情報メディア基盤ユニット

座標変換、変数の有効範囲、関数

情報メディア学科佐藤尚

今日の内容

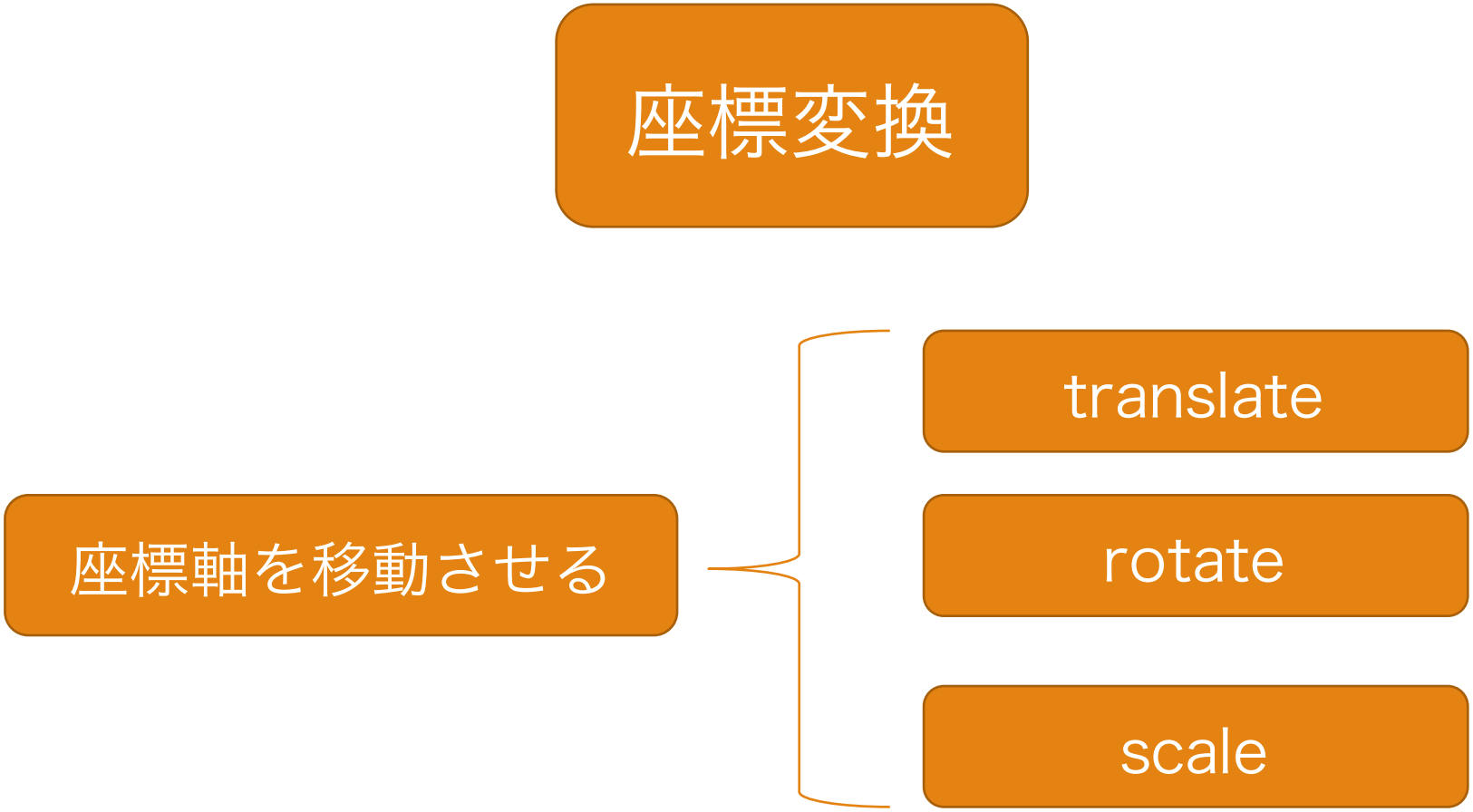
プログラムが動かない-Σヽ(｀д´;)ノ うぉぉぉ！

座標変換
（前回の続き）

変数の有効範囲

関数その1

座標変換



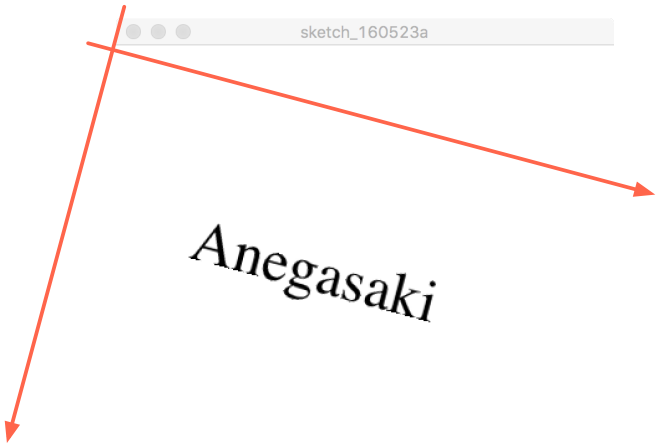
