Python 小技之繁花曲线

原创 轩辕御龙 Python技术 3月3日

前几天逛朋友圈的时候，无意间刷到同学这样一条内容：

朋友圈截图

不知道大家有没有眼熟的感觉，反正是勾起了我不少回忆。

这种叫做“万花尺”的小玩意儿小时候应该不少人都玩过。一个大圆套一个小圆，圆与圆之间通过齿轮啮合在一起。

只需选中一个点，拿一支笔随着圆移动，就可以画出各种复杂的曲线，不同的曲线又可以进一步呈现出奇妙的图形。并且换用不同颜色的笔芯还可以使得图形更加丰富多彩（如上图所示）。这种图形还有一个更加文艺、好听的名字叫“繁花曲线”。

正好最近学到了 Python 的 turtle 模块，对于画这样的圆啊、曲线啊什么的再合适不过了。要用 Python 构建一个类似的图形，我们首先得要考察一下我们到底要画什么。

通过观察图形（当然也可以是通过观察万花尺的结构），我们可以很容易地发现：不论图形怎样变化，最终得到的图形大体上总是一个圆。换句话说，整个图形的大体框架就是一整个圆，其他的各种曲线都是在此基础上进一步曲折变换来的，因此我们第一步先要画一个圆。

turtle 画圆

仅仅是想用 turtle 模块来画圆的话很简单。

首先导入模块：

```
import turtle
```

生成画笔的实例，并调用已有的 `circle` 方法：

```
pen = turtle.Turtle()
pen.circle(100)
```

画圆

这里方法的参数 100 指定的是所画圆的半径。此外该方法还有 `extent` 和 `steps` 两个参数，前者指定绘制圆的角度（单

位为角度），后者我们放在后面来介绍。

我们还可以用这个方法来画一个太极图：

```
# pen = turtle.Turtle()
pen.clear()
pen.up() # 将画笔从画布上提起。即在画笔移动过程中不画出笔迹
pen.setpos(0, -100) # 将画笔移动到画布的这个位置
pen.down() # 将画笔放到画布上。即之后画笔的移动都会留下笔迹

# 画阴鱼
pen.fillcolor("black")
pen.begin_fill()
pen.circle(100, 180)
pen.end_fill()

# pen.fillcolor("black")
pen.begin_fill()
pen.circle(50, 180)
pen.end_fill()

pen.fillcolor("white")
pen.begin_fill()
pen.circle(-50, 180)
pen.end_fill()

# 画阳鱼
pen.circle(-100, 180)

# 画阳眼
pen.up()
pen.setpos(0, 50-15)
pen.down()
# pen.fillcolor("white")
pen.begin_fill()
pen.circle(15)
pen.end_fill()

# 画阴眼
pen.up()
pen.setpos(0, -50-15)
pen.down()
pen.fillcolor("black")
pen.begin_fill()
pen.circle(15)
pen.end_fill()
```

虽然太极图确实画出来了，但是可以发现，太极图本身结构并不复杂，而我们的代码需要把画笔多次抬起、放下，反复调用 `circle` 方法，十分繁琐。同时最致命的一个问题是，调用 `circle` 方法画出的图形一定是圆周或部分圆周，但我们真正要画的万花尺图形却并非都是由圆周曲线组成的，占大头的多是各种椭圆线、螺旋线。

所以到这里我们就遇到了一个问题：调用 `circle` 方法，方便确实是方便，但真要想画出一条复杂的曲线， `circle` 方法就无法给我们提供想要的灵活和自由度。

我们需要思考灵活度更高的绘图方法。

利用方程画图

还是画圆

上一小节我们讲到 `circle` 方法还有一个参数 `steps` ，但留了一个悬念没有说明参数的用途。

顾名思义， `steps` 就是指“画圆的过程分为几步”。

实际上， `circle` 方法并不是真的画出了一个完美的“圆”，而仅仅是使用多边形模拟的一个“近似的圆”，就像当年祖冲之计算圆周率用的方法一样。

知道了这一点，接下来就好办了。我们想画出一个圆也可以用这种方法，只要把圆周上的很多个点用线段连起来即可。

但是关键是首先要找出圆周上的若干个点。

我们可以回忆一下中学时代圆锥曲线的内容：圆的 x、y 两个坐标可以通过关于半径 r 和角度 θ 的两个参数方程分别确定。

用公式表达圆心在原点上的圆周坐标为：

$$\begin{cases} x = r \times \cos(\theta) \\ y = r \times \sin(\theta) \end{cases}$$

