

```
int radius = 45;      // 頭の半径
int ny = y - bodyHeight - neckHeight - radius; // 首のy

size(170, 480);
strokeWeight(2);
background(204);
ellipseMode(RADIUS);

// 首
stroke(102);
line(x+2, y-bodyHeight, x+2, ny);
line(x+12, y-bodyHeight, x+12, ny);
line(x+22, y-bodyHeight, x+22, ny);
// アンテナ
line(x+12, ny, x-18, ny-43);
line(x+12, ny, x+42, ny-99);
line(x+12, ny, x+78, ny+15);
// 胴体
noStroke();
fill(102);
ellipse(x, y-33, 33, 33);
fill(0);
rect(x-45, y-bodyHeight, 90, bodyHeight-33);
fill(102);
rect(x-45, y-bodyHeight+17, 90, 6);
// 頭
fill(0);
ellipse(x+12, ny, radius, radius);
fill(255);
ellipse(x+24, ny-6, 14, 14);
fill(0);
ellipse(x+24, ny-6, 3, 3);
fill(153);
ellipse(x, ny-8, 5, 5);
ellipse(x+30, ny-26, 4, 4);
ellipse(x+41, ny+6, 3, 3);
```

# 5

## 反応

Response

マウスやキーボードといったデバイスからの入力に反応するプログラムは止まらずに動き続けている必要があります。そのためには、draw()関数のなかにコードを配置します。

### Example 5-1:draw() 関数

draw() がどう作用するか、次の例を実行して確かめてください。

```
void draw() {  
    // フレームカウントをコンソールに表示  
    println("I'm drawing");  
    println(frameCount);  
}
```

結果は以下のようになります。

```
I'm drawing  
1  
I'm drawing  
2  
I'm drawing  
3  
...
```

draw() ブロック内のコードは、Stop ボタンを押すかウィンドウを閉じてプログラムを停止するまで、何度も繰り返し実行されます。ブロック内のコードが上から下へと実行されていくて一巡する期間をフレームといいます。デフォルトは1秒間に60フレームで、この値は変更可能です(Example 7-2を参照)。先ほどの例では、“I'm drawing”というメッセージに続いて、frameCount 変数の値(1, 2, 3……)がprintln() 関数によって表示されました。これらのテキストが表示されたエディタの下にある黒い領域がコンソールです。

### Example 5-2:setup() 関数

ループ状態の draw() を補完する存在が setup() です。setup() 関数はプログラムが動き始めたときに、1回だけ実行されます。

```
void setup() {  
    println("I'm starting");  
}  
  
void draw() {  
    println("I'm running");  
}
```

このコードを実行すると、コンソールには次のように出力されます。

```
I'm starting  
I'm running  
I'm running  
I'm running  
...
```

プログラムを止めるまで、“I'm running”というテキストが出続けます。

典型的なプログラムでは、setup() 内のコードは開始時の値を定義するために使われます。多くの場合、最初の1行はsize() 関数で、その後に、塗り色や線の色の設定、画像やフォントの読み込みといった処理が続きます。

setup() と draw() の使い方は分かったと思いますが、コードを配置できる場所はもう1か所あります。setup() と draw() の外側に変数を置くことができるのです。setup() の中に宣言された変数は draw() のなかでは使えないで、両方の関数で使いたい変数は関数の外で宣言します。このような変数をグローバル変数といい、プログラム中のどこででも使うことができます。

Processing がコードを実行する順番を整理すると、次のとおりです。

1. setup() と draw() の外側で宣言された変数を作成
2. setup() 内のコードを1度だけ実行
3. draw() 内のコードを繰り返し実行

### Example 5-3:setup() と draw()

次の例はすべての要素を含んでいます。

```
int x = 280;  
int y = -100;  
int diameter = 380;  
  
void setup() {  
    size(480, 120);  
    fill(102);  
}
```

```
void draw() {
    background(204);
    ellipse(x, y, diameter, diameter);
}
```

