小学生から始める

プログラミング教室

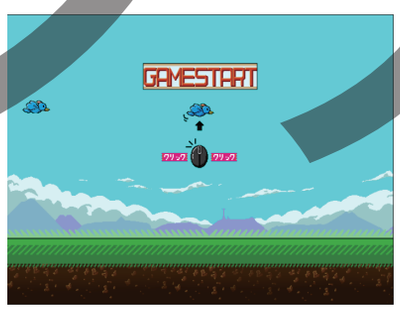
ミドルC

Kid's programmer -master course-

ミドルコース

ゆけ！　ブルーバード（基礎編）

～１枚絵の画像配置を学ぼう～



１回目授業	**年**	**月**	**日**
２回目授業	**年**	**月**	**日**

名前

KID'S プログラミングラボ

ICT活用教育アドバイザー

推奨ブラウザ……Google Chrome 最新バージョン

◆文科省　ICT活用教育アドバイザー事務局掲載
学校ICT化サポート事業者
◆長期コースによる、プログラミングの普及

1日目

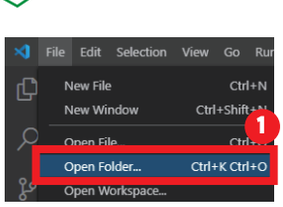
2カ月かけて、トリがジャンプしてパイプを通りぬけるゲームを作成しましょう。
今月は、1枚絵の座標を指定してキャンバスにオブジェクトを表示するしくみを学びます。

1 完成ゲームをプレーしてみよう！

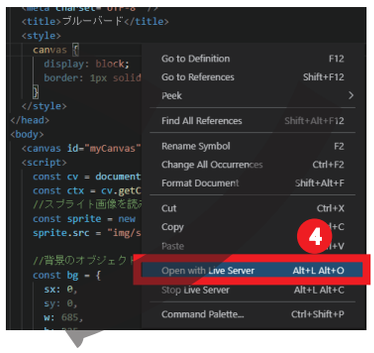
Visual Studio Code の拡張機能「Live Server」を使います。

1 Live Server でプログラムを実行しよう

使用ファイル：ミドルC > ブルーバード.html



1. **File＞Open Folder** を選ぶ
2. 「ミドル C」を選び、
   「フォルダーの選択」をクリック
   （※ミドル C をダブルクリックしない。うまく選べません）

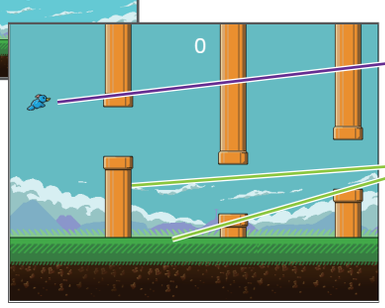


3. ミドル C の中身が表示されるので
   ブルーバード.html を選ぶ
4. コードの上で右クリックし、
   Open with Live Server をクリック
   ※画面右下にこれが出ていたら、ここをクリックしても OK

※「Open with Live Server」が表示されない場合は、
Visual Studio Code を一度閉じ、再度立ち上げます

完成ゲーム （今月は途中まで作ります）

ゲームスタート画面がはじめに表示されています。
画面をクリックしてスタートしましょう。

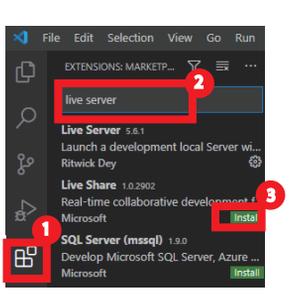


トリには重力がかかっていて、加速しながら下に落ちていきます。
クリックするとジャンプすることができます。

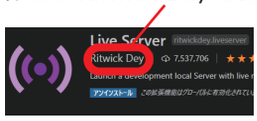
パイプ・地面に**トリ**がぶつかると、ゲームオーバーです。

※Live Server がインストールされていない場合、ネットワークにつなぎ、右の手順で作業してください

Live Server のインストール手順



1. Visual Studio Code を開き、拡張機能をクリック
2. 表示された検索欄に「live server」と入力
3. 一番上に表示されている「Live Server」をインストール
   (作者が Ritwick Dey であることを確認します)



2 ゲームの土台となる骨組みの作成

1 HTML を記述しよう

使用ファイル：ミドル C >1 日目 > 練習 > 練習 1.html

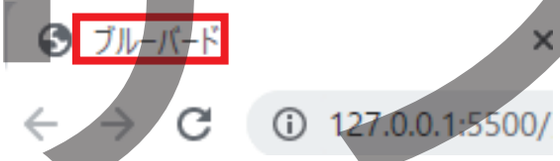
まず「**html:5**」と書きましょう。Visual Studio Code が**自動で html、head、body タグを用意**します。

赤文字部分を記述しよう！（以下同じ）

```
<!DOCTYPE html>                          // この文書を HTML5 で記述することを宣言する
<html lang="ja">                         // 文書の言語を指定する (ja は Japanese の略 )
  <head>
    <meta charset="UTF-8">               // 文字コードには UTF-8 を設定する
                                         （PC によっては最後に / が入る場合がありますが、問題ありません）
    <title> ブルーバード </title>        // ページのタイトルを記述する
  </head>
  <body>

  </body>
</html>
```

※自動で用意されたコードのうち、
左に記載がないものは削除し、
内容が違うものは修正しましょう。

保存（ctrl+s）



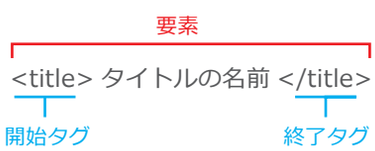
★できているかチェックをしよう！

☑ p.1 で完成ゲームを確認したときと同じように、
Open with Live Server をクリックしましょう（※）。

ウィンドウに、ゲームのタイトルが表示されましたか？
（※）この後のページでも、同じようにチェックします

■ HTML で骨組みを作る

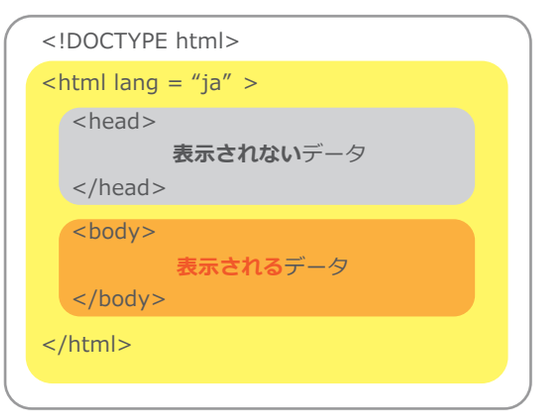
<head></head> や <title></title> などを**タグ**とよびます。
開始タグと終了タグで囲んだ内容を**要素**とよびます。



作成したプログラムは右のように、
「入れ子構造」になっています。

<head> の中には、タイトルや文字コードなど、
Web ブラウザに**表示されない**データを記述します。

<body> の中には、画像や文章など、
Web ブラウザに**表示される**コンテンツを記述します。



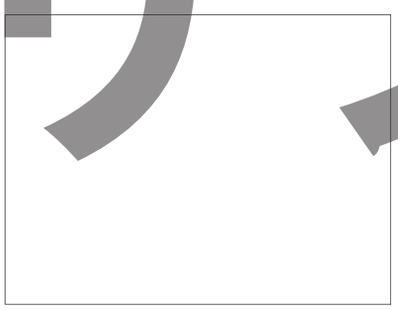
2 キャンバスを作成しよう

使用ファイル：ミドル C>1 日目 > 練習 > 練習 2.html

<canvas> を使用して、キャンバスを作成していきましょう。

```
<head>
   <meta charset="UTF-8" >
   <title> ブルーバード </title>
   <style>
    canvas {
      display: block;          // キャンバス要素の表示形式を「block」にする
                               （難しいので、詳しい説明は省略します）
      border: 1px solid #000;  // キャンバスの枠線を 1px の黒い 1 本線にする
    }
   </style>
</head>
<body>
   <canvas id="myCanvas" width="640" height="480"></canvas>
</body>                        // キャンバスに名前をつけ、横幅と縦幅を指定する
```

