



# Girls Who Code At Home

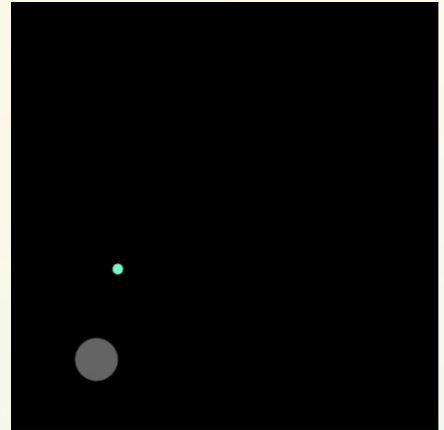
流星キャッチャーゲーム: Part 4

流星をつかまえる

## 学習概要

第3回では、p5.jsで変数を作成して使用し、指定された速度で画面上流星を移動させる方法を学びました。今回は、双方向のゲームにしてみましょ。このパートでは、マウスの動きに連動するキャッチャーをプログラミングして、プレイヤーの（操作）入力をゲームに反映させる方法を学習します。このキャッチャーは、半透明の白い楕円で、マウスの動きにあわせて動きます。[ここ](#)をクリックすると、このアクティビティが終了するまでに、習得できる内容をプレビューできます。

このアクティビティに取り掛かる前に、「流星キャッチャーゲームシリーズ」の[パート1](#)、[パート2](#)、[パート3](#)をすでに終了している必要があります。



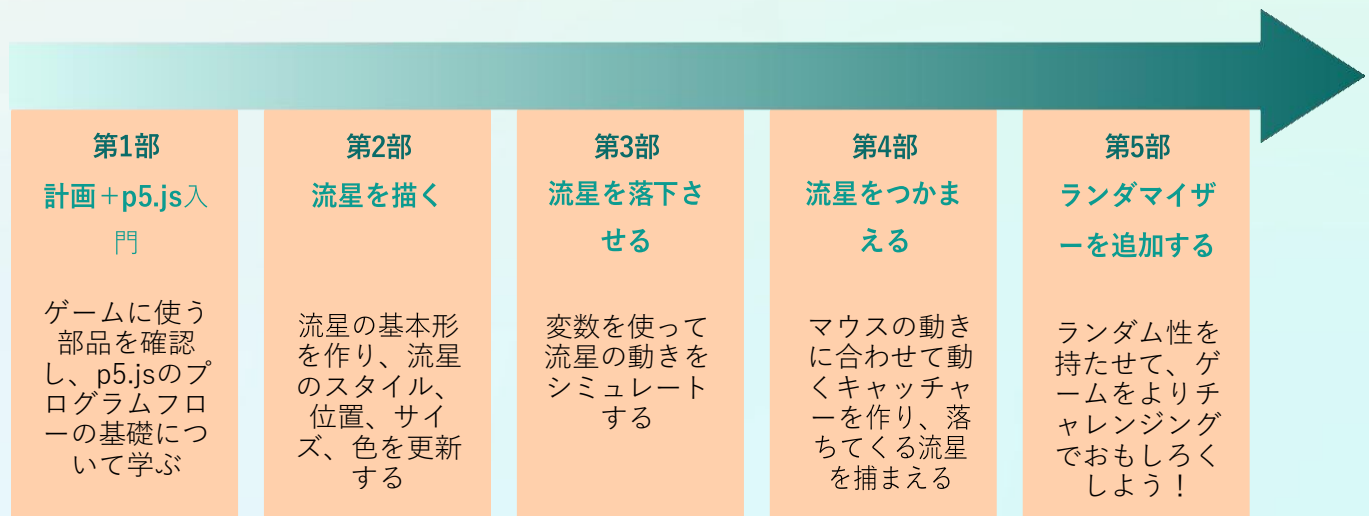
## 学習目標

このアクティビティが終わるころには、次のことができるようになります

- 基本的な動きをプログラムでシミュレートする方法を説明する。
- 変数と算術演算子を使って、さまざまな動作を要素にプログラムする。

## 教材

- [p5.js オンラインエディタ](#)
- [流星キャッチャーゲームサンプルプロジェクト](#)
- [流星キャッチャーゲームその4 参考資料](#)



流星キャッチャーゲームの[動画チュートリアル](#)のパート4、パート5も参照しながら、すすめることもできます。

## Women in Tech スポットライト: Rebecca Cohen Palacios



Image Source:  
[GamesIndustry.biz](https://www.gamesindustry.biz)

