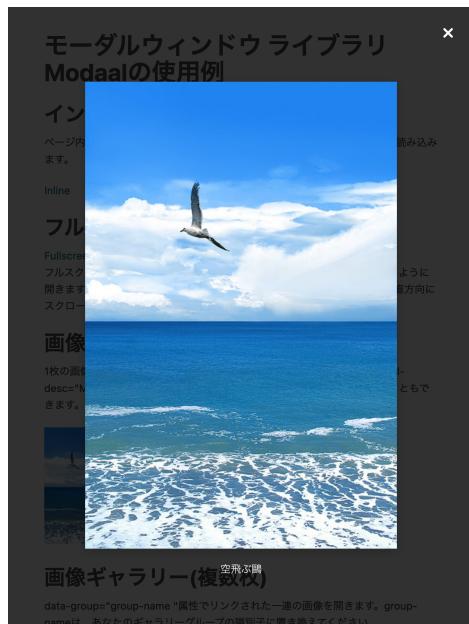
ポーラーの広場 宮沢賢治

そのころわたくしは、モリオ市の博物局に勤めて居ました。

十八等官でしたから役所のなかでも、ずうっと下の方でしたし俸給もほんのわずかでしたが、受持ちが機本の収集や整理で生れ付き好きなことでしたから、くっちは毎日ずいぶん愉快にはたらきました。殊にそのころ、モリオ市一高浜虚子 構物園に堀え直すというので、その景色のいいまわりにアカシアを植えたり、小地図が、切符売場や信号所の建物のついたまま、わたくしどもの役所の方へまわって

▼追随メニュー▲

♣ モーダルウィンドウ



♣ モバイルメニュー

左側サイト (白背景)	右側サイト (黒背景)
文學紹介	文學紹介
ポラーノの広場 宮沢賢治	ポラーノの広場 宮沢賢治
坊ちゃん 夏目漱石	坊ちゃん 夏目漱石
杜子春 芥川龍之介	杜子春 芥川龍之介
高瀬舟 森鷗外	高瀬舟 森鷗外

♣ 種類ごとに絞り込む

ラーメン人気店紹介

全国各地の美味しいラーメン店を紹介します。

気になるラーメン名を入力 全てを表示

[京都市] 櫻幕ラーメン 櫻幕ラーメンは京都市東区にある醤油ラーメンが人気のお店。280円とお財布に優しいお店である。	[京都市] 柳風ラーメン 柳風ラーメンは京都市北区にある味噌ラーメンが人気のお店。280円とお財布に優しいお店である。	[東京都] 鳳凰ラーメン 鳳凰ラーメンは東京都西区にある塩ラーメンが人気のお店。380円とお財布に優しいお店である。
[仙台市] 藤帰ラーメン 藤帰ラーメンは仙台市北区にある塩ラーメンが人気のお店。580円と良心的な価格設定が嬉しい。	[福岡市] 鳳凰ラーメン 鳳凰ラーメンは福岡市東区にある塩ラーメンが人気のお店。980円と高価だが行く価値がある。	[東京都] 柳風ラーメン 柳風ラーメンは東京都西区にある塩ラーメンが人気のお店。980円と高価だが行く価値がある。

♣ グラフを描画する

ラーメン人気店紹介

全国各地の美味しいラーメン店を紹介します。

[京都市] 櫻幕ラーメン 櫻幕ラーメンは京都市東区にある醤油ラーメンが人気のお店。280円とお財布に優しいお店である。	店名 櫻幕ラーメン
	住所 京都市東区桜幕1-2-3
	電話 075-222-1234
	営業時間 10:00 - 23:00
	定休日 年中無休

京都より徒歩40分。
円山公園で営業しております。
ご来店、お待ちいたしております。

総合評価 ★★★★☆



第 2 章

トップページの機能拡張

この章では、トップページに関する次のような機能拡張を行っていきます。

ファビコン、アップルタッチアイコン / 検索エンジン用の説明文 / リンクの無効化 / 写真のポップアップ / 写真をスライドさせる / 全画面ヒーローイメージ / スクロールを促す為の表示 / スクロールすると出現するメニュー / 光りながら出現する文字 / 見出しの装飾 / 装飾的なボタン / トップに戻るボタン

また、JavaScript の使い方についても触れてきます。

2.1 ファビコンとアップルタッチアイコン

ファビコンとは、ブラウザのタブの左側に表示される小さなアイコンのことです。



アップルタッチアイコンとは、iPhone / iPad でサイトをホーム画面に登録すると表示されるアイコンです。

どちらもサイトを印象づけ、実装も容易ですので、ご利用下さい。



♣ アイコンの作成

ファビコン、アップルタッチアイコンの作成は、様々なファビコンを一括生成。favicon generator^{*1}を利用すると簡単に行えます。

1. ファビコン、アップルタッチアイコンにしたい正方形の画像を作成します。512x512 がお薦めサイズとのことですので、このサイズに予めしておきます。
2. 「画像ファイルを選択」ボタンを押して、作成した画像をサイトにアップします。
3. 「ファビコン一括生成」ボタンを押します。

*1 <https://ao-system.net/favicongenerator/>

4. さまざまな大きさの画像が作成され表示されるので、画像の下側にある「ファビコンダウンロード」ボタンを押します。
5. ダウンロードされた画像たちを解凍(展開)します。たくさんの画像がありますが、使うのは二種類、favicon.icoと、apple-touch-icon-180x180.pngです。
6. この二つの画像ファイルを、imagesディレクトリに置き、以下のようなコードを<head>内に追加します。するとブラウザのタブの左側にファビコンが、iPhoneの「ホーム画面に追加」した際に、アップルタッチアイコンが表示されるようになります。

▼index.html

```
<head>
  <!-- ファビコン -->
  <link rel="icon"
        href="images/favicon.ico"
        type="image/ico">

  <!-- アップルタッチアイコン -->
  <link rel="apple-touch-icon"
        href="images/apple-touch-icon-180x180.png"
        type="image/png"
        sizes="180x180">
</head>
```

2.2 検索エンジン用の説明文

ウェブサイトを検索すると、そのサイトの簡単な説明文が表示されます。特に指定しなくとも、Googleがそれらしい説明文を提示しますが、自分で書くこともできますので、やってみましょう。

作成は簡単で、<head>内に次のように書くだけです。

▼index.html

```
<head>
  <!-- 検索エンジン用の説明文 -->
  <meta name="description"
        content="概ね80文字前後のサイトの紹介文が検索時に表示されます。">
</head>
```

2.3 リンクの無効化

リンク先のページが準備されていると良いのですが、次回更新の予告など、何らかの理由でリンクを無効化したい時もあります。

style="pointer-events: none;"と書くことでリンクを無効化できます。

```
<a href="post04.html" style="pointer-events: none;">
  次回予告記事（居心地のいい部屋にあるもの）
</a>
```

(インラインスタイルを用いるよりは、HTMLではclass="disabled"とクラス名を付与し、CSSでa.disabled { pointer-events: none; }と書いたほうが良いです。)

2.4

写真をポップアップウィンドウで表示する

ウェブサイトに写真を掲載することはよくあります。タグを使うことで、写真を載せることができます。より効果的に写真を見せることができるよう、先人の方が様々な有益な「ライブラリ」を提供してくださっています。

ライブラリ

ライブラリとは、図書館、図書室、資料室、書庫、書斎、蔵書、文庫、選書、双書などの意味を持つ英単語。ITの分野では、ある特定の機能を持ったコンピュータプログラムを他のプログラムから呼び出して利用できるように部品化し、そのようなプログラム部品を複数集めて一つのファイルに収納したものをライブラリという。^{*2}

様々なライブラリがありますが、ここでは Magnific Popup をご紹介いたします。

「画面上にある小さな写真」をクリックしたときに、「大きく表示させる」ことができます。

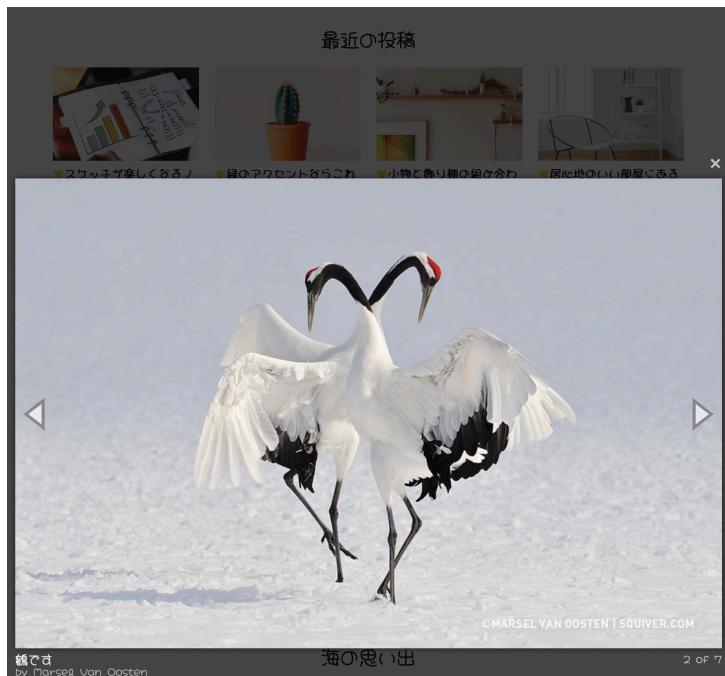
動物園の思い出

動物園に行ってきました。いろいろな動物に会えました。



▲図 2.1: 小さな写真

^{*2} 出典: IT用語辞典 (<https://e-words.jp>)



▲図2.2: クリックすると大きく表示される

♣ JavaScript の簡単な紹介

今まで、HTMLとCSSを使って、ウェブサイトを創ってきましたが、動きのあるウェブサイトを創るために、JavaScriptを使います。

JavaScriptとは

JavaScriptとは、主にWebページに組み込まれたプログラムをWebブラウザ上で実行するため用いられるプログラミング言語。ページ内の要素に動きや効果を加えたり、閲覧者の操作に即座に反応して何らかの処理を行ったりするのに用いられる。^{*3}

今回の例ですが、ブラウザで読み込んだHTMLの要素を操作する機能がJavaScriptにはありますので、これによってクリックされた写真を大きく表示したりします。

始めて触れる方もいらっしゃるかもしれませんので、簡単に概要をご紹介いたします。より詳しくは巻末の参考文献等をご覧ください。

それでは、練習としてindex.htmlとaisatsu.jsを次のように書きましょう。

▼index.html

```
<html>
<head>
```

*3 出典: IT用語辞典 (<https://e-words.jp>)

```
<!-- (略) -->
</head>
<body>
<!-- (略) -->

<!-- aisatsu.js という JavaScript プログラムを、読み込む -->
<script src="aisatsu.js"></script>
</body>
</html>
```

▼aisatsu.js

```
//「おはよう」と、アラート(警報)を出すプログラム
alert("おはよう");
```



▲図 2.3: 実行例

`<script src="aisatsu.js"></script>`で、aisatsu.js を読み込んでいます。`<script>`タグは、いろいろなところに記述できますが、基本的には `</body>`の直前にするのがお薦めです。

また、素の JavaScript ^{*4}をより使い易くするライブラリとして、jQuery があります。

jQuery とは

jQuery とは、Web ブラウザ上で動作する JavaScript ライブラリの一つ。簡潔な記述で豊富な機能 (Web ページの要素に演出効果やアニメーションなどを追加したり、スタイルやイベント起動の設定や変更など) を活用できる。また様々な機能を実現する豊富な対応プラグインが公開されている。jQuery は独特の記法を使い、複数の処理を容易に組み合わせられるようになっている。機能のほとんどは「\$」あるいは「jQuery」という名前のオブジェクトのメソッドとして定義されている。

最初に\$関数に CSS セレクタに似た記法で \$("h1") のようにページ内の操作したい DOM (Document Object Model) 要素を指定すると、その要素を内部に持つ jQuery オブジェクトが返されるため、これに実行したいメソッドを指定する。例えば、`$("#aisatsu").text("こんにちは")`と書くと、#aisatsu 要素の中身を、「こんにちは」に変更できる。

それでは、素の JavaScript と、jQuery の違いを見てみましょう。

▼index.html

```
<body>
<h1 id="aisatsu">おはよう</h1>
```

^{*4} 素の JavaScript のことを、Vanilla JS と呼ぶこともあります

```
<script src="ohayou_vanilla.js"></script>
</body>
```

▼aisatsu_vanilla.js

```
// 素のJavaScript(Vanilla JS)での例
document.getElementById("#ohayou").innerText = "こんにちは";
```

▼index.html

```
<body>
  <h1 id="aisatsu">おはよう</h1>

  <script src="aisatsu_jquery.js"></script>
</body>
```

▼aisatsu_jquery.js

```
// jQuery の例
$("#aisatsu").text("こんにちは");
```

どちらも、index.html 内の id="aisatsu" が付与された要素を「こんにちは」に書き換えます。

従来はブラウザ間の挙動の相違が著しく、また JavaScript 自体の機能も整っていなかったため、各ブラウザ間の差違を埋め、簡潔に容易に記述することができる jQuery が盛んに用いられました。今日では、ブラウザの互換性の向上や、JavaScript の機能向上に伴い、素の JavaScript や上位互換となる TypeScript を使って書くケースも増えて参りましたが、ここでは jQuery で書かれた多くの優秀なライブラリがありますので、使っていきましょう。

ウェブサイトにjQueryを取り込むには、CDNを使うと便利です。

.....
CDN とは

CDN【Content Delivery Network】コンテンツデリバリネットワークウェブ上で送受信されるコンテンツをインターネット上で効率的に配達するために構築されたネットワーク^{*5}

.....

有名な CDNとして、cdnjs^{*6}や、jsDelivr^{*7}があります。ここでは、cdnjsを使ってみましょう。

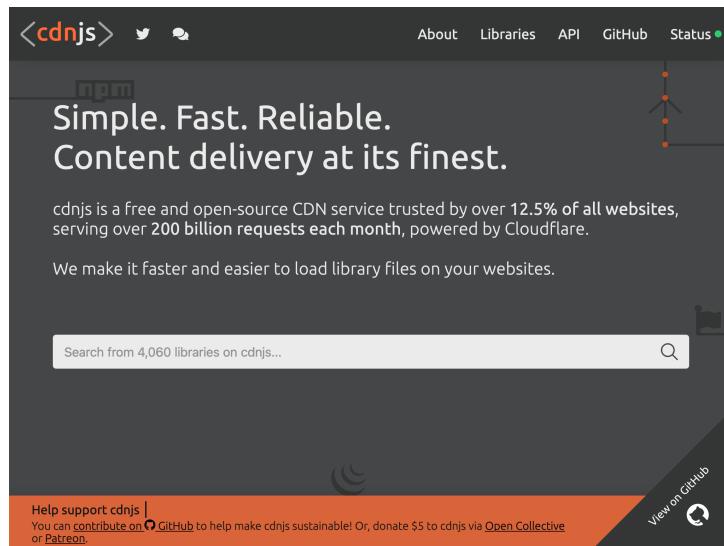
1. cdnjs^{*8}のサイトにアクセスします。

*5 出典: IT用語辞典 (<https://e-words.jp>)

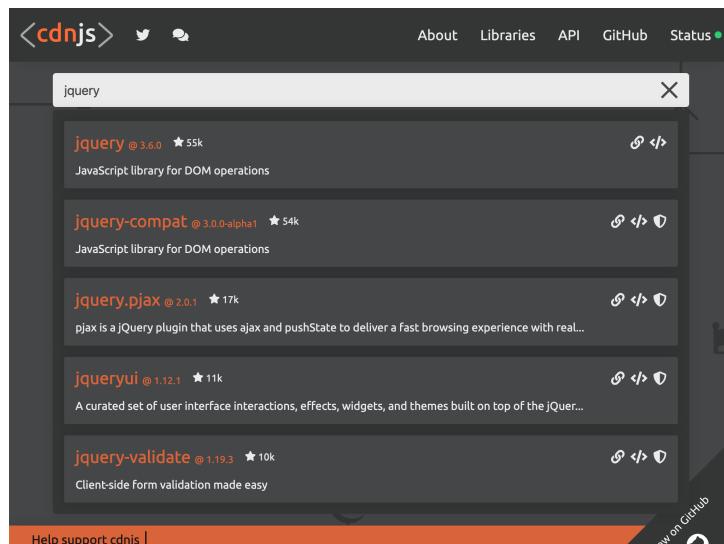
*6 <https://cdnjs.com/>

*7 <https://www.jsdelivr.com/>

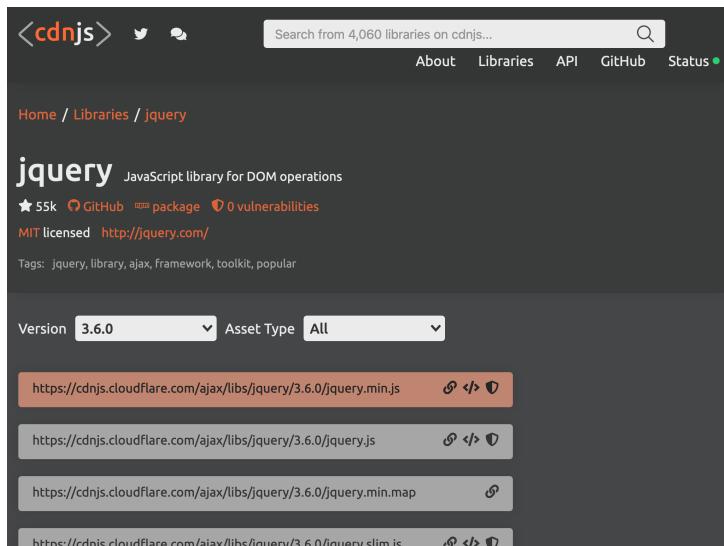
*8 <https://cdnjs.com/>



2. 中央にある検索窓に **jQuery** と入力、検索します。
候補が表示されますので、一番上の **jquery @ 3.6.1** をクリックします。



3. **jQuery** のいくつかの種類が表示されますが、ここでは一番上に表示されている **jquery.min.js** を使うことにしましょう。</>ボタンをクリックすると、コピーできますので、エディタに貼り付けます。



▼index.html

```
<body>
<!-- (略) -->

<!-- jQueryをCDNから読み込む -->
<script
  src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"
  integrity="sha512-894YE6QWD5I59HgZ0GReFYm4dnWc1Qt5NtvYSaNcOP+u1T9qYdvdihz0PPSiiqn
>/+/3e7Jo4EaG7TubfWGUrMQ=="
  crossorigin="anonymous"
  referrerpolicy="no-referrer">
</script>
</body>
```

少し長いですので、以下のように省略して書くこともできます。

▼index.html

```
<!-- jQueryをCDNから読み込む -->
<script
  src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js">
</script>
```

jQuery のファイルの種類について

jquery.min.js, jquery.js, jquery.min.map, jquery.slim.js, jquery.slim.min.js, jquery.slim.min.map と 6 つのファイルが表示されました。拡張子が .js となっているものは、JavaScript のファイルで、.map となっているものは、「ソースマップ」と呼ばれるファイルです。

ブラウザが実行する JavaScript ソースは、開発者が作成した元のソースから何らかの方法で変換される場合があります。ソースマップは変換後のソースと元のソースを関連付けるファイルであり、^{*9} プログラムのデバッグ時に活用されます。

^{*9} 出典: ソースマップを使用する (https://developer.mozilla.org/ja/docs/Tools/Debugger/How_to/Use_a_source_map)

拡張子が .js となっているものの内、jquery.js と jquery.min.js は同じものです。人に読み易く書かれている jquery.js を、圧縮（ミニファイ）してブラウザで速く読み込めるようにしたものが jquery.min.js です。

jquery.slim.js と jquery.slim.min.js は、全ての機能を備えた jquery.js から、利用機会の少ないアニメーション機能などを削除したものです。

♣ 写真ポップアップ機能を実装する

公式サイト^{*10} に詳しく紹介されておりますが、簡単に使い方を説明します。

Magnific Popup や Slick、Vegas など、よく使われる有名なライブラリは、CDN に公開されています。簡潔に記述でき、応答速度の改善も図ることができますので、活用していきましょう。

それでは、写真ポップアップのためのライブラリ、Magnific Popup を使って行きましょう。HTML の全体は次のようになります。CSS は head タグに、JavaScript は body タグの一番最後に書きます。

▼index.html

```
<head>
  <!-- Magnific Popup -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/magnific-popup@1.1.0/dist/magnific-popup.css">
  <link rel="stylesheet" href="magnific-popup-custom.css">
</head>

<body>
  <section class="zoo">
    <h2>動物園の思い出</h2>
    <p>動物園に行ってきました。いろいろな動物に会えました。</p>

    <div>
      <a href="images/tora.webp" title="大きな虎です">
        
      </a>
      <a href="images/tsuru.webp" title="鶴です">
        
      </a>
      <a href="images/zou.webp" title="象さん 象さん お鼻が長いのね">
        
      </a>
      <a href="images/saru.webp" title="お猿さんです">
        
      </a>
      <a href="images/hakucho.webp" title="白鳥の湖">
        
      </a>
      <a href="images/araiguma.webp" title="洗熊です">
        
      </a>
      <a href="images/kuma.webp" title="熊さんです">
        
      </a>
    </div>
  </section>
```

*10 <https://dimsemenov.com/plugins/magnific-popup/>

```
<!-- jQuery -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>
<!-- Magnific Popup -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/magnific-popup.js/1.1.0/jquery.magnific-popup.min.js"></script>
<script src="magnific-popup-custom.js"></script>
</body>
```

♣ HTML の解説

HTML を次のように書き、写真ギャラリー用の画像を、image ディレクトリに入れておきます。サムネイル画像（小さな画像）も準備しておきます。

▼写真表示用の HTML

```
<div>
  <a href="images/tora.webp" title="大きな虎です">
    
  </a>
</div>
```

で使われている tora_s.webp は、元の画像ファイル tora.webp を小さくしたもので、「サムネイル」と呼ばれる画像です。小さな画像を用意することで、ウェブサイトを見る人が心地よく閲覧できるようになります。

作成者の意図に応じて、写真の大きさや動きをカスタマイズすることができます。そのために次のコードを書き、magnific-popup-custom.css と magnific-popup-custom.js を読み込んでいます。

▼index.html

```
<link rel="stylesheet" href="magnific-popup-custom.css">
<script src="magnific-popup-custom.js"></script>
```

♣ カスタマイズ用の CSS

Magnific Popup をカスタマイズする為の CSS は、一例として次のように書きます。

▼magnific-popup-custom.css

```
/* 画像の大きさを指定 */
.zoo div {
  /* 横一杯に使う */
  grid-column: 1 / -1;

  /* aタグのためのコンテナ(箱)にする */
  display: grid;
  grid-gap: 10px;

  /* grid-auto-fill */
  grid-template-columns: repeat(auto-fill, minmax(180px, 1fr));
}

.zoo div a img {
  vertical-align: bottom;
}
```

```
/* 写真をクリックしたときに、ズームアップする為の設定 */
.mfp-with-zoom .mfp-container,
.mfp-with-zoom.mfp-bg {
    opacity: 0;
    backface-visibility: hidden;
    transition: all 0.3s ease-out;
}

.mfp-with-zoom.mfp-ready .mfp-container {
    opacity: 1;
}

.mfp-with-zoom.mfp-ready.mfp-bg {
    opacity: 0.8;
}

.mfp-with-zoom.mfp-removing .mfp-container,
.mfp-with-zoom.mfp-removing.mfp-bg {
    opacity: 0;
}
```

grid-auto-fill

```
grid-template-columns: repeat(auto-fill, minmax(180px, 1fr));
```

と書くことで、グリッドの列を生成できます。 <https://www.webprofessional.jp/difference-between-auto-fill-and-auto-fit/> に分かりやすい説明がございますので、ご覧になつてください。

♣ カスタマイズ用の JavaScript

Magnific Popup をカスタマイズする為の JavaScript は、次のように書きます。

▼ magnific-popup-custom.js

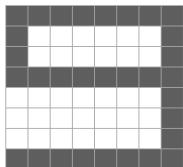
```
$(".zoo div").each(function () {
    $(this).magnificPopup({
        // 基本設定いろいろ
        delegate: "a",
        type: "image",
        tLoading: "Loading image# % curr % ...",
        mainClass: "mfp-img-mobile",
        gallery: {
            enabled: true,
            navigateByImgClick: true,
            preload: [0, 1],
            arrowMarkup: `<button title="%title%" type="button" class="mfp-arrow mfp-arrow-%dir%"></button>`,
            tPrev: "Previous(Left arrow key)",
            tNext: "Next(Right arrow key)",
            tCounter: '<span class="mfp-counter"> %curr% of %total% </span>'
        },
        image: {
```

```
tError: '<a href="%url%"> The image# %curr% </a> could not be loaded.',  
titleSrc: function (item) {  
    return `${item.el.attr('title')} <small> by Marsel Van Oosten </small>`;  
},  
  
// クリックしたときにズームアップするための設定  
mainClass: "mfp-with-zoom",  
zoom: {  
    enabled: true,  
    duration: 300,  
    easing: "ease-in-out",  
    opener: function (openerElement) {  
        return openerElement.is("img") ? openerElement : openerElement.find("img");  
    }  
}  
});  
});
```

以上で、Magnific Popup を使った写真ギャラリーの作成は完了です。

【コラム】コンピュータによる画像表現

ラスター形式



コンピュータで画像を表現するにはどのようにしたら良いのでしょうか。左は $8 \times 8 = 64$ 個の画素を使って作成した 数字の「9」の画像です。画素が光っていない黒を 0、光っている白を 1 とすると、

00000000 01111110 01111110 00000000 11111110 11111110 11111110
00000000

の 64 bit (= 8 Byte) で表現できます。

白黒画像でしたらこれで完了ですが、カラー写真では、フルカラー 16,777,216 色を表現するために、赤 256 段階 (=8 ビット)、緑 256 段階 (=8 ビット)、青 256 段階 (=8 ビット) が必要ですから、8 Byte $\times 8 \times 8 \times 8 = 512$ Byte となります。

iPhone では、 $3,024 \times 4,032 = 12,192,768$ もの画素数で写真を撮ることができ、さらに赤 1,024 段階 (=10 ビット)、緑 1,024 段階 (=10 ビット)、青 1,024 段階 (=10 ビット) の 1,073,741,824 色が表現できます。なので、写真一枚を保存するために、 $3,024 \times 4,032 \times 10 \times 10 \times 10 = 12,192,768,000$ bit (=1,524,096,000 Byte = 1,488,375 kB = 1,453 MB = 1.4 GB) もの容量が必要となります。数十枚写真を撮るだけで、容量一杯となってしまいます。

そこで、写真の情報量をそのまま保存するのではなく、「圧縮」して保存することにしましょう。先ほどの数字の「9」の例ですと、

黒黒黒黒黒黒黒 黒白白白白白白白 黑白白白白白白黑 黑黒黒黒黒黒黒 白白白白白白白白黒
白白白白白白白白 黑白白白白白白白 黑黒黒黒黒黒黒黒

とそのまま表現する代わりに、同じ色が続いているところは色の数を記すことにすると

黒9白6黒2白6黒9白7黒1白7黒1白7黒9

と 22 文字で表現できます。さらに黒の次は白なので、省略できそうです。すると、

9 6 2 6 9 7 1 7 1 7 9

と 11 文字で表現できます。

そのまま表現していた際には 64 文字必要でしたが、 $11 \div 64 = 17\%$ と 約 1 / 6 で済みました。これが圧縮の原理です。(ランレンジス圧縮と呼ばれ、FAX などで用いられています。)

FAX などで用いられているランレンジス圧縮ですが、繰り返しが少ないと効率が悪化するという弱点を抱えています。そこで、より優れた様々な圧縮アルゴリズムが考案されています。例えば、iPhone では HEVC(High Efficiency Video Coding) 方式を採用し、一枚の写真を保存するために必要な 1.4 GB を、数 MB と、数百分の一に圧縮しています。

次世代画像形式のWebP

様々な画像形式が考案され、写真用の JPEG, 図やイラストのための GIF, PNG が主流となりましたが、こうした中登場したのが、次世代画像形式の WebP です。

WebP とは、グーグル社が開発・公開している画像ファイル形式の一つ。標準のファイル拡張子は「.webp」。Web ページへ埋め込む静止画像に適した画像形式として、既存の JPEG や GIF、PNG の置き換えが可能である。

JPEG のような写真に適した高圧縮率の非可逆圧縮と、GIF や PNG のような図表やイラストに適した可逆圧縮の両方に対応する。透過 PNG のようなピクセル単位の透過度（アルファチャンネル）が非可逆圧縮でも利用でき、GIF アニメーションのような簡単なアニメーション機能（フルカラー画像や非可逆圧縮も可）にも対応する。【出典：IT用語辞典】

ウェブ制作会社 ICS が提供する技術情報メディアがあります。HTML / CSS / JavaScript を中心とした記事が多数掲載されています。WebPについて書かれた記事もございましたので、引用してご紹介いたします。^{*11}

次世代画像形式の WebP、そして AVIF へ

長い間、Web の静止画に関しては「写真の JPEG、ロゴやイラストの GIF、透過画像の PNG」という明確な使い分けが確立されてきました。WebP はこのすべてを置き換えることができる次世代のフォーマットです。

WebP は JPEG/GIF/PNG(APNG) をカバーする魅力的なフォーマット

WebP を使うことで、これまで用途や画像の特徴ごとに使い分けが必要だったフォーマットの一本化が可能になります。主な特徴を簡単に紹介しましょう。

- 高い圧縮率：同等画質の JPEG と比較して 20~30% のサイズ削減（JPEG の置き換え）
- 不可逆圧縮と透過アニメーションの併用（透過アニメーションでも画質を犠牲にしてサイズを削減できる）（GIF/PNG の置き換え）
- 画質劣化のない可逆圧縮もサポート（GIF/PNG の置き換え）



さらに次世代のフォーマット、AVIF も

- 多様な色空間やサンプリング方式をサポート
- WebP よりもさらに高画質でコンパクト（同じサイズでも画質が高く、JPEG に特有のブロックノイズも発生しない）
- Amazon・Netflix・Google・Microsoft・Mozilla 等の幅広い企業によるコンソーシアムが共

同で開発（Facebook や Apple も後から参画）



ImageMagick *12 等のツールを導入することで、簡単に画像形式を変換することができますし、また、ネット上でオンラインで変換してくれるサイトもございます。

画像の表示も速くなり、利用者に快適に閲覧してもらえますので、使っていきましょう。

2.5 写真をスライドさせる

slick^{*13} というライブラリを使うと、写真をスライドさせることができます。公式サイトにも詳しく例や説明がありますので、ご覧ください。



*5 次世代画像形式の WebP、そして AVIF へ (<https://ics.media/entry/201001/>)

*6 ImageMagick の使いかた 日本語マニュアル (<https://imagemagick.biz/>)

*13 <https://kenwheeler.github.io/slick/>

ここでは、Grid Of Sliders^{*14}として、Jack HarnerさんがCode Penに公開している例をもとに実装します。コード中にコメントを入れておりますので、写真などを変更されてご活用下さい。

▼index.html

```
<head>
  <!-- slick-carousel -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick.min.css">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick-theme.min.css">
  <link rel="stylesheet" href="slick-custom.css">
</head>

<body>
  <section class="sea">
    <div class="slider-wrapper">
      <h2>海の思い出</h2>
      <p>海に行ってきました。波の音はとっても心落ち着きます。</p>

      <div class="slider">
        <div class="card">
          
          <h3>空飛ぶ鷗</h3>
        </div>
        <!-- (略) -->
        <div class="card">
          
          <h3>幾星霜に耐えた老松</h3>
        </div>
      </div>
    </div>
  </section>

  <!-- jQuery -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>
<script>
  <!-- Slick -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick.min.js"></script>
  <script src="slick-custom.js"></script>
</body>
```

▼slick-custom.css

```
.slider-wrapper {
  grid-column: 2;
  max-width: 100%;
  min-width: 0px;
  justify-self: center;
}

.card {
  color: black;
  text-align: center;
  display: inline-flex !important;
  align-items: center;
```

*14 <https://codepen.io/jackharner/pen/gyyeEd>

```

justify-content: center;
flex-direction: column;
max-width: 100%;
min-width: 0px;
margin: 1em; /* 写真に少し間隔を入れる為 */
}

.card img {
  max-width: 100%;
}

.title {
  font-size: 4em;
  text-align: center;
  margin: 0.25em;
}

/* 戻る矢印、次へ矢印 */
.slick-prev,
.slick-next {
  position: absolute;
  top: 42%;
  cursor: pointer;
  border-top: 2px solid #666;
  border-right: 2px solid #666;
  height: 15px;
  width: 15px;
}

.slick-prev {
  left: -1.5%;
  transform: rotate(-135deg);
}

.slick-next {
  right: -1.5%;
  transform: rotate(45deg);
}

.slick-prev::before,
.slick-next::before {
  content: "";
}

```

▼slick-custom.js

```

$(".slider").slick({
  slidesToShow: 3,          // 画面に見せるスライドの枚数
  slidesToScroll: 3,        // 1回のスクロールで移動するスライドの枚数
  dots: true,              // 画面下部に案内用ドット（点）を表示
  arrows: true,             // 左右の矢印表示
  autoplay: true,            // 自動再生
  autoplaySpeed: 2000,       // 自動再生速度(ms)
  rows: 1,                  // スライド表示に用いる行数
  pauseOnHover: true,        // ホバー時に停止するか否か
  responsive: [
    {
      breakpoint: 768,        // 端末の横幅が768px以下の見せ方

```

```

        settings: {
          slidesToShow: 2,
          slidesToScroll: 2,
        }
      },
      {
        breakpoint: 428,      // 端末の横幅が428px以下の見せ方
        settings: {
          slidesToShow: 1,
          slidesToScroll: 1,
        }
      }
    ]
  );
}
);

```

2.6 全画面のヒーローイメージ

サイトを印象づける「ヒーローイメージ」。これを全画面にしてみましょう。^a

JavaScript を使って、何枚かの絵が自動で切り替わるようになります。Vegas という素敵な「ライブラリ」が公開されていますので、それを使うと簡単に実装できます。

いつものように、CSS は head タグに、JavaScript は body タグの一番最後に書きます。

^a 卷末の参考文献「動く Web デザインアイディア帳」より引用、適宜改変しています。



▼index.html

```

<head>
  <!-- Vegas - Fullscreen Backgrounds and Slideshows. -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/vegas/2.5.4/vega>
>as.min.css">
  <link rel="stylesheet" href="vegas-custom.css">

```

```

</head>

<body>
  <div id="vegas_slider">
    <h1 class="glow text">Best place to visit in the world</h1>
    <a href="https://github.com/Atelier-Mirai/wave-improve" class="btn btn-gradient">
      <span>ソースコードは<br class="mobile only">こちら</span>
    </a>
  </div>

  <!-- jQuery -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>
  <!-- Vegas -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/vegas/2.5.4/vegas.min.js"></script>
</body>

```

▼vegas-custom.css

```

#vegas_slider {
  /* スライダー全体の縦幅を全画面にする */
  width: 100%;
  height: 100vh;
}

#vegas_slider h1 {
  position: absolute;
  z-index: 2;
  top: 40%;
  left: 50%;
  transform: translate(-50%, -50%);
  text-align: center;
  font-size: 6vw; /* 6vw は、幅の 6% という単位指定 */
  letter-spacing: 0.2em;
  text-transform: uppercase; /* (小文字でも)大文字に変換する */
  color: white;
  width: 80%;
}

@media(min-width: 768px) {
  #vegas_slider h1 {
    font-size: 48px; /* 大画面で大きく成りすぎるのを防ぐ */
  }
}

#vegas_slider a {
  grid-column: 2 / 4;
  position: absolute;
  z-index: 2;
  top: 60%;
  left: 50%;
  transform: translate(-50%, -50%);
  text-align: center;
  letter-spacing: 0.1em;
  text-transform: uppercase; /* (小文字でも)大文字に変換する */
  color: white;
}

```

```
}
```

▼vegas-custom.js

```
$('#vegas_slider').vegas({
  overlay: true,
  transition: 'blur',
  transitionDuration: 2000,
  delay: 10000,
  animationDuration: 20000,
  animation: 'kenburns',
  slides: [{ src: './images/room.webp'},
            { src: './images/room02.webp'},
            { src: './images/room03.webp'},
            { src: './images/room04.webp'},
            { src: './images/room05.webp'}]
});
```

2.7 スクロールを促す為の表示

ヒーローイメージが全画面になりましたが、このサイトを閲覧した方がどのようにしたら次のページが見られるのか、もしかすると戸惑うかもしれません。UI/UX の観点から、下へのスクロールを促すようにしてみましょう。CSS アニメーション機能を使って実現します。^a

^a 巻末の参考文献「動く Web デザインアイディア帳」より引用、適宜改変しています。



▼index.html

```
<head>
  <link rel="stylesheet" href="scroll.css">
</head>

<body>
  <!-- スクロールを促す -->
  <div class="scrolldown">
    <span>Scroll</span>
  </div>
</body>
```

▼scroll.css

```
/* スクロールを促す
-----*/
.scrolldown {
  grid-row: vegas;
  justify-self: center;
  align-self: end;
  padding-bottom: 60px;
}

/* 表示位置 */
.scrolldown {
  position: relative;
```

```

}

/* Scrollという文字表示の設定 */
.scrolldown span {
  color: var(--sakurairo);
  font-size: 1rem;
  letter-spacing: 0.05em;
}

/* 線の表示 */
.scrolldown::after {
  content: "";
  position: absolute;
  top: 220px;
  left: 35px;
  width: 2px;
  height: 30px;
  background: var(--sakurairo);
  animation: pathmove 1.4s ease-in-out infinite;
  opacity: 0;
}

/* 高さ・位置・透過度が変化して、線が上から下に動く */
@keyframes pathmove {
  0% {
    height: 0;
    top: 25px;
    opacity: 0;
  }
  30% {
    height: 30px;
    opacity: 1;
  }
  100% {
    height: 0;
    top: 60px;
    opacity: 0;
  }
}

