

### 変数宣言、型変換、文字出力

```
void setup() {
  int a;
  float x;

  a = 12;
  x = 3.0/2;

  println("Hello");
  println( a );
  println( a, x );
  a = x;
  a = int( x );
  println( "a=", a, ",x=", x );
}
```

整数型変数 a を宣言する  
 実数型変数 x を宣言する  
 a に 12 を代入する  
 x に 3.0/2 の計算結果 1.5 を代入する (3/2 では 1 になる)  
 文字列「Hello」を表示する  
 a の値を表示する  
 a と x の値を並べて表示する (間に空白が自動的に入る)  
**エラー** 実数を整数型変数に代入することはできない  
 int() を使って実数を整数に変換してから、整数型変数に代入する  
 変数と文字列を並べて表示する

### 画像の表示

```
PImage f;

void settings() {
  f = loadImage("lena.jpg");
  size(f.width, f.height);
}

void setup() {
  image(f, 0, 0);
}
```

画像オブジェクト f を宣言する  
 画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 実行画面を原画像と同じサイズにする  
 実行画面に画像 f を貼る

### 画像の濃淡変換 (画像を明るくする)

```
PImage f;

void settings() {
  f = loadImage("lena.jpg");
  size(f.width, f.height);
}

void setup() {
  int x, y;
  float r, g, b;
  color c;

  for(y = 0; y < f.height; y++){
    for(x = 0; x < f.width; x++){
      r = red( f.get(x, y) );
      g = green( f.get(x, y) );
      b = blue( f.get(x, y) );
      r += 100;
      g += 100;
      b += 100;
      c = color(r, g, b);
      f.set(x, y, c);
    }
  }
  f.save("lena2.jpg");
  image(f, 0, 0);
}
```

画像オブジェクト f を宣言する  
 画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 実行画面を原画像と同じサイズにする  
 整数型変数 x, y を宣言する  
 実数型変数 r, g, b を宣言する  
 color 型変数 c を宣言する  
 y を 0 から画像縦幅-1 まで 1 ずつ増やす  
 x を 0 から画像横幅-1 まで 1 ずつ増やす  
 r に、座標(x,y)の画素の画素値 (赤成分) を代入する  
 g に、座標(x,y)の画素の画素値 (緑成分) を代入する  
 b に、座標(x,y)の画素の画素値 (青成分) を代入する  
 r を 100 増やす  
 g を 100 増やす  
 b を 100 増やす  
 RGB 値から color 型の値をつくり c に代入する  
 画像 f の座標(x,y)の画素値を c の色にする  
 画像 f を画像ファイル (lena2.jpg) へ書き出す  
 実行画面に画像 f を貼る

### 平均値フィルタ

```

PImage f, g;
void settings() {
  f = loadImage("lena.jpg");
  f.filter(GRAY);
  g = createImage(f.width, f.height, RGB);
  size(f.width*2, f.height);
}
void setup() {
  int x, y;
  float a;
  for(y = 1; y < f.height; y++){
    for(x = 1; x < f.width; x++){
      a = red(f.get(x-1,y-1)) + red(f.get(x,y-1)) + red(f.get(x+1,y-1))
        + red(f.get(x-1,y)) + red(f.get(x,y)) + red(f.get(x+1,y))
        + red(f.get(x-1,y+1)) + red(f.get(x,y+1)) + red(f.get(x+1,y+1));
      a = a / 9;
      g.set(x, y, color(a));
    }
  }
  image(f, 0, 0);
  image(g, f.width, 0);
}

```

画像オブジェクト f, g を宣言する

画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 画像 f をグレースケール (白黒濃淡) に変換する  
 画像 f と同じサイズで画像 g を用意する  
 実行画面のサイズを原画像の 2 つ分 (横幅 2 倍) にする

整数型変数 x, y を宣言する  
 実数型変数 a を宣言する  
 y を 1 から画像縦幅-2 まで 1 ずつ増やす  
 x を 1 から画像横幅-2 まで 1 ずつ増やす  
 座標(x,y)の近傍 9 画素の画素値の和を a に代入する  
 近傍 9 画素の平均値を a に代入する  
 画像 g の座標(x,y)の画素値を a にする