もし、あなたが「エルダースクロール：ブレイズ」「アサシンクリード オリジンズ」「アサシンクリード シンジケート」「Shape Up (Kinect 版)」などのビデオゲームをプレイしたことがあれば、既にレベッカが制作に関わった作品に触れたことがあるということになります！ レベッカが、ターニャ・ショートと共同で設立した[Pixelles](#)という組織は、ゲーム業界の女性増加にも貢献しています。

Pixellesは、ゲーム開発に携わる女性を増やすことを目的とした非営利団体です。レベッカは、ビデオゲームがまだ「男の子のもの」として認識されていることに加え、女性への偏見やサポート不足、ガラスの天井（資質・実績があっても女性やマイノリティを一定の職位以上には昇進させようとしない組織内の障壁）のためにゲーム業界から去っていく女性の割合が多いことに気づきました。Pixellesは、意欲的な女性、中堅女性たち向けに、

毎月の無料ワークショップ、メンターシップ、「ゲームを作ってみよう」講座、キャリア促進、ネットワーキング等々を提供しています。

レベッカは、Pixellesの共同ディレクターを務める傍ら、現在は[Behaviour Interactive社](#)でシニアUIデザイナーとして働いています。Ubisoft社でゲーム業界に入る前は、グラフィックおよびウェブデベロッパーとして6年間で勤務していました。

レベッカがPixellesの活動を通じてゲーム業界のジェンダーギャップをどのように解消しようとしているのかをもっと詳しく知りたい人は、次のリンクを参照ください（英語）。

- ["Pixelles is Helping Mid-Career Mothers Stay in Games"](#)
- ["Top 7 Reasons Women Quit Game Development"](#)
- ["Montreal non-profit gives women a chance to break into male-dominated video game industry"](#)
- ["Empowerment Through Game Development": The Pixelles Game Incubator"](#)

## 考えてみましょう

コンピュータサイエンティストであることは、単にコーディングが得意というだけではありません。偉大なコンピュータサイエンティストは、勇気、レジリエンス（困難や脅威に直面している状況に対して、「うまく適応できる能力」、創造性や目標を構築していくという強さも持っているのです。レベッカや彼女の仕事も、例外ではありません。



RESILIENCE

ゲーム業界で女性が直面する苦悩は何ですか？Pixellesはどのように女性のキャリアを支えているのでしょうか？

あなたの考えたことを家族や友人と共有しましょう。他の人にもレベッカのことを知ってもらい、考えたことを議論してみましょう

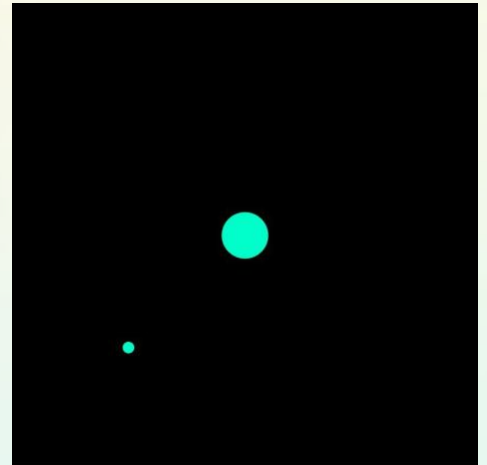
## ステップ1: キャッチャーを追加する (5～10分)

### キャッチャーを描く (3～5分)

まず、**ellipse()** 関数を使ってキャッチャーを画面に描画する必要があります。流星のときと同じように、キャッチャーの幅と高さを変数に格納します。次のステップでは、x と y の位置に特別な変数を使用するので、これらの値を格納するための変数を作成する必要はありません。

- ❑ **setup()** の上に新しい変数を追加して、キャッチャーの幅と高さを格納します。キャッチャーは円形なので、幅と高さと同じ値を使用します。サンプルでは変数に *catcherDiameter* という名前をつけましたが、好きな名前にしても大丈夫です。ただ、コードの中で変数名を間違えないようにしてください。
- ❑ キャッチャー幅と高さの変数に**40**の値を割り当てます。
- ❑ **draw()** 関数で、**ellipse()** 関数を使用してキャッチャーを描画します。これを流星のコードの下に追加します。これがキャッチャーであることを忘れないように、短いコメントを追加しておいてください。
- ❑ xとyのパラメータを**200**に設定し、幅と高さのパラメータを上記で作成した変数に設定します。
- ❑ コードを実行します。

