

程序设计

Week 8

基本算法—迭代

一组指令以循环的方式重复执行

将复杂的问题化为简单的问题

使用旧值推新值

基本算法—递归

把问题进行分解

自己调用自己

递归公式

递归出口

基本算法

问题：有人想知道一年内一对兔子可繁殖成多少对，便筑了一道围墙把一对兔子关在里面。已知一对兔子每一个月可以生一对小兔子，而一对兔子出生后两个月就开始生小兔子。假如一年内没有发生死亡，则一对兔子一年内能繁殖成多少对？

基本算法—递归

斐波那契数列：

月数	1	2	3	4	5	6	7	8
兔子对数	1	1	2	3	5	8	13	21



$$Fibonacci(n) = \begin{cases} 1 & n = 1, 2 \\ Fibonacci(n-1) + Fibonacci(n-2) & n > 2 \end{cases}$$

基本算法—递归与迭代的比较

相同点：

递归和迭代都使用循环

不同点：

递归需重复调用自身函数

迭代需重复执行某段代码

基本算法—递归与迭代的比较

例子：求阶乘

迭代：

$$Factorial(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1) \times (n-2) \dots \times 2 \times 1 & \text{if } n > 0 \end{cases}$$

递归：

$$Factorial(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times Factorial(n-1) & \text{if } n > 0 \end{cases}$$

基本算法—递归与迭代的比较

迭代算法：Factorial(n)

F=1, i=1

while(i<=n)

{

 F=F*i

 i=i+1

}

return F

递归算法：Factorial(n)

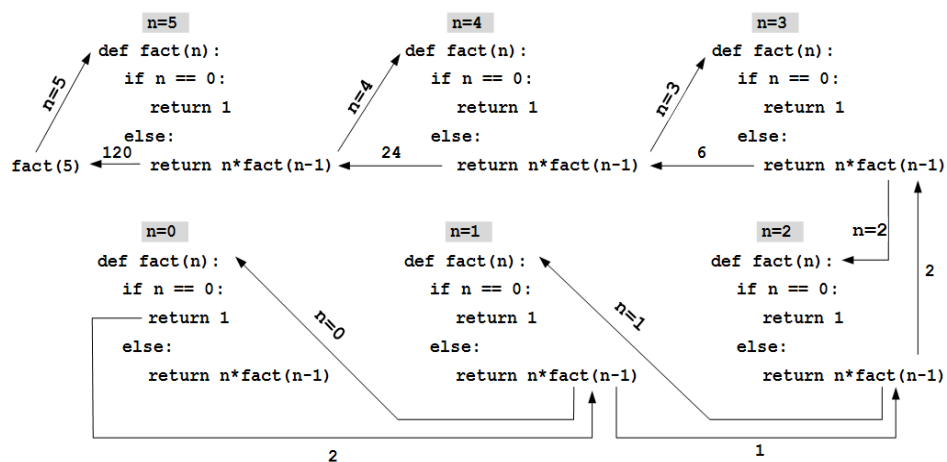
if (n=0)

 return 1

else

 return n*Factorial(n-1)

阶乘计算




递归

微实例5.32：字符串反转。

对于用户输入的字符串s，输出反转后的字符串。

这个问题的基本思想是把字符串看作一个递归对象。

```
1 def reverse(s):  
2     return reverse(s[1:]) + s[0]
```



代码的复用和模块化设计

代码复用和模块化设计

当程序的长度在百行以上，如果不划分模块就算是最好的程序员也很难理解程序含义程序的可读性就已经很糟糕了。

解决这一问题的最好方法是将一个程序分割成短小的程序段，每一段程序完成一个小的功能。

无论面向过程和面向对象编程，对程序合理划分功能模块并基于模块设计程序是一种常用方法，被称为“模块化设计”。

代码复用和模块化设计

模块化设计一般有两个基本要求：

紧耦合：尽可能合理划分功能块，功能块内部耦合紧密；

松耦合：模块间关系尽可能简单，功能块之间耦合度低。

使用函数只是模块化设计的必要非充分条件，根据计算需求合理划分函数十分重要。一般来说，完成特定功能或被经常复用的一组语句应该采用函数来封装，并尽可能减少函数间参数和返回值的数量。

