変数宣言、型変換、文字出力

```
void setup() {
  int   a;                          整数型変数 a を宣言する
  float x;                          実数型変数 x を宣言する
  a = 12;                           a に 12 を代入する
  x = 3.0/2;                        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）
  println("Hello");                 文字列「Hello」を表示する
  println( a );                     a の値を表示する
  println( a, x );                  a と x の値を並べて表示する（間に空白が自動的に入る）
  a = x;                            エラー　実数を整数型変数に代入することはできない
  a = int( x );                     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
  println( "a=", a, ",x=", x );     変数と文字列を並べて表示する
}
```

画像の表示

```
int  xsize=320, ysize=320;          xsize と ysize に原画像の横幅と縦幅を設定する
                                    この値は、使用する画像に合わせて変更する
void settings() {
  size(xsize, ysize);               実行画面を原画像と同じサイズにする
}
void setup() {
  PImage f;                         画像オブジェクト f を宣言する
  f = loadImage("lenna.jpg");       画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  image(f, 0, 0);                   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）

```
int  xsize=320, ysize=320;          xsize と ysize に原画像の横幅と縦幅を設定する
void settings() {
  size(xsize, ysize);               実行画面を原画像と同じサイズにする
}
void setup() {
  PImage f;                         画像オブジェクト f を宣言する
  int   x, y;                       整数型変数 x, y を宣言する
  float r, g, b;                    実数型変数 r, g, b を宣言する
  color c;                          color 型変数 c を宣言する
  f = loadImage("lenna.jpg");       画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  for(y = 0; y < ysize; y++){       y を 0 から画像縦幅-1 まで 1 ずつ増やす
    for(x = 0; x < xsize; x++){       x を 0 から画像横幅-1 まで 1 ずつ増やす
      r =   red( f.get(x, y) );       r に、座標(x,y)の画素の画素値（赤成分）を代入する
      g = green( f.get(x, y) );       g に、座標(x,y)の画素の画素値（緑成分）を代入する
      b =  blue( f.get(x, y) );       b に、座標(x,y)の画素の画素値（青成分）を代入する
      r += 100;                       r を 100 増やす
      g += 100;                       g を 100 増やす
      b += 100;                       b を 100 増やす
      c = color(r, g, b);             RGB 値から color 型の値をつくり c に代入する
      f.set(x, y, c);                 画像 f の座標(x,y)の画素値を c の色にする
    }
  }
  f.save("lenna2.jpg");             画像 f を画像ファイル（lenna2.jpg）へ書き出す
  image(f, 0, 0);                   実行画面に画像 f を貼る
}
```

平均化フィルタ

