**Processing で初めてのプログラミングに挑戦！**

**目次**

[1. Processing(プロセシング)とは？ 2](#_rlialwlcadvk)

[2. 簡単なプログラミングをしてみよう 4](#_od1p1dmblpxo)

[2.1. プログラムの基本 5](#_ukrp1b1crby)

[2.2. コメントを書こう～こっそりメモ～ 5](#_4amqhys5nwii)

[2.3. 最初のプログラムの解説 6](#_j6mn8aqj316e)

[3. プログラミングの基礎 7](#_hkolyzky1fzi)

[3.1. 変数～数字の入った箱～ 7](#_u29ezj768p3a)

[3.2. 四則演算～たす・ひく・かける・わる・あまり～ 9](#_vaaa7sybiabf)

[3.3. 論理演算  
～○○よりも大きい？小さい？、「かつ」と「または」～ 10](#_svkuh39abm05)

[3.4. 制御構文～「もし○○のとき」と「○○である間は」～ 13](#_229u73y976gr)

[3.5. 図形を動かしてみよう 17](#_rq87ynyhs7tg)

[3.6. 乱数 19](#_xg9syd8mnu36)

[3.7. 指定した位置に描画してみよう 20](#_f7azok4b19mo)

[4. 応用編 21](#_4uxztzpkplwr)

[4.1. 関数 21](#_72q3bao6a8ug)

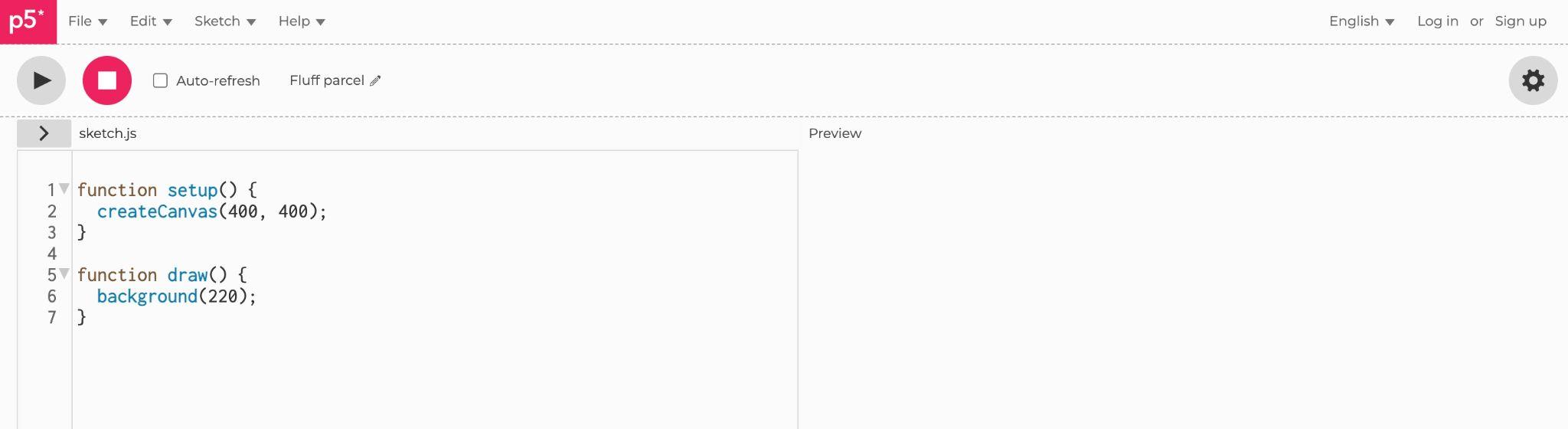
[4.2. 再帰関数 22](#_vic3w4tnohp3)

[4.3. 自由課題 25](#_28scl02dpzdo)

