

計算機実習 III

第1回： 静止画の作成1

担当： 佐藤

2018年4月10日(火)

講義概要

● 使用言語

▪ Processing(プロセッシング)

- Javaをベースに視覚的な表現を作りやすくしたプログラミング言語
- Processingの公式Webサイト(<https://processing.org/download/>)からダウンロード可能
- 講義室のPCにはあらかじめインストール済み

● 講義の構成

- 3部構成(1部あたり5回)
- 各部の最後の講義回→その部で学習した内容の「総合演習」

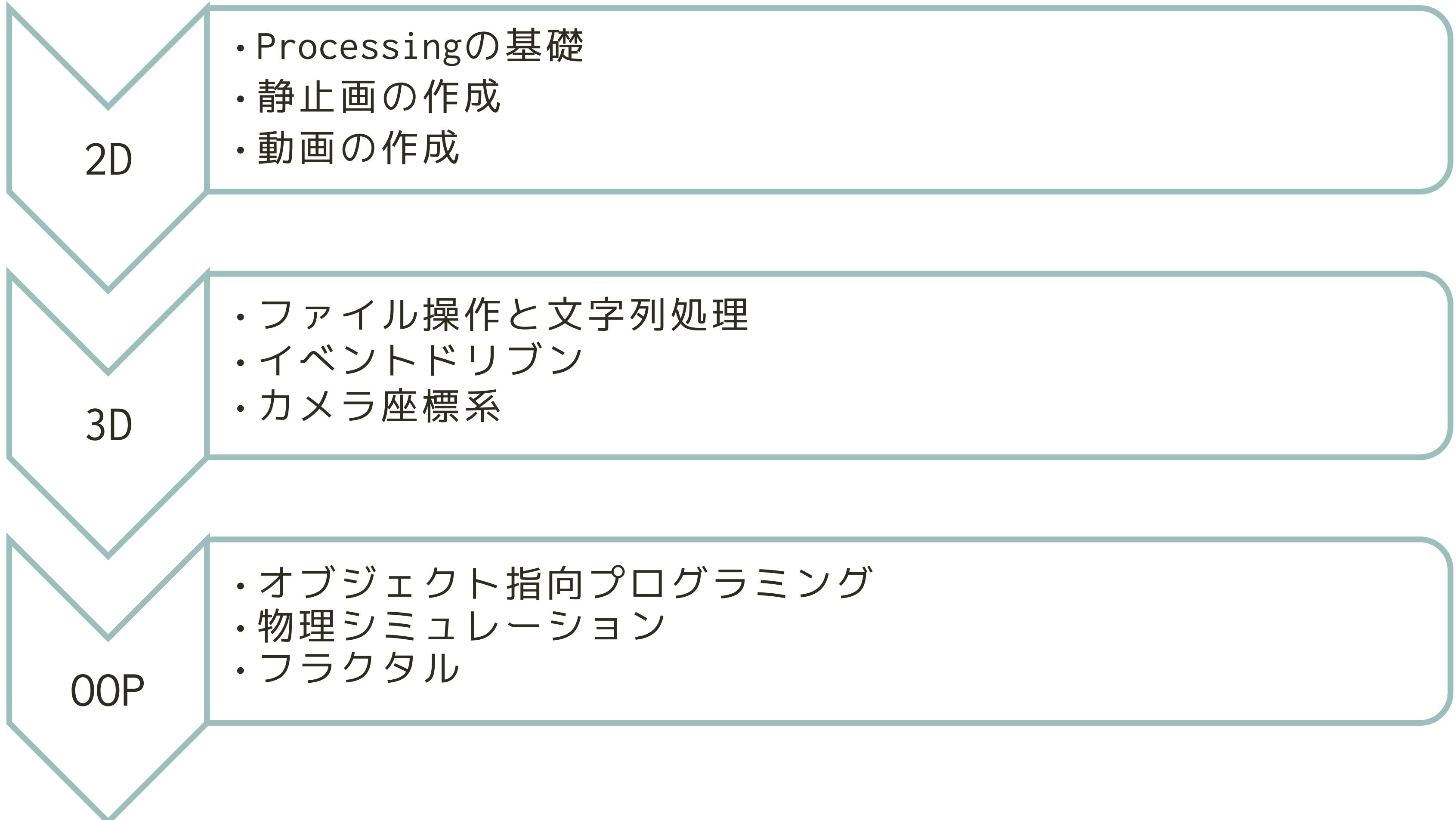
● 講義の流れ(総合演習回を除く)

- ① 前回課題の解説
- ② 学習内容の説明
- ③ 課題に取り組む(すべての課題を終えれば退出可)

● 教員

- 水山先生(第3部: 11~15回)
- 高橋先生(第2部: 6~10回)
- 佐藤 (第1部: 1~ 5回)

講義計画(変更もあり得る)