保存（ctrl+s）



★できているかチェックをしよう！

☑ 左のようなキャンバスが表示されましたか？

●ためしてみよう！

☑ キャンバスの枠線の太さを変えてみましょう！

■ キャンバスを作り、デザインを加える

●<body> 内の記述

ここに <canvas> を記述すると、キャンバスを作成することができます。
width と height で、横幅と縦幅をピクセル単位で指定します。
id=”myCanvas”は、後で作る JavaScript をこのキャンバスに結びつけるために記述しています。

●<head> 内の記述

ここに <style> を記述すると、CSS でデザインを加えることができます。

CSS の書き方

セレクタ（どこの）
プロパティ（何を）
値（どうする）

```
セレクタ   プロパティ      値
canvas {
          background  : #fff999;   背景色をカラーコードで指定して、黄色にする
          display     : block;     キャンバス要素の表示形式を「block」にして
          margin      : 0 atuo     外余白を「0 auto」にすることで、画面中央に配置する
}
```

文法を覚えよう！

この章で登場したタグで、HTML の基本的な骨組みを作ります。
すべての HTML ファイルに必要な定型文として、覚えましょう。

<!DOCTYPE html> / HTML5であることを宣言

「doctype 宣言」と呼び、文書が HTML5 で作成されていることを宣言しています。

<html lang = “ja”> ～</html> / HTML文書であることを示す

html タグで囲まれた内容が、HTML 文書であることを示します。
開始タグにある lang 属性は、言語 (lang) が日本語 (ja) であることを示します。

<head> ～</head> / 基本的な情報を示す

head 要素内には、ブラウザに表示されない情報を記述します。

<body> ～</body> / コンテンツの中身を記述する

body 要素内には、ブラウザに表示される情報（web サイトのコンテンツ）を記述します。

<meta charset = “UTF-8”> / 文字コードを設定する

※meta は設定用タグのため、終了タグは不要

meta タグは属性とともに利用し、文書に関する情報を表します。
ここでは charset 属性に、国際的に最も利用されている文字コードの「UTF-8」を指定しています。

<style> ～</style> / CSSを記述する

今回は CSS を HTML ファイル内に記述しています。CSS は <head> 内に、<style> タグで囲んで書きます。
（CSS を、HTML とは別のファイルに記述する方法もあります）

CSS の書式

「セレクタ（どこの）」「プロパティ（何を）」「値（どうする）」を指定します。
セレクタの範囲を {}（中かっこ）でくくり、
その中にプロパティと値を：と；で区切って入力します。

例 div 要素の、文字のサイズを 18px に、文字の色を青にする⇒

```
セレクタ
  div {
        プロパティ    値
        font-size : 18px;
              color: blue;
  }
```

<canvas>～</canvas> / 図形や絵を描画できるようにする

<canvas> タグを使うと、HTML と JavaScript で図形やグラフなどを描けるようになります。
この機能を Canvas API と呼び、ゲームやアニメーションを作るときに使われます。

- id 属性
  id で、canvas 要素に名前をつけることができます。
  つけた名前は後で、この要素を JavaScript で参照するときに使います。
- width 属性・height 属性
  width でキャンバスの横幅を、height で縦幅を指定できます。

3 HTML に JavaScript を記述しよう

使用ファイル：ミドル C >1 日目 > 練習 > 練習 3.html

HTML と CSS を使って、ゲームの土台となるキャンバスを作りました。
ここからは、<body> 内に <script> タグを追加して JavaScript を記述していきましょう。

```
<body>
  <canvas id="myCanvas" width="640" height="480"></canvas>
  <script>
    const cv = document.getElementById("myCanvas");  //myCanvas を参照する
    const ctx = cv.getContext("2d");                  // コンテキストオブジェクトを取得する
  </script>
</body>
```

※詳しい説明はこのページの下にあります

保存 （ctrl+s）

変数 const（コンスト）： 値を再代入しないときに使う

ここでは変数 cv・変数 ctx を **const** で宣言しています。**変数に値を再び代入しない**ときに使います。
役割は、値を再び代入して起きるエラーを防ぐことです。

■ <script> タグで JavaScript を記述する

<script> タグで囲んだ部分は、JavaScript のプログラムです。

① getElementById メソッドで、**HTML と関連付ける**

② getContext メソッドで、**描画機能を有効にする**

●getElementById

HTML 内で、id 属性を持つ要素を参照します。
ここでは <canvas> の id 属性” myCanvas” を参照し、変数 cv(canvas の略) に代入しています。

●getContext

コンテキストオブジェクト（Canvas2D コンテキスト）を取得し、
変数 ctx(context の略) に代入します。

描画機能メソッドは、さまざまなものがあります。
今回のプログラムでは、「ctx.drawImage」を使って画像をキャンバスに表示させます
（後ろのページで作ります）。

Canvas を用いてゲームを作るときに毎回必要な手順なので、一連の流れを覚えましょう。

4 3つの関数を記述しよう

ミドル C >1 日目 > 練習 > 練習 4.html 使用ファイル

ゲーム実行の土台として動き続ける、３つの関数を用意します。

```
const cv = document.getElementById("myCanvas");
const ctx = cv.getContext("2d");
// オブジェクトを描画する関数
function drawAll() {                                    ←②
                                （下に説明があります。以下同じ）
        ctx.fillStyle = "aqua";                         // 色をアクア色にする
        ctx.fillRect(0, 0, cv.width, cv.height);        // キャンバスサイズの四角形を描く
}
// オブジェクトを更新する関数
function update() {}                                    ←③
// ゲームをループし続ける関数
function loop() {
        drawAll();
        update();
        requestAnimationFrame(loop);
}
loop();                                                 ←①
```

保存（ctrl+s）

★できているかチェックをしよう！

☑ 左のようなキャンバスが表示されましたか？

●ためしてみよう！

☑ キャンバスの色を変えてみましょう！
（例：” yellow”, “lightgreen”）

■ ゲーム実行の土台となる、３つの関数

① loop()……requestAnimationFrame() により、一度実行するとループし続ける関数。
drawAll() と update() をここに入れることで、ゲーム実行中ずっと関数が動き続ける。

② drawAll()……図形を描写する関数を集めた関数。背景に入る空（そら）の絵をここで設定します。

③ update()……オブジェクトの動きを表現する関数。
関数の内容は、後ろのページで定義します。

文法を覚えよう！

新しい文法が出てきました。役割をおさえておきましょう。

<script>～</script> / JavaScriptを記述できるようにする

JavaScript プログラムを HTML 文書内に埋め込むために使うタグです。

const　変数名 / 再代入できない変数を設定する

値を再代入できない変数を宣言するキーワードです。**const** で宣言した変数は、後から値を代入したり変更することはできません。　※変数に値を再代入するときは、**let** キーワードを使用します。

```
const age = 12;
age = 10; //=> エラー
```

```
let age = 12;
age = 10;
console.log(age); //=> 10
```

getElementById(); / HTMLとJavaScriptをつなげる

HTML 内の、指定した id を持つ要素を取得するメソッド。HTML と JavaScript を関連付けます。

getContext(); / 描画機能を有効にする

キャンバスで、描画機能を有効にするメソッドです。

function　関数名(引数){} / 関数を定義する

Python では関数の定義に def を使いますが、JavaScript では function を使います。

```
function add(){
        console.log( 2 + 3 );
}
add(); //=> 5
```