実行画面左に画像 f (原画像) を貼る  
 実行画面右に画像 g (変換後画像) を貼る

### 1 次微分フィルタ

```

PImage f, g1, g2, g3;
void settings() {
  f = loadImage("lena.jpg");
  f.filter(GRAY);
  g1 = createImage(f.width, f.height, RGB);
  g2 = createImage(f.width, f.height, RGB);
  g3 = createImage(f.width, f.height, RGB);
  size(f.width*2, f.height*2);
}
void setup() {
  int x, y;
  float dx, dy, norm;
  for(y = 1; y < f.height-1; y++){
    for(x = 1; x < f.width-1; x++){
      dx = red(f.get(x+1,y)) - red(f.get(x,y));
      dy = red(f.get(x,y+1)) - red(f.get(x,y));
      norm = sqrt(dx*dx + dy*dy);
      dx = abs(dx);
      dy = abs(dy);
      g1.set(x, y, color(dx));
      g2.set(x, y, color(dy));
      g3.set(x, y, color(norm));
    }
  }
  image(f, 0, 0);
  image(g1, f.width, 0);
  image(g2, 0, f.height);
  image(g3, f.width, f.height);
}

```

画像オブジェクト f, g1, g2, g3 を宣言する

画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 画像 f をグレースケールに変換する  
 画像 f と同じサイズで画像 g1 を用意する  
 画像 f と同じサイズで画像 g2 を用意する  
 画像 f と同じサイズで画像 g3 を用意する  
 実行画面のサイズを原画像の 4 つ分 (横・縦 2 倍) にする

整数型変数 x, y を宣言する  
 実数型変数 dx, dy, norm を宣言する  
 y を 1 から画像縦幅-2 まで 1 ずつ増やす  
 x を 1 から画像横幅-2 まで 1 ずつ増やす  
 座標(x,y)の横方向の微分値を dx に代入する  
 座標(x,y)の縦方向の微分値を dy に代入する  
 ベクトル(dx,dy)の大きさを norm に代入する  
 dx を絶対値にする  
 dy を絶対値にする  
 画像 g1 の座標(x,y)の画素値を dx にする  
 画像 g2 の座標(x,y)の画素値を dy にする  
 画像 g3 の座標(x,y)の画素値を norm にする

実行画面左上に画像 f (原画像) を貼る  
 実行画面右上に画像 g1 (横方向 1 次微分) を貼る  
 実行画面左下に画像 g2 (縦方向 1 次微分) を貼る  
 実行画面右下に画像 g3 (ベクトルの大きさ) を貼る

### ラプラシアン (2次微分) フィルタ

```

PImage f, g;

void settings() {
  f = loadImage("lena.jpg");
  f.filter(GRAY);
  g = createImage(f.width, f.height, RGB);
  size(f.width*2, f.height);
}

void setup() {
  int x, y;
  float a;
  for(y = 1; y < f.height-1; y++){
    for(x = 1; x < f.width-1; x++){
      a =
        + red(f.get(x,y-1))
        + red(f.get(x-1,y)) - 4*red(f.get(x,y)) + red(f.get(x+1,y))
        + red(f.get(x,y+1));

      a = abs(a);
      g.set(x, y, color(a));
    }
  }
  image(f, 0, 0);
  image(g, f.width, 0);
}

```

画像オブジェクト f, g を宣言する

画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 画像 f をグレースケールに変換する  
 画像 f と同じサイズで画像 g を用意する  
 実行画面のサイズを原画像の横幅の 2 倍にする

整数型変数 x, y を宣言する  
 実数型変数 a を宣言する

y を 1 から画像縦幅-2 まで 1 ずつ増やす  
 x を 1 から画像横幅-2 まで 1 ずつ増やす

座標(x,y)のラプラシアンの値を a に代入する  
 a を絶対値にする  
 画像 g の座標(x,y)の画素値を a にする

実行画面左に画像 f (原画像) を貼る  
 実行画面右に画像 g (2次微分) を貼る

### 画像の平行移動

画像 f を x 軸方向に 80 画素、y 軸方向に 30 画素、平行移動する。

```

PImage f, g;

void settings() {
  f = loadImage("lena.jpg");
  f.filter(GRAY);
  g = createImage(f.width, f.height, RGB);
  size(f.width*2, f.height);
}

void setup() {
  int X, Y;
  int x, y;
  float a;
  for(y = 0; y < g.height; y++){
    for(x = 0; x < g.width; x++){
      X = x - 80;
      Y = y - 30;
      a = red(f.get(X, Y));
      g.set(x, y, color(a));
    }
  }
  image(f, 0, 0);
  image(g, f.width, 0);
}

