クリエイティブプログラミング

「情報II」第2章

Contents

クラス：　　番号：　　氏名：

図形の描画

すでに「情報I」で一度プログラミングについては一通りのことは学習していますが、ここでは､もう一度プログラミングの考え方を定着するように演習していきたいと思います。「情報I」とは言語は変えますが、言語が変わっても本質は変わらないことを体験しましょう。

（教科書II：p.116 – p.117 , 教科書I：p.134 – p.135 , p.160 – p.161）

■ 情報システムの開発とプログラミング

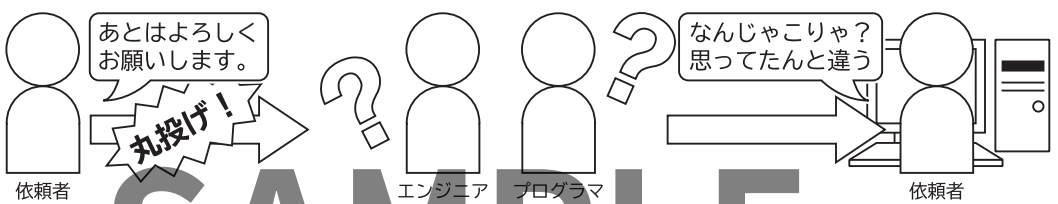
プログラミングを学ぶ意味

情報や情報技術を用いた問題解決は、社会のあらゆる分野、あらゆる場面で必須のもの

例）会社で提供するサービスをアプリで提供したい / 会社の業務を効率化させたい

→実際に開発するのは情報の専門家（エンジニア／プログラマ）

→専門家に内容や意図（要件定義）を正しく伝えなければ正しく開発されない



情報システムの開発

情報システムは一般に次のような工程で開発される

要件定義	どのような情報システムを作るかを決める
外部設計	画面や操作方法などを設計する
内部設計	システムの内部でのデータの構造や情報の扱い方を設計する
プログラミング	設計をもとに実際に具体的に開発をすすめる
テスト	設計どおりに動くかをテストする
運用	実際に情報システムを使う

プログラミングや設計を体験することで、情報システムの開発の全体像をつかもう

→情報や情報技術を用いた問題解決の見通しが見えるようになる

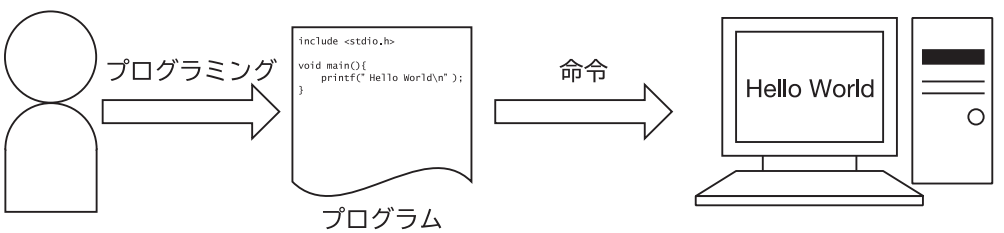
■ プログラミング言語

プログラミングとプログラミング言語

プログラミングとは

プログラム＝ コンピュータに対する命令（処理）を記述したもの

プログラミング＝ プログラムを記述すること

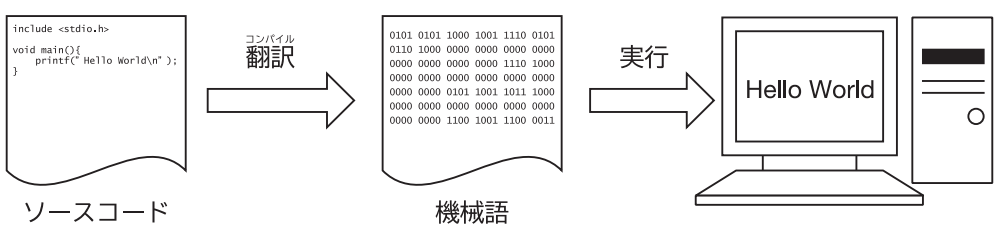


※すべてのコンピュータはプログラムにしたがって動作する

プログラミング言語

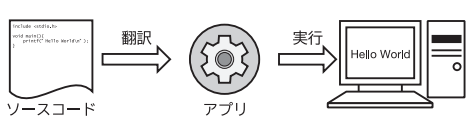
プログラミング言語＝ コンピュータプログラムを記述するための言語

コンピュータは機械語しか理解できない → プログラミング言語を機械語に変換して実行

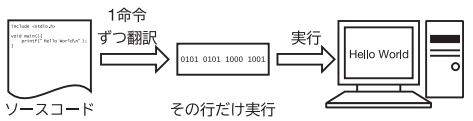


コンパイラ言語とインタプリタ言語

●コンパイラ言語



●インタプリタ言語



その都度翻訳の必要がないため、コンパイラ言語で作成されたアプリの方が高速に動作
インタプリタ言語は1行ずつ実行するため、エラーを発見しやすい

■ p5.jsの利用

p5.jsの起動

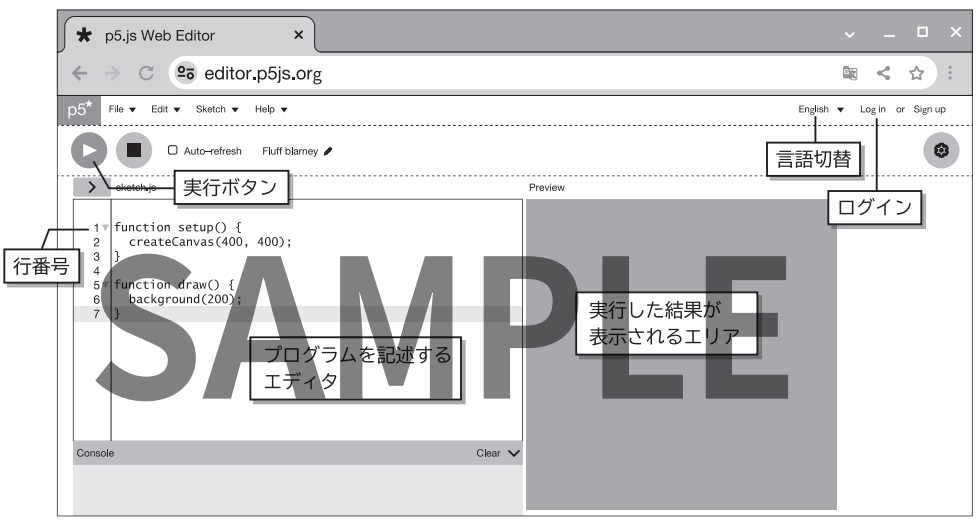
p5.js＝ 電子アートとビジュアルデザインを目的としたプログラミング環境

※JavaScriptと呼ばれる言語で記述する

p5.jsの起動

```
https://editor.p5js.org
```

※このURLで起動することができる



ステージの用意

```
1 function setup(){
2   createCanvas(400, 400);
3 }
4
5 function draw(){
6   background(220);
7 }
```

{ }中の命令が最初に一度だけ実行
キャンバスサイズを横400,縦400に

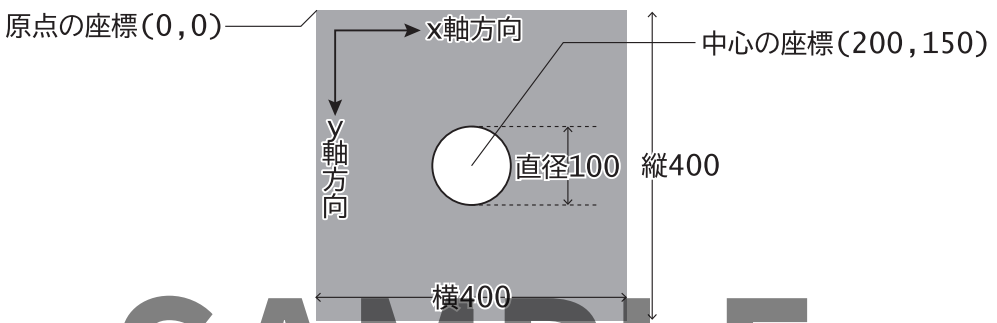
{ }中の命令がずっと繰り返し実行
背景色を220（グレー）に設定

これを実行すると、プレビュー画面に400×400のグレーのキャンバスが表示される

図形の描画

ステージの中央に円を描いてみよう

```
1 function setup(){
2   createCanvas(400, 400);
3 }
4
5 function draw(){
6   background(220);
7   circle(200, 200, 100);     (200,200)を中心に直径100の円
8 }
```



プログラム記述の際の注意点

♦1つの命令は必ず「;」で終わること

♦括弧を対応させること → (...)　{...} など（※複数行にまたがる場合がある）

※エラーになる場合のほとんどは、上記2つのどちらか → 特にこの2つに気をつけよう

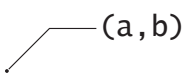
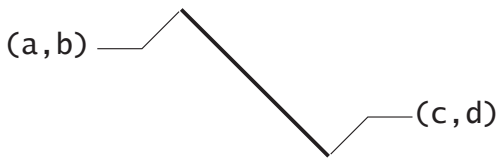
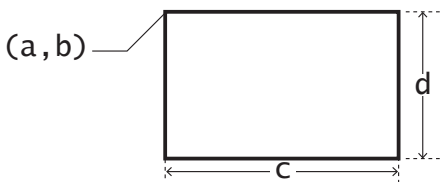
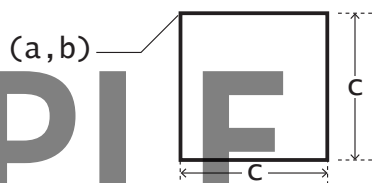
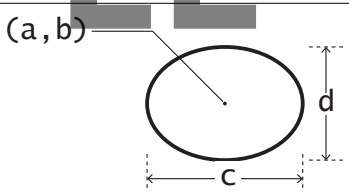
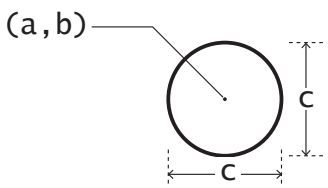
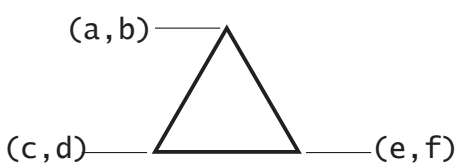
括弧を書くときは、先に括弧を閉じてから中を書くようにすると書き忘れなくて済む

括弧の対応

特に初{ }の対応が混乱を招く → 字下げを使ってまとまりを示すと便利

```
function xxxxxx(){
  while(...) {
    if(...) {
      ......
      if(...) {
        ......
      }
    }
  }
}
```

