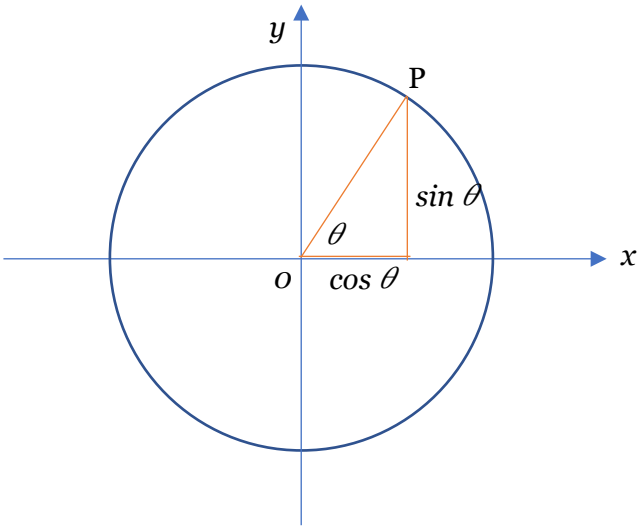
視覚表現のためのプログラミング

「円運動」

■円の座標

円周上を動く点の座標は三角関数から簡単に導くことができる。



半径 1 の単位円上を移動する点 P の座標は

$x=\cos\theta$

$y=\sin\theta$

となります。

三角関数は、高校「数学 I」で「三角比」として学習しています。
円を描くためには、この式を JavaScript で表現すれば良いだけです。

■描画のための要素

まず、どのような要素（パラメータ）があるか考えてみる

- キャンバスの大きさ
- 半径
- 点 P の座標(x,y)
- 角度 θ

※JavaScript の中では角度はラジアン表記になるので、必ず変換が必要になる

これらを JavaScript の言葉で置き換えていきます。

・キャンバスの大きさ

```
<canvas id="cv" width="500" height="500"></canvas>
```

ここでは、幅 500 ピクセル、縦 500 ピクセルのキャンバスを用意しています

・半径

```
var r＝200;
```

半径 r は 200 ピクセル

・点 P の座標(x,y)

```
var x, y;
```

点 P の座標は理解しやすいように x と y をそのまま変数名として使います。

・角度 θ

```
var degree=0;
var radian;
```

角度は degree をそのまま変数名として使います。deg とか略す場合もあります。同じく radian もそのまま変数名として使っています。これも rad と略して使う場合もあります。

・角度をラジアン表記に変換する式

```
radian = degree * Math.PI/180;
```

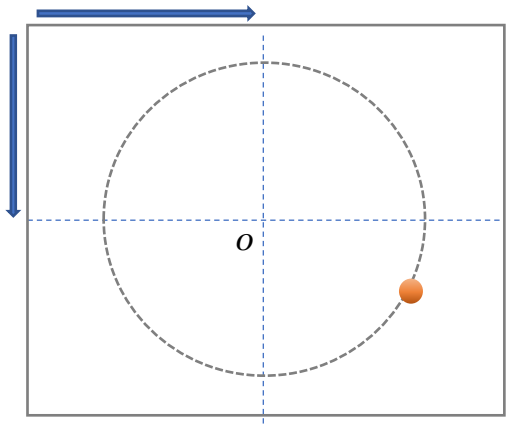
JavaScript の中では、角度は常にラジアン表記なので、度数をラジアンに変換する必要があります。

◆点 P の座標を求める

点 P の各座標は上の図より三角関数でだせる

```
x = r * Math.cos(radian) ;
y = r * Math.sin(radian) ;
```

この点 P の座標を実際に描く場合、キャンバスの中央を円の中心 O にする必要があります。
この場合、円の中心点 O の位置は、キャンバスの横幅、タテのそれぞれ 1/2 の位置になります。



この中心点 O は、座標の原点（左上）から、キャンバスの「幅」または「縦」の 1/2 をそれぞれプラス方向に足します。

すると、上の式は、

```
x = r * Math.cos(radian) + cv.width/2;
y = r * Math.sin(radian) + cv.height/2;
```

■円を描く

単位円の周りを点 P として動く円

半径は 10、赤色

```
context.clearRect(0, 0, cv.width, cv.height);
context.beginPath();
context.fillStyle = 'red';
context.arc(x, y, 10, 0, Math.PI*2);
context.fill();
context.closePath();
```

◆アニメーションとして描く

前回はアニメーションを描くときに時間ごとに処理を呼び出す「setInteval()」を使いました。
今回は、アニメーション描画専用の「requestAnimationFrame()」を使います。名前の通り、アニメーションフレームを要求する命令です。

連続して円を描くために function loop()の中から再帰で呼び出します。

```
function loop(){
  :
  : 円の描画
  :
 requestAnimationFrame(loop);
}
```