requestAnimationFrame(関数名); / 一定の間隔で処理を繰り返す

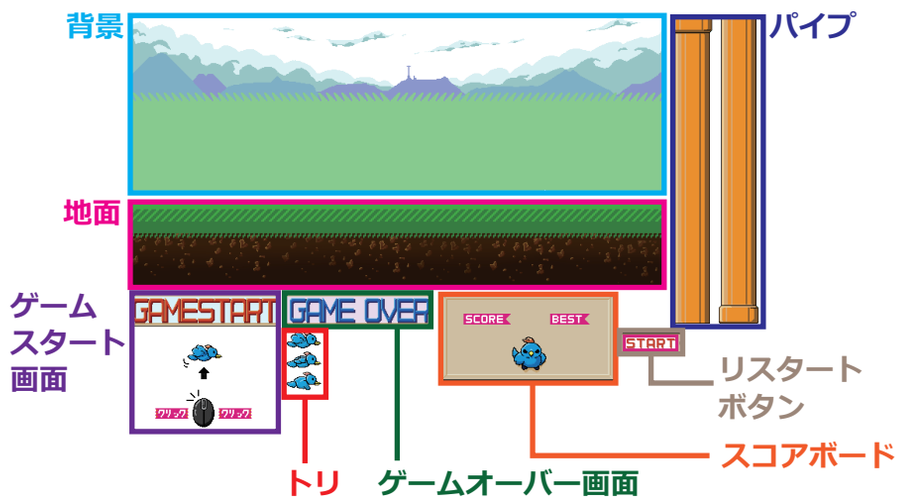
requestAnimationFrame() を使用すると、およそ 60fps（※）の間隔で繰り返し処理を実行できます。
これにより、なめらかなゲームアニメーションをキャンバスに描くことができます。
主に、コールバック関数（定義した関数自身）を引数にとって使用します。

（※fps……1 秒に見せる画像のコマ数）

```
function loop(){
        console.log( ” こんにちは” );
        requestAnimationFrame(loop);
}
loop();
```

3 背景オブジェクト

使う画像が img フォルダ内にあります。「sprite.png」を開いて見てみましょう。
大きな 1 枚絵の中に、画像がいろいろ入っています。どの部分を使うか指定して、表示します。



1 枚絵から表示する仕組み「スプライト」

どんなもの？
必要な画像を 1 枚絵にまとめ、
使う部分を指定して、
そこだけ表示する仕組みです

この仕組みにする理由は？
1 枚絵にまとめることで、
画像の読み込みが速くなるという
メリットがあります

1 スプライト画像を読み込もう

使用ファイル：ミドル C >1 日目 > 練習 > 練習 5.html

1 枚絵を、まず変数として読み込みます。
作成した変数にはパス（画像がある場所の情報）を追加します。

```
const cv = document.getElementById("myCanvas");
const ctx = cv.getContext("2d");
// スプライト画像を読み込む
const sprite = new Image();          // 画像のオブジェクトを作成する
sprite.src = "img/sprite.png";       // 画像にパスを追加する
```

保存（ctrl+s）

■ 画像を読み込む方法

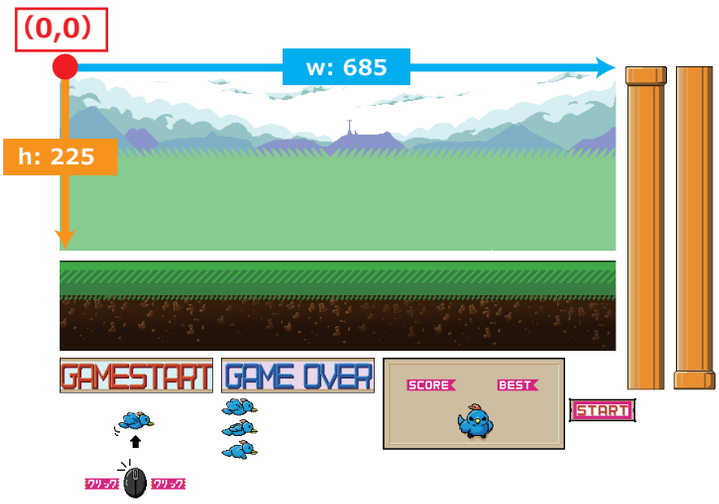
次の 2 つの手順を使います。

- **const sprite = new Image();**
  画像用のオブジェクトを作り出し、変数 sprite に代入します。
- **sprite.src = “img/sprite.png”**
  作成した sprite オブジェクトにパスを通します。
  パスとはフォルダ上にある画像の位置です。

2 背景のオブジェクトを作ろう

使用ファイル：ミドル C >1 日目 > 練習 > 練習 6.html

背景のスプライトは、1 枚絵の中の左上にあります。
座標は以下のようになります。




```
const bg = {
      sx: 0,
      sy: 0,
      w: 685,
      h: 225
};
```

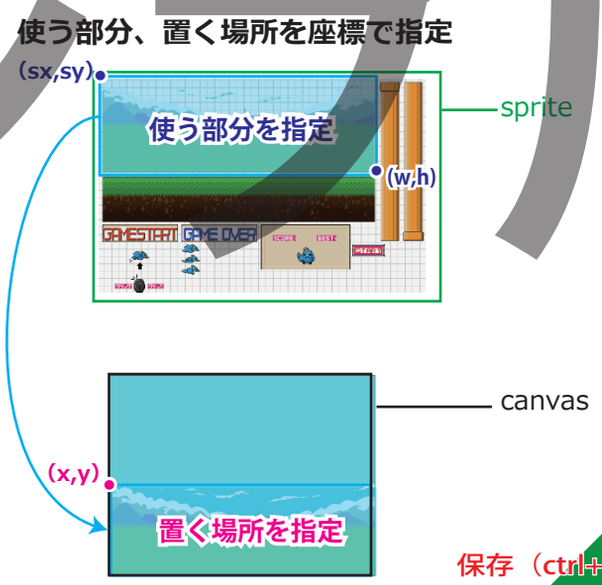
※引数の単位はピクセルです。

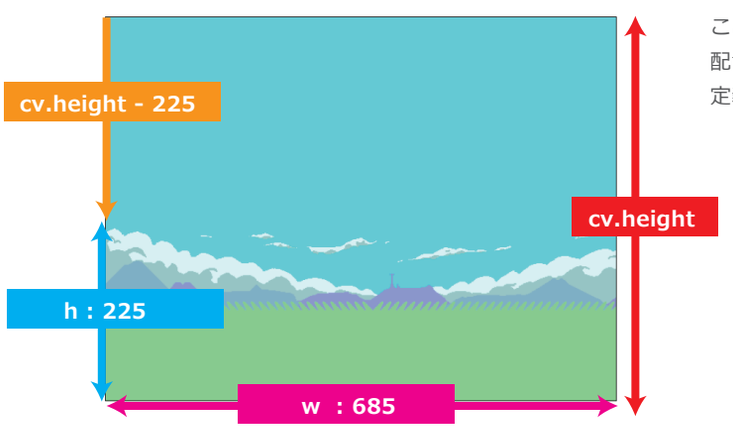
上の図のとおりに背景のオブジェクトを記述していきます。オブジェクトは {} で定義します。

```
const sprite = new Image();
sprite.src = "img/sprite.png";
// 背景のオブジェクト
const bg = {
      sx: 0,
      sy: 0,
      w: 685,
      h: 225,
      x: 0,
      y: cv.height - 225,
};
```

使う部分を指定

置く場所を指定





ここではまだ画像を配置しませんが、
配置するときに必要な座標を定義しています。
定義した座標は左の通りです。

3 背景を配置しよう

使用ファイル：ミドル C>1 日目 > 練習 > 練習 7.html

drawImage() を使用して画像を配置しましょう。
drawImage() の使い方については次のページで学びます。

```
// 背景のオブジェクト
const bg = {
    sx: 0,
    sy: 0,
    w: 685,
    h: 225,
    x: 0,
    y: cv.height - 225,
    draw: function () {
        ctx.drawImage(
            sprite,
            this.sx, this.sy, this.w, this.h,
            this.x, this.y, this.w, this.h
        );
    },
};
// オブジェクトを描画する関数
function drawAll() {
        ctx.fillStyle = "aqua";
        ctx.fillRect(0, 0, cv.width, cv.height);
        bg.draw();
}
```

