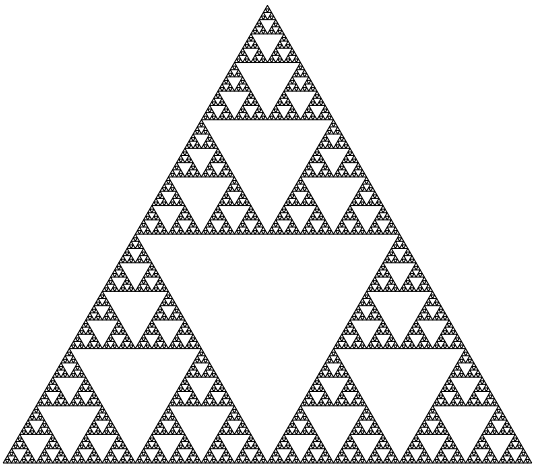
8 フラクタル図形



ここまでに学んだ，関数の再帰的定義を利用して，**フラクタル図形**と呼ばれる図形を描いてみよう．

フラクタルとは，正確な数学的定義は難しいが，「自己相似な図形」を意味する．図形のある部分が，図形全体と相似になっている．例えば，右の**シェルピンスキーのガスケット**は有名なフラクタル図形の 1 つである．正三角形の中に正三角形を描く操作を無限回行って出来る図形であるから，n 回行ってできる図形の極限として定義される．実際には，右の図は有限回で操作を打ち切って出来る近似である．

8.1 タートルグラフィックス

turtle モジュールについては既に 1 度紹介したが，今回はこの機能をたくさん利用するので改めてここに載せておこう．この他の関数を詳しく知りたい場合は，`import turtle` の後で `help(turtle)` を実行すると良い．

タートルを動かす関数	
`forward(d)`, `fd(d)`	タートルが頭を向けている方へ，タートルを距離 `d` だけ移動.
`backward(d)`, `bk(d)`, `back(d)`	タートルが頭を向けている方と逆方向へ，タートルを距離 `d` だけ移動．**タートルの向きは変えない**.
`right(a)`, `rt(a)`	タートルの向きを角度 `a`（度）だけ右回転.
`left(a)`, `lt(a)`	タートルの向きを角度 `a`（度）だけ左回転.
`setposition(x, y)`, `setpos(x, y)`, `goto(x, y)`	タートルを座標 `(x, y)`（絶対位置）へ移動. （ペンが下りている場合は線も引く．以下他の関数も同じ）
`setx(x)`	タートルの x 座標を `x` に指定.
`sety(y)`	タートルの y 座標を `y` に指定.
`setheading(a)`	タートルの向きを角度 `a`（絶対値，0° は右向き）に指定.
`home()`, `seth(a)`	タートルの位置を初期化（座標 `(0, 0)` に移動し，右向き）.
`circle(r, extent, steps)`	半径 `r` の円弧を，中心角 `extent` 度だけ描く．中心はタートルの左 90° 方向に距離 `r` の点．円は内接する正 `steps` 角形で近似される．`extent`，`steps` は省略可能.

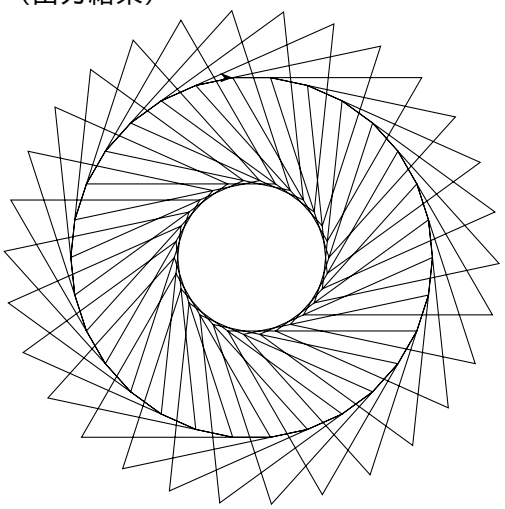
ペンを制御する関数	
pendown(), pd(), down()	ペンを下ろす（移動の度に線が引かれるようになる）.
penup(), pu(), up()	ペンを上げる（線が描かれなくなる）.
pensize(w), width(w)	線の太さを w に設定．引数を与えない場合は現在の太さを返す.
pencolor(c)	ペンの色を c に設定．引数を省略すると現在のペン色を返す．c は"red"や"#ff0000"や (1.0,0,0)（RGB 値）などで指定.
pen(p)	ペンの属性を辞書 p で設定．引数を省略すると現在のペンの属性を辞書で返す．（現在のペンの状態の記録/復元に使う）
塗りつぶし	
begin_fill(), end_fill()	begin_fill() 実行時のタートルの位置から，end_fill() 実行時までに描かれた図形を塗りつぶす．開始点と終了点が異なる場合はその 2 点を結んでから塗りつぶす.
fillcolor(c)	塗りつぶしの色を設定．変数 c は pencolor(c) と同様.
その他	
undo()	最後の移動を取り消す．取り消し可能な回数の上限は setundobuffersize(s) で指定可能（デフォルトは 1000 回）.
speed(s)	タートルの移動アニメーションの速さを s（0 から 10 までの整数）に設定．数字が大きいほど速い．ただし 0 が最速（アニメーションなし）．引数を与えない場合は現在の速さを返す.
tracer(n)	タートルの移動中のスクリーン更新回数を通常の 1/n 倍に設定する．n=0 の場合は更新を行わない（アニメーションなし）.
update()	スクリーンを更新する．tracer(0) の場合に必要.
reset()	タートルの描いたものを全て消去し，タートルやペンの状態を全て初期化.
clear()	タートルの描いたものを全て消去する．タートルやペンの状態はそのまま.
hideturtle(), ht()	タートル自身（矢印のような図形）を描画しないように設定．タートルを隠すと線の描画はかなり速くなるので，複雑な図を描くときには推奨.
showturtle(), st()	タートル自身（矢印のような図形）を描画するように設定.
screensize(w,h)	キャンバスの幅と高さを変更する．描画ウィンドウの大きさは変更されないので，画面外を見るためにはスクロールバーを使う．引数を省略すると現在の幅と高さを返す.

♣ **注意 50** 重要！ Spyder で turtle モジュールを利用する場合は，まずコンソール画面で「%gui tk」を実行すること．

例 27

```
1  import turtle as t
2
3  t.speed(10)
4  # t.tracer(0)
5
6  n = 12
7  for i in range(360//n):
8      # 円周上を移動
9      t.fd(25)
10     t.rt(n)
11     # 正三角形を描く
12     for j in range(3):
13         t.fd(125)
14         t.rt(120)
15
16 # t.update()
```

（出力結果）



問 30

(1) 上のプログラムを実行せよ．

(2) 亀がのそのそ動くのを観ているのは，それはそれで楽しくもあるのだが，複雑な図形を描いてもらおうというときには少々急いでいただきたいと思うかもしれない．上のプログラムでコメントアウトしてある`# t.tracer(0)` や`# t.update()` のコメントを外して，これらの行も実行させるとどうなるか，確かめてみよ．

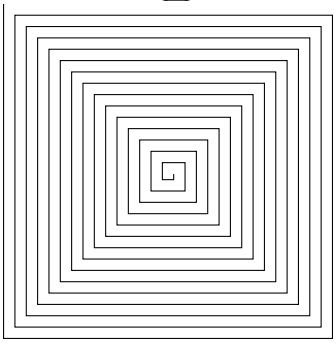
♣ **注意 51** 複雑な図形を描く際に，コードに誤りがあって描画結果が望むものにならなかった場合は，タートルが動いている所が見える方が原因の特定に役立つ場合もある．その場合は，`tracer(0)` は実行しない方が良いだろう．

例 28 再帰を利用してタートルグラフィックスで図形を描く関数を定義してみよう．

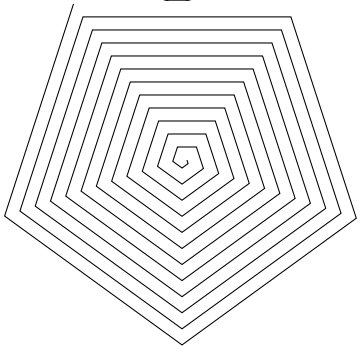
```
1  import turtle as t
2
3  t.speed(0)
4  t.ht()
5  t.tracer(0)
6
7  def spiral(a, d=5, n=0, n_max=300):
8      if n < n_max:
9          t.fd(n)
10         t.rt(a)
11         spiral(a, d, n+d, n_max)
12         t.update()
```

