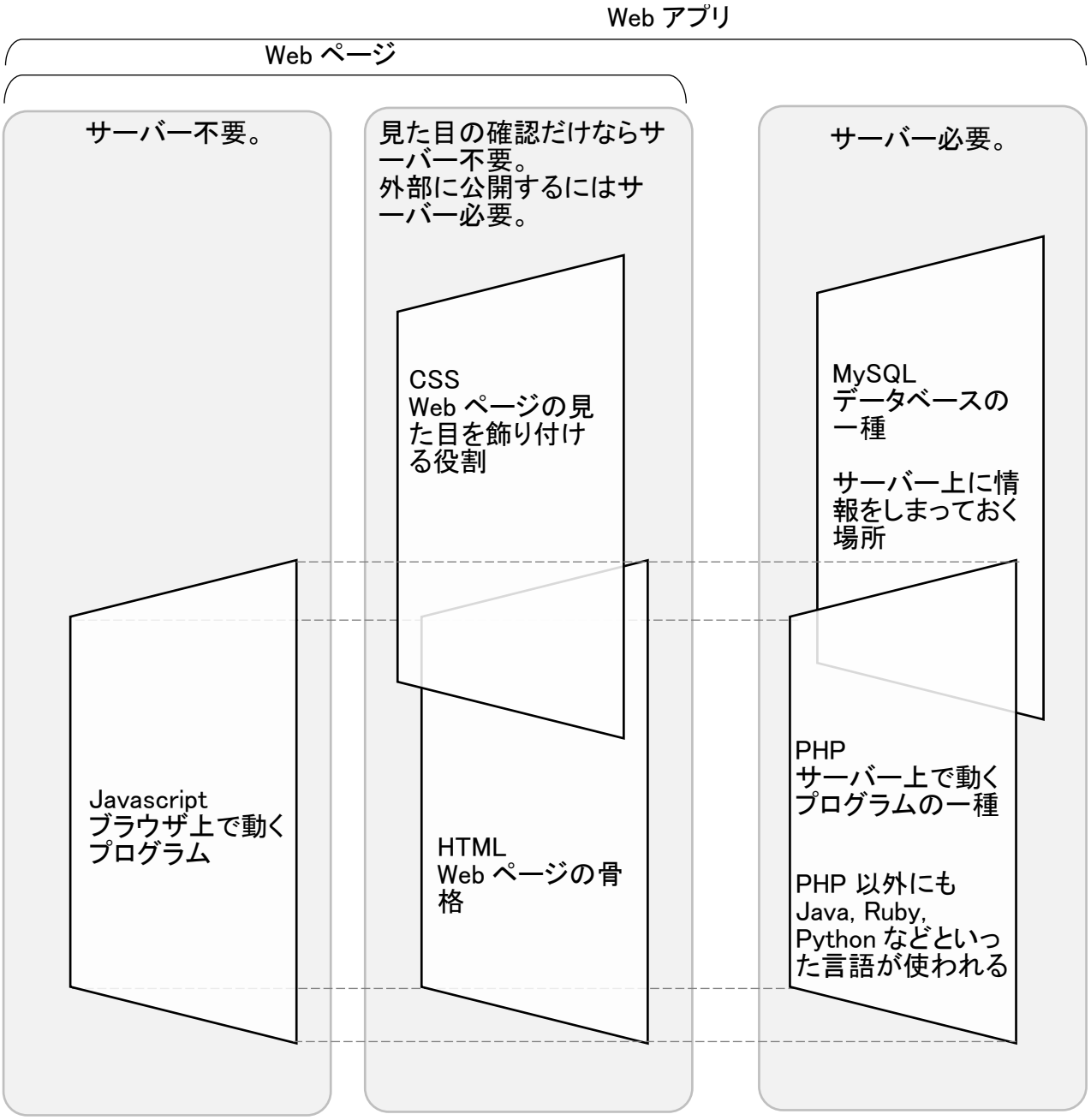
目次

序章　概要と準備

Web ページ・Web アプリの仕組み



動的な見た目などが欲しい場合は必要。
（例）
・入力のチェック
・マウスを当てたら色が変わる・絵が変わる
・時間などのカウントダウン

静的なページはこれだけで作れる。

情報を入力してもらい、それに基づいて情報処理して結果を返すようなシステムには不可欠。
（例）
・予約システム
・ショッピングカート
・SNS
などオンラインアプリ全般

本テキストで取り扱う内容

JavaScript は Web サイトに動きを持たせることができるプログラミング言語です。今ではその用途が Web 以外の用途にも広がりを見せています。

JavaScript を学ぶために必要なものは「Web ブラウザ」と「コードエディタ」の２つのみです。

●Web ブラウザ

Chrome、Edge、Safari など、Web サイト閲覧に使うソフトウェア。
本テキストでは Chrome を使って説明をします。

●コードエディタ

プログラムを書くために使うソフトウェア。
文字を打てて保存できれば良いのでテキストエディタ（Windows に標準で入っている「メモ帳」など）でも問題ないですが、コードエディタだとコードを補完してくれるような便利機能が搭載されているのでおすすめです。

本テキストでは Visual Studio Code を使って説明します。
Visual Studio Code（以下 VSCode と略します）には、インストールして使うタイプのものとインストール不要の Web 版（https://vscode.dev にアクセスして使う）タイプがあります。

Web ブラウザもコードエディタも無料で使えます。
例えば、Chrome と VSCode は Windows でも Mac でも使えます。

VSCode のインストールについては次節で説明します。

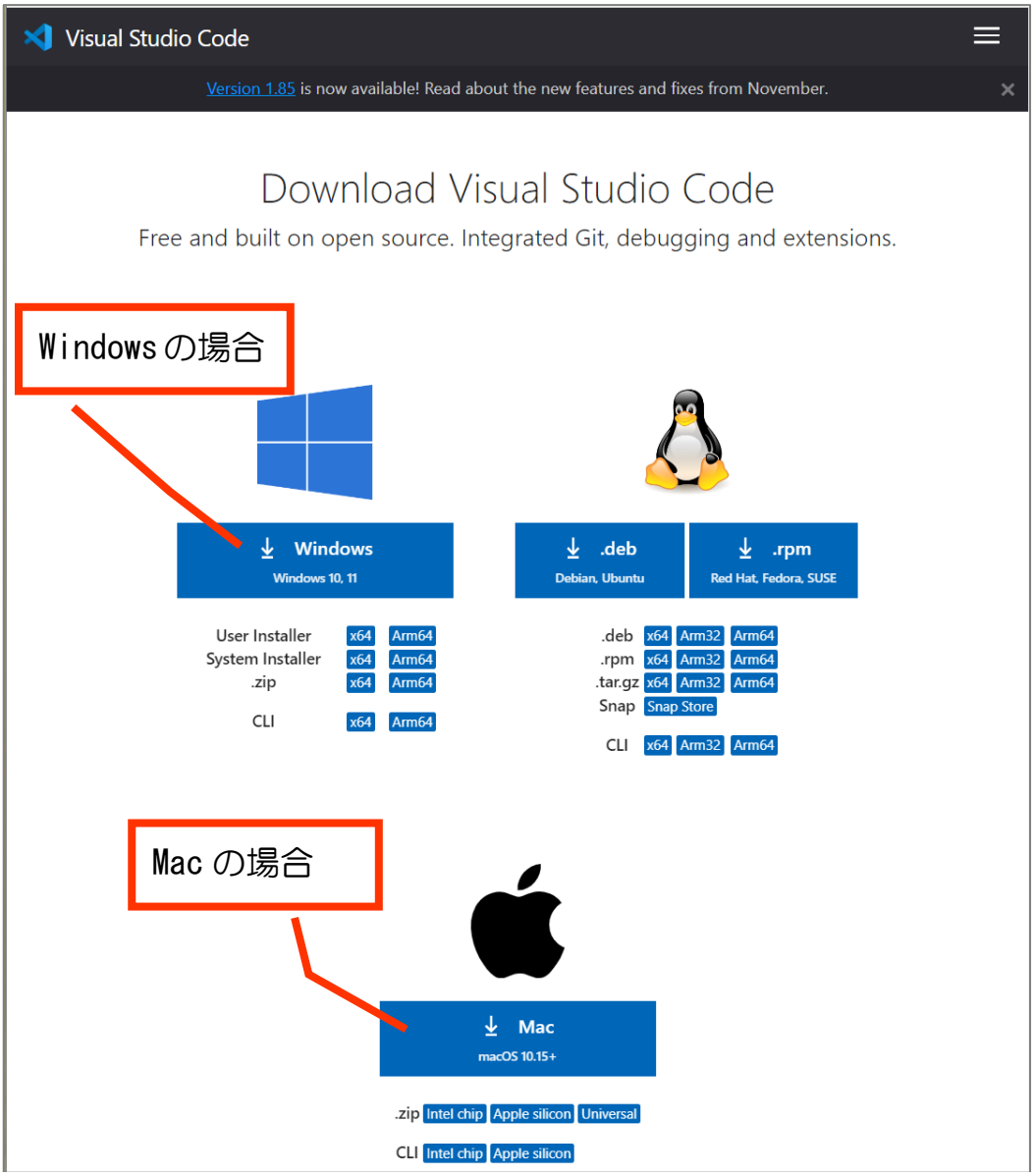
一般的にプログラミング言語を学習するためには各プログラム言語を扱うための環境構築が必要です。ですが、JavaScript は環境構築の必要がないため、初めてプログラミング言語を学習する人にとってとてもハードルが低く、手軽に始められます。

序章－１　コードエディタ　Visual Studio Code の入手とインストール

序章－１－１　パソコン本体にインストールする場合

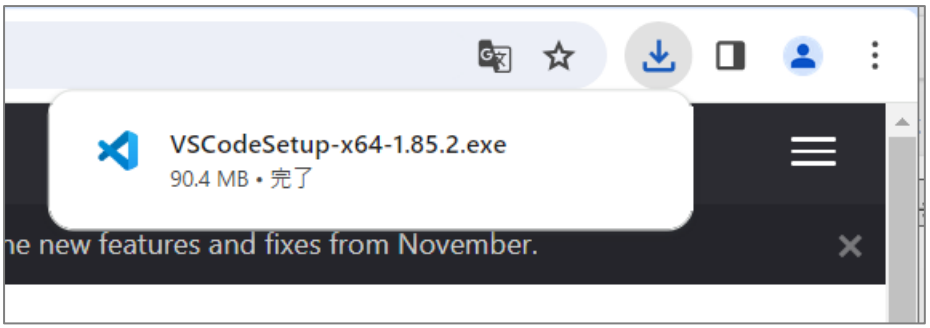
Visual Studio Code のダウンロードページに接続してください。
https://code.visualstudio.com/download

各環境（OS）に合ったものをダウンロードしてください。



※ Web サイトの表示やインストール画面については執筆時のものですので、本テキストとは異なる表記になる場合もあります。インストーラをダウンロードして入手してからインストールするという大きな流れは変わりませんので、表記が少々異なっても推測しながら読み替えて進めてください。

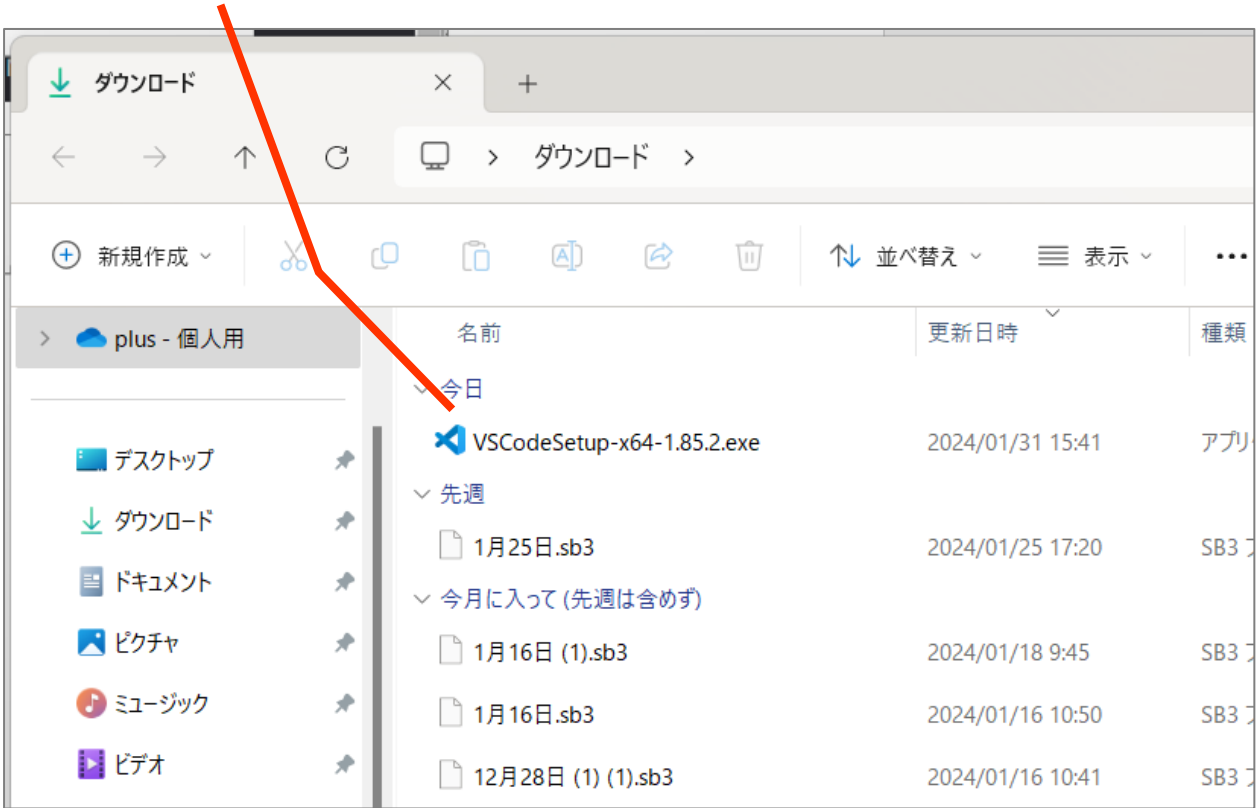
ダウンロードが進みます。完了したら、ダウンロードフォルダを開いてください。



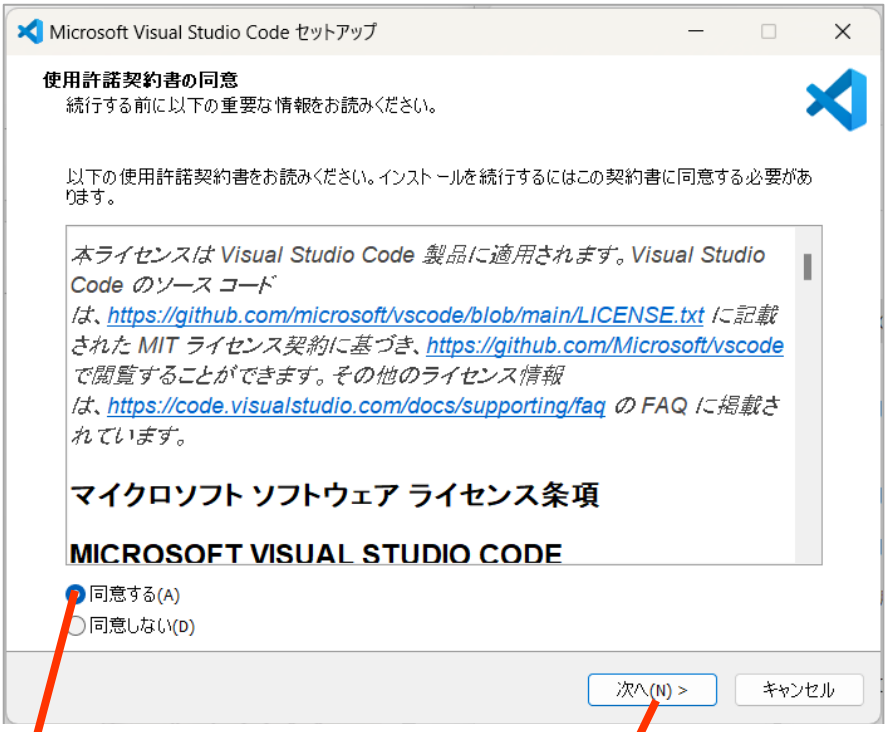
以下、Windows 環境の場合で説明します。

「VSCodeSetup-x64-xxxxx.exe」をダブルクリックしてください。インストーラが起動します。

「xxxxx」の部分はバージョン番号なので、入手時期によって異なります。



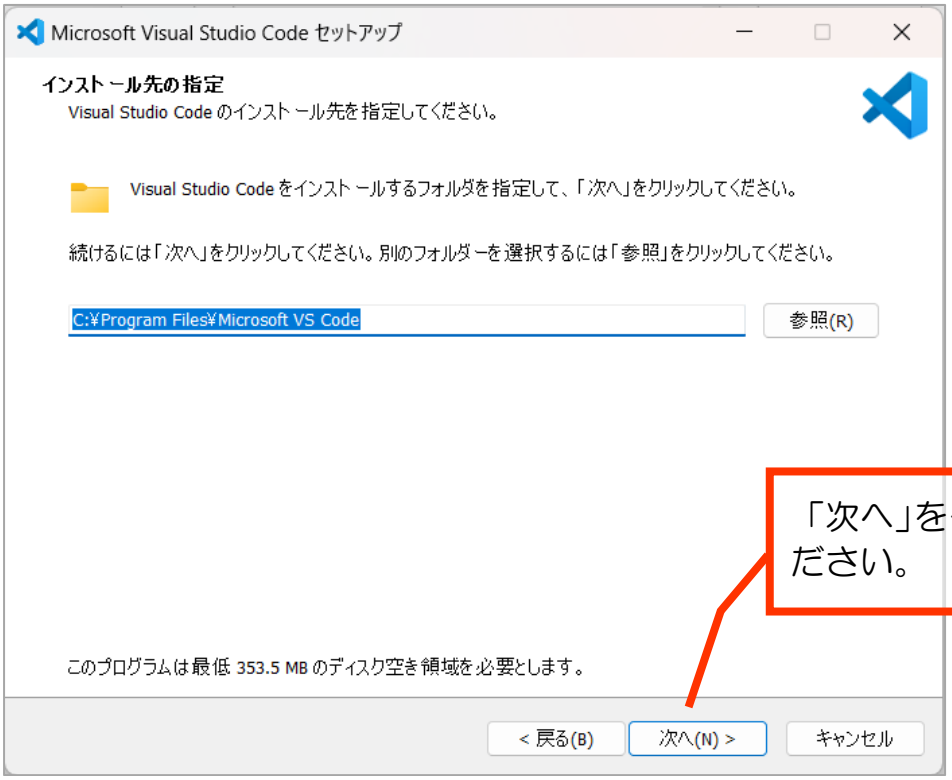
ユーザーアカウント制御のウィンドウが出たら、「はい」をクリックして進んでください。



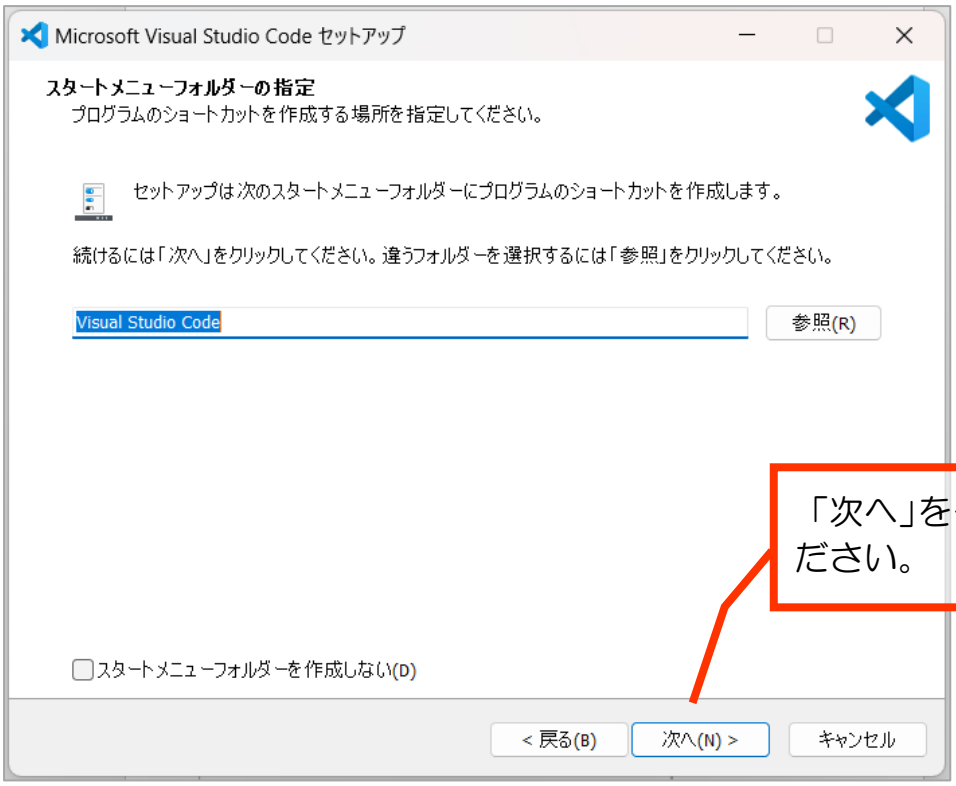
「同意する」をチェックしてください。

「次へ」をクリックしてください。

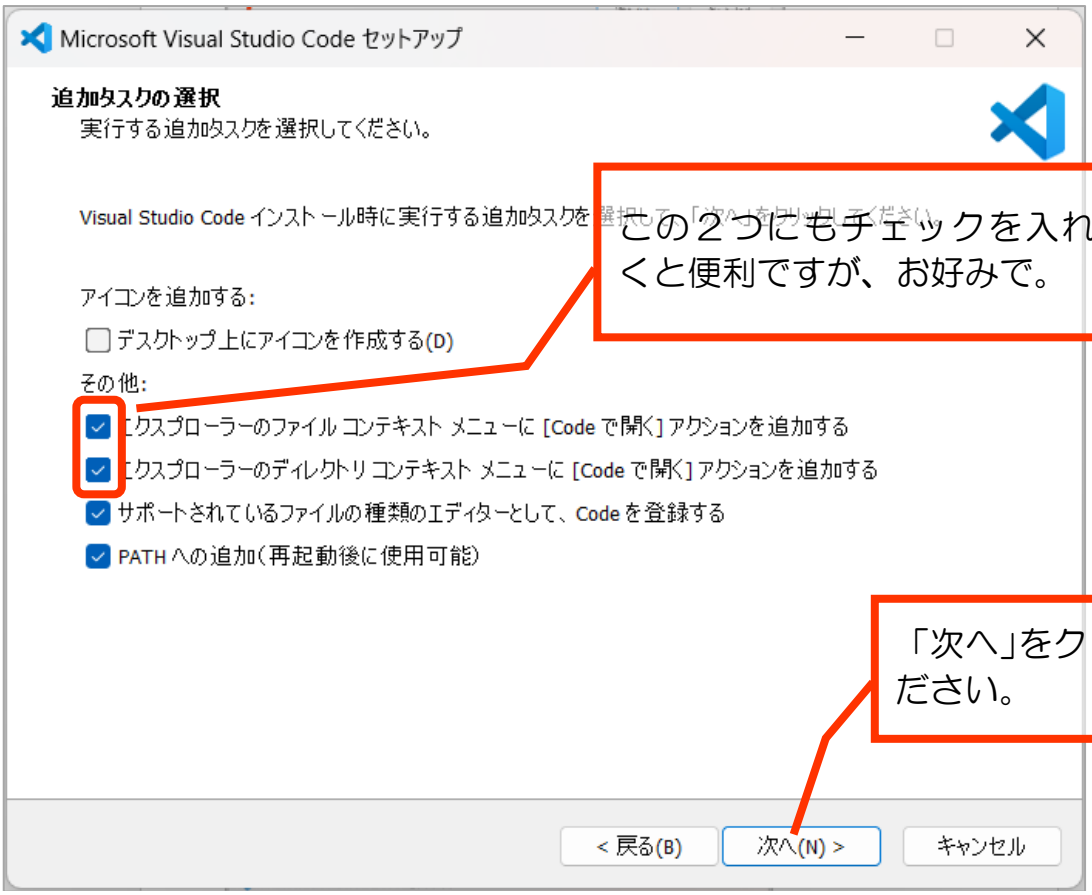
インストール先指定のウィンドウでは、そのまま「次へ」をクリックして下さい。



「次へ」をクリックしてください。

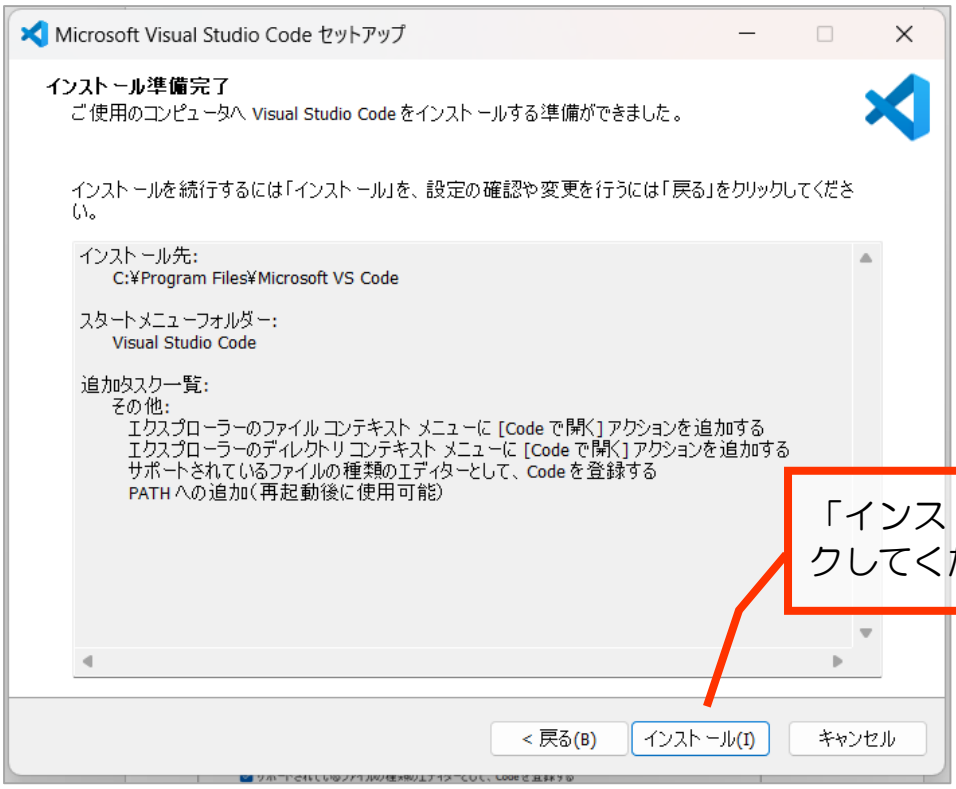


「次へ」をクリックしてください。



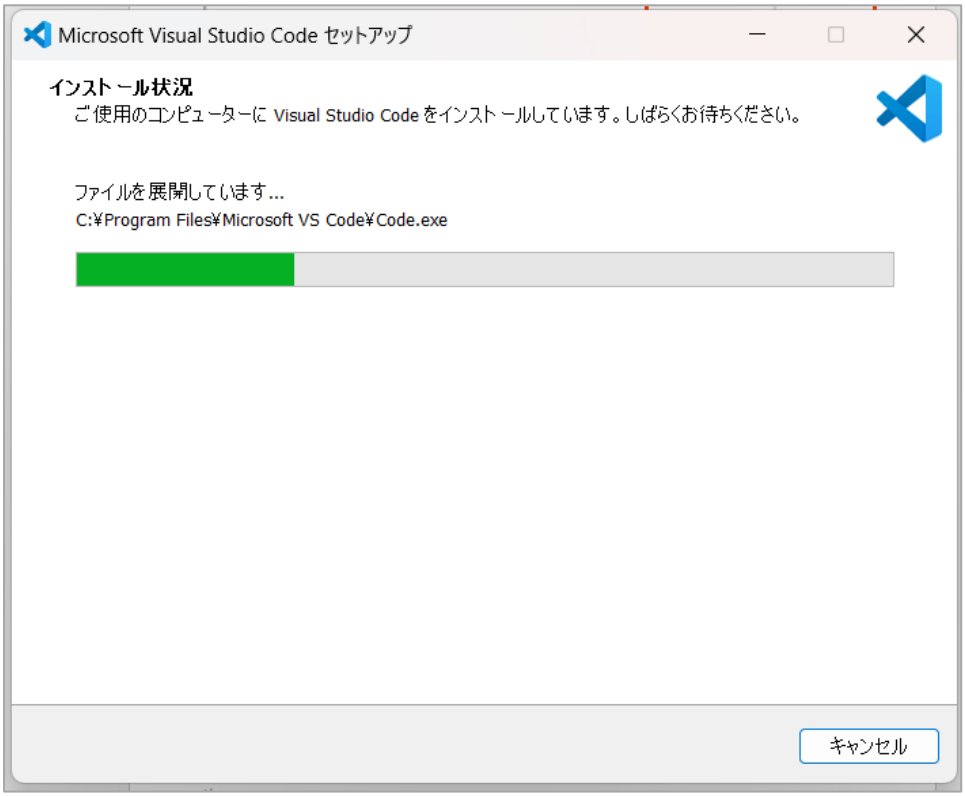
この２つにもチェックを入れておくと便利ですが、お好みで。

「次へ」をクリックしてください。

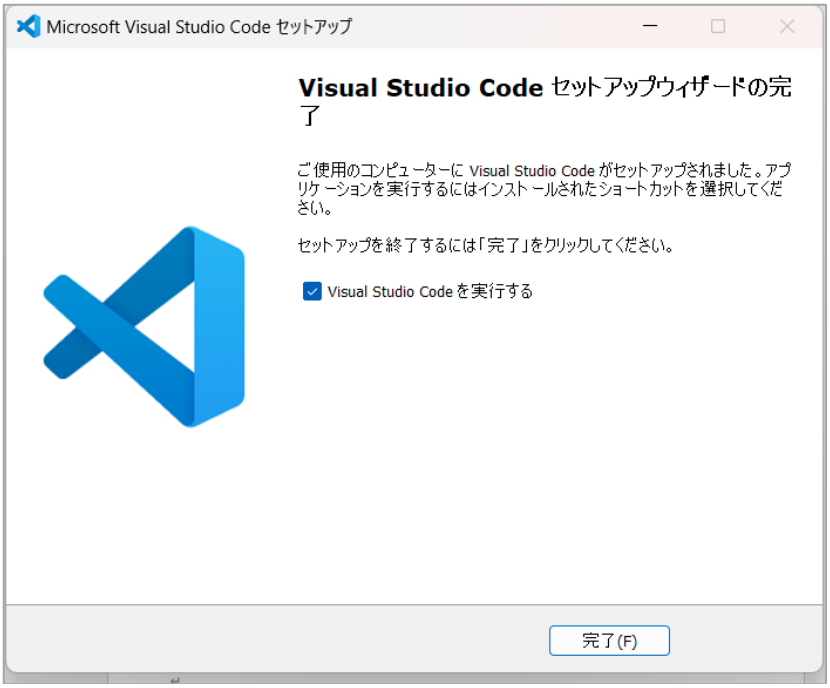


「インストール」をクリックしてください。

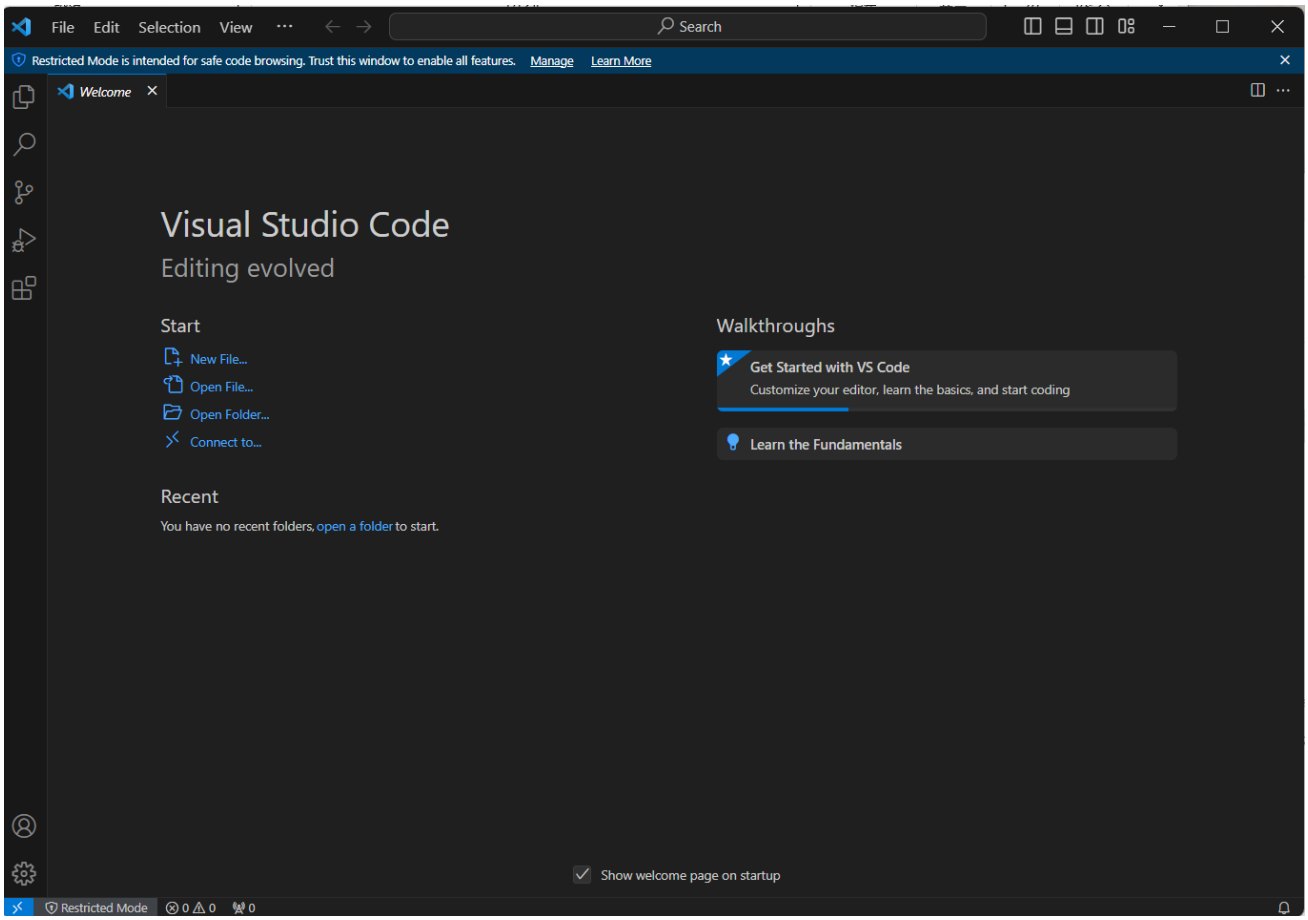
インストールが進みますので、待ちましょう。



インストールが完了しました。「完了」をクリックしてください。



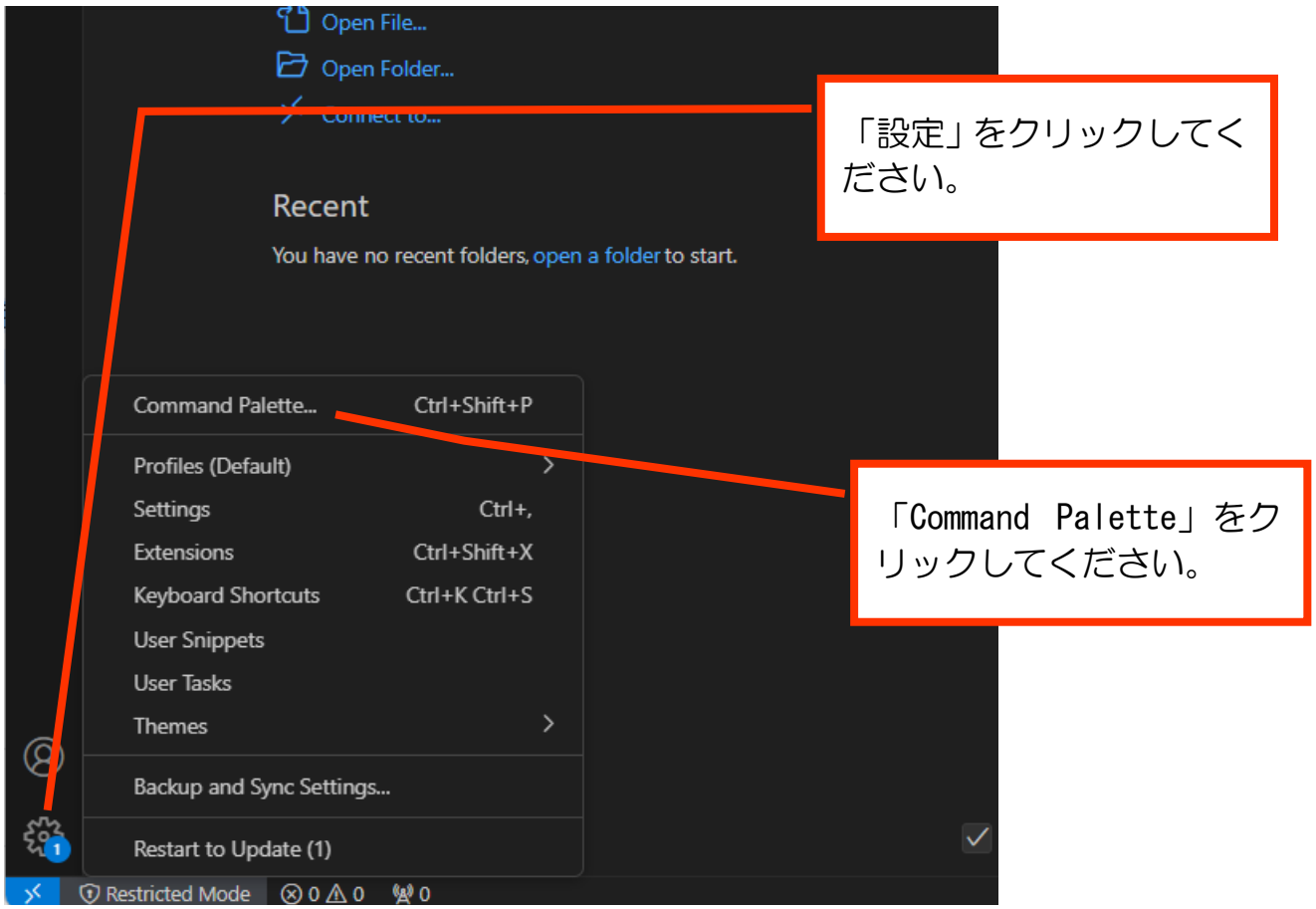
Visual Studio Code が起動し、次のようなウィンドウが現れます。



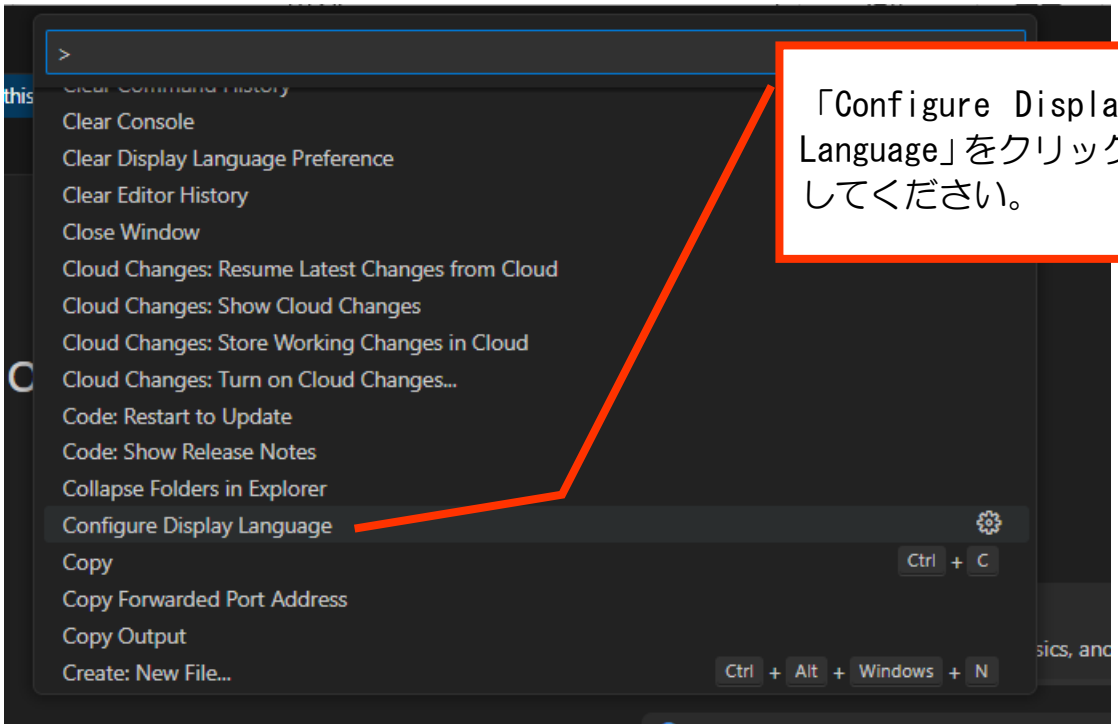
メニューを日本語化したい場合は、１－２－３へ進んでください。

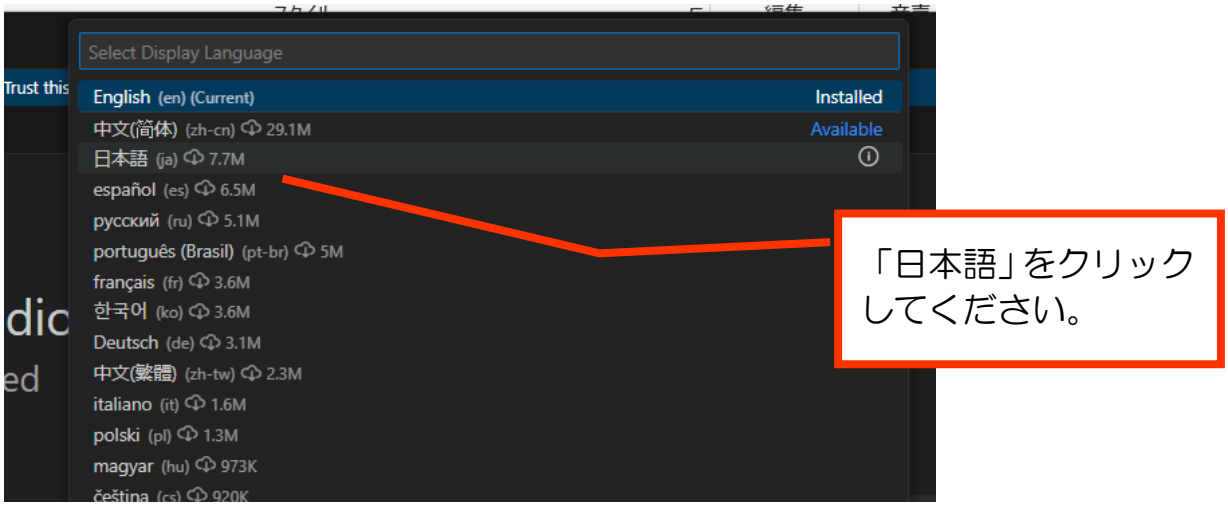
序章－１－２　Visual Studio Code の日本語化

メニューを日本語化しましょう。

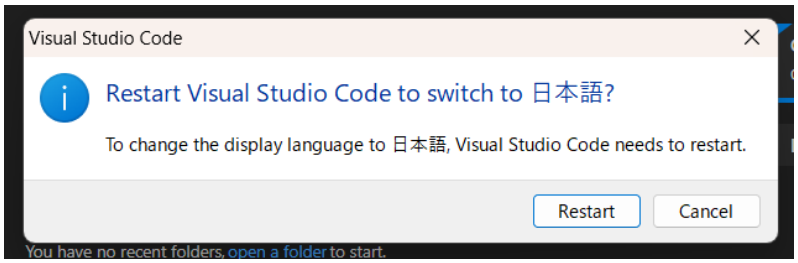


「C」で始まる項目が見えるところまでスクロールして、


>
Clear Console
Clear Display Language Preference
Clear Editor History
Close Window
Cloud Changes: Resume Latest Changes from Cloud
Cloud Changes: Show Cloud Changes
Cloud Changes: Store Working Changes in Cloud
Cloud Changes: Turn on Cloud Changes...
Code: Restart to Update
Code: Show Release Notes
Collapse Folders in Explorer
Configure Display Language
Copy
Ctrl + C
Copy Forwarded Port Address
Copy Output
Create: New File...
Ctrl + Alt + Windows + N
「Configure Display Language」をクリックしてください。

