Canvas 授業

Canvas の使用例

http://festival.lattexplus.com/

http://experience.mausoleodiaugusto.it/en/intro

https://tkmh.me/

Canvas はロジックの組み立てが大事！(復習)

CSS

```
①
.area{          ②          ③
    background-color:red;
}
```

① class="area" がついている HTML タグの（どこの）
② background-color（背景色）を（何を）
③ red（赤色）にしたい（どうしたい）

JQ

```
$(".btn").on("click",function(){   ①
                イベント名

    $(".text").fadeOut(3000)
        メソッドとセレクタは「.」で繋げる

});.btn がクリックされたら function(){} の中身を順番に実行！！
```

① class="btn がついている HTML タグがクリックされたら
(いつ・どのタイミングで)
② class="text" がついている HTML タグを（何を）
③ 3 秒かけてだんだん消したい（どうしたい）

■ CSS

命令したい場所を決めて（どこの）
命令したい内容（何をどうしたい）を書く！

■ JavaScript（jQuery）

命令をするタイミングを指定し（いつ・イベント）
命令をする場所を決めて（どこを・セレクタ）
実際に命令する内容を書く（どうしたい・メソッド）

5 歳の息子に初めてのお使いを頼むのと同じ！

例)
お昼になったら（イベント）
近所のスーパーに行って（セレクタ）
大根を買ってきて！（メソッド）

Canvas を扱ってみよう！

1、HTML 上に Canvas 要素を書く。
2、JavaScript で描画のための準備をする。
3、描画処理を書く。

1-1、Canvas 描画の準備をする

canvas 上に描画を色々としたい場合、「canvas タグ（場所）」に「描画（命令）」をする、となりますので、Canvas タグを何度も呼び出すことになります。
何度呼び出されてもいいように、変数に canvas タグを入れておきましょう！


```HTML
<canvas id="×××" width="1024" height="760"></canvas>

//id      →Canvas 要素に命名するユニーク ID
//width   →canvas 要素の幅
//height →canvas 要素の高さ

※ CSS で幅と高さを指定すると、おかしなことになるので注意！！
(※ http://jsdo.it/castero/vo2w)
```

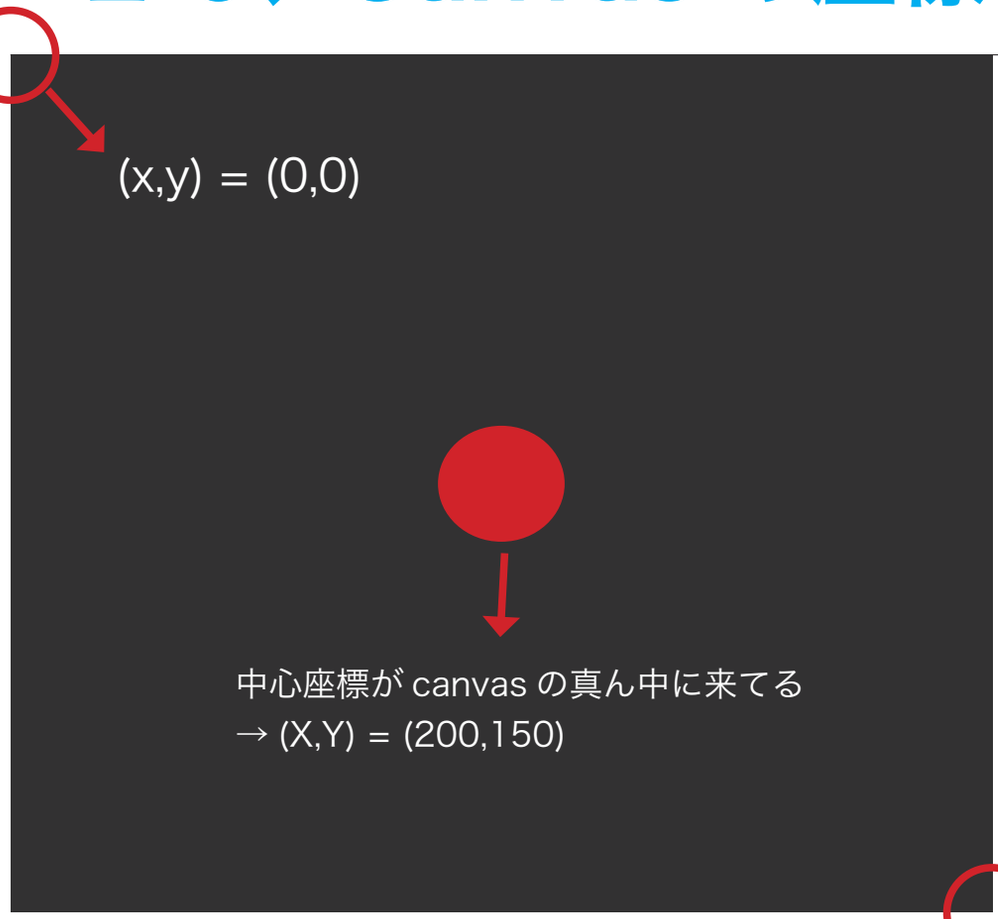
1-1、Canvas 描画の準備をする

・canvas タグを書いただけでは、描画は実行されない！
・getContext メソッドを実行して初めて描画が出来る！
（変数 ctx に canvas タグを自由に操作できる機能がたくさん入ったことになります）


```
//JavaScript で書くと
const can = document.getElementById("Canvas タグの ID");
const ctx = can.getContext("2d");

//jQuery で書くと
const can = $("Canvas タグの ID")[0];
const ctx = can.getContext("2d");
```

jQuery

2-0、Canvas の座標の考え方



※ canvas 要素の width=400,height=300 とした場合

Canvas 要素は左上を
x 座標 :y 座標 = 0 : 0
として考えます。
描画した図形・線の中心点が x 座標 y 座標
として定義されます。

(x,y) = (400,300)

2-1、Canvas で図形を描く時の手順

0、Canvas 要素をセレクタ指定

1、**「どんな見た目の図形を描くのか」**を決めて
(・色は？線の太さは？ etc……..)

2、**「どこにどんな形の図形を描くのか」**を決めて
(・座標位置は？丸？四角？・etc……..)

