Python程序设计

陈远祥

chenyxmail@gmail.com

北京邮电大学 电子工程学院

Python程序设计

上周主要内容

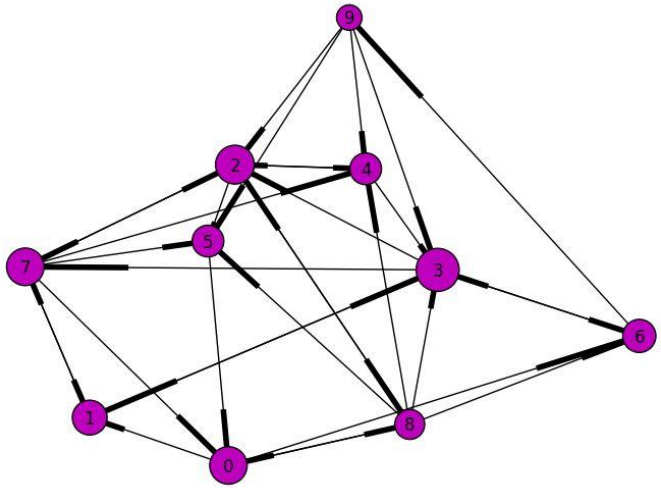
- 程序设计语言基础知识
- Python语言的发展概述
- Python语言中的类型：数字类型，字符串类型，元组类型，列表类型，文件类型，字典类型

Python程序设计

本周主要内容

- Python语法元素分析
- 布尔值
- 程序的控制结构（顺序结构，分支结构，循环结构）
- 异常处理机制

Python语法元素分析



Python语法元素分析

- Python语言采用严格的“缩进”来表明程序的格式框架
- 缩进指每一行代码开始前的空白区域，用来表示代码之间的包含和层次关系
  - ✓ 1个缩进=4个空格，用以在Python中标明代码的层次关系
  - ✓ 缩进是Python语言中表明程序框架的唯一手段
  - ✓ 单层缩进，多层缩进

Python语法元素分析

```
#e10.2CalHamlet.py
excludes = {"the","and","of","you","a","i","my","in"}
def getText():
    txt = open("hamlet.txt", "r").read()
    txt = txt.lower()
    for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_‘{|}~':
        txt = txt.replace(ch, " ")    #将文本中特殊字符替换为空格
    return txt
hamletTxt = getText()
words  = hamletTxt.split()
counts = {}
for word in words:
    counts[word] = counts.get(word,0) + 1
for word in excludes:
    del(counts[word])
items = list(counts.items())
items.sort(key=lambda x:x[1], reverse=True)
for i in range(10):
    word, count = items[i]
    print ("{0:<10}{1:>5}".format(word, count))
```

Python语法元素分析

- 注释
  - ✓ 注释：程序员在代码中加入的说明信息，不被计算机执行，提升代码的可读性
- 注释的两种方法：
  - ✓ 单行注释以#开头

```
# Here are the comments
```

  - ✓ 多行注释以"""开头和结尾

```
"""
This is a multiline comment
used in Python
"""
```

Python语法元素分析

- 程序元素
  - ✓ 常量：程序中值不发生改变的元素
  - ✓ 变量：程序中值可以发生改变的元素
- Python语言允许采用大写字母、小写字母、数字、下划线和汉字等字符及其组合给变量命名，但名字的首字符不能是数字，中间不能出现空格，长度没有限制
  - ✓ 标识符对大小写敏感，python和Python是两个不同的名字
- 命名保证程序元素的唯一性

Python语法元素分析

- 保留字，也称为关键字，指被编程语言内部定义并保留使用的标识符
- 程序员编写程序不能定义与保留字相同的标识符
- 每种程序设计语言都有一套保留字，保留字一般用来构成程序整体框架、表达关键值和具有结构性的复杂语义等
- 掌握一门编程语言首先要熟记其所对应的保留字

Python语法元素分析

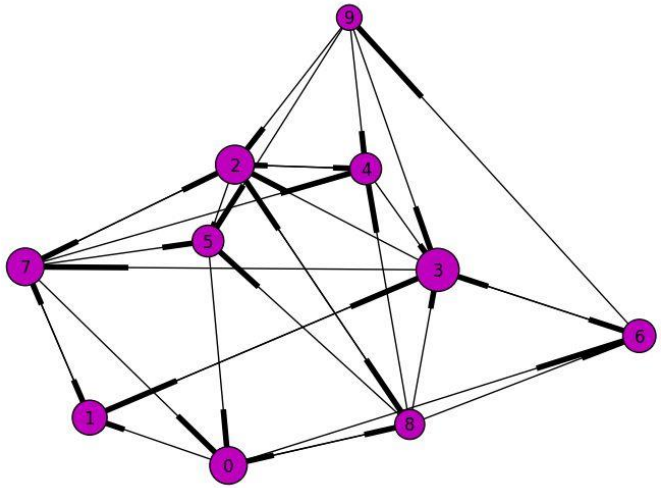
- Python3. x保留字列表(33个)

and	elif	import	raise
as	else	in	return
assert	except	is	try
break	finally	lambda	while
class	for	nonlocal	with
continue	from	not	yield
def	global	or	True
del	if	pass	False
			None

Python语法元素分析

- 赋值语句
  - ✓ Python语言中，=表示“赋值”，即将等号右侧的值计算后将结果值赋给左侧变量，包含等号（=）的语句称为“赋值语句”
- 同步赋值语句：同时给多个变量赋值
- <变量1>, …, <变量N>=<表达式1>, …, <表达式N>
  - ✓ a, b, c, d=1, 2, 3, 4

布尔值



布尔值

- 布尔数据类型只有两种值:True和False
- 大写字母T或F开头
- 可以保存在变量中，不能作为变量名
  - ✓ true
  - ✓ True
  - ✓ True=2?
  - ✓ a=True

比较操作符

- 比较操作符比较两个值，求值为一个布尔值

操作符	含义
==	等于
!=	不等于
<	小于
>	大于
<=	小于等于
>=	大于等于

- ✓ 'hello' == 'hello'
- ✓ 'hello' == 'Hello'
- ✓ 42 == 42.0
- ✓ 42 == '42'

二元布尔操作符

- and和or操作符总是接受两个布尔值（或表达式），所以它们被认为是“二元”操作符
- 如果两个布尔值都为True，and操作符就将表达式求值为True，否则求值为False