（出力結果）（前の実行結果を消すためには `t.reset()`↵）

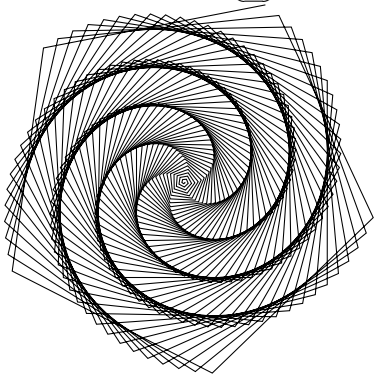
`spiral(90)`↵



`spiral(72)`↵



`spiral(71, d=1)`↵



♣ **注意 52** 関数 `spiral()` の定義の中で，引数のところに `n=0`，`d=5`，`n_max=300` のように書かれている．これは**デフォルト引数**と呼ばれる機能で，関数を利用する際に明示的に値を指定しない場合には，予め指定してあった**デフォルト値**[注 25]が自動的に利用される．

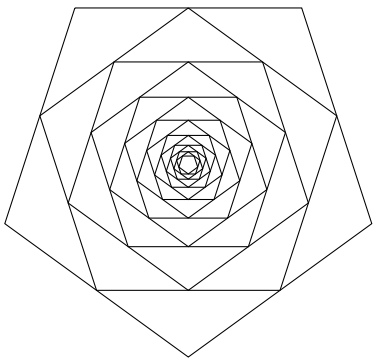
一筆書きでは描けないような図形を描く場合には，`penup()`, `pendown()` を利用して線を引かない移動と線を引く移動を制御する．

例 29

```
1   import turtle as t
2   import math
3
4   t.speed(0)
5   t.ht()
6   t.tracer(0)
7
8   def polygon(n, e=200, m=15):
9       if m > 0:
10          # 1 辺の長さ e の正 n 角形を描く
11          for i in range(n):
12              t.fd(e)
13              t.rt(360/n)
14
15          t.penup()
16          t.fd(e/2)
17
18          # 次の辺の長さは余弦定理で求めた
19          c = math.cos(math.radians(360/n))
20          next_e = e * (2 + 2*c)**0.5 /2
21          t.rt(360/n/2)
22          t.pendown()
23
24          polygon(n, next_e, m-1)
```

（出力結果）

`polygon(5)`↵



[注 25] 普通は，イミュータブルな値をデフォルト値に設定する．リストなどの変更可能なデータ型を設定してしまうと，関数の外でそのリストが変更を受けた場合にはデフォルト値にも影響するので注意が必要である．例えば，`def f(x, l=LIST_A):`のような関数を作ったとき，関数の外で `LIST_A[0] = 5` のような書き換えを行うことは望ましくない（そもそも `l=LIST_A` のような指定自体が望ましくないが…）．

8.2 パッケージの追加インストール

Python には，外部ライブラリのインストールと管理を行うための「pip」というツールが付属している．デフォルトではインストールされていない外部のパッケージをインストールしたり，各種ライブラリのバージョンアップを行うときに利用する．

pip の使い方
コマンドプロンプトなどから，以下のコマンドを実行すれば利用できる．

- `pip install` ライブラリ名
  PyPI（Pyton Package Index）という Python 公式サイトのサービスに登録されているライブラリを指定してインストールする．
- `pip install URL`・ファイル名など
  PyPI に登録されていないパッケージを，URL やファイル名を直接指定してインストールする．
- `pip install --upgrade` ライブラリ名
  既にインストール済のパッケージのバージョンを確認し，更新する．

♣ **注意 53** Windows 環境では，pip 単体でインストールしようとしても，途中でインストールに失敗するパッケージも存在する．その場合は，インストールのための「wheel ファイル」と呼ばれる形式のファイルが配布されていることがあるので，それを探してダウンロードし，ファイル名を指定して `pip install` を実行する．

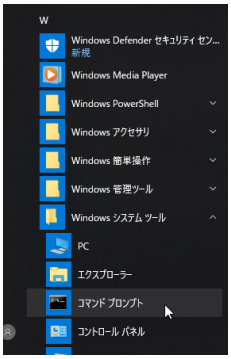
- Unofficial Windows Binaries for Python Extension Packages
  `https://www.lfd.uci.edu/~gohlke/pythonlibs/`

というサイトでは，（非公式ビルドだが）Windows 用の wheel ファイルがたくさん配布されているので知っておくと良い．

タートルグラフィックスは，手軽に動かせて絵を描く実験をするのには良いパッケージだが，せっかく作った絵を出力する機能を持っていない．そこで，今回はタートルで描いた図を SVG 形式（Scalable Vector Grapchics）で保存する拡張機能「`turtlesvg` パッケージ」を用意した．この教室の PC にはインストール済である．自宅などでも図を作ってみたい場合には，GitHub (`https://github.com/NSatoh-azb/lecture2018-01`) にこの講座のサイトを準備してあるので，以下の手順で利用可能である：

- Git というバージョン管理ソフトウェアのインストールが必要である．Git for Windows をインターネットで検索し，インストールしておく．
- 「コマンドプロンプト」を起動する．

（下の図は Windows 10 の場合のスタートメニュー）



- 以下のコマンドを実行する：

  `pip install git+https://github.com/NSatoh-azb/lecture2018-01`

次のような表示になればインストール成功である．

```
C:\Users\NSatoh>pip install git+https://github.com/NSatoh-azb/lecture2018-01
Collecting git+https://github.com/NSatoh-azb/lecture2018-01
  Cloning https://github.com/NSatoh-azb/lecture2018-01 to c:\users\ 略
Installing collected packages: turtlesvg
  Running setup.py install for turtlesvg ... done
Successfully installed turtlesvg-1.3
```

インストールした「turtlesvg」パッケージは，Python 付属のタートルモジュールの代わりに呼び出して利用する．

turtlesvg パッケージの使い方

- `import turtlesvg as ttl` のようにインポートする．
- `t = ttl.MyTurtle()` を実行して，`MyTurtle` オブジェクトを生成する．
- 通常の turtle モジュールと同じように，「`t.forward(100)`」「`t.left(60)`」などを実行してタートルを動かす．
  タートルモジュールの全ての機能には対応していない．undo() などが使えないので注意．
- 絵を描き終わったら，`t.penup()` を実行（これを忘れると最後に描いた線が保存されない）．
- `t.save_as_svg(ファイル名)` を実行すると，「ファイル名」に SVG 形式で保存する．
  ※ 現在実行しているファイルと同じフォルダに保存される．
  ※ ファイル名は，「`'hoge.svg'`」のように，最後に「`.svg`」を付けること．

簡単に言えば，これまでのプログラムで

```
1  import turtle as t
```

と書いていた箇所を，

```
1  import turtlesvg as ttl
2  t = ttl.MyTurtle()
```

に書き換えればそのまま動く．絵を描かせたあと，

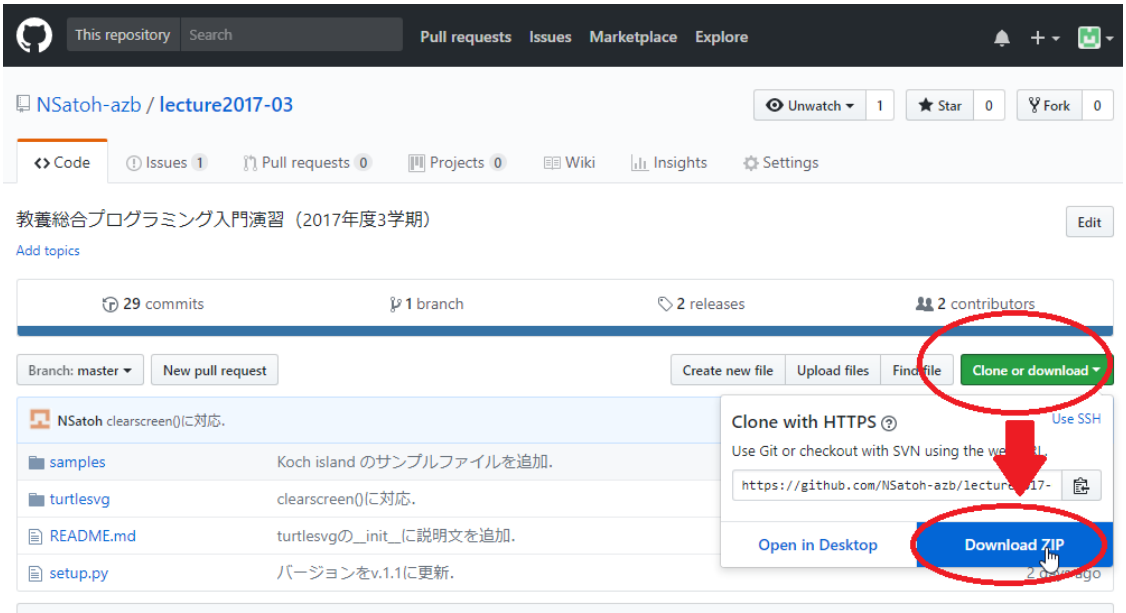
```
1  t.penup()
2  t.save_as_svg('ファイル名.svg')
```