3、実際の描画を実行！

2-2、どんな見た目の図形を描くのか？？

ctx.fillStyle・・・・・・・塗りつぶし色の指定
ctx.strokeStyle・・・・・線の色の指定
ctx.lineWidth・・・・・・線の太さの指定

■書き方例

```
ctx.fillStyle = "#f90";
ctx.strokeStyle = "#090";
ctx.lineWidth = 9;
```

http://www.html5.jp/canvas/ref.html
「2d コンテキストのプロパティ」を参照

3-2、どこにどんな形の図形を描くのか？（四角）

■図形を描画 - 長方形などの四角形 -（1,2 つめの引数は中心点の座標）
※共通して引数は以下の通り

（図形の左上点の X 座標 , 図形の左上点の Y 座標 , 図形の幅 , 図形の高さ）

■書き方例

```
ctx.fillRect(100, 100, 30, 30 ); // 塗りつぶし
ctx.strokeRect(100, 100, 30, 30 ); // 枠線のみ
ctx.clearRect(100, 100, 50, 50 ); // 消す
```

※線付き四角形を作るときは **fillRect()** と **StrokeRect()** を両方使う。

4-1、図形をアニメーションさせる

■ setInterval() や requestAnimationFrame() を使用

```
<script>
  var timer = setInterval(function(){
   ctx.fillStyle="#fff";
   ctx.clearRect(0,0,300,300);
   ctx.fillStyle="#f00";
   ctx.fillRect(30+count,30+count,30,30);
   count++;
   if(count>200){
    clearInterval(timer);
   }
  },100);
</script>
```

演習

Player

※一番左端でマウスに合わせて上下に動く

小菅

小菅を倒すシューティングゲーム

制作工程 - プレイヤー編 -

drawImage()でcanvas上にPlayer画像配置

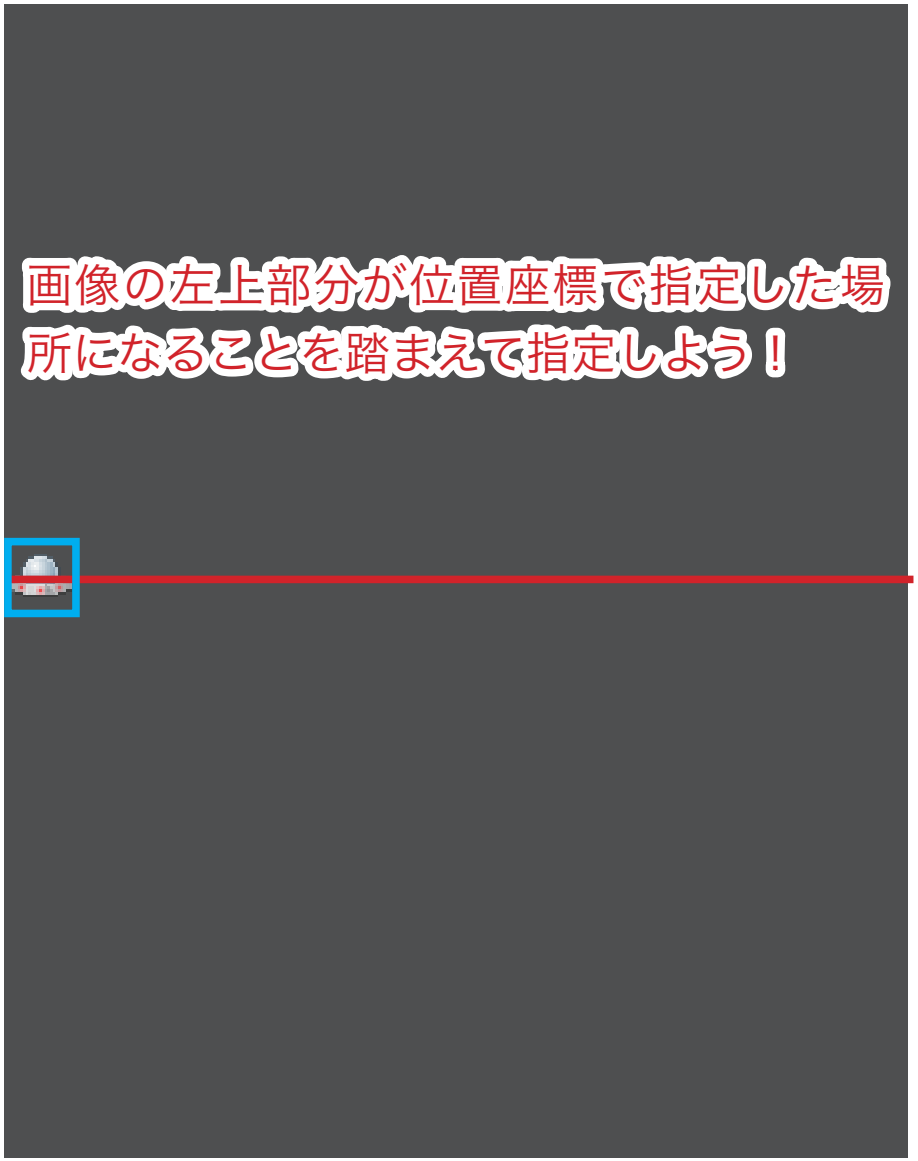
ctx.drawImage(配置画像 ,X 軸位置 ,Y 軸位置 , 横幅 , 高さ);

※ http://www.html5.jp/canvas/ref/method/drawImage.html
※動画も配置可能

まずは Player 機を Canvas 上に描いてみましょう。
考え方としては「canvas 上の**「どこに」「どんな大きさで」**画像を表示するかを設定する、というものになります。（大きさは省略可能）
drawImage() が実行される時点で画像データがきちんと読み込めている状態であるかどうかが注意点になります。

※まずは canvas エリアの一番左、上下中央に配置します！

drawImage()でcanvas上にPlayer画像配置




```
const newImage = $("<img>");
newImage.attr("src","images/ufo.gif");
newImage.on("load",function(){
      ctx.drawImage(this,0,can.height/2);
});
```

※ X 座標・Y 座標の値を変更して、player 機の位置が変わることを確認してみてください！

キーヴォード操作で player 機を動かす

can.on("mousemove", 関数名);

次に canvas 上でマウスを動かして player 機を動かす機能を入れてみましょう。マウスの動きは、mousemove イベントを使って定義可能です。

「マウスが canvas 上で mousemove されたときだけ」player 機を動かしたい！

イベントオブジェクト

```
$(window).on("mousemove",function(e){
    // 実行内容
});
```

function() のところによく書かれているのを見る「e」。
これは**イベントオブジェクト**といって、そのイベントが発生した場所における各種データのまとまりです。