# **Processing(プロセシング)とは？**

初心者向けのプログラミング学習環境です。今日はウェブブラウザでプログラムを実行して学んでいきます。まずは以下の URL にブラウザでアクセスしましょう。

<https://editor.p5js.org/>



| 👆**ヒント** 初期設定では表示言語が英語になっています。右上の「English ▼」をクリックして日本語に変更するとわかりやすいです。 |  |
| --- | --- |

| 画面左上の再生ボタンをクリックすると、書いたプログラムが実行されます。停止ボタンをクリックするとリセットされます。試しに再生ボタンをクリックしてみましょう。 |  |
| --- | --- |
| プレビューに、右図のような灰色の四角形が表示されていれば、成功です。これであなたも、もうプログラマーの一員ですね。 |  |

# 簡単なプログラミングをしてみよう

まずは、いちばん簡単なプログラムを書いてみましょう。

以下のようなプログラムに書き換えてみます。

**📄プログラム ① いちばん簡単なプログラム**

| function setup() {  createCanvas(400, 400);  }  function draw() {  background(255, 255, 255);  fill(255, 0, 0);  ellipse(200, 200, 200, 200);  } |
| --- |

(赤色の部分を書き換えよう)

| 完成したら、再生ボタンをクリックして、実行結果をみてみましょう。  日本の国旗のような画像が表示されましたか？ |  |
| --- | --- |

| 👆**ヒント** 自動更新 にチェック✅️をいれておきましょう。いちいち再生ボタンをクリックしなくても、プレビューにすぐに結果が反映されるようになります。 |
| --- |

## プログラムの基本

ほとんどのプログラミング言語は行単位で上から順番に実行されます。

先のコードでは、 setup が実行され、 createCanvas が実行され、 draw が実行され...というように、上から順番に実行されていきます。

## コメントを書こう～こっそりメモ～

最初のうちはチンプンカンプンになりがちです。英語で記述しないといけないというのも、私たち日本語話者には難しいポイントです。大抵のプログラミング言語にはコメントという構文があり、コメントに書いた内容は実行時に無視されます。Processing では // スラッシュ2個以降はコメントとみなされて、実行時には無視されます。

**例① 一行コメントの書き方**

| // これはコメントです  background(255, 255, 255); |
| --- |

**例② 行末コメントの書き方**

| background(255, 255, 255); // 行末に書いてもOK |
| --- |

これからそれぞれの行で何をしているか解説していきます。適宜自分のプログラムにコメントを書いていくとわかりやすくなるでしょう。

## 最初のプログラムの解説

createCanvas

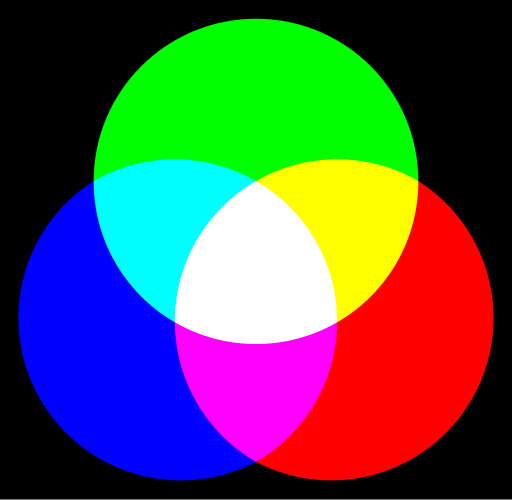
Processing では描画する範囲のことをキャンバス(canvas)と呼びます。

先のコードでは 400x400 サイズのキャンバスを作成しろ、と命令しています。キャンバスの左上が x,y 座標の (0,0) で、右下が (400,400) です。

| **🤔試してみよう①** より大きなキャンバス(例えば 800x800)に変えてみよう |
| --- |

background

背景の色を指定する命令です。多くのプログラミング言語では加法混合で赤・緑・青の原色の値を 0-255 の範囲で指定して色を表現します。例えば 255, 255, 255 は白になります。逆に 0, 0, 0 は黒になります。 255, 0, 0 は赤...というように、3つの値の組み合わせによって色を表現します。



