

# Python语言程序设计

## Design and Programming of The Python Language

---

主讲教师：张小东

联系方式：[z\\_xiaodong7134@163.com](mailto:z_xiaodong7134@163.com)

答疑地点：哈尔滨工业大学(威海)研学楼院423

# 第2章 控制结构与异常处理

## 主要内容

- 顺序与条件分支控制结构
- 循环程序设计
- 异常处理
- 函数
- 函数式编程
- 结构化程序设计思想

## ===顺序结构===

【定义】程序按照语句的书写次序自上而下顺序执行

【例2-1】输入圆的半径，计算圆的周长与面积

pi为math中定义的常量

```
from math import *  
r=float(input("输入圆半径: "))  
c=2*pi*r  
s=pi*r**2  
print("圆周长为: %.2f"%c,";圆的面积为: %.2f"%s)
```

## ===分支控制结构===

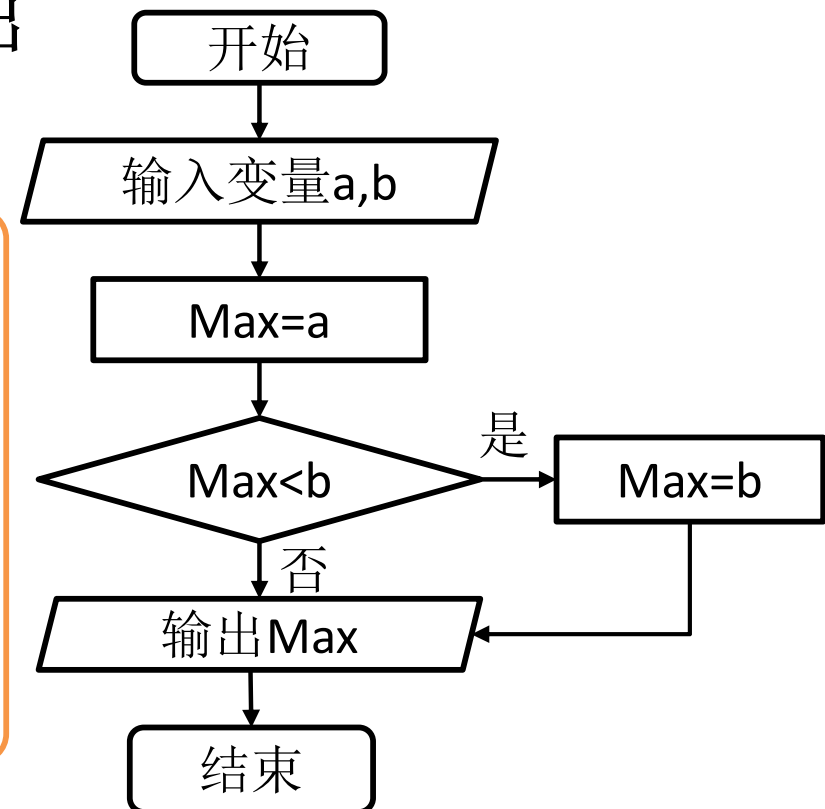
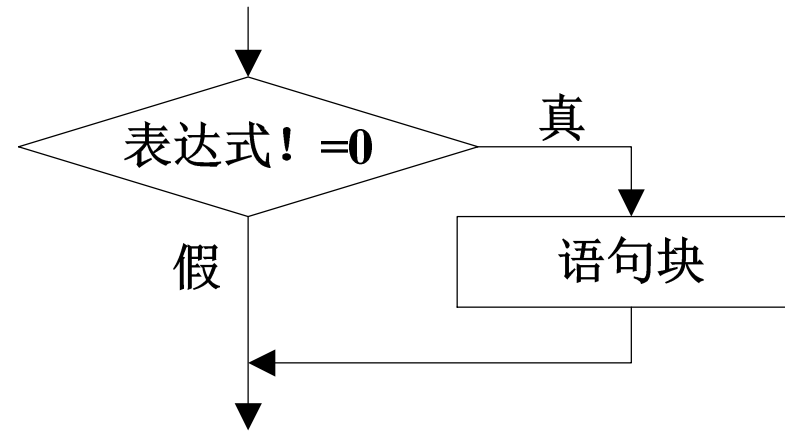
## ◆ 一路分支结构

【语句格式】

**if<条件表达式>:**  
**<语句块>**

【例2-2】从键盘输入两个数，输出它们的最大值

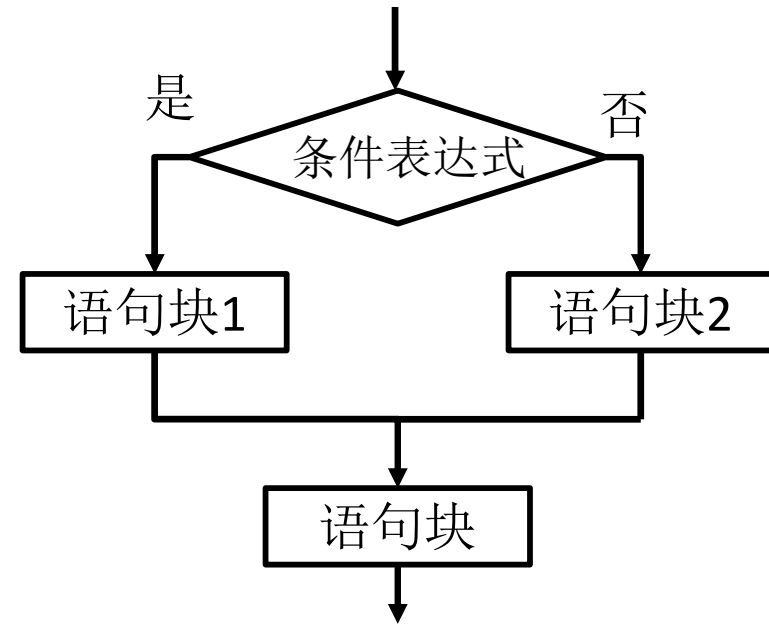
```
a=int(input("输入第一个数: "))
b=int(input("输入第二个数: "))
Max=a
if Max<b:
    Max=b
print("max=",Max)
```