図形を描画する関数

図形	関数	
点	point(a,b);	
線	line(a,b,c,d);	
四角形	rect(a,b,c,d);	
正方形	square(a,b,c);	
楕円	ellipse(a,b,c,d);	
円	circle(a,b,c);	
三角形	triangle(a,b,c,d,e,f);	

図形の塗りつぶしと枠線

塗りつぶし

次のfill()関数またはnoFill()関数の後ろに書いた図形はここで指定した色になる

関数	説明
fill(R, G, B)	図形の塗り色をR,G,Bで設定（各0 ～ 255で設定）
noFill()	塗りつぶしをなしにすることで、枠線だけの図形を描ける

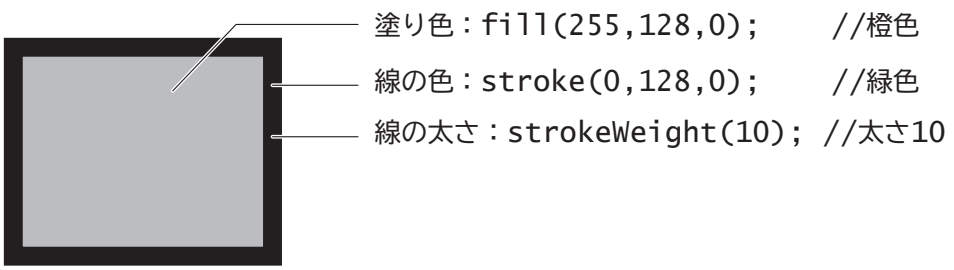
枠線

下のstroke()関数やstrokeweight()関数で図形の枠線の色や太さを指定できる

関数	説明
stroke(R, G, B)	図形の枠線の色をR,G,Bで設定（各0 ～ 255で設定）
strokeWeight(W)	図形の枠線の太さをWの太さにする（0も指定可能）
noStroke()	枠線をなしにすることができる

例

```
 1 function setup(){
 2   createCanvas(400, 400);
 3   noLoop();                    draw()関数をループさせない
 4 }
 5
 6 function draw(){
 7   background(220);
 8   fill(255, 128, 0);           塗り色を橙色に設定
 9   stroke(0, 128, 0);           枠線の色を緑色に設定
10   strokeWeight(10);            枠線の太さを10に設定
11   circle(200, 200, 100);
12 }
13
```

※今回は一度しか描画すればよいだけ → noLoop()関数でループしないようにした

テキストの描画

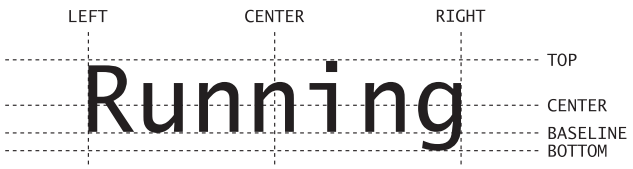
text()関数を使うと、キャンバス上に文字を表示させることができる

関数	説明
text('text',x,y)	座標(x,y)から「text」という文字を表示させられる

テキストの設定

textSize()関数、teztAlign()関数を使うと、文字サイズや位置の基準を設定

関数	説明
textSize(s)	sに指定した文字サイズになる
textAlign(h,v)	hに水平方向の配置（LEFT , CENTER , RIGHT） vに垂直方向の配置（TOP , BOTTOM , CENTER , BASELINE）




```
1 function setup(){
2   createCanvas(400,400);
3   noLoop();                    draw()関数をループさせない
4 }
5
6 function draw(){
7   background(220);
8   textSize(24);                文字サイズを24に設定
9   textAlign(CENTER,CENTER)     文字の基準を中央に設定
10  text('Information',200,200); (200,200)の位置に文字を表示
11 }
```


課題

今日学習した図形を組み合わせて、絵を描いてみよう。

必ずどこかに文字も表示させること。

提出方法

「ファイル→共有」から出てきた**フルスクリーン**のURLを提出

→URL部分をクリックすると「クリップボードへコピーしました！」となる→これでOK

→クリップボードにコピーされた状態で、提出フォームにURLペースト

プログラミングをする姿勢

失敗を恐れないこと

プログラミングは、間違えてもコンピュータは絶対に壊れない
→間違ったら何度でもやり直せばよいだけ

何度も失敗しながら、うまくいく方法を探っていこう！

少しずつすすめていくこと

プログラミングは一気にすべてのコードを書かず、少し書いては実行しよう
→一気に書いてしまうと、どこが間違っているのかが分かりづらい

少し書いては実行、少し書いては実行を繰り返しながら作っていこう！

振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。
□p5.jsで図形を描くことができるようになった
□p5.jsで文字を表示させることができるようになった
□p5.jsでの座標の向きと対応がわかるようになってきた

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

変数と代入文

プログラムの中で、数値や文字列を入れておく入れ物を変数といいます。変数を使うことで、同じ値を繰り返し使うことができます。今回は、変数を使って簡単なアニメーションをつくってみましょう。

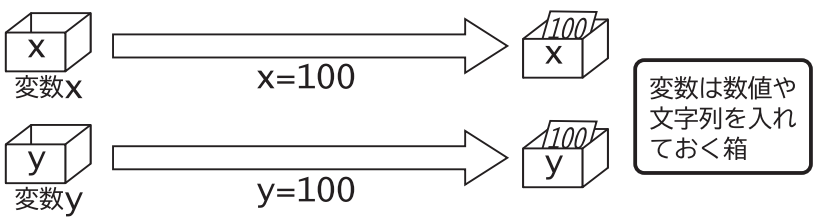
（教科書Ⅰ：p.134 – p.135 , p.160 – p.161）

■ 変数と代入式

変数とは

プログラムの中で、数値や文字列を入れておく入れ物を変数という

→変数を使うことで、同じ値を繰り返し使うことができる



変数の宣言

変数は、使う際にあらかじめ宣言をしておく必要がある

```
let 変数名 = 初期値;
```

変数を使ったプログラム

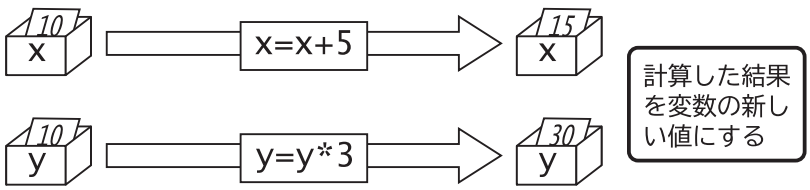
```
 1 let d = 100;                    円の直径を格納する変数
 2 let x = 200;                    x座標を格納する変数
 3 let y = 200;                    y座標を格納する変数
 4
 5 function setup(){
 6   createCanvas(400,400);
 7 }
 8
 9 function draw(){
10   background(220);
11   circle(x, y, d);              数値ではなく変数を使って座標を設定
12 }
```

例題1

変数の値をいろいろと変化させ、変数の意味を理解しよう

代入文

プログラミングにおける"="は、「右辺の計算結果を左辺の変数に代入する」という意味



代入文を使ったプログラム

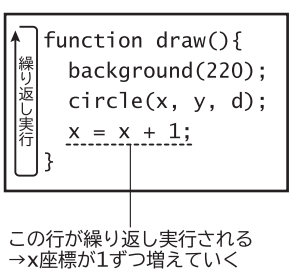
円を画面の左から右へ移動させるプログラム

```
let d = 10;
let x = 10;
let y = 200;

function setup(){
  createCanvas(400,400);
}

function draw(){
  background(220);
  circle(x, y, d);
  x = x + 1;
}
```

xに1を加えたものを新しいxとする

draw()関数内の{...}内のコードは繰り返し何度も実行される

→変数の値が変わるとアニメーションする



例題2

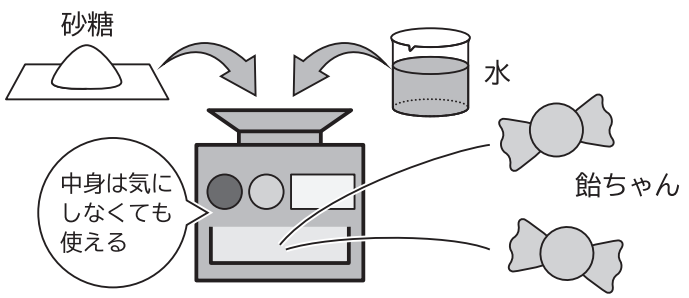
初期位置やxやyの変化量の値などをいろいろと変えて動きの違いを確かめてみよう

■ 関数

関数とは

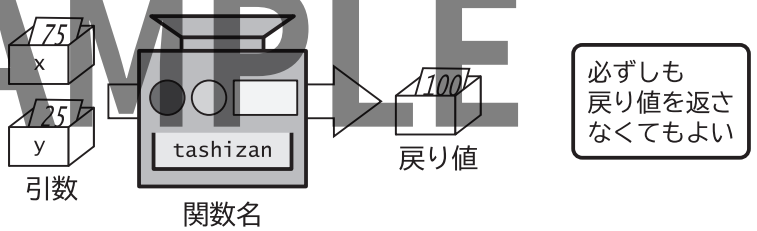
関数の概念

関数＝ **引数**を代入すると、定められた処理を実行し、結果を返す一連の命令群



材料を入れると、製品を返してくれる機械のようなものをイメージするとよい
→機械の中身がどうなっているかということは、利用者はあまり気にしなくてもよい

プログラミングにおける関数



関数名：機械の名前（好きな名前を付けてもよい）
引数（ひきすう）：機械に入れる材料にあたるもの（複数の引数を指定してもよい）
戻り値：機械によって作られた製品にあたるもの（必ずしも戻り値がある必要はない）

関数の利用

```
var p = 5;
var q = 7;

function tashizan(a, b){
  let x = a + b;
  return x;
}

r = tashizan(p,q);
```

5 7（関数定義の引数 a, b の上）
戻す 5
対応 7
関数定義
関数呼出

※このプログラムを実行した後、**r**には12が入っている

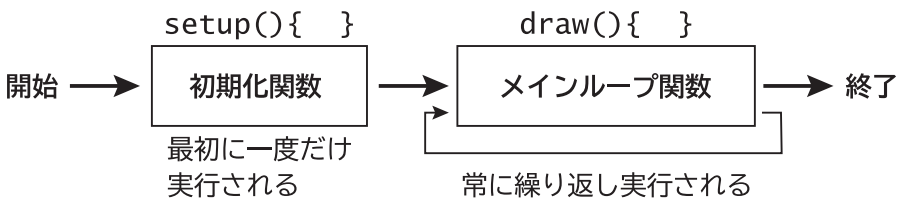
関数型プログラミング

組み込み関数

これまでに使用してきたcreateCanvas()もbackground()も関数の一つ
→プログラミング言語に事前に用意され、定義不要で使えるものを**組み込み関数**という