```

画像オブジェクト f, g を宣言する

画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 画像 f をグレースケールに変換する  
 画像 f と同じサイズで画像 g を用意する  
 実行画面のサイズを原画像の横幅の 2 倍にする

整数型変数 X, Y を宣言する 画像 f 上の座標(X,Y)  
 整数型変数 x, y を宣言する 画像 g 上の座標(x,y)  
 実数型変数 a を宣言する

y を 0 から画像縦幅-1 まで 1 ずつ増やす  
 x を 0 から画像横幅-1 まで 1 ずつ増やす  
 座標(x,y)から x 軸方向に-80 画素、y 軸方向に-30  
 画素平行移動した座標を(X,Y)に設定する

画像 f の座標(X,Y)の画素値を a に代入する  
 画像 g の座標(x,y)の画素値を a にする

実行画面左に画像 f (原画像) を貼る  
 実行画面右に画像 g (変換後画像) を貼る

画像の拡大縮小 (ニアレストネイバー (最近隣内挿法))

原点を中心にして画像 f を x 軸方向に 1.5 倍、y 軸方向に 0.8 倍に拡大する。

PImage f, g;

画像オブジェクト f, g を宣言する

```
void settings() {
  f = loadImage("lena.jpg");
  f.filter(GRAY);
  g = createImage(f.width, f.height, RGB);
  size(f.width*2, f.height);
}

void setup() {
  int X, Y;
  int x, y;
  float a, Xf, Yf;
  for(y = 0; y < g.height; y++){
    for(x = 0; x < g.width; x++){
      Xf = x / 1.5;
      Yf = y / 0.8;
      X = int(Xf + 0.5);
      Y = int(Yf + 0.5);
      a = red(f.get(X, Y));
      g.set(x, y, color(a));
    }
  }
  image(f, 0, 0);
  image(g, f.width, 0);
}
```

画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 画像 f をグレースケールに変換する  
 画像 f と同じサイズで画像 g を用意する  
 実行画面のサイズを原画像の横幅の 2 倍にする

整数型変数 X, Y を宣言する 画像 f 上の座標(X,Y)  
 整数型変数 x, y を宣言する 画像 g 上の座標(x,y)  
 実数型変数 a, Xf, Yf を宣言する

y を 0 から画像縦幅-1 まで 1 ずつ増やす  
 x を 0 から画像横幅-1 まで 1 ずつ増やす  
 座標(x,y)の、x 座標を 1/1.5 倍、  
 y 座標を 1/0.8 倍した座標を (Xf, Yf) に設定する  
 Xf を四捨五入して整数にした値を X に代入する  
 Yf を四捨五入して整数にした値を Y に代入する  
 画像 f の座標(X,Y)の画素値を a に代入する  
 画像 g の座標(x,y)の画素値を a にする

実行画面左に画像 f (原画像) を貼る  
 実行画面右に画像 g (変換後画像) を貼る

画像の拡大縮小 (バイリニア補間 (双線形補間))

原点を中心にして画像 f を x 軸方向に 1.5 倍、y 軸方向に 0.8 倍に拡大する。

PImage f, g;

画像オブジェクト f, g を宣言する

```
void settings() {
  f = loadImage("lena.jpg");
  f.filter(GRAY);
  g = createImage(f.width, f.height, RGB);
  size(f.width*2, f.height);
}

void setup() {
  int X, Y;
  int x, y;
  float a, Xf, Yf, s, t;
  for(y = 0; y < g.height; y++){
    for(x = 0; x < g.width; x++){
      Xf = x / 1.5;
      Yf = y / 0.8;
      X = int(Xf);
      Y = int(Yf);
      s = Xf - X;
      t = Yf - Y;
      a = (1-s)*(1-t) * red(f.get(X, Y))
        + s*(1-t) * red(f.get(X+1, Y))
        + (1-s)* t * red(f.get(X, Y+1))
        + s * t * red(f.get(X+1, Y+1));
      g.set(x, y, color(a));
    }
  }
  image(f, 0, 0);
  image(g, f.width, 0);
}
```

画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 画像 f をグレースケールに変換する  
 画像 f と同じサイズで画像 g を用意する  
 実行画面のサイズを原画像の横幅の 2 倍にする

