情報メディア基盤ユニット用資料（2013 年 4 月 23 日分）

# Processing 言語による情報メディア入門

条件分岐（if 文）

神奈川工科大学情報メディア学科　佐藤尚

## 条件分岐

今までのプログラムでは、並んでいる順番に命令を実行していました。このようは命令の実行の仕方を逐次処理と読んでいます。コンピュータのプログラムの実行の仕方には、この逐次処理を含めて、下の 3 つのものがあります。

1. 逐次処理
2. 条件分岐処理
3. 繰り返し処理

条件分岐処理と繰り返し処理が、コンピュータのプログラムを強く特徴づける処理となっています。これにより､大量のデータ処理や、一見すると複雑な処理が実行できます。ドラクエや Watson のような複雑なシステムも、原理的にはこの 3 つの処理を組み合わせて出来ています。

日常でも､「もしお腹がすいたら何かを食べる、そうでなくもしのどが渇いていたら水を飲む、そうでなければ昼寝をする」(If I am hungry then eat some food, otherwise if I am thirsty, drink some water, otherwise, take a nap) などのような使い方をすることがあります。これに類似したものが条件分岐処理です。つまり、ある条件が満たされているときに実行する処理を指定するのが条件分岐処理の基本的な考え方です。

ここで述べる、条件分岐処理の表し方は、Processing 言語だけでなく、C 言語系の言語ではほぼ共通の書き方（構文）になっています。

C 言語系の例

- C++
- Java
- C#
- JavaScript

Watson って、知っていますか？ IBM という会社が作ったシステムなのです。どのようなものか調べてみて下さい。ちょっと古い話題でしょうか？今だと GPS 将棋？

Processing 言語などのプログラミング言語では、英語が命令文やその組み合わせ方に大きな影響を与えています。英語ネイティブな人の方が有利？

## 条件式（論理式）

Processing 言語をはじめとして、多くのプログラミング言語では、条件分岐において条件式や論理式と呼ばれる考え方が出てきます。つまり、ある条件の時に、ある処理を行ったり、行わなかったりするので、その条件を指定する必要が出てきます。この条件を指定するに利用されるものが条件式です。条件式や論理式と言うと難しく感じるかも知れませんが、簡単にいうと、数学で出ている不等式のようなものです。

条件式の一番単純なものは、次の表に挙げるようなものです。数学での記号と同じ記号を使っているものをありますが、一部異なる記号を使っているものがあります。特に、両辺の値が等しいかどうかを調べるときに使う ==（等値演算子）は =（代入演算子）と勘違いとしやすいので、注意して下さい。

この辺は、数学の不等式や等式同じような扱いとなります。

なんらかの計算を表す記号を演算子 (operator) と呼びます。+ や * も演算子ですし、両辺の値が等しいかという計算をする == も演算子です。演算子に適用する値のことをオペランド (operand) と呼ぶことがあります。

**条件式に出てくる演算子**

| 記号 | 意味 | 使用例 | 使用例の意味 |
|---|---|---|---|
| > | 大きい | mouseX > 100 | mouseX の値が 100 より大きい |
| < | 小さい | mouseY < 200 | mouseY の値が 200 より小さい |
| == | 等しい<br>同じ | mouseButton == LEFT | mouseButton の値が LEFT と等しい |
| >= | 以上 | mouseY >= 100 | mouseY の値が 100 以上 |
| <= | 以下 | mouseX <= width/2 | mouseX の値が width/2 以下 |
| != | 等しくない<br>異なっている<br>同じではない | mousePressed != true | mousePressed の値が true と異なっている |

この表に示した条件式は、変数の値によって、正しい (true) か間違っている (false) かが決まります。そのため、条件式のことを論理式と呼ぶこともあります。

