

Python语言程序设计

## 第5章 函数和代码复用

---





# 本课概要

# 第5章 函数和代码复用



- 5.1 函数的定义与使用
- 5.2 实例7: 七段数码管绘制
- 5.3 代码复用与函数递归
- 5.4 模块4: PyInstaller库的使用
- 5.5 实例8: 科赫雪花小包裹



# 第5章 函数和代码复用

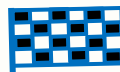
## 方法论

- Python基本代码抽象即函数的使用方法



## 实践能力

- 学会编写带有函数并复用代码的程序





# 前课复习



# 数字类型及操作

- 整数类型的无限范围及4种进制表示
- 浮点数类型的近似无限范围、小尾数及科学计数法
- +、-、\*、/、//、%、\*\*、二元增强赋值操作符
- abs()、divmod()、pow()、round()、max()、min()
- int()、float()、complex()



python

语言程序设计

```
#DayDayUpQ3.py
```

```
dayup = 1.0
```

```
dayfactor = 0.01
```

```
for i in range(365):
```

```
    if i % 7 in [6,0]:
```

```
        dayup = dayup*(1-dayfactor)
```

```
    else:
```

```
        dayup = dayup*(1+dayfactor)
```

```
print("工作日的力量: {:.2f} ".format(dayup))
```

**for..in.. (计算思维)**

天好  
天好  
向學  
上習

#DayDayUpQ4.py

```
def dayUP(df):
```

```
    dayup = 1
```

```
    for i in range(365):
```

```
        if i % 7 in [6,0]:
```

```
            dayup = dayup*(1 - 0.01)
```

```
        else:
```

```
            dayup = dayup*(1 + df)
```

```
    return dayup
```

```
dayfactor = 0.01
```

```
while dayUP(dayfactor) < 37.78:
```

```
    dayfactor += 0.001
```

```
print("工作日的努力参数是: {:.3f}".format(dayfactor))
```

def..while..

("笨办法"试错)

天好  
天好  
向學  
上習



# 字符串类型及操作

- 正向递增序号、反向递减序号、<字符串>[M:N:K]
- +、\*、len()、str()、hex()、oct()、ord()、chr()
- .lower()、.upper()、.split()、.count()、.replace()
- .center()、.strip()、.join() 、.format()格式化



## #TextProBarV3.py

```
import time
```

```
scale = 50
```

```
print("执行开始".center(scale//2, "-"))
```

```
start = time.perf_counter()
```

```
for i in range(scale+1):
```

```
    a = '*' * i
```

```
    b = '.' * (scale - i)
```

```
    c = (i/scale)*100
```

```
    dur = time.perf_counter() - start
```

```
    print("\r{:^3.0f}%[{}->{}]{:.2f}s".format(c,a,b,dur),end='')
```

```
    time.sleep(0.1)
```

```
print("\n"+"执行结束".center(scale//2, '-'))
```



# 程序的分支结构

- 单分支 *if* 二分支 *if-else* 及紧凑形式
- 多分支 *if-elif-else* 及条件之间关系
- *not and or > >= == <= < !=*
- 异常处理 *try-except-else-finally*



```
#CalBMI.py
```

```
height, weight = eval(input("请输入身高(米)和体重\ (公斤)[逗号隔开]: "))
```

```
bmi = weight / pow(height, 2)
```

```
print("BMI 数值为: {:.2f}".format(bmi))
```

```
who, nat = "", ""
```

```
if bmi < 18.5:
```

```
    who, nat = "偏瘦", "偏瘦"
```

```
elif 18.5 <= bmi < 24:
```

```
    who, nat = "正常", "正常"
```

```
elif 24 <= bmi < 25:
```

```
    who, nat = "正常", "偏胖"
```

```
elif 25 <= bmi < 28:
```

```
    who, nat = "偏胖", "偏胖"
```

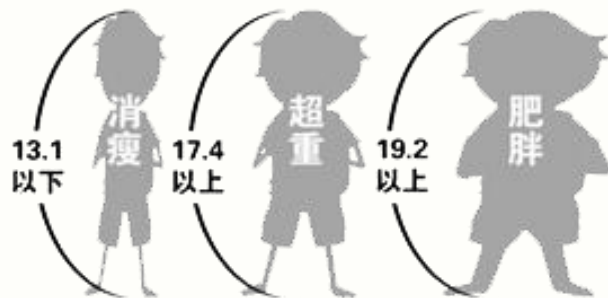
```
elif 28 <= bmi < 30:
```

```
    who, nat = "偏胖", "肥胖"
```

```
else:
```

```
    who, nat = "肥胖", "肥胖"
```

```
print("BMI 指标为:国际'{0}', 国内'{1}'".format(who, nat))
```



# 程序的循环结构

- *for...in* 遍历循环: 计数、字符串、列表、文件...
- *while* 无限循环
- *continue* 和 *break* 保留字: 退出当前循环层次
- 循环 *else* 的高级用法: 与 *break* 有关



```
#CalPiV2.py
```

```
from random import random
```

```
from time import perf_counter
```

```
DARTS = 1000*1000
```

```
hits = 0.0
```

```
start = perf_counter()
```

```
for i in range(1, DARTS+1):
```

```
    x, y = random(), random()
```

```
    dist = pow(x ** 2 + y ** 2, 0.5)
```

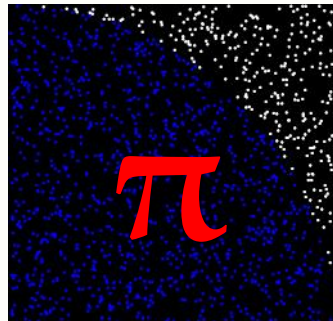
```
    if dist <= 1.0:
```

```
        hits = hits + 1
```

```
pi = 4 * (hits/DARTS)
```

```
print("圆周率值是: {}".format(pi))
```

```
print("运行时间是: {:.5f}s".format(perf_counter()-start))
```

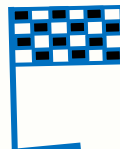
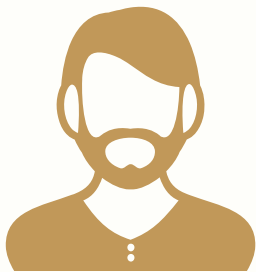




# 练习与作业

# 第5章 函数和代码复用

练习





## 5.1 函数的定义与使用

---



# 函数的定义与使用

- 函数的理解与定义
- 函数的使用及调用过程
- 函数的参数传递
- 函数的返回值
- 局部变量和全局变量
- lambda函数



python

语言程序设计



# 函数的理解和定义

#DayDayUpQ4.py

```
def dayUP(df):  
    dayup = 1  
    for i in range(365):  
        if i % 7 in [6,0]:  
            dayup = dayup*(1 - 0.01)  
        else:  
            dayup = dayup*(1 + df)  
    return dayup
```

dayfactor = 0.01

while dayUP(dayfactor) < 37.78:

dayfactor += 0.001

print("工作日的努力参数是: {:.3f}".format(dayfactor))

**def..while..**

**(“笨办法”试错)**



# 函数的定义

函数是一段代码的表示

- 函数是一段具有特定功能的、可重用的语句组
- 函数是一种功能的抽象，一般函数表达特定功能
- 两个作用：降低编程难度 和 代码复用

# 函数的定义

函数是一段代码的表示

*def* <函数名>(<参数(0个或多个)>) :

<函数体>

*return* <返回值>

# 函数的定义

函数名

参数

*def* fact(n) :

s = 1

*for* i *in* range(1, n+1):

s \*= i

*return* s

返回值

计算 n!