```

【コラム】position: absolute と position: relative について

position: absolute と position: relative については、UX MILK さんの記事 <https://uxmilk.jp/63409> が分かりやすくお勧めです。以下に引用いたします。

CSS を記述するときに position プロパティを利用して、要素の位置をずらすことがあります。そのときに出てくるのが「position: absolute (絶対位置)」「position: relative (相対位置)」です。なんとなく使っていたけど、「絶対位置」「相対位置」と言われてもなかなかピンとこない人もいるのではないかでしょうか。今回は実際に配置した例を見て、「absolute」と「relative」について確認していきましょう。

♣ position プロパティとは

まずは、「position: absolute」「position: relative」を使ううえで欠かせない position プロパティについて説明します。

position プロパティは、「要素を配置する基準」を指定するためのプロパティです。「position: absolute」と「position: relative」はこの「基準」の場所を区別するためにあるということを理解していると、この先の説明もわかりやすくなると思います。

♣ absolute とは

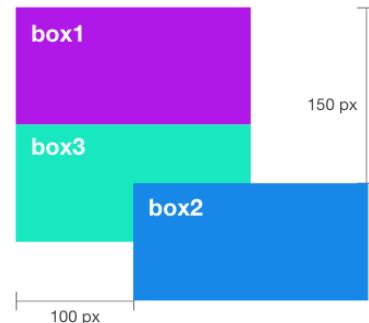
「position: absolute」は移動するときの基準がウィンドウ、または親要素になります。

つまり複数の要素がある場合でも「position: absolute」で指定すると、他の要素を無視して左上（top:0 left:0 の位置）から移動するということです。

実際に配置した例を見ていきましょう。200px × 100px の box を3つ並べました。

次に、青色の box2 に { position: absolute; top:150px; left:100px } を指定します。

すると、box2 はウィンドウの左上を基準（起点）にして、上から 150px、左から 100px の位置に移動しました。さらに position: absolute を指定した要素は高さがなくなり、浮いたような状態になるため、box3 は box2 を無視して位置を詰めています。



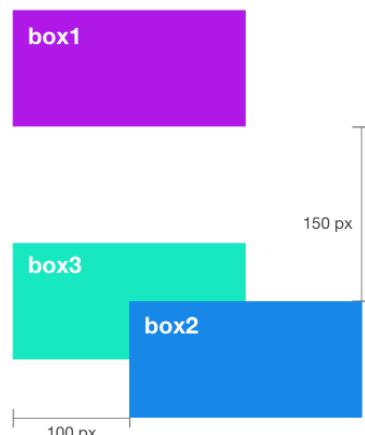
♣ relative とは

relative は移動するときの基準が元いた位置になります。つまり position を記述する前に配置されていた場所から移動するということです。

box2 に、{ position: relative; top:150px; left:100px } を指定した例を見ていきましょう。

box2 は自分が元々いた位置を基準にして、上から 150px、左から 100px の位置に移動しています。

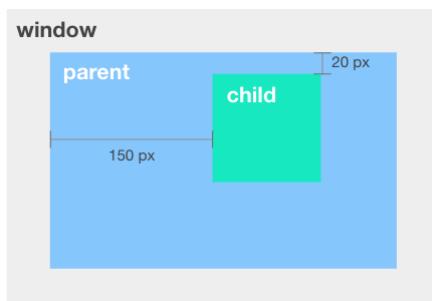
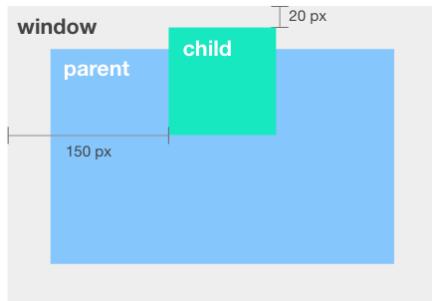
また、position: absolute とは違い、移動させた要素の高さが残るため、box3 は位置を詰めずそのままの位置に表示されています。



♣ absolute と relative の使い方

実際にウェブサイトを制作すると、親要素を基準にして子要素を移動させたいという場合が多いです。しかし要素に空白を残したくないからと、子要素にだけ「position: absolute」を記述してしまうと、思ったように表示することができません。たとえば、parent(青いボックス)の子要素である child(緑のボックス)に { position: absolute; top:20px; left:150px } を指定すると、右のように表示されます。これは、親要素ではなく画面左上を起点に子要素が移動しているからです。

親要素を基準にするためには、親要素に「position: relative」を記述します。こうすることで、子要素が親要素を基準に移動することができます。



♣ まとめ

「position: absolute」と「position: relative」は、最初は混乱するかもしれません、少しでも理解しておくと、実際に position を使ってレイアウトするときやソースコードを参考にしたときにすんなりと記述しやすくなります。ぜひ覚えておきましょう。

2.8 スクロールすると出現するメニュー

全画面でヒーローイメージが表示されるようになりました。スクロールを促すよう、アニメーションも実装しましたので、今度はスクロールすると出現するメニューを作成していきましょう。



WAVE

[TOP](#) [ABOUT](#) [CONTACT](#)

まず、HTML は次のように書きます。

▼ index.html

```
<!-- ヘッダー（ロゴとナビゲーションメニュー）-->
<header class="header hidden">
  <!-- ロゴ -->
  <a class="logo" href="index.html">
    
    <span>WAVE</span>
  </a>
  <!-- ナビゲーションメニュー -->
  <nav class="menu">
    <ul>
      <li>
        <a href="index.html">
          <i class="fa-solid fa-house fa-lg fa-fw"></i>
          トップ
        </a>
      </li>
      <li>
        <a href="about.html">
          <i class="fa-brands fa-pagelines fa-lg fa-fw"></i>
          サイトについて
        </a>
      </li>
      <li>
        <a href="contact.html">
          <i class="fa-solid fa-envelope fa-lg fa-fw"></i>
          お問い合わせ
        </a>
      </li>
    </ul>
  </nav>
</header>
```

サイト名の隣には、ロゴマークも表示させるようにしています。

ナビゲーションメニューは、普通に書いています。タグは一行で書くこともできますが、**HTML整形ツール**^{*15}を使って、読みやすく整えています。^{*16}

それぞれのメニュー項目は、Font Awesome を使って、アイコンも付けています。Font Awesome では、クラス名を付与することで細かい調整も可能です。fa-lg は 1.33 倍、la-2x は 2 倍の大きさ、fa-fw は右に少し間隔が空きます。

より詳しい説明は **Font Awesome アイコンの使い方と便利な機能のまとめ**^{*17} にございますので、ご覧になってください。

それでは、次に CSS を見ていきましょう。^{*18}

▼ header.css

```
/* ふわっと出現させるためのCSS
-----
.header.upward {
  position: fixed;
```

*15 <https://u670.com/pikamap/htmlseikei.php>

*16 紙幅に納まるようにもできました。

*17 <https://coliss.com/articles/build-websites/operation/work/font-awesome-guide-and-useful-tricks.html>

*18 CSS / JavaScript は 卷末の参考文献「動く Web デザインアイディア帳」より引用適宜改変。

```

        animation: upwardAnimation 0.5s forwards;
    }

/* ヘッダーが画面上部に上がって消えていく動き */
@keyframes upwardAnimation {
    from {
        opacity: 1;      transform: translateY(0);
    }
    to {
        opacity: 0;      transform: translateY(-100px);
    }
}

.header.downward {
    position: fixed;
    animation: downwardAnimation 0.5s forwards;
}

/* ヘッダーが画面上部から下に現れてくる動き */
@keyframes downwardAnimation {
    from {
        opacity: 0;      transform: translateY(-100px);
    }
    to {
        opacity: 1;      transform: translateY(0);
    }
}

```

この CSS で把握しておきたいのは、「CSS アニメーション」です。突然メニューが現れ、突然消えるのではなく、余韻を持たせるかのように、ゆっくりと表示され、そして消えていくようにしています。

ヘッダーが現れる動きを見ていきましょう。`animation: downwardAnimation 0.5s` と書き、0.5 秒かけて `downwardAnimation` を実行します。

`downwardAnimation` には、アニメーションの開始時 (`from`) には、`opacity: 0; transform: translateY(-100px);` ですので、透明で上に隠れている状態です。アニメーション終了時 (`to`) には、`opacity: 1; transform: translateY(0);` となり、鮮明に姿を現わします。

CSS アニメーションの使用^{*19} や `animation`^{*20} に詳しい説明がございますので、ご覧になって下さい。

それでは、JavaScript を見ていきましょう。

▼ navigation.js

```

1 const fixedAnimationForHeaderMenu = () => {
2     // スクロール量を取得
3     let scroll = $(window).scrollTop();
4
5     // 300px以上、スクロールしたら
6     if (scroll >= 300) {
7         // ヘッダーを表示
8         $(".header").removeClass("hidden");
9         $(".header").addClass("shown");
10        // 上から現れるよう、動きのためのクラス名を付与
11        $(".header").removeClass("upward");

```

*19 https://developer.mozilla.org/ja/docs/Web/CSS/CSS_Animations/Using_CSS_animations

*20 <https://developer.mozilla.org/ja/docs/Web/CSS/animation>

```

12    $(".header").addClass("downward");
13 }
14 // そうでなければ
15 else {
16     // 画面上部に消えていく動きのためのクラス名を付与
17     $(".header").removeClass("downward");
18     $(".header").addClass("upward");
19 }
20 }
21
22 // スクロールイベント発火で、fixedAnimationForHeaderMenu を呼ぶ。
23 $(window).scroll(fixedAnimationForHeaderMenu);

```

最後の 23 行目で、画面スクロールされた時に関数 `fixedAnimationForHeaderMenu` が呼ばれるようにしています。1 行目から 20 行目までは、実際に呼ばれる `fixedAnimationForHeaderMenu` 関数の定義が書かれています。スクロール量を取得、もし 300px 以上スクロールしていたら、メニュー出現のためのクラス名を付与し、そうでなければ、消失用のクラス名を付与するという、分かりやすいコードです。

補足説明

全画面のヒーローイメージ表示との兼ね合いで HTML で `<header class="hidden">` と書いています。`.hidden { display: none; }` と非表示にしておき、スクロールしたら `class="hidden"` を取り除き、`class="shown"` を付与しています。`.shown { display: grid; }` ですので、ヘッダーメニューが表示されるようになります。

新しい関数の書き方 アロー関数のご紹介

エクマスクリプト ECMAScript

JavaScript は、Ecma International (旧欧州電子計算機工業会 European Computer Manufacturers Association) によって仕様が定められています。第一版から始まり、第五版の ECMAScript 5th edition(ES5) まで順調に機能向上が図られてきました。そして仕様に大きな変更が加えられ、ECMAScript 2015(ES2015) と呼ばれるようになりました (グレゴリオ暦に因んだ命名で、二千十五版ではありません)。それ以降、毎年行われる様々な仕様の追加が行われ、ES2016, ES2017...、となります。

アロー関数

ES2015において、新しい関数の書き方「アロー関数」が導入されました。^{*21 *22}

▼新しく導入されたアロー関数

```
// アロー関数
const add = (a, b) => {
    return a + b;
}
```

*21 => が、「矢印、アロー」に見えるので。

*22 アロー関数式 (https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Functions/Arrow_functions)

比較のために、従来の関数宣言も掲載します。

▼ 従来の関数宣言

```
// 従来の関数宣言
function add(a, b) {
    return a + b;
}
```

使い方はどちらも同じで、次のようにして結果を表示できます。

▼ 関数の呼び出し

```
// 関数の呼び出しと結果の表示
const answer = add(9, 23);
alert(answer);
```

従来の関数とアロー関数への変形

JavaScript の関数は、「第一級オブジェクト」と呼ばれる関数を変数に代入できる性質を持ちます。ですので、次のように変数 add に関数 tashizan を代入して使うことができます。

```
// 変数 add に 関数tashizanを代入する
const add = function tashizan(a, b) {
    return a + b;
}

// 関数の呼び出しと結果の表示
const answer = add(9, 23);
alert(answer);
```

変数 add に代入して呼び出すことができるのであれば、わざわざ関数に tashizan と命名する必要も無いので、名前を省けます（「無名関数」と言います）。

```
// 変数 add に 無名関数を代入する
const add = function (a, b) {
    return a + b;
}
```

function と毎回書くのも文字が長いので省略して、代わりに => を () の後に書きます。

▼ アロー関数の完成

```
const add = (a, b) => {
    return a + b;
}
```

始めての方には、function というキーワードの存在が分かりやすいかと思いますが、関数を簡単に定義できるよう「アロー関数」が導入されました。従来の関数とは細かい動作の面で差違がございますが、これから主流として広く用いられていますので、ぜひ使ってみてください。

2.9 光りながら出現する文字

キャッチフレーズを光るように出現させるために CSS アニメーション機能を使います。そして、手書きでコードを記述するのは大変なので、支援するための JavaScript を自分で書いて行きます。

コードの解説も行いましたので、参考にしていただければ幸いです。

それでは準備です。HTML の <head>内に CSS を、</body>の直前に <script> を書き、必要な JavaScript ファイルを読み込みます。



▼index.html

```
<head>
  <link rel="stylesheet" href="glow-text.css">
</head>

<body>
  <h1 class="glow text">Best place to visit in the world</h1>

  <!-- jQuery -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>
<script>
  <!-- glow text-->
  <script src="glow-text.js"></script>
</body>
```

CSS は次のようにになります。^{*23}

▼glow-text.css

```
/* 文字を光りながら出現させるためのCSS
-----
.glow.text span {
  opacity: 0;
}

/* アニメーションで透過度を0から1に変化させ、text-shadowをつける */
.glow.text.shine span {
  animation: glow_anime_on 1s ease-out forwards;
}

@keyframes glow_anime_on {
  0% { opacity:0; text-shadow: 0 0 0 #fff, 0 0 0 #fff; }
  50% { opacity:1; text-shadow: 0 0 10px #fff, 0 0 15px #fff; }
  100% { opacity:1; text-shadow: 0 0 0 #fff, 0 0 0 #fff; }
}
```

先に紹介した CSS アニメーションを使い、文字の周りに白い影をつけることで、あたかも光っているように見えるという仕組みです。

^{*23} CSS / JavaScript は 巻末の参考文献「動く Web デザインアイディア帳」より引用適宜改変。

さらに順番に光っているように見せたいので、次のように

```
<h1 class="glow text shine">
  <span style="animation-delay: 0.0s;">B</span>
  <span style="animation-delay: 0.1s;">e</span>
  <span style="animation-delay: 0.2s;">s</span>
  <span style="animation-delay: 0.3s;">t</span>
</h1>
```

と一文字ずつ `animation-delay` を付けると、光りながら文字が出現します。

CSS アニメーションの仕組みが分かったところで、次は JavaScript の出番です。

```
<h1 class="glow text">Best place to visit in the world</h1>
```

と書かれた一文字ずつを取り出して、`B` のようにしたいのですが、手作業で行うのは大変です。生産性向上のために、次のコードを用意しましょう。

▼ glow-text.js

```
1 // .glow.text に shineクラス名を付与する関数定義
2 const addShineClassName = () => {
3   let element = $(".glow.text");
4   element.each(() => {
5     let elemPosition = element.offset().top - 50;
6     let scroll      = $(window).scrollTop();
7     let windowHeight = $(window).height();
8
9     // .glow.text要素の位置までスクロールされたなら、shineクラスを付与する。
10    if (scroll >= elemPosition - windowHeight) {
11      element.addClass("shine");
12    } else {
13      element.removeClass("shine");
14    }
15  });
16 }
17
18 // 画面スクロール時に呼び出す関数を記述する
19 $(window).scroll(() => {
20   addShineClassName();
21 });
22
23 $(window).on('load', () => {
24   const DELAY_TIME = 0.1; // 一文字ずつの遅延時間
25
26   $(".glow.text").each((index, element) => {
27     let delay_initial_value = 0 + (index * text.length * DELAY_TIME);
28     let text                = $(element).text();
29     let textbox              = "";
30     text.split('').forEach((t, i) => {
31       let delay = delay_initial_value + i * DELAY_TIME;
32       textbox += `<span style="animation-delay:${delay}s;">${t}</span>`;
33     });
34     $(element).html(textbox);
35   });
36
37   addShineClassName();
38 });
```

前半は先ほどのメニュー表示処理と類似していますので、32行目から解説します。

`$(window).on('load', () => {処理いろいろ})` は、「画像や動画など全ての読み込みが完了したら、『処理いろいろ』を実行する書き方です。`() => {処理いろいろ}` は、「無名関数」と呼ばれる関数で、先のアロー関数の紹介もお読みください。

26行目の左辺`$(".glow.text")`は、HTML内に書かれた`.glow.text`クラスの要素を取得するjQueryでの記法です。これに続く`.each`メソッド(手続き・処理・手法)で、個々の要素を取り出し処理します。

28行目は、要素の中身(コンテンツ)を取り出して、`text`という変数に代入しています。`$element`が`<h1>Best</h1>`の場合、`text`は`Best`となります。

それでは、この`Best`から、一文字ずつ取り出して、`B`のようにする処理を行っていきましょう。

29行目で、処理済のものを格納する変数として`textbox`を宣言しています。最初は処理が終わったものがないので、空っぽ('')です。

30行目から実際の処理を行います。`text.split('')`で、`split`という名前の通り、一文字ずつばらばらにされます。そして`.forEach((t, i)`へ送られます。`.forEach((t, i)`メソッドの引数`t`にはばらばらにされた文字一つ一つが渡されています(`B, e, s, t`と一文字ずつ渡されます)。`i`には何番目の文字であるかが渡されています(`0, 1, 2, 3`と`0`から数えます)。

31行目は、`delay`を算出しています。順番に`0.0, 0.1, 0.2, 0.3`となります。

32行目の`+=`は「自己代入演算子」です。一般にプログラミング言語では、`age = age + 1`のように、自分自身の値に`1`を加えたものを、自分自身に代入することがよくあります。例えば、年齢`(age)10`歳の人は来年には`11`歳になります。このような処理は良く出てくるので、`age = age + 1`を簡潔に書けるよう`age += 1`という記法が用意されています。

同じく32行目の右辺を見ていきます。

``${t}``は、
` B `ととてもよく似ています。
違いを探すと、 `${delay}` が、`0.0`に `${t}` が、`B`に変わっています。変数`delay`は、`0.0, 0.1, 0.2, 0.3`と、変数`t`は、`B, e, s, t`と変化します。JavaScriptでは、「文字列リテラル」と呼ばれる「変数を文字列の中に埋め込み展開する」記法が用意されています。``<span ${delay} ${t} ``と、展開したい変数を文字列の中に埋め込むことで、楽に記述できます。ちなみに前後を囲んでいる記号`'`は、「バッククオート」と呼ばれます。普段よく見かける`'(クォーテーション)`や`"(ダブルクオーテーション)`とは逆に、左上から右下に向かが傾いているので、「バッククオート」と呼ばれます。

34行目は、jQueryで、要素を書き換えるメソッドです。

```
<h1 class="glow text">Best</h1>
```

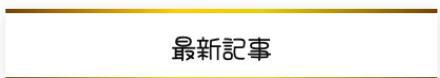
であった`element`が次のように書き換わります。

```
<h1 class="glow text">
  <span style="animation-delay:.0s;">B</span>
  <span style="animation-delay:.1s;">e</span>
  <span style="animation-delay:.2s;">s</span>
  <span style="animation-delay:.3s;">t</span>
</h1>
```

最後に37行目で`class="glow text"`を`class="glow text shine"`とクラス名を付与します。

2.10 見出しの装飾

せっかくですので、綺麗に見出しを装飾してみましょう。簡単に CSS を書くだけで実現できます。



最新記事

HTML はとても簡潔です。

▼index.html

```
<h2>最新記事</h2>
```

CSS は次の通りです。

▼text.css

```
h2 {
    position: relative;
    padding: 1rem 1.5rem;
    box-shadow: 0 2px 14px rgba(0, 0, 0, .1);
}

h2:before,
h2:after {
    position: absolute;
    left: 0;
    width: 100%;
    height: 4px;
    content: '';
    background-image: linear-gradient(135deg,
        #704308 0%,
        #ffce08 40%,
        #e1ce08 60%,
        #704308 100%);
}

h2:before {
    top: 0;
}

h2:after {
    bottom: 0;
}
```

こちらのコードは、CSS 見出しデザイン参考 100 選^{*24} からの引用です。

また、CSS のコピペだけ！おしゃれな見出しのデザイン例まとめ 68 選^{*25} にも素敵なデザインが紹介されていますので、ご参考になさってください。

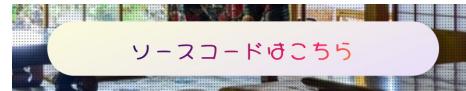
*24 <https://jajaaan.co.jp/css/css-headline/>

*25 <https://saruwakakun.com/html-css/reference/h-design>

2.11 装飾的なボタンを作る

従来はお絵描きソフトを使って作成していたボタンも、CSSで作成できるようになりました。

CSSボタンデザイン120個以上！どこよりも詳しく作り方を解説！^aというサイトにて、数多く紹介されておりますが、その中から、グラデーションを使った綺麗なボタンと、金塊のように輝くボタンをご紹介します。



^a <https://jajaaan.co.jp/css/button/>



▼index.html

```
<a href="https://github.com/Atelier-Mirai/wave-improve" class="gradient button">
  <span>ソースコードはこちら</span>
</a>

<a href="#" class="gold button">
  <span><i class="fas fa-angle-double-up fa-lg fa-fw"></i>Page Top</span>
</a>
```

aタグに、class="gradient button"、class="gold button"とクラス名を付与して、実際の装飾はCSSでいろいろ行っていきます。

▼button.css

```
=====
ボタンの為の装飾指定
参考: CSSボタンデザイン120個以上！どこよりも詳しく作り方を解説！
      https://jajaaan.co.jp/css/button/
=====

/* ボタンの基本形
-----
.button {
  /* 横幅と高さを指定して表示できるようにする */
  display: inline-block;

  /* 書体に関する指定 */
  /* 大きさを 16px ~ 24pxまで可変にする */
  font-size: clamp(16px, 4vw, 24px);
  letter-spacing: 0.1em; /* 文字と文字の間を少し空ける */
  text-decoration: none; /* 下線などの装飾が付かないようにする */

  /* 文字の配置に関する指定 */
  text-align: center; /* 文字は中央に揃える */
  vertical-align: middle; /* 縦方向も中央に揃える */

  /* ボタンの形に関する指定 */
  border-radius: 0.5rem; /* 角を少し丸くする */
  padding: 0.8rem 1.2rem; /* 内側に少し詰め物をして間隔を空ける */

  /* アニメーションに関する指定 */
  transition: all 0.3s; /* 少し時間をあけて変化するようにする */
```

```

}

/* グラデーションボタン
-----
.gradient.button {
    /* 背景色を桜色のグラデーションにする */
    background-image: linear-gradient(-20deg, #e9defa 0%, #f7cbea 100%);
}

/* マウスを重ねたときの指定 */
.gradient.button:hover {
    box-shadow: 0 5px 15px #bc33f5; /* ボタンに紫色の影を付ける */
}

/* 文字の色の指定 */
.gradient.button span {
    background: linear-gradient( /* 虹色にする */
        -225deg,
        #e60012 14%,
        #f39800 28%,
        #fff100 42%,
        #009944 56%,
        #0068B7 70%,
        #1d2088 84%,
        #cfa7cd 100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
}

/* 金塊のようなボタン
-----
.a.gold.button {
    color: #b1921b; /* 文字の色 */
    text-shadow: -1px -1px 1px #fffffd9; /* 文字に影を付ける */

    /* 枠線の太さと色を、上右下左の時計回りの順に指定する */
    border-top: none;
    border-right: 4px solid #cea82c;
    border-bottom: 10px solid #987c1e;
    border-left: 4px solid #ffed8b;

    border-radius: 0; /* 金塊なので角は丸めない */
    background: linear-gradient(-45deg, /* 金塊のようなグラデーション */
        #ffd75b 0%,
        #ffff5a0 30%,
        #ffffabe 40%,
        #fffffdb 50%,
        #ffff5a0 70%,
        #fdd456 100%);
}

/* マウスを重ねたときの指定 */
.a.gold.button:hover {
    /* 上に 3px 空白を入れて、下の枠線を 3px 減らすことで、
       押してへこんだように見せる */
    margin-top: 3px;
    border-bottom: 7px solid #987c1e;
}

```

2.12 トップに戻るボタン

トップに戻るボタンを実装してみましょう。
ボタンの装飾については先ほど扱いましたので、機能面について触れていきます。



▼index.html

```
<head>
  <link rel="stylesheet" href="button.css">
  <link rel="stylesheet" href="page-top.css">
</head>

<body>
  <footer>
    <p id="page-top">
      <a href="#" class="gold button">
        <span><i class="fas fa-angle-double-up fa-lg fa-fw"></i>Page Top</span>
      </a>
    </p>
    &copy; WAVE
  </footer>

  <!-- jQuery -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>
  <!-- page top -->
  <script src="page-top.js"></script>
</body>
```

先に紹介したボタン装飾の為の `button.css` とトップに戻るボタンの為の `page-top.css`、そして `page-top.js` を読み込んでいます。^{*26}

▼page-top.css

```
/* 戻るボタンを右下に固定*/
#page-top {
  position: fixed;
  right: 10px;
  bottom: 10px;
  z-index: 2000;
  /*はじめは非表示*/
  opacity: 0;
  transform: translateY(100px);
}

/* 上に上がる動き */
#page-top.upward {
  animation: upward-animation 0.5s forwards;
}
@keyframes upward-animation {
  from {
```

^{*26} 卷末の参考文献「動くWebデザインアイディア帳」より引用、適宜改変しています。

```

        opacity: 0;
        transform: translateY(100px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

/* 下に下がる動き */
#page-top.downward {
    animation: downward-animation 0.5s forwards;
}
@keyframes downward-animation {
    from {
        opacity: 1;
        transform: translateY(0);
    }
    to {
        opacity: 1;
        transform: translateY(100px);
    }
}

```

`position: fixed;` を使って、戻るボタンを右下に固定しています。

「上に上がる動き」「下に下がる動き」は、**CSS アニメーション**機能を使っています。`animation: upward-animation 0.5s forwards;` と書き、`upward-animation` を 0.5 秒間行っています。`upward-animation` は、`@keyframes upward-animation` で定義されたアニメーションです。`opacity: 0;`(透明) から `opacity: 1;`(不透明) まで変化します。

CSS アニメーションについては、[CSS アニメーションの使用^{*27}](#) に詳しい説明がございます。ぜひご欄になってください。

▼page-top.js

```

// ウィンドウをスクロールした際の処理を、関数定義する。
//-----
const pageTopAnimation = () => {
    // scroll という変数に、ウィンドウのスクロール量を取得して、代入する。
    let scroll = $(window).scrollTop();
    // もしスクロール量が200px以上ならば
    if (scroll >= 200){
        // #page-top に付与した downward というクラス名を除く
        $("#page-top").removeClass("downward");
        // #page-top に upward というクラス名を付与する
        $("#page-top").addClass("upward");

        // そうではなくて、もし #page-topに upward というクラス名が付与されていたら
    } else if ($("#page-top").hasClass("upward")){
        // upward というクラス名を除き
        $("#page-top").removeClass("upward");
        // downward というクラス名を#page-topに付与する
        $("#page-top").addClass("downward");
    }
}

```

^{*27} https://developer.mozilla.org/ja/docs/Web/CSS/CSS_Animations/Using_CSS_animations

```
// 画面をスクロールした際にどの関数を呼ぶか記述する
//-----
$(window).scroll(() => {
    pageTopAnimation();
});

// ページが読み込まれた際にどの関数を呼ぶか記述する
//-----
$(window).on("load", () => {
    pageTopAnimation();
});

// #page-topをクリックした際の設定
//-----
$("#page-top a").on("click", () => {
    $("body, html").animate(
        { scrollTop: 0 }, // ページトップまでスクロール
        500              // ページトップまで500msかけてスクロールする。
    );
    return false;      // リンク自体の無効化。
});
```

「トップに戻る」動作は、最後の `$("#page-top a").on("click", () => {...})` で充分ですが、より魅力的になるよう、jQuery を使い「イベント」と呼ばれる特定の条件下でクラス名を付与するようになっています。そして、特定のクラス名が付与された際の様々な動きは、主に CSS アニメーション機能で行うようになっています。

第3章

様々なフッターの装飾

この章では、様々なフッターの装飾を行っていきます。

- ・文字のグラデーション
- ・可変の文字サイズ
- ・グラデーションで背景画像を彩る
- ・菱形のナビゲーションメニュー
- ・スムーズスクロール

この章では、様々なフッターの装飾を行っていきます。もともとのフッターは次のようにしました。

© WAVE

立派にフッターの役目は果たしていますが、作成者のただ一行のみで淋しい気がいたします。ロゴやナビゲーションメニュー、住所や電話番号なども載せ、彩り豊かに機能強化を図りましょう。



3.1 文字のグラデーション

右のように虹色の文字を作成することもできます。

CSS でテキストを彩る装飾サンプル集^a を参考に実装していきましょう。



^a <https://1-notes.com/css-text-design/>

▼ index.html

```
<span>WAVE</span>
```

▼ index.html

```
<span>WAVE</span>
```

▼ logo.css

```
span {  
    color: transparent;  
    background: repeating-linear-gradient(45deg,  
        #e60012 0.1em 0.2em,  
        #f39800 0.2em 0.3em,  
        #fff100 0.3em 0.4em,  
        #009944 0.4em 0.5em,  
        #0068B7 0.5em 0.6em,  
        #1d2088 0.7em 0.8em,  
        #cfa7cd 0.8em 0.9em);  
    -webkit-background-clip: text; /* Chrome での表示用 */  
    background-clip: text;  
    font-weight: bold;  
    letter-spacing: 10px;  
}
```

簡単な HTML と CSS で実装できます。他にも様々な装飾例が掲載されていますので、ご参考になれば幸いです。

3.2 可変の文字サイズ

端末の画面幅によって、文字の大きさを変更したい。そういった場面もあるかと思います。少し前にはなりますが、「今すぐ使える CSS レシピブック」という書籍には以下のように紹介されております。

♣ メディアクエリを使う方法

▼ CSS

```
p {  
    font-size: 12px;  
}  
  
@media (min-width:320px) {  
    p {  
        font-size: calc((24 - 12) * ((100vw - 320px) / (960 - 320)) + 12px);  
    }  
}
```

```

}

@media (min-width: 960px) {
  p {
    font-size: 24px;
  }
}

```

画面幅 320px で最小値が 12px、画面幅 960px で最大値 24px を取るのは明確です。画面幅 320px ~960px で、徐々に拡大する為に calc 関数を使い、フォントサイズを計算します。calc 関数の引数 $(24 - 12) * ((100vw - 320px) / (960 - 320))$ について説明します。

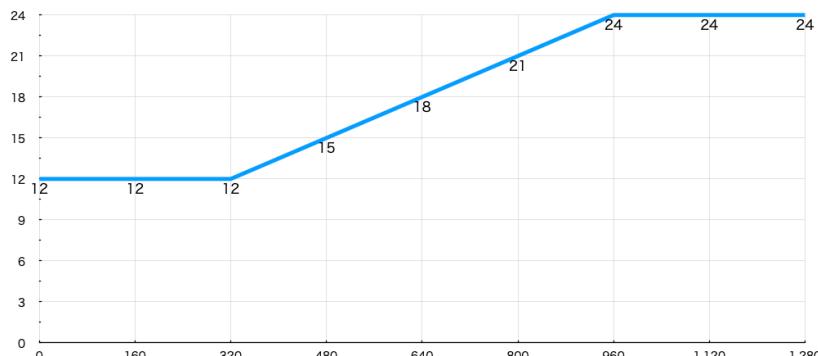
画面幅が 320px から 960px まで 640px 増えた際に、フォントサイズは 12px から 24px まで 12px 増加します。つまり、

画面幅が 160px 増えたならば、フォントサイズ 3px 増加させ、

画面幅が 320px 増えたならば、フォントサイズ 6px 増加させ、

画面幅が 480px 増えたならば、フォントサイズ 9px 増加させるように、比例配分すれば良いのです。

100vw で現在の画面幅が取得できますので、 $(100vw - 320px)$ で画面幅の増分が得られます。これを、 $(960 - 320)$ で割ることで、画面幅の増分が占める 640px への割合が得られますので、 $(24 - 12)$ に掛けることで、フォントサイズを何ピクセル大きくすればよいかが得られます。これに最小値の 12px を足せば、現在の画面幅 100vw で指定すべきフォントサイズが得られます。



♣ clamp 関数を使う方法

メディアクエリを使ってレスポンシブサイズを実現できましたが、少し面倒でした。`clamp()`*¹関数を使うと、より簡便に実現できます。

`clamp()` は CSS の関数で、値を上限と下限の間に制限します。`clamp()` によって、定義された最大値と最小値の間の値を選択することができます。最小値、推奨値、最大値の 3 つの引数を取ります。

使い方は次のように使います。

*¹ <https://developer.mozilla.org/ja/docs/Web/CSS/clamp>

```
p {
  font-size: clamp(最小値, 推奨値, 最大値);
}
```

最小値は 12px、最大値は 24px ですが、推奨値はどのように記述すれば良いでしょうか。

画面幅に応じて変化する単位として vw があります。1vw は画面幅の 1% を表します。

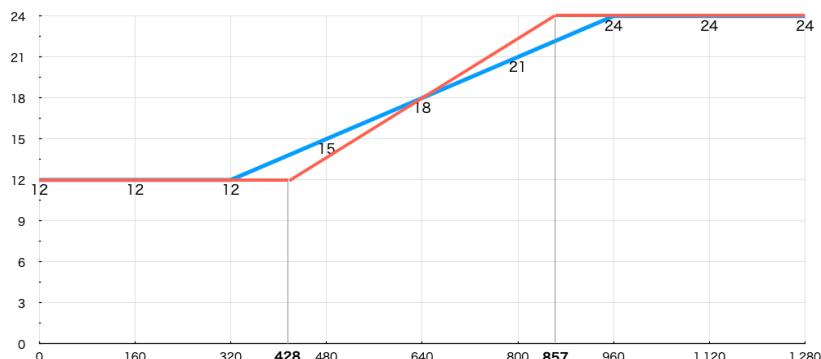
ここでは、画面幅 640px の時に フォントサイズ 18px になるようにしましょう。 $18 / 640 = 2.8\%$ なので次の CSS で実現できそうです。

```
p {
  font-size: clamp(12px, 2.8vw, 24px);
}
```

コードが書けたので、ブラウザで確認してみます。画面幅に応じて `<p>` タグのフォントサイズが変化し、美味く実装できているようですが、少し最小値や最大値の点で違いもあることが分かります。

最小値 12px になるときの画面幅を計算すると、 $12px / 2.8vw * 100 = 428px$ となります。

最大値 24px になるときの画面幅を計算すると、 $24px / 2.8vw * 100 = 857px$ となります。



♣ clamp() Calcurator

clamp 関数を使う方法をご紹介いたしましたが、少し計算が面倒でした。[clamp\(\) Calcurator^{*2}](#) というサイトでは、最小フォントサイズ、最大フォントサイズ、最小ビューポート幅、最大ビューポート幅を入力するだけで、簡単に clamp 関数の値を算出してくれます。

^{*2} <https://chrisburnell.com/clamp-calculator/>

The screenshot shows a web application titled "clamp() Calculator". At the top right, there is a navigation bar with links to "Home", "About", "All Posts", and "Projects". On the left side, there is a small logo of a bird in flight. The main content area has a title "clamp() Calculator" and a subtitle "Handy little tool for calculating viewport-based clamped values." Below this, there are four input fields arranged in a 2x2 grid:

- Minimum Font Size (px): 16
- Maximum Font Size (px): 18
- Minimum Viewport Width (px): 500
- Maximum Viewport Width (px): 1000

Below these fields is a dark blue button labeled "Calculate!". Underneath the button, there is a section titled "The Result" containing a text box with the following CSS code:

```
clamp(1rem, 0.875rem + 0.4vw, 1.125rem)
```

画像は、最小フォントサイズを 16px に、最大フォントサイズを 18px に、最小ビューポート幅を 500px に、最大ビューポート幅を 1000px にする例です。

`clamp(1rem, 0.875rem + 0.4vw, 1.125rem)` と算出されていますので、そのままコピーして使うことが出来ます。

簡単に可変文字サイズを実装できますので、是非使ってみてください。

3.3 グラデーションで背景画像を彩る

美しいグラデーションを CSS で実装^{*3} というサイトがございます。こちらを参考に写真とグラデーションを重ね合わせ、フッターの背景にします。



▲ 図 3.1: 元の背景画像

*3 <https://www.webcreatorbox.com/tech/css-gradient>



▲図 3.2: 淡いグラデーション



▲図 3.3: 重ね合わせた背景

▼ footer.css

```
footer {  
    background: linear-gradient(45deg, #2ca9e128, #ffec4760) fixed,  
              url('sea.webp') 75% 75%;  
    background-size: cover;  
    width: 100vw;  
}
```

簡単に実現できますので、よろしければお使いください。

3.4 菱形のナビゲーションメニュー

それでは、最後に菱形のナビゲーションメニューを創っていきましょう。

まずは、HTML を書いていきます。



▼ index.html

```
<nav>  
  <ul>  
    <li><a href="index.html"><i class="fa-solid fa-house-chimney"></i></a></li>  
    <li><a href="#kiji">    記事一覧<br></a></li>  
    <li><a href="#about">    会社案内<br></a></li>  
    <li><a href="#contact">    お問い合わせ<br></a></li>  
  </ul>  
</nav>
```

いたって普通のナビゲーションメニューです。Font Awesome を使い、`<i class="fa-solid fa-house-chimney"></i>` で家のアイコンを表示させ、ページ内リンクとして `#kiji`、`#about`、`#contact` を設定しています。

▼ footer.css

```
nav ul {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-template-rows: 1fr;
  gap: 25px;

  list-style: none;
  padding: 0;
}
```

CSS は、まず `display: grid;` とし、`` をコンテナ（箱）にし、グリッドアイテムである``を横に四つ並べます。

▼ footer.css

```
nav ul li {
  display: grid;
  grid-template-columns: 64px;
  grid-template-rows: 64px;

  background-color: var(--kurohairo);

  transform: rotate(45deg);
}
```

`` 要素ですが、内包する `<a>` 要素の為のコンテナとなります。列、行ともに 64px のグリッドを設定します。そして背景色を黒羽色にし、45 度回転させます。一辺 64px の正方形の対角線の長さは約 90px ですから、 $90\text{px} \times 4 = 360\text{px}$ の幅を持つナビゲーションメニューができます。

▼ footer.css

```
nav ul li a {
  justify-self: center;
  align-self: center;

  color: var(--sakurairo);
  font-size: 14px;
  text-decoration: none;
  text-align: center;

  transform: rotate(-45deg);
}
```

次に `<a>`要素です。グリッドアイテムとなっていきますので、`<justify-self: center; align-self: center; >` で、グリッドトラックの中央に配置します。文字色や文字サイズなどの調整を行います。そして、このままですると、文字も 45 度傾いたままですので、`transform: rotate(-45deg);` で逆方向に回転させ、文字が水平になるようにします。

▼ footer.css

```
nav ul li:hover {
  background: var(--botaniro);
}
```

最後に `hover` させた際の色を指定して、CSS は完了です。

3.5 スムーズスクロール

記事として、ページ内リンクを設定できます。クリックすると、<h2 id="kiji">記事</h2>に飛びます。

このときに「瞬間移動」するのではなく、滑らかにスクロールさせたいものです。様々な手法がありますが、ここでは簡単な方法として、smooth scroll polyfills を使った方法をご紹介します。

▼ index.html

```
<html>
  <head>
    <!-- Smooth Scroll -->
    <script src="https://cdn.jsdelivr.net/npm/smooth-scroll@16.1.3/dist/smooth-scroll->
    .polyfills.min.js"></script>
  </head>

  <body>
    <!-- (略) -->

    <!-- Smooth Scroll -->
    <script>
      let scroll = new SmoothScroll('a[href*="#"]', { easing: 'easeInOutQuint' });
    </script>
  </body>
</html>
```

<head>内に、CDN から スクリプトを取得するよう記述します。そして、</body> の直前に、一行コードを記述するのみで完了です。

「polyfills」という名称の通り、もともと一部のブラウザでの CSS の挙動を補正するための JavaScript でした。今春より全てのブラウザで CSS のみでスムーズスクロールができるようになっていきます。

scroll-behavior^{*4} として、MDN に 解説もございますので、よろしければご覧になってください。

*4 <https://developer.mozilla.org/ja/docs/Web/CSS/scroll-behavior>

第4章

記事ページの機能拡張

この章では、写真をセピア調にする / 枠を付ける / タブ機能 / 続きを読むボタンについて、実装していきます。

4.1 写真をセピア調にする / 枠を付ける

「思い出」の写真のように、セピア調にして枠を付けてみましょう。

簡単な HTML と CSS だけでも実現できます。

少し前の記事ですが、CSS3】で写真に色々装飾を加えてみた^aが参考になります。

^a https://tanepa.net/css3_01/

The screenshot shows a blog post titled "スケッチが楽しくなるノート" (Sketches that make it fun) dated July 25th. The main image is a sepia-toned photograph of an open notebook. One page contains a bar chart with arrows pointing upwards, and the other page has various hand-drawn sketches and notes. To the right of the main content are two sidebar sections: "PROFILE" featuring a profile picture and a bio, and "PICKUP" showing smaller thumbnail images and their titles.