| **👊チャレンジ①**  黄色を表示させたい時は、３つの値をどう設定するべきだろう？  正解だと思う数値に、プログラムを書き換えてみよう。 |
| --- |

fill

塗りつぶしの色を指定する命令です。色の指定は background と同様で、3つの値で表現します。

ellipse

円を描画させる命令です。fill で指定された色で塗りつぶされて描画されます。4つの値は描画する座標の位置と大きさを指定します。x, y, dx, dy はそれぞれ中心の x,y 座標、dx は横幅、dy は高さです。

| **🤔試してみよう②** 値を変えてみて、どのような丸が描けるか確認しよう。 |
| --- |

| **🤔試してみよう③**  色の違う円を複数描いてみよう。 |
| --- |

# プログラミングの基礎

## 変数～数字の入った箱～

変数は、値を格納しておく箱のようなもののことです。Processing では let 構文で変数を宣言できます。変数には = で値を代入できます。

**例③ 色をあらわす変数**

| let R = 255; // 赤(Red)  let G = 255; // 緑(Green)  let B = 255; // 青(Blue)  background(R, G, B); |
| --- |

上記のように書き換えた場合、R という変数と G という変数、B という変数を定義して、それぞれに 255 を格納しています。変数を定義すると、以降の行ではその変数を参照できるようになり、 background に R と G と B の値を渡すことができるようになります。

変数に変数を代入することもできます。

**例④ 変数に変数を代入する**

| let x = 0;  let y =10;  x = y;  print(x) // x … 10  print(y) // y … 10 |
| --- |

数学では x = y と書くと x と y は同じ値だという意味になりますが、プログラミングでは代入する(y の値を x にコピーする)という意味になるので、注意しましょう。

変数に何が代入されているか知りたい時は print を使うと左下のコンソールに表示されるので、わからなくなったらコンソールに表示して確認してみるとよいでしょう。

**例⑤ printの使い方**

| let x = 5;  print(x); // 5 が表示される |
| --- |

## 四則演算～たす・ひく・かける・わる・あまり～

数学の四則演算や、剰余を計算できます。

**例⑥ 四則演算と剰余の計算**

| let x = 1 + 1;// … 2 足し算  x = 10 - 8;// … 2 引き算  x = 2 \* 2; // … 4 掛け算  x = 10 / 5; // … 2 割り算  x = 10 % 4; // … 2 剰余 |
| --- |

通常の四則演算同様、掛け算と割り算が足し算と引き算より優先されます。括弧をつけることで優先順位を変更できます。

**例⑦ 計算の順序**

| let x = 2 + 3 \* 4; // … 14  x = (2 + 3) \* 4; // … 20 |
| --- |

| **👊チャレンジ②**  太郎と花子のお小遣い帳をつけてみよう  太郎は1000円、花子は1000円持っています。  花子は太郎に返す時に倍にする約束で500円借りました。花子はやりくりの結果小遣いを1.2倍にすることができました。  花子は返済日に約束通りの金額を太郎に支払いました。  太郎と花子の小遣いを表示してみよう。 |
| --- |

## 論理演算 ～○○よりも大きい？小さい？、「かつ」と「または」～

コンピュータは2進数といって0と1の羅列で制御されています。 0 を false 、1 を true と表現します。

**例⑧ 論理演算～○○よりも大きい？小さい？～**

| let x = 2;  x > 0; // … true x は 0よりも大きい？  x < 0; // … false x は0よりも小さい？  x >= 2; // … true x は2以上？  x <= 3; // … false x は3以下？  x == 2; // … true x は2と等しい  x != 2; // … true x は2と等しくない |
| --- |

この true と false は boolean (ブーリアン)といって、以下のような論理演算ができます。

論理**和(OR)**

| || | true | false |
| --- | --- | --- |
| true | true | true |
| false | true | false |

論理**積(AND)**

| && | true | false |
| --- | --- | --- |
| true | true | false |
| false | false | false |