# 函数的定义

$$y = f(x)$$

- 函数定义时，所指定的参数是一种**占位符**
- 函数定义后，如果不经**调用**，不会被执行
- 函数定义时，参数是输入、函数体是处理、结果是输出 (IPO)





# 函数的使用及调用过程

# 函数的调用

调用是运行函数代码的方式

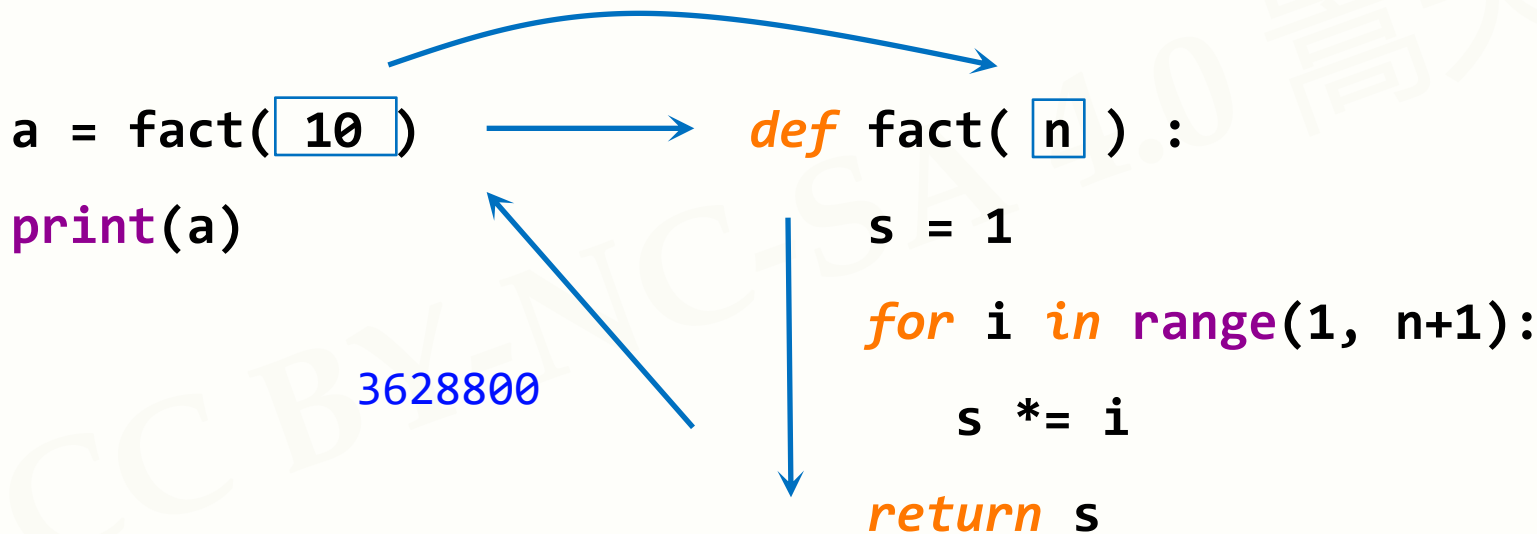
```
def fact(n) :           函数的定义
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
```

fact(10)

函数的调用

- 调用时要给出实际参数
- 实际参数替换定义中的参数
- 函数调用后得到返回值

# 函数的调用过程





# 函数的参数传递

# 参数个数

函数可以有参数，也可以没有，但必须保留括号

*def* <函数名>() :

<函数体>

*return* <返回值>

*def* fact() :

*print*("我也是函数")

# 可选参数传递

函数定义时可以为某些参数指定默认值，构成可选参数

```
def    <函数名>(<非可选参数>, <可选参数>) :
```

```
    <函数体>
```

```
    return    <返回值>
```

# 可选参数传递

可选参数



```
def fact(n, m=1) :
```

```
    s = 1
```

```
    for i in range(1, n+1):
```

```
        s *= i
```

```
    return s//m
```

计算  $n!//m$

```
>>> fact(10)
```

```
3628800
```

```
>>> fact(10,5)
```

```
725760
```

# 可变参数传递

函数定义时可以设计可变数量参数，既不确定参数总数量

```
def    <函数名>( <参数>, *b ) :
```

```
    <函数体>
```

```
    return  <返回值>
```



# 可变参数传递

可变参数

```
def fact(n, *b) :
```

```
    s = 1
```

```
    for i in range(1, n+1):
```

```
        s *= i
```

```
    for item in b:
```

```
        s *= item
```

```
    return s
```

```
>>> fact(10,3)
```

```
10886400
```

```
>>> fact(10,3,5,8)
```

```
435456000
```

计算 n!乘数

# 参数传递的两种方式

函数调用时，参数可以按照位置或名称方式传递

```
def fact(n, m=1) :  
    s = 1  
    for i in range(1, n+1):  
        s *= i  
    return s//m
```

>>> fact(10, 5)

725760

>>> fact(m=5, n=10)

725760

位置传递

名称传递



# 函数的返回值

# 函数的返回值

函数可以返回0个或多个结果

- **return**保留字用来传递返回值
- 函数可以有返回值，也可以没有，可以有**return**，也可以没有
- **return**可以传递0个返回值，也可以传递任意多个返回值

# 函数的返回值

函数调用时，参数可以按照位置或名称方式传递

```
def fact(n, m=1) :  
    s = 1  
    for i in range(1, n+1):  
        s *= i  
    return s//m, n, m
```

```
>>> fact( 10, 5 )
```

元组类型

```
(725760, 10, 5)
```

```
>>> a,b,c = fact(10,5)
```

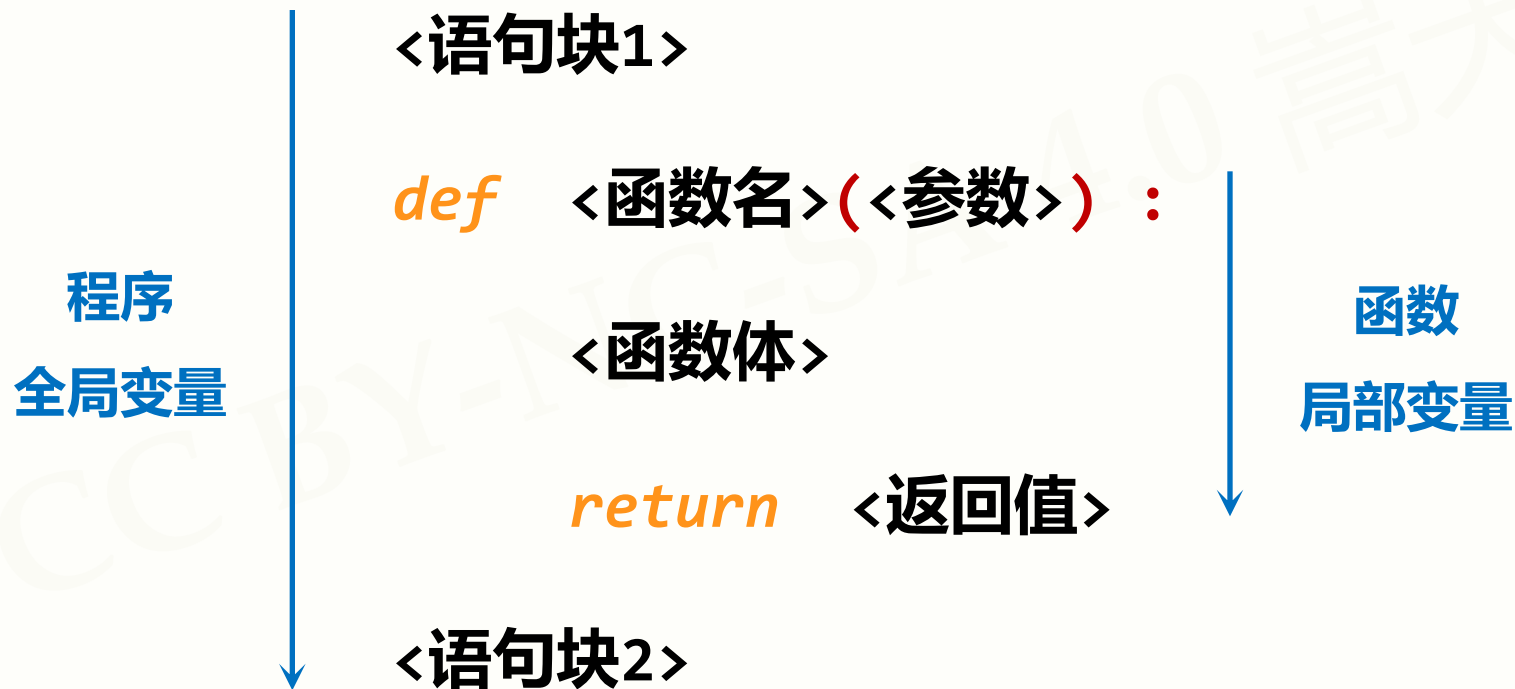
```
>>> print(a,b,c)
```

```
725760 10 5
```