若圆心坐标为 (a, b)，则公式表达应为：

先导入要用到的模块 `math` ：

```
import math
```

这个模块中写好了 `cos` 和 `sin` 的函数实现，我们直接拿来用就好。

上述第二组表达式写成函数就是：

```
# 计算圆周的 x、y 坐标
def cor_x_y(r, theta, a=0, b=0,):
    # 把角度表示的 theta 转换为弧度表示。
    # 因为 math 中的 cos 和 sin 都要求输入为弧度
    rad = degreeToRadian(theta) # 该函数稍后实现

    x = r * math.cos(rad) + a
    y = r * math.sin(rad) + b

    return (x, y)
```

其中用到了一个将角度转换为弧度的函数，实现起来也很简单：

```
def degreeToRadian(degree):
    return degree * math.pi / 180
```

调用函数来画图试试：

```
pen.clear()

# 遍历圆周上的 180 个点
for i in range(0,362,2):
    if i != 0:
        pen.setpos(*cor_x_y(100, i))
    else:
        pen.up()
        pen.setpos(*cor_x_y(100, i))
        pen.down()
```

用多边形近似画圆

完美！完全看不出跟真正圆的区别嘛哈哈~

画一条曲线

通过上面“以方画圆”的测试，我们可以得知用 `turtle` 模块描出轮廓点的方式可以非常好地模拟出圆形，自然而然别的曲线也不例外。

为了尽量减少本文中出现公式的频率，此处删减约一千字推导过程，于是我们得到了繁花曲线的坐标公式（来自

Wikipedia，“Spirograph”词条）：

繁花曲线数学原理

其中，R 为大圆半径；r 为小圆半径；k 为小圆半径与大圆半径之比，即 r/R；ρ 为画笔到小圆圆心的距离；l（注意是小写字母L）为画笔到小圆圆新的距离与小圆半径之比，即 ρ/r；t 即对应坐标相对圆心的弧度。显然，k 和 l 都应该是介于 0 和 1 之间的实数。

上述公式实现如下：

```
# 用 d 表示画笔到小圆圆心的距离
def cor_x_y_Spiro(R, r, l, theta):
    k = r/R
    ef = 1 - k

    rad = degreeToRadian(theta)
    x = R*(ef*math.cos(rad) + l*k*math.cos(ef/k*rad))
    y = R*(ef*math.sin(rad) - l*k*math.sin(ef/k*rad))

    return (x, y)
```

曲线重复的周期可以这样确定：将 k 化为最简分数，此时分子的大小 n 即为曲线的周期。也就是说我们只需遍历 n 个圆周即可画出闭合的曲线。

```
for i in range(0,360*3+2,2):
    if i != 0:
        pen.setpos(*cor_x_y_Spiro(100, 30, 0.6, i))
    else:
        pen.up()
        pen.setpos(*cor_x_y_Spiro(100, 30, 0.6, i))
        pen.down()
```

为了增加程序的灵活性，我们还需要实现一个函数用以求得这个周期数。而首先我们应当实现一个函数来求大圆半径和小圆半径的最大公约数（即 highest common factor，hcf）。这里我们用的是欧几里得算法，又称“辗转相除法”：

```
def hcf(x, y):
    if x == y:
        result = x
    elif x > y:
```

```
        result = hcf(x-y, y)
    else:
        result = hcf(x, y-x)

    return result

```

用小圆半径除以该最大公约数，即可得到周期数：

```
def periods(R, r):
    div = hcf(R, r)

    return r//div
```