を実行すればSVGファイルに画像が保存される．

問 31 ここまでのプログラムのうち，「spiral」「polygon」を新たにインストールしたturtlesvgパッケージ用に書き換えたものを，

- https://github.com/NSatoh-azb/lecture2018-01

にアップロードしてある．インターネットブラウザを利用して上記サイトにアクセスし，「Clone or download」ボタンから「Download ZIP」を選択してダウンロードせよ．



保存したZIP（圧縮形式）ファイルを解凍したら，「samples」フォルダの中から「spiral.py」「polygon.py」を自分のpythonファイル保存フォルダにコピーして，Spyder上で実行してみよ．プログラムを保存しているフォルダ内にSVGファイルが作成されていることを確認せよ．（SVGファイルの表示のしかたは右ページを参照）

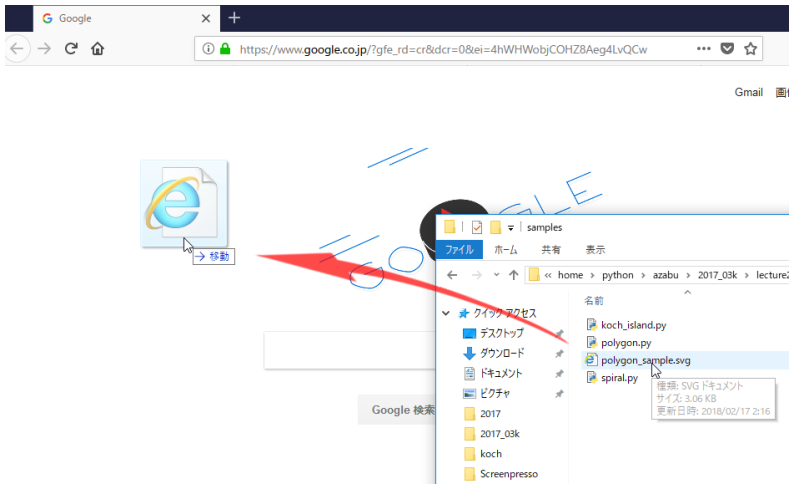
8.3 SVG 画像の見方

SVG(Scalable Vector Graphics) は，XML(Extensible Markup Language) をベースとした画像形式で，web ブラウザを利用して表示させることが出来る．

問 32 次の手順で，前問で作成した SVG 画像を表示せよ．

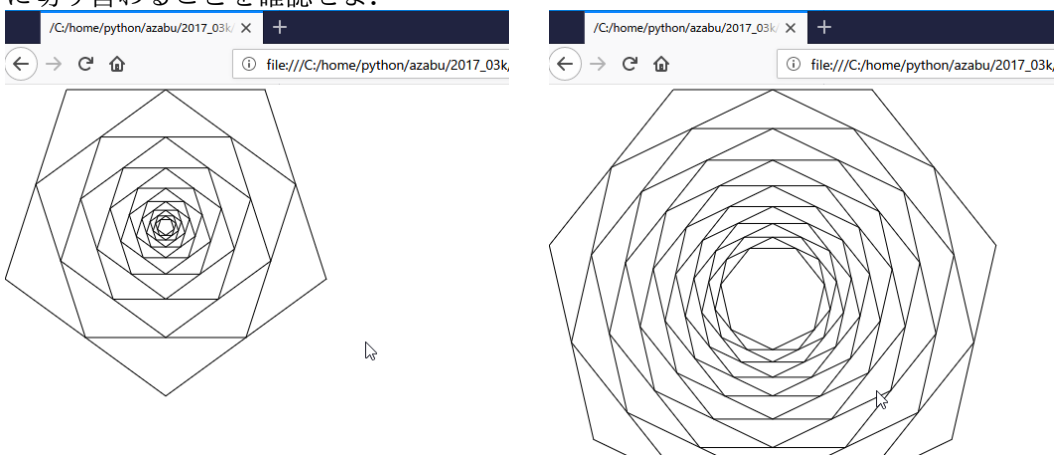
(1) FireFox や Chrome などの web ブラウザを起動せよ．

(2) 作成した SVG ファイルをブラウザ上にドラッグ＆ドロップせよ．



(3) **ブラウザはそのまま閉じずに**，Spyder に戻って，同じプログラムをパラメータを変更して実行し，SVG 画像を上書きせよ．（例えば，前問で `polygon(5)` を実行して保存したなら，`polygon(7)` などを再び実行し，同じファイル名で SVG を上書きせよ．）

(4) 再びブラウザに戻り，「F5」キーを押して表示を更新し，画像が新たに作成したものに切り替わることを確認せよ．



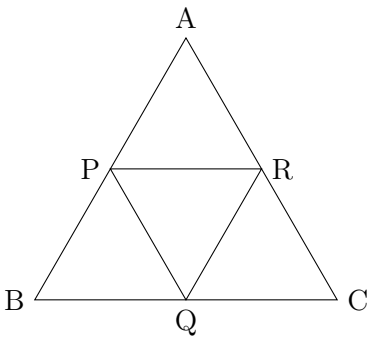
♣ **注意 54** 最後の画像の更新方法は，今日の作業でパラメーターを変えながらいろいろな画像を作って表示するときに役立つ．覚えておくといいだろう．

8.4 シェルピンスキーのガスケットの描き方

それでは，まずはシェルピンスキーのガスケットを実際に描いてみよう．正三角形の 3 頂点 A, B, C の座標を使って，次のような再帰アルゴリズムによって描くことが出来る．

sierpinski(A, B, C):

- AB, BC, CA の中点 P, Q, R の座標を計算する．
- 正三角形 PQR を描く．
- `sierpinski(A,P,R)`, `sierpinski(B,Q,P)`, `sierpinski(C,R,Q)` を呼び出す．



このアルゴリズムは，次々と自分自身を呼び出すので，終わることはない．実際に作るときは再帰の深さに制限を設けて，有限回で止めた図形で近似することになる．

問 33 [提出課題 11] シェルピンスキーのガスケットを描画するための関数を以下のように定義した．このプログラムの続き（実際にこの関数を利用して描画させる部分）を作り，1 辺の長さが 600 の正三角形の内側にシェルピンスキーのガスケットを描け．
（サンプルファイル「`toi34_sierpinski.py`」にヒント付きで下の内容は入力されている．）

```
import turtlesvg as ttl
t = ttl.MyTurtle()

def middle_pt(A, B):
    # AB の中点を返す
    return ( (A[0] + B[0]) / 2, (A[1] + B[1]) / 2 )

def draw_triangle(A, B, C):
    t.penup()
    t.goto(A)
    t.pendown()

    t.goto(B)
    t.goto(C)
    t.goto(A)

def sierpinski(A, B, C, n=5):
    if n > 0:
        P = middle_pt(A, B)
        Q = middle_pt(B, C)
        R = middle_pt(C, A)
        draw_triangle(P, Q, R)

        sierpinski(A, P, R, n-1)
        sierpinski(B, Q, P, n-1)
        sierpinski(C, R, Q, n-1)

B = (-300, -300)
```

♣ **注意 55** タートルの速度やアニメーションの制御に関するコードを入れ忘れると，描画に時間がかかるので注意せよ．

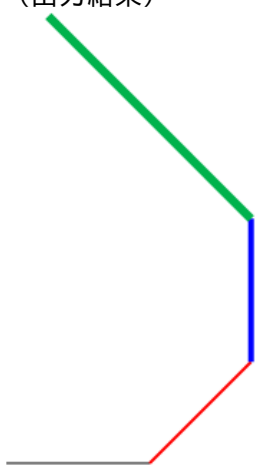
8.5 色の付け方

タートルモジュールには，線の色を変更する機能と，動いて出来た多角形の内部を塗りつぶす機能がある．

例 30

```
1  import turtlesvg as ttl
2  t = ttl.MyTurtle()
3
4  t.fd(100)
5
6  t.left(45)
7  t.pensize(2)
8  t.pencolor('red')
9
10 t.fd(100)
11
12 t.left(45)
13 t.pensize(4)
14 # RGB 値(0〜255)を 16進数で指定
15 t.pencolor('#0000FF')
16
17 t.fd(100)
18
19 t.left(45)
20 t.pensize(6)
21 # RGB 値（0〜1）を順番に指定
22 t.pencolor(0, 0.7, 0.3)
23
24 t.fd(200)
```

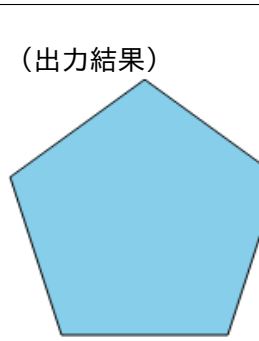
（出力結果）



例 31

```
1  import turtlesvg as ttl
2  t = ttl.MyTurtle()
3
4  t.fillcolor('skyblue')
5  t.begin_fill()
6
7  for i in range(5):
8      t.fd(100)
9      t.left(72)
10
11 t.end_fill()
```

