

Python艺术编程05

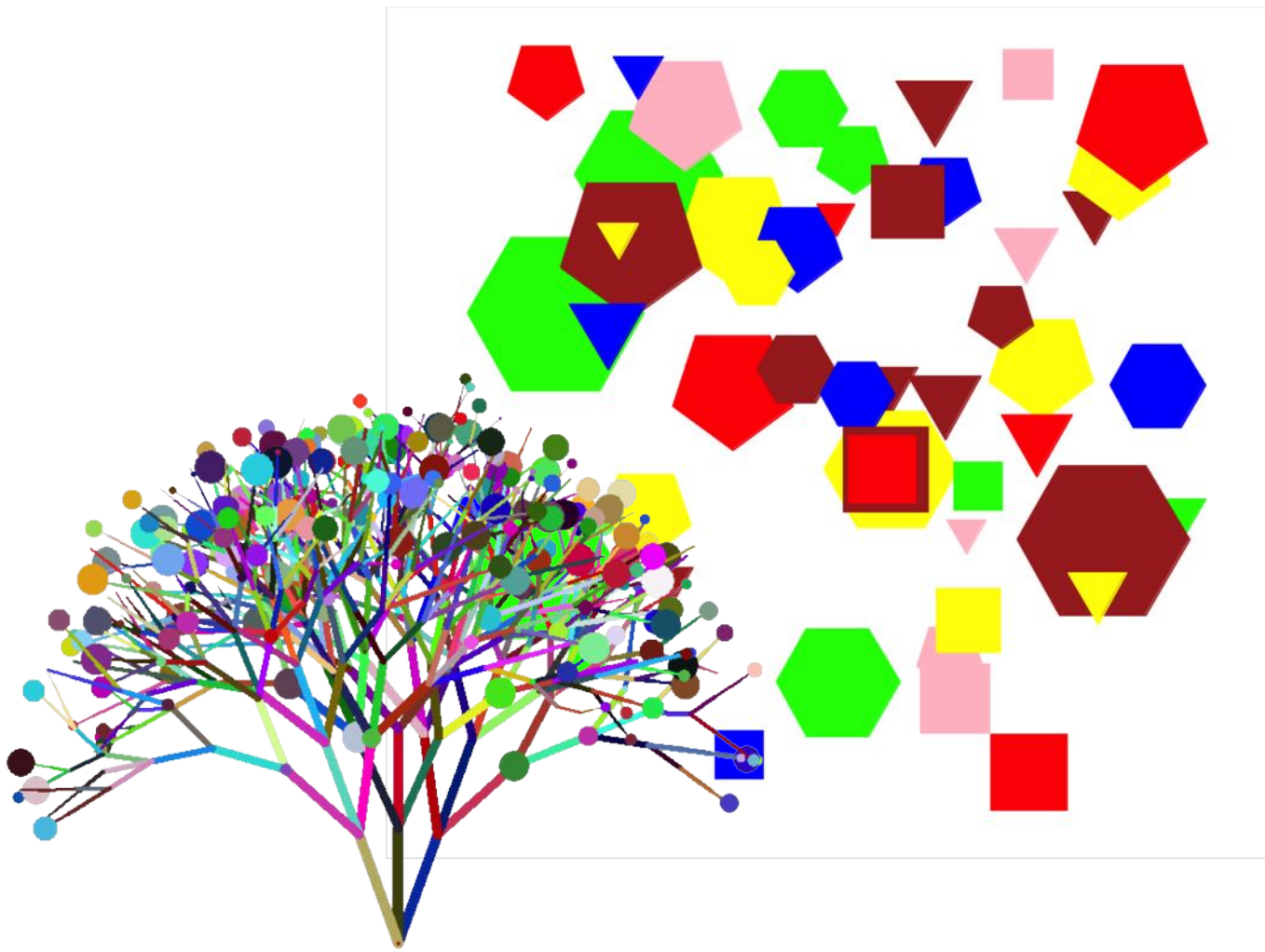
——函数定义和分形树

北京大学 陈斌

2018.10.08

目录

- 函数的基本概念
- 组合图形
- 上机练习
- 递归的概念
- 递归函数
- 绘制简单二叉树

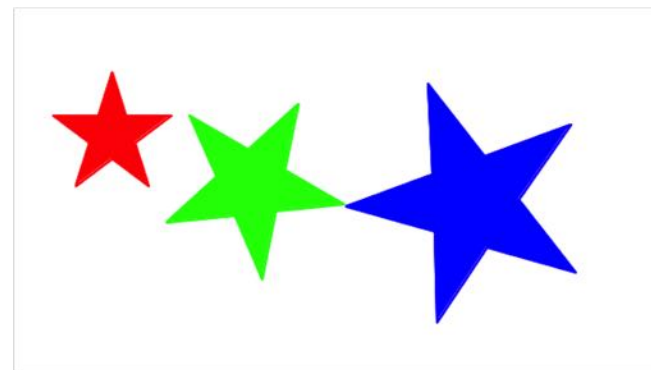


代码复用情境

- 有时候我们需要反复使用某些代码
 - 比如组合图形出现多个五角星
- 如果到处拷贝这些代码，会出现弊端
 - 程序变得冗长，可读性差
 - 一旦需要修改或扩充，要在各处同步改代码，容易出错，可维护性差

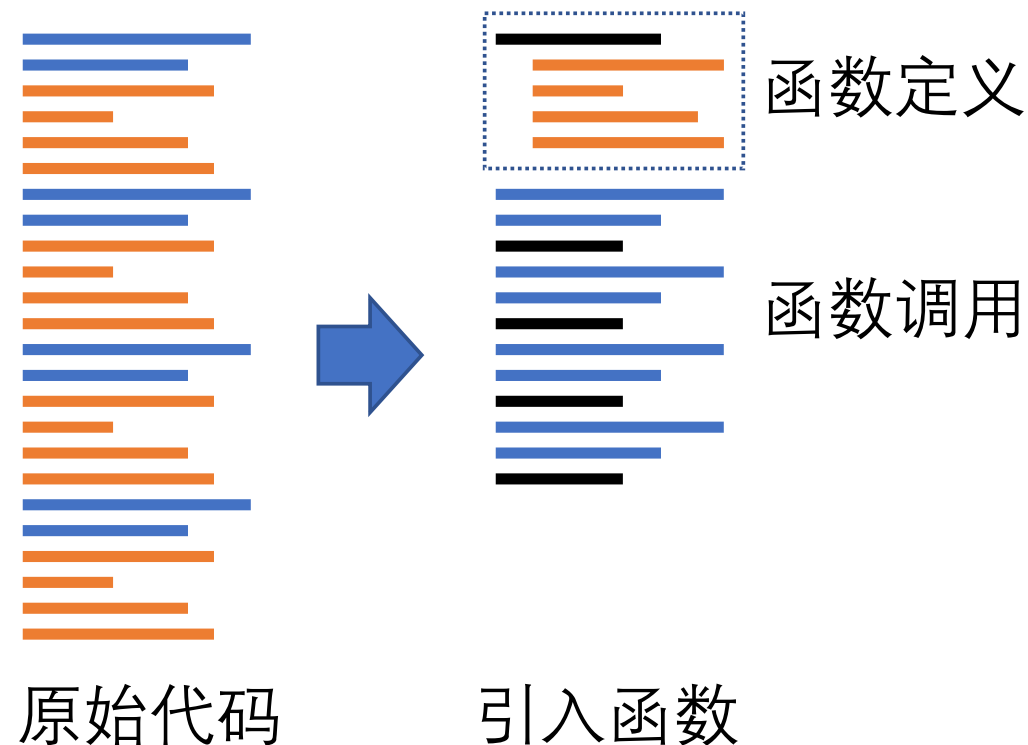
```
1  import turtle
2
3  t = turtle.Turtle()
4
5  t.color('red')
6  t.begin_fill()
7  for i in range(5):
8      t.forward(20)
9      t.left(72)
10     t.forward(20)
11     t.right(144)
12 t.end_fill()
13
14 t.penup()
15 t.forward(60)
16 t.right(30)
17 t.pendown()
18
19 t.color('green')
20 t.begin_fill()
21 for i in range(5):
22     t.forward(30)
23     t.left(72)
24     t.forward(30)
25     t.right(144)
26 t.end_fill()
27
28 t.penup()
29 t.forward(80)
30 t.left(50)
31 t.pendown()
32
33 t.color('blue')
34 t.begin_fill()
35 for i in range(5):
36     t.forward(40)
37     t.left(72)
38     t.forward(40)
39     t.right(144)
40 t.end_fill()
41
42 t.hideturtle()
43 turtle.done()
```