## ===分支控制结构===

## ◆ 二路分支结构

【语句格式】

**if<条件表达式>:****<语句块1>****else:****<语句块2>**

【例2-3】从键盘输入两个数，输出它们的最大值。

```
a=int(input("输入第一个数: "))
b=int(input("输入第二个数: "))
if a<b:
    print("max=",b)
else:
    print("max=",a)
```

## ===分支控制结构===

## ◆ 二路分支结构

【例2-4】编写程序，解一元二次方程 $a*x^2+bx+c=0$ 。用户输入系数 $a, b, c$ ，如果有实根计算实根并显示，如果没有，显示“没有实根”

开方函数

```
from math import *
a=float(input('输入a:'))
b=float(input('输入b:'))
c=float(input('输入c:'))
beta=b*b-4*a*c
if beta>=0:
    x1=(-b+sqrt(beta))/(2*a)
    x2=(-b-sqrt(beta))/2/a
    print("x1=",x1,";x2=%.2f"%x2)
else:
    print("没有实根")
```

## ===分支控制结构===

## ◆ 多路分支结构

## 【语句格式】

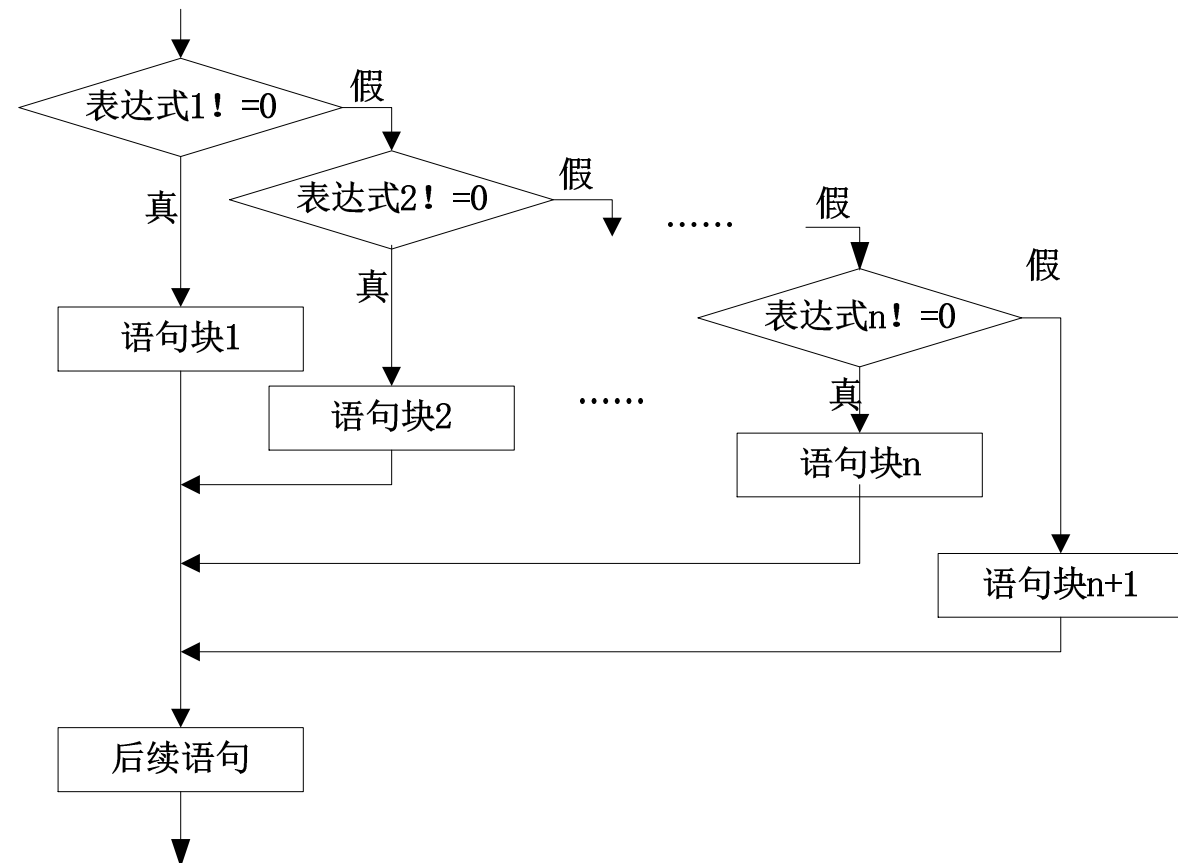
**if<条件1>:**  
    <语句块1>

**elif<条件2>:**  
    <语句块2>

.....