# 局部变量和全局变量

# 局部变量和全局变量



# 局部变量和全局变量

```
n, s = 10, 100
```



n和s是全局变量

```
def fact(n):
```

```
    s = 1
```



fact()函数中的n和s是局部变量

```
    for i in range(1, n+1):
```

```
        s *= i
```

```
    return s
```

运行结果

>>>

```
print(fact(n), s)
```



n和s是全局变量

3628800 100



# 局部变量和全局变量

## 规则1: 局部变量和全局变量是不同变量

- 局部变量是函数内部的占位符，与全局变量可能重名但不同
- 函数运算结束后，局部变量被释放
- 可以使用`global`保留字在函数内部使用全局变量

# 局部变量和全局变量

```
n, s = 10, 100
```

```
def fact(n) :
```

```
    s = 1
```

```
    for i in range(1, n+1):
```

```
        s *= i
```

```
    return s
```

```
print(fact(n), s)
```

fact()函数中s是局部变量

与全局变量s不同

运行结果

>>>

3628800 100

此处局部变量s是3628800

此处全局变量s是100

# 局部变量和全局变量

```
n, s = 10, 100
```

```
def fact(n) :
```

fact()函数中使用global保留字声明

```
    global s
```

← 此处s是全局变量s

```
    for i in range(1, n+1):
```

```
        s *= i
```

运行结果

```
    return s
```

← 此处s指全局变量s

>>>

```
print(fact(n), s) ← 此处全局变量s被函数修改
```

362880000 362880000

# 局部变量和全局变量

**规则2: 局部变量为组合数据类型且未创建，等同于全局变量**

`ls = ["F", "f"]` ← 通过使用[]真实创建了一个全局变量列表ls

`def func(a) :`

`ls.append(a)` ←

此处ls是列表类型，未真实创建  
则等同于全局变量

`return`

`func("C")` ←

全局变量ls被修改

`print(ls)`

**运行结果**

`>>>`

`['F', 'f', 'C']`

# 局部变量和全局变量

`ls = ["F", "f"]` ← 通过使用[]真实创建了一个全局变量列表ls

*def* func(a) :

`ls = []`

此处ls是列表类型，真实创建  
ls是局部变量

`ls.append(a)`

*return*

`func("C")`

← 局部变量ls被修改

`print(ls)`

运行结果

>>>

['F', 'f']

# 局部变量和全局变量

## 使用规则

- 基本数据类型，无论是否重名，局部变量与全局变量不同
- 可以通过global保留字在函数内部声明全局变量
- 组合数据类型，如果局部变量未真实创建，则是全局变量



lambda函数



# lambda函数

## lambda函数返回函数名作为结果

- lambda函数是一种匿名函数，即没有名字的函数
- 使用 *lambda* 保留字定义，函数名是返回结果
- lambda函数用于定义简单的、能够在一行内表示的函数



# lambda函数

〈函数名〉 = *lambda* 〈参数〉: 〈表达式〉

等价于

*def* 〈函数名〉(〈参数〉) :

〈函数体〉

*return* 〈返回值〉

# lambda函数

```
>>> f = lambda x, y : x + y
```

```
>>> f(10, 15)
```

```
25
```

```
>>> f = lambda : "lambda函数"
```

```
>>> print(f())
```

```
lambda函数
```

# lambda函数的应用

## 谨慎使用lambda函数

- lambda函数主要用作一些特定函数或方法的参数
- lambda函数有一些固定使用方式，建议逐步掌握
- 一般情况，建议使用`def`定义的普通函数



## 单元小结

# 函数的定义与使用

- 使用保留字`def`定义函数，`lambda`定义匿名函数
- 可选参数(赋初值)、可变参数(\*b)、名称传递
- 保留字`return`可以返回任意多个结果
- 保留字`global`声明使用全局变量，一些隐式规则



## 5.2 实例7: 七段数码管绘制

---

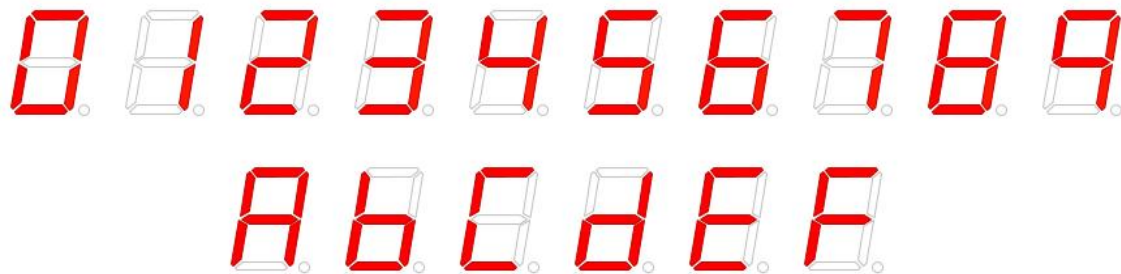
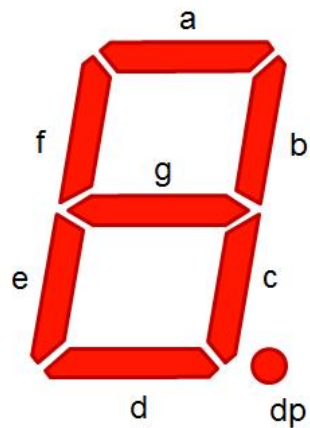