上記例であれば、mouse イベントが発生した場所（地点）における様々なデータが、e という変数の中に入っているのです！

※例
座標位置・・・・e.offsetX

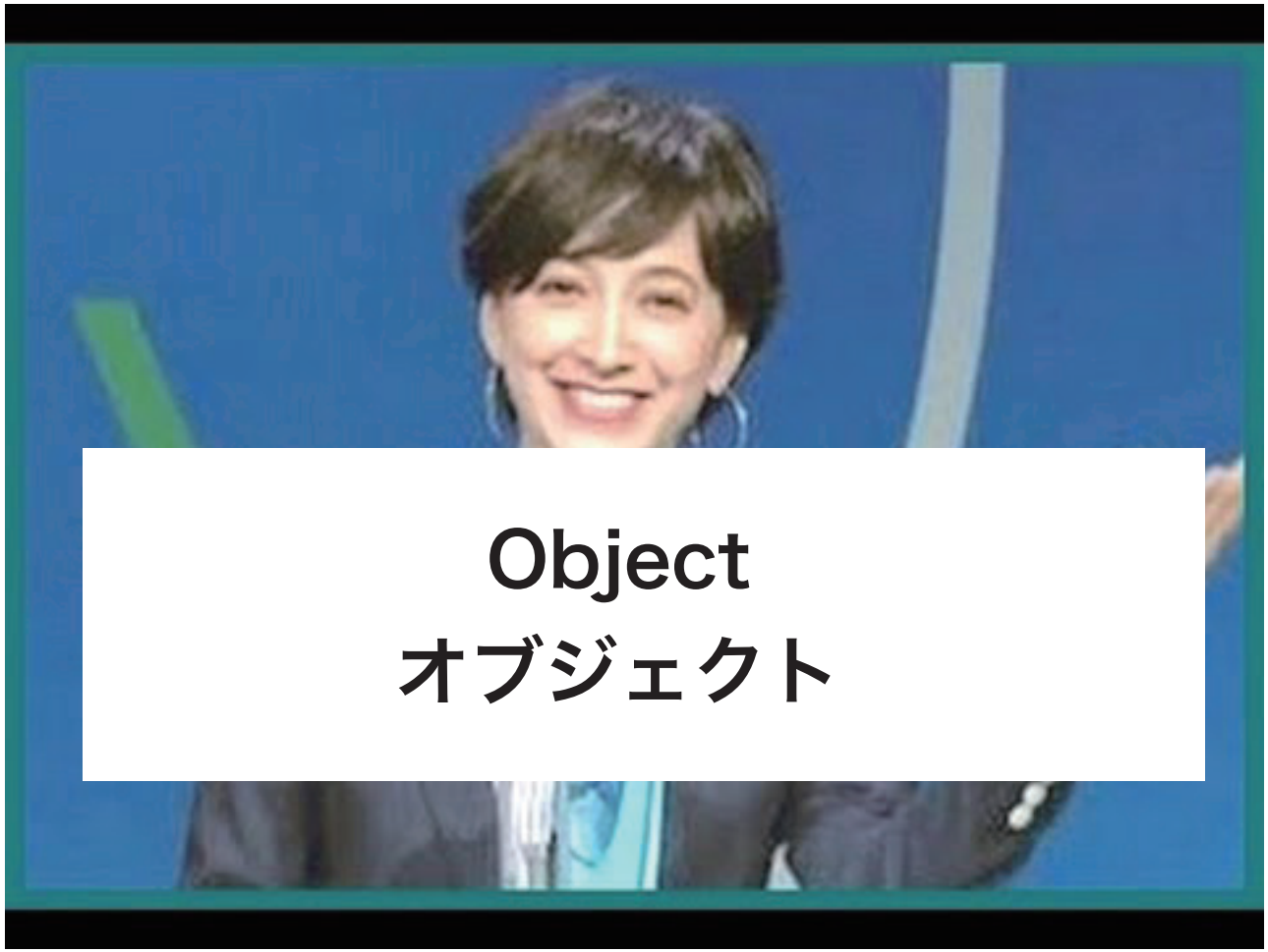

▼KeyboardEvent {isTrusted: true, key: "ArrowLeft", code: "ArrowLeft", location: 0, ctrlKey: false, …}
altKey: false
bubbles: true
cancelBubble: false
cancelable: true
charCode: 0
code: "ArrowLeft"
composed: true
ctrlKey: false
currentTarget: null
defaultPrevented: fals
detail: 0
eventPhase: 0
isComposing: false
isTrusted: true
key: "ArrowLeft"
keyCode: 37
location: 0
metaKey: false
▶path: (4) [body, html, document, Window]
repeat: false
returnValue: true
shiftKey: false
▶sourceCapabilities: InputDeviceCapabilities {firesTouchEvents: false}
▶srcElement: body
▶target: body
timeStamp: 17453.19999998901
type: "keydown"
▶view: Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window, …}
which: 37
▶__proto__: KeyboardEvent
例：keyCode →それぞれのキーに割り振られている番号のこと！
(例えば、enter キーなら 13 番 !)

マウスを動かす度にufo の縦軸位置が変わる
→位置座標の管理方法は？？

必要な分だけ変数をたくさん作ってもいいけど・・・面倒！


Omotenasi
オモテナシ


Omo
オモテナ


Object
オブジェクト

オブジェクト

```
let ufo = {
        posX:0,//X 座標
        posY:can.height/2,//Y 座標
        flag:false,// 動くかどうかの flag
        img:"images/ufo.gif"// 画像データ管理
        // 最初は画像の URL を格納
}
```

オブジェクトは、特定のものに関するデータの集まり・固まりのことです。
{ } を使って、複数のデータを key - value 型でカンマ区切りで管理します。

オブジェクトのデータへのアクセス方法

ufo.posY・・・・・ ドット記法
ufo["posY"]・・・・ブラケット記法

```
console.log(ufo.posY);
console.log(ufo["posY"]);
```

→どちらの方法でもデータ表示できます。

※キーの名前は予約語などの利用は通常不可。
ブラケット記法使用した場合、不正に使えてしまう。

実はみなさんも今まで触れていたオブジェクト

Math オブジェクト	数値演算に関するデータとデータ操作方法がまとまっている。（じゃんけんアプリでやった！！）
Date オブジェクト	時間に関するデータと、データ操作方法がまとまっている。(new Date() して、getMonth() 的な)
DOM オブジェクト	HTML タグに関するデータと、データ操作方法がまとまっている（$("body").CSS("color","red))

JavaScript →様々なデータを操作

オブジェクト
→様々なデータの集まり + データ操作方法・関数の集まり

※例
alert()・・・window オブジェクトに紐づいた関数

オブジェクトは後からデータの設定も可能

```
const ufo = {
    posX:0,
    posY:can.height/2,
    flag:false,
    img:"images/ufo.gif",
};
```