これだけでは、複雑な条件を表すことが出来ないので、単純な条件式を組み合わせて複雑な条件式や論理式を作り出すことで、複雑な条件判定を行います。単純な条件式（論理式）を組み合わせる糊のような役目をするものが論理演算子と呼ばれるものです。ちょっと堅い言い方も知れませんが、日常でのも、「または」とか「かつ」とか言って、少し複雑な条件を表現することがありますよね。この「または」や「かつに」にあたるものが、論理演算子です。

**複雑な条件判定を作る演算子**

| 記号 | 式 | 意味 | 使用例 |
|---|---|---|---|
| && | P && Q | かつ<br>and | (mousePressed == true) && (mouseButton == RIGHT) |
| \|\| | P \|\| Q | または<br>or | (mouseX < width/2) \|\| (mouseY < height/2) |
| ! | ! P | 否定<br>ではない<br>not | !( (mouseX < width/2) \|\| ( mouseY < height/2)) |

**論理式の例**

| | |
|---|---|
| x == 6 | x != y |
| x < 7 | x > 8 |
| x <= 10 | y >= x |
| (7 < x) && (x < 20) | (x == 6) \|\| (y == 8) |
| !(x < 6) | !((x == y) && (y > 8)) |

## 論理式の便利な性質

数式と言うと難しく感じるかもしれませんが、数式にすると機械的な計算で、色々なことがわかることがあります。論理式にも

≧、≦や≠などの記号が使えないので、複数の記号を組み合わせて使っています。「以上」は「大きい」または「等しい」のどちらかですよね。

日本語では、true は真（正しい）、false は偽（正しくない）という意味です。プログラムの作成時には、true や false という値を利用したことが多いので、Processing 言語では、true と false の 2 種類の値を持つ Boolean 型というデータが用意されています。

単純なものから、順々に複雑なものを作り出すことを構成的方法と呼びます。

英語では、「または」は or、「かつ」は and です。

この辺の堅い表現は、履修要項などに出ています。

なれるまでは、複雑な論理式を構成する一つ一つの式を括弧で括った方がわかりやすいと思います。
(mouseX < width/2) || (mouseY < height/2)
と
mouseX < width/2 || mouseY < height/2
を比べてみて下さい。

Processing 言語では、7 < x <20 のような不等式を利用することが出来ないので、&& 演算子を使用して、
(7<x) && (x < 20)
のような書き方をします。

機械的な計算で色々なことをがわかるのが、数式（数学）を利用する、大きなメリットです。

知っていると便利な性質があります。

例えば、数学的には否定の否定はもとの値に戻ります。つまり、Pを論理式とすると、「!(!P)」はPと同じ値になります。このような性質は次の表のようにまとめることが出来ます。表の中では、PやQは論理式とします。

論理式の持っている色々な性質は数理論理学で取り扱われます。

**論理式の持っている性質**

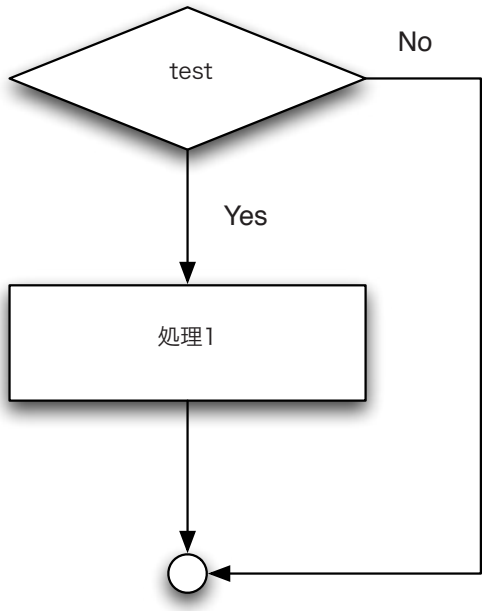
| 元の論理式 | 値の等しい論理式 |
|---|---|
| P && true | P |
| P && false | false |
| P \|\| true | true |
| P \|\| false | P |
| !(!P) | P |
| P == true | P |
| P == false | !P |
| !(P \|\| Q) | (!P) && (!Q) |
| !(P && Q) | (!P) \|\| (!Q) |