（出力結果）



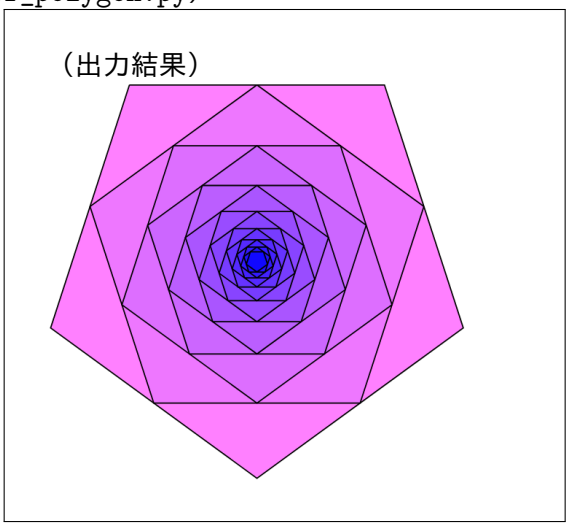
♣ **注意 56** RGB（Red Green Blue）とは，その名の通り「赤」「緑」「青」の割合で色を表す色の表現方法の 1 つである．タートルモジュールでの色の指定に用いることが出来るのは，「0〜255(= FF) の 16 進数」を 3 つ並べた色文字列（`'#FFCC77'` など）と，「0〜1 の float」を 3 つ組にしたタプル（`(0.7, 0.2, 0)` など）の 2 通りである．また，基本的な web カラーの色名（`'black'`, `'red'` など）も利用できる．

再帰的に繰り返し図形を描きながら，色を少しずつ変化させると綺麗な模様になる．

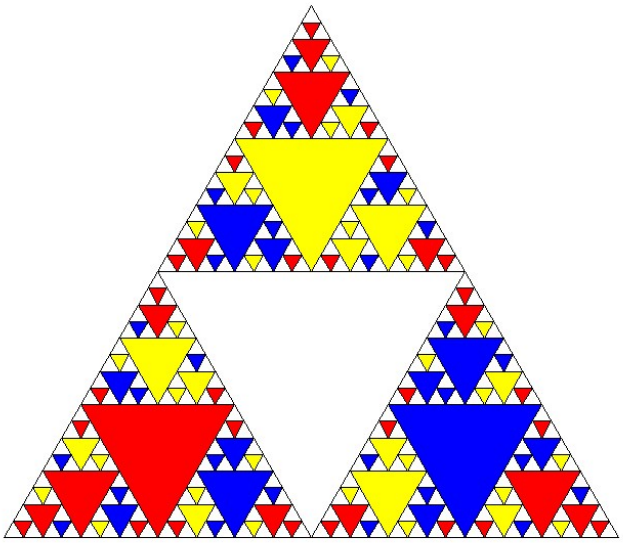
例 32 （サンプルファイル：rei32_color_polygon.py）

```
import turtlesvg as ttl
import math

t = ttl.MyTurtle()

def color_polygon(n, e=200, m=15):
    if m > 0:
        # R：0〜1, G：0〜0.5,B：1（固定）
        t.fillcolor(m/15, m/30, 1)
        t.begin_fill()
        for i in range(n):
            t.fd(e)
            t.rt(360/n)
        t.end_fill()
        t.penup()
        t.fd(e/2)

        # 次の辺の長さは余弦定理
        a = math.radians(360/n)
        c = math.cos(a)
        next_e = e * (2 + 2*c)**0.5 /2
        t.rt(360/n/2)
        t.pendown()

        color_polygon(n, next_e, m-1)

color_polygon(5)
```

（出力結果）



関数 sierpinski() の引数に色を追加して，呼び出しごとに色の種類を指定すると，例えば次のような色分けができる．



線の色・線の太さ・塗りつぶし色の指定や，頂点の作り方などを各自で工夫して面白い図形を描いてみよ．

8.6 一筆書きでシェルピンスキーガスケット

前節までで，「ペンを上げて座標をジャンプしながら三角形を何度も描く」ことでシェルピンスキーのガスケットを描いたが，同じ辺を何度も通ることを許せば，もっと短いプログラムで，ペンを離さずに一筆書きで描くことも出来る．

問 34 下のプログラムは，シェルピンスキーのガスケットを一筆で描くプログラムである．`t.tracer(0)` を使わずに `var_sierpinski(100,4)` などを実行して，タートルの移動する様子を観察して，どのように描かれるかを考察せよ．（サンプルファイル:toi34_var_sierpinski.py）

```
1  import turtlesvg as ttl
2  t = ttl.MyTurtle()
3
4  t.speed(10)
5
6  def var_sierpinski(length, n):
7      if n > 0:
8          for i in range(3):
9              t.fd(length)
10             t.left(120)
11             var_sierpinski(length * 0.5, n-1)
```

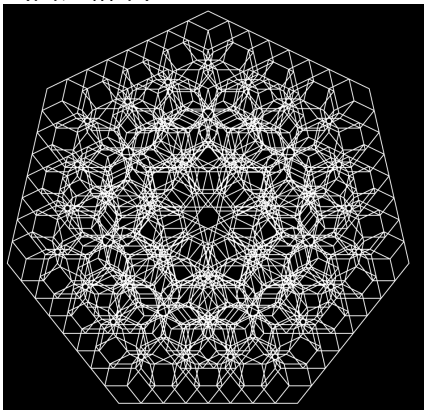
問 35 上のプログラムで，10 行目と 11 行目を入れ替えたら（つまり，タートルの向きを変える操作と，次の `var_sierpinski` を呼び出す操作の順番を逆にしたら），描かれる図形はどのようになるだろうか．2 人で予想を立てた上で，`var_sierpinski(100,5)` などを実行せよ．

♣ **注意 57** このように，ちょっとした工夫を加えると色々な図形が描ける．上の一筆書きシェルピンスキーガスケットは，他にも工夫できそうなところが多いので，k 角形版を作っておこう：（サンプルファイル：chuui57_var_sierpinski_k.py）

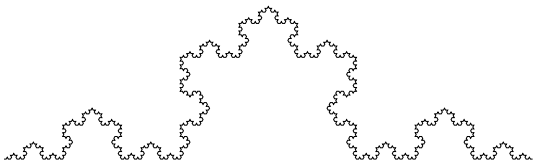
```
1  import turtlesvg as ttl
2  t = ttl.MyTurtle()
3  t.speed(10)
4
5  def var_sierpinski_k(length, k, n, r=0.5):
6      if n > 0:
7          for i in range(k):
8              t.fd(length)
9              t.left(360/k)
10             var_sierpinski_k(length * r,
11                              k, n-1, r)
12
13 t.bgcolor('black')
14 t.pencolor('white')
15 var_sierpinski_k(100, 7, 4, r=0.5)
```

例えば 7 角形でやってみると右の図のようになる．他にも各自で色々アレンジを試してみよ．

（出力結果）



8.7 コッホ曲線

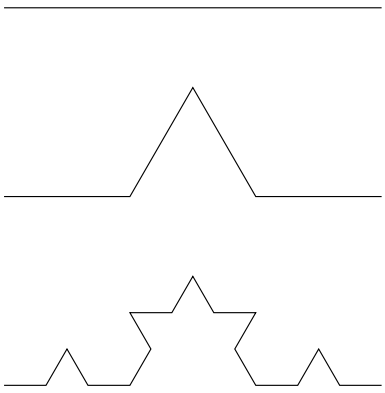


再帰アルゴリズムによるフラクタル図形の描き方が理解できた所で，同様の作り方で他の図形描いてみよう．

右の図形は，**コッホ曲線** (Koch curve) と呼ばれるフラクタル図形である．これは，線分から出発し，「線分を 3 等分し，分割した 2 点を頂点とする正三角形を作る」という操作を繰り返して出来る図形である．従って，初めに線分の両端の 2 点 A, B の座標が与えられたとき，先ほどのシェルピンスキーのガスケットと同じ要領で考えるなら，次のような再帰アルゴリズムによって描くことが出来る．

`koch(n, A, B)`:

- `n == 0` なら，AB を結ぶ．
- `n > 0` のとき：
  - AB の 3 等分点 P, R の座標を計算する．
  - P, R を頂点とする正三角形のもう 1 つの頂点 Q の座標を計算する．
  - `koch(n-1, A, P)`, `koch(n-1, P, Q)`, `koch(n-1, Q, R)`, `koch(n-1, R, B)` を呼び出す．