※ const 使ってますが、オブジェクト代入している場合、その中身の値（プロパティ）は後から変更可能です。

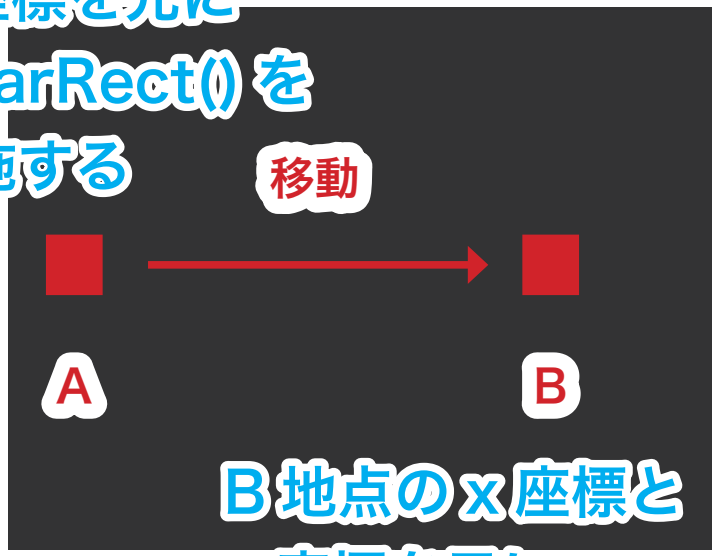
```
const ufoDraw=function(){
    const newImage = $("<img>").
attr("src",ufo.img);
    newImage.on("load",function(){
        ctx.drawImage(this,ufo.posX,ufo.
posY);
        ufo.img = this;
        ufo.width=this.width;
        ufo.height=this.height;
    });
}

$(window).on("load",ufoDraw);
```

→ページ読み込み時に ufo 表示するための関数実行。

clearRect（canvas 上の描画物消去）

ctx.clearRect(基準Ｘ座標 , 基準Ｙ 座標 , 消去エリアの幅 , 消去エリアの高さ)

A 地点の x 座標と
y 座標を元に
clearRect() を
実施する



canvas 上の描画物は DOM オブジェクトのように扱えないため、animate などが効かない！

→ A から B への移動を表現する場合、**A に描かれている内容を一度消去して、そのあと B に新しく描き直す**ことで、「移動」を表現します。
（clearRect → drawImage を繰り返す）

ここまでの演習

十字キーを動かしたら、Player 機が上下に動くようにする !!

■注意点

・canvas エリアからはみ出して Player 機が表示されないようにする。
・Player 機が重複して表示されないようにする。
・マウスの Y 座標 = ufo の上下中央の位置、となるようにする。

ここまでの演習２

canvas の一番右に敵の画像を表示してみよう！

■注意点

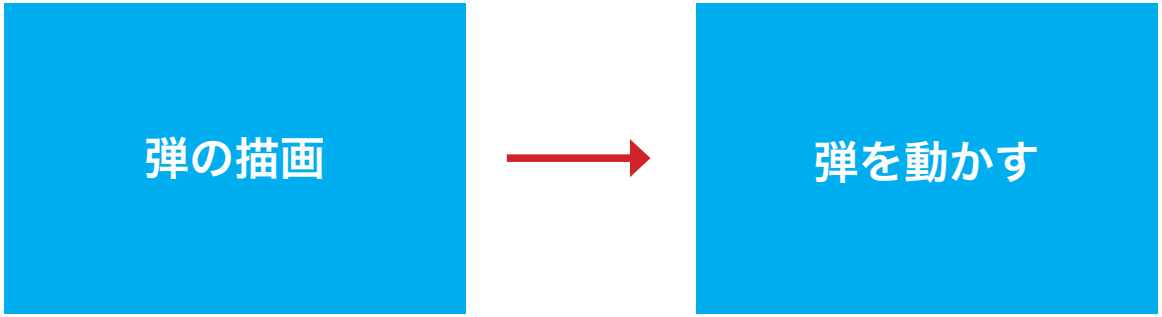
・敵の画像は **images/stamp10.png** です。

・静止して表示すれば OK なので、マウスの動きに追従する必要はありません。

（flag は不要です）

Player 機から弾を発射させる！

制作工程 - 弾の発射から弾を動かすまで -



弾の座標情報は Player の座標に依存する

画面外に出た弾はデータから削除

fillRect() でcanvas上に四角形の描画

ctx.fillRect(X 軸位置 , Y 軸位置 , 横幅 , 高さ);

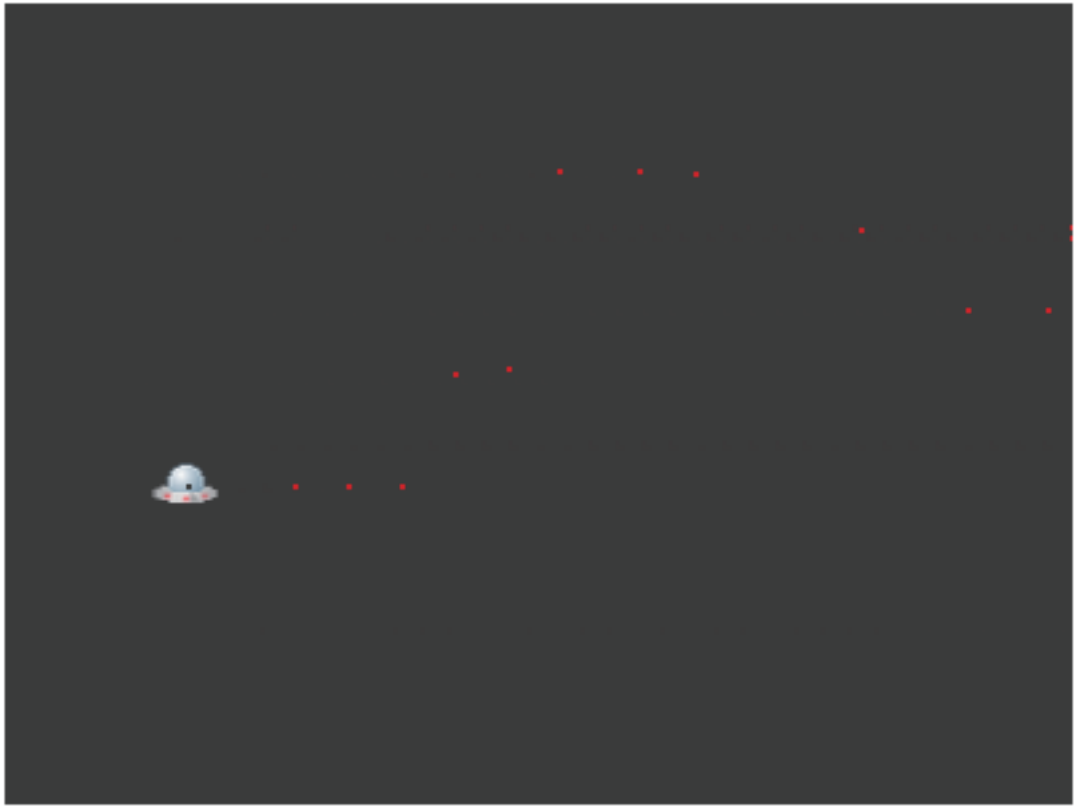
※ http://www.html5.jp/canvas/ref/method/fillRect.html