**elif<条件n>:**  
    <语句块n>

**else:**  
    <语句块n+1>



## ===分支控制结构.多路分支结构===

【例2-5】将百分制转换为5分制。转换规则为：**90分及以上者转为5分，80分及以上者转为4分，70分及以上者为3分，60分及以上者2分，不及格为1分**

```
a=int(input("请输入百分制成绩:"))
b=0
if(a<0 or a>100):
    b=-1
elif(a>=90):
    b=5
elif(a>=80):
    b=4
```



分支控制结构有几种基本形式?都是什么?

```
elif(a>=70):
    b=3
elif(a>=60):
    b=2
else:
    b=1
if(b== -1):
    print('输入错误')
else:
    print(b)
```



## ===分支控制结构.多路分支结构===

【例2-6】编写算法对输入的一个整数，判断它能否被3，5，7整除，并输出以下信息之一：

- (1) 能同时被3，5，7整除；
- (2) 能被其中两数（要指出哪两个）整除；
- (3) 能被其中一个数（要指出哪一个）整除；
- (4) 不能被3，5，7任一个整除。

```
n=int(input("Please enter a number:"))
k=(n % 3==0)+(n % 5==0)+(n % 7==0)
if k==3:
    print("All!")
elif k==2:
    print("two!")
elif k==1:
    print("one!")
else:
    print("none!")
```

如何区分被哪几个整除了？

算法分析：

- (1) k的范围是0~3可以表示四种情况。
- (2) 题目要求：八种情况！所以k的范围应该是0~7。

## ===分支控制结构.多路分支结构===

【例2-6】编写算法对输入的一个整数，判断它能否被3，5，7整除。

➤ 算法改进：

$$k=(n \% 3==0)+(n \% 5==0)*2+(n \% 7==0)*4$$

```
n=int(input("Please enter a number:"))
k=(n % 3==0)+(n % 5==0)*2+(n % 7==0)*4
if k==7:
    print("All!")
elif k==6:
    print("5 and 7!")
elif k==5:
    print("3 and 7!")
```

```
elif k==4:
    print("7!")
elif k==3:
    print("3 and 5!")
elif k==2:
    print("5!")
elif k==1:
    print("3!")
else:
    print("none!")
```

## ===分支控制结构===

## ◆ 分支嵌套

【例2-7】改造例2-4，使其可包含二次方程、一次方程或者构不成方程的判定并能进行求解。

## ➤ 问题分析

(1) 当 $a=0, b=0$ 则构不成方程

(2) 当 $a=0, b \neq 0$ 则构成一次方程，结果为 $x=-c/b$

(3) 当 $a \neq 0$ ，计算 $\text{beta}=b*b-4*a*c$ ，若 $\text{beta} \geq 0$ ，则可求出方程的两个实根；若 $\text{beta} < 0$ ，则 $\text{deta}=\text{sqrt}(-\text{beta})$

实部： $\text{real}=-b/(2*a)$ ； 虚部： $\text{imag}=\text{deta}/(2*a)$

输出方程有复根： $\text{complex}(\text{real}, \text{imag})$ 和 $\text{complex}(\text{real}, -\text{imag})$

```

from math import *
a=float(input('输入a:'))
b=float(input('输入b:'))
c=float(input('输入c:'))
if(a==0):
    if(b==0):
        print('输入a=0,b=0不能构成方程! ')
    else:
        x=-c/b
        print('输入为一元一次方程, 根为: ',x)
else:
    beta=b*b-4*a*c
    if(beta>=0):
        x1=-b+sqrt(beta)
        x2=-b-sqrt(beta)
        print('x1=%0.2f'%x1,'x2=%0.2f'%x2)
    else:
        deta=sqrt(-beta)
        real=float('%10.3f'%(-b/(2*a)))
        imag=float('%10.3f'%(deta/(2*a)))
        print('x1=',complex(real,imag),';x2=',complex(real,-imag))

```



讨论：  
何为嵌套？学习要  
时注意什么？

# 第2章 控制结构与异常处理

## 主要内容

- 顺序与条件分支控制结构
- 循环程序设计
- 异常处理
- 函数
- 函数式编程
- 结构化程序设计思想

## ===循环程序设计===

## ◆ 列表—集合类数据类型

【定义】在一对方括号中用逗号隔开的若干数据。

【成员特点】

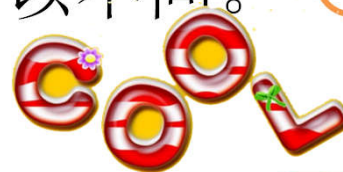
- 任何合法数据类型。
- 多个成员的数据类型可以不同。

```
>>> l=[1,2,3, 'zxd']  
>>> l  
[1, 2, 3,'zxd']
```

【索引】访问方式

- 正向索引：0~L-1 (列表长度)
- 逆向索引：-1~-L
- 区间访问：[start: **end**: interval]

```
>>> s = [1,2,3,4,5,6,7,8]  
>>> print(s[0])  
>>> print(s[-3:])  
>>> print(s[4:8])  
>>> print(s[1:8:2])
```



## ===循环程序设计===

## ◆ 列表—集合类数据类型

## 【相关操作】

```
>>> lst1=[1,3,4]
>>> lst2=[2,5]
>>> lst1+lst2
```

```
>>> 2 in lst2
>>> lst*3
>>> len(lst1)
```

加(adding)—连接两个序列

***s+t***

乘(multiplying)—重复连接同一个序列

***s\*n***

检查某个元素是否属于这序列(index)

***x in s***

计算序列长度(len)

***len(s)***

找出最大元素和最小元素(min/max)


***min(s)/max(s)***

## 【用列表表示多维数据】

➤ 列表的每个维度长度可以不同

➤ 列表的元素类型可以不同

```
>>> M=[[1,2,3],
        [4,5,6]]
```

 问题：  
M的引用方式？



讨论：

python中一切皆对象，在列表的学习中如何使用？

***s.index(x[,start[,end]])***

***s.count(x)***

***s.append(x)***

## ===循环程序设计===

## ◆ for循环

## 【常用格式】

for<variable> in range(begin, end, step):  
    <循环体>/<语句块>

【例2-8】求1~n之间正整数的平方和。n由用户输入。

## ➤ 问题分析

$$\text{sum}=1^2+2^2+3^2+\dots+n^2$$

## ➤ 计算模型

$$\text{sum}+=i*i \quad i \in [1,n]$$

```
n=int(input('input n:'))
sum=0
for i in range(1,n+1,1):
    sum+=i*i
print('sum=',sum)
```