让我们把上面的代码封装一下：

```
import turtle
import math

class Spiro:
    def __init__(self, R, r, l):
        self.R = R
        self.r = r
        self.l = l
        self.pen = turtle.Turtle()


    def drawSingleSpiro(self):
        # 周期数 p
        p = self.periods()

        for i in range(0, 360*p + 2, 2):
            if i != 0:
                self.pen.setpos(*self.cor_x_y_Spiro(i))
            else:
                self.pen.up()
                self.pen.setpos(*self.cor_x_y_Spiro(i))
                self.pen.down()


    def hcf(self, x, y):
        if x == y:
            result = x
        elif x > y:
            result = self.hcf(x-y, y)
        else:
            result = self.hcf(x, y-x)
        return result
```

```
        return result


    def periods(self):
        div = self.hcf(self.R, self.r)
        return self.r//div


    def cor_x_y_Spiro(self, theta):
        k = self.r/self.R
        ef = 1 - k

        rad = self.degreeToRadian(theta)
        x = self.R*(ef*math.cos(rad) + self.l*k*math.cos(ef/k*rad))
        y = self.R*(ef*math.sin(rad) - self.l*k*math.sin(ef/k*rad))

        return (x, y)


    def degreeToRadian(self, degree):
        return degree * math.pi / 180

s = Spiro(100, 30, 0.6)
s.drawSingleSpiro()
turtle.mainloop()
```

两相切圆

这样就可以画出一条完整的曲线了。

完整实现

修改一下 `drawSingleSpiro` 方法的接口，增加一个参数 `pencolor` 来指定单条曲线的画笔颜色。同时去掉原 `Spiro` 类中的 `l` 属性——根据我们使用万花尺的经验，这一个参数应当是可变的——改为在某个随机处理函数中指定。

当然考虑到修改代码的方便性，本次修改仅仅是在随机处理函数中对 `l` 属性重新赋值。

最后，应当画多少条完整曲线也应由用户在初始化实例时自由指定。

代码如下：

```
import turtle
import math
import random

class Spiro:
```

```
    def __init__(self, R, r, l, num, color):
        self.R = R
        self.r = r
        self.l = l
        self.num = num
        self.pen = turtle.Turtle()
        self.pen.pencolor(color)
        turtle.colormode(1.0)


    def drawSingleSpiro(self):
        # 周期数 p
        p = self.periods()

        for i in range(0, 360*p + 2, 2):
            if i != 0:
                self.pen.setpos(*self.cor_x_y_Spiro(i))
            else:
                self.pen.up()
                self.pen.setpos(*self.cor_x_y_Spiro(i))
                self.pen.down()


    def drawWhole(self):
        for s in range(self.num):
            if s != 0:
                self.randomSetting()
                self.drawSingleSpiro()
            else:
                self.drawSingleSpiro()


    def randomSetting(self):
        self.l = random.random()

        r = random.random()
        g = random.random()
        b = random.random()
        self.pen.pencolor((r, g, b))


    def hcf(self, x, y):
        if x == y:
            result = x
        elif x > y:
            result = self.hcf(x-y, y)
        else:
            result = self.hcf(x, y-x)
        return result

```

```
    def periods(self):
        div = self.hcf(self.R, self.r)
        return self.r//div


    def cor_x_y_Spiro(self, theta):
        k = self.r/self.R
        ef = 1 - k

        rad = self.degreeToRadian(theta)
        x = self.R*(ef*math.cos(rad) + self.l*k*math.cos(ef/k*rad))
        y = self.R*(ef*math.sin(rad) - self.l*k*math.sin(ef/k*rad))

        return (x, y)


    def degreeToRadian(self, degree):
        return degree * math.pi / 180

s = Spiro(100, 30, 0.6, 5, "pink")
s.drawWhole()
turtle.mainloop()
```

完整的繁花曲线图

小 tips

大圆半径与小圆半径尽量互质，得到的图形会更加复杂精巧哦~

> 示例代码：https://github.com/JustDoPython/python-100-day/tree/master/Spiro

参考资料

https://en.wikipedia.org/wiki/Spirograph

PS: 公号内回复「Python」即可进入 Python 新手学习交流群，一起**100天计划！**

-END-