# "七段数码管绘制"问题分析

# 问题分析

## 七段数码管





# 问题分析

## 七段数码管绘制

- 需求：用程序绘制七段数码管，似乎很有趣
- 该怎么做呢？

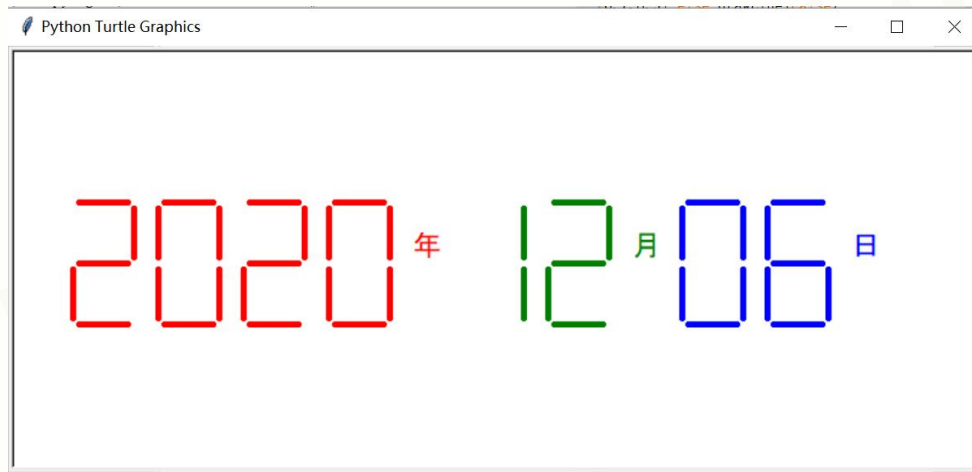
turtle绘图体系



七段数码管绘制

# 问题分析

## 七段数码管绘制时间





# "七段数码管绘制"实例讲解(上)

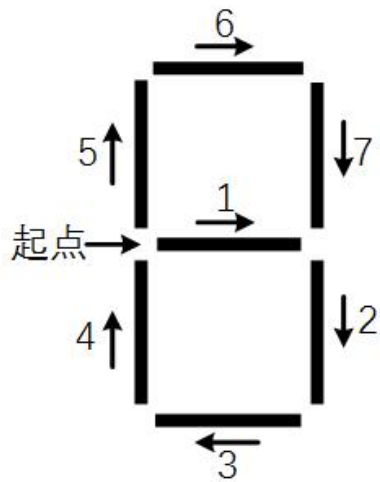
# 七段数码管绘制

## 基本思路

- **步骤1：绘制单个数字对应的数码管**
- **步骤2：获得一串数字，绘制对应的数码管**
- **步骤3：获得当前系统时间，绘制对应的数码管**

# 七段数码管绘制

## 步骤1: 绘制单个数码管

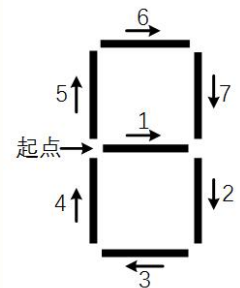


- 七段数码管由7个基本线条组成
- 七段数码管可以有固定顺序
- 不同数字显示不同的线条

```

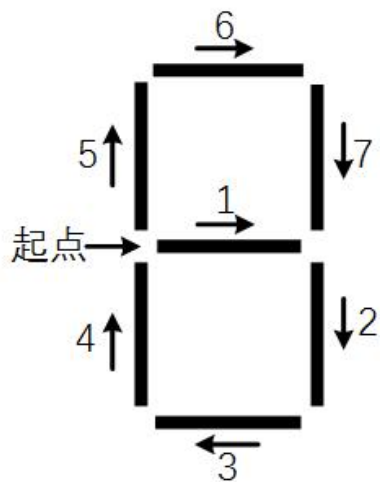
import turtle
def drawLine(draw):    #绘制单段数码管
    turtle.pendown() if draw else turtle.penup()
    turtle.fd(40)
    turtle.right(90)
def drawDigit(digit): #根据数字绘制七段数码管
    drawLine(True) if digit in [2,3,4,5,6,8,9] else drawLine(False)
    drawLine(True) if digit in [0,1,3,4,5,6,7,8,9] else drawLine(False)
    drawLine(True) if digit in [0,2,3,5,6,8,9] else drawLine(False)
    drawLine(True) if digit in [0,2,6,8] else drawLine(False)
    turtle.left(90)
    drawLine(True) if digit in [0,4,5,6,8,9] else drawLine(False)
    drawLine(True) if digit in [0,2,3,5,6,7,8,9] else drawLine(False)
    drawLine(True) if digit in [0,1,2,3,4,7,8,9] else drawLine(False)
    turtle.left(180)
    turtle.penup() #为绘制后续数字确定位置
    turtle.fd(20)  #为绘制后续数字确定位置

```

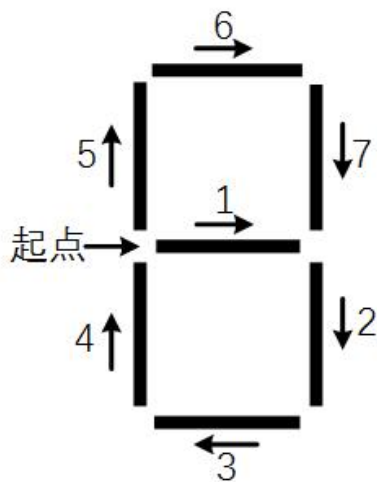


# 七段数码管绘制

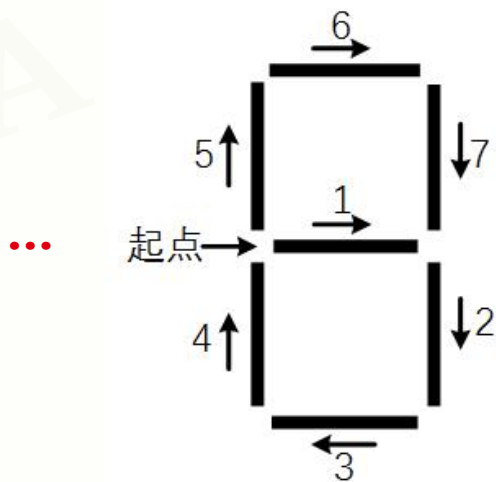
## 步骤2: 获取一段数字, 绘制多个数码管



第1个



第2个



第N个

```
import turtle
def drawLine(draw):    #绘制单段数码管
    ... (略)
def drawDigit(digit): #根据数字绘制七段数码管
    ... (略)
def drawDate(date):   #获得要输出的数字
    for i in date:
        drawDigit(eval(i)) #通过eval()函数将数字变为整数
def main():
    turtle.setup(800, 350, 200, 200)
    turtle.penup()
    turtle.fd(-300)
    turtle.pensize(5)
    drawDate('20201206')
    turtle.hideturtle()
    turtle.done()
main()
```





**准备好电脑，与老师一起编码吧！**



# "七段数码管绘制"实例讲解(下)

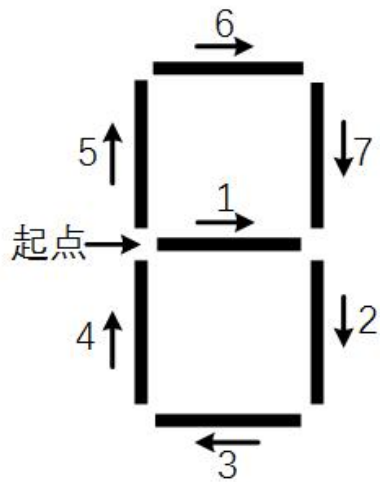
# 七段数码管绘制

## 基本思路

- 步骤1：绘制单个数字对应的数码管
- 步骤2：获得一串数字，绘制对应的数码管
- 步骤3：获得当前系统时间，绘制对应的数码管

# 七段数码管绘制

## 绘制漂亮的七段数码管



- 增加七段数码管之间线条间隔

```
import turtle
```

```
def drawGap():    #绘制数码管间隔
```

```
    turtle.penup()
```

```
    turtle.fd(5)
```

```
def drawLine(draw):    #绘制单段数码管
```

```
    drawGap()
```

```
    turtle.pendown() if draw else turtle.penup()
```

```
    turtle.fd(40)
```

```
    drawGap()
```

```
    turtle.right(90)
```

```
def drawDigit(digit): #根据数字绘制七段数码管
```

```
    drawLine(True) if digit in [2,3,4,5,6,8,9] else drawLine(False)
```

```
    drawLine(True) if digit in [0,1,3,4,5,6,7,8,9] else drawLine(False)
```

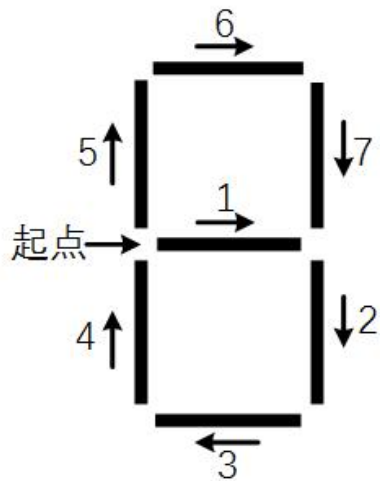
```
    drawLine(True) if digit in [0,2,3,5,6,8,9] else drawLine(False)
```

```
    drawLine(True) if digit in [0,2,6,8] else drawLine(False)
```

```
    ... (略)
```