開発環境

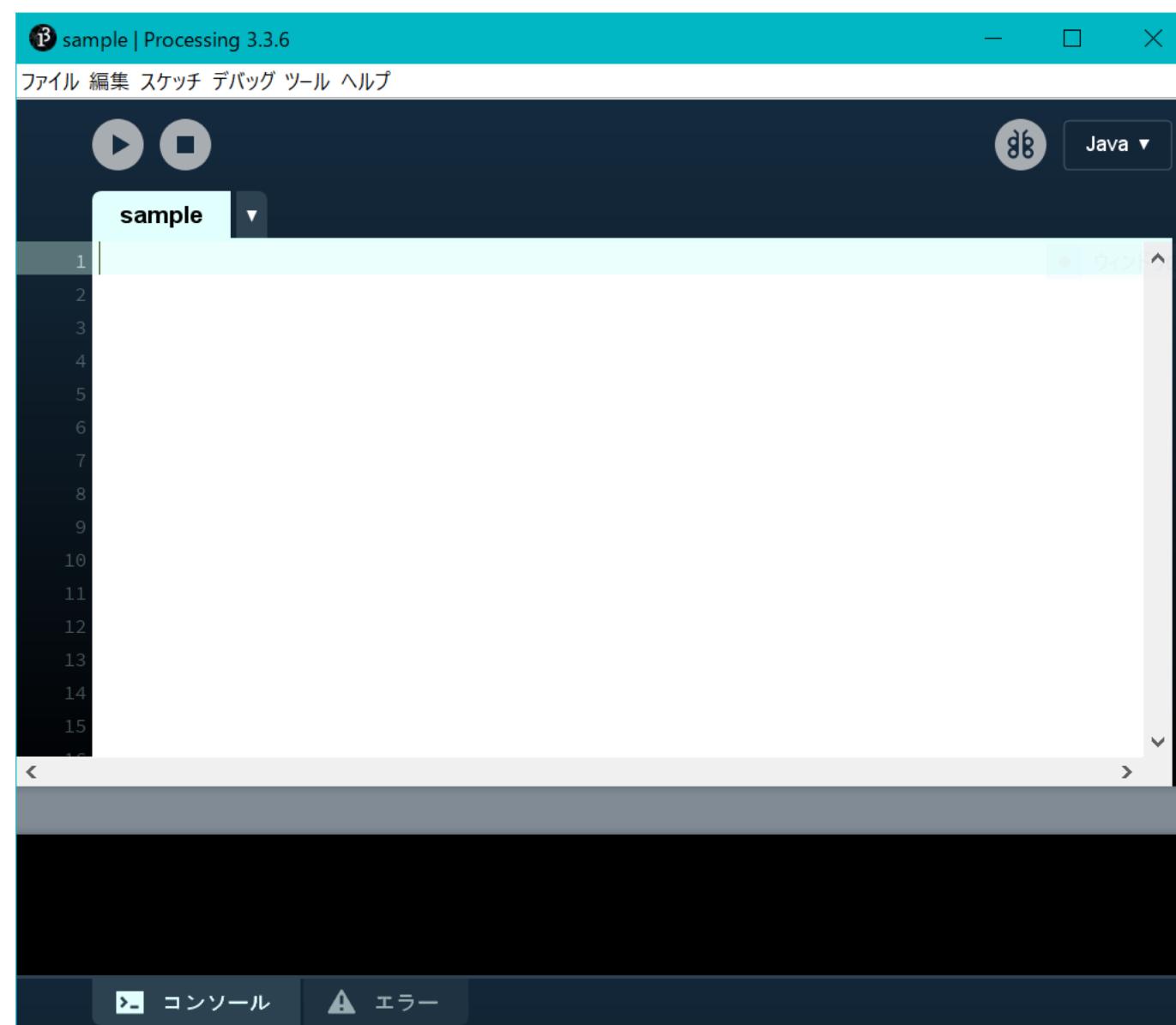


2018年4月10日(火)

PDE

● Processing開発環境(Processing Development Environment: PDE)

- メニュー
- ツールバー
- タブ
- テキストエディタ
- メッセージエリア
- コンソール



メニュー

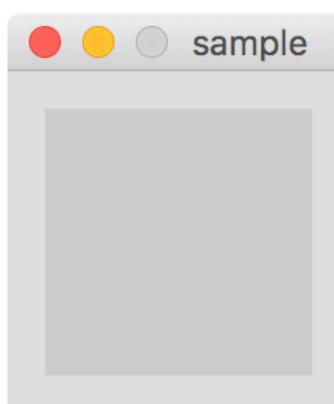
テキストエディタ

コンソール

メッセージエリア

スケッチ

- Processingで作成されたプログラムを **スケッチ** という
- スケッチの作成手順
 - ① コーディング
 - テキストエディタにコードを書く
 - ② 実行 ⇄ デバッグ
 - ツールバーの「▶」ボタンを押すと実行
 - ツールバーの「■」ボタンを押すと停止
 - ③ 保存
 - メニューの「ファイル」→「名前を付けて保存」
 - そのスケッチ専用のフォルダ(**スケッチフォルダ**)が作成される
- スケッチを実行すると **ディスプレイウィンドウ** と呼ばれる新規ウィンドウが開く



実行したままデバッグ可能。ただし、再度実行しないと反映されない

全部完成してから保存するのではなく、コードを書いたところまでを保存しながら作成すること！

小演習(1)

- 次の3つを順に行なさい

- ① Processingを起動し， PDEの構成を確認しなさい
- ② ツールバーの▶を押すとディスプレイウィンドウが表示されることを確認しなさい
- ③ ツールバーの■を押すとディスプレイウィンドウが閉じられることを確認しなさい

スケッチの管理

【重要】

● 本講義で作成するスケッチの保存場所

- Z:\My Documents\Processing\computerProgramming3\xyyyyyy_zz
 - ① ZドライブのMy Documentsフォルダの直下に「Processing」フォルダを作成
 - ② 「Processing」フォルダの直下に「computerProgramming3」フォルダを作成
 - ③ 「computerProgramming3」フォルダの直下に「xyyyyyyy_zz」フォルダを作成

● xyyyyyy_zz

- x: 自分の青山メールの最初の文字
- yyyyyyy: 自分の青山メールの2文字目以降の数字の並び
 - (例)1234567
- zz: 講義回
 - (例)第1回 → 「01」

● 「xyyyyyyy_zz」フォルダの直下に毎回の講義で作成する各スケッチを保存

● 保存したスケッチの開き方

- スケッチフォルダの直下にある拡張子が「.pde」のファイルを開く

小演習(2)

- 本講義で作成するスケッチフォルダの保存場所を全15回分作成しなさい
 - 完成したら教員またはTAに確認してもらうこと

小演習(3)

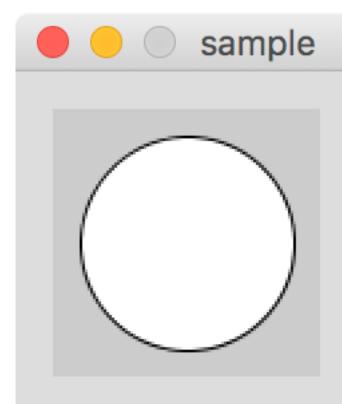
円を描く

- テキストエディタ内に次のコードを記述して実行・保存してみよう!

- スケッチフォルダ名: 「pr01_01」

```
ellipse(50, 50, 80, 80);
```

- 中心が左から50ピクセル、上から50ピクセルの位置にある、幅80ピクセル、高さ80ピクセルの円を描く



コード補完

● 準備

- ・メニューの「ファイル」→「設定」で開かれるウィンドウの中の「コードの補完」にチェックを入れると使用可能になる

● (例) 関数のコード補完

- ・Processingが用意する関数を使う際、1文字以上打ち込んだ後に**Ctrl+スペース**で関数の候補リストが表示される
 - 候補リストの表示を消すにはEscキーを押す
- ・上下の矢印キーで使いたい関数を選択すると、コード補完が行われる

小演習(4)

- メニューの「ファイル」→「新規」で新しいPDEのウィンドウを開きなさい。その後、「e」と1文字打った時点でコード補完を行い、`ellipse()`を補完しなさい
 - 以下のコードが補完されれば成功

```
ellipse(,,,)
```

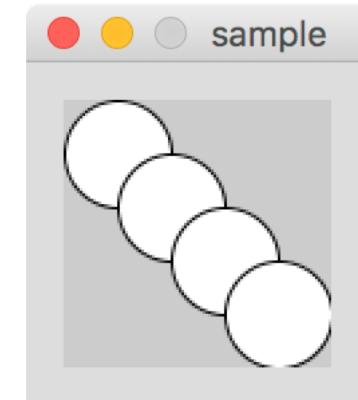
識別子の変更

- 外部から見たプログラムの振る舞いを変えずにプログラム内部の構造を改善することをリファクタリングという
- リファクタリングの1つに識別子(変数名や関数名などの名前)の変更がある
- Processingのエディタはスケッチ全体に渡って識別子の一括変更を行う機能がある
 - 識別子の上にカーソルを持って行く → 右クリック → 「名前の変更…」 → 現れるダイアログに新しい変数名を入力 → 「Rename」ボタンを押す