func_star.py



解决方案：函数（functions）

- 我们把这些重复代码单独收集起来，组成一个“函数”对象，并赋予一个名称
- 在需要用到这些代码的时候就通过名称来“呼叫”这些“函数”
- 前者称为函数定义（define）
- 后者称为函数调用（call）



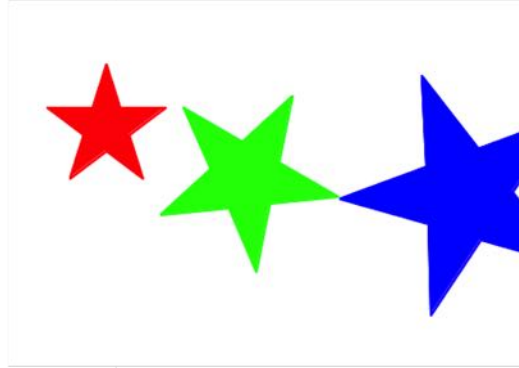
定义函数：def语句

- 函数定义语句def
 - `def` <函数名称>(<参数表>):
 - <语句块>
 - [`return` <返回值>]
- 几个要素
 - `def`关键字
 - 函数名称，后跟一对圆括号
 - (可选的) 参数表
 - 语句块
 - (可选的) 返回值

func_star2.py

```
1 import turtle
2
3
4 def star(size, color):
5     t.color(color)
6     t.begin_fill()
7     for i in range(5):
8         t.forward(size)
9         t.left(72)
10        t.forward(size)
11        t.right(144)
12    t.end_fill()
13
14
15 t = turtle.Turtle()
16
17 star(20, 'red')
18 t.penup()
19 t.forward(60)
20 t.right(30)
21 t.pendown()
22 star(30, 'green')
23 t.penup()
24 t.forward(80)
25 t.left(50)
26 t.pendown()
27 star(40, 'blue')
28
29 t.hideturtle()
30 turtle.done()
```

函数定义



函数的参数

- 如果代码块里没有可供调节的选项，可以定义没有参数的简单函数
- 一般函数会带有可供调节的参数，参数可以有多个
 - 如画五角星的函数，包含两个参数：大小size和颜色color

func_star2.py

```
4  def star(size, color):  
5      t.color(color)  
6      t.begin_fill()  
7      for i in range(5):  
8          t.forward(size)  
9          t.left(72)  
10         t.forward(size)  
11         t.right(144)  
12     t.end_fill()
```

函数的返回值

- 有时候函数会有返回值
 - 如math模块中求平方根的函数`math.sqrt(n)`返回n的平方根
- `return`语句负责结束函数执行，并返回值
- `return`语句可以根据需要，出现在语句块中的任何位置

```
1  import math
2
3  s = math.sqrt(120)
4  print(s)
5
6
7  def my_abs(n):
8      if n >= 0:
9          return n
10     else:
11         return -n
12
13
14 s = my_abs(-10)
15 print(s)
```

函数定义中的代码块

- 由于函数定义def语句仅仅是把代码块“打包封装”
- def语句执行的时候，代码块并不会被执行
- 所以，在执行def语句的时候，除非语句块中包含了明显的语法错误
- Python解释器是不会检查语句块中其它错误的。

func_star2.py

```
1  import turtle
2
3
4  def star(size, color):
5      t.color(color)
6      t.begin_fill()
7      for i in range(5):
8          t.forward(size)
9          t.left(72)
10         t.forward(size)
11         t.right(144)
12     t.end_fill()
13
14
15 t = turtle.Turtle()
16
17 star(20, 'red')
```

并不会出现
“t未定义”
的错误

调用函数： call function

- def定义了函数之后，函数名称仅代表这个“函数对象”
- 如果需要执行语句块代码，需要有如下的要素
 - 函数名称，后加括号
 - 括号内放置参数的具体值
- 没有或者不需要返回值
 - func(a,b,c) #如调用star
- 获取返回值
 - v = func(a,b,c)

func_call.py

```
1 def my_abs(n):
2     if n >= 0:
3         return n
4     else:
5         return -n
6
7
8 # 函数对象
9 s = my_abs
10 print("function object:", s)
11
12 # 加括号和参数, 调用函数
13 s = my_abs(-10)
14 print("call function:", s)
```