表达式	求值为
True and True	True
True and False	False
False and True	False
False and False	False

not操作符

- 和and、or不同，not操作符只作用于一个布尔值（或表达式）。not操作符求值为相反的布尔值

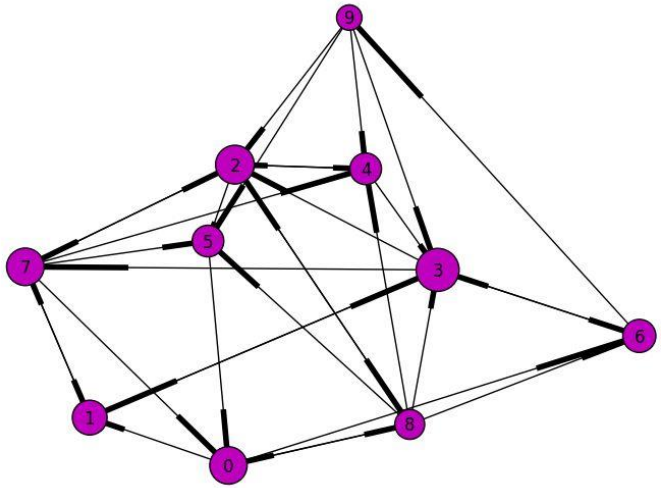
表达式	求值为
not True	False
not False	True

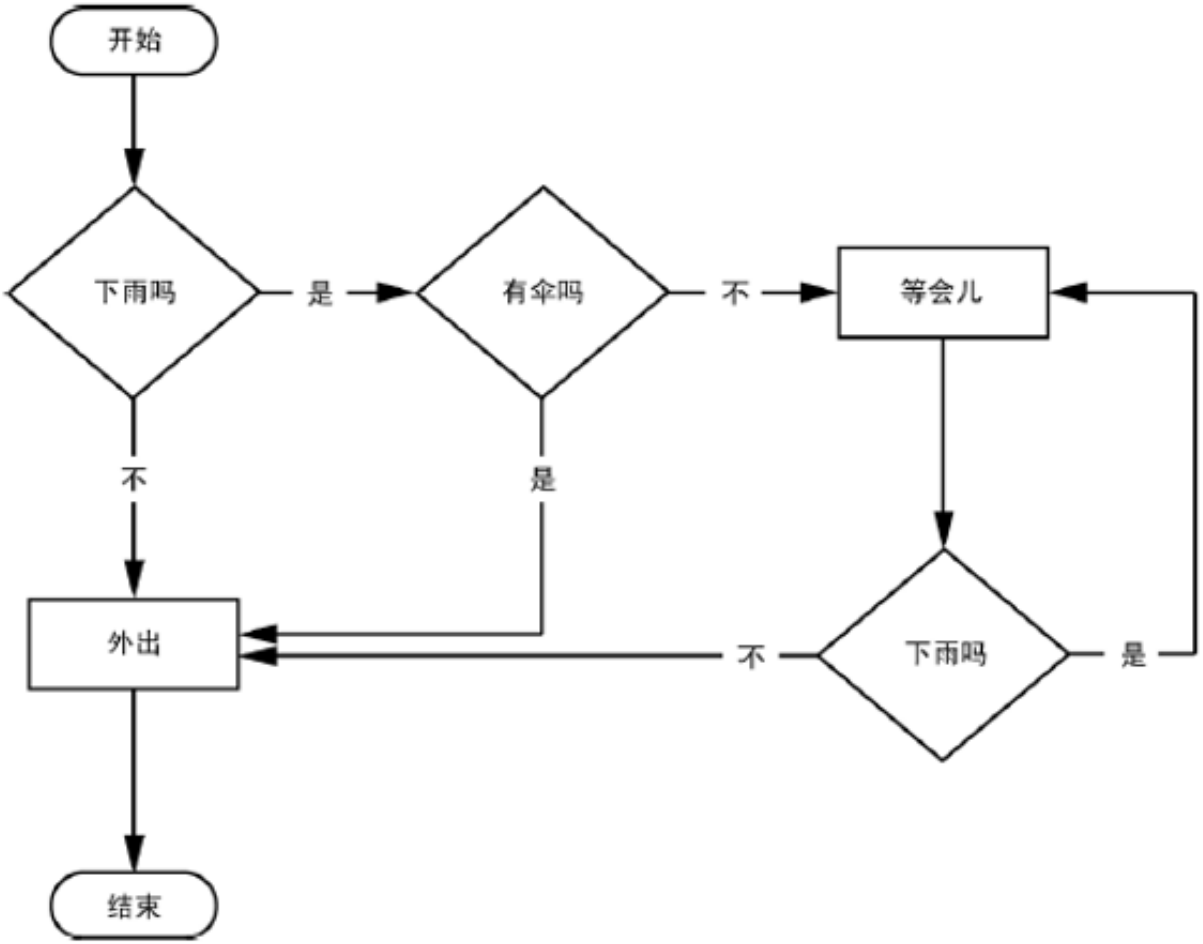
混合布尔和比较操作符

- 比较操作符和布尔操作符一起，在表达式中使用
  - ✓ （4 < 5） and （5 < 6）

- 操作符的顺序：
  - ✓ Python先求值not操作符，然后是and操作符，然后是or操作符

程序的控制结构