これらを組み合わせると

**例⑨ 論理演算～「かつ」～**

| let x = 2;  4 > x && x > 0; // … true x は 0 より大きい、かつ4よりも小さい |
| --- |

実行順は左から順ですが、論理演算が最後に実行されます。つまり、 4 > x がまず評価され、次に x > 0 が評価され、最後に true && true のような形で論理積が計算されます。

**例⑩ 論理演算～「かつ」と「または」～**

| let x = 2;  let y = 4;  let z = 5  // x が0よりも大きいか、y が5よりも大きい  x > 0 || y > 5 // … true  // x は2と等しいかつ、y が4と等しいかつ、z が5と等しい  x == 2 && y == 4 && z == 5 // … true |
| --- |

| **👊チャレンジ③**  与えられた変数が偶数が判定してみよう。 |
| --- |

| **👊チャレンジ④**  与えられた変数が偶数かつ5の倍数かどうか判定してみよう。 |
| --- |

## 制御構文～「もし○○のとき」と「○○である間は」～

特定の条件の時のみ命令を実行したい時に使うのが if（もし○○のとき） です。

**例⑪ if（もし○○のとき）**

| let x = 2;  if (x == 2) { // x が2と等しい時だけ円を描画する  ellipse(200, 200, 200, 200);  } |
| --- |

else（それ以外の時には～） を組み合わせると条件を満たさない場合の命令を指定できます。

**例⑫ if（もし○○のとき） と else（それ以外の時には～）**

| let x == 4;  if (x == 2) {  ellipse(200, 200, 200, 200);  } else { // x が2以外の場合大きい円を描画する  ellipse(200, 200, 400, 400);  } |
| --- |

また、if と else は組み合わせることもできます。

**例⑬ if（もし○○のとき） と else（それ以外の時には～） の組み合わせ**

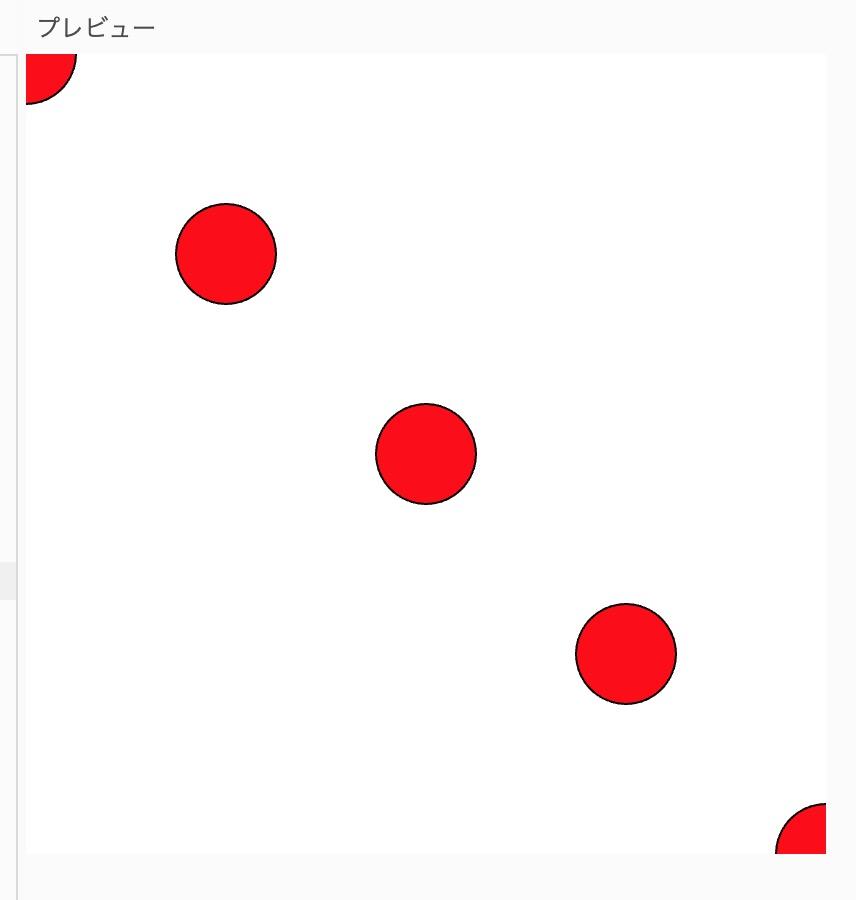
| let x == 5;  if (x == 1) { // x が1と等しいなら 100x100 の円  ellipse(200, 200, 100, 100);  } else if (x == 2) { // x が2と等しいなら 200x200 の円  ellipse(200, 200, 200, 200);  } else { // それ以外は 400x400 の円  ellipse(200, 200, 400, 400);  } |
| --- |