## ===循环程序设计===

## ◆ for循环

【例2-9】 建立3\*3矩阵并输出。

## ➤ 问题分析

建立一个空列表，向空列表里插入子列表，向子列表里添加元素，从而完成二维矩阵的生成

## #建立矩阵

```
a=[]  
for i in range(3):  
    a.append([])  
    for j in range(3):  
        v=int(input("input element:"))  
        a[i].append(v)  
print(a)
```

## #输出矩阵

```
for i in range(len(a)):  
    for j in range(len(a[i])):  
        print(a[i][j], end=" ")  
    print()
```

## ===循环程序设计-for===

## 【一般格式】

for<variable> in <可迭代对象的集合>:  
    <循环体>/<语句块>

else:  
    <语句块>

【例2-10】求一组数：23, 59, 1, 20, 15, 5, 3的和及平均值。



问题：从一个包含正负数的列表中挑出为负的数进行累加，并输出结果？

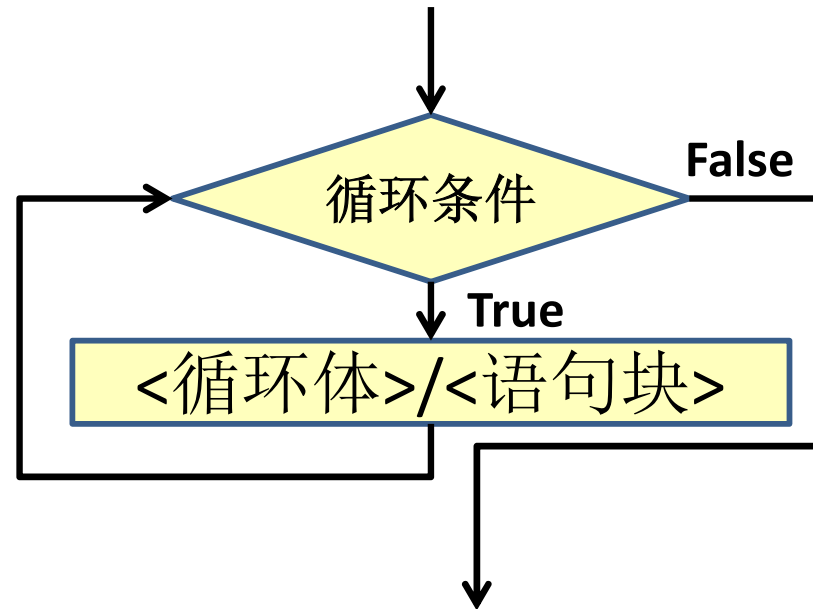
```
list1=[23,59,1,20,15,5,3]
k=0
sum1=0
for i in list1:
    sum1+=i
    k+=1
else:
    print('和为: ',sum1)
    print('平均值为: ',sum1/k)
```

## ===循环程序设计===

## ◆ while循环

## 【格式】

```
while<循环条件>:
    <循环体>/<语句块>
else:
    <语句块>
```



【例2-11】利用下述公式计算e的近似值。要求最后一项的值小于 $10^{-6}$ 即可

➤ 数学模型:  $e \approx 1 + 1/1! + 1/2! + \dots + 1/n!$

➤ 计算模型

$$\begin{cases} u=1 \\ ev=1 \\ u=u/i & i \in [1, n] \\ ev+=u & 1/u > 10E-6 \end{cases}$$

```
u=1;ev=1;i=1
while(u>10e-6):
    u=u/i
    ev+=u
    i+=1
print("e=",ev)
```

## ===循环程序设计===

## ◆ 循环和分支的嵌套

【例2-12】寻找自幂数。自幂数：对于n位数，它的各位数字的n次方加起来和仍等于这个数。如 $1^3+5^3+3^3=153$ ，153就是一个三位数自幂数

➤ 计算模型：设n位数为k，digit为n位数某位上的值

1) 找n位数自幂数，k取值空间为 $[10^{n-1}, 10^n-1]$

2)  $m=k$ ;

3) 各位数字的n次方相加，循环执行：

$\text{digit}=m\%10$ ;

$\text{total}+=\text{pow}(\text{digit},n)$ ;

$m=m//10$

3) if  $\text{total}==k$ : print(k)

## ===循环程序设计-循环和分支的嵌套===

## 【例3-11】

```
n=int(input("输入位数 【1,2,3,4,5,6】 :"))
while(0<n<7):
    start=pow(10,n-1)
    end=pow(10,n)
    print(n,"位自幂数为: ")
    for k in range(start,end):
        m=k
        total=0
        while m:
            digit=m%10
            total+=pow(digit,n)
            m//=10
        if(k==total):
            print(k,end=' ')
    n=int(input("\n输入位数 【1,2,3,4,5,6】 :"))
else:
    print("输入位数不在范围内，程序结束！ ")
```

## ===循环程序设计===

## ◆ 循环中的特殊语句

【break】 中断循环

【continue】 跳过当前剩余语句，执行下一次循环

【pass】 什么也不做

【例2-13】 对指定列表[1,2,3,4,5,6,7]中非2的倍数的数值求和

```
l=[1,2,3,4,5,6,7]
y=0
for item in l:
    if item%2==0:
        pass
    else:
        y+=item
print(y)
```

## ===循环程序设计-循环中的特殊语句===

【break】 中断循环

【continue】 结束某轮循环

【例2-14】 判断某个数是否为质数

```
n=int(input('输入某个数: '))
flag=1
if n==2:
    print('质数! ')
else:
    for i in range(2,n):
        if n%i==0:
            flag=0
            break
    if flag:
        print('质数! ')
    else:
        print('非质数! ')
```

```
n=int(input('输入某个数: '))
flag=1
if n==2:
    print('质数! ')
else:
    for i in range(2,n):
        if n%i:
            continue
        flag=0
        break
    if flag:
        print('质数! ')
    else:
        print('非质数! ')
```