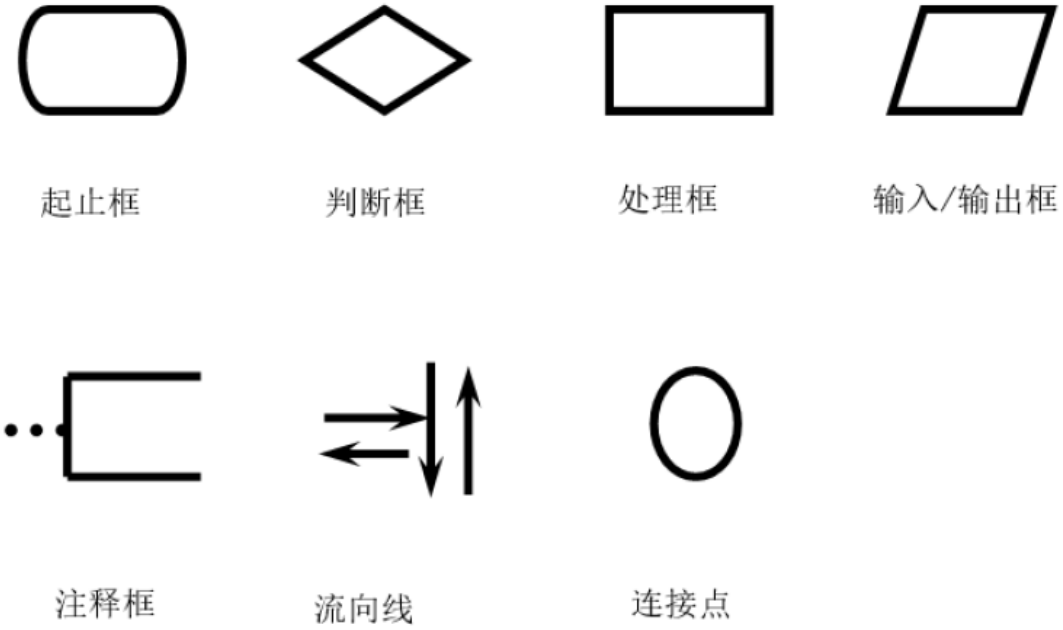




一张流程图，告诉你如果下雨要做什么

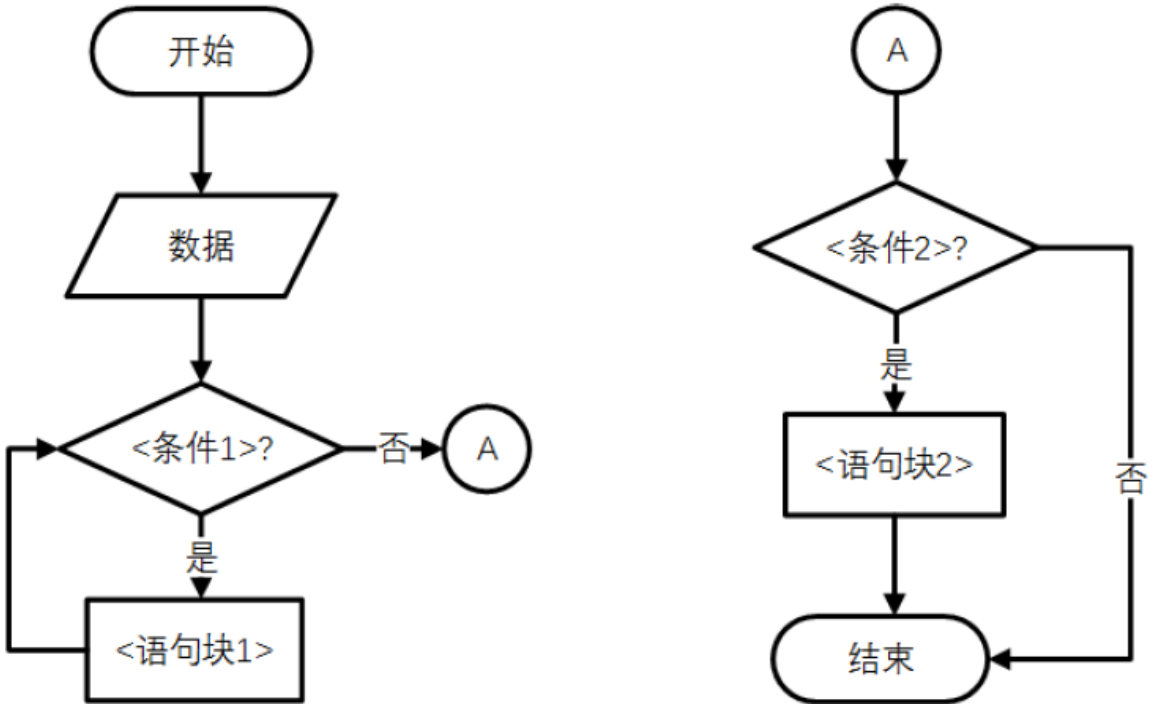
程序流程图

- 程序流程图用一系列图形、流程线和文字说明描述程序的基本操作和控制流程，它是程序分析和过程描述的最基本方式
- 流程图的基本元素包括7种

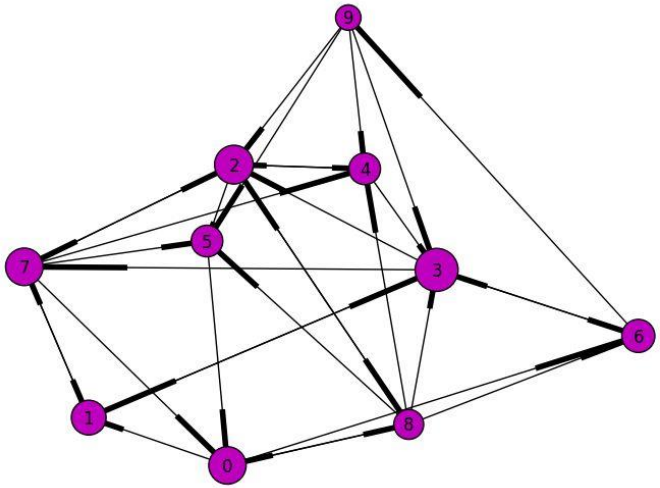


程序流程图

- 程序流程图示例：由连接点A连接的一个程序



程序的基本结构

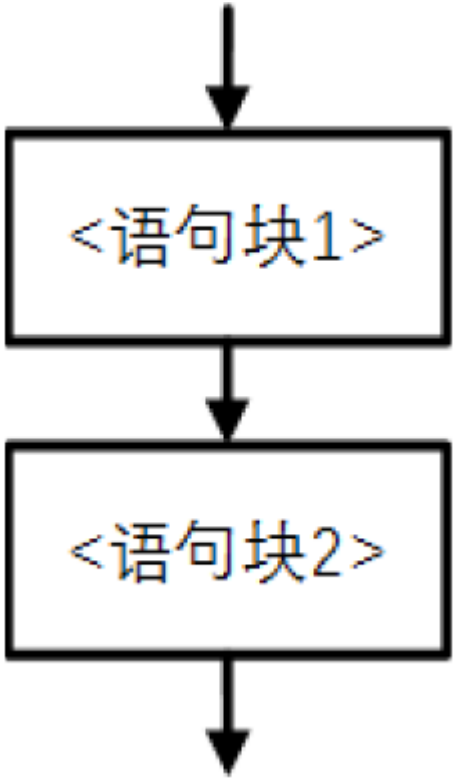


程序的基本结构

- 顺序结构是程序的基础，但单一的顺序结构不可能解决所有问题
- 程序由三种基本结构组成：
  - ✓ 顺序结构
  - ✓ 分支结构
  - ✓ 循环结构
- 任何算法(程序)都可以由这三种基本结构组合来实现