## 追いかける

動き続けるコードができたところで、次はマウスの位置を追跡し、その情報を使って画面上の図形を動かしてみましょう。

### Example 5-4: マウスを追跡

mouseX変数はx座標、mouseY変数はy座標を保持しています。



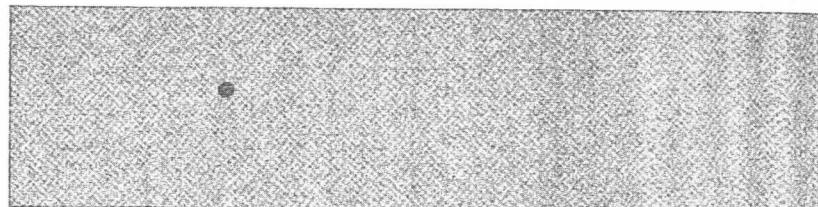
```
void setup() {
    size(480, 120);
    fill(0, 102);
    noStroke();
}

void draw() {
    ellipse(mouseX, mouseY, 9, 9);
}
```

draw()の中のコードが実行されるたびに、新しい円がひとつ描かれます。上の画像は、マウスをあちこち移動させて円の位置を変えた結果できたものです。塗り(fill)を半透明に設定しているので、マウスをゆっくり動かした部分は円が重なって濃く見えます。円が飛び飛びになっているところは、マウスを速く動かした区間です。

### Example 5-5: 追ってくる点

次のプログラムも、draw()が実行されるたびに新しい円がひとつ描かれる、先ほどとほぼ同じ内容です。違うのは1か所だけ。最新の円だけが画面上に残るよう、draw()の先頭でbackground()関数を実行して、画面をリフレッシュしています。



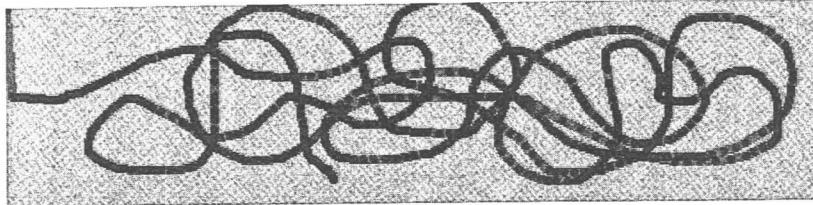
```
void setup() {
    size(480, 120);
    fill(0, 102);
    noStroke();
}

void draw() {
    background(204);
    ellipse(mouseX, mouseY, 9, 9);
}
```

background()関数はウィンドウ全体をクリアします。draw()の中では他の関数よりも前に置かないと、描いた图形が消えてしまいます。

### Example 5-6: 連続的に描く

pmouseX変数とpmouseY変数は、ひとつ前のフレームにおけるマウスの位置を記憶しています。mouseXやmouseYと同様に、draw()が実行されるたびに値が更新されます。これらの変数を組み合わせると、現在の位置と前回の位置がつながっている、切れ目のない線を描くことができます。



```
void setup() {  
    size(480, 120);  
    strokeWeight(4);  
    stroke(0, 102);  
}  
  
void draw() {  
    line(mouseX, mouseY, pmouseX, pmouseY);  
}
```

#### Example 5-7: 太さを変えながら描く

pmouseXとpmouseYはマウスのスピードの計算にも使えます。現在のマウス位置と前回のマウス位置から移動距離を求めれば、(フレームの更新時間は一定なので)それがスピードを表しています。マウスがゆっくり移動しているとき、この距離は縮まります。逆にマウスが速く動くと、距離は広がります。次の例に登場するdist()はこの計算を簡単にしてくれる関数です。dist()を使って得た値をstrokeWeight()に渡すことで、スピードを線の太さに変えます。



```
void setup() {  
    size(480, 120);  
    stroke(0, 102);  
}  
  
void draw() {  
    float weight = dist(mouseX, mouseY, pmouseX, pmouseY);  
    strokeWeight(weight);  
    line(mouseX, mouseY, pmouseX, pmouseY);  
}
```

#### Example 5-8: ゆっくり行こう

Example 5-7では、マウスから得た値をそのまま画面上の位置に変換しました。この方法では動きが速すぎると感じたかもしれません。より滑らかなモーションを作り出すために、反応を遅らせてみましょう。このテクニックはイージング(easing)と呼ばれます。イージングは現在地と目的地を示す2つの値を使います(図5-1)。1フレーム進むごとに、現在地は目的地へ少しづつ近づいていきます。

```
float x;  
float easing = 0.01;  
float diameter = 12;  
  
void setup() {  
    size(220, 120);  
}  
  
void draw() {  
    float targetX = mouseX;  
    x += (targetX - x) * easing;  
    ellipse(x, 40, diameter, diameter);  
    println(targetX + " : " + x);  
}
```