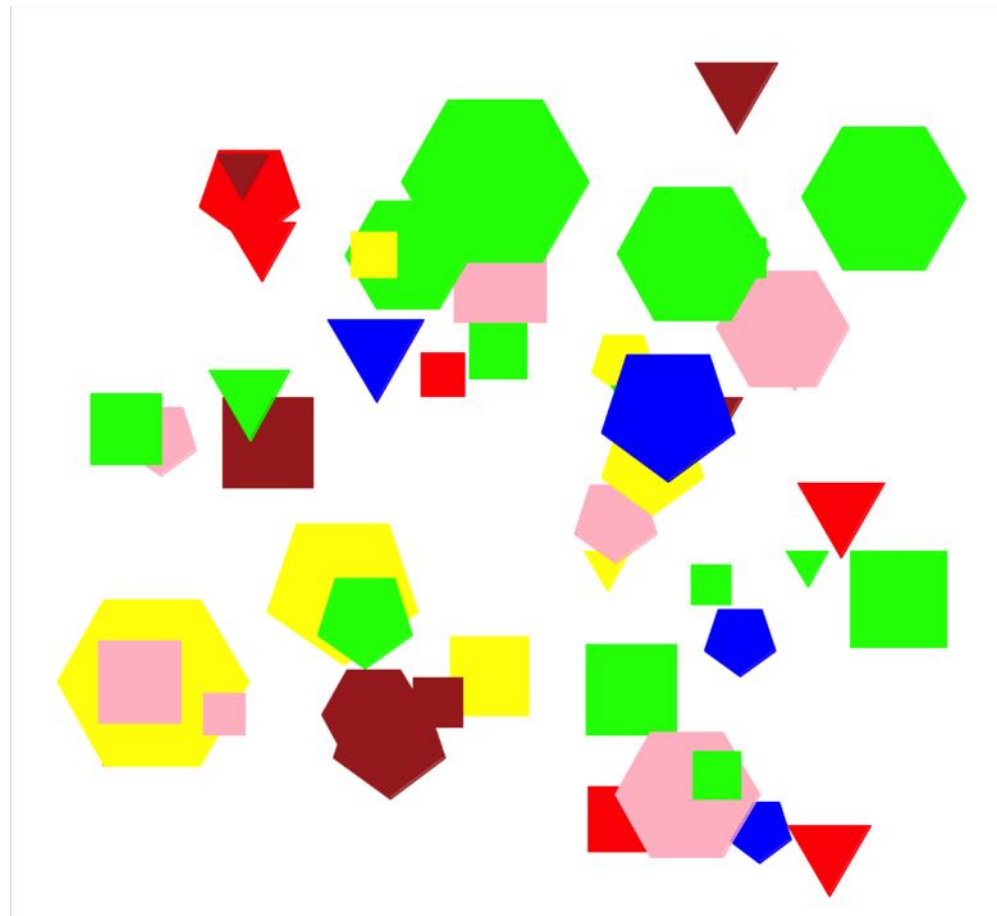
随机数模块random

- 产生一定范围内的随机数
 - `random.randint(min,max)`
- 从列表中随机选择
 - `random.choice(list)`

```
>>> import random
>>> colors=['red', 'green', 'blue', 'brown', 'pink']
>>> c=random.choice(colors)
>>> c
'green'
>>> n=random.randint(10,20)
>>> n
19
```

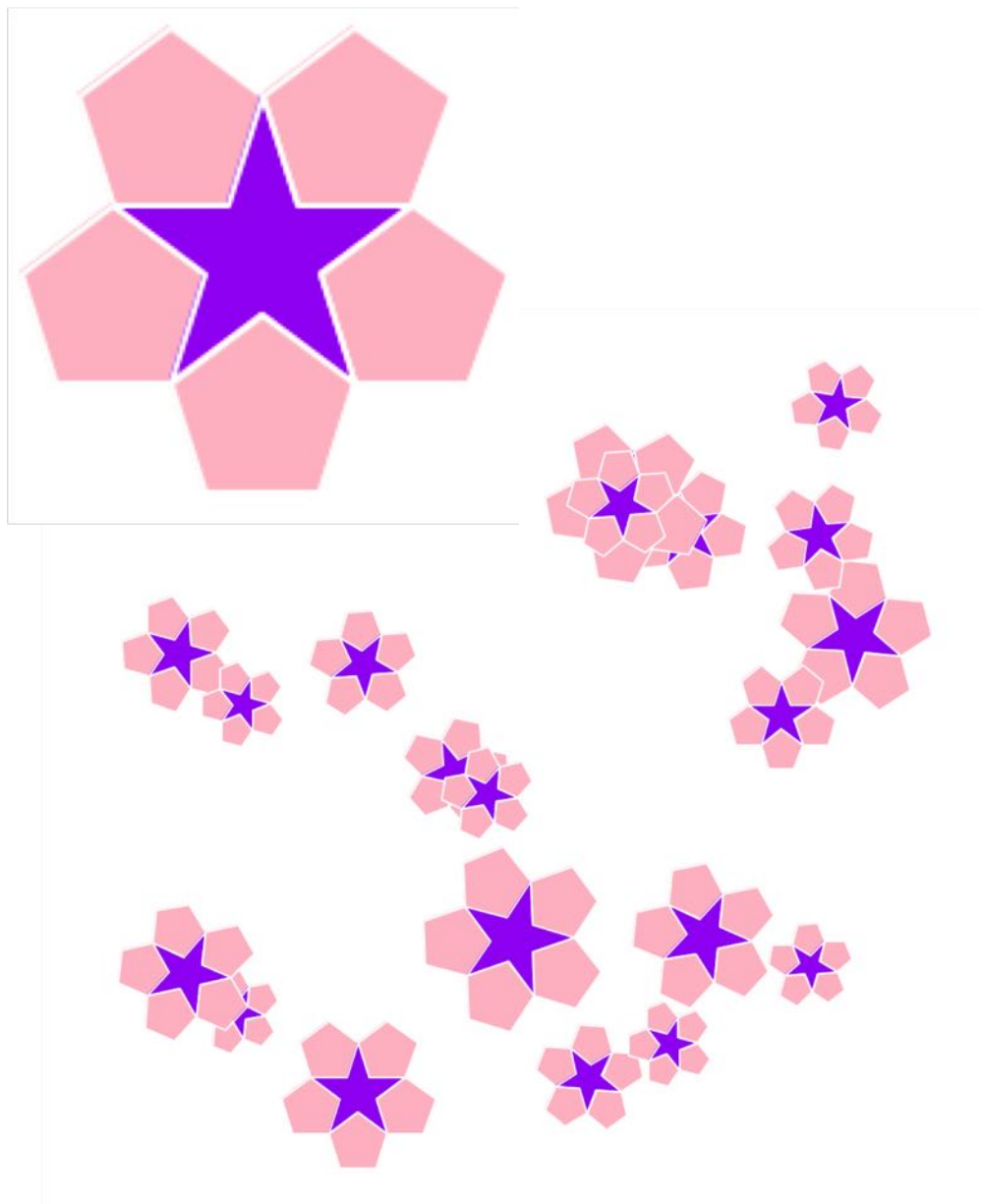
上机练习

- 定义一个多边形函数
 - `def polygon(n,size,color):`
 - 绘制正n边形，边长为size，填充颜色color
- 编写一个程序，绘制现代时尚几何多边形色块抽象装饰画
 - 随机模块random
 - `t.goto(x,y)`
- 可以进一步修改程序
 - 如：将H4中的曲线定义为函数，组合进随机图案中来





函数调用函数

- 只要def定义过的函数对象都可以被调用
- 也可以从函数里调用另一个函数
- 例如flower函数
 - 有参数size
 - 调用了star画紫色五角星
 - 调用了polygon画粉色五边形





程序: func_call2.py



```
1 import turtle
2 import random
3 import math
4
5
6 def polygon(n, size, color):
7     t.fillcolor(color)
8     t.begin_fill()
9     for i in range(n):
10         t.forward(size)
11         t.right(360 / n)
12     t.end_fill()
13
14
15 def star(size, color):
16     t.fillcolor(color)
17     t.begin_fill()
18     for i in range(5):
19         t.forward(size)
20         t.left(72)
21         t.forward(size)
22         t.right(144)
23     t.end_fill()
```

```
26 def flower(size):
27     for i in range(5):
28         t.forward(2 * size * (1 + math.sin(18 * math.pi / 180)))
29         t.right(36)
30         polygon(5, size, 'pink')
31         t.right(108)
32         star(size, 'purple')
33
34
35 t = turtle.Turtle()
36 t.speed(0)
37 t.pencolor('white')
38 for i in range(20):
39     t.penup()
40     x = random.randint(-200, 200)
41     y = random.randint(-200, 200)
42     t.goto(x, y)
43     t.pendown()
44     head = random.randint(1, 9)
45     t.setheading(head * 40)
46     size = random.randint(10, 25)
47     flower(size)
48 t.hideturtle()
49 turtle.done()
```



递归：函数调用自己？

- 函数可以调用自己吗？
 - 从Python语言的函数定义角度来说
 - def在前，调用在后，所以函数可以调用自己，称为“递归调用”
- 那会不会有什么问题？

recursion.py

```
1  def tell_story():  
2      print("从前有座山，山上有座庙，庙里有个老和尚，他在讲：")  
3      tell_story()  
4  
5  
6  tell_story()
```