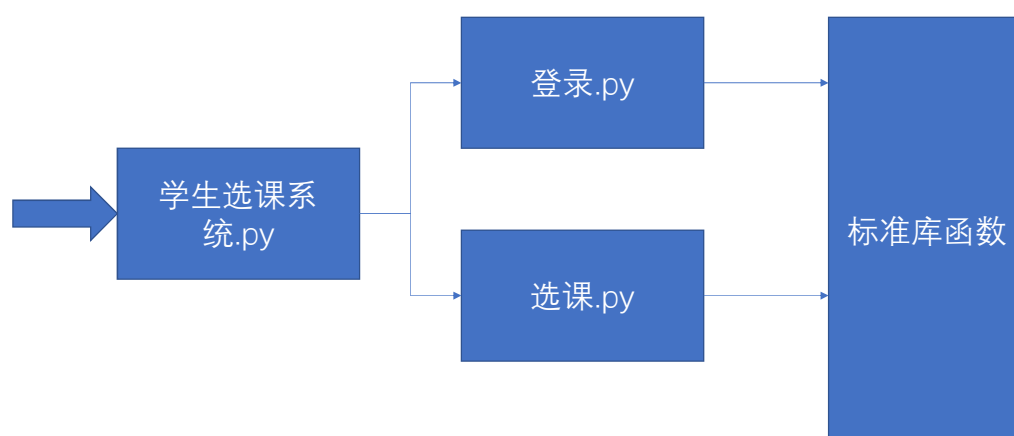
代码复用和模块化设计

主要有如下作用：

代码重用：我们知道当一段代码需要用到两次的时候，我们就需要写一个函数了这是一个道理。

避免变量名的冲突：每个模块都将变量名封装进了自己包含的软件包，这可以避免变量名的冲突。除非使用精确导入。

代码复用和模块化设计



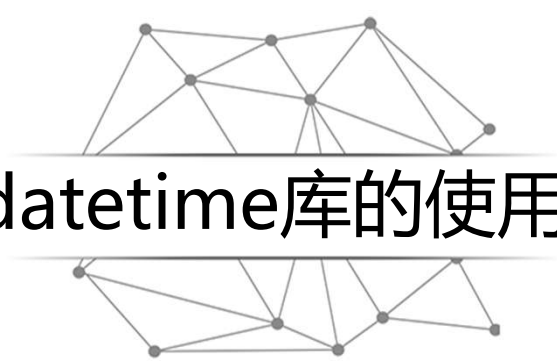
代码复用和模块化设计

导入整个模块：

```
import XXX
```

导入特定属性：

```
from XXX import XXX
```



datetime库的使用

面向对象

面向对象的高级语言：

Object-Oriented Programing (OOP)

OOP使用类作为程序的基本形态

类中包括数据和对数据的操作

对象是类的实例

编程语言——高级语言

封装：

封装是指把对象的属性和操作结合在一起，构成一个独立体。

继承：

继承 (Inheritance) 是指子类可以拥有父类的属性和行为，这里的类 (Class) 就是对象继承提高了软件代码的复用性。

编程语言——高级语言

多态性：

多态性（Polymorphism）是指对象可以具有不同的行为。多态性机制不仅为软件的结构设计提供了灵活性，还减少了信息冗余，提高了软件的可扩展性。



datetime库概述

一个处理时间的标准函数库datetime；

可以从系统中获得时间；

提供一系列由简单到复杂的时间处理方法；

以格林威治时间为基础

包含datetime.MINYEAR和datetime.MAXYEAR两个常量。

datetime库概述

datetime库以类的方式提供多种日期和时间表达方式：

- datetime.date：日期表示类，可以表示年、月、日等
- datetime.time：时间表示类，可以表示小时、分钟、秒、毫秒等
- datetime.datetime：日期和时间表示的类，功能覆盖date和time类
- datetime.timedelta：时间间隔有关的类
- datetime.tzinfo：与时区有关的信息表示类

datetime

使用datetime.now()获得当前日期和时间对象，使用方法如下：

datetime.now()