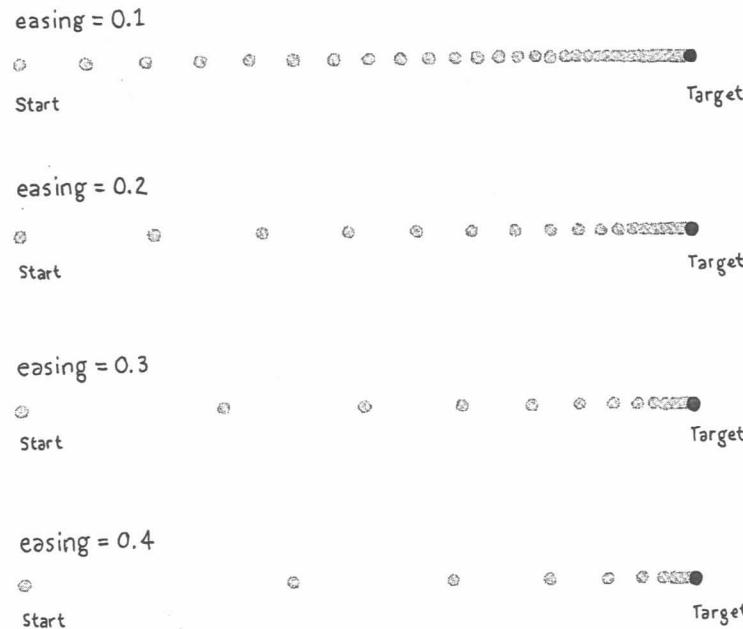
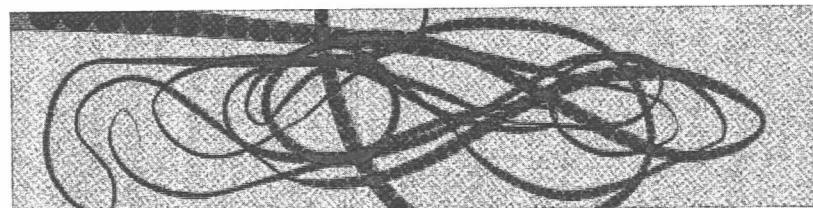


図5-1 イージング

このプログラムの一番重要な部分は `x +=` で始まる1行です。まず、現在地と目的地の差を計算し、そこに easing 変数を掛けてから `x` に加えることで目的地へと近づけます。

#### Example 5-9: イージングで線を滑らかに

次の例はイージングを Example 5-7 に適用したものです。比べると、線がより滑らかになっているのがわかるでしょう。



```
float x;
float y;
float px;
```

```
float py;
float easing = 0.05;

void setup() {
    size(480, 120);
    stroke(0, 102);
}

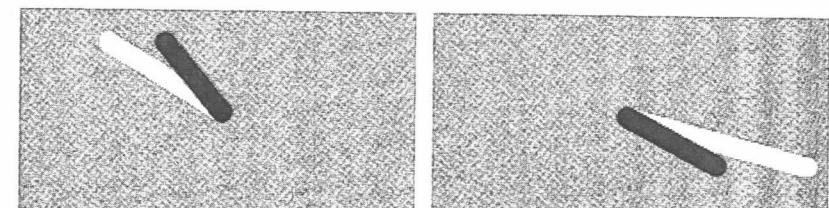
void draw() {
    float targetX = mouseX;
    x += (targetX - x) * easing;
    float targetY = mouseY;
    y += (targetY - y) * easing;
    float weight = dist(x, y, px, py);
    strokeWeight(weight);
    line(x, y, px, py);
    py = y;
    px = x;
}
```

## マッピング

図形を描くために数値を扱っていると、ある範囲から別の範囲へ変換したくなることがあります。

#### Example 5-10: 値の範囲を変更

通常、`mouseX`変数の範囲は0とウィンドウ幅の間です。しかし、値を異なる範囲に変換して使いたい時があるかもしれません。その場合、`mouseX`をなんらかの値で割ってからまた別の値を加える（小さくしてからずらす）という計算が必要です。



```

void setup() {
    size(240, 120);
    strokeWeight(12);
}

void draw() {
    background(204);
    stroke(255);
    line(120, 60, mouseX, mouseY); // 白い線
    stroke(0);
    float mx = mouseX/2 + 60;
    line(120, 60, mx, mouseY); // 黒い線
}

```

この種の変換を行うもっと良い手段が `map()` 関数です。値がある範囲から別の範囲へ移します。`map()` の1つ目のパラメータは変換したい変数、2つ目と3つ目はその変数の最小値と最大値、4つ目と5つ目は変換後の最小値と最大値です。`map()` 関数によって面倒な計算を省略でき、コードが分かりやすくなります。