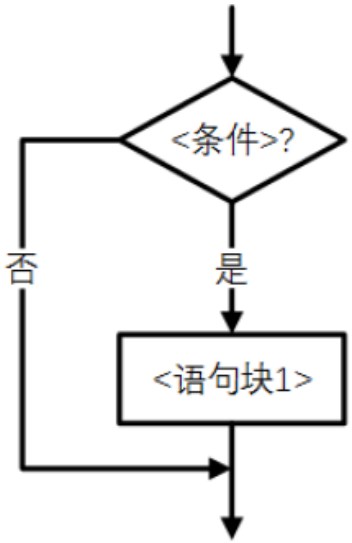
程序的基本结构

- 顺序结构是程序按照线性顺序依次执行的一种运行方式

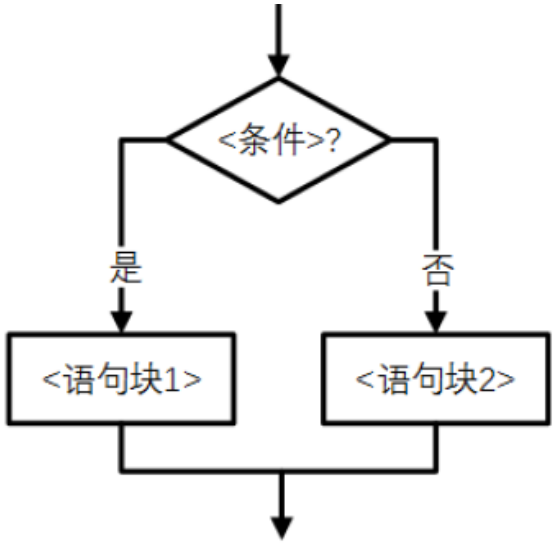


程序的基本结构

- 分支结构是根据条件判断结果而选择不同向前执行路径的一种运行方式
- 可分为单分支结构和二分支结构，二分支结构组合形成多分支结构



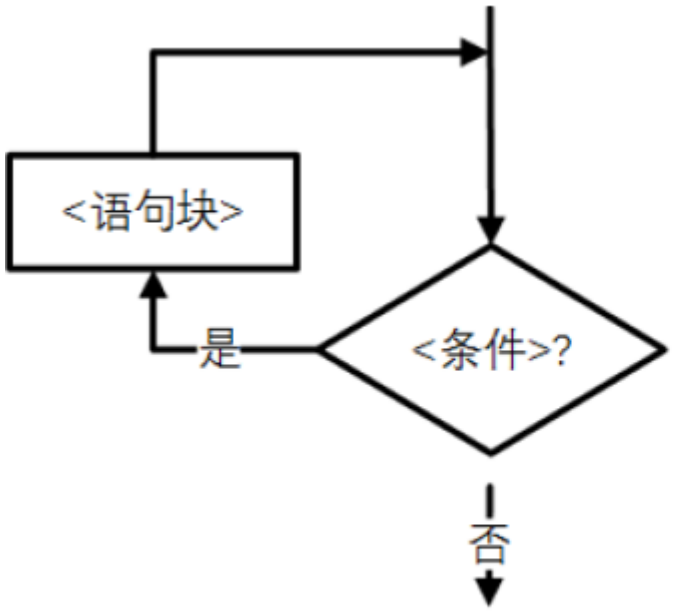
单分支结构



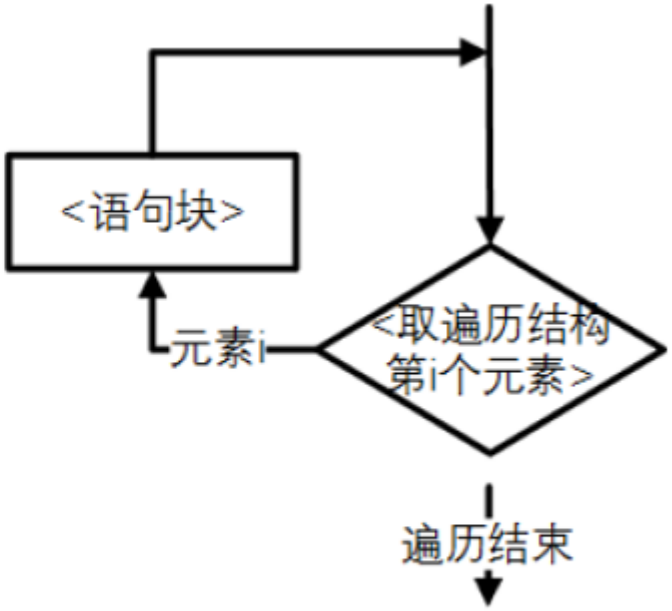
二分支结构

程序的基本结构

- 循环结构是程序根据条件判断结果向后反复执行的一种运行方式，根据循环体触发条件不同，包括条件循环和遍历循环结构



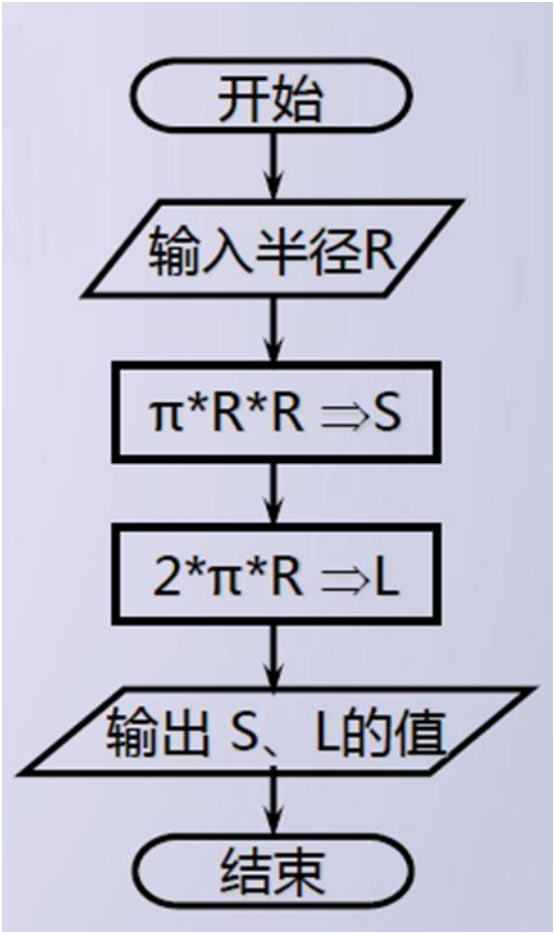
条件循环



遍历循环

程序的基本结构

- 求给定半径R的圆面积和周长
- 数学问题
  - ✓ 圆面积 $S = \pi R^2$
  - ✓ 圆周长 $L = 2\pi R$
  - ✓ 顺序结构

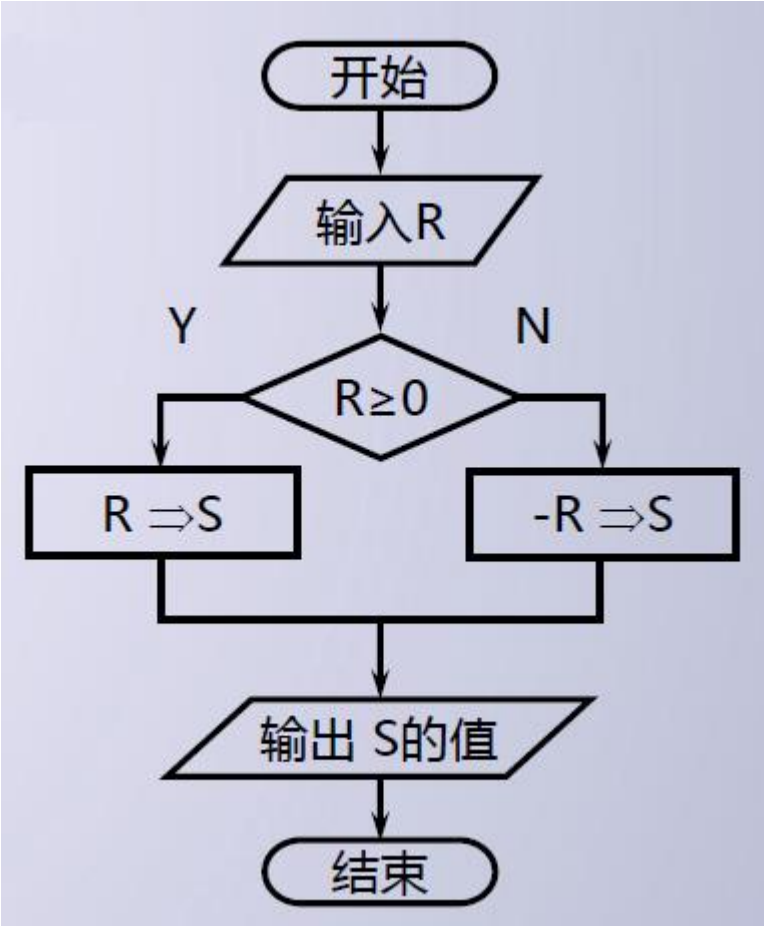


程序的基本结构

- 求给定数R的绝对值
- 数学问题

$$|R| = \begin{cases} R & R \geq 0 \\ -R & R < 0 \end{cases}$$

分支结构



程序的基本结构

- 给定K值，求1到K连加和。
- 数学问题