同じ処理を繰り返す時に使うのが while（○○である間は） です。

**例⑭ while（○○である間は）**

| let i = 0;  while (i < 5) {  ellipse(i \* 100, i \* 100, 50, 50);  i = i + 1;  } |
| --- |

**結果**



while の括弧の中に、論理式を書いて、それが true の間繰り返し実行されます。

順を追って説明すると、

①まずは i が0からスタートして (0,0) の位置の円が描かれ、i に i + 1(つまり 0 + 1)を代入します。

②次に i < 5 が評価され、i (= 1) は5よりも小さいので (100,100) の位置に円が描かれ、i に i + 1(つまり 1 + 1)を代入します。

③これらを繰り返し、i が5になると i < 5 が false となるので実行が終了します。

上記の while を簡単にに書くために使う、 for という構文もあります。

**例⑮ for**

| for (let i = 0; i < 5; i = i + 1) {  ellipse(i \* 100, i \* 100, 50, 50);  } |
| --- |

これは先ほどの while で書いたものと同じ結果になります。for の中には初期化する際の式(let i =0)と、条件式(i < 5)と、繰り返し時の処理(i = i + 1)を記述します。

注意!! 繰り返し構文は、書き方を間違えると無限ループに入ってしまいます。無限ループに入るとフリーズすることがあるので、その時はブラウザを再起動しましょう。自動更新のチェック✅️を外して、再生ボタンをクリックして実行するとよいでしょう。

| **👊チャレンジ⑤**  以下のようなドット模様を描画してみよう。 |
| --- |

## 図形を動かしてみよう

ここまでは静止した図形を描画してきました。

Processing では 、setup は1度だけ実行され、draw は繰り返し実行されるという仕組みになっています。なので、draw の中を変数や制御構文を使って時間的に変化させることで図形を動かすことができます。

以下のプログラムを書いてみましょう。

**📄プログラム ② 図形を動かすプログラム**

| let x = 0;  let l = 400;  function setup() {  createCanvas(l, l);  }  function draw() {  background(255, 255, 255);  fill(255, 0, 0);  ellipse(x, x, 50, 50);  x = x + 1;  if (x > l) {  x = 0;  }  } |
| --- |

赤い丸が左上から右下に動きましたか？

最初に x を定義して、draw が繰り返し実行される毎に1ずつ加算されていきます。そしてx が l よりも大きくなったら0に戻すことにより、また0から繰り返し実行される、という仕組みです。

## 乱数

これまで座標等の数値は固定の値として与えていましたが、ランダムな値を使ってみましょう。以下のようなプログラムに書き換えます。

**📄プログラム ③ ランダムに図形を動かすプログラム**

| let l = 400;  function setup() {  createCanvas(l, l);  frameRate(10);  }  function draw() {  background(255, 255, 255);  fill(255, 0, 0);  ellipse(random(l), random(l), 50, 50);  } |
| --- |

random は指定した値までの大きさのランダムな数値を1つ返してくれます。今回はキャンバスの大きさの中でランダムな位置を指定して丸を描画するプログラムにしました。

| **👊チャレンジ⑥**  丸の色もランダムにしてみよう。 |
| --- |

## 指定した位置に描画してみよう

ここまではプログラムを実行したら眺めているだけでしたが、Processing では何らかのアクションを元に描画することができます。以下のようなプログラムに書き換えます。