#### Example 5-11: `map()` 関数でマッピング

このプログラムは Example 5-10 を `map()` を使って書き直したものです。

```

void setup() {
    size(240, 120);
    strokeWeight(12);
}

void draw() {
    background(204);
    stroke(255);
    line(120, 60, mouseX, mouseY); // 白い線
    stroke(0);
    float mx = map(mouseX, 0, width, 60, 180);
    line(120, 60, mx, mouseY); // 黒い線
}

```

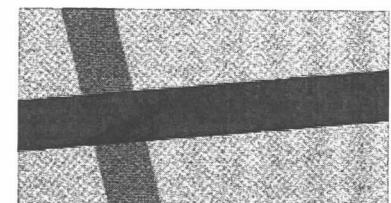
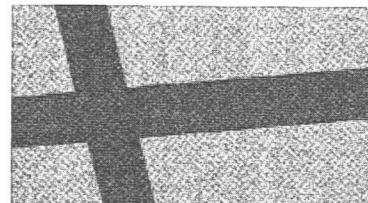
`map()` 関数を使うことで、ある変数の最小値と最大値が明確になり、その結果、コードが理解しやすくなります。この例では、`mouseX` の元の値(0からwidth)が 60 から 180 の範囲にコンバートされます。このあと登場する多くの例で、`map()` 関数が有効に使われています。

## クリック

Processing はマウスの位置だけでなく、マウスボタンの状態も把握しています。ボタンが押されると、`mousePressed` 変数が変化します。この変数の型はブーリアン型と呼ばれ、取り得る値は真(true) か偽(false) のどちらか一方です。ボタンが押されると、`mousePressed` は真となります。

#### Example 5-12: マウスをクリック

`mousePress` 変数を `if` 文と一緒に使って、あるコードを実行すべきかそうでないかを判定します。詳しい説明の前に、このコードを試してください。



```

void setup() {
    size(240, 120);
    strokeWeight(30);
}

```

```

void draw() {
    background(204);
    stroke(102);
    line(40, 0, 70, height);
    if (mousePressed == true) {
        stroke(0);
    }
    line(0, 70, width, 50);
}

```

if ブロックのなかのコードは、マウスボタンが押されているときだけ実行されます。ボタンが押されていなければ、ブロック内のコードは無視されます。4章で取り上げたforループと同様に、if文も式をテストして真か偽を判断します。

```
if (test) {  
    真のとき実行されるコード  
}
```

コンピュータはカッコのなかの式 (test) を評価し、真か偽かを判断します (Example 4-6を思い出してください)。それが真ならばブロック内のコードが実行され、偽ならばされません。

==という記号は、左側と右側の値が等しいかどうかをテストするときに使います。等しければ真です。==と代入演算子 (=) は別のもので、==は「両者は等しいか？」を問い合わせ、=は変数に値をセットします。

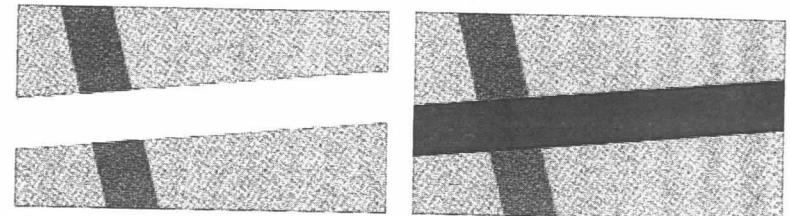
[NOTE] 経験豊富なプログラマーであっても、==と書くべきところで=と書いてしまうミスをしてかすことがあります。Processingはこのミスを見つけてくれないので、注意しましょう。

Example 5-12 の if 文は次のように書くこともできます。

```
if (mousePressed) {  
  
    mousePressedのようなブーリアン変数は、それ自身が真か偽を示すので、==演算子を使って明示的に比較する必要はありません。
```

### Example 5-13: クリックされていないことを検出する

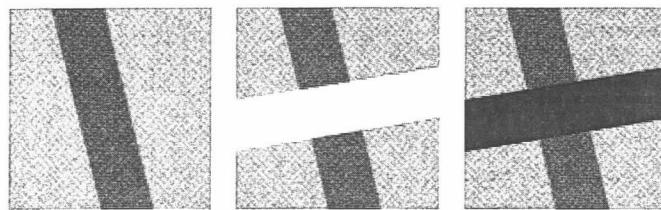
単一の if ブロックを使って書けるのは、コードを実行するかしないかの選択だけです。if ブロックに else ブロックを追加することで、2つの選択肢から1つを選ぶプログラムが可能になります。else ブロックのコードは、if ブロックのテストが偽のときに実行されます。次の例では、マウスボタンが押されていないときは白の線、押されている間は黒い線が画面上に現れます。



```
void setup() {  
    size(240, 120);  
    strokeWeight(30);  
}  
  
void draw() {  
    background(204);  
    stroke(102);  
    line(40, 0, 70, height);  
    if (mousePressed) {  
        stroke(0);  
    } else {  
        stroke(255);  
    }  
    line(0, 70, width, 50);  
}
```