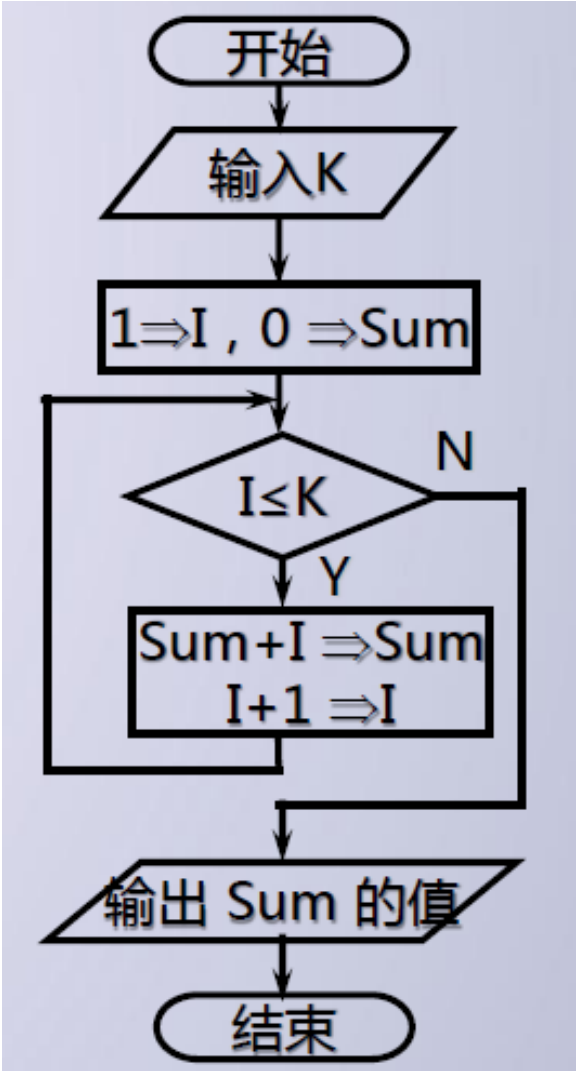
  Sum=1+2+3+…+K

  1 →I

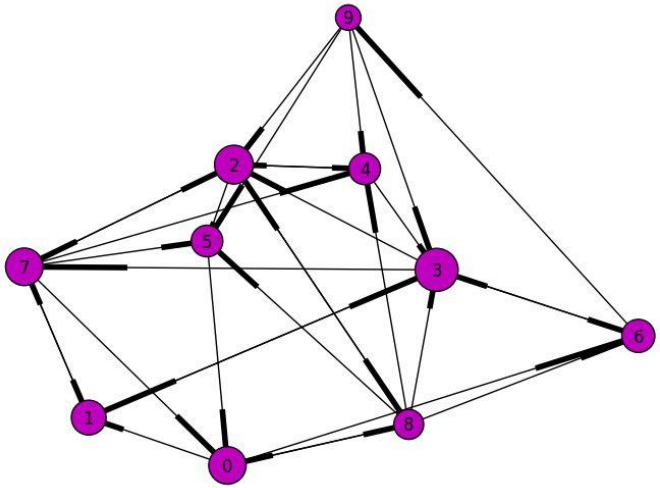
  0 →Sum

  Sum+I →Sum（I=1, 2, 3, …K）

  I+1 →I
- Sum里保存1+2+3+…+K的连加和
- 重复进行某种运算，运算对象有规律地变化，采用循环结构



程序的分支结构



程序的分支结构

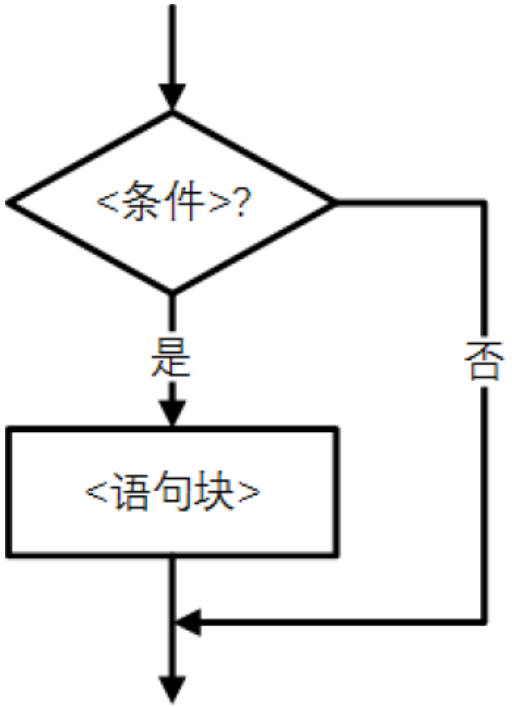
- 单分支结构：if语句
- Python中if语句的语法格式如下：

```
if <条件>:
    语句块
```

- 语句块是if条件满足后执行的一个或多个语句序列，语句块中语句通过与if所在行形成缩进表达包含关系
- if语句首先评估<条件>的结果值，如果结果为True，则执行语句块里的语句序列，然后控制转向程序的下一条语句；如果结果为False，语句块里的语句会被跳过

程序的分支结构

- 单分支结构：if语句
- if语句中语句块执行与否依赖于条件判断。但无论什么情况，控制都会转到if语句后与该语句同级别的下一条语句

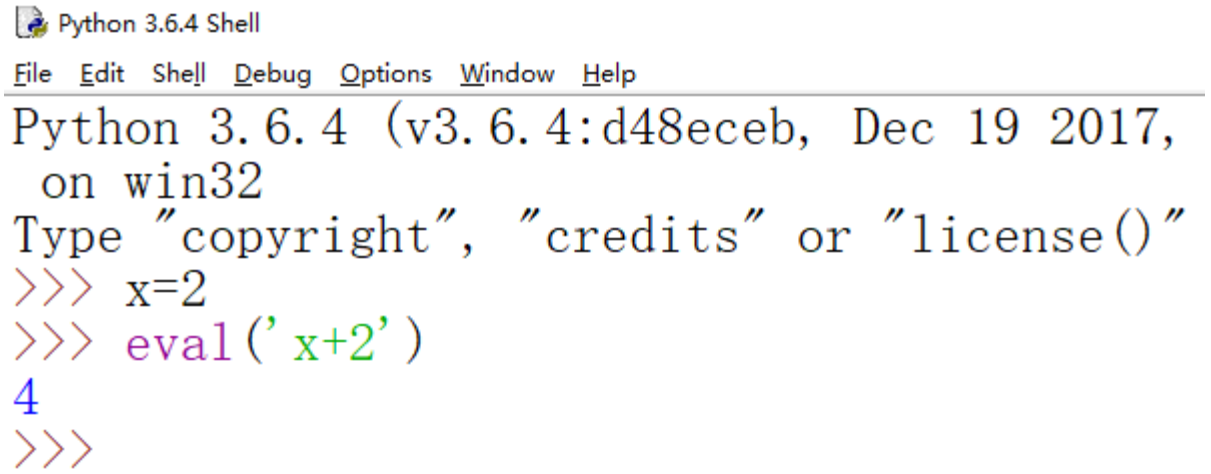


程序的分支结构

- PM2.5空气质量提醒(1)
  - ✓ if PM2.5值 >=75，打印空气污染警告
  - ✓ if 35<= PM2.5值 <75，打印空气污染警告
  - ✓ if PM2.5值 <35，打印空气质量优，建议户外运动
  - ✓ 输出：打印空气质量提醒

