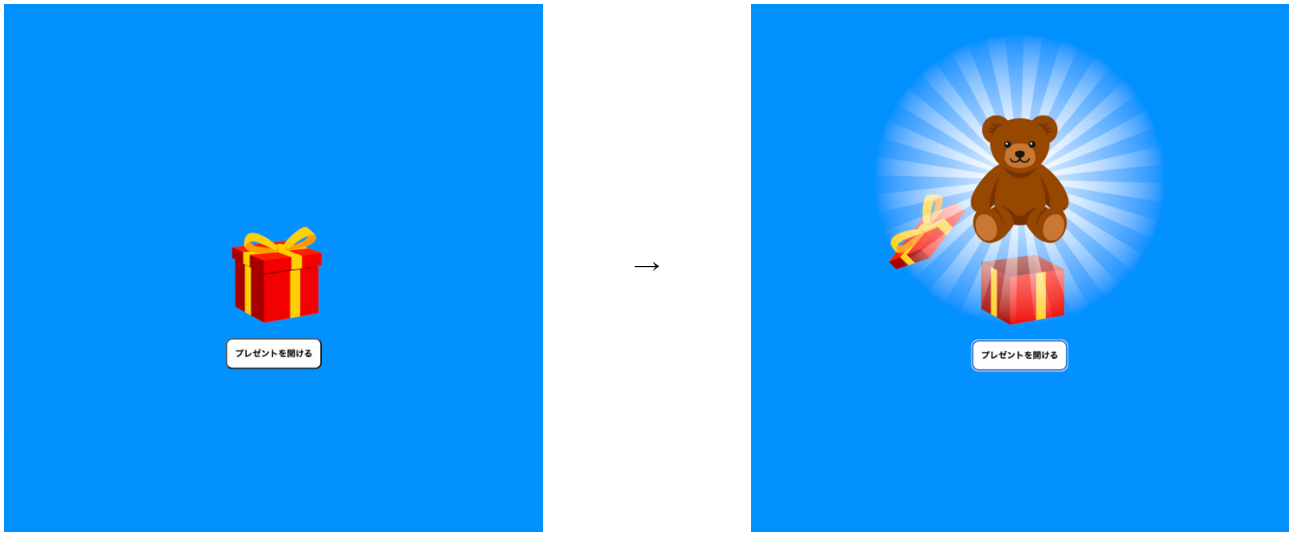
プログラミング基礎演習 / 説明資料 / G1

v1.0

G1 / プレゼントガチャ

DOM 操作を利用してアイテムをランダムに取り出すプレゼントガチャを作成します。
開封前と開封後の演出を作成し、アイテムは配列からランダムに取得・表示します。



準備：

1）素材データをダウンロード => 「 **https://pglec.com/g1/step1.zip** 」

2）pg フォルダに boxgame フォルダを作成して、ダウンロードした素材データをフォルダに入れる

```
boxgame/
 └ step1/
     ├ css/      ・・・ CSS ファイルを格納
     ├ js/       ・・・ JavaScript ファイルを格納
     ├ img/      ・・・ 画像ファイルを格納
     └ sound/    ・・・ 音声ファイルを格納
```

開発手順：

Step1：開封前の画面作成（HTML＋CSS）
Step2：開封後の画面作成１ - 蓋を配置する（HTML＋CSS）
Step3：開封後の画面作成２ - アイテムと光のエフェクトを配置する（HTML＋CSS）
Step4：開封ボタンの設置とクリック判定（JS）
Step5：開封ボタンを押したら中身を表示する（JS）
Step6：開封時にアニメーションさせる（CSS）
Step7：開封時に効果音を鳴らす（JS）
Step8：JavaScript からアイテム画像を出力（JS）
Step9：配列からアイテム画像を出力（JS）
Step10：配列からランダムにアイテム画像を出力（JS）

G1-Step1：開封前の画面表示（HTML＋CSS）

箱と蓋の画像を配置します。箱と蓋の画像が順番に表示されます。

HTML > <body> / 追加

```
  <body>
    <div id="main">
        <!-- 箱 -->
        <img src="img/box.png" id="box" />
        <!-- 蓋 -->
        <img src="img/lid.png" id="lid" />
    </div>
  </body>
```

次に、css/app.css にコンテナを作り、箱と蓋の画像を重ねて配置します。

CSS / 新規

```
body {
  background: #0091ff;
  padding: 20px 0;
  text-align: center;
}
/* メインコンテナ */
#main {
  display: grid;
  place-items: center;
  margin: 0 auto;
  width: 500px;
  height: 500px;
}
/* コンテナ内の子要素を同じ位置に配置 */
#main > * {
  grid-area: 1 / 1;
}
/* 箱*/
#box {
  width: 250px;
  align-self: end;
}
/* 蓋 */
#lid {
  width: 260px;
  align-self: end;
}
```