### Example 5-14:複数のマウスボタン

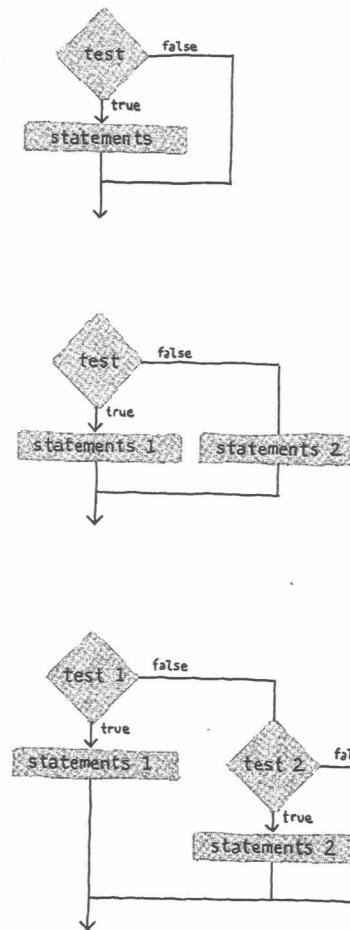
マウスに2個以上のボタンがある場合、Processingはどのボタンが押されたかも見ています。mouseButton変数の値はLEFT、CENTER、RIGHTのどれかです。押されたボタンを判定するために==演算子を使います。



```
void setup() {
    size(120, 120);
    strokeWeight(30);
}

void draw() {
    background(204);
    stroke(102);
    line(40, 0, 70, height);
    if (mousePressed) {
        if (mouseButton == LEFT) {
            stroke(255);
        } else {
            stroke(0);
        }
        line(0, 70, width, 50);
    }
}
```

if～elseの構造はいくつでも増やすことができます（図5-2）。連結して、条件が異なるテストを順番に行ったり、ifブロックのなかにifブロックを埋め込んで、より複雑な判定を行うことが可能です。



```
if (test) {
    statements
}
```

```
if (test) {
    statements 1
} else {
    statements 2
}
```

```
if (test 1) {
    statements 1
} else if (test 2) {
    statements 2
}
```

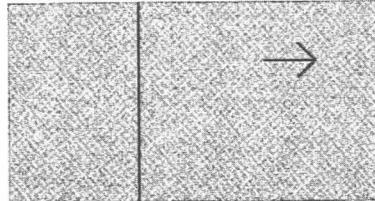
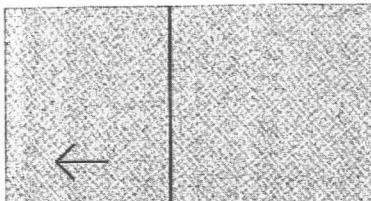
図5-2 if～else構造によって、どのブロックのコードを実行すべきかを決定

### カーソルの位置

ウィンドウ内でのカーソル位置を知るために、if文を使って mouseXと mouseYの値を調べてみましょう。

### Example 5-15:カーソルを探せ

カーソルが線のどちら側にあるかを調べてその結果を表示し、その線をカーソルのほうへ動かします。



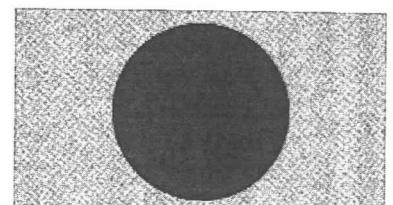
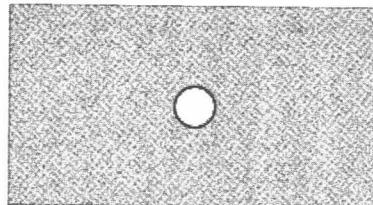
```
float x;
int offset = 10;
void setup() {
    size(240, 120);
    x = width/2;
}

void draw() {
    background(204);
    if (mouseX > x) {
        x += 0.5;
        offset = -10;
    }
    if (mouseX < x) {
        x -= 0.5;
        offset = 10;
    }
    line(x, 0, x, height);
    line(mouseX, mouseY, mouseX + offset, mouseY - 10);
    line(mouseX, mouseY, mouseX + offset, mouseY + 10);
    line(mouseX, mouseY, mouseX + offset*3, mouseY);
}
```