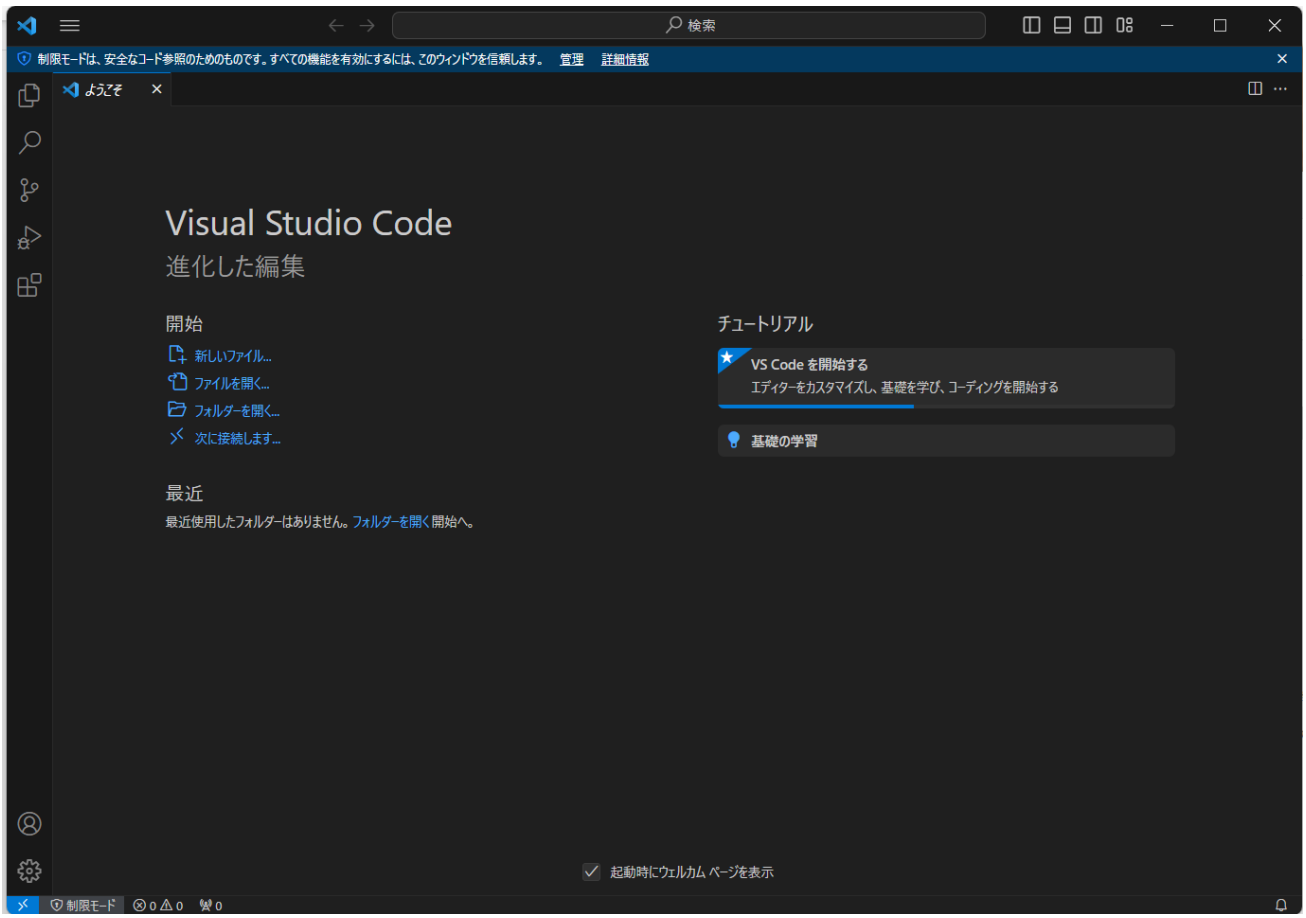


こんなふうに聞かれたら、[Restart]をクリックしてください。



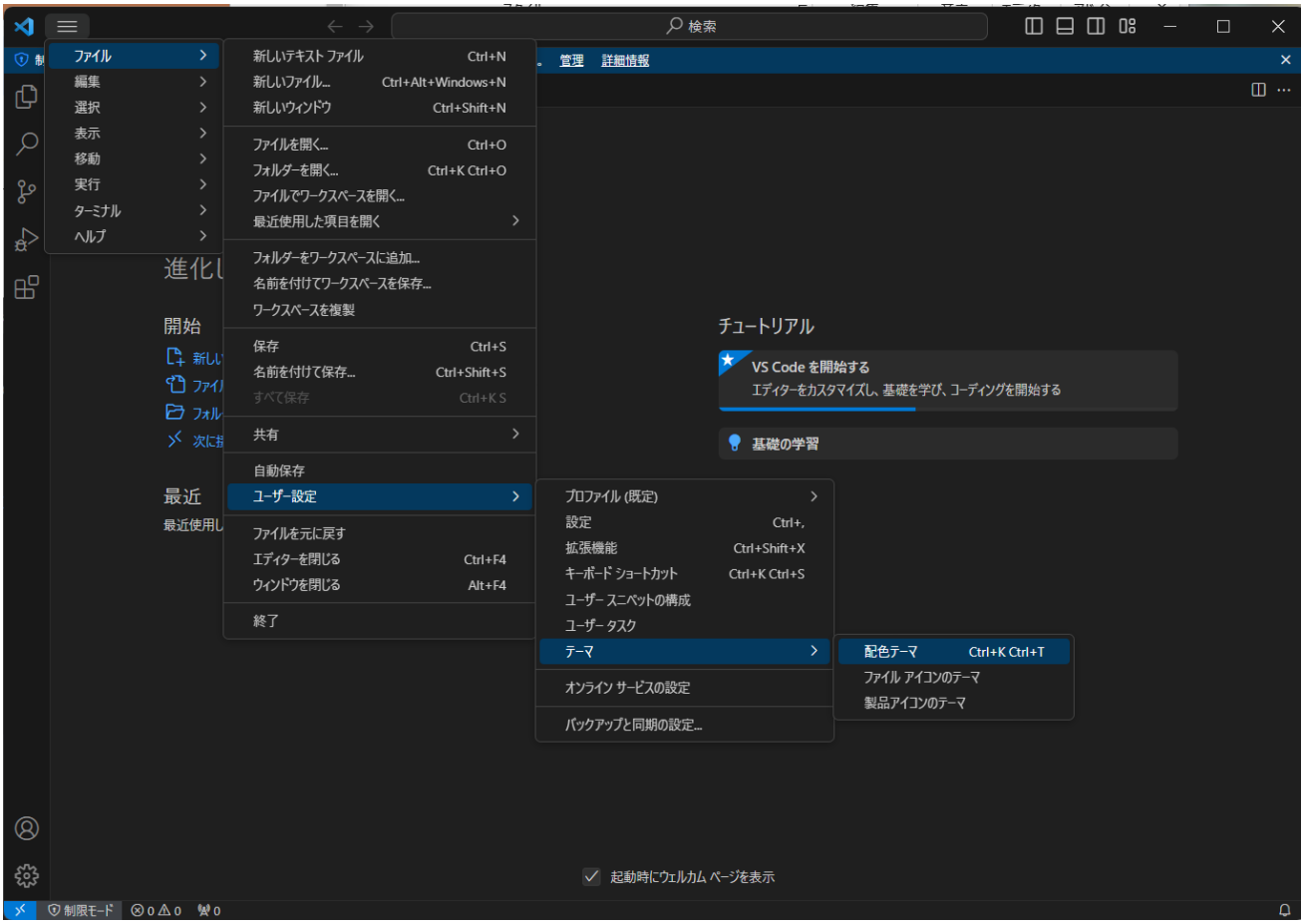
Visual Studio Code が再起動されます。

日本語表記になりました。


検索
制限モードは、安全なコード参照のためのものです。すべての機能を有効にするには、このウィンドウを信頼します。 管理 詳細情報
ようこそ
Visual Studio Code
進化した編集
開始
新しいファイル...
ファイルを開く...
フォルダーを開く...
次に接続します...
チュートリアル
VS Code を開始する
エディターをカスタマイズし、基礎を学び、コーディングを開始する
基礎の学習
最近
最近使用したフォルダーはありません。フォルダーを開く開始へ。
起動時にウェルカム ページを表示
制限モード

【参考】テーマ（見た目）の変更

このテキストでは、背景が白いテーマにしています。



メニュー＞ファイル＞ユーザー設定＞テーマ＞配色テーマ　とたどってください。

「Light(Visual Studio)」「ライト モダン」など、Light 系にすると背景が白（または白っぽい）ものになります。

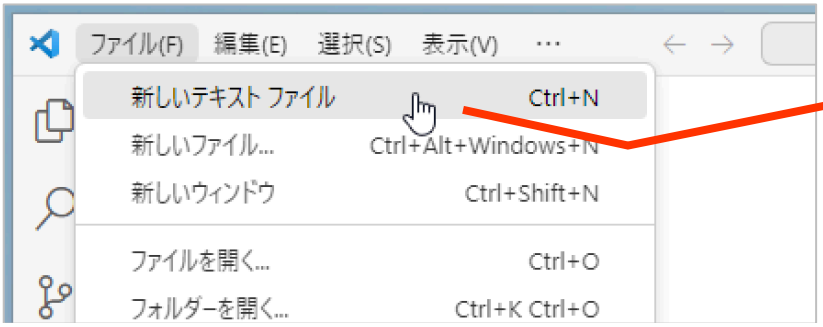
お好みのテーマをお使いください。

なお、本書中 Visual Studio Code 以外のコードエディタを使用して作成したものもあり、配色などが Visual Studio Code では再現できないものもあります。ご了承ください。

以下、Visual Studio Code を VS Code と略記することもあります。

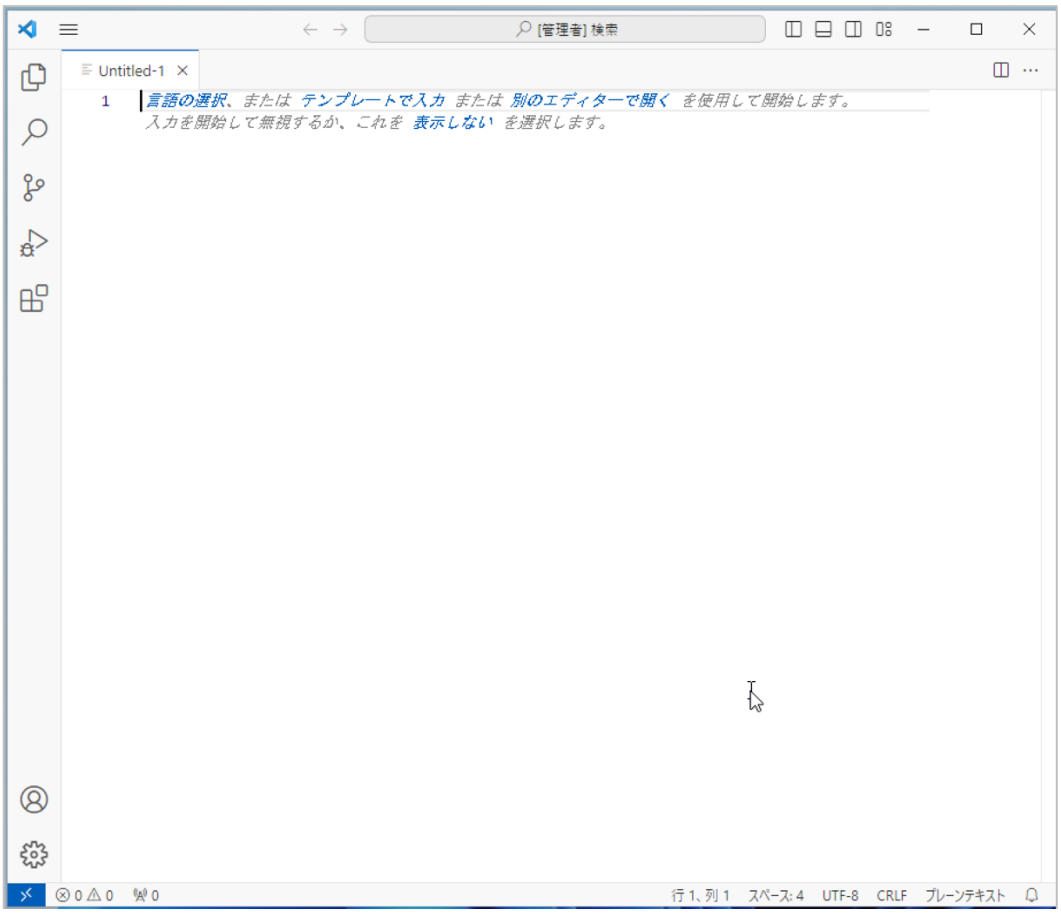
序章－２　VS Code で新規ファイルを作り、保存する

次の章で HTML を学びますので、準備として新規ファイルを作成しましょう。
VS Code を起動した状態で、新規ファイルを作成してください。

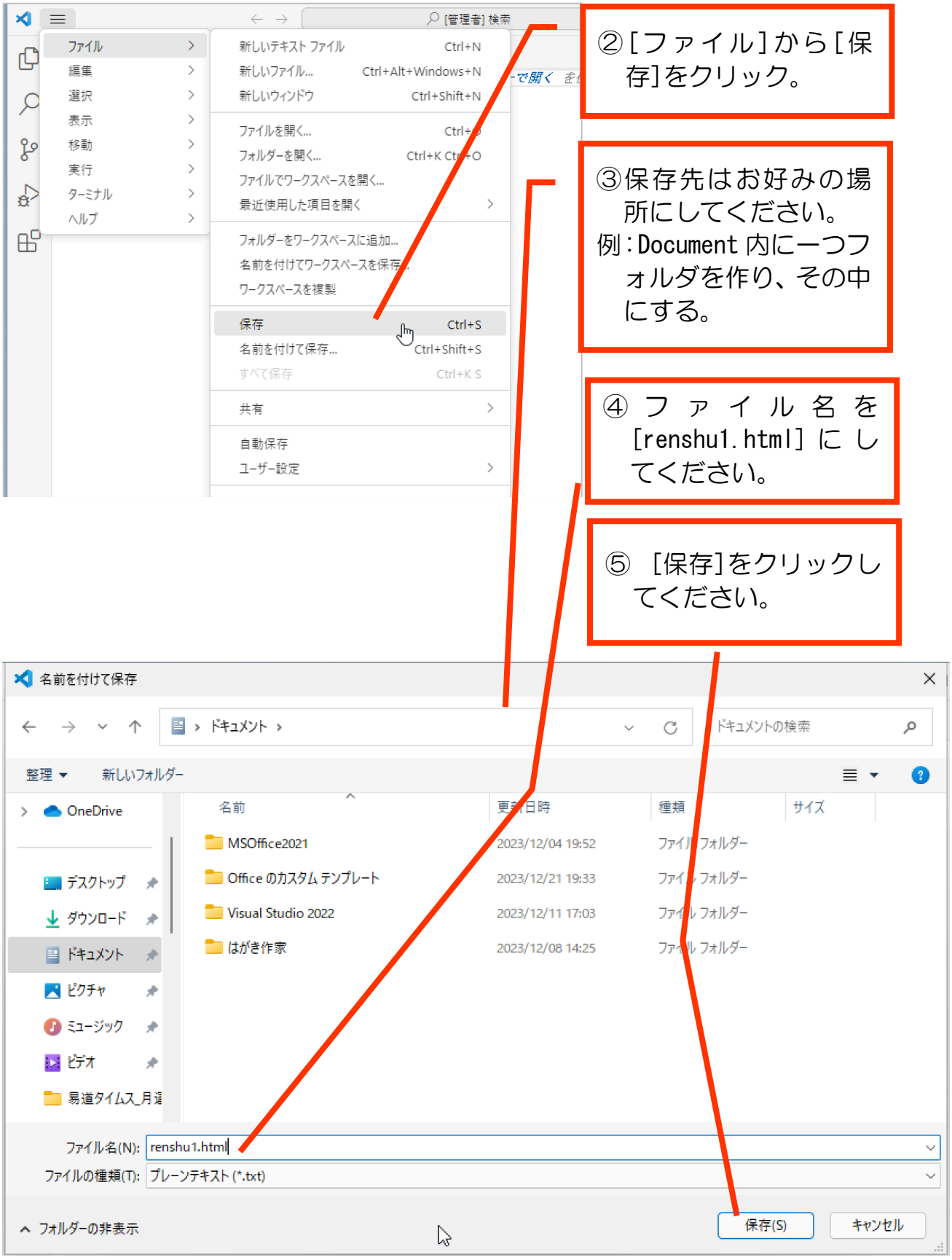


① [ファイル]から[新しいテキストファイル]をクリック。

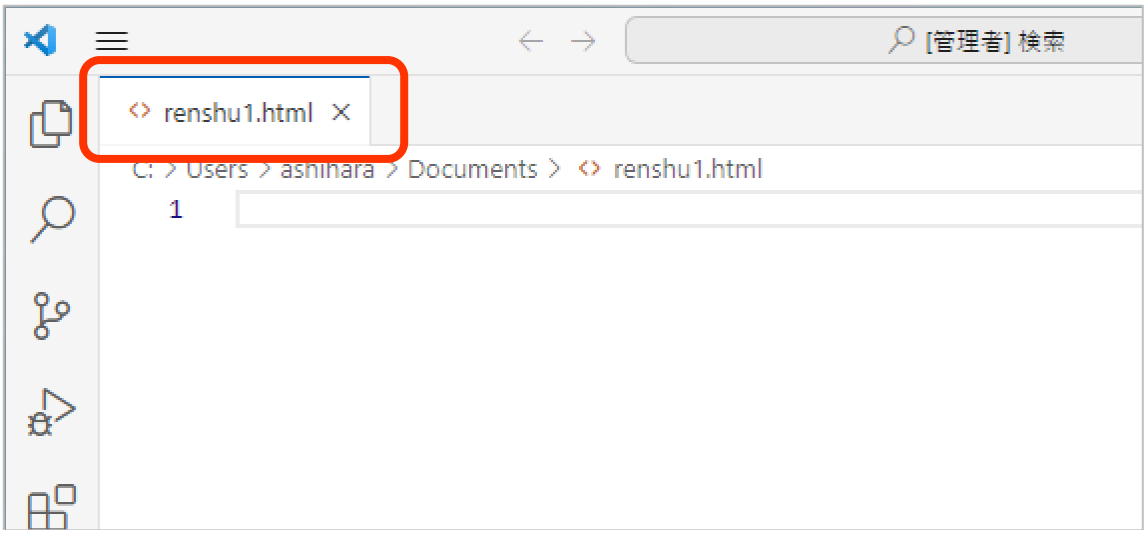
下図のように空白のファイルができあがります。



任意の場所に保存してください。



名前がついて保存されたファイルは、タブの部分にファイル名が確認できます。



第１章　HTML と CSS で遊ぼう

HTML…Web サイトの骨格・構造
CSS……Web サイトの見た目・飾り付け

１―１　ファイルの新規作成と大枠の準備

①コードエディタにて新規ファイルを作成し、任意のフォルダ内に
「kouza01.html」という名前で保存してください。

②下記のように打ち込んでください。

（参考）VSCode（Visual Studio Code）を使用している場合は、１行目に半角で「!」を打ち込み［Enter］（Mac は［Return］）キーを打つだけで全て入力されます。

この数字はエディタの行番号のため、打ち込む必要はありません。

```
1   <!DOCTYPE html>
2   <html lang="ja">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Document</title>
7   </head>
8   <body>
9
10  </body>
11  </html>
12
```

これは Web サイトを記述する際の HTML の大枠になります。
実のところ、JavaScript の学習のためだけであれば最低限次の記述だけでも構いません。

```
1   <html>
2       <body>
3
4       </body>
5   </html>
6
```

ですが、「!」を打つだけで全部入力される便利な機能があるなら使うに越したことはありません。

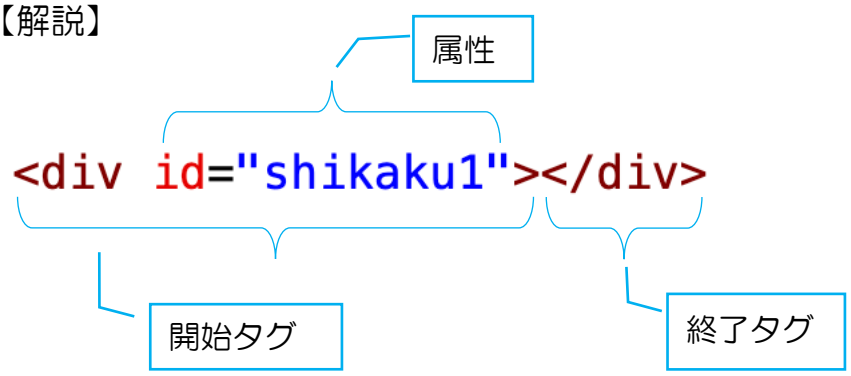
1―2　領域をつくる

① 下記のように追記してください。
「div タグ」というもので領域を用意します。

```
 7    </head>
 8    <body>
 9        <div id="shikaku1"></div>
10    </body>
11    </html>
```

追加

【解説】



div タグは、領域を示すためのもの
開始タグと終了タグのペアで使われます。
開始タグの中に属性を追加することができ、この例では id 属性を入れています。
id 属性は、このタグを識別するために id を割り当てる用途に使います。

1―3　CSS にて見た目をつくる　幅・高さ・ぬりつぶし

CSS（Cascading Style Sheets）は Web サイトのデザインを整えるためのスタイルシート言語です。主に Web ページの色、文字サイズ、レイアウトなどを指定し、HTML と組み合わせて Web サイトを視覚的に魅力的にします。CSS は、プログラミング言語ではありませんが、Web 制作で HTML とセットで使われる重要な技術です。（引用元：Google 検索にて AI による概要）

① 下記のように追記してください。この例では<head>の領域に<style>タグにて CSS を記述する領域を設けました。
<style>は<head>に記述しなければいけないというわけではありませんが、<head>に記述されるケースが一般的です。

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=d
    <title>Document</title>
    <style>
        #shikaku1{

        }
    </style>
</head>
<body>
    <div id="shikaku1"></div>
</body>
</html>
```

追加

この波線は VSCode の機能で自動的に表示されているもので、記述の必要はあり

【解説】

```
    <style>
        #shikaku1{

        }
    </style>
</head>
<body>
    <div id="shikaku1"></div>
</body>
```

id 名は「#」で始める。

{で始まり、}で終わる。
この間に、この id に対するスタイルの情報を記述する。

この id が割り当てられているタグに対してスタイルを指定する。

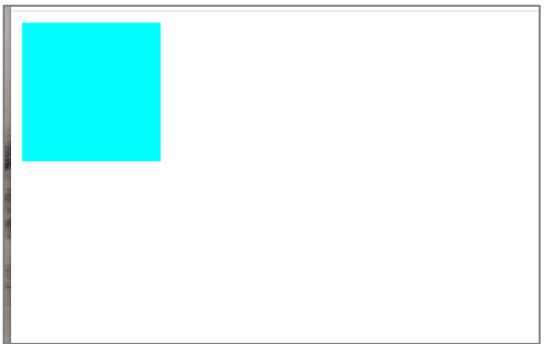
② 次のように追記してください。

```
 8        #shikaku1{
 9            width: 100px;
10            height: 100px;
11            background-color: aqua;
12        }
```

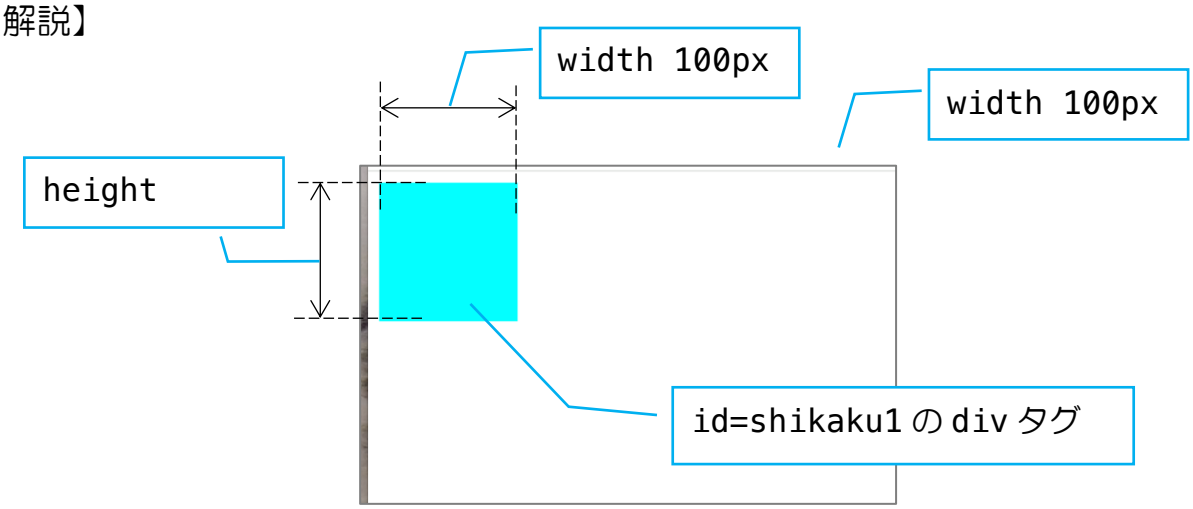
追加

この水色の四角 VSCode の機能で自動的に表示されているもので、記述の必要はありません。

ブラウザで表示して、下図のように見えれば OK です。



【解説】



幅 100px，高さ 100px、background-color で塗りつぶし色を指定しています。

なお、VSCode では background-color: と打っただけで色の候補がたくさん出てきます。（CSS 記述部の時だけ）

【課題】
（１）　横長にしてみましょう。それができたら、縦長にしてみましょう。
（２）　塗りつぶしの色を変えてみましょう。

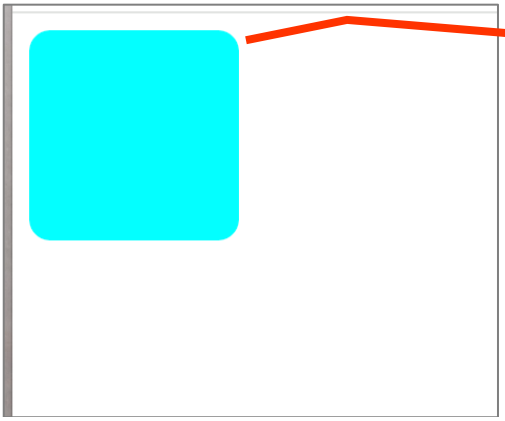
1―4　角の丸み

1-4-1　角の丸み　４つ角すべて

角に丸みをつけてみましょう。
下記のように追記してください。

```
 7      <style>
 8          #shikaku1{
 9              width: 100px;
10              height: 100px;
11              background-color: aqua;
12              border-radius: 10px;
13          }
14      </style>
```

追加



４つの角に丸みがつけば OK です。

【解説】
border-radius で半径のサイズを指定できます。
サイズの単位は px または%が主に使われます。（他に em、rem、vw、vh なども使えるようです）

【演習１】
半径を 50%にしてみましょう。
どうなるか、確認してみてください。

【演習２】
今 50%の半径ですが、もっと小さな数値や大きな数値に変えてみて、どうなるか見てみましょう。

1-4-2　角の丸み　４つ角別々

４つの角それぞれ別々にも指定できます。

【演習】
① まずは次のコードに変更してみてください。どんな見た目になりますか？

```
11          background-color: aqua;
12          border-radius: 50% 10%;
```

変更

② 次のように変更するとどうなりますか？

```
11          background-color: aqua;
12          border-radius: 50% 10% 30%;
```

変更

③ 次のように変更するとどうなりますか？

```
11          background-color: aqua;
12          border-radius: 50% 10% 30% 20%;
```

変更

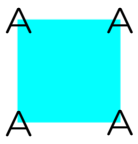
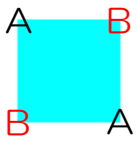
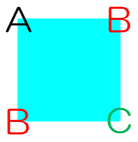
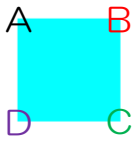
④ 次のように変更するとどうなりますか？

```
12          border-radius: 50% 10% 30% 20% 5%;
```

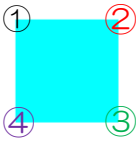
変更

結果のまとめは次のページです。

【まとめ】
　半径の指定の数値の個数によって、設定される角は次のようになることがわかりましたか？

	コード	円くなる場所
１つ	`border-radius: A;`	
２つ	`border-radius: A B;`	
３つ	`border-radius: A B C;`	
４つ	`border-radius: A B C D;`	
５つ	`border-radius: A B C D E;`	 何も変わらない

＜　覚え方　＞



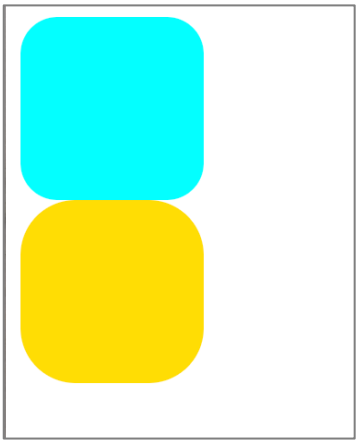
左上から時計回りが基本
４つ未満のとき（③・④が書かれていない場合）、対角と同じになる。
例えば、③が書かれていない場合は対角の①と同じ。
④が書かれていない場合は対角の②と同じ。

５つ以上は文法的に間違っているので命令としては意味をなさない。

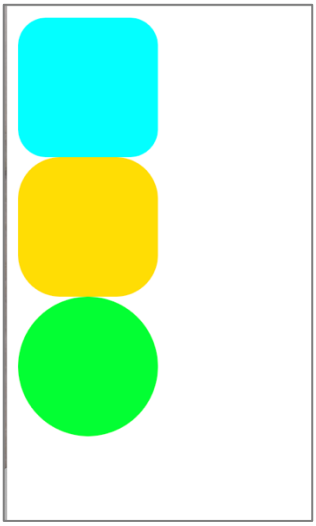
（参考）VSCode に慣れる
　VSCode などのコードエディタを使用している場合、何文字か打つと候補が出てきます。出てきた中から選ぶとスペルミスを防げますので、積極的に活用することをおススメします。

1-5　課題

【課題　1-5-1】
もう１個同じものをつくってみましょう。
位置の設定はしなくても、下図のように縦に並びます。
２個目の色と半径は任意です。１個目も含めていろいろな色を試してみましょう。
id は shikaku2 にしてください。
わからないときは周囲の人と相談してみましょう。



【課題　1-5-2】
もう１個（つまり３個目）も作ってみましょう。
id は shikaku3 にしてください。
円にしてください。
色も変えてください。



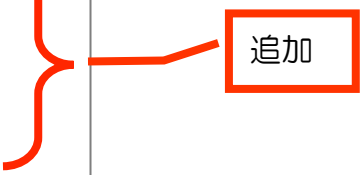
解答例

【課題　1-5-1】

```
 7      <style>
 8          #shikaku1{
 9              width: 100px;
10              height: 100px;
11              background-color: aqua;
12              border-radius: 20%;
13          }
14          #shikaku2{
15              width: 100px;
16              height: 100px;
17              background-color: rgb(255, 221, 0);
18              border-radius: 30%;
19          }
20      </style>
21  </head>
22  <body>
23      <div id="shikaku1"></div>
24      <div id="shikaku2"></div>
25  </body>
```

【課題　1-5-2】

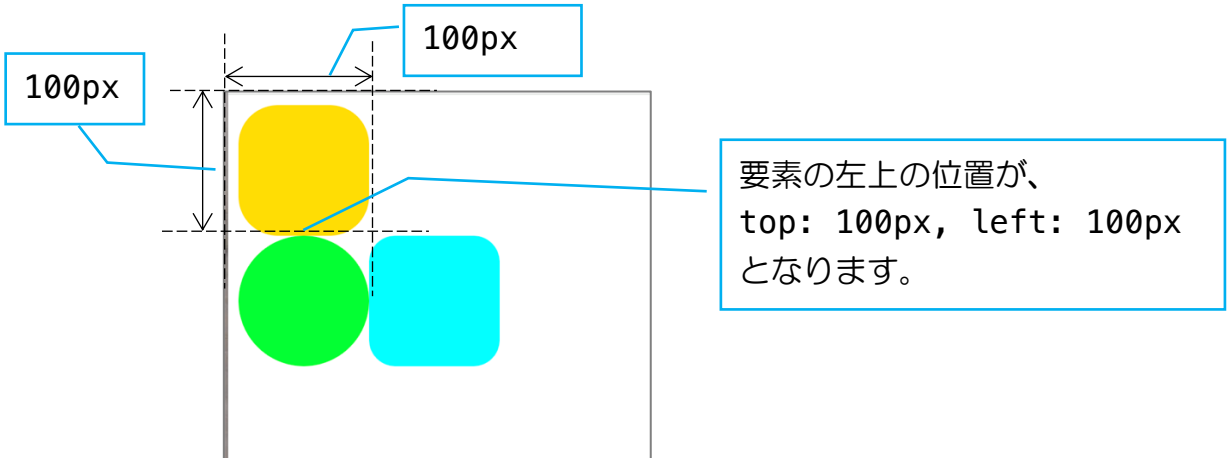
```
16              height: 100px;
17              background-color: rgb(255, 221, 0);
18              border-radius: 30%;
19          }
20          #shikaku3{
21              width: 100px;
22              height: 100px;
23              background-color: rgb(0, 255, 51);
24              border-radius: 50%;
25          }
26      </style>
27  </head>
28  <body>
29      <div id="shikaku1"></div>
30      <div id="shikaku2"></div>
31      <div id="shikaku3"></div>
32  </body>
```



1-6　位置指定

下図のように 1 個目の位置を変えてみます。

上から 100px，左から 100px の位置に指定する方法を使います。



位置情報を扱う場合、何を基準とするか意識しておきます。
今回は body を基準として#shikaku1 の位置を決めます。

① そこで、CSS に次のように body を位置基準とするコードを追加してください。

```
 7      <style>
 8          body{
 9              position: relative;
10          }
11          #shikaku1{
```

追加

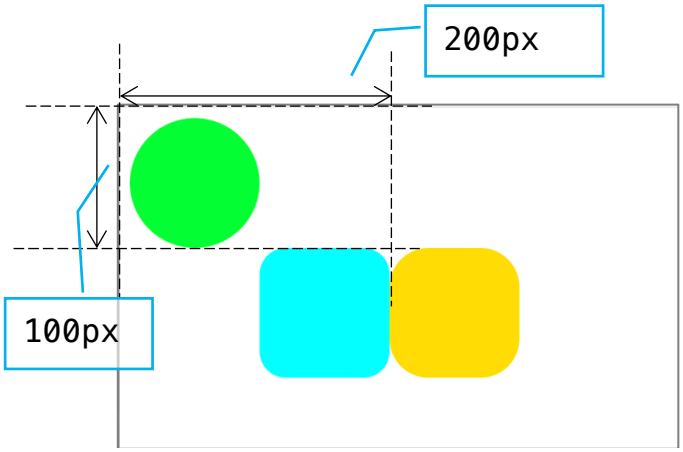
② 次に、#shikaku1 に次のコードを追加してください。

```
10          }
11          #shikaku1{
12              position: absolute;
13              top: 100px;
14              left: 100px;
15              width: 100px;
16              height: 100px;
17              background-color: aqua;
18              border-radius: 20%;
19          }
```

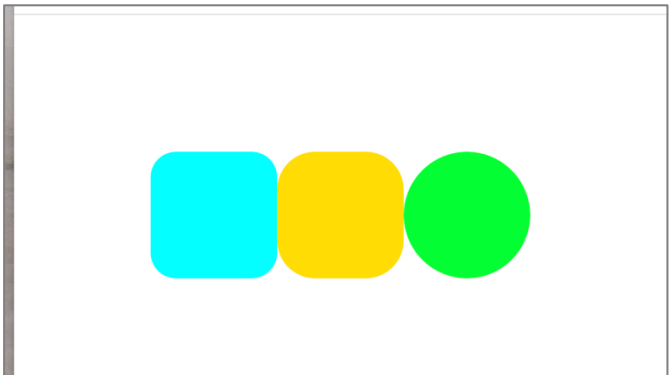
追加

position: absolute;、top, left ともに 100px;
position: absolute;は、position: relative;要素を基準にした絶対位置です。

【課題１-６-１】
（１）　#shikaku2 を#shikaku1 の右に表示してみよう
　　表示すると下図のようになります。



（２）　#shikaku３を#shikaku２の右に表示してみよう
　　表示すると下図のようになります。
　　座標は想像がつくと思いますので、下図には示しません。お考えください。



いずれの課題も、わからないときは周囲の人と相談してみましょう。

解答例は次ページです。

（まとめ）
　位置指定、色、半径、id を使って CSS から HTML の部品を指定する方法を学びました。

解答例【課題１-６-１】

（１）

```
#shikaku2{
    position: absolute;
    top: 100px;
    left: 200px;
    width: 100px;
    height: 100px;
    background-color: rgb(255, 221, 0);
    border-radius: 30%;
}
```

追加

（２）

```
#shikaku3{
    position: absolute;
    top: 100px;
    left: 300px;
    width: 100px;
    height: 100px;
    background-color: rgb(0, 255, 51);
    border-radius: 50%;
}
```

追加

第２章　JavaScript こと始め

この章では、マウスを動かすとブラウザ上の要素が連動して動く仕組みを学びます。

2-1　マウスの動きに連動

プログラムを使うことで、「マウスが動いた」というようなイベントに対応して Web ブラウザ上のものを変化させることができます。
いよいよ JavaScript（ジャヴァスクリプト）に触れることになります。

前章で作った「kouza01.html」をコピーして、「kouza02-1.html」としてください。

① 下図のように、JavaScript を書くエリアを作ってください。

```
<body>
    <div id="shikaku1"></div>
    <div id="shikaku2"></div>
    <div id="shikaku3"></div>
    <script>

    </script>
</body>
```

追加

script タグの開始・終了のペアで挟まれたエリアにプログラムをこれから書いていきます。

② 下図のように追記してください。
HTML の id=”shikaku1”の要素を JavaScript から扱えるように、定数 shikaku1 を定義します。

```
<body>
    <div id="shikaku1"></div>
    <div id="shikaku2"></div>
    <div id="shikaku3"></div>
    <script>
        const shikaku1 = document.querySelector('#shikaku1');
    </script>
```

一致させる。
#は id の意味

追加

③ マウスが動いたという「イベント」を検知するしくみを下図のように追加してください。

```
<script>
    const shikaku1 = document.querySelector('#shikaku1');

    window.addEventListener('mousemove', e=>{

    }, false);
</script>
```

追加

「addEventListener」はイベントの発生を検知する仕組みです。
「'mousemove'」はマウスが動いたイベントを指定しています。

④ ひきつづき、イベント発生時に行いたい処理を下図のように追記してください。

```
        window.addEventListener('mousemove', e=>{
            shikaku1.style.left = e.clientX + "px";
        }, false);
    </script>
```

追加

【解説】

```
shikaku1.style.left = e.clientX + "px";
```

shikaku1 の style の left、すなわち、これ

```
#shikaku1{
    position: absolute;
    top: 100px;
    left: 100px;
    width: 100px;
```

を、e.clientX（マウスポインタの x 座標）px に変更します。

マウスが動くたびにこのイベントが発生し、そのたびに left にマウスの x 座標がセットされます。ですので、shikaku1 がマウスの横の動きに追従します。

【課題】
マウスの縦方向の動きにも追従するようにしてみましょう。

```
window.addEventListener('mousemove', e=>{
    shikaku1.style.left = e.clientX + "px";
    ここに追記してください
}, false);
```

マウスのｙ座標は、e.clientY です。
ヒントなしでできる人は、ぜひチャレンジしてください。

ヒントが必要な人は、下図の①～③を埋めるように作ってみましょう。

```
window.addEventListener('mousemove', e=>{
    shikaku1.style.left = e.clientX + "px";
    shikaku1.style.[①] = e.[②] + [③];
}, false);
```

```
#shikaku1{
    position: absolute;
    top: 100px;
    left: 100px;
    width: 100px;
```

縦方向の位置ですから、
style ではこの項目ですよ。

解答例は次のページです。

課題　解答例