小演習(5)

- テキストエディタ内に次のコードを記述して実行しなさい。その後、識別子の一括変更機能を使ってループ変数iをjに変更して実行・保存しなさい
 - スケッチフォルダ名：「pr01_02」

```
for (int i = 1; i <= 4; i++) {  
    ellipse(i * 20, i * 20, 40, 40);  
}
```



```
for (int j = 1; j <= 4; j++) {  
    ellipse(j * 20, j * 20, 40, 40);  
}
```



```
for (jnt j = 1; j <= 4; j++) {  
    ellipse(j * 20, j * 20, 40, 40);  
}
```

Ctrl+Fでワード検査/置換機能を使い「検索」にi、「置換」にjと入力して「すべて置換」ボタンを押した結果、変更する必要な無いintのiまでjになってしまふ！

デバッグ

- メニューの「デバッグ」→「デバッグを有効化」でデバッグが起動する
 - ツールバーに以下2つのボタンが追加される
 - ステップ(▶|): コードを1行ずつ実行
 - 続行(|▶): 次のブレークポイントまで実行
 - 「Variables」というウィンドウが表示される
 - 変数の値が確認できる
- 実行中のプログラムを一時停止させる箇所(行単位)をブレークポイントという
- ブレークポイントの設定/解除手順
 - ① ブレークポイントを設定/解除したい行を選択する(カーソルを移動してクリック)
 - ② メニューの「デバッグ」→「ブレークポイントを切り替え」
 - コードの左側が「◆」になれば設定, 行番号になれば解除

小演習(6)

- pr01_02を開きデバッグモードを起動しなさい。次に、ellipse()が書かれた行にブレークポイントを設定しなさい。その後、ステップ実行して変数jの値の変化をVariablesウィンドウで確認しなさい

リファレンス

- Processingには公式のリファレンスが存在し，Webページとして公開されている
 - リファレンスには，Processingで用いられる様々な機能について個別に詳細な説明や使い方が書かれている
- PDEからリファレンスへアクセス可能
 - メニューの「ヘルプ」→「リファレンス」

小演習(7)

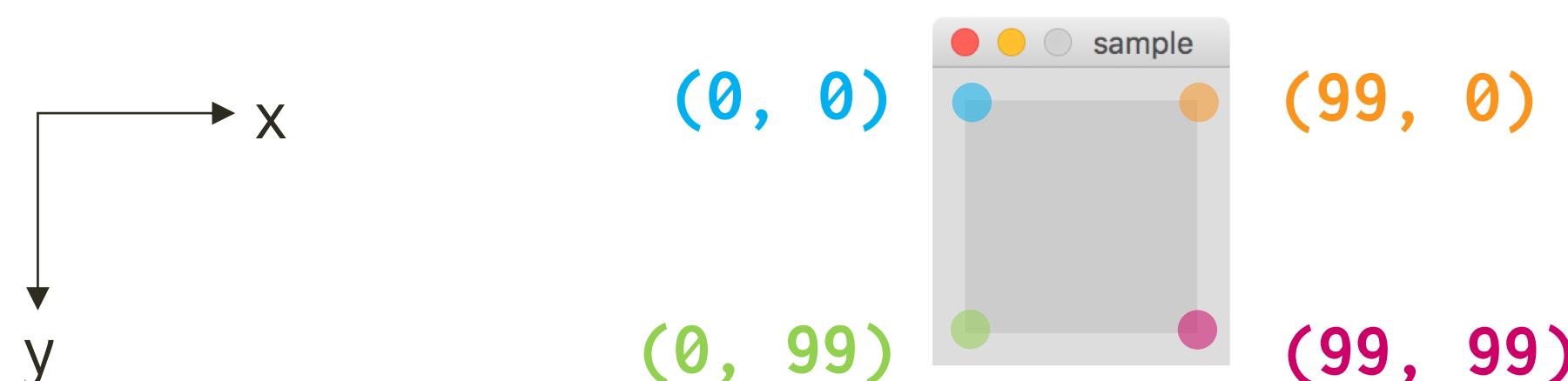
- `ellipse()`の詳細をリファレンスで調べなさい

座標，線，基本的な図形

2018年4月10日(火)

座標

- コンピュータの画面は、**ピクセル**と呼ばれる発光体が格子状に並んだものであり、各ピクセルは**座標**で示すことができる
- ピクセルの座標を (x, y) と表す
 - x : ディスプレイウィンドウの左端からの距離(ピクセル数)
 - y : ディスプレイウィンドウの上端からの距離(ピクセル数)
- (例)ディスプレイウィンドウのサイズが 100×100 ピクセル
 - 左上隅の座標: $(0, 0)$
 - 右下隅の座標: $(99, 99)$

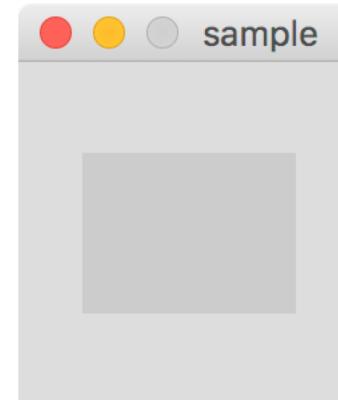


size(), point()

- スケッチの構成単位は **関数** であり、あらかじめ用意された関数を使って基本的な図形を描くことができる
- **size()** は、ディスプレイウィンドウの幅と高さを設定する関数
 - ・ 第1引数：幅
 - ・ 第2引数：高さ

例

```
size(80, 60);
```

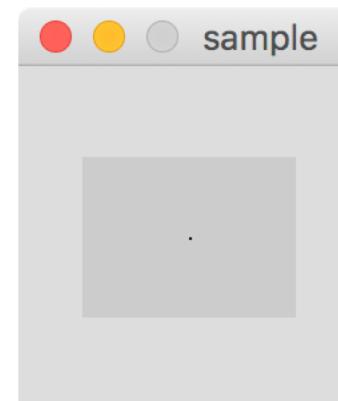


size()が指定されていない場合、ディスプレイウィンドウのサイズは **100×100** ピクセルになる

- **point()** は、ディスプレイウィンドウ内に点を描く関数
 - ・ 第1引数：x座標
 - ・ 第2引数：y座標

例

```
size(80, 60);
point(40, 30);
```



原則として設定サイズとディスプレイウィンドウのサイズは一致する。しかし、ディスプレイウィンドウの最小サイズはOS依存であるため、最小サイズより小さいサイズを設定した場合、描画領域は最小サイズのディスプレイウィンドウの一部分になる