初期化関数、メインループ関数

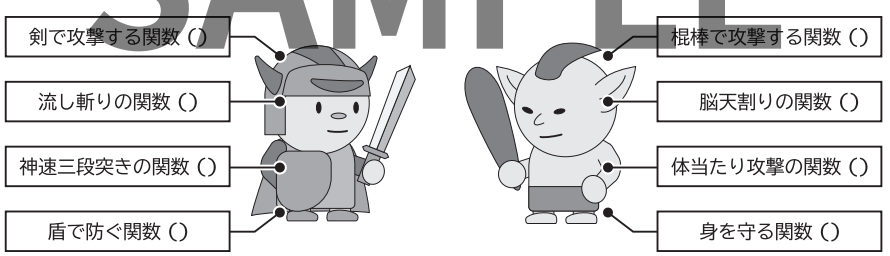
p5.jsで使われている、setup()、draw()も関数の一つ　→　特殊な関数
setup()関数は**初期化関数**と呼ばれ、開始して最初に一度だけ実行される
draw()関数は**メインループ関数**と呼ばれ、実行中常に繰り返される



※メインループ関数が無限ループする性質を利用してアニメーションを作成

関数型プログラミング

よく使う機能をまとめておくことで、同じプログラムを再利用できる
機能をまとめておくことで、プログラム本体もすっきりと見やすいものにできる



※いくつかの関数をまとめたライブラリもある　→　公開されているものもたくさんある
→ライブラリを自分のプログラムに読み込むことで効率的にプログラミングが可能

変数の通用範囲

letで定義された変数は、それが定義されたブロック（{...}）内でしか通用しない

```
let a,b,c;
function kansuuA(){
  let s,t,u;
  ......
}
function kansuuB(){
  let x,y,z;
  ......
}
```

変数a、b、cはkansuuA()内でもkansuuB内でも通用する
変数s、t、uはkansuuA()内でしか通用しない
変数x、y、zはkansuuB()内でしか通用しない

関数を使ったプログラム

指定した座標に、一定の大きさの円を描くプログラム

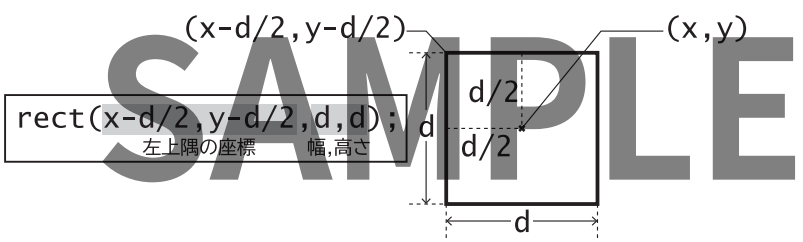
```
 1 let d = 100;
 2
 3 function setup(){
 4   createCanvas(400,400);
 5 }
 6
 7 function maru(x, y){
 8   circle(x, y, d);
 9 }
10
11 function draw(){
12   background(220);
13   maru(100, 100);
14 }
```

円の直径だけは先に宣言

ここから関数maru()を定義開始
座標(x,y)に直径dの円を描く

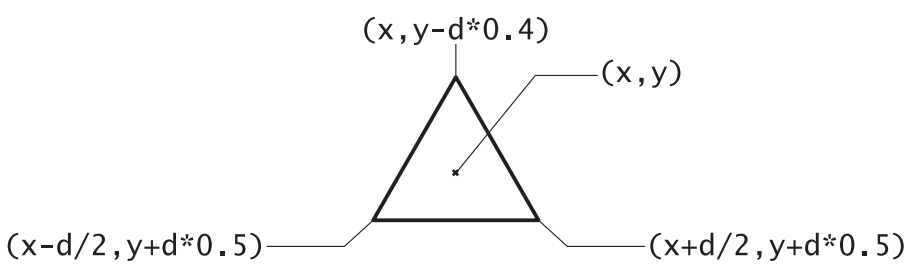
例題3

次の図を参考に、関数shikaku()を定義し、(200,200)を中心に正方形を描こう



例題4

次の図を参考に、関数sankaku()を定義し、(300,300)を中心に三角形を描こう



課題

maru()、shikaku()、sasnkaku()をいくつか配置し、模様を描こう

変数の命名規則

変数命名の必要性

変数を使うとき、変数には名前をつける必要がある
→何の値が入っているのかが分かりやすい名前を付ける必要がある
→aaaやdataなど、どのようなデータが入っているかわからないものはNG

命名規則4ケース

変数名の中に空白（スペース）は使えない
→`"user name"`や`"average of score"`
変数名の命名規則はおおよそ次の4つのいずれかになる

スネークケース	`user_name`	`average_of_score`
ローワーキャメルケース	`userName`	`averageOfScore`
アッパーキャメルケース	`UserName`	`AverageOfScore`
チェインケース	`user-name`	`average-of-score`

変数には分かりやすい名前を付けよう

振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。
□p5.jsで図形を描くことができるようになった
□p5.jsで文字を表示させることができるようになった
□p5.jsでの座標の向きと対応がわかるようになってきた

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

条件分岐

コンピュータで問題解決をする手順をアルゴリズムといいます。アルゴリズムは、逐次処理、条件分岐、繰り返しの3つの要素の組み合わせで実現されています。ここでは、条件によって処理が分かれる条件分岐について学びます。

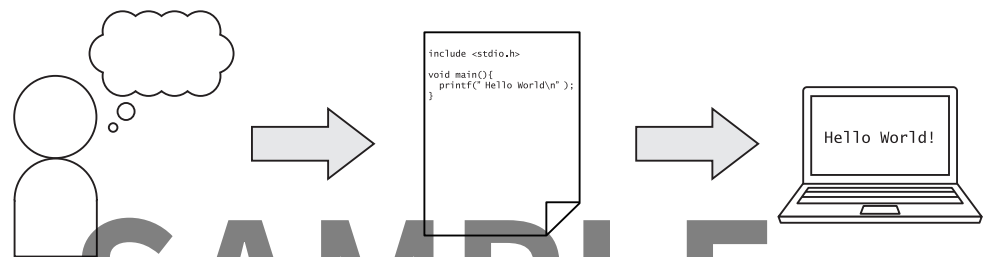
（教科書I：130 – p.132 , p.162）

■ アルゴリズムとは

アルゴリズムとは

アルゴリズム＝ 問題を解決するための考え方と手順

※プログラミングとは、アルゴリズムにしたがってプログラムを記述すること

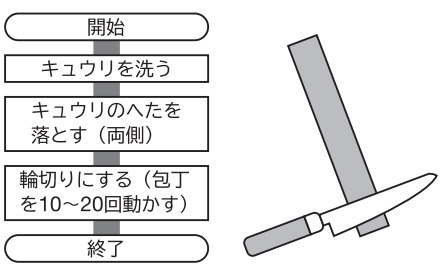


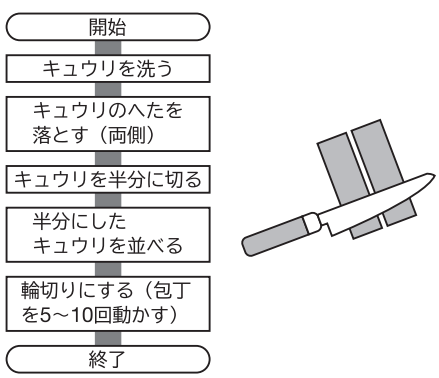
アルゴリズムを考える　　プログラムを記述する　　コンピュータで実行

よいアルゴリズム

同じ問題を解決するにしても、アルゴリズム次第で処理の効率は大きく変わる

●キュウリの輪切りアルゴリズム



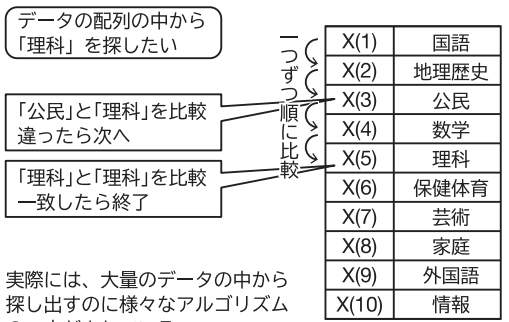


※一見、半分に切る方が処理が増えているように見えるが、包丁を動かす回数は減少

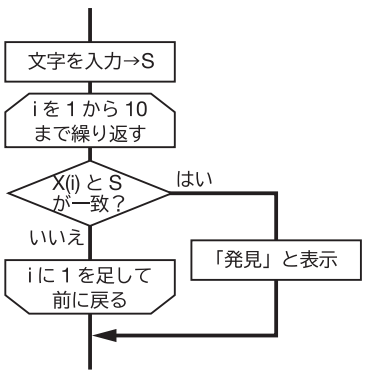
どうすればより効率的に処理ができるかを考えよう

コンピュータの得意なこと

- ♦デジタル情報を扱うため、情報を演算によって加工することが得意
- ♦決められた手順を、条件を判断しながら繰り返し処理することが得意

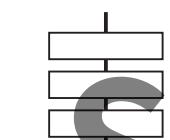
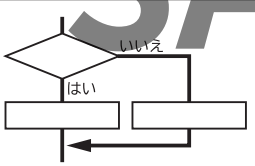
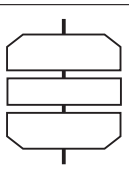


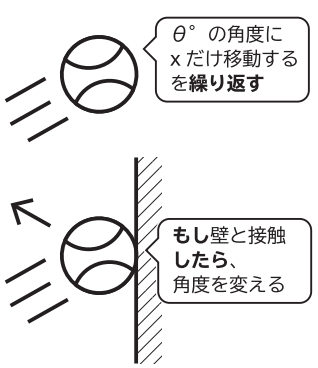
実際には、大量のデータの中から探し出すのに様々なアルゴリズムの工夫がされている。