```
window.addEventListener('mousemove', e=>{
    shikaku1.style.left = e.clientX + "px";
    shikaku1.style.top = e.clientY + "px";
}, false);
```

追加

2-2　クリックされた

クリックされたイベントに対しての処理を学びましょう。
前節で作った「kouza02-1.html」をコピーして、「kouza02-2.html」としてください。
kouza02-2.html のマウスムーブイベントは削除してしまいましょう。
id="shikaku1"がクリックされたときの処理を作っていきますので、「const shikaku1 = （以下略）」の命令は残しておいてください。
下図のようになります。

```
    <script>
        const shikaku1 = document.querySelector('#shikaku1');



    </script>
</body>
```

① id="shikaku1"に対するクリックイベントを次のように追加してください。

```
    <script>
        const shikaku1 = document.querySelector('#shikaku1');

        shikaku1.addEventListener('click', e=>{

        }, false);
```

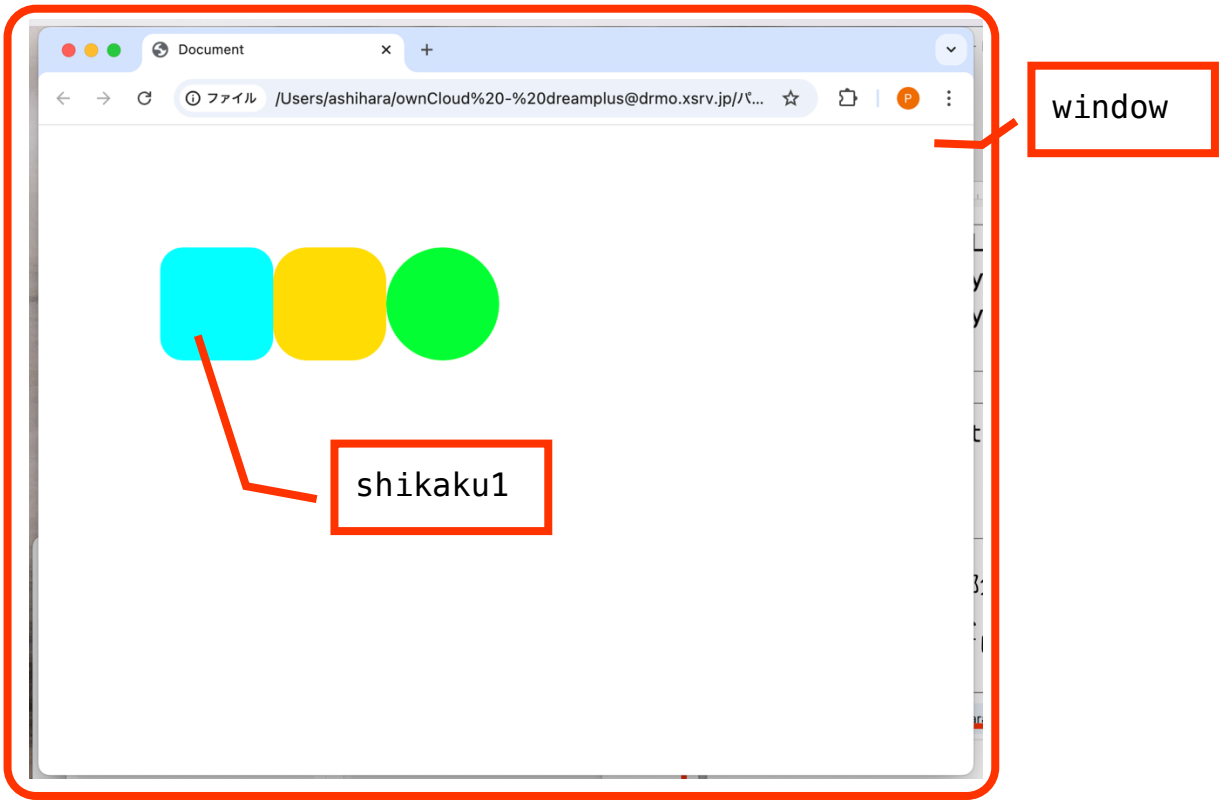
追加

kouza02-1.html で作った「mousemove」との違いについて見てみましょう。

```
window.addEventListener('mousemove', e=>{
    shikaku1.style.left = e.clientX + "px";
    shikaku1.style.top = e.clientY + "px";
}, false);
```

```
shikaku1.addEventListener('click', e=>{

}, false);
```

上図にて赤線を引いた部分、「window」と「shikaku1」が違いますね。
「window.～」であれば、window に対してのイベントを検知
「shikaku1.～」であれば、shikaku1 に対してのイベントを検知
ということになります。

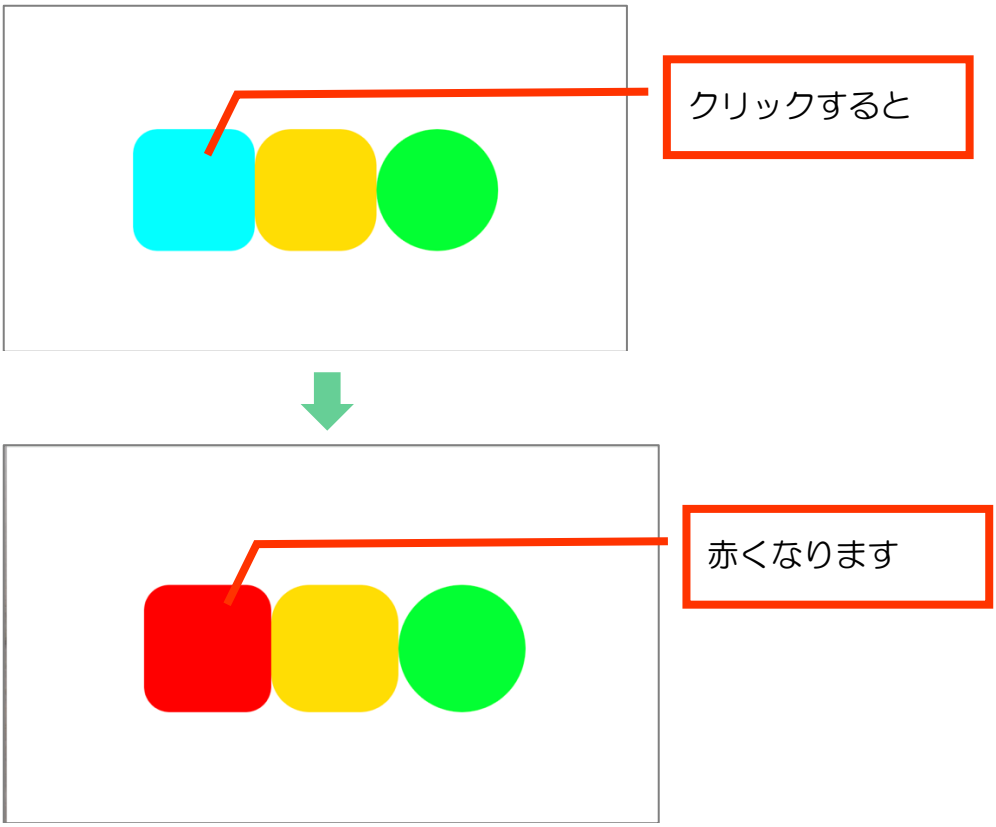


② id="shikaku1"の要素の塗りつぶし色が赤くなるようにします。下図のように命令を追加してください。

```
shikaku1.addEventListener('click', e=>{
    shikaku1.style.backgroundColor = "#ff0000";
}, false);
```

追加

③ id="shikaku1"の要素をクリックして、色が変わることを確認してください。



2-3　色の表現方法について

2-3-1　６ケタのカラーコード

　色は文字に限らず、背景の塗りつぶしや線の色など、全てに共通です。

　色を指定している、#から始まる６文字は、左から２ケタずつ赤・緑・青の要素の強さを示しています。

```
# ff 00 00
  赤 緑 青
```

光の三原色（Red・赤、Green・緑、Blue・青）のそれぞれの強さを 0～F の 16 段階で表現します。（16進数（しんすう））

0 はその色の要素が 0（最小）、F はその色の要素が最大です。

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

最小←　→最大

一つの色要素を 16 進数２ケタで表すので、16 x 16 = 256 段階で表現できます。
ff 00 00
は、赤が最大・緑が０・青も０なので、赤になります。

#00ff00 なら、緑になります。

2-3-2　３ケタの省略形（カラーコード）

６ケタで表す色のコードは、３ケタに省略することもできます。

例えば、「#f00」のように表現します。
３ケタの場合は、R、G、B のそれぞれ２ケタ分を１ケタにまとめた状態となります。

#f00 = #ff0000

です。
　６ケタの場合は 16 段階が６ケタなので、16x16x16x16x16x16 = 16,777,216 色を表現できますが、
　３ケタの場合は 16 段階が６ケタなので、16x16x16 = 4,096 色になります。

2-3-3　rgb

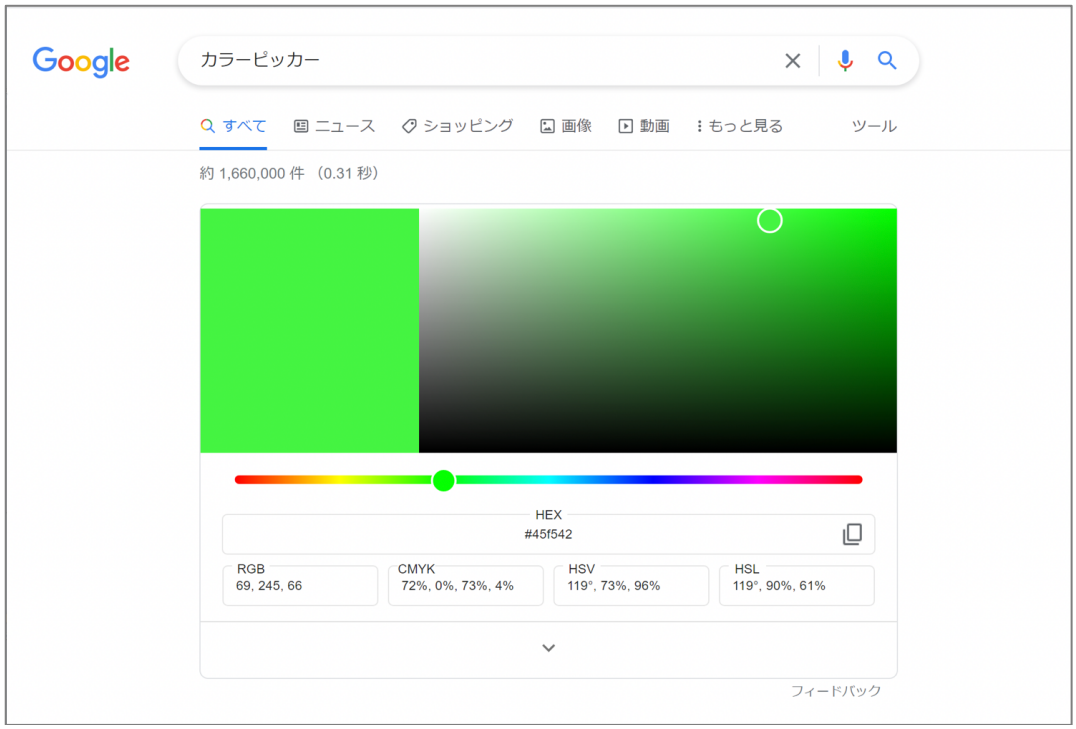
１-５の解答例では rgb(255, 255, 0)の表現方法が出てきています。
６ケタの表現方法と対比させると下記のようになります。

```
# ff 00 00
  赤 緑 青

rgb(255, 0, 0)
    赤  緑  青
```

rgb 形式では、10 進法で 0～255（つまり 16 進法の 00 ～ff）を用いて各三原色の強度を表現します。

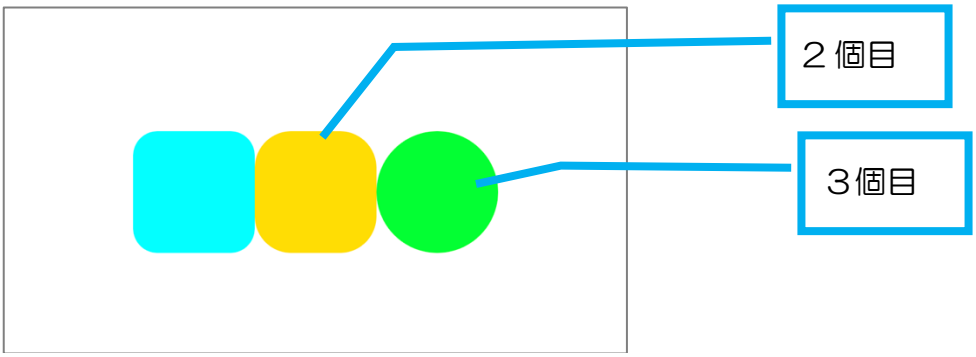
2-3-4　カラーピッカー

色のコード指定は、Google で「カラーピッカー」を検索すると使いやすい項目が出てきます。（下図参照）
下図の「HEX」の部分が６ケタのコード、「RGB」の部分が rgb のコードです。
ほぼどんな色でも指定できます。



2-4　課題

２個目、３個目にも、クリックしたら色が変わる仕組みを追加してください。



（ヒント　というか手順）
２個目に関しては、下記の穴埋めでつくってみましょう。

まずは定数 shikaku2 を定義し、２個目の要素の id を指定してください。
こうすることで、JavaScript 内で 2 個目の要素を shikaku2 という定数で制御するための準備ができます。

```
<script>
    const shikaku1 = document.querySelector('#shikaku1');
    const shikaku2 = document.querySelector('  ①  ');

    shikaku1.addEventListener('click', e=>{
```

追加

次に、イベントリスナーをつかってクリックされた時の処理を記述してください。
shikaku1 に対してのイベントリスナーを参考にして、下記の②～④を埋める感じで作成してみましょう。④には任意のカラーコードを入れてください。６ケタの記述でも rgb の記述でもかまいません。

```
shikaku1.addEventListener('click', e=>{
    shikaku1.style.backgroundColor = "#ff0000";
}, false);

  ②  .addEventListener('click', e=>{
    ②  .style.  ③  = ④ ;
}, false);
```

コードが書けたら、思った通りに動くかどうか動作確認してください。

２個目までできたら、３個目もつくってください。
ノーヒントとしますが、手順としてはこれまでと同じです。
const での定義、イベントリスナーでクリックされたときの処理の記述、となります。

解答例は次のページです。

2-4課題の解答例

2個目まで

```
<script>
    const shikaku1 = document.querySelector('#shikaku1');
    const shikaku2 = document.querySelector('#shikaku2');

    shikaku1.addEventListener('click', e=>{
        shikaku1.style.backgroundColor = "#ff0000";
    }, false);

    shikaku2.addEventListener('click', e=>{
        shikaku2.style.backgroundColor = "#ff99ff";
    }, false);
```

追加

追加

① #shikaku2
② shikaku2
③ backgrouncColor
④ 任意のコード　サンプルとしては”#ff99ff”

3個目

```
<script>
    const shikaku1 = document.querySelector('#shikaku1');
    const shikaku2 = document.querySelector('#shikaku2');
    const shikaku3 = document.querySelector('#shikaku3');

    shikaku1.addEventListener('click', e=>{
        shikaku1.style.backgroundColor = "#ff0000";
    }, false);

    shikaku2.addEventListener('click', e=>{
        shikaku2.style.backgroundColor = "#ff99ff";
    }, false);

    shikaku3.addEventListener('click', e=>{
        shikaku3.style.backgroundColor = "#ff9900";
    }, false);
```

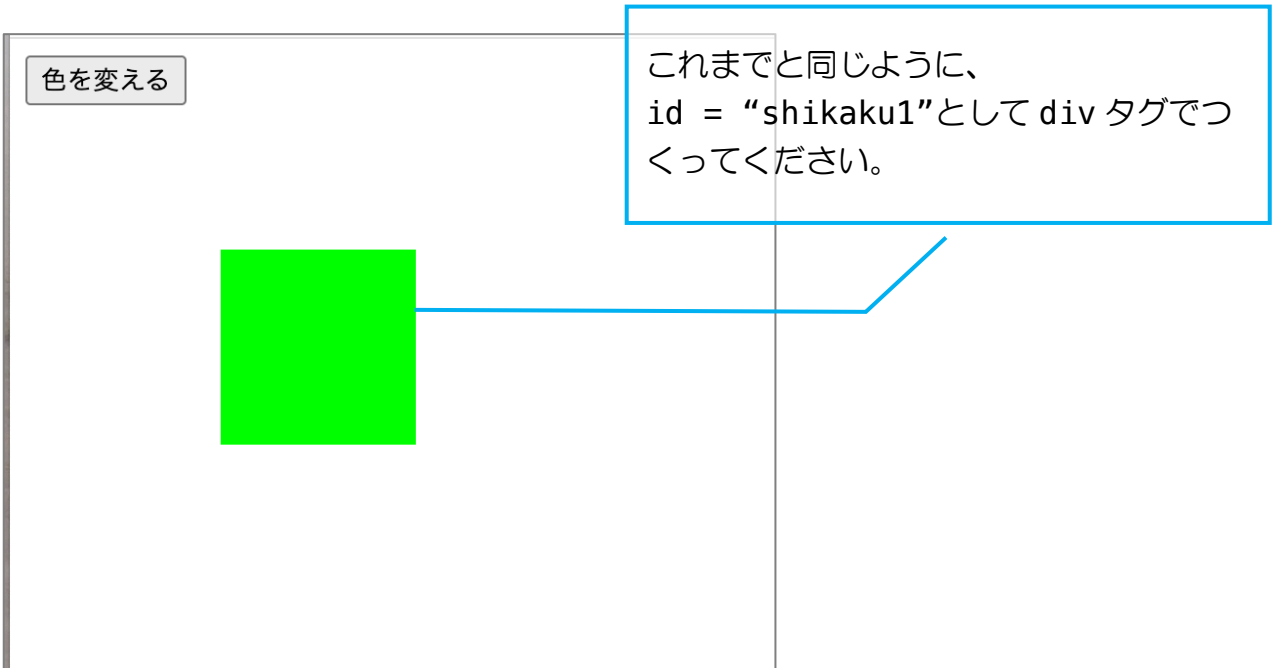
追加

追加

2-5　ボタンを追加してみよう【演習】

2-5-1　部品の用意

下図の見た目になるような html ファイルを作成してください。ファイル名は kouza02-4.html としてください。



id = “shikaku1”の style は、
上から 100px、左から 100px、幅 100px、高さ 100px。
塗りつぶし色は#00ff00。
position: absolute;の設定を忘れずに。

body に対しての position: relative;の設定を忘れずに。

［色を変える］ボタンは、html にて button タグを使ってください。下記のコードです。style は指定しなくて良いです。

```
<button id="btn">色を変える</button>
```

解答例は次のページです。

kouza02-4.html　解答例

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        body{
            position: relative;
        }
        #shikaku1{
            position: absolute;
            top: 100px;
            left: 100px;
            width: 100px;
            height: 100px;
            background-color: #00ff00;
        }
    </style>
</head>
<body>
    <div id="shikaku1"></div>
    <button id="btn">色を変える</button>
</body>
</html>
```

2-5-2　ボタンクリックで色を変えよう

［色を変える］ボタンをクリックしたら id=”shikaku1”の塗りつぶし色が黄色になるようにしてください。

　黄色は６ケタのコードで表すと#ffff00 です。

解答例は次のページです。

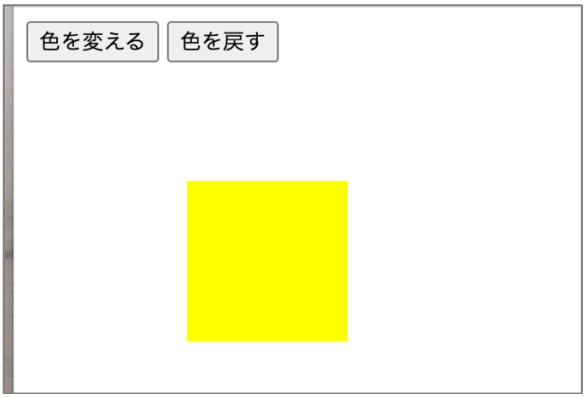
kouza02-4.html　［色を変える］ボタンクリックの処理　解答例

```
<body>
    <div id="shikaku1"></div>
    <button id="btn">色を変える</button>
    <script>
        const shikaku1 = document.querySelector("#shikaku1");
        const btn = document.querySelector("#btn");

        btn.addEventListener('click', e=>{
            shikaku1.style.backgroundColor = "#ffff00";
        }, false);

    </script>
</body>
```

追加

2-5-3　元の色に戻すボタンを作ってみよう

完成例としては下図のような感じです。



先ほどのコードに追加してください。

［色を戻す］ボタンはここに追加でよいです。id は btn2 にしておきましょう。

```
<body>
    <div id="shikaku1"></div>
    <button id="btn">色を変える</button>
    <script>
        const shikaku1 = document.querySelector("#shikaku1");
        const btn = document.querySelector("#btn");

        btn.addEventListener('click', e=>{
            shikaku1.style.backgroundColor = "#ffff00";
        }, false);

    </script>
```

btn2 を扱う定数を追加

btn2 に対するイベントリスナーを作りましょう。

kouza02-4.html　［色を戻す］ボタンクリックの処理　解答例

```
<body>
    <div id="shikaku1"></div>
    <button id="btn">色を変える</button>
    <button id="btn2">色を戻す</button>
    <script>
        const shikaku1 = document.querySelector("#shikaku1");
        const btn = document.querySelector("#btn");
        const btn2 = document.querySelector("#btn2");

        btn.addEventListener('click', e=>{
            shikaku1.style.backgroundColor = "#ffff00";
        }, false);
        btn2.addEventListener('click', e=>{
            shikaku1.style.backgroundColor = "#00ff00";
        }, false);

    </script>
</body>
```

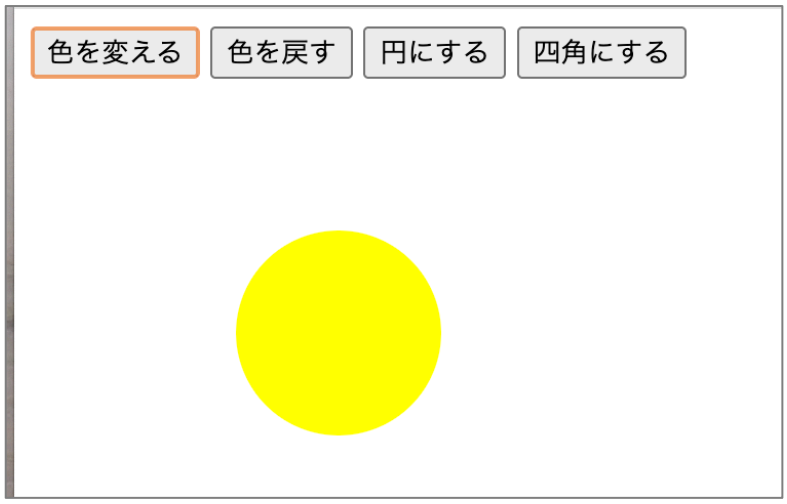
追加

追加

追加

2-5-4　円にする・四角に戻す　２つのボタンをつくろう

完成例としては下図のような感じです。



先ほどのコードに追加してください。

（ヒント）
［円にする］ボタンは id="btn3"とし、クリックされたら shikaku1 の半径を50%にする処理とします。JavaScript から半径を指定するには下記のようにします。

```
shikaku1.style.borderRadius = "50%";
```

同様に、［四角にする］ボタンは id="btn4"とし、半径を0%にしてください。

kouza02-4.html　［円にする］［四角にする］ボタンクリックの処理　解答例

```
<body>
    <div id="shikaku1"></div>
    <button id="btn">色を変える</button>
    <button id="btn2">色を戻す</button>
    <button id="btn3">円にする</button>
    <button id="btn4">四角にする</button>
    <script>
        const shikaku1 = document.querySelector("#shikaku1");
        const btn = document.querySelector("#btn");
        const btn2 = document.querySelector("#btn2");
        const btn3 = document.querySelector("#btn3");
        const btn4 = document.querySelector("#btn4");

        btn.addEventListener('click', e=>{
            shikaku1.style.backgroundColor = "#ffff00";
        }, false);
        btn2.addEventListener('click', e=>{
            shikaku1.style.backgroundColor = "#00ff00";
        }, false);
        btn3.addEventListener('click', function(){
            shikaku1.style.borderRadius = "50%";
        }, false);
        btn4.addEventListener('click', function(){
            shikaku1.style.borderRadius = "0%";
        }, false);

    </script>
```

追加

追加

追加

（※注意）
`const shikaku1 = document.querySelector("#shikaku1");`
などの定数定義を書き忘れても eventListner は働くことがありますが、正しい動作をしなくなる可能性を内包してしまうのでちゃんと記述しておくことを推奨します。

2-6　ボタンを有効・無効にしてみよう

ボタンの有効・無効は button タグに disable 属性を設定することで実現できます。

JavaScript から制御するには、対象に対して.disable を true か false を指定します。

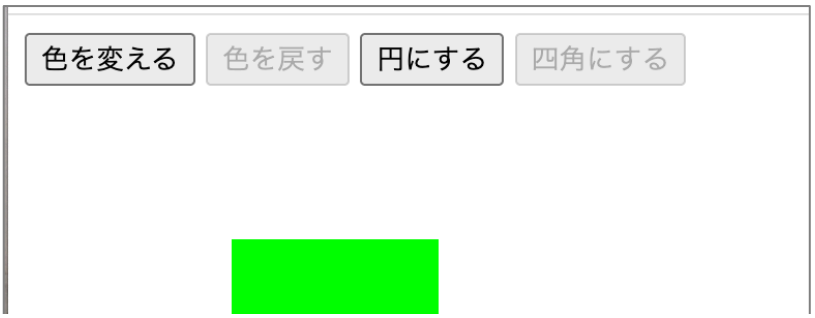
①まずは、btn2 と btn4 を無効にしてみましょう。下記のように追記してください。

```
    <div id="shikaku1"></div>
    <button id="btn">色を変える</button>
    <button id="btn2" disabled>色を戻す</button>
    <button id="btn3">円にする</button>
    <button id="btn4" disabled>四角にする</button>
    <script>
```

追加

追加

②ブラウザで見ると、下図のように無効になります。色が薄くなり、クリックできない状態です。



③［色を変える］ボタンがクリックされたときにそのボタンを無効にしてみましょう。下図のように disabled を true にしてあげます。

```
btn.addEventListener('click', e=>{
    shikaku1.style.backgroundColor = "#ffff00";
    btn.disabled = true;
}, false);
btn2.addEventListener('click', e=>{
```

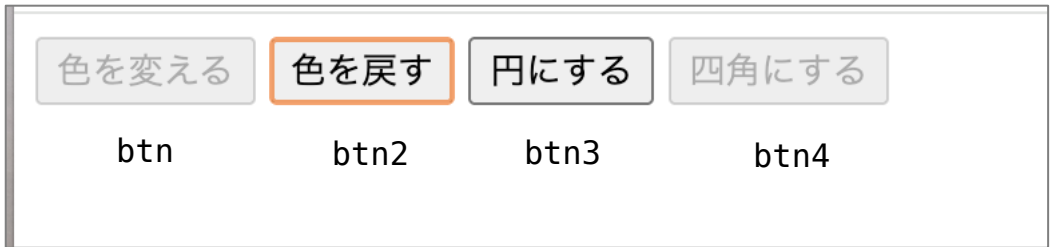
追加

④同じタイミングで［色を戻す］ボタンが有効になるようにしてみましょう。下図のように disabled を false にしてあげます。

```
btn.addEventListener('click', e=>{
    shikaku1.style.backgroundColor = "#ffff00";
    btn.disabled = true;
    btn2.disabled = false;
}, false);
```

追加

これで色を戻せるようになりましたね。
残りのボタンにも設定を追加してみましょう。



btn2 がクリックされたら、btn2 は無効、btn1 を有効に
btn3 がクリックされたら、btn3 は無効、btn4 を有効に
btn4 がクリックされたら、btn4 は無効、btn3 を有効に

自力での挑戦をおすすめします。

解答例は次のページです。

【参考】
イベントの種類がわかりやすく掲載されているページをご紹介します。
https://qiita.com/mzmz__02/items/873118fbd8723c44956d

残りのボタンの解答例

```
    btn2.addEventListener('click', e=>{
        shikaku1.style.backgroundColor = "#00ff00";
        btn.disabled = false;
        btn2.disabled = true;
    }, false);
    btn3.addEventListener('click', function(){
        shikaku1.style.borderRadius = "50%";
        btn3.disabled = true;
        btn4.disabled = false;
    }, false);
    btn4.addEventListener('click', function(){
        shikaku1.style.borderRadius = "0%";
        btn3.disabled = false;
        btn4.disabled = true;
    }, false);
```

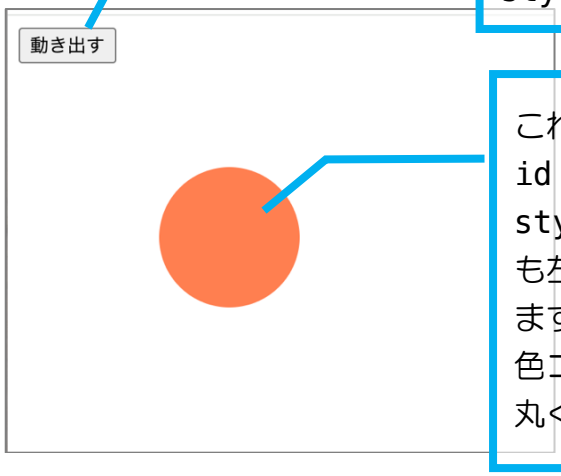
第３章　定期的に繰り返す処理と条件分岐

この章では、定期的に繰り返す処理を学びます。
下に**動き続ける**処理を例にとして学んでいきましょう。

3-1　準備と復習

【演習１】
いきなりの演習ですが、これまでの復習です。
下図のように表示されるコードを書いてください。
`kouza03-1.html` というファイル名にしましょう。

要素は二つ、［動き出す］ボタンと●。



［動き出す］ボタン。`button` タグでつくってください。
`id` は `move_btn` にしておきましょう。
`style` の設定は不要です。

これまでと同様、`div` タグで作ってください。
`id` もこれまでと同様、`shikaku1` で良いです。
`style` の設定、`position` は `absolute`、上からも左からも `100px` の位置、幅も高さも `100px` とします。塗りつぶし色はサンプルでは `coral` です。
色コードではなく、色の名称でも指定できます。
丸くする方法はわかりますね？復習してください。

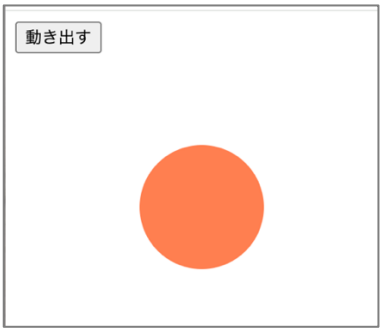
`body` に対しての `position` を `relative` にするのを忘れずに。

解答例は次のページに示します。

解答例　【演習１】kouza03-1.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        body{
            position: relative;
        }

        #shikaku1{
            position: absolute;
            top: 100px;
            left: 100px;
            width: 100px;
            height: 100px;
            background-color: coral;
            border-radius: 50%;
        }
    </style>
</head>
<body>
    <div id="shikaku1"></div>
    <button id="move_btn">動き出す</button>
</body>
</html>
```

【演習２】script エリアを作り、［動き出す］ボタンと●を定数に割り当ててください。
　それぞれ、id と同じ定数名にするとわかりやすいです。



解答例は次のページです。

解答例　kouza03-1.html　【演習２】スクリプトエリアと定数追加

```
<body>
    <div id="shikaku1"></div>
    <button id="move_btn">動き出す</button>
    <script>
        const shikaku1 = document.querySelector("#shikaku1");
        const move_btn = document.querySelector("#move_btn");

    </script>
</body>
```

追加

3-2　変数の定義

①下方向に動かしたい・・・ということは、上下方向の座標の値を扱うことになります。

上下方向の座標の値を扱う「変数」をｙとして定義しましょう。初期値もついでに設定しておきましょう。

●の最初の top の値が y の初期値になります。

次のように追記してください。

変数は let を使って宣言します。（const ではありません。）

```
        #shikaku1{
            position: absolute;
            top: 100px;
            left: 100px;
            width: 100px;
            height: 100px;
            background-color: coral;
            border-radius: 50%;
        }
    </style>
</head>
<body>
    <div id="shikaku1"></div>
    <button id="move_btn">動き出す</button>
    <script>
        const shikaku1 = document.querySelector("#shikaku1");
        const move_btn = document.querySelector("#move_btn");
        let y = 100;

    </script>
```

これをｙの初期値にする。

ｙの初期値の定義を追加

②「動き続ける」ということで、移動量も定義します。下記のように追記してください。ひとまず5にしておきます。

```
<script>
    const shikaku1 = document.querySelector("#shikaku1");
    const move_btn = document.querySelector("#move_btn");
    let y = 100;
    let dy = 5;
```

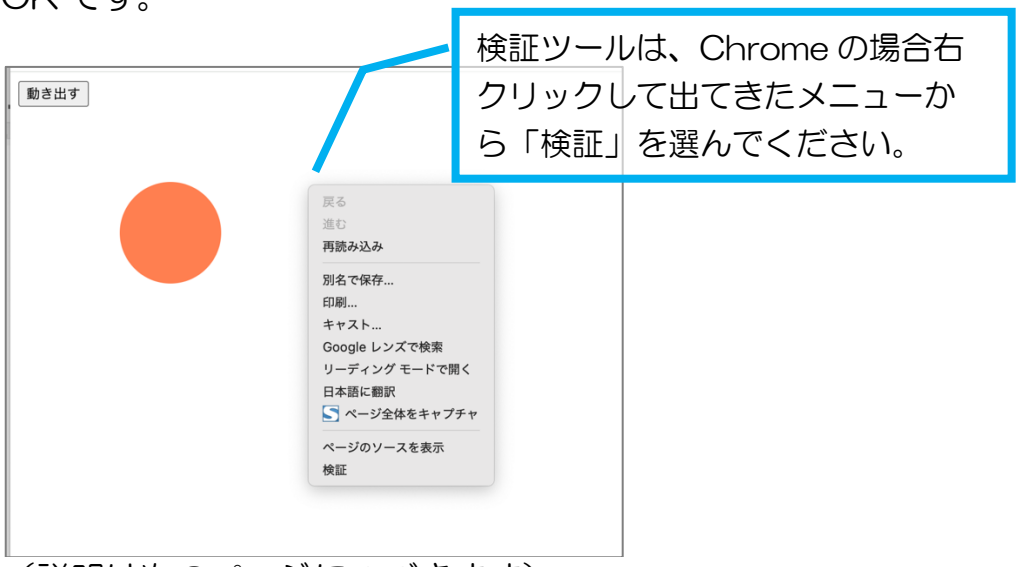
追加

3-3　クリックイベント

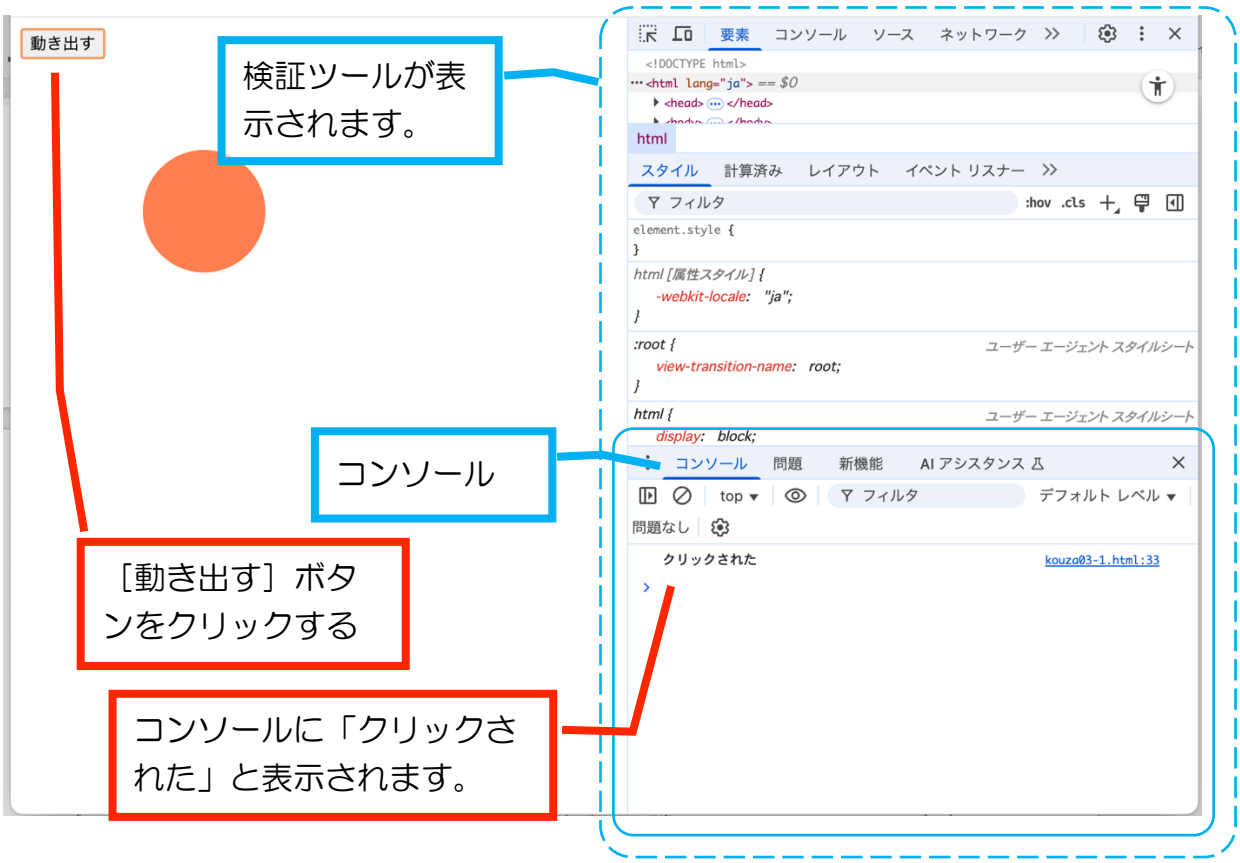
①ボタンが押されたときのイベントリスナーを追加してください。穴埋めにしました。解答例は次のページです。

```
    let y = 100;
    let dy = 5;

    move_btn.(1)(2), (3){
        console.log("クリックされた");
    }, (4));
```

　埋めることができたら、Web ブラウザの検証ツール（開発ツール）を使ってクリックされたイベントを検知していることを確認してください。
　ボタンをクリックしたときに、コンソールに「クリックされた」と表示されればOK です。



（説明は次のページにつづきます）



【前ページの解答例】

(1) `addEventListener`
(2) `'click'`
(3) `e=>`
(4) `false`

②クリックされたら dy の値分下に動くようにしてみましょう。下図のように追記してください。

```
move_btn.addEventListener('click', e=>{
    // console.log("クリックされた");
    y += dy;
    shikaku1.style.top = y + "px";
}, false);
```

コメント化。命令として機能しないようにしておきましょう。削除してもかまいません。

追加

【解説】

y += dy;

元々の y の値に dy を加えます。
y を dy ずつ増やす処理になります。

shikaku1.style.top = y + "px";

変更になった y の値を shikaku1 の style の top に設定しています。
shikaku1 の上からの位置を y px にする処理になります。
ここでは dy = 5 としていますので、ボタンをクリックするたびに 5px 分下にチョコっと動きます。

ブラウザで動作確認してください。

3-4　関数化と setInterval・clearInterval

①このあとの手順で「動き続ける」仕組みをつくりますが、そのための事前準備として前節でつくった「少しだけ動く」命令を関数化しておきましょう。

関数にすると、命令のかたまりをひとまとめにしておくことができます。

変更前

```
move_btn.addEventListener('click', e=>{
    // console.log("クリックされた");
    y += dy;
    shikaku1.style.top = y + "px";
}, false);
```

この命令を関数化します。

変更後

```
move_btn.addEventListener('click', e=>{
    move();
}, false);

function move(){
    y += dy;
    shikaku1.style.top = y + "px";
}
```

move()関数の中身に持っていきます。

move()という関数を呼ぶ構造に書き換えてください。

move()関数をつくります。

関数を呼び出す形に書き換えても、先ほどと同じ動きをすることをブラウザで確認してください。

②move()関数を繰り返し呼び出せば、動かし続けることができるようになります。
　そのしくみを setIntervel という関数（JavaScript にあらかじめ用意されている関数）を使ってつくります。

変更前

```
move_btn.addEventListener('click', e=>{
    move();
}, false);
```

move()関数が呼ばれるのは１回だけ

変更後

```
move_btn.addEventListener('click', e=>{
    timeId = setInterval("move()", 10);
}, false);
```

move()関数を 10ms ごとに繰り返し呼び続ける

繰り返し呼び出す対象の関数名

呼び出すインターバル。単位はミリ秒。ms。

timeId は後ほど、呼び出し続けている状況を停止するときに使います。
変数としての宣言も追加しておきましょう。
（宣言していなくても動くことは動きますが、バグの内包になりかねないのでお勧めしかねます。）

```
<script>
    const shikaku1 = document.querySelector("#shikaku1");
    const move_btn = document.querySelector("#move_btn");
    let y = 100;
    let dy = 5;
    let timeId;
```

追加

　ブラウザで［動き出す］ボタンを押して、動き続けるようになっていることを確認してください。

【演習】
　いつまでも動き続けているとメモリや電力の無駄になります。［止める］ボタンを作り、クリックされたら止まるようにしてください。

　［止める］ボタンはわかりやすく id = “stop_btn”にしましょう。
　stop_btn に対して addEventlistner をつくり、クリックされたイベントの処理を書いてください。
　setInterval で動いている状態を止めるには、次のコードを使ってください。

```
clearInterval(timeId);
```

　解答例は次のページに示します。

　動作確認の際、動きが速すぎる場合は dy の値を小さくして動くスピードを遅くしてみましょう。

演習の解答例

html 部

```
<body>
    <div id="shikaku1"></div>
    <button id="move_btn">動き出す</button>
    <button id="stop_btn">止める</button>
    <script>
```

追加

JavaScript 部　ボタンを定数に割り当てる部分

```
<script>
    const shikaku1 = document.querySelector("#shikaku1");
    const move_btn = document.querySelector("#move_btn");
    const stop_btn = document.querySelector("#stop_btn");
    let y = 100;
    let dy = 1;
```

追加

JavaScript 部　動きを止める部分