作用：返回一个datetime类型，表示当前的日期和时间，精确到微秒。

```
>>> from datetime import datetime
>>> today = datetime.now()
>>> today
datetime.datetime(2016, 9, 20, 10, 29, 43, 928549)
```

datetime

使用`datetime.utcnow()`获得当前日期和时间对应的UTC（世界标准时间）时间对象，使用方法如下：

`datetime.utcnow()`

作用：返回`datetime`类型，表示当前日期和时间的UTC表示，精确到微秒。

```
>>> today = datetime.utcnow()
>>> today
datetime.datetime(2016, 9, 20, 2, 35, 1, 427954)
```

datetime

`datetime.now()`和`datetime.utcnow()`都返回一个`datetime`类型的对象，也可以直接使用`datetime()`构造一个日期和时间对象，使用方法如下：

`datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0)`

作用：返回一个`datetime`类型，表示指定的日期和时间，可以精确到微秒。

datetime库解析

调用datetime()函数直接创建一个datetime对象，表示2016年9月16日22:33，32秒7微秒，执行结果如下：

```
>>> someday = datetime(2016,9,16,22,33,32,7)
>>> someday
datetime.datetime(2016, 9, 16, 22, 33, 32, 7)
```

程序已经有了一个datetime对象，进一步可以利用这个对象的属性显示时间，为了区别datetime库名，采用上例中的someday代替生成的datetime对象

datetime库解析

属性	描述
someday.min	固定返回datetime的最小时间对象，datetime(1,1,1,0,0)
someday.max	固定返回datetime的最大时间对象，datetime(9999, 12, 31, 23, 59, 59, 999999)
someday.year	返回someday包含的年份
someday.month	返回someday包含的月份
someday.day	返回someday包含的日期
someday.hour	返回someday包含的小时
someday.minute	返回someday包含的分钟
someday.second	返回someday包含的秒钟
someday.microsecond	返回someday包含的微秒值

datetime库解析

datetime对象有3个常用的时间格式化方法，如表所示

属性	描述
<code>someday.isoformat()</code>	采用ISO 8601标准显示时间
<code>someday.isoweekday()</code>	根据日期计算星期后返回1-7,对应星期一到星期日
<code>someday.strftime(format)</code>	根据格式化字符串format进行格式显示的方法

isoformat()和isoweekday()方法的使用如下：

```
>>> someday = datetime(2016,9,16,22,33,32,7)
>>> someday.isoformat()
'2016-09-16T22:33:32.000007'
>>> someday.isoweekday()
5
```

datetime库解析

strftime()方法是时间格式化最有效的方法，几乎可以以任何通用格式输出时间

```
>>> someday.strftime("%Y-%m-%d %H:%M:%S")
'2016-09-16 22:33:32'
```

datetime库解析

格式化字符串	日期/时间	值范围和实例
%Y	年份	0001~9999, 例如: 1900
%m	月份	01~12, 例如: 10
%B	月名	January~December, 例如: April
%b	月名缩写	Jan~Dec, 例如: Apr
%d	日期	01 ~ 31, 例如: 25
%A	星期	Monday~Sunday, 例如: Wednesday
%a	星期缩写	Mon~Sun, 例如: Wed
%H	小时 (24h制)	00 ~ 23, 例如: 12
%I	小时 (12h制)	01 ~ 12, 例如: 7
%p	上/下午	AM, PM, 例如: PM
%M	分钟	00 ~ 59, 例如: 26
%S	秒	00 ~ 59, 例如: 26

datetime库解析

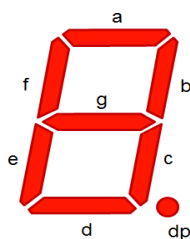
strftime()格式化字符串的数字左侧会自动补零, 上述格式也可以与print()的格式化函数一起使用

```
>>>from datetime import datetime
>>>now = datetime.now()
>>>now.strftime("%Y-%m-%d")
'2016-09-20'
>>>now.strftime("%A, %d. %B %Y %I:%M%p")
'Tuesday, 20. September 2016 01:53PM'
>>>print("今天是{0:%Y}年{0:%m}月{0:%d}日".format(now))
今天是2016年09月20日
```