座標軸

図形や文字列を表示する時の基準

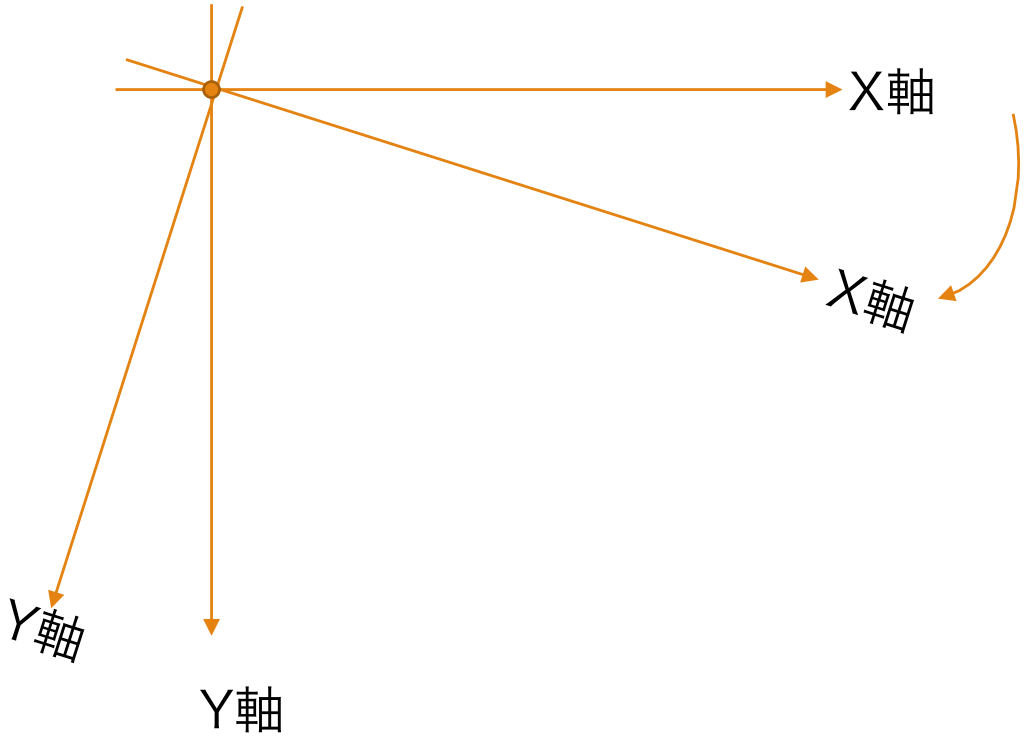
```
text("Anegasaki",100,height/2);
```

デフォルトの座標軸

座標軸を15度回転
rotate(radians(15))
rotate(PI/12);