```
int  xsize=320, ysize=320;                  xsize と ysize に原画像の横幅と縦幅を設定する
void settings() {
  size(xsize*2, ysize);                     実行画面のサイズを原画像の 2 つ分（横幅 2 倍）にする
}
void setup() {
  PImage f, g;                              画像オブジェクト f, g を宣言する
  int   x, y;                               整数型変数 x, y を宣言する
  float a;                                  実数型変数 a を宣言する
  f = loadImage("lenna.jpg");               画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  f.filter(GRAY);                           画像 f をグレースケール（白黒濃淡）に変換する
  g = createImage(xsize, ysize, RGB);       画像 f と同じサイズで画像 g を用意する
  for(y = 1; y < ysize; y++){               y を 1 から画像縦幅-2 まで 1 ずつ増やす
    for(x = 1; x < xsize; x++){               x を 1 から画像横幅-2 まで 1 ずつ増やす
      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));
                                                座標(x,y)の近傍 9 画素の画素値の和を a に代入する
      a = a / 9;                                近傍 9 画素の平均値を a に代入する
      g.set(x, y, color(a));                    画像 g の座標(x,y)の画素値を a にする
    }
  }
  image(f, 0, 0);                           実行画面左に画像 f（原画像）を貼る
  image(g, xsize, 0);                       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

```
int  xsize=320, ysize=320;                  xsize と ysize に原画像の横幅と縦幅を設定する
void settings() {
  size(xsize*2, ysize*2);                   実行画面のサイズを原画像の 4 つ分（横･縦 2 倍）にする
}
void setup() {
  PImage f, g1, g2, g3;                     画像オブジェクト f, g1, g2, g3 を宣言する
  int   x, y;                               整数型変数 x, y を宣言する
  float dx, dy, norm;                       実数型変数 dx, dy, norm を宣言する
  f = loadImage("lenna.jpg");               画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  f.filter(GRAY);                           画像 f をグレースケールに変換する
  g1 = createImage(xsize, ysize, RGB);      画像 f と同じサイズで画像 g1 を用意する
  g2 = createImage(xsize, ysize, RGB);      画像 f と同じサイズで画像 g2 を用意する
  g3 = createImage(xsize, ysize, RGB);      画像 f と同じサイズで画像 g3 を用意する
  for(y = 1; y < ysize-1; y++){             y を 1 から画像縦幅-2 まで 1 ずつ増やす
    for(x = 1; x < xsize-1; x++){             x を 1 から画像横幅-2 まで 1 ずつ増やす
      dx = red(f.get(x+1,y  )) - red(f.get(x,y));       座標(x,y)の横方向の微分値を dx に代入する
      dy = red(f.get(x  ,y+1)) - red(f.get(x,y));       座標(x,y)の縦方向の微分値を dy に代入する
      norm = sqrt(dx*dx + dy*dy);                 ベクトル(dx,dy)の大きさを norm に代入する
      dx = abs(dx);                               dx を絶対値にする
      dy = abs(dy);                               dy を絶対値にする
      g1.set(x, y, color(dx));                    画像 g1 の座標(x,y)の画素値を dx にする
      g2.set(x, y, color(dy));                    画像 g2 の座標(x,y)の画素値を dy にする
      g3.set(x, y, color(norm));                  画像 g3 の座標(x,y)の画素値を norm にする
    }
  }
  image(f, 0, 0);                           実行画面左上に画像 f（原画像）を貼る
  image(g1, xsize, 0);                      実行画面右上に画像 g1（横方向 1 次微分）を貼る
  image(g2, 0, ysize);                      実行画面左下に画像 g2（縦方向 1 次微分）を貼る
  image(g3, xsize, ysize);                  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

```
int  xsize=320, ysize=320;                    xsize と ysize に原画像の横幅と縦幅を設定する
void settings() {
  size(xsize*2, ysize);                       実行画面のサイズを原画像の横幅の 2 倍にする
}
void setup() {
  PImage f, g;                                画像オブジェクト f, g を宣言する
  int   x, y;                                 整数型変数 x, y を宣言する
  float a;                                    実数型変数 a を宣言する
  f = loadImage("lenna.jpg");                 画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  f.filter(GRAY);                             画像 f をグレースケールに変換する
  g = createImage(xsize, ysize, RGB);         画像 f と同じサイズで画像 g を用意する
  for(y = 1; y < ysize-1; y++){               y を 1 から画像縦幅-2 まで 1 ずつ増やす
    for(x = 1; x < xsize-1; x++){               x を 1 から画像横幅-2 まで 1 ずつ増やす
      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));
                                                座標(x,y)のラプラシアンの値を a に代入する
      a = abs(a);                               a を絶対値にする
      g.set(x, y, color(a));                    画像 g の座標(x,y)の画素値を a にする
    }
  }
  image(f, 0, 0);                             実行画面左に画像 f（原画像）を貼る
  image(g, xsize, 0);                         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。

```
int  xsize=320, ysize=320;                    xsize と ysize に原画像の横幅と縦幅を設定する
void settings() {
  size(xsize*2, ysize);                       実行画面のサイズを原画像の横幅の 2 倍にする
}
void setup() {
  PImage f, g;                                画像オブジェクト f, g を宣言する
  int   X, Y;                                 整数型変数 X, Y を宣言する　画像 f 上の座標(X,Y)
  int   x, y;                                 整数型変数 x, y を宣言する　画像 g 上の座標(x,y)
  float a;                                    実数型変数 a を宣言する
  f = loadImage("lenna.jpg");                 画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  f.filter(GRAY);                             画像 f をグレースケールに変換する
  g = createImage(xsize, ysize, RGB);         画像 f と同じサイズで画像 g を用意する
  for(y = 0; y < ysize; y++){                 y を 0 から画像縦幅-1 まで 1 ずつ増やす
    for(x = 0; x < xsize; x++){                 x を 0 から画像横幅-1 まで 1 ずつ増やす
      X = x - 80;                               座標(x,y)から x 軸方向に-80 画素、y 軸方向に-30
      Y = y - 30;                               画素平行移動した座標を(X,Y)に設定する
      a = red(f.get(X, Y));                     画像 f の座標(X,Y)の画素値を a に代入する
      g.set(x, y, color(a));                    画像 g の座標(x,y)の画素値を a にする
    }
  }
  image(f, 0, 0);                             実行画面左に画像 f（原画像）を貼る
  image(g, xsize, 0);                         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。