# 七段数码管绘制

## 步骤3: 获取系统时间, 绘制七段数码管



- 使用time库获得系统当前时间
- 增加年月日标记
- 年月日颜色不同

```
import turtle, time
...(略)
def drawDate(date):    #data为日期, 格式为 '%Y-%m=%d+'
    turtle.pencolor("red")
    for i in date:
        if i == '-':
            turtle.write('年',font=("Arial", 18, "normal"))
            turtle.pencolor("green")
            turtle.fd(40)
        elif i == '=':
            turtle.write('月',font=("Arial", 18, "normal"))
            turtle.pencolor("blue")
            turtle.fd(40)
        elif i == '+':
            turtle.write('日',font=("Arial", 18, "normal"))
        else:
            drawDigit(eval(i))

def main():
...(略)
```

```
import turtle, time
```

```
...(略)
```

```
def drawDate(date):
```

```
...(略)
```

```
def main():
```

```
    turtle.setup(800, 350, 200, 200)
```

```
    turtle.penup()
```

```
    turtle.fd(-300)
```

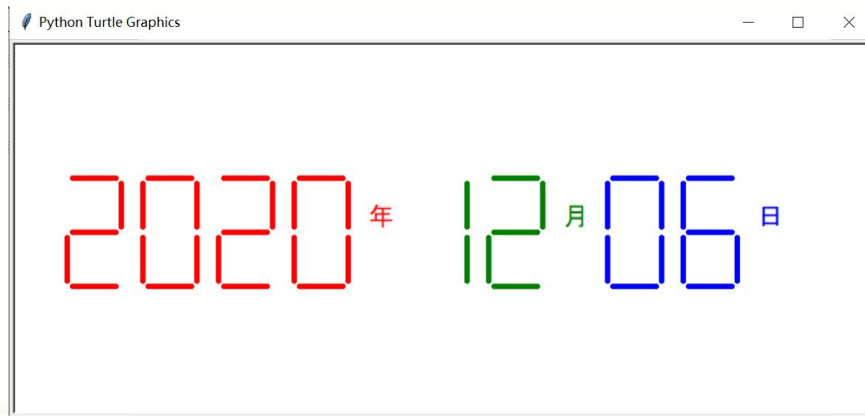
```
    turtle.pensize(5)
```

```
    drawDate(time.strftime('%Y-%m=%d+', time.gmtime()))
```

```
    turtle.hideturtle()
```

```
    turtle.done()
```

```
main()
```





**准备好电脑，与老师一起编码吧！**



# "七段数码管绘制"举一反三

```
import turtle, time
```

```
...(略)
```

```
def drawLine(draw):
```

```
    drawGap()
```

```
    turtle.pendown() if draw else turtle.penup()
```

```
    turtle.fd(40)
```

```
    drawGap()
```

```
    turtle.right(90)
```

```
def drawDigit(digit):
```

```
    drawLine(True) if digit in [2,3,4,5,6,8,9] else drawLine(False)
```

```
    drawLine(True) if digit in [0,1,3,4,5,6,7,8,9] else drawLine(False)
```

```
    drawLine(True) if digit in [0,2,3,5,6,8,9] else drawLine(False)
```

```
    drawLine(True) if digit in [0,2,6,8] else drawLine(False)
```

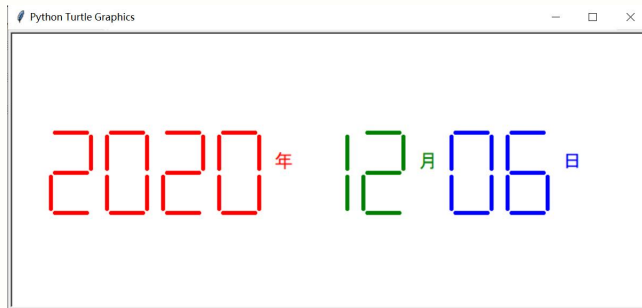
```
    turtle.left(90)
```

```
    drawLine(True) if digit in [0,4,5,6,8,9] else drawLine(False)
```

```
    drawLine(True) if digit in [0,2,3,5,6,7,8,9] else drawLine(False)
```

```
    drawLine(True) if digit in [0,1,2,3,4,7,8,9] else drawLine(False)
```

```
...(略)
```



# 举一反三

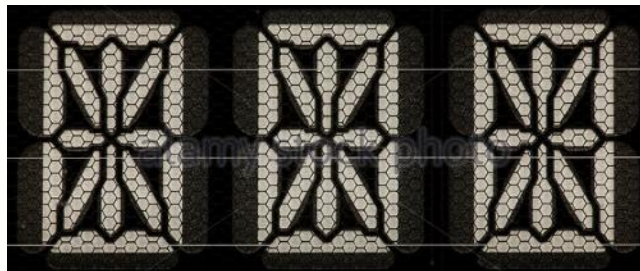
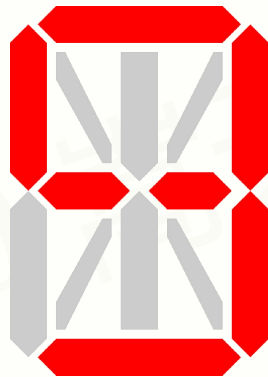
## 理解方法思维

- **模块化思维：确定模块接口，封装功能**
- **规则化思维：抽象过程为规则，计算机自动执行**
- **化繁为简：将大功能变为小功能组合，分而治之**

# 举一反三

## 应用问题的扩展

- 绘制带小数点的七段数码管
- 带刷新的时间倒计时效果
- 绘制高级的数码管



## 5.3 代码复用与函数递归

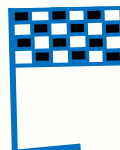
---



# 代码复用与函数递归



- 代码复用与模块化设计
- 函数递归的理解
- 函数递归的调用过程
- 函数递归实例解析





# 代码复用与模块化设计



# 代码复用

## 把代码当成资源进行抽象

- **代码资源化：程序代码是一种用来表达计算的“资源”**
- **代码抽象化：使用函数等方法对代码赋予更高级别的定义**
- **代码复用：同一份代码在需要时可以被重复使用**

# 代码复用

函数 和 对象 是代码复用的两种主要形式

**函数：**将代码命名  
在代码层面建立了初步抽象

**对象：**属性和方法

$\langle a \rangle . \langle b \rangle$  和  $\langle a \rangle . \langle b \rangle ()$

在函数之上再次组织进行抽象



抽象级别

# 模块化设计


## 分而治之

- 通过函数或对象封装将程序划分为模块及模块间的表达
- 具体包括：主程序、子程序和子程序间关系
- 分而治之：一种分而治之、分层抽象、体系化的设计思想

# 模块化设计

## 紧耦合 松耦合

- **紧耦合：两个部分之间交流很多，无法独立存在**
- **松耦合：两个部分之间交流较少，可以独立存在**
- **模块内部紧耦合、模块之间松耦合**



# 函数递归的理解

# 递归的定义

函数定义中调用函数自身的方式



# 递归的定义

## 两个关键特征

- **链条：计算过程存在递归链条**
- **基例：存在一个或多个不需要再次递归的基例**

# 递归的定义

## 类似数学归纳法

- 数学归纳法
  - 证明当 $n$ 取第一个值 $n_0$ 时命题成立
  - 假设当 $n_k$ 时命题成立，证明当 $n=n_{k+1}$ 时命题也成立
- 递归是数学归纳法思维的编程体现