“好”的递归：必须有终止条件

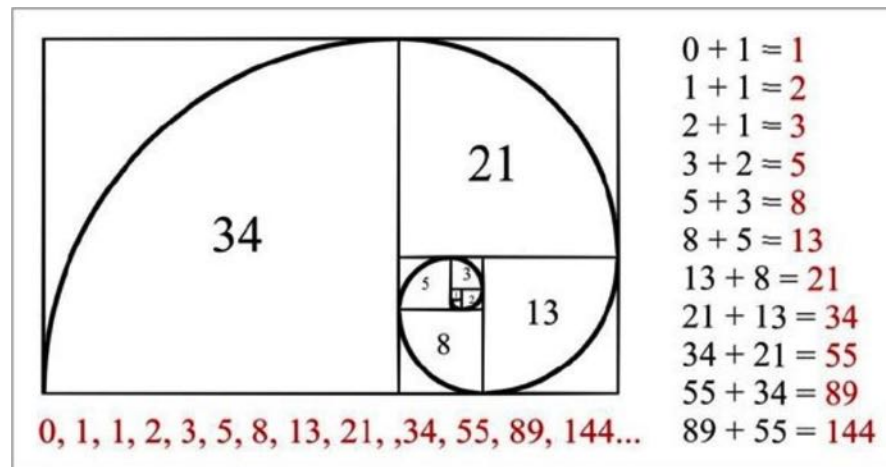
- 递归调用很有用，但要有个终止结束条件，结束时不再调用自身
- 一般是通过参数来控制，递归调用时让参数向终止条件演进
 - 例子：参数 n ，结束条件是 $n==0$ ，递归调用时参数为 $n-1$
 - 无论多大的 n ，总会变为 0

recursion2.py

```
1  def tell_story(n):
2      if n > 0:
3          print("从前有座山，山上有座庙，庙里有个老和尚，他在讲：")
4          tell_story(n - 1)
5      else:
6          print("讲完了！")
7
8
9  tell_story(10)
```

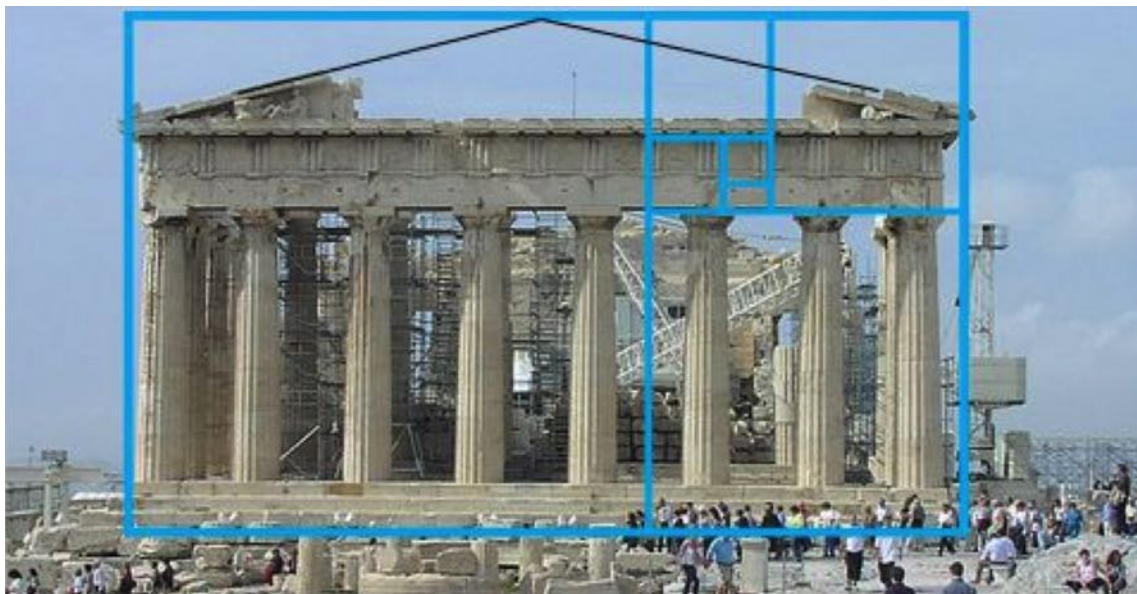
典型例子：斐波那契数列

- 斐波那契数列的发现者，是意大利数学家列昂纳多·斐波那契 (Leonardo Fibonacci)
- 从第3项开始，每一项都等于前两项之和
- 相邻两项之比，无限趋近于“黄金分割数” 0.618....
- 在自然界中有很多实例
- 黄金分割比例广泛应用于美术



艺术中的黄金分割Golden Ratio

建筑艺术



摄影构图



递归函数： fibonacci

- 观察fibonacci数列的定义
- 一个典型的递归定义
- 可以写出递归函数fibonacci
- 这里，递归结束条件是什么？
- 请分析fibonacci(5)的调用？

$$F_0 = 0$$

$$F_1 = 1$$

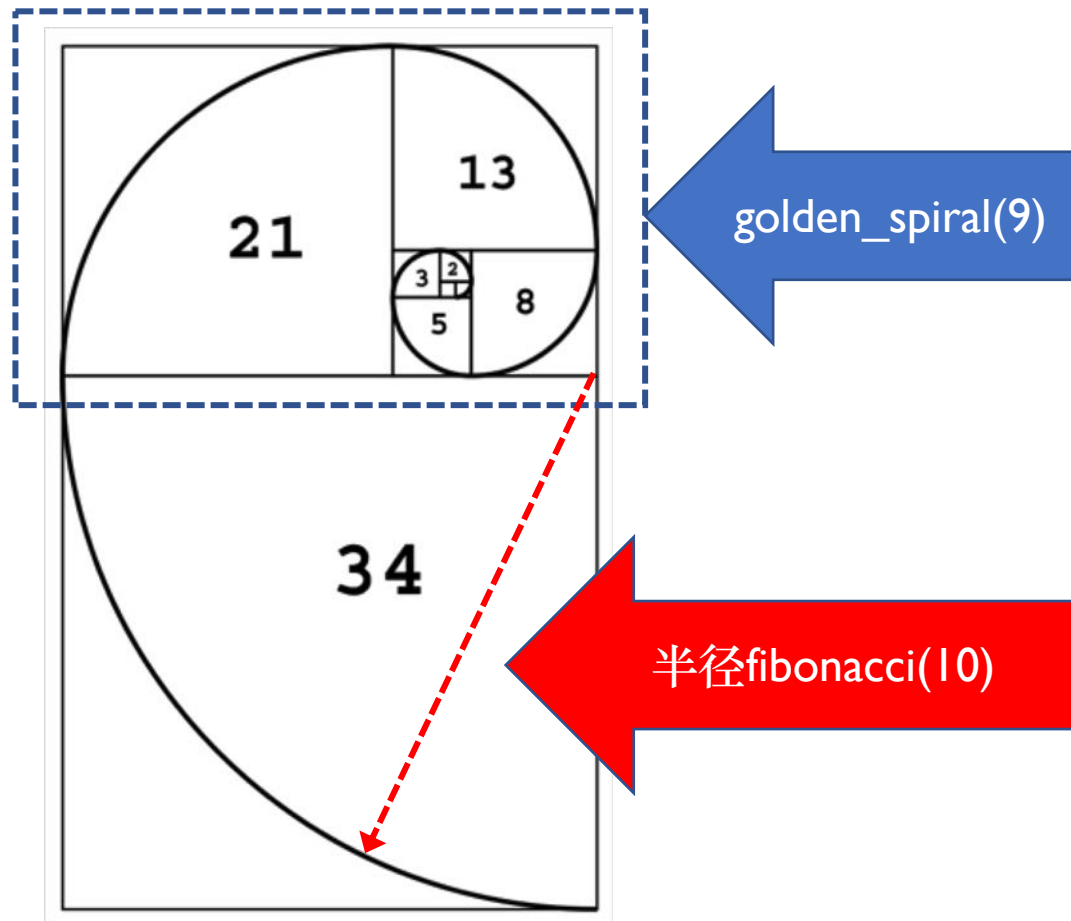
$$F_n = F_{n-1} + F_{n-2}, n > 1$$

```
1  def fibonacci(n):  
2      if n == 1:  
3          return 0  
4      elif n == 2:  
5          return 1  
6      else:  
7          return fibonacci(n - 1) + fibonacci(n - 2)  
8  
9  
10     for i in range(1, 12):  
11         print(i, fibonacci(i))
```

fibonacci.py


画出这条“黄金螺线”