キャンバスの真ん中に青緑色の丸が見えるはずですが。

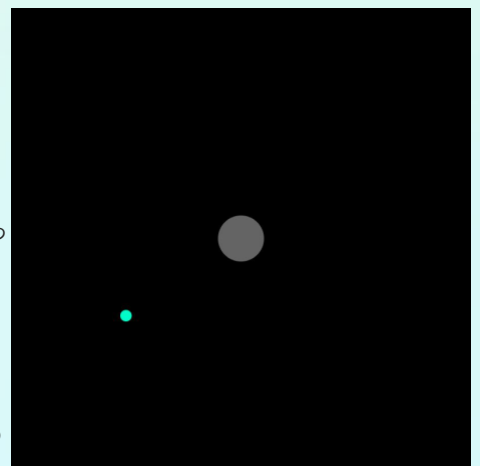


### キャッチャーの色を変える (3～5 分)

キャッチャーは白色で半透明にし、キャッチャーを通して流星を見ることができるようになりたいと思います。これは、**fill()** 関数をもう一度使う必要があることを意味します。以前に、**fill()** は呼び出された後のすべての図形を指定した色で塗りつぶすものと説明しました。これは、絵筆をきれいにして別の絵の具をつけるという作業のようです。

- ❑ キャッチャーの**ellipse()** 関数の上に新しい**fill()** 関数を作成します。これで、これ以降のすべての図形に新しい色が適用されます。
- ❑ RGB 値を追加して、キャッチャーを白にします。
- ❑ 透明度を制御する4番目のパラメータを追加し、この値を**100**に設定します。このオプションのパラメータはアルファ値と呼ばれます。0から255の間の任意の値を設定でき、0は完全に透明、255は不透明になります。いくつかの異なる値をいじってみて、キャッチャーの表示がどのように変わるかを試すことができます。
- ❑ コードを実行します。

キャッチャーの色が半透明の白に変わるのが確認できるはずですが。下の画像では、キャッチャーがグレーに見えることがあります。半透明のため、背景色の一部が透けて見えるのです。



参考資料の2ページ目でコードを確認することを忘れないでください。

## ステップ2：プレイヤーの入力値の追加(2～4分)

今のところ、キャッチャーはあまり動いていません。プレイヤーがマウスを使って、キャッチャーを動かせるようにしましょう。そのためには、キャッチャーのXとYの値が、マウスのXとYの値に連動するように必要があります。

幸運なことにp5.jsの2つの変数はそのようなことができるように作られています。変数`mouseX`と`mouseY`で、マウスの水平方向と垂直方向の位置を取得することができます。[サンプルのスケッチ](#)でマウスを動かしてみて、値がどのように変化するかを見てみましょう。



```
mouseY = 94.47236180904522  
mouseX = 308
```

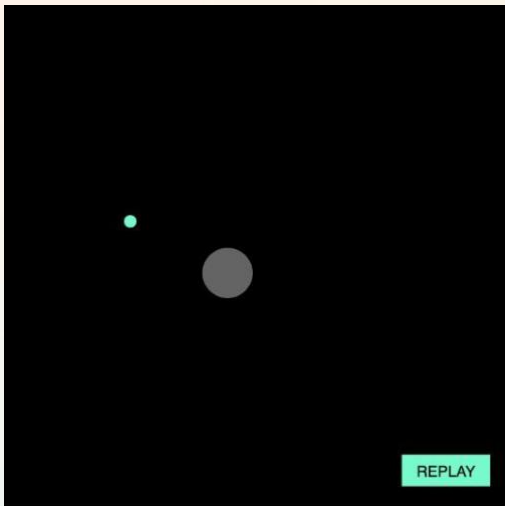
`mouseX` と `mouseY` を使って、キャッチャーの動きをマウスの動きに連動させることができます。つまり、マウスの `x` と `y` の位置はキャッチャーの `x` と `y` に一致することを意味します。それでは、キャッチャーを変更してみましょう。

- ☐ キャッチャーの`ellipse()`関数の`x`の現在値を`mouseX`に置き換える
- ☐ キャッチャーの`ellipse()`関数の`y`の現在値を`mouseY`に置き換える

## ステップ 3: コードのテスト (2-5分)

ここまでに書いたものをテストして、プログラムが思い通りに動くことを確認しましょう。Play ボタンをクリックして、スケッチを実行してください。次のようなことが確認できるでしょう。

- キャッチャーがマウスの動きに連動している
- キャッチャーは白色で半透明である
- Replay ボタンは表示されていない



サンプルスケッチを実行するには[ここ](#)をクリックしてください。

**注:** 流星の挙動をリセットできるように、リプレイボタンを設けました。このボタンがないと流星が画面の下まで落ちると、黒画面だけが表示されることになります。次のパートでは、この点を修正する方法を網羅する予定ですが、今回のパートには含まれていません。