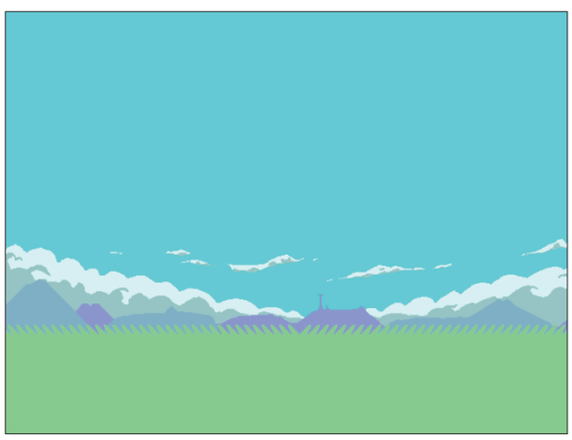
画像を配置する

//draw メソッドを定義
//drawImage メソッド

① // スプライト画像を選択し、（次のページに説明があります）
② // 使用する部分を決める
③ // その画像をキャンバス上に配置する
bg オブジェクトを指す

//draw メソッドの実行

保存（ctrl+s）



★できているかチェックをしよう！

☐ 背景がキャンバスに表示されましたか？

■ drawImage() で画像を表示する

● **ctx.drawImage(){sprite, sx, sy, w, h, x, y, w, h}**

drawImage メソッドで、1 枚絵の中にある画像をキャンバスに表示するステップです。

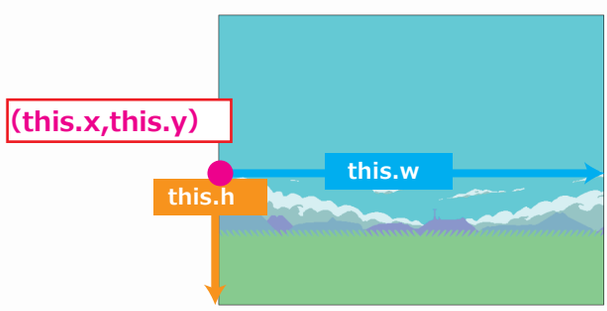
```
const bg = {
        省略
        ctx.drawImage(
            1 枚絵
            sprite,                                          ←①
            1 枚絵の、指定の部分を使う
            this.sx, this.sy, this.w, this.h,                ←②
                     基点      横幅    高さ
            キャンバスの、指定の場所に置く
            this.x, this.y, this.w, this.h                   ←③

        );
```

① 変数 **sprite** を選択しています。
p.8 で定義したとおり、1 枚絵の画像を読み込んだものです。

② 引数 (**this.sx, this.sy**) は使用部分の基点座標、引数 (**this.w, this.h**) はその横幅と高さです。
この 4 つの座標を使い、1 枚絵から使用する部分を指定します。

③ **this** は日本語で「この」を意味します。ここでの this は **bg オブジェクト**自身を表します。
this.sx では、bg オブジェクトの sx の値である 0 が参照されます。

引数 (**this.x, this.y**) は画像をキャンバスに配置する位置の、基点座標です。
そこから（**this.w,this.h**）で画像の横幅と高さを指定して、配置します。



文法を覚えよう！

drawImage() を使用した画像の描画の仕方を復習しましょう。
同じ仕組みで、これからほかのオブジェクトを作っていきます

new Image(); / 画像のオブジェクトを生成する

new Image() を使用すると画像要素を生成することができます。
そこにパスのソース（ファイルの場所）を設定することで、画像データを読み込むことができるようになります。

この 2 つの動作を実行したのち、drawImage() を使用して画像を配置します。流れをセットで覚えましょう。

```
const img = new Image();
img.src = 'myImage.png';
ctx.drawImage(img, 0, 0);
```

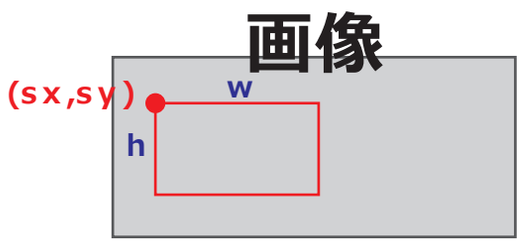
変数名 = {}; / オブジェクトを作成する

JavaScript でオブジェクトを作成するには、{}(中かっこ) を使用します。
オブジェクトのキーと値を：で区切ったものを {} の中に書きます。
作成したオブジェクトは、変数名とキーをドット (.) でつないで参照することができます。


```
human オブジェクト
const human = {     キー     値
                  name: '太郎' ,
                  age  :   32,
                  gender: '男'
                  };
console.log(human.name);  //=> 太郎
console.log(human.age) ; //=> 32
```

ctx.drawImage(image, sx, sy, w, h, x, y, w, h) / 画像を描く

画像 image から取り出したい部分の開始値を、座標 (sx, sy) に記述します。



指定した部分の画像を、キャンバスの座標 (x,y) の場所に配置します。

4 地面オブジェクト

地面のスプライトは、1 枚絵の中の真ん中にあります。
座標を確認しましょう。


```
const bg = {
        sx: 0,
        sy: 240,
        w: 685,
        h: 110,
};
```

1 地面のオブジェクトを作成しよう

ミドル C >1 日目 > 練習 > 練習 8.html 使用ファイル

上の図のとおりに地面のオブジェクトを記述していきます。

```
// 背景のオブジェクト
省略
// 地面のオブジェクト
const ground = {
        sx: 0,
        sy: 240,
        w: 685,
        h: 110,
        x: 0,
        y: cv.height - 110,
        dx: 2,          (dx は来月のプログラムで使います)
};
```

保存（ctrl+s）

2 地面を配置しよう

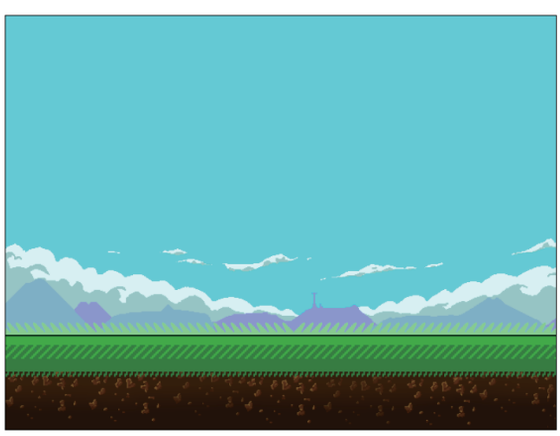
ミドル C>1 日目 > 練習 > 練習 9.html 使用ファイル

背景を配置したときと同様に、draw メソッドを地面オブジェクトに定義します。

```
// 地面のオブジェクト
const ground = {
        sx: 0,
        sy: 240,
        w: 685,
        h: 110,
        x: 0,
        y: cv.height - 110,
        dx: 2,
        draw: function () {
          ctx.drawImage(
            sprite,
            this.sx, this.sy, this.w, this.h,
            this.x, this.y, this.w, this.h
          );
        },
};
// オブジェクトを描画する関数
function drawAll() {
        ctx.fillStyle = "aqua";
        ctx.fillRect(0, 0, cv.width, cv.height);
        bg.draw();
        ground.draw();
}
```

// スプライト画像を選択し、
// 使用する部分を決める
// その画像をキャンバス上に配置する

保存（ctrl+s）

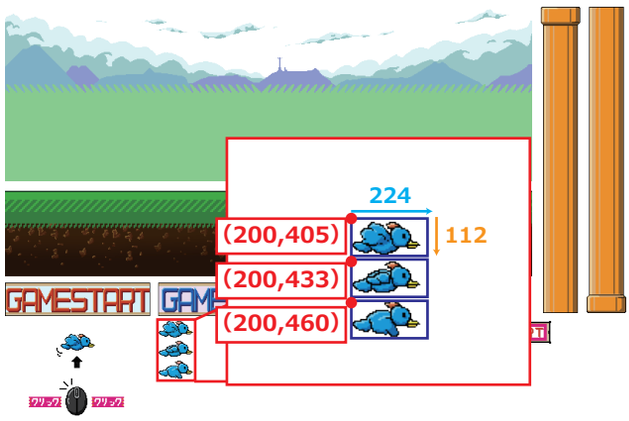


★できているかチェックをしよう！

☐ 地面がキャンバスに表示されましたか？

5 トリのオブジェクト

トリの画像は、３つあります。これらを切り替えて、飛んでいるように見せます。
画像３つがそれぞれ座標を持つので、animation 配列に３つのオブジェクトを作ります。




```
const bg = {
      animation: [
        { sx: 200, sy: 405 },
        { sx: 200, sy: 433 },
        { sx: 200, sy: 460 },
      ],
      w: 224,
      h: 112,
};
```