# 函数递归的调用过程

# 递归的实现

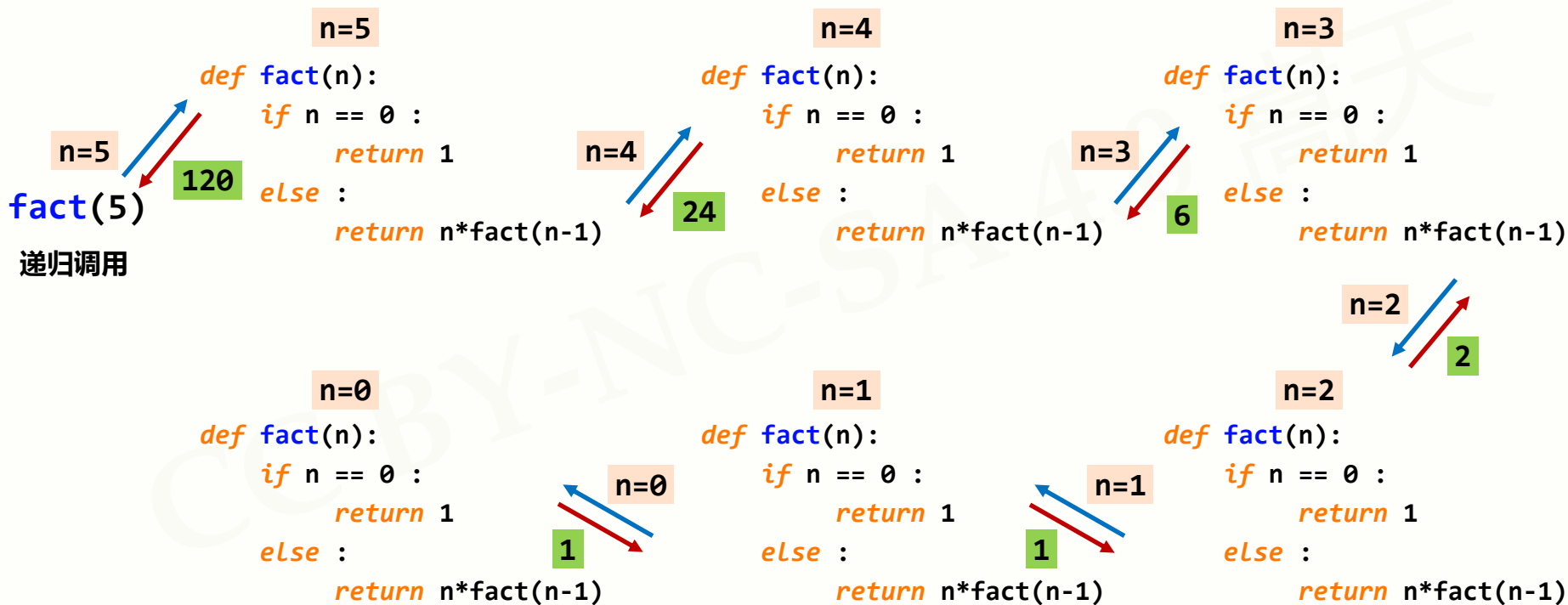
```
def fact(n):  
    if n == 0 :  
        return 1  
    else :  
        return n*fact(n-1)
```

# 递归的实现

## 函数 + 分支语句

- 递归本身是一个函数，需要函数定义方式描述
- 函数内部，采用分支语句对输入参数进行判断
- 基例和链条，分别编写对应代码

# 递归的调用过程





# 函数递归实例解析

# 字符串反转

将字符串s反转后输出

```
>>> s[::-1]
```

- 函数 + 分支结构

```
def rvs(s):
```

```
    if s == "" :
```

```
        return s
```

```
    else :
```

```
        return rvs(s[1:])+s[0]
```

- 递归链条

- 递归基例

# 斐波那契数列

一个经典数列

CC BY-NC-SA 4.0 高天

# 斐波那契数列

$$F(n) = F(n-1) + F(n-2)$$

- 函数 + 分支结构

```
def f(n):
```

```
    if n == 1 or n == 2 :
```

```
        return 1
```

```
    else :
```

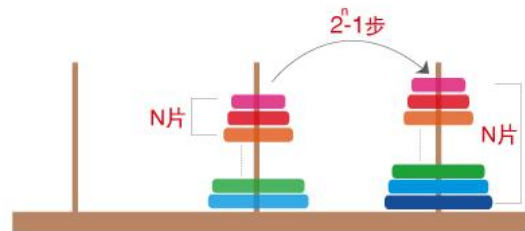
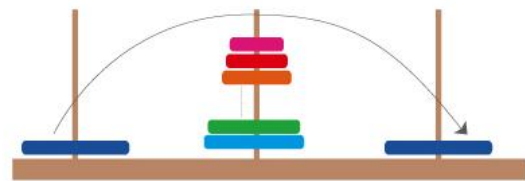
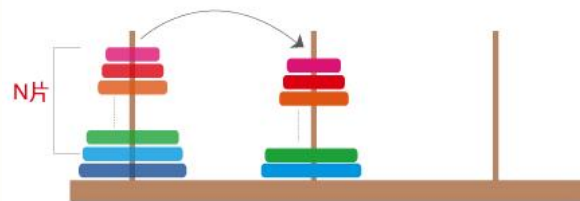
```
        return f(n-1) + f(n-2)
```

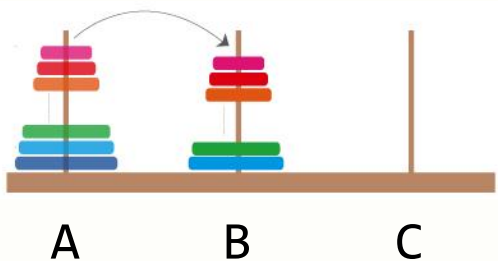
- 递归链条

- 递归基例



# 汉诺塔





# 汉诺塔

- 函数 + 分支结构
- 递归链条
- 递归基例

```
count = 0
```

```
def hanoi(n, src, dst, mid):
```

```
    global count
```

```
    if n == 1 :
```

```
        print("{}: {}-> {}".format(1, src, dst))
```

```
        count += 1
```

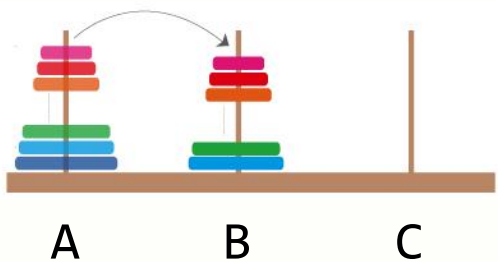
```
    else :
```

```
        hanoi(n-1, src, mid, dst)
```

```
        print("{}: {}-> {}".format(n, src, dst))
```

```
        count += 1
```

```
        hanoi(n-1, mid, dst, src)
```



# 汉诺塔

```
count = 0
def hanoi(n, src, dst, mid):
    ... (略)
hanoi(3, "A", "C", "B")
print(count)
```

>>>

1:A->C

2:A->B

1:C->B

3:A->C

1:B->A

2:B->C

1:A->C

7



## 单元小结

# 代码复用与函数递归

- 模块化设计：松耦合、紧耦合
- 函数递归的2个特征：基例和链条
- 函数递归的实现：函数 + 分支结构

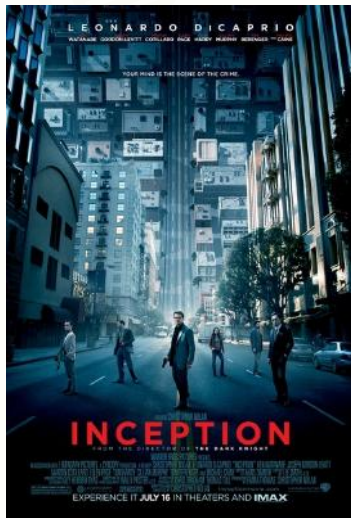


# 如何理解递归呢？

递归很简单，无非就是一个函数调用自己而已...

- 看过《盗梦空间》吗？本质上就是递归...
- 学过数学归纳法吗？本质上就是递归...
- 听过这个故事吗？本质上就是递归...

"从前有座山，山里有座庙，庙里有个老和尚在讲故事..."