# 第2章 控制结构与异常处理

## 主要内容

- 顺序与条件分支控制结构
- 循环程序设计
- 异常处理
- 函数
- 函数式编程
- 结构化程序设计思想



## ==异常==



异常：程序中产生的错误

后果：如果异常对象未被处理或捕获，程序就会用所谓的回溯(Traceback, 一种错误信息)终止执行。

```
>>> num
```

```
Traceback (most recent call last):
```

```
File "<pyshell#0>", line 1, in <module>
```

```
num
```

**NameError: name 'num' is not defined**



```
>>> 1/0
```

```
Traceback (most recent call last):
```

```
File "<pyshell#3>", line 1, in <module>
```

```
1/0
```

**ZeroDivisionError: division by zero**

**==异常==****【一般格式】**

```
try:
    <statements1>
except <name1>:          #捕获异常name1
    <statements2>
except <name2, name3>:   #捕获异常name2, name3
    <statements3>
except <name4> as e:     #捕获异常name4, e作为其实例
    <statements4>
except:                  #捕获其它所有异常
    <statements5>
else:                    #无异常
    <statements6>
finally:                 #无论有否异常发生, 保证执行
    <statements7>
```

## ==异常==

## ◆ 按异常类名捕获异常

【例2-15】输入两整数，打印它们相除之后的结果，若输入的不是整数或除数为0，进行异常处理

```
k=0
while k<4:
    try:
        x=int(input('请输入第一个整数: '))
        y=int(input('请输入第二个整数: '))
        print('x/y=',x/y)
    except ValueError:
        print('请输入一个整数.')
    except ZeroDivisionError:
        print('除数不能为零.')
    k+=1
```

## ==异常==

## ◆ 使用异常实例

【例2-16】输入两整数，打印它们相除之后的结果，若输入的不是整数或除数为0，进行异常处理

```
k=0
while k<4:
    try:
        x=int(input('请输入第一个整数: '))
        y=int(input('请输入第二个整数: '))
        print('x/y=',x/y)
    except (ValueError, ZeroDivisionError) as e:
        print(e)
    k+=1
```

## ==异常==



思考题：  
异常语句的语法结构是什么？请例举你知道的异常？

## ==异常==

## ◆ 自定义异常类

【格式】

```
class SomeCustomException(Exception):  
    pass
```

异常类名称

继承自 **Exception**

【抛出异常类】（引发异常）

**raise <class>**     #创建并抛出类的实例**raise <instance>** #抛出类的实例

【异常处理代码】可以写在except语句里

## ==异常-自定义异常类==

【例2-17】输入与输出某个人的姓名、年龄、月收入，根据每个项目的约束条件，引发异常。约定名字长度必须在2-20字符之间，年龄在18-60之间，月工资大于800元，否则引发异常。

```
class StrExcept(Exception):
    pass
```

```
while True:
    try:
        x=input("名字(2-20字符):")
        if len(x)<2 or len(x)>20:
            raise StrExcept
        y=int(input("年龄(18-60):"))
        if y<18 or y>60:
            raise MathExcept
        z=int(input("月收入(大于800):"))
        if z<800:
            raise MathExcept
```

满足条件抛出异常

```
class MathExcept(Exception):
    pass
```

```
print('姓名: ',x)
print('年龄: ',y)
print('年收入: ',z*12)
break
except StrExcept:
    print('输入名称异常')
except MathExcept:
    print('输入数值异常')
except Exception as e:
    print('输入',e)
```

捕捉异常

## ==异常==

## ◆ assert语句（断言）

【用途】期望满足用户指定的条件。当用户定义的约束不满足时触发AssertionError异常。它是条件式的raise语句。

## 【一般形式】

assert <test>, <data> # <data> 可选

逻辑表达式

<test>为假时的提示信息

## 【等效代码】

```
if not <test>:  
    raise AssertionError(<data>)
```

【使用方向】 Assert语句是用来收集用户定义的约束条件，而不是捕捉内在的程序设计错误



## ==异常==

【例2-18】求x与y的最大公约数，使用assert语句来约束x、y取值为大于1正整数。

```
while True:
    try:
        x=int(input('请输入第一个数: '))
        y=int(input('请输入第二个数: '))
        assert x>1 and y>1,'x与y必须大于1'
```

并未抛出异常

```
a=x
b=y
if a<b:
    a,b=b,a
while b!=0:
    temp=a%b
    a=b
    b=temp
else:
    print('%s和%s的最大公约数
        为: %s'%(x,y,a))
    break
except Exception as e:
    print('捕捉到异常',e)
```