▼post01.html

```
<figure class="sepia rotate-3 frame">
  
</figure>
```

▼frame.css

```
/* 写真をセピア調にする */
.sepia.frame img {
  filter: sepia(90%);
}
```

```
/* 枠を付ける */
.frame {
    border: solid 8px var(--sakurairo); /* 桜色の枠線を付けます */
    box-shadow: 0 0 0 1px #ccc,           /* 箱に灰色と黒い影を付けます */
                1px 3px 8px 0 #25252555;
    margin-bottom: 0.5em;                 /* 下に少し間隔を取ります */
}

/* 写真を右上に少し回転する */
.rotate-3 {
    transform: rotate(-3deg)
}
```

<figure>は、写真の他、プログラムコードや詩など、独立した図版を示すタグです。class="sepia rotate-3 frame" とクラス名を付与し、CSS で装飾を行っていきます。

色の変更は、filter プロパティで行えます。

filter は CSS のプロパティで、ぼかしや色変化などのグラフィック効果を要素に適用します。フィルターは画像、背景、境界の描画を調整するためによく使われます。CSS 標準に含まれているものは、定義済みの効果を実現するためのいくつかの関数です。

ぼかしや輝度、コントラストなどさまざまな調整を行うことができます。filter^{*1} に丁寧な説明や実例がございますので、ご覧ください。

写真の枠は、border プロパティで行っています。box-shadow プロパティで影もつけて、雰囲気を出しています。

写真を少し傾けることは、transform プロパティで行っています。

transform は CSS のプロパティで、与えられた要素を回転、拡大縮小、傾斜、移動することできます。これは、CSS の視覚整形モデルの座標空間を変更します。

transform^{*2} に丁寧な説明や実例がございますので、ご覧ください。

*1 <https://developer.mozilla.org/ja/docs/Web/CSS/filter>

*2 <https://developer.mozilla.org/ja/docs/Web/CSS/transform>

4.2 タブ機能

タブ機能を使うと、コンパクトに画面内に納めることができます。

タブ機能は、一つは「見出し」部分、もう一つはその「内容」部分と二つの部品から構成されます。

見出しが選択されると、それに対応する内容の部分を表示し、それ以外の内容の部分を非表示にするのが基本的な仕組みです。



緑のアクセント 小物と飾り棚

緑のアクセントならこれ

① 令和三年八月二十五日

サボテン（シャボテン、仙人掌、霸王樹）は、サボテン科に属する植物の総称である。北アメリカと中央アメリカを中心に2000種以上ある。その多くは多肉植物であるため、多肉植物の別名として使われることもあるが、サボテン科以外の多肉植物をサボテンと呼ぶのは誤りである。棘の部分は葉や茎が変化したものであると考えられている。

日本には16世紀後半に南蛮人によって持ち込まれたのが初めとされている。彼らが「ウチワサボテン」の茎の切り口で畳や衣服の汚れをふき取り、樹液をシャボン（石鹼）としてつかっていたため「石鹼のようなもの」という意味で「石鹼体（さぼんてい）」と呼ばれる。

英語でサボテンを表す Cactus（カクタス）は、古代イラン語で棘だらけの植

続きを読む

タブ機能のための HTML は次のようにになります。

```
<!-- タブの見出し部分 -->
<ul class="tab menu">
  <li><a href="#midori" class="active">緑のアクセント</a></li>
  <li><a href="#komono">小物と飾り棚</a></li>
</ul>

<!-- タブの内容部分 -->
<article id="midori" class="tab segment active">
  <h1>緑のアクセントならこれ</h1>
  <!-- (略) -->
</article>

<article id="komono" class="tab segment">
  <h1>小物と飾り棚の組み合わせ</h1>
  <!-- (略) -->
</article>
```

CSS は 次のようになります。見出し部分は active クラスの有無で色を変えるように、内容部分は active クラスの有無で表示・非表示を変えるようにします。

```
/* 見出し部分 */
.tab.menu li a {
  display: block;
  background: var(--natsukazeiro);
}

.tab.menu li.active a {
  background: var(--sakurairo);
}

/* 内容部分 */
.tab.segment {
  display: none;
}

.tab.segment.active {
  display: block;
}
```

そして、``と、見出しがクリックされた際に、このアンカータグから、`#midori`を取得し、`<article id="komono">`に `active` クラスを付与することで、内容部分を表示させる役割を担っているのが、次の JavaScript です。

```
// 任意のタブにURLからリンクするための関数
const showSegmentByHashLink = (locationHashLink) => {
  if (!locationHashLink) { return false; } // 引数が与えられないときは戻る。

  // タブ設定
  $('.tab.menu li').find('a').each(function() { // タブ内のaタグ全てを取得
    let id = $(this).attr('href'); // aタグのhref属性値を取得
    // (=表示させたいタブセグメントのID)

    // もしaタグのhref属性が、リンク元の指定されたURLのハッシュリンクと等しければ、
    if(id === locationHashLink){
      let containingElement = $(this).parent(); // タブ内のaタグの親要素liを取得
      $('.tab.menu li').removeClass("active"); // タブ内のliに付与された
                                                // activeクラスを取り除く
      $(containingElement).addClass("active"); // liにactiveクラスを付与
      $(".tab.segment").removeClass("active"); // タブセグメントのactiveクラスを取り除く
    }
  });
}

// タブをクリックした際に、以下が実行される。
$('.tab.menu a').on('click', function() {
  let id = $(this).attr('href'); // リンクのhref属性を取得
  // (=表示させたいタブセグメントのID)
  showSegmentByHashLink(id); // タブセグメントを表示する
  return false; // aタグをクリックした際の通常動作
                // (リンク先へのジャンプ)を無効にする。
});

// ページ読み込み完了時に、以下が実行される。
$(window).on('load', () => {
  let locationHashLink = location.hash; // URLのフラグメント識別子(ハッシュリンク)
```

```
>)を取得
    showSegmentByHashLink(locationHashLink); // 設定したタブセグメントの読み込み
});
```

4.3 続きを読む機能

コピペができる！ CSS と html のみで作る「続きを読む」の開閉ボタン^{*3} を元に実装します。

```
<figure class="readmore-box">
  <input type="checkbox" id="readmore-01">
  <label for="readmore-01"></label>
  <div class="readmore-container">
    <p>
      サボテン（仙人掌、霸王樹）は、サボテン科に属する植物の総称である。
    </p>
    <!-- (略) -->
  </div>
</article>
```

チェックボックスと、ラベルの関係に着目してください。

何も CSS が適用されていないときには、`<input type="checkbox" id="readmore-01">`、`<label for="readmore-01"></label>` とあるように、チェックボックスが表示されているだけです。（ラベル部分の文字をクリックすることでも、チェックボックスのオンオフを切り替えることが出来ますが、ラベルの文字が空なので押せなくなっています。）

```
.readmore-box {
  position: relative;
}

.readmore-box label {
  position: absolute;
  z-index: 1;
  bottom: 0;
  width: 100%;
  height: 140px; /* グラデーションの高さ */
  cursor: pointer;
  text-align: center;
  /* 以下グラデーションは背景を自身のサイトに合わせて設定してください */
  background: linear-gradient(to bottom, rgba(254, 244, 244, 0) 0%, rgba(254, 244, 244, 0.95) 100%);
}

.readmore-box input:checked + label {
  background: inherit; /* 開いた時にグラデーションを消す */
}

.readmore-box label::after {
  line-height: 2.5rem;
  position: absolute;
  z-index: 2;
```

^{*3} <https://copypet.jp/502/>

```

bottom: 20px;
left: 50%;
width: 16rem;
font-family: "Font Awesome 5 Free";
content: "\f13a 続きを読む";
font-weight: bold;
transform: translate(-50%, 0);
letter-spacing: 0.05em;
border-radius: 20px;

/* https://coco-factory.jp/ugokuweb/wp-content/themes/ugokuweb/data/1-6/1-6.html#Flower より グラデーション ボタン */
border-color: transparent;
color: var(--sakurairo);
background: linear-gradient(270deg, #3bade3 0%, #9844b7 50%, #44ea76 100%);
background-size: 200% auto;
background-position: right center;
box-shadow: 0 5px 10px rgba(250, 108, 159, 0.4);
}

.readmore-box input {
display: none;
}

.readmore-box .readmore-container {
overflow: hidden;
height: 250px; /* 開く前に見えている部分の高さ */
transition: all 0.5s;
}

.readmore-box input:checked + label {
/* display: none ; 閉じるボタンを消す場合解放 */
}

.readmore-box input:checked + label:after {
font-family: "Font Awesome 5 Free";
content: "\f139 閉じる";
font-weight: bold;
}

.readmore-box input:checked ~ .readmore-container {
height: auto;
padding-bottom: 80px; /* 閉じるボタンのbottomからの位置 */
transition: all 0.5s;
}

```

装飾も入っていますので、長い CSS と成っています。

- `.readmore-box label {}` で、ラベルの高さ (=グラデーションの高さ) を設定する。
- `.readmore-box input:checked + label {}` で、チェックされたときにグラデーションを消す。
- `.readmore-box label::after {}` で、疑似要素を使って何もなかったラベルをボタンの形に整形する。

と、少しづつ追って行くと良いです。装飾に関する部分を割愛すると理解しやすいでしょう。

ポイントとなるのは、`input:checked + label` と、隣接セレクタを使った CSS の部分で、これにより JavaScript を用いることなく HTML / CSS のみで 続きを読む機能を実現しています。

第 5 章

サイトについてページの機能拡張

- ・表を互い違いに色を塗る
- ・見出しの均等割付
- ・電話番号、メールアドレスにリンクを設定する
- ・ルビをふる
- ・地図を載せる
- ・紹介動画を掲載する

といった機能拡張を行っていきます。

5.1 表を互い違いに色を塗る

表のそれぞれの行が、別々の色になっていると見易いので、`nth-child` を使って実現します。

▼table.css

```
/* 奇数行と偶数行とで、それぞれで色を変える */
tr:nth-child(odd) th { background: #a1dfbc; color: black; }
tr:nth-child(even) th { background: #00ba13; color: white; }
tr:nth-child(odd) td { background: #bfcd5; color: black; }
tr:nth-child(even) td { background: #4470ec; color: white; }
```

5.2 見出しの均等割付

均等割付を行うには、`text-align-last: justify;` と宣言します (Safari でも、今秋提供されるバージョン 16.0 から対応しています)。

▼table.css

```
th {
  width: 5em;
  text-align: left;          /* 左揃え */
  text-align-last: justify; /* 均等割付 */
}
```

5.3 電話番号、メールアドレスにリンクを設定する

電話番号をクリックすると電話発信を、メールアドレスをクリックするとメールソフトが立ち上がるようになります。リンクを設定できます。

▼about.html

```
<table>
  <tr>
    <th>サイト名</th>
    <td>WAVE</td>
  </tr>
  <tr>
    <th>開設日</th>
    <td>四月一日</td>
  </tr>
  <tr>
    <th>住所</th>
    <td>名古屋市桜区桜町1丁目1番地<br>桜ビル1階</td>
  </tr>
  <tr>
    <th>電話番号</th>
    <td><a href="tel:052-373-4649">052-373-4649</a></td>
  </tr>
  <tr>
    <th>メール<br>アドレス</th>
    <td><a href="mailto:contact@example.com">contact@example.com</a></td>
  </tr>
  <tr>
    <th>運営方針</th>
    <td>ときどきまつり更新中。最新記事の一覧はトップページに掲載しています。</td>
  </tr>
</table>
```

aタグに着目してください。

- で電話番号へのリンクを設定
- でメールアドレスへのリンクを設定します。

以上を行うと、次のようになります。

サイト名	WAVE
開設日	四月一日
住所	名古屋市桜区桜町1丁目1番地 桜ビル1階
電話番号	052-373-4649
メールアドレス	contact@example.com
運営方針	こさじさまったり更新中。最新記事の一覧はトップページに掲載しています。

5.4 ルビを振る

<ruby>タグと<rt>タグを使って、漢字の読みがなを振ることもできます。

綺麗なインテリアや可愛い文房具など、

日常の中で見つけたお気に入りのものを
まとめていのサイトです。

▼about.html

```
<p>
    綺麗なインテリアや可愛い文房具など、<br>
    日常の中で見つけたお気に入りのものを<br>
    <ruby>纏<rt>まと</rt></ruby>めているサイトです。
</p>
```

5.5 地図を掲載する

ウェブサイトに案内地図を載せたい場面があります。Google Map を使い、iPhone では縦長に、iPad では横長に載せることにしましょう。

会社案内

地下鉄桜線 桜駅1番出口より徒歩三分、交通
至便の地にあります。



▲ 図 5.1: 地図 (iPhone)

会社案内

地下鉄桜線 桜駅1番出
口より徒歩三分、交通
至便の地にあります。



▲ 図 5.2: 地図 (iPad)

▼about.html

```
<!-- 地図の取り扱いがしやすいよう コンテナで包みます。 -->
<div class="map-container">
```

```

<div class="note">
  <h2>会社案内</h2>
  <p>地下鉄桜線 桜駅1番出口より徒歩三分、交通至便の地にあります。</p>
</div>

<div class="map">
  <iframe
    src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3264.098229592375!2d>136.9490909522327!3d35.10425798023631!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m>2!1s0x60037b0d5843e2af%3A0x282e3c573e3f00d8!2z44Kz440h440A54-I55CyI0mHjuS4puW6lw!5e0!>3m2!1sja!2sjp!4v1592795030975!5m2!1sja!2sjp"
    style="border:0;"
    loading="lazy">
  </iframe>
</div>
</div>

```

▼ map.css

```

/* モバイル
-----
/* iPhone では、一列 x 二行 で配置します */
.map-container {
  display: grid;
  grid-template-columns: 1fr;
  grid-template-rows: auto auto;
  gap: 10px;
}

/* 紹介文章を一行目に表示します */
.map-container .note {
  grid-column: 1;
  grid-row: 1;
}

/* 地図を二行目に表示します */
.map-container .map {
  grid-column: 1;
  grid-row: 2;
}

/* 地図の縦横比の指定 */
.map-container .map {
  /* iPhone では、縦長(横2に対して縦3の比率)で表示します */
  aspect-ratio: 2 / 3;
}

/* 地図をグリッド内で最大表示します */
.map-container .map iframe {
  width: 100%;
  height: 100%;
}

/* デスクトップ
-----
@media (min-width: 768px) {
  /* iPad や Mac では、二列 x 一行 で配置します */
  .map-container {

```

```
display: grid;
grid-template-columns: 2fr 1fr;
grid-template-rows: auto;
}

/* 紹介文章を二列目に表示します */
.map-container .note {
  grid-column: 2;
  grid-row: 1;
}

/* 地図を一列目に表示します */
.map-container .map {
  grid-column: 1;
  grid-row: 1;
}

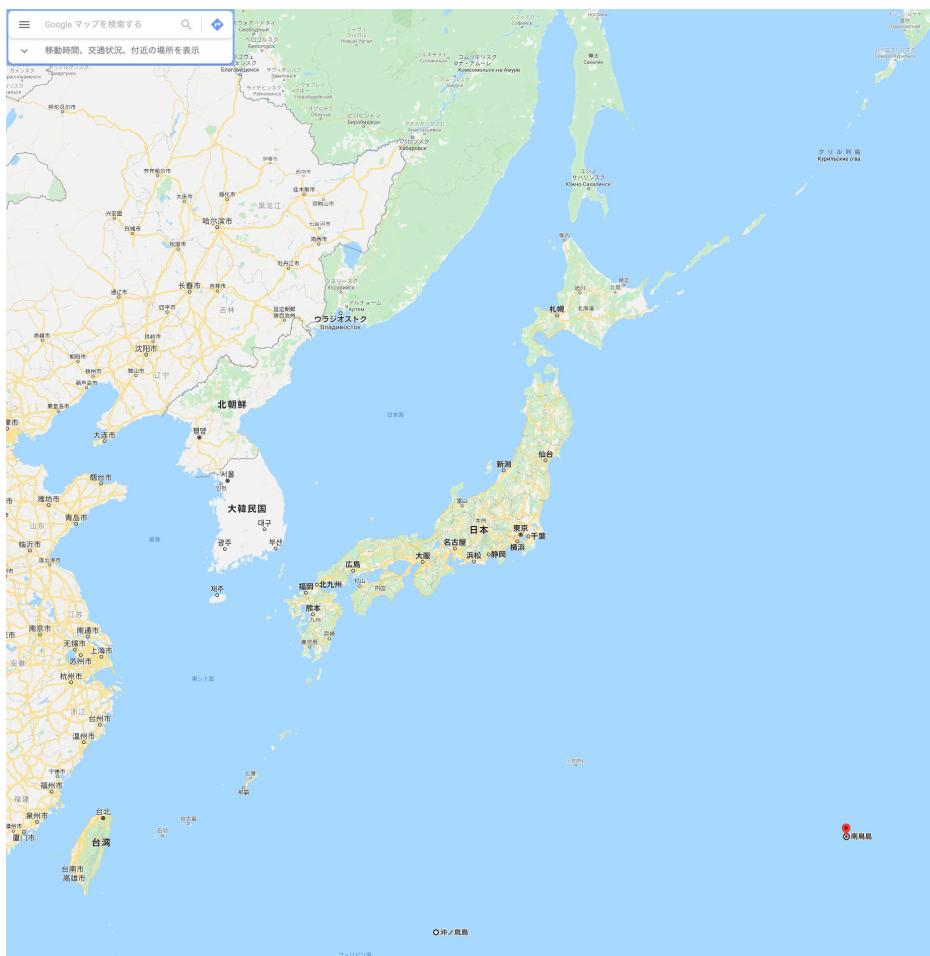
/* 地図の縦横比の指定 */
.map-container .map {
  /* iPad や Mac では、横長(横16に対して縦9の比率)で表示します */
  aspect-ratio: 16 / 9;
}
}
```

HTML コード中、`<iframe src="https://www.google.com/maps/(略)</iframe>`と書かれている部分が、Google Map を埋め込んでいる部分です。`embed?pb=!1m18!1m12!1m3!1d(略)` が、「住所」に相当しています。

CSS コード中、地図の縦横比を決定するために `aspect-ratio` プロパティを使用します（従来は `padding-top` によるハックにより実現していました）。

♣ 地図の埋め込み方法

1. Google Map^{*1} を開きます。



2. 左上の検索窓に、場所名や住所を入力します。

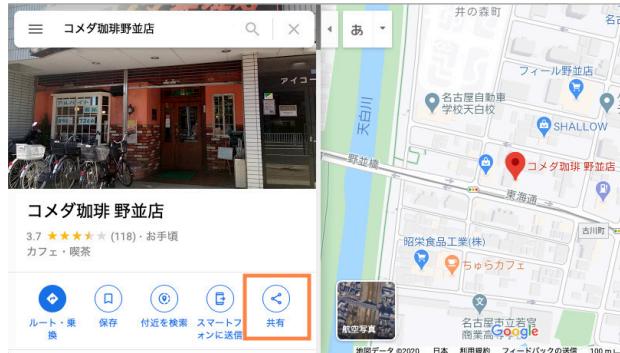


3. 「コメダ珈琲野並店」と入力し、虫眼鏡ボタンを押します。



4. コメダ珈琲 野並店が表示されました。埋め込み用のデータを表示させる為に、右下の「共有」ボタンを押します。

*1 <https://www.google.co.jp/maps>



5. ここでは自分の Web サイトの中に Google Map を埋め込みたいので、「地図を埋め込む」をクリックします。



6. 「HTMLをコピー」をクリックして、エディタに貼り付けます。



7. 不要なコードもあるので、必要な部分だけを残して完成です。

5.6 紹介動画を掲載する

ウェブサイトに動画を掲載したいことも良くあります。写真と同様にサーバーにアップロードして配信することもできますが、動画は容量が嵩みますので、動画共用サイト (YouTube や Vimeo など) にアップロードし、その URL を自分のウェブサイトに埋め込んで使うことがお勧めです。

一日社長 ノンちゃん

「一日社長 ノンちゃん」です。のんびり寛い
でいます。



▲図 5.3: YouTube (iPhone)



▲図 5.4: YouTube (iPad)

▼ about.html

```
<!-- YouTubeの取り扱いがしやすいよう コンテナで包みます。 -->
<div class="video-container">
  <div class="note">
    <h2>一日社長 ノンちゃん</h2>
    <p>「一日社長 ノンちゃん」です。のんびり<ruby>寛<rt>くつ</rt></ruby>いでいます。</p>
  </div>

  <div class="video">
    <iframe
      src="https://www.youtube.com/embed/0Ao1GBrdvzg?rel=0&showinfo=0"
      frameborder="0"
      allowfullscreen>
    </iframe>
  </div>
</div>
```

HTML の構造が、先に掲載した地図と全く同様になっていることにお気付きでしょうか。iPhone では縦長に、iPad では横長にと、表示方法も同様ですので、CSS も同じものを使うことができます。HTML のクラス名のみが異なりますので、.map-container を.video-container に、.map を.video に変更したら完了です。

♣ YouTube 埋め込み方法

1. 埋め込みたい YouTube 動画に移動します。
2. 動画の URL の末尾にある動画 ID をメモします。
<https://www.youtube.com/watch?v=0Ao1GBrdvzg> であれば 動画 ID は 0Ao1GBrdvzg です。
3. 以下の HTML の動画 ID の部分を置き換えて完了です。

▼ html

```
<iframe src="https://www.youtube.com/embed/動画ID"
        frameborder="0"
        allowfullscreen>
</iframe>
```

4. ?rel=0&showinfo=0" と付けるのもお勧めです。

▼ html

```
<iframe src="https://www.youtube.com/embed/動画ID?rel=0&showinfo=0"
        frameborder="0"
        allowfullscreen>
</iframe>
```

- 動画再生した後に関連動画が表示されなくなります。
- 動画再生中の表題が非表示になり、すっきりします

第6章

お問い合わせページの機能拡張

Netlify 公開したウェブサイトのために、* スパム対策* 必須項目設定* 連絡先自動入力* iPhone で自動拡大を防ぐ* 投稿成功時の表示処理* 記入例 (プレースホルダ) の表示などを行っていきます。

6.1

ネットリファイ Netlify

Netlify は無料でサイトを公開できるサービスです。会員登録なしでも利用可能ですが、登録により次のような様々な機能が使えるようになります。

- Git(ギット) と連携したデプロイ (配備) ができる。
- サイトを永久的に公開できる。
- サイト名を好きな名前に変更できる。
- 問い合わせフォームの投稿を、メールで通知できる。

紙幅の都合により登録方法等の解説は省きますが、登録等は容易ですのでご利用下さい。

6.2

お問い合わせフォームの機能拡張

CSS 疑似クラスを活用した、モダンでインタラクティブなフォームの作り方^{*1}や、今どきの入力フォームはこう書く！ HTML コーダーがおさえるべき input タグの書き方まとめ^{*2} にとても良い記事がございます。引用・抜粋しつつ作成いたしました。是非ご一読下さい。

*1 <https://ics.media/entry/200413/>

*2 <https://ics.media/entry/11221/>

お問い合わせ

ご意見、ご感想などがございましたら、以下の欄にご記入の上、
送信してください。記事に関するご質問などもお気軽に寄せく
ださい。

お問い合わせ内容 (必須)

素敵な記事をいつもありがとうございます。
秋の紅葉の記事を読みたいです。

36 文字入力しました。

お名前 (必須)

櫻木 薫

メールアドレス (必須)

sakuragi@example.com

電話番号

09012455678

OKです！

送信

▲図6.1: お問い合わせフォームの作成例

お問い合わせを承りました

担当者よりご連絡差し上げますので、しばらくお待ちください。

▲図6.2: 送信ボタン押下後の表示

♣ HTML

▼ contact.html

```

1 <!-- 問い合わせフォーム -->
2 <form action="./success.html" method="post" name="contact" netlify-honeypot="bot-field">
>d" data-netlify="true">
3   <input type="hidden" name="form-name" value="contact">
4   <label class="hidden" for="subject">件名</label>
5   <input type="hidden" id="subject" name="subject" value="Webサイトから「お問い合わせ」がありました。">
>   <label class="hidden" for="referrer_url">お問い合わせ前に見ていたURL</label>
7   <input type="hidden" id="referrer_url" name="referrer_url" value="">
8
9   <p class="hidden">
10    <label>Don't fill this out if you're human:</label>
11    <input name="bot-field">
12  </p>
13
14  <p>
15    <textarea autocapitalize="none" autocorrect="off" id="message" name="message" placeholder="とっても為になりました。ありがとうございます。" required="required"></textarea>
>area>
17    <label class="name" for="message">お問い合わせ内容</label>
18    <span id="message_size"></span>
19  </p>
20
21  <p>
22    <input type="text" autocomplete="name" class="input" id="name" name="name" placeholder="山田 太郎" required="required">
23    <label class="name" for="name">お名前</label>
24  </p>
25
26  <p>
27    <input type="email" autocomplete="email" class="input" id="email" name="email" placeholder="taro@example.com" required="required">
28    <label class="name" for="email">メールアドレス</label>
29    <span class="error message">taro@example.comの形式で入力してください。</span>
30  </p>
31
32  <p>
33    <input type="tel" autocomplete="tel" class="input" id="tel" maxlength="11" minlength="7" name="tel" pattern="^(\d|-|\\)(\\d){10}$" placeholder="0523734649">
34    <label class="name" for="tel">電話番号</label>
35    <span class="success message">OKです！</span>
36    <span class="error message">09012345678、052-373-4649、0120(000)999の形式で入力してください</span>
37  </p>
38
39  <input type="submit" id="submit" name="" value="送信">
40 </form>
41
42 <!-- contact form validate -->
43 <link rel="stylesheet" href="vendor/contact_form_validate.css">
44 <script src="vendor/contact_form_validate.js"></script>

```

少し大きなHTMLになったので要点を解説します。

▼ contact.html

```
<form action="./success.html" method="post" name="contact" netlify-honeypot="bot-field">
<d> data-netlify="true">
```

`form` 要素に様々な属性が追記されています。

`action="./success.html"` で、フォーム送信後、`success.html` が表示されるようになります。

`method="post"` は、フォーム送信時に、入力したデータが見えないように送る指定です。

`name="contact"` と書くことで、サーバ側でこのフォームのデータを `params[:contact]` のような形で取得することができるようになります。

`netlify-honeypot="bot-field"` は、スパム対策用の記述です。 `bot` と呼ばれる機械プログラムにより投稿フォームが送信されることがあります。その際、`honeypot`(蜜壺) を仕掛けておくことで、機械によるフォームデータの入力と、人によるフォームデータの入力を区別できます。

`data-netlify="true"` と書くことで、通常サーバ側でフォームデータを取り出す処理を行って管理者へのメール通知などを行わなければならないところを、GUI 上で簡単な設定を行うだけで Netlify が通知してくれるようになります。

▼ contact.html

```
<input type="hidden" name="form-name" value="contact">
<label class="hidden" for="subject">件名</label>
<input type="hidden" id="subject" name="subject" value="Webサイトから「お問い合わせ」">
<br>がありました。">
<label class="hidden" for="referrer_url">お問い合わせ前に見ていたURL</label>
<input type="hidden" id="referrer_url" name="referrer_url" value="">
<script>
  document.getElementById("referrer_url").value = document.referrer;
</script>
```

`<input type="hidden">` は、人の目には隠されている (hidden) 入力枠です。プログラムからデータを送信したい際に用います。`label` と `input` は、組にして用います。書き方は二種類あり、一つは `label` タグの中に、`input` タグを書く方法です。

```
<label>
  件名
  <input id="subject">
</label>
```

もう一つの方法は、`input` タグに `id` 属性を付け、それを `label` タグで参照する方法です。

```
<label for="subject">件名</label>
<input id="subject">
```

コーディングやデザイン上の要請から、こちらの方法を用いています。

```
<input type="hidden" id="referrer_url" name="referrer_url" value="">
<script>
  document.getElementById("referrer_url").value = document.referrer;
</script>
```

は、お問い合わせ前に見ていた URL を得るためのコードです。`input` タグの `value` 属性は、`""` と空になっていますが、`JavaScript` コードを書くことで、お問い合わせページの直前に見ていた URL を取得し、`input` タグの `value` 属性を与えてています。

```
<p class="hidden">
  <label>Don't fill this out if you're human:</label>
  <input name="bot-field">
</p>
```

は、`honey-pot`(蜜壺)です。

`class="hidden"`と、`hidden` クラスを付与することにより、人の目には見えないようになります。それにもかかわらず、`<input name="bot-field">`に何か値がセットされていたならば、ボットと呼ばれる機械プログラムからのスパムであると判定できます。

```
<textarea autocapitalize="none" autocorrect="off" id="message" name="message" placeholder="とっても為になりました。ありがとうございます。" required="required"></textarea>
<label class="name" for="message">お問い合わせ内容</label>
<span id="message_size"></span>
```

`autocapitalize="none"`は、自動的に先頭の文字を大文字にすることを無効にするために書きます。英語圏では自動的に先頭の文字が大文字になるのは有用ですが、日本語圏においてはむしろ邪魔になりますので、無効化します。

`autocorrect="off"`は、自動的に英単語のつづりを修正する機能です。これも英語圏では有用ですが、日本語圏においてはむしろ邪魔になりますので、無効化します。

`name="message"`と名前付けを行い、サーバ側でこのフィールドのデータを、`params[:message]`などのように取り出すことができるようになります。

`placeholder="とっても為になりました"`は、入力枠に表示される記入例で、例があると、サイトの閲覧者にとって何を入力すべきか、分かりやすくなります。

`required="required"`は、必須項目を現す属性です。ブーリアン属性と呼ばれ、必須の有無を表します。`required` または `required=""` と書くこともできます。

``は、後ほど JavaScript で文字数を数え、表示するために用意している要素です。

```
<input type="text" autocomplete="name" class="input" id="name" name="name" placeholder="山田 太郎" required="required">
<label class="name" for="name">お名前</label>
```

`autocomplete="name"`と書くことで、iPhone に登録されている連絡先が自動入力されるようになります。

```
<input type="email" autocomplete="email" class="input" id="email" name="email" placeholder="taro@example.com" required="required">
<label class="name" for="email">メールアドレス</label>
<span class="error message">taro@example.comのような形式で入力してください。</span>
```

`type="email"`と書くことで、iPhone で入力する際、@マーク付きのキーボードが表示されるので楽になります。

また、メールアドレスの形式が異なる場合に、エラーメッセージを表示するようにしています。

```
<input type="tel" autocomplete="tel" class="input" id="tel" maxlength="11" minlength="7" name="tel" pattern="^(\d|-|(|\|))*$" placeholder="0523734649">
<label class="name" for="tel">電話番号</label>
```

```
<span class="success message">OKです！</span>
<span class="error message">09012345678、052-373-4649、0120(000)999のような形式で入力
>してください。</span>
```

`type="tel"`と書くことで、iPhoneで入力する際、電話番号のためのキーボードが表示されるので楽になります。`maxlength="11" minlength="7"`と書くことで、電話番号の桁数を最大11桁、最小7桁に指定します。`pattern="^(\d|-|(\(|\)))*$"`は、入力データを「正規表現」と呼ばれる記法で制約しています。数字と-(ハイフン)、()(かっこ)のみを受け付けるように指定しています。

♣ CSS

▼ contact_form_validate.css

```
form .hidden {
  display: none;
}

form {
  max-width: 600px;
  margin: 0 auto;
  padding: 24px;
}

form p {
  display: flex;
  flex-direction: column;
  margin-bottom: 2rem;
  position: relative;
}

form p:focus-within label.name {
  transform: translateY(0) scale(0.8);
}

form label.name {
  display: block;
  order: 1;
  transition: transform 0.2s;
  transform: translateY(32px) scale(1);
  transform-origin: 0 100%;
  padding: 7px;
  font-size: 14px;
  line-height: 1;
}

@media (min-width: 768px) {
  form label.name {
    transform: translateY(34px) scale(1);
  }
}

form input, form textarea {
  order: 2;
  width: 100%;
  border: none;
  font-size: 16px;
  padding: 8px;
```

```
border-bottom: 1px solid #333333;
background: rgba(255, 255, 255, 0.5);
}

form .message.error { color: red; }
form .message.success { color: green; }

form input:invalid,
form textarea:invalid {
  box-shadow: none;
}

form input:invalid~.error.message,
form textarea:invalid~.error.message {
  display: block;
}

form input:invalid~.success.message,
form textarea:invalid~.success.message {
  display: none;
}

form input:valid~.error.message,
form textarea:valid~.error.message {
  display: none;
}

form input:valid~.success.message,
form textarea:valid~.success.message {
  display: block;
}

form input::placeholder,
form textarea::placeholder {
  color: transparent;
}

form input:placeholder-shown~.error.message,
form input:placeholder-shown~.success.message,
form textarea:placeholder-shown~.error.message,
form textarea:placeholder-shown~.success.message {
  display: none;
}

form input:not(:placeholder-shown)~label.name,
form textarea:not(:placeholder-shown)~label.name {
  transform: translateY(0) scale(0.8);
}

form textarea~label.name {
  padding-left: 7px;
}

form .message {
  padding: 0 8px 8px 8px;
  position: absolute;
  bottom: -36px;
  z-index: -1;
```

```
    font-size: 0.9rem;
}

form input[required]~label.name::after,
form textarea[required]~label.name::after {
    content: '(必須)';
    color: red;
    font-size: small;
    margin-left: 5px;
}

form p.required {
    display: block;
    text-align: right;
    margin-bottom: 0;
}

form #submit {
    width: 120px;
    height: 40px;
    border: none;
    background-color: green;
    font-weight: bold;
    font-size: 1rem;
    color: #fff;
    cursor: pointer;
    transition: opacity 0.2s;
}

form #submit:disabled {
    background-color: #999;
    color: #ddd;
    cursor: not-allowed;
}

form #submit:not(:disabled):hover {
    opacity: 0.8;
}

form #message_size {
    font-size: 12px;
    transform: translateY(140px)
}
```