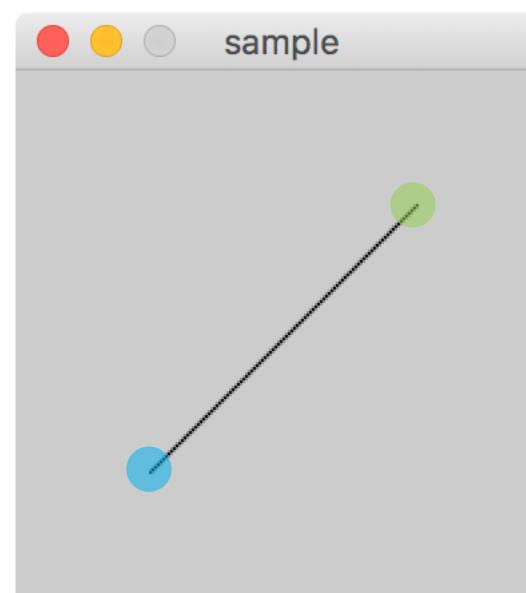
line()

- **line()**は、2点を結ぶ線を描く関数

- 第1引数: 点のx座標, 第2引数: 点のy座標
- 第3引数: 点のx座標, 第4引数: 点のy座標

例

```
size(200, 200);
line(50, 150, 150, 50);
```

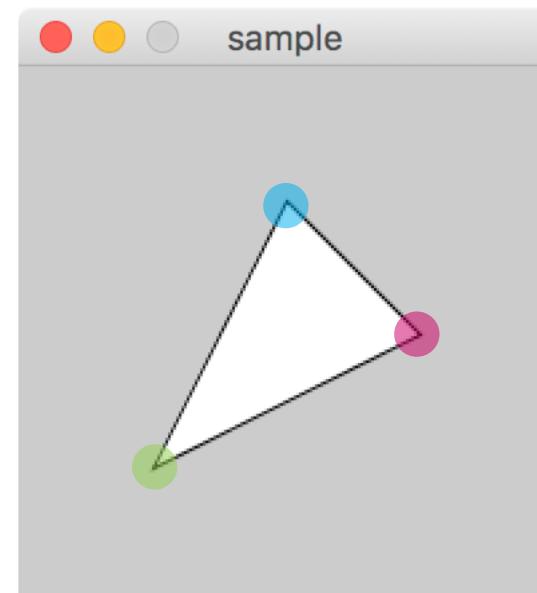


triangle(), quad()

- **triangle()**は，3点を結ぶ三角形を描く関数
 - 6つの引数(3つの点)
- **quad()**は，4点を結ぶ四角形を描く関数
 - 8つの引数(4つの点)

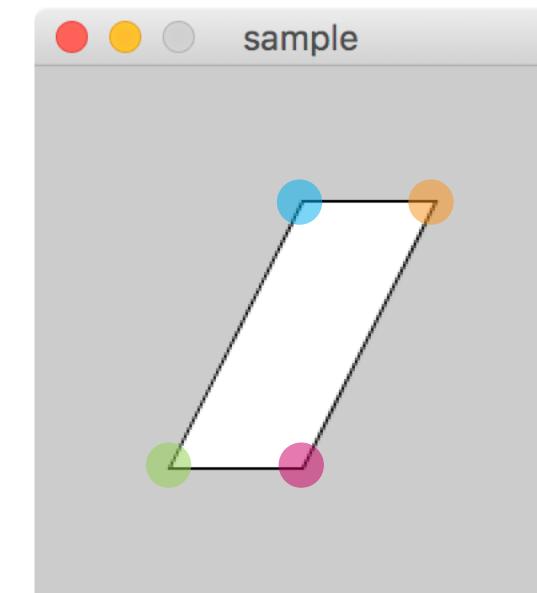
例

```
size(200, 200);
triangle(100, 50, 50, 150, 150, 100);
```



例

```
size(200, 200);
quad(100, 50, 50, 150, 100, 150, 150, 50);
```



rect(), ellipse()

● rect() は、長方形を描く関数

- ・第1引数：左上の角の点のx座標，第2引数：左上の角の点のy座標
- ・第3引数：幅
- ・第4引数：高さ

● ellipse() は、橢円を描く関数

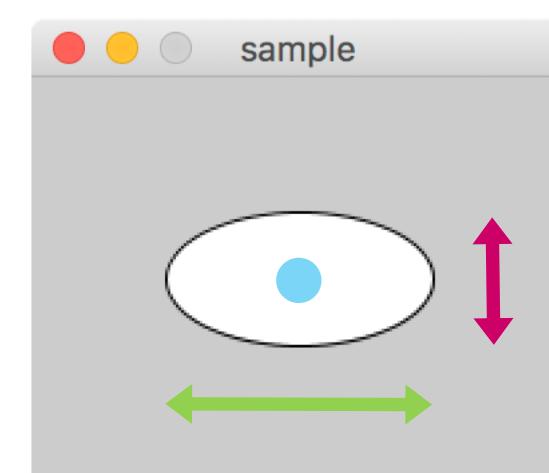
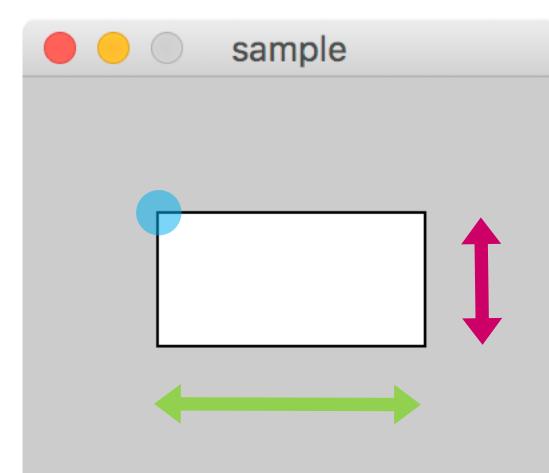
- ・第1引数：中心のx座標，第2引数：中心のy座標
- ・第3引数：幅
- ・第4引数：高さ

例

```
size(200, 150);
rect(50, 50, 100, 50);
```

例

```
size(200, 150);
ellipse(100, 75, 100, 50);
```



rectMode()

- **rectMode()**は、 rect()の基準点の位置を変更する関数。引数は1つで4つの種類がある

- **CORNER**

- rect()の第1引数と第2引数が表す基準点を長方形の左上の角の点にする
 - **デフォルト**

- **CORNERS**

- rect()の第1引数と第2引数が表す基準点を長方形の一方の角の点にし，第3引数と第4引数が表す基準点を反対側の角の点にする

- **CENTER**

- rect()の第1引数と第2引数が表す基準点を，第3引数が幅，第4引数が高さの長方形の中心にする

- **RADIUS**

- rect()の第1引数と第2引数が表す基準点を，第3引数が半分の幅，第4引数が半分の高さの長方形の中心にする

ellipseMode()