**📄プログラム ④ 指定した位置に図形を動かすプログラム**

| function setup() {  createCanvas(400, 400);  }  function mouseClicked() {  background(255, 255, 255);  fill(255, 0, 0);  ellipse(mouseX, mouseY, 50, 50);  } |
| --- |

mouseClicked は、マウスがクリックされたら｛｝（波かっこ）の内容を実行するものです。また、mouseX と mouseY は特殊な変数で、現在のカーソルの位置を x,y 座標で取得できます。よって、マウスがクリックされたらマウスの位置に丸を描画するプログラムになります。

| **🤔試してみよう④** background(255, 255, 255); の行を消すとどうなるだろう？ |
| --- |

# 応用編

## 関数

これまで使ってきた setup や draw は**関数**といいます。ellipse なども、**関数**です。

かっこの中で指定している x, y, dx, dy を**引数**といいます。また、これらは**プリミティブ関数**といって Processing があらかじめ定義してくれている関数です。

それらとは別に、自分で関数を定義できます。

**例⑯ 自作関数**

| function add(x, y) {  return x + y;  } |
| --- |

return の後の式のことを**返り値**といい、関数を呼び出すとその値が返されます。何も返さないと、処理を打ち切ります。

**例⑰ 自作関数の呼び出し方**

| function draw() {  background(255, 255, 255);  fill(255, 0, 0);  ellipse(add(50, 100), add(50, 100), 50, 50);  } |
| --- |

プリミティブ関数と同様にかっこで呼び出せます。

上記では、 ellipse に対して 150 と 150 が渡されます。

### 

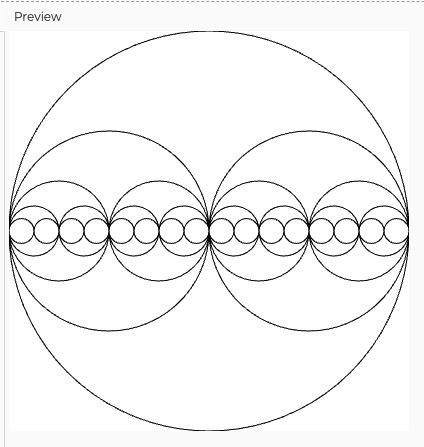
## 再帰関数

定義した関数の中で、関数をふたたび呼び出すような関数のことを**再帰関数**といいます。

**📄プログラム ⑤ 再帰関数を使ったプログラム**

| function setup() {  createCanvas(400, 400);  background(255, 255, 255);  circles(200, 200, 200, 4);  }  function circles(x, y, r, n) {  ellipse(x, y, r \* 2, r \* 2);  if (n < 1) {  return;  }  let dr = r / 2;  circles(x + dr, y, dr, n - 1);  circles(x - dr, y, dr, n - 1);  } |
| --- |

**結果**



このような、拡大しても似た形が出てくるような図形を**フラクタル図形**といいます。

上記のコードは

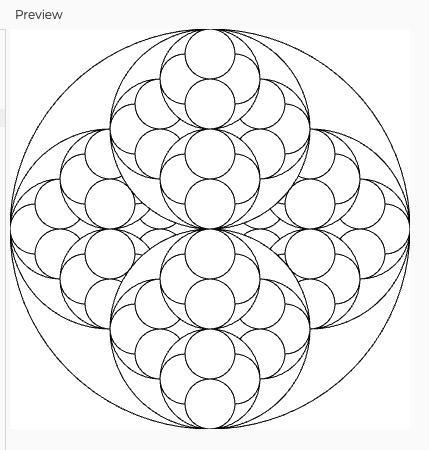
①最初に circles に 200, 200, 200, 4 がそれぞれ x, y, r, n に渡されるので、200, 200 に 400x400 の円が描画され、最初の circles で 300, 200 に 200x200 の円が描画され、