線分の 3 等分点は容易に求められるが，正三角形の 3 つ目の頂点を直接求めるためには，ベクトルなどを利用して計算する必要がある．

しかし，今の場合は一筆書きで描ける図形なので，座標を直接指定しなければ移動できない `goto()` を利用しなくても，進む向きと距離を指定すれば移動できる `forward()`, `left()`, `right()` などを利用すれば簡単に描ける．すなわち，

`koch(n, l)`:

- `n == 0` なら，距離 `l` だけ移動．
- `n > 0` なら，向きを変更させながら，`koch(n-1, l/3)` を 4 回呼び出す．

これだけで良い．

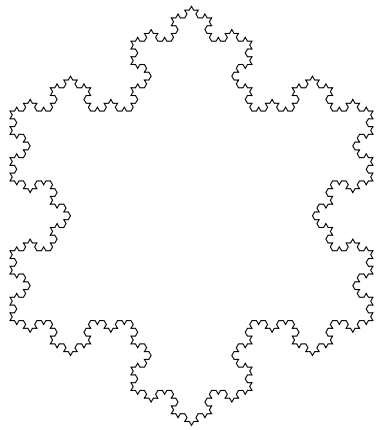
問 36　[提出課題 12] コッホ曲線を描画するプログラムを以下のように作った．空欄を埋めて，プログラムを完成させよ．また，点の取り方を正三角形に限らずに色々変えてみる（二等辺三角形にしてみる，四角形にしてみる，もっと歪んだ形にしてみる）とどのような図形が描けるか．実験してみよ．

```
1   import turtlesvg as ttl
2   t = ttl.MyTurtle()
3
4   def koch(n, l):
5       if n == 0:
6           t.fd(l)
7       else:
8           koch(n-1, l/3)
9           [          ]
10          koch(n-1, l/3)
11          [          ]
12          koch(n-1, l/3)
13          [          ]
14          koch(n-1, l/3)
15
16  t.penup()
17  t.goto(-300,-200)
18  t.pendown()
19
20  t.speed(0)
21  #t.tracer(0)
22
23  koch(4, 600)
24  t.update()
```

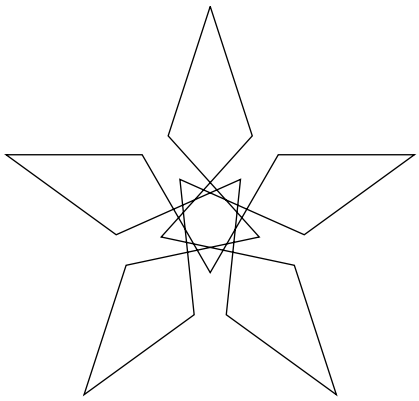
♣ **注意 58**　上のプログラムでは，タートルの動き方を目視するために敢えて`#t.tracer(0)`をコメントアウトしてある．何度も実行したい場合や，再帰の深さを増やしたい場合にはこのコメントを外した方が良い．

問 37　上で作ったコッホ曲線のプログラムを利用して，次のような図形を描け．

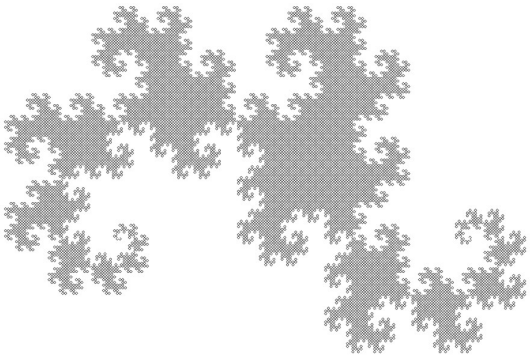
(1)



(2)

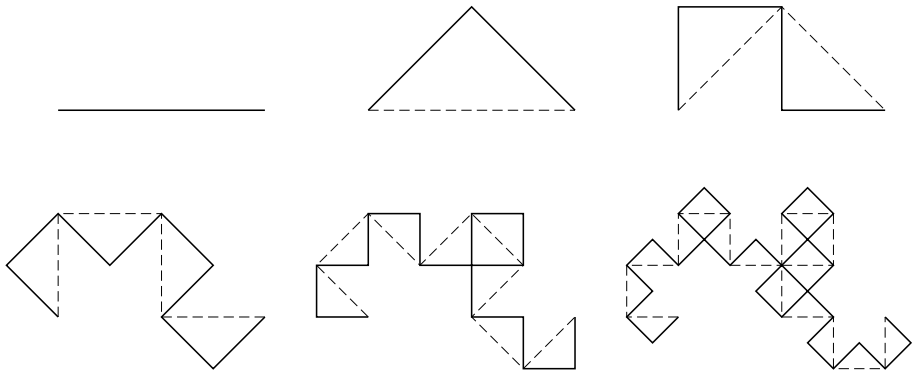


8.8 ドラゴン曲線（時間に余裕がある人は描いてみよ）



右の図形は，**ドラゴン曲線**と呼ばれるフラクタル図形である．これは，コッホ曲線と同様に線分から出発し，その両端を直角でない頂点に持つような直角二等辺三角形を作ることを繰り返して出来る図形である．

コッホ曲線と違う点は，直角三角形を作る向きを交互に逆にしていくことである．生成されていく途中経過を下に示しておこう．



三角形を線分のどちら側に作るかは，条件分岐させてしまうのが簡単だろう．そこで，関数 dragon には，このための符号 sgn を引数に加えることにしよう．アルゴリズムは次のようになる．

dragon(n, l, sgn=1):

- n == 0 なら，l だけ進む．
- n > 0 のとき：
  - sgn == 1 のときは左，右の順に回転．
    sgn == -1 のときは右，左の順に回転．
  - 回転しながら dragon(n-1, l/(2**0.5),1) と dragon(n-1, l/(2**0.5),-1) を呼び出す．
  - 終了する前に，**向きを戻す**．

本質的にはコッホ曲線と同様のプログラムで描くことが出来る．

問 38 以下の空欄を埋めて，ドラゴン曲線を描画するプログラムを完成させよ．

```
1  import turtlesvg as ttl
2  t = ttl.MyTurtle()
3
4  def dragon(n, l, sgn):
5      if n == 0:
6          t.fd(l)
7      else:
8          if [          ]
9              [          ]
10             dragon(n-1, l/(2**0.5), 1)
11             [          ]
12             dragon(n-1, l/(2**0.5), -1)
13             [          ]
14         else:
15             [          ]
16             dragon(n-1, l/(2**0.5), 1)
17             [          ]
18             dragon(n-1, l/(2**0.5), -1)
19             [          ]
20
21
22 t.speed(10)
23 # t.tracer(0)
24
25 t.penup()
26 t.goto(-200,0)
27 t.pendown()
28
29 n = 10
30 dragon(n,500,1)
31 t.update()
```

♣ **注意 59** タートルの状態を取得する関数がいくつかある．ドラゴン曲線やコッホ曲線をアレンジするのに役立つ場合もあるので紹介しておこう：

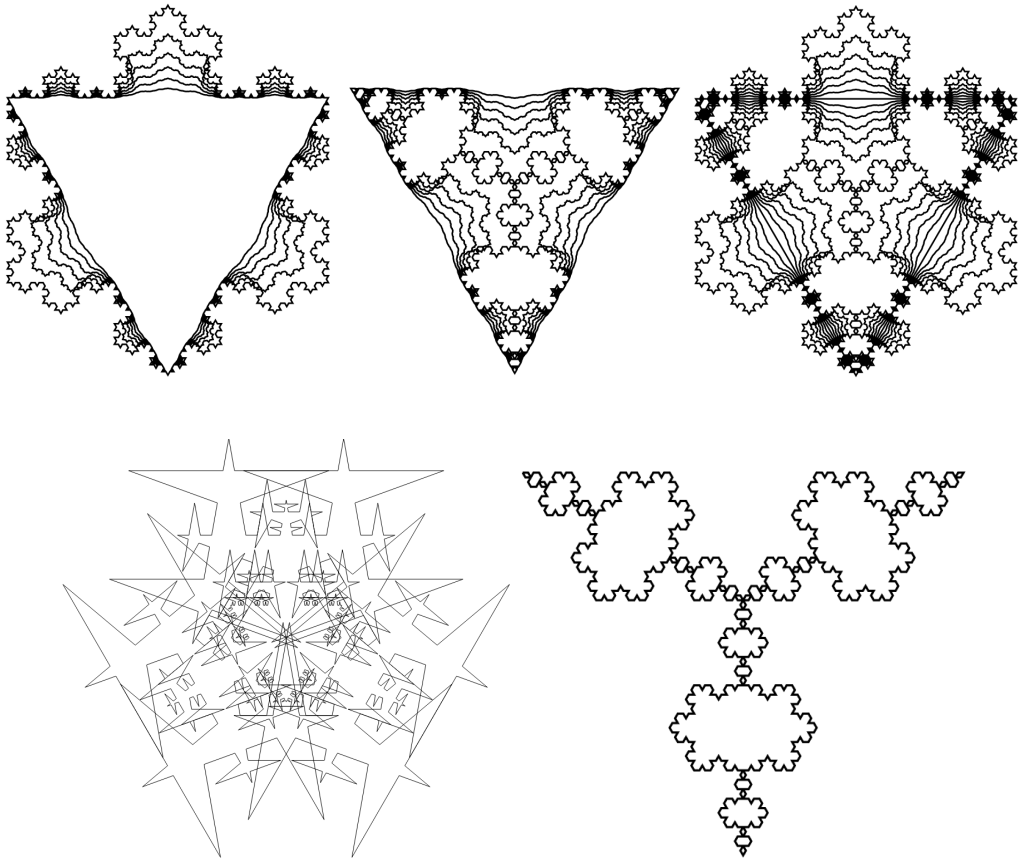
タートルの状態を取得する関数	
`position()`, `pos()`	タートルの現在位置を座標の組（タプル）で返す（正確には，`Vec2D` クラスのベクトルオブジェクトとして返す）．
`xcor()`	タートルの現在位置の x 座標を返す．
`ycor()`	タートルの現在位置の y 座標を返す．
`heading()`	タートルの現在の向き（角度）を返す．
`towards(x, y)`	タートルの現在位置から座標 `(x, y)` へ向かう角度を返す．
`distance(x, y)`	タートルの現在位置から座標 `(x, y)` までの距離を返す．