次に、弾を描くための練習として、canvas 上に四角形を描いてみましょう。
X 軸、Y 軸位置で指定した座標は、四角形の左上部分に該当します。

※色などの変更も可能（ fillRect() の前に必ず記述）
ctx.fillStyle ="**#f00**"; // 赤への変更

１発１発の弾オブジェクトの作成

```
const ballData = {
        speed:5, // 弾の発射スピード
        width:10, // 弾の幅
        height:10, // 弾の高さ
        posX:0, // 弾の X 位置情報
        posY:0, // 弾の Y 位置情報
        color:"#d00", // 弾の色
}
```

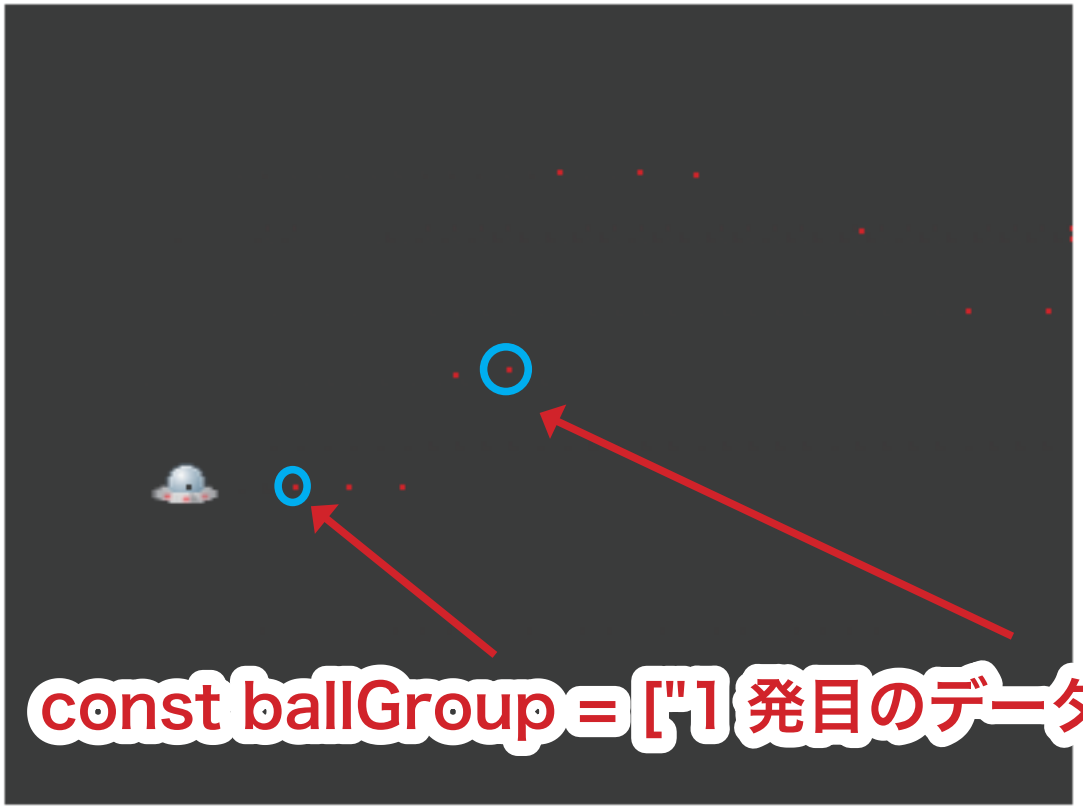


弾は複数打ちたい！

複数の弾の位置座標管理はどうすればいい？
オブジェクト１つだと上書きされちゃう・・・

そうだ！配列だ！

配列を使って弾のデータを個別で管理



マウスクリックの度に配列に ballData オブジェクトのデータを複製し、新しい弾のデータを push!!!

Object.assign()

JavaScript では、オブジェクトを他の変数にコピー複製すると、元オブジェクトへの参照がコピーされるため、元あるいはコピーオブジェクトのプロパティ変更をすると、双方のプロパティが変更されてしまいます。

※例

```
const ballData = {
    speed:5,
    w:10,
    h:10,
    color:"#d00"
}
ballData2 = ballData;
ballData2.color="#00f";
console.log(ball);//color → #f00 になっている！
```

これだと状況に応じて弾の見た目を変更したりすることができない！

Object.assign()

const copyObject = Object.assign({ }, baseObject);

Object.assign() を使用すると、元のオブジェクトからデータのみをコピーし、複製オブジェクトを作成することができます。
ちなみに、複数のオブジェクトをがっちゃんこして、新しいオブジェクトを作成することも可能です。

```
※例
const baseKosuge = {age:35,gender:"men"};
const newKosuge = Object.assign({prefecture:"Saitama"},bas
eKosuge);
console.log(newKosuge);
→{age:35,gender:"men",prefecture:"Saitama"};
```

弾を出力する（Object.assign() を活用）

この時点ではまだ弾は出力されるだけで
動かないので注意！！

```
const ballData = {
        speed:5,// 弾のスピード
        width:10,// 弾の横幅サイズ
        height:10,// 弾の縦幅サイズ
        posX:0, // 発射時に設定のため後で調整
        posY:0, // マウス位置に依存するので後で調整
        color:"#d00",
}
const ballGroup = []; // 弾を格納する配列を作成
```

弾を出力する（Object.assign() を活用）

この時点ではまだ弾は出力されるだけで
動かないので注意！！

```
const shootBall = function(e){
        const newShootBall =
                Object.assign({ },ballData);
        newShootBall.posY =
                ufo.posY
            + (ufo.height/2 - newShootBall.height/2);
        ctx.fillStyle = newShootBall.color;
        ctx.fillRect(
                newShootBall.posX,
                newShootBall.posY,
                newShootBall.width,
                newShootBall.height
        );
        ballGroup.push(newShootBall);
}
// クリックしたら弾発射
$(can).on("mousedown",shootBall);
```