## 5.4 模块4: PyInstaller库的使用

---





# PyInstaller库基本介绍



# PyInstaller库概述

将.py源代码转换成无需源代码的可执行文件

.py文件



PyInstaller



- Windows (exe文件)
- Linux
- Mac OS X

# PyInstaller库概述

**PyInstaller库是第三方库**

- **官方网站: <http://www.pyinstaller.org>**
- **第三方库: 使用前需要额外安装**
- **安装第三方库需要使用pip工具**

# PyInstaller库的安装

(cmd命令行) `pip install pyinstaller`

```
命令提示符 - pip install pyinstaller
C:\Users\Tian Song>pip install pyinstaller
Collecting pyinstaller
  Downloading PyInstaller-3.3.1.tar.gz (3.5MB)
    0% |          | 10kB 93kB/s eta 0:0
    0% |          | 20kB 65kB/s eta 0:
    0% |          | 30kB 78kB/s eta 0:
    1% |          | 40kB 49kB/s eta 0:
    1% |          | 51kB 61kB/s eta 0:
    1% |          | 61kB 74kB/s eta 0:
    2% |          | 71kB 86kB/s eta 0:
    2% |          | 81kB 98kB/s eta 0:
    2% |          | 92kB 111kB/s eta 0
    2% |          | 102kB 94kB/s eta 0
    3% |          | 112kB 104kB/s eta
```

```
命令提示符
99% |          |
100% |          | 8.3MB 11kB/s
Installing collected packages: pefile, altgraph, macholib, py
win32, pypiwin32, pyinstaller
  Running setup.py install for pefile ... done
  Running setup.py install for pyinstaller ... done
Successfully installed altgraph-0.15 macholib-1.9 pefile-2017
.11.5 pyinstaller-3.3.1 pypiwin32-223 pywin32-223

C:\Users\Tian Song>
```



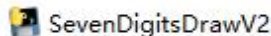
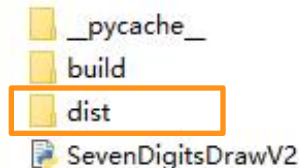
# PyInstaller库使用说明

# 简单的使用

(cmd命令行) **pyinstaller -F <文件名.py>**

命令提示符 - pyinstaller -F SevenDigitsDrawV2.py

```
D:\PYECourse>pyinstaller -F SevenDigitsDrawV2.py
140 INFO: PyInstaller: 3.3.1
140 INFO: Python: 3.6.4
140 INFO: Platform: Windows-10-10.0.15063-SP0
156 INFO: wrote D:\PYECourse\SevenDigitsDrawV2.spec
156 INFO: UPX is not available.
172 INFO: Extending PYTHONPATH with paths
['D:\\PYECourse', 'D:\\PYECourse']
172 INFO: checking Analysis
172 INFO: Building Analysis because out00-Analysis.toc is non
existent
172 INFO: Initializing module dependency graph...
172 INFO: Initializing module graph hooks...
172 INFO: Analyzing base_library.zip ...
```



# PyInstaller库常用参数

参数	描述
-h	查看帮助
--clean	清理打包过程中的临时文件
-D, --onedir	默认值, 生成dist文件夹
-F, --onefile	在dist文件夹中只生成独立的打包文件
-i <图标文件名.ico>	指定打包程序使用的图标(icon)文件

# 使用举例

`pyinstaller -i curve.ico -F SevenDigitsDrawV2.py`



+



SevenDigitsDrawV2

=



SevenDigitsDrawV2

# 5.5 实例8: 科赫雪花小包裹

---







# "科赫雪花小包裹"问题分析

# 科赫雪花

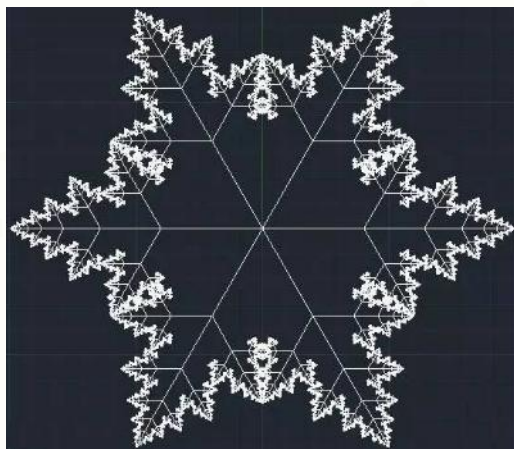
## 高大上的分形几何



- 分形几何是一种迭代的几何图形，广泛存在于自然界中

# 科赫雪花

科赫曲线，也叫雪花曲线



# 科赫雪花绘制

## 用Python绘制科赫曲线

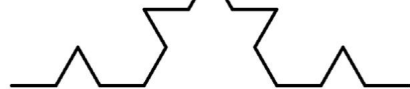
0阶科赫曲线



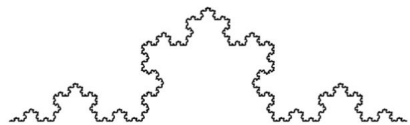
1阶科赫曲线



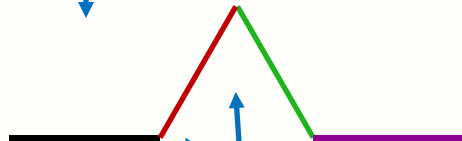
2阶科赫曲线



5阶科赫曲线



取 $1/3$ 长



60度

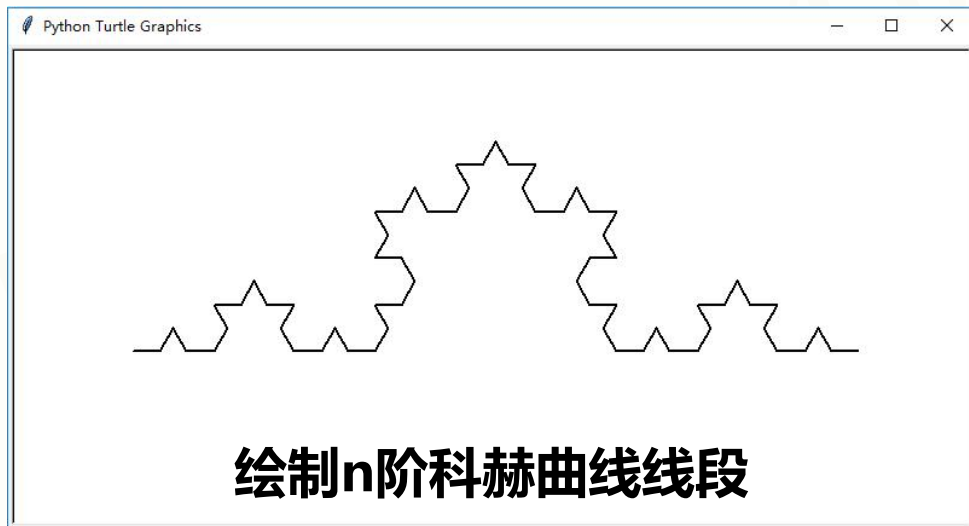
每分隔一次为一阶



# "科赫雪花小包裹"实例讲解(上)

# 科赫雪花小包裹(上)

## 科赫曲线的绘制



# 科赫雪花小包裹(上)

```
#KochDrawV1.py
```

```
import turtle
```

```
def koch(size, n):
```

```
    if n == 0:
```

```
        turtle.fd(size)
```

```
    else:
```

```
        for angle in [0, 60, -120, 60]:
```

```
            turtle.left(angle)
```

```
            koch(size/3, n-1)
```

## 科赫曲线的绘制

- 递归思想：函数+分支
- 递归链条：线段的组合
- 递归基例：初始线段

# 科赫雪花小包裹(上)

```
#KochDrawV1.py
import turtle
def koch(size, n):
    if n == 0:
        turtle.fd(size)
    else:
        for angle in [0, 60, -120, 60]:
            turtle.left(angle)
            koch(size/3, n-1)
def main():
    turtle.setup(800,400)
    turtle.penup()
    turtle.goto(-300, -50)
    turtle.pendown()
    turtle.pensize(2)
    koch(600, 3)      # 3阶科赫曲线, 阶数
    turtle.hideturtle()
main()
```