②次に 350, 200 に 100x100 の円が描画され...というように繰り返されて、

③n が 1 よりも小さくなったら(つまり5回実行されたら)処理を終了するような挙動になります。

最初はどういう風に実行されているのかわからないかもしれません。その時は、その都度手計算で、行毎にどのように値が変化するのか書き出してみると理解しやすくなります。

| **👊チャレンジ⑧** n = 3 の時に、上のように上下にも円を表示できるように書き換えてみよう。 |
| --- |

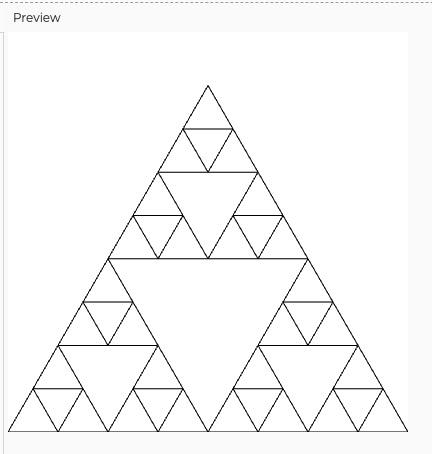


## 

## 自由課題

ここまでできたらあとは自由にプログラムを書いてみましょう。

アイデアが浮かばない人のために、以下に参考になるような図形を示します。



**シェルビンスキのギャスケット(n=3)**

必要になる関数

triangle

x1, y1, x2, y2, x3, y3

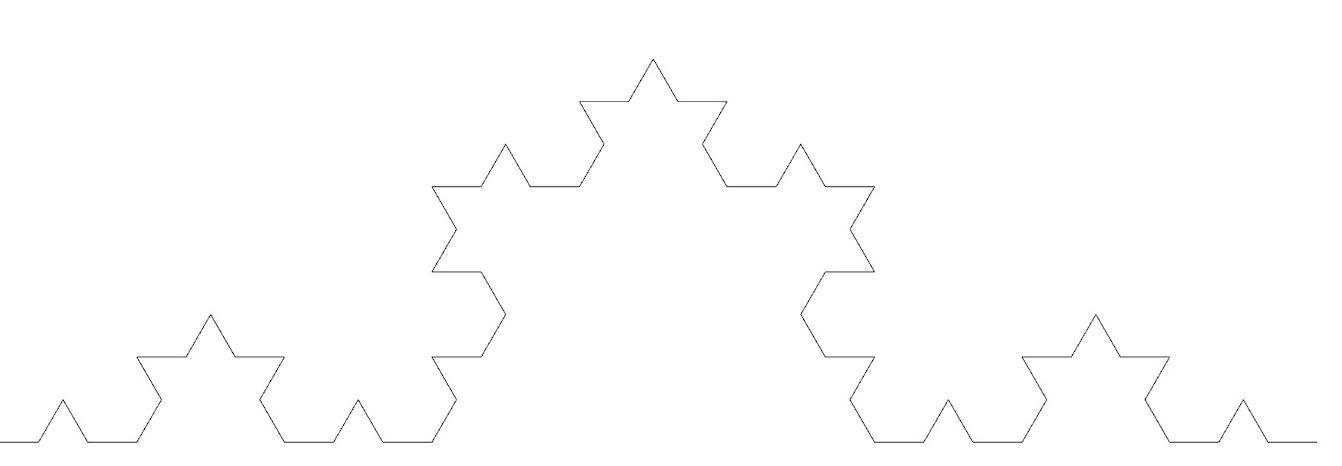
3点を指定して三角形を描画

sin, cos, tan

それぞれ三角関数

PI

円周率



コッホ曲線(n=4)

line

x1, y1, x2, y2

2点を指定して線を描画

この他にも Processing にはたくさんの関数が用意されています。

<https://p5js.org/reference/>

上記 URL でその一覧を確認できます(英語)

<https://p5js.org/examples/>

また、サンプルもたくさんあるので面白そうなプログラムを真似して、変えてみよう。