結果：

解説

CSS のレイアウト手法には、グリッドレイアウト（grid） や絶対配置（absolute） などがあります。 グリッドレイアウトは要素を規則的に整列させるのに適しており、ページ全体の構造をきれいに整えやすいのが特徴です。 一方、絶対配置は要素を座標で自由に配置でき、アニメーションや表現の演出など動的なデザインに向いています。 今回は、ボックスを固定的に並べて配置したい目的のため、グリッドレイアウトを採用します。

G1-Step2：開封後の画面表示 1（HTML＋CSS）

HTML / 変更

```
<img src="img/lid.png" id="lid" class="open" />
```

CSS / 追加

```
#lid.open {
  transform: translate(-150px, -30px) rotate(-45deg);
}
```

結果：

解説

- #lid.open は、「id が#lid かつ class が.open」の場合に適用されるスタイル指定です。
- class=”open” を class=””（未指定）に書き換えると、開封前の表示になります。
- transform は、配置している要素に対して、CSS アニメーション・3D 表現の中心的なプロパティで、 要素の位置・回転・拡大縮小・傾きなどを制御できます。

G1-Step3：開封後の画面表示 2（HTML＋CSS）

開封時に表示する中身、アイテム（クマ）とエフェクト（光）を追加します。

HTML / 追加

```
<!-- 光 -->
<img src="img/light.png" id="light" class="open" />
<!-- アイテム -->
<img src="img/item_kuma.png" id="item" class="open" />
```

CSS / 追加

```
/* 光 */
#light {
  width: 500px;
  opacity: 0;
}
/* 光(開封後) */
#light.open {
  opacity: 1;
}
/* アイテム */
#item {
  width: 200px;
  align-self: start;
  margin-top: 150px;
  opacity: 0;
}
/* アイテム(開封後) */
#item.open {
  opacity: 1;
}
```

結果：

解説

- opacity は、不透明度を指定するプロパティです。
  0 で完全に透明、1 で完全に不透明になります。
  今回は開封前に opacity: 0; として見えなくし、
  開封後に opacity: 1; にして表示させています。
- class=”open” を class=””（未指定）に書き換えると、
  開封前の表示になります。

G1-Step4：開封ボタンの設置とクリック判定

HTML > <body> / 追加

```
<button id="btn">プレゼントを開ける</button>
```

CSS / 追加

```
/* ボタン */
#btn {
  padding: 12px;
  font-weight: bold;
  background-color: #fff;
  border-radius: 10px;
  cursor: pointer;
}
```

JS / 新規

```
// ボタン要素を取得
const btn = document.querySelector("#btn");

// アイテムをゲットする関数
function getItem() {
    console.log("アイテムをゲットしました！");
}
// ボタンがクリックされたら getItem 関数を実行
btn.addEventListener('click', getItem);
```

結果：

「プレゼントを開ける」ボタンを押下したら、「アイテムをゲットしました！」と DevTool のコンソールに表示される。

G1-Step5：開封ボタンを押したら中身を表示する

「プレゼントを開ける」ボタンを押下したら、プレゼントの箱が開くようにします。
class=”open” は、開封後に表示する要素です。
したがって、開封前は非表示（class 未指定）にして、開封後（class=”open”指定）に表示します。

HTML > <body> / 変更

```
<img src="img/box.png" id="box" />
<img src="img/lid.png" id="lid" />
<img src="img/light.png" id="light" />
<img src="img/item_kuma.png" id="item" />
```

JS / 追加

```
// 要素を取得
const box = document.querySelector("#box");
const lid = document.querySelector("#lid");
const light = document.querySelector("#light");
const item = document.querySelector("#item");
const btn = document.querySelector("#btn");

// アイテムをゲットする関数
function getItem() {
    console.log("アイテムをゲットしました！");
    // 開封
    lid.classList.add('open');
    light.classList.add('open');
    item.classList.add('open');
}

// ボタンがクリックされたら getItem 関数を実行
btn.addEventListener('click',getItem);
```

結果：

・開封ボタンを押下すると開封後の画像が表示されます。
・DevTool で開封後に class=”open” が付与されていることを確認してください。

解説：

・***.classList.add('open')は、#***の要素に .open クラスを追加して対応する CSS を有効にします。
削除する方法もあります。 ***.classList.remove('open')は、.open クラスを削除します。

G1-Step6：開封時にアニメーションさせる

現在、蓋・光・アイテムの 3 つは開封ボタン押下後に、即時に表示切替しています。
これらにアニメーションを追加して開封の演出をします。

はじめに

Web 上でアニメーションを実現する方法は、CSS・JavaScript・Canvas などいくつかあります。 CSS アニメーションでは直線的で単純な UI の演出表現に向いています。複雑なアニメーションは JavaScript で実現することができます。今回は、演出の基礎として CSS アニメーションを採用します。