表の最後の２つのことをドモルガンの法則と呼びます。

## 一番簡単な条件分岐処理

条件分岐に if という英単語を利用することは、英語圏の人に取っては、自然ですよね。

一番簡単な条件処理は、指定された条件が正しいときに実行する命令を指定するものです。Processing 言語では、条件式を test としたときに、以下のような if という単語を利用して表現されます。

| Processing 言語での表現 | 処理の流れ |
|---|---|
| if(test){<br>処理 1<br>} |  |

右の図では、ひし形の部分で条件式部分を表し、長方形の部分で処理する命令文があることを表しています。このような図のことをフローチャート (flow chart) や流れ図と呼んでいます。
処理の内容を理解するために、使用されることがあります。

サンプル 3-1 は、マウスボタンが押されているときに、黒色（fill(0)）の円を描画する（ellipse(mouseX,mouseY,20,20)）ものです。

**条件分岐処理 (if のみ ) サンプル 3-1**

```
void setup(){
  size(400,400);
  smooth();
}
```

{ と } で囲まれている部分が一つの塊（ブロック）を作っています。

```
void draw(){
  background(255);
  if(mousePressed == true){
    fill(0);
    ellipse(mouseX,mouseY,20,20);
  }
}
```

サンプル 3-1 の中では、“if(test){” の test の部分には、条件式と呼ばれるものが書かれます。サンプル 3-1 では、この条件式にあたる部分は、”mousePressed == true” です。この条件式の意味は、mousePressed という変数の値が true と等しい (==) というものです。

サンプル 3-2 もサンプル 3-1 と同じような構造のプログラムです。プログラムを見て、どのような動作のプログラムかわかりますか？打ち込んで、実行して見ましょう。

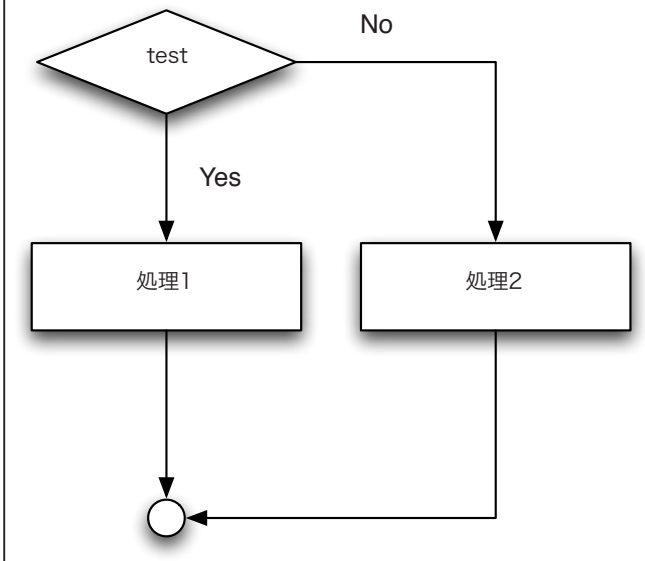
**条件分岐処理 (if のみ ) サンプル 3-2**

```
void setup(){
  size(400,400);
}

void draw(){
  background(255);
  if(mouseX < width/2){
    fill(0);
    rect(0,0,width/2,height);
  }
}
```

## もう少し複雑な条件分岐処理 (A or B)

もう少し複雑な条件分岐処理は、指定された条件が正しいときに実行する命令と、その条件が正しくないときに実行する命令を指定するものです。Processing 言語では、このような処理は if と else という英単語を利用した文として表現されます。

システム変数 mousePressed は、マウスボタンが押されている時には値が true となり、そうでないときには値が false となります。つまり、データ型が Boolean となっている変数です。このような Boolean 型の変数を論理 (型) 変数と呼ぶことがあります。

| Processing 言語での表現 | 処理の流れ |
| --- | --- |
| if(test){<br>処理 1<br>}else{<br>処理 2<br>} |  |