七段数码管绘制

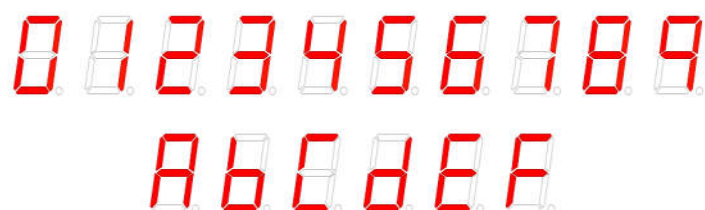
七段数码管绘制

七段数码管 (seven-segment indicator) 由7段数码管拼接而成，每段有亮或不亮两种情况，改进型的七段数码管还包括一个小数点位置，如图所示。



七段数码管绘制

七段数码管能形成 $2^7=128$ 种不同状态，其中部分状态能够显示易于人们理解的数字或字母含义，因此被广泛使用。图5.5给出了十六进制中16个字符的七段数码管表示。



七段数码管绘制

每个0到9的数字都有相同的七段数码管样式，因此，可以通过设计函数复用数字的绘制过程。进一步，每个七段数码管包括7个数码管样式，除了数码管位置不同外，绘制风格一致，也可以通过函数复用单个数码段的绘制过程。

七段数码管绘制

实例代码7.1

e7.1DrawSevenSegDisplay.py

```

1  #e7.1DrawSevenSegDisplay.py
2  import turtle, datetime
3  def drawLine(draw): #绘制单段数码管
4      turtle.pendown() if draw else turtle.penup()
5      turtle.fd(40)
6      turtle.right(90)
7  def drawDigit(d): #根据数字绘制七段数码管
8      drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
9      drawLine(True) if d in [0,1,3,4,5,6,7,8,9] else drawLine(False)
10     drawLine(True) if d in [0,2,3,5,6,8,9] else drawLine(False)
11     drawLine(True) if d in [0,2,6,8] else drawLine(False)
12     turtle.left(90)
13     drawLine(True) if d in [0,4,5,6,8,9] else drawLine(False)
14     drawLine(True) if d in [0,2,3,5,6,7,8,9] else drawLine(False)
15     drawLine(True) if d in [0,1,2,3,4,7,8,9] else drawLine(False)

```

七段数码管绘制

实例代码7.1

e7.1DrawSevenSegDisplay.py

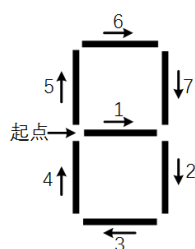
```

16  turtle.left(180)
17  turtle.penup()
18  turtle.fd(20)
19  def drawDate(date): #获得要输出的数字
20      for i in date:
21          drawDigit(eval(i)) #注意：通过eval()函数将数字变为整数
22  def main():
23      turtle.setup(800, 350, 200, 200)
24      turtle.penup()
25      turtle.fd(-300)
26      turtle.pensize(5)
27      drawDate(datetime.datetime.now().strftime('%Y%m%d'))
28      turtle.hideturtle()
29  main()