アニメーションの記述方法

CSS 例

```
#test {
  animation: <名前> <再生時間> <動きの加速・減速> <再生までの待機時間> <状態保持指定>;
}
@keyframes <名前>  {
 ・・・ここにアニメーション内容を記述する・・・
}
```

（1）蓋のアニメーション

蓋が 0.6 秒かけて動的に開くアニメーション
ease-out は、最初が速く最後に減速します。forwards は、アニメーション終了時の状態を保持します。

CSS / 変更

```
#lid.open {
  animation: lidOpen 0.6s ease-out forwards;
}

@keyframes lidOpen {
  0% {
    transform: translate(0, 0) rotate(0deg);
  }
  100% {
    transform: translate(-150px, -30px) rotate(-45deg);
  }
}
```

（2）光のアニメーション

・0.5 秒後に 0.3 秒かけて出現するアニメーション

・5 秒で 1 周、ずっと回転するアニメーション

2 つのアニメーションを使っています。その場合、「,」で区切り複数指定します。

CSS / 追加

```css
#light.open {
  animation: lightAppear 0.3s ease-out 0.5s forwards,
             lightRotate 5s linear 0.5s infinite;
}

@keyframes lightAppear {
  0% { opacity: 0; }
  100% { opacity: 1; }
}

@keyframes lightRotate {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}
```

（3）アイテムのアニメーション

・0.5 秒後に開始し、下から 100px の位置から 0px に上昇して出現するアニメーション
また、小さい状態（0.3 倍）から拡大 60%地点で少し大きく（1.1 倍）跳ねる効果を入れ
最終的に通常サイズ（1 倍）で表示しています

CSS / 追加

```css
#item.open {
  animation: itemAppear 0.8s ease-out 0.5s forwards;
}

@keyframes itemAppear {
  0% {
    opacity: 0;
    transform: translateY(100px) scale(0.3);
  }
  60% {
    transform: translateY(0) scale(1.1);
  }
  100% {
    opacity: 1;
```

```
    transform: translateY(0) scale(1);
  }
}
```

注意点：

HTML のレイヤー順序は、HTML で後に書かれた要素ほど上に表示されます（z-index を指定していない場合）しかし、CSS で transform を使ったアニメーションを適用すると、その要素に新しい「スタッキングコンテキスト」が作成され、見た目上のレイヤー順序が変わります。

G1-Step7：音を鳴らす

開封時の演出として 効果音（sound/open.mp3）を鳴らします。
以下の HTML と JavaScript のコードを適切な箇所に追加してください。

HTML / 追加

```
<audio src="sound/open.mp3" id="sound"></audio>
```

JS / 追加

```
// 要素を取得
const sound = document.querySelector("#sound");

// 音の再生
sound.play();
```

G1-Step8：JavaScript からアイテム画像を出力（JS）

現在、アイテムはクマの画像だけが表示される仕組みになっています。
別の画像を表示するために JavaScript から画像を出力する形式に変更します。

HTML / 変更

```
<img src="" id="item" />
```

JS / 追加

```
// 画像の src を設定
item.src = 'img/' + 'item_tokei.png';
```

G1-Step9：配列からアイテム画像を出力（JS）

画像ファイル名をあらかじめ配列で用意して、要素番号を指定することで、その画像を出力します。

JS / 追加&変更

```
// アイテム画像の配列
const items = [
    'item_kuma.png',
    'item_jisho.png',
    'item_origami.png',
    'item_tokei.png',
    'item_hanataba.png'
];

// 画像の src を設定
    item.src = 'img/' + items[1];
```

JavaScript の配列（Array）では、要素の番号（インデックス）は 0 から始まります。
つまり、
・最初の要素は items[0]
・2 番目の要素は items[1]
・3 番目の要素は items[2] …　というように数えます。
したがって、上のコードでは items[1] が 2 番目の画像（item_jisho.png）を指定しています。
「最初の要素は 1 ではなく 0 番目」である点に注意してください。

G1-Step10：配列からランダムにアイテムを出力（JS）

今回のゴールは、プレゼントガチャを作ることです。つまり、アイテムをランダムに表示することが必要です。以前に学習した「Math.random()」などを使い、配列の中からランダムにアイテム画像を表示してみましょう。

G1-Step11：ブラッシュアップ

これまでに作成したプログラムを、より完成度の高いものに仕上げましょう。以下の改善ポイントを参考に、自分で改善点や方法を考え、実際にコードを書いてみてください。

・開封後にリセットできるようにし、もう一度開封できるようにする・
・アイテムにレアリティ（希少度）を設定し、「レアなアイテム」を 20%の確率で出現させる
・効果音が開封直後に鳴ってしまうため、少し遅らせて再生するように調整する
・演出をさらに追加し、より華やかで楽しい見た目にする