スクロールバー、ボタン、チェックボックスといったグラフィカルユーザーインターフェイスを備えたプログラムを書くために、カーソルが今どの領域にあるかを把握する方法を考えてみましょう。次の2つは、カーソルが円や長方形の内側にあるときだけ反応するプログラムの例です。再利用性を考慮した変数の使い方になっていて、異なる大きさの円や長方形に対する判定に使用できます。

### Example 5-16: 円の境界

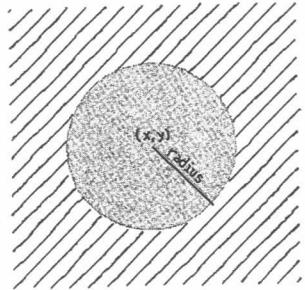
まずdist()関数を使って円の中心からカーソルまでの距離を調べます。その値が円の半径よりも小さければ、カーソルは円の内側にあるということです(図5-3)。この例では、カーソルを円に重ねると、その円が膨らみます。



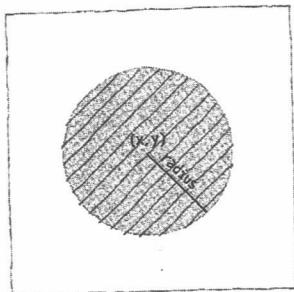
```
int x = 120;
int y = 60;
int radius = 12;

void setup() {
    size(240, 120);
    ellipseMode(RADIUS);
}

void draw() {
    background(204);
    float d = dist(mouseX, mouseY, x, y);
    if (d < radius) {
        radius++;
        fill(0);
    } else {
        fill(255);
    }
    ellipse(x, y, radius, radius);
}
```



`dist(x, y, mouseX, mouseY) > radius`

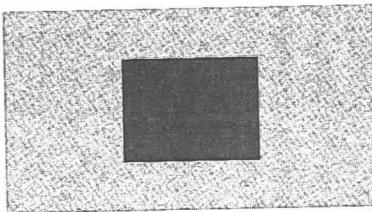
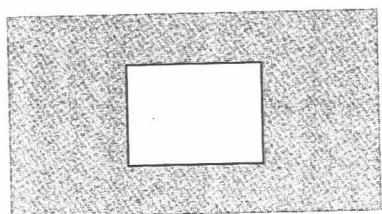


`dist(x, y, mouseX, mouseY) < radius`

図5-3 円に対する重なりの判定

#### Example 5-17:長方形の境界

長方形の内側にカーソルがあることを知るために、別のアプローチが必要です。4つの辺すべてについてカーソルがしかるべき側にあるかを調べ、すべて真ならば、カーソルは内側と判定します（図5-4）。ひとつひとつのステップは単純なのですが、1本のプログラムにまとめると少し複雑に見えるかもしれません。



```

int x = 80;
int y = 30;
int w = 80;
int h = 60;

void setup() {
    size(240, 120);
}

void draw() {
    background(204);
    if ((mouseX > x) && (mouseX < x+w) &&
        (mouseY > y) && (mouseY < y+h)) {
        fill(0);
    } else {
        fill(255);
    }
    rect(x, y, w, h);
}

```

if文のカッコの中身がこれまでになく複雑です。`(mouseX > x)`のようなテストが4つも`&&`でつなぎ合わされています。`&&`は論理演算のANDを表す記号で、すべての式が真であることを確認するために使われています。もしひとつでも偽があれば、このテスト全体が偽となって、長方形の色が黒に変わります。`&&`の詳しい説明はリファレンス（202ページ）にあります。

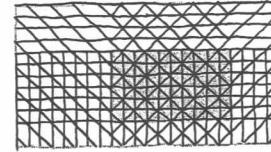
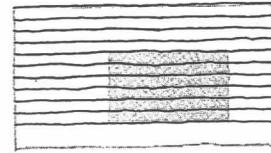
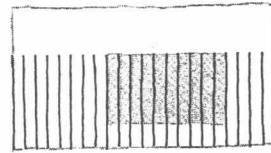
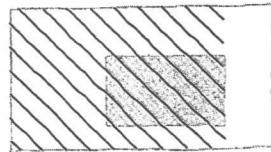
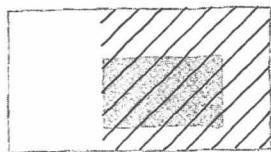
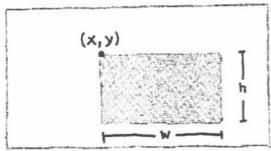


