Try1） 正多角形と角度の研究

Try1） ハンドルを切って車で多角形を描こう。

try01_1.pyを使って、角度を変えながら、何角形になるか表を埋めていってみよう。

	60°	72°	90°	120°
何角形？				

このように試行錯誤して数学的な法則を見つけることを　**実験数学**　と言います。

Try1） ハンドルを切って車で多角形を描こう。

try01_1.py だとセット数分だけ角度を書き換えないといけないので、めんどくさい！
そこで変数というものを使って、書き換えを自動でできるようにソースコードを改良しましょう。それが下のコード **try01_2.py** です。

```
import turtle
#いちいち実験のたびに角度を書き換えるのがめんどくさいので、一か所だけ書き換えれば良いように、改良したソースコード。
#kakudoという変数にいれるだけ。

kakudo = 120
turtle.forward(200)
turtle.left(kakudo)
turtle.forward(200)
turtle.left(kakudo)
turtle.forward(200)
turtle.left(kakudo)
turtle.forward(200)
turtle.left(kakudo)

turtle.done()
```

この部分を書き換えるだけで良くなった。
以下の文中での kakudo は
自動的に 120 に置き換わる。

Try1） ハンドルを切って車で多角形を描こう。

「変数」について説明します。数学では「変数」は「変わる数」のことです。x とか y で書くことが多いですね。ほとんどの人が 1 つの文字を思い浮かべるでしょう。

でも、プログラミングでは「アルファベットで書いたひとカタマリの単語」なら1つの変数になれるのです。また、数学では「 x = 3 」は「 x は 3 と等しい。」という意味でしたが、プログラミングではちょっと違います。「x に 3 という数を入力(セット)する。」という意味です。

変数名(ひとカタマリの単語) ＝ 具体的な数

で、「変数」に数をセットすることができるわけです。

```
import turtle
#いちいち実験のたびに角度を書き換えるのがめんどくさいので、一か所だけ書き換えれば良いように、改良したソースコード。 kakudoという変数にいれるだけ。

kakudo = 120
turtle.forward(200)
turtle.left(kakudo)
turtle.forward(200)
turtle.left(kakudo)
turtle.forward(200)
turtle.left(kakudo)
turtle.forward(200)
turtle.left(kakudo)

turtle.done()
```

kakudo という文字列が
1 つの変数になっており、
kakudo という変数に 120をセットした。
これによって、これ以降の
kakudo はすべて120になって左折する。

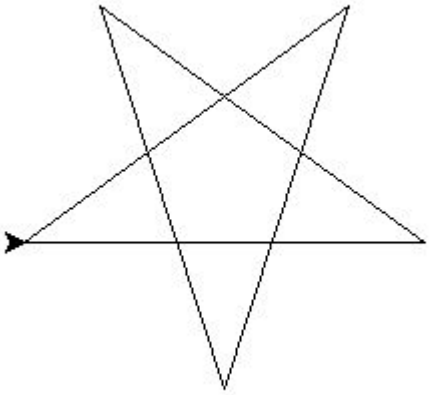
Try1） ハンドルを切って車で多角形を描こう。

kakudo を変更しながら、もっといろいろな形を描画してみよう。でも、「進む＋回転する」のセットが4セットでは面白くないね・・・。でも、同じコードを何回も打つのはイヤだね。そこで役立つのが、**ループ構文** です。ループ構文を使うと何セットでもできます。ひとまず 20 セットにしてみよう。下の **try01_3.py** がその改良版です。

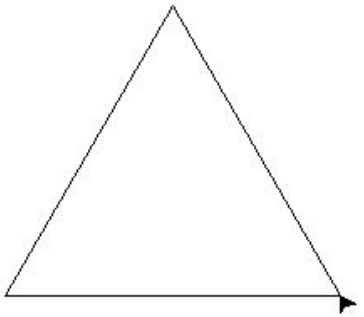
```
import turtle

# kakudo は折れ曲がる角度。いろいろな角度に書き換えてみよう。
kakudo = 144

# このコードではタートルは、20回セット分進む。下の20のところを書き換えてみよう。
for i in range(0,20):
    turtle.forward(200)
    turtle.left(kakudo)
turtle.done()
```