思い通りに動きませんか？デバックのコツを紹介します。

- あなたのコードは正しい波括弧の中にありますか？
- コードの各行の末尾にセミコロンをつけていますか？
- 変数名や関数名のスペルは正しく書けましたか？JavaScriptは大文字と小文字を区別することを忘れないでください
- 関数は正しい位置と順序で配置されていますか？プログラムの流れは順序が重要であることを忘れないでください！
- `ellipse()` 関数の上に、`fill()` 関数が、流星とキャッチャー用に別々に作られていますか？
- キャッチャーが表示されない場合は、 $\alpha$  の値を大きくしてください。
- キャッチャーの `ellipse()` 関数の正しいパラメータ位置に、`mouseX` と `mouseY` を追加しましたか？



3ページ目の参考資料でコードを確認することを忘れないでください。



## ステップ4：理解度の確認

このコードによって、キャッチャーの動作がどのように変わるかを説明してください。

```
ellipse(200, 200, mouseX, mouseY);
```



3ページ目の参考資料でコードを確認することを忘れないでください。

## ステップ5：「キャッチ」の概念化（2分）

前回のステップでは、プレイヤーが、どのようにゲームに関わるかについて学びました。今度は、ゲーム内の、さまざまな構成要素が互いに作用しあう方法を考えましょう。いつキャッチャーが流星を「キャッチ」したのか、いつ流星が画面の底に「落ちた」のかを判断する方法を学習しましょう。

何かをキャッチするとき、実際には何が起こるのでしょうか？友達にボールを投げたり、または、そのボールが床を転がって友達にキャッチしてもらうことを想像してみてください。ボールが相手の体に触れ、相手がそれを掴んだときに、ボールを「キャッチ」したと言います。または、ボールが友達の手や足（前足）に当たった、または、ボールが彼らを横切ったとも言うでしょう。

流星とキャッチャーが交差したかどうかをテストするコードを書きましょう。これは衝突検出とも呼ばれ、2つの図形が接触することを表す言葉です。ここでは、プログラムの中で衝突検出を考慮する方法をいくつか紹介します。

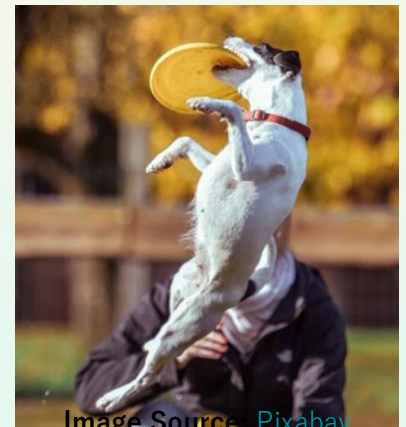
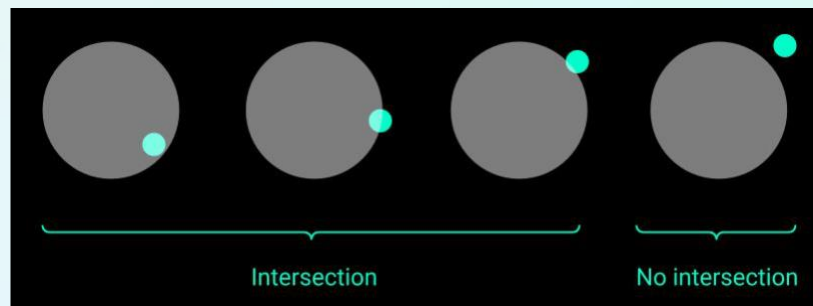


Image Source: [Pixabay](#)



このプログラムでアクセスできる情報のうち、役に立ちそうなものを考えてみましょう。流星のxとyの位置、キャッチャーのxとyの位置、そしてそれぞれの大きさがわかっています。流星の位置は常に変化していますが、それらの値にはxとyの変数から取得することができます。またしても変数の出番です。