```
int  xsize=320, ysize=320;                  xsize と ysize に原画像の横幅と縦幅を設定する

void settings() {
  size(xsize*2, ysize);                     実行画面のサイズを原画像の横幅の 2 倍にする
}
void setup() {
  PImage f, g;                              画像オブジェクト f, g を宣言する
  int   X, Y;                               整数型変数 X, Y を宣言する　画像 f 上の座標(X,Y)
  int   x, y;                               整数型変数 x, y を宣言する　画像 g 上の座標(x,y)
  float a, Xf, Yf;                          実数型変数 a, Xf, Yf を宣言する

  f = loadImage("lenna.jpg");               画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  f.filter(GRAY);                           画像 f をグレースケールに変換する
  g = createImage(xsize, ysize, RGB);       画像 f と同じサイズで画像 g を用意する

  for(y = 0; y < ysize; y++){               y を 0 から画像縦幅-1 まで 1 ずつ増やす
    for(x = 0; x < xsize; x++){               x を 0 から画像横幅-1 まで 1 ずつ増やす
      Xf = x / 1.5;                             座標(x,y)の、x 座標を 1/1.5 倍、
      Yf = y / 0.8;                             y 座標を 1/0.8 倍した座標を(Xf,Yf)に設定する

      X  = int(Xf + 0.5);                       Xf を四捨五入して整数にした値を X に代入する
      Y  = int(Yf + 0.5);                       Yf を四捨五入して整数にした値を Y に代入する

      a = red(f.get(X, Y));                     画像 f の座標(X,Y)の画素値を a に代入する
      g.set(x, y, color(a));                    画像 g の座標(x,y)の画素値を a にする
    }
  }
  image(f, 0, 0);                           実行画面左に画像 f（原画像）を貼る
  image(g, xsize, 0);                       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。

```
int  xsize=320, ysize=320;                  xsize と ysize に原画像の横幅と縦幅を設定する

void settings() {
  size(xsize*2, ysize);                     実行画面のサイズを原画像の横幅の 2 倍にする
}
void setup() {
  PImage f, g;                              画像オブジェクト f, g を宣言する
  int   X, Y;                                 整数型変数 X, Y を宣言する　画像 f 上の座標(X,Y)
  int   x, y;                                 整数型変数 x, y を宣言する　画像 g 上の座標(x,y)
  float a, Xf, Yf, s, t;                      実数型変数 a, Xf, Yf, s, t を宣言する

  f = loadImage("lenna.jpg");                 画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  f.filter(GRAY);                             画像 f をグレースケールに変換する
  g = createImage(xsize, ysize, RGB);         画像 f と同じサイズで画像 g を用意する

  for(y = 0; y < xsize; y++){                 y を 0 から画像縦幅-1 まで 1 ずつ増やす
    for(x = 0; x < ysize; x++){                 x を 0 から画像横幅-1 まで 1 ずつ増やす
      Xf = x / 1.5;                               座標(x,y)の、x 座標を 1/1.5 倍、
      Yf = y / 0.8;                               y 座標を 1/0.8 倍した座標を(Xf,Yf)に設定する

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

      s  = Xf - X;                                内分比 s を求める（Xf の小数部）
      t  = Yf - Y;                                内分比 t を求める（Yf の小数部）

      a =   (1-s)*(1-t) * red(f.get(X  , Y  ))      座標(Xf,Yf)の隣接 4 画素の画素値から
          +    s *(1-t) * red(f.get(X+1, Y  ))      座標(Xf,Yf)の画素値をバイリニア補間
          + (1-s)*   t  * red(f.get(X  , Y+1))      で求めて a に代入する
          +    s *   t  * red(f.get(X+1, Y+1));
      g.set(x, y, color(a));                      画像 g の座標(x,y)の画素値を a にする
    }
  }
  image(f, 0, 0);                             実行画面左に画像 f（原画像）を貼る
  image(g, xsize, 0);                         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。