要点を解説していきます。

```
form input[required]~label.name::after,
form textarea[required]~label.name::after {
    content: '(必須)';
    color: red;
    font-size: small;
    margin-left: 5px;
}
```

`input[required]` は、属性セレクタです。 `required` 属性が設定されている要素を選択します。
～は、間接セレクタです。 `required` 属性が設定されている要素の後に続く `label.name` に対して疑似要素`::after`を指定、赤色で(必須)を挿入しています。これをやりたいために、`<input>`タグの

「後」に `<label>` タグを書いていることにも着目してください。

```
form label.name {
  display: block;
  order: 1;
  transition: transform 0.2s;
  transform: translateY(32px) scale(1);
  transform-origin: 0 100%;
  padding: 7px;
  font-size: 14px;
  line-height: 1;
}
```

少し長い CSS ルールですが、ポイントとなるのは `transform: translateY(-72px) scale(1);` です。

```
<input type="text" id="name" required="required">
<label class="name" for="name">お名前</label>
```

と書いたことで、入力枠の「後」に、見出しがきます。この見出しを 72px 上にずらすことで、入力枠の「前」に、見出しがくるようにしています。

```
form .message.error { color: red; }
form .message.success { color: green; }

form input:invalid~.error.message,
form textarea:invalid~.error.message {
  display: block;
}

form input:invalid~.success.message,
form textarea:invalid~.success.message {
  display: none;
}
```

入力に誤りがあった場合、エラーメッセージを赤で、正常であった場合、緑で表示します。着目して欲しいのは `:invalid` 擬似クラス です。フォームの検証（バリデーション）をブラウザが行った際、妥当でない入力の場合には、`:invalid` となりますので、CSS により、赤くエラーメッセージを表示させることができます。

```
form #submit {
  background-color: green;
}

form #submit:disabled {
  background-color: #999;
  cursor: not-allowed;
}
```

同様に「送信ボタン」も、全入力欄が妥当な場合には緑色、そうでない場合には灰色となり送信ボタンを押せないようにしています。

♣ JavaScript

▼ contact_form_validate.js

```
/*
 *-----*
 * ユーザーが何かを入力するたびに、文字数を表示する。
 *-----*/
// 文字数を返す関数
let getCharacterLength = (str) => {
  return [...str].length;
}

// 文字数の表示
const message      = document.getElementById('message');
const message_size = document.getElementById('message_size');
message.addEventListener("input", (event) => {
  let size = getCharacterLength(message.value);
  message_size.innerText = `${size} 文字入力しました。`;
});

/*
 *-----*
 * フォーム全体の妥当性を判定する
 *-----*/
let validate = () => {
  let validForm      = document.querySelector("form:valid");
  let submitButton   = document.getElementById("submit");
  submitButton.disabled = (validForm === null);
};

// (送信ボタンが押せないよう) 初回読み込み時に、validate関数を実行。
validate();

// フォームに入力されたら、validate関数を実行
document.querySelectorAll("input, textarea").forEach((input) => {
  input.addEventListener("input", validate);
});

/*
 *-----*
 * どの記事を見てからの問い合わせか分かるよう、参照元URLを取得する
 *-----*/
document.getElementById("referrer_url").value = document.referrer;

/*
 *-----*
 * フォーム送信前にタブを閉じる際に、確認アラートを表示する。
 *-----*/
let confirmationAlert = (event) => {
  // Cancel the event as stated by the standard.
  event.preventDefault();
  // Chrome requires returnValue to be set.
  event.returnValue = '';
};

// ページ離脱しようとした際に、アラートを表示する。
window.addEventListener('beforeunload', confirmationAlert, false);

// 但し、#submit が押された際には、アラートを表示させない。
document.getElementById('submit').addEventListener('click', () => {
  window.removeEventListener('beforeunload', confirmationAlert, false);
});
```

JavaScript の解説を行っていきます。

文字数を数える

「ユーザーが何かを入力するたびに、文字数を表示する。」機能を設けてあります。

`str.length` では、一部の絵文字等で正しく取得できない場合があるため、JavaScript で正確に文字数を数えるためには、次のような関数により実現します。

お問い合わせ内容 (必須)

こんにちは

5 文字入力しました。

```
let getCharacterLength = (str) => {
  return [...str].length;
}
```

詳細は [String length^{*3}](#) をご覧ください。

文字数が数えられるようになりましたので、イベントリスナを準備し、表示させています。

フォームの妥当性

「フォーム全体の妥当性を判定する」機能の説明です。フォームの必須項目が入力されていないか、形式が誤っている場合には送信ボタンを押せないようにします。

送信

妥当なデータが入力されているなら、送信ボタンを押せるようにします。
サイトの利用者にも分かりやすくなっています。

送信

フォームの妥当性は `document.querySelector("form:valid")`; で取得できます。これにより送信ボタンの有効・無効を切り替えて実現しています。

CSS 疑似クラスを活用した、モダンでインタラクティブなフォームの作り方^{*4} や、クライアント側のフォーム検証^{*5}に詳細が記されておりますので、ご参考になるかと思います。

参照元 URL の取得

`document.referrer` で URL が取得できるので、お問い合わせ前に見ていた URL を`<input type="hidden" id="referrer_url" name="referrer_url" value="">` に渡しています。

*3 https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/String/length

*4 <https://ics.media/entry/200413/>

*5 https://developer.mozilla.org/ja/docs/Learn/Forms/Form_validation

確認アラートの表示

フォーム送信前に他のページへ行くと入力中のデータは失われてしまいます。そうならないよう確認アラートを表示しましょう。



Window: beforeunload イベント^{*6}を使います。

beforeunload イベントは、ウィンドウ、文書、およびそのリソースがアンロードされる直前に発生します。文書はまだ表示されており、この時点ではイベントはキャンセル可能です。このイベントによって、ウェブページがダイアログボックスを表示し、ユーザーにページを終了するかどうかの確認が求めることができます。ユーザーが確認すれば、ブラウザは新しいページへ遷移し、そうでなければ遷移をキャンセルします。

ページ離脱時にアラート表示^{*7}も参考になります。

^{*6} https://developer.mozilla.org/ja/docs/Web/API/Window/beforeunload_event

^{*7} https://qiita.com/naoki_koreeda/items/bf0f512dbd91b450c671

第 7 章

モーダルウィンドウ

文字や写真、動画の表示に使えるモーダルウィンドウの例をご紹介いたします。

7.1 モーダルウィンドウ

モーダルウィンドウを実現するためのライブラリは多数ございますが、その中から、MODAAL^{*1} をご紹介いたします。

Modaal は WCAG 2.0 Level AA アクセシブル・モーダルウィンドウ・プラグインです。

なぜ他のモーダルプラグインなのか？

品質、柔軟性、アクセシビリティの適切な組み合わせのプラグインを見つけるのは困難です。私たちは、さまざまなプロジェクトで使えるものを開発し、アクセシブルなウェブを目指すことができれば、おもしろいと思いました。

DeepL^{*2} による翻訳

Modaal の使い方はとても簡単です。

1. HTML に (通常時には非表示の) コンテンツを用意する。
2. クリックした際の処理を jQuery を書く。

以上でモーダルウィンドウを実現できます。

Modaal のデモサイトは英語ですので、以下の機能について翻訳してご紹介いたします。

- インライン（基本）
- フルスクリーン
- 画像（一枚）
- 画像ギャラリー（複数枚）
- 動画（YouTube / Vimeo）
- iframe

*1 <https://humaan.com/modaal/>

*2 <https://www.deepl.com/ja/translator>

- 確認画面

7.2 基本準備

基本となる以下のHTMLを用意します。

▼index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Modaal 使用例</title>
    <meta name="viewport" content="width=device-width">

    <!-- 簡単に見栄えの良いページを作るためのスタイルシート -->
    <link rel="stylesheet" href="https://unpkg.com/sakura.css/css/sakura.css">
    <!-- Modaal を CDN から読み込む -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/Modaal/0.4.4/>
>css/modaal.min.css">

    <style>
      /* 通常時にモーダルウィンドウ内のコンテンツを表示させないために */
      .hidden { display: none !important; }
      /* スタイルシート「sakura」が干渉するので上書き */
      img { margin-bottom: 0; }
    </style>
  </head>

  <body>
    <h1>モーダルウィンドウ ライブラリ Modaalの使用例</h1>

    <!-- jQuery を CDN から読み込みます -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></
>/script>
    <!-- Modaal を CDN から読み込みます -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Modaal/0.4.4/js/modaal.min.js>
>"></script>
    <!-- Modaal 設定用の JavaScript(jQuery)です -->
    <script src="modaal-custom.js"></script>
  </body>
</html>
```

「簡単に見栄えの良いページを作るためのスタイルシート」として [sakura.css^{*3}](#) を使います。

そして、Modaal のためのスタイルシートを CDN から読み込みます。

<style>タグ内では、二つのルールを宣言しています。一つは、通常時にモーダルウィンドウ内のコンテンツを表示させないためのもので、もう一つは、スタイルシート「sakura」が干渉するので上書きするためのものです。

</body>の直前に、三つの<script>タグを書き、必要なJavaScriptを読み込みます。一つ目はjQueryです。Modaalは、jQueryの利用を前提に書かれていますので、読み込みが必要です。

^{*3} <https://oxal.org/projects/sakura/>

二つ目は Modaal 本体です。CDN から読み込みます。

三つ目は `modaal-custom.js` としていますが、名前は任意です。Modaal 設定用の JavaScript(jQuery) で、特定の要素をクリックした際に、どのようなモーダルウィンドウを実現したいのか、簡単なコードを書くためのファイルです。

7.3 インライン

♣ 基本的な使い方

それでは、実際に使い始めていきましょう。以下のコードを追加してください。

```
<h2>インライン（基本）</h2>
<p>
  ページ内の既存の要素（IDを使用）からコンテンツを取得し、
  コンテンツに読み込みます。
</p>
<a href="#inline" class="inline">Inline</a>
<div id="inline" class="hidden">
  ここに書かれたINLINEコンテンツが表示されます
</div>
```

ブラウザで見ると右のようになります。`class="hidden"`により非表示となっていることが確認できます。

モーダルウィンドウ ライブライ Modaalの使用例

インライン（基本）

ページ内の既存の要素（IDを使用）からコンテンツを取得し、コンテンツに読み込みます。

Inline

`Inline` をクリックしたときに、コンテンツが表示されるようにするために、`modaal-custom.js`に次のコードを記述しましょう。

▼ `modaal-custom.js`

```
// inline (基本)
$(".inline").modaal();
```

`Inline` をクリックすると右のようになります。

右上の「x」ボタンをクリックすると、モーダルウィンドウを閉じることができます。

モーダルウィンドウ ライブライ Modaalの使用例

ここに書かれたINLINEコンテンツが表示されます

ページ内の既存の要素（IDを使用）からコンテンツを取得し、コンテンツに読み込みます。

Inline

♣ 複数のモーダルウィンドウを用意したい場合

複数のモーダルウィンドウを用意したい場合には次のコードを書きます。

```
<a href="#inline1" class="inline">Inline1</a>
<div id="inline1" class="hidden">インラインコンテンツ その 1</div>

<a href="#inline2" class="inline">Inline2</a>
<div id="inline2" class="hidden">インラインコンテンツ その 2</div>

<a href="#inline3" class="inline">Inline3</a>
<div id="inline3" class="hidden">インラインコンテンツ その 3</div>
```

に対応する <div id="inline1">が開くようになります。

♣ まとめ

1. HTML コードを書く。

```
<a href="#inline" class="inline">Inline</a>
<div id="inline" class="hidden">コンテンツ</div>
```

2. コンテンツが最初から見えてしまわぬよう、`class="hidden"`として非表示にする。
3. `class="inline"`要素をクリックした際に、インライン表示のモーダルウィンドウとなるよう、簡単なコードを書く。
4. に対応する <div id="inline">が開くようになる。

7.4 フルスクリーン

フルスクリーンで開くモーダルウィンドウも作成できます。そのために次の HTML を用意します。

```
<h2>フルスクリーン</h2>
<a href="#fullscreen" class="fullscreen">Fullscreen</a>
<p>
  フルスクリーンモードでは、
  Modaalウィンドウがビューポート全体に広がるように開きます。
  コンテンツがウィンドウの高さを超える場合は、
  ダイアログが垂直方向にスクロールし、
  すべてのコンテンツにアクセスできるようになります。
</p>
<div id="fullscreen" class="hidden">
  ここに書かれたINLINEコンテンツがフルスクリーンで表示されます
</div>
```

ブラウザで見ると右のようになります。

フルスクリーン

Fullscreen
フルスクリーンモードでは、Modaalウィンドウがビューポート全体に広がるように開きます。コンテンツがウィンドウの高さを超える場合は、ダイアログが垂直方向にスクロールし、すべてのコンテンツにアクセスできるようになります。

Fullscreen をクリックしたときに、コンテンツがフルスクリーン表示されるようにするために、modaal-custom.js に次のコードを記述しましょう。

▼ modaal-custom.js

```
// フルスクリーン
$(".fullscreen").modaal({
  fullscreen: true
});
```

Fullscreen をクリックすると右のようになります。

ここに書かれたINLINEコンテンツがフルスクリーンで表示されます



7.5 画像

画像が開くモーダルウィンドウも作成できます。そのためには、次の HTML を用意します。

```
<h2>画像(一枚)</h2>
<p>
  一枚の画像を開きます。
  開いた画像の下にラベルを表示したり、
  data-modaal-desc="My Image Description "を使って
  アクセス可能なラベルを表示することもできます。
</p>
```

```
<a href="images/sea01.webp" class="image" data-modaal-desc="空飛ぶ鷗">
  
</a>
```

と、画像ファイル名を指定しています。「サムネイル」（親指の爪）画像と呼ばれるもので、その名の通り、小さな画像ファイルを作って、最初はそれを表示させています。そして、サムネイル画像がクリックされた際には、 で指定した大きな画像を表示させます。

ブラウザで見ると右のようになります。

画像(一枚)

1枚の画像を開きます。開いた画像の下にラベルを表示したり、data-modaal-desc="My Image Description "を使ってアクセス可能なラベルを表示することもできます。



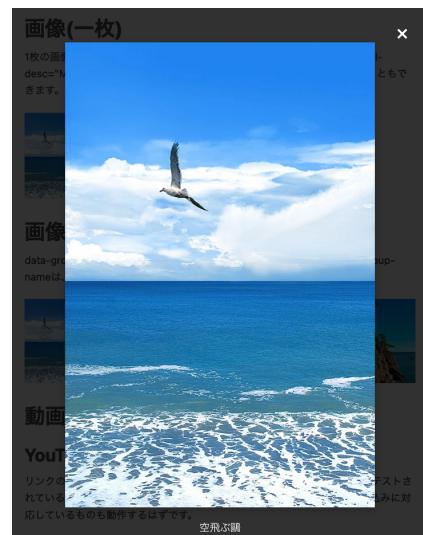
「空飛ぶ鷗」の画像をクリックしたときに、「空飛ぶ鷗」の画像が表示されるようにするために、modaal-custom.js に次のコードを記述しましょう。

▼ modal-custom.js

```
// 画像(一枚)
$('.image').modaal({
  type: 'image'
});
```

「空飛ぶ鷗」の画像をクリックすると右のようになります。

data-modaal-desc="空飛ぶ鷗" と記述したことで、写真の下に補足説明も掲載されます。



7.6 画像ギャラリー(複数枚)

関連性のある複数枚の画像を開くためのモーダルウィンドウも作成できます。そのためには、次のHTMLを用意します。

```
<h2>画像ギャラリー(複数枚)</h2>
<p>
  data-group="group-name" 属性でリンクされた一連の画像を開きます。
  group-nameは、あなたのギャラリーグループの識別子に置き換えてください。
</p>
<div>
  <a href="images/sea01.webp" class="gallery" data-group="sea"
      data-modaal-desc="空飛ぶ鷗">
    
  </a>
  <a href="images/sea02.webp" class="gallery" data-group="sea"
      data-modaal-desc="打ち寄せる波">
    
  </a>
  <a href="images/sea03.webp" class="gallery" data-group="sea"
      data-modaal-desc="透明な南の海">
    
  </a>
  <a href="images/sea04.webp" class="gallery" data-group="sea"
      data-modaal-desc="夜明け前">
    
  </a>
  <a href="images/sea05.webp" class="gallery" data-group="sea"
      data-modaal-desc="断崖にて">
    
  </a>
</div>
```

画像一枚で表示する例とほぼ同じですが、それぞれの画像リンクに共通して data-group="sea"と書かれていることに着目してください。グループ名が sea で共通しているので、海に関する一塊の写真ギャラリーとして、Modaal は扱うようになります。

ブラウザで見ると右のようになります。

画像ギャラリー(複数枚)

data-group="group-name" 属性でリンクされた一連の画像を開きます。group-nameは、あなたのギャラリーグループの識別子に置き換えてください。



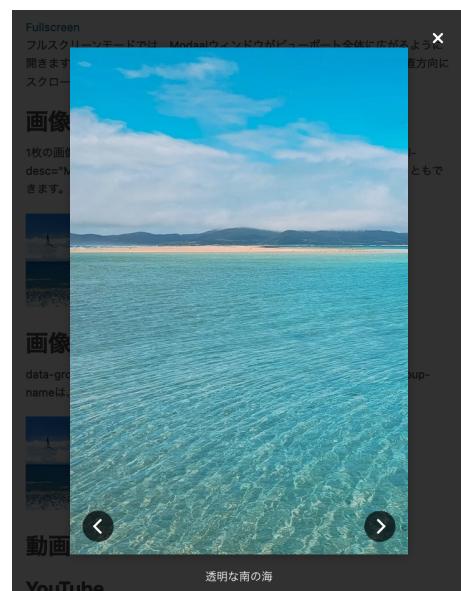
写真ギャラリーを実現するためには、modaal-custom.js に次のコードを記述しましょう。

▼ modaal-custom.js

```
// 画像ギャラリー(複数枚)
$('.gallery').modaal({
  type: 'image'
});
```

「空飛ぶ鷗」の画像をクリックするとモーダルウィンドウで写真ギャラリーが表示されます。写真下の「>」をクリックすることで次の写真へと移動できます。

真ん中の「透明な南の海」を見ている状態です。



7.7 動画

YouTube や Vimeo など動画が開くモーダルウィンドウも作成できます。

♣ YouTube

```
<h3>YouTube</h3>
<p>
  リンクのhref属性で指定された埋め込み動画をiframeに読み込みます。
  現在テストされているフォーマットは、YoutubeとVimeoです。
  その他、iframeでの埋め込みに対応しているものも動作するはずです。
</p>
<p>
  Modalのビデオタイプは、VimeoとYoutubeの両方で徹底的にテストされています。
  最良の結果を得るためにには、URLフォーマットが以下のようになっていることを
  確認してください。私たちはこのURLをiframeに移植し、
  そこから各サービスプロバイダーが必要な再生をすべてコントロールします。
</p>
<p>
  https://www.youtube.com/embed/cBJyo0tgLnw
  最後のIDはあなたのユニークなビデオIDです。
  これは、youtubeの動画で「共有」を選択し、
  「埋め込み」をクリックすると表示されます。
  提示されたコンテンツの中にこのURLがあります。
</p>

<a href="https://www.youtube.com/embed/cBJyo0tgLnw" class="youtube">
  
</a>
```

<p>タグ内に説明が書かれていますので、ご自身の動画を埋め込む場合には、適宜、動画 ID を変更して実行してください。

ブラウザで見ると右のようになります。

YouTube

リンクのhref属性で指定された埋め込み動画をiframeに読み込みます。現在テストされているフォーマットは、YoutubeとVimeoです。その他、iframeでの埋め込みに対応しているものも動作するはずです。

Modaalのビデオタイプは、VimeoとYoutubeの両方で徹底的にテストされています。最良の結果を得るためにには、URLフォーマットが以下のようになっていることを確認してください。私たちはこのURLをiframeに移植し、そこから各サービスプロバイダーが必要な再生をすべてコントロールします。

<https://www.youtube.com/embed/cBjyo0tgLnw>、最後のIDはあなたのユニークなビデオIDです。これは、youtubeの動画で「共有」を選択し、「埋め込み」をクリックすると表示されます。提示されたコンテンツの中にこのURLがあります。



「YouTube」の画像をクリックしたときに、動画が再生されるようにするために、modaal-custom.jsに次のコードを記述しましょう。

▼ modaal-custom.js

```
// 動画(YouTube)
$('.youtube').modaal({
  type: 'video'
});
```

「YouTube」の画像をクリックすると右のようになります。



♣ Vimeo

```
<h3>Vimeo</h3>
<p>
  https://player.vimeo.com/video/109626219
  最後のIDはお客様固有のビデオIDです。
  これは、vimeoの動画で「共有」を選択し（一般的に右側に表示されます）、
  「埋め込み」内のコンテンツを選択することで確認できます。
  埋め込みコードの一一番最初にあるsrc=""の中に必要なURLがあります。
</p>
<a href="https://player.vimeo.com/video/109626219" class="vimeo">
  
</a>
```

ブラウザで見ると右のようになります。

Vimeo

https://player.vimeo.com/video/109626219、最後のIDはお客様固有のビデオIDです。これは、vimeoの動画で「共有」を選択し（一般的に右側に表示されます）、「埋め込み」内のコンテンツを選択することで確認できます。埋め込みコードの一一番最初にあるsrc=""の中に必要なURLがあります。



「Vimeo」の画像をクリックしたときに、動画が再生されるようにするために、modaal-custom.jsに次のコードを記述しましょう。

▼ modaal-custom.js

```
// 動画(Vimeo)
$('.vimeo').modaal({
  type: 'video'
});
```

「Vimeo」の画像をクリックすると右のようになります。



7.8

iframeで他のウェブサイトの内容を開く

iframeを使うと、モーダルウィンドウで他のウェブサイトの内容を開くことができます。まずは次のHTMLを用意します。

```
<h2>iframeで他のウェブサイトの内容を開く</h2>
<p>
  リンクのhref属性で定義されたURLを、iframeに読み込みます。
  このためには、モーダルの幅と高さも設定する必要があります。
</p>
<a href="http://humaan.com" class="iframe">iframe</a>
```

ブラウザで見ると右のようになります。

iframeで、他のウェブサイトの内容を開く

リンクのhref属性で定義されたURLを、iframeに読み込みます。このためには、モーダルの幅と高さも設定する必要があります。

iframe

iframeをクリックしたときに、リンク先のサイト「<http://humaan.com>」をモーダルウィンドウ内で表示されるためには、modaal-custom.jsに次のコードを記述しましょう。

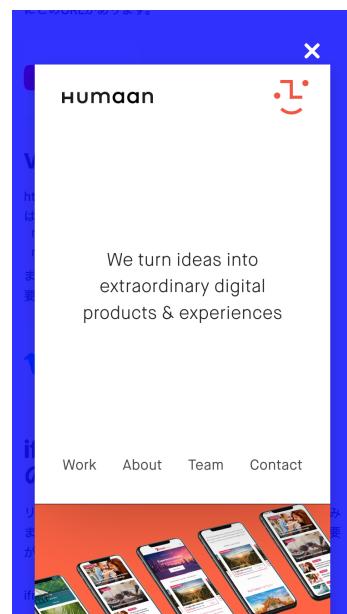
▼ modaal-custom.js

```
// iframe
let w = $(window).width(); // 画面幅を取得
let h = $(window).height(); // 画面高を取得
let bc = '';
if (w >= 768) { // iPad以上の画面幅の場合
  w = w * 0.8; // w *= 0.8; と省略して書くことも出来ます
  h = h * 0.8;
  bc = '#ff0000'; // 背景色を赤色にする
} else { // iPhoneで閲覧の場合
  bc = '#0000ff'; // 背景色を青色にする
}

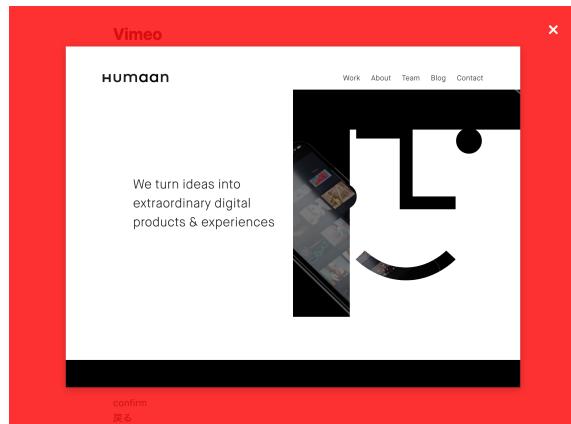
$('.iframe').modaal({
  type: 'iframe',
  width: w,
  height: h,
  background: bc
});
```

今までより少しコードが長くなりました。`<p>`タグ内の説明として「モーダルの幅と高さも設定する必要があります」と書かれているように、画面幅を取得し、iPhoneかiPadかによって、モーダルウィンドウの大きさと背景色を変更するようにしています。

iframe をクリックした例 (iPhone)



iframe をクリックした例 (iPad)



7.9 確認画面

最後の例は、モーダルウィンドウで開く確認画面です。準備として次の HTML を用意します。

```
<h2>確認画面</h2>
<p>
  ユーザーに特定のアクションの「確認」または「取消」を促すモーダルなウィンドウ。
  必要に応じて、コールバックイベントを含むコンテンツをプッシュすることができます。
  デフォルトでは、一度開くと、
  アクション（確認／取消など）が選択されるまで閉じることができません。
</p>
<a href="#" class="confirm">confirm</a>
```

また、`sakura.css` の干渉補正のために、以下も追加します。

```
<style>
/* sakura.css の干渉補正 */
.modaal-confirm-btn.modaal-cancel { color: #555; }
</style>
```

プラウザで見ると右のようになります。

確認画面

ユーザーに特定のアクションの「確認」または「取消」を促すモーダルなウィンドウ。必要に応じて、コールバックイベントを含むコンテンツをプッシュすることができます。デフォルトでは、一度開くと、アクション（確認／取消など）が選択されるまで閉じることができません。

confirm

`confirm` をクリックしたときに、確認画面がモーダル表示されるようにするためにには、`modaal-custom.js` に次のコードを記述しましょう。

▼mодаal-custom.js

```
// 確認画面
$('.confirm').mодаал({
  type: 'confirm',
  confirm_button_text: '確認',
  confirm_cancel_button_text: '取消',
  confirm_title: '確認画面です',
  confirm_content: '<p>登録してもよろしいですか</p>',
  confirm_callback: () => {
    alert('登録しました');
  },
  confirm_cancel_callback: () => {
    alert('取り消しました');
  }
});
```

`confirm` をクリックすると右のようになります。



「確認」をクリックすると「登録しました」とメッセージが表示され、一見動作しているように思いますが、`alert('登録しました');` により表示させているのみです。実用に供するには、実際の登録処理をコーディングして下さい。

第8章

文字の演出効果

8.1 シャッフルテキスト

手軽にテキストシャッフル演出ができる JavaScript ライブラリ「shuffle-text」を公開^{*1} という記事がございます。

JavaScript ライブラリ「shuffle-text」を公開しました。shuffle-text はテキストシャッフル（文字列がランダムで切り替わる演出）の表現を行うためのライブラリで、SPA（シングル・ページ・アプリケーション）やゲームの演出やスペシャルコンテンツなどの演出に役立ちます。

元記事は、NPM を利用して使う方法が説明されておりましたが、<https://github.com/ics-iked/a/shuffle-text> で公開されている examples から、HTML と JavaScript を使った基本的な使い方をご紹介いたします。

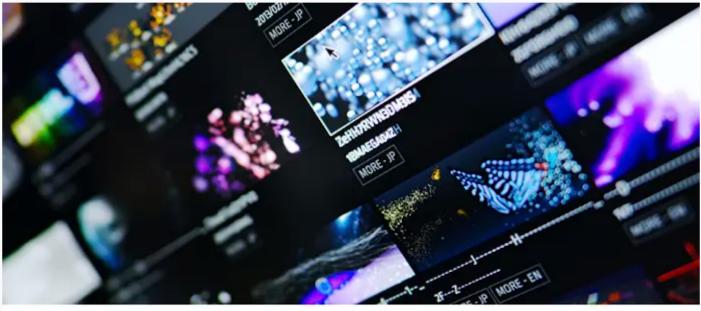
マウスカーソルをのせると、文字がシャッフルされる演出がなされます。動作例^{*2}とソースコード^{*3}です。ご参考になれば幸いです。

*1 <https://ics.media/entry/15498/>

*2 https://wave-improve.netlify.app/shuffle_text/index.html

*3 https://github.com/Atelier-Mirai/wave-improve/tree/master/shuffle_text

shuffle-text



手軽にテキストシャッフル演出ができる
JavaScriptライブラリ「shuffle-text」を公開

shuffle-textはテキストシャッフル（文字列がランダムで切り替わる演出）の表現を行うためのライブラリです。ウェブサイトでマウスが触れたときに演出ができます。

使い方

シャッフルしたい要素に class="shuffle" とクラス名を付与します。

- 01/01 「トップページ」更新しました。
- 01/02 「会社案内」更新しました。
- 01/03 「お問い合わせ」更新しました。
- 01/04 「最新情報」更新しました。

再読み込み
shuffle-text.js is under MIT Licence.

© copyright 2012-2021,clockmaker.jp

▲図 8.1: 作成例

まずは、次のように index.html を作成いたします。

▼index.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>shuffle-text</title>
6   <meta name="viewport" content="width=device-width">
7
8   <!-- 簡単に見栄えの良いページを作るためのスタイルシート -->
9   <link rel="stylesheet" href="https://unpkg.com/sakura.css/css/sakura.css">
10 </head>
11
12 <body>
```

```

13 <h1><a href="https://ics.media/entry/15498/">shuffle-text</a></h1>
14 
15 <p>
16   shuffle-textはテキストシャッフル（文字列がランダムで切り替わる演出）の
17   表現を行うためのライブラリです。
18   ウェブサイトでマウスが触れたときに演出ができます。
19 </p>
20
21 <h3>使い方</h3>
22 <p>シャッフルしたい要素に class="shuffle" とクラス名を付与します。</p>
23
24 <ul>
25   <li class="shuffle">01/01 「トップページ」更新しました。</li>
26   <li class="shuffle">01/02 「会社案内」更新しました。</li>
27   <li class="shuffle">01/03 「お問い合わせ」更新しました。</li>
28   <li class="shuffle">01/04 「最新情報」更新しました。</li>
29 </ul>
30
31 <!-- 再読み込みボタン -->
32 <button id="reload">再読み込み</button>
33 <script>
34   document.getElementById("reload").addEventListener("click", () => {
35     location.reload();
36   });
37 </script>
38
39 <footer>
40   <p>shuffle-text.js is under MIT Licence.</p>
41   <p>&copy; copyright 2012-2021,<a href="http://clockmaker.jp/">clockmaker.jp</a></p>
42 </footer>
43
44 <!-- JavaScript を読み込む -->
45 <script src="shuffle-text.js"></script>
46 <script src="shuffle-text-custom.js"></script>
47 </body>
48 </html>

```

<li class="shuffle">01/01 「トップページ」更新と、シャッフルしたい要素に **class="shuffle"** とクラス名を付与する点がポイントです。

</body> 直前に、シャッフルテキスト本体である **shuffle-text.js** と、簡単に効果を適用できるための **shuffle-text-custom.js** を呼び出すようコーディングしたら完成です。

iPhone / iPad で何度も効果を楽しめるよう、際読み込みボタンも用意しています。

▼ index.html

```

31 <!-- 再読み込みボタン -->
32 <button id="reload">再読み込み</button>
33 <script>
34   document.getElementById("reload").addEventListener("click", () => {
35     location.reload();
36   });
37 </script>

```

.shuffle クラスに、簡単に効果を適用するための `shuffle-text-custom.js` です。ご興味ある方のために、コメントもつけておりますので、お読みください。

▼ `shuffle-text-custom.js`

```

1 // ページ読み込み時に実行されるよう、イベントリスナを指定。
2 window.addEventListener('load', init);
3
4 // 初期化処理を行う関数
5 function init() {
6   // ShuffleTextのインスタンス達を格納する配列
7   let effectList = [];
8   // shuffle クラスが付与された全ての要素を取得する。
9   // (<li class="shuffle">)と書かれた要素全て)
10  let elementList = document.querySelectorAll('.shuffle');
11
12  // elementListの全てのメンバーに対して、繰り返し処理を行う。
13  for (let i = 0; i < elementList.length; i++) {
14
15    // i番目のメンバーを取得して、elementという変数に代入。
16    let element = elementList[i];
17    // カスタムdata属性を付与する。
18    // <li class="shuffle">01/01「トップページ」更新</li>
19    // という元の要素を、
20    // <li class="shuffle" data-index="1">01/01「トップページ」更新</li>
21    // に変更する。カスタムdata属性 data-index="1" が付与されている。
22    element.dataset.index = i;
23    // ShuffleTextクラスのインスタンスを作成し、effectListに格納する。
24    effectList[i] = new ShuffleText(element);
25
26    // マウスを載せたときに再生するよう、イベントリスナを指定する。
27    element.addEventListener('mouseenter', function () {
28      // <li>要素は、次のようになっている。
29      // <li class="shuffle" data-index="1">01/01「トップページ」更新</li>
30      // this.dataset.index と書くと data-index="1" の 1 を取得できる。
31      effectList[this.dataset.index].start();
32      // effectList[1].start(); と等価だが、
33      // シャッフル効果を付けたい要素が複数ある場合に、
34      // this.dataset.index で それぞれの要素を指定できる。
35    });
36
37    // マウスを離した時に再生するよう、イベントリスナを指定する。
38    element.addEventListener('mouseleave', function () {
39      effectList[this.dataset.index].start();
40    });
41
42    // ページを読み込んだときに、初回を再生する。
43    effectList[i].start();
44  }
45}
```