Try1） ハンドルを切って車で多角形を描こう。

kakudo = 144 のとき



kakudo = 120 のとき

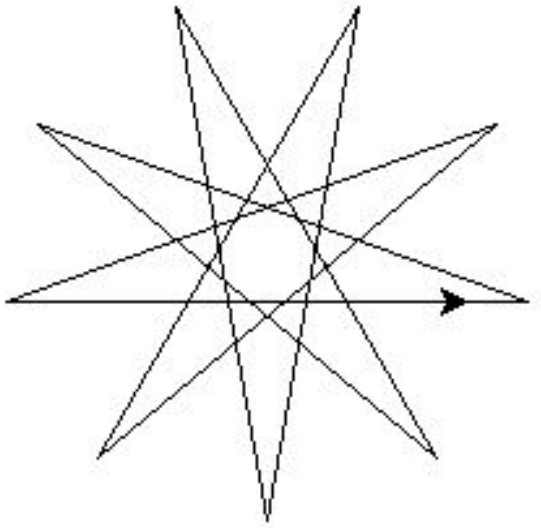


kakudo を変更して、右の図形を描画できるかな？？

try01_3.py のコードを使って試行錯誤してみよう。

英語ではトライアンドエラーと言います。

kakudo = ？？のとき



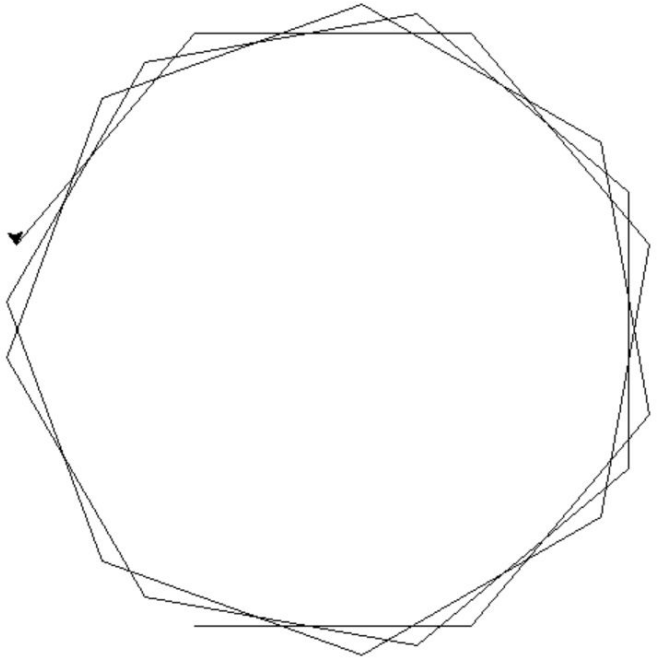
Try1） ハンドルを切って車で多角形を描こう。

kakudo = 50 のときは、下の図形のように、タートルはぐるぐる回って、同じところに戻らない。

range(0,20) のところを range(0,100) にすれば 100 セット動きます。

やってみると、どうなるかな？

kakudo = 50 のとき



Try1） ハンドルを切って車で多角形を描こう。

kakudo やセット数を変更して図形を描画してみたね。そこで、kakudo と図形の関係についてどんなことがわかるか考えてみよう。

- 凸多角形（辺と辺とが交差しない）になるのはどんな時？

- 「正〇〇角形を描画しなさい」といわれたら、kakudo はどうやって設定する？
  例えば正17角形を描画できる？

- kakudo=72 にすると星形になる。とんがっているところは5つだ。
  では、kakudo=50 の場合は、とんがっているところは何個かな？

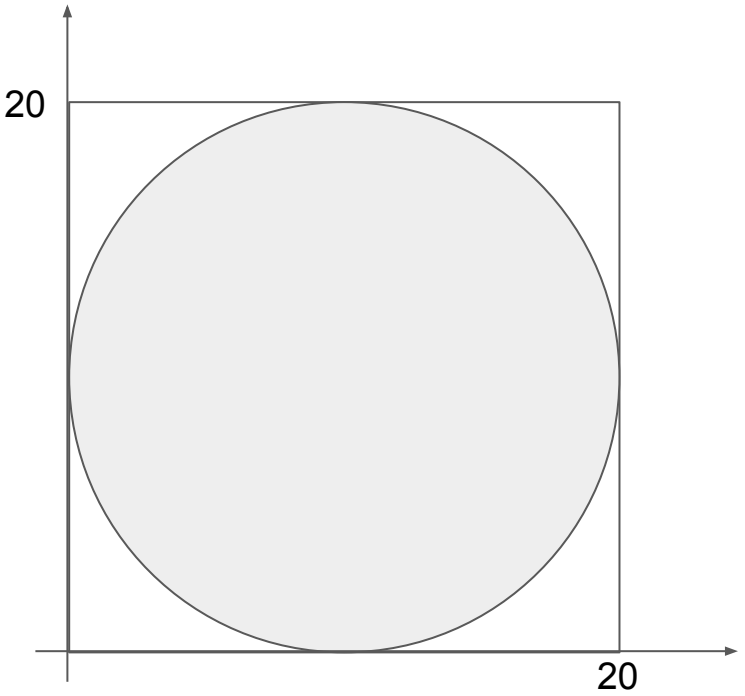
Try2） でたらめなものから、円周率を計算する（モンテカルロ法）

Try2） でたらめなものから円周率を求めよう。

右図を見て下さい。

一辺 20 の正方形の中に、円が内接していますね。

（内側にぴったり接すること。）



π (パイ) を円周率とします。

円の面積は、100×π

正方形の面積は、400

ですね。

ということは、

「正方形に対する円の面積比（つまり、円の面積は正方形の何倍か？ってこと）」は

$$100\pi \div 400 = 0.25\pi = \pi/4$$

です。

Try2） でたらめなものから円周率を求めよう。