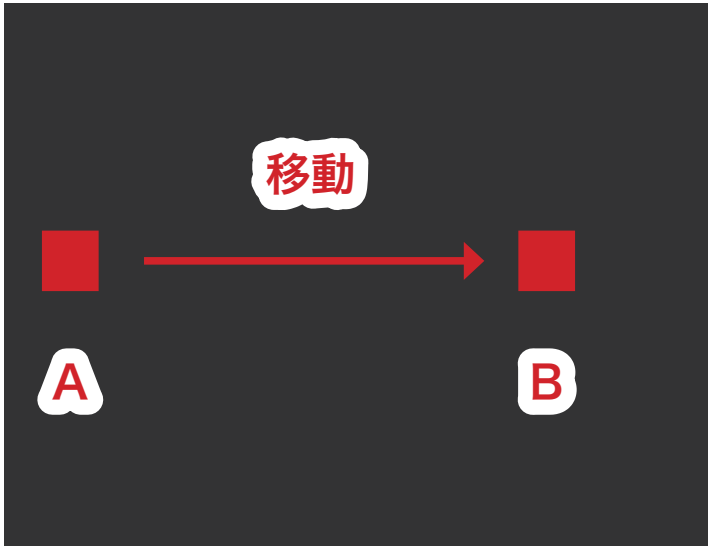
弾を動かす

弾を動かす！！

基本的には Player 機を動かすときの考え方と同じになります。

弾の場合は画面外に消えるまで弾を動かし続けなければならないため、

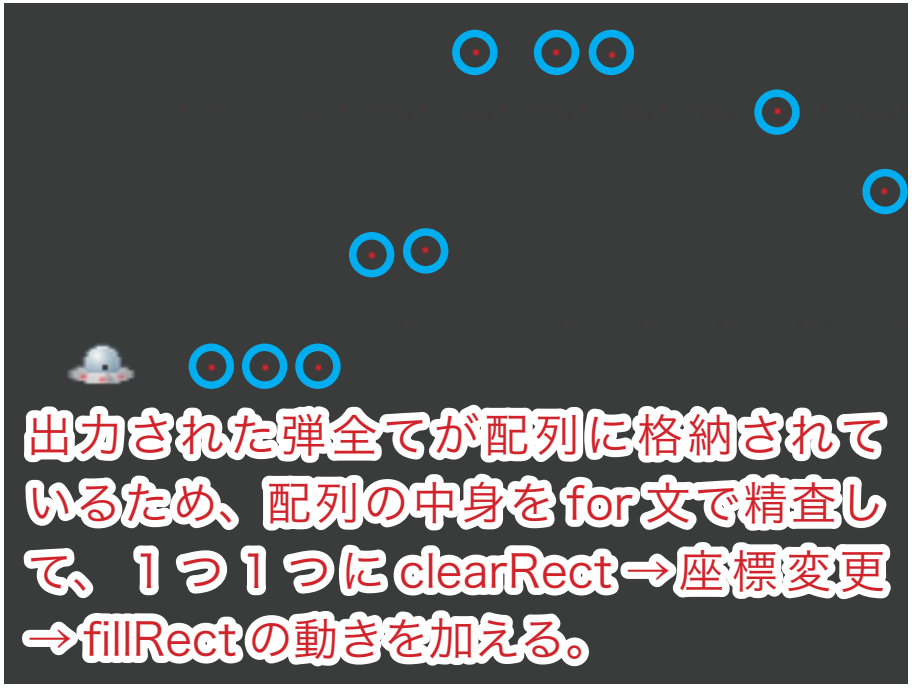
動かす関数を作ってループで動かす仕組みが必要です。



canvas 上の物は DOM オブジェクトのように扱えないため、animate などが効かない！

→ A から B への移動を表現する場合、**A に描かれている内容を一度消去して、そのあと B に新しく描き直す**ことで、「移動」を表現します。
（clearRect → drawImage を繰り返す）

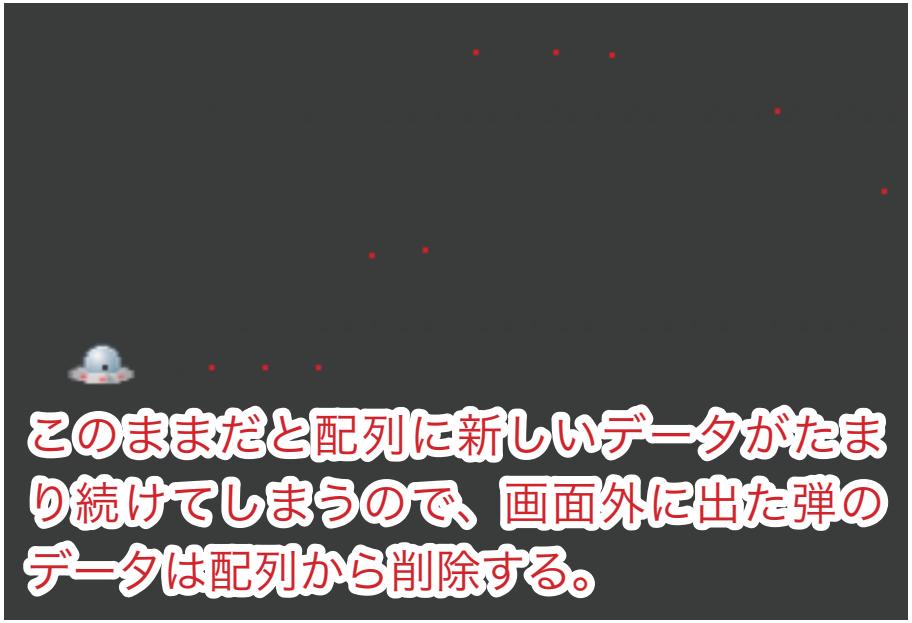
弾を動かす



出力された弾全てが配列に格納されているため、配列の中身をfor文で精査して、１つ１つにclearRect→座標変更→fillRectの動きを加える。

```
const moveBall = function(){
      for(let i=0; i < ballGroup.length; i++){
            const ball = ballGroup[i];
            ctx.clearRect(ball.posX, ball.posY,
ball.width, ball.height);
            ctx.fillStyle = ball.color;
            ball.posX += ball.speed;
            ctx.fillRect(ball.posX, ball.posY, ball.
width, ball.height);
      }
};
```