シャッフルテキスト本体です。

シャッフル効果が日本語になるよう ポラーノの広場 (宮沢賢治) に変更した他は、ほぼ原作者 池田泰延さんのコードのままです。

▼shuffle-text.js

```

1 (function (global, factory) {
2   typeof exports === 'object' && typeof module !== 'undefined' ? module.exports = factory()
3   : typeof define === 'function' && define.amd ? define(factory) :
4   : globalThis !== 'undefined' ? globalThis : global || self, global.ShuffleText = factory();
5 });
6
7 }(this, (function () {
8   'use strict';
9
10  /**
11   * ShuffleTextはDOMエレメント用ランダムテキストクラスです。
12   * @author Yasunobu Ikeda
13   * @since 2012-02-07
14   */
15
16  let ShuffleText = (function () {
17    // DOMエレメントです。
18    function ShuffleText(element) {
19      let _a;
20      // シャッフル効果が日本語になるよう ポラーノの広場(宮沢賢治)に変更
21      this.sourceRandomCharacter = "あのイーハトーヴォのすきとおった風、夏でも底に冷々たさをもつ青いそら、うつくしい森で飾られたモリーオ市、郊外のぎらぎらひかる草の波";
22      // this.sourceRandomCharacter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890";
23      this.emptyCharacter        = "-"; // 空白に用いる文字列です
24      this.duration              = 600; // エフェクトの実行時間（ミリ秒）
25      this._isRunning            = false;
26      this._originalStr          = "";
27      this._originalLength       = 0;
28      this._timeCurrent          = 0;
29      this._timeStart             = 0;
30      this._randomIndex          = [];
31      this._element               = null;
32      this._requestAnimationFrameId = 0;
33      this._element               = element;
34      this.setText(_a = element.textContent) !== null && _a !== void 0 ? _a : "";
35    }
36
37    // シャッフル対象となるテキストを設定します。
38    ShuffleText.prototype.setText = function (text) {
39      this._originalStr  = text;
40      this._originalLength = text.length;
41    };
42
43    Object.defineProperty(ShuffleText.prototype, "isRunning", {
44      get: function () {
45        return this._isRunning; // 再生中かどうかを示すブール値です。
46      },
47      enumerable: false,
48      configurable: true
49    });
50
51    // Play effect. 再生を開始します。
52    ShuffleText.prototype.start = function () {
53      let _this = this;
54      this.stop();
55      this._randomIndex = [];
56      let str = "";
57      for (let i = 0; i < this._originalLength; i++) {
58        let rate = i / this._originalLength;
59        this._randomIndex[i] = Math.random() * (1 - rate) + rate;
60        str += this.emptyCharacter;
61      }
62      this._element.textContent = str;
63      this._start();
64    };
65
66    /**
67     * エフェクトを停止します。
68     */
69    ShuffleText.prototype.stop = function () {
70      this._cancel();
71    };
72
73    /**
74     * エフェクトを強制的に終了します。
75     */
76    ShuffleText.prototype.cancel = function () {
77      this._cancel();
78    };
79
80    /**
81     * エフェクトを強制的に再開します。
82     */
83    ShuffleText.prototype.resume = function () {
84      this._resume();
85    };
86
87    /**
88     * エフェクトを強制的に再開します。
89     */
90    ShuffleText.prototype.pause = function () {
91      this._pause();
92    };
93
94    /**
95     * エフェクトを強制的に再開します。
96     */
97    ShuffleText.prototype.resume = function () {
98      this._resume();
99    };
100
101   /**
102    * エフェクトを強制的に再開します。
103    */
104    ShuffleText.prototype.pause = function () {
105      this._pause();
106    };
107  });
108
109  /**
110   * エフェクトを強制的に再開します。
111   */
112  ShuffleText.prototype.resume = function () {
113    this._resume();
114  };
115
116  /**
117   * エフェクトを強制的に再開します。
118   */
119  ShuffleText.prototype.pause = function () {
120    this._pause();
121  };
122
123  /**
124   * エフェクトを強制的に再開します。
125   */
126  ShuffleText.prototype.resume = function () {
127    this._resume();
128  };
129
130  /**
131   * エフェクトを強制的に再開します。
132   */
133  ShuffleText.prototype.pause = function () {
134    this._pause();
135  };
136
137  /**
138   * エフェクトを強制的に再開します。
139   */
140  ShuffleText.prototype.resume = function () {
141    this._resume();
142  };
143
144  /**
145   * エフェクトを強制的に再開します。
146   */
147  ShuffleText.prototype.pause = function () {
148    this._pause();
149  };
150
151  /**
152   * エフェクトを強制的に再開します。
153   */
154  ShuffleText.prototype.resume = function () {
155    this._resume();
156  };
157
158  /**
159   * エフェクトを強制的に再開します。
160   */
161  ShuffleText.prototype.pause = function () {
162    this._pause();
163  };
164
165  /**
166   * エフェクトを強制的に再開します。
167   */
168  ShuffleText.prototype.resume = function () {
169    this._resume();
170  };
171
172  /**
173   * エフェクトを強制的に再開します。
174   */
175  ShuffleText.prototype.pause = function () {
176    this._pause();
177  };
178
179  /**
180   * エフェクトを強制的に再開します。
181   */
182  ShuffleText.prototype.resume = function () {
183    this._resume();
184  };
185
186  /**
187   * エフェクトを強制的に再開します。
188   */
189  ShuffleText.prototype.pause = function () {
190    this._pause();
191  };
192
193  /**
194   * エフェクトを強制的に再開します。
195   */
196  ShuffleText.prototype.resume = function () {
197    this._resume();
198  };
199
200  /**
201   * エフェクトを強制的に再開します。
202   */
203  ShuffleText.prototype.pause = function () {
204    this._pause();
205  };
206
207  /**
208   * エフェクトを強制的に再開します。
209   */
210  ShuffleText.prototype.resume = function () {
211    this._resume();
212  };
213
214  /**
215   * エフェクトを強制的に再開します。
216   */
217  ShuffleText.prototype.pause = function () {
218    this._pause();
219  };
220
221  /**
222   * エフェクトを強制的に再開します。
223   */
224  ShuffleText.prototype.resume = function () {
225    this._resume();
226  };
227
228  /**
229   * エフェクトを強制的に再開します。
230   */
231  ShuffleText.prototype.pause = function () {
232    this._pause();
233  };
234
235  /**
236   * エフェクトを強制的に再開します。
237   */
238  ShuffleText.prototype.resume = function () {
239    this._resume();
240  };
241
242  /**
243   * エフェクトを強制的に再開します。
244   */
245  ShuffleText.prototype.pause = function () {
246    this._pause();
247  };
248
249  /**
250   * エフェクトを強制的に再開します。
251   */
252  ShuffleText.prototype.resume = function () {
253    this._resume();
254  };
255
256  /**
257   * エフェクトを強制的に再開します。
258   */
259  ShuffleText.prototype.pause = function () {
260    this._pause();
261  };
262
263  /**
264   * エフェクトを強制的に再開します。
265   */
266  ShuffleText.prototype.resume = function () {
267    this._resume();
268  };
269
270  /**
271   * エフェクトを強制的に再開します。
272   */
273  ShuffleText.prototype.pause = function () {
274    this._pause();
275  };
276
277  /**
278   * エフェクトを強制的に再開します。
279   */
280  ShuffleText.prototype.resume = function () {
281    this._resume();
282  };
283
284  /**
285   * エフェクトを強制的に再開します。
286   */
287  ShuffleText.prototype.pause = function () {
288    this._pause();
289  };
290
291  /**
292   * エフェクトを強制的に再開します。
293   */
294  ShuffleText.prototype.resume = function () {
295    this._resume();
296  };
297
298  /**
299   * エフェクトを強制的に再開します。
300   */
301  ShuffleText.prototype.pause = function () {
302    this._pause();
303  };
304
305  /**
306   * エフェクトを強制的に再開します。
307   */
308  ShuffleText.prototype.resume = function () {
309    this._resume();
310  };
311
312  /**
313   * エフェクトを強制的に再開します。
314   */
315  ShuffleText.prototype.pause = function () {
316    this._pause();
317  };
318
319  /**
320   * エフェクトを強制的に再開します。
321   */
322  ShuffleText.prototype.resume = function () {
323    this._resume();
324  };
325
326  /**
327   * エフェクトを強制的に再開します。
328   */
329  ShuffleText.prototype.pause = function () {
330    this._pause();
331  };
332
333  /**
334   * エフェクトを強制的に再開します。
335   */
336  ShuffleText.prototype.resume = function () {
337    this._resume();
338  };
339
340  /**
341   * エフェクトを強制的に再開します。
342   */
343  ShuffleText.prototype.pause = function () {
344    this._pause();
345  };
346
347  /**
348   * エフェクトを強制的に再開します。
349   */
350  ShuffleText.prototype.resume = function () {
351    this._resume();
352  };
353
354  /**
355   * エフェクトを強制的に再開します。
356   */
357  ShuffleText.prototype.pause = function () {
358    this._pause();
359  };
360
361  /**
362   * エフェクトを強制的に再開します。
363   */
364  ShuffleText.prototype.resume = function () {
365    this._resume();
366  };
367
368  /**
369   * エフェクトを強制的に再開します。
370   */
371  ShuffleText.prototype.pause = function () {
372    this._pause();
373  };
374
375  /**
376   * エフェクトを強制的に再開します。
377   */
378  ShuffleText.prototype.resume = function () {
379    this._resume();
380  };
381
382  /**
383   * エフェクトを強制的に再開します。
384   */
385  ShuffleText.prototype.pause = function () {
386    this._pause();
387  };
388
389  /**
390   * エフェクトを強制的に再開します。
391   */
392  ShuffleText.prototype.resume = function () {
393    this._resume();
394  };
395
396  /**
397   * エフェクトを強制的に再開します。
398   */
399  ShuffleText.prototype.pause = function () {
400    this._pause();
401  };
402
403  /**
404   * エフェクトを強制的に再開します。
405   */
406  ShuffleText.prototype.resume = function () {
407    this._resume();
408  };
409
410  /**
411   * エフェクトを強制的に再開します。
412   */
413  ShuffleText.prototype.pause = function () {
414    this._pause();
415  };
416
417  /**
418   * エフェクトを強制的に再開します。
419   */
420  ShuffleText.prototype.resume = function () {
421    this._resume();
422  };
423
424  /**
425   * エフェクトを強制的に再開します。
426   */
427  ShuffleText.prototype.pause = function () {
428    this._pause();
429  };
430
431  /**
432   * エフェクトを強制的に再開します。
433   */
434  ShuffleText.prototype.resume = function () {
435    this._resume();
436  };
437
438  /**
439   * エフェクトを強制的に再開します。
440   */
441  ShuffleText.prototype.pause = function () {
442    this._pause();
443  };
444
445  /**
446   * エフェクトを強制的に再開します。
447   */
448  ShuffleText.prototype.resume = function () {
449    this._resume();
450  };
451
452  /**
453   * エフェクトを強制的に再開します。
454   */
455  ShuffleText.prototype.pause = function () {
456    this._pause();
457  };
458
459  /**
460   * エフェクトを強制的に再開します。
461   */
462  ShuffleText.prototype.resume = function () {
463    this._resume();
464  };
465
466  /**
467   * エフェクトを強制的に再開します。
468   */
469  ShuffleText.prototype.pause = function () {
470    this._pause();
471  };
472
473  /**
474   * エフェクトを強制的に再開します。
475   */
476  ShuffleText.prototype.resume = function () {
477    this._resume();
478  };
479
480  /**
481   * エフェクトを強制的に再開します。
482   */
483  ShuffleText.prototype.pause = function () {
484    this._pause();
485  };
486
487  /**
488   * エフェクトを強制的に再開します。
489   */
490  ShuffleText.prototype.resume = function () {
491    this._resume();
492  };
493
494  /**
495   * エフェクトを強制的に再開します。
496   */
497  ShuffleText.prototype.pause = function () {
498    this._pause();
499  };
500
501  /**
502   * エフェクトを強制的に再開します。
503   */
504  ShuffleText.prototype.resume = function () {
505    this._resume();
506  };
507
508  /**
509   * エフェクトを強制的に再開します。
510   */
511  ShuffleText.prototype.pause = function () {
512    this._pause();
513  };
514
515  /**
516   * エフェクトを強制的に再開します。
517   */
518  ShuffleText.prototype.resume = function () {
519    this._resume();
520  };
521
522  /**
523   * エフェクトを強制的に再開します。
524   */
525  ShuffleText.prototype.pause = function () {
526    this._pause();
527  };
528
529  /**
530   * エフェクトを強制的に再開します。
531   */
532  ShuffleText.prototype.resume = function () {
533    this._resume();
534  };
535
536  /**
537   * エフェクトを強制的に再開します。
538   */
539  ShuffleText.prototype.pause = function () {
540    this._pause();
541  };
542
543  /**
544   * エフェクトを強制的に再開します。
545   */
546  ShuffleText.prototype.resume = function () {
547    this._resume();
548  };
549
550  /**
551   * エフェクトを強制的に再開します。
552   */
553  ShuffleText.prototype.pause = function () {
554    this._pause();
555  };
556
557  /**
558   * エフェクトを強制的に再開します。
559   */
560  ShuffleText.prototype.resume = function () {
561    this._resume();
562  };
563
564  /**
565   * エフェクトを強制的に再開します。
566   */
567  ShuffleText.prototype.pause = function () {
568    this._pause();
569  };
570
571  /**
572   * エフェクトを強制的に再開します。
573   */
574  ShuffleText.prototype.resume = function () {
575    this._resume();
576  };
577
578  /**
579   * エフェクトを強制的に再開します。
580   */
581  ShuffleText.prototype.pause = function () {
582    this._pause();
583  };
584
585  /**
586   * エフェクトを強制的に再開します。
587   */
588  ShuffleText.prototype.resume = function () {
589    this._resume();
590  };
591
592  /**
593   * エフェクトを強制的に再開します。
594   */
595  ShuffleText.prototype.pause = function () {
596    this._pause();
597  };
598
599  /**
600   * エフェクトを強制的に再開します。
601   */
602  ShuffleText.prototype.resume = function () {
603    this._resume();
604  };
605
606  /**
607   * エフェクトを強制的に再開します。
608   */
609  ShuffleText.prototype.pause = function () {
610    this._pause();
611  };
612
613  /**
614   * エフェクトを強制的に再開します。
615   */
616  ShuffleText.prototype.resume = function () {
617    this._resume();
618  };
619
620  /**
621   * エフェクトを強制的に再開します。
622   */
623  ShuffleText.prototype.pause = function () {
624    this._pause();
625  };
626
627  /**
628   * エフェクトを強制的に再開します。
629   */
630  ShuffleText.prototype.resume = function () {
631    this._resume();
632  };
633
634  /**
635   * エフェクトを強制的に再開します。
636   */
637  ShuffleText.prototype.pause = function () {
638    this._pause();
639  };
640
641  /**
642   * エフェクトを強制的に再開します。
643   */
644  ShuffleText.prototype.resume = function () {
645    this._resume();
646  };
647
648  /**
649   * エフェクトを強制的に再開します。
650   */
651  ShuffleText.prototype.pause = function () {
652    this._pause();
653  };
654
655  /**
656   * エフェクトを強制的に再開します。
657   */
658  ShuffleText.prototype.resume = function () {
659    this._resume();
660  };
661
662  /**
663   * エフェクトを強制的に再開します。
664   */
665  ShuffleText.prototype.pause = function () {
666    this._pause();
667  };
668
669  /**
670   * エフェクトを強制的に再開します。
671   */
672  ShuffleText.prototype.resume = function () {
673    this._resume();
674  };
675
676  /**
677   * エフェクトを強制的に再開します。
678   */
679  ShuffleText.prototype.pause = function () {
680    this._pause();
681  };
682
683  /**
684   * エフェクトを強制的に再開します。
685   */
686  ShuffleText.prototype.resume = function () {
687    this._resume();
688  };
689
690  /**
691   * エフェクトを強制的に再開します。
692   */
693  ShuffleText.prototype.pause = function () {
694    this._pause();
695  };
696
697  /**
698   * エフェクトを強制的に再開します。
699   */
700  ShuffleText.prototype.resume = function () {
701    this._resume();
702  };
703
704  /**
705   * エフェクトを強制的に再開します。
706   */
707  ShuffleText.prototype.pause = function () {
708    this._pause();
709  };
710
711  /**
712   * エフェクトを強制的に再開します。
713   */
714  ShuffleText.prototype.resume = function () {
715    this._resume();
716  };
717
718  /**
719   * エフェクトを強制的に再開します。
720   */
721  ShuffleText.prototype.pause = function () {
722    this._pause();
723  };
724
725  /**
726   * エフェクトを強制的に再開します。
727   */
728  ShuffleText.prototype.resume = function () {
729    this._resume();
730  };
731
732  /**
733   * エフェクトを強制的に再開します。
734   */
735  ShuffleText.prototype.pause = function () {
736    this._pause();
737  };
738
739  /**
740   * エフェクトを強制的に再開します。
741   */
742  ShuffleText.prototype.resume = function () {
743    this._resume();
744  };
745
746  /**
747   * エフェクトを強制的に再開します。
748   */
749  ShuffleText.prototype.pause = function () {
750    this._pause();
751  };
752
753  /**
754   * エフェクトを強制的に再開します。
755   */
756  ShuffleText.prototype.resume = function () {
757    this._resume();
758  };
759
760  /**
761   * エフェクトを強制的に再開します。
762   */
763  ShuffleText.prototype.pause = function () {
764    this._pause();
765  };
766
767  /**
768   * エフェクトを強制的に再開します。
769   */
770  ShuffleText.prototype.resume = function () {
771    this._resume();
772  };
773
774  /**
775   * エフェクトを強制的に再開します。
776   */
777  ShuffleText.prototype.pause = function () {
778    this._pause();
779  };
780
781  /**
782   * エフェクトを強制的に再開します。
783   */
784  ShuffleText.prototype.resume = function () {
785    this._resume();
786  };
787
788  /**
789   * エフェクトを強制的に再開します。
790   */
791  ShuffleText.prototype.pause = function () {
792    this._pause();
793  };
794
795  /**
796   * エフェクトを強制的に再開します。
797   */
798  ShuffleText.prototype.resume = function () {
799    this._resume();
800  };
801
802  /**
803   * エフェクトを強制的に再開します。
804   */
805  ShuffleText.prototype.pause = function () {
806    this._pause();
807  };
808
809  /**
810   * エフェクトを強制的に再開します。
811   */
812  ShuffleText.prototype.resume = function () {
813    this._resume();
814  };
815
816  /**
817   * エフェクトを強制的に再開します。
818   */
819  ShuffleText.prototype.pause = function () {
820    this._pause();
821  };
822
823  /**
824   * エフェクトを強制的に再開します。
825   */
826  ShuffleText.prototype.resume = function () {
827    this._resume();
828  };
829
830  /**
831   * エフェクトを強制的に再開します。
832   */
833  ShuffleText.prototype.pause = function () {
834    this._pause();
835  };
836
837  /**
838   * エフェクトを強制的に再開します。
839   */
840  ShuffleText.prototype.resume = function () {
841    this._resume();
842  };
843
844  /**
845   * エフェクトを強制的に再開します。
846   */
847  ShuffleText.prototype.pause = function () {
848    this._pause();
849  };
850
851  /**
852   * エフェクトを強制的に再開します。
853   */
854  ShuffleText.prototype.resume = function () {
855    this._resume();
856  };
857
858  /**
859   * エフェクトを強制的に再開します。
860   */
861  ShuffleText.prototype.pause = function () {
862    this._pause();
863  };
864
865  /**
866   * エフェクトを強制的に再開します。
867   */
868  ShuffleText.prototype.resume = function () {
869    this._resume();
870  };
871
872  /**
873   * エフェクトを強制的に再開します。
874   */
875  ShuffleText.prototype.pause = function () {
876    this._pause();
877  };
878
879  /**
880   * エフェクトを強制的に再開します。
881   */
882  ShuffleText.prototype.resume = function () {
883    this._resume();
884  };
885
886  /**
887   * エフェクトを強制的に再開します。
888   */
889  ShuffleText.prototype.pause = function () {
890    this._pause();
891  };
892
893  /**
894   * エフェクトを強制的に再開します。
895   */
896  ShuffleText.prototype.resume = function () {
897    this._resume();
898  };
899
900  /**
901   * エフェクトを強制的に再開します。
902   */
903  ShuffleText.prototype.pause = function () {
904    this._pause();
905  };
906
907  /**
908   * エフェクトを強制的に再開します。
909   */
910  ShuffleText.prototype.resume = function () {
911    this._resume();
912  };
913
914  /**
915   * エフェクトを強制的に再開します。
916   */
917  ShuffleText.prototype.pause = function () {
918    this._pause();
919  };
920
921  /**
922   * エフェクトを強制的に再開します。
923   */
924  ShuffleText.prototype.resume = function () {
925    this._resume();
926  };
927
928  /**
929   * エフェクトを強制的に再開します。
930   */
931  ShuffleText.prototype.pause = function () {
932    this._pause();
933  };
934
935  /**
936   * エフェクトを強制的に再開します。
937   */
938  ShuffleText.prototype.resume = function () {
939    this._resume();
940  };
941
942  /**
943   * エフェクトを強制的に再開します。
944   */
945  ShuffleText.prototype.pause = function () {
946    this._pause();
947  };
948
949  /**
950   * エフェクトを強制的に再開します。
951   */
952  ShuffleText.prototype.resume = function () {
953    this._resume();
954  };
955
956  /**
957   * エフェクトを強制的に再開します。
958   */
959  ShuffleText.prototype.pause = function () {
960    this._pause();
961  };
962
963  /**
964   * エフェクトを強制的に再開します。
965   */
966  ShuffleText.prototype.resume = function () {
967    this._resume();
968  };
969
970  /**
971   * エフェクトを強制的に再開します。
972   */
973  ShuffleText.prototype.pause = function () {
974    this._pause();
975  };
976
977  /**
978   * エフェクトを強制的に再開します。
979   */
980  ShuffleText.prototype.resume = function () {
981    this._resume();
982  };
983
984  /**
985   * エフェクトを強制的に再開します。
986   */
987  ShuffleText.prototype.pause = function () {
988    this._pause();
989  };
990
991  /**
992   * エフェクトを強制的に再開します。
993   */
994  ShuffleText.prototype.resume = function () {
995    this._resume();
996  };
997
998  /**
999   * エフェクトを強制的に再開します。
1000  */
1001  ShuffleText.prototype.pause = function () {
1002    this._pause();
1003  };
1004
1005  /**
1006   * エフェクトを強制的に再開します。
1007  */
1008  ShuffleText.prototype.resume = function () {
1009    this._resume();
1010  };
1011
1012  /**
1013   * エフェクトを強制的に再開します。
1014  */
1015  ShuffleText.prototype.pause = function () {
1016    this._pause();
1017  };
1018
1019  /**
1020   * エフェクトを強制的に再開します。
1021  */
1022  ShuffleText.prototype.resume = function () {
1023    this._resume();
1024  };
1025
1026  /**
1027   * エフェクトを強制的に再開します。
1028  */
1029  ShuffleText.prototype.pause = function () {
1030    this._pause();
1031  };
1032
1033  /**
1034   * エフェクトを強制的に再開します。
1035  */
1036  ShuffleText.prototype.resume = function () {
1037    this._resume();
1038  };
1039
1040  /**
1041   * エフェクトを強制的に再開します。
1042  */
1043  ShuffleText.prototype.pause = function () {
1044    this._pause();
1045  };
1046
1047  /**
1048   * エフェクトを強制的に再開します。
1049  */
1050  ShuffleText.prototype.resume = function () {
1051    this._resume();
1052  };
1053
1054  /**
1055   * エフェクトを強制的に再開します。
1056  */
1057  ShuffleText.prototype.pause = function () {
1058    this._pause();
1059  };
1060
1061  /**
1062   * エフェクトを強制的に再開します。
1063  */
1064  ShuffleText.prototype.resume = function () {
1065    this._resume();
1066  };
1067
1068  /**
1069   * エフェクトを強制的に再開します。
1070  */
1071  ShuffleText.prototype.pause = function () {
1072    this._pause();
1073  };
1074
1075  /**
1076   * エフェクトを強制的に再開します。
1077  */
1078  ShuffleText.prototype.resume = function () {
1079    this._resume();
1080  };
1081
1082  /**
1083   * エフェクトを強制的に再開します。
1084  */
1085  ShuffleText.prototype.pause = function () {
1086    this._pause();
1087  };
1088
1089  /**
1090   * エフェクトを強制的に再開します。
1091  */
1092  ShuffleText.prototype.resume = function () {
1093    this._resume();
1094  };
1095
1096  /**
1097   * エフェクトを強制的に再開します。
1098  */
1099  ShuffleText.prototype.pause = function () {
1100    this._pause();
1101  };
1102
1103  /**
1104   * エフェクトを強制的に再開します。
1105  */
1106  ShuffleText.prototype.resume = function () {
1107    this._resume();
1108  };
1109
1110  /**
1111   * エフェクトを強制的に再開します。
1112  */
1113  ShuffleText.prototype.pause = function () {
1114    this._pause();
1115  };
1116
1117  /**
1118   * エフェクトを強制的に再開します。
1119  */
1120  ShuffleText.prototype.resume = function () {
1121    this._resume();
1122  };
1123
1124  /**
1125   * エフェクトを強制的に再開します。
1126  */
1127  ShuffleText.prototype.pause = function () {
1128    this._pause();
1129  };
1130
1131  /**
1132   * エフェクトを強制的に再開します。
1133  */
1134  ShuffleText.prototype.resume = function () {
1135    this._resume();
1136  };
1137
1138  /**
1139   * エフェクトを強制的に再開します。
1140  */
1141  ShuffleText.prototype.pause = function () {
1142    this._pause();
1143  };
1144
1145  /**
1146   * エフェクトを強制的に再開します。
1147  */
1148  ShuffleText.prototype.resume = function () {
1149    this._resume();
1150  };
1151
1152  /**
1153   * エフェクトを強制的に再開します。
1154  */
1155  ShuffleText.prototype.pause = function () {
1156    this._pause();
1157  };
1158
1159  /**
1160   * エフェクトを強制的に再開します。
1161  */
1162  ShuffleText.prototype.resume = function () {
1163    this._resume();
1164  };
1165
1166  /**
1167   * エフェクトを強制的に再開します。
1168  */
1169  ShuffleText.prototype.pause = function () {
1170    this._pause();
1171  };
1172
1173  /**
1174   * エフェクトを強制的に再開します。
1175  */
1176  ShuffleText.prototype.resume = function () {
1177    this._resume();
1178  };
1179
1180  /**
1181   * エフェクトを強制的に再開します。
1182  */
1183  ShuffleText.prototype.pause = function () {
1184    this._pause();
1185  };
1186
1187  /**
1188   * エフェクトを強制的に再開します。
1189  */
1190  ShuffleText.prototype.resume = function () {
1191    this._resume();
1192  };
1193
1194  /**
1195   * エフェクトを強制的に再開します。
1196  */
1197  ShuffleText.prototype.pause = function () {
1198    this._pause();
1199  };
1200
1201  /**
1202   * エフェクトを強制的に再開します。
1203  */
1204  ShuffleText.prototype.resume = function () {
1205    this._resume();
1206  };
1207
1208  /**
1209   * エフェクトを強制的に再開します。
1210  */
1211  ShuffleText.prototype.pause = function () {
1212    this._pause();
1213  };
1214
1215  /**
1216   * エフェクトを強制的に再開します。
1217  */
1218  ShuffleText.prototype.resume = function () {
1219    this._resume();
1220  };
1221
1222  /**
1223   * エフェクトを強制的に再開します。
1224  */
1225  ShuffleText.prototype.pause = function () {
1226    this._pause();
1227  };
1228
1229  /**
1230   * エフェクトを強制的に再開します。
1231  */
1232  ShuffleText.prototype.resume = function () {
1233    this._resume();
1234  };
1235
1236  /**
1237   * エフェクトを強制的に再開します。
1238  */
1239  ShuffleText.prototype.pause = function () {
1240    this._pause();
1241  };
1242
1243  /**
1244   * エフェクトを強制的に再開します。
1245  */
1246  ShuffleText.prototype.resume = function () {
1247    this._resume();
1248  };
1249
1250  /**
1251   * エフェクトを強制的に再開します。
1252  */
1253  ShuffleText.prototype.pause = function () {
1254    this._pause();
1255  };
1256
1257  /**
1258   * エフェクトを強制的に再開します。
1259  */
1260  ShuffleText.prototype.resume = function () {
1261    this._resume();
1262  };
1263
1264  /**
1265   * エフェクトを強制的に再開します。
1266  */
1267  ShuffleText.prototype.pause = function () {
1268    this._pause();
1269  };
1270
1271  /**
1272   * エフェクトを強制的に再開します。
1273  */
1274  ShuffleText.prototype.resume = function () {
1275    this._resume();
1276  };
1277
1278  /**
1279   * エフェクトを強制的に再開します。
1280  */
1281  ShuffleText.prototype.pause = function () {
1282    this._pause();
1283  };
1284
1285  /**
1286   * エフェクトを強制的に再開します。
1287  */
1288  ShuffleText.prototype.resume = function () {
1289    this._resume();
1290  };
1291
1292  /**
1293   * エフェクトを強制的に再開します。
1294  */
1295  ShuffleText.prototype.pause = function () {
1296    this._pause();
1297  };
1298
1299  /**
1300   * エフェクトを強制的に再開します。
1301  */
1302  ShuffleText.prototype.resume = function () {
1303    this._resume();
1304  };
1305
1306  /**
1307   * エフェクトを強制的に再開します。
1308  */
1309  ShuffleText.prototype.pause = function () {
1310    this._pause();
1311  };
1312
1313  /**
1314   * エフェクトを強制的に再開します。
1315  */
1316  ShuffleText.prototype.resume = function () {
1317    this._resume();
1318  };
1319
1320  /**
1321   * エフェクトを強制的に再開します。
1322  */
1323  ShuffleText.prototype.pause = function () {
1324    this._pause();
1325  };
1326
1327  /**
1328   * エフェクトを強制的に再開します。
1329  */
1330  ShuffleText.prototype.resume = function () {
1331    this._resume();
1332  };
1333
1334  /**
1335   * エフェクトを強制的に再開します。
1336  */
1337  ShuffleText.prototype.pause = function () {
1338    this._pause();
1339  };
1340
1341  /**
1342   * エフェクトを強制的に再開します。
1343  */
1344  ShuffleText.prototype.resume = function () {
1345    this._resume();
1346  };
1347
1348  /**
1349   * エフェクトを強制的に再開します。
1350  */
1351  ShuffleText.prototype.pause = function () {
1352    this._pause();
1353  };
1354
1355  /**
1356   * エフェクトを強制的に再開します。
1357  */
1358  ShuffleText.prototype.resume = function () {
1359    this._resume();
1360  };
1361
1362  /**
1363   * エフェクトを強制的に再開します。
1364  */
1365  ShuffleText.prototype.pause = function () {
1366    this._pause();
1367  };
1368
1369  /**
1370   * エフェクトを強制的に再開します。
1371  */
1372  ShuffleText.prototype.resume = function () {
1373    this._resume();
1374  };
1375
1376  /**
1377   * エフェクトを強制的に再開します。
1378  */
1379  ShuffleText.prototype.pause = function () {
1380    this._pause();
1381  };
1382
1383  /**
1384   * エフェクトを強制的に再開します。
1385  */
1386  ShuffleText.prototype.resume = function () {
1387    this._resume();
1388  };
1389
1390  /**
1391   * エフェクトを強制的に再開します。
1392  */
1393  ShuffleText.prototype.pause = function () {
1394    this._pause();
1395  };
1396
1397  /**
1398   * エフェクトを強制的に再開します。
1399  */
1400  ShuffleText.prototype.resume = function () {
1401    this._resume();
1402  };
1403
1404  /**
1405   * エフェクトを強制的に再開します。
1406  */
1407  ShuffleText.prototype.pause = function () {
1408    this._pause();
1409  };
1410
1411  /**
1412   * エフェクトを強制的に再開します。
1413  */
1414  ShuffleText.prototype.resume = function () {
1415    this._resume();
1416  };
1417
1418  /**
1419   * エフェクトを強制的に再開します。
1420  */
1421  ShuffleText.prototype.pause = function () {
1422    this._pause();
1423  };
1424
1425  /**
1426   * エフェクトを強制的に再開します。
1427  */
1428  ShuffleText.prototype.resume = function () {
1429    this._resume();
1430  };
1431
1432  /**
1433   * エフェクトを強制的に再開します。
1434  */
1435  ShuffleText.prototype.pause = function () {
1436    this._pause();
1437  };
1438
1439  /**
1440   * エフェクトを強制的に再開します。
1441  */
1442  ShuffleText.prototype.resume = function () {
1443    this._resume();
1444  };
1445
1446  /**
1447   * エフェクトを強
```

```
54 }
55 this._timeStart = new Date().getTime();
56 this._isRunning = true;
57 this._requestAnimationFrameId = requestAnimationFrame(function () {
58     _this._onInterval();
59 });
60 if (this._element) {
61     this._element.textContent = str;
62 }
63 };
64
65 // Stop effect. 停止します。
66 ShuffleText.prototype.stop = function () {
67     this._isRunning = false;
68     cancelAnimationFrame(this._requestAnimationFrameId);
69 };
70
71 // メモリ解放のためインスタンスを破棄します。
72 ShuffleText.prototype.dispose = function () {
73     cancelAnimationFrame(this._requestAnimationFrameId);
74     this._isRunning = false;
75     this.duration = 0;
76     this._originalStr = "";
77     this._originalLength = 0;
78     this._timeCurrent = 0;
79     this._timeStart = 0;
80     this._randomIndex = [];
81     this._element = null;
82     this._requestAnimationFrameId = 0;
83 };
84
85 // インターバルハンドラーです。
86 ShuffleText.prototype._onInterval = function () {
87     let _this = this;
88     this._timeCurrent = new Date().getTime() - this._timeStart;
89     let percent = this._timeCurrent / this.duration;
90     let str = "";
91     for (let i = 0; i < this._originalLength; i++) {
92         if (percent >= this._randomIndex[i]) {
93             str += this._originalStr.charAt(i);
94         } else if (percent < this._randomIndex[i] / 3) {
95             str += this.emptyCharacter;
96         } else {
97             str += this.sourceRandomCharacter.charAt(Math.floor(Math.random() * this.sourc
eRandomCharacter.length));
98         }
99     }
100    if (percent > 1) {
101        str = this._originalStr;
102        this._isRunning = false;
103    }
104    if (this._element) {
105        this._element.textContent = str;
106    }
107    if (this._isRunning) {
108        this._requestAnimationFrameId = requestAnimationFrame(function () {
109            _this._onInterval();
110        });
111 }
```

```
111     }
112 };
113     return ShuffleText;
114 }());
115     return ShuffleText;
116 })) );
117 }
```


第9章

桜吹雪

古来より日本人の心をとらえてはなさない「さくら」。春の桜吹雪、夏の虫の乱舞、秋の紅葉、冬のしんしんと降り積もる雪。季節を彩る演出を、JavaScriptを使って実現します。

9.1 HTML

作例例は右のようになります。さくらの花びらがひらひらと舞う演出効果を、CSSとJavaScriptを使って実現しています。

//blankline <https://actyway.com/8351> (現在リンク切れ) を元に適宜改変いたしました。動作例^aとソースコード^bです。ご参考になれば幸いです。

^a https://wave-improve.netlify.app/sakura_fubuki/index.html

^b https://github.com/Atelier-Mirai/wave-improve/tree/master/sakura_fubuki



▼index.html

```
<!DOCTYPE html>
<html>
  <head>
```

```
<meta charset="utf-8">
<title>桜吹雪</title>
</head>

<body>
<!-- 略 -->

<!-- 桜吹雪の効果 -->
<link rel="stylesheet" href="sakura.css">
<script src="sakura.js"></script>
</body>
</html>
```

HTML は今回の主眼ではございませんので省略しておりますが、`</body>` 直前に桜吹雪のための CSS と JavaScript を読み込んでいます。

9.2 CSS

`sakura.css` は、桜の花びらの色やひらひら回転する CSS アニメーションのための CSS です。

▼ `sakura.css`

```
html,
body {
    overflow-x: hidden;
    overflow-y: clip;
    position: relative;
}

/* 花びらの形 */
.hana {
    position: absolute;
    height: 0;
    width: 0;
    border: 10px solid transparent;
    border-radius: 15px;
    border-top-right-radius: 0;
    border-bottom-left-radius: 0;
}

.hana::after {
    content: "";
    display: block;
    position: absolute;
    top: -7px;
    left: -7px;
    height: 0;
    width: 0;
    border: 10px solid transparent;
    border-radius: 15px;
    border-top-right-radius: 0;
    border-bottom-left-radius: 0;
    transform: rotate(15deg);
}
```

```

/* 花びらの色 */
.t1, .t1::after { border-color: #fef4f4; }
.t2, .t2::after { border-color: #ffd1d9; }
.t3, .t3::after { border-color: #ffc0cb; }
.t4, .t4::after { border-color: #ffc0cb; }
.t5, .t5::after { border-color: #ffafbd; }

/* 花びらがひらひら回転する動き */
.a1 { animation: a1 12s infinite; }
.a2 { animation: a2 10s infinite; }
.a3 { animation: a3 9s infinite; }
.a4 { animation: a4 9s infinite; }
.a5 { animation: a5 8s infinite; }

@keyframes a1 {
  from { transform: rotate( 0deg) scale(.9); }
  50% { transform: rotate(270deg) scale(.9); }
  to { transform: rotate( 1deg) scale(.9); }
}
@keyframes a2 {
  from { transform: rotate( -90deg) scale(.8); }
  50% { transform: rotate(-360deg) scale(.8); }
  to { transform: rotate( -89deg) scale(.8); }
}
@keyframes a3 {
  from { transform: rotate( 30deg) scale(.7); }
  50% { transform: rotate(300deg) scale(.7); }
  to { transform: rotate( 29deg) scale(.7); }
}
@keyframes a4 {
  from { transform: rotate(-120deg) scale(.6); }
  50% { transform: rotate(-390deg) scale(.6); }
  to { transform: rotate(-119deg) scale(.6); }
}
@keyframes a5 {
  from { transform: rotate( 60deg) scale(.5); }
  50% { transform: rotate(330deg) scale(.5); }
  to { transform: rotate( 59deg) scale(.5); }
}

```

♣ 花びらの形

.hana で、一枚の花びらの半分 (楕円のような形) を作成します。そして .hana::after では、
`transform: rotate(15deg);` と書かれている点に着目してください。もう半分を 15 度傾けて結合
することで、一枚の花びらを完成させています。

♣ 花びらの色

`.t1, .t1::after { border-color: #fef4f4; }` で、花びらの色を設定します。.t1 ~ .t5
まで、淡い桜色から濃い桜色まで五種類の花びらの色を付けています。

♣ 花びらがひらひら回転する動き

CSS アニメーションを用いて、回転する動きと少しづつ大きさが変化する動きを実現しています。
`.a1 { animation: a1 12s infinite; }` で、花びらのアニメーションを設定します。.a1 ~ .a5

まで、五種類の花びらの動きを付けています。

`.a1 { animation: a1 12s infinite; }` は、アニメーション `a1` を 12 秒掛けて実行します。
アニメーション `a1` は、以下の動きをします。

```
@keyframes a1 {
  from { transform: rotate( 0deg) scale(.9); }
  50% { transform: rotate(270deg) scale(.9); }
  to   { transform: rotate( 1deg) scale(.9); }
}
```

最初 (`from`) は、回転角 0 度 (`rotate(0deg)`) で、中ほど (50%) は、回転角 270 度 (`rotate(270deg)`) となり、最後に (`to`) は、回転角 1 度 (`rotate(1deg)`) になるまで回転します。
`scale(.9)` とありますので、`.hana` で作成した花びらの大きさより少し小さ目 (90%) の花びらとなります。

9.3 JavaScript

JavaScript は、(主に) ブラウザ上で動くプログラミング言語です。HTML の要素を取得し、追加や編集、削除など様々な制御することができるので、動きのあるウェブサイトを作ることが出来るようになります。

今回の桜吹雪の例では、花びら用の `<div>` を 50 枚生成し、上から下へ落としていきます。左右への揺らぎも表現したいので、一定回数右へ移動したら左へ移動するようにしています。

JavaScript に関しては、初心者の方向けに、コードの意味を逐次解説した [スラスラ読める JavaScript ふりがなプログラミング^a](#) がございますので、ご一読下さい。

^a <https://www.amazon.co.jp/dp/4295003859>



今回作成した `sakura.js` では、`class` 構文を用いて、花びら座標計算用オブジェクトを作成し、花びら `<div>` に反映させるようにしています。せっかくの機会ですので、ここで少し「オブジェクト指向」についてご紹介します。

♣ オブジェクト指向

オブジェクト指向は、ソフトウェア開発とコンピュータプログラミングのために用いられる考え方である。元々は特定のプログラミングパラダイムを説明するために考案された言葉であり、その当時の革新的技術であった GUI (グラフィカル・ユーザーインターフェース) とも密接に関連していた。明確な用語としては 1970 年代に誕生し、1990 年頃にはソフトウェア開発の総合技術としての共通認識を確立

している。

オブジェクトとは、**データ構造とその専属手続きを一つにまとめたもの** / **情報資源とその処理手順を一つにまとめたもの**を指している。データとプロセスを個別に扱わずに、双方を一体化したオブジェクトを基礎要素にし、メッセージと形容されるオブジェクト間の相互作用を重視して、ソフトウェア全体を構築しようとする考え方がオブジェクト指向である。