右の図では、条件式 test が true の時を Yes、false の時を No で表しています。

サンプル 3-3 は、マウスボタンが押されているときには黒色（fill(0)）を、そうでない場合には灰色 (fill(170)) の円を描画する（ellipse(mouseX,mouseY,20,20)）ものです。

**条件分岐処理 (if と else) サンプル 3-3**

```
void setup(){
  size(400,400);
  smooth();
}

void draw(){
  background(255);
  if(mousePressed == true){
    fill(0);
  }else{
    fill(170);
  }
  ellipse(mouseX,mouseY,20,20);
}
```

サンプル 3-4 もサンプル 3-3 と同じような構造のプログラムです。プログラムを見て、どのような動作のプログラムかわかりますか？打ち込んで、実行して見ましょう。このサンプルでは、条件分岐のための条件が「mouseX<width/2」となっています。この条件はどのような条件になっているか、わかりますか？これが OK なら、このプログラムの動作の理解も簡単だと思います。

**条件分岐処理 (if と else) サンプル 3-4**

```
void setup(){
  size(400,400);
}

void draw(){
  background(255);
  fill(0);
  if(mouseX < width/2){
    rect(0,0,width/2,height);
  }else{
    rect(width/2,0,width/2,height);
  }
}
```

サンプル 3-1 やサンプル 3-3 での条件式は、「mousePressed == true」となっています。システム変数 mousePressed は true か false のどちらかの値をとる論理変数です。従って、「論理式の持っている性質」を使うと、mousePressed だけで良いことになります。従って、サンプル 3-3 は次の様に書き換えることが出来ます。

論理式は true か false のどちらかに値が決まればよいので、mousePressed のように論理変数単体でも、論理式となります。

**条件分岐処理 (if と else) サンプル 3-3'**

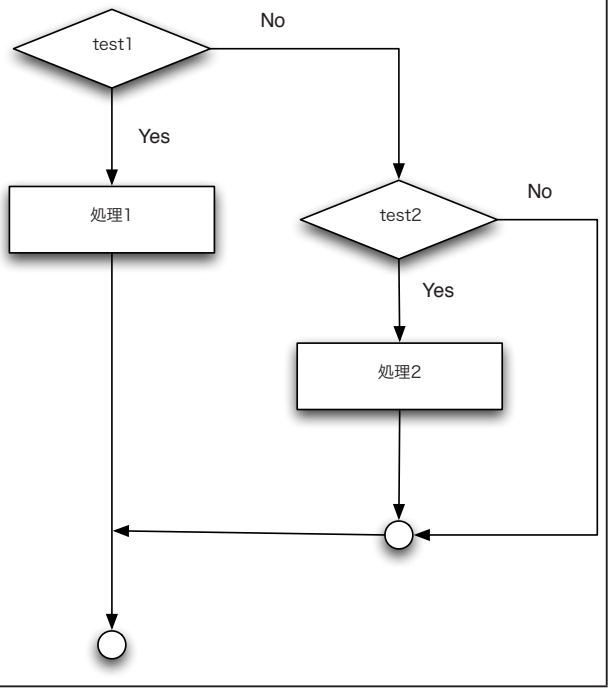
```
void setup(){
  size(400,400);
  smooth();
}

void draw(){
  background(255);
  if(mousePressed){
    fill(0);
  }else{
    fill(170);
  }
  ellipse(mouseX,mouseY,20,20);
}
```

「mousePressed == true」 と書くか、単に「mousePressed」と書くかは、どちらでも良いと思います。個人的は、シンプルな後者の方が好きです。

## 条件分岐処理の重ね ( その 1)

条件分岐処理は、重ねて処理をすることが出来ます。つまり、条件を順番に試していき、実行すべき命令を決めることが出来ます。

このような処理は、次に示す、if と else if という英単語の組み合わせで表現します。

| Processin 言語の表現 | 処理の流れ |
| --- | --- |
| if(test1){<br>処理 1<br>}else if(test2){<br>処理 2<br>} |  |