## 科赫曲线的绘制



# 科赫雪花小包裹(上)

```
#KochDrawV2.py
import turtle
def koch(size, n):
    ...(略)
def main():
    turtle.setup(600,600)
    turtle.penup()
    turtle.goto(-200, 100)
    turtle.pendown()
    turtle.pensize(2)
    level = 3          # 3阶科赫雪花, 阶数
    koch(400, level)
    turtle.right(120)
    koch(400, level)
    turtle.right(120)
    koch(400, level)
    turtle.hideturtle()
main()
```

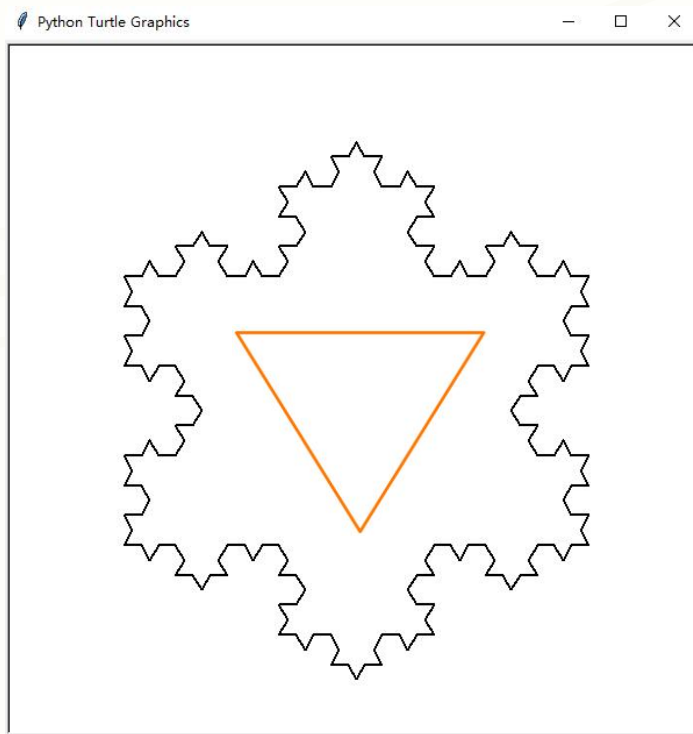
科赫曲线的绘制



科赫雪花的绘制

# 科赫雪花小包裹(上)

```
#KochDrawV2.py
import turtle
def koch(size, n):
    ... (略)
def main():
    turtle.setup(600,600)
    turtle.penup()
    turtle.goto(-200, 100)
    turtle.pendown()
    turtle.pensize(2)
    level = 3      # 3阶科赫雪花, 阶数
    koch(400, level)
    turtle.right(120)
    koch(400, level)
    turtle.right(120)
    koch(400, level)
    turtle.hideturtle()
main()
```



**准备好电脑，与老师一起编码吧！**



# "科赫雪花小包裹"实例讲解(下)



# 科赫雪花小包裹(下)

打包才能上路...

```
pyinstaller -i curve.ico -F KochDrawV2.py
```

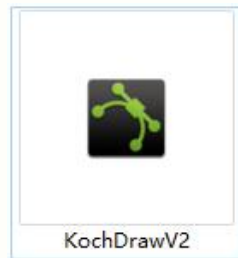


+



KochDrawV2

=

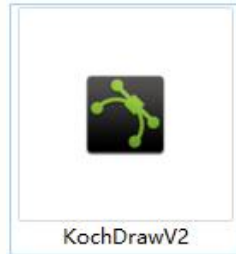


- 对编写后的科赫雪花代码进行打包处理

# 科赫雪花小包裹(下)

```
命令提示符 - pyinstaller -i curve.ico -F KochDrawV2.py

D:\PYECourse>pyinstaller -i curve.ico -F KochDrawV2.py
62 INFO: PyInstaller: 3.3.1
62 INFO: Python: 3.6.4
62 INFO: Platform: Windows-10-10.0.15063-SP0
62 INFO: wrote D:\PYECourse\KochDrawV2.spec
62 INFO: UPX is not available.
62 INFO: Extending PYTHONPATH with paths
['D:\PYECourse', 'D:\PYECourse']
62 INFO: checking Analysis
62 INFO: Building Analysis because out00-Analysis.toc is non
existent
62 INFO: Initializing module dependency graph...
62 INFO: Initializing module graph hooks...
62 INFO: Analyzing base_library.zip ...
```



**准备好电脑，与老师一起编码吧！**



"科赫雪花小包裹"举一反三



```
#KochDrawV2.py
```

```
import turtle
```

```
def koch(size, n):
```

```
    if n == 0:
```

```
        turtle.fd(size)
```

```
    else:
```

```
        for angle in [0, 60, -120, 60]:
```

```
            turtle.left(angle)
```

```
            koch(size/3, n-1)
```

```
def main():
```

```
    turtle.setup(600,600)
```

```
    turtle.penup()
```

```
    turtle.goto(-200, 100)
```

```
    turtle.pendown()
```

```
    turtle.pensize(2)
```

```
    level = 3          # 3阶科赫雪花, 阶数
```

```
    koch(400, level)
```

```
    turtle.right(120)
```

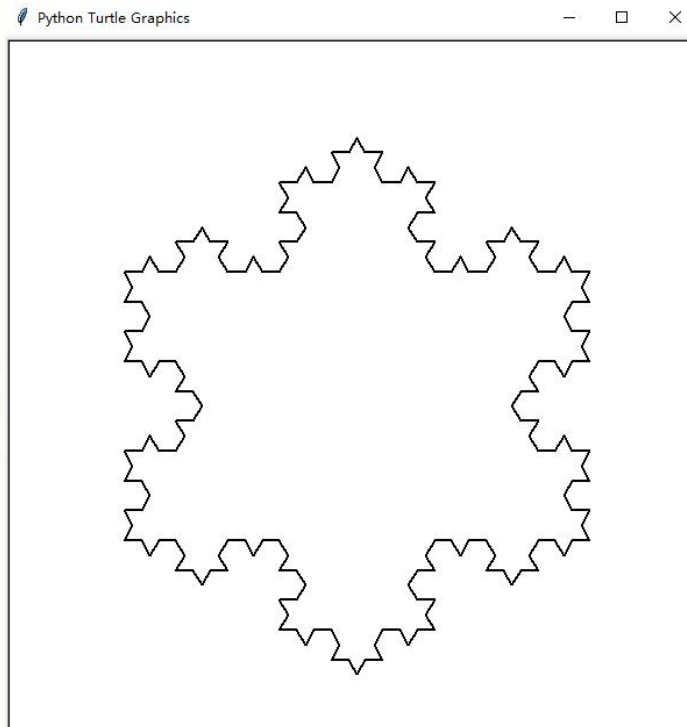
```
    koch(400, level)
```

```
    turtle.right(120)
```

```
    koch(400, level)
```

```
    turtle.hideturtle()
```

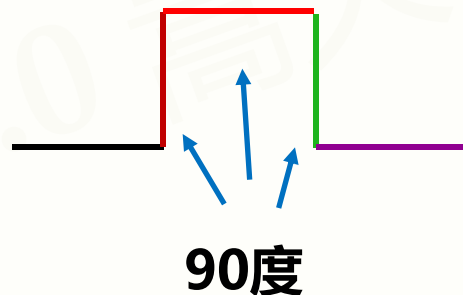
```
main()
```



# 举一反三

## 绘制条件的扩展

- 修改分形几何绘制阶数
- 修改科赫曲线的基本定义及旋转角度
- 修改绘制科赫雪花的基础框架图形



# 举一反三

## 分形几何千千万

- 康托尔集、谢尔宾斯基三角形、门格海绵...
- 龙形曲线、空间填充曲线、科赫曲线...
- 函数递归的深入应用...