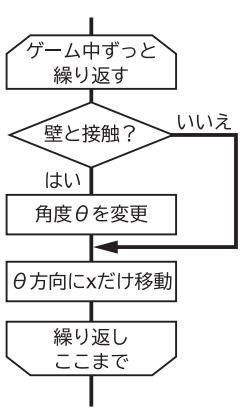


基本的なアルゴリズム

どんなプログラムも、次の3つの要素の組み合わせで実現される

逐次処理		処理が順番に行なわれる
条件分岐		条件により処理が分かれる
繰り返し		条件が成り立つ間、処理を繰り返す





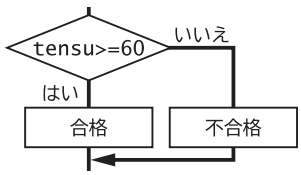
■ 条件分岐（if文）

if文

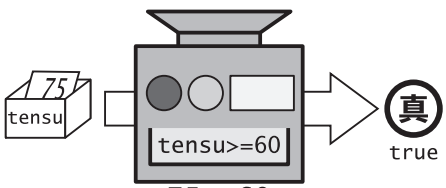
条件によって処理を分けたい場合、if文を使う

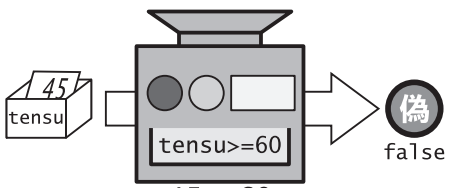
```
if ( [条件式] ) {
  [条件式が真の場合の処理];
} else {
  [条件式が偽の場合の処理];
}
```

```
if(tensu>=60){
  kekka="合格";
}else{
  kekka="不合格";
}
```



条件式の考え方





※比較演算子には「==（等しい）」「!=（等しくない）」「>」「<」「>=」「<=」が使える

より複雑な条件式

演算	書式	説明	使用例
否定	!条件式	「条件式」**ではない**	if(!(a==b))
論理積	条件式A && 条件式B	「条件式A」**かつ**「条件式B」	if(x>=a && x<=b)
論理和	条件式A || 条件式B	「条件式A」**または**「条件式B」	if(x<=a || x>=b)

準備

Classroomからリンクを開き、左上の</>から編集モードにしよう

```
let x = 10;
let y = 150;
let vx = 1;

function setup(){
  createCanvas(400,400);
}

function draw(){
  reset();          ステージを真っ白にする独自関数
  x = x + vx;
  ball(x,y);        (x,y)を中心に直径10の円を描く独自関数
}
```

※この状態から、プログラムを改良していくこととする

例題1

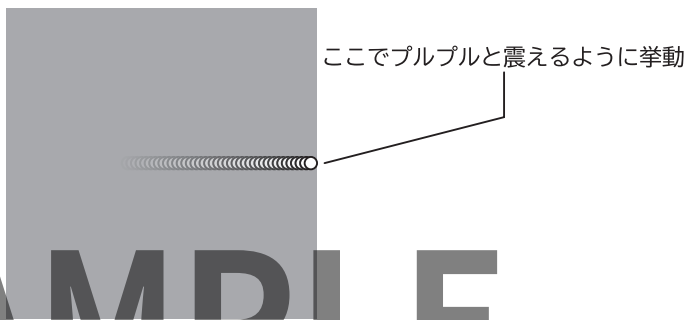
右端まで行ったらはね返るようにプログラムを作りたい

→下のように書き換えてみよう

```
 9 function draw(){
10   reset();
11   if (x < 400) {
12     x = x + vx;
13   } else {
14     x = x - vx;
15   }
16   ball(x, y);
17 }
```

もし**x**が400より小さいなら{ }内を実行
　xに**vx**を足したものを新しい**x**とする
もし**x**が400より小さくなければ{ }内を実行
　xに**vx**を引いたものを新しい**x**とする
13行目の{に対応した括弧を閉じるための}

これを実行すると、x= 400のところでプルプルと震えていることがわかる



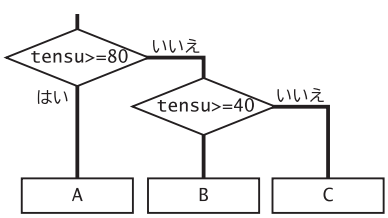
実際にコードを入力して、失敗することを確かめよう

→なぜ静止してしまうかを考えてみよう

if文で更に分岐を増やしたい場合

if文で分岐した先で更に分岐させたい場合、else if文を使う

```
if ( [条件式1] ) {
  [条件式1が真の場合の処理];
} else if ( [条件式2] ) {
  [条件式2が真の場合の処理];
} else {
  [条件式2が偽の場合の処理];
}
```



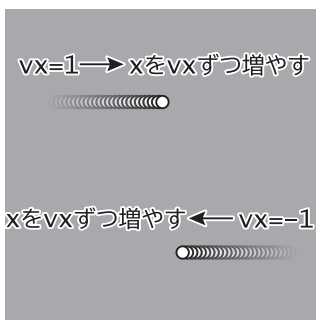
※else if文はさらにいくつでも追加していくことができる

例題2

きちんとはね返るようにするには、次のようにすればよい

```
 9 function draw(){
10   reset();
11   if (x > 400) {
12     vx = -vx;
13   }
14   x = x + vx;
15   ball(x, y);
16 }
```

もしxが400より大きいなら{ }内を実行
　vxをマイナスにする
11行目の{に対応した括弧を閉じるための}
xにvxだけ足したものを新しいxとする



例題3

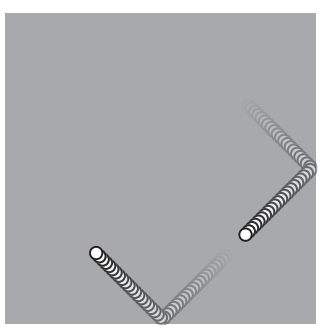
画面左端でも同様にはね返るようにしてみよう

※ヒント：「xが0より小さくなったら」というif文を追加する

課題

縦方向にも画面上をはね返るようにしてみよう。

※ヒント：変数vy（y方向の速度）を追加し、y = y + vyをプログラムに追加する



※できあがったら、条件式を工夫することでコードの行数を減らせないか考えてみよう

データの入れ換え

変数aに入っている値と変数bに入っている値を入れ換える場合

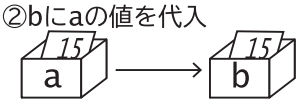
失敗例

```
a=10;
b=15;
a=b;  ①
b=a;  ②
```

①aにbの値を代入

この時点でaの値が15に

②bにaの値を代入

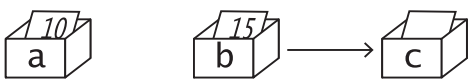


aが15なので、bには15が入る

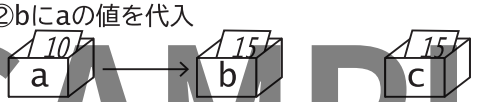
正解

```
a=10;
b=15;
c=b;  ①
b=a;  ②
a=c;  ③
```

①cにbの値を代入



②bにaの値を代入



③aにcの値を代入

別の変数に退避しておくことで変数の交換ができる

振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。

□条件分岐の考え方を理解した

□if文を使うことができるようになった

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

順次繰り返し

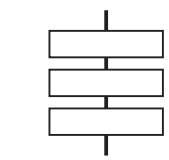
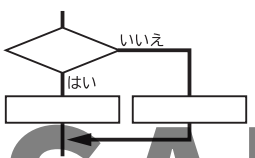
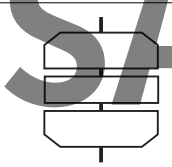
すべてのプログラムは、逐次処理、条件分岐、繰り返しの3つの組み合わせでつくられています。ここでは、繰り返しを学び、繰り返しと条件分岐の組み合わせによってコンピュータが動いていることを実際に確かめます。

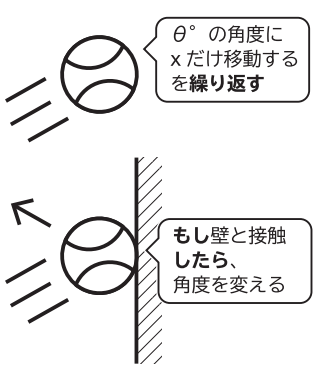
（教科書Ⅰ：p.130 – p.132 , p.162）

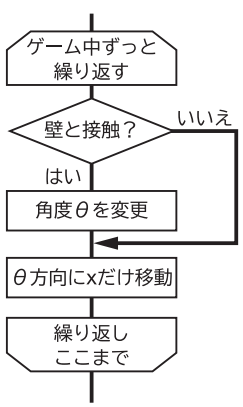
■ 基本的なアルゴリズム

基本的なアルゴリズム

どんなプログラムも、次の3つの要素の組み合わせで実現される

逐次処理		処理が順番に行なわれる
条件分岐		条件により処理が分かれる
繰り返し		条件が成り立つ間、処理を繰り返す





■ 順次繰り返し文（for文）

for文

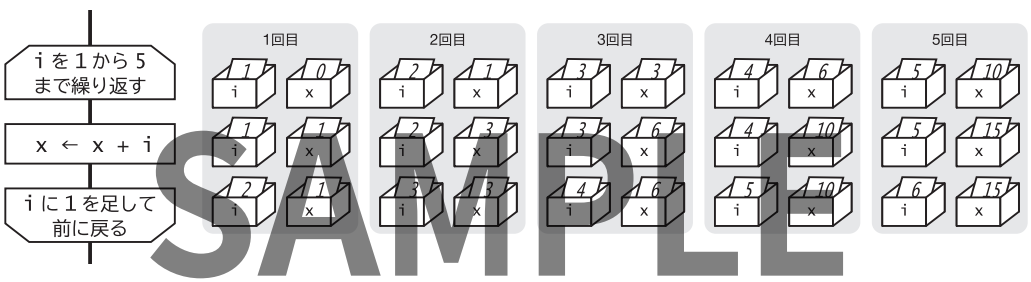
一定の回数、同じ処理を繰り返したい場合、for文を使う

```
for( [カウンタ初期値] ; [条件式] ; [カウンタ更新式] ){
    [繰り返す処理] ;
}
```

[条件式]を満たしている間、{ }内の処理を実行し、満たさなくなれば終了する
処理が終了したところで[カウンタ更新式]を実行し、再び[条件式]をチェックする
[条件式]を満たさなくなった（**偽**になった）ところで処理を終了する

```
let x = 0;
for(let i = 1; i <= 5; i++){
  x += i;
}
```

※i++ は i = i + 1 と同じ意味
※x += i は x = x + i と同じ意味



6回目の条件式が「6<=5」となり、**偽**となる　→　繰り返しを抜ける

準備

Classroomからリンクを開き、左上の</>から編集モードにしよう

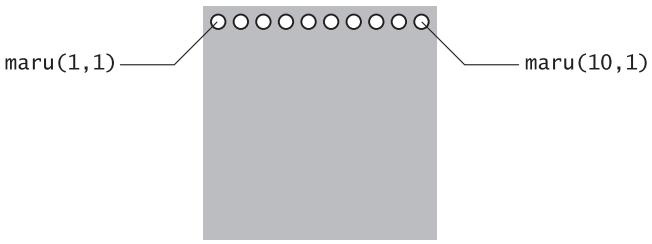
```
1 function setup() {
2   createCanvas(200, 200);
3   background(220);
4   noLoop();
5 }
6
7 function draw() {
8   maru(1,1);
9 }
```