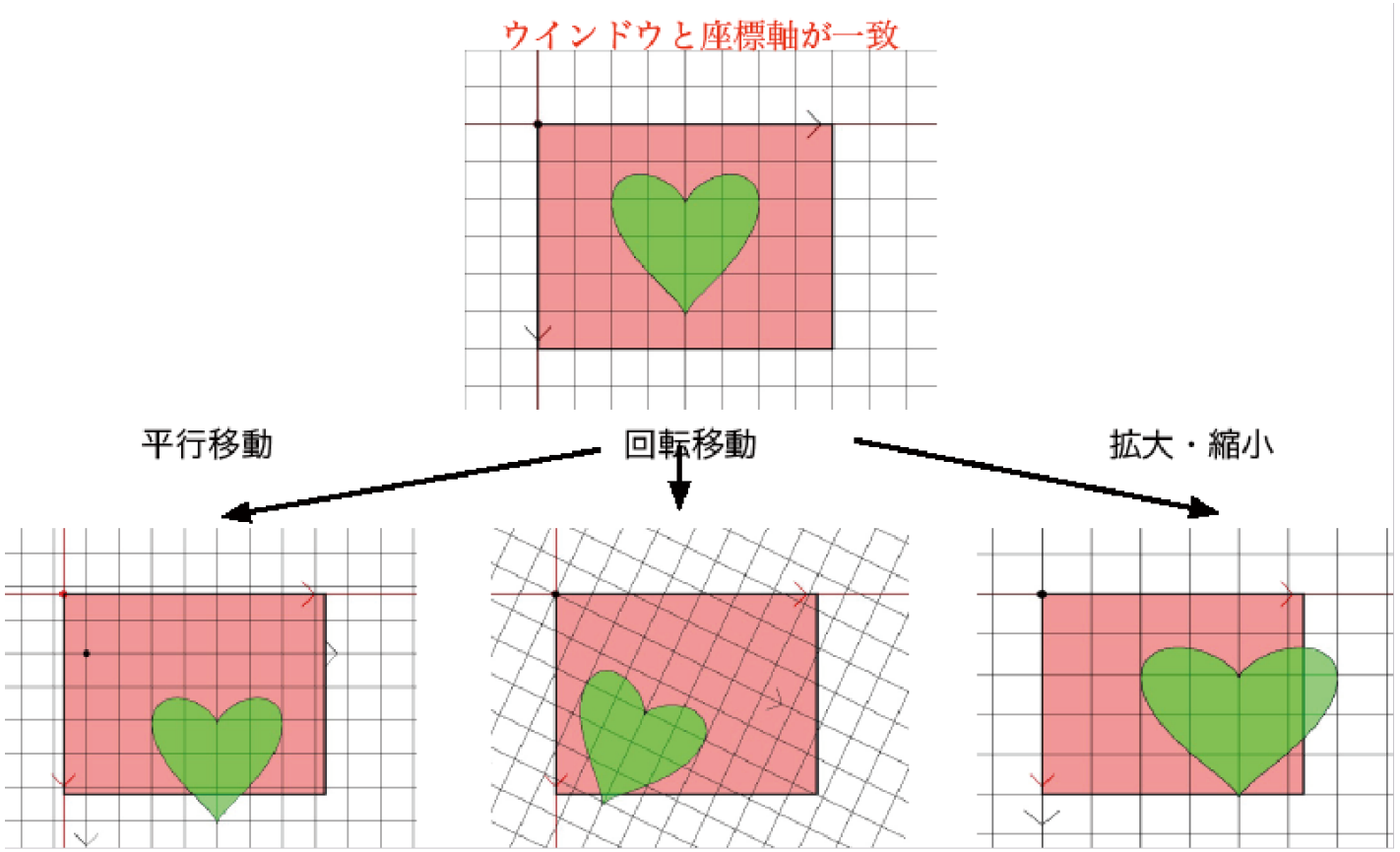


回転角度の正の方向



こっち向きが
正の方向

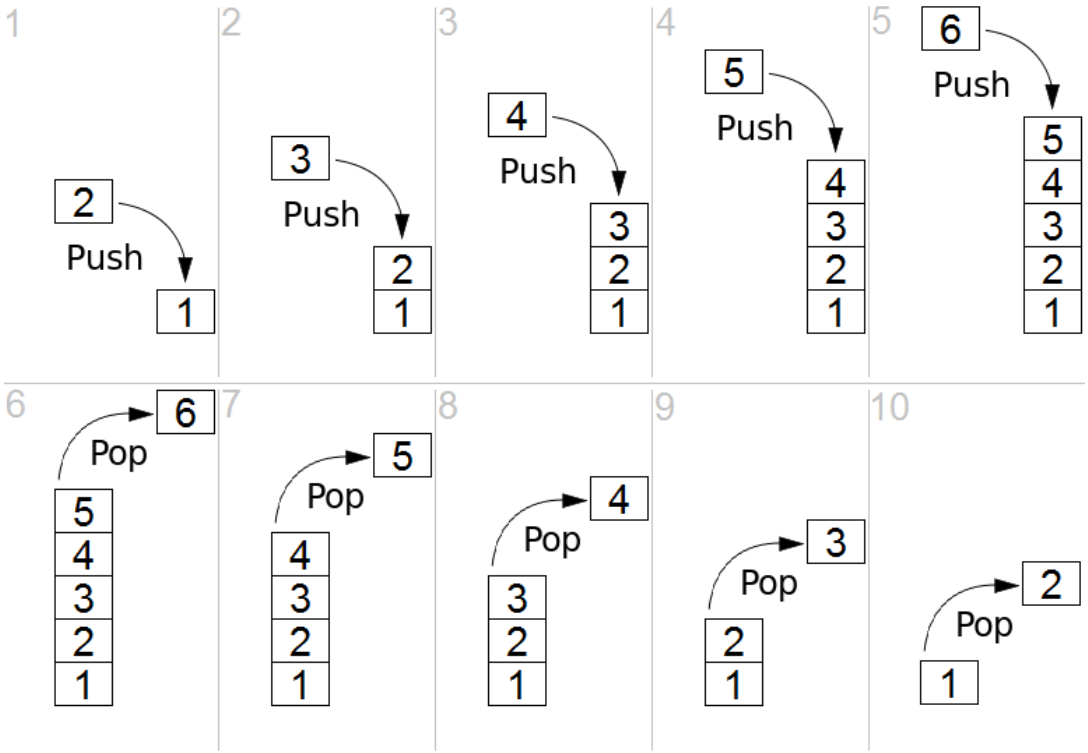
座標変換



関数名など	説明
hour()	現在の時間の時（0～23の整数）を返す関数。
minute()	現在の時間の分(0～59の整数)を返す関数。
second()	現在の時間の秒(0～59の整数)を返す関数。
year()	現在の年を返す関数。
month()	現在の月(1～12の整数)を返す関数。
day()	現在の日(1～31の整数)を返す関数。
millis()	プログラムを実行してからの時間をミリ秒単位で返す。

関数名など	説明
translate	座標軸の原点を移動する
rotate	現在の原点の場所を中心に座標軸を回転させる
scale	座標軸の目盛の基準を拡大・縮小させる
resetMatrix	座標軸の位置を初期状態に戻す ウインドの左上が原点、X軸は水平方向、Y軸は垂直方向
pushMatrix	現在の座標軸の状態をスタックに積む
popMatrix	座標軸の状態をスタックの先頭の状態にする、 スタックからは取り除かれる

スタック


1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
6
Pop
5
4
3
2
1
7
5
Pop
4
3
2
1
8
4
Pop
3
2
1
9
3
Pop
2
1
10
2
Pop
1