サンプル 3-5 は、マウスボタンが押されていない時 (mousePressed == false) には、中を塗りつぶさない（noFill()）で円を描画 (ellipse(width/2,height/2,100,100)) します。そうでない時（” マウスボタンが押されていないとき ” ではないとき、つまりつまりマウスボタンが押されているとき）に、押されているボタンが右ボタン（mouseButton == RIGHT）であれば、赤色 (fill(250,20,20)) の円を描画 (ellipse(width/2,height/2,100,100)) します。ところで、サンプル 3-5 を実行しているときに、マウスの
左ボタンを押したら、どのような表示になるかわかりますか？

システム変数 mouseButton は、LEFT、CENTER、RIGHT のどれかの値を取ります。どのマウスボタンが押されているかによって、値が変わります。

**条件分岐処理 (if と else if) サンプル 3-5**

```
void setup(){
  size(400,400);
  smooth();
}

void draw(){
  background(255);
  if(mousePressed == false){
    noFill();
    ellipse(width/2,height/2,100,100);
  }else if(mouseButton == RIGHT){
    fill(250,20,20);
    ellipse(width/2,height/2,100,100);
  }
}
```

サンプル 3-6 もサンプル 3-5 と同じような構造のプログラムです。プログラムを見て、どのような動作のプログラムかわかりますか？打ち込んで、実行して見ましょう。

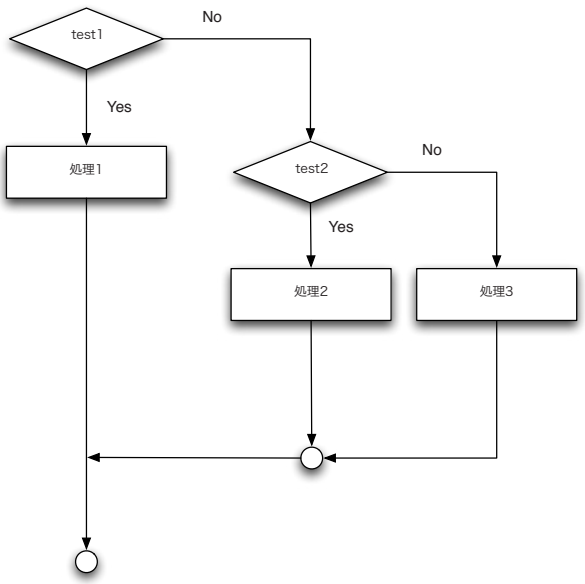
**条件分岐処理 (if と else if) サンプル 3-6**

```
void setup(){
  size(300,300);
}

void draw(){
  background(255);
  line(width/3,0,width/3,height);
  line(2*width/3,0,2*width/3,height);
  fill(0);
  if(mouseX < width/3){
    rect(0,0,width/3,height);
  }else if(mouseX > 2*width/3){
    rect(2*width/3,0,width/3,height);
  }
}
```

このサンプルのように、マウスカーソルの位置に応じて、表示を行うことをロールオーバー処理と呼びます。

## 条件分岐処理の重ね ( その 2)

条件分岐処理は、もっと重ねて処理をすることが出来ます。基本的には好きなだけ if ～ else という構造をつなげていき、沢山の条件を試していくことが出来ます。ここでは、2 つの条件式を持つ場合を示します。始めに条件式 test1 の条件が成立しているどうかを調べます。

| Processing 言語での表現 | 処理の流れ |
|---|---|
| if(test1){<br>  処理 1<br>}else if(test2){<br>  処理 2<br>}else{<br>  処理 3<br>} |  |