## ==异常==



思考题：

自定义异常分几个步骤？语法结构是什么？

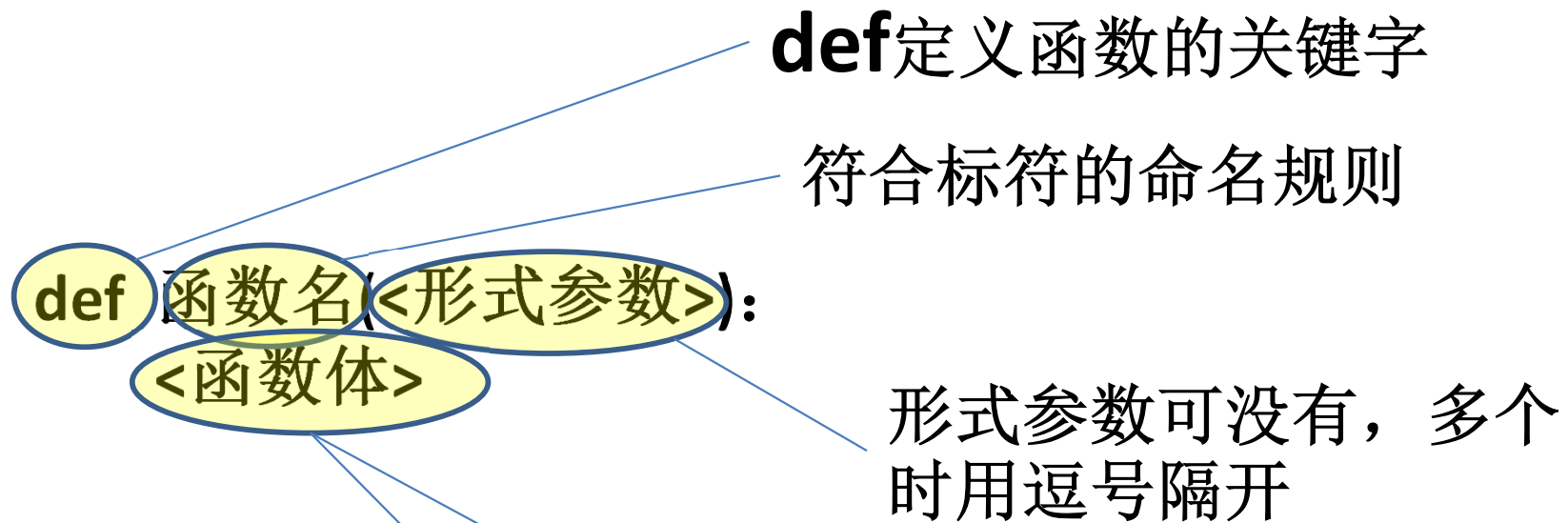
# 第2章 控制结构与异常处理

## 主要内容

- 顺序与条件分支控制结构
- 循环程序设计
- 异常处理
- 函数
- 函数式编程
- 结构化程序设计思想

## ===函数-定义===

## ◆ 函数定义的一般格式



函数体所有语句必须相对于第一行缩进

如果有返回值，使用**return**语句，其格式为：

**return** <表达式>

## ===函数-定义===

【例2-19】编写函数求出区间[i, j]内所有整数的和

```
def mySum(i,j):  
    s=0  
    for k in range(i,j+1):  
        s=s+k  
    return s
```

◆ 参数传递方式：位置绑定

实参与形参按出现的位置一一对应

info('张三',30,'男')

```
def info(name,age,sex):  
    print('name:',name,'age:',age,'sex:',sex)
```

## ===函数-参数===

## ◆ 参数传递方式：关键字绑定

实参与形参采用“形式参数名=数值”对应

```
def info(name,age,sex):  
    print('name:',name,'age:',age,'sex:',sex)
```

```
info(age=30, name='张三',sex='男')
```

## ◆ 为形参指定默认值

```
info(age=30, name='张三')
```

```
def info(name,age,sex='男'):  
    print('name:',name,'age:',age,'sex:',sex)
```

## ===函数-参数===

## ◆ 设定两种可变长参数

元组变长参数

字典变长参数

```
def 函数名(arg1, arg2,...,*tuple_args, **dic_arg)
```

```
def tup1(*s): # 参数为元组
    for i in s:
        print(i)
```

```
t=(1,2,3) # 定义元组
tup1(t):
tup1(1,2,3)
```

```
def defcountry():
    cc={}
    while True:
        cou=input("country name:")
        if(cou=="火星"):
            break
        cap=input("its capital:")
        cc[cou]=cap
    return cc
```

```
def showc(**c):
    for i in c:
        print(c.items())
```

```
c={}
c=defcountry()
showc(**c)
```

## ===函数-返回值===

## ◆ 返回多个数值

【例2-20】编写函数，计算三门课程的总分和平均分

```
def calc_grade(math, english, chinese):  
    Sum=math+english+chinese  
    Avg=float(Sum/3)  
    return Sum,Avg
```

```
>>> a,b=calc_grade(88,76,85)  
>>> a  
249  
>>> b  
83.0
```



## ===函数-调用===

## ◆ 函数的调用

## 【格式】

函数名(<实际参数>)

## 【函数出现的位置】

(1) 作为单独的语句出现，如

```
>>> calc_grade(88,76,85)  
  
(249, 83.0)
```

(2) 出现在表达式里，如

```
a,b=calc_grade(88,76,85)
```

(3) 作为实际参数出现在其他函数中，如

```
M=max(5000, mySum(1,100))
```

## ===函数===

【例2-21】 已知平面上若干点的坐标：

$a_0(1,2)$ ,  $a_1(-1,3)$ ,  $a_2(2,1.5)$ ,  $a_3(-2,0)$ ,  $a_4(4,2)$

计算任意两点的距离并生成距离矩阵，其中，矩阵元素  $(i, j)$  表示  $a_i$  和  $a_j$  之间的距离，最后输出距离矩阵和两点之间最大距离。

```
from math import *  
def d(x1,y1,x2,y2):  
    return sqrt((x1-x2)**2+(y1-y2)**2)
```

## ===函数的定义===

## 【2-21】

```
def ma():
    x=[1,-1,2,-2,4]
    y=[2,3,1.5,0,2]
    dd=[]
    s=0
    for i in range(len(x)):
        dd.append([])
        for j in range(len(x)):
            v=d(x[i],y[i],x[j],y[j])
            dd[i].append(v)
            if s<dd[i][j]:
                s=dd[i][j]
    for i in range(len(x)):
        for j in range(len(x)):
            print("%5.2f"%dd[i][j],end=" ")
        print()
    print("max=%5.2f"%s)
```

# 第2章 控制结构与异常处理

## 主要内容

- 顺序与条件分支控制结构
- 循环程序设计
- 异常处理
- 函数
- 函数式编程
- 结构化程序设计思想

## ===函数式编程===

## ◆ 匿名函数 lambda

【格式】lambda 参数列表: 语句块

```
>>> f=lambda x,y:x+y
>>> f(1,2)      #3
>>> f(3,8)      #11
```