整数型変数 X, Y を宣言する 画像 f 上の座標(X,Y)  
 整数型変数 x, y を宣言する 画像 g 上の座標(x,y)  
 実数型変数 a, Xf, Yf, s, t を宣言する

y を 0 から画像縦幅-1 まで 1 ずつ増やす  
 x を 0 から画像横幅-1 まで 1 ずつ増やす  
 座標(x,y)の、x 座標を 1/1.5 倍、  
 y 座標を 1/0.8 倍した座標を (Xf, Yf) に設定する  
 Xf を整数化(小数点以下切り捨て)した値を X に代入する  
 Yf を整数化した値を Y に代入する  
 内分比 s を求める (Xf の小数部)  
 内分比 t を求める (Yf の小数部)

座標(Xf, Yf)の隣接 4 画素の画素値から  
 座標(Xf, Yf)の画素値をバイリニア補間  
 で求めて a に代入する

画像 g の座標(x,y)の画素値を a にする

実行画面左に画像 f (原画像) を貼る  
 実行画面右に画像 g (変換後画像) を貼る

## 画像の回転

原点を中心にして画像 f を 20 度回転する。

```

PImage f, g;

void settings() {
  f = loadImage("lena.jpg");
  f.filter(GRAY);
  g = createImage(f.width, f.height, RGB);
  size(f.width*2, f.height);
}

void setup() {
  int X, Y;
  int x, y;
  float a, Xf, Yf, s, t;
  float rad;

  rad = radians(20);

  for(y = 0; y < g.height; y++){
    for(x = 0; x < g.width; x++){
      Xf = x*cos(rad) + y*sin(rad);
      Yf = -x*sin(rad) + y*cos(rad);

      X = int(Xf);
      Y = int(Yf);

      s = Xf - X;
      t = Yf - Y;

      a = (1-s)*(1-t) * red(f.get(X, Y))
        + s*(1-t) * red(f.get(X+1, Y))
        + (1-s)*t * red(f.get(X, Y+1))
        + s*t * red(f.get(X+1, Y+1));
      g.set(x, y, color(a));
    }
  }
  image(f, 0, 0);
  image(g, f.width, 0);
}

```

画像オブジェクト f, g を宣言する

画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 画像 f をグレースケールに変換する  
 画像 f と同じサイズで画像 g を用意する  
 実行画面のサイズを原画像の横幅の 2 倍にする

整数型変数 X, Y を宣言する 画像 f 上の座標(X,Y)  
 整数型変数 x, y を宣言する 画像 g 上の座標(x,y)  
 実数型変数 a, Xf, Yf, s, t を宣言する  
 実数型変数 rad を宣言する

角度 20 度をラジアン単位に変換した値を rad に代入する

y を 0 から画像縦幅-1 まで 1 ずつ増やす

x を 0 から画像横幅-1 まで 1 ずつ増やす

座標(x,y)を、原点を中心にして-20 度回転したときの  
 座標を(Xf, Yf)に設定する

Xf を整数化(小数点以下切り捨て)した値を X に代入する  
 Yf を整数化した値を Y に代入する

内分比 s を求める (Xf の小数部)

内分比 t を求める (Yf の小数部)

座標(Xf, Yf)の隣接 4 画素の画素値から  
 座標(Xf, Yf)の画素値をバイリニア補間  
 で求めて a に代入する

画像 g の座標(x,y)の画素値を a にする

実行画面左に画像 f (原画像) を貼る

実行画面右に画像 g (変換後画像) を貼る

## 2 値図形の面積を求める

```

PImage f;

void settings() {
  f = loadImage("lena.jpg");
  f.filter(GRAY);
  size(f.width, f.height);
}

void setup() {
  int x, y;
  int S = 0;
  f.filter(THRESHOLD, 0.47);

  for(y = 0; y < f.height; y++){
    for(x = 0; x < f.width; x++){
      if( red(f.get(x, y)) == 255 ){
        S++;
      }
    }
  }
  println("Image Area =", f.width*f.height);
  println("Object Area =", S);
  image(f, 0, 0);
}