条件式の値が true となる場合は、条件が成立していると言うことがあります。false となる場合には、条件が不成立と呼ぶこともあります。

サンプル 3-7 は、この条件分岐処理の重ねの例題です。

### 条件分岐処理 (if と else if の多段の重ね ) サンプル 3-7

```
void setup(){
  size(400,400);
  smooth();
}

void draw(){
  background(255);
  if(mousePressed == false){
    noFill();
  }else if(mouseButton == LEFT){
    fill(250,20,20);
  }else{
    fill(20,250,20);
  }
  ellipse(width/2,height/2,100,100);
}
```

サンプル 3-7 は、マウスボタンが押されていない時 (mousePressed == false) には、中を塗りつぶさない（noFill()）で、そうでない時 (" マウスボタンが押されていないとき " ではないとき、つまりマウスボタンが押されているとき）に、押されているボタンが左ボタン（mouseButton == LEFT）であれば赤色 (fill(250,20,20))、そうでないときには緑色 (fill(20,250,20)) で、円を描画 (ellipse(width/2,height/2,100,100)) します。

サンプル 3-8 もサンプル 3-7 と同じような構造のプログラムです。プログラムを見て、どのような動作のプログラムかわかりますか？打ち込んで、実行して見ましょう。

条件が複雑になってくると、日本語で説明することが、段々面倒になってきます。そのため、Processing 言語などのプログラミング言語や、フローチャートのような処理の流れを図式化（可視化）する手法を利用して表現（記述）することが重要になってきます。

### 条件分岐処理 (if と else if の多段の重ね ) サンプル 3-8

```
int xLeft;
void setup(){
  size(300,300);
}
void draw(){
  background(255);
  line(width/3,0,width/3,height);
  line(2*width/3,0,2*width/3,height);
  fill(0);
  if(mouseX < width/3){
    xLeft = 0;
  }else if(mouseX < 2*width/3){
    xLeft = width/3;
  }else{
    xLeft = 2*width/3;
  }
  rect(xLeft,0,width/3,height);
}
```

サンプル 3-7 とサンプル 3-8 は、2 つの条件を重ねたものです。これは 2 つの条件に限らず、もっと増やすことが出来ます。ここでは、もう 1 つ条件を加えた、同じような動作をするサンプルを示します。

### 条件分岐処理 (if と else if の多段の重ね ) サンプル 3-9

```
int xLeft;
int w4; // width の 4 分の 1 の値を保存する

void setup(){
  size(300,300);
  w4 = width/4;
}

void draw(){
  background(255);
  line(w4,0,w4,height);
  line(2*w4,0,2*w4,height);
  line(3*w4,0,3*w4,height);
  fill(0);
  if(mouseX < w4){
    xLeft = 0;
  }else if(mouseX < 2*w4){
    xLeft = w4;
  }else if(mouseX < 3*w4){
    xLeft = 2*w4;
  }else{
    xLeft = 3*w4;
  }
  rect(xLeft,0,w4,height);
}
```

width/4 という式が沢山出てくるので、int 型変数 w4 に width/4 の値を保存しています。

今までに説明したことを利用して作ったサンプルを示します。このプログラムでやっていることは、

setup での処理

1. ウインドウを表示する

draw での処理

1. 背景を白にする
2. 真ん中に十字上に線を表示する
3. もしマウスカーソルが左上のコーナーにいれば、そのコーナーに黒色の矩形を表示する。
4. もしマウスカーソルが右上のコーナーにいれば、そのコーナーに黒色の矩形を表示する。
5. もしマウスカーソルが左下のコーナーにいれば、そのコーナーに黒色の矩形を表示する。
6. もしマウスカーソルが右下のコーナーにいれば、そのコーナーに黒色の矩形を表示する。

**条件分岐処理 (if と else if の多段の重ね ) サンプル 3-10**

```
void setup(){
  size(400,400);
}

void draw(){
  background(255);
  line(width/2,0,width/2,height);
  line(0,height/2,width,height/2);
  fill(0);
  if(mouseX < width/2 && mouseY < height/2){
    rect(0,0,width/2,height/2);
  }else if(mouseX >= width/2 && mouseY < height/2){
    rect(width/2,0,width/2,height/2);
  }else if(mouseX < width/2 && mouseY >= height/2){
    rect(0,height/2,width/2,height/2);
  }else{
    rect(width/2,height/2,width/2,height/2);
  }
}
```