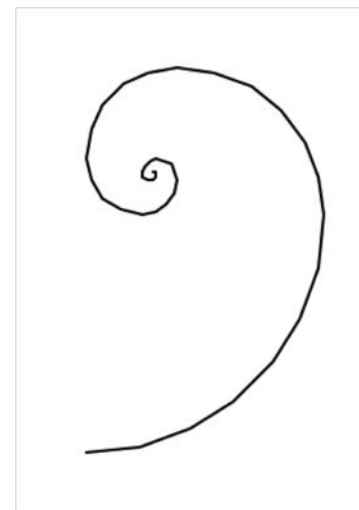
- 海龟作图circle函数
 - `circle(<半径>, <角度>)`
- 右图是10阶黄金螺线
 - 由半径为`fibonacci(10)`的1/4圆弧
 - 加上9阶黄金螺线构成
- 递归函数`golden_spiral(n)`:
 - `circle(fibonacci(10), 90)`
 - `golden_spiral(n-1)`



程序： fibonacci2.py

- 看看如何写出golden_spiral函数？（代码已遮挡）
- 注意：递归结束条件

```
1  import turtle
2
3
4  def fibonacci(n):
5      if n == 1:
6          return 0
7      elif n == 2:
8          return 1
9      else:
10         return fibonacci(n - 1) + fibonacci(n - 2)
11
12
13 def golden_spiral(n):
14     
15
16
```

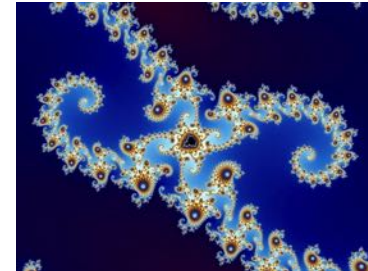
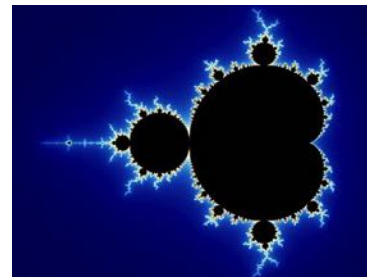


fibonacci2.py

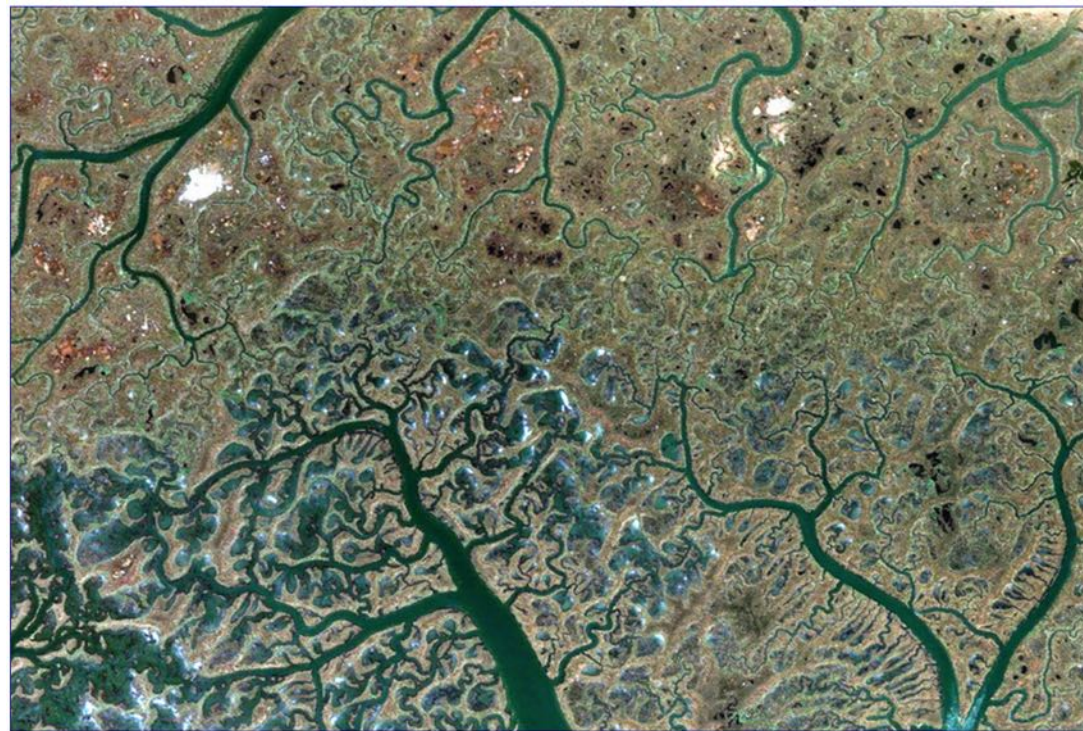
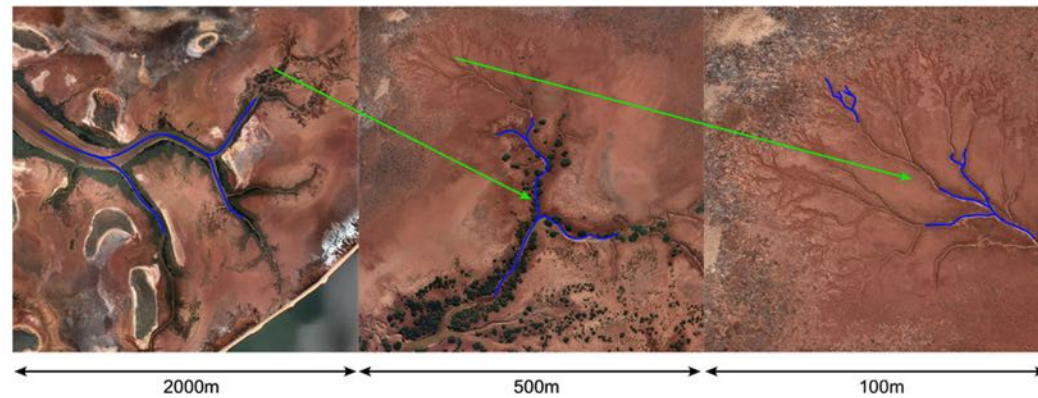
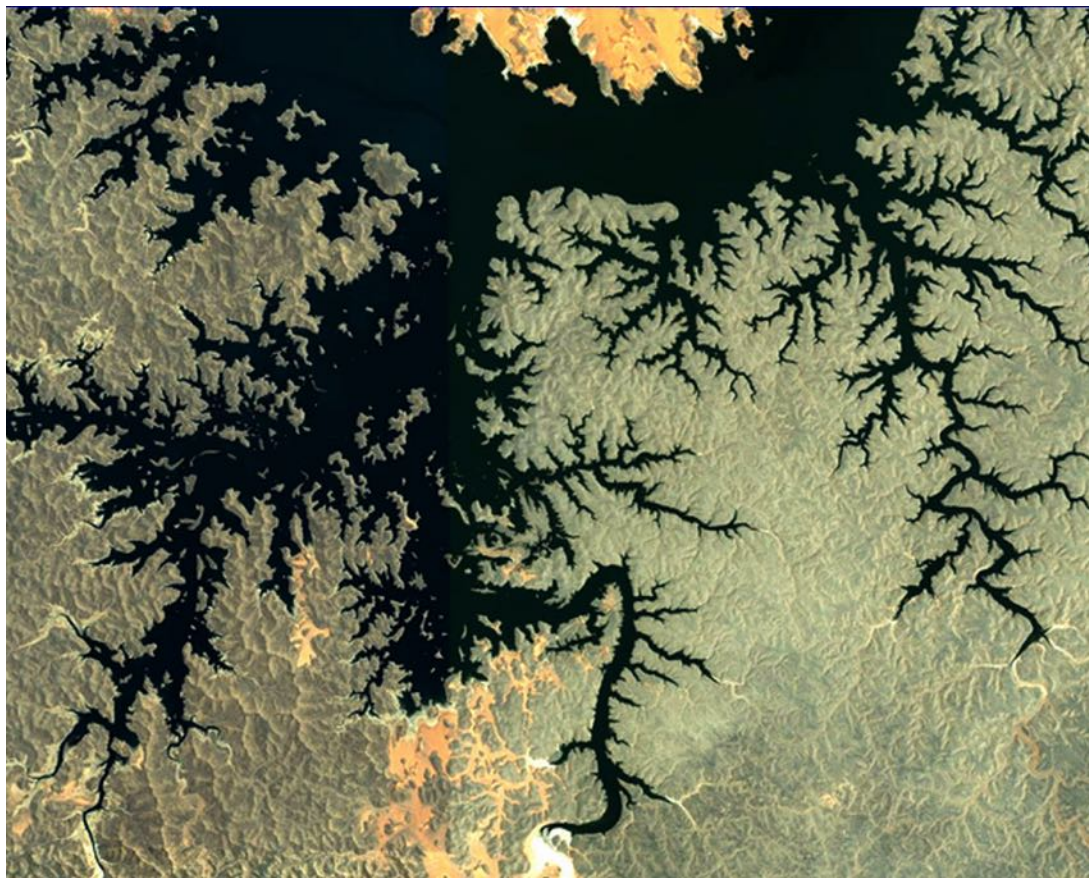
```
19  t = turtle.Turtle()
20
21  golden_spiral(12)
22
23  t.hideturtle()
24  turtle.done()
```