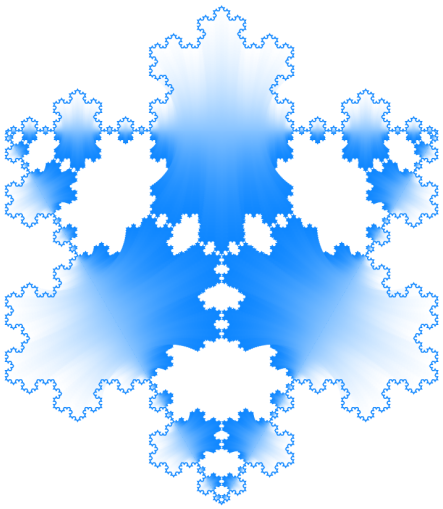
実際，ドラゴン曲線で点の取り方を直角二等辺三角形に限らずに色々変えてみるとどのような図形が描けるかなど，実験してみると良い．

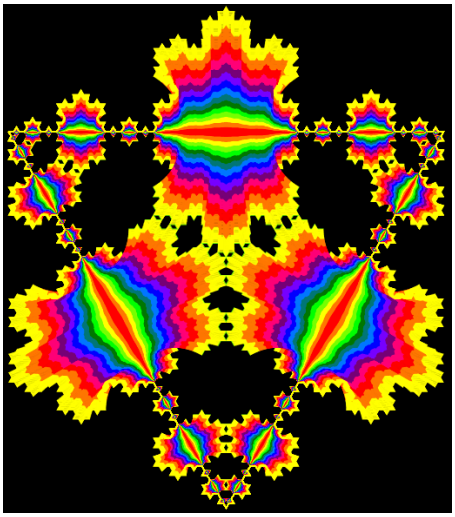
8.9 次回予告

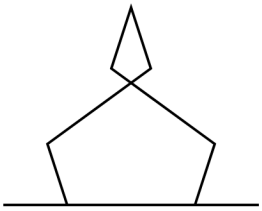
次回は，これまでに扱った図形を参考に（しなくても良いが），各自にフラクタル図形を作って提出してもらう．シェルピンスキーのガスケット，コッホ曲線，ドラゴン曲線などを自由にアレンジし，または各自が調べたその他のフラクタルを活用し，オリジナルの図形を作成する．各自１つ作成するのが理想ではあるが，共同で作りグループで１つでも良いことにする．いくつかの作品は論集に掲載することを考えているので情熱をもって取り組んでほしい．数学１教室のプリンターではカラー印刷はできないが，色を用いた作品も歓迎する．

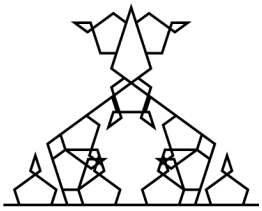
アレンジのヒントになるように，いくつか例を挙げておこう．これらを描くためのコードは来週紹介することにして，ここでは結果の絵だけ挙げておく．どのように考えればこのような絵が出来るのか．次回までに考えておくと作品を作るのに役立つはずである．

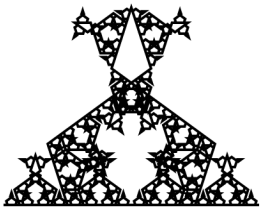












8.10 前回の解答

[提出課題 11]

```
27  # 問 34 答えの部分のみ
28  B = (-300, -300)
29  C = ( 300, -300)
30  A = (0, -300 + 150 * 3**0.5)
31
32  sierpinski(A, B, C, n=5)
33  draw_triangle(A, B, C)
```

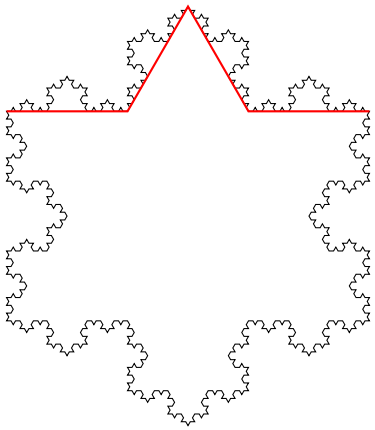
[提出課題 12]

```
1   # 問 37
2
3   import turtlesvg as ttl
4   t = ttl.MyTurtle()
5
6   def koch(n, l):
7       if n == 0:
8           t.fd(l)
9       else:
10          koch(n-1, l/3)
11          t.left(60)
12          koch(n-1, l/3)
13          t.right(120)
14          koch(n-1, l/3)
15          t.left(60)
16          koch(n-1, l/3)
17
18  t.penup()
19  t.goto(-300,-200)
20  t.pendown()
21
22  t.speed(0)
23  #t.tracer(0)
24
25  koch(4, 600)
26  t.update()
```

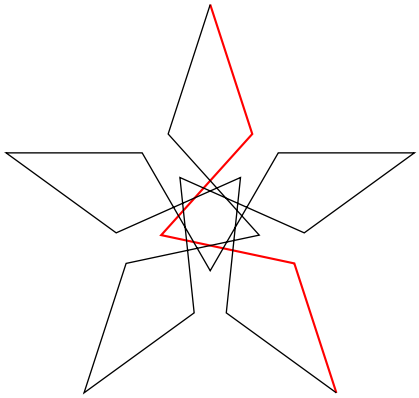
問 37 解答

下のように見れば，コッホ曲線を組み合わせることで描けることが分かるだろう．

(1)



(2)



よって，問 36 で定義した `koch(n,l)` を使って，次のようにすれば良い．

```
# (1)

for i in range(3):
    koch(3, 300)
    t.left(144)

t.update()
```

```
# (2)

t.left(108)
for i in range(5):
    koch(1, 300)
    t.left(144)

t.update()
```

問 38 解答

```
def dragon(n, l, sgn):
    if n == 0:
        t.fd(l)
    else:
        if sgn > 0:
            t.lt(45)
            dragon(n-1, l/(2**0.5), 1)
            t.rt(90)
            dragon(n-1, l/(2**0.5), -1)
            t.lt(45)
        else:
            t.rt(45)
            dragon(n-1, l/(2**0.5), 1)
            t.lt(90)
            dragon(n-1, l/(2**0.5), -1)
            t.rt(45)
```

9 フラクタル作品を作ろう

問 1 [提出課題 最終] シェルピンスキーのガスケット・コッホ曲線・ドラゴン曲線（または他に自由に選んだフラクタル曲線）を組み合わせたり，動かし方を変更したり，思いつく様々なアレンジを適用しながら曲線をプレビューし，気に入った形になったら印刷して提出せよ（turtlesvg の `.save_as_svg(filename)` を利用して SVG ファイルを作り，chrome または FireFox を利用して印刷すること）．その際の Python コードも印刷すること．

♣ **注意 60**

- 数学関数を利用したい場合は，`import math` で math モジュールを読み込んで利用するとよい（`math.cos(math.radians(72))` などの形で数学関数が利用できるようになる）．
- 優秀な作品は来年度の論集に掲載を考えている．そのため，色なども自由に用いて描いて良い．ただし，残念ながら数学 1 教室のプリンターではカラー印刷ができないので，今日提出する印刷物は白黒になる．
- （黒インクを大量に消費してしまうので，完成作品の背景色を黒にしたい場合でも，今日の提出用の印刷物は背景色を白に設定しておいてほしい）