1 トリのオブジェクトを作ろう

ミドル C>1 日目 > 練習 > 練習 10.html 使用ファイル

３つの画像を用意するため、animation という配列（リスト）にそれぞれの画像座標を作成します。


```
// トリのオブジェクト
const bird = {
      animation: [
        { sx: 200, sy: 405 },
        { sx: 200, sy: 433 },
        { sx: 200, sy: 460 },
      ],
      w: 50,
      h: 26,
      x: 50,
      y: 150,
      frame: 0,
};
```

animation 配列を作成

最初のフレーム

保存（ctrl+s）

■ フレームを使用して画像切り替え

●frame

３つの画像を切り替えるには、frame を使用します。

frame が 0 のとき、**animation 配列** 0 番目 (animation[0]) の 
frame が 1 のとき、 
frame が 2 のとき、  が表示されるように指定します。

2 トリの画像を配置しよう

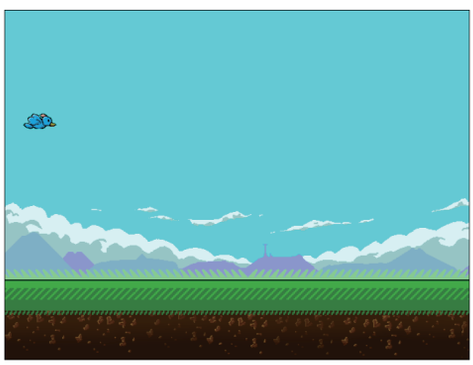
使用ファイル：ミドル C>1 日目 > 練習 > 練習 11.html

トリを配置するときは、３つのうちどれにするかを決めます。
そのために、draw メソッド内に bird 変数を作ります。

```
// トリのオブジェクト
const bird = {
  ═══省略═══
  y: 150,
  frame: 0,
  draw: function () {
    let bird = this.animation[this.frame];          // どのスプライトにするか frame の値で選ぶ
    ctx.drawImage(
      sprite,                                       // スプライト画像を選択し、
      bird.sx, bird.sy, this.w, this.h,             // 使用する部分を決める
      this.x - this.w/2, this.y - this.h/2, this.w, this.h   // その画像をキャンバス上に配置する
    );
  },
};
// オブジェクトを描画する関数
function drawAll() {
  ═══省略═══
    ground.draw();
    bird.draw();
}
```

画像を適切に配置する指定をしています。
詳しくは来月学習します。

保存（ctrl+s）



★できているかチェックをしよう！

☑ キャンバスに、トリが表示されましたか？

●ためしてみよう！

☑ frame の値を 1 や 2 に変更してみよう！
（トリの絵が代わるのを確認できたら 0 に戻しましょう）

■ 配列（リスト）の値を参照する流れ

●this.animation[this.frame]

frame の値は 0 ～ 2 です。0 のときは this.animation[0] となります。
animation 配列の 0 番目は { sx: 200 sy: 405 } なので、bird = { sx: 200, sy: 405 } になります。

続く drawImage 変数内で、bird.sx と bird.sy はこの bird 変数の値を参照します。

2日目

ゲームスタートとゲームオーバーに使うオブジェクトを作ります。
また、トリに重力がかかって下に落ちるようにします。

1 ゲームスタート画面


```
const gameStart = {
        sx: 0,
        sy: 360,
        w: 190,
        h: 180,
};
```


1 ゲームスタート画面を作成しよう

使用ファイル：ミドル C >2 日目 > 練習 > 練習 1.html

上の値の通りに、ゲームスタート画面のオブジェクトを記述していきます。

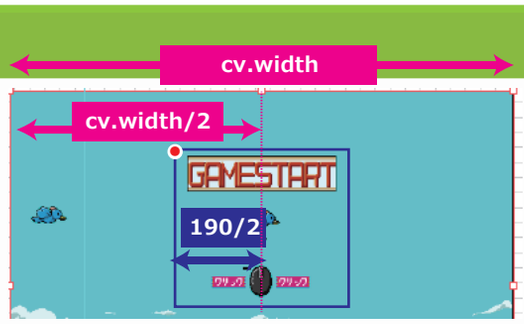
```
// トリのオブジェクト
省略
// ゲームスタート画面
const gameStart = {
        sx: 0,
        sy: 360,
        w: 190,
        h: 180,
        x: cv.width / 2 - 190 / 2,
        y: 80,
};
```

保存（ctrl+s）

■ キャンバスの左右中央に表示する座標指定

- x: cv.width / 2 - 190 / 2

cv.width / 2……キャンバス横幅の半分
190 / 2…ゲームスタート画像の横幅の半分



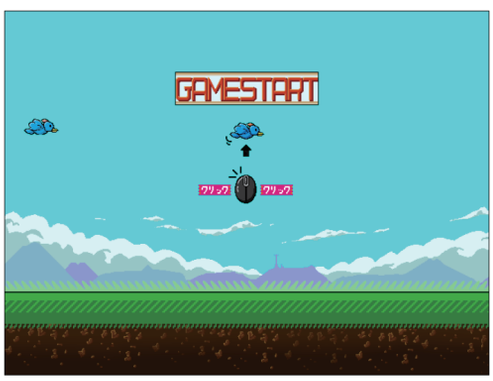
2 ゲームスタート画面を配置しよう

使用ファイル：ミドル C >2 日目 > 練習 > 練習 2.html

draw メソッドでゲームスタート画面を配置していきましょう。

```
// ゲームスタート画面
const gameStart = {
      省略
      y: 80,
      draw: function () {
        ctx.drawImage(
          sprite,                                    // スプライト画像を選択し、
          this.sx, this.sy, this.w, this.h,          // 使用する部分を決める
          this.x, this.y, this.w, this.h             // その画像をキャンバス上に配置する
        );
      },
  };

// オブジェクトを描画する関数
function drawAll() {
      省略
      bird.draw();
      gameStart.draw();
    }
```

保存（ctrl+s）



★できているかチェックをしよう！

☐ キャンバスに、ゲームスタート画面が表示されましたか？

2 ゲームオーバー画面

1 ゲームオーバー画面を作成する

使用ファイル：ミドル C >2 日目 > 練習 > 練習 3.html

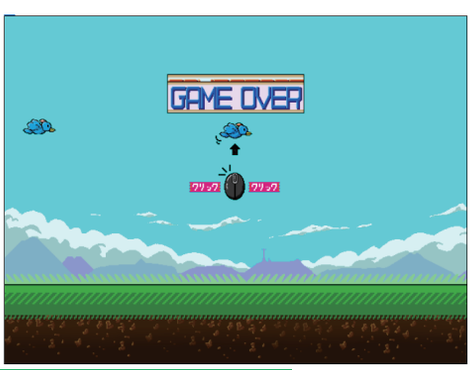
ゲームオーバー画面のプログラムは、ゲームスタート画面とほとんど同じです。
コピー・貼り付けで複製しましょう。そのあと、数値を下のように変更します。

```
// ゲームスタート画面
        省略
// ゲームオーバー画面
const gameOver = {
        sx: 200,
        sy: 360,
        w: 190,
        h: 45,
        x: cv.width / 2 - 190 / 2,
        y: 90,
        draw: function () {
            ctx.drawImage(
              sprite,
              this.sx, this.sy, this.w, this.h,
              this.x, this.y, this.w, this.h
            );
        },
};
// オブジェクトを描画する関数
function drawAll() {
        省略
        bird.draw();
        gameStart.draw();
        gameOver.draw();
}
```



```
// スプライト画像を選択し、
// そのスプライト画像の使用する範囲を決める
// その画像をキャンバス上に配置する
```