分形树：自相似递归图形

- 分形Fractal，是1975年由Mandelbrot开创的新学科
- 通常被定义为“一个粗糙或零碎的几何形状，可以分成数个部分，且每一部分都（至少近似地）是整体缩小后的形状”，即具有**自相似**的性质。
- 自然界中能找到众多具有分形性质的物体
 - 海岸线、山脉、闪电、云朵、雪花、树
 - <http://paulbourke.net/fractals/googleearth/>
 - <http://recursivedrawing.com/>

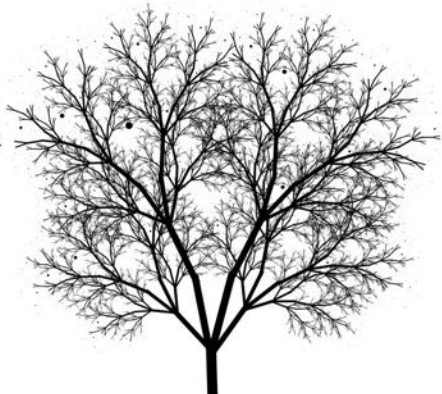


自相似的河流/海岸线



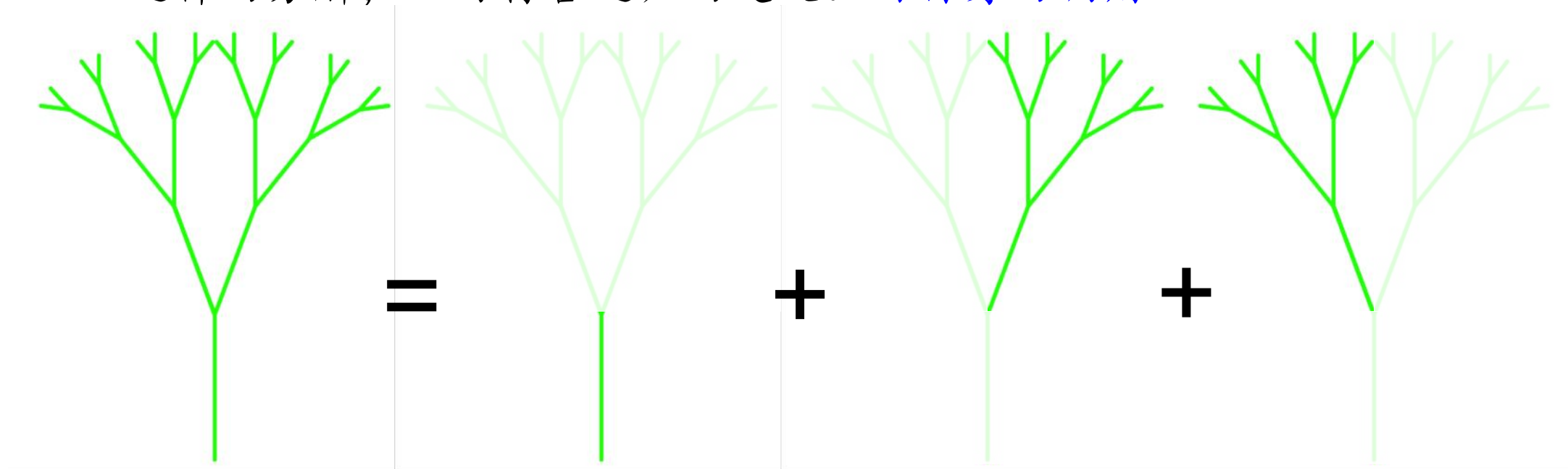
分形树：自相似递归图形

- 自然现象中所具备的分形特性，使得计算机可以通过分形算法生成非常逼真的自然场景，下面我们以树为例做一个粗糙的近似
- 我们发现，一棵树的每个分叉和每条树枝，实际上都具有整棵树的外形特征（也是逐步分叉的）