## ◆ 内嵌函数及其作用域

【例2-22】内嵌函数

```
def f1():
    x=y=2
    def f2():
        y=3
        print('f2:x=',x)
        print('f2:y=',y)
    f2()
    print('f1:x=',x)
    print('f1:y=',y)
```

## ===函数式编程===

## 【递归思想与递归函数】

把一个复杂的大问题逐步转换为与原问题相似的小问题，在求解小问题时，又用到原问题的求解方式，直到分解为可以简单或直接求解的小问题，求得小问题的解后，再回归，直到把大问题解决。

递归算法的设计要点：

- (1) 递推公式
- (2) 递归结束条件

## 【例2-23】编程求n!

$$f(n) = \begin{cases} 1, & n=1 \\ n*f(n-1), & n>1 \end{cases}$$

结束条件  
递推公式

```
def f(n):
    if n==1:
        return 1
    return n*f(n-1)
```

## ===函数式编程===

## 【例2-24】汉诺(Hanoi)塔问题。

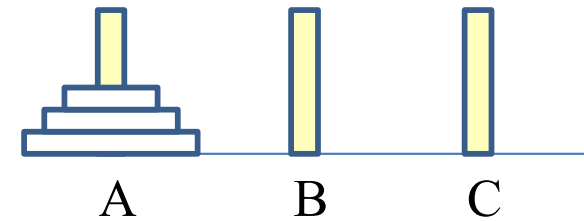


图 汉诺塔

借助B将n个盘子从A移到C = { 将一个盘子从A移到C  
   { 借助C将n-1个盘子从A移到B  
   { 将一个盘子从A移到C  
   { 借助A将n-1个盘子从B移到C

$n=1$  — 结束条件

$n>1$  — 递推公式


```
def Hanoi(n,ch1,ch2,ch3):
    if n==1:
        print(ch1,'->',ch3)
    else:
        Hanoi(n-1,ch1,ch3,ch2)
        print(ch1,'->',ch3)
        Hanoi(n-1,ch2,ch1,ch3)
```

## ===函数式编程===

## ➤ 计算模型

$$\begin{cases} f(n) = \text{move}(1) & n = 1 \\ f(n) = 2f(n-1) + \text{move}(1) & n > 1 \end{cases}$$

谨慎使用递归算法，因为它们的简洁可能会掩盖其低效率的事实。




5000亿年



1秒1个盘子

$$2^{64}-1=18466744073709551615\text{秒}$$

```
def Hanoi(n,ch1,ch2,ch3):
    if n==1:
        print(ch1,'->',ch3)
    else:
        Hanoi(n-1,ch1,ch3,ch2)
        print(ch1,'->',ch3)
        Hanoi(n-1,ch2,ch1,ch3)
```

- n为规模，也是计算规模
- 核心操作为移动盘子
- 依据递推公式，两次递推之间，执行一次移动操作，因此有如下推导过程：

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &= 2[2T(n-2)+1]+1=2^2T(n-2)+2+1 \\ &= 2^2[2T(n-3)+1]+2+1=2^3T(n-3)+2^2+2+1 \\ &\dots\dots \\ &= 2^{i-1}[2T(n-i)+1]+2^{i-2}+2^{i-3}\dots+2^0=2^iT(n-i)+2^{i-1} \\ &\dots\dots \\ &= 2^{n-1}T(n-(n-1))+2^{n-1}-1=2^{n-1}T(1)+2^{n-1}-1=2^n-1 \end{aligned}$$



## ===函数式编程===

## ◆ 高阶函数

【定义】一个函数可以作为参数传给另外一个函数，或者一个函数的返回值为另外一个函数，满足其一则为高阶函数。

## ➤ 内置高阶函数

**zip**函数:可以同时遍历多个序列,遍历次数为最短序列长度

**map**函数:可以根据提供的函数对指定序列做映射。

```
# zip函数  
a = [1,2,3]  
b = ['a','b','c']  
for x,y in zip(a,b):  
    print(x,y)
```

```
# 将序列中的元素平方  
print(*map(lambda x:x*x, [1,2,3]))
```

```
# 过滤序列中的偶数  
print(*filter(lambda x: x%2==1, [1,2,3]))
```

**filter(function,iterable):** 根据函数对给定的可迭代对象进行过滤。

## ===函数式编程===



思考题：

现在列表[1,2,3,4,5,6,7,8],请过滤掉其中奇数？

## ===函数式编程===

## ➤ 内置高阶函数

【例2-25】以普通编程方式实现计算列表中正数之和

```
lt=[2,-4,9,-5,6,13,-12,-3]
s=0
for i in range(len(lt)):
    if lt[i]>0:
        s+=lt[i]
print("sum=",s)
```

```
from functools import *
lt=[2,-4,9,-5,6,13,-12,-3]
s=filter(lambda x:x>0,lt)
sum1=reduce(lambda x,y:x+y,s)
print("sum=",sum1)
```

【例2-26】以内置高阶函数实现计算列表中正数之和



思考题：  
试使用函数式编程完成huffman编码。

## ===函数式编程===

## ◆ 自定义高阶函数

```
def bar():  
    print("in the bar..")  
def foo(func):  
    func()  
    print("in the foo..")  
  
foo(bar)
```

## ===函数式编程===

## ◆ 自定义高阶函数

【闭包】函数的嵌套定义。在函数内部定义一个嵌套函数，将嵌套函数视为一个对象，将内嵌函数作为包含它的函数的返回结果（相当于返回一段函数代码）。

## 【例2-27】使用闭包



如何省掉这两个变量？

```
def func_lib():  
    def add(x,y):  
        return x+y  
    return add  
fadd=func_lib()  
print(fadd(1,2))
```

```
def add(x,y):  
    return x+y  
def func_lib(func):  
    return func  
res= func_lib(add)  
res(3,5)
```