保存（ctrl+s）

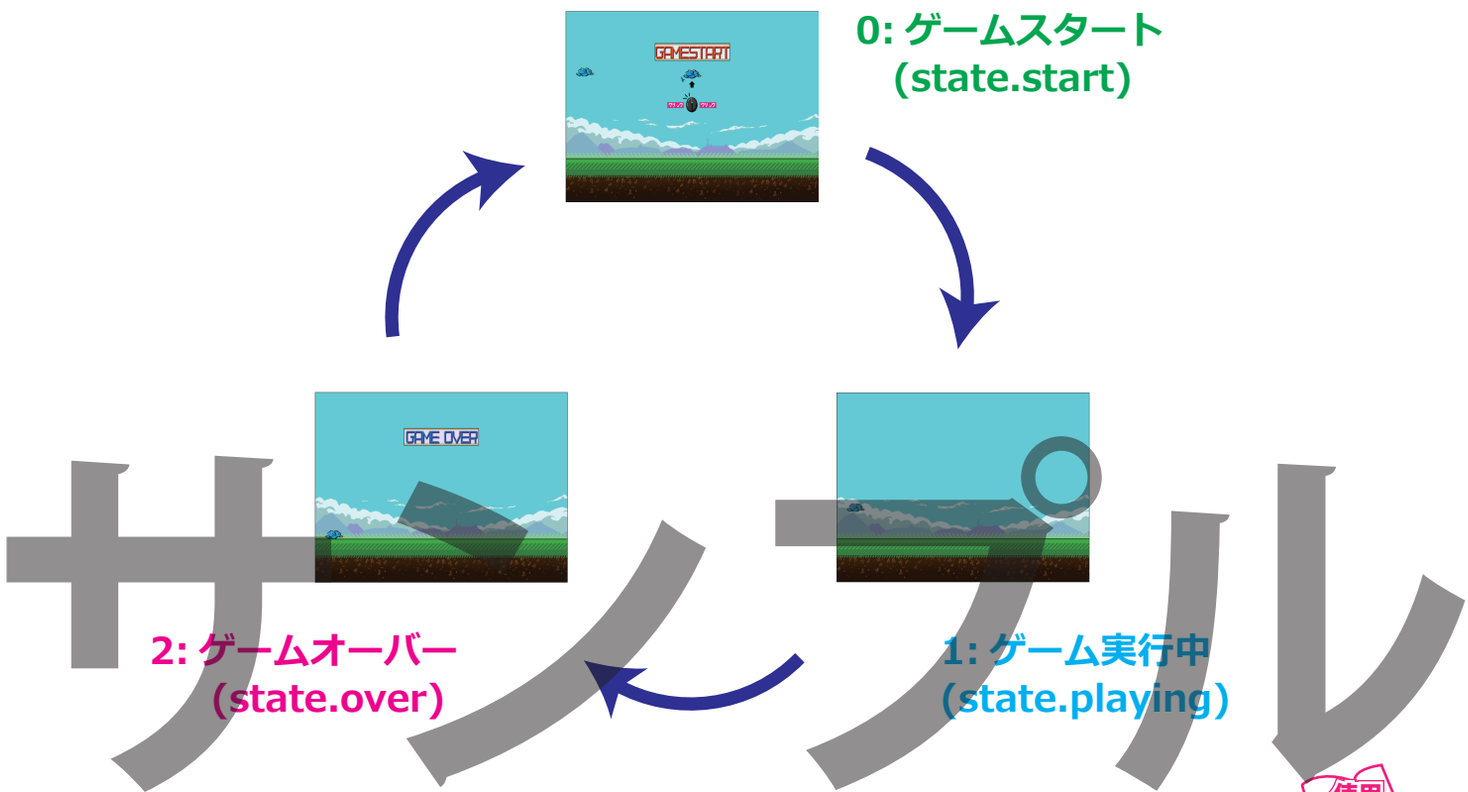


★できているかチェックをしよう！

☐ キャンバスに、ゲームオーバー画面が表示されましたか？

3 ゲーム状態の管理

今回のゲームには、ゲームスタート、ゲーム実行中、ゲームオーバーの３つの状態があります。
画面をクリックすると、状態が切り替わるようにします。



1 変数を設定しよう

使用ファイル：ミドル C >2 日目 > 練習 > 練習 4.html

ゲームの状態を表す state オブジェクトを作成します。
３つの状態を表す変数に加え、現在の状態を表す変数 now も用意します。

```
// スプライト画像を読み込む
const sprite = new Image();
sprite.src = "img/sprite.png";
// ゲームの状態
const state = {
        now: 0,                    // 現在の状態
        start: 0,                  // ゲームスタート
        playing: 1,                // ゲーム実行中
        over: 2,                   // ゲームオーバー
};
```

保存（ctrl+s）

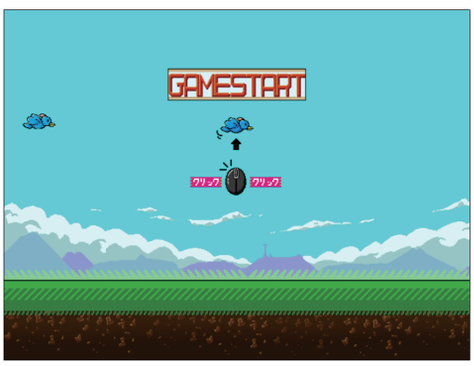
2 ゲーム終了用の関数の定義

使用ファイル：ミドル C >2 日目 > 練習 > 練習 5.html

if 文を使用して、もしゲームの状態がゲームスタートならゲームスタート画面を、
ゲームオーバーならゲームオーバー画面を出すようにしましょう。

```
// ゲームスタート画面
const gameStart = {
    省略
    draw: function () {
      if (state.now == state.start) {          // もし現在の状態がゲームスタートなら
        ctx.drawImage(
            sprite,
            this.sx, this.sy, this.w, this.h,
            this.x, this.y, this.w, this.h
        );
      }
    },
  };
// ゲームオーバー画面
const gameOver = {
    省略
    draw: function () {
      if (state.now == state.over) {           // もし現在の状態がゲームオーバーなら
        ctx.drawImage(
          sprite,
          this.sx, this.sy, this.w, this.h,
          this.x, this.y, this.w, this.h
        );
      }
    },
  };
```

保存（ctrl+s）



★できているかチェックをしよう！

☐ ゲームスタートの画面になりましたか？

●ためしてみよう！

☐ now の値を 1 や 2 に変更してみよう！
（画像が代わるのを確認できたら 0 に戻しましょう）

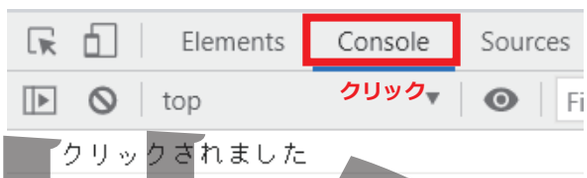
3 クリックしたときの関数を作ろう

使用ファイル：ミドル C >2 日目 > 練習 > 練習 6.html

addEventListener() を使い、キャンバスをクリックすると関数が実行されるようにしましょう。

```
// スプライト画像を読み込む
const sprite = new Image();
sprite.src = "img/sprite.png";
// クリックしたときの関数
cv.addEventListener("click", function (event) {   // キャンバス (cv) をクリックすると
    console.log(" クリックされました " );          // コンソールに文字を表示する
});
```

保存（ctrl+s）



★できているかチェックをしよう！

☑ p.1 で完成ゲームを確認したときと同じように、
Open with Live Server をクリックしましょう。

「**Ctrl + Shift + I**」を押して、コンソールを確認します。
コンソールが開けたらキャンバス上をクリックしましょう。
左の画面のように、
「クリックされました」と表示されましたか？