さて、この正方形の中にランダムに、点を打つ装置を考えます。

「点が、円の中に入る確率」は「正方形に対する円の面積比」と同じはずです。

だから、この確率を4倍すれば円周率 π になります。

コンピュータは、ランダムで同じことを繰り返すのが得意なのです。

そこで、コンピュータに 1 万個の点を打たせて、この確率を求めてしまおう。

円の中に入る確率は、円の中に入った数÷ 1 万個　だから

円周率 π　≒　4 × 円の中に入った点の数 ÷ 1万

となるはず。

Try2） でたらめなものから円周率を求めよう。

下の **try02.py** を実行して、いろいろやってみよう。

```
import random as rd

in_circle = 0        # 円の中に入った点の数
point_all = 10000    # 点の個数（好きな数字に書き換えよう。）

for i in range(0, point_all):
    x = rd.uniform(0,20)     # 0≦ x ≦20の範囲の点の座標をランダムに作る。
    y = rd.uniform(0,20)     # 0≦ y ≦20の範囲の点の座標をランダムに作る。

    if (x-10)**2+(y-10)**2 < 100:   # 円の中に入る判定。この式は高校数学で学ぶよ。
        in_circle += 1

print(f"円周率は {4*in_circle/point_all} です。")

#本当の円周率は π = 3.1415926535…
```

Try2） でたらめなものから円周率を求めよう。

10回シミュレーションしてみて、

円周率を導き出してみよう。

	4×円の中に入った点の個数÷全部の点の数
1 回目	
2 回目	
3 回目	
4 回目	
5 回目	
6 回目	
7 回目	
8 回目	
9 回目	
10 回目	
平均	

このように、ランダム（でたらめ）に点を打つことによって、必要なものを確率的に計算する方法を **モンテカルロ法** といいます。

平均の計算はPythonでやってもいいぞ！

```
print(f'{sum([3.1332,3.1364,3.1472,…])/10}')
```

ここに10個の結果をカンマで書く