## 少し意味のあるサンプルその 1

FF などのゲームで世界地図上を移動する際に、一番下まで移動すると一番真上に現れます。次のサンプル 3-11 は、これと同じように、移動する直線が一番下まで移動したら、その次は一番上から出てくるようにするプログラムです。X 軸に平行な移動する直線が一番下に到達したら、この直線を一番上に移動させています。int 型変数 y が、直線を描く高さ（Y 座標の値）を表しています。

**条件分岐処理を利用した直線の移動 サンプル 3-11**

```
int y;// 水平な直線を描くときの、y 座標の値を保存している変数

void setup(){
  size(100,300);
  y = 0;
}

void draw(){
  background(255);
  stroke(0);
  line(0,y,width,y);
  y = y+1;
  if(y >= height){
    y = 0;
  }
}
```

## 少し意味のあるサンプルその 2

条件分岐処理を利用すると、壁に円がぶつかったときに跳ね返るような処理を作り出すことが出来ます。これを実現したプログラムが次のサンプルプログラムです。

**条件分岐処理を利用した壁での反射サンプル 3-12**

```
int xCenter; // 円の中心の X 座標
int radius;  // 円の半径
int speed;   // 円の移動速度。正の値の時は、左から右に移動する。
             // 負の時は、右から左に移動する。

void setup(){
  size(400,200);
  smooth();
  xCenter = width/2;
  radius = 20;
  speed = 1;
}

void draw(){
  background(255);
  fill(170);
  ellipse(xCenter,height/2,2*radius,2*radius);
  xCenter = xCenter + speed;
  if(((xCenter + radius) >= width) || ((xCenter-radius) < 0)){
    speed = -speed;
  }
}
```

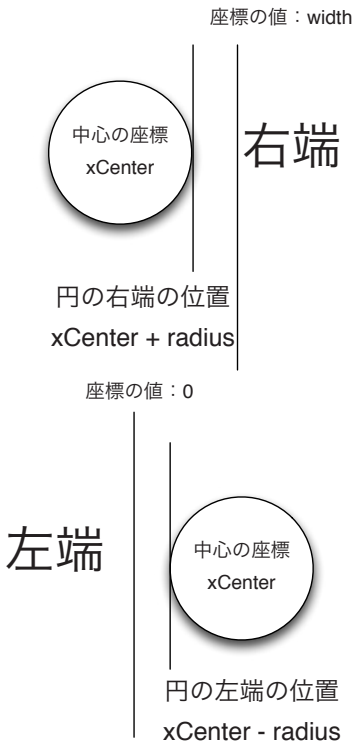
移動する円は、左端と右端に到達したときに移動方向を変えれば、両端の壁にぶつかって移動する動作を再現出来ます。円が左端に到達した場合には、円の一番左側の位置が 0 よりも小さくなっているはずです。また、円が右端に到達した場合には、円の一番右側の位置が width 以上になっています。この 2 つの条件のどちらかが成立している時に、移動方向を変更します。移動方向を正反対にするためには、移動速度に -1 をかければ OK です。

円が両端に到達した時の移動方向の変更の処理が難しいと感じる人は、サンプル 3-12' のように明示的に変数 speed の値を変更した方が理解しやすいかも知れません。

**条件分岐処理を利用した壁での反射サンプル 3-12'**

```
int xCenter; // 円の中心の X 座標
int radius;  // 円の半径
int speed;   // 円の移動速度。正の値の時は、左から右に移動する。
             // 負の時は、右から左に移動する。
```