```

画像オブジェクト f を宣言する

画像 f へ画像ファイル (lena.jpg) のデータを読み込む  
 画像 f をグレースケールに変換する  
 実行画面を原画像と同じサイズにする

整数型変数 x, y を宣言する

整数型変数 S を宣言し、0 で初期化する

filter 関数を用いて画像 f を 2 値画像に変換する

※filter 関数では、しきい値を 0~1 の実数で与える

しきい値を画素値 120 にする場合は、 $120 \div 255 = 0.47$  になる

y を 0 から画像縦幅-1 まで 1 ずつ増やす

x を 0 から画像横幅-1 まで 1 ずつ増やす

画像 f の座標(x,y)の画素が白画素であれば、  
 S の値を 1 増やす

画像 f の全画素数を表示する

物体の面積 S を表示する

実行画面に画像 f を貼る

**相違度 SSD (Sum of Squared Differences 差分 2 乗和) の計算**

<pre> PImage f, t;  void settings() {   f = loadImage("lena.jpg");   t = loadImage("lena2.jpg");    f.filter(GRAY);   t.filter(GRAY);   size(f.width*2, f.height); }  void setup() {   int x, y;   float a, ssd = 0;   for(y = 0; y &lt; f.height; y++){     for(x = 0; x &lt; f.width; x++){       a = red(f.get(x,y)) - red(t.get(x,y));       ssd += a*a;     }   }   println("SSD=", ssd);   image(f, 0, 0);   image(t, f.width, 0); } </pre>	<p>画像オブジェクト f, g を宣言する</p> <p>対象画像 f へ画像ファイル(lena.jpg)のデータを読み込む          テンプレート画像 t へ画像ファイル(lena2.jpg)のデータを読み込む <b>画像 f と t は同じサイズでなければならない</b>          画像 f をグレースケール (白黒濃淡) に変換する          画像 t をグレースケール (白黒濃淡) に変換する          実行画面のサイズを原画像の横幅の 2 倍にする</p> <p>整数型変数 x, y を宣言する          実数型変数 a, ssd を宣言し、ssd の初期値を 0 にする          y を 0 から画像縦幅-1 まで 1 ずつ増やす          x を 0 から画像横幅-1 まで 1 ずつ増やす          座標(x,y)における画像 f と t の画素値の差を a に代入する          a の 2 乗を ssd に加える</p> <p>計算された SSD の値を表示する          実行画面左に対象画像 f を貼る          実行画面右にテンプレート画像 t を貼る</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**相違度 SSD を用いたテンプレートマッチング** 画像サイズが大きいほど処理時間が長くなるので注意せよ。

<pre> PImage f, t;  void settings() {   f = loadImage("lena.jpg");   t = loadImage("eye.jpg");    f.filter(GRAY);   t.filter(GRAY);   size(f.width+t.width, f.height); }  void setup() {   int x, y, X, Y;   int mx=0, my=0;   float a, ssd, ssdmin = 1000000000;    for(Y = 0; Y &lt; f.height-t.height; Y++){     for(X = 0; X &lt; f.width-t.width; X++){       ssd = 0;       for(y = 0; y &lt; t.height; y++){         for(x = 0; x &lt; t.width; x++){           a = red(f.get(x+X,y+Y)) - red(t.get(x,y));           ssd += a*a;         }       }       if(ssd &lt; ssdmin){         mx = X; my = Y;         ssdmin = ssd;       }     }   }   image(f, 0, 0);   image(t, f.width, 0);   stroke(255, 0, 0);   noFill();   rect(mx, my, xtmp, ytmp); } </pre>	<p>画像オブジェクト f, t を宣言する</p> <p>対象画像 f へ画像ファイル(lena.jpg)のデータを読み込む          テンプレート画像 t へ画像ファイル(template.jpg)のデータを読み込む          画像 f をグレースケール (白黒濃淡) に変換する          画像 t をグレースケール (白黒濃淡) に変換する          実行画面の横幅を画像 f と t を合わせたサイズにする</p> <p>整数型変数 x, y, X, Y を宣言する          整数型変数 mx, my を宣言する          実数型変数 a, ssd, ssdmin を宣言し、ssdmin の初期値を十分に大きな値にする          Y を 0 から(画像 f 縦幅-画像 t 縦幅-1)まで 1 ずつ増やす          X を 0 から(画像 f 横幅-画像 t 横幅-1)まで 1 ずつ増やす          ssd を 0 で初期化する          y を 0 から画像 t 縦幅-1 まで 1 ずつ増やす          x を 0 から画像 t 横幅-1 まで 1 ずつ増やす          画像 f の座標(x+X,y+Y)の画素値と画像 t の座標(x,y)の画素値の差を a に代入し、a の 2 乗を ssd に加える</p> <p>現在の座標(X,Y)における ssd が ssdmin より小さければ、          mx, my それぞれに X, Y の値を代入し、          ssdmin に ssd の値を代入する</p> <p>実行画面左に対象画像 f を貼る          実行画面右にテンプレート画像 t を貼る          線を赤色 (R=255, G=0, B=0) に設定する          図形塗り潰しを無しに設定する          テンプレート画像と最もマッチングした箇所を四角形で囲む</p>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------