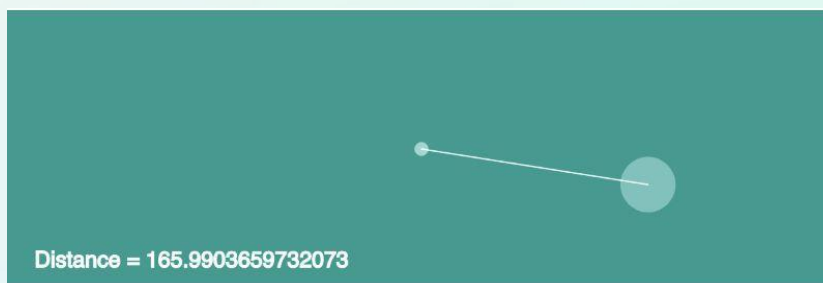
## ステップ6：距離の計算（10分～15分）

### dist()関数の紹介（3分～5分）

上記の情報を使って、キャッチャーが、流星からどれだけ離れているかを計算することができます。p5には、このような計算をしてくれる関数、[dist\(\)](#)関数があります。この関数は2点間の距離を計算します。あとはパラメータを入力するだけです。以下は、構文規則です。

| JAVASCRIPT                         | DESCRIPTION   |
|------------------------------------|---|
| <code>dist(x1, y1, x2, y2);</code> | <ul style="list-style-type: none"><li>→ <b>dist</b>: 関数名</li><li>→ <b>()</b>: 括弧を使用して、プログラムに関数を呼び出す必要があることを伝えます。時には、関数のパラメータや入力を括弧の中に入れることもあります</li><li>→ <b>x1</b>: 1 番目の点の x 座標</li><li>→ <b>y1</b>: 1 番目の点の y 座標</li><li>→ <b>x2</b>: 2 点目の x 座標</li><li>→ <b>y2</b>: 2 点目の y 座標</li><li>→ <b>;</b>: p5.jsのコード行はすべてセミコロンで終わらなければならない</li></ul> |

[デモスケッチ](#)を動かしてみましょう。中央の円とマウスの円の間の**dist()**関数から返された値が表示されています。マウスの円が中心の円と交差するまで動かしてみて、距離の値が変化することを確認してください。



**考えてみましょう：**どの距離であれば「交差」とすると定義しますか？今後、この定義について、また考えてみましょう。その定義に基づいて、"キャッチすること"を追跡できるように条件を書きます。



## ステップ6：距離の計算（続）

### コードに `dist()` を追加する (5-8 分)

プロジェクトのスケッチに `dist()` 関数を追加してみましょう。まず、関数から返される値（キャッチャーの中心と流星の中心との距離）を格納するための変数を作成します。次に、関数を追加します。最後に、新しい関数である `print()` 関数を使用して、距離の値が変化するのを追跡します。

- `setup()` の上に新しい変数を追加して、距離の値を格納します。変数に `distance` という名前をつけましたが、好きな名前にすればよいでしょう。ただ、後でコードの中で正しく参照するようにしてください。この変数に値を代入する必要はありません。
- キャッチャーのコードの下にある `dist()` 関数を呼び出します。このコードが、何をするのか短いコメントを追加しておくといいいでしょう。
- `x1` と `y1` のパラメータに、流星の `x` と `y` の位置を格納する変数を設定します。
- `x2` と `y2` のパラメータを `mouseX` と `mouseY` に設定する。
- `dist()` 関数の結果を `distance` 変数に格納します。



4ページ目の参考資料でコードを確認することを忘れないでください。

## ステップ6：距離の計算（続）

### dist()の値をコンソールに表示する(3-5 分)

これで、プログラムは流星の中心とキャッチャーの中心との距離を常に計算していることになります。しかし、これらの値を確認するにはどうしたらよいのでしょうか？最も簡単な方法は、**print()** 関数です。これは、単語や数字のような英数字の値をエディタの下にあるコンソールに表示します。以下はその構文です。

| JAVASCRIPT                                  | DESCRIPTION   |
|---|---|
| <pre>print('Distance = ' + distance);</pre> | <ul style="list-style-type: none"><li>→ <b>print</b>: コンソールにメッセージを出力する関数名</li><li>→ <b>()</b>: 括弧を使用して、プログラムに関数を呼び出す必要があります。時には、関数のパラメータや入力を括弧の中に入れることもある</li><li>→ <b>"</b>: 一重引用符または二重引用符は、プログラムに対して文字列やテキストを印刷することを伝えます。どちらの引用符を使ってもかまいませんが、一貫性を保つようにする</li><li>→ <b>+</b>: 要素を結合して印刷する</li><li>→ <b>distance</b> を表示する。変数 distance に格納されている現在値を表示する</li><li>→ <b>;</b> p5.js のコード行はすべてセミコロンで終わらなければならない</li></ul> |