- **ellipseMode()**は、 ellipse()の基準点の位置を変更する関数。引数は1つで4つの種類がある
 - **CORNER**
 - ellipse()の第1引数と第2引数が表す基準点を楕円の外接長方形(バウンディングボックス)の左上の角の点にする
 - **CORNERS**
 - ellipse()の第1引数と第2引数が表す基準点を楕円の外接長方形(バウンディングボックス)の一方の角の点にし、 第3引数と第4引数が表す基準点を反対側の角の点にする
 - **CENTER**
 - ellipse()の第1引数と第2引数が表す基準点を、 第3引数が幅、 第4引数が高さの楕円の中心にする
 - **デフォルト**
 - **RADIUS**
 - ellipse()の第1引数と第2引数が表す基準点を、 第3引数が半分の幅、 第4引数が半分の高さの楕円の中心にする

arc(1)

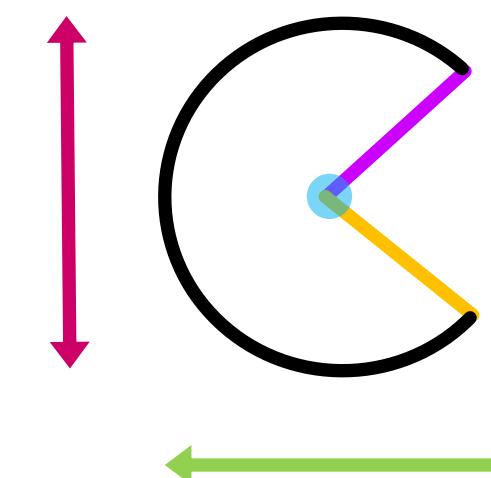
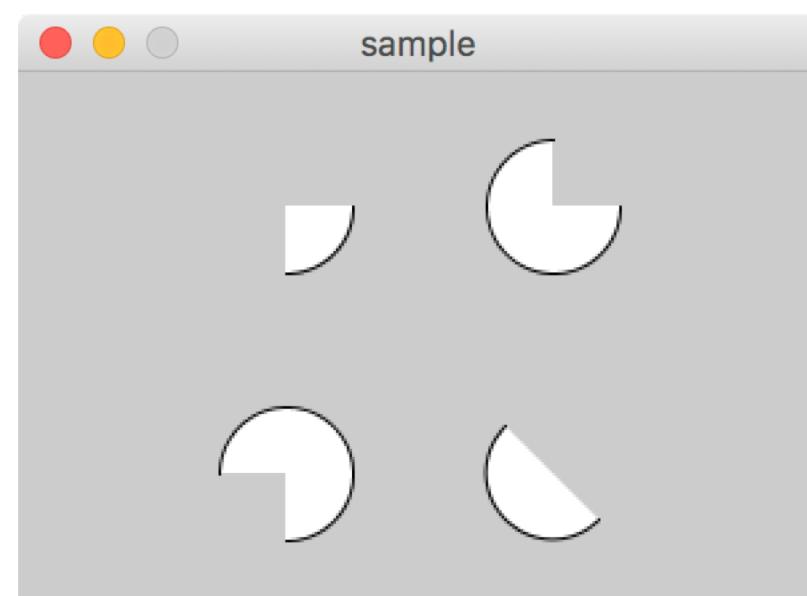
arc()の基本的な使い方

- **arc()**は、弧(円周の一部分)を描く関数

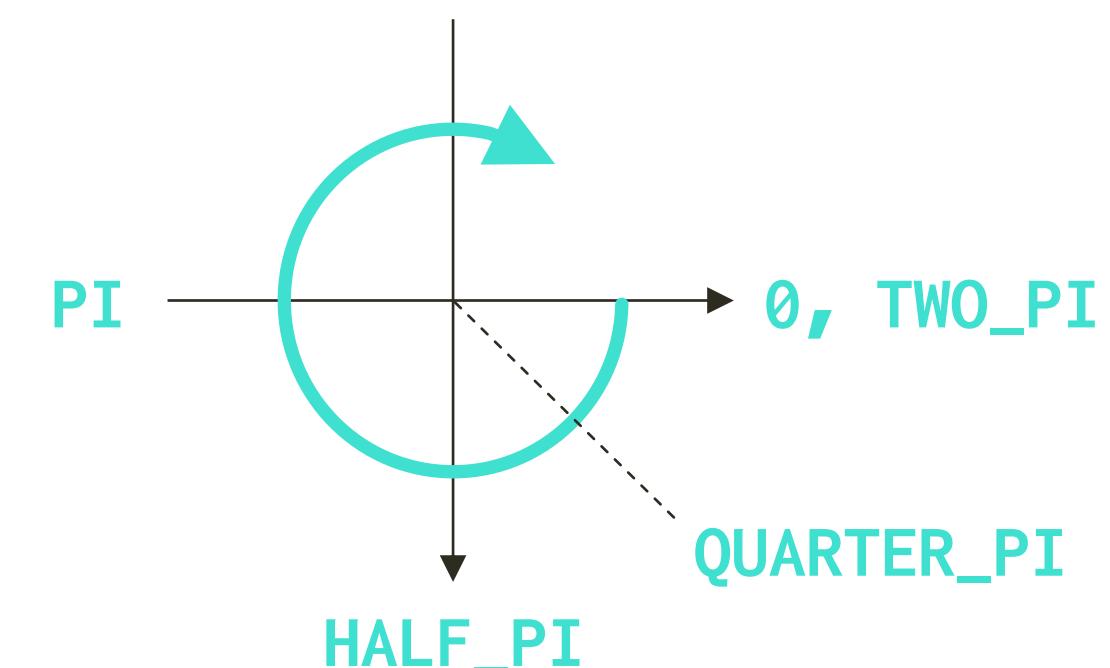
- 第1引数: 中心のx座標, 第2引数: 中心のy座標
- 第3引数: 幅, 第4引数: 高さ
- 第5引数: 弧の始まりの角度[rad], 第6引数: 弧の終わりの角度[rad]
 - 第5引数の値 \leq 第6引数の値

例

```
size(300, 200);
arc(100, 50, 50, 50, 0, HALF_PI);
arc(200, 50, 50, 50, 0, PI + HALF_PI);
arc(100, 150, 50, 50, PI, TWO_PI + HALF_PI);
arc(200, 150, 50, 50, QUARTER_PI, PI + QUARTER_PI);
```



PI + HALF_PI



arc(2)

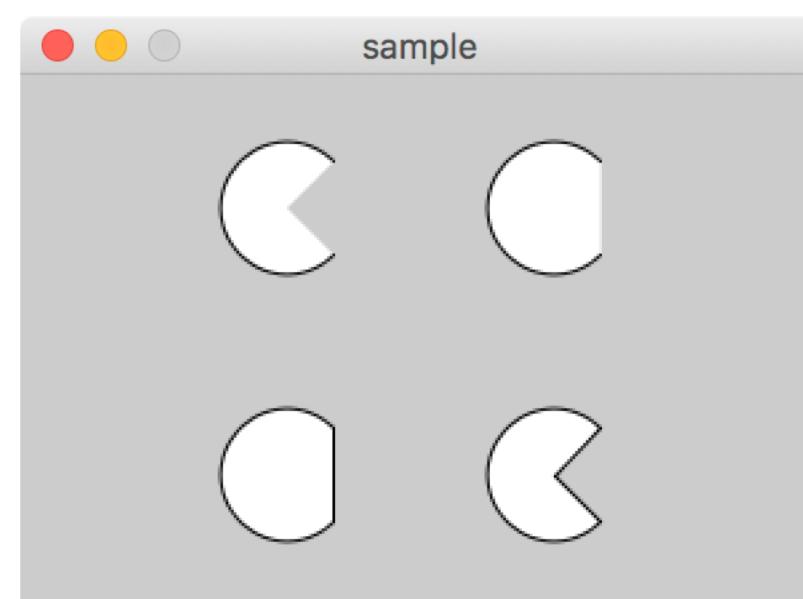
arc()のモード

- Processingでは、同じ名前で引数の数が異なる関数が存在する
 - (例)arc()は、7番目の引数を指定することで描き方を変えることができる
 - 7番目の引数: モード