オブジェクト指向(object-oriented)という言葉自体は、1972年から80年にかけてプログラミング言語「Smalltalk」を開発したゼロックス社パロアルト研究所の計算機科学者アラン・ケイが、その言語設計を説明する過程で誕生している。大学院時代のケイがプログラミング言語「Simula」に感化されて日夜プログラミング・アーキテクチャの思索に耽っていた1967年頃、今何をしているのかと尋ねてきた知人に対して「object-oriented programmingだよ」とその時の造語で答えたのが原点であるという。

オブジェクト指向プログラミング(OOP)とは、「**オブジェクト**」という概念に基づいたプログラミングパラダイムの一つである。オブジェクトは、任意個数のフィールド（属性、プロパティまたは変数）で構成されるデータと、任意個数の（メソッドまたは関数）で構成されるコードのひとまとまりで構成される。

オブジェクトの特徴として、オブジェクト自身の手続きが自身のデータフィールドを読み書きできることが挙げられる（オブジェクトにはthisやselfという概念がある）。また、OOPでは、相互に作用するオブジェクトを組み合わせてプログラムを設計する。OOP言語のありかたは多様であるが、最も一般的といえるものは、オブジェクトがクラス（型）のインスタンス（実例）であり、また、オブジェクトの型もクラスとして規定されるクラスベースといわれるものである。^{*1}

♣ オブジェクト指向 練習

それでは、理解を深めるために練習していきましょう。

JavaScriptも、オブジェクト指向言語ですので、クラス（型・雛型）からインスタンス（実例）を作成し、コーディングすることが出来ます。

簡単な例ですが、四角形とその面積を求める例です。

▼ sakura.js

```

1 // 長方形クラス(型)の宣言
2 class Rectangle {
3     // 構築子
4     constructor(height, width) {
5         this.width = width; // 幅
6         this.height = height; // 高さ
7     }
8
9     // 面積を求めるメソッド(関数)
10    area() {
11        return this.width * this.height;
12    }
13 }
14
15 // 実際に使ってみる
16 // 幅10cm, 高さ20cm の長方形のインスタンス(実例)を作成
17 rect1 = new Rectangle(10, 20);
18
19 // 幅と高さを表示させてみる

```

^{*1} 出典: ウィキペディア

```

20 console.log(rect1.width); // => 10 と幅が表示される
21 console.log(rect1.height); // => 20 と高さが表示される
22
23 // 面積を求めるために、area メソッドを呼び出す
24 console.log(rect1.area()); // => 200 と面積が表示される
25
26 // 高さを半分にして、正方形にしてみる
27 rect1.height /= 2;
28 console.log(rect1.height); // => 10 と高さが半分になっている
29
30 // もう一度面積を表示させてみる
31 console.log(rect1.area()); // => 100 と面積が表示される

```

クラスを作成せずに、単純に次のように書いても、面積を求めることができます。

▼ sakura.js

```

1 // 面積を求める関数を定義
2 let area = (width, height) => {
3   return width * height;
4 }
5
6 // 四角形の幅と高さ
7 let width1 = 10
8 let height1 = 20;
9
10 // 幅と高さを表示させてみる
11 console.log(width1); // => 10 と幅が表示される
12 console.log(height1); // => 20 と高さが表示される
13
14 // 面積を求めるために、area 関数に幅と高さを渡す
15 console.log(area(width1, height1)); // => 200 と面積が表示される
16
17 // 高さを半分にして、正方形にしてみる
18 height1 /= 2;
19 console.log(height1); // => 10 と高さが半分になっている
20
21 // もう一度面積を表示させてみる
22 console.log(area(width1, height1)); // => 100 と面積が表示される

```

しかしながら、四角形1、四角形2、四角形3・・・とたくさんの図形がある場合にはどうでしょう。あるいは、三角形や円の面積を求める時もあるかもしれません。

クラスを使ったコーディングの有用性が掴めるのではないでしょうか。

♣ sakura.js のご紹介

それでは、花びらクラスを用いたコードのご紹介です。花びらの位置情報と、それを操作するためのメソッド(関数)、そして実際に画面表示するには、花びら div 要素(DOM)に反映させが必要ですから、そのためのメソッドも備わっています。

紙幅の都合上、詳細は割愛いたしますが、雰囲気だけでも掴んでいただければ幸いです。

▼ sakura.js

```

1 /**
2  櫻の花びらが舞い散るJavaScript
3  https://actyway.com/8351 を元に作成
4 */

```

```

5
6 (
7 () => {
8 // =====
9 // 定数宣言等
10 // =====
11 const NUMBER_OF_HANABIRAS = 50; // 花びらの枚数
12 const FPS = 24; // 一秒間に24回 動かす
13 const HANABIRA_HEIGHT = 30; // 花びらの高さ 回転するので最大値は 30px
14 const HANABIRA_WIDTH = 30; // 花びらの幅 回転するので最大値は 30px
15 const HANABIRA_Z_BASE = 10000; // 花びらの z-index の基準値
16
17 // ウィンドウの高さ
18 const windowHeight = window.innerHeight;
19 // ウィンドウの幅(スクロールバー除く)
20 const windowWidth = document.documentElement.clientWidth
21 // スクロール位置
22 let scroll = document.documentElement.scrollTop || document.body.scrollTop;
23 // スクロール時のイベント登録(スクロール時に花びらがウィンドウ内に納まる為に)
24 document.addEventListener('scroll', () => {
25   scroll = document.documentElement.scrollTop || document.body.scrollTop;
26 }, false);
27
28 // =====
29 // 亂数関数
30 // min 以上 max 以下の乱数を返す (integer)
31 // min 以上 max 未満の乱数を返す (float)
32 // =====
33 let rand = (min, max, type = "integer") => {
34   if(type === "integer"){
35     return Math.floor(Math.random() * (max-min+1)) + min;
36   } else {
37     return Math.random() * (max-min) + min;
38   }
39 };
40
41 // =====
42 // 花びらクラスの宣言
43 // =====
44 class Hanabira {
45   // コンストラクタ(構築子)
46   constructor(id, x, y, z, tremorMax, fallingSpeed, className) {
47     this.id = id;
48     this.x = x;
49     this.y = y;
50     this.z = z;
51     this.tremorMax = tremorMax;
52     this.fallingSpeed = fallingSpeed;
53     this.className = className;
54
55     this.direction = "right";
56     this.tremorCount = 0;
57   }
58
59   // 際大揺らぎ回数に達しているか ?
60   isTremorMax() {
61     return (this.tremorCount === this.tremorMax)
62   }
}

```

```
63 // 揺らぎ方向転換
64 directionSwitch() {
65     if (this.direction === "right") {
66         this.direction = "left";
67     } else {
68         this.direction = "right";
69     }
70 }
71
72 // 花びらの位置に関して
73 // 空中にいるか？(ウィンドウ内か？)
74 isInTheAir() {
75     return (this.y < scroll + windowHeight - HANABIRA_HEIGHT);
76 }
77
78 // 地面に着いたか？
79 isOnTheGround() {
80     return !this.isInTheAir();
81 }
82
83
84 // 右端にいるか？
85 isOnTheRightEdge() {
86     return (this.x + HANABIRA_WIDTH >= windowHeight);
87 }
88
89 // 左端にいるか？
90 isOnTheLeftEdge() {
91     // 花びら幅の半分の位置なら、左端と見做す。
92     return (this.x <= HANABIRA_WIDTH / 2);
93 }
94
95 // 花びらの x, y 座標を更新する
96 move() {
97     // 花びらの位置がウィンドウ内なら
98     if (this.isInTheAir()) {
99         // 同一方向へtremorMax回移動したなら、移動方向を反転させる
100        if (this.isTremorMax()) {
101            this.directionSwitch();
102            this.tremorCount = 0;
103        }
104
105        // 左右に移動する
106        let deltaX = rand(0.3, 0.6, "float");
107        let signFlag = (this.direction === "right" ? +1 : -1);
108        this.x += signFlag * deltaX;
109
110        // もし右端にいるなら、左端に移動する
111        if (this.isOnTheRightEdge()) {
112            this.x = HANABIRA_WIDTH / 2;
113        }
114
115        // もし左端にいるなら、右端に移動する
116        if (this.isOnTheLeftEdge()) {
117            this.x = windowHeight - HANABIRA_WIDTH;
118        }
119
120        // 移動回数を増やす
```

```

121     this.tremorCount++;
122
123     // 落下速度分を加える
124     this.y += this.fallingSpeed;
125
126     // もし地面に着いているなら、上に戻す
127 } else if (this.isOnTheGround()) {
128     this.y = scroll;
129     this.x = rand(0, windowHeight - HANABIRA_WIDTH);
130 }
131 }
132
133 // 位置情報を DOM に反映させる
134 applyPositionInformation(domHanabira) {
135     domHanabira.setAttribute('style', `top: ${this.y}px; left: ${this.x}px; z-index: ${this.z};`);
136 }
137 }
138
139 // =====
140 // 初期化処理
141 // 桜の花びらのための新しい div 要素を作成し、body の末尾に追加
142 // 作業用の 花びらインスタンスも生成する
143 // =====
144 const divSakura = document.createElement("div");
145 document.body.appendChild(divSakura);
146
147 let domHanabiras = []; // 花びら要素の配列
148 let jsHanabiras = []; // 花びらjsオブジェクトの配列
149 // それぞれの花びらについて、位置等の初期設定を行う
150 for(let i = 0; i < NUMBER_OF_HANABIRAS; i++){
151     let id          = i;
152     let x           = rand(HANABIRA_WIDTH / 2, windowHeight - HANABIRA_WIDTH);
153     let y           = rand(-500, 0) + scroll;
154     let z           = HANABIRA_Z_BASE + i;
155     let tremorMax   = rand(15, 50);
156     let fallingSpeed = rand(1, 3);
157     let className   = `hana t${rand(1, 5)} a${rand(1, 5)}`;
158     let jsHanabira = new Hanabira(id, x, y, z, tremorMax, fallingSpeed, className);
159     jsHanabiras[i] = jsHanabira;
160
161     // 花びらの div を作る
162     let domHanabira = document.createElement('div');
163     // 初期表示位置を設定する
164     jsHanabira.applyPositionInformation(domHanabira);
165     // ID 付与
166     domHanabira.id = i;
167     // ランダムに生成した花びらの色とアニメーションのための css class を設定する
168     domHanabira.setAttribute('class', jsHanabira.className);
169     // 作成した花びらをDOMに追加、画面に表示されるようにする
170     divSakura.appendChild(domHanabira);
171     // 扱いやすくするために、花びら要素達を配列に格納
172     domHanabiras[i] = domHanabira;
173 }
174
175 // =====
176 // メイン処理
177 // 生成したそれぞれの花びらの位置情報を更新し、画面に反映する。

```

```

178 // =====
179 setInterval(() => {
180   for(let jsHanabira of jsHanabiras) {
181     // 各花びらに対し、位置情報の更新処理を行う
182     jsHanabira.move();
183
184     // js オブジェクトの位置情報を、domオブジェクトモデルの位置に反映する。
185     let id      = jsHanabira.id;
186     let domHanabira = domHanabiras[id];
187     jsHanabira.applyPositionInformation(domHanabira);
188   }
189 }, 1000 / FPS);
190 }
191 )();

```

9.4 春夏秋冬

春の桜吹雪、夏の蛍の乱舞、秋の紅葉、冬のしんしんと降り積もる雪。日本は、四季折々の豊かな自然に恵まれた豊かな国です。

桜吹雪は上記で実現いたしました。

[particles.js^{*2}](#) というライブラリを利用すると、夏の蛍や冬の雪のような効果を簡単に実現できます。

♣ 夏の蛍

[ホタルが舞う^{*3}](#) にて紹介されていますので、参考になさってください。

♣ 秋の紅葉

[JS と CSS で落ち葉をひらひらと舞わせるエフェクトを実装する方法^{*4}](#) にて紹介されていますので、参考になさってください。

♣ 冬の雪

[particles.js【公式】^{*5}](#) にて紹介されています。右上 CodePen より、ソースコードも見られますので、参考になさってください。

*2 <https://vincentgarreau.com/particles.js/>

*3 <https://coco-factory.jp/ugokuweb/move02/5-7/>

*4 <https://web-dev.tech/front-end/javascript/autumn-leaves-falling-effect/>

*5 <https://vincentgarreau.com/particles.js/#snow>

第 10 章

スクロールしたら要素を表示させる

スクロールに伴い、見出しや画像などがされると、動きがある印象深いサイトになります。CSS と JavaScript を使って、実現していきます。

10.1 HTML

スクロールしたらフワッと要素を表示させるスクリプトの使い勝手を良くしてみる^a を元に適宜改変いたしました。作例例は右のようになります。緑色の箱が並んでいるだけでございますので、動作例^bをごらんになってください。また ソースコード^c もござります。

詳しい解説が元のサイトにございますので、ここでは、HTML / CSS / JavaScript それぞれのソースコードを掲載するのみで、説明は割愛いたします。

簡単に実装できますので、是非、お使いになってみてください。

スクロールしたらフワッと要素を表示させる

<http://noze.space/archives/415> より引用改変

要素1 up	要素2 down	要素3 up 400	要素4 down 400	要素5 up 600
要素6 down 600	要素7 up 800	要素8 down 800	要素9 ltr	要素10 rtl
要素11 ltr 800	要素12 rtl 800	要素13 up #elem13	要素14 up trigger13	要素15 up trigger13 800
要素16 scale_up	要素17 scale_down	要素18 rotate_l	要素19 rotate_r trigger18	要素20 rotate_r trigger18 800

^a <http://noze.space/archives/415>
^b https://wave-improve.netlify.app/scroll_animation/index.html
^c https://github.com/Atelier-Mirai/wave-improve/tree/master/scroll_animation

▼index.html

```
<!DOCTYPE html>
<html>
  <head>
```

```

<meta charset="utf-8">
<title>スクロールでフワッと要素表示</title>

<!-- 簡単に見栄えの良いページを作るためのスタイルシート -->
<link rel="stylesheet" href="https://unpkg.com/sakura.css/css/sakura.css">
<!-- スクロールアニメーション用のスタイルシートなど -->
<link rel="stylesheet" href="scroll_animation.css">
</head>

<body>
<section>
  <h1>スクロールしたらフワッと要素を表示させる</h1>
  <p><a href="http://noze.space/archives/415">
    http://noze.space/archives/415 より引用改変</a></p>

  <div class="sa up">要素1<br>up</div>
  <div class="sa down">要素2<br>down</div>
  <div class="sa up" data-sa_delay="400">要素3<br>up<br>400</div>
  <div class="sa down" data-sa_delay="400">要素4<br>down<br>400</div>
  <div class="sa up" data-sa_delay="600">要素5<br>up<br>600</div>
  <div class="sa down" data-sa_delay="600">要素6<br>down<br>600</div>
  <div class="sa up" data-sa_delay="800">要素7<br>up<br>800</div>
  <div class="sa down" data-sa_delay="800">要素8<br>down<br>800</div>
  <div class="sa rtl">要素9<br>ltr</div>
  <div class="sa rtl">要素10<br>rtl</div>
  <div class="sa rtl" data-sa_delay="800">要素11<br>ltr<br>800</div>
  <div class="sa rtl" data-sa_delay="800">要素12<br>rtl<br>800</div>
  <div class="sa up" id="elem13">要素13<br>up<br>#elem13</div>
  <div class="sa up" data-sa_trigger="#elem13">
    要素14<br>up<br>trigger13
  </div>
  <div class="sa up" data-sa_trigger="#elem13" data-sa_delay="800">
    要素15<br>up<br>trigger13<br>800
  </div>
  <div class="sa scale_up">要素16<br>scale_up</div>
  <div class="sa scale_down">要素17<br>scale_down</div>
  <div class="sa rotate_l" id="elem18">要素18<br>ratate_l</div>
  <div class="sa rotate_r">要素19<br>ratate_r<br>trigger18</div>
  <div class="sa rotate_r" data-sa_delay="800">
    要素20<br>ratate_r<br>trigger18<br>800
  </div>
</section>

<!-- スクロールアニメーション用のスクリプトを読み込む -->
<script src="scroll_animation.js"></script>
</body>
</html>

```

10.2 CSS

```

/* sakura.css 補正 */
body { max-width: 90vw; }

/* グリッドレイアウトで、div要素を配置する */

```

```

section {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));
  gap: 20px;
  margin-bottom: 200px;
}
h1, p { grid-column: 1 / -1; }

div { background: #c0f4c7; height: 200px;
      font-size: 24px; text-align: center; }

/* sa(スクロールアニメーション)用のクラス指定 */
.sa          { opacity: 0; transition: all .5s ease; }
.sa.show    { opacity: 1; transform: none; }
.ltr         { transform: translate(-100px, 0); }
.rtl         { transform: translate(100px, 0); }
.up          { transform: translate(0, 100px); }
.down        { transform: translate(0, -100px); }
.scale_up   { transform: scale(.5); }
.scale_down { transform: scale(1.5); }
.rotate_l   { transform: rotate(180deg); }
.rotate_r   { transform: rotate(-180deg); }

```

10.3 JavaScript

```

const scroll_animation_class      = 'sa';
const scroll_animation_show_class = 'show';
const trigger_margin_default     = 300;

let scroll_animation_elements
  = document.querySelectorAll(`.${scroll_animation_class}`);

let scroll_animation_function = function () {
  // 表示タイミング位置を指定
  // 例) data-sa_margin="200"
  // その要素が画面の下から200px上に上がって来たら表示する
  let data_margin  = `${scroll_animation_class}_margin`;
  // 表示タイミングの基準の要素を指定
  // 例) data-sa_trigger="#elem1"
  // #elem1の出現と同時に、その要素も表示させる
  let data_trigger = `${scroll_animation_class}_trigger`;
  // 表示タイミング時間を指定
  // 例) data-sa_delay="500"
  // 500ms 後にその要素を表示する
  let data_delay   = `${scroll_animation_class}_delay`;

  // saクラスの要素を対象に処理を行う
  for(let i = 0; i < scroll_animation_elements.length; i++){
    let elm = scroll_animation_elements[i];

    // もしdata-sa_marginに値があれば、その数値に置き換える
    let trigger_margin = trigger_margin_default;
    if (elm.dataset[data_margin] != null) {
      trigger_margin = parseInt(elm.dataset[data_margin]);
    }
  }
}

```

```
}

// data-sa_trigger属性に値があれば指定の要素、
// 無ければその要素自身の位置を判定基準にする
let show_position = 0;
if (elm.dataset[data_trigger]) {
    show_position
        = document
            .querySelector(elm.dataset[data_trigger])
            .getBoundingClientRect()
            .top + trigger_margin;
} else {
    show_position = elm.getBoundingClientRect().top + trigger_margin;
}

// data-sa_delay があれば、その時間だけずらして表示する。
if (window.innerHeight > show_position) {
    let delay = (elm.dataset[data_delay]) ? elm.dataset[data_delay] : 0;
    setTimeout(function(index) {
        scroll_animation_elements[index].classList
            .add(scroll_animation_show_class);
        }.bind(null, i), delay);
    }
}
}

// イベントリスナー
window.addEventListener('load', scroll_animation_function);
window.addEventListener('scroll', scroll_animation_function);
```

第 11 章

固定フッター

iPhone アプリでよくあるように画面下にメニューを配置します。CSS と JavaScript を使って、実現していきます。

11.1 HTML

注文住宅なら京都市で設計施工を行う工務店 garDEN^{*1} を元に適宜改変いたしました。下の作成例のように、画面下にメニューが配置されます。動作例^{*2}をごらんになってください。また ソースコード^{*3} もございます。

モバイル端末に向いたデザインですので、画面幅によって、CSS / JavaScript の適用を制御する必要がありますが、ここでは簡単化のために、割愛してございます。

▼ index.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
```

*1 <https://gar-den.jp/>

*2 https://wave-improve.netlify.app/fixed_footer/index.html

*3 https://github.com/Atelier-Mirai/wave-improve/tree/master/fixed_footer

```
4 <meta charset="utf-8">
5 <title>Fixed Footer</title>
6 <!-- Font Awesome -->
7 <link rel="stylesheet" href="https://use.fontawesome.com/releases/v6.2.0/css/all.css">
8 >
9 <!-- 簡単に見栄えの良いページを作るためのスタイルシート -->
10 <link rel="stylesheet" href="https://unpkg.com/sakura.css/css/sakura.css">
11 <!-- Fixed Footer -->
12 <link rel="stylesheet" href="fixed_footer.css">
13 </head>
14
15 <body>
16   <h1><a href="index.html">文學堂</a></h1>
17
18   <section>
19     <h2><ruby>文學紹介</h2>
20     <p class="lead text">文學堂では古今の様々な作家による名作を紹介しています。</p>
21
22     <article id="pora-no">
23       <h3>ポーラーの広場 宮沢賢治</h3>
24       <p>そのころわたくしは、モリオ市博物局に勤めて居りました。</p>
25     </article>
26
27     <article id="botchan">
28       <h3>坊ちゃん 夏目漱石</h3>
29       <p>親譲りの無鉄砲で小供の時から損ばかりしている。</p>
30     </article>
31
32     <article id="toshishun">
33       <h3>杜子春 芥川龍之介</h3>
34       <p>或春の日暮です。</p>
35     </article>
36
37     <article id="takasebune">
38       <h3>高瀬舟 森鷗外</h3>
39       <p>高瀬舟は京都の高瀬川を上下する小舟である。</p>
40   </section>
41
42   <footer id="fixed_footer">
43     <nav>
44       <ul>
45         <li><a href="#pora-no">
46           <i class="fa-solid fa-book fa-lg"></i><br>
47           ポーラーの広場</a></li>
48         <li><a href="#botchan">
49           <i class="fa-solid fa-calendar-days fa-lg"></i><br>
50           坊ちゃん</a></li>
51         <li><a href="#toshishun">
52           <i class="fa-solid fa-house-chimney fa-lg"></i><br>
53           杜子春</a></li>
54         <li><a href="#takasebune">
55           <i class="fa-solid fa-ship fa-lg"></i><br>
56           高瀬舟</a></li>
57       </ul>
58       <div class="telephone">
59         <a href="tel:052-373-4649">
60           <i class="fa-solid fa-phone"></i>
```

```

61      電話をかける<small>9:00～18:00 / 日曜祝日休</small>
62      052-373-4649
63    </a>
64  </div>
65 </nav>
66</footer>
67
68<!-- jQuery -->
69<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></scr>
70</script>
71<!-- Fixed Footer -->
72<script src="fixed_footer.js"></script>
73
74<!-- Smooth Scroll -->
75<script src="https://cdn.jsdelivr.net/npm/smooth-scroll@16.1.3/dist/smooth-scroll.polyfills.min.js"></script>
76<script>
77let scroll = new SmoothScroll('a[href*="#"]', { easing: 'easeInOutQuint' });
78</script>
79</body>
79</html>

```

画面下部のボタンをクリックした際に、各文学作品にスクロールするようにします。文学作品の

タグはもう少し必要ですが、紙幅の関係で割愛しています。

作品紹介、メニュー項目を用意し、必要な CSS / JavaScript を読み込むという、分かりやすい HTML と成っています。

11.2 CSS

▼fixed_footer.css

```

1 #fixed_footer {
2   position: fixed;
3   width: 100%;
4   z-index: 1000;
5   transition-property: all;
6   transition-duration: 0.3s;
7   transition-timing-function: ease;
8   transition-delay: 0.3s;
9 }
10
11 /* 最初は電話をかけるを下に隠しておく */
12 #fixed_footer { bottom: -95px; }
13 /* 電話をかけるが上に現れる */
14 #fixed_footer.active { bottom: 0; }
15
16 #fixed_footer.active span {
17   display: inline-block; margin: 6px 0; }
18
19 /* CSS Grid で ボタン項目を並べる */
20 #fixed_footer nav ul {
21   display: grid;
22   grid-template-columns: repeat(4, 1fr);
23   grid-auto-flow: column;

```

```
24
25     overflow: hidden;
26     margin: 0 -1px;
27     padding: 0;
28     list-style: none;
29 }
30
31 #fixed_footer nav ul li a {
32     vertical-align: middle;
33     text-align: center;
34     text-decoration: none;
35     display: block;
36     background: #0d0d0d;
37     padding-top: 12px;
38     color: #fef4f4;
39     font-size: 11px;
40     height: 60px;
41     border-left: 1px solid #fef4f4;
42     position: relative;
43 }
44
45 #fixed_footer nav ul li a span {
46     display: inline-block;
47     margin: 12px 0;
48 }
49
50 #fixed_footer nav ul li a::after {
51     content: "";
52     display: block;
53     border: 15px solid transparent;
54     border-left-color: #ffec47;
55     border-top-color: #ffec47;
56     width: 0;
57     height: 0;
58     position: absolute;
59     left: 0;
60     top: 0;
61 }
62
63 #fixed_footer nav .telephone {
64     background: #0d0d0d;
65     text-align: center;
66     border-top: 1px solid #fef4f4;
67 }
68
69 #fixed_footer nav .telephone a {
70     color: #fef4f4;
71     text-decoration: none;
72     width: 100%;
73     display: inline-block;
74     padding: 10px;
75 }
76
77 #fixed_footer nav .telephone a small {
78     display: block;
79     font-size: 9px;
80 }
```

```

82 /* sakura.css による干渉を補正 */
83 #fixed_footer { width: 100%; left: 0; }
84 li { margin-bottom: 0; }
85 a:hover { border: none; }

```

少し長い CSS で恐縮ですが、大半は形を整えるためのものです。

機能面でのポイントとしては、まず、先頭に書かれた `#fixed_footer { position: fixed; }` が挙げられます。これによりスクロールに関わらず画面下部に固定して表示されます。

また、作成例では隠れていますが、ある程度スクロールすると、「電話をかける」ためのリンクも出現します。それを実現しているのが、11-14 行目です。`active` クラスの有無で位置を調整するようにしています。

19 行目からは、CSS Grid Layout を使って、ボタン項目を並べています。

53-55 行目では、ボタン項目の装飾として黄色の三角形を作成しています。

11.3 JavaScript

▼ fixed_footer.js

```

1 /* 150px以上のスクロールで activeクラスを付与 電話をかける を表示させる
2 -----
3 let fixed_footer = () => {
4   const $fixed_footer = $('#fixed_footer');
5   const trigger_position = 150;
6   let current_position = $(this).scrollTop();
7
8   if (current_position > trigger_position) {
9     $fixed_footer.addClass('active');
10 } else {
11   $fixed_footer.removeClass('active');
12 }
13 }
14
15 /* 画面スクロールやページ読み込みの際に fixed_footer 関数を呼ぶ
16 -----
17 $(window).scroll(() => { fixed_footer(); });
18 $(window).on("load", () => { fixed_footer(); });

```

スクロール量を取得し、`active` クラスを付与するという、簡潔なスクリプトです。

第 12 章

追隨メニュー

画面右にあってスクロールに関わらず、ずっと存在しているメニューです。CSS のみで実現します。

12.1 HTML

注文住宅なら京都市で設計施工を行う工務店 garDEN^{*1} を元に適宜改変いたしました。下の作成例のように、画面下にメニューが配置されます。動作例^{*2}をごらんになってください。また ソースコード^{*3} もございます。

文學堂

文學紹介

文學堂では古今の様々な作家による名作を紹介しています。

ポラーノの広場 宮沢賢治

そのころわたくしは、モリーオ市の博物局に勤めて居りました。

十八等官でしたから役所のなかでも、ずっと下の方でしたし俸給もほんとうに少しかでしたが、受持ちが標本の採集や整理で生れ付き好きなことでしたから、くしは毎日すいぶん愉快にはたらきました。殊にそのころ、モリーオ市で高瀬舟植物園に拵え直すというので、その景色のいいまわりにアカシヤを植えたり地面が、切符売場や信号所の建物のついたまま、わたくしどもの役所の方へまわって

iPad や Mac など比較的大きな画面を持つ端末向けのデザインですので、画面幅によって、CSS の適用を制御する必要がありますが、ここでは簡単化のために、割愛してございます。

▼index.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
```

*1 <https://gar-den.jp/>

*2 https://wave-improve.netlify.app/follow_menu/index.html

*3 https://github.com/Atelier-Mirai/wave-improve/tree/master/follow_menu

```
5 <title>Fixed Footer</title>
6 <!-- Font Awesome -->
7 <link rel="stylesheet" href="https://use.fontawesome.com/releases/v6.2.0/css/all.css">
8 <!-- 簡単に見栄えの良いページを作るためのスタイルシート -->
9 <link rel="stylesheet" href="https://unpkg.com/sakura.css/css/sakura.css">
10 <!-- Follow Menu -->
11 <link rel="stylesheet" href="follow_menu.css">
12 </head>
13
14 <body>
15 <h1><a href="index.html">文學堂</a></h1>
16
17 <section>
18   <h2><ruby>文學紹介</h2>
19   <p class="lead text">文學堂では古今の様々な作家による名作を紹介しています。</p>
20
21   <article id="pora-no">
22     <h3>ポーラーの広場 宮沢賢治</h3>
23     <p>そのころわたくしは、モリオ市博物局に勤めて居りました。</p>
24   </article>
25
26   <article id="botchan">
27     <h3>坊ちゃん 夏目漱石</h3>
28     <p>親譲りの無鉄砲で小供の時から損ばかりしている。</p>
29   </article>
30
31   <article id="toshishun">
32     <h3>杜子春 芥川龍之介</h3>
33     <p>或春の日暮です。</p>
34   </article>
35
36   <article id="takasebune">
37     <h3>高瀬舟 森鷗外</h3>
38     <p>高瀬舟は京都の高瀬川を上下する小舟である。</p>
39   </article>
40 </section>
41
42 <nav class="follow_menu">
43   <ul>
44     <li><a href="#pora-no">
45       <i class="fa-solid fa-book fa-lg"></i><br>
46       ポーラーの広場</a></li>
47     <li><a href="#botchan">
48       <i class="fa-solid fa-calendar-days fa-lg"></i><br>
49       坊ちゃん</a></li>
50     <li><a href="#toshishun">
51       <i class="fa-solid fa-house-chimney fa-lg"></i><br>
52       杜子春</a></li>
53     <li><a href="#takasebune">
54       <i class="fa-solid fa-ship fa-lg"></i><br>
55       高瀬舟</a></li>
56   </ul>
57   <div class="telephone">
58     <a href="tel:052-373-4649">
59       <i class="fa-solid fa-phone"></i>
60       電話をかける<small>9:00～18:00 / 日曜祝日休</small>
61       052-373-4649
62 </div>
```

```

62      </a>
63    </div>
64  </nav>
65
66 <!-- jQuery -->
67 <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></scr>
68 >ipt>
69 <!-- Smooth Scroll -->
70 <script src="https://cdn.jsdelivr.net/npm/smooth-scroll@16.1.3/dist/smooth-scroll.polyfills.min.js"></script>
71 <script>
72   let scroll = new SmoothScroll('a[href*="#"]', { easing: 'easeInOutQuint' });
73 </script>
74 </body>
75 </html>

```

先ほどの「固定フッター」とほぼ同じ HTML です。追随メニューは、フッターではないので`<footer id="fixed_footer">`が無くなりました。そして CSS から扱いやすいよう、`<nav class="follow_menu">` とクラス名を付与しています。

12.2 CSS

▼fixed_footer.css

```

1 .follow_menu {
2   position: fixed;
3   right: 40px;
4   bottom: 40px;
5   z-index: 10000;
6 }
7
8 .follow_menu ul {
9   list-style: none;
10 }
11
12 .follow_menu ul li {
13   display: block;
14   box-sizing: border-box;
15   width: 100px;
16   height: 100px;
17   border-radius: 50%;
18   position: relative;
19 }
20
21 .follow_menu ul li:nth-child(1) {
22   background: #e7609e; border: 1px solid #e7609e; } /* 牡丹色 */
23 .follow_menu ul li:nth-child(2) {
24   background: #67a70c; border: 1px solid #67a70c; } /* 早苗色 */
25 .follow_menu ul li:nth-child(3) {
26   background: #3d6eda; border: 1px solid #3d6eda; } /* 移色 */
27 .follow_menu ul li:nth-child(4) {
28   background: #a61017; border: 1px solid #a61017; } /* 紅葉色 */
29

```

```

30 .follow_menu ul li a {
31   font-size: 1.5rem;
32   text-align: center;
33   text-decoration: none;
34   color: #fef4f4; /* 桜色 */
35   display: block;
36   box-sizing: border-box;
37   padding: 20px 10px;
38   line-height: 20px;
39   position: relative;
40
41   /* sakura.css の干渉補正 */
42   border-bottom: none;
43 }
44
45 .follow_menu ul li a:hover::after {
46   opacity: 1;
47   border-width: 7px;
48 }
49
50 .follow_menu ul li a::after {
51   content: "";
52   display: block;
53   width: 98px;
54   height: 98px;
55   border: 0 solid #fef4f4; /* 桜色 */
56   border-radius: 50%;
57   position: absolute;
58   left: 0;
59   top: 0;
60   box-sizing: border-box;
61   opacity: 0;
62   transition-property: all;
63   transition-duration: 0.3s;
64   transition-timing-function: ease;
65 }
66

```

機能面でのポイントとしては、まず、先頭に書かれた `.follow_menu { position: fixed; }` が挙げられます。これによりスクロールに関わらず位置を固定して表示されます。

21行目の`.follow_menu ul li:nth-child(1)` 以下で、それぞれのボタンに対し、色を設定しています。

15-17行目で、`100px` の円のボタンにしていますが、45-56行目で `98px` と少し小さめの円を用意することで、`:hover` の際に 桜色の枠線が現れるようにしています。

12.3 縦書きにする

せっかくの文学作品ですので、縦書きにしたいと思われる方もいらっしゃると思います。

```

p {
  writing-mode: vertical-rl;
}

```

で、簡単に縦書きにすることができます。

少し前の記事とはなりますが、[日本らしさを表現 CSS で文字の縦書きに挑戦^{*4}](#) により詳しい解説がございますので、ご参考にしていただければと思います。

^{*4} <https://www.webcreatorbox.com/tech/writing-mode>

第 13 章

モバイルメニュー

iPhone のためのモバイルメニューの実装です。CSS と JavaScript を使って実現します。

13.1 HTML

巻末の参考文献「作って学ぶ HTML&CSS モダンコーディング」を元に適宜改変しております。下の作成例のように、画面右上にボタンが配置され、これを押すとメニューが開きます。[動作例^{*1}](#)をごらんになってください。また [ソースコード^{*2}](#) もございます。



▼index.html

```
1 <!DOCTYPE html>
2 <html>
```

*1 https://wave-improve.netlify.app/mobile_menu/index.html

*2 https://github.com/Atelier-Mirai/wave-improve/tree/master/mobile_menu

```
3 <head>
4 <meta charset="utf-8">
5 <title>Mobile Menu</title>
6 <meta name="viewport" content="width=device-width">
7
8 <!-- Font Awesome -->
9 <link rel="stylesheet"
10   href="https://use.fontawesome.com/releases/v6.2.0/css/all.css">
11 <!-- 簡単に見栄えの良いページを作るためのスタイルシート -->
12 <link rel="stylesheet" href="https://unpkg.com/sakura.css/css/sakura.css">
13 <!-- ハンバーガーボタン -->
14 <link rel="stylesheet" href="hamburgers.css">
15 <!-- モバイルメニュー用のスタイルシート -->
16 <link rel="stylesheet" href="mobile_menu.css">
17 </head>
18
19 <body>
20 <header class="header">
21   <h1><a href="index.html">文學堂</a></h1>
22
23   <!-- https://jonsuh.com/hamburgers/ の例 -->
24   <button class="navbtn hamburger hamburger--spin">
25     <span class="hamburger-box">
26       <span class="hamburger-inner"></span>
27     </span>
28   </button>
29
30   <!-- ナビゲーションメニュー -->
31   <nav class="nav">
32     <ul>
33       <li><a href="#pora-no">
34         <i class="fa-solid fa-book fa-lg fa-fw"></i>ポラーノの広場</a></li>
35       <li><a href="#botchan">
36         <i class="fa-solid fa-calendar-days fa-lg fa-fw"></i>坊ちゃん</a></li>
37       <li><a href="#toshishun">
38         <i class="fa-solid fa-house-chimney fa-lg fa-fw"></i>杜子春</a></li>
39       <li><a href="#takasebune">
40         <i class="fa-solid fa-ship fa-lg fa-fw"></i>高瀬舟</a></li>
41     </ul>
42   </nav>
43 </header>
44
45 <!-- 文學紹介 -->
46 <section>
47   <h2>文學紹介</h2>
48   <p class="lead text">文學堂では古今の作家による名作を紹介しています。</p>
49
50   <article id="pora-no">
51     <h3>ポラーノの広場 宮沢賢治</h3>
52     <p>そのころわたくしは、モリオ市博物局に勤めて居りました。</p>
53   </article>
54
55   <article id="botchan">
56     <h3>坊ちゃん 夏目漱石</h3>
57     <p>親譲りの無鉄砲で小供の時から損ばかりしている。</p>
58   </article>
59
60   <article id="toshishun">
```

```

61   <h3>杜子春 芥川龍之介</h3>
62   <p>或春の日暮です。</p>
63 </article>
64
65 <article id="takasebune">
66   <h3>高瀬舟 森鷗外</h3>
67   <p>高瀬舟は京都の高瀬川を上下する小舟である。</p>
68 </article>
69 </section>
70
71 <!-- ハンバーガーボタンを押されたら、ナビゲーションメニューを開く -->
72 <script src="mobile_menu.js"></script>
73 <!-- Smooth Scroll -->
74 <script src="https://cdn.jsdelivr.net/npm/smooth-scroll@16.1.3/dist/smooth-scroll.polyfills.min.js"></script>
75 <script>
76   let scroll = new SmoothScroll('a[href*="#"]', { easing: 'easeInOutQuint' });
77 </script>
78 </body>
79 </html>

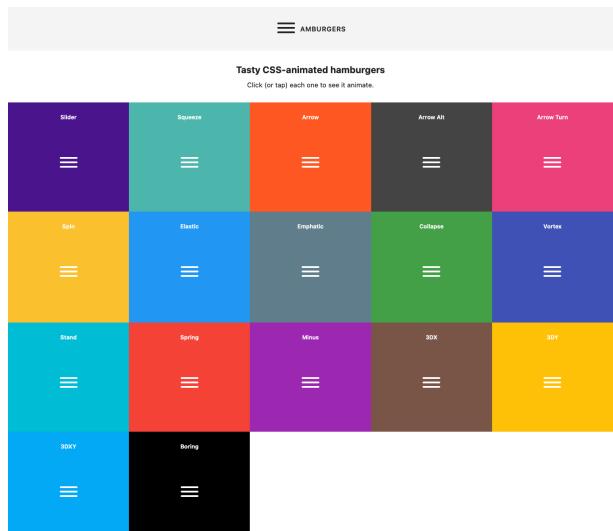
```