（1列目,1行目）に直径10の円を描く関数

※この状態からプログラムを改良していくこととする

例題1

次の図のように、丸を10個等間隔に並べたい



よくない例

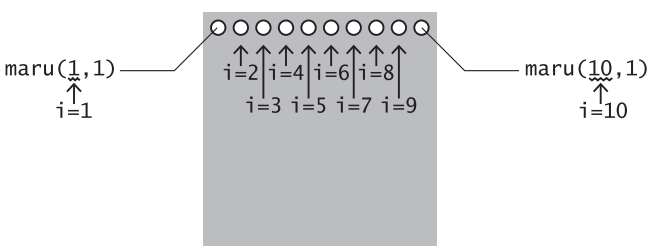
次のようにしても同じ結果を得られるが、たいへん非効率　→　タイプミスの可能性も

```
 7 function draw() {
 8   maru(1,1);
 9   maru(2,1);
...   ......
17   maru(10,1);
18 }
```


正解

for文を使い、カウンタ変数**i**が1から10になるまで繰り返す

```
 7 function draw() {
 8   for(let i=1; i<=10; i++){
 9     maru(i,1);
10   }
11 }
```



例題2

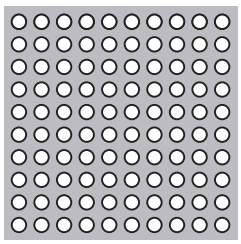
maru(1,**i**)とすることで縦一列に並ぶことを確認してみよう

例題3

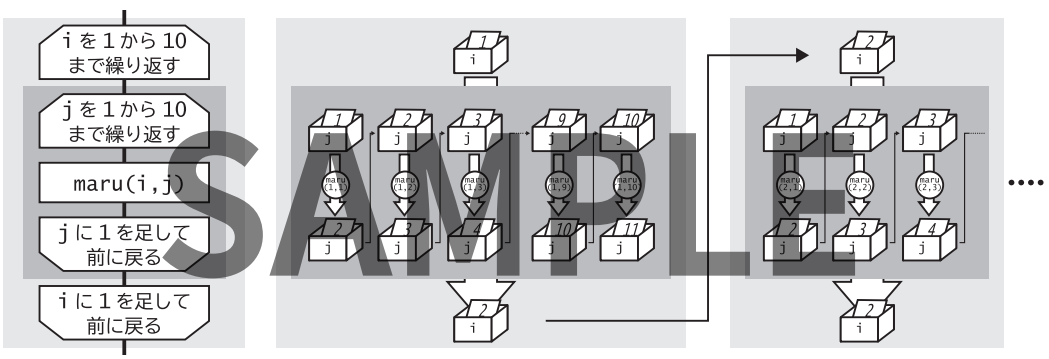
maru(**i**,**i**)とすることで斜めに並ぶことを確認してみよう

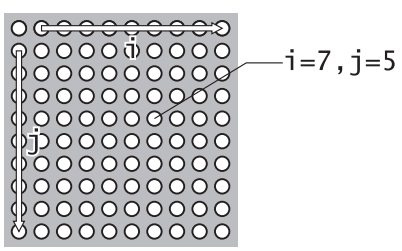
例題4

次の図のように、丸を100個等間隔に並べたい



```
 7 function draw() {
 8   for(let i=1; i<=10; i++){
 9     for(let j=1; j<=10; j++){
10       maru(i,j);
11     }
12   }
13 }
```





例題5

上のプログラムの9行目を次のように書き換えるとどうなるか、実際にやって確かめよう

```
 9     for(let j=1; j<=i; j++){
```

例題6

［例題5］の状態で、更に8行目を次のように書き換えるとどうなるか、確かめてみよう

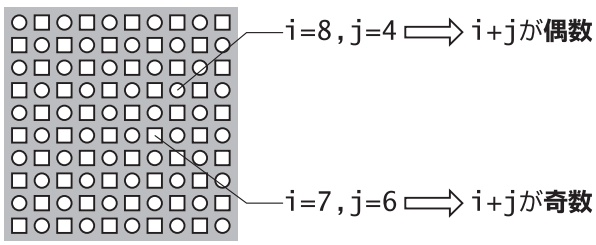
```
 8     maru(10-i,j)
```

※他にも8行目の( )内を書き換えていろんなパターンを試してみよう

■ 順次繰り返しと条件分岐の組み合わせ

例題7

下のような千鳥模様を作りたい



考え方

i+jが**偶数**なら○（maru(**i**,**j**)）、奇数なら□（shikaku(**i**,**j**)）を表示させる

→2で割った余り（(**i**+**j**)%2）が0なら**偶数**、1なら**奇数**（**a**%**b**は**a**を**b**で割った余り）

```
 7 function draw() {
 8   for(let i=1; i<=10; i++){
 9     for(let j=1; j<=10; j++){
10       if( (i+j)%2 == 0 ){
11         maru(i,j);
12       } else {
13         shikaku(i,j);
14       }
15     }
16   }
17 }
```


課題

i+jを3で割った余りが0なら○、1なら△、2なら□を表示させてみよう

```
10       if (                    ) {
11         maru(i, j);
12       } else if(                    ) {
13         sankaku(i, j);
14       } else {
15         shikaku(i,j);
16       }
```

※空欄になっているところがどうなるかを考えてみよう

代入演算子とインクリメント演算子

`i = i + 1;`のように変数に決まった値を足す代入式がよく出てくる
→代入演算子を使うことでより簡単に書くことができる

代入演算子

たとえば、`i = i + 1;`は次のように表すことができる

```
i += 1;
```

"+="は右辺の値を左辺の変数に加算するという意味
同様に、"−="、"*="、"/="のように減算、乗算、除算でも使える

インクリメント演算子

`i = i + 1;`は、定型的に次のように書くことがある

```
i++;
```

for文などで、入力する文字数が減って効率的

振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。
□繰り返しの考え方を理解した
□for文を使うことができるようになった
□配列の考え方を理解した

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

配列

同じような変数をいくつも作りたい場合、一つ一つ変数を定義していたのではプログラムが煩雑になってしまいます。そこで、配列を使うと変数をまとめて扱えるのでとても便利です。ここでは、配列の基本的な考え方について学びます。

（教科書Ⅰ：p.138 – p.139）

■ 配列

配列とは

配列＝ 同じ型のデータを一列に並べたもので、添字（そえじ）を使って取り出すことができる

a[]という配列に上記のように値が入っていたとする → aの4番目の値が27という意味

※配列の添字は、0番目から始まることに注意

定義の方法

上記の例だと、次のように一括で設定することができる（変数名に[]を付ける）

```
let A = [32,73,79,85,27,30,55,91,75,67]
```

配列には、文字列も設定可能

```
let A = ["太郎","花子","健太","弘子","二郎"]
```

空の配列をつくることもできる

```
let A = new Array(10);
```

配列の使い方

```
1 let A = [32,73,79,85,27,30,55,91,75,67]
2 console.log(A[4])                          配列の4番目の値
```

※console.log()は、コンソールに結果を表示する関数（正確にはメソッド）

このプログラムで表示される値は何だろう？

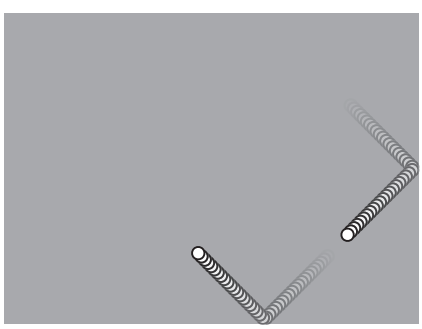
配列の添字は0番目から始まることに注意しよう

準備

Classroomからリンクを開き、左上の</>から編集モードにしよう

```
let x = 10;
let y = 10;
let vx = 1;
let vy = 1;

function setup() {
  createCanvas(400, 300);
}

function draw() {
  reset();
  for(let i=0; i<1; i++){
    x = x + vx;
    y = y + vy;
    ball(x,y);
    if(x>=400 || x<=0){
      vx = -vx;
    }
    if(y>=300 || y<=0){
      vy = -vy;
    }
  }
}
```


※この状態から、プログラムを改良していくこととする



※このプログラムを実行すると、ボール1つが画面上をはね返りながら飛び回る

目標

たくさんのボールがランダムに飛び回るプログラムを作りたい

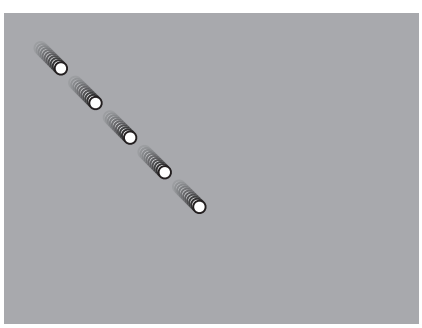
例題1

プログラムを次のように修正して、5個のボールが連なって動くプログラムを作る

```
let x = [10,20,30,40,50];
let y = [10,20,30,40,50];
let vx = [1,1,1,1,1];
let vy = [1,1,1,1,1];

function setup() {
  createCanvas(400, 300);
}

function draw() {
  reset();
  for(let i=0; i<5; i++){
    x[i] = x[i] + vx[i];
    y[i] = y[i] + vy[i];
    ball(x[i],y[i]);
    if(x[i]>=400 || x[i]<=0){
      vx[i] = -vx[i];
    }
    if(y[i]>=300 || y[i]<=0){
      vy[i] = -vy[i];
    }
  }
}
```

変数**x**を配列にして初期値を設定
変数**y**を配列にして初期値を設定
変数**vx**を配列にして初期値を設定
変数**vy**を配列にして初期値を設定

配列の長さ分だけ繰り返し
変数をすべて配列のi番目とする



※5つのボールが連なって動くようになる

例題2

それぞれのボールの初期値をいろいろな値に変更してみよう

■ 乱数

乱数

乱数＝ サイコロを投げるように、次に何が出るかわからない数字のこと

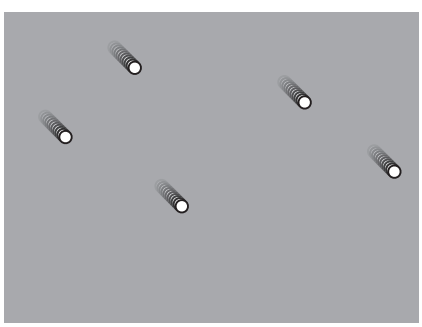
乱数の生成

random(10);	実数の0.000 ～ 9.999の乱数を生成する
random(-5,5);	実数の-5.000 ～ 4.999の乱数を生成する
int(random(10));	整数の0 ～ 9までの乱数を生成する