```
        shikaku1.style.top = y + "px";
    }

    stop_btn.addEventListener('click', function(){
        clearInterval(timeId);
    }, false);

</script>
```

追加

【課題】
●をクリックしたときも止まるようにしてください。

解答例は次のページです。

解答例　3-4【課題】

```
    shikaku1.addEventListener('click', function(){
        clearInterval(timeId);
    }, false);
</script>
```

追加

3-5　条件による処理　if

ある程度下に行ったらまた上から出てくるようにしてみます。
「ある程度下にいった」という条件を判定することになります。

move()関数の中に、次のように追記してください。

```
function move(){
    y += dy;
    shikaku1.style.top = y + "px";

    if(y > 700){
        y = 50;
    }
}
```

追加

ブラウザで動作確認してみましょう。

【解説】

```
if(y > 700){
    y = 50;
}
```

もしyが700より大きい場合は、yを50にするという命令です。

【課題】
kouza03-1をコピーしてkouza03-1aを作ってください。
下方向ではなく右方向に動き続け、ある程度右（例えば700px）まで行ったら左から出てくるように書き換えてください。
横方向は変数xを定義してください。横方向への移動量も同様にdxとしましょう。

解答例　3-5【課題】

```
<body>
    <div id="shikaku1"></div>
    <button id="move_btn">動き出す</button>
    <button id="stop_btn">止める</button>
    <script>
        const shikaku1 = document.querySelector("#shikaku1");
        const move_btn = document.querySelector("#move_btn");
        const stop_btn = document.querySelector("#stop_btn");
        let x = 100;
        let dx = 5;
        let timeId;

        move_btn.addEventListener('click', e=>{
            timeId = setInterval("move()", 10);
        }, false);

        function move(){
            x += dx;
            shikaku1.style.left = x + "px";

            if(x > 700){
                x = 50;
            }
        }

        stop_btn.addEventListener('click', function(){
            clearInterval(timeId);
        }, false);

        shikaku1.addEventListener('click', function(){
            clearInterval(timeId);
        }, false);
    </script>
</body>
```

3-6　キー入力の受け取り

3-6-1　キー入力を受け取る

新しいファイルを作り、kouza03-2.html としてください。
押されたキーが判別できていることをコンソールで見てみましょう。以下のコードを書いてください。

```
<body>
    <script>
        let keydown = '';

        window.addEventListener('keydown', e => {
            keydown = e.key;
            console.log(keydown);
        }, false);
        window.addEventListener('keyup', e => {
            keydown = '';
        }, false);
    </script>
</body>
```

【解説】
イベントリスナーで'keydown'（キーが押された）と'keyup'（キーが離された）ことを受け取っています。

let keydown として定義した変数は、押されたキーのキーコードを保持するためのものです。初期値は''として何も入っていない状態にしています。

'keydown'のイベントが発生したら、変数 keydown に e.key を代入しています。e.key は、発生したイベントのキーコードです。
console.log(keydown)はコンソールに keydown の値を表示します。これで押されたキーのキーコードをコンソールで見ることができます。

'keyup'のイベントが発生した場合は変数 keydown の中身を''として何も入っていない状態にします。

ブラウザで動作確認してください。
キーを押すたびに、コンソールに押されたキーのキーコードが表示されます。

3-6-2　受け取ったキーコードを表示してみよう

①表示する場所を次のように追加してください。

②また、その場所を JavaScript で制御できるように定数としての定義を追加してください。

```
<body>
    <div id="hyouji"></div>
    <script>
        let keydown = '';
        const hyouji = document.querySelector("#hyouji");
```

追加①

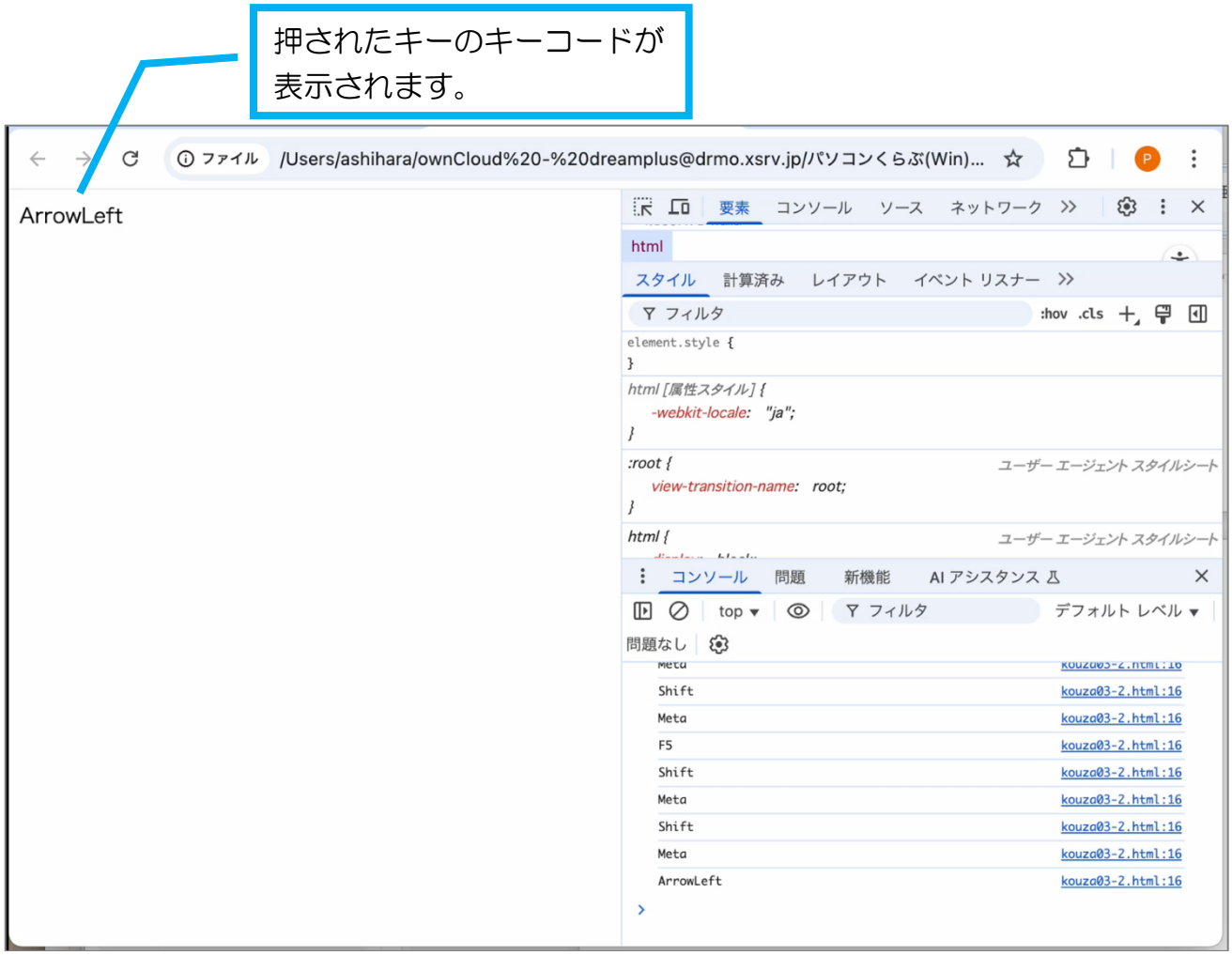
追加②

③押されたキーのキーコードが表示されるように、キーが押された時のイベントリスナーに追加してください。

```
    window.addEventListener('keydown', e => {
        keydown = e.key;
        console.log(keydown);
        hyouji.textContent = keydown;
    }, false);
```

追加③

動作確認してください。


押されたキーのキーコードが
表示されます。
ファイル
/Users/ashihara/ownCloud%20-%20dreamplus@drmo.xsrv.jp/パソコンくらぶ(Win)...
ArrowLeft
要素
コンソール
ソース
ネットワーク
html
スタイル
計算済み
レイアウト
イベント リスナー
フィルタ
:hov .cls
element.style {
}
html [属性スタイル] {
-webkit-locale: "ja";
}
:root {
ユーザー エージェント スタイルシート
view-transition-name: root;
}
html {
ユーザー エージェント スタイルシート
コンソール
問題
新機能
AI アシスタンス
top
フィルタ
デフォルト レベル
問題なし
Shift kouza03-2.html:16
Meta kouza03-2.html:16
F5 kouza03-2.html:16
Shift kouza03-2.html:16
Meta kouza03-2.html:16
Shift kouza03-2.html:16
Meta kouza03-2.html:16
ArrowLeft kouza03-2.html:16

3-7　キーボードで動かす

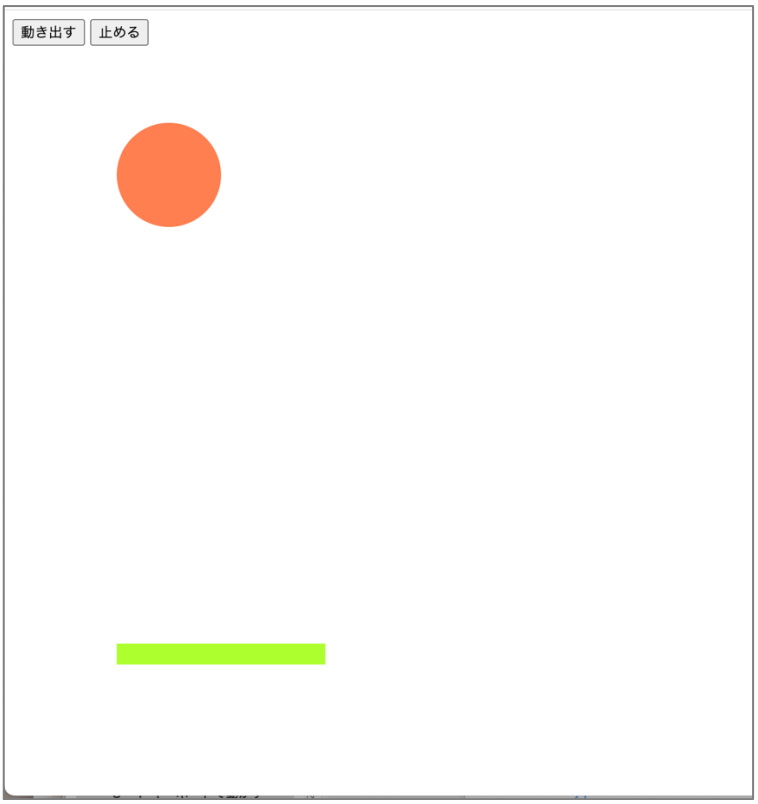
　前節で学んだキー入力を受け取るしくみを使って、html 内の要素をキー入力で動かす方法を学びましょう。

　kouza03-1.html をコピーし、kouza03-3.html としてください。

3-7-1　準備・復習

【演習（１）】
下図に示すような、黄緑色のバーを追加してください。

　id は bar1 としましょう。（任意ですので別名でもかまいませんが、本テキスト内では以下 bar1 として進めます。）
　上から 600px、左から 100px の位置、幅 200px，高さ 20px、塗りつぶし色 greenyellow;です。



解答例は次のページに示します。

解答例【演習（１）】

```
        #bar1{
            position: absolute;
            top: 600px;
            left: 100px;
            width: 200px;
            height: 20px;
            background-color: greenyellow;
        }

    </style>
</head>
<body>
    <div id="shikaku1"></div>
    <div id="bar1"></div>
    <button id="move_btn">動き出す</button>
    <button id="stop_btn">止める</button>
    <script>
```

追加

追加

【演習（２）】

押されたキーを取得する仕組みと、キーが離された時の仕組みを JavaScript に追加してください。押されたキーが判別されていることを console.log で確認できるようにし、動作確認してください。

３-６を参考にしてください。

押されたキーのキーコードを代入するための変数 keydown の定義も必要です。

解答例は次のページに示します。

解答例【演習（２）】

押されたキーのキーコードを代入するための変数定義

```
<script>
    const shikaku1 = document.querySelector("#shikaku1");
    const move_btn = document.querySelector("#move_btn");
    const stop_btn = document.querySelector("#stop_btn");
    let y = 100;
    let dy = 1;
    let timeId;
    let keydown = '';
```

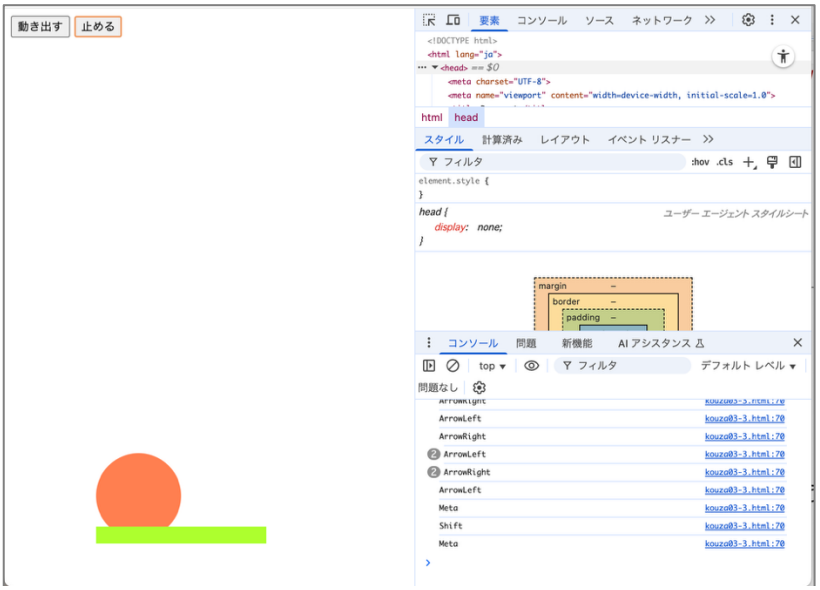
追加

キーが押されたときのキーコードを変数に代入する部分とコンソール表示、キーが離された時の処理

```
    window.addEventListener('keydown', e => {
        keydown = e.key;
        console.log(keydown);
    }, false);
    window.addEventListener('keyup', e => {
        keydown = '';
    }, false);

</script>
```

追加

キー入力を取得できていることを動作確認してください。



キーボードを押して、表示されることを確認。

3-7-2　キー入力で動かす

①bar1 を JavaScript から扱えるようにしましょう。

②bar1 の横方向の位置の値となる変数 bar_x を定義し、初期値を 100 にしましょう。

③bar1 の移動量の値となる変数 bar_dx を定義し、値を 5 にしましょう。

```
<script>
    const shikaku1 = document.querySelector("#shikaku1");
    const move_btn = document.querySelector("#move_btn");
    const stop_btn = document.querySelector("#stop_btn");
    const bar1 = document.querySelector("#bar1");
    let y = 100;
    let dy = 1;
    let timeId;
    let keydown = '';
    let bar_x = 100;
    let bar_dx = 5;
```

追加①

追加②

追加③

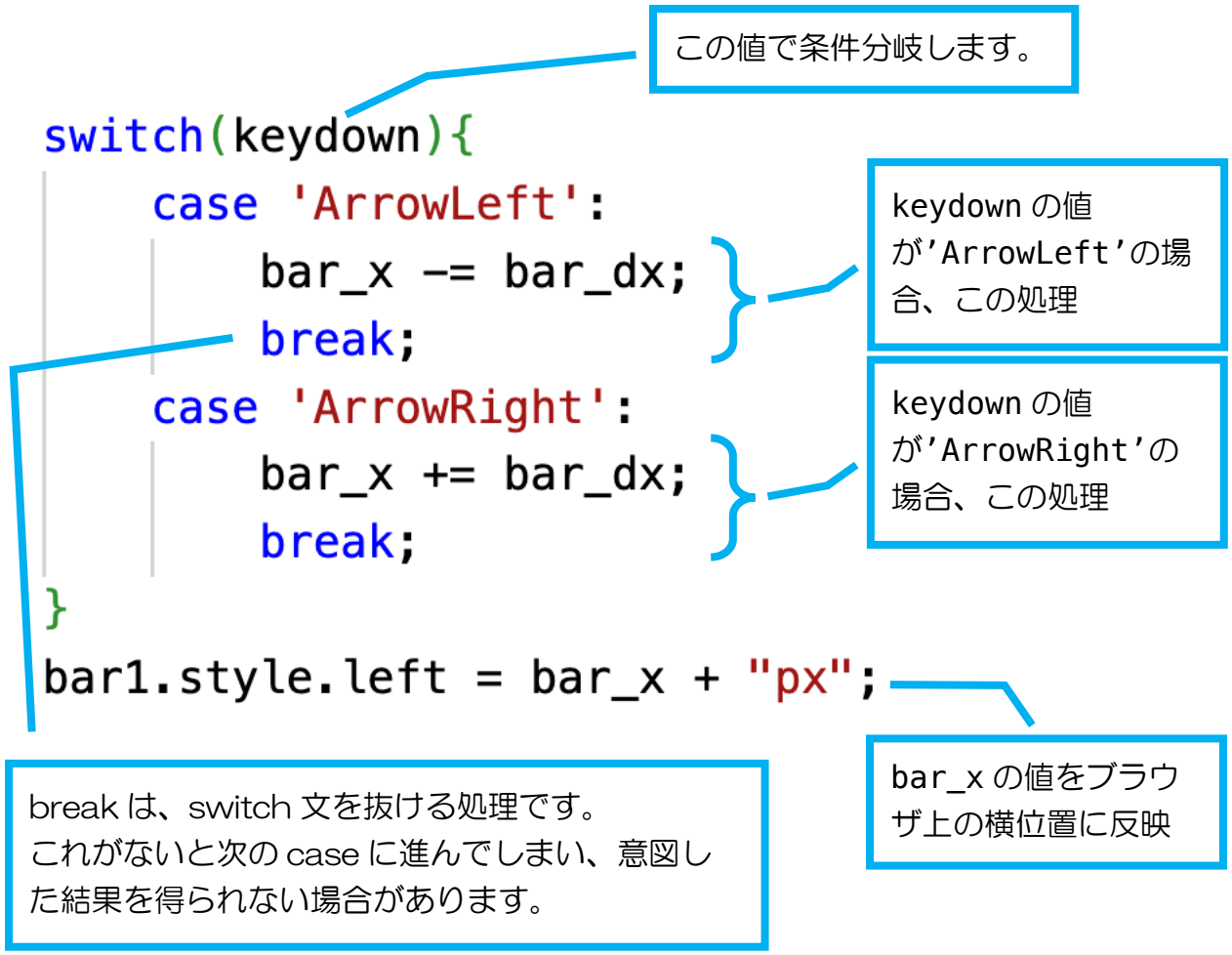
④ずっと動いている move 関数内で動かす処理をつくりましょう。
　押されたキーの情報は変数 keydown に入っています。
　左向き矢印キーの場合は左へ動き、右向き矢印キーの場合は右へ動くように、下記のように追加してください。

```
function move(){
    y += dy;
    shikaku1.style.top = y + "px";

    if(y > 700){
        y = 50;
    }

    switch(keydown){
        case 'ArrowLeft':
            bar_x -= bar_dx;
            break;
        case 'ArrowRight':
            bar_x += bar_dx;
            break;
    }
    bar1.style.left = bar_x + "px";

}
```

追加

bar1 が矢印キーで左右に動くことを動作確認してください。

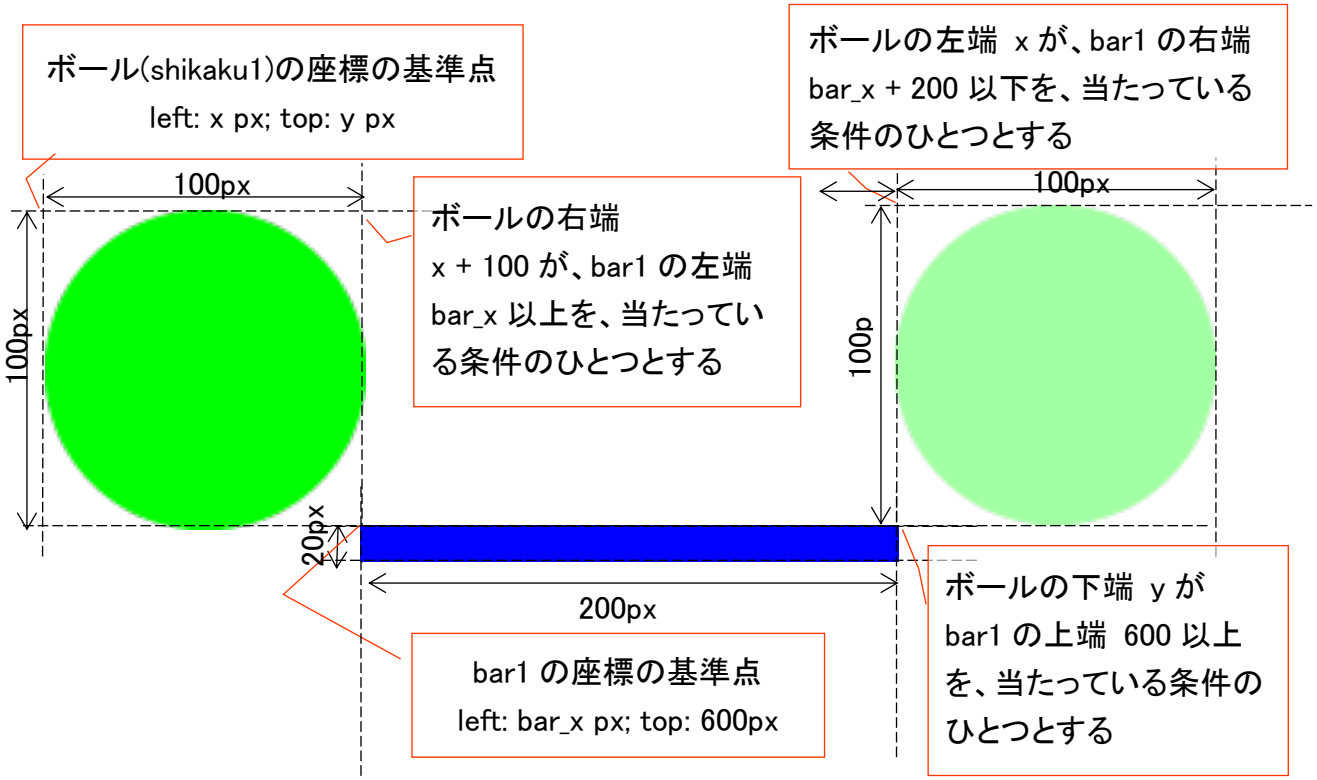
【解説】
switch 文が初めて出てきました。
if 文と同様に、条件分岐する構文です。



3-8　ボールがバーに当たったら跳ね返る

【考え方】

bar の上端に当たる判定。この場合、 y 方向（上下の動く方向）を逆にします。



したがって、

(x + 100 >= bar_x) && (x <= bar_x + 200) && (y + 100 >= 600) の場合、

dy = -dy

という処理になります。

以上をコードにしましょう。

①ボール（shikaku1）の横方向の位置の値を x として定義し、初期値を 100 としましょう。

```
const bar1 = document.querySelector("#bar1");
let x = 100;
let y = 100;
let dy = 1;
```

追加。

② move()関数内でボールの横方向の座標に x を反映させましょう。
　現時点ではボールは横方向に動きませんので、事実上はあまり意味をなしませんが、将来的に横方向にも動かす場合はこれは必要です。

```
function move(){
    shikaku1.style.left = x + "px";
    y += dy;
    shikaku1.style.top = y + "px";
```

追加。

③当たり判定と当たった時の挙動を下記のように move()関数の中に追加してください。

```
    bar1.style.left = bar_x + "px";

    if((x+100 >= bar_x) && (x <= bar_x+200) && (y+100 >= 600)){
        dy = -dy;
    }
```

追加。

　動作確認してみましょう。跳ね返れば OK です。
　これで、ちょっとしたゲームくらいなら作れそうですね。

【補足説明】&& は「かつ」

（条件 A）&&（条件 B）　と書いた場合、条件 A と条件 B の両方を満たす（両方が真）かどうかを評価します。

ちなみに、「または」は || です。
（条件 A）||（条件 B）　と書いた場合、条件 A と条件 B のどちらかを満たす（どちらかが真）かどうかを評価します。

第４章　配列

この章では、配列を使うことを学びます。画像の表示も合わせて学びます。
まずは、一定時間で画像が次々に切り替わるスライドショーをつくってみましょう。

フォルダをひとつ作りましょう。フォルダ名は任意ですが、サンプルでは slide というフォルダ名にします。

その中に３つの画像ファイル photo01.jpg、photo02.jpg、photo03.jpg を入れてください。(「https://drmpls.com/javascript_super_intro/」からダウンロード・または講師からもらってください。）ダウンロードした画像のファイル名は適宜変更してください。

同じフォルダ内に slideshow01.html を新規に作ってください。
下記のように img タグ（id="image"とする）を書いてください。
script にはその img タグを定数 image に割り当ててください。

```
<body>
    <img id="image">
    <script>
        const image = document.querySelector("#image");

    </script>
```

4-1　配列　（スライドショー編）

これからつくるもののサンプル
https://js.drmpls.com/slide/slideshow01.html

これまで出てきた変数や定数は、ひとつの値しか入れられませんでした。
配列には、値を複数入れることができます。値を入れる引き出しがたくさんあるタンスのようなものをイメージしてください。

では、slideshow01.html の JavaScript に次の配列を追加してください。フォルダにある３つの画像ファイル名です。

```
<script>
    const image = document.querySelector("#image");
    let photos = ["photo01.jpg", "photo02.jpg", "photo03.jpg"];
```

追加

理解を深めるために、コンソールで表示する命令も追加してください。

```
let photos = ["photo01.jpg", "photo02.jpg", "photo03.jpg"];
console.log(photos);
```

追加

ブラウザで表示し、検証ツールのコンソールを見てみましょう。

`Array(3)`・・・・ ３つの要素がある配列という意味です。
`0, 1, 2`・・・・・ インデックス番号です。1 番目の要素はインデックス番号が `0` です。2 番目が 1、3 番目が 2 です。
`length: 3`・・・・ 配列の長さ＝要素の数です。この場合は 3 です。

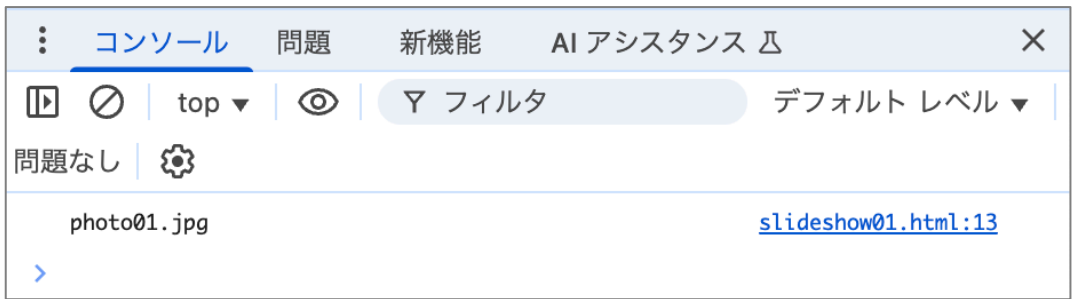
配列から中身の１つの要素を取り出したい場合は、次のように配列名とインデックス番号で指定します。
配列 `photos` から最初の要素を取り出したい場合、`photos[0]`と指定します。

実際にやってみましょう。
`console.log` の中身を次のように書き換えてください。

```
let photos = ["photo01.jpg", "photo02.jpg", "photo03.jpg"];
console.log(photos[0]);
```

最初の要素を指定するように書き換え

コンソールで見てください。配列 photos の最初の要素が表示されましたね。



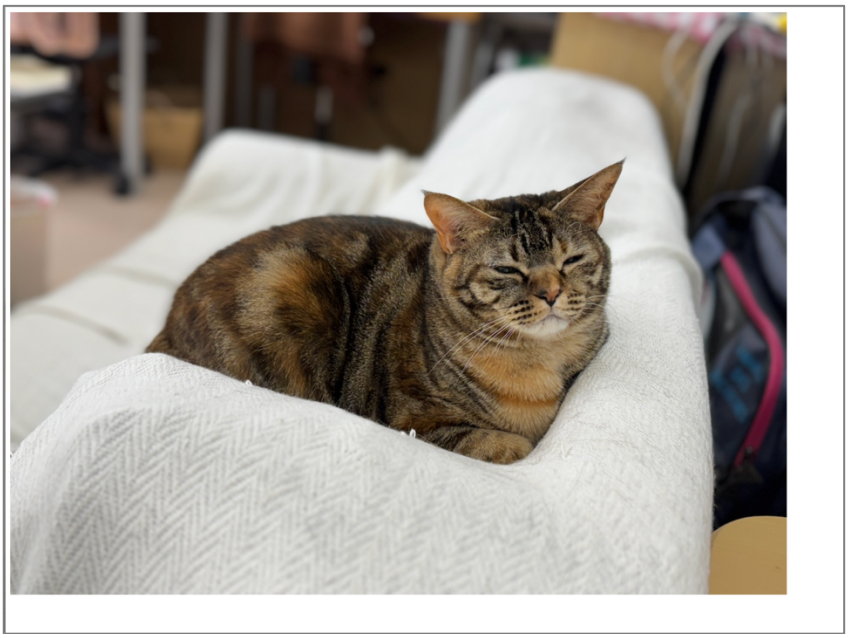
4-2　画像表示　（スライドショー編）

配列 photos の中のひとつの画像ファイル名を使って画像表示してみましょう。
下記のように追記してください。
id="image"の画像表示に配列 photos の最初の要素を指定します。

```
let photos = ["photo01.jpg", "photo02.jpg", "photo03.jpg"];
// console.log(photos[0]);

image.src = photos[0];
```

追加

ブラウザで表示してみましょう。



【演習】

```
image.src = photos[0];
```

上記のコードで配列 photos のインデックス番号を書き換えて、２番目の画像、３番目の画像が表示されることを確認してください。

解答例は省略します。

配列のインデックス番号を指定することで配列の要素を指定できることを実感できることと思います。
画像を次々に切り替えるには、これを使います。

4-3　一定時間で切り替えるしくみをつくる　（スライドショー編）

しくみとしては、

photos[0] → photos[1] → photos[2] → photos[3] → ・・・・

というようにインデックスを切り替えていけば画像が次々と切り替わっていきます。

そこで、image に画像をセットする関数をつくり、それを setInterval で一定時間ごとに呼び出すようにします。

①まず、インデックス用の変数 idx を定義し、初期値0にしましょう。
②setInterval を使う時のお決まりの変数 timeId も用意しましょう。

```
const image = document.querySelector("#image");
let photos = ["photo01.jpg", "photo02.jpg", "photo03.jpg"];
// console.log(photos[0]);
let idx = 0;
let timeId;
```

追加①

追加②

③最初に表示される画像をセットしましょう。

```
let idx = 0;
let timeId;

image.src = photos[idx++];
```

追加

もともとは photos[0]になっていました。
idx の初期値を0にしていますので、ページを開くと今までと変わることなく photos[0]の画像が表示されます。

idx++ の ++ は、1 増やす命令です。
このコードが実行された（photos[0]がセットされた）あとに idx の値を1増やして idx の値が1になります。
ですので、photos[idx]が実行されるときは今表示された画像の次の画像になります。

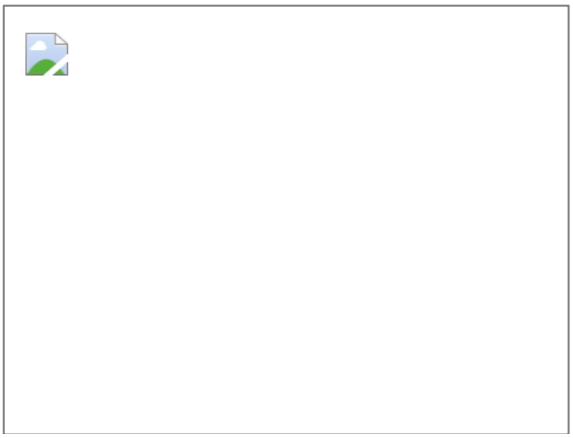
④setInterval で slide()関数を 2 秒ごとに呼ぶようにしましょう。
slide()関数には、画像を表示する命令を入れてください。③で追加した命令と同じです。

```
image.src = photos[idx++];

timeId = setInterval('slide()', 2000);

function slide(){
    image.src = photos[idx++];

}
```

追加

ここまでできたら、ブラウザで表示してみましょう。

画像が 2 秒ごとに切り替われば OK です。
が、３枚表示されたら次は下図のように表示されなくなりますね。



⑤idx が２を超えたら idx = 0 にしましょう。
ヒントなしでできれば素晴らしいです。挑戦してみてください。
解答例は次のページです。

```
function slide(){
    image.src = photos[idx++];

    if(idx > 2) idx = 0;
}
```

追加

if 文の中身が１行だけの場合は、{　}を省略して簡略化した書き方ができます。

【課題１】［止める］ボタンを追加し、スライドショーを停止させられるようにしてください。

【課題２】［始める］ボタンを追加し、スライドショーを始められるようにしてください。ページを読み込んだときにはスライドショーは始まらず、今回追加する［始める］ボタンをクリックして初めて動き出すようにしてください。

【課題３】［始める］ボタンが押されてスライドショーが動いている間は［始める］ボタンを無効（非活性）にしてください。［
　止める］ボタンが押されたら［始める］ボタンを有効（活性）にしてください。

完成例【課題１】

```
<body>
    <img id="image">
    <button id="stop_btn">止める</button>
    <script>
        const image = document.querySelector("#image");
        const stop_btn = document.querySelector("#stop_btn");
        let photos = ["photo01.jpg", "photo02.jpg", "photo03.jpg"];
        // console.log(photos[0]);
        let idx = 0;
        let timeId;

        image.src = photos[idx++];

        timeId = setInterval('slide()', 2000);

        function slide(){
            image.src = photos[idx++];

            if(idx > 2) idx = 0;
        }

        stop_btn.addEventListener('click', e=>{
            clearInterval(timeId);
        }, false);


    </script>
```

追加

追加

追加

完成例【課題２】

```
<body>
    <img id="image">
    <button id="stop_btn">止める</button>
    <button id="start_btn">始める</button>
    <script>
        const image = document.querySelector("#image");
        const stop_btn = document.querySelector("#stop_btn");
        const start_btn = document.querySelector("#start_btn");
        let photos = ["photo01.jpg", "photo02.jpg", "photo03.jpg"];
        // console.log(photos[0]);
        let idx = 0;
        let timeId;

        image.src = photos[idx++];

        start_btn.addEventListener('click', e=>{
            timeId = setInterval('slide()', 2000);
        }, false);

        function slide(){
            image.src = photos[idx++];

            if(idx > 2) idx = 0;

        }

        stop_btn.addEventListener('click', e=>{
            clearInterval(timeId);
        }, false);
```

追加

追加

イベントリスナーの中に setInterval を入れるように変更

完成例【課題３】

```
start_btn.addEventListener('click', e=>{
    timeId = setInterval('slide()', 2000);
    start_btn.disabled = true;
}, false);

function slide(){
    image.src = photos[idx++];

    if(idx > 2) idx = 0;

}

stop_btn.addEventListener('click', e=>{
    clearInterval(timeId);
    start_btn.disabled = false;
}, false);
```

追加

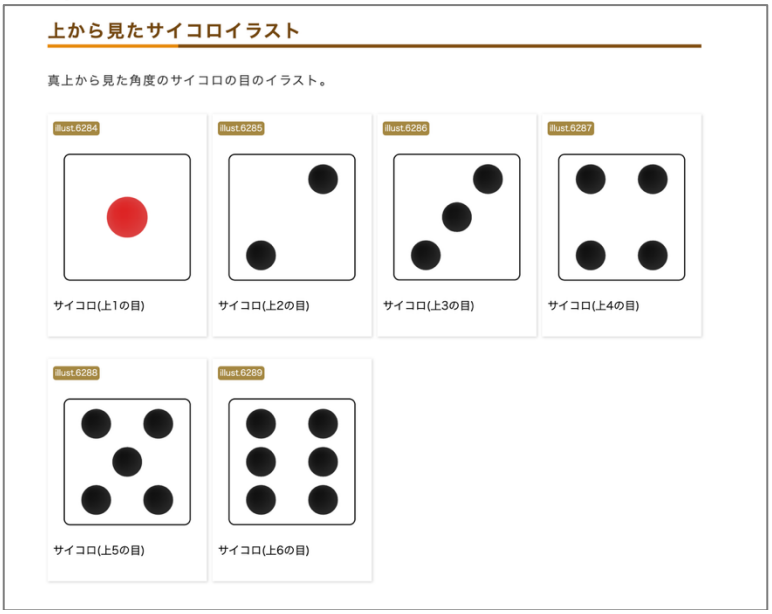
追加

4-4　サイコロを作ってみる　（サイコロ編）

これから作るもののサンプルはこちら。
https://js.drmpls.com/saikoro/saikoro.html

　前節までの知識を使うと、６つの目がコロコロと切り替わるサイコロを作ることができます。
　［サイコロを振る］ボタンで動かし、［止める］ボタンで止める。
　前節のスライドショーとほとんど同じということがわかりますね。

　サイコロの６つの目のイラストは次から入手してください。
　無料で配布してくださっているサイトです。ありがたいことです。
https://chicodeza.com/freeitems/saikoro-illust.html



　saikoro フォルダーを作り、サイコロの６つのイラストを入れてください。

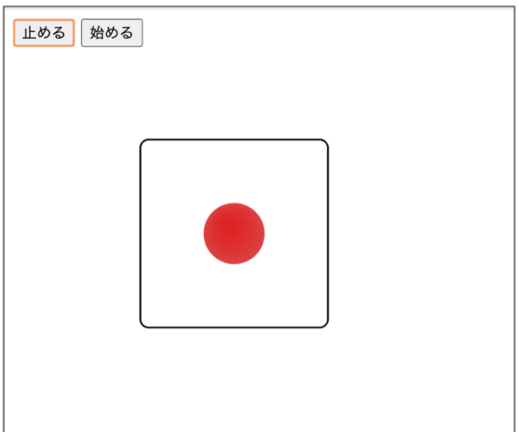
【演習】
次の手順で、サイコロを作ってみましょう。

①新規 html ファイルを作り、saikoro フォルダーに saikoro.html として保存してください。

②サイコロの１の目のイラストを表示させましょう。
id は'saikoro'にしておきましょう。
top, left ともに 100px の位置にしてください。
body に position: relative;を忘れずに、id='saikoro'の画像には position: absolute;を忘れずに。

③［サイコロを振る］ボタン（id='start_btn'）、［止める］ボタン（id='stop_btn'）を追加しましょう。
④［サイコロを振る］ボタンをクリックするとサイコロの目が 1～6 まで順に切り替わり続け、［止める］ボタンをクリックすると切り替わりが止まるようにしましょう。
　前節のスライドショーと同様のしくみで作れます。切り替わりの時間は短い方が良いですが、作成時点では動作確認のためにも 1 秒にしておきましょう。完成時にもっと短時間にします。
　ボタンの有効・無効もスライドショーと同様にしてください。

⑤画像が意外と大きいので、サイズを小さくしたい場合は幅（または高さ）を style にて適当なサイズに指定してください。下図の例は幅を 200px にしたものです。



解答例は以下です。

saikoro.html　　style の部分

```
    <style>
        body{
            position: relative;
        }
        #image{
            position: absolute;
            top: 100px;
            left: 100px;
            width: 200px;
        }
    </style>
</head>
```

saikoro.html　html、script の部分

```
<body>
    <img id="image">
    <button id="stop_btn">止める</button>
    <button id="start_btn">始める</button>
    <script>
        const image = document.querySelector("#image");
        const stop_btn = document.querySelector("#stop_btn");
        const start_btn = document.querySelector("#start_btn");
        let photos = ["saikoro-illust1.png",
                      "saikoro-illust2.png",
                      "saikoro-illust3.png",
                      "saikoro-illust4.png",
                      "saikoro-illust5.png",
                      "saikoro-illust6.png"];
        let idx = 0;
        let timeId;

        image.src = photos[idx++];

        start_btn.addEventListener('click', e=>{
            timeId = setInterval('slide()', 1000);
            start_btn.disabled = true;
        }, false);

        function slide(){
            image.src = photos[idx++];
            if(idx > 5) idx = 0;
        }

        stop_btn.addEventListener('click', e=>{
            clearInterval(timeId);
            start_btn.disabled = false;
        }, false);

    </script>
</body>
```

見やすさのため改行しています。改行しなくてもかまいません。

4-5　ランダムに変わるようにしよう　（サイコロ編）

　現状では、インデックス番号 idx が
０ → １ → ２ → ３ → ４ → ５ → ６ → ０ → １ → ２ → ３ →……
という規則的なループです。

slide()関数の中身を次のように書き換えてください。

```
function slide(){
    idx = Math.floor(Math.random() * 6);
    image.src = photos[idx];
}
```

【解説】

Math.random()
　０～1 未満の範囲でランダムな値を生成する関数です。
　６をかけることにより、０～６未満の範囲のランダムな数値が生み出されます。

Math.floor()
　整数に切り捨てる関数です。
　Math.random()の値は０～６未満の数値ですので、整数に切り捨てることによって０～５のどれかの数字になります。

変更前に idx++になっていた部分は、idx になりました。
idx の値がランダムになりますので、次の番号へ増加させる必要がなくなったためです。

【課題】
切り替わり時間が現状では 1 秒です。
これを 10ms に変更してください。

解答例

```
start_btn.addEventListener('click', e=>{
    timeId = setInterval('slide()', 10);
    start_btn.disabled = true;
}, false);
```

変更

4-6　回転させてみよう　（サイコロ編）

要素に角度をつけることを繰り返すと回転させることができます。

例えば 30° 傾けるには、対象の［要素］に対して次のように記述します。

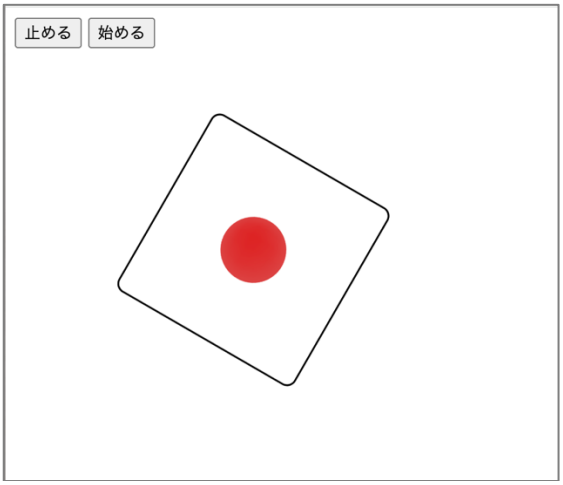
［要素］.style.transform = “rotate(30deg)”

試しに、次の１行を追加してください。
サイコロの画像を表示している id=”image”の img タグに 30° の角度をつけるコードです。