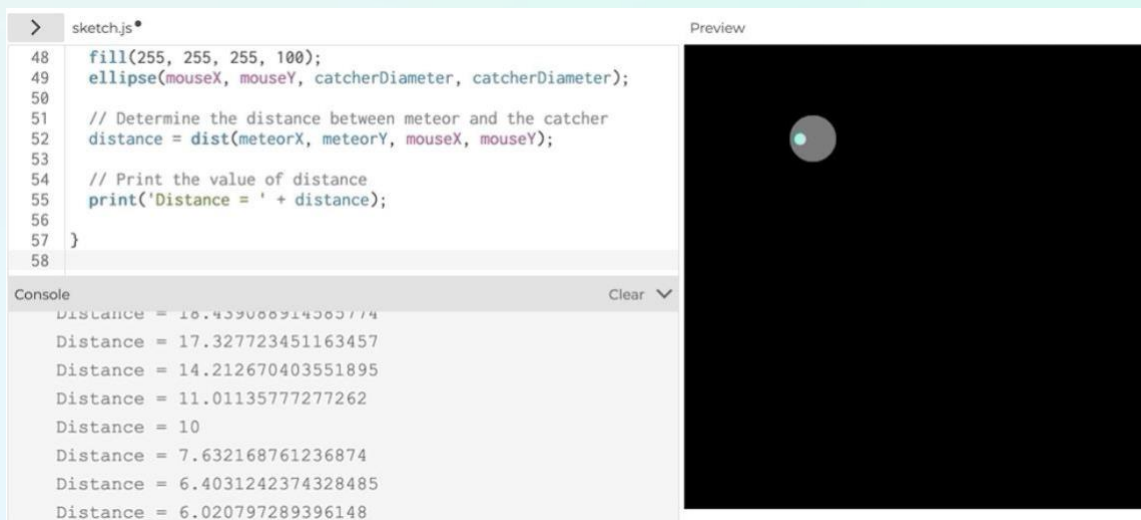
これをスケッチに書き込んでみましょう。

□ **dist()**関数の下に次のコードを追加して、distance 変数の値を出力します。

**print('Distance = ' + distance);**

□ スケッチを実行します。

テキスト**Distance** =と変化する一連の値がコンソールにプリントされるはずです。



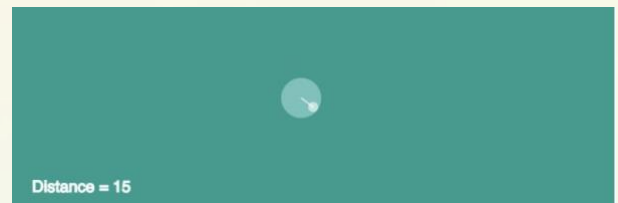
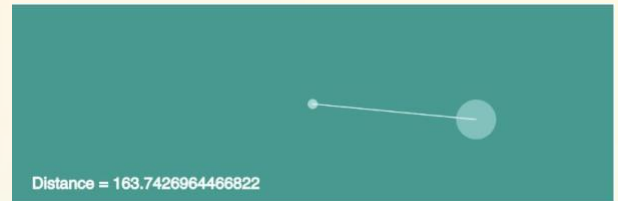
## ステップ7：交差点の判定（5分）

### 距離値を用いた交差点の定義（2～3分）

キャッチャーと流星が交差したときに、プログラムが、ある動作をするようにしましょう。そのためには、交差点を表す数値が必要です。**distance** に格納された値（自分の好きな他の名前を使っている可能性があることを忘れないでください）を使用して、キャッチャーと流星が交差するタイミングを決定します。

先ほどの[距離感のデモスケッチ](#)をもう一度考えてみてください。キャッチャーと流星が重なっている、あるいはほとんど重なっているとき、距離はどのような値になるのでしょうか？

デモスケッチを見てみると、距離が**15**未満であれば「交差」とであると判定できるでしょう。この情報をもとに、条件式を使って、プログラム内での決定をすることができます。



### 条件文の復習（2分）

交差が発生した場合に、何をすべきかをスケッチに指示することができます。そのためには、条件文が必要です。条件文はプログラムの流れを制御することができます。条件文は、ある条件が真か 偽かをチェックするために **if** ステートメントを使用します。もし条件が真であれば、コンピュータは **if** 文の中のコードを実行します。