```

七段数码管绘制

实例代码定义了drawDigit()函数，该函数根据输入的数字d绘制七段数码管，结合七段数码管结构，每个数码管的绘制采用图所示顺序。

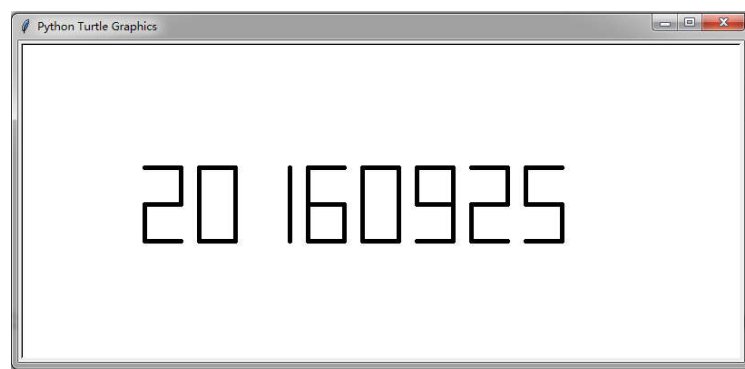


七段数码管绘制

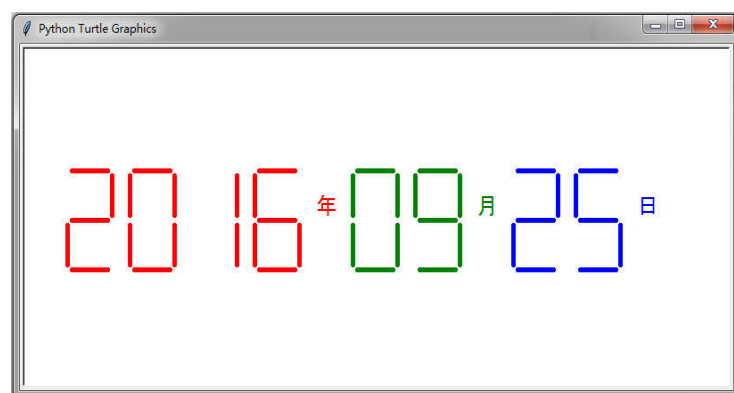
绘制起点在数码管中部左侧，无论每段数码管是否被绘制出来，turtle画笔都按顺序“画完”所有7个数码管。对于给定数字d，哪个数码段被绘制出来采用if...else...语句判断。

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

七段数码管绘制



七段数码管绘制



实例代码7.2 e7.2DrawSevenSegDisplay.py

```

1  #e7.2DrawSevenSegDisplay.py
2  import turtle, datetime
3  def drawGap(): #绘制数码管间隔
4      turtle.penup()
5      turtle.fd(5)
6  def drawLine(draw): #绘制单段数码管
7      drawGap()
8      turtle.pendown() if draw else turtle.penup()
9      turtle.fd(40)
10     drawGap()
11     turtle.right(90)
12 def drawDigit(d): #根据数字绘制七段数码管
13     drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
14     drawLine(True) if d in [0,1,3,4,5,6,7,8,9] else drawLine(False)
15     drawLine(True) if d in [0,2,3,5,6,8,9] else drawLine(False)
16     drawLine(True) if d in [0,2,6,8] else drawLine(False)
17     turtle.left(90)
18     drawLine(True) if d in [0,4,5,6,8,9] else drawLine(False)
19     drawLine(True) if d in [0,2,3,5,6,7,8,9] else drawLine(False)
20     drawLine(True) if d in [0,1,2,3,4,7,8,9] else drawLine(False)
21     turtle.left(180)
22     turtle.penup()
23     turtle.fd(20)
24


```

实例代码7.2 e7.2DrawSevenSegDisplay.py

```

25 def drawDate(date):
26     turtle.pencolor("red")
27     for i in date:
28         if i == '-':
29             turtle.write('年',font=("Arial", 18, "normal"))
30             turtle.pencolor("green")
31             turtle.fd(40)
32         elif i == '=':
33             turtle.write('月',font=("Arial", 18, "normal"))
34             turtle.pencolor("blue")
35             turtle.fd(40)
36         elif i == '+':
37             turtle.write('日',font=("Arial", 18, "normal"))
38         else:
39             drawDigit(eval(i))
40 def main():
41     turtle.setup(800, 350, 200, 200)
42     turtle.penup()
43     turtle.fd(-350)
44     turtle.pensize(5)
45     drawDate(datetime.datetime.now().strftime('%Y-%m=%d+'))
46     turtle.hideturtle()
47 main()

```



科赫曲线绘制

科赫曲线绘制

自然界有很多图形很规则，符合一定的数学规律，例如，蜜蜂蜂窝是天然的等边六角形等。科赫(Koch)曲线在众多经典数学曲线中非常著名，由瑞典数学家冯·科赫(H·V·Koch)于1904年提出，由于其形状类似雪花，也被称为雪花曲线。

科赫曲线绘制

科赫曲线的基本概念和绘制方法如下：

正整数 n 代表科赫曲线的阶数，表示生成科赫曲线过程的操作次数。科赫曲线初始化阶数为0，表示一个长度为 L 的直线。对于直线 L ，将其等分为三段，中间一段用边长为 $L/3$ 的等边三角形的两个边替代，得到1阶科赫曲线，它包含四条线段。进一步对每条线段重复同样的操作后得到2阶科赫曲线。继续重复同样的操作 n 次可以得到 n 阶科赫曲线。