- OPEN
- CHORD
- PIE

例

```
size(300, 200);
arc(100, 50, 50, 50, QUARTER_PI, TWO_PI - QUARTER_PI);
arc(200, 50, 50, 50, QUARTER_PI, TWO_PI - QUARTER_PI, OPEN);
arc(100, 150, 50, 50, QUARTER_PI, TWO_PI - QUARTER_PI, CHORD);
arc(200, 150, 50, 50, QUARTER_PI, TWO_PI - QUARTER_PI, PIE);
```



radians()

- **radians()**は、角度を「度」から「ラジアン」へ変換する関数

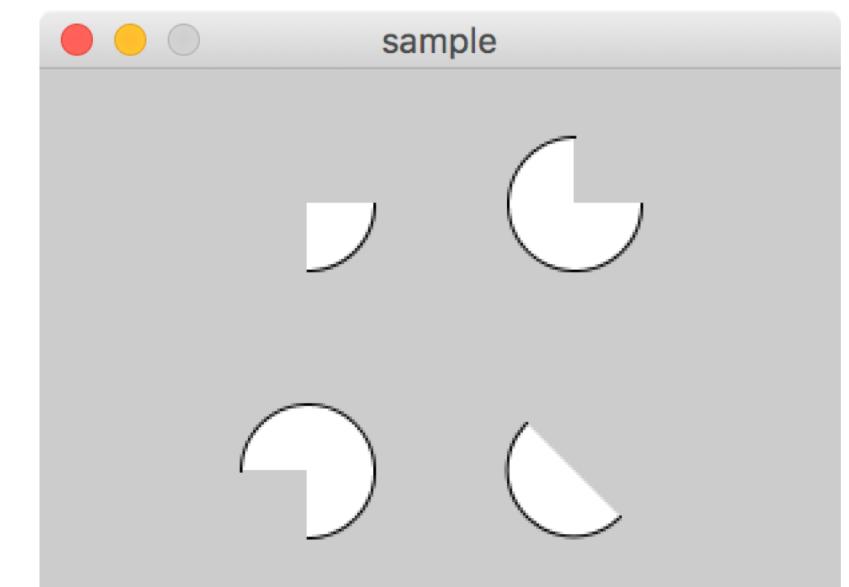
例

```
size(300, 200);
arc(100, 50, 50, 50, 0, HALF_PI);
arc(200, 50, 50, 50, 0, PI + HALF_PI);
arc(100, 150, 50, 50, PI, TWO_PI + HALF_PI);
arc(200, 150, 50, 50, QUARTER_PI, PI + QUARTER_PI);
```



例

```
size(300, 200);
arc(100, 50, 50, 50, 0, radians(90));
arc(200, 50, 50, 50, 0, radians(270));
arc(100, 150, 50, 50, radians(180), radians(450));
arc(200, 150, 50, 50, radians(45), radians(225));
```



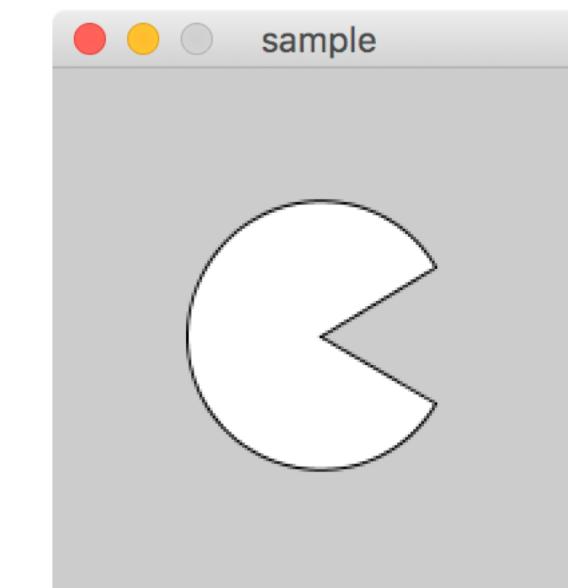
setup()

- Processingは、`setup()`の中のコードを**最初に一度**だけ実行する
- `setup()`を用いることで自作関数が定義できるようになる

例

```
void setup() {  
    size(200, 200);  
    drawPackman(width / 2, height / 2,  
                width / 2, height / 2,  
                30, 330);  
}  
  
void drawPackman(float x, float y,  
                 float w, float h,  
                 float aStart, float aEnd) {  
    arc(x, y,  
        w, h,  
        radians(aStart), radians(aEnd),  
        PIE);  
}
```

setup()の中でsize()を書く場合、必ず**1行目**にすること！



Processingであらかじめ用意されている図形を描画する関数の引数の型は基本的に**float**(決めてからずにリファレンスでチェックすること！)

多角形，曲線



2018年4月10日(火)

beginShape(), endShape()

- 複数の点をつなぎ合わせて複雑な形を描く

① **beginShape();**

- 新しい形を描き始める合図

② **vertex();** (複数回)

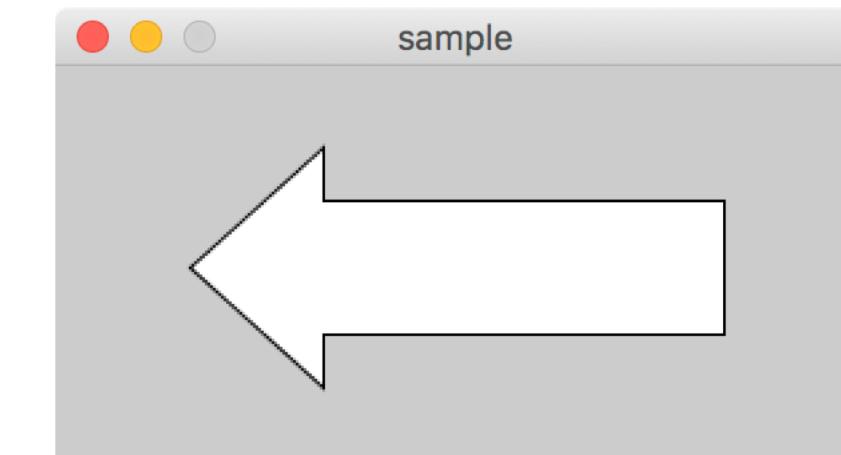
- 図形の輪郭を表す点
- 引数に座標を指定

③ **endShape();**

- 描き終わりの合図

例

```
size(300, 150);
beginShape();
vertex(50, 75);
vertex(100, 30);
vertex(100, 50);
vertex(250, 50);
vertex(250, 100);
vertex(100, 100);
vertex(100, 120);
vertex(50, 75);
endShape(CLOSE);
```