```
let idx = 0;
let timeId;

image.style.transform = "rotate(30deg)";
```

追加

ブラウザで見ると次のようになります。



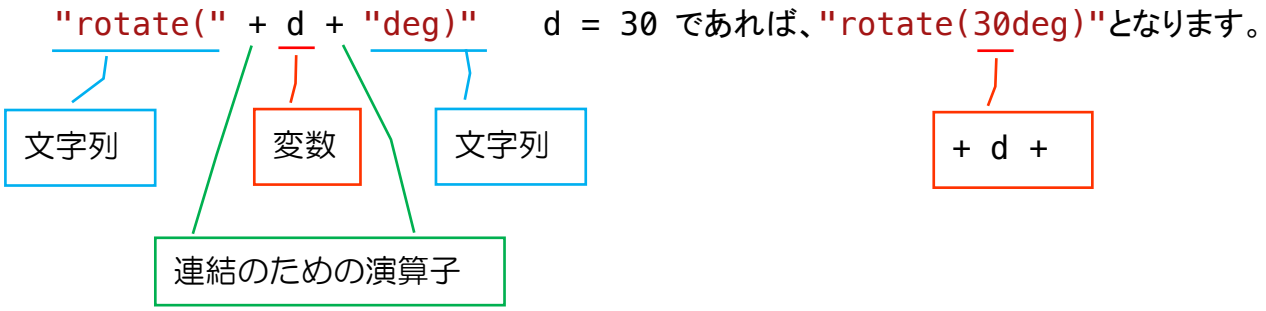
回転させるには角度を次々に変えていく必要がありますので、角度を扱う変数 d を定義して次のように書き換えてみましょう。

```
let idx = 0;
let timeId;
let d = 30;
image.style.transform = "rotate(" + d + "deg)";
```

追加

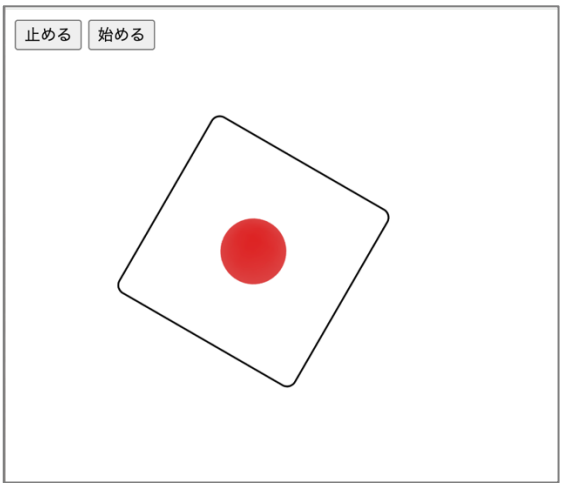
変更

変更した部分の表現が慣れないと難しいと思いますが、文字列と合わせて変数を扱うには「+」を使って文字列と変数をつなぐ方法をとります。

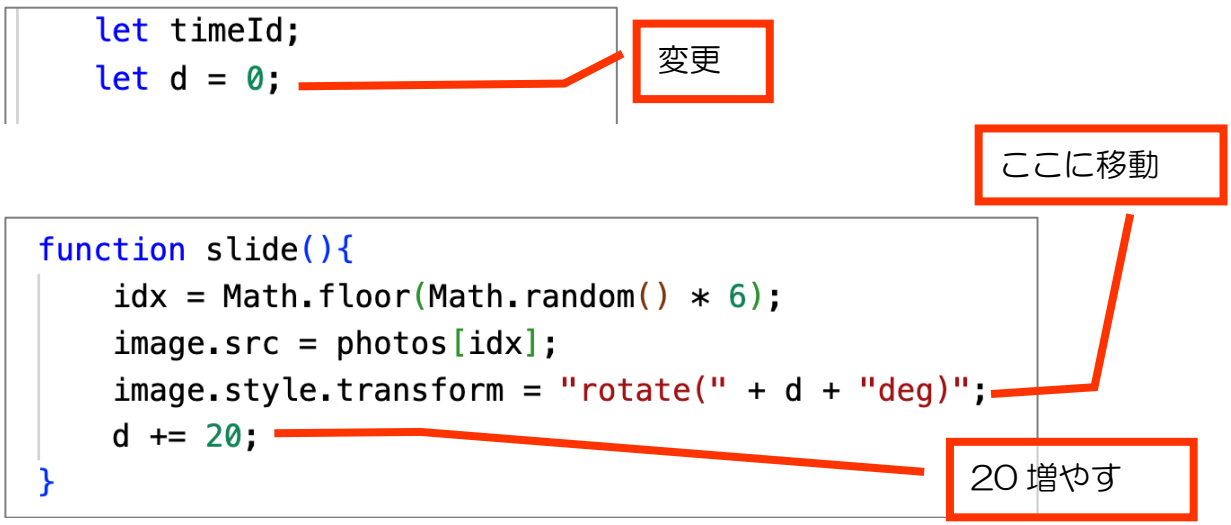


"と"で挟まれている部分が文字列です。

ブラウザで表示して先ほどと変わらないことを確認してください。



では、実際に回転させてみましょう。
d の初期値は 0 にして、角度をつける部分を slide() 関数の中に移動、そのあとに角度を増やす命令を追加してください。



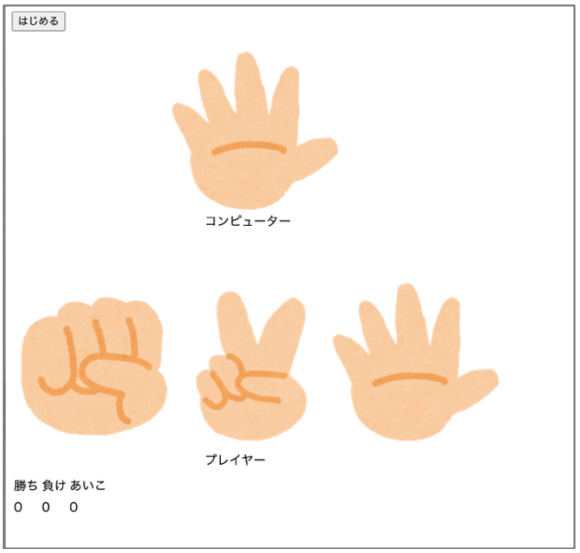
ブラウザで動作確認してみましょう。

回転が速すぎる場合は、setInterval の時間間隔を 100ms くらいに下げてみましょう。

第５章　実践：じゃんけん

　下図に示すようなじゃんけんゲームをつくってみましょう。これまでの知識でほとんど出来上がります。

サンプル　　https://js.drmpls.com/janken/janken.html



【ゲームの内容】

［はじめる］ボタンをクリックするとコンピューターの手がランダムに切り替わります。コンピューターの手が切り替わっている最中は、［はじめる］ボタンを無効にします。

　プレイヤー側のグー・チョキ・パーのどれかをクリックするとコンピューター側の手が決まり（動きが止まる）ます。プレイヤー側の手はクリックしたもの以外は見えなくします。

　勝ち負けの判定結果が表示されます。画面左下の「勝ち　負け　あいこ」のカウントを更新します。［はじめる］ボタンを有効にします。何度でも勝負ができ、その結果がカウントされます。

5-1　表示部分をつくる

① janken フォルダを作ってください。

②グー・チョキ・パーの画像は下記からダウンロード、または講師かから入手してください。

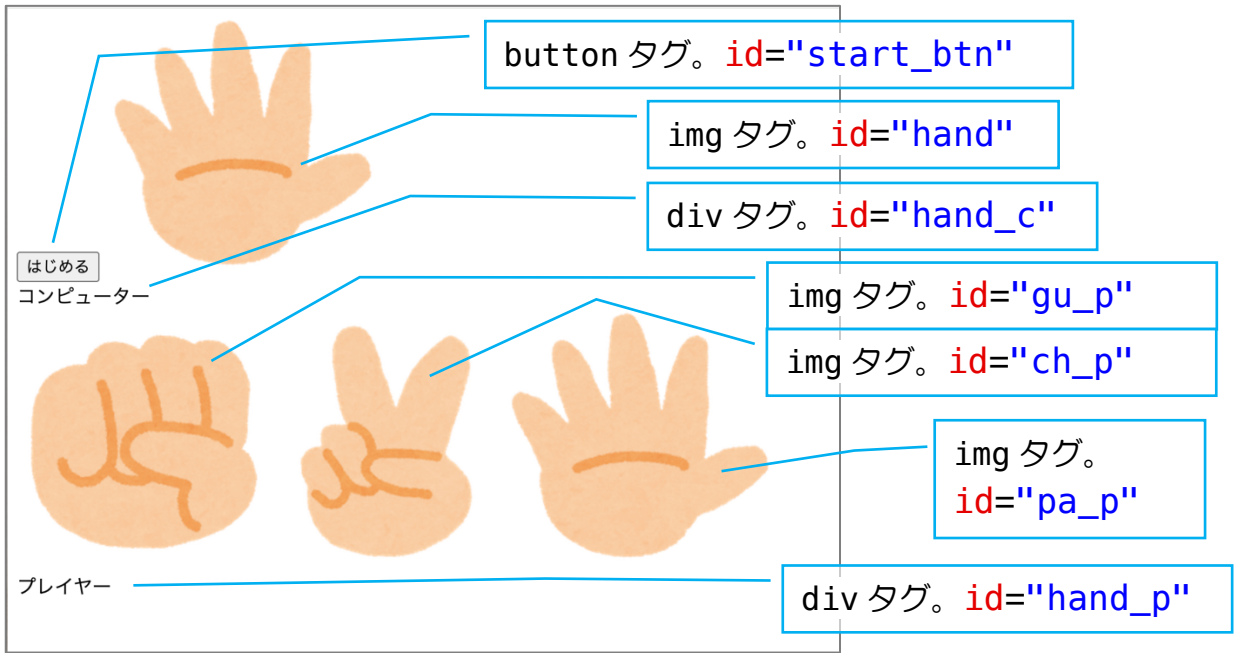
https://drmpls.com/javascript_super_intro/

ダウンロードしたイラストは janken フォルダに入れてください。

（※使用する画像は「いらすとや」から入手し、大きさを調整したものです。）

③ janken.html を作り、janken フォルダに保存してください。

④ janken.html に html の基本構造をつくり、［はじめる］ボタンコンピューター側の手ひとつ、プレイヤー側の手３つ、表示するところまでつくってください。style は次の手順⑤で指定することとして、id は下図の案内のようにしてください。



完成例は次のページです。

④の完成例

```
<body>
    <button id="start_btn">はじめる</button>
    <img id="hand" src="janken_pa.png"">
    <div id="hand_c">コンピューター</div>
    <img id="gu_p" src="janken_gu.png">
    <img id="ch_p" src="janken_choki.png">
    <img id="pa_p" src="janken_pa.png">
    <div id="hand_p">プレイヤー</div>
</body>
```

⑤ style の設定をしてください。
body に対しては、position: relative;
start_btn に対しては、position: absolute; のみでいいです。
hand に対しては、上から 50px、左から 200px にしてください。
hand_c に対しては、上から 260px、左から 250px にしてください。

ここまでの指定で下図のようになります。（コードの完成例は次のページ）
プレイヤー側の手にコンピューター側の手が隠されて見えていません。

⑥プレイヤー側の３つの手は「コンピューター」という文字の下に配置したいので、３つをまとめる div を下図のように追加してください。「プレイヤー」の文字も一緒にまとめてしまいましょう。

```
    <img id="hand" src="janken_pa.png"">
    <div id="hand_c">コンピューター</div>
    <div id="player">
        <img id="gu_p" src="janken_gu.png">
        <img id="ch_p" src="janken_choki.png">
        <img id="pa_p" src="janken_pa.png">
        <div id="hand_p">プレイヤー</div>
    </div>
</body>
```

始まりの div タグ追加

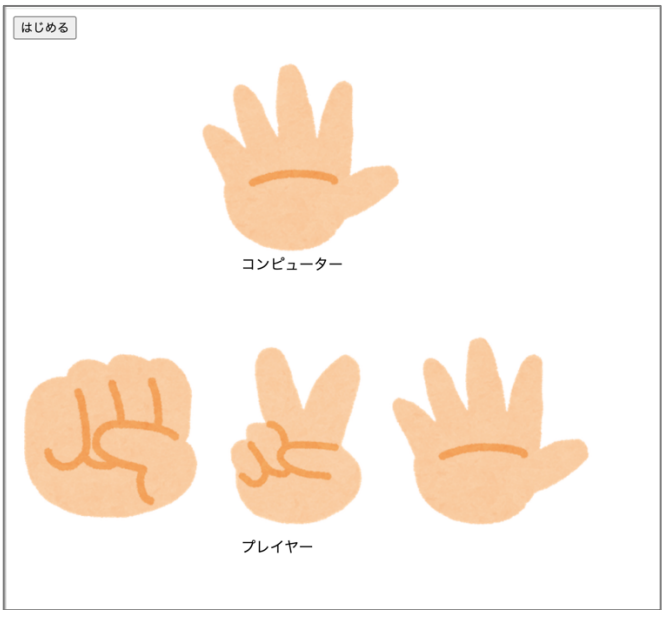
終わりの div タグ追加

構造をわかりやすくするために、インデントを１段階入れて引っ込める。タブキー１回分。

完成例⑤

```
<style>
    body{
        position: relative;
    }
    #start_btn{
        position: absolute;
    }
    #hand{
        position: absolute;
        top: 50px;
        left: 200px;
    }
    #hand_c{
        position: absolute;
        top: 260px;
        left: 250px;
    }
</style>
```

⑦ player に対して、style を上から 350px にしてください。
hand_p に対しては、左から 250px にしてください。
　下図のような見た目になります。



　完成例は次のページです。

完成例⑦

```
    #hand_c{
        position: absolute;
        top: 260px;
        left: 250px;
    }
    #player{
        position: absolute;
        top: 350px;
        left: 250px;
    }
    #hand_p{
        position: absolute;
        left: 250px;
    }
</style>
```

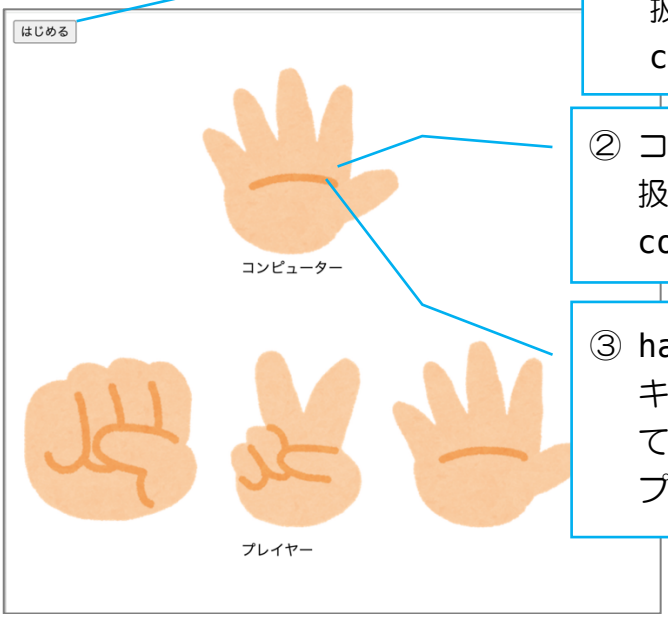
追加

5-2　［はじめる］ボタンがクリックされたときの挙動

【演習１】
　［はじめる］ボタンがクリックされたとき、hand（コンピューター側の手）がランダムにパラパラと切り替わるようにしてください。

　前章のサイコロと同様の方法でできます。

（ヒント・手順）



① ［はじめる］ボタンを JavaSript 内で扱うための定数を定義してください。
`const start_btn = ……`

② コンピューター側の手を JavaSript 内で扱うための定数を定義してください。
`const hand = ……`

③ hand にパラパラと表示させるグー・チョキ・パーの３つの画像を配列として定義してください。（配列名は任意ですが、サンプルでは te としました）

④ 配列の要素を指定するためのインデックスとなる変数（idx）と setInterval を管理する変数（timeId）も定義してください。idx = 0 としておきましょう。

⑤ start_btn に対してクリックされた時のイベントリスナーを設定し、move()関数（次の手順で作る関数）を 100ms のインターバルで呼ぶようにしてください。

⑥ move()関数をつくりましょう。
idx の値を 0～2のランダムな値に設定します。
配列 te のインデックス番号を idx として hand の画像に反映してください。

解答例　5-2【演習１】

```
<script>
    const start_btn = document.querySelector("#start_btn");
    const hand = document.querySelector("#hand");
    const te = ["janken_gu.png", "janken_choki.png", "janken_pa.png"];
    let idx = 0;
    let timeId;

    start_btn.addEventListener('click', e=>{
        timeId = setInterval('move()', 100);
    }, false);

    function move(){
        idx = Math.floor(Math.random() * 3);
        hand.src = te[idx];
    }


</script>
```

5-3　プレイヤー側の手がクリックされたときの処理

【演習】

（1）プレイヤー側のどれかの手がクリックされたらコンピューターの手の動きが止まるようにしてください。

（ヒント）
イベントの処理にて clearInterval を使うとできますね。
３つの手それぞれにイベントを設定する必要があります。

（2）プレイヤー側のどれかの手がクリックされたときの処理に、クリックした手以外の２つが消える（見えなくなる）処理を追加してください。

（ヒント）

```
[要素].style.visibility = "hidden"
```

とすることで見えなくできます。

（3）[はじめる] ボタンクリックで、消えた手がまた見えるようにしてください。

見えるようにするには

```
[要素].style.visibility = "visible"
```

です。

解答例

（1）

定数追加部分

```
<script>
    const start_btn = document.querySelector("#start_btn");
    const hand = document.querySelector("#hand");
    const gu_p = document.querySelector("#gu_p");
    const ch_p = document.querySelector("#ch_p");
    const pa_p = document.querySelector("#pa_p");
    const te = ["janken_gu.png", "janken_choki.png", "janken_pa.png"];
    let idx = 0;
```

追加

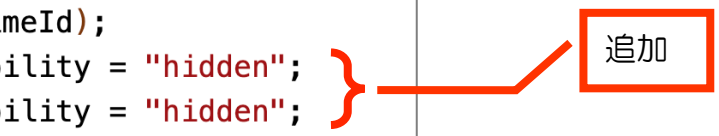
イベント処理部分

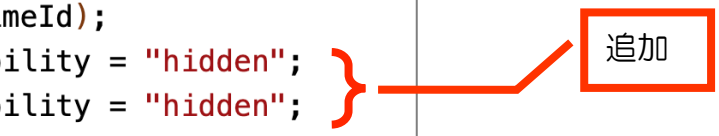
```
    gu_p.addEventListener('click', e=>{
        clearInterval(timeId);
    }, false);

    ch_p.addEventListener('click', e=>{
        clearInterval(timeId);
    }, false);

    pa_p.addEventListener('click', e=>{
        clearInterval(timeId);
    }, false);


</script>
```

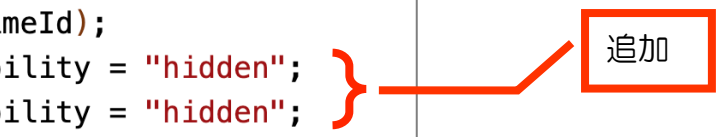
追加

（2）

```
gu_p.addEventListener('click', e=>{
    clearInterval(timeId);
    ch_p.style.visibility = "hidden";
    pa_p.style.visibility = "hidden";
}, false);

ch_p.addEventListener('click', e=>{
    clearInterval(timeId);
    pa_p.style.visibility = "hidden";
    gu_p.style.visibility = "hidden";
}, false);

pa_p.addEventListener('click', e=>{
    clearInterval(timeId);
    gu_p.style.visibility = "hidden";
    ch_p.style.visibility = "hidden";
}, false);
```







（3）

```
start_btn.addEventListener('click', e=>{
    gu_p.style.visibility = "visible";
    ch_p.style.visibility = "visible";
    pa_p.style.visibility = "visible";
    timeId = setInterval('move()', 100);
}, false);
```

5-4　勝ち負けの判定、表示

　勝ち負けの判定をするためには、コンピューター側の手が止まった時の手とプレイヤー側の手を比較します。

　プレイヤーの各手がクリックされたイベント関数内で勝ち負け判定をすれば良さそうですね。

例として、クリックされたプレイヤー側の手がグーだった場合
　コンピューターがグー（idx=0）なら、あいこ
　コンピューターがチョキ（idx=1）なら、勝ち
　コンピューターがパー（idx=2）なら、負け
という処理をすれば良いです。

　次の手順で作っていきましょう。

① 結果を表示するための領域を div タグで作ってください。id=”kekka”としておきましょう。
　style は、上から 150px、左からも 150px、フォントサイズは 10rem としてください。
　動作確認用に「勝ち」を表示してください。下図のような見た目になれば OK です。確認後は「勝ち」を消しておいてください。

```
        #kekka{
            position: absolute;
            top: 150px;
            left: 150px;
            font-size: 10rem;
        }
    </style>
</head>
<body>
    <button id="start_btn">はじめる</button>
    <img id="hand" src="janken_pa.png"">
    <div id="hand_c">コンピューター</div>
    <div id="player">
        <img id="gu_p" src="janken_gu.png">
        <img id="ch_p" src="janken_choki.png">
        <img id="pa_p" src="janken_pa.png">
        <div id="hand_p">プレイヤー</div>
    </div>
    <div id="kekka">勝ち</div>
```

追加

追加

この「勝ち」の文字は、動作確認後に削除してください。

② クリックされたプレイヤー側の手がグーだった場合の処理を追加してください。
コンピューターがグー（idx=0）なら、kekka に「あいこ」を表示
コンピューターがチョキ（idx=1）なら、kekka に「勝ち」を表示
コンピューターがパー（idx=2）なら、kekka に「負け」を表示
という処理です。

kekka に文字を表示するには、

```
kekka.textContent = "あいこ";
```

のようにしてください。

③ ［はじめる］ボタンがクリックされたら kekka の文字を消してください。

kekka の文字を消すには、

```
kekka.textContent = "";
```

のように空文字を入れてください。

④ 文字の色を指定してください。「あいこ」は緑（#00ff00）、「勝ち」は赤（#ff0000）、「負け」は青（#0000ff）とします。

kekka の文字色を指定するには、

```
kekka.style.color = "#00ff00";
```

とします。

解答例　②

kekka を JavaScript で操作できるように定数割り当て

```
const pa_p = document.querySelector("#pa_p");
const kekka = document.querySelector("#kekka");
const te = ["janken_gu.png", "janken_choki.png", "janken_pa.png"];
let idx = 0;
```

追加

勝敗判定と表示

```
gu_p.addEventListener('click', e=>{
    clearInterval(timeId);
    ch_p.style.visibility = "hidden";
    pa_p.style.visibility = "hidden";
    if(idx == 0){
        kekka.textContent = "あいこ";
    } else if(idx == 1){
        kekka.textContent = "勝ち";
    } else if(idx == 2){
        kekka.textContent = "負け";
    }
}, false);
```

追加

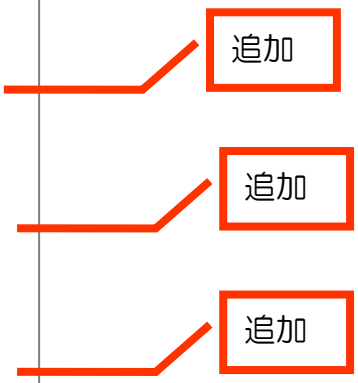
解答例　③

```
start_btn.addEventListener('click', e=>{
    gu_p.style.visibility = "visible";
    ch_p.style.visibility = "visible";
    pa_p.style.visibility = "visible";
    kekka.textContent = "";
    timeId = setInterval('move()', 100);
}, false);
```

追加

解答例　④

```
gu_p.addEventListener('click', e=>{
    clearInterval(timeId);
    ch_p.style.visibility = "hidden";
    pa_p.style.visibility = "hidden";
    if(idx == 0){
        kekka.textContent = "あいこ";
        kekka.style.color = "#00ff00";
    } else if(idx == 1){
        kekka.textContent = "勝ち";
        kekka.style.color = "#ff0000";
    } else if(idx == 2){
        kekka.textContent = "負け";
        kekka.style.color = "#0000ff";
    }
}, false);
```



⑤ プレイヤー側のチョキとパーにも勝ち負けあいこの判定を組み込んでください。

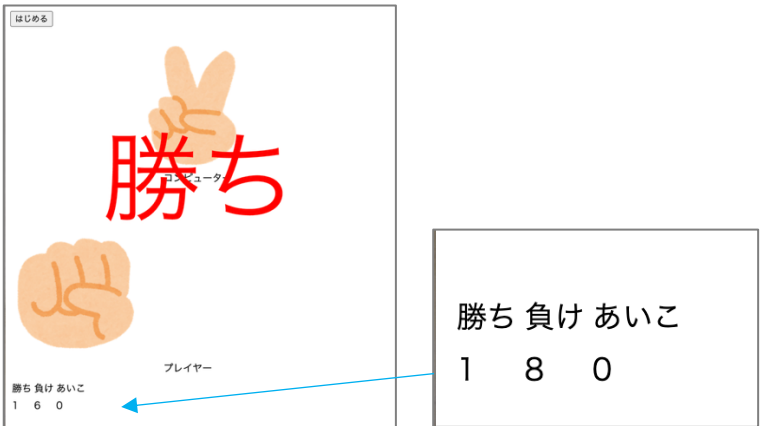
```
ch_p.addEventListener('click', e=>{
    clearInterval(timeId);
    pa_p.style.visibility = "hidden";
    gu_p.style.visibility = "hidden";
    if(idx == 0){
        kekka.textContent = "負け";
        kekka.style.color = "#0000ff";
    } else if(idx == 1){
        kekka.textContent = "あいこ";
        kekka.style.color = "#00ff00";
    } else if(idx == 2){
        kekka.textContent = "勝ち";
        kekka.style.color = "#ff0000";
    }
}, false);

pa_p.addEventListener('click', e=>{
    clearInterval(timeId);
    gu_p.style.visibility = "hidden";
    ch_p.style.visibility = "hidden";
    if(idx == 0){
        kekka.textContent = "勝ち";
        kekka.style.color = "#ff0000";
    } else if(idx == 1){
        kekka.textContent = "負け";
        kekka.style.color = "#0000ff";
    } else if(idx == 2){
        kekka.textContent = "あいこ";
        kekka.style.color = "#00ff00";
    }
}, false);
```

追加

追加

5-5　勝ち・負け・あいこの数を表示

勝ち・負け・あいこの数をカウントして表示するしくみを追加します。



２行３列の表として表示します。表は「table タグ」を使います。

① ２行３列の表を下記のように追加してください。
位置は上から 600px とします。

style 部

```
    #cnt{
        position: absolute;
        top: 600px;
    }
</style>
```

追加

html に table 追加部

追加

```
<div id="kekka"></div>
<table id="cnt">
    <tr><td>勝ち</td><td>負け</td><td>あいこ</td></tr>
    <tr><td id="kachi">0</td><td id="make">0</td><td id="aiko">0</td></tr>
</table>
<script>
```

ブラウザで見ると下図のようになります。

<table>～</table>の部分が「表」になる

<tr>～</tr>の部分が１行

<td>～</td>の部分は表のデータ

② 勝ち・負け・あいこの回数を表示する部分３箇所を JavaScript から操作するための定数に割り当ててください。

```
<tr><td id="kachi">0</td><td id="make">0</td><td id="aiko">0</td></tr>
```

const kachi = ……

const make = ……

const aiko = ……

③ 勝ち・負け・あいこの回数を計上するための変数を定義して０で初期化してください。

変数名は任意ですが、②で kachi, make, aiko を使っていますので使えません。回答例では win, lose, draw を使っています。

④ 勝った時には win を１増やし、負けたときは lose を１増やし、あいこのときは draw を１増やしてください。そして kachi には win の値を表示、make には lose の値を表示、aiko には draw の値を表示してください。

この処理を、まずはグーの場合にだけ追加してください。

解答例　②と③

```
const kekka = document.querySelector("#kekka");
const kachi = document.querySelector("#kachi");
const make = document.querySelector("#make");
const aiko = document.querySelector("#aiko");
const te = ["janken_gu.png", "janken_choki.png", "janke
let idx = 0;
let timeId;
let win = 0;
let lose = 0;
let draw = 0;
```

追加　②

追加　③

解答例　④

```
gu_p.addEventListener('click', e=>{
    clearInterval(timeId);
    ch_p.style.visibility = "hidden";
    pa_p.style.visibility = "hidden";
    if(idx == 0){
        draw++;
        aiko.textContent = draw;
        kekka.textContent = "あいこ";
        kekka.style.color = "#00ff00";
    } else if(idx == 1){
        win++;
        kachi.textContent = win;
        kekka.textContent = "勝ち";
        kekka.style.color = "#ff0000";
    } else if(idx == 2){
        lose++;
        make.textContent = lose;
        kekka.textContent = "負け";
        kekka.style.color = "#0000ff";
    }
}, false);
```

追加

追加

追加

5-6　関数化

チョキとパーの場合にも同様の処理を追加することになりますが、同じ命令が何度も出てきて冗長です。

改めてグーの場合のコードを示します。勝ち・負け・あいこの場合の処理はチョキとパーにもそのまま同じコードを使うことになります。

```
    if(idx == 0){
        draw++;
        aiko.textContent = draw;
        kekka.textContent = "あいこ";
        kekka.style.color = "#00ff00";
    } else if(idx == 1){
        win++;
        kachi.textContent = win;
        kekka.textContent = "勝ち";
        kekka.style.color = "#ff0000";
    } else if(idx == 2){
        lose++;
        make.textContent = lose;
        kekka.textContent = "負け";
        kekka.style.color = "#0000ff";
    }
}, false);
```

(a)あいこの場合

(b)勝ちの場合

(c)負けの場合

そこで、その命令のかたまりを関数化して、必要なときにその関数を呼ぶ方法にしてみましょう。

① あいこの場合（上図で示す(a)）の４行を切り取り、下図のように新たに`draw_disp()`という関数の中身に入れてください。

```
function draw_disp(){
    draw++;
    aiko.textContent = draw;
    kekka.textContent = "あいこ";
    kekka.style.color = "#00ff00";
};
```

切り取って貼り付ける部分

そして、切り取った４行があった部分では`draw_disp()`を呼ぶようにしてください。

```
    if(idx == 0){
        draw_disp();
    } else if(idx == 1){
        win++;
        kachi.textContent = win;
```

関数を呼ぶ

② 同様に、⑹勝ちの場合を`win_disp()`という関数に、⑹負けの場合を`lose_disp()`という関数にして、それぞれのケースで呼び出すしくみに書き換えてください。

動作確認して、関数化する前と動作に違いがないことを確認してください。
確認できたら、次に進んでください。

解答例は次のページです。

③ チョキとパーの場合も、`draw_disp()`、`win_disp()`、`lose_disp()`を呼ぶしくみに書き換えてください。
関数はすでにありますので、呼び出し部のみ変更するだけで良いです。

動作確認して、関数化する前と動作に違いがないことを確認してください。

解答例は次の次のページです。

解答例②

呼び出し部

```
if(idx == 0){
    draw_disp();
} else if(idx == 1){
    win_disp();
} else if(idx == 2){
    lose_disp();
}
```

関数を呼ぶように書き換え

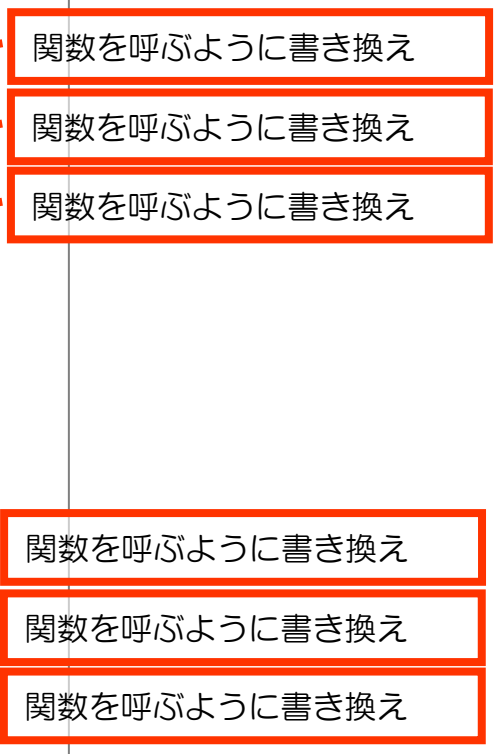
関数を呼ぶように書き換え

関数部　新たに作成。関数の中身は切り取ってきて貼り付け。

```
function win_disp(){
    win++;
    kachi.textContent = win;
    kekka.textContent = "勝ち";
    kekka.style.color = "#ff0000";
};

function lose_disp(){
    lose++;
    make.textContent = lose;
    kekka.textContent = "負け";
    kekka.style.color = "#0000ff";
};
```

解答例③

```
ch_p.addEventListener('click', e=>{
    clearInterval(timeId);
    pa_p.style.visibility = "hidden";
    gu_p.style.visibility = "hidden";
    if(idx == 0){
        lose_disp();
    } else if(idx == 1){
        draw_disp();
    } else if(idx == 2){
        win_disp();
    }
}, false);

pa_p.addEventListener('click', e=>{
    clearInterval(timeId);
    gu_p.style.visibility = "hidden";
    ch_p.style.visibility = "hidden";
    if(idx == 0){
        win_disp();
    } else if(idx == 1){
        lose_disp();
    } else if(idx == 2){
        draw_disp();
    }
}, false);
```



5-7　バグ修正

勝ち・負け・あいこの結果が表示されているときにプレイヤーの手をクリックしてみてください。

結果の数字が増えてしまうことに気づきますね。

この現象を無くすためには、勝ち・負け・あいこの数をカウント増加させる部分に工夫を加えます。

現状では無条件で増加させています。これを次のようにするのはいかがでしょうか？

（１）［はじめる］ボタンを押されたら不活性（disabled=true）にする
（２）プレイヤーの手のクリックイベントの最後に［はじめる］ボタンを活性化（disabled=false）
（３）［はじめる］ボタンが不活性の時だけ勝ち・負け・あいこの数をカウント増加

5-7-1【演習】

（１）（２）の処理を追加してください。

解答例は次のページです。

動作確認では、（１）（２）の処理が問題なく期待通りにできていることを確認してください。

解答例

ボタンを無効（非活性）にする

```
start_btn.addEventListener('click', e=>{
    gu_p.style.visibility = "visible";
    ch_p.style.visibility = "visible";
    pa_p.style.visibility = "visible";
    kekka.textContent = "";
    start_btn.disabled = true;
    timeId = setInterval('move()', 100);
}, false);
```

追加

ボタンを有効（活性）にする
グーの部分

```
gu_p.addEventListener('click', e=>{
    clearInterval(timeId);
    ch_p.style.visibility = "hidden";
    pa_p.style.visibility = "hidden";
    if(idx == 0){
        draw_disp();
    } else if(idx == 1){
        win_disp();
    } else if(idx == 2){
        lose_disp();
    }
    start_btn.disabled = false;
}, false);
```

追加

パー・チョキの部分もグーと同様に、関数最後に追加してください。
最後の追加部のみ示すと下図のようになります。（グーの場合と全く同じ）

```
    }
    start_btn.disabled = false;
}, false);
```

追加

5-7-2　（3）の処理

（3）［はじめる］ボタンが不活性の時だけ勝ち・負け・あいこの数をカウント増加させる処理をつくります。

　if 文を使うことはわかると思いますが、条件は次のようにします。まずは「あいこ」の場合につくってください。

```
function draw_disp(){
    if(start_btn.disabled)draw++;
    aiko.textContent = draw;
    kekka.textContent = "あいこ";
    kekka.style.color = "#00ff00";
};
```

追加

【解説】

if(start_btn.disabled)は、
if(start_btn.disabled == true)と同じ意味です。

if 文の条件の部分は true か false かを評価する部分です。
start_btn.disable は元々true か false かのどちらかの値しか入りません。
ですので、「== true」を書かなくても問題ないのです。
というか、、「== true」があると「頭痛が痛い」みたいな感じになります。

■動作確認では、「あいこ」になったときにプレイヤーの手をさらにクリックして、「あいこ」のカウントが増えないことを確認してください。

【演習】
　win_disp()と lose_disp()にも同様の変更を加えてください。
　動作確認して、勝ち・負け・あいこのどの結果になっても結果が出ている最中はどのカウントも増えないことを確認してください。

解答例は次のページです。

じゃんけんゲーム、完成です。

演習の解答例

```
function win_disp(){
    if(start_btn.disabled)win++;
    kachi.textContent = win;
    kekka.textContent = "勝ち";
    kekka.style.color = "#ff0000";
};

function lose_disp(){
    if(start_btn.disabled)lose++;
    make.textContent = lose;
    kekka.textContent = "負け";
    kekka.style.color = "#0000ff";
};
```

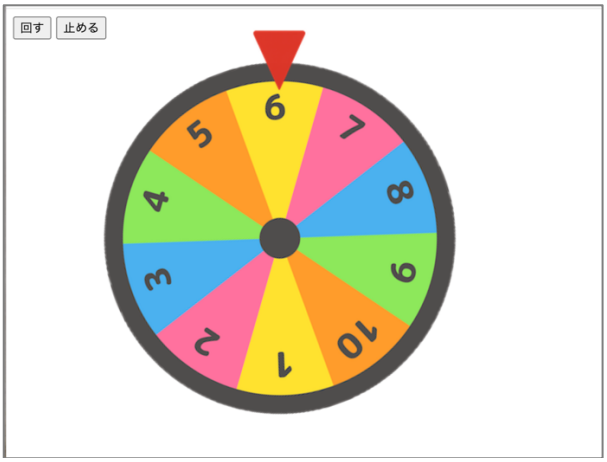
第６章　ルーレット

６-１　【演習】ルーレットをつくってみよう

下図のようなルーレットを作ってください。

サンプル　　https://js.drmpls.com/roulette/roulette.html

第４章の「サイコロ」の知識でできます。



【作成手順・ヒント】
■手順１　見た目をつくりましょう。

画像は下記からダウンロード、または講師かから入手してください。

https://drmpls.com/javascript_super_intro/

ダウンロードしたイラストは roulette フォルダをつくって入れてください。

roulette.html を新規につくり、roulette フォルダに保存しましょう。
画像やボタンを表示してください。

 roulette.png	position: absolute; 上から 50px, 左から 100px に配置。 画像の元ネタ「イラスト AC」 https://www.ac-illust.com/main/detail.php?id=24994562&word=カラフルなルーレット#goog_rewarded から入手したものをサイズ調整しました。
 probe.png	position: absolute; 左から 246px に配置。

［回す］［止める］ボタンは style 設定不要です。
body には position: relative;の style 設定を忘れずに。
　各要素の id 名は独自に決めてください。

■手順２　プログラムをつくりましょう
- ［回す］［止める］ボタン、ルーレットの３つを JavaScript で扱えるように定数に割り当てましょう。定数名は独自に決めてください。
- ［回す］ボタンがクリックされたときのイベントリスナーをつくり、ルーレットの角度を変える関数を適当なインターバルで呼び出してください。サイコロを回転させた「4-6」あたりを参考にしましょう。
- ［止める］ボタンクリックされたときのイベントリスナーをつくり、回転を止めましょう。

完成例　6-1【演習】

style 部

```
<style>
    body{
        position: relative;
    }
    #roulette{
        position: absolute;
        top: 50px;
        left: 100px;
    }
    #probe{
        position: absolute;
        left: 246px;
    }
</style>
```

html および script 部

```
<body>
    <img id="roulette" src="roulette.png">
    <img id="probe" src="probe.png">
    <button id="start_btn">回す</button>
    <button id="stop_btn">止める</button>
    <script>
        const roulette = document.querySelector("#roulette");
        const start_btn = document.querySelector("#start_btn");
        const stop_btn = document.querySelector("#stop_btn");
        let timeId;
        let d = 0;

        start_btn.addEventListener('click', e=>{
            timeId = setInterval('move()', 10);
            start_btn.disabled = true;
        }, false);

        stop_btn.addEventListener('click', e=>{
            clearInterval(timeId);
            start_btn.disabled = false;
        }, false);

        function move(){
            roulette.style.transform = "rotate(" + d + "deg)";
            d += 20;
        }

    </script>
</body>
```

6-2　ゆっくりと止まるようにしてみよう

ピタッと止まるのではなく、だんだん回転速度が遅くなって止まるようにしてみます。
［止める］ボタンがクリックされたら徐々に減速させるので、回転の増分を少しずつ小さくしていく必要があります。

現在、増分は d += 20 として 20 ずつ増えています。（20 が増分です。）
増分を変数にして、少しずつ小さくするしくみを作っていきましょう。

① 増分の変数を dx として定義してください。後々使用する timeIdSlow も定義しておいてください。

```
let timeIdSlow;
let d = 0;
let dx;
```

追加
追加

② ［回す］ボタンがクリックされたタイミングで dx を 20 にしましょう。もともとの増分です。

```
start_btn.addEventListener('click', e=>{
    dx = 20
    timeId = setInterval('move()', 50);
    start_btn.disabled = true;
}, false);
```

追加

インターバルは 10 だと速過ぎの感じもあるので 50 にしています。

③ move()関数内で 20 直打ちで増やしていた部分を dx を増やすように書き換えてください。

```
function move(){
    roulette.style.transform = "rotate(" + d + "deg)";
    d += dx;
}
```

dx に変更

④ ［止める］ボタンがクリックされたときのイベントリスナーを次のように変更してください。

```
stop_btn.addEventListener('click', e=>{
    clearInterval(timeId);
    timeIdSlow = setInterval('slowdown()', 50);
}, false);
```

追加

もともとあった clearInterval は削除してください。
slowdown()内で使うようにします。

slowdown()関数はこれから作ります。

⑤ slowdown()関数を作っていきます。
まずは次のようにしてください。
move()関数とほとんど同じですが、増分をプラスする直前に増分に 0.96 を掛けて少し減らします。

```
function slowdown(){
    roulette.style.transform = "rotate(" + d + "deg)";
    dx *= 0.96;
    d += dx;
}
```

増分を減らす処理

動作確認してください。
回転させてから［止める］ボタンをクリックすると、ゆっくりと減速していって動かなくなります。

ただし、見た目が止まっているだけで、setInterval で slowdown 関数は呼び出し続けられていますから実は動き続けています。

確認のため、ログ出力を追加して動かしてみましょう。

```
function slowdown(){
    roulette.style.transform = "rotate(" + d + "deg)";
    dx *= 0.96;
    d += dx;
    console.log(d);
}
```

追加

コンソールを見ると、見た目が止まったあとも d の値がどんどん表示されて動き続けていることがわかります。

⑥ 増分がある程度小さくなったら clearInterval で止めるようにしましょう。