画面外に出た弾のデータは削除

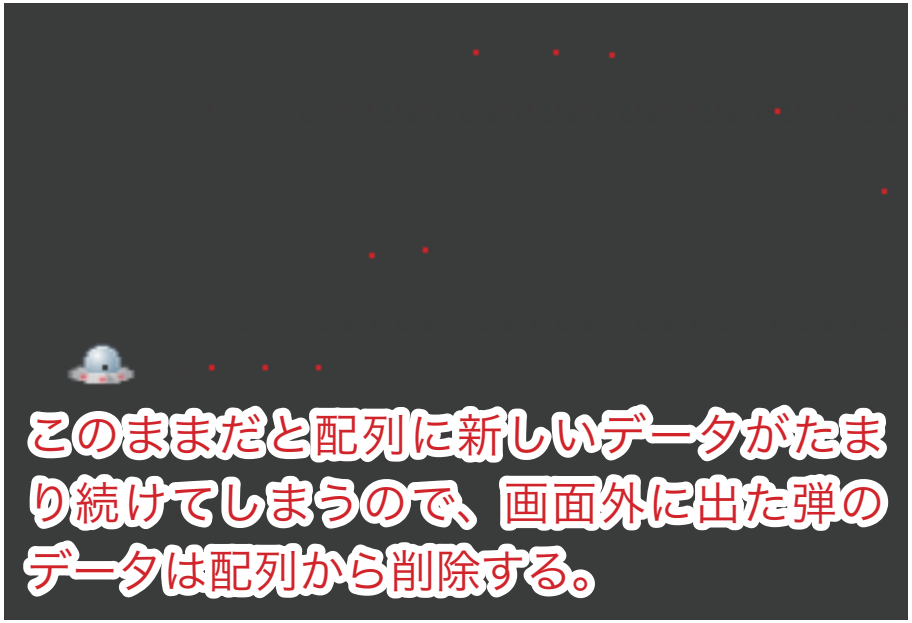


このままだと配列に新しいデータがたまり続けてしまうので、画面外に出た弾のデータは配列から削除する。

```
const deleteBall=function(){
      for(let i = 0; i < ballGroup.length; i++) {
            if (ballGroup[i].posX >= can.width) {
                  ballGroup.splice(i,1);
            }
      }
};
```

※ splice(a,b)・・・配列の a 番目から b 個分データ削除するメソッド。

setInteval 関数でループで関数実行




```
setInterval(function(){
        moveBall();
        deleteBall();
},100);
```

ここまでの演習

十字キーを動かしたら、Player 機が上下に動き、かつ弾を自由自在に発射させる！！！

■注意点

・配列に弾のデータがたまりすぎないように削除処理をきちんと行う！

・位置の管理の仕方間違えると canvas 上に描画が残るので注意。

当たり判定

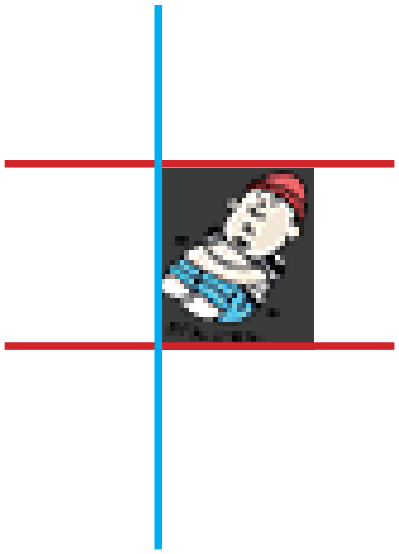
弾の右端の座標が敵の左端の座標以上になり

弾の上端が敵下端以下にあり

弾の下端が的上端以上にある

↓

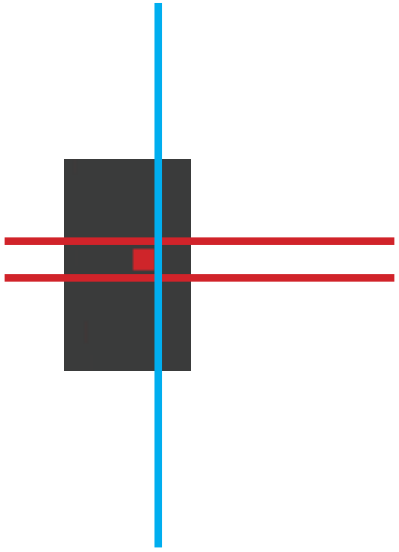
当たり！と判定できる。



Y 座標：ememy.posY

Y 座標：ememy.posY+ememy.height

X 座標：ememy.posX

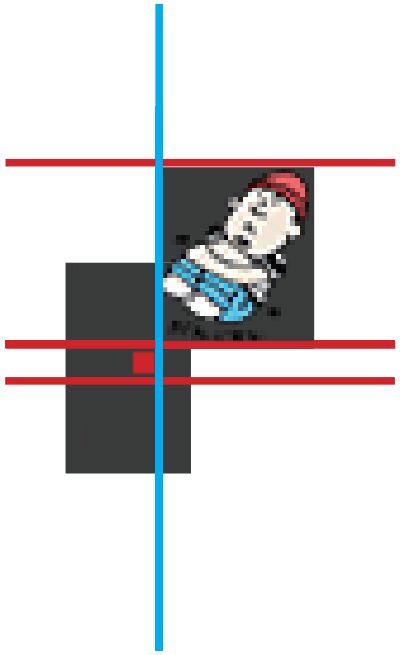


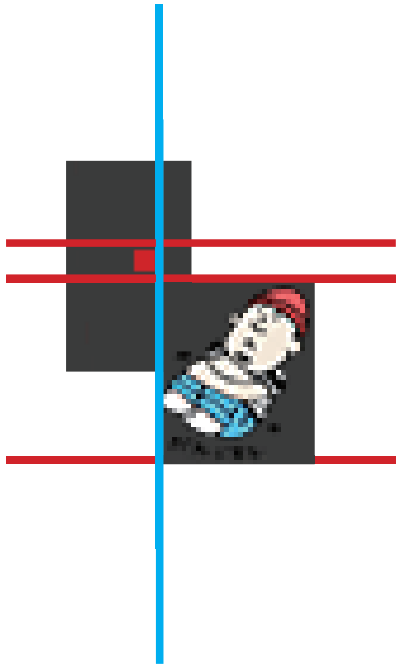
Y 座標：ballGroup[i].posY

Y 座標：ballGroup[i].posY+ballGroup[i].height

X 座標：ballGroup[k].posX + ballGroup[k].width

ballRight > enemyLeft





ballTop < enemyBottom

ballBottom > enemyTop

```
    const hitJudge = function(){
        for(let k=0;k<ballGroup.length;k++){
            // 弾の上下左右の座標をわかりやすく変数にして管理（この辺オブジェクトにしても OK）
            const ballLeft = ballGroup[k].posX;
            const ballRight = ballLeft + ballGroup[k].width;
            const ballTop = ballGroup[k].posY;
            const ballBottom = ballTop + ballGroup[k].height;
            // 敵の上下左右の座標をわかりやすく変数にして管理（この辺オブジェクトにしても OK）
            const enemyLeft = enemy.posX;
            const enemyTop = enemy.posY;
            const enemyBottom = enemyTop + enemy.height;

            if((ballRight >= enemyLeft) && (ballTop <= enemyBottom) && (ballBottom
>= enemyTop)){
                ctx.clearRect(ballLeft, ballTop, ballGroup[k].width, ballGroup[k].height);
                ballGroup.splice(k,1);
                console.log(" あたった ");
            }
        }
    };
```