# 第2章 控制结构与异常处理

## 主要内容

- 顺序与条件分支控制结构
- 循环程序设计
- 异常处理
- 函数
- 函数式编程
- 结构化程序设计思想

## ===结构化程序设计思想===

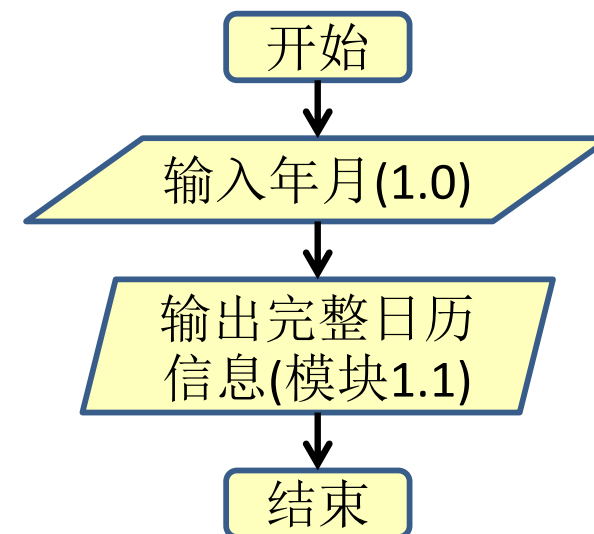
## ◆ 自顶向下逐步求精的思想

【例2-27】已知1980年1月1日是星期二，现在要求根据用户输入的年份( $\geq 1980$ )、月份在屏幕上打印出当月的日历，运行效果如下：

```
input year(yyyy):2017
input month(m):3
           March    2017
```

```
-----
Sun Mon Tue Wed Thr Fri Sat
    1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

## 第一步





## ===结构化程序设计思想===

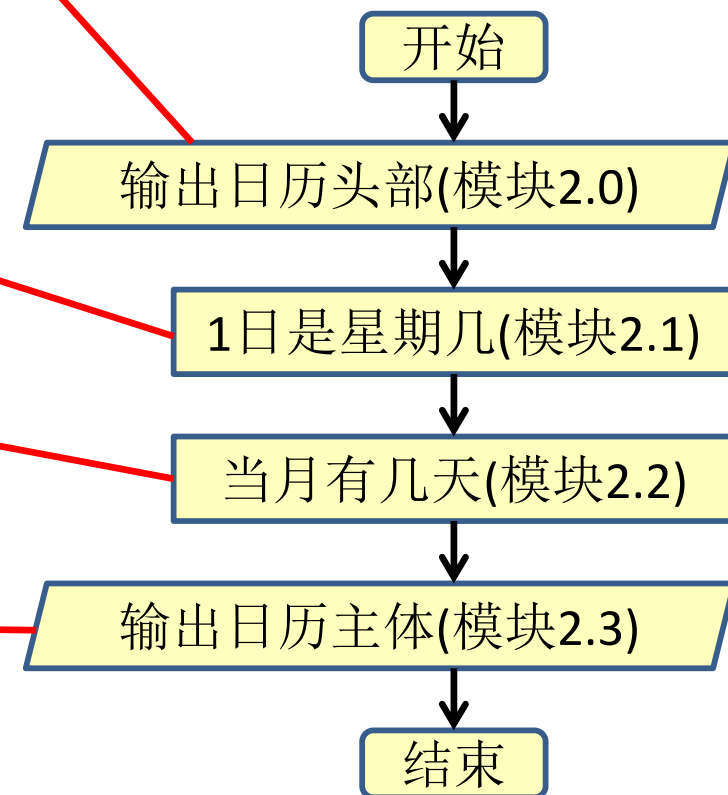
## ◆ 自顶向下逐步求精的思想

```
def pM(m,y):
    print('\t',gmN(m),' ',y)
    print('-----')
    print(' Sun Mon Tue Wed Thr Fri Sat ')
    #计算星期几
    sD=gD(m,y)
    for i in range(0,sD):
        print(' ',end=" ")
    #计算这个月有多少天
    sDm=gDm(m,y)
    for i in range(1,sDm+1):
        if i<10:
            tme=' %d'%i
        else:
            tme=' %d'%i
        print(tme,end=" ")
        if (i+sD)%7==0:
            print()
```

```
input year(yyyy):2017
input month(m):3
```

March 2017						
Sun	Mon	Tue	Wed	Thr	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

## 第二步



## ===结构化程序设计思想===

```
def gmn(m):
    mn={1:"January",2:"February",
        3:"March",4:"April",5:"May",
        6:"June",7:"Jnly",8:"August",
        9:"September",10:"October",
        11:"November",12:"Deceember"}
    return mn[m]
```

```
def gDm(m,y):
    lm1=[1,3,5,7,8,10,12]
    lm2=[4,6,9,11]
    if m in lm1:
        return 31
    if m in lm2:
        return 30
    if m==2:
        if ly(y):
            return 29
        else:
            return 28
    return 0
```

```
def ly(y):
    return (y%4==0 and y%100!=0) or y%400==0
```

【例4-4】已知1980年1月1日是星期二，现在要求根据用户输入的年份( $\geq 1980$ )、月份在屏幕上打印出当月的日历，运行效果如下：

```
def gD(m,y):
    t=1
    for i in range(1980,y):
        if ly(i):
            t+=366
        else:
            t+=365
    for i in range(1,m):
        t+=gDm(i,y)
    return t%7
```

# 本章小结

- 顺序结构与分支控制
- 循环程序设计
- 异常处理
- 函数与函数式编程
- 结构化程序的设计思想