```
function slowdown(){
    roulette.style.transform = "rotate(" + d + "deg)";
    dx *= 0.96;
    d += dx;
    console.log(d);
    if(dx < 0.05){
        clearInterval(timeIdSlow);
    }
}
```

追加。dx が 0.05 を下回ったら繰り返し呼び出しているものを止める。

動作確認してください。見た目が止まったあと、コンソールに出力される d の値も止まることを確認しましょう。

⑦ 止まった後、［回す］ボタンを有効（活性）にしましょう。

```
function slowdown(){
    roulette.style.transform = "rotate(" + d + "deg)";
    dx *= 0.96;
    d += dx;
    console.log(d);
    if(dx < 0.05){
        clearInterval(timeIdSlow);
        start_btn.disabled = false;
    }
}
```

追加

これで、止まったあとにまた回転させることができるようになります。

6-3　バグ修正

6-3-1　減速中

減速中に［止める］ボタンをさらに押すと、どんなことが起きるでしょう？
実際に何度も押してみても、特になにも問題ないように感じます。

ところが、そのあとに［回す］ボタンをクリックしてみてください。
回り続けないで、ブレーキがかかって自動的に止まってしまいます。
しかも、見た目は止まっているのに、コンソールで見ると d の値は変化し続けていることが見えます。

この現象を解消するために、［止める］ボタンを押して減速している最中は［止める］ボタンも無効化（非活性）にしましょう。
そして、［回す］ボタンがクリックされたときに有効（活性）にしましょう。

自力で挑戦してみてください。
解答例は次のページです。

解答例

［止める］ボタンを無効化（非活性）

```
function slowdown(){
    stop_btn.disabled = true;
    roulette.style.transform = "rotate(" + d + "deg)";
    dx *= 0.96;
```

追加

［止める］ボタンを有効化（活性）

```
start_btn.addEventListener('click', e=>{
    dx = 20
    timeId = setInterval('move()', 50);
    start_btn.disabled = true;
    stop_btn.disabled = false;
}, false);
```

追加

6-3-2　回す前

回転させる前に［止める］ボタンをクリックしたときも、そのあと［回す］ボタンをおしても回らないというおかしな挙動になります。

ですので、初期状態で［止める］ボタンを無効（非活性）にしておきましょう。

解答例は次のページです。

解答例

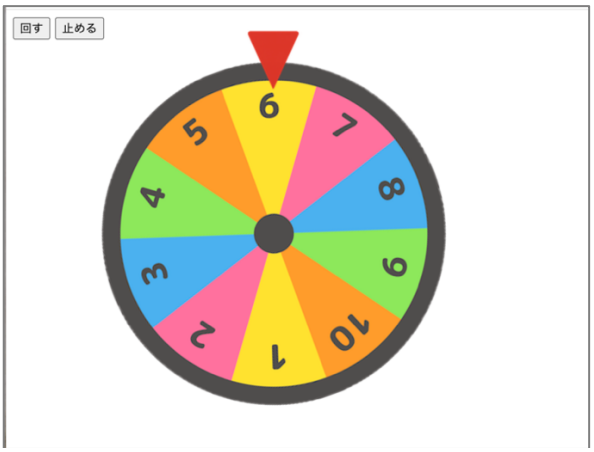
```
<button id="start_btn">回す</button>
<button id="stop_btn" disabled>止める</button>
<script>
```

追加

6-4　出た値を利用したい場合

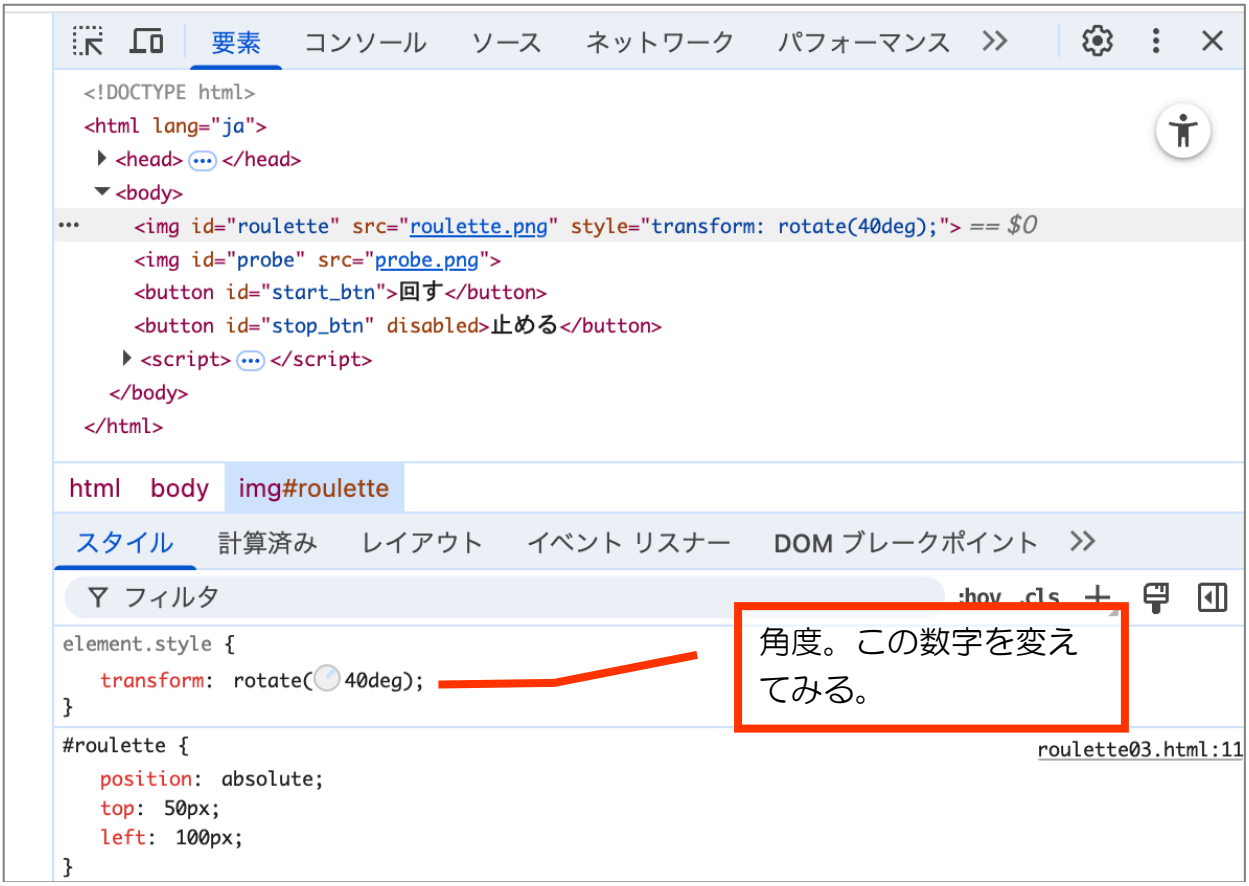
6-4-1　角度と値の関係

　実際にゲームなどでこのようなルーレットを使う場合、ルーレットが止まった時の値を人間であれば目で見て認識できますが、コンピューターではどうしたら良いでしょうか？
　例えば下図ですと６を示していますが、プログラムではそれをどのように判断すればよいでしょうか。　１分ほど考えてみてください。

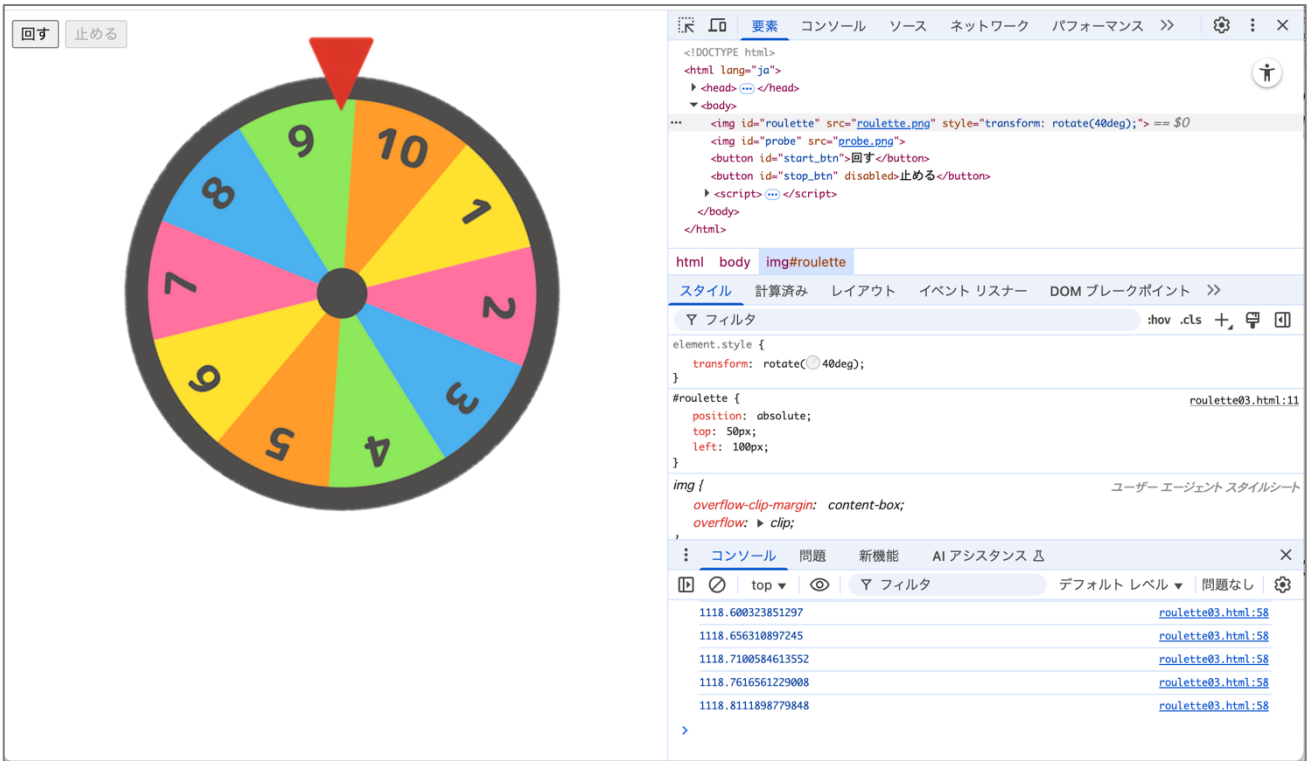


　答えのひとつとして・・・回転している角度での判断が良いでしょう。
　理解を深めるため、回転させてブラウザにて検証ツールを見てみましょう。

　次ページの図で示すように、id="roulette"の img タグを選択すると、スタイル欄に transform: rotate(xxx deg);を見ることができます。



例えば角度を 40deg にすると、下図のようになります。ルーレットでは９ですね。



角度をいろいろと変えてみると角度とルーレットの値の関係を実感できることと思います。

360 度で１回転です。ルーレットの値は１～１０なので、１０等分して一つの値は 36 度。
右に回転していくので、0～36 度の範囲だとルーレットの値は１０、36～72 度だと9、72～108 度だと8、・・・となりますね。

プログラムで条件判定していくにあたり、0～36 度と 36～72 度だと境目の 36 度がどちらの範囲になるか困ってしまいますので、0以上 36 度未満、36 度以上 72 度未満、・・・というルールにしてみましょう。

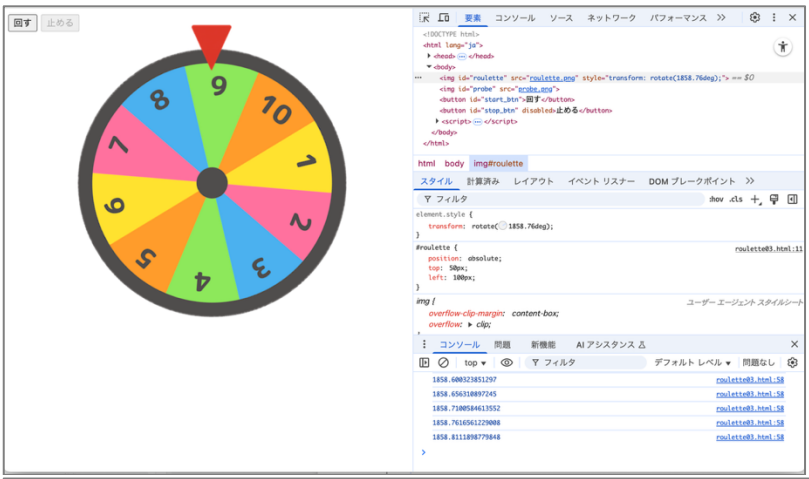
対応表を作ると以下のようになります。

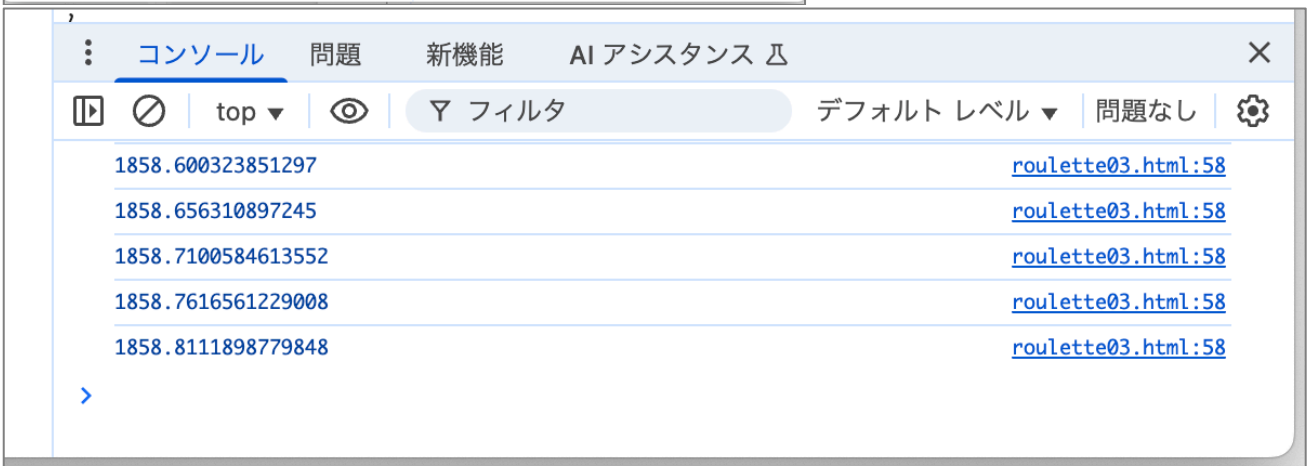
角度	ルーレットの値
0 以上 36 未満	10
36 以上 72 未満	9
72 以上 108 未満	8
108 以上 144 未満	7
144 以上 180 未満	6
180 以上 216 未満	5
216 以上 252 未満	4
252 以上 288 未満	3
288 以上 324 未満	2
324 以上	1

6-4-2　止まった時の角度

角度はプログラム内では d です。
［とまる］ボタンを押すとコンソールに d の値が表示されます。（function slowdown 内で console.log(d)としているため）

実際に止まった時のコンソールの例を下図に示します。





1858 とかになっています。
全然０～360 度の範囲ではありません。
なぜかというと、d はずっと増え続けるからです。360 を超えても増え続けます。
例えば 370 度になったとすると、これは 10 度と同じことです。
360 度以下の角度に変換（換算）するには、360 で割った余りを利用します。

そこで、d を 360 で割った余りを入れておく変数を用意し、止まった時の角度として評価することにします。

6-4-3　ルーレットの値を表示してみる

① 確認のためだけですので、style の設定などは不要で div で表示領域だけつくってください。id=”kekka”としましょう。
　また、JavaScript からこの表示領域を扱えるように定数 const kekka = ……もつくりましょう。

```
<button id="start_btn">回す</button>
<button id="stop_btn" disabled>止める</button>
<div id="kekka"></div>
<script>
    const roulette = document.querySelector("#roulette");
    const start_btn = document.querySelector("#start_btn");
    const stop_btn = document.querySelector("#stop_btn");
    const kekka = document.querySelector("#kekka");
    let timeId;
    let timeIdSlow;
```

追加

追加

② d を 360 で割った余りを入れるための変数も用意しましょう。stop_deg という変数名にします。

```
let d = 0;
let dx;
let stop_deg;
```

追加

③ ルーレットの回転が止まったら id="kekka"の表示領域に stop_deg の値を表示します。
　次のように stop_deg に d を 360 で割った余りを入れます。
　そして switch 文を使って条件分岐しましょう。まずは 10 と 9 の部分のみ示します。

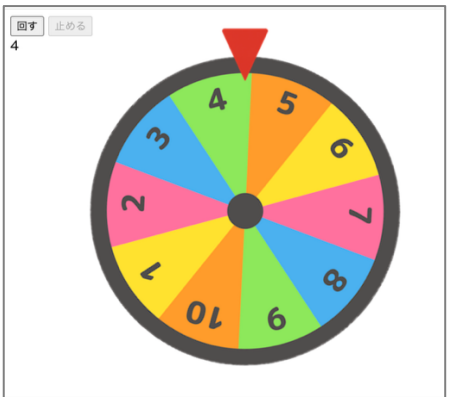
```
if(dx < 0.05){
    clearInterval(timeIdSlow);
    start_btn.disabled = false;

    stop_deg = d % 360;
    switch(true){
        case stop_deg < 36:
            kekka.textContent = 10;
            break;
        case 36 <= stop_deg && stop_deg < 72:
            kekka.textContent = 9;
            break;
    }
}
```

追加

【演習】
8、7、6、・・・・、2、1も表示できるようにコードを追加してください。
6-4-1の最後に条件を表にまとめましたので参考にしてください。

解答例は次のページです。

　ルーレットの値が表示領域に正しく表示されることを確認してください。



④ 【演習】一度結果を表示したあと［回す］ボタンを押してもその値が表示されっぱなしです。［回す］ボタンが押されたら一度消えるようにしてください。

【演習】③解答例

```
switch(true){
    case stop_deg < 36:
        kekka.textContent = 10;
        break;
    case 36 <= stop_deg && stop_deg < 72:
        kekka.textContent = 9;
        break;
    case 72 <= stop_deg && stop_deg < 108:
        kekka.textContent = 8;
        break;
    case 108 <= stop_deg && stop_deg < 144:
        kekka.textContent = 7;
        break;
    case 144 <= stop_deg && stop_deg < 180:
        kekka.textContent = 6;
        break;
    case 180 <= stop_deg && stop_deg < 216:
        kekka.textContent = 5;
        break;
    case 216 <= stop_deg && stop_deg < 252:
        kekka.textContent = 4;
        break;
    case 252 <= stop_deg && stop_deg < 288:
        kekka.textContent = 3;
        break;
    case 288 <= stop_deg && stop_deg < 324:
        kekka.textContent = 2;
        break;
     case 324 <= stop_deg:
        kekka.textContent = 1;
        break;
}
```

追加

【演習】④解答例

```
start_btn.addEventListener('click', e=>{
    dx = 20
    timeId = setInterval('move()', 50);
    start_btn.disabled = true;
    stop_btn.disabled = false;
    kekka.textContent = '';
}, false);
```

追加

第７章　ライオンを避けてネズミをゲットするゲーム

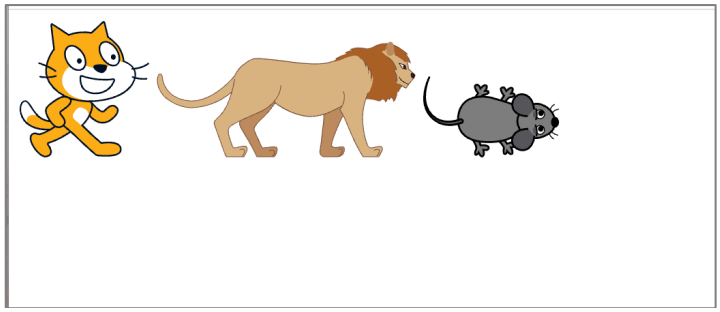
ライオンが左右に行ったり来たりしているところをライオンに当たらないようにすりぬけてネズミのところまで行けたらクリア、という単純なゲームを作ってみましょう。（パソコン教室ドリームプラスでの Scratch の体験メニューとしてお馴染みのネタです。）

サンプル　　https://js.drmpls.com/lion/lion.html

ライオンフォルダをつくり、lion.html をそこに新規作成・保存してください。使用する画像素材画像は下記からダウンロード、または講師かから入手してライオンフォルダに入れてください。
https://drmpls.com/javascript_super_intro/

7-1　表示する

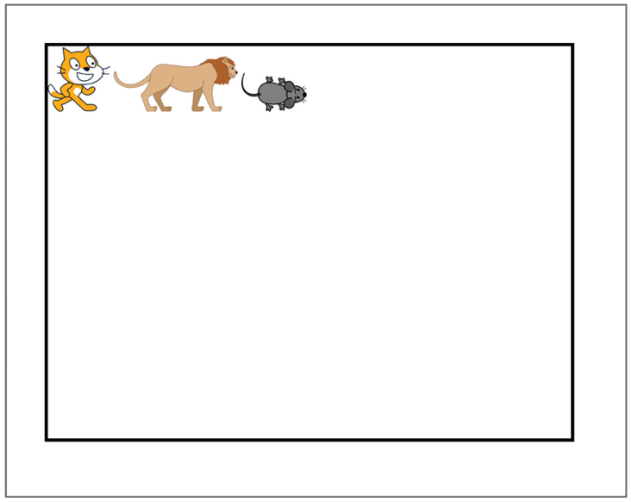
① まずは使用する画像を表示するだけ表示してみましょう。楽勝ですね。



それぞれ id は次のようにしておきましょう。
ねこ：　cat
ライオン：　lion1
ねずみ：　mouse

② 四角い枠を作りましょう。div タグで、上からも左からも 50px の位置に配置します。大きさは幅 800px、高さ 600px にしましょう。

その中に３つの画像を入れてください。下図のようになります。
３つの画像にはまだ style は設定していません。



ここまでの完成例は次のページです。

完成例

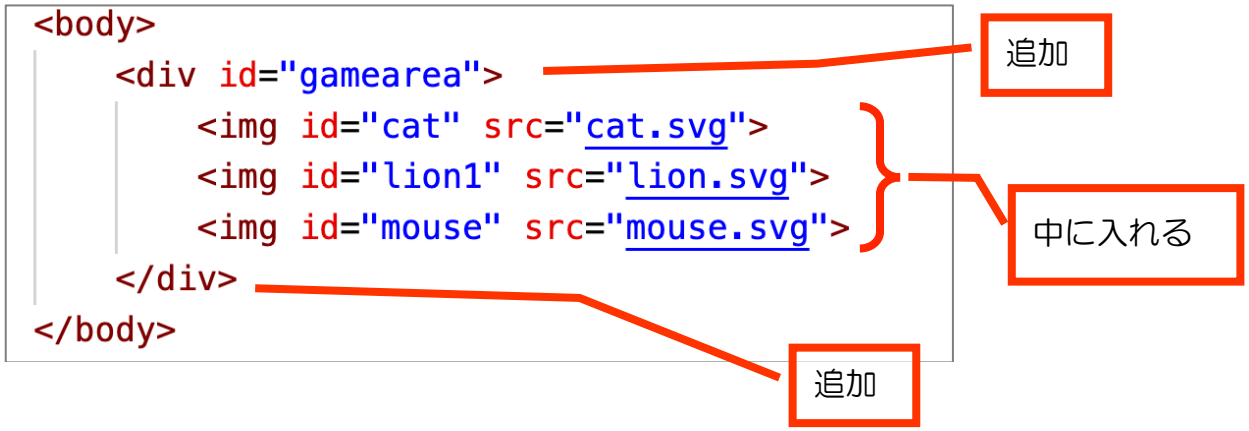
①

```
<body>
    <img id="cat" src="cat.svg">
    <img id="lion1" src="lion.svg">
    <img id="mouse" src="mouse.svg">
</body>
```

②

html 部

```
<body>
    <div id="gamearea">
        <img id="cat" src="cat.svg">
        <img id="lion1" src="lion.svg">
        <img id="mouse" src="mouse.svg">
    </div>
</body>
```



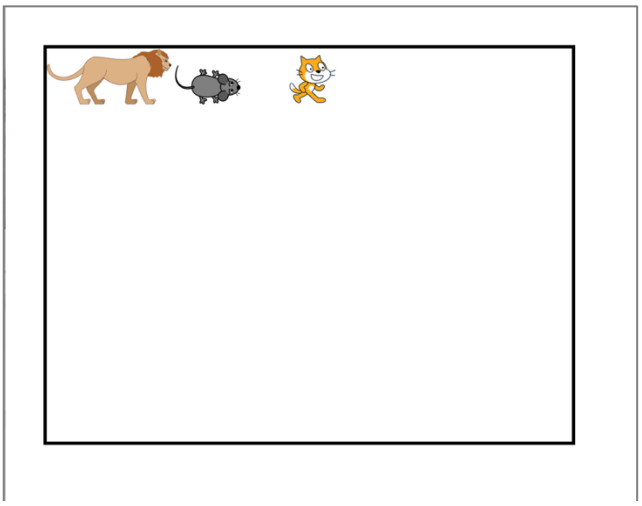
style 部

```
<style>
    body{
        position: relative;
    }
    #gamearea{
        position: absolute;
        top: 50px;
        left: 50px;
        width: 800px;
        height: 600px;
        border: solid 5px ■black;
    }
```

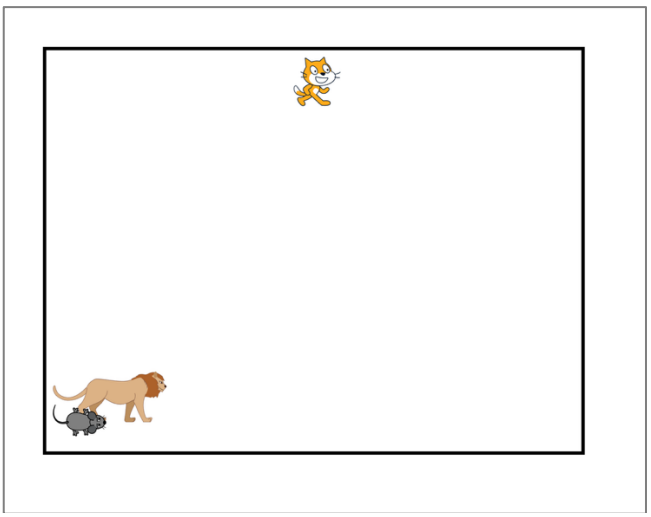
③ ネコの位置を次のように設定してください。

```
#cat{
    position: absolute;
    top: 10px;
    left: 370px;
    width: 70px;
}
```

下図のように、枠内で上から 10px、左から 370px の位置になります。
width を 70px にしているのは、大きさを横幅 70px にすることで少し小さめにするためです。



【演習】ライオンとねずみを下図のように配置してください。数値はいろいろと試して、下図と似たような見た目になれば OK です。完成例は次のページに示しますが、完成例通りの数値じゃなくても OK です。



完成例

```
#lion1{
    position: absolute;
    top: 480px;
    left: 10px;
    width: 170px;
}
#mouse{
    position: absolute;
    top: 530px;
    left: 10px;
    width: 80px;
}
```

④　［開始］［停止］ボタンも枠の外に表示してください。

枠の外ですので、gamearea の外側に追加です。

```
<body>
    <button id="start_btn">開始</button>
    <button id="stop_btn">停止</button>
    <div id="gamearea">
        <img id="cat" src="cat.svg">
        <img id="lion1" src="lion.svg">
        <img id="mouse" src="mouse.svg">
    </div>
```

追加

7-2　ライオンを行ったり来たりさせる

第３章で跳ね返る仕組みをつくりました。復習を兼ねて作ってみましょう。

① ［開始］ボタンがクリックされたらライオンを左に動き続けるようにしましょう。
（ヒント）
［開始］ボタンクリックを検知するイベントリスナーにてライオンを左に動かし続ける処理を行います。setInterval を使います。

また、［停止］ボタンがクリックされたらライオンの動きが止まるようにしましょう。
（ヒント）
［停止］ボタンクリックを検知するイベントリスナーにて、clearInterval です。

前章までと同様、［開始］ボタンがクリックされたら［開始］ボタンは無効に、［停止］ボタンがクリックされたら［開始］ボタンを有効にするしくみも入れておきましょう。

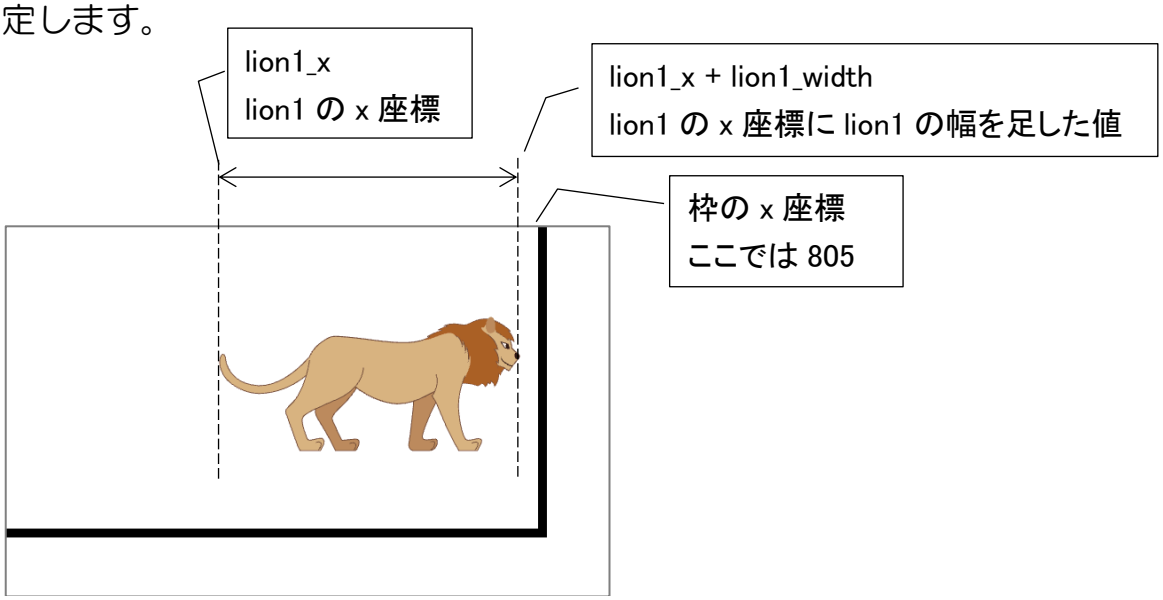
①を完成させてから、次の②に取り組んでください。

② 枠の右端で逆方向に動くようにしましょう。

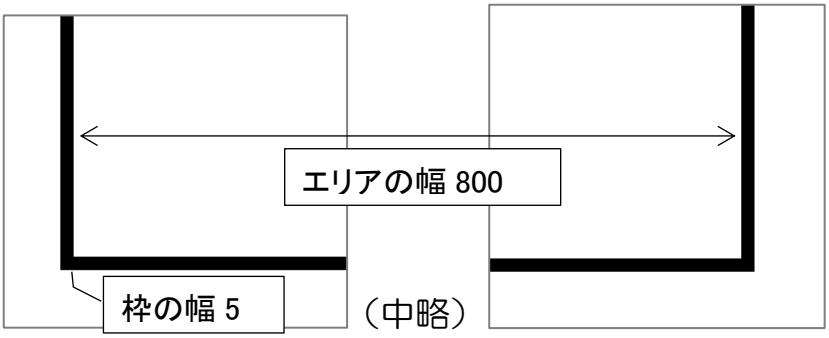
```
let lion1_dx = 10;
let lion1_width = 170;
```

lion1 の幅を定義しておきます。
style で指定した幅と同じ値です。

右端は lion1 の x 座標に lion1 の幅を足した値が枠の右端を超えたかどうかで判定します。



枠の右端の x 座標は枠で囲まれたエリアの横幅＋枠の幅になりますので 805 です。



実際に 805 で判定するとライオンが少し枠をハミ出ているようにも見えます（試してみてください。）ので、純粋にエリアの幅である 800 で判定することにします。
つまり、「lion1_x + lion1_width が 800 を超えたら」という条件を判定します。

```
function move(){
    lion1_x += lion1_dx;
    lion1.style.left = lion1_x + "px";

    if(800 <= lion1_x + lion1_width) lion1_dx = -lion1_dx;
};
```

追加

増分を反転させることで逆方向に動くようにします。

【演習】
枠の左端でも逆方向に動くようにしてください。
枠の左端の x 座標は 0 です。
つまり、「lion1_x <= 0」という条件です。

```
if(800 <= lion1_x + lion1_width) lion1_dx = -lion1_dx;
```

上記の条件に、「||」（または）で追加してください。

```
if(追加する条件 || 800 <= lion1_x + lion1_width) lion1_dx = -
lion1_dx;
```

ライオンの向きはずっと右向きのままですが、左右の端に着いたら逆方向に動くようになることを確認してください。

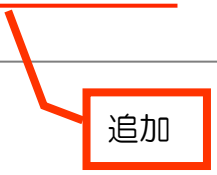
解答例は次のページです。

完成例①

```
<script>
    const start_btn = document.querySelector("#start_btn");
    const stop_btn = document.querySelector("#stop_btn");
    const lion1 = document.querySelector("#lion1");

    let timeId;
    let lion1_x = 10;
    let lion1_dx = 10;
    let lion1_width = 170;
    let muki = 1;

    start_btn.addEventListener('click', e=>{
        start_btn.disabled = true;
        timeId = setInterval('move()', 100);
    }, false);

    function move(){
        lion1_x += lion1_dx;
        lion1.style.left = lion1_x + "px";
    };

    stop_btn.addEventListener('click', e=>{
        clearInterval(timeId);
        start_btn.disabled = false;
    }, false);

</script>
```

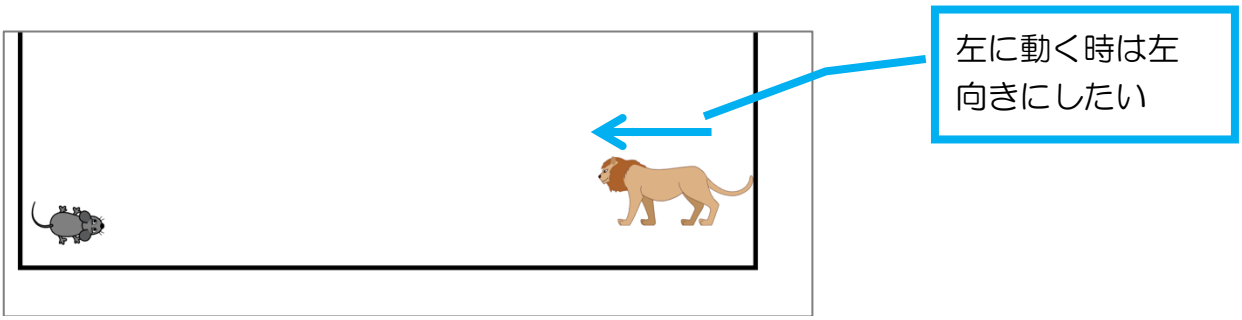
完成例②　【演習】

```
    if(lion1_x <= 0 || 800 <= lion1_x + lion1_width) lion1_dx = -lion1_dx;
};
```



7-3　ライオンの向きを変える

左に動く時は左向きにしたいですね。



枠に触れた時の処理に向きを反転させる処理を追加していきます。

① 処理が増えるので、処理の部分を{ }でくくりましょう。

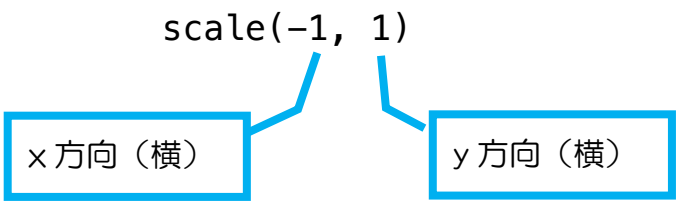
```
if(lion1_x <= 0 || 800 <= lion1_x + lion1_width){
    lion1_dx = -lion1_dx;
}
```

② 画像の左右を反転させる処理は、次のように style.transform にて scale を設定します。

```
if(lion1_x <= 0 || 800 <= lion1_x + lion1_width){
    lion1_dx = -lion1_dx;
    lion1.style.transform = "scale(-1, 1)";
}
```

追加

scale は大きさ（拡大・縮小）に使う属性ですが、-（マイナス）をつけるとマイナス方向へ作用させることになります。「１」は等倍です。



動作確認して、右の枠に触れたらライオンが左を向いて左に進むことを確認してください。

③ 左の枠に触れたとき、左向きのまま右方向へ動き出します。
scale(-1, 1)は左向きの設定ですので、当然ですね。

そこで、左の枠にふれたときには scale(1, 1)にする必要があります。
muki という変数を定義し枠に触れるたびに 1 と-1 を切り替わるようにすればよさそうです。
まずは変数を定義し、１で初期化しましょう。

```
let lion1_dx = 10;
let lion1_width = 170;
let muki = 1;
```

追加

④ 向きを反転させる処理として、次のように追加・変更してください。

```
if(lion1_x <= 5 || 805 <= lion1_x + lion1_width){
    lion1_dx = -lion1_dx;
    muki = -muki;
    lion1.style.transform = "scale(" + muki + ", 1)";
}
```

向きの反転を追加

-1 だった部分を、変数 muki を使うように変更。

動作させて、枠に触れた時にライオンの向きが進行方向に向くことを確認してください。

7-4　【演習】ねずみを動かす

ねずみについても、ライオンと同様に左右に動き続けるようにしてください。枠に触れたら向きを変えてください。

■ヒントと手順

① ライオンは下図に示すように定数と変数を定義しました。
ネズミも同様の一式を作りましょう。変数名・定数名は任意です。

```
const lion1 = document.querySelector("#lion1");

let timeId;
let lion1_x = 10;
let lion1_dx = 10;
let lion1_width = 170;
let muki = 1;
```

timeId は不要です。
ネズミを定数で扱えるようにすること、ネズミの位置・増分・幅・向きを定義しましょう。

② ライオンを動かし続けている関数 move() 内に、ネズミを動かし続けるためのコードを追加してください。

解答例は次のページです。

解答例

```
<script>
    const start_btn = document.querySelector("#start_btn");
    const stop_btn = document.querySelector("#stop_btn");
    const lion1 = document.querySelector("#lion1");
    const mouse = document.querySelector("#mouse");

    let timeId;
    let lion1_x = 10;
    let lion1_dx = 10;
    let lion1_width = 170;
    let muki = 1;

    let mouse_x = 10;
    let mouse_dx = 10;
    let mouse_width = 80;
    let mukiM = 1;
```

① 定数追加

① 変数追加

```
  function move(){
      lion1_x += lion1_dx;
      lion1.style.left = lion1_x + "px";

      if(lion1_x <= 0 || 800 <= lion1_x + lion1_width){
          lion1_dx = -lion1_dx;
          muki = -muki;
          lion1.style.transform = "scale(" + muki + ", 1)";
      }

      mouse_x += mouse_dx;
      mouse.style.left = mouse_x + "px";

      if(mouse_x <= 0 || 800 <= mouse_x + mouse_width){
          mouse_dx = -mouse_dx;
          mukiM = -mukiM;
          mouse.style.transform = "scale(" + mukiM + ", 1)";
      }

  };
```

② 追加

7-5　ネコを上下左右キーで動かす

3-7を参考に、猫を上下左右キーで動かす仕組みをつくりましょう。

① 猫を操作できるように定数を定義しましょう。

```
const mouse = document.querySelector("#mouse");
const cat = document.querySelector("#cat");
```

追加

② 押されたキーのキーコードを入れておく変数を定義しましょう。

```
let mouse_width = 80;
let mukiM = 1;

let keydown = '';
```

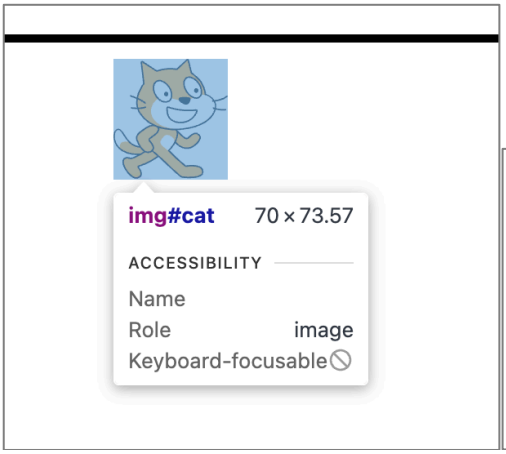
追加

③ キーが押されたときのキーコードを変数に代入する処理、キーが離された時の処理を追加しましょう。

```
    window.addEventListener('keydown', e=>{
        keydown = e.key;
    }, false);
    window.addEventListener('keyup', e=>{
        keydown = '';
    }, false);

</script>
```

追加

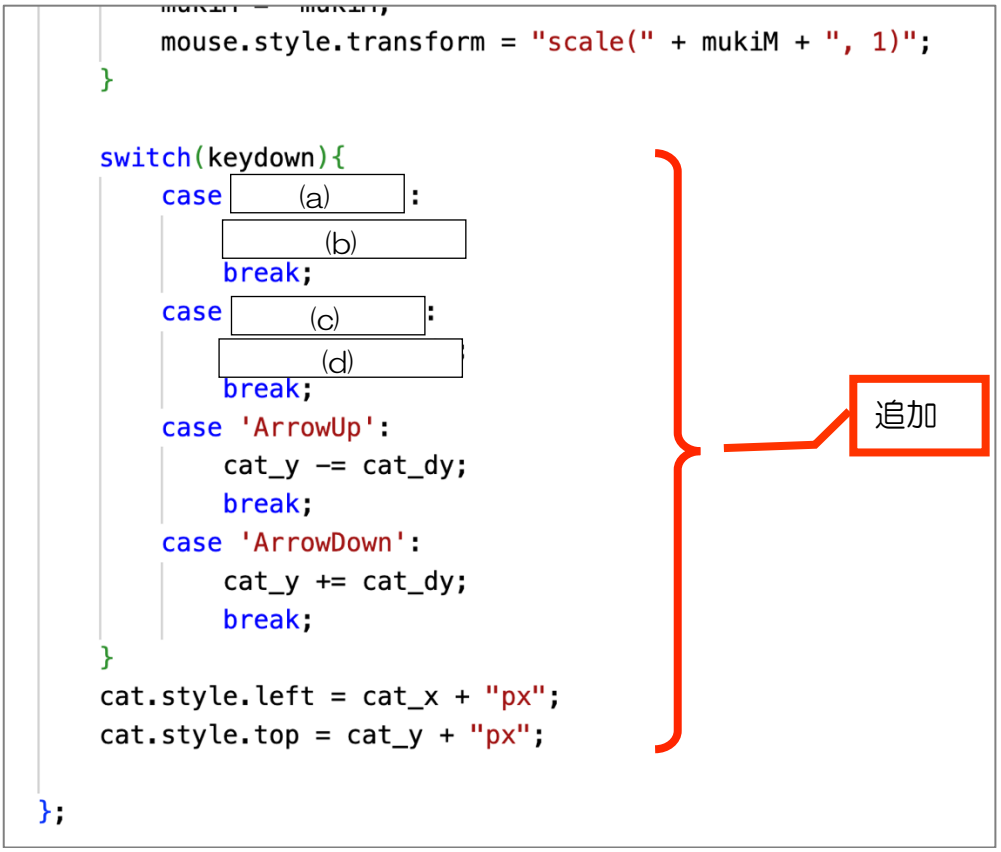
④ 猫の座標・変化量・幅・高さを定義しましょう。
　なお、高さはわからなかったので、検証ツールでみてみたところ、73.57でしたのでそのように定義します。