程序的分支结构

- eval()函数
  - ✓ eval(<字符串>)函数是Python语言中一个十分重要的函数，它能够以Python表达式的方式解析并执行字符串，将返回结果输出



程序的分支结构

4-1.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\4-1.py (3.6.4)

File Edit Format Run Options Window Help

```
PM = eval(input("请输入PM2.5数值："))
if 0<= PM < 35:
    print("空气优质，快去户外运动!")
if 35 <= PM <75:
    print("空气良好，适度户外活动！")
if 75 <= PM:
    print("空气污染，请小心！")
```

程序的分支结构

- 二分支结构：if-else语句
- if-else语句用来形成二分支结构，语法格式如下：

```
if<条件>:
    <语句块1>
else:
    <语句块2>
```

- <语句块1>是在if条件满足后执行的一个或多个语句序列
- <语句块2>是if条件不满足后执行的语句序列

程序的分支结构

- PM2.5空气质量提醒(2)

4-2.py - F:\北邮课程\python程序设计\python程序设计-北理工\source\4-2.py (3.6.4)

File Edit Format Run Options Window Help

```
PM = eval(input("请输入PM2.5 数值: "))
if PM >= 75:
    print("空气存在污染，请小心！")
else:
    print("空气没有污染，可以开展户外运动!")
```

程序的分支结构

- 二分支结构还有一种更简洁的表达方式，适合通过判断返回特定值，语法格式如下
  - ✓ <表达式1> if <条件> else <表达式2>

if else-简洁.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\if else-简洁.py (3.6.4)

File Edit Format Run Options Window Help

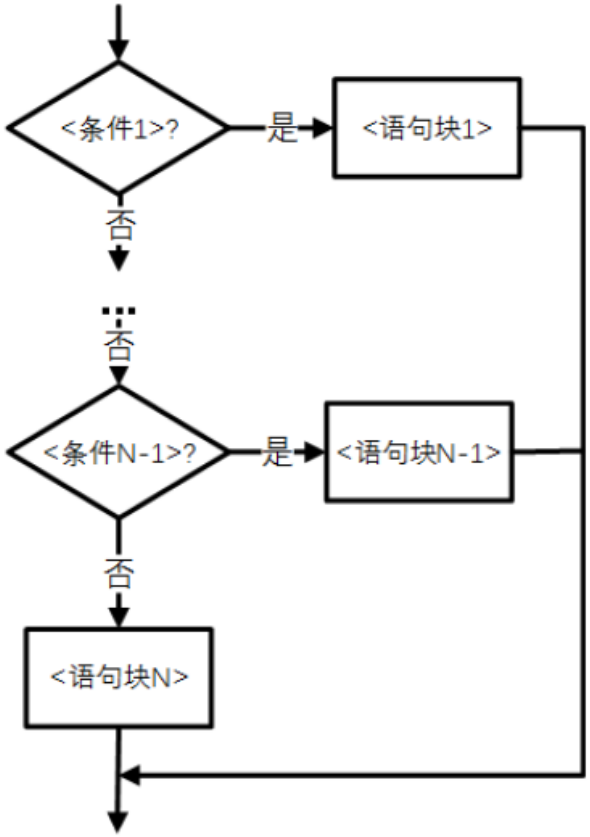
```
PM = eval(input("请输入PM2.5 数值: "))
print("空气存在污染，请小心！") if PM >= 75 else print("空气没有污染，可以开展户外运动!")
```

- if…else的紧凑结构非常适合对特殊值处理的情况

程序的分支结构

- 多分支结构：if-elif-else语句
- if-elif-else描述多分支结构，语句格式如下：

```
if <条件1>:
    <语句块1>
elif <条件2>:
    <语句块2>
...
else:
    <语句块N>
```



程序的分支结构

- 多分支结构是二分支结构的扩展，这种形式通常用于设置同一个判断条件的多条执行路径
- Python依次评估寻找第一个结果为True的条件，执行该条件下的语句块，同时结束后跳过整个if-elif-else结构，执行后面的语句。如果没有任何条件成立，else下面的语句块被执行。else子句是可选的

程序的分支结构

- PM2.5空气质量提醒(3)

File Edit Format Run Options Window Help

```
PM = eval(input("请输入PM2.5数值: "))
if 0<= PM < 35:
    print("空气优质，快去户外运动!")
elif 35 <= PM <75:
    print("空气良好，适度户外活动！")
else:
    print("空气污染，请小心！")
```

程序的分支结构

- 分支结构能改变程序控制流
  - ✓ 优点：可以开发更先进算法
  - ✓ 缺点：算法设计复杂
- 找出三个数字中最大者的程序设计为例
  - ✓ 输入：三个数值
  - ✓ 处理：三者最大算法
  - ✓ 输出：打印最大值
- 计算机如何确定哪个是用户输入的最大值？