例題3

x[]の初期値を乱数で設定したい　→　次のように書き換えてみよう

```
 1 let x = new Array(5);                  xの箱だけを用意（配列の長さは5）
...
 6 function setup(){
 7   createCanvas(400,300);
 8   for(let i=0; i<5; i++){              iを使って5回繰り返し
 9     x[i] = random(400);                  xのi番目に0 ～ 399の乱数を代入
10   }
11 }
...
```

※5つのボールの**x**の初期値が乱数になる

例題4

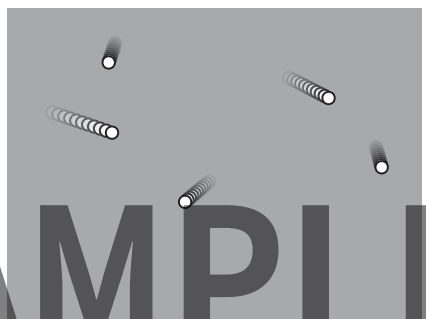
yの初期値も0 ～ 300の乱数に設定してみよう

※ヒント：2行目を1行目同様に変更、11行目からのfor文の{　}内に1行追加

例題5

横方向の速度の初期値を−2 ～ 2の乱数で設定してみよう

```
...
 3 let vx = new Array(5);          vxの箱だけを用意（配列の長さは5）
...
 6 function setup(){
 7   createCanvas(400,300);
 8   for(let i=0; i<5; i++){
 9     x[i] = random(400);
10     y[i] = random(300);
11     vx[i] = random(-2,2);       −2 ～ 2の乱数を生成、vxのi番目に
12   }
13 }
...
```



※横方向の速さが乱数になる

例題6

y方向の速さも−2 ～ 2までの乱数で設定してみよう

1か-1を生成したい

乱数を使って1または-1のどちらかを生成する場合、以下のようにする

```
int(random(2))*2-1;
```

random(2)で0~1.999が生成される→int()で整数部分を取り出す
→int(random(2))で0または1が生成される
2倍すると、0または2が生成→そこから1を引くと-1または1が生成

右または左を設定したいなどの際に便利

課題

ボール100個がランダムに飛び回るアニメーションを作ってみよう

```
let x = [redacted];
let y = [redacted];
let vx = [redacted];
let vy = [redacted];

function setup() {
  createCanvas(400, 300);
  for(let i=0; i<[redacted]; i++){
    x[i] = [redacted];
    y[i] = [redacted];
    vx[i] = [redacted];
    vy[i] = [redacted];
  }
}

function draw() {
  reset();
  for(let i=0; i<[redacted]; i++){
    ...
  }
}
```


振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。

□配列の考え方を理解した

□配列を使ったプログラムを組むことができるようになった

□順次繰り返し文（for文）を使うことで、配列を効率よく利用できることを理解した

□乱数の考え方を理解した

□プログラムの中で乱数を使うことができた

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

アーティスティックプログラミング

ここまで学んだ知識を応用すると、p5.jsを使ってアートな作品を制作することができます。今日は、いくつかのアート作品を制作し、プログラミングでなければ制作できないアート作品とはどのようなものかを体験してもらいたいと思います。

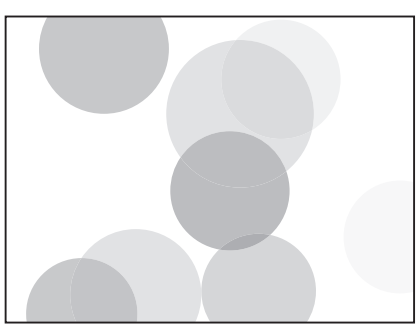
（教科書I：132 – p.139 , p.160 – p.163）

■ アーティスティックプログラミング

作例1

色とりどりの図形が次々と現れては消えていくプログラムを作ってみよう

```
 1 function setup() {
 2   createCanvas(400, 300);
 3   frameRate(5);
 4 }
 5
 6 function draw() {
 7   screenClear();
 8   let r = random(255);          赤の強さを乱数で0~255に
 9   let g = random(255);          緑の強さを乱数で0~255に
10   let b = random(255);          青の強さを乱数で0~255に
11   let a = random(128);          図形の不透明度を乱数で0~128に
12   fill(r,g,b,a);                aは不透明度
13   let x = random(400);          円のx座標を乱数で0~400に
14   let y = random(300);          円のy座標を乱数で0~300に
15   let d = random(20,150);       円の直径を乱数で20~150に
16   circle(x,y,d);                座標(x,y)に直径dの円を描く
17 }
18
19 function screenClear(){        画面を少しずつ消していく関数
20   noStroke();                   枠せんなし
21   fill(255,255,255,32);         塗り色を白色で不透明度を32に
22   rect(0,0,400,300);            画面全体に不透明な四角を描く
23 }
```

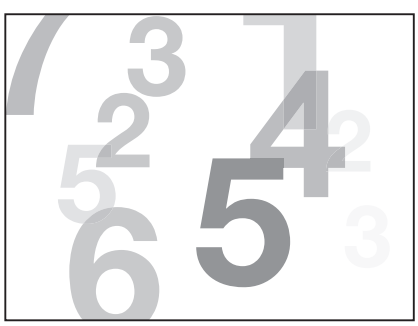
作例2

［作例1］を改良し、0~9の数字がランダムに表示されるようにする

```
 1 function setup() {
 2   createCanvas(400, 300);
 3   frameRate(5);
 4 }
 5
 6 function draw() {
 7   screenClear();
 8   let r = random(255);
 9   let g = random(255);
10   let b = random(255);
11   let a = random(128);
12   fill(r,g,b,a);
13   let s = random(64,128);
14   let n = int(random(10));
15   let x = random(400);
16   let y = random(300);
17   textSize(s);
18   textAlign(CENTER,CENTER);
19   text(n,x,y);
20 }
21
22 function screenClear(){
23   noStroke();
24   fill(255,255,255,32);
25   rect(0,0,400,300);
26 }
```

文字のサイズを乱数で64~128に
表示させる数字0~9を乱数で生成

文字サイズを設定
文字を表示する基準を中央に
表示させる数字**n**を座標(**x**,**y**)に



作例3

［作例1］［作例2］の図形や数字の出現位置をマウスの位置にする

```
13   let x = mouseX;
14   let y = mouseY;
```

x座標を現在のマウスのX座標にする
y座標を現在のマウスのY座標にする

作例4

［作例1］［作例2］を改良し、マウスクリックした位置に円や数字を描く

```
 1 function setup() {
 2   createCanvas(400, 300);
 3   frameRate(5);
 4 }
 5
 6 function draw() {
 7   screenClear();
 8 }
 9
10 function mouseClicked(){
11   let r = random(255);
12   let g = random(255);
13   let b = random(255);
14   let a = random(128);
15   fill(r,g,b,a);
16   let x = mouseX;
17   let y = mouseY;
18   let d = random(20,150);
19   circle(x,y,d);
20 }
21
22 function screenClear(){
23   noStroke();
24   fill(255,255,255,32);
25   rect(0,0,400,300);
26 }
```

screenClear()はクリックと無関係に実行したい

mouseClicked()関数がマウスクリックしたときに実行される特殊関数

x座標をマウスのX座標に

y座標をマウスのX座標に

マウスイベント駆動

下表はマウスのボタンをクリックした瞬間や、押した瞬間、離した瞬間に動作する関数

function mouseClicked(){ }	マウスボタンをクリックしたときに動作する
function mousePressed(){ }	マウスボタンが押されたときに動作する
function mouseReleased(){ }	マウスボタンが離されたときに実行する
function mouseMoved(){ }	マウスが動いたときに実行する
function mouseDragged(){ }	マウスがドラッグされたときに実行する

課題

［作例4］を改良し、マウスをドラッグして円の大きさを決める

```
let x = 0;
.....                                  y,r,g,b,a すべて初期値を0に

function setup() {
  .....                                ［作例1］そのまま
}

function draw() {
  screenClear();
}

function mousePressed(){
  x = [redacted];                      xをマウスのX座標に
  y = [redacted];                      yをマウスのY座標に
  r = [redacted];                      r,g,b,aを乱数で設定
  .....                                （［作例1］を参考に）
}

function mouseReleased(){
  fill(r,g,b,a);
  let d = abs(x - mouseX);             マウスを押したxからドラッグした
  circle(x,y,d);
}

function screenClear(){
  .....                                ［作例1］そのまま
}
```


振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。

□プログラミングを使ってアート作品を作ることの楽しさを実感することができた

□これまで学んできたプログラミングの知識をより定着させることができた

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

インタラクティブプログラミング

前項では、アーティスティックな作品を制作しましたが、ここからはプログラミングを用いて「情報デザイン」を意識したコンテンツの制作につなげたいと思います。今回はマウスをクリックしたりドラッグしたりすることで反応するコンテンツをつくっていきます。

（教科書II：p.28 – p.29 , p.44 – p.47 , 教科書I：p.132 – p.139 , p.160 – p.163）

■ インタラクティブプログラミング

画像の読み込み

画像のアップロード

プログラム中で画像を扱いたいため、まず画像をアップロードしたい

プログラムエディタ左上の「>」を開くと、スケッチファイルの一覧が表示される

→一覧の上にある［+］から「ファイルアップロード」を選びファイルをアップロード

例題1

Classroomからダウンロードした画像をスケッチファイルにアップロードしよう

画像の読み込み

画像を読み込むには時間がかかるため、プログラム実行時に最初に実行する必要がある

→特殊関数preload()関数内で画像の読み込み処理をする必要がある

→読み込んだ画像を格納しておくための変数を用意しよう

例題2

画像を読み込み表示させるプログラムを作ってみよう

```
let smile1;                                  画像を格納する変数を宣言

function preload(){
  smile1 = loadImage('smile1.png');          画像の読み込み
}

function setup() {
  createCanvas(200, 200);
}

function draw() {
  image(smile1,50,50);                       座標(50,50)に画像表示
}
```