二叉树的递归分解

- 可以把二叉树分解为三个部分：树干、左边的小树、右边的小树
 - 这样的分解，正好符合递归的定义：对自身的调用



二叉树

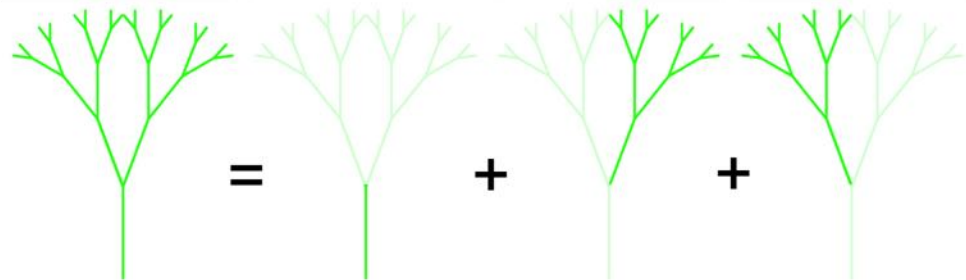
树干

倾斜的右小树

倾斜的左小树

程序: tree.py

```
1  import turtle
2
3
4  def tree(branch_len):
5      if branch_len > 5: # 树干太短不画, 即递归结束条件
6          t.forward(branch_len) # 画树干
7          t.right(20) # 右倾斜20度
8          tree(branch_len - 15) # 递归调用, 画右边的小树, 树干减15
9          t.left(40) # 向左回40度, 即左倾斜20度
10         tree(branch_len - 15) # 递归调用, 画左边的小树, 树干减15
11         t.right(20) # 向右回20度, 即回正
12         t.backward(branch_len) # 海龟退回原位置
```



二叉树

树干

倾斜的右小树

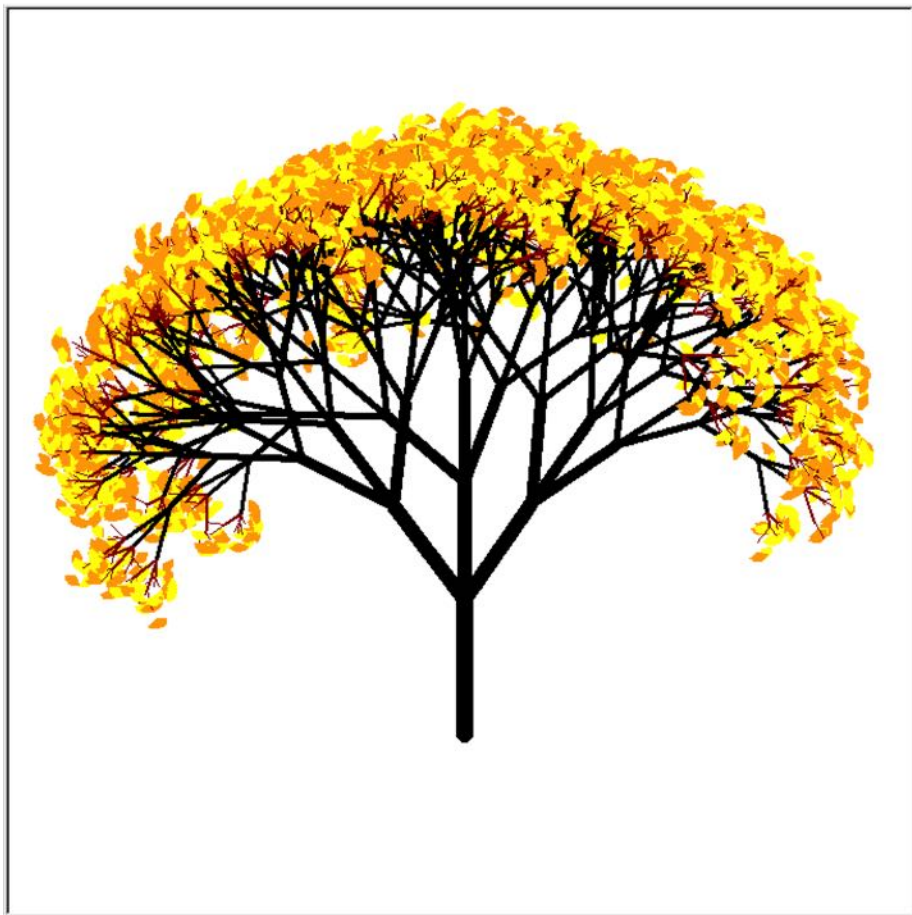
倾斜的左小树

```
15 t = turtle.Turtle()
16 t.left(90)
17 t.penup()
18 t.backward(100)
19 t.pendown()
20 t.pencolor('green')
21 t.pensize(2)
22 tree(75) # 画树干长度75的二叉树
23 t.hideturtle()
24 turtle.done()
```

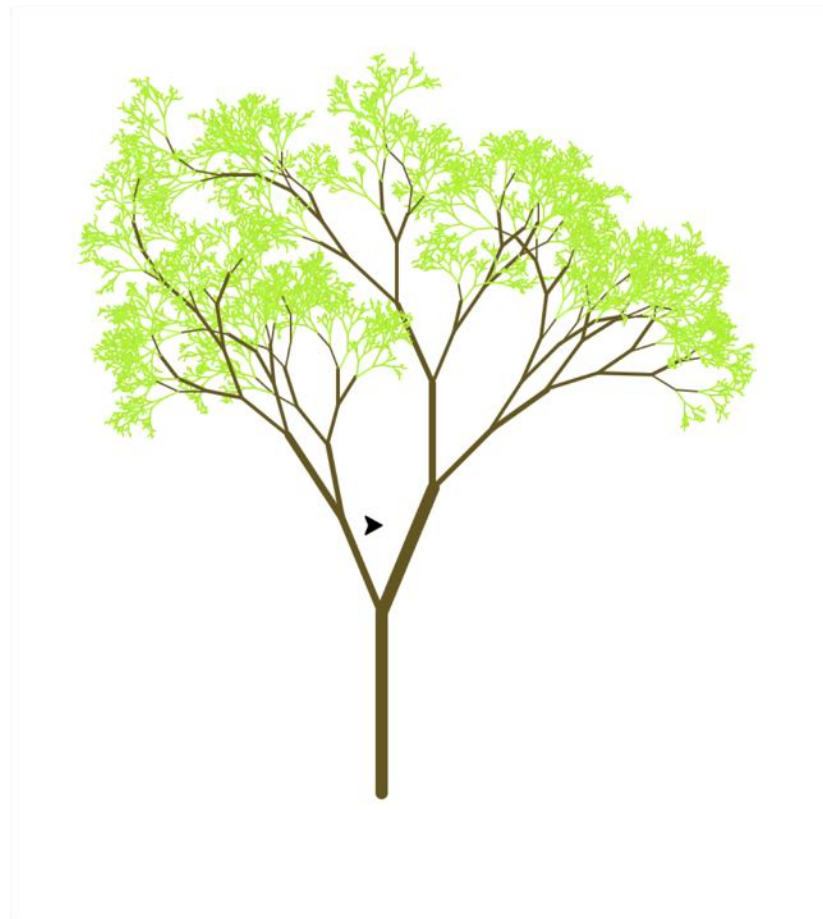
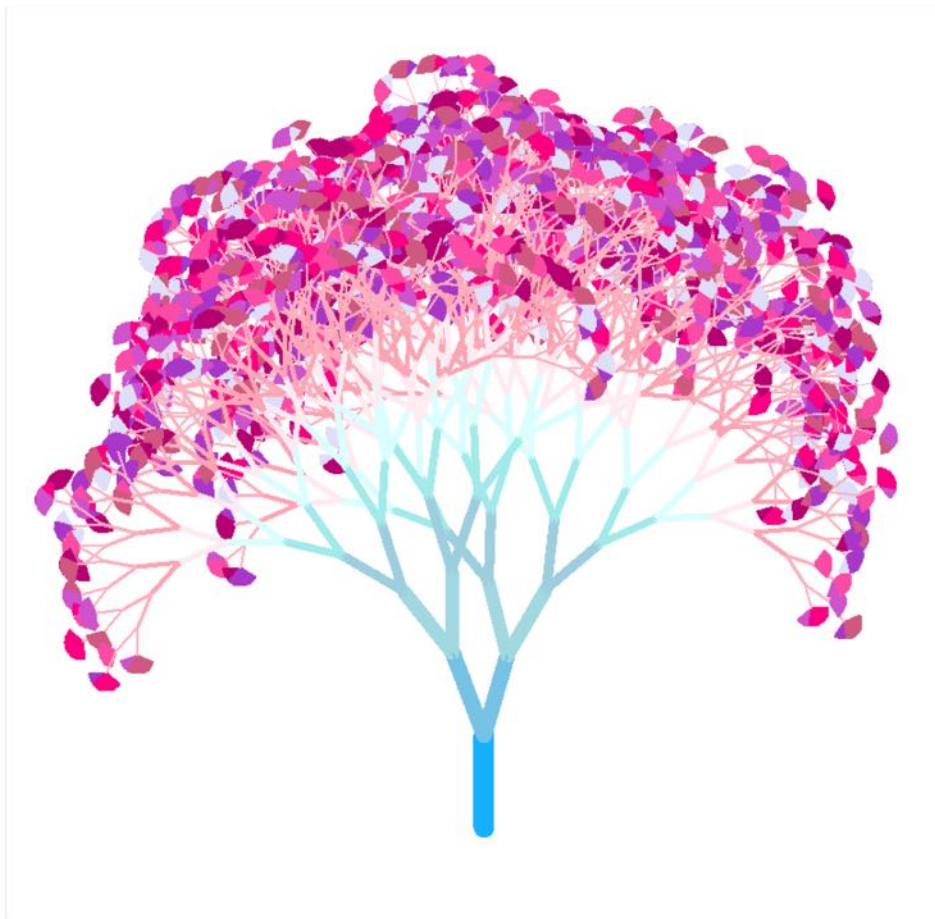
【H6】 分组作业： 艺术分形树