以下では，アレンジのヒントになるように，前回示した例たちのコードを紹介していこう．各コード例では省略するが，冒頭に次の 4 行があるとせよ：

```
1  import turtlesvg as ttl
2  t = ttl.MyTurtle()
3  t.speed(10)
4  t.tracer(0)
```

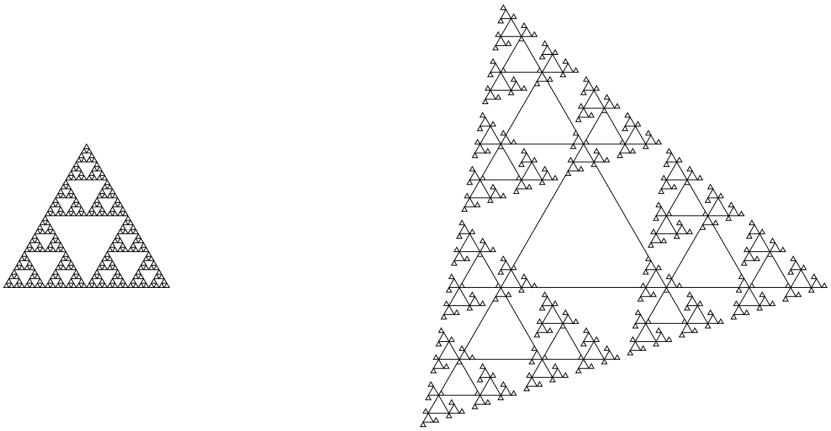
9.1 シェルピンスキーのガスケットのアレンジ

例 33 （問 **35** の解答）タートルの向きを変える操作と再帰呼び出しとの順序を入れ替えると次のようになる：

```
5   def sierpinski1(l, n):
6       if n > 0:
7           for i in range(3):
8               t.fd(l)
9               t.left(120)
10              sierpinski1(l * 0.5, n-1)
11
12  sierpinski1(200, 6)
```

```
5   def sierpinski2(l, n):
6       if n > 0:
7           for i in range(3):
8               t.fd(l)
9               sierpinski2(l * 0.5, n-1)
10              t.left(120)
11
12  sierpinski2(200, 6)
```

（出力結果）

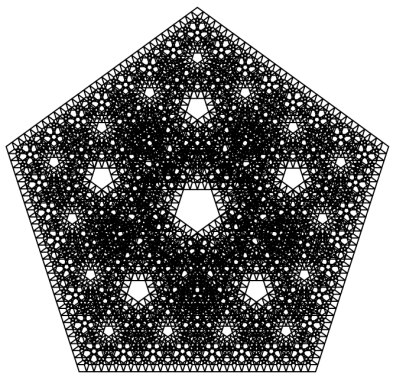


前回作った「k 角形版」でも同じようなアレンジは出来るだろう．また，毎回同じ `sierpinski` を呼び出すのではなく，また，三角形と五角形を組み合わせたり，`sierpinski1` と `sierpinski2` を交互に呼び出したり，など色々なことが考えられそうである．

（k 角形版：再掲）

```
1   def sierpinski_k(length, k, n, r=0.5):
2       if n > 0:
3           for i in range(k):
4               t.fd(length)
5               t.left(360/k)
6               sierpinski_k(length * r,
7                            k, n-1, r)
8
9   sierpinski_k(200, 5, 6, r=0.5)
```

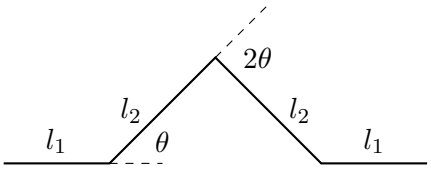
（出力結果）



9.2 コッホ曲線のアレンジ

例 34 コッホ曲線で，線分の長さ l を 3 等分したところに正三角形以外のの形も作れるように，回転する角度を変更しよう．すなわち，

- l_1 進んで，左に θ 回転
- l_2 進んで，右に 2θ 回転
- l_2 進んで，左に θ 回転
- l_1 進む

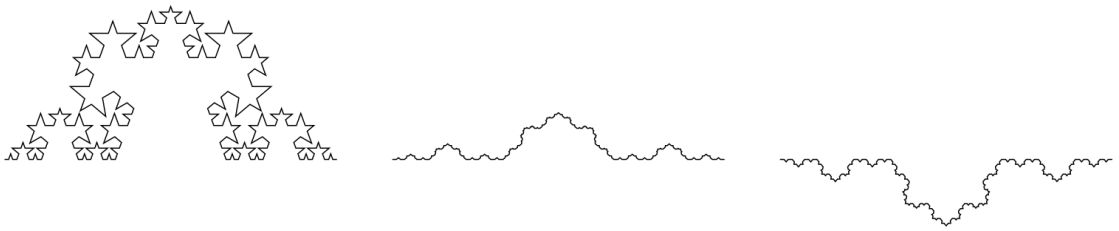


となるように変更しよう．ここでは $l_1 = \frac{l}{3}$ とし，l_2 の長さは三角関数を利用して求める．

```
1   def koch_angle(n, length, theta):
2       if n == 0:
3           t.fd(length)
4       else:
5           l1 = length/3
6           l2 = l1 / (2*math.cos(math.radians(theta)))
7
8           koch_angle(n-1, l1, theta)
9           t.left(theta)
10          koch_angle(n-1, l2, theta)
11          t.right(2*theta)
12          koch_angle(n-1, l2, theta)
13          t.left(theta)
14          koch_angle(n-1, l1, theta)
```

（出力結果 例）

(1) `koch_angle(4,300,70)` (2) `koch_angle(4,300,40)` (3) `koch_angle(4,300,-50)`



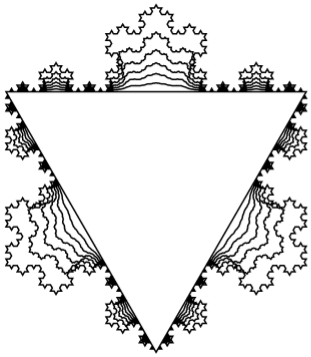
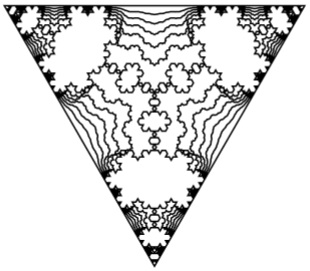
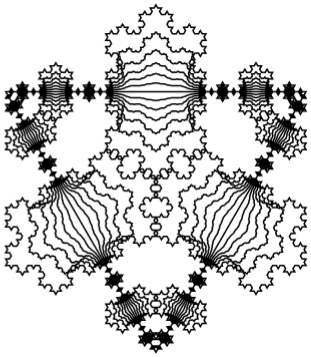
♣ **注意 61** 負の角を指定すれば，回転の向きを逆方向にできる．もちろん，`theta = 60` のときは通常のコッホ曲線を描く．

例 35 `koch_angle` で角度を少しずつ変化させたものを，重ねて描かせてみよう．

```
1   def multiple_koch_island(n=4, length=200,
2                            theta_min=-60, theta_max=60, theta_step=10):
3       for theta in range(theta_min, theta_max + 1, theta_step):
4           for i in range(3):
5               koch_angle(n, length, theta)
6               t.right(120)
```

例として，10° 刻みの実行結果を示しておこう．

- `multiple_koch_island(theta_min=0, theta_max=60, theta_step=10)`
- `multiple_koch_island(theta_min=-60, theta_max=0, theta_step=10)`
- `multiple_koch_island(theta_min=-60, theta_max=60, theta_step=10)`

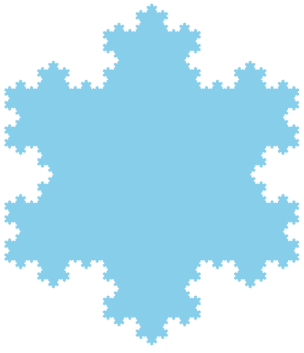
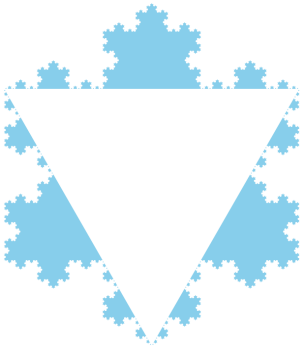