endShape()の引数にCLOSEを指定すると、最初の点と最後の点が直線で結ばれ閉じた形になる

beginShape()のパラメータ

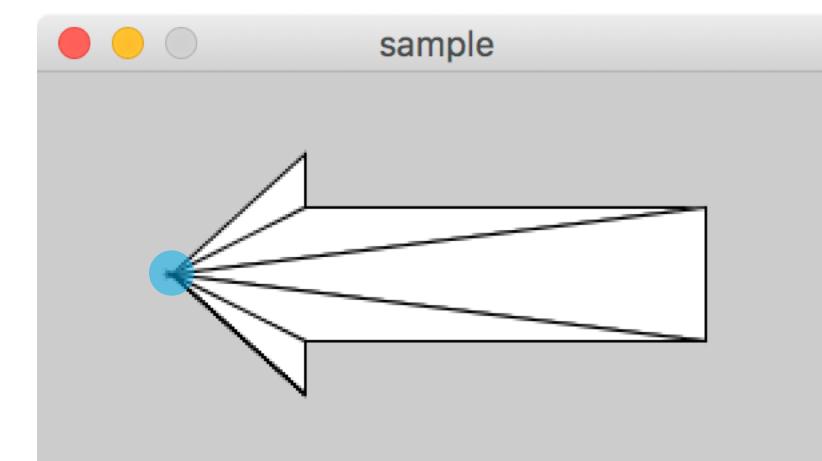
- beginShape()には、次のパラメータが指定可能

- POINTS
- LINES
- TRIANGLES
- TRIANGLE_FAN
- TRIANGLE_STRIP
- QUADS
- QUADS_STRIP

例

```
size(300, 150);
beginShape(TRIANGLE_FAN);
vertex(50, 75);
vertex(100, 30);
vertex(100, 50);
vertex(250, 50);
vertex(250, 100);
vertex(100, 100);
vertex(100, 120);
vertex(50, 75);
endShape();
```

最初の点とそれ以外のすべての点の間に線が引かれる



curve()(1)

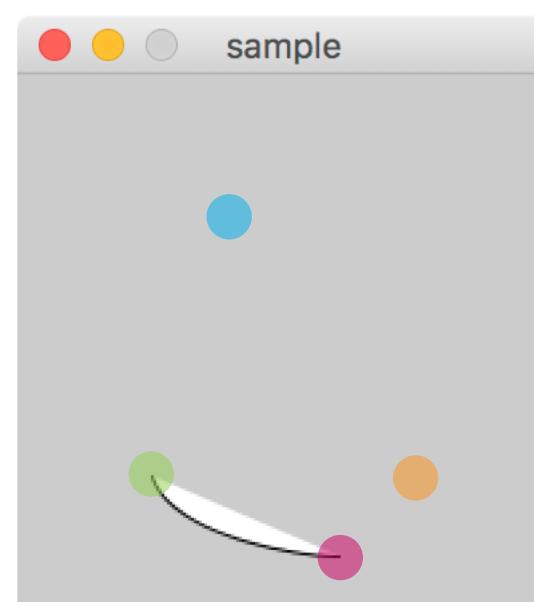
- **curve()**は、曲線を描く関数

- 8つの引数で4つの点の座標を指定
- 指定された4点を結ぶ曲線の**中間の2点の間**の部分を描く

例

```
size(200, 200);
curve(80, 50, 50, 150, 120, 180, 150, 150);
```

曲線の形状を定義する点を**制御点(コントロールポイント)**という。curve()では**最初の点と最後の点**が制御点



中間の2点を結ぶ部分以外の曲線は描かれない！

curve()(2)

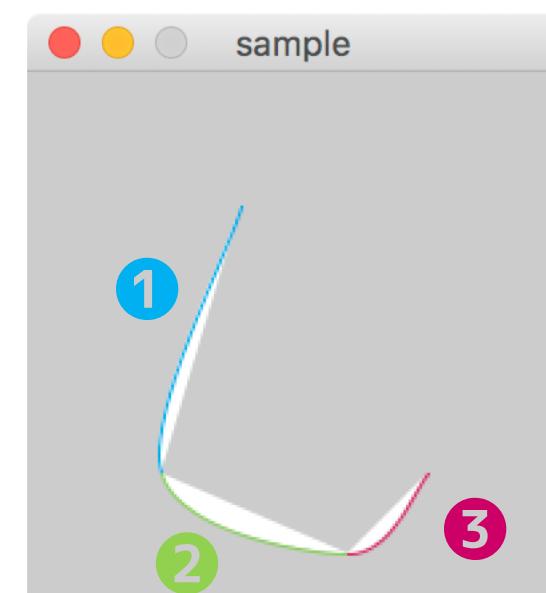
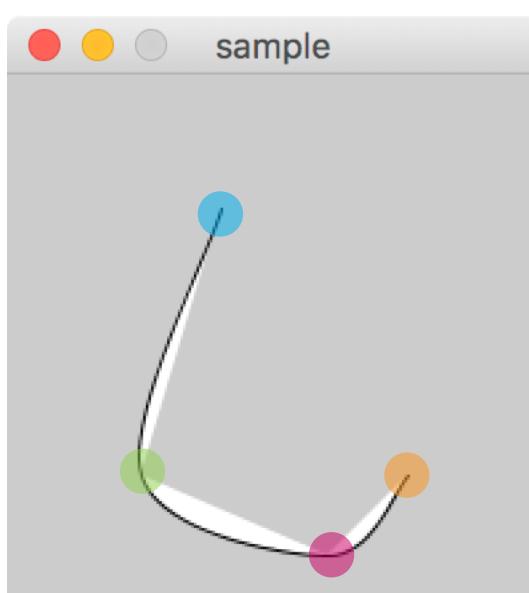
曲線全体を描く方法

- 次のようにしてcurve()の曲線全体を描くことができる

例

```
size(200, 200);
curve(80, 50, 80, 50, 50, 150, 120, 180);      // ①
curve(80, 50, 50, 150, 120, 180, 150, 150); // ②
curve(50, 150, 120, 180, 150, 150, 150, 150); // ③
```

曲線を構成する点に沿ってcurve()を繰り返し使い徐々に曲線を描いていく。最初と最後の点(制御点)は**2回**続ける。これらを除く点は、引数の順番が1つずつ繰り上がる