■クリックでプログラムが実行される仕組み

● **cv.addEventListener(" click" ,function(event){ 処理する内容 })**

addEventListener() は、さまざまなイベント処理を実行するメソッドです。

「イベント処理」とは、なにかしらのイベントが起きたときに実行する処理です。

例
- マウスでクリックされたとき（" click"）
- マウスがオブジェクトに乗ったとき (" mouseover")
- マウスでカーソルを移動させたとき（" mousemove"）

実行する処理には一般的に、関数 function(event){} を使います。
引数 event はイベントオブジェクトと呼ばれ、イベント発生時の情報をもつオブジェクトです。
今月のプログラムでは使用しませんが、来月はこれを使います。

●ためしてみよう！

☑ " click" の値を" mousemove" に変更して実行してみよう！
（確認ができたら" click" に戻しておきましょう）

4 switch 文で条件分岐しよう

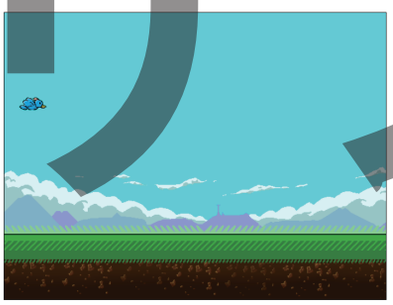
使用ファイル：ミドル C >2 日目 > 練習 > 練習 7.html

キャンバスをクリックしたときにゲームの状態に合わせて実行するプログラムが変わるようにします。

```
// クリックしたときの関数
cv.addEventListener("click", function (event) {
    console.log(” クリックされました” );
    switch (state.now) {
        case state.start:
            state.now = state.playing;
            break;
        case state.playing:
            break;
        case state.over:
            state.now = state.start;
            break;
    }
});
```

```
//state.now を対象の値にします
// もし state.now と state.start が同じなら
  //state.now を state.playing にする
  //switch 文から抜ける
// もし state.now と state.playing が同じなら
  //switch 文から抜ける
// もし state.now と state.over が同じなら
  //state.now を state.start にする
  //switch 文から抜ける
```

保存（ctrl+s）



★できているかチェックをしよう！

☐ クリックすると「ゲームスタート画面」が消えるようになりましたか？

■switch 文で処理を分ける

● **switch(対象値){case 値 1 : 処理 , case 値 2 : 処理 , case 値 3: 処理・・・}**

switch 文は、対象値がほかの値と一致するか調べ、その結果で処理を分けます。
ここでは、**state.now（現在の状態）**を対象値としています。

(1) 最初にある「**state.now** == **state.start（ゲームスタート）**」が成り立つとき、
最初の case 文以下の処理が実行されます。

(2) 処理が実行されると **state.now** の値が **state.playing(ゲーム実行中)** となり、
その後 break 文が実行され、switch 文から抜け出します。

＼case 文の処理の最後に break 文を入れる／

break 文がないと、次の case 文が比較され、
switch 文が終了するか break がどこかに出てくるまで、プログラムが実行され続けてしまいます。

文法を覚えよう！

この章ではイベント処理や switch 文など大切な文法を学習しました。

addEventListener(”イベント名”, 関数名(引数)); / イベントに合わせて関数を実行する

addEventListener は、特定のイベントに合わせて関数を実行します。

```
function sayHello(){
    alert("Hello!");   //=>「Hello!」のポップアップを表示する
}
const button = document.getElementById('button');
button.addEventListener('click', sayHello);    // ボタンをクリックすると sayHello() を実行する
```

console.log(引数); / コンソールに表示する

console.log(引数) を使用すると、引数の値をコンソールに表示することができます。

コンソール画面は、ブラウザを開いて「Ctrl + Shift + I」を押すと出てきます。

「この値は何だろう？」と思ったら、コンソールに表示して確認しましょう。よく使う機能です。

```
const total =  63 + 63;
console.log(total);        //=>  126 をコンソールに表示する
```

switch文 / 条件にあった式を実行する

switch 文は、対象となる値が、いずれかの値と一致するかどうかを調べ、処理を分けることができます。

```
let age = 10;
switch (age) {                          //age を対象の値にする
  case 10:                              //age が 10 の場合
    console.log(” 10 才です” );          //「10 才です」と表示する
    break;                              //switch 文から抜ける
 case 9:                                //age が 9 の場合
    console.log(” 9 才です” );           //「9 才です」と表示する
    break;                              //switch 文から抜ける
 default:                               // それ以外の場合
    console.log(” ひみつです” )          //「ひみつです」と表示する
}
```

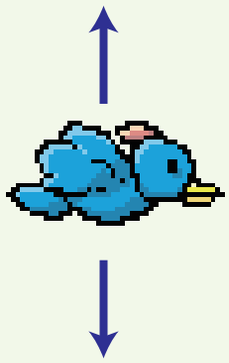
上記の文は age が 10 なため、case 10 のプログラムが実行されコンソールに「10 才です」と表示されます。break を書かない場合は次の case 文の条件式が実行されるようになるので、ひとつひとつの case 文の最後には break; を書きましょう。どの値にもあてはまらないときは default: に指定した処理が実行されるようになります。

同様に条件分岐ができる文法には if 文もありますが、実行される処理の違いはありませんので、書きやすいほうでプログラムを記述しましょう。

4 トリの動きを追加

トリに新しい情報を加えて、動きを追加します。

３つの値で、トリが加速しながら下に落ちる動きと
画面をクリックするとジャンプする動きを表現します。



- **gravity**
  トリにかかる重力の値。
- **jump**
  クリックしたときのジャンプ力を示す値
- **speed**
  トリの移動スピードを記録する値

1 変数を追加しよう

使用ファイル：ミドル C >2 日目 > 練習 > 練習 8.html

トリのオブジェクト、トリにかかる重力、クリックしたときのトリのジャンプ力、
トリが移動するスピードの３つを追加します。

```
// トリのオブジェクト
const bird = {
        省略
        y: 150,
        frame: 0,
        gravity: 0.25,
        jump: 4.5,
        speed: 0,
        draw: function () {
```

保存（ctrl+s）

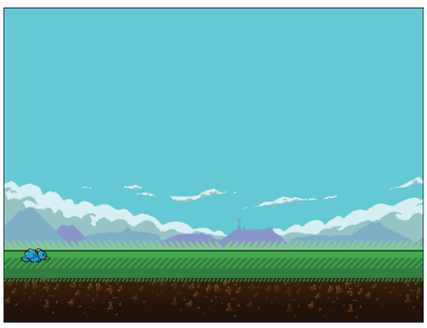
2 トリに重力がかかるようにしよう

使用ファイル：ミドル C>2 日目 > 練習 > 練習 9.html

トリに新たに update メソッドを追加します。
ゲームの状態に応じたトリの動きの変化を、アップデートし続けます。

```
// トリのオブジェクト
const bird = {
  ═══省略═══
    jump: 4.5,
    speed: 0,
    update: function () {
      if (state.now == state.start) {      // もし現在の状態がゲームスタートの場合 ←① 
        this.y = 150;                      // トリの y 座標を 150 にする
      } else {                             // それ以外の場合 ←②
        this.speed += this.gravity;        // トリのスピードに重力分加算する
        this.y += this.speed;              // トリの座標をスピード分移動させる
      }
    },
  ═══省略═══
// オブジェクトを更新する関数
function update() {
    bird.update();
}
```

保存（ctrl+s）



★できているかチェックをしよう！

☑ 画面をクリックしてゲームをスタートします。
トリが下に落ちるようになりましたか？
（画面から出ていきます）

●ためしてみよう！

☑ gravity の値を変更して実行してみよう！

■ ゲームの状態に応じてトリの動きを変える

●if(条件){ 処理１ }else{ 処理２ }

① もしゲームの状態がゲームスタートなら、トリの y 座標を 150 にして始めの位置に置きます。

② それ以外（ゲーム実行中／ゲームオーバー）なら、y 座標を大きくして落ちてくようにします。

【加速して落ちる仕組み】

update メソッドはゲーム実行中ずっと高速で動き続けています。
動くたびに this.speed に gravity が追加され、それがトリの y 座標に追加されるので、
落ちるスピードが加速していきます。