程序的分支结构

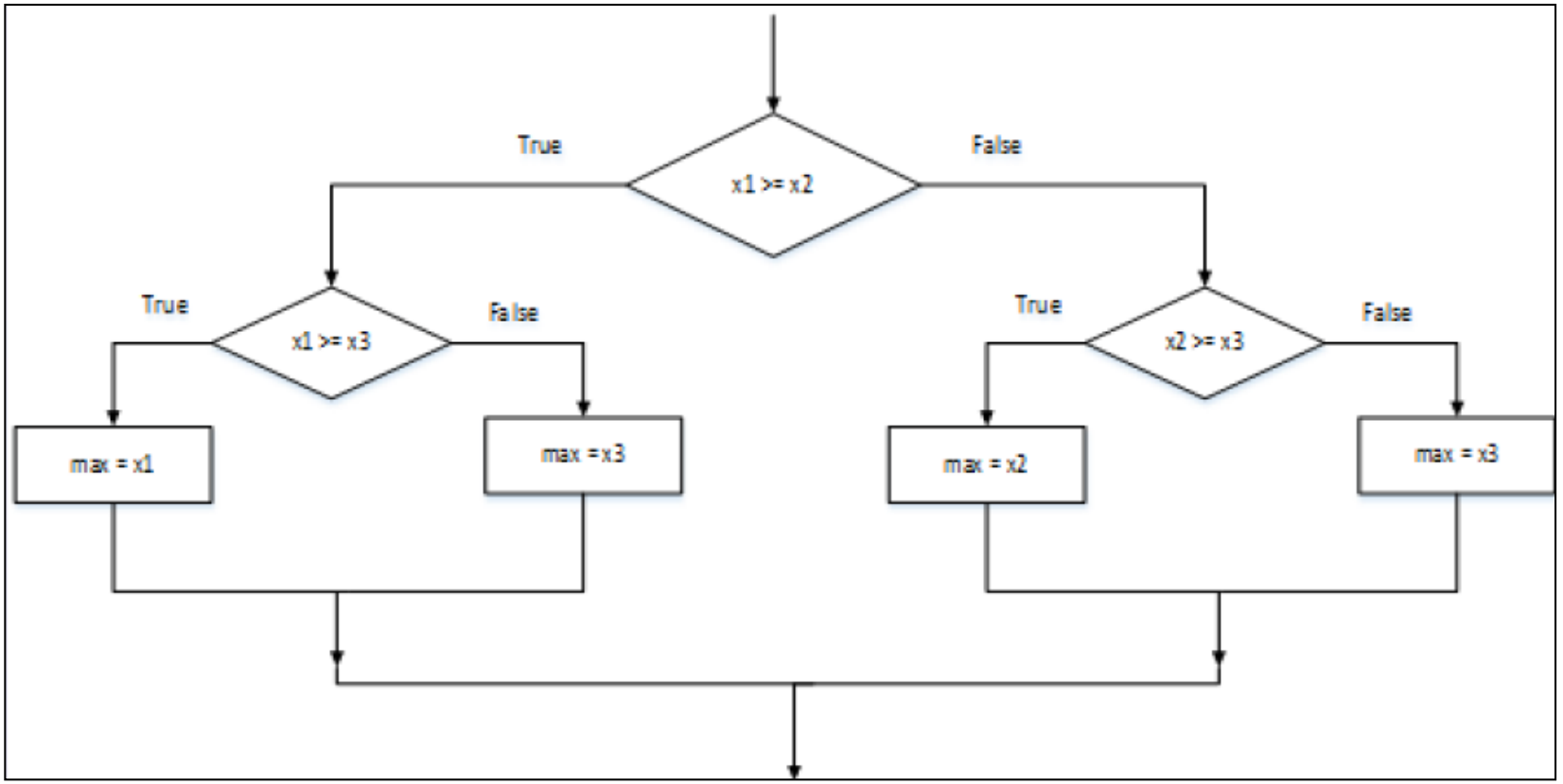
- 策略1：通盘比较
  - ✓ 通盘比较，即将每一个值与其他所有值比较以确定最大值

```
if x1 >= x2 and x1 >= x3:
    max = x1
elif x2 >= x1 and x2 >= x3:
    max = x2
else:
    max = x3
```

- 存在的问题：
  - ✓ 目前只有三个值，比较简单
  - ✓ 如果是五个值比较，表达式包含四个and，比较复杂
  - ✓ 每个表达式结果没有被互相利用，效率低(x1与x2比较了两次)

程序的分支结构

- 策略2：决策树
  - ✓ 决策树方法可以避免冗余比较



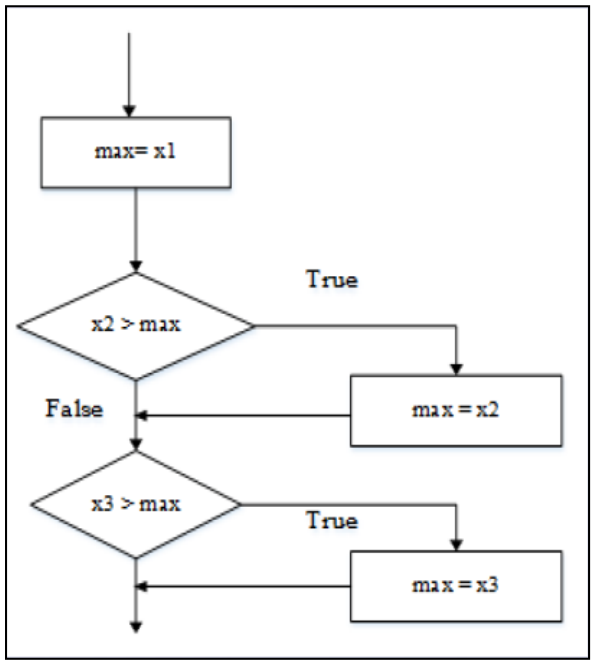
程序的分支结构

- 策略2：决策树
  - ✓ 决策树方法可以避免冗余比较
  - ✓ 先判断x1>x2，如果成立再判断x1>x3，否则判断x2>x3
  - ✓ 虽然效率高，但设计三个以上的方案，复杂性会爆炸性地增长

```
if x1 >= x2:
    if x1 >= x3:
        max = x1
    else:
        max = x3
else:
    if x2 >= x3:
        max = x2
    else:
        max = x3
```

程序的分支结构

- 策略3：顺序处理
  - ✓ 逐个扫描每个值，保留最大者
  - ✓ 以max变量保存当前最大值，完成最后一个扫描时，max就是最大值



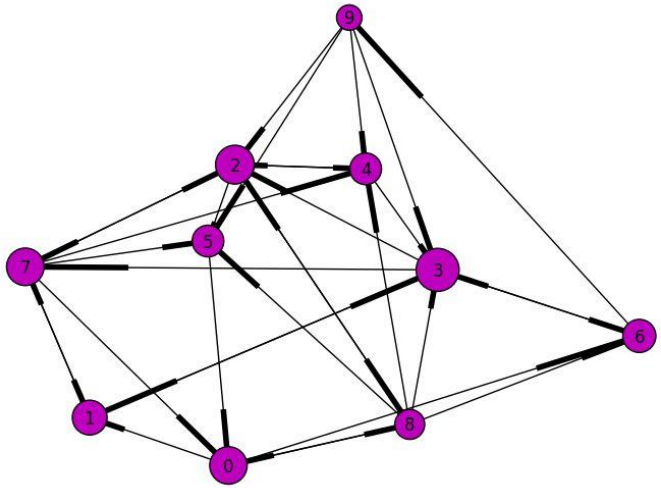

```
max = x1
if x2 > max:
    max = x2
if x3 > max:
    max = x3
```

借助循环结构，我们可以实现n个数最大者求值，高效、易读

程序的分支结构

- 策略4：Python终极解决方案
  - ✓ 使用Python内置的max函数
  - ✓ max（x1, x2, x3）

程序的循环结构



程序的循环结构

- for循环
  - ✓ Python可以使用for语句循环遍历整个序列的值
  - ✓ 在for循环中，循环变量var遍历了队列中的每一个值，循环的语句体为每个值执行一次

```
for <var> in <sequence>:
    <body>
```

  - ✓ Measure string，测量列表中每个字符的长度

程序的循环结构

Measure string.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\Measure string.py (3.6.4)

File Edit Format Run Options Window Help

```
# Measure string:
words=["毛泽东思想","邓小平理论","三个代表重要思想","科学发展
for w in words:
    print(w, len(w))
```

程序的循环结构

- for循环在执行过程中，直接在序列上进行遍历，而非在内存中生成一个新的序列拷贝进行
  - ✓ Remove string，去除长度大于10的字符串
  - ✓ 不同缩进代表不同的含义