- 修改分形树程序，增加如下功能：
 - 树枝的粗细可以变化，随着树枝缩短，也相应变细
 - 树枝的颜色可以变化，当树枝非常短的时候，使之看起来像树叶的颜色
 - 让树枝倾斜角度在一定范围内随机变化，如15~45度之间，左右倾斜也可不一样，做成你认为最好看的样子
 - 树枝的长短也可以在一定范围内随机变化，使得整棵树看起来更加逼真

学生作业精选：分形树



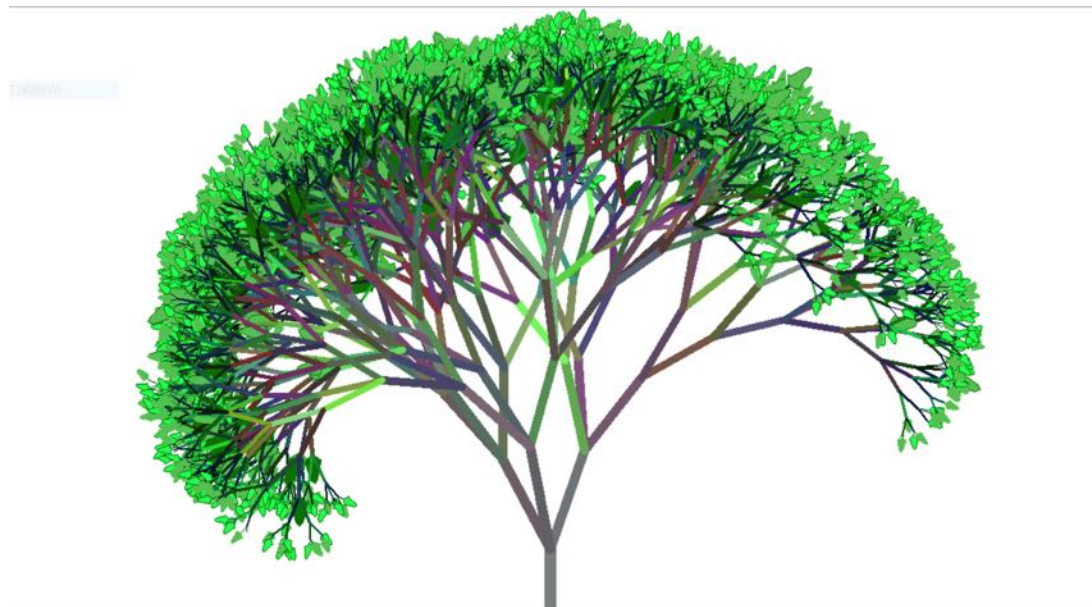
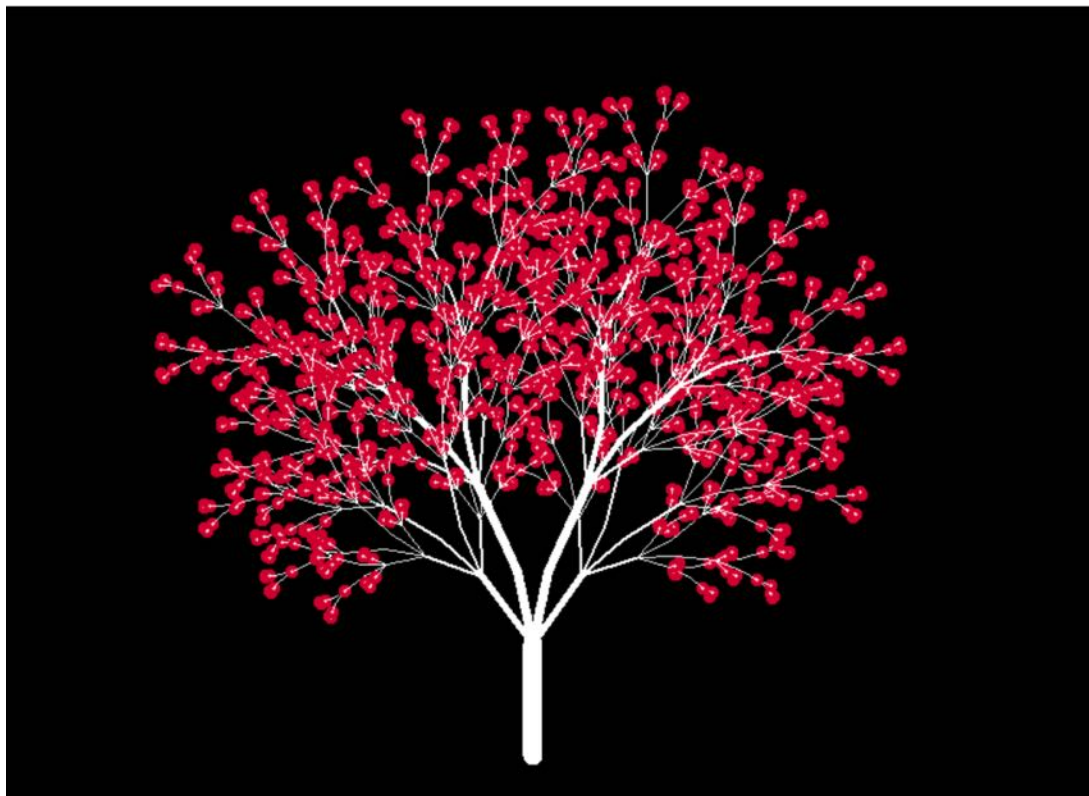
学生作业精选：分形树



学生作业精选：分形树



学生作业精选：分形树



学生作业精选：分形树