このプログラムを複雑にしていくと、Pong のようなゲームを作ることが出来ます。上手くゲームをデザインすれば、今までの知識でもゲームを作ることが出来ます。



**円板が壁にあたる時**

速度がベクトルで表されている場合は、移動方向を正反対にするためには、-1 をかければ求めることが出来ます。

このサンプル 3-12' では、左端でぶつかるか、右端でぶつかるかで処理を変えています。

```
void setup(){
  size(400,200);
  smooth();
  xCenter = width/2;
  radius = 20;
  speed = 1;
}

void draw(){
  background(255);
  fill(170);
  ellipse(xCenter,height/2,2*radius,2*radius);
  xCenter = xCenter + speed;
  if((xCenter + radius) >= width){   // 左端で衝突
    speed = -1;
  }else if((xCenter-radius) < 0){    // 右端で衝突
    speed = 1;
  }
}
```

## 変数の値の変更

サンプル 3-11 やサンプル 3-12 などでは、「y=y+1;」や「xCenter = xCenter + speed;」などのように、右辺で計算した値を左辺の変数に代入する形の式が出てきます。これらの式でやりたいことをよく考えると、「変数の y の値を 1 増やす」や「変数 xCenter の値を speed だけ増やす」などになります。このような、「変数の値を増やす（減らす）」などの処理は、プログラム中で良く使用される式となっています。このような、意図をハッキリさせるために、Processing 言語では特別な式の書き方が用意されています。これを、次の表にまとめておきます。

「意図」は、人間にとっても、コンピュータ側にとっても、有効な情報になっています。

C 言語系のプログラミング言語でも同じ機能が提供されています。

**複合代入演算子、増分演算子、減分演算子**

| 普通の記法 | 意図を発揮させた記法 | 記法の使用例 | |
|---|---|---|---|
| 変数 = 変数 + 値 ; | 変数 += 値 ; | x = x+3; | x += 3; |
| 変数 = 変数 * 値 ; | 変数 *= 値 ; | x = x*5; | x *= 5; |
| 変数 = 変数 - 値 ; | 変数 -= 値 ; | x = x-2; | x -= 2; |
| 変数 = 変数 / 値 ; | 変数 /= 値 ; | x = x / 7; | x /= 7; |
| 変数 = 変数 % 値 ; | 変数 %= 値 ; | x = x % 5; | x %= 5; |
| 変数 = 変数 + 1; | 変数 ++; | x = x+1; | x++; |
| 変数 = 変数 + 1; | ++ 変数 ; | x = x+1; | ++x; |
| 変数 = 変数 -1; | 変数 --; | x = x-1; | x--; |
| 変数 = 変数 -1; | -- 変数 ; | x = x-1; | --x; |

この表の上の 5 つは複合代入演算子と呼ばれています。

++ は増分演算子、-- は減分演算子と呼ばれています。正確には、++ 変数を前置増分演算子、変数 ++ を後置増分演算子と呼んで区別をしています。この 2 つは良く似ていますが、少しだけ挙動が異なります。-- も同じです。

サンプル 3-11 とサンプル 3-12 を、増分演算子や複合代入演算子を使用して書きかえたものを示します。とめておきます。

### 条件分岐処理を利用した直線の移動 サンプル 3-11'

```
int y;

void setup(){
  size(100,300);
  y = 0;
}

void draw(){
  background(255);
  stroke(0);
  line(0,y,width,y);
  y++;  // この部分を増分演算子で書きかえ
  if(y >= height){
    y = 0;
  }
}
```

### 条件分岐処理を利用した壁での反射サンプル 3-12"

```
int xCenter;
int radius;
int speed;

void setup(){
  size(400,200);
  smooth();
  xCenter = width/2;
  radius = 20;
  speed = 1;
}

void draw(){
  background(255);
  fill(170);
  ellipse(xCenter,height/2,2*radius,2*radius);
  xCenter += speed; // この部分を複合代入演算子で書きかえ
  if(((xCenter + radius) >= width) || ((xCenter-radius) < 0)){
    speed = -speed;
  }
}
```