3 ゲームオーバーの追加

使用ファイル：ミドル C>2 日目 > 練習 > 練習 10.html

トリが地面に着いたらゲームオーバーになるようにします。

```
// トリのオブジェクト
const bird = {
    省略
    update: function () {
        if (state.now == state.start) {
            this.y = 150;
        } else {
            this.speed += this.gravity;
            this.y += this.speed;
            if (this.y + this.h / 2 >= cv.height - ground.h) {  // もしトリが地面についたとき
                this.y = cv.height - ground.h - this.h / 2;     // トリの座標を地面と同じにする
                if (state.now == state.playing) {               // もしゲーム実行中なら
                    state.now = state.over;                     // ゲームオーバーにする
                }
            }
        }
    },
```

①

②

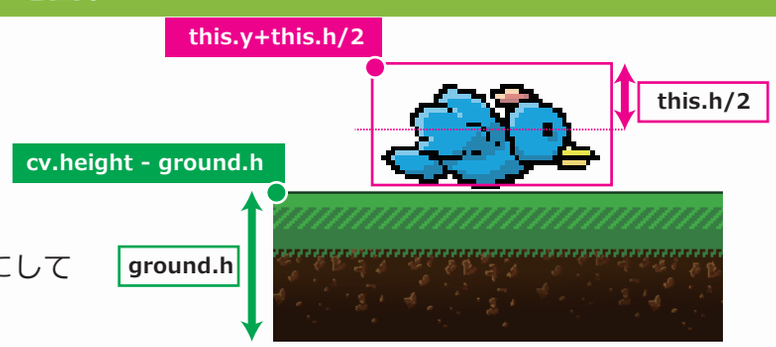
★できているかチェックをしよう！

- ☐ トリが地面の位置で止まりましたか？
- ☐ トリが地面に着いたら
  ゲームオーバーの表示が出ましたか？

■ トリを止め、ゲームオーバーの表示を出す

① トリの y 座標が地面に着いたら、
その高さにトリを置きます。

② そのときゲームの状態が
「ゲーム実行中」だった場合、
ゲームの状態を「ゲームオーバー」にして
終了するようにします。



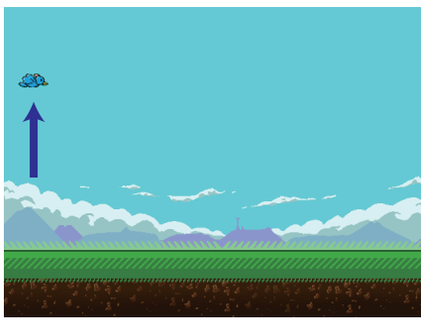
4 トリをジャンプさせよう

使用ファイル：ミドル C>2 日目 > 練習 > 練習 11.html

キャンバスをクリックしたときにトリがジャンプするようにしましょう。

```
// クリックしたときの関数
cv.addEventListener("click", function (event) {
        console.log(" クリックされました ");
        switch (state.now) {
          省略
          case state.playing:
            bird.flap();
            break;
          省略
// トリのオブジェクト
const bird = {
          省略
        jump: 4.5,
        speed: 0,
        flap: function () {
          this.speed = -this.jump;
        },
        update: function () {
```


保存（ctrl+s）



★できているかチェックをしよう！

☐ 画面をクリックすると
トリがジャンプするようになりましたか？

●ためしてみよう！

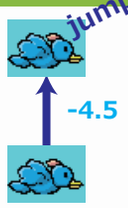
☐ トリのジャンプ力を変更して実行してみよう！

■ トリがジャンプする仕組み

flap メソッドが実行されると、
トリのジャンプ力にマイナスを掛け算した値が
移動スピード「this.speed」に入ります。
マイナスの値は上への動きになるので、ジャンプするようになります。

ジャンプ力が 4.5 のとき

```
flap: function () {
        this.speed = -this.jump;
                     -4.5
      },
```

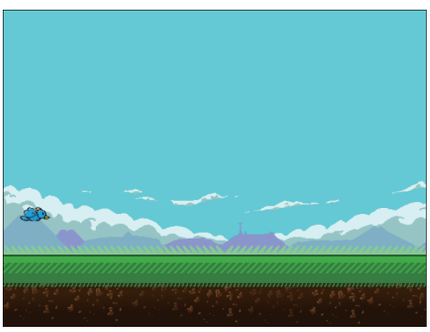


5 トリのスピードをリセットする

使用ファイル：ミドル C >2 日目 > 練習 > 練習 12.html

前のページでは、ゲームオーバーして再び実行したとき、トリが一気に下まで落ちてしまいました。
これを修正するために、トリのスピードをリセットします。

```
// クリックしたときの関数
cv.addEventListener("click", function (event) {
    console.log(" クリックされました ");
    switch (state.now) {
        省略
        case state.over:  ←――――――――――――― ②
            state.now = state.start;
            bird.speedReset();
            break;
        }
});
        省略
// トリのオブジェクト
const bird = {
        省略
        jump: 4.5,
        speed: 0,
        speedReset: function () {  ←――――― ①
          this.speed = 0;
        },
        flap: function () {
          this.speed = -this.jump;
        },
```

★できているかチェックをしよう！

☐ ゲームオーバーになったあと、再びスタートすると
トリが下に落ちるスピードがリセットされましたか？

■ 下降スピードリセットの仕組み

① トリにスピードリセットのメソッドを用意します。
② 状態がゲームオーバーのとき、キャンバスをクリックするとこのメソッドが実行されます。
下に落ちるスピードを元に戻してから、ゲームスタートの状態に移ります。

早く終わったら

チャレンジ課題：図形の描写

図形を描くプログラムを、自分で考えて作ってみましょう

1 HTML を作成しよう

練習　ミドル C>2 日目 > チャレンジ > チャレンジ 1 練習 .html

完成例　ミドル C>2 日目 > チャレンジ > チャレンジ 1 完成 .html

HTML を作成し、キャンバスを作りましょう。
キャンバスのサイズは、横 640・縦 480 にします。

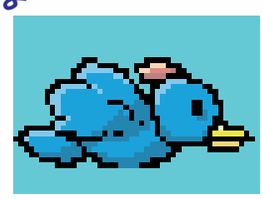
2 drawImage() で画像を表示しよう（ヒント：p.10 ～ 12）

練習　ミドル C>2 日目 > チャレンジ > チャレンジ 2 練習 .html

完成例　ミドル C>2 日目 > チャレンジ > チャレンジ 2 完成 .html

drawImage() を使った画像表示の練習です。
img フォルダに 1 枚の画像があります。数値を参考に、船と背景を表示してみましょう。

＼ブルーバードより／



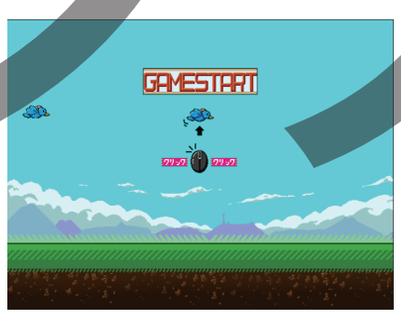
1 枚絵から画像を配置する方法は、わかったかな？
来月は、次の仕組みを作ってゲームを完成させます。

- **パイプや地面の動き**
- **当たり判定**
- **トリの複雑な動き**
- **スコア機能　　など**

お楽しみに！

ミドルコース

ゆけ！　ブルーバード（基礎編）
～１枚絵の画像配置を学ぼう～



KID's プログラミングラボ

ICT活用教育アドバイザー

◆文科省　ICT活用教育アドバイザー事務局掲載
学校ICT化サポート事業者
◆長期コースによる、プログラミングの普及