演習

当たり判定を実施して、弾が当たった時は弾を削除しよう。

■注意点

・配列から該当要素を削除し、インデックスを詰める場合は、splice() を使用するとよいです。

https://msdn.microsoft.com/ja-jp/library/wctc5k7s(v=vs.94).aspx

array.splice(2,1);

→配列の３つから数えて１つ要素を削除する。かつ、空いたところを詰める。

課題

課題内容

オブジェクトの扱い習得が今回の講義目的なので、新しいオブジェクトデータを使ってゲームに機能追加させる。

■機能追加例

・敵も弾を発射させる。（**敵の弾オブジェクト**を追加）

・**アイテムオブジェクト**を作成し、アイテムをゲットしたら弾が大きくなるようにする。

・敵を複数出現（種類複数でも OK）させるようにする。

■プラスアルファ要素例

・スコア機能、ゲームの時間制限、を追加する。

・3 発敵に弾を当てたらクリアにする。

・自分に 3 発弾が当たったらゲームオーバーにする。

シューティングだとテンション上がらない方は

オブジェクトを使ったデータ管理していたらシューティングゲーム以外でも OK にします！

■ゲーム例

・パーティクル表現を使ったゲーム。
・アルカノイドやピンボールなどのボードゲームチックなもの。
・

■参考 URL

・http://jscanvas.hatenablog.com/
・http://jsdo.it/　や https://codepen.io/ で canvas 関連調べる。

Developer Tools のブレイクポイント機能

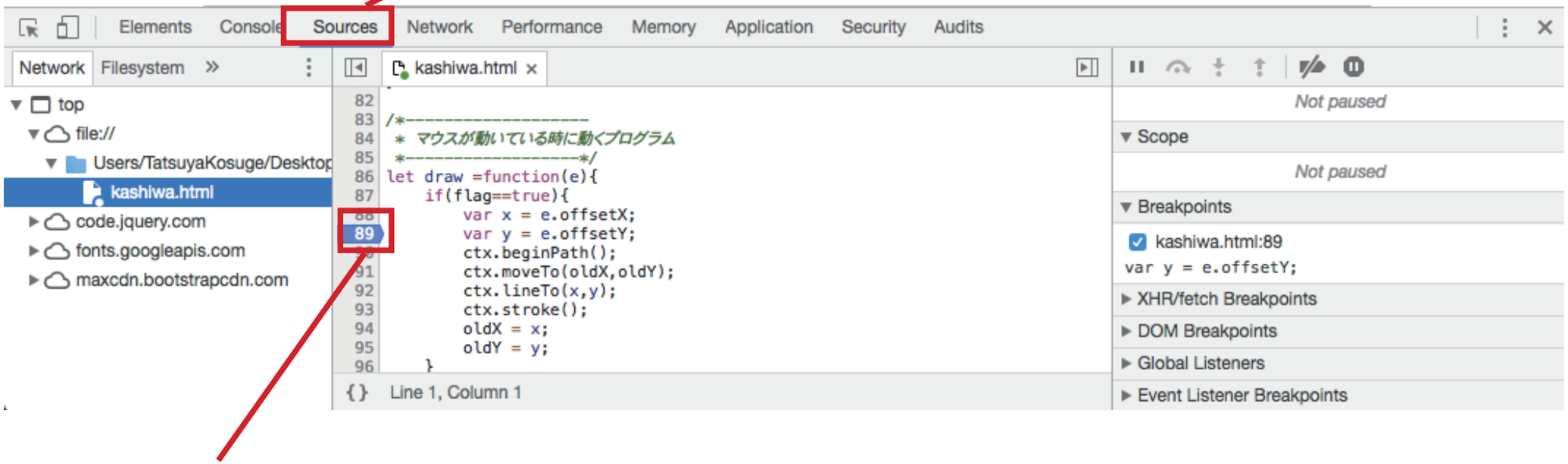
Developer Tools のブレイクポイント機能を使ってデバッグ上手に！

指定した関数が動かない！？そんなときはありませんか？？
そんなときは、Developer Tools のブレイクポイントを使うと、かなり便利に効率的にデバッグを行うことができます。

今回のうちに、ブレイクポイント機能の使い方をマスターしましょう！

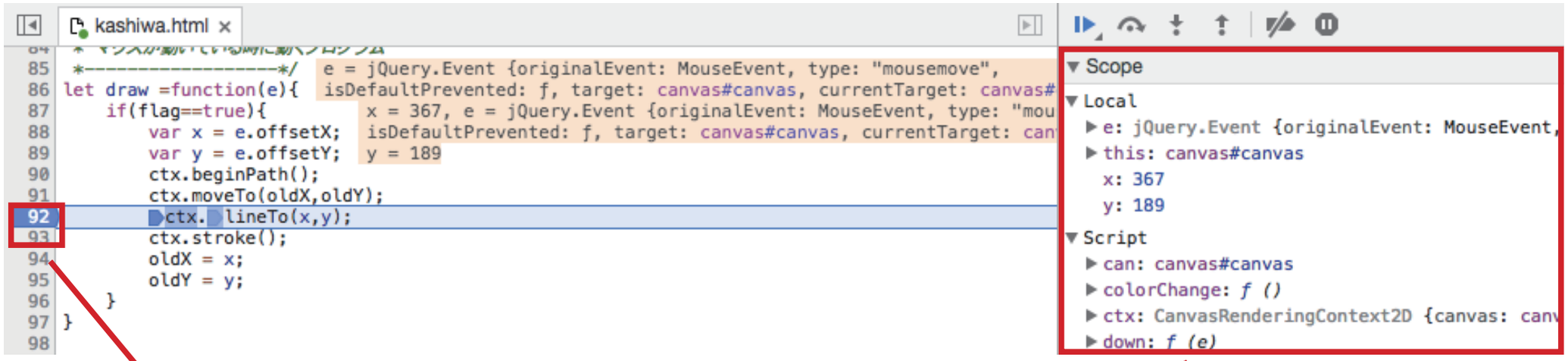
Developer Tools のブレイクポイント機能

Sources を選択



プログラムの動きを止めたい行にマーカーを打ちます。その行が読み込まれた時点で、一度動きを止めることができます。

Developer Tools のブレイクポイント機能



プログラムの動きを止めたい行にマーカーを打ちます。その行が読み込まれた時点で、一度動きを止めることができます。

動きを止めた時点で、各変数のスコープ範囲や、具体的な変数の中身などを確認することができます。
（scope 欄）

ありがとうございました