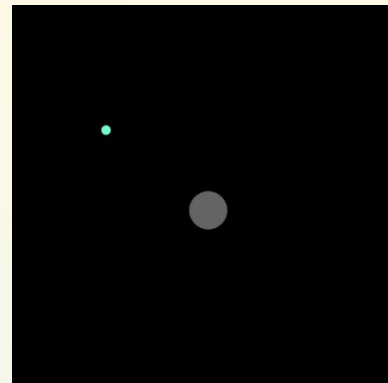
以下の例では、JavaScriptの条件文の構文を説明しています（p5.jsはJavaScript用のライブラリであることを忘れないでください）。これは、マウスのxの位置が100より大きい場合、キャンバスの左上隅に円を描くようにプログラムに指示しています。

| JAVASCRIPT  | DESCRIPTION  |
|---|--|
| <pre>if(mouseX &gt; 100){<br/>  ellipse(0,0,50,50);<br/>}</pre> | <ul style="list-style-type: none"><li>→ <b>if</b>: プログラムに条件付きであることを伝えるためのキーワード</li><li>→ <b>()</b>: その中にあるものが評価する条件の一部であることをプログラムに伝える</li><li>→ <b>mouseX &gt; 100</b>: スケッチが真か偽かを判断するためにテストする条件式。文の設定には、&gt; (より大きい) や &lt;= (以下) などの<a href="#">比較演算子</a>や、   (or) や &amp;&amp; (and) などの<a href="#">論理演算子</a>を使用する</li><li>→ <b>{}</b>: 条件が真である場合に実行するコードの行をプログラムに伝える</li><li>→ <b>ellipse(0,0,50,50)</b>: 条件が「真 (true)」と評価された場合に実行されるステートメントです。ここに他の if 文を入れることも可能</li><li>→ <b>::</b>: p5.jsのコード行はすべてセミコロンで終わらなければならない。</li></ul> |

## ステップ8：キャッチャーの条件設定（5～10分）

### キャッチャーコンディションの計画（2～4分）

最初に作成する条件は、キャッチャーのためのものです。流星とキャッチャーが交差するかどうかをテストする **if** 文を書く必要があります。コードを書く前に、何が起きたいかを考えてみましょう。もう一度ゲームをプレイして、条件分岐の疑似コードを書いてみてください。できるだけ具体的に書いてください。**パート1の計画セクション**で書いた疑似コードを使って、始めることができます。始める前に、[サンプルのスケッチ](#)を確認してください。



**ヒント：**距離の変数と流星の y 位置は、前のステップで作成したものを使用することができます。また、交差点を表す数値も必要です。



参考資料の5ページ目でコードを確認することを忘れないでください

### キャッチャーの条件文の追加（3～5分）

次に、疑似コードを実際のコードに変換します。

- 変数 `distance` の値をチェックする **if** 文を追加する。
- 流星の **y** 位置を更新するコード行を追加します。条件文の中の中括弧の中にあることを確認してください！
- コードを実行します。
- 流星とキャッチャーを交差させてみてください。交差すると、現在の流星が消え、画面上部に「新しい」流星が表示されるはずです。

上記のように動作しなかった場合は、コンソールに出力される距離の値を確認してください。キャッチャーと流星が交差したとき、その値は条件文に指定された範囲に収まっていますか？収まっていない場合は、その数値を変更してください。

**print()**関数は、デバッグにとっても便利です。もし、スケッチの中の変数の値が正しいか確信が持てない場合は、その変数の下に**print()**ステートメントを記述して、ダブルチェックをしてください。

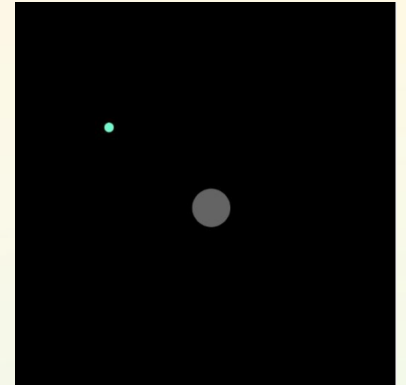


参考資料の5ページ目でコードを確認することを忘れないでください

## ステップ9：画面下部の条件設定（5～10分）

### 画面下部の条件を計画する（2～3分）

では、流星が画面の下と交差したかどうかをテストする条件文を書きましょう。流星が画面の下部と交差した場合、プログラムが流星をキャンパスの上部に再描画するようにしましょう。コード内でキャンパスの高さや幅をすばやく参照したい場合は、数値の代わりにキーワードの **height** や **width** を使用します。始める前に[スケッチの例](#)をチェックしてください。



キャンパスの高さは400ピクセルであることを忘れないでください。

この条件に対する疑似コードを書いてください。以前と同じように、できるだけ具体的に書くようにしましょう。前のステップで書いた疑似コード、または、パート **1** の計画セクションを使用することもできます。