※setup()関数はpreload()関数が終了してから実行される

※image()関数で実際に画像を表示させることができる

マウスの状態に応じて画像を切り替え

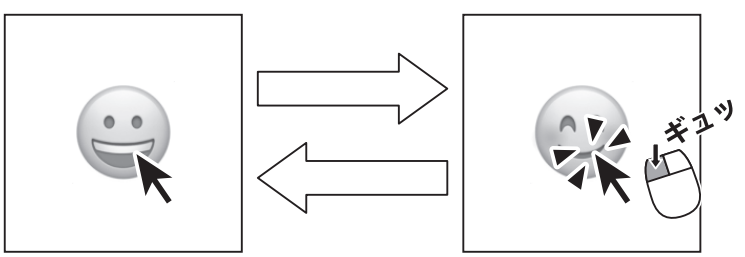
マウスのボタンを押している間画像を切り替え

マウスのボタンを押している状態は**mouseIsPressed**という変数に格納される
マウスのボタンが押されていると**mouseIsPressed**の値が「true」となる
マウスのボタンが押されていないと**mouseIsPressed**の値は「false」となる
→if文の条件式としてそのまま使うことができる

例題3

マウスを押している間、画像が切り替わるようにプログラムしよう

```
let smile1,smile2;                          画像を格納する変数を追加

function preload(){
  smile1 = loadImage('smile1.png');
  smile2 = loadImage('smile2.png');         'smile2.png'の読込
}

function setup() {
  createCanvas(200, 200);
}

function draw() {
  if(mouseIsPressed){                       マウスが押されていれば真
    image(smile2,50,50);                    smile2を表示
  } else {
    image(smile1,50,50);                    smile1を表示
  }
}
```

ブール値

mouseIsPressed変数はtrueかfalseの値しか持たない（**ブール変数**）
ブール変数が真の場合の条件式の「==true」を省略できる
偽の場合の条件式は「!**mouseIsPressed**」のように前に!を付ける

マウスをクリックするたびに画像を切り替え

マウスをクリックするたびに実行するには`mouseClicked()`関数を定義すればよい
状態を管理する変数（**flag**）を用意し、**flag**の値に応じて`if`文で処理を切り替え

例題4

マウスをクリックするたびに画像が切り替わるようにプログラムしよう
flagの状態が1なら**smile1**を、**flag**の状態が2なら**smile2**を表示する

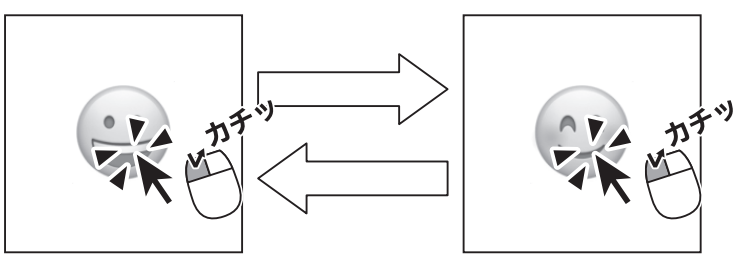
```
 1 let smile1,smile2;
 2 let flag = 1;
 3
 4 function preload(){
 5   smile1 = loadImage('smile1.png');
 6   smile2 = loadImage('smile2.png');
 7 }
 8
 9 function setup() {
10   createCanvas(200, 200);
11   image(smile1,50,50);
12 }
13
14 function draw() {
15
16 }
17
18 function mouseClicked(){
19   if(flag == 1){
20     image(smile1,50,50);
21     flag = 2;
22   }else{
23     image(smile2,50,50);
24     flag = 1;
25   }
26 }
```

flagの初期値を1にする

最初に**smile1**を表示

何もいらない

flagが1なら{}内を実行
smile1を表示
flagを2にする
flagが2なら{}内を実行
smile2を表示
flagを1にする



※一般に、このような切り替え機構を**トグル**という

キー操作で画像を動かす

キーボードのキーを押した際に実行するにはkeyPressed()関数を定義すればよい
押されたキーは**key**または**keyCode**変数に記録されているため、if文で場合分け

画像を配列に格納

方向別の画像を配列に格納したい　→　画像のファイル名に添字を付けるとよい
例えば、「turtle_1.png」のように画像ファイルのファイル名に添字の数字を入れる
→"turtle"+i+".png"とすることで、間に数字を入れて画像を指定することが可能

例題5

画像を配列に格納してみよう
変数**dir**は、方向を示す（↑：0　←：1　↓：2　→：3）

```
let turtle = new Array(4);
let x = 0;
let y = 0;
let dir = 0;

function preload() {
  for (let i = 0; i < 4; i++) {
    turtle[i]=loadImage("turtle_"+i+".png"); 配列に画像読込
  }
}

function setup() {
  createCanvas(200, 200);
  background(255);
  frameRate(10);
}

function draw() {
  background(255);
  image(turtle[dir], x, y);                 画像を表示
}
```


※2行目～ 4行目の初期値をいろいろと変更し、画像が切り替わることを確かめよう

key / keyCode

アルファベットや数字など、通常のキーは**key**変数に記録される

→例）(**key** == "x") → xを押した場合の判定

矢印キーのような特殊キーは**keyCode**変数に記録される

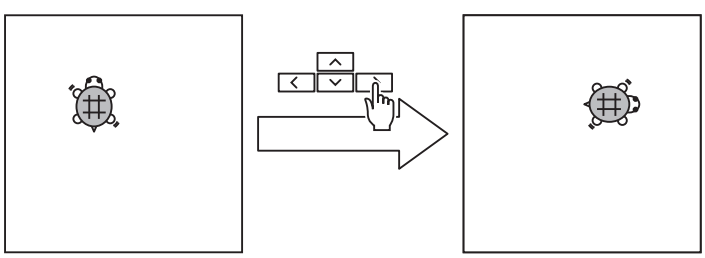
キー	値	キー	値
enterキー	ENTER	↑	UP_ARROW
returnキー	RETURN	←	LEFT_ARROW
TABキー	TAB	↓	DOWN_ARROW
escキー	ESCAPE	→	RIGHT_ARROW

課題

［例題5］の続きをプログラミングしよう

キーボードのカーソルキー（↑←↓→）でキャラクターが動くプログラムを書いてみよう

```
22
23 function keyPressed() {
24   if (keyCode == UP_ARROW) {
25     ____;                          座標を動かす
26     dir = _;                       方向
27   } else if (keyCode == LEFT_ARROW) {
28     ____;                          座標を動かす
29     dir = _;                       方向
30   } else if (keyCode == DOWN_ARROW) {
31     ____;                          座標を動かす
32     dir = _;                       方向
33   } else if (keyCode == RIGHT_ARROW) {
34     ____;                          座標を動かす
35     dir = _;                       方向
36   }
37 }
```

発展

更に、キャンバスをはみ出さないようにするにはどうすればよいかを考えてみよう

プログラムを字下げをする理由

プログラムの中で字下げをしている箇所が目立つ

→{と}の対応箇所が分かりやすいようにしている

●字下げを使わない場合

```
void draw(){
for(int i=1; i<=10; i=i+1){
for(int j=1; j<=10; j=j+1){
maru(i,j);
}
}
}
```

括弧がどのように対応しているかが
分かりにくい

●字下げを使った場合

```
void draw(){
  for(int i=1; i<=10; i=i+1){
    for(int j=1; j<=10; j=j+1){
      maru(i,j);
    }
  }
}
```

括弧がどのように対応しているかが
分かりやすく、構造を認識しやすい

字下げをするには、TABを使おう

キーボードのTABキー⇥を押すと、字下げ位置を揃えることができる

```
void draw(){
|for(int i=1; i<=10; i=i+1){
maru(i,j);
}
}
```

⇥ TABキーを押すと

```
void draw(){
  |for(int i=1; i<=10; i=i+1){
maru(i,j);
}
}
```

TABは文書処理ソフトウェアで位置揃えするにも使える

振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。

□プログラミングを使ってコンテンツを作成することに興味を持つことができた

□マウスのクリックやキーボードの操作によって反応するプログラムを作ることができた

□さまざまなコンテンツが作成されているしくみを理解することができた

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

画面遷移コンテンツのプログラミング

前項では、マウスやキーボードで反応するコンテンツのプログラミングを行ないました。ここでは、ボタンのクリックで画面を遷移していくコンテンツを作成していきたいと思います。ここまでできれば、本当にいろいろなコンテンツの作成ができるようになります。

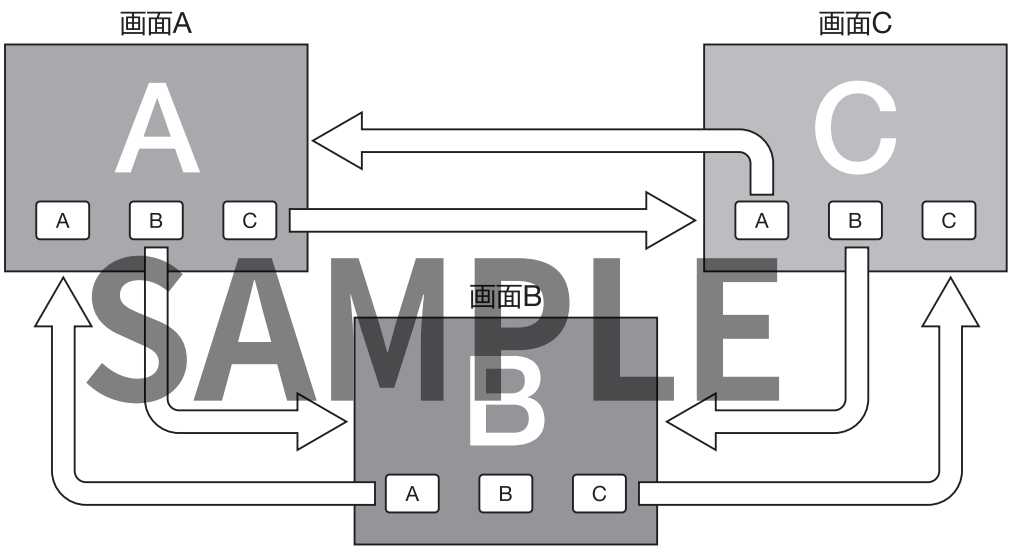
（教科書II：p.28 – p.29 , p.44 – p.47 , 教科書I：p.132 – p.139 , p.160 – p.163）