動かない理由

無限ループ

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

void draw(){
  background(255);
  fill(128);
  int x = 0;
  while(x >= 0){
    x = x + int(random(10));
    ellipse(x,height/2,10,10);
  }
}
```

デバッガを使うとデバッグ(debug)が楽になる

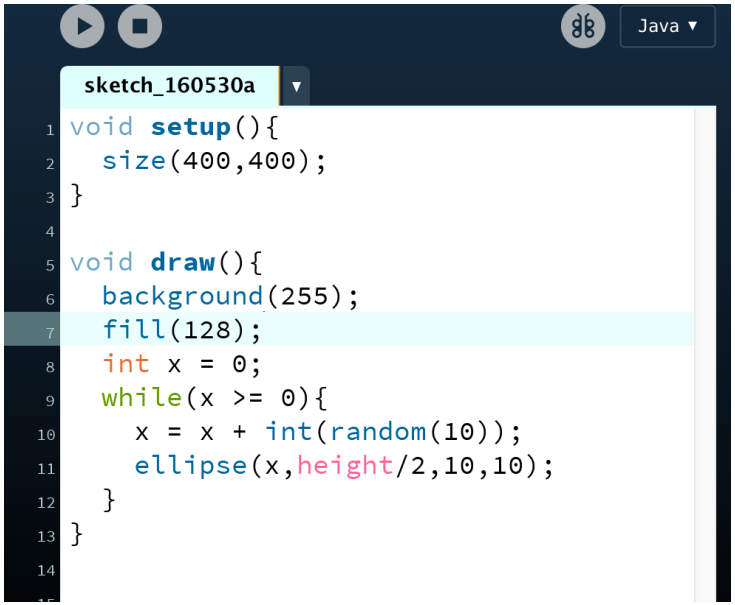
こんな時に便利

- 問題がありそうな処理のところで処理を止めて動きを確かめたい。
- 変数の値の変化を確認したい。
- 少しずつ動かしながら、動作を確認したい。

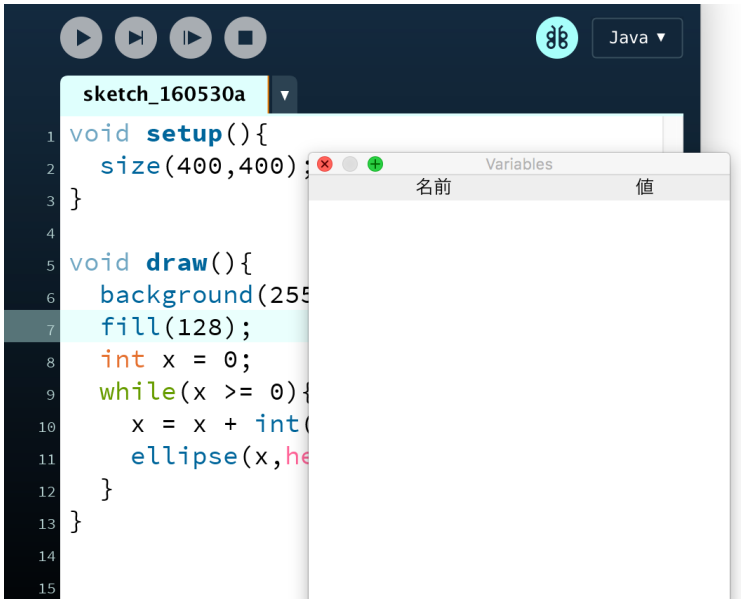
デバッグモードで出来ること

- 動きを確認したい処理のあたりにブレークポイントを設定し、実行。
  - ブレークポイント＝プログラムの動作を止めたい場所
- ステップ実行しながら、インスペクターで変数を確認しながら、期待の処理が行われている確認。

通常のウインドウ



デバッグ状態のウインドウ



具体的には

1. ステップ実行：プログラムを1ステップ毎に実行できる機能
   - stepボタンを押すと、1ステップ毎に実行。(左端に実行行にマークがつく)
2. ブレークポイント:実行を停止したい行を設定できる機能
   - ブレークポイントの設定：設定したい行にカーソルを合わせて、
   - Runボタンを押すと、ブレークポイントを設定した行でプログラムが一時停止する。
3. インスペクター：変数名、値を確認できる機能
   - 変数名と値を確認できる

変数の有効範囲

大域変数
（グローバル変数）

局所変数
（ローカル変数）

区別
どこで変数が
宣言されたか？

違い
どの範囲で使うことが
出来るのか？

大域変数（グローバル変数）

プログラムの先頭部分

変数宣言した後は、基本的にはプログラム中のどこでもOK

大域変数の使いすぎには注意

どこで変数が
宣言されたか？

どの範囲で使うことが
出来るのか？

局所変数（ローカル変数）

ブロックの中で変数宣言

宣言したブロック内で利用可能