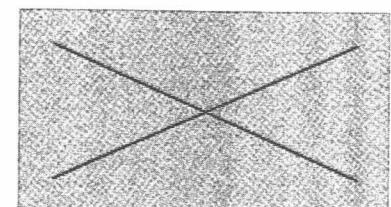
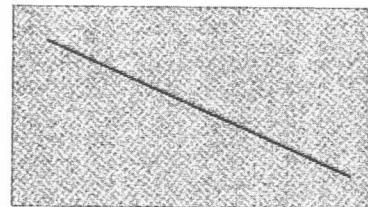
図 5-4 長方形に対する重なりの判定

## キーボードからの入力

Processing はキーボードの状態を見ていて、押されているキーの有無と、最後に押されたキーの情報を得ることができます。`mousePressed`変数と同様にキーが押されると真になる`keyPressed`という変数があります。どのキーも押されていないときの`keyPressed`は偽です。

### Example 5-18: キーを叩く

キーを押すと2本目の線が描かれます。



```
void setup() {
    size(240, 120);
}

void draw() {
    background(204);
    line(20, 20, 220, 100);
    if (keyPressed) {
        line(220, 20, 20, 100);
    }
}
```

最後に押されたキーがどれかは、`key`変数を見るとわかります。`key`のデータ型は`char`です。これは`character`の省略形ですが「チャー」と発音されることが多いようです。`char`型の変数には英数字を1つだけ格納できます。6章で説明する文字列はダブルクオート ("") で囲って表記しますが、`char`型はシングルクオート ('') です。

`char`型変数の宣言と代入は次のようにします。

```
char c = 'A'; // 宣言し、'A'を代入
```

次のようなコードはエラーとなります。

```
char c = "A"; // エラー! (char型の変数に文字列を代入しようとしている)
char h = A; // エラー! (シングルクオートがない)
```

ブーリアン型変数の `keyPressed` はキーから指が離れると偽に変化しますが、`key` 変数の値は次のキーが押されるまで変わりません。次の例では、`key` 変数の値を文字として描きます。キーが押されて値が更新されると、画面上の文字も書き換えられます。Shift や Alt といった文字以外のキーが押されたときは何も表示されません。

#### Example 5-19: 文字を描く

ここで3つの新しい関数が登場します。`textSize()` は文字の大きさを、`textAlign()` はテキストの水平位置を指定する関数です。`text()` 関数で文字を描きます。詳しい説明は Example 6-6 を見てください。



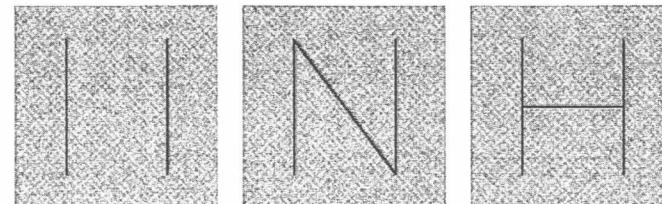
```
void setup() {
    size(120, 120);
    textSize(64);
    textAlign(CENTER);
}

void draw() {
    background(0);
    text(key, 60, 80);
}

if 文を使ってどのキーが押されたか判定し、それに応じて何かを表示するプログラムを考えてみましょう。
```

#### Example 5-20: 特定のキーに反応する

押されたキーが H か N のときだけ反応するよう、`==` 演算子を使って `key` 変数の値をテストします。



```
void setup() {
    size(120, 120);
}

void draw() {
    background(204);
    if (keyPressed) {
        if ((key == 'h') || (key == 'H')) {
            line(30, 60, 90, 60);
        }
        if ((key == 'n') || (key == 'N')) {
            line(30, 20, 90, 100);
        }
        line(30, 20, 30, 100);
        line(90, 20, 90, 100);
    }
}
```

Shift や Caps Lockと一緒に押されることも想定して、小文字と大文字の両方をチェックする必要があります。2つのテストを論理演算の OR を表す記号 `||` でつなぎました。この if 文の働きを普通の言葉で説明すると「もし h キーまたは H キーが押されたら」となります。AND (`&&`) で連結した式はすべてが真のときだけ真となりましたが、OR の場合は1つでも真ならその式全体も真です。