参考資料の5ページ目でコードを確認することを忘れないでください

### スクリーン底面の条件を追加する（3～5分）

では、疑似コードを実際のコードに変換してみましょう。

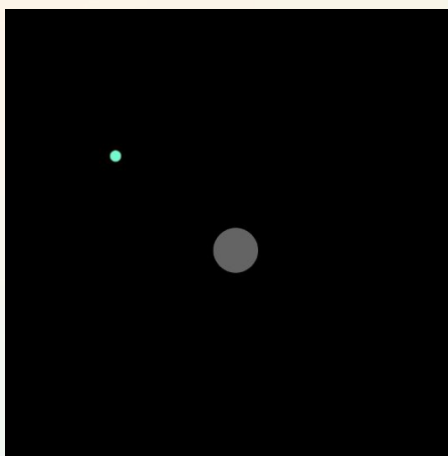
- **if** 文と条件式を書く
- 条件式が真の場合に実行されるステートメントを追加します。条件文の中括弧の中にあることを確認する
- コードを実行する
- 流星が画面の下部と交差するのを待つ。

交差すると、現在の流星が消えて、「新しい」流星が画面の上部に現れるはずです。

## ステップ10: コードのテスト (5-10分)

ここまでに書いたものをテストして、プログラムが計画した通りに動くことを確認しましょう。Play ボタンをクリックして、スケッチを動かしてみましょう。以下のようなことが確認できるでしょう。

- 距離の値がコンソールに表示される
- キャッチャーと流星が交差したときに、流星をキャンパスの上部に再描画する
- 流星とキャンパスの底が交差するとき、流星をキャンパスの上部に再描画する



サンプルスケッチを実行するには、[ここ](#)をクリックしてください。

思い通りに動作しませんか？デバッグのコツをご紹介します。

- あなたのコードは正しい波括弧の中にありますか？
- コードの各行の末尾にセミコロンをつけていますか？
- 変数名や関数名のスペルは正しいですか？
- 関数は正しい位置と順序で配置されていますか？プログラムの流れは順序が重要であることを忘れないでください！
- `dist()` 関数のパラメータは正しいですか？
- `print()` でコンソールに値を出力し、`if` 文のチェックを行います。



参考資料の6ページ目でコードを確認することを忘れないでください



## ステップ11：理解度の確認

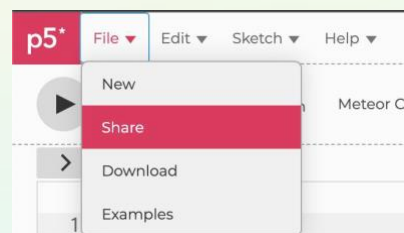
例えば、流星の外縁にかろうじてキャッチャーが触れたときも流星を「キャッチした」ということにしましょう。この場合、最初の条件文の式中の値を増やすべきでしょうか？減らすべきでしょうか？



参考資料の7ページ目で自分のアイデアを確認してください

## ステップ12：Girls Who Code at Home プロジェクトを共有(5 分)

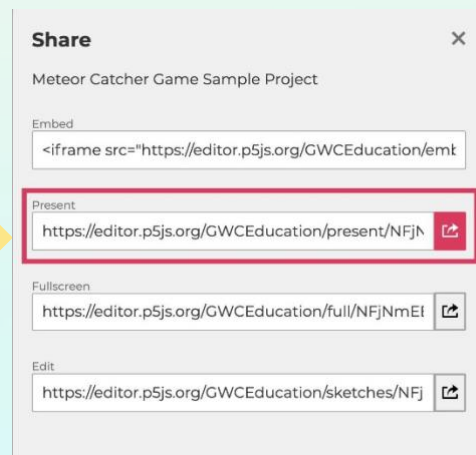
私たちはあなたの作品を見ることを楽しみにしています。また、ほかの人も同じように思っていることでしょう。あなたのゲームを、私たちにシェアしてください。また、[@girlsswhocode](#) [#codefromhome](#)のタグをお忘れなく！私たちのアカウントで紹介するかもしれませんよ！



以下の手順でプロジェクトを共有することができます。

- 最初にプロジェクトを保存してください。
- **ファイル**メニューのドロップダウンメニューから、「共有」を選択します。
- ドロップダウンメニューから「リンク」を選択します。
- プレゼントリンクをコピーして、共有したい場所に貼り付けてください。

Project Link



これからもGirls Who Code at Homeのプロジェクトにご期待ください！