```
int  xsize=320, ysize=320;                  xsize と ysize に原画像の横幅と縦幅を設定する

void settings() {
  size(xsize*2, ysize);                     実行画面のサイズを原画像の横幅の 2 倍にする
}
void setup() {
  PImage f, g;                              画像オブジェクト f, g を宣言する
  int   X, Y;                               整数型変数 X, Y を宣言する　画像 f 上の座標(X,Y)
  int   x, y;                               整数型変数 x, y を宣言する　画像 g 上の座標(x,y)
  float a, Xf, Yf, s, t;                    実数型変数 a, Xf, Yf, s, t を宣言する
  float rad;                                実数型変数 rad を宣言する

  f = loadImage("lenna.jpg");               画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  f.filter(GRAY);                           画像 f をグレイスケールに変換する
  g = createImage(xsize, ysize, RGB);       画像 f と同じサイズで画像 g を用意する

  rad = radians(20);                        角度 20 度をラジアン単位に変換した値を rad に代入する

  for(y = 0; y < ysize; y++){               y を 0 から画像縦幅-1 まで 1 ずつ増やす
    for(x = 0; x < xsize; x++){               x を 0 から画像横幅-1 まで 1 ずつ増やす
      Xf =  x*cos(rad) + y*sin(rad);            座標(x,y)を、原点を中心にして-20 度回転したときの
      Yf = -x*sin(rad) + y*cos(rad);            座標を(Xf,Yf)に設定する

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

      s  = Xf - X;                              内分比 s を求める（Xf の小数部）
      t  = Yf - Y;                              内分比 t を求める（Yf の小数部）

      a =   (1-s)*(1-t) * red(f.get(X  , Y  ))    座標(Xf,Yf)の隣接 4 画素の画素値から
          +    s *(1-t) * red(f.get(X+1, Y  ))    座標(Xf,Yf)の画素値をバイリニア補間
          + (1-s)*   t  * red(f.get(X  , Y+1))    で求めて a に代入する
          +    s *   t  * red(f.get(X+1, Y+1));
      g.set(x, y, color(a));                    画像 g の座標(x,y)の画素値を a にする
    }
  }
  image(f, 0, 0);                           実行画面左に画像 f（原画像）を貼る
  image(g, xsize, 0);                       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

```
int  xsize=320, ysize=320;                  xsize と ysize に原画像の横幅と縦幅を設定する

void settings() {
  size(xsize, ysize);                       実行画面を原画像と同じサイズにする
}
void setup() {
  PImage f;                                 画像オブジェクト f を宣言する
  int  x, y;                                整数型変数 x, y を宣言する
  int  S = 0;                               整数型変数 S を宣言し、0 で初期化する

  f = loadImage("lenna.jpg");               画像 f へ画像ファイル（lenna.jpg）のデータを読み込む
  f.filter(GRAY);                           画像 f をグレースケールに変換する
  f.filter(THRESHOLD, 0.47);                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÷255=0.47 になる

  for(y = 0; y < ysize; y++){               y を 0 から画像縦幅-1 まで 1 ずつ増やす
    for(x = 0; x < xsize; x++){               x を 0 から画像横幅-1 まで 1 ずつ増やす
      if( red(f.get(x, y)) == 255 ){            画像 f の座標(x,y)の画素が白画素であれば、
        S++;                                    S の値を 1 増やす
      }
    }
  }
  println("Image =", xsize*ysize);           画像 f の全画素数を表示する
  println("Object=", S);                     物体の面積 S を表示する
  image(f, 0, 0);                           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