先ほどとほぼ同じ HTML です。変更点として<header>にハンバーガーメニューのための<button>を追加しています。

ハンバーガーの作成方法はいろいろありますが、ここでは [Tasty CSS-animated hamburgers^a](#) というサイトで提供されているものを使うことにします。

ボタン開閉効果として十七種類が用意されています。ハンバーガーが回転しながら×印に変化していく spin を使いたいので <button class="hamburger hamburger--spin"> と書いています。

使用法の紹介もなされておりますので好みに合わせてご利用下さい。



^a <https://jonsuh.com/hamburgers/>

13.2 CSS

▼ mobile_menu.css

```

1 /* ヘッダー */
2 .header {

```

```
3  display: grid;
4  grid-template-columns: 1fr 70px;
5  grid-template-rows: 100px;
6 }
7
8 /* ナビゲーションメニュー */
9 .header .nav {
10   position: fixed;
11   inset: 0 -100% 0 100%; /* 画面右外に移動 */
12   transition: transform 0.3s;
13   background: #0d0d0ddd; /* 黒羽色 */
14
15   display: grid;
16   align-items: center;
17 }
18
19 .header .nav ul {
20   list-style: none;
21   padding: 0;
22
23   display: grid;
24   gap: 40px;
25   justify-content: center;
26   align-items: center;
27   text-align: center;
28 }
29
30 .header .nav ul a {
31   font-size: larger;
32   color: #ffec47; /* 菜の花色 */
33 }
34
35 /* メニュー開放時 */
36 .open .navbtn { z-index: 100; }
37 .open .nav { transform: translate(-100%, 0); }
38
39 /* hamburgers.css のハンバーガーボタン色を上書き */
40 .hamburger-inner,
41 .hamburger-inner::before,
42 .hamburger-inner::after {
43   background: #a61017; /* 京緋色 */
44 }
45
46 .hamburger.is-active .hamburger-inner,
47 .hamburger.is-active .hamburger-inner::before,
48 .hamburger.is-active .hamburger-inner::after {
49   background: #2ca9e1; /* 天色 */
50 }
51
52 /* sakura.css の干渉補正 */
53 .navbtn,
54 .navbtn:hover,
55 .navbtn:focus:enabled {
56   background: transparent !important;
57 }
```

ヘッダーとナビゲーションメニューを、CSS グリッドレイアウトを使って、配置します。

注目点は、まず 11 行目の `inset: 0 -100% 0 100%;` です。ナビゲーションメニューを画面右外

へと追いやることで、見えないようにしています。ハンバーガーボタンが押されると JavaScript により、.open クラスが付与されるので、37 行目の .open .nav { transform: translate(-100%, 0); } により、画面右側からメニューが現れる仕組みです。

ナビゲーションメニューの背景色は黒羽色、リンクの色は菜の花色、ハンバーガーボタンの色は京緋色と天色にしています。お好みで変更なさってください。

13.3 JavaScript

▼mobile_menu.js

```
1 // ハンバーガーボタンが押された際に、メニューを表示する
2 document.querySelector('.navbtn.hamburger').addEventListener('click', () => {
3   document.querySelector('html').classList.toggle('open');
4   document.querySelector('.navbtn.hamburger').classList.toggle('is-active');
5 });
6
7 // ナビゲーションメニュー内のリンクがクリックされたときに、メニューを閉じる
8 document.querySelectorAll('nav a').forEach((link) => {
9   link.addEventListener('click', () => {
10     document.querySelector('html').classList.toggle('open');
11     document.querySelector('.navbtn.hamburger').classList.toggle('is-active');
12   });
13 });
```

ハンバーガーボタンが押された際に、`open` クラスを付与することにより、メニューを表示します。`is-active` クラスを付与することにより、ボタンの形状を、ハンバーガーから×へと変化します。

ナビゲーションメニュー内のそれぞれのリンクがクリックされたときに、メニューを閉じるようにします。`open` クラスを取り除き、ボタン形状もハンバーガーへと変化させて、完了です。

第 14 章

画像の絞り込み

沢山の画像から目的のものを抽出する。そういう機能を備えている JavaScript ライブラリとして「Muuri」があります。ここでは、架空のラーメン店の紹介を例に、使い方のご紹介を致します。

14.1 Muuri

Muuri^{*1}公式サイトによると、次のように紹介されています。

Muuri は、レスポンシブで、ソート可能で、フィルタリング可能で、ドラッグ可能なレイアウトを作成します。

最近では、JavaScript を一行も使わずに、かなり素晴らしいレイアウトを構築することができます。しかし、時には CSS だけでは十分でないことがあります。そこで Muuri の登場です。Muuri の核心は、あなたの想像力によってのみ制限されるレイアウトエンジンです。お望みであれば、ウェブワーカーで非同期に、どんな種類のレイアウトでも構築することができます。

アニメーションやインタラクティブ機能（フィルタリング、ソート、ドラッグ＆ドロップ）をレイアウトに散りばめる必要があるかもしれません。これらの追加機能のほとんどは Muuri のコアに組み込まれており、追加のライブラリを探したり、何度も車輪を作り直したりする必要はありません。

Muuri の長期的なゴールは、比類ないパフォーマンスと、複雑さのほとんどを抽象化した、素晴らしいレイアウトを構築するためのシンプルな API を提供することです。（翻訳 DeepL）

Muuri の豊富な機能の中から、絞り込み機能のご紹介を、架空の全国各地の有名ラーメン店の紹介を例に行っていきます。動作例^{*2}やソースコード^{*3}もご活用下さい。

*1 <https://muuri.dev/>

*2 <https://wave-improve.netlify.app/ramen/muuri.html>

*3 <https://github.com/Atelier-Mirai/wave-improve/tree/master/ramen>

ラーメン人気店紹介

全国各地の美味しいラーメン店を紹介します。

気になるラーメン名を入力 全てを表示



[京都市] 櫻幕ラーメン

櫻幕ラーメンは京都市東区にある醤油ラーメンが人気のお店。280円とお財布に優しいお店である。



[京都市] 柳風ラーメン

柳風ラーメンは京都市北区にある味噌ラーメンが人気のお店。280円とお財布に優しいお店である。



[東京都] 凰風ラーメン

凰風ラーメンは東京都西区にある塩ラーメンが人気のお店。380円とお財布に優しいお店である。



[仙台市] 藤鳴ラーメン

藤鳴ラーメンは仙台市北区にある塩ラーメンが人気のお店。580円と良心的な価格設定が嬉しい。



[福岡市] 凰風ラーメン

凰風ラーメンは福岡市東区にある塩ラーメンが人気のお店。980円と高価だが価値ある逸品。



[東京都] 柳風ラーメン

柳風ラーメンは東京都西区にある塩ラーメンが人気のお店。980円と高価だが価値ある逸品。



[福岡市] 菊蓋ラーメン

菊蓋ラーメンは福岡市北区にある醤油ラーメンが人気のお店。280円とお財布に優しいお店である。



[大阪市] 梅鶯ラーメン

梅鶯ラーメンは大阪市南区にある豚骨ラーメンが人気のお店。280円とお財布に優しいお店である。



[京都市] 紅葉ラーメン

紅葉ラーメンは京都市北区にある醤油ラーメンが人気のお店。880円と高価だが価値ある逸品。



[札幌市] 柳風ラーメン

柳風ラーメンは札幌市北区にある塩ラーメンが人気のお店。780円と良心的な価格設定が嬉しい。



[大阪市] 柳風ラーメン

柳風ラーメンは大阪市南区にある塩ラーメンが人気のお店。780円と良心的な価格設定が嬉しい。



[札幌市] 牡丹ラーメン

牡丹ラーメンは札幌市西区にある塩ラーメンが人気のお店。280円とお財布に優しいお店である。



[仙台市] 凰風ラーメン

凰風ラーメンは仙台市南区にある塩ラーメンが人気のお店。680円と良心的な価格設定が嬉しい。



[福岡市] 梅鶯ラーメン

梅鶯ラーメンは福岡市南区にある塩ラーメンが人気のお店。680円と良心的な価格設定が嬉しい。



[福岡市] 凰風ラーメン

凰風ラーメンは福岡市南区にある醤油ラーメンが人気のお店。680円と良心的な価格設定が嬉しい。



[仙台市] 凰風ラーメン

凰風ラーメンは仙台市南区にある醤油ラーメンが人気のお店。380円とお財布に優しいお店である。



[東京都] 茅蒲ラーメン

茅蒲ラーメンは東京都東区にある塩ラーメンが人気のお店。480円とお財布に優しいお店である。



[京都市] 芒月ラーメン

芒月ラーメンは京都市北区にある塩ラーメンが人気のお店。680円と良心的な価格設定が嬉しい。



[東京都] 柳風ラーメン

柳風ラーメンは東京都西区にある醤油ラーメンが人気のお店。780円と良心的な価格設定が嬉しい。



[京都市] 櫻幕ラーメン

櫻幕ラーメンは京都市西区にある豚骨ラーメンが人気のお店。380円とお財布に優しいお店である。



[札幌市] 松鶴ラーメン

松鶴ラーメンは札幌市西区にある豚骨ラーメンが人気のお店。480円とお財布に優しいお店である。



[福岡市] 櫻幕ラーメン

櫻幕ラーメンは福岡市東区にある塩ラーメンが人気のお店。680円と良心的な価格設定が嬉しい。



[東京都] 松鶴ラーメン

松鶴ラーメンは東京都南区にある塩ラーメンが人気のお店。780円と良心的な価格設定が嬉しい。



[仙台市] 萩猪ラーメン

萩猪ラーメンは仙台市南区にある味噌ラーメンが人気のお店。880円と高価だが価値ある逸品。

14.2 HTML

それでは、HTML を書いていきましょう。公式サイトによると以下のように書くと良さそうです。

▼ index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>人気のラーメン店</title>

<!-- 簡単に見栄えの良いページを作るためのスタイルシート -->
<link rel="stylesheet" href="https://unpkg.com/sakura.css/css/sakura.css">
<link rel="stylesheet" href="style.css">
</head>

<body>
<h1>ラーメン人気店紹介</h1>
<p>全国各地の美味しいラーメン店を紹介します。</p>
<hr class="divider">

<!-- 絞り込み -->
<section class="grid-wrapper">
<div class="form-group">
<input type="text"
       id="search-field"
       class="form-input search-field"
       placeholder="所在地やラーメン店名などを入力">
<select class="form-select filter-field">
<option value="">全てを表示</option>
<option value="shoyu">醤油ラーメンを表示</option>
<option value="miso">味噌ラーメンを表示</option>
<option value="shio">塩ラーメンを表示</option>
<option value="tonkotsu">豚骨ラーメンを表示</option>
</select>
</div>

<!-- ラーメン店紹介 -->
<div class="grid">
<div class="item">
<div class="item-content">
<!-- Safe zone, enter your custom markup
This can be anything.
Safe zone ends -->
</div>
</div>
</div>
</section>

<!-- Muuri -->
<script src="https://cdn.jsdelivr.net/npm/muuri@0.9.5/dist/muuri.min.js"></script>
<script src="muuri_filter.js"></script>
</body>
</html>
```

絞り込みのための入力欄を用意し、紹介するラーメン店の画像や説明文、最後に Muuri ライブライ

と絞り込み用のスクリプトを読み込むというシンプルな形です。

中核となるラーメン店の紹介ですが、

```
<div class="item">
  <div class="item-content">
    <!-- Safe zone, enter your custom markup
    This can be anything.
    Safe zone ends -->
  </div>
</div>
```

と書かれているように、内部には好きなHTMLをマークアップできるようすで、お店の紹介として、次のようなマークアップを行うことにしましょう。

```
<div class="item" data-color="shoyu" data-title="櫻幕ラーメンは 京都市東区にある 醤油ラーメンが人気のお店。280円とお財布に優しいお店である。">
  <div class="item-content">
    <div class="card">
      <figure class="card-image">
        
      </figure>
      <h2 class="card-title">
        [京都市] 櫻幕ラーメン
      </h2>
      <p class="card-desc">
        櫻幕ラーメンは 京都市東区にある 醤油ラーメンが人気のお店。280円とお財布に優しいお店である。
      </p>
    </div>
  </div>
</div>
```

`data-color="shoyu"` は、醤油、味噌、塩、豚骨とラーメンのスープの色で絞り込みが行えるようにしていきます。`data-title="櫻幕ラーメンは 京都市東区にある 醤油ラーメンが人気のお店。280円とお財布に優しいお店である。` は、利用者からの文字入力による絞り込みを行うためのもので、「京都市」や「櫻幕ラーメン」などと入力すると、該当するラーメン店が抽出されます。

これを二十四店舗分作成し、

```
<!-- ラーメン店紹介 -->
<div class="grid">
  <div class="item">一軒目のラーメン店</div>
  <div class="item">二軒目のラーメン店</div>
  <div class="item">三軒目のラーメン店</div>
  .
  .
  .
  <div class="item">二十四軒目のラーメン店</div>
</div>
```

とすれば完成ですが、少し大変です。

そこで次のような、Rubyスクリプトで作り上げることと致しましょう。

▼ramen.rb

```

city_list = %w(札幌市 仙台市 東京都 京都市 大阪市 福岡市)
ward_list = %w(北区 東区 南区 西区)
shurui_list = %w(豚骨 醤油 味噌 塩)
name_list = %w(松鶴 梅鶯 櫻幕 藤帰 菖蒲 牡丹 萩猪 芒月 菊盆 紅葉 柳風 凰凰)
price_list = %w(280 380 480 580 680 780 880 980)

(1..24).to_a.each.with_index(1) do |_, index|
  city = city_list.sample
  address = "#{city}#{ward_list.sample}"
  shurui = shurui_list.sample
  shurui_romaji = { 豚骨: "tonkotsu",
                     醤油: "shoyu",
                     味噌: "miso",
                     塩: "shio" }[shurui.to_sym]
  name = name_list.sample
  price = price_list.sample.to_i
  phrase = if price < 500
    "お財布に優しいお店である"
  elsif price < 800
    "良心的な価格設定が嬉しい"
  else
    "高価だが価値ある逸品"
  end
  comment = "#{name}ラーメンは #{address}にある #{shurui}ラーメンが人気のお店。#{price}円と#{phrase}。"
  idx = sprintf("%02d", index)

  html = <<<EOS
  <div class="item" data-color="#{shurui_romaji}" data-title="#{comment}">
    <div class="item-content">
      <div class="card">
        <figure class="card-image">
          
        </figure>
        <h2 class="card-title">
          [#{city}] #{name}ラーメン
        </h2>
        <p class="card-desc">
          #{comment}
        </p>
      </div>
    </div>
  EOS

  puts html
end

```

Ruby は、まつもとひろゆきさんが創られたプログラミング言語で、柔軟性とその優れた書き味からとても人気があります。環境をお持ちでないかたは、 [paiza.io^{*4}](https://paiza.io) から実行できます。
出力結果を貼り付けたら、HTML の完成です。 ^{*5}

^{*4} <https://paiza.io/ja/projects/new>

^{*5} ちなみに店名は花札から取っています。

14.3 CSS

▼ style.css

```
/* sakura.css の補正 */
body { max-width: 90%; }

/* 区切り線 ラーメンらしく雷門の模様に */
.divider { background-image: url("images/raimon_ss.webp");
            height: 20px;
            border: none;
            overflow: none; }

/* 24店舗をグリッドで配置する */
.grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
    gap: 20px;
}

/* ラーメンの種類に応じて枠線を付ける */
div[data-color="shoyu"]     { border: 5px solid #674196; }
div[data-color="miso"]      { border: 5px solid #f19072; }
div[data-color="shio"]       { border: 5px solid #2ca9e1; }
div[data-color="tonkotsu"] { border: 5px solid #ffec47; }

/* Muuri 補正 */
.item { transform: none !important; }

/* ラーメンカードに適宜余白等設定 */
.card .card-image { margin: 0; }
.card .card-image img { width: 100%; height: auto; }
.card h2,
.card p { padding: 0.5em; }
.card h2 { margin: 0; font-size: 1.5em; }
```

特段、凝ったことはしていない普通のスタイルシートです。CSS グリッドレイアウトにより二十四店舗を配置し、ラーメンのスープの色に応じて枠線を付け、種類を分かりやすくしています。^{*6}

14.4 JavaScript

公式サイトによると、次のように書くことで、絞り込みが行えるようです。

▼ muuri_filter.js

```
document.addEventListener('DOMContentLoaded', () => {
    let grid      = null;
    let wrapper   = document.querySelector('.grid-wrapper');
    let searchField = wrapper.querySelector('.search-field');
    let filterField = wrapper.querySelector('.filter-field');
    let gridElem  = wrapper.querySelector('.grid');
```

^{*6} ラーメンの画像は Google 検索により落としてきたものであり、また無作為にデータを作っていることから画像と種類が対応しないものもありますが、それは許容しましょう。

```

let searchAttr  = 'data-title';
let filterAttr  = 'data-color';
let searchFieldValue;
let filterFieldValue;

// Init the grid layout
grid = new Muuri(gridElem, {
  dragEnabled: false
});

grid._settings.layout = {
  horizontal:  false,
  alignRight:  false,
  alignBottom: false,
  fillGaps:    false
};
grid.layout();

// Set initial search query, active filter, active sort value and active layout.
searchFieldValue = searchField.value.toLowerCase();
filterFieldValue = filterField.value;

// Search field event binding
searchField.addEventListener('keyup', () => {
  let newSearch = searchField.value.toLowerCase();
  if (searchFieldValue !== newSearch) {
    searchFieldValue = newSearch;
    filter();
  }
});

// Filtering
const filter = () => {
  filterFieldValue = filterField.value;
  grid.filter( (item) => {
    let element = item.getElement(),
        isSearchMatch = !searchFieldValue ? true : (element.getAttribute(searchAttr) || '').toLowerCase().indexOf(searchFieldValue) > -1,
        isFilterMatch = !filterFieldValue ? true : (element.getAttribute(filterAttr) || '') === filterFieldValue;
    return isSearchMatch && isFilterMatch;
  });
}

// Filter field event binding
filterField.addEventListener('change', filter);
});

```

簡単ではございましたが、Muuri による絞り込み機能のご紹介でした。ご活用いただければ幸いです。

第 15 章

グラフを描画する

様々なグラフを描画したい場面もあるかと思います。定番の JavaScript ライブラリとして「chart.js」があります。ここでは、架空のラーメン店の紹介を例に、使い方のご紹介を致します。

15.1 Chart.js

Chart.js^{*1}公式サイトによると、次のように紹介されています。

シンプル、クリーン、魅力的な HTML5 ベースの JavaScript チャート。Chart.js は、アニメーションやインタラクティブなグラフをあなたの Web サイトに無料で簡単に取り入れることができます。(翻訳 DeepL)

Chart.js は、棒グラフ各種、線グラフ各種、バブル図や円グラフなど、多様なグラフを描画することができます。公式サイトにはサンプルや使い方に関する説明も充実しています。



*1 <https://www.chartjs.org>

ここでは、架空の全国各地の有名ラーメン店の紹介としてレーダーチャートの例を紹介します。動作例^{*2}やソースコード^{*3}もご活用下さい。

ラーメン人気店紹介

全国各地の美味しいラーメン店を紹介します。

	店名 桜幕ラーメン
	住所 京都市東区桜幕1-2-3
	電話 075-222-1234
	営業時間 10:00 - 23:00
	定休日 年中無休

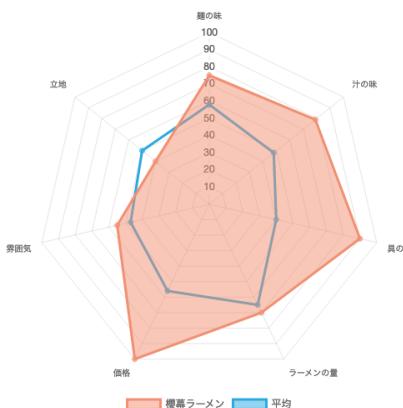
[京都市] 桜幕ラーメン

桜幕ラーメンは 京都市東区にある 醤油ラーメンが人気のお店。280円とお財布に優しいお店である。



京都駅より徒歩40分。
円山公園で営業中です。
ご来店、お待ちいたしております。

総合評価 ★★★★☆



桜幕ラーメンは 京都市東区にある 醤油ラーメンが人気のお店。280円とお財布に優しいお店である。

店名の由来は、京都の桜の名所、円山公園に花見客の紅白幕で賑わうところからと、店主の櫻木薫さんは語る。

国産小麦を丁寧に打ち、醤油をベースに仕立てた汁は滋味がなく澄み渡っており、具材に至っては十年もの研究を経て現在のものが得られたという逸品。腹一杯食べたい方には百円追加で大盛サービスも行っている。価格は創業当時から変わらぬ破格の280円。これで採算が取れるのか心配になるほどであるが、公園内の屋台で営業しているため、家賃がかからぬ分を還元しているとのこと。落ち着いて食べるには少しせわしない雰囲気と京都駅から徒歩40分というアクセスが評価を落としているが、京都に立ち寄ったならば是非食したい逸品である。

「散る桜 残る桜も 散る桜」 良寛和尚の詠んだ名句そのままが伝わる店主との対話を終えて、心に何か温かいものを受け取った筆者であった。

Copyright © 美しいラーメン調査委員会 All rights reserved.

*2 <https://wave-improve.netlify.app/ramen/chart.html>

*3 <https://github.com/Atelier-Mirai/wave-improve/tree/master/ramen>

15.2 HTML

それでは、HTML を書いていきましょう。

▼ index.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>人気のラーメン店</title>

    <!-- Font Awesome -->
    <link href="https://use.fontawesome.com/releases/v6.2.0/css/all.css"
        rel="stylesheet">

    <!-- 簡単に見栄えの良いページを作るためのスタイルシート -->
    <link rel="stylesheet" href="https://unpkg.com/sakura.css/css/sakura.css">
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <h1>ラーメン人気店紹介</h1>
    <p>全国各地の美味しいラーメン店を紹介します。</p>

    <hr class="divider">

    <section>
        <!-- 店舗紹介 -->
        <div class="grid">
            <div class="item" data-color="shoyu" data-title="櫻幕ラーメンは 京都市東区にある 醤油ラーメンが人気のお店。280円とお財布に優しいお店である。">
                <!-- (略) -->
            </div>

            <table>
                <!-- (略) -->
            </table>

            <div class="map">
                <!-- (略) -->
            </div>
        </div>
    </section>

    <hr class="divider">

    <section class="report">
        <h2>総合評価
            <!-- 今すぐ使えるCSSレシピブックより ★を付ける -->
            <div class="rating" data-rate="3.5">
                <span class="star"></span>
                <span class="star"></span>
                <span class="star"></span>
                <span class="star"></span>
                <span class="star"></span>
            </div>
        </h2>
    </section>

```

```
<!-- グラフ描画領域 -->
<div class="chart-container">
  <canvas id="myChart"></canvas>
</div>

<!-- 評価記事 -->
<article class="note">
  <!-- (略) -->
</article>
</section>

<hr class="divider" style="margin-top: 150px;">

<footer>
  <p class="copyright">
    Copyright © 美味しいラーメン調査委員会 All rights reserved.
  </p>
</footer>

<!-- jQuery -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></scr>
<script>
  <!-- Chart.js -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.9.1/chart.min.js"></scr>
<script>
  <script src="chart.js"></script>
</body>
```

ここでポイントは、グラフ描画のために書かれた次のコードです。

```
<!-- グラフ描画領域 -->
<div class="chart-container">
  <canvas id="myChart"></canvas>
</div>

<!-- jQuery -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js"></scr>
<script>
  <!-- Chart.js -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.9.1/chart.min.js"></scr>
<script>
  <script src="chart.js"></script>
```

描画のための `<canvas>` 要素を用意し、画面幅に応じてレスポンシブにグラフの大きさを変更できるよう、親要素として `<div class="chart-container">` を書いています。

Chart.js は jQuery を利用しているので、jQuery と Chart.js を CDN から読み込みます。最後の `chart.js` は、グラフデータや表示設定等が書かれたスクリプトファイルです。

15.3 JavaScript

▼ chart.js

```
// レーダーチャートの説明 (Charts.js 公式)
// https://www.chartjs.org/docs/latest/charts/radar.html

// 色の設定
const colorSet = {
    // 線の色に用いる
    red : '#ff251e',
    orange: '#f19072',
    yellow: '#ffec47',
    green : '#67a70c',
    blue : '#2ca9e1',
    purple: '#674196',
    grey : '#9eala3',

    // 背景色に用いる
    red_a : '#ff251e80',
    orange_a: '#f1907280',
    yellow_a: '#ffec4780',
    green_a : '#67a70c80',
    blue_a : '#2ca9e180',
    purple_a: '#67419680',
    grey_a : '#9eala380',
}

// Setup
const data = {
    labels: ['麵の味', '汁の味', '具の味', 'ラーメンの量', '価格', '雰囲気', '立地'],
    datasets: [
        {
            label: '櫻幕ラーメン',
            data: [75, 79, 90, 70, 100, 55, 40],
            backgroundColor: colorSet.orange_a, // 背景色
            borderColor: colorSet.orange, // 線の色
            fill: true, // 塗りつぶす
        },
        {
            label: '平均',
            data: [58, 48, 40, 65, 56, 47, 50],
            backgroundColor: colorSet.blue_a,
            borderColor: colorSet.blue,
            fill: false,
        }
    ]
};

// Config
const config = {
    type: 'radar', // グラフの種類 レーダーチャート
    data: data, // 上で定義した data オブジェクト読み込み 描画する
    options: {
        plugins: {
            legend: {
                position: 'bottom', // 凡例は図の下に表示
            }
        }
    }
};
```

```
        },
    },
    maintainAspectRatio: false, // 親要素の大きさに合わせ、図を描画する
    elements: {
        line: {
            borderWidth: 3           // 線の太さは 3px
        }
    },
    scales: {
        r: {
            suggestedMin: 0,      // 最小値 0
            suggestedMax: 100     // 最大値 100 の図を描画する
        }
    }
};

// Canvas要素を取得
const ctx = document.getElementById('myChart');
// 取得したCanvas要素に、config に基づき、図を描画する
const myChart = new Chart(ctx, config);
```

レーダーチャートの描画方法について ?? に詳しい説明がございますので、それを参考にしつつ、作成いたしました。

線の色や背景色を定義し、ラーメンに関する各データ系列を準備、グラフの種類等細かい調整を行って描画するコードとなっています。

15.4 ★による評価

ラーメンの味などをレーダーチャートを使って分かりやすく表示することができました。総合評価として、★を使って評価したいと思います。

少し前の書籍ではございますが、「今すぐ使えるCSS レシピブック」に掲載されておりますので、ご紹介いたします。

総合評価 ★★★★☆

```
<div class="rating" data-rate="3.5">
    <span class="star"></span>
    <span class="star"></span>
    <span class="star"></span>
    <span class="star"></span>
    <span class="star"></span>
    <span class="star"></span>
</div>
```

```
.rating { display: inline; }
.star { font-size: 3rem; margin: 0 .05em; }

.star::before {
    content: '\f005';
    color: #ffec47;
    font-weight: 900;
    font-family: 'Font Awesome 6 Free';
```

```
}

.rating[data-rate="0"] .star:nth-child(n+1)::before,
.rating[data-rate="0.5"] .star:nth-child(n+1)::before,
.rating[data-rate="1"] .star:nth-child(n+2)::before,
.rating[data-rate="1.5"] .star:nth-child(n+2)::before,
.rating[data-rate="2"] .star:nth-child(n+3)::before,
.rating[data-rate="2.5"] .star:nth-child(n+3)::before,
.rating[data-rate="3"] .star:nth-child(n+4)::before,
.rating[data-rate="3.5"] .star:nth-child(n+4)::before,
.rating[data-rate="4"] .star:nth-child(n+5)::before,
.rating[data-rate="4.5"] .star:nth-child(n+5)::before {
  color: #9ea1a3;
  font-weight: 400;
}

.rating[data-rate="0.5"] .star:nth-child(1)::before,
.rating[data-rate="1.5"] .star:nth-child(2)::before,
.rating[data-rate="2.5"] .star:nth-child(3)::before,
.rating[data-rate="3.5"] .star:nth-child(4)::before,
.rating[data-rate="4.5"] .star:nth-child(5)::before {
  content: '\f5c0';
  color: #ffec47;
  font-weight: 900;
  font-family: 'Font Awesome 6 Free';
}
```

以上、簡単ではございましたが、Chart.js によるグラフ描画のご紹介でした。ご活用いただければ幸いです。

第 16 章

プログレッシブウェブアプリ (PWA)

プログレッシブウェブアプリ (Progressive web apps, PWA) は、新しいウェブ API と伝統的なプログレッシブな拡張戦略を使用して、クロスプラットフォームのウェブアプリケーションにネイティブアプリと同様の使い勝手をもたらすウェブアプリのことです。



プログレッシブウェブアプリ (Progressive web apps, PWA) と呼ばれる技術があります。ブラウザから見るだけであったウェブサイトを、まるで「アプリ」であるかのように見せられるようになります。

どのようなものか、概要を把握されたい方のために、MDN Web Docs^aにより、引用して、掲載いたします。少し難しいなど感じられた方は、さらっと眺めていただいて実装方法に移っていただきても結構です。

^a https://developer.mozilla.org/ja/docs/Web/Progressive_web_apps

16.1 プログレッシブウェブアプリ とは

プログレッシブウェブアプリ (Progressive web apps, PWA) は、新しいウェブ API と伝統的なプログレッシブな拡張戦略を使用して、クロスプラットフォームのウェブアプリケーションにネイティブアプリと同様の使い勝手をもたらすウェブアプリのことです。

ウェブアプリを PWA と呼ぶには、技術的に言えば、安全なコンテキスト (HTTPS)、1 つ以上のサービスワーカー、マニフェストファイルを持つべきです。

♣ 安全なコンテキスト (HTTPS)

このウェブアプリケーションは、安全なネットワーク上で提供しなければなりません。安全なサイトにすることは、良い習慣であるだけでなく、特にユーザーが安全な取引を行う必要がある場合には、ウェブアプリケーションを信頼できるサイトとして確立することにもつながります。位置情報やサービスワーカーなどの PWA に関連する機能のほとんどは、アプリが HTTPS を使用して読み込まれた場合にのみ利用できます。

♣ サービスワーカー

サービスワーカーとは、ウェブブラウザーがネットワークのリクエストや資産のキャッシュに介入し、その方法を制御することができるスクリプトのことです。サービスワーカーを使用することで、

ウェブ開発者は信頼できる高速なウェブページやオフライン操作を作成することができます。

♣ マニフェストファイル

アプリがユーザーにどのように表示されるかを制御し、プログレッシブウェブアプリを確実に発見できるようにする JSON ファイルです。アプリの名前、開始 URL、アイコン、その他ウェブサイトをアプリのような形式に変換するために必要なすべての詳細が記述されています。

16.2 PWA の利点

♣ 発見可能性

最終的な目的は、ウェブアプリが検索エンジンでより適切に表現され、公開されやすく、カタログ化とランク付けされ、ブラウザにメタデータを使用してそれらに特別な機能を提供することです。

一部の機能は、Open Graph のような独自の技術によって特定のウェブベースのプラットフォームすでにできるようになっています。これは HTML の <head> ブロック内で <meta> タグを使って同様のメタデータを指定するフォーマットを提供しています。

ここで関連するウェブ標準はウェブアプリマニフェスト (Web app manifest) です。これは、名前、アイコン、スプラッシュ画面、テーマカラーなどのアプリの機能を JSON 形式のマニフェストファイルで定義します。これは、アプリ一覧や端末のホーム画面などの場面で使用するためのものです。

♣ インストール可能性

ウェブアプリの使い勝手の中心となるのは、ユーザーがホーム画面にアプリのアイコンを表示し、タップしてアプリを開くことができるネイティブコンテナーであり、基盤となるプラットフォームとうまく統合されていることです。

最近のウェブアプリは、ウェブアプリマニフェストに設定されたプロパティや、最近のスマートフォンブラウザーで利用できるウェブアプリのインストール (en-US) と呼ばれる機能によって、このようなネイティブアプリの感覚を持つことができます。

♣ リンク可能性

ウェブの最も強力な機能の 1 つは、特定の URL でアプリにリンクできることです。アリストアは不要で、複雑なインストールプロセスも不要です。これは今までそうでした。

♣ ネットワーク非依存性

最新のウェブアプリケーションは、ネットワークの信頼性が低い、あるいは存在しない場合でも動作します。ネットワーク非依存性の基本的な考え方は、以下の通りです。

ネットワークが利用できない場合でも、サイトを再訪してそのコンテンツを取得できる。ユーザーが過去に一度でもアクセスしたことのあるコンテンツであれば、接続性が悪い状況でも閲覧できる。接続がない状況で、ユーザーに表示するものを制御することができます。これは、ページリクエストを制御するサービスワーカー (オフラインでの保存など)、ネットワークリクエストに対するレスポンスをオフラインで保存する Cache API (サイトの資産を保存するのにとても便利)、アプリケーションデータをオフラインで保存するウェブストレージや IndexedDB などのクライアントサイドのデータストレージ技術など、さまざまな技術を組み合わせて実現されています。

♣ プログレッシブエンハンスメントの対応

最近のウェブアプリは、十分な機能を備えたブラウザでは優れた使い勝手を、機能の劣るブラウザでは（見劣りするものの）許容できる使い勝手を提供するように開発することができます。私は、プログレッシブエンハンスメントなどのベストプラクティスを用いて、何年も前からこれを行ってきました。プログレッシブエンハンスメントを使用することで、PWA はクロスブラウザーに対応します。つまり開発者は、PWA の一部の機能や技術の実装が、ブラウザの実装ごとに異なることを考慮する必要があります。

♣ 再エンゲージ可能性

ネイティブプラットフォームの大きな利点は、ユーザーがアプリを見ていない時や端末を使用していない時でも、アップデートや新しいコンテンツによってユーザーを簡単に再エンゲージできることです。最近のウェブアプリでは、ページを制御するサービスワーカー、サーバーからサービスワーカーを介してアプリに直接アップデートを送信する Web Push API、システム通知を生成する Notifications API などの新しい技術を使用することで、このようなことが可能になっていますが、ユーザーがウェブブラウザーを積極的に使用していないときにも、ユーザーの関心を引くことができます。

♣ レスポンシブ性

レスポンシブウェブアプリでは、メディアクエリーやビューポートなどの技術を用いて、デスクトップ、モバイル、タブレットなど、あらゆるフォームファクターに対応する UI を実現しています。

♣ 安全性

HTTPS を利用し、セキュリティを考慮してアプリを開発していれば、ウェブプラットフォームは、コンテンツが改ざんされていないことを確認すると同時に、盗み見されることを防ぐ安全な配信メカニズムを提供します。

また、アプリの URL がサイトのドメインと一致するため、ユーザーは正しいアプリをインストールしているかどうかを簡単に確認することができます。これは、アプリストアのアプリとは大きく異なります。アプリストアには似たような名前のアプリがいくつもあり、中には自分のサイトをベースにしたものもあるため、混乱を招くだけです。ウェブアプリは、そのような混乱を解消し、ユーザーに最高の使い勝手を提供します。

16.3 実装方法

技術的概要を掲載いたしましたが、実装方法は容易です。ここでは「アプリ」らしく、「全画面表示」できることを企図して、若干のHTMLとJavaScriptのコードを書き加えましょう。

♣ HTML

HTMLには、`<head>`タグ内に次のように記述します。

```
<!-- PWA (Progressive Web Apps) -->
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="#477294">
<meta name="theme-color" content="#477294">
<meta name="apple-mobile-web-app-title" content="WAVE">
<link rel="manifest" href="/manifest.json">
<script src="app.js"></script>
```

`content="#477294"`は、iPhoneのホーム画面に追加した際の色ですので、お好みの色を指定します。`content="WAVE"`は、ご自身の作成したアプリケーション名です。ここでは、「WAVE」としています。`<link rel="manifest" href="/manifest.json">`は、マニフェスト(宣言書・声明書)ファイルと呼ばれるファイルを読み込みます。例えば次のように書きます。

▼ manifest.json

```
1 {
2     "name": "WAVE",
3     "short_name": "WAVE",
4     "start_url": "/",
5     "background_color": "#ffffff",
6     "theme_color": "#477294",
7     "display": "standalone",
8     "icons": [
9         {
10             "src": "images/icon-512x512.png",
11             "sizes": "512x512",
12             "type": "image/png"
13         }
14     ]
15 }
```

アプリの名称や色、使うアイコンの画像ファイルを指定しています。`"display": "standalone"`と書くことで、ネイティブアプリのようになります。

最後に書かれている、`<script src="app.js"></script>`では、`app.js`ファイルを読み込んでいます。アプリ起動時に、`sw.js`を登録しています。