```
let keydown = '';
let cat_x = 370;
let cat_dx = 10;
let cat_width = 70;
let cat_y = 10;
let cat_dy = 10;
let cat_height = 73.57;
```

追加

⑤ キーが押された時の猫の動きをつくりましょう。
move()関数内に下図のように追加してください。
復習を兼ねて、穴埋めにしていますので、その部分は考えて埋めてみましょう。



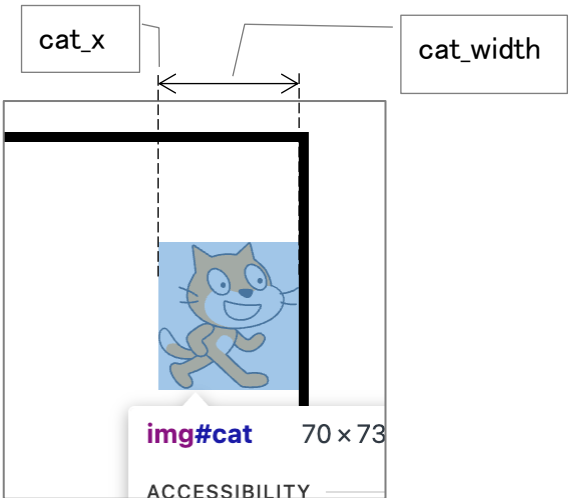
完成例は次のページです。

完成例⑦

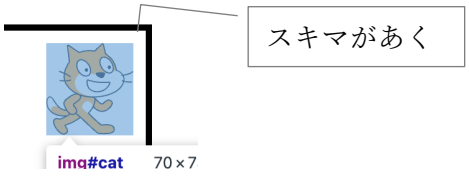
```
        mouse.style.transform = "scale(" + mukiM + ", 1)";
    }

    switch(keydown){
        case 'ArrowLeft':
            cat_x -= cat_dx;
            break;
        case 'ArrowRight':
            cat_x += cat_dx;
            break;
        case 'ArrowUp':
            cat_y -= cat_dy;
            break;
        case 'ArrowDown':
            cat_y += cat_dy;
            break;
    }
    cat.style.left = cat_x + "px";
    cat.style.top = cat_y + "px";

};
```

7-6　猫が枠からハミ出ないようにする

＜考え方＞
　右端の枠に当たった場合の事例で考えてみましょう。
　ライオンやネズミと同様、右端は x = 805 です。
　ですので、右に移動中の場合に cat_x + cat_width >= 805 の条件になったら移動量を打ち消します。



※　ライオンが右端に当たるときの判定は 800 にしましたが、猫を 800 で判定すると枠の手前で止まって隙間が開いてしまうため、805 としました。



① 次のコードを追加してください。

```
switch(keydown){
    case 'ArrowLeft':
        cat_x -= cat_dx;
        break;
    case 'ArrowRight':
        cat_x += cat_dx;
        if(cat_x + cat_width >= 800) cat_x -= cat_dx;
        break;
    case 'ArrowUp':
        cat_y -= cat_dy;
```

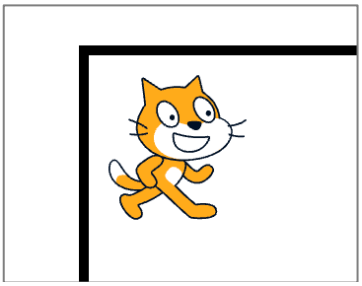
追加

もともとは右へ移動させるために cat_dx を追加していますが、

打ち消すために cat_dx を減算します。

プラスマイナス0なので、右矢印キーを押し続けても動きません。

② 左へ移動中に左端の枠に当たった場合にもハミ出さないようにしましょう。
枠の太さを考慮すると猫が枠に当たる場合の x 座標は 5 になりそうです。



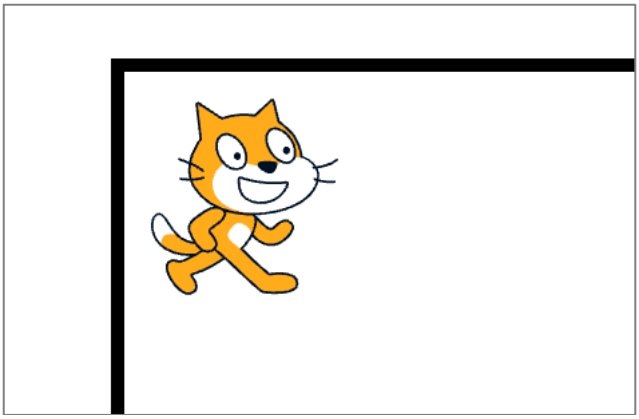
次のように追加してください。

```
    switch(keydown){
        case 'ArrowLeft':
            cat_x -= cat_dx;
            if(cat_x <= 5) cat_x += cat_dx;
            break;
        case 'ArrowRight':
            cat_x += cat_dx;
            if(cat_x + cat_width >= 805) cat_x -= cat_dx;
            break;
```

追加

動作確認してみましょう。
左端まで行って止まるでしょうか？

なんと、驚いたことに隙間が空いたまま、これ以上左へ行けません。



検証ツールで見ると、left: 10px; になっています。

left が 10px で、それ以上左へ行かない。

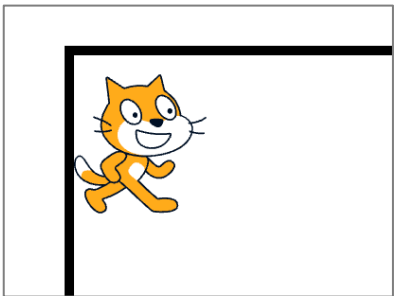
```
<button id="stop_btn">停止</button>
<div id="gamearea">
    <img id="cat" src="cat.svg" style="left: 10px; top: 10px;"> == $0
    <img id="lion1" src="lion.svg" style="left: 230px; transform: scale(1, 1);">
    <img id="mouse" src="mouse.svg" style="left: 670px; transform: scale(-1, 1);">
```

不思議です。
ということで、判定を 0 未満に修正してみましょう。

```
case 'ArrowLeft':
    cat_x -= cat_dx;
    if(cat_x < 0) cat_x += cat_dx;
    break;
```

変更

左へ進み続けると、枠にピタリとくっついたところでそれ以上左へ行かなくなりました。

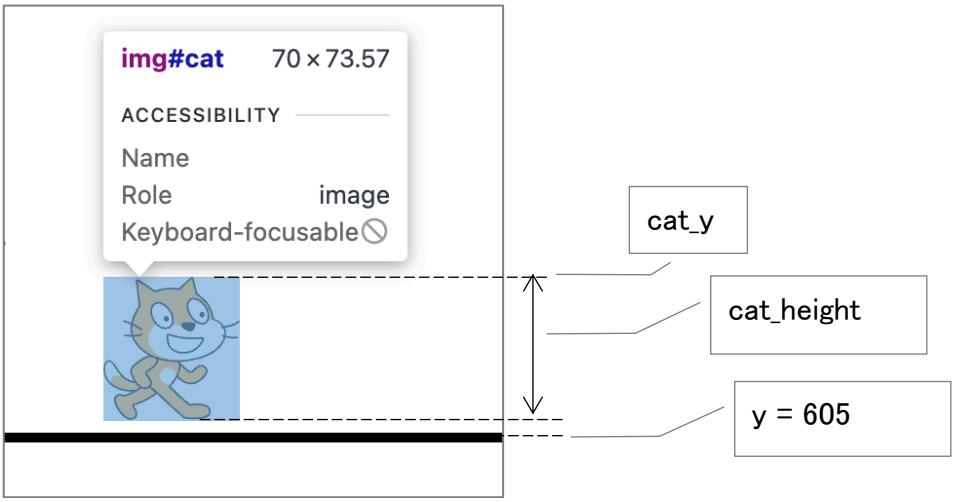


検証ツールで見ても left が 0px になっていることが確認できます。

```
<div id="gamearea">
    <img id="cat" src="cat.svg" style="left: 0px; top: 10px;"> == $0
    <img id="lion1" src="lion.svg" style="left: 580px; transform: scale(
```

こういう状況を見ると、ライオンやネズミが左端で跳ね返る部分を 5 で判定しましたが、0 未満でも良いかもしれません。
各自、課題として取り組んでみてください。

③ 下端より下に行かないようにする考え方も、右端のときと同じです。
ゲームエリアの高さは 600 です。枠の幅 5 も考慮して、下に移動中の場合に cat_y + cat_height >= 605 の条件になったら移動量を打ち消します。



下記のように追加してください。

```
    case 'ArrowDown':
        cat_y += cat_dy;
        if(cat_y + cat_height >= 605) cat_y -= cat_dy;
        break;
}
```

追加

下の枠線からハミ出て動いていかないことを動作確認してください。

④ 【演習】上の枠線も同様に、上端をハミ出ないようにしてください。

解答例　④【演習】

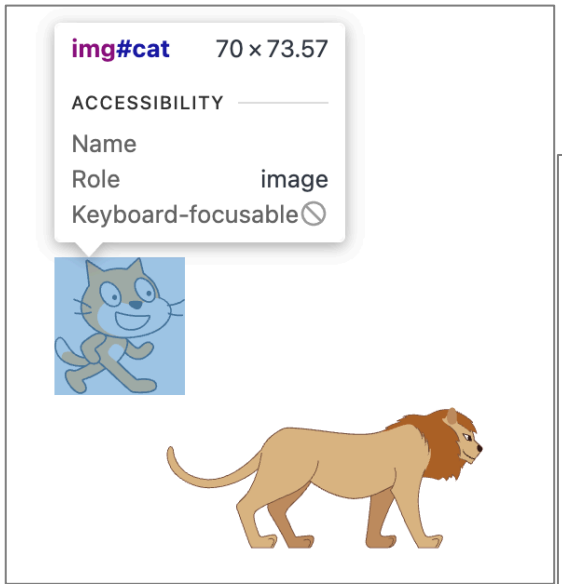
```
    case 'ArrowUp':
        cat_y -= cat_dy;
        if(cat_y < 0) cat_y += cat_dy;
        break;
    case 'ArrowDown':
```

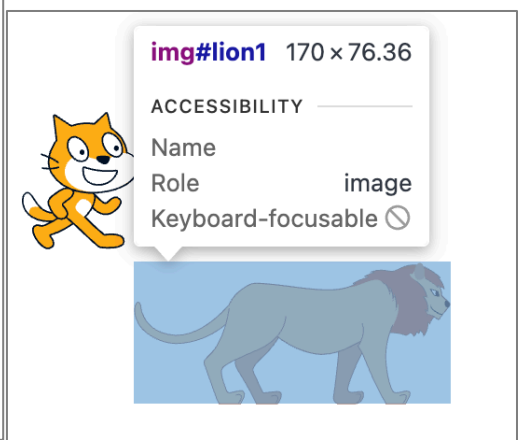
追加

7-6　猫とライオンの当たり判定

猫もライオンも形が複雑です。形に沿っての当たり判定をつくるのは大変困難ですので、猫もライオンも四角形で判定することにします。

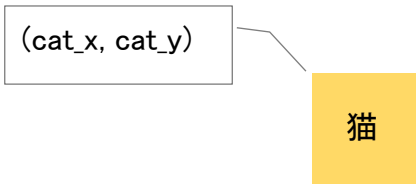
実際には触れていなくても当たりの判定になってしまうこともあり、厳しい判定になりますが…



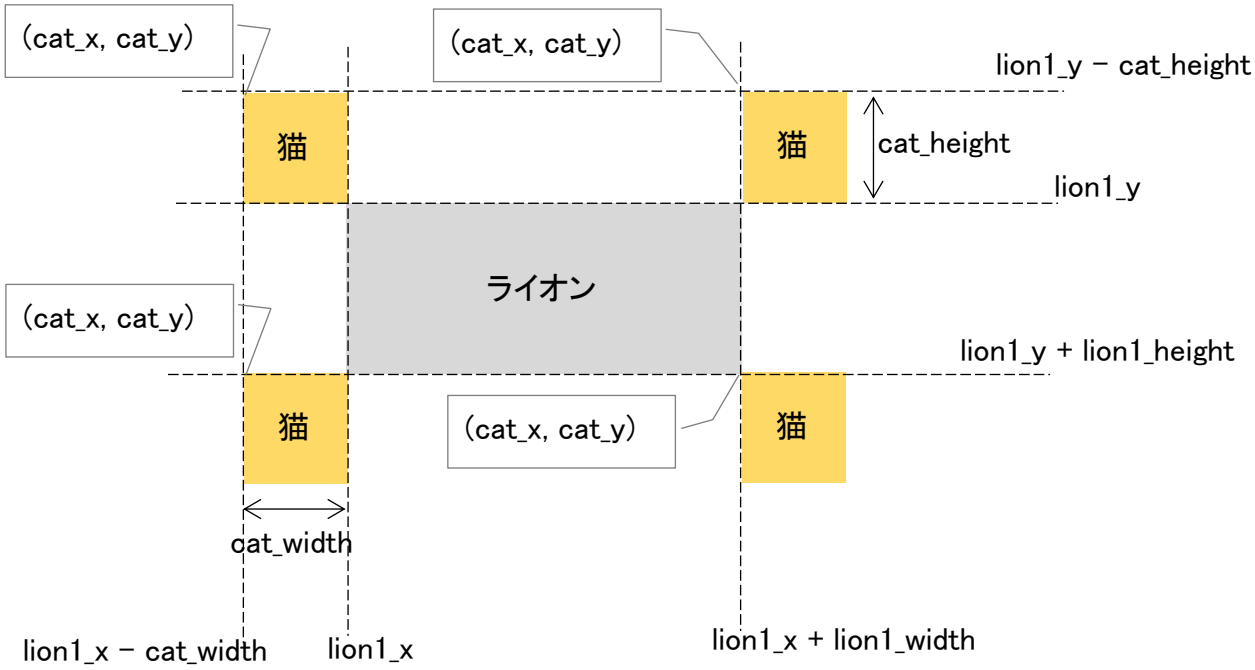


考え方としては下図のようになります。

猫の座標は左上の角が基準になり、次のようになります。



ライオンの上端・下端・左端・右端と猫の座標との関係を下図のように整理して考えて当たり判定をつくります。



横方向については、
lion1_x - cat_width <= cat_x でライオンの左側との当たり判定
cat_x <= lion1_x + lion1_width　でライオンの右側との当たり判定

縦方向については、
lion1_y - cat_height <= cat_y　でライオンの上側との当たり判定
cat_y <= lion1_y + lion1_height　でライオンの下側との当たり判定

lion1_y と lion1_height は未定義でしたので追加してください。

```
    let lion1_dx = 10;
    let lion1_width = 170;
    let muki;
    let lion1_y = 480;
    let lion1_height = 76.36;
```

追加

①【課題】
　条件が整理できましたので、if 文を使って当たり判定のコードを追加してください。当たった場合、ひとまずコンソールに「当たった」と表示するようにしてください。

　（ヒントというかアドバイス）
　条件は横方向で２つ・縦方向で２つの合計４つの条件をすべて「かつ」でつなぎます。
　概念的に示すと、条件の４つを（条件 A）（条件 B）（条件 C）（条件 D）とするなら、

```
if(（条件 A）&&（条件 B）&&（条件 C）&&（条件 D）){
　　コンソールに「当たった」と表示する
}
```

　また、ひとつひとつの条件が長めですので、条件をつなぐ部分で適度に改行してかまいません。
　概念的に示すと、次のようになります。

条件を２行にした例
```
if(（条件 A）&&（条件 B）
　　&&（条件 C）&&（条件 D）){
　　コンソールに「当たった」と表示する
}
```

条件４行にした例
```
if(（条件 A）
　　&&（条件 B）
　　&&（条件 C）
　　&&（条件 D）){
　　コンソールに「当たった」と表示する
}
```

　わかると思いますが、move()内に追加してください。

解答例は次のページです。

解答例　①【課題】

```
if((lion1_x - cat_width <= cat_x) && cat_x <= lion1_x + lion1_width
    && lion1_y - cat_height <= cat_y && cat_y <= lion1_y + lion1_height){
    console.log("当たった");
}
```

7-7　「ゲームオーバー」と表示

ライオンに当たったら「ゲームオーバー」と表示されるようにしていきます。

【課題】

手順１　HTML 側にて、div タグで表示領域をつくり、id を割り当てましょう。
表示する場所・文字の大きさ・色などを決めるために、ひとまず「ゲームオーバー」を仮表示させてください。
CSS にて、場所・大きさ・色を調整しましょう。
場所・大きさ・色が決まったら、仮表示は削除してください。

手順２　JavaScript 側にて、手順１で作った表示領域を操作するための定数を定義してください。

手順３　当たり判定が真の場合に手順１で作った表示領域に「ゲームオーバー」を表示してください。

手順４　「ゲームオーバー」を表示のタイミングの際に、場所・色を指定してください。CSS の該当部分をコメントにして動作確認してください。

手順５　［開始］ボタンクリックのタイミングで「ゲームオーバー」の表示を消してください。

解答例は次のページです。

解答例　【課題】

手順１

表示部分 HTML 部

```
<body>
    <button id="start_btn">開始</button>
    <button id="stop_btn">停止</button>
    <div id="gamearea">
        <img id="cat" src="cat.svg">
        <img id="lion1" src="lion.svg">
        <img id="mouse" src="mouse.svg">
        <div id="disp">ゲームオーバー</div>
    </div>
```

gamearea 内に追加

CSS 部　見た目を確認しながら調整した結果、次のようにしてみました。

```
    #disp{
        position: absolute;
        top: 200px;
        left: 10px;
        font-size: 7rem;
        color: red;
    }
</style>
```

場所・大きさ・色が決まったら、仮表示は削除

```
    <img id="mouse" src="mouse.svg">
    <div id="disp"></div>
</div>
```

仮表示していた「ゲームオーバー」を削除

手順２

```
const mouse = document.querySelector("#mouse");
const cat = document.querySelector("#cat");
const disp = document.querySelector("#disp");
```

追加

手順３

```
if((lion1_x - cat_width <= cat_x) && cat_x <= lion1_x + lion1_width
    && lion1_y - cat_height <= cat_y && cat_y <= lion1_y + lion1_height){
    console.log("当たった");
    disp.textContent = "ゲームオーバー";
}
```

追加

手順４

JavaScript 部　場所・文字色の指定

```
if((lion1_x - cat_width <= cat_x) && cat_x <= lion1_x + lion1_width
    && lion1_y - cat_height <= cat_y && cat_y <= lion1_y + lion1_height){
    console.log("当たった");
    disp.textContent = "ゲームオーバー";
    disp.style.top = "200px";
    disp.style.left = "10px";
    disp.style.color = "red";
}
```

追加

CSS 部　該当部分コメントアウト

```
#disp{
    position: absolute;
    /* top: 200px;
    left: 10px; */
    font-size: 7rem;
    /* color: red; */
}
```

手順５

```
start_btn.addEventListener('click', e=>{
    start_btn.disabled = true;
    timeIdL1 = setInterval('lion1_move()', 100);
    timeIdM = setInterval('mouse_move()', 100);
    timeIdC = setInterval('cat_move()', 100);
    muki = 1;
    mukiM = 1;
    disp.textContent = '';
}, false);
```

追加

7-8　ゲームオーバーで停止させる

【課題１】
ゲームオーバーが表示されたタイミングでも、ライオンとネズミは動き続けています。
全体の返し処理も止めるようにしてください。
そして、全体が止まった時は［開始］ボタンを有効にしてください。

【課題２】
［開始］ボタンをクリックした時に猫がスタート位置へ行くようにコードを追加してください。

解答例は下に示しますが、頑張って見ずに挑戦してみましょう。

【課題１】解答例

```
if((lion1_x - cat_width <= cat_x) && cat_x <= lion1_x + lion1_width
    && lion1_y - cat_height <= cat_y && cat_y <= lion1_y + lion1_height){
    console.log("当たった");
    disp.textContent = "ゲームオーバー";
    disp.style.top = "200px";
    disp.style.left = "10px";
    disp.style.color = "red";
    clearInterval(timeId);
    start_btn.disabled = false;
}
```

追加

【課題２】解答例

```
start_btn.addEventListener('click', e=>{
    start_btn.disabled = true;
    timeId = setInterval('move()', 100);
    disp.textContent = '';
    cat_x = 370;
    cat_y = 10;
}, false);
```

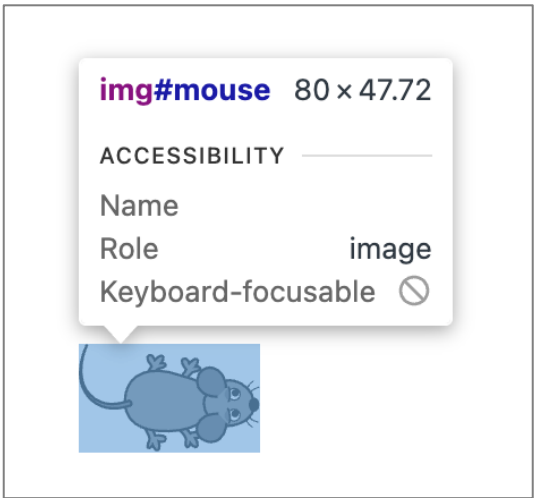
追加

7-8　ネズミとの当たり判定と「クリア」表示、微調整

7-8-1　当たり判定と「クリア」表示

【演習】
　考え方はライオンとの当たり判定と同じです。
　ネズミの縦方向の位置（mouse_y にしましょうか）、高さ（mouse_height にしましょうか）を定義し、ライオンとの当たり判定を参考に「クリア」と表示してください。
　クリアを表示する位置は、調整してください。
　色は青や緑にしてください。

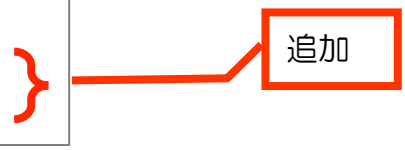
　ネズミの高さは検証ツールで確認してください。下図に示した通りになりますが…。



解答例【演習】

変数定義部

```
let mouse_x = 10;
let mouse_dx = 10;
let mouse_width = 80;
let mukiM = 1;
let mouse_y = 530;
let mouse_height = 47.72;
```



当たり判定部

```
    if((mouse_x - cat_width <= cat_x) && cat_x <= mouse_x + mouse_width
        && mouse_y - cat_height <= cat_y && cat_y <= mouse_y + mouse_height){
        disp.textContent = "クリア";
        disp.style.top = "200px";
        disp.style.left = "230px";
        disp.style.color = "blue";
        clearInterval(timeId);
        start_btn.disabled = false;
    }
};
```

7-8-2　微調整

【演習】
　ライオンやネズミの動き、猫の動きとキー操作の反応をもう少し滑らかにしてみましょう。
　setInterval でインターバル時間が現状では 100 になっています。
　これを 10 にしてみてください。
　その状態で動作確認すると、動きがメチャクチャ速くなることを感じられるとおもいます。
　そのままでは速すぎますので、ライオン、ネズミ、猫の移動距離を小さめの数字に調整して適切な速さにしてください。

　解答例は次のページです。

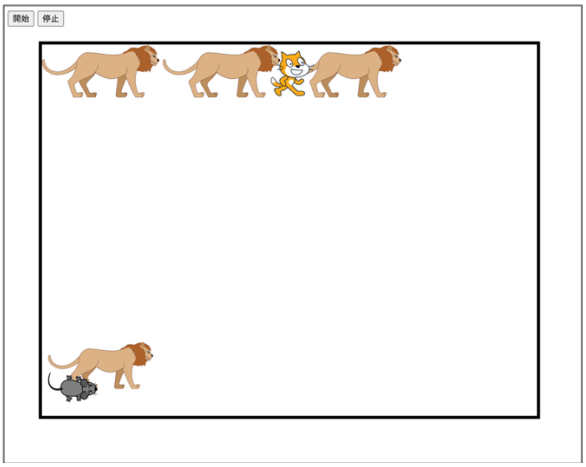
解答例【演習】　調整したところを ← で示します。

この解答例と違う数値でもかまいません。

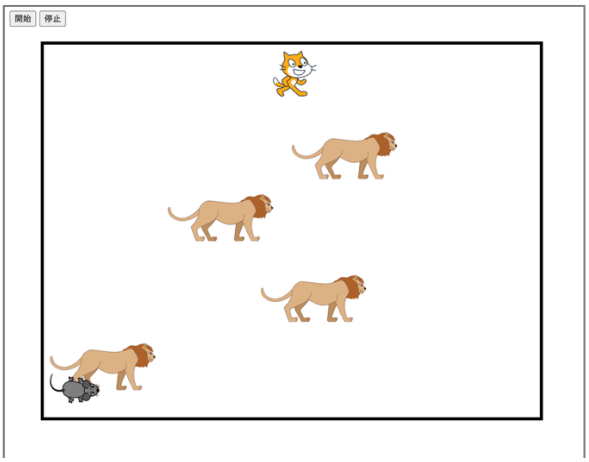
```
let timeId;
let lion1_x = 10;
let lion1_dx = 2; ←
let lion1_width = 170;
let muki = 1;
let lion1_y = 480;
let lion1_height = 76.36;

let mouse_x = 10;
let mouse_dx = 2; ←
let mouse_width = 80;
let mukiM = 1;
let mouse_y = 530;
let mouse_height = 47.72;

let keydown = '';
let cat_x = 370;
let cat_dx = 2;  ←
let cat_width = 70;
let cat_y = 10;
let cat_dy = 2;  ←
let cat_height = 73.57;

start_btn.addEventListener('click', e=>{
    start_btn.disabled = true;
    timeId = setInterval('move()', 10);
    disp.textContent = '';
    cat_x = 370;
    cat_y = 10;
}, false);
```

7-9　ライオンを増やす

【課題１】　ライオンをあと３頭追加してください。
　ひとまず追加するだけで良いです。id は lion2、lion3、lion4 にしましょう。



【課題２】大きさや位置をだいたい下図に示すような感じにしてください。
lion1 の CSS をコピーして top と left を調整するとよいです。



解答例

【課題１】

```
<div id="gamearea">
    <img id="cat" src="cat.svg">
    <img id="lion1" src="lion.svg">
    <img id="lion2" src="lion.svg">
    <img id="lion3" src="lion.svg">
    <img id="lion4" src="lion.svg">
    <img id="mouse" src="mouse.svg">
    <div id="disp"></div>
</div>
```

追加

【課題２】数値は違ってもかまいません

```
#lion1{
    position: absolute;
    top: 480px;
    left: 10px;
    width: 170px;
}
#lion2{
    position: absolute;
    top: 370px;
    left: 350px;
    width: 170px;
}
#lion3{
    position: absolute;
    top: 240px;
    left: 200px;
    width: 170px;
}
#lion4{
    position: absolute;
    top: 140px;
    left: 400px;
    width: 170px;
}
```

追加

7-9-1　複数のライオンを配列で扱う

追加した３頭のライオンを行ったり来たりさせます。
lion1 のコードをコピーして残り３つに対して「行ったり来たり」「当たり判定」を作ることもできますが、コード全体が長々となります。
そこで、位置情報と移動量を配列にして扱うようにしましょう。

① まずは追加した３頭のライオンを JavaScript で扱えるように定数として定義してください。

```
<script>
    const start_btn = document.querySelector("#start_btn");
    const stop_btn = document.querySelector("#stop_btn");
    const lion1 = document.querySelector("#lion1");
    const lion2 = document.querySelector("#lion2");
    const lion3 = document.querySelector("#lion3");
    const lion4 = document.querySelector("#lion4");
    const mouse = document.querySelector("#mouse");
    const cat = document.querySelector("#cat");
    const disp = document.querySelector("#disp");
```

追加

② それぞれのライオンで異なる量（横位置、変化量、向き、縦位置）を配列で値を持たせましょう。 ← が変更する部分です。

```
let timeId;
let lion1_x = [10, 350, 200, 400];
let lion1_dx = [2, 3, 4, 2];
let lion1_width = 170;
let muki = [1, 1, 1, 1];
let lion1_y = [480, 370, 240, 140];
let lion1_height = 76.36;
```

【解説】

```
let lion1_x = [10, 350, 200, 400];
```

CSS との対応は図のとおり。lion1_x は left の値。
lion1_dx は変化量なので CSS には記載なし。独自に決めました。
向きも別々に指定する必要があります。
lion1_y は top の値です。

```
#lion1{
    position: absolute;
    top: 480px;
    left: 10px;
    width: 170px;
}
#lion2{
    position: absolute;
    top: 370px;
    left: 350px;
    width: 170px;
}
#lion3{
    position: absolute;
    top: 240px;
    left: 200px;
    width: 170px;
}
#lion4{
    position: absolute;
    top: 140px;
    left: 400px;
    width: 170px;
}
```

③ ４頭を扱う定数をまとめて扱う配列 lions も定義しておきましょう。

```
const lion1 = document.querySelector("#lion1");
const lion2 = document.querySelector("#lion2");
const lion3 = document.querySelector("#lion3");
const lion4 = document.querySelector("#lion4");
const lions = [lion1, lion2, lion3, lion4];
```

追加

③ move()関数の中のライオンを動かしている部分を変更していきます。
この部分です。

```
lion1_x += lion1_dx;
lion1.style.left = lion1_x + "px";

if(lion1_x <= 0 || 800 <= lion1_x + lion1_width){
    lion1_dx = -lion1_dx;
    muki = -muki;
    lion1.style.transform = "scale(" + muki + ", 1)";
}
```

４頭のライオンに対応させるため、for 文でのループ処理にします。

```
for(let i = 0; i < lion1_x.length; i++){
    lion1_x += lion1_dx;
    lion1.style.left = lion1_x + "px";

    if(lion1_x <= 0 || 800 <= lion1_x + lion1_width){
        lion1_dx = -lion1_dx;
        muki = -muki;
        lion1.style.transform = "scale(" + muki + ", 1)";
    }
}
```

ここはまだそのまま。次の手順で変更します。

lion1_x.length は配列 lion1_x の長さ（要素の数）です。ここでは要素が４つ（ライオン４頭分）に相当します。

④ ループの中で、配列の部分を変更します。
配列にしたのは、lion1_x、lion1_dx、muki、lion1_y、lions でしたね。

```
for(let i = 0; i < lion1_x.length; i++){
    lion1_x[i] += lion1_dx[i];
    lions[i].style.left = lion1_x[i] + "px";

    if(lion1_x[i] <= 0 || 800 <= lion1_x[i] + lion1_width){
        lion1_dx[i] = -lion1_dx[i];
        muki[i] = -muki[i];
        lions[i].style.transform = "scale(" + muki[i] + ", 1)";
    }
}
```

ここまでの段階で、ライオン４頭がそれぞれ別々の速さで左右に行ったり来たりすることを確認してください。

7-9-2　当たり判定もループの中へ

当たり判定も効かなくなっています。
ループの中へ入れて、配列にすべき部分は配列に変更しましょう。

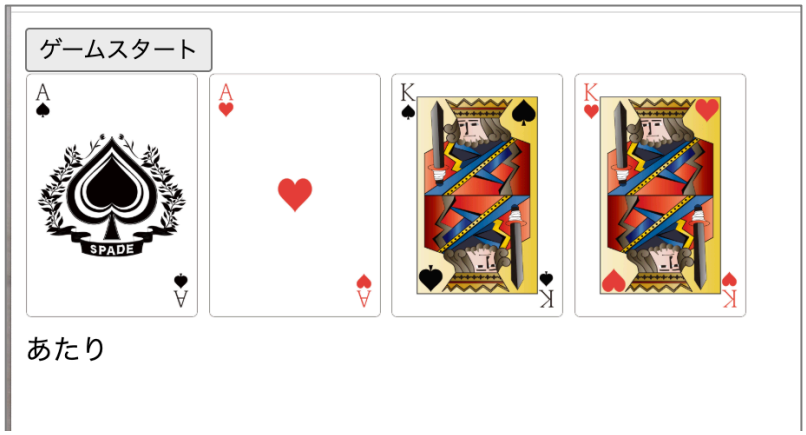
```
for(let i = 0; i < lion1_x.length; i++){
    lion1_x[i] += lion1_dx[i];
    lions[i].style.left = lion1_x[i] + "px";

    if(lion1_x[i] <= 0 || 800 <= lion1_x[i] + lion1_width){
        lion1_dx[i] = -lion1_dx[i];
        muki[i] = -muki[i];
        lions[i].style.transform = "scale(" + muki[i] + ", 1)";
    }

    if((lion1_x[i] - cat_width <= cat_x) && cat_x <= lion1_x[i] + lion1_width
        && lion1_y[i] - cat_height <= cat_y && cat_y <= lion1_y[i] + lion1_height){
        console.log("当たった");
        disp.textContent = "ゲームオーバー";
        disp.style.top = "200px";
        disp.style.left = "10px";
        disp.style.color = "red";
        clearInterval(timeId);
        start_btn.disabled = false;
    }
}
```

ループの中へ入れた。
lion1_x、lion1_y を配列に変更。

４頭全てのライオンで当たり判定が正常に機能することを確認してください。

これで完成です。
ライオンやネズミ、猫の動く速さを調整してみたり、ライオンを増やしたり、いろいろとアレンジしてみてください。

第８章　神経衰弱　簡易版



カード４枚のみで神経衰弱を作ってみましょう。

サンプル　　https://js.drmpls.com/shinkei/shinkei.html

8-1　表示部分をつくる

① shinkei フォルダを作ってください。

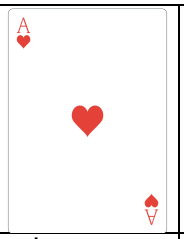
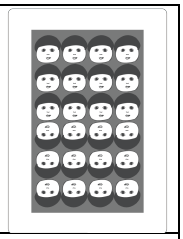
②カードの画像は下記からダウンロード、または講師かから入手してください。

https://drmpls.com/javascript_super_intro/

ダウンロードしたイラストは shinkei フォルダに入れてください。

（※使用する画像は「チコデザ」https://chicodeza.com/freeitems/torannpu-illust.html から入手し、大きさを調整したものです。）

必要な画像は次の５つです。ファイル名も下記の通りに書き換えてください。（書き換える必要がある場合）

				
sa.png	sk.png	ha.png	hk.png	ura.png

③ shinkei.html を作り、shinkei フォルダに保存してください。

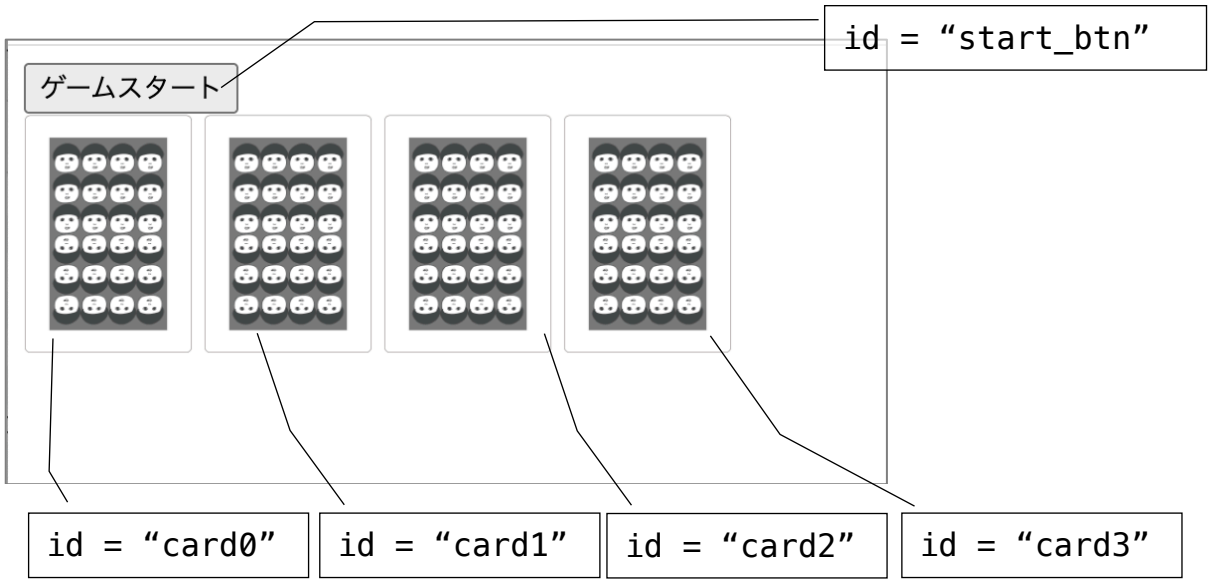
④ 【演習】

shinkei.html に html の基本構造をつくり、［ゲームスタート］ボタンと裏面のカードを４つ表示してください。

ボタンは button タグ、画像表示に img タグを使い、id は下図のとおりにしてください。

カード４枚はひとつの div の中にまとめてください。

style の指定は特に必要ありません。



⑤ 【演習】
表示した５つの部品を JavaScript で扱えるように定数で定義してください。
定数名は id 名と同じにしておくとわかりやすいでしょう。

解答例は次のページです。

解答例　④　【演習】

```
<body>
    <button id="start_btn">ゲームスタート</button>
    <div>
        <img id="card0" src="ura.png">
        <img id="card1" src="ura.png">
        <img id="card2" src="ura.png">
        <img id="card3" src="ura.png">
    </div>

</body>
```

追加

解答例　⑤　【演習】

```
    </div>
    <script>
        const start_btn = document.querySelector('#start_btn');
        const card0 = document.querySelector('#card0');
        const card1 = document.querySelector('#card1');
        const card2 = document.querySelector('#card2');
        const card3 = document.querySelector('#card3');
    </script>
</body>
```

追加

8-2　表側のカード（ファイル名）配列をシャッフルする　準備編

神経衰弱ですから、伏せられたカードは順番がバラバラです。
ここでは、表側のカードを配列にしておき、その中身をランダムに入れ替える処理を作ります。

① まずは対象となる配列を作りましょう。表側のカードの画像名を下記のように配列にしてください。

```
    const card3 = document.querySelector('#card3');
    let cards = ['sa.png', 'sk.png', 'ha.png', 'hk.png'];
</script>
```

追加

② 配列をシャッフルする関数を作ります。この関数に配列を渡せば、中身をランダムに入れ替えて返してくれる機能にします。

まず関数の外枠をつくりましょう。
関数名は shuffle です。引数（ひきすう）として配列をひとつ受け取ります。

```
    function shuffle(array){

    }

</script>
```

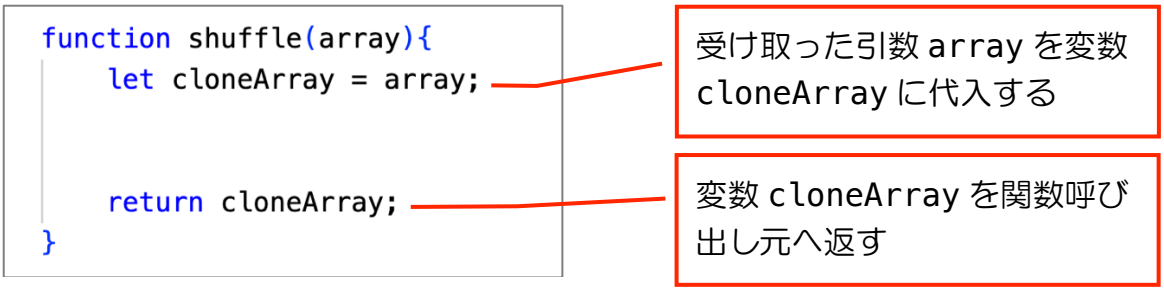
関数名

関数が受け取る引数

※ 関数にする理由は、ひとかたまりの処理を関数化（ひとまとめ）しておくと、全体の流れが把握しやすくなること、複数の箇所で同じ処理をするときに共通して利用できること、などの利点があるためです。

※ 仕組みはわからなくても、この関数を使って配列の中身（要素）をシャッフルできれば OK です。

③ 受け取った引数を関数内部の変数にセットする部分、値を返す部分をつくりましょう。



この段階では、この関数は受け取ったものをそのまま何も加工（処理）せずに呼び出し元へ返します。

```
return cloneArray;
```

の cloneArray の部分、「戻り値」と言います。
値を返すので「返り値」と呼ぶ場合もありますが、「返り血」を連想してしまう（人を刀で切ったり、刃物で刺したりしたときに切られた人の血をブシャーっと自分に浴びるシーン）ので「戻り値」と呼ぶほうを推奨される…と聞いたことがあります。

④ 渡したらただそのまま返ってくる、という状況を一度実感して見ましょう。
［ゲームスタート］ボタンがクリックされたときのイベントリスナーをつくり、コンソールに返ってきた値を表示してみます。

【演習】復讐がてら、［ゲームスタート］ボタンがクリックされたときのイベントリスナーをつくってください。
まずは枠組みだけで良いです。

クリックされたときのイベント処理は何度も出てきていますので、そろそろ自力で作れるのではないだろうか……？との期待をこめて。

解答例は次のページです。

解答例　④【演習】

```
start_btn.addEventListener('click', e=>{

}, false);

function shuffle(array){
```

追加

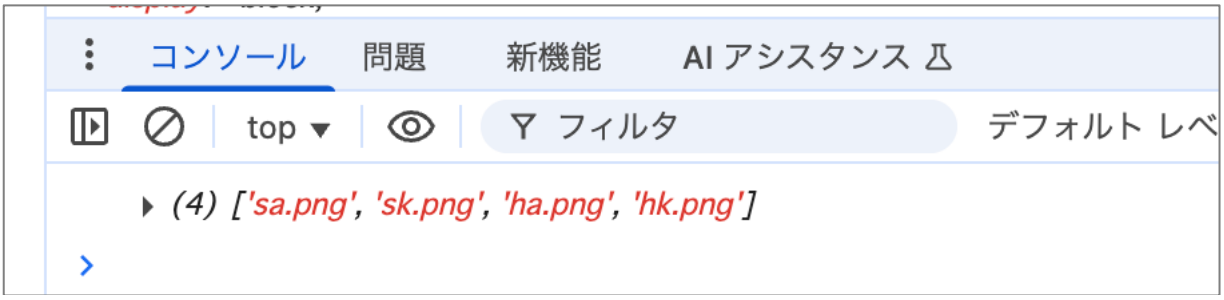
⑤ 配列 cards を shuffle 関数に渡し、返ってきた値を元々の cards に代入しましょう。
　この cards をコンソールに表示してみます。