検出が面倒なキーがあります。ShiftやAlt、カーソルキーのような、特定の文字に紐付けられていないキーです。Processingはこれらのキーをコード化し、文字キーとは別の方法で処理します。まず、key変数の中身がコード化されたキーかをチェックし、それからkeyCode変数を参照してどのキーかを特定します。よく使われるkeyCodeの値はALT、CONTROL、SHIFT、それからカーソルキーのUP、DOWN、LEFT、RIGHTです。

#### Example 5-21: カーソルキーで動かす

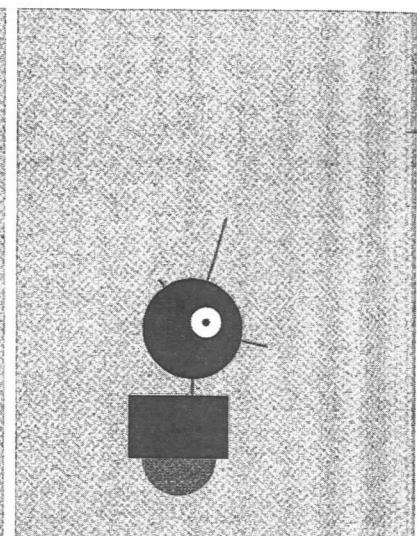
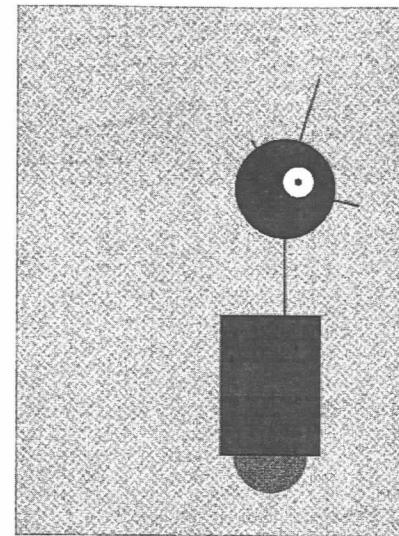
左右どちらかのカーソルキーに反応して、図形を動かします。

```
int x = 215;

void setup() {
    size(480, 120);
}

void draw() {
    if (keyPressed && (key == CODED)) { // それはコード化されたキーか
        if (keyCode == LEFT) { // それは左方向キーか
            x--;
        } else if (keyCode == RIGHT) { // それは右方向キーか
            x++;
        }
    }
    rect(x, 45, 50, 50);
}
```

## Robot 3: Response



マウスの動きに反応して形や位置が変化するロボットを作りましょう。Robot 2(4章)で導入した変数をプログラムの実行中に変更します。draw() ブロック内のコードは毎秒数十回実行され、フレームごとに mouseXと mousePressed の値が形や位置を表す変数に反映されます。マウスを動かすとロボットは左右に移動します。mouseXの値をそのままロボットの位置にしてしまうと変化が速すぎるので、イージングのテクニックを使って自然な動きを作っています。マウスボタンが押されると、neckHeightとbodyHeightが変化し、ロボットは縮みます。

```
float x = 60; // X座標
float y = 440; // Y座標
int radius = 45; // 頭の半径
int bodyHeight = 160; // 脳の高さ
int neckHeight = 70; // 首の高さ

float easing = 0.02;

void setup() {
    size(360, 480);
    strokeWeight(2);
    ellipseMode(RADIUS);
}
```

```
void draw() {
    int targetX = mouseX;
    x += (targetX - x) * easing;
    if (mousePressed) {
        neckHeight = 16;
        bodyHeight = 90;
    } else {
        neckHeight = 70;
        bodyHeight = 160;
    }
    float ny = y - bodyHeight - neckHeight - radius;
    background(204);

    // 首
    stroke(102);
    line(x+12, y-bodyHeight, x+12, ny);

    // アンテナ
    line(x+12, ny, x-18, ny-43);
    line(x+12, ny, x+42, ny-99);
    line(x+12, ny, x+78, ny+15);

    // 胴体
    noStroke();
    fill(102);
    ellipse(x, y-33, 33, 33);
    fill(0);
    rect(x-45, y-bodyHeight, 90, bodyHeight-33);

    // 頭
    fill(0);
    ellipse(x+12, ny, radius, radius);
    fill(255);
    ellipse(x+24, ny-6, 14, 14);
    fill(0);
    ellipse(x+24, ny-6, 3, 3);
}
```

# 6

## メディア Media

Processingを使って描けるのは単純な図形だけではありません。プログラムにラスタ画像、ベクタファイル、フォントといった要素を取り入れることで、写真、緻密なグラフ、多様な書体といった表現手段が使えるようになります。