■ 画面遷移コンテンツのプログラミング

画面遷移コンテンツ

今回は、次のような3つの画面を相互に遷移するコンテンツを考える

画面上のボタンをクリックすると、それぞれに対応した画面が表示されるようにしたい

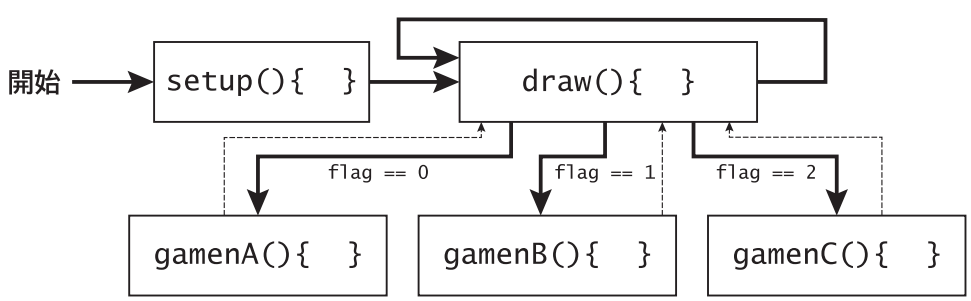


プログラムのイメージ

現在どの画面にいるのかを示すフラグ（**flag**）を用意

→**flag**の値に応じて、画面を切り替える

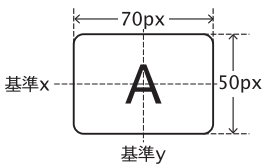
→画面描画はそれぞれ関数を用意し、関数内で描画させる



画面の用意

ボタンを表示する関数を用意する

画面サイズを(400,300)とする
ボタンのサイズは(70,50)とし、中に文字を入れる
中心座標と中に表示する文字列を引数とした関数として設計



例題1

ボタンを表示する関数showButton()を定義してみよう

```
 9 function showButton(x,y,t){      座標と文字列を引数とする
10   fill(255);                    塗り色を白に
11   rectMode(CENTER);             中心座標を基準とする
12   rect(x,y,70,50,5,5,5,5);      四角形の表示（角丸5）
13   textAlign(CENTER,CENTER);     文字の基準を中央に
14   textSize(24);                 文字サイズを24に
15   fill(0);                      塗り色（文字色）を黒に
16   text(t,x,y);                  (x,y)に文字列tを表示
17 }
```

ボタンの配置

ボタンをそれぞれ次の位置に配置する
A：(75,250)　B：(200,250)　C：(325,250)

例題2

setup()関数内で画面サイズを(400,300)としよう
draw()関数内でボタン3つを配置してみよう

```
 1 function setup() {
 2   createCanvas(400,300);        画面サイズ(400,300)に
 3 }
 4
 5 function draw() {
 6   showButton(75,250,"A");       ボタンAを配置
 7   showButton(200,250,"B");      ボタンBを配置
 8   showButton(325,250,"C");      ボタンCを配置
 9 }
```

各画面用関数の作成

画面表示の詳しい内容は後で作成するとして、とりあえず画面を表示する関数を用意
それぞれ、背景色のみ変更するかたちで用意しよう

例題3

画面表示するための関数を用意しよう

```
21 function gamenA(){                  画面Aを表示する関数
22   background(128,0,0);              背景色を赤に
23 }
24
25 function gamenB(){                  画面Bを表示する関数
26   background(0,128,0);              背景色を緑に
27 }
28
29 function gamenC(){                  画面Cを表示する関数
30   background(0,0,128);              背景色を青に
31 }
```

例題4

初期画面を画面Aにしてみよう


```
1 function setup() {
2   createCanvas(400,300);
3   gamenA();                          画面Aを呼び出す
4 }
```

例題5

画面B、画面Cも表示されることを確認しよう

フラグとは

プログラムにおいて特定の動作をさせるための条件づけに使うしるし
→英語の旗（＝flag）が語源　→　「旗が揚がっていればこの動作をする」
そこから転じて、物語の展開を予兆させる要素として使われるようにも
→死亡フラグが揚がると、キャラクターが死亡するストーリー展開に

いまや日常用語になったフラグはプログラミングで使われていた言葉

画面遷移のしくみをつくる

フラグの用意

フラグの変数**flag**を用意し、その値に応じて画面を切り替える

flagの値と画面の対応　→　0：gamenA　1：gamenB　2：gamenC

例題6

変数**flag**を用意し、**flag**によって画面が切り替わるしくみを実装してみよう

```
 1 let flag = 0;                                  変数flagを宣言
…
 8 function draw() {
 9   if(flag == 0){                               もしflagが0なら
10     gamenA();                                    画面Aを表示
11   }else if(flag == 1){                         もしflagが1なら
12     gamenB();                                    画面Bを表示
13   }else{                                       そうでなければ
14     gamenC();                                    画面Cを表示
15   }
16   showButton(75,250,"A");
17   showButton(200,250,"B");
18   showButton(325,250,"C");
19 }
```


例題7

1行目の**flag**の初期値を1、2に変更することで画面が変化することを確認しよう

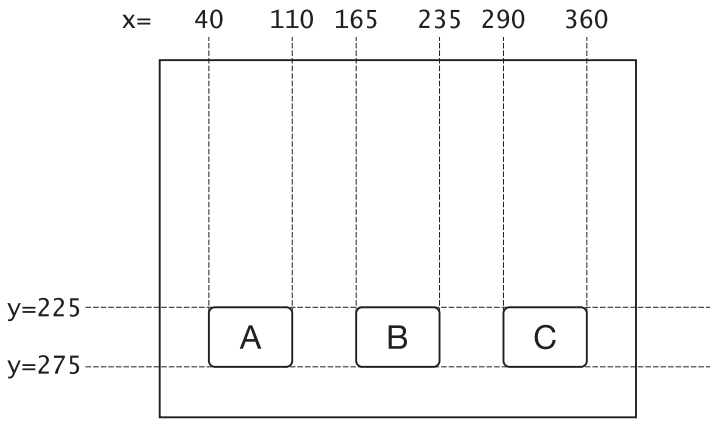
switch文

if文の分岐が同じような条件で続く場合、swith文を使うとスッキリする

```
switch(flag){                    変数flagの値が各caseの値なら、以下を実行
  case 0:                        flag == 0 ならば
    gamenA();                      画面Aを表示
    break;                         switchブロックを抜ける
  case 1:                        flag == 1 ならば
    gamenB();                      画面Bを表示
    break;                         switchブロックを抜ける
  case 2:                        flag == 2  ならば
    gamenC();                      画面Cを表示
    break;                         switchブロックを抜ける
}
```

ボタンクリックのしくみをつくる

mouseClicked()関数に、クリックした位置に応じて反応するしくみを実装する
→マウスの座標(mouseX,mouseY)がボタンの範囲に入っているかをif文で判定



ボタンAの範囲：

```
mouseX>=40 && mouseX<=110 && mouseY>=225 && mouseY<=275
```

同様に、他のボタンの範囲がどのように表されるかを考えてみよう

例題8

ボタンをクリックするとボタンに応じて**flag**の値が変化するようにしてみよう

```
31 function mouseClicked(){
32   if( ボタンAの範囲 ){
33     flag = 0;
34   } else if( ボタンBの範囲 ){
35     flag = 1;
36   } else if( ボタンCの範囲 ){
37     flag = 2;
38 }
```

※本当はもう少し効率的な方法はあるのだが、それは別の章で学ぼう

課題

画面A、画面B、画面Cをデザインし、簡単なコンテンツを制作しよう
※この章で学んできたことを活かして自由にコンテンツを作ってみよう

関数を別ファイルに分ける方法

関数を駆使することで、メインのプログラムがよりスッキリする
関数が増えてくると、プログラム自体が長くなってしまう
→細かい関数などは別ファイルに分けておくと便利

別ファイルの作成方法

画面上の［>］でスケッチファイル一覧を出し、［+］からファイルを追加
→拡張子「.js」でファイルを作成し、関数を記述する
→「index.html」内でファイルを読み込ませる必要がある
→6行目に以下のコードを追加

```
<script src="functions.js" type="text/javascript"></script>
```

※ここでは例としてファイル名を「functions.js」としている
→任意の名前に変えて作ってもよい（拡張子「.js」とする）
→複数ファイルに分けることも可能

振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。
□プログラミングを使ってコンテンツを作成することに興味を持つことができた
□マウスのクリックで画面遷移するプログラムを作ることができた
□プログラミングは少しずつ組み立てて作っていくものであることを実感できた

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

章末問題

[問題]

次のそれぞれのプログラムを実行したとき、画面上にはどのような文字列または数値が表示されますか。ただし、println文は(　　)内の内容を画面上に表示させる命令であるとします。

(1)

```
let x,y;
x = 10;
y = 10;
x = x + y;
println(x);
```

(2)

```
let x,y,z;
x = 10;
y = 20;
z = x;
x = y;
y = z;
println(x);
println(y);
```

x=

y=

(3)

```
let answer = f_calc(1,2);
println(answer);

function f_calc(a,b){
    return a + b;
}
```

(4)

```
let tensu = 45;
let kekka;
if ( tensu >=60 ){
    kekka = 'OK';
} else{
    kekka = 'NG';
}
println(kekka);
```

(5)

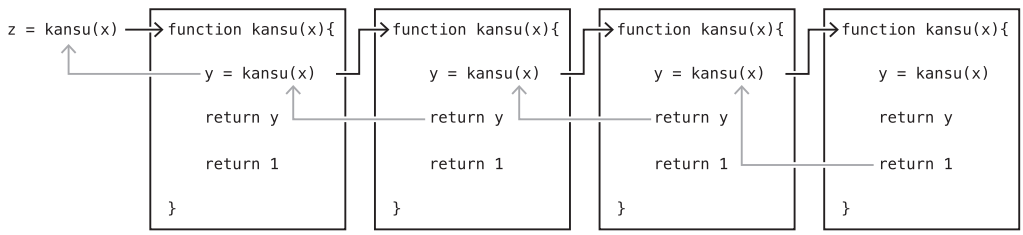
```
let answer = f_calc();
println(answer);

function f_calc(){
    let x = 0;
    for( let i = 1; i <= 5; i = i + 1 ){
        x = x + 1;
}
```

コラム～再帰関数

■ 再帰関数

再帰関数

再帰関数＝ 関数の中で自分自身の呼び出しが含まれているもの



※必ず終了条件が必要　→　終了条件がなければ無限ループに陥る

再帰関数を使わない場合

1からnまでの整数を足し合わせるプログラムは次のようになる

```
let x = gokei(100);

function gokei(n){
  let s = 0
  for(let i=1; i<=n; i++){
    s += i;
  }
  return s;
}
```


再帰関数を使った場合

1からnまでの整数を足し合わせるプログラムは次のようになる

```
let x = gokei(100);

function gokei(n){
  if(n < 1){
    return n;
  }
  return n + gokei(n-1)
}
```

nを1減らして
同じ関数を再度実行

SAMPLE

2025年度「情報Ⅱ」ワークブック見本
CQ出版社