▼ app.js

```
1 =====
2   サービスワーカーの登録を行う
3   // https://jam25.jp/javascript/about-pwa/
4   // https://laboradian.com/create-offline-site-using-sw/
5 =====
6 if ('serviceWorker' in navigator) {
7   navigator.serviceWorker.register('sw.js')
8     .then(() => (registration) {
```

```

9     // 登録成功
10    console.log(`Service Worker の登録に成功しました。スコープ: ${registration.scope}`);
11  });
12  }).catch((error) => {
13    // 登録失敗
14    console.log(`Service Worker の登録に失敗しました。${error}`);
15  });

```

`sw.js` は、「サービスワーカー」と呼ばれるものです。様々な機能を実装できますが、ここでは単純に電波が届かないときの為のキャッシュのみを行っています。

▼ sw.js

```

1 =====
2 オフラインでも使えるよう、キャッシュを取得し、レスポンスする。
3 // https://jam25.jp/javascript/about-pwa/
4 // https://laboradian.com/create-offline-site-using-sw/
5 =====
6 const CACHE_DYNAMIC_VERSION = 'dynamic-v1';
7 self.addEventListener('fetch', (event) => {
8   console.log('[Service Worker] Fetching something ...');
9   event.respondWith(
10     // キャッシュの存在チェック
11     caches.match(event.request).then((response) => {
12       // キャッシュ内に該当レスポンスがあれば、それを返す
13       if (response) {
14         return response;
15       } else {
16         // キャッシュがなければリクエストを投げて、レスポンスをキャッシュに入れる
17         return fetch(event.request).then((res) => {
18           return caches.open(CACHE_DYNAMIC_VERSION).then((cache) => {
19             // 最後に res を返せるように、ここでは clone() する必要がある
20             cache.put(event.request.url, res.clone());
21             return res;
22           })
23         }).catch(() => {
24           // エラーが発生しても何もしない
25         });
26       }
27     }));
28   });

```

より詳しく解説された記事が以下にございますので、参考にされてください。

- PWA とは？ 実装方法・作り方を企業事例をもとに解説！ ^{*1}
- Service Worker を使ってオフラインでも閲覧できるウェブページを作る方法^{*2}

*1 <https://jam25.jp/javascript/about-pwa/>

*2 <https://laboradian.com/create-offline-site-using-sw/>

付録 A

珠玉の名著のご紹介

学習に役立つ書籍のご案内です。ご参考になれば幸いです。 *1

A.1 HTML / CSS の学習に

♣ CSS グリッドで作る HTML5 & CSS3 レッスンブック



本書は CSS グリッドを基礎にした Web ページ制作を行うための解説書です。CSS グリッドを基礎にすると、Web ページ制作がシンプルになります。サンプルを作りながら一歩一歩着実に学習することにより、モバイルファーストで本格的なレスポンシブに対応した実践的な Web 制作に関する知識がひと通り得られます。

- これから HTML5 & CSS3 を使った Web サイトの構築を学ぶ人
- 最新の CSS グリッドに関する知識を得たい人に最適の一冊です。

♣ 作って学ぶ HTML&CSS モダンコーディング



モバイルファースト&レスポンシブで、サンプルサイトを制作していく過程を実際に操作しながら学びます。サイトはパーツ単位で作成し、章ごとに 1 つのパーツを作成していきます。ヘッダー / ヒーロー / フッター / 記事 / ナビゲーションなど、各パーツの作成にあたっては、パーツのレイアウトを実現する CSS の選択肢を示し、適切なものを選択して作成します。HTML は最新の「HTML Living Standard」に準拠し、CSS では従来から活用してきたメディアクエリの他、Flexbox、CSS Grid などのレイアウトのコントロール、CSS 関数を使いこなします。

*1 紹介文は書籍紹介からの引用・改変です。

♣ HTML5&CSS3 デザイン 現場の新標準ガイド【第2版】



フロントエンドエンジニアはじめ、Web 制作に関わっている人のための HTML5/CSS3 ガイドブックです。HTML と CSS の最新仕様を整理するとともに、主要ブラウザの対応状況など、現時点でのポイントに留意して制作を進めていけばよいか、現場で必要不可欠な情報を解説しています。

HTML / Web ページの作成とメタデータ / コンテンツのマークアップ / CSS の適用 / ボックスのレイアウト / グリッドレイアウト / テーブル / テキスト / エンベディッド・コンテンツ / フォーム / 特殊効果

A.2 JavaScript の学習に

♣ JavaScript Primer 迷わないための入門書



これから JavaScript を学びたい人が、ECMAScript 2015 以降をベースにして一から JavaScript を学べる書籍です。

この書籍は、JavaScript の仕様に対して真剣に向き合って書かれています。入門書であるからといって、極端に省略して不正確な内容を紹介することは避けています。そのため、JavaScript の熟練者であっても、この書籍を読むことで発見があるはずです。

♣ スラスラ読める JavaScript ふりがなプログラミング



「プログラムの読み方をすべて載せる(ふりがなをふる)」という手法で究極のやさしさを目指した、まったく新しい JavaScript(ジャバスクript)の入門書です。本書内に登場するプログラムの読み方をすべて載せ、さらに、漢文訓読の手法を取り入れ、読み下し文を用意。プログラムの1行1行が何を意味していく、どう動くのかが理解できます。

この手法により「プログラムが読めず、自分がいま何をしているか分からぬ」といったプログラミング入門者が挫折する原因を解決しました。

♣ 動く Web デザインアイディア帳 / 動く Web デザインアイディア帳 実践編



本書は Web サイトを動かすことが苦手な右脳系ウェブデザイナーが、サイトを動かす第1歩を踏み出すための「動きの逆引き事典」です。近年のウェブサイトで使用されている基本的な動きの原理や仕組みをサンプルコードと共に紹介します。

付録 B

参考リンク集

B.1 タイピング練習

♣ タイピングクラブ^{*1}

「F」「J」のホームポジションから始まり、数字や記号に至るまで滑らかに入力できるよう練習できるサイト。円滑に文字入力できるよう、毎日こつこつ練習がお勧めです。

B.2 プログラミング学習に

♣ 開発者向けのウェブ技術 - MDN Web Docs より

チュートリアル^{*2} / ウェブ入門^{*3} / HTML の基本^{*4} / CSS の基本^{*5} / ウェブのしくみ^{*6} / JavaScript の基本^{*7}

♣ オンライン学習サイト

ドットインストール^{*8}

すべてのレッスンは3分以内の動画で提供されており、無理なく気軽に学べます。

Progate^{*9}

紙の本よりも直感的で、動画よりも学びやすい、「スライド学習」を採用した学習サイト。自分のペースで学習できること、復習しやすいことが強みです。

paiza^{*10}

1本3分の動画と練習問題で効率的に学ぶ、オンラインでプログラミングしながらスキルアップできる、プログラミング入門学習コンテンツです。

*1 <https://www.typing.com/student/lessons>

*2 <https://developer.mozilla.org/ja/docs/Web/Tutorials>

*3 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web

*4 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web/HTML_basics

*5 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web/CSS_basics

*6 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web/How_the_Web_works

*7 https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web/JavaScript_basics

B.3 写真・イラスト素材

♣ ぱくたそ

人気の写真素材を無料でダウンロード「ぱくたそ」写真は33,119枚使って楽しい、見て楽しい。高解像度で美しい日本の写真が沢山あります。<https://www.pakutaso.com>

♣ プロドットフォト pro.photo

雑誌・出版業界や広告業界などで活躍しているプロカメラマンが撮影した高品質の写真素材を無料のフリー素材として公開。商用利用も可能です。<https://pro-photo.jp>

♣ ベイツ・イメージズ BEIZ images

背景や壁紙に使える画像素材を無料でダウンロード。高画質&高解像度の風景やテクスチャーなど、Web、DTP、動画、アプリ、背景等に使える商用利用可能な素材です。<https://www.beiz.jp>

♣ あしなり 足成

アマチュアカメラマンが撮影した写真を素材として提供。商用利用も全て無料。クレジット表記も不要で、Web、紙、動画等、写真素材として利用可能。<http://www.ashinari.com>

♣ Pixabay

どこでも使える無料のイメージとビデオ。Pixabayは創作に意欲的なコミュニティであり、著作権フリーな画像や映像をお使いいただけます。海外サイトです。<https://pixabay.com/ja/>

♣ 写真 AC

高品質な写真素材が無料でダウンロードできます。加工や商用利用もOK! 登録やダウンロードが一 手間掛かります。<https://www.photo-ac.com>

♣ かわいいフリー素材集 いらすとや

いらすとやは季節のイベント・動物・子供などのかわいいイラストが沢山見つかるフリー素材サイトです。<https://www.irasutoya.com>

♣ イラスト AC

おしゃれでかわいい無料イラスト・人物・フレーム・動物・年賀状などの素材がフリー。AI・EPSやJPEG・PNG形式の画像も無料でダウンロードOK!<https://www.ac-illust.com>

B.4 洗練されたサイト集

♣ WebDesignClip

Webデザインの参考となる品質の高い国内のWebデザイン・クリップ集です。Web制作におけるアイデア・技術に優れたサイトをクリップしています。<https://www.webdesignclip.com>

付録 C

HTML / CSS / JavaScript 簡易まとめ

HTML / CSS / JavaScript に関する簡易なまとめです。タグを使用して作成される HTML 要素を一覧表示しています。考えているものを見つけやすいように、機能別にグループ化しています。

HTML 要素リファレンス^{*1}より、抄訳しております。引用元にはより詳細な解説や使用例等が掲載されておりますので、是非ご活用下さい。

C.1 HTML 簡易まとめ

HTML は Hyper Text Markup Language 超文書印付け言語 の意味で、ウェブサイトにおける文書構造の記述に用います。

♣ コメント

プログラミング言語では、ソースコード中に記述されるがコードとしては解釈されない、人に向けた文字列をコメントといいます。主にコードの記述者が別の開発者などにコードの意味や動作、使い方、注意点等について注釈や説明を加える為に使われます。^{*2}

HTML では、コメントは以下のように記述します。

記述例	説明
<!-- コメント -->	コメント

♣ メインルート

要素	説明
<html>	HTML 文書においてルート（基点）となる要素（トップレベル要素）であり、ルート要素とも呼ばれます。他の全ての要素は、この要素の子孫として配置します。

♣ 文書メタデータ

メタデータは、ページに関する情報のことです。これは検索エンジンやブラウザなどが利用する、お

*1 <https://developer.mozilla.org/ja/docs/Web/HTML/Element>

*2 出典：IT 用語辞典

よりページの描画を支援するスタイル、スクリプト、データといった情報を含みます。スタイルやスクリプトのメタデータはページ内で定義するか、それらの情報を持つ別のファイルへのリンクとして定義します。

要素	説明
<head>	文書に関する機械可読な情報 (metadata)、たとえば題名、スクリプト、スタイルシートなどを含みます。
<link>	外部リソースへのリンク要素です。現在の文書と外部のリソースとの関係を指定します。この要素は CSS へのリンクに最もよく使用されますが、サイトのアイコン (favicon スタイルのアイコンと、モバイル端末のホーム画面やアプリのアイコンの両方) の確立や、その他のことにも使用されます。
<meta>	他のメタ関連要素 (base / link / script / style / title) で表すことができない任意の metadata を提示します。
<style>	文書あるいは文書の一部分のスタイル情報を含みます。
<title>	題名要素です。ブラウザのタイトルバーやページのタブに表示される文書の題名を定義します。

♣ 区分化ルート

要素	説明
<body>	HTML 文書のコンテンツを示す要素で、<body>要素は一つだけ配置できます。

♣ コンテンツ区分

コンテンツ区分要素は、文書のコンテンツを論理的な断片に体系づけます。ページのコンテンツでヘッダーやフッターのナビゲーション、あるいはコンテンツのセクションを識別する見出しなどの、大まかなアウトラインを作成するために区分要素を使用します。

要素	説明
<address>	これを含んでいる HTML が個人、団体、組織の連絡先を提供していることを示します。
<article>	文書、ページ、アプリケーション、サイトなどの内で自己完結しており、(集合したものの中で) 個別に配信や再利用を行うことを意図した構成物を表します。
<aside>	文書のメインコンテンツと間接的な関係しか持っていない文書の部分を表現します。
<footer>	直近の区分コンテンツまたは <body> 要素のフッターを表します。フッターには通常、そのセクションの著作者に関する情報、関連文書へのリンク、著作権情報等を含めます。
<header>	導入部やナビゲーション等のグループを表すコンテンツです。見出し要素だけでなく、ロゴ、検索フォーム、作者名、その他の要素を含むこともできます。

要素	説明
<h1> <h2> <h3> <h4> <h5> <h6>	セクションの見出しを 6 段階で表します。 <h1>が最上位で、<h6>が最下位です。
<main>	文書の <body> の主要な内容を表します。主要な内容とは、文書の中心的な主題、またはアプリケーションの中心的な機能に直接関連または拡張した内容の範囲のことです。
<nav>	現在の文書内の他の部分や他の文書へのナビゲーションリンクを提供するためのセクションを表します。ナビゲーションセクションの一般的な例としてメニュー、目次、索引などがあります。
<section>	文書の自立した一般的なセクション (区間) を表します。そのセクションを表現するより意味的に具体的な要素がない場合に使用します。

♣ テキストコンテンツ

テキストコンテンツ要素は、開始タグ `<body>` と終了タグ `</body>` の間にあるコンテンツでブロックやセクションを編成します。これらの要素はコンテンツの用途や構造を識別するものであり、アクセシビリティ や SEO のために重要です。

要素	説明
<code><div></code>	フローコンテンツの汎用コンテナです。CSS を用いて何らかのスタイル付けがされる（例えば、スタイルが直接適用されたり、親要素にグリッドなど何らかのレイアウトモデルが適用されるなど）までは、コンテンツやレイアウトには影響を与えません。
<code><figure></code>	図表などの自己完結型のコンテンツを表します。任意で <code>figcaption</code> 要素を使用してキャプション（見出し）を付けることができます。
<code><figcaption></code>	親の <code>figure</code> 要素内にあるその他のコンテンツを説明するキャプション（見出し）や凡例を表します。
<code></code>	項目の順序付きリストを表します。ふつうは番号付きのリストとして表示されます。
<code></code>	項目の順序なしリストを表します。一般的に、行頭記号を伴うリストとして描画されます。
<code></code>	リストの項目を表すために用いられます。
<code><p></code>	テキストの段落を表します。
<code><hr></code>	段落レベルの要素間において、テーマの意味的な区切りを表します。例えば、話の場面の切り替えや、節内での話題の転換などです。

♣ インライン文字列意味付け

インラインテキストセマンティクス要素は、単語、行、あるいは任意のテキスト範囲の意味、構造、スタイルを定義します。

要素	説明
<code><a></code>	アンカー要素は、 <code>href</code> 属性を用いて、別のウェブページ、ファイル、メールアドレス、同一ページ内の場所、または他の URL へのハイパーリンクを作成します。
<code>
</code>	文中に改行（キャリッジリターン）を生成します。詩や住所など、行の分割が重要な場合に有用です。
<code></code>	注目付け要素です。要素の内容に読み手の注意を惹きたい場合で、他の特別な重要性が与えられないものに使用します。
<code></code>	強調されたテキストを示します。入れ子にすることができる、入れ子の段階に応じてより強い程度の強調を表すことができます。
<code><i></code>	興味深いテキスト要素です。何らかの理由で他のテキストと区別されるテキストの範囲を表します。
<code></code>	強い重要性要素です。内容の重要性、重大性、または緊急性が高いテキストを表します。ブラウザは一般的に太字で描画します。
<code><small></code>	著作権表示や法的表記のような、注釈や小さく表示される文を表します。既定では、 <code>small</code> から <code>x-small</code> のように、一段階小さいフォントでテキストが表示されます。
<code></code>	記述コンテンツの汎用的な行内コンテナであり、何かを表すものではありません。 <code>class</code> または <code>id</code> 属性を使用して、スタイル付けのために使用することができます。

♣ 画像とマルチメディア

HTML は 画像、音声、映像といった、さまざまなマルチメディアソースをサポートします。

要素	説明
<code></code>	文書に画像を埋め込みます。
<code><audio></code>	文書内に音声コンテンツを埋め込むために使用します。
<code><video></code>	映像要素です。文書中に映像再生に対応するメディアプレイヤを埋め込みます。

♣ SVG と MathML

SVG と MathML のコンテンツを、`<svg>` および `<math>` 要素を使用して直接 HTML 文書に埋め込むことができます。

要素	説明
<code><svg></code>	SVG(Scalable Vector Graphics) 形式の図形描画のための要素です。直線、矩形、円、橢円、多角形、折れ線やベジェ曲線の描画が可能です。
<code><math></code>	数式を記述する際に用います。

♣ スクリプティング

動的なコンテンツやウェブアプリケーションを作成するために、HTML ではスクリプト言語を使用できます。もっとも有名な言語は、JavaScript です。

要素	説明
<code><canvas></code>	<code><canvas></code> 要素 と Canvas スクリプティング API や WebGL API を使用して、グラフィックやアニメーションを描画することができます。
<code><script></code>	実行できるコードやデータを埋め込むために使用します。ふつうは JavaScript のコードの埋め込みや参照に使用されます。
<code><noscript></code>	このページ上のスクリプトの種類に対応していない場合や、スクリプトの実行がブラウザで無効にされている場合に表示する HTML の部分を定義します。

♣ 表 (テーブル)

以下の要素は、表形式のデータを作成および制御するために使用します。

要素	説明
<code><table></code>	表形式のデータ、つまり、行と列の組み合わせによるセルに含まれたデータによる二次元の表で表現される情報です。
<code><caption></code>	表のキャプション（またはタイトル）を指定します。
<code><colgroup></code>	表内の列のグループを定義します。
<code><col></code>	表内の列を定義して、全ての一般セルに共通の意味を定義するために使用します。この要素は通常、 <code>colgroup</code> 要素内にみられます。
<code><thead></code>	表の列の見出しを定義する行のセットを定義します。
<code><tbody></code>	表本体要素 (<code>tbody</code>) は、表の一連の行 (<code>tr</code> 要素) を内包し、その部分が表 (<code>table</code>) の本体部分を構成することを表します。
<code><tr></code>	表内でセルの行を定義します。行のセルは <code>td</code> (データセル) および <code>th</code> (見出しセル) 要素を混在させることができます。
<code><th></code>	表のセルのグループ用のヘッダーであるセルを定義します。
<code><td></code>	表でデータを包含するセルを定義します。これは表モデルに関与します。
<code><tfoot></code>	表の一連の列を総括する行のセットを定義します。

♣ フォーム

利用者がデータを記入してウェブサイトやアプリケーションに送信することを可能にするフォームを作成するために組み合わせて用いる要素です。^{*3}

^{*3} フォームに関する多くの情報が、HTML フォームガイド (<https://developer.mozilla.org/ja/docs/Learn/Forms>) に掲載されています。

要素	説明
<fieldset>	フォーム内のラベル (label) などのようにいくつかのコントロールをグループ化するために使用します。
<legend>	fieldset の内容のキャプション (見出し) を表すために用います。
<form>	サーバに情報を送信するための対話型コントロールを含む文書の区間を表します。
<button>	クリックできるボタンを表し、フォームや、文書で単純なボタン機能が必要なあらゆる場所で使用することができます。
<input>	利用者からデータを受け取るための、フォーム用の対話的なコントロールを作成するために使用します。
<textarea>	複数行のプレーンテキスト編集コントロールを表し、問い合わせフォーム等、利用者が大量の自由記述テキストを入力できるようにするときに便利です。
<label>	ユーザーインターフェイスの項目のキャプション (見出し) を表します。
<datalist>	他のコントロールで利用可能な値を表現する一連の option 要素を含みます。
<select>	選択式のメニューを提供するコントロールを表します。
<optgroup>	select 要素内の、選択肢 (option) のグループを作成します。
<option>	select 要素、optgroup 要素、datalist 要素内で項目を定義するために使われます。
<output>	出力要素 (output) です。サイトやアプリが計算結果やユーザー操作の結果を挿入することができるコンテナ要素です。
<meter>	既知の範囲内のスカラ値、または小数値を表します。
<progress>	タスクの進捗状況を表示します。プログレスバーとしてよく表示されます。

C.2 CSS 簡易まとめ

CSS は、Cascading Style Sheets の略で、ウェブサイトにおける装飾などの為に用います。CSS: カスケーディングスタイルシート^{*4}より、抄訳しています。引用元には詳細な解説や使用例等が掲載されています。是非ご活用下さい。

♣ 構文

カスケーディングスタイルシート (CSS) 言語の基本的な狙いは、ブラウザがページの要素を、色、位置、装飾などの特定の特性をもって描けるようにすることです。その為に、**プロパティ** (人がどのような特性か考えることのできる名前) と、その特性をどのようにブラウザが操作しなければならないか表す**値**の組で表現します。これを**宣言**と呼びます。ページの要素を選択する条件である**セレクタ**により、それぞれの宣言を文書のそれぞれの部品に適用できるようにします。

▼ CSS 構文

```
セレクタ {
  プロパティ1: 値;
  プロパティ2: 値;
  プロパティ3: 値;
}
```

▼ CSS の例

```
header, p.intro {
  background-color: red;
  border-radius: 3px;
```

*4 <https://developer.mozilla.org/ja/docs/Web/CSS>

}

♣ セレクタ

基本セレクタ

全称セレクタ

全ての要素を選択します。任意で、特定の名前空間に限定したり、全ての名前空間を対象にしたりすることができます。

例: * は文書の全ての要素を選択します。

要素型セレクタ

指定されたノード名を持つ全ての要素を選択します。

例: input はあらゆる <input> 要素を選択します。

クラスセレクタ

指定された class 属性を持つ全ての要素を選択します。

例: .index は "index" クラスを持つあらゆる要素を選択します。

ID セレクタ

ID 属性の値に基づいて要素を選択します。文書中に指定された ID を持つ要素は 1 つしかないはずです。

例: #toc は "toc" という ID を持つ要素を選択します。

属性セレクタ

指定された属性を持つ要素を全て選択します。

構 文: [attr] [attr=value] [attr~=value] [attr|=value] [attr^=value]
[attr\$=value] [attr*=value]

例: [autoplay] は autoplay 属性が（どんな値でも）設定されている全ての要素を選択します。

グループ化セレクタ

セレクタリスト

, (カンマ) はグループ化の手段であり、一致する全てのノードを選択します。

例: div, span は と <div> の両要素に一致します。

子孫結合子

半角空白 結合子は、第 1 の要素の子孫にあたるノードを選択します。

例: div span は <div>要素内の要素を全て選択します。

子結合子

> 結合子は、第 1 の要素の直接の子に当たるノードを選択します。

例: ul > li は 要素の内側に直接ネストされた 要素を全て選択します。

一般兄弟結合子

~ 結合子は兄弟を選択します。つまり、第 2 の要素が第 1 の要素の後にあり（直後でなくとも構わない）、両者が同じ親を持つ場合です。

例: p ~ span は <p> 要素の後にある 要素を全て選択します。

隣接兄弟結合子

+ 結合子は隣接する兄弟を選択します。つまり、第 2 の要素が第 1 の要素の直後にあり、両者が同じ親を持つ場合です。

例: `h2 + p` は `<h2>` 要素の後にすぐに続く `<p>` 要素を全て選択します。

擬似表記

擬似クラス

`: 表記`により、文書ツリーに含まれない状態情報によって要素を選択できます。

例: `a:visited` は利用者が訪問済みの `<a>` 要素を全て選択します。

疑似要素

`:: 表記`は、HTML に含まれていない存在(エンティティ)を表現します。

例: `p::first-line` は全ての `<p>` 要素の先頭行を選択します。

コメント

CSS では、コメントは以下のように記述します。

記述例	説明
<code>/* コメント */</code>	コメント

✿ 良く使う CSS プロパティのご案内

CSS には、100 以上ものプロパティがあり、そしてそれぞれのプロパティが取り得る値も個々に決められています。CSS リファレンス^{*5}に全てが紹介されていますので、詳しくはそちらをご覧ください。ここでは、主なもののみを簡単にご紹介します。

要素	説明
<code>background</code>	色、画像、原点と寸法、反復方法など、背景に関する全てのスタイルプロパティを一括で設定します。
<code>border</code>	要素の境界(枠線)を設定します。これは <code>border-width</code> / <code>border-style</code> / <code>border-color</code> の値を設定します。
<code>border-radius</code>	要素の境界(枠線)の外側の角を丸めます。1つの半径を設定すると円の角になり、2つの半径を設定すると楕円の角になります。
<code>box-shadow</code>	要素のフレームの周囲にシャドウ(影)効果を追加します。
<code>color</code>	要素のテキストやテキスト装飾における前景色の色の値を設定します。
<code>display</code>	要素をブロック要素とインライン要素のどちらとして扱うかを設定します。およびその子要素のために使用されるレイアウト、例えば フローレイアウト、グリッド、フレックスなどを設定します。
<code>filter</code>	ぼかしや色変化などのグラフィック効果を要素に適用します。フィルターは画像、背景、境界の描画を調整するためによく使われます。
<code>font-family</code>	選択した要素に対して、フォントファミリ名や総称ファミリ名の優先順位リストを指定します。明朝体、ゴシック体など、書体名を設定します。
<code>font-size</code>	フォントの大きさを定義します。
<code>font-style</code>	通常体(normal)、筆記体(italic)、斜体(oblique)のどれでスタイル付けるかを設定します。
<code>font-weight</code>	フォントの太さ(あるいは重み)を指定します。
<code>height</code>	要素の高さを指定します。
<code>line-height</code>	行ボックスの高さを設定します。これは主にテキストの行間を設定するために使用します。

*5 <https://developer.mozilla.org/ja/docs/Web/CSS/Reference>

要素	説明
<code>list-style-type</code>	リスト項目要素のマーカーを設定します(円、文字、独自のカウンタースタイルなど)。
<code>margin</code>	要素の全四辺のマージン領域を設定します。
<code>max-height</code>	要素の最大高を設定します。
<code>max-width</code>	要素の最大幅を設定します。
<code>object-fit</code>	<code></code> や <code><video></code> などの中身を、コンテナにどのようにめ込むかを設定します。
<code>object-position</code>	<code>object-fit</code> プロパティと併用し、ボックス内における置換要素の配置を指定することができます。
<code>padding</code>	要素の全四辺のパディング領域を一度に設定します。
<code>position</code>	文書内で要素がどのように配置されるかを設定します。
<code>text-align</code>	ブロック要素または表セルボックスの水平方向の配置を設定します。
<code>text-decoration</code>	テキストの装飾的な線の表示を設定します。
<code>text-shadow</code>	テキストに影を追加します。文字列及びその装飾に適用される影のカンマで区切られたリストを受け付けます。
<code>vertical-align</code>	インラインボックス、インラインブロック、表セルボックスの垂直方向の配置を設定します。
<code>width</code>	要素の幅を設定します。
<code>z-index</code>	位置指定要素とその子孫要素、またはフレックスアイテムの z 順を定義します。より大きな <code>z-index</code> を持つ要素はより小さな要素の上に重なります。

グリッドレイアウト関係のプロパティ

要素	説明
<code>grid-template-columns</code>	列の線名と列幅のサイズ変更機能を定義します。
<code>grid-template-rows</code>	行の線名と行高のサイズ変更機能を定義します。
<code>grid-auto-flow</code>	自動配置されたアイテムがどのようにグリッドに流れていくかを指定します。
<code>grid-column</code>	グリッド列の中におけるグリッドアイテムの寸法と位置を指定し、線、区間、なし(自動)をグリッド配置に適用されることで、グリッド領域の列の開始と終了の端を指定します。
<code>grid-row</code>	グリッド行の中におけるグリッドアイテムの寸法と位置を指定し、線、区間、なし(自動)をグリッド配置に適用されることで、グリッド領域の行の開始と終了の端を指定します。
<code>gap</code>	行や列の間のすき間(溝)を定義します。これは <code>row-gap</code> および <code>column-gap</code> の一括指定です。
<code>align-self</code>	グリッド領域内のアイテムの垂直方向の配置を指定します。
<code>justify-self</code>	グリッド領域内のアイテムの水平方向の配置を指定します。

C.3 JavaScript 簡易まとめ

JavaScript は、主にブラウザで用いられるプログラミング言語で、動的なサイトの作成に用います。

*6

♣ コメント

JavaScript は、一行コメントと複数行コメントが用意されています。

コード例	説明
// xxx	一行コメント
/* xxx */	複数行コメント

♣ データ構造

変数とは、コンピュータプログラムのソースコードなどで、データを一時的に記憶しておくための領域に固有の名前を付けたもの。 *7

JavaScript では、定数宣言用の「`const`」と、変数宣言用の「`let`」が用意されています。

コード例	説明
<code>const x</code>	変数宣言。x に値の再代入はできない
<code>let x</code>	変数宣言。const と似ているが、x に値を再代入できる
<code>var x</code>	変数宣言。古い変数宣言方法（今は使わない）

♣ リテラル

リテラル (literal) とは、直値、直定数とも呼ばれ、コンピュータプログラムのソースコードなどの中に、特定のデータ型の値を直に記載したものである。また、そのように値をコードに書き入れるために定められている書式のことという。 *8

コード例	説明
<code>true</code> または <code>false</code>	真偽値
<code>123</code>	10進数の整数リテラル
<code>123n</code>	巨大な整数を表す BigInt リテラル
<code>1_2541_0000</code>	日本の人口など、大きな数は _ で区切ると読みやすくなる
<code>0b10</code>	2進数の整数リテラル
<code>0x30A2</code>	16進数の整数リテラル
<code>[x, y]</code>	x と y を初期値にもつ配列オブジェクトを作成
<code>{ k: v }</code>	プロパティ名が k、 プロパティの値が v のオブジェクト（連想配列）を作成

*6 出典：JavaScript Primer 迷わないとめる入門書 (<https://jsprimer.net>) など

♣ 文字列

文字列とは、文字を並べたもの。コンピュータ上では、数値など他の形式のデータと区別して、文字の並びを表すデータを文字列といふ。^{*9}

コード例	説明
"xxx"	ダブルクオートの 文字列リテラル 。
'xxx'	シングルクオートの 文字列リテラル 。
`xxx`	テンプレート文字列リテラル。改行を含んだ入力が可能
`\${x}`	テンプレート文字列リテラル中の変数 x の値を展開する

♣ 演算子

演算子とは、数学やプログラミングなどで式を記述する際に用いる、演算内容を表す記号などのこと。様々な演算子が定義されており、これを組み合わせて式や命令文を構成する。^{*10}

以下の表は優先順位の最も高いもの(21)から最も低いもの(1)の順に並べられている。^{*11}

優先順位	演算子の種類	結合性	演算子	優先順位	演算子の種類	結合性	演算子
21	グループ化	n/a	(…)	12	小なり	左から右	… < …
20	メンバへのアクセス	左から右	… . …		小なりイコール		… <= …
	計算値によるメンバへのアクセス	左から右	… […]		大なり		… > …
	new (引数リスト付き)	なし	new … (…)		大なりイコール		… >= …
	関数呼び出し	左から右	… (…)		in		… in …
	オプショナルチェイニング	左から右	?.		instanceof		… instanceof …
19	new (引数リストなし)	右から左	new …	11	等価	左から右	… == …
18	後置インクリメント	なし	… ++		不等価		… != …
	後置デクリメント		… --		厳密等価		… === …
17	論理 NOT	右から左	! …		厳密不等価		… !== …
	ビットごとの NOT		~ …		10 ビット単位 AND	左から右	… & …
	単項 +		+ …		9 ビット単位 XOR	左から右	… ^ …
	単項 -		- …		8 ビット単位 OR	左から右	… …
	前置インクリメント		++ …		7 論理 AND	左から右	… && …
	前置デクリメント		-- …		6 論理 OR	左から右	… …
	typeof		typeof …		5 Null 合体	左から右	… ?? …
	void		void …		4 条件	右から左	… ? … : …
	delete		delete …				… = …
	await		await …				… += …
16	べき乗	右から左	… ** …	3 代入			… -= …
15	乗算	左から右	… * …				… **= …
	除算		… / …				… *= …
	剰余		… % …				… /= …
14	加算	左から右	… + …				… %= …
	減算		… - …				… <= …
	左ビットシフト		… << …				… >= …
13	右ビットシフト	左から右	… >> …				… >>= …
	符号なし		… >>> …				… &= …
	右ビットシフト						… ^= …

*11 出典: 演算子の優先順位 (https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Operators/Operator_Precendence)

♣ 制御構造

プログラムの流れを制御するための構文です。繰り返しのための「**for 文**」、条件分岐のための「**if 文**」などが用意されています。

例	説明
<code>while(x){}</code>	while ループ。 x が true なら反復処理を行う。 繰り返し回数が不明な際に用いると効果的
<code>for(let x=0;x < y ;x++){}</code>	for ループ。 x < y が true なら反復処理を行う。 繰り返し回数が分かる時に使うと効果的
<code>for(const p in o){}</code>	for...in ループ。 オブジェクト (o) のプロパティ (p) に対して反復処理を行う
<code>for(const x of iter){}</code>	for...of ループ。 イテレータ (iter) の反復処理を行う
<code>if(x){/*A*/*}else{/*B*/}</code>	条件式。 x が true なら A の処理を、 それ以外なら B の処理を行う
<code>switch(x){case "A":{/*A*/*} "B":{/*B*/}}</code>	switch 文。 x が "A" なら A の処理を、 "B" なら B の処理を行う
<code>x ? A: B</code>	条件 (三項) 演算子。 x が true なら A の処理を、 それ以外なら B の処理を行う
<code>break</code>	break 文。 現在の反復処理を終了しループから抜け出す。
<code>continue</code>	continue 文。 現在の反復処理を終了し次のループに行く。
<code>try{}catch(e){}finally{}</code>	try...catch 構文
<code>throw new Error("xxx")</code>	throw 文

♣ データアクセス

プログラミング言語 Pascal の開発者 ニクラウス・ヴィルト氏による、「プログラミング」 = 「データ構造」 + 「アルゴリズム」は、広く知られています。

配列とオブジェクト (≡連想配列) という主要なデータ構造にアクセスするために、次の構文が用意されています。

コード例	説明
<code>array[0]</code>	配列へのインデックスアクセス
<code>obj["x"]</code>	オブジェクトへのプロパティアクセス (ブラケット記法)
<code>obj.x</code>	オブジェクトへのプロパティアクセス (ドット記法)

♣ 関数宣言

関数とは、コンピュータプログラム上で定義されるサブルーチンの一種で、数学の関数のように与えられた値 (引数) を元に何らかの計算や処理を行い、結果を呼び出し元に返すもののこと。^{*12}

サンプル	説明
<code>function f(){}</code>	関数宣言
<code>const f = function(){};</code>	関数式
<code>const f = () => {};</code>	Arrow Function の宣言
<code>function f(x, y){}</code>	関数における仮引数の宣言
<code>function f(x = 1, y = 2){}</code>	デフォルト引数、引数が渡されていない場合の初期値を指定する。
<code>clasX{}</code>	クラス宣言
<code>const X = clasX{};</code>	クラス式

♣ モジュール

大きなプログラムを作る際、小さな部品（モジュール）を組み合わせて作ると、管理しやすく、部品の再利用もできるので便利です。JavaScript にも、特定のファイルで定義した関数を、他のファイルでも使えるようにする仕組みが用意されています。

コード	説明
<code>import x from "./x.js"</code>	デフォルトインポート
<code>import { x } from "./x.js"</code>	名前付きインポート
<code>export default x</code>	デフォルトエクスポート
<code>export { x }</code>	名前付きエクスポート

♣ その他

コード	説明
<code>x;</code>	文
<code>{ }</code>	ブロック文

【コラム】金の延棒クイズ 【解答】

最後までお読みください、ありがとうございます。金の延棒クイズの解答です。

2回鉄を入れて、金の延棒を1と2と4の大きさに分割します。

一日目のお支払いには、1の延棒を渡します。

二日目のお支払いには、2の延棒を渡して、先に渡した1の延棒は返してもらいます。

三日目のお支払いには、1の延棒も渡します。

四日目のお支払いには、大きな4の延棒を渡し、2と1の延棒は返してもらいます。

五日目のお支払いには、1の延棒も渡します。

六日目のお支払いには、2の延棒を渡して、先に渡した1の延棒は返してもらいます。

七日目のお支払いには、全ての延棒を渡します。

延棒の有無を0と1で表すと二進数と対応しています。

意外なところに潜む二進数。探してみてくださいね。

金の延棒	日当
421	
001	1
010	2
011	3
100	4
101	5
110	6
111	7

終わりに

本書では、少し豊かなウェブサイトを目指して、いろいろご紹介いたしましたが、いかがでしたでしょうか？ コード部分が多くなってしまい、もう少し解説できると良かったのですが、お役立て頂ければとても仕合せです。

「福祉」。「福」「祉^{*13}」どちらも「めぐみ、さいわい」という意味を持ちます。

「熱き心、^{たくま}逞^{かいな}しき腕、冷静な頭脳」

学生時代に言われた言葉ですが、福祉を生きる者は、人としての熱い思い、暖かい心を持ち、その上で、冷静な判断力を以て、力強く行動するのだと。

「工学」の「工」は、「天の 理^{もつ}」を、地に下ろす」意味です。

技術の産物としての社会ではなく、世界を^{かがや}耀^{かがや}かせるために技術を用いてください。技術に使われるのではなく、技術を使いこなし、人の道に役立てる人となってください。

令和の御世を生きる皆さんのが素晴らしい人生を生き、素晴らしい日本を創ることを願って筆を置きます。

いやきか
彌榮

*13 「祉い」と書いて、「さいわい」と読みます。天からの恵みがその身に止まる意味です。

JavaScript で創る豊かなウェブサイト

令和四年九月廿三日 ver 3.0

著 者 アトリエ未来

発行者 早乙女 遙香

連絡先 contact@atelier-mirai.net

© 令和四年 アトリエ未来