このアニメーションをブラウザが読み込まれたときに動作を開始するように、常にブラウザの動作を見続ける（聞き続ける）コマンド「addEventListener」を置きます。これはインタラクティブでよく使われる命令です。

たとえば、マウスのクリックを検知する際などにも使われる。
ここでは、「ブラウザが load されたら、イベントを開始する」ように、その動作を見続けています。

```
window.addEventListener('load', draw, false);
```

draw()は、この描画全体を処理する function です

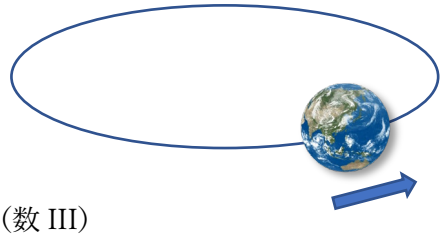
それでは、全体を組み立てます。

```
<canvas   id="cv" width="500" height="500"></canvas>

<script>
 window.addEventListener('load', draw, false);

function draw(){
    var r = 200;
    var x,y;
    var degree = 0;
    var radian;
    var canvas = document.getElementById('cv');
    var context = cv.getContext('2d');

        function loop(){
                degree += 1;
                radian = degree * Math.PI/180;

                x = r * Math.cos(radian) + cv.width/2;
                y = r * Math.sin(radian) + cv.height/2;

                context.clearRect(0, 0, cv.width, cv.height);
                context.beginPath();
                context.fillStyle = 'red';
                context.arc(x, y, 10, 0, Math.PI*2);
                context.fill();
                context.closePath();

                requestAnimationFrame(loop);
        }
        loop();
}
draw();
</script>
```

loop の中で loop 自身を
呼び出している（再帰）

■楕円運動

それでは、楕円に動くものをアニメーションするためにはどうしたら良いでしょうか



「楕円の方程式」があります。(数 III)

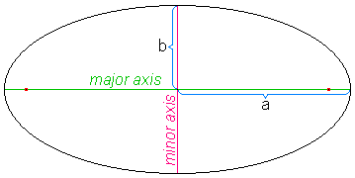
$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$

これを x と y の式になおすと、

$$x=a\cos\theta$$

$$y=b\sin\theta$$

となり、楕円の座標が導けます。



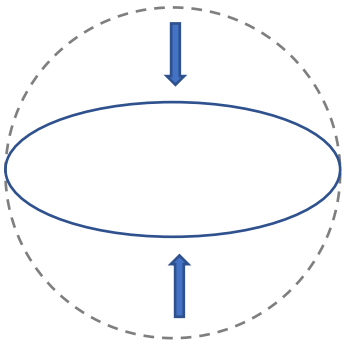
これは**正攻法の解答**です。

科学問題などで厳密に答えを出す際に必要となります。

でも、**即時性**が求められるゲームなどの世界では、計算に時間がかかっていたら敵にやられてしまいます。そこで、正確性が求められない世界では、いかに早く解答を出すかをさまざま考えます。また、楕円の方程式を調べる時間も開発では省きたいものです。

そこで、**アイディア**を絞り出します。

円を縦方向につぶせば「楕円」に見えます。



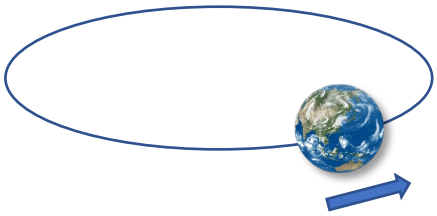
つまり、y 軸方向の大きさを半分にする。式で言えば y の値に 1/2 を掛ければ、単純にこの図形になります。

円のソースコードで言えば、この部分に 1/2 を掛けます。ただ、それだけです。

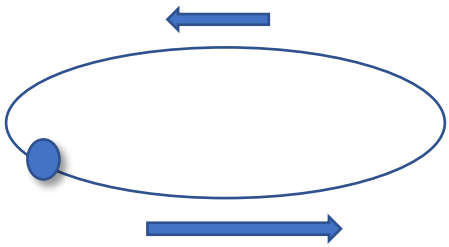
```
y = r * Math.sin(radian) + cv.height/2;
```

■楕円運動の応用（円錐運動）

　たとえば、地球の公転運動を描く際にも応用できます。
つぶれた円運動にすれば、ほとんど奥行きを持った円運動に見えます。



　厳密には、こうした円錐運動を 3 次元で見ているときには、視点との距離の関係で、手前は速く動いて、奥は遅く動きます。



　しかし、簡単なアニメーションではその必要がありません。十分に回転運動をしているように見えます。距離によって「見かけ」の速度変化を表現するには非常に複雑な式が必要になり、その処理に時間がかかります。