```
start_btn.addEventListener('click', e=>{
    cards = shuffle(cards);
    console.log(cards);
}, false);
```

関数に渡して、戻ってきた値を自分自身に入れ替える

コンソールに表示する

　表示した様子を示します。元の cards の内容と変わりません。shuffle 関数内ではなにも処理をされずにそのまま返ってきたことがわかります。



　こうすることで、shuffle 関数の戻り値を確認できる準備もできました。

8-3　表側のカード（ファイル名）配列をシャッフルする　処理編

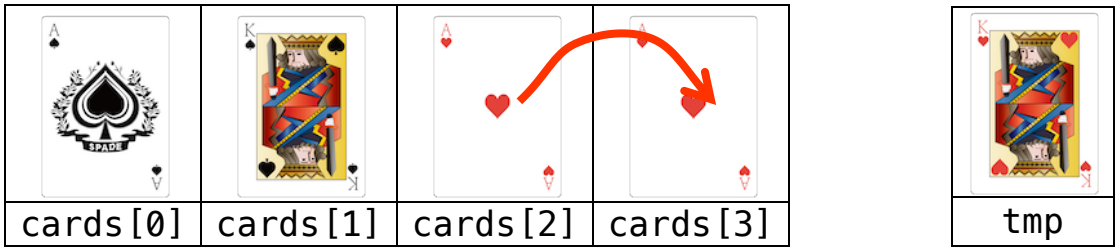
シャッフルする処理をつくります。

8-3-1　処理のしくみについて

処理の仕組みを説明します。
配列 cards の中身は下図のとおりです。一時的な保管場所の tmp も用意しておきます。

■ 末尾（インデックス番号 3）の要素から順番に処理していきます。
(1) 末尾（インデックス番号 3）の要素を tmp に入れます。コピーになります。

(2)交換対象を cards[0]～cards[3]のうちのどれかに決め、それを cards[3] に入れます。cards[3]はそれで上書きされます。（図では cards[2]を例にしました。）このとき、自分自身（cards[3]）で上書きされるケースもあります。



(3)交換対象に tmp を入れます。交換対象はそれで上書きされます。（図では cards[2]を例にしました。）

■ 次に末尾の１つ前（インデックス番号 2）の要素を処理していきます。
(1)末尾の１つ前（インデックス番号 2）の要素を tmp に入れます。コピーになります。

(2)交換対象を cards[0]～cards[2]のうちのどれかに決め、それを cards[2]に入れます。cards[2]はそれで上書きされます。（図では cards[0]を例にしました。）このとき、自分自身（cards[2]）で上書きされるケースもあります。

(3)交換対象に tmp を入れます。交換対象はそれで上書きされます。（図では cards[0]を例にしました。）

これで、末尾から２つまでは確実に入れ替わったことになります。
（入れ替わらない場合もありますが、処理は実施済みです。）

同様に、インデックス番号１を処理し、次にインデックス番号０について処理することで、全ての要素について処理が完了してシャッフルの作業が終わります。

この流れを shaffle 関数内にコーディング（プログラムとして書く）します。

8-3-2　shuffle 関数のシャッフル部分のコーディング

① 繰り返す処理を for 文で用意してください。

```
function shuffle(array){
    let cloneArray = array;

    for(let i = cloneArray.length - 1; i >= 0; i-- ){

    }

    return cloneArray;
}
```

追加

配列の長さ
ここでは要素が４つの配列が渡ってきているので、4。

最後の要素のインデックス番号は、要素数-1

配列 cloneArray の最後の要素から、ループするごとに i を一つずつ減らします。i が0以上の間は（0になるまで）処理を繰り返します。

② 0～3 のどれかを変数 rand に入れます。

```
for(let i = cloneArray.length - 1; i >= 0; i-- ){
    let rand = Math.floor(Math.random() * (i + 1));

}
```

追加

4-5で Math.randam()と Math.floor の説明をしていますので、改めて確認してください。

ループが繰り返されるたびに、rand は次のようになります。
i = 3 のときは rand は 0~3 のどれか
i = 2 のときは rand は 0~2 のどれか
i = 1 のときは rand は 0~1 のどちらか
i = 0 のときは rand は 0~0 なので 0 にしかならない

③ 入れ替えの処理を追加しましょう。

```
for(let i = cloneArray.length - 1; i >= 0; i-- ){
    let rand = Math.floor(Math.random() * (i + 1));
    let tmp = cloneArray[i];
    cloneArray[i] = cloneArray[rand];
    cloneArray[rand] = tmp;
}
```

追加

処理の仕組みでの(1)に相当

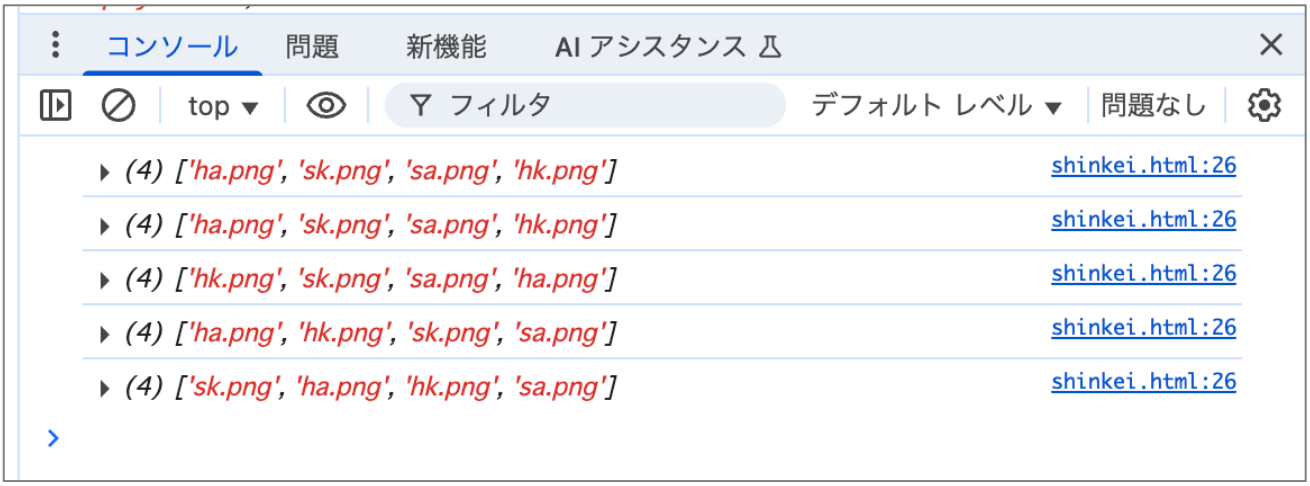
処理の仕組みでの(2)に相当

処理の仕組みでの(2)に相当

④ 動作確認してみましょう。

［ゲームスタート］ボタンをクリックして、コンソールで配列の内容を見てください。

［ゲームスタート］ボタンをクリックするたびに、配列の中身の順番が異なっていることが確認できれば OK です。



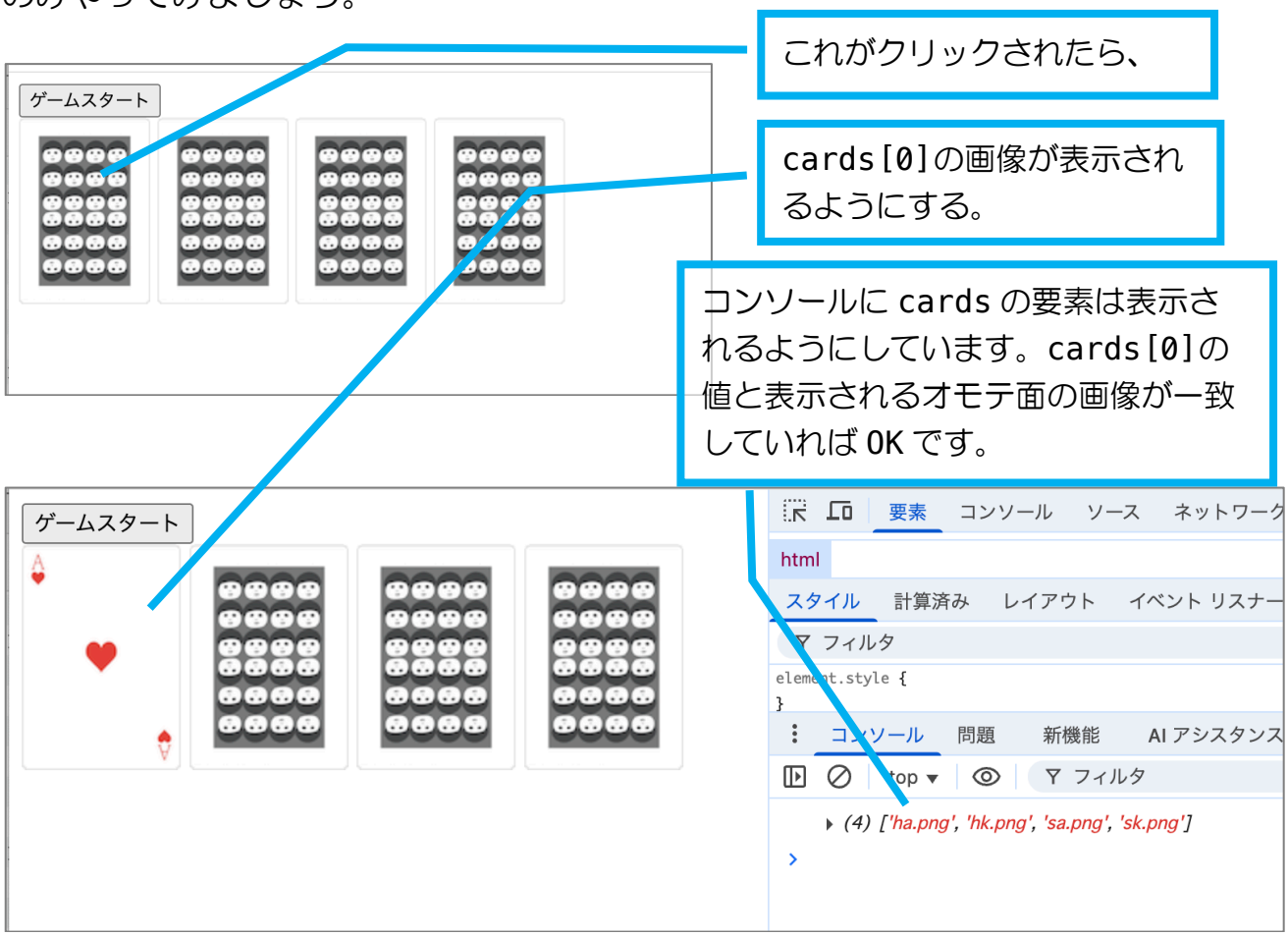
偶然にも同じになることもあります。

何度も繰り返し試してみてください。同じものが多数続かないようであれば大丈夫です。

8-4　クリックされたらオモテ側を表示

8-4-1　とりあえず card0 のみ

【課題】
　クリックされたらオモテ側が表示されるようにしてください。とりあえず card0 のみやってみましょう。



解答例は次のページです。

解答例　8-4-1【課題】

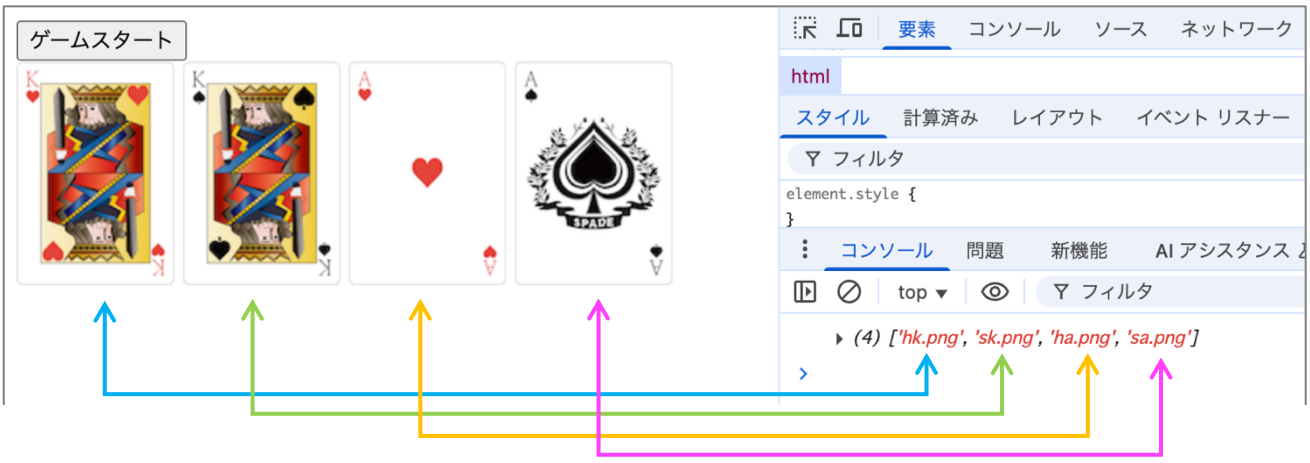
```
    card0.addEventListener('click', function(){
        card0.src = cards[0];

    }, false);

</script>
```

追加

8-4-2　残り3つもオモテ側が表示されるようにしよう

【課題】card0 に対して作ったイベントリスナーを参考に、残りの3つもクリックされたらオモテ側が表示されるようにしてください。

　開かれたカードとコンソールに表示される cards の要素の並び順が一致していることを確認してください。



解答例は次のページです。

　なお、現時点では開かれたカードはそのまま開かれた状態から変化しません。［ゲームスタート］ボタンをクリックしても、裏面になるようなコードは書いていませんのでオモテ面のままです。
　裏面にするようなコードは、この先の段階で追加していきます。

解答例　8-4-2【課題】

```
    card0.addEventListener('click', function(){
        card0.src = cards[0];

    }, false);

    card1.addEventListener('click', function(){
        card1.src = cards[1];

    }, false);

    card2.addEventListener('click', function(){
        card2.src = cards[2];

    }, false);

    card3.addEventListener('click', function(){
        card3.src = cards[3];

    }, false);

</script>
```

追加

8-4-3　ループ化

長ったらしいので、処理をループ化しましょう。４つのイベントは、0, 1, 2, 3 が違うだけですので、ループ化しやすいです。

```
    card0.addEventListener('click', function(){
        card0.src = cards[0];

    }, false);

    card1.addEventListener('click', function(){
        card1.src = cards[1];

    }, false);

    card2.addEventListener('click', function(){
        card2.src = cards[2];

    }, false);

    card3.addEventListener('click', function(){
        card3.src = cards[3];

    }, false);
```

cards[i]としておいて i を 0～3 まで変化させさせれば対応できます。

card0～card3 は cardi としたとしても cardi という変数名になってしまい、変数 i を直接使えません。
そこで、配列化します。

① 次のように定数 card0～card3 を配列に入れてしまいましょう。

```
const card0 = document.querySelector('#card0');
const card1 = document.querySelector('#card1');
const card2 = document.querySelector('#card2');
const card3 = document.querySelector('#card3');
const card_list = [card0, card1, card2, card3];
```

追加

こうすれば、card_list[i]とすることで i を 0～3 まで変化させることで

card_list[0]は card0
card_list[1]は card1
card_list[2]は card2
card_list[3]は card3

になり、ループ化に対応できます。

② for ループを作りましょう。

for ループ追加

```
for(let i = 0; i < card_list.length; i++){
    card_list[i].addEventListener('click', function(){
        card_list[i].src = cards[i];
    }, false);
}

card1.addEventListener('click', function(){
    card1.src = cards[1];

}, false);

card2.addEventListener('click', function(){
    card2.src = cards[2];

}, false);

card3.addEventListener('click', function(){
    card3.src = cards[3];

}, false);
```

card0 に対してのイベントの部分を部分的に変更

この部分は削除してください。

これまでと変わらない動作であることを確認してください。

【演習】
［ゲームスタート］ボタンをクリックしたときに、４枚すべてが ura.png を表示するようにコードを追加してください。

解答例【演習】

```
start_btn.addEventListener('click', e=>{
    for(let i = 0; i< card_list.length; i++){
        card_list[i].src = 'ura.png';
    }
    cards = shuffle(cards);
    console.log(cards);
}, false);
```

追加

for ループを使わずに、card0.src = ‘ura.png’…という感じで４枚分のコードを書いても構いません。４枚程度ですから。
ただ、これがカードフルセットの５２枚の場合は５２行書くことになってしまいますので、for ループを使った方がスマートに書けます。

8-5　めくられたカードの情報を保持しておく

① めくられたカードの情報を保持しておくための空の配列を定義しましょう。

```
const card_list = [card0, card1, card2, card3];
let cards = ['sa.png', 'sk.png', 'ha.png', 'hk.png'];
let mekuri = [];          // めくられたカードの値を格納 [位置, 値]を格納
```

追加

これはわかりやすくするためのコメントなので、書かなくてもよいです。

② めくるたびに配列 mekuri に要素を追加しましょう。

```
for(let i = 0; i < card_list.length; i++){
    card_list[i].addEventListener('click', function(){
        card_list[i].src = cards[i];
        mekuri.push([i, cards[i].substring(1,2)]);
        console.log(mekuri);
    }, false);
}
```

追加

動作確認のためのログ出力

【解説】

配列.push(要素)　配列に要素を追加するメソッド。
ここでは、[i, cards[i].susbstring(1,2)]という要素を追加しています。
[インデックス番号,左から２文字目]を追加することになります。

文字列.substring(開始インデックス,終了インデックス)

文字列から文字を切り出します。開始インデックスから終了インデックスの一つ前までの文字を切り出します。
例として、cards = ['sa.png', 'sk.png', 'ha.png', 'hk.png']の場合で説明します。

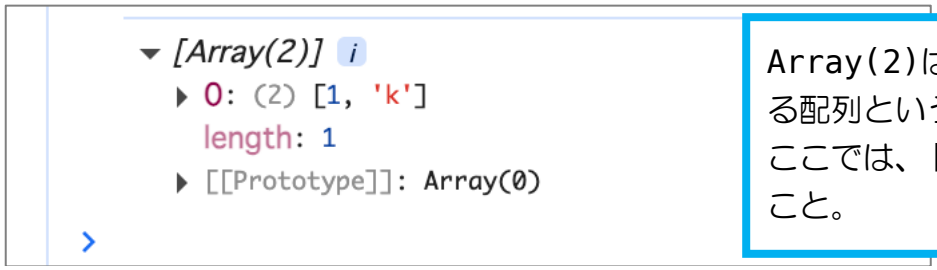
例えば i=0 の場合、cards[0]は'sa.png'です。

cards[0].substring(1,2)は、'sa.png'という文字列のインデックス 1 から 2 のひとつ前（すなわち 1）までの文字列を切り出すので、'a'になります。

③ 動作確認してみましょう。

例えば、下図のように左から２枚目をクリックしたとします。

▼にすると[Array(2)]の中身を見ることができます。

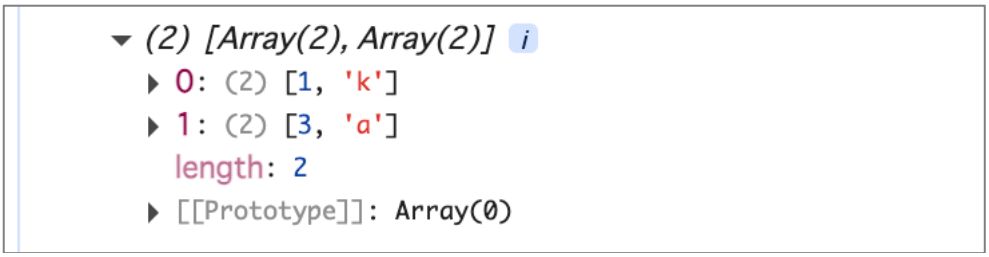


Array(2)は、要素を２つ含んでいる配列という意味。
ここでは、[1, 'k']という配列のこと。

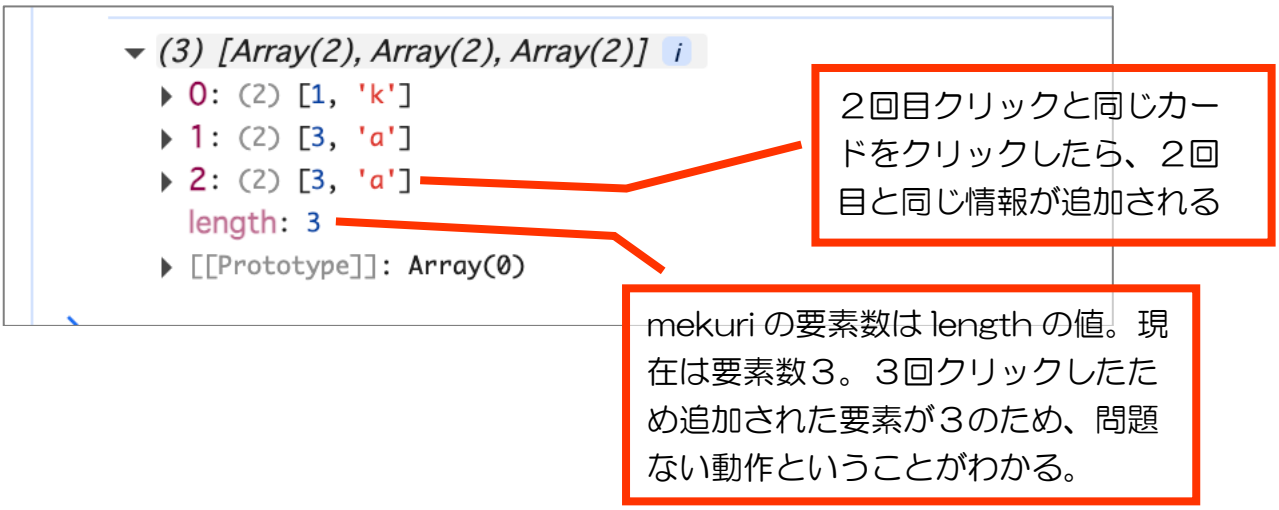
例では、[1, 'k']です。インデックス番号が１、値は k という情報がひとつ入りました。

つづけて、別のカードをクリックしてみましょう。

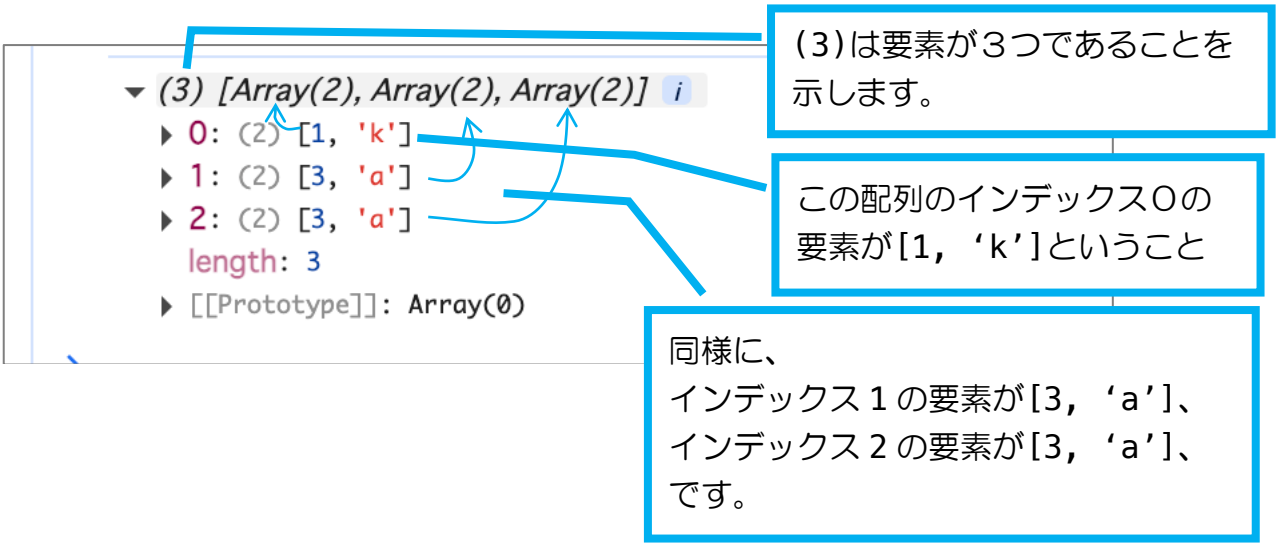
インデックス番号が３、値はａという情報が要素として追加されたことがわかります。



なお、現時点ではクリックするたびに追加されますので、すでにオモテ面が表示されているカードをまたクリックしても情報が追加されます。



要素が３つくらい入ると説明しやすくなります。



8-6　２枚めくったときに同じかどうかの判定

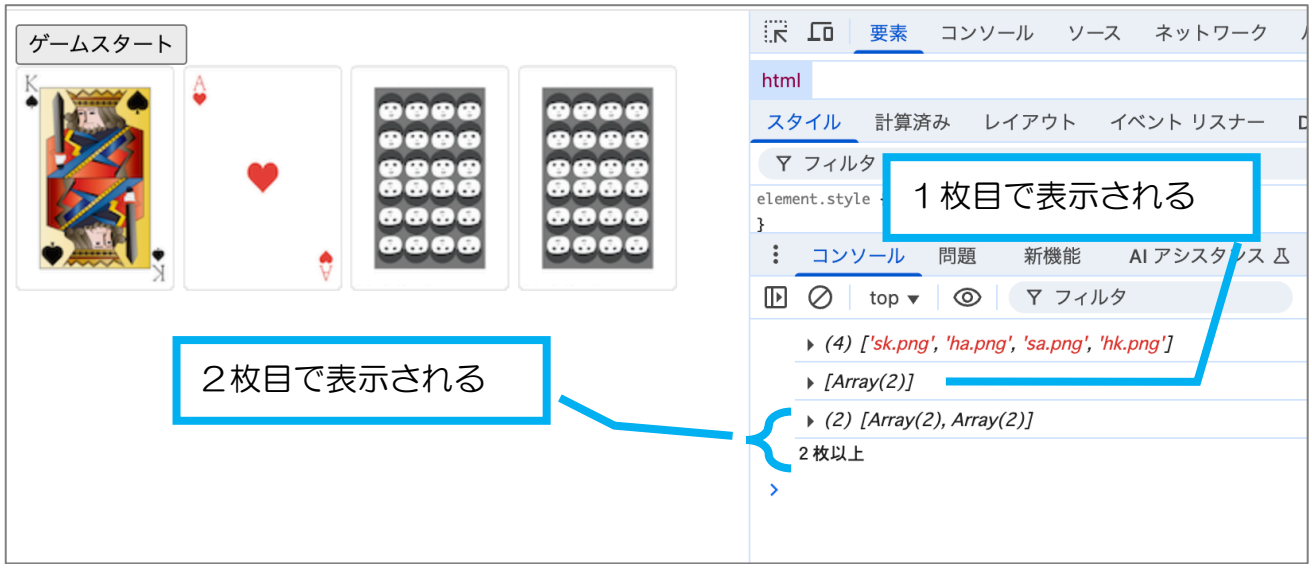
２枚目かどうかの判断は、配列 mekuri の要素数が２になれば２枚目であることがわかります。
判定には、幅を持たせて mekuri.length > 1 で評価してみましょう。

① めくったときに mekuri に要素追加した直後に判定するコードを追加してください。

```
for(let i = 0; i < card_list.length; i++){
    card_list[i].addEventListener('click', function(){
        card_list[i].src = cards[i];
        mekuri.push([i, cards[i].substring(1,2)]);
        console.log(mekuri);
        if(mekuri.length > 1){
            console.log('２枚以上');
        }
    }, false);
}
```

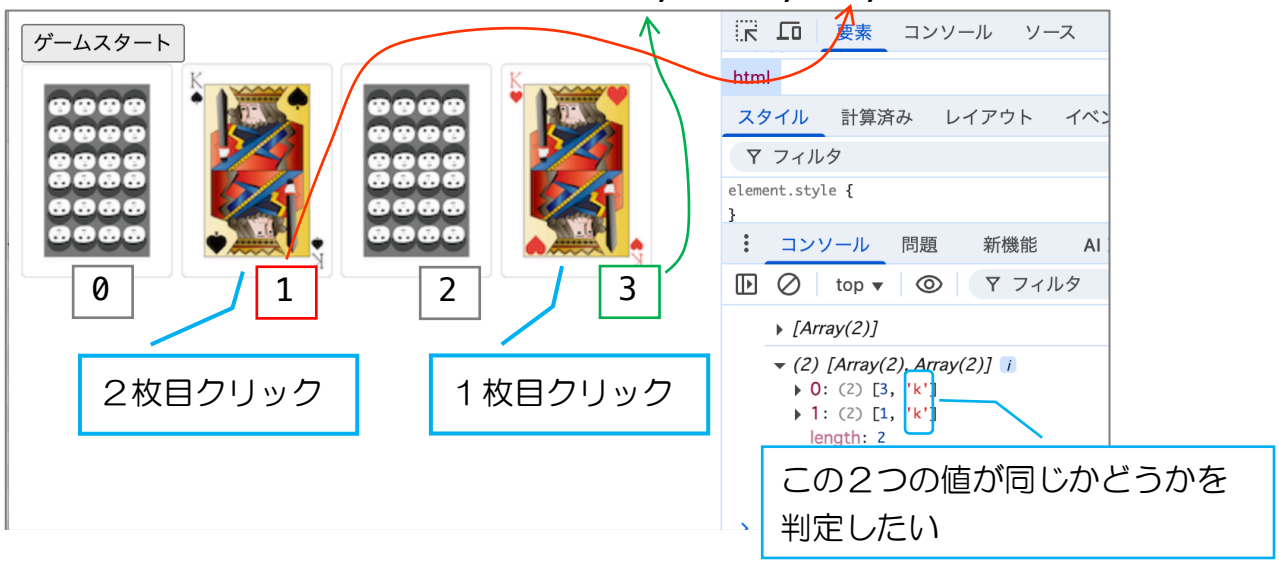
追加

２枚目をめくったときにコンソールに「２枚以上」と表示されることを確認してください。

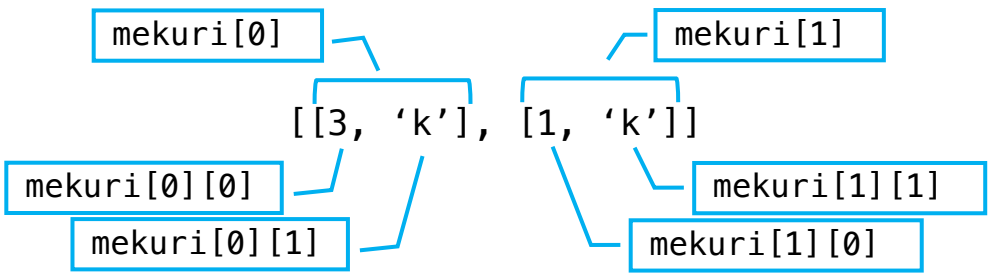


mekuri.length > 1 で判定していますので、３枚目でも同じように表示されますが、今は気にする必要はありません。

② 配列 mekuri の要素数が２のときに、同じ数値かどうかを判定しましょう。
mekuri には、２枚めくった時点で２つの配列が入っています。
例として下図のような状態の場合、[[3, ‘k’], [1, ‘k’]]です。



配列の中に配列が入っていて、慣れないと複雑に感じますが、



つまり、mekuri[0][1] == mekuri[1][1]を評価して同じかどうかを判定します。

下記のように追加してください。
alert はポップアップするメッセージボックス（ダイアログボックス）を表示します。

```
console.log(mekuri);
if(mekuri.length > 1){
    console.log('２枚以上');
    if(mekuri[0][1] == mekuri[1][1]){
        alert('あたり');
    }
}
```

追加

なお、３枚目以降の動きは想定していない作りです。動作確認を何度も行うと意図しない動きになることがあります。　そこで…

③ ２枚目の処理で配列 mekuri をリセットしましょう。
mekuri の要素を空にするまたは要素数を０にするという処理をします。
ここでは、要素数を０にする命令を追加してみましょう。

```
if(mekuri.length > 1){
    console.log('２枚以上');
    if(mekuri[0][1] == mekuri[1][1]){
        alert('あたり');
        mekuri.length = 0;
    }
}
```

追加

（ mekuri.length = 0 の代わりに、mekuri = [] でもいいです。）

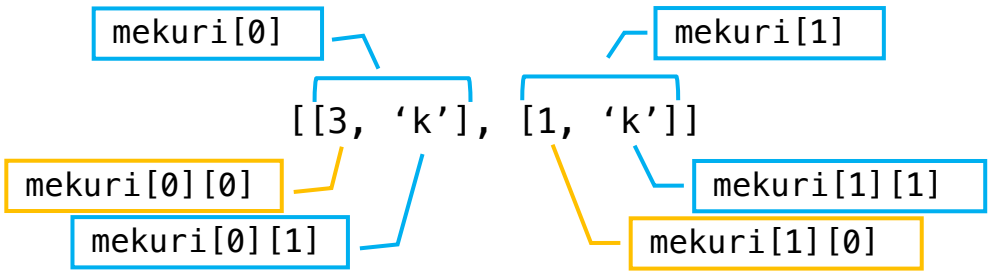
動作確認してください。
３枚目をクリックしたとき、コンソールに表示される mekuri の要素が１つになっていれば OK です。

④ 違ったらまた裏にします。
はずれたことがわかるように「はずれ」のダイアログボックスを出してから裏にする命令を書きましょう。mekuri の要素を空にする命令も入れましょう。

```
if(mekuri.length > 1){
    console.log('２枚以上');
    if(mekuri[0][1] == mekuri[1][1]){
        alert('あたり');
        mekuri.length = 0;
    } else {
        alert('はずれ');
        card_list[mekuri[0][0]].src = 'ura.png';
        card_list[mekuri[1][0]].src = 'ura.png';
        mekuri.length = 0;
    }
}
```

追加

mekuri[0][0], mekuri[1][0]を使っている意味は何でしょうか？



めくられたカードの位置を示す部分だからです。

動作確認すると、ハズレのときに２枚目がウラのままに見えます。
実はオモテになっているはずなのですが、すぐにウラになってしまうためにオモテが見えないような状況になっています。

8-7　１秒の待ち時間のあとにウラにする

① 待ち時間を設定するには、setTimeout というしくみを使います。

これまでに何度か使っている setInterval と似たものですが、

setInterval はインターバル時間ごとに繰り返し実行する設定
setTimeout はインターバル時間後に 1 度だけ実行する設定

になります。

下記のように、カードを裏にする部分と mekuri の要素を空にする部分を setTimeout の中に入れてください。

```
} else {
    alert('はずれ');
    setTimeout(function(){
        card_list[mekuri[0][0]].src = 'ura.png';
        card_list[mekuri[1][0]].src = 'ura.png';
        mekuri.length = 0;
    }, 1000);
}
```

追加

追加

8-8　オモテのカードをクリックしたときに mekuri へ追加しない処理

現状では、すでにめくられてオモテが表示されているカードをクリックしても配列 mekuri へその情報が追加されてしまいます。
ウラが表示されている場合だけ配列 mekuri へ情報を追加するためには、カードがウラとオモテのどちらが表示されている状態なのかを把握しておく必要があります。

① そこで、新たに配列 omote を定義しましょう。
配列 omote には、オモテのときが true，ウラのときが false を入れておくことにします。

②［ゲームスタート］がクリックされたときに４枚とも false にしましょう。

```
let cards = ['sa.png', 'sk.png', 'ha.png', 'hk.png'];
let mekuri = [];          // めくられたカードの値を格納 [位置, 値]を格納
let omote = [];           // カードがオモテかウラかの状態

start_btn.addEventListener('click', e=>{
    for(let i = 0; i< card_list.length; i++){
        card_list[i].src = 'ura.png';
    }
    omote = [false, false, false, false];
    cards = shuffle(cards);
    console.log(cards);
}, false);
```

追加 ①

追加 ②

③ カードがクリックされたときの処理をウラのときだけに限定し、配列 omote の該当する要素を true にしましょう。

```
for(let i = 0; i < card_list.length; i++){
    card_list[i].addEventListener('click', function(){
        if(!omote[i]){
            omote[i] = true;
            card_list[i].src = cards[i];
            mekuri.push([i, cards[i].substring(1,2)]);
            console.log(mekuri);
            if(mekuri.length > 1){
                console.log('２枚以上');
                if(mekuri[0][1] == mekuri[1][1]){
                    alert('あたり');
                    mekuri.length = 0;
                } else {
                    alert('はずれ');
                    setTimeout(function(){
                        card_list[mekuri[0][0]].src = 'ura.png';
                        card_list[mekuri[1][0]].src = 'ura.png';
                        mekuri.length = 0;
                    }, 1000);
                }
            }
        }
    }, false);
}
```

追加

追加

omote[i] = true; でクリックされたカードのオモテウラ情報をオモテにしています。

④ ２枚めくってハズレだったときに、見た目がウラになるタイミングで配列 omote の該当する部分もウラ（false）にしましょう。

⑤［ゲームスタート］がクリックされたときに mekuri の要素数も 0 にしましょう。

```
start_btn.addEventListener('click', e=>{
    mekuri.length = 0;      // mekuriを確実に初期化（要素なし）にしておく
    for(let i = 0; i< card_list.length; i++){
        card_list[i].src = 'ura.png';
    }
    omote = [false, false, false, false];
    cards = shuffle(cards);
    console.log(cards);
}, false);
```

追加

8-9　「あたり」「はずれ」の表示

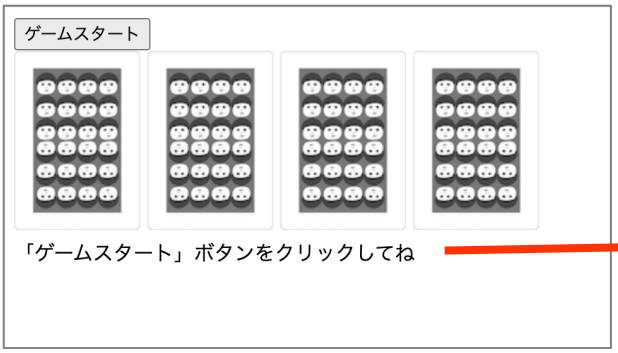
「あたり」「はずれ」の表示を、ダイアログボックスではなくカードの下に表示するようにしましょう。

① まずは表示領域をつくりましょう。［ゲームスタート］のクリックを促すようにします。id=”kekka”としておきましょう。

```
<body>
    <button id="start_btn">ゲームスタート</button>
    <div>
        <img id="card0" src="ura.png">
        <img id="card1" src="ura.png">
        <img id="card2" src="ura.png">
        <img id="card3" src="ura.png">
    </div>
    <div id="kekka">「ゲームスタート」ボタンをクリックしてね</div>
```

追加

見た目はこうなります。



追加された表示領域

② 上記表示領域を JavaScript で操作できるように定数で定義しましょう。

```
<script>
    const start_btn = document.querySelector('#start_btn');
    const card0 = document.querySelector('#card0');
    const card1 = document.querySelector('#card1');
    const card2 = document.querySelector('#card2');
    const card3 = document.querySelector('#card3');
    const kekka = document.querySelector('#kekka');
    const card_list = [card0, card1, card2, card3];
```

追加

③ alert にしている部分を表示領域に表示するように変更してください。

```
if(mekuri.length > 1){
    console.log('２枚以上');
    if(mekuri[0][1] == mekuri[1][1]){
        kekka.textContent = 'あたり';
        mekuri.length = 0;
    } else {
        kekka.textContent = 'はずれ';
        setTimeout(function(){
            card_list[mekuri[0][0]].src = 'ura.png';
```

変更

変更

④ ついでに、「あたり」「はずれ」判定後にまためくりはじめたとき（mekuri の要素数が１以下）のときは何も表示しないようにしましょう。

```
if(mekuri.length > 1){
    console.log('２枚以上');
    if(mekuri[0][1] == mekuri[1][1]){
        kekka.textContent = 'あたり';
        mekuri.length = 0;
    } else {
        kekka.textContent = 'はずれ';
        setTimeout(function(){
            card_list[mekuri[0][0]].src = 'ura.png';
            card_list[mekuri[1][0]].src = 'ura.png';
            omote[mekuri[0][0]] = false;
            omote[mekuri[1][0]] = false;
            mekuri.length = 0;
        }, 1000);
    }
} else {
    kekka.textContent = '';
}
```

else 節を追加

8-9　［ゲームスタート］でカードが出現するようにしよう

この時点でほとんど完成ですが、［ゲームスタート］をクリックする前でもカードをクリックできてしまい、オモテが表示されてしまいます。そして、意図しない動作になることもあります。

そこで、最初は下図のようにカードが表示されていない状態にしておき…

［ゲームスタート］がクリックされたらカードが見えるようにしたいと思います。

5-3で style の属性のひとつ visibility を使うことで見えたり見えなかったりできることを学びました。ここでも使ってみましょう。

① 次のように、まずは画像を非表示にします。

画像４枚分、
非表示に変更

```
<div>
    <img id="card0" style="visibility: hidden;">
    <img id="card1" style="visibility: hidden;">
    <img id="card2" style="visibility: hidden;">
    <img id="card3" style="visibility: hidden;">
</div>
```

非表示ですので、画像ファイル名も不要なため削除しました。

② ［ゲームスタート］がクリックされたらカードが見えるようにしましょう。

```
start_btn.addEventListener('click', e=>{
    mekuri.length = 0;      // mekuriを確実に初期化
    for(let i = 0; i< card_list.length; i++){
        card_list[i].src = 'ura.png';
    }
    omote = [false, false, false, false];
    cards = shuffle(cards);
    console.log(cards);
    card0.style.visibility = "visible";
    card1.style.visibility = "visible";
    card2.style.visibility = "visible";
    card3.style.visibility = "visible";
}, false);
```

これで、４枚バージョンの神経衰弱はひとまず完成としましょう。

これをベースに、５２枚のフルバージョンへの挑戦もおもしろいかと思います。

制作・著作
パソコン教室 ドリームプラス
2025 年 6 月 7 日　試作版
2025 年 7 月 19 日　第 1 版