例 36　上の例で，角度の刻みをもっと細かく（1° 刻みなどに）して，線を描く代わりに内部を塗りつぶしすようにして，角度ごとに色を変化させてみるとグラデーションのようなカラフルな図形が描ける．まず，内部の塗りつぶし方を考えておこう．

```
1   def color_test1(n=5, length=400):
2       t.pu()
3
4       t.fillcolor('skyblue')
5       t.begin_fill()
6
7       for i in range(3):
8           koch_angle(n, length, 60)
9           t.right(120)
10
11      t.end_fill()
```

```
1   def color_test2(n=5, length=400):
2       t.pu()
3
4       t.fillcolor('skyblue')
5       t.begin_fill()
6
7       for i in range(3):
8           koch_angle(n, length, 60)
9           t.right(120)
10
11      # 正三角形を描いて図形を閉じる
12      t.right(60)
13      for i in range(3):
14          t.fd(length)
15          t.left(120)
16      t.left(60)
17
18      t.end_fill()
```

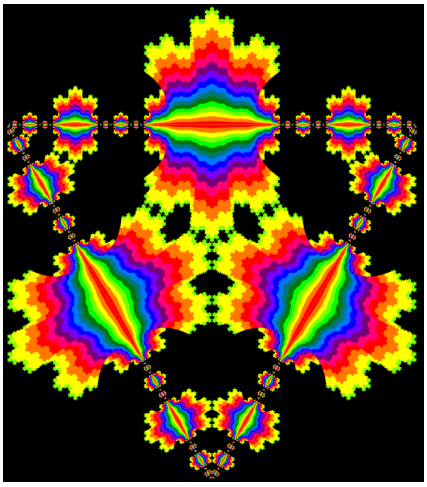
（出力結果）

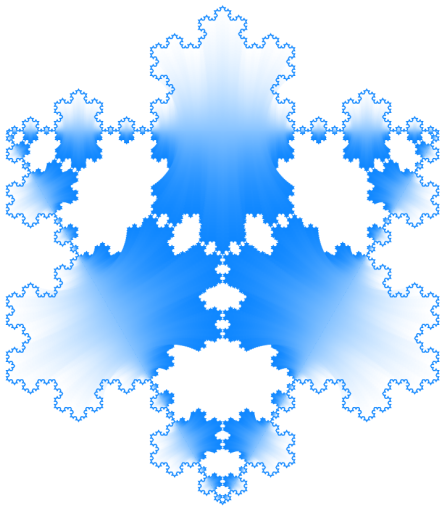
begin_fill(), end_fill() は，その間にタートルが動いてできた図形の「内部」を塗りつぶすので，コッホ曲線を 3 つ並べたあと，最後に逆向きにたどって正三角形を描かせることで，「トゲの中だけ」塗るような塗りつぶし方ができる．

例 37 今の塗りつぶしを，角度の大きい方から順に実行して，徐々に色を変化させてみよう．あらかじめ色のリストを準備しておく方法と，角度に応じて計算した数値で RGB 値を指定する方法を紹介しておく：

```
def multiple_koch_color1(n=5, length=400, theta_max=60, theta_step=1):
    t.pu()

    colors = ["#ff0000", "#ff7700", "#ffff00", "#77ff00", "#00ff00",
              "#007700", "#007777", "#0077ff", "#0000ff", "#7700ff",
              "#770077", "#ff0077"]

    t.bgcolor('black')

    # 外側
    for theta in range(theta_max, 0, -theta_step):
        color = colors[int(abs(theta)/4) % 12]
        t.fillcolor(color)
        t.begin_fill()
        for i in range(3):
            koch_angle(n, length, theta)
            t.right(120)

        t.right(60)
        for i in range(3):
            t.fd(length)
            t.left(120)
        t.left(60)
        t.end_fill()

    # 内側
    for theta in range(-theta_max, 1, theta_step):
        color = colors[int(abs(theta)/4) % 12]
        t.fillcolor(color)
        t.begin_fill()
        for i in range(3):
            koch_angle(n, length, theta)
            t.right(120)

        t.right(60)
        for i in range(3):
            t.fd(length)
            t.left(120)
        t.left(60)
        t.end_fill()
```

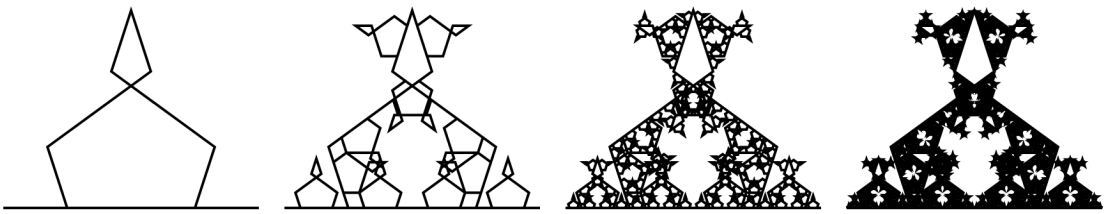


```
def multiple_koch_color2(n=5, length=400, theta_max=60, theta_step=1):
    t.pu()

    # 外側
    for theta in range(theta_max, 0, -theta_step):
        R = (135 + theta*2) / 255
        G = (195 + theta) / 255
        B = 255 / 255
        t.fillcolor(R, G, B)
        t.begin_fill()
        for i in range(3):
            koch_angle(n, length, theta)
            t.right(120)

        t.right(60)
        for i in range(3):
            t.fd(length)
            t.left(120)
        t.left(60)
        t.end_fill()

    # 内側
    for theta in range(-theta_max, 1, theta_step):
        R = (135 + theta*2) / 255
        G = (195 + theta) / 255
        B = 255 / 255
        t.fillcolor(R, G, B)
        t.begin_fill()
        for i in range(3):
            koch_angle(n, length, theta)
            t.right(120)

        t.right(60)
        for i in range(3):
            t.fd(length)
            t.left(120)
        t.left(60)
        t.end_fill()

    # 一番外側の輪郭を描く
    t.pendown()
    R = (135 - 60*2) / 255
    G = (195 - 60) / 255
    B = 255 / 255
    t.pencolor(R, G, B)
    for i in range(3):
        koch_angle(n, length, 60)
        t.right(120)
```



例 38 例 34 では，コッホ曲線で正三角形を作る部分を，「角度を変えた別の三角形」に変更した．これを更に，「三角形とは限らない形」に変更する例も考えておこう．

```
def var_koch(length, n):
    if n == 0:
        t.fd(length)
    else:
        var_koch(length*0.25, n-1)
        var_koch(length*0.25, n-1)
        var_koch(length*0.25, n-1)
        t.left(72)
        var_koch(length*0.25, n-1)
        t.left(72)
        var_koch(length*0.25, n-1)
        var_koch(length*0.25, n-1)
        t.right(72)
        var_koch(length*0.25, n-1)
        t.right(144)
        var_koch(length*0.25, n-1)
        t.right(72)
        var_koch(length*0.25, n-1)
        var_koch(length*0.25, n-1)
        t.left(72)
        var_koch(length*0.25, n-1)
        t.left(72)
        var_koch(length*0.25, n-1)
        var_koch(length*0.25, n-1)
        var_koch(length*0.25, n-1)

for n in range(1, 5):
    var_koch(100, n)
    t.pu()
    t.fd(10)
    t.pd()
```



♣ **注意 62** 上に挙げたものは一例に過ぎない[注 26]．タートルの動かし方の工夫次第で色々な絵が作れるだろう．どのように描かれているのかよく分からない場合は，タートルの動きが見えるように `tracer(0)` を切って，`var_koch(100, 1)` と `var_koch(100, 2)` を実行してみると良いだろう．

[注 26] この変形コッホ曲線は，エッシャーの「天使と悪魔」の絵に通じるものがあるように思える．真ん中の白い部分が，天使のシルエットのように見えてきませんか？

9.3 ドラゴン曲線のアレンジ

例 39 ドラゴン曲線は，コッホ曲線よりアレンジがしづらいが，これも角度の変更をしてみよう．長さ l の線分に対し，通常のドラゴン曲線は角度 45°，長さ $\frac{l}{\sqrt{2}}$ の線分を描く．これを，角度 θ, 長さ $l_1 = a \times l$ のように，θ, a を指定して色々な曲線が描けるように変更したものを挙げておく：

```
import math

def var_dragon(n, l, theta, next_scale, sgn):
    if n == 0:
        t.fd(l)
    else:
        # 現在の向きを記憶
        current_angle = t.heading()
        # 現在の位置を記憶
        current_pos = t.pos()

        t.lt(theta * sgn)
        var_dragon(n-1, l*next_scale, theta, next_scale, 1)

        # 次の位置を計算
        c = math.cos(math.radians(current_angle))
        s = math.sin(math.radians(current_angle))
        next_x = current_pos[0] + l*c
        next_y = current_pos[1] + l*s
        next_angle = t.towards(next_x, next_y)
        next_length = t.distance(next_x, next_y)

        t.setheading(next_angle)
        var_dragon(n-1, next_length, theta, next_scale, -1)

        # 向きを復元
        t.setheading(current_angle)

t.pencolor('white')
t.bgcolor('black')
n = 20 # 20だとわりと時間がかかる
var_dragon(n, 500, 80, 0.37, 1)
```