```
int  xsize=320, ysize=320;                          xsize と ysize に原画像の横幅と縦幅を設定する

void settings() {
  size(xsize*2, ysize);                             実行画面のサイズを原画像の横幅の 2 倍にする
}
void setup() {
  PImage f, t;                                      画像オブジェクト f, g を宣言する
  int    x, y;                                      整数型変数 x, y を宣言する
  float a, ssd = 0;                                 実数型変数 a, ssd を宣言し、ssd の初期値を 0 にする
  f = loadImage("lenna.jpg");                       対象画像 f へ画像ファイル(lenna.jpg)のデータを読み込む
  t = loadImage("lenna2.jpg");                      テンプレート画像 t へ画像ファイル(lenna2.jpg)のデータを
                                                    読み込む　画像 f と t は同じサイズでなければならない
  f.filter(GRAY);                                   画像 f をグレイスケール（白黒濃淡）に変換する
  t.filter(GRAY);                                   画像 t をグレイスケール（白黒濃淡）に変換する
  for(y = 0; y < ysize; y++){                       y を 0 から画像縦幅-1 まで 1 ずつ増やす
    for(x = 0; x < xsize; x++){                       x を 0 から画像横幅-1 まで 1 ずつ増やす
      a = red(f.get(x,y)) - red(t.get(x,y));          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
      ssd += a*a;                                     a の 2 乗を ssd に加える
    }
  }
  println("SSD=", ssd);                             計算された SSD の値を表示する
  image(f, 0, 0);                                   実行画面左に対象画像 f を貼る
  image(t, xsize, 0);                               実行画面右にテンプレート画像 t を貼る
}
```

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。

```
int  xsize=320, ysize=320;                          xsize と ysize に原画像の横幅と縦幅を設定する
int  xtmp=40, ytmp=40;                              xtmp と ytmp にテンプレート画像の横幅と縦幅を設定する

void settings() {
  size(xsize+xtmp, ysize);                          実行画面の横幅を画像 f と t を合わせたサイズにする
}
void setup() {
  PImage f, t;                                      画像オブジェクト f, t を宣言する
  int    x, y, X, Y;                                整数型変数 x, y, X, Y を宣言する
  int    mx=0, my=0;                                整数型変数 mx, my を宣言する
  float a, ssd, ssdmin = 1000000000;                実数型変数 a, ssd, ssdmin を宣言し、ssdmin の初期値を
                                                    充分に大きな値にする
  f = loadImage("lenna.jpg");                       対象画像 f へ画像ファイル(lenna.jpg)のデータを読み込む
  t = loadImage("template.jpg");                    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
  f.filter(GRAY);                                   画像 f をグレイスケール（白黒濃淡）に変換する
  t.filter(GRAY);                                   画像 t をグレイスケール（白黒濃淡）に変換する
  for(Y = 0; Y < ysize-ytmp; Y++){                  Y を 0 から(画像 f 縦幅-画像 t 縦幅-1)まで 1 ずつ増やす
    for(X = 0; X < xsize-xtmp; X++){                  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 = 0;                                        ssd を 0 で初期化する
      for(y = 0; y < ytmp; y++){                      y を 0 から画像 t 縦幅-1 まで 1 ずつ増やす
        for(x = 0; x < xtmp; x++){                      x を 0 から画像 t 横幅-1 まで 1 ずつ増やす
          a = red(f.get(x+X,y+Y)) - red(t.get(x,y));    画像 f の座標(x+X,y+Y)の画素値と画像 t の座標
          ssd += a*a;                                   (x,y)の画素値の差を a に代入し、a の 2 乗を ssd
        }                                               に加える
      }
      if(ssd < ssdmin){                               現在の座標(X,Y)における ssd が ssdmin より小さければ、
        mx = X; my = Y;                                 mx, my それぞれに X, Y の値を代入し、
        ssdmin = ssd;                                   ssdmin に ssd の値を代入する
      }
    }
  }
  image(f, 0, 0);                                   実行画面左に対象画像 f を貼る
  image(t, xsize, 0);                               実行画面右にテンプレート画像 t を貼る
  stroke(255, 0, 0); noFill();                      線の色を赤色、図形塗り潰しなしに設定する
  rect(mx, my, xtmp, ytmp);                         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
}
```