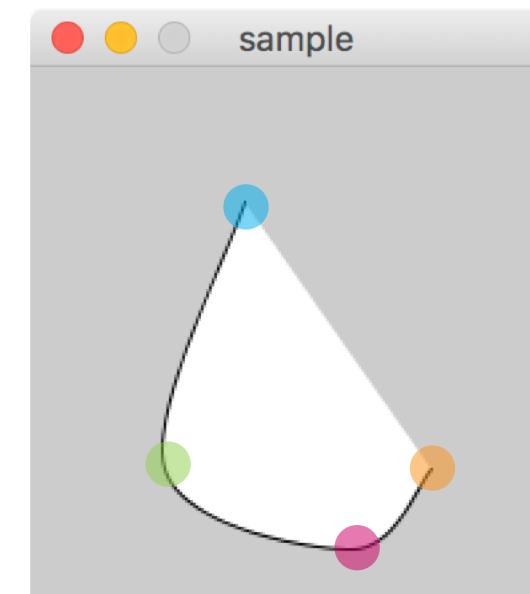
curveVertex()

- vertex()をcurveVertex()になると，点の間が曲線で結ばれる

例

```
size(200, 200);
beginShape();
curveVertex(80, 50);
curveVertex(80, 50);
curveVertex(50, 150);
curveVertex(120, 180);
curveVertex(150, 150);
curveVertex(150, 150);
endShape();
```

curve()と同様に，
制御点は2回続ける！

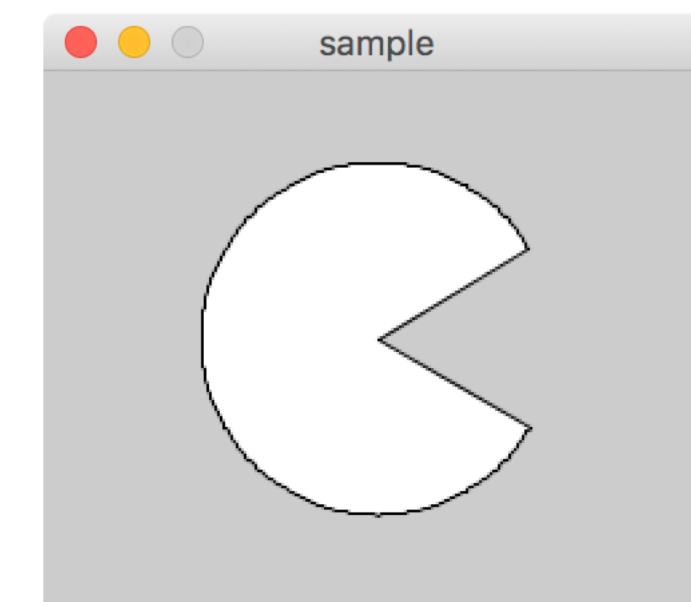


vertex()とcurveVertex()の併用

- vertex()とcurveVertex()を切り替えることで、直線と曲線が混在する図形を描くことができる

例

```
boolean isCrlPnt = true; // control point
float pX = 0, pY = 0; // pointX, pointY
size(250, 200);
float r = height / 3; // radius
beginShape();
vertex(width / 2, height / 2);
for (int i = 30; i < 330; i++) {
    float theta = radians(float(i));
    pX = width / 2 + r * cos(theta);
    pY = height / 2 + r * sin(theta);
    if (isCrlPnt) {
        vertex(pX, pY);
        isCrlPnt = false;
    }
    curveVertex(pX, pY);
}
curveVertex(pX, pY);
vertex(width / 2, height / 2);
endShape();
```



- 直線→曲線
 - vertex(x, y); → curveVertex(x, y);
- 曲線→直線
 - curveVertex(x, y); → vertex(x, y);

切り替え所の点を2回続ける

補足



2018年4月10日(火)

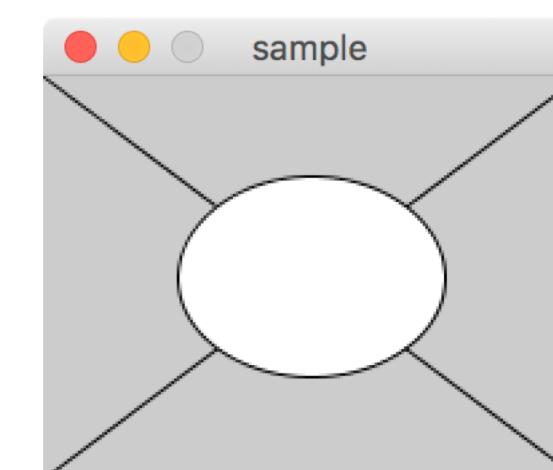
システム変数(1)

width, height

- Processingであらかじめ用意されている、変数宣言なしで使える特殊な変数(システム変数)にwidthとheightがある
 - width: ウィンドウの幅
 - height: ウィンドウの高さ
 - いずれもint型
- size()が実行されると、自動的にwidthとheightに値が設定される
 - size()を省略した場合、自動的に「size(100, 100)」が実行される
→ widthとheightの値は、それぞれ100になる

例

```
size(200, 150);
line(0, 0, width, height);
line(width, 0, 0, height);
ellipse(width / 2, height / 2, width / 2, height / 2);
```



算術関数

関数名	機能
abs(a)	aの絶対値を求める
sqrt(a)	aの平方根を求める
sq(a)	aの2乗を求める
pow(a, b)	aのb乗を求める
exp(a)	e(ネイピア数)のa乗を求める
log(a)	自然対数を求める
round(a)	小数点以下を四捨五入する
floor(a)	小数点以下を切捨てる
ceil(a)	小数点以下を切上げる
min(a, b)	最小値を求める
max(a, b)	最大値を求める
constrain(a, min, max)	aをminとmaxの間に制限する
mag(x, y)	原点から座標(x, y)までの距離を求める
dist(x1, y1, x2, y2)	2点間の距離を求める
lerp(a, b, c)	aとbの間の比cで指定される値を求める
norm(a, b, c)	aを範囲b-cから0-1の範囲へ変換する
map(a, b, c, d, e)	aを範囲b-cから別の範囲d-eへ変換する

minとmaxには配列を渡すこともできる!

三角関数

引数の順番に
注意(yが先)!

関数名	機能
$\sin(a)$	正弦(サイン)を求める
$\cos(a)$	余弦(コサイン)を求める
$\tan(a)$	正接(タンジェント)を求める
$\text{asin}(a)$	逆正弦(アークサイン)を求める
$\text{acos}(a)$	逆余弦(アークコサイン)を求める
$\text{atan}(a)$	逆正接(アークタンジェント)を求める
$\text{atan2}(y, x)$	座標を指定して角度を求める
$\text{degrees}(a)$	角度の単位を「ラジアン」から「度」へ変換する
$\text{radians}(a)$	角度の単位を「度」から「ラジアン」へ変換する

最後の「s」を忘れずに!

型変換関数

関数名	機能
char()	文字(1文字)に変換
str()	文字列に変換
int()	整数に変換
float()	浮動小数点数に変換
byte()	1バイトのデータに変換
boolean()	0はfalseに， その他はtrueに変換