```
for(int i=0;i < 10;i++){
  // 変数iはこの中で有効
}
```

```
void draw(){
  background(255);
  fill(128);
  int x = 0;
  while(x < width){
    x = x + int(random(10));
    ellipse(x,height/2,10,10);
  }
}
```

大域変数と同じ名前の局所変数を宣言するとどうなるか？

グローバル変数と局所変数は同時に存在をするが、宣言したブロック内部では、局所変数にしかアクセス出来ない。

関数

特定の処理を行う受付窓口

内部の細かい処理内容を隠す（隠蔽）

プログラム(パンの作成関数)

あんぱん

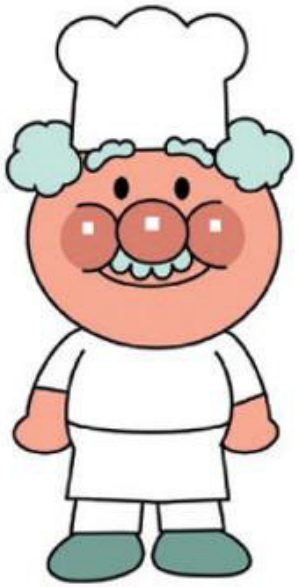
カレーパン

成果物(パン)

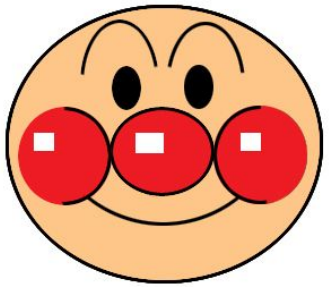
関数

特定の処理を行う受付窓口

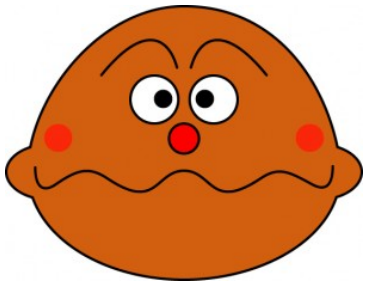
内部の細かい処理内容を隠す（隠蔽）



プログラム（パンの作成関数）

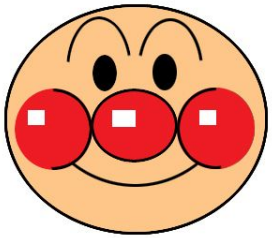


あんパン



カレーパン

あんパンの作り方



こねる

材料を用意
粉などをまぜる
こねる

発酵

生地を整形
放置

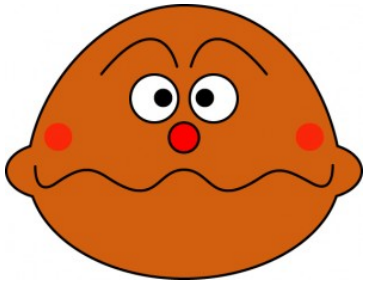
あんパンのたね作り

生地をとる
あんを詰める
丸める

焼く

オーブンを温める
たねを入れる
しばらく放置
取り出す

カレーパンの作り方



こねる

材料を用意
粉などをまぜる
こねる

発酵

生地を整形
放置

カレーパンのたね作り

生地をとる
カレーを詰める
丸める

焼く

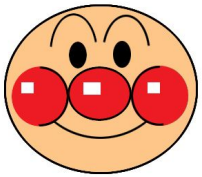
油を入れる
たねをいれれる
しばらく放置
取り出す

関数を使わないと

処理の区切りがわかりづらく、重複が多い

こねる
材料を用意
粉などをまぜる
こねる
発酵
生地を整形
放置
あんパンのたね作り
生地をとる
あんを詰める
丸める
焼く
オーブンを温める
たねを入れる
しばらく放置
取り出す

こねる
材料を用意
粉などをまぜる
こねる
発酵
生地を整形
放置
カレーパンのたね作り
生地をとる
カレーを詰める
丸める
焼く
オーブンを温める
たねを入れる
しばらく放置
取り出す



こねる

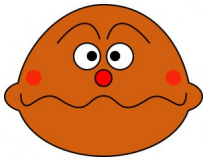
発酵

あんパンの
たね作り

焼く

カレーパンの
たね作り

揚げる



関数による
プログラムの整理

関数：makeあんパン
　こねる
　発酵
　あんパンのたね作り
　焼く

関数：makeカレーパン
　こねる
　発酵
　カレーパンのたね作り
　揚げる

関数：こねる
　材料を用意
　粉を混ぜる
　こねる
関数：発酵
　生地を整形
　放置
関数：あんパンのたね作り
　生地を適量とる
　あんを入れる
　整形
関数：焼く
　オーブンを温める
　たねを入れる
　一定時間待つ
　取り出す
関数：カレーパンのたね作り
　生地を適量とる
　カレーを入れる
　整形
　以下略

関数宣言

同じ処理はひとかたまりに–＞関数を利用

関数名で使えるもの：変数と同じ

```
void 関数名(){
        処理内容を書く
}
```

```
関数名(); // <-処理内容を実行できる
```

データを与えて処理を行う

```
void 関数名(データ型名  引数名,…){
    処理内容を書く
}

 関数名(引数に与えるデータ,…);  // <-処理内容を実行できる
```

授業時に配布した資料

http://www.sato-lab.jp/imfu/index.html

においてあります。