科赫曲线绘制

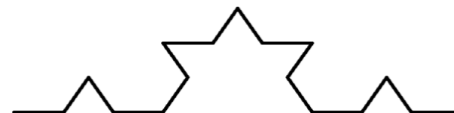
0阶科赫曲线



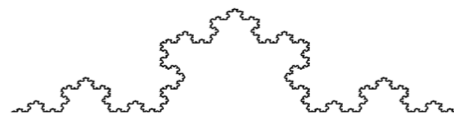
1阶科赫曲线



2阶科赫曲线



5阶科赫曲线



科赫曲线绘制

科赫曲线属于分形几何分支，它的绘制过程体现了递归思想，绘制过程代码。

实例代码8.1

e8.1DrawKoch.py

```

1 #e8.1DrawKoch.py
2 import turtle
3 def koch(size, n):
4     if n == 0:
5         turtle.fd(size)
6     else:
7         for angle in [0, 60, -120, 60]:
8             turtle.left(angle)
9             koch(size/3, n-1)
10 def main():
11     turtle.setup(800,400)
12     turtle.speed(0) #控制绘制速度
13     turtle.penup()
14     turtle.goto(-300, -50)
15     turtle.pendown()
16     turtle.pensize(2)
17     koch(600,3) # 0阶科赫曲线长度, 阶数
18     turtle.hideturtle()
19 main()

```

科赫曲线绘制

科赫曲线的雪花效果

实例代码8.2

e8.2DrawKoch.py

```

1 #e8.2DrawKoch.py
2 import turtle
3 def koch(size, n):
4     if n == 0:
5         turtle.fd(size)
6     else:
7         for angle in [0, 60, -120, 60]:
8             turtle.left(angle)
9             koch(size/3, n-1)
10 def main():
11     turtle.setup(600,600)
12     turtle.speed(0)
13     turtle.penup()
14     turtle.goto(-200, 100)
15     turtle.pendown()
16     turtle.pensize(2)
17     level = 5
18     koch(400,level)
19     turtle.right(120)
20     koch(400,level)
21     turtle.right(120)
22     koch(400,level)
23     turtle.hideturtle()
24 main()

```



Python内置函数

Python内置函数

Python解释器提供了68个内置函数，其中，前36个已经将结果，需要掌握。

<code>abs()</code>	<code>id()</code>	<code>round()</code>	<code>compile()</code>	<code>locals()</code>
<code>all()</code>	<code>input()</code>	<code>set()</code>	<code>dir()</code>	<code>map()</code>
<code>any()</code>	<code>int()</code>	<code>sorted()</code>	<code>exec()</code>	<code>memoryview()</code>
<code>ascii()</code>	<code>len()</code>	<code>str()</code>	<code>enumerate()</code>	<code>next()</code>
<code>bin()</code>	<code>list()</code>	<code>tuple()</code>	<code>filter()</code>	<code>object()</code>
<code>bool()</code>	<code>max()</code>	<code>type()</code>	<code>format()</code>	<code>property()</code>
<code>chr()</code>	<code>min()</code>	<code>zip()</code>	<code>frozenset()</code>	<code>repr()</code>
<code>complex()</code>	<code>oct()</code>		<code>getattr()</code>	<code>setattr()</code>
<code>dict()</code>	<code>open()</code>		<code>globals()</code>	<code>slice()</code>
<code>divmod()</code>	<code>ord()</code>	<code>bytes()</code>	<code>hasattr()</code>	<code>staticmethod()</code>
<code>eval()</code>	<code>pow()</code>	<code>delattr()</code>	<code>help()</code>	<code>sum()</code>
<code>float()</code>	<code>print()</code>	<code>bytearray()</code>	<code>isinstance()</code>	<code>super()</code>
<code>hash()</code>	<code>range()</code>	<code>callable()</code>	<code>issubclass()</code>	<code>vars()</code>
<code>hex()</code>	<code>reversed()</code>	<code>classmethod()</code>	<code>iter()</code>	<code>import()</code>