Remove string.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\Remove string.py (3.6.4)

File Edit Format Run Options Window Help

```
# Remove string :
words=["毛泽东思想","邓小平理论","三个代表重要
for w in words:
    if len(w)>10:
        words.remove(w)
    print(words)
```

程序的循环结构

- 遍历结构可以是字符串、文件、组合数据类型或range()函数：

循环N次	遍历文件fi的每一行	遍历字符串s	遍历列表ls
for i in range(N):	for line in fi:	for c in s:	for item in ls:
<语句块>	<语句块>	<语句块>	<语句块>

- range() 函数可创建一个整数列表
  - ✓ range(10)
  - ✓ range(1,10)
  - ✓ range(1,10,2)
- #range遍历，打印列表中所有字符

程序的循环结构

range遍历.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\range遍历.py (3.6.4)

File Edit Format Run Options Window Help

```
words=["毛泽东思想","邓小平理论","三个代表重要思
for i in range(len(words)):
    print(words[i],end=',')
```

程序的循环结构

- 遍历循环还有一种扩展模式，使用方法如下：

```
for <循环变量> in <遍历结构>:
        <语句块1>
else:
        <语句块2>
```

- 当for循环正常执行之后，程序会继续执行else语句中内容。else语句只在循环正常执行之后才执行并结束，可以在<语句块2>中放置判断循环执行情况的语句。
- #for test，测试是否把列表中所有字符打印出

程序的循环结构

for test.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\for test.py (3.6.4)

File Edit Format Run Options Window Help

```
words=["毛泽东思想","邓小平理论","三个代表重
for i in range(len(words)):
    print(words[i])
else:
    print("循环正常结束")
```

程序的循环结构

- for循环-求平均数

  步骤：

  输入数字的个数n

  将sum初始化为0

  循环n次：

  输入数字x

  将x加入sum中

  将sum/n作为平均数输出出来

- #average，求平均数

程序的循环结构

average.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\average.py (3.6.4)

File Edit Format Run Options Window Help

```
#average
n=eval(input("How many numbers?"))
sum=0.0
for i in range(n):
    x=eval(input("Enter a numbers>>"))
    sum=sum+x
print("\nThe average is",sum/n)
```

程序的循环结构

- for循环缺点
  - ✓ 程序开始时必须提供输入数字总数
  - ✓ 大规模数字求平均值需要用户先数清楚个数
  - ✓ for循环是需要提供固定循环次数的循环方式
  - ✓ Python提供了另一种循环模式即无限循环，不需要提前知道循环次数，即我们提到的当型循环也叫条件循环

程序的循环结构

- 无限循环：
  - ✓ 无限循环一直保持循环操作直到特定循环条件不被满足才结束，不需要提前知道确定循环次数
  - ✓ Python通过保留字while实现无限循环

```
while <condition>:
        <body>
```

  - ✓ while语句中<condition>是布尔表达式
  - ✓ <body>循环体是一条或多条语句
  - ✓ 在while循环中，条件总是在循环顶部被判断，即在循环体执行之前，这种结构又被称为前测循环

程序的循环结构

- 使用while循环完成从0到100的求和，并打印平均值
  - ✓ #average2，输出0到100，并求平均值
- 如果循环体忘记累加i，条件判断一直为真，循环体将一直执行，这就是所谓的死循环程序
- 这时通常使用ctrl+c来终止一个程序

程序的循环结构

average2.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\average2.py (3.6.4)

File Edit Format Run Options Window Help

```
#average,计算从0到100的求和，并打印平均值
i=0 #0到100的数字，初始值为0
n=0 #有多少个数字
sum=0 #和
while i<=100:
    print(i)
    sum=sum+i
    n=n+1
    i=i+1
print("\nThe average is",sum/n)
```

程序的循环结构

- 循环保留字：break和continue
- break用来跳出最内层for或while循环，脱离该循环后程序从循环后代码继续续执行
- break语句跳出了最内层while循环，但仍然继续执行外层循环。每个break语句只有能力跳出当前层次循环
- #break，从0开始累加，超过100时候停止

程序的循环结构

break.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\break.py (3.6.4)

File Edit Format Run Options Window Help

```
#累加直到100
sum=0
number=0
while number<20:
    number+=1
    sum+=number
    if sum>100:
        break
print("The number is",number)
print("The sum is",sum)
```

程序的循环结构

- continue用来结束当前当次循环，即跳出循环体中下面尚未执行的语句，但不跳出当前循环
- 对于while循环，继续求解循环条件。而对于for循环，程序流程接着遍历循环列表
- 区别：continue语句只结束本次循环，而不终止整个循环的执行。而break语句则是结束整个循环过程，不再判断执行循环的条件是否成立
- #continue，找出0到9之间的奇、偶数并打印

程序的循环结构

continue.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\continue.py (3.6.4)

File Edit Format Run Options Window Help

```
for number in range(10):
    if number%2==0:
        print("Found an even number",number)
        continue #break
    print("Found an odd number",number)
```

程序的循环结构

- for/while 中的else用法
  - ✓ for循环和while循环中都存在一个else扩展用法
  - ✓ <for… else: …> <while… else: …>语句与循环的搭配使用，else:后的表达式在for循环列表遍历完毕后或while 条件语句不满足的情况下执行
  - ✓ #for else

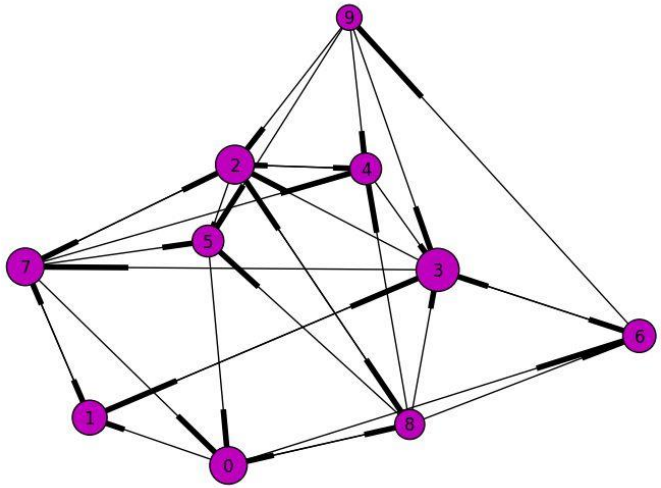
程序的循环结构

for else.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\for else.py (3.6.4)

File Edit Format Run Options Window Help

```
for s in "PYTHON":
    if s=="T":
        continue
    print(s)
else:
    print("正常退出")
```

通用循环构造方法



通用循环构造

- 交互式循环，交互式循环是无限循环的一种，允许用户通过交互的方式重复程序的特定部分
  - ✓ 询问是否输入数据并根据输入数据求平均，伪代码如下

```
初始化sum为0
初始化count为0
初始化moredata为"yes"
当moredata值为"yes"时
输入数字x
将x加入sum
count值加1
询问用户是否还有moredata需要处理
输出sum/count
```

  - ✓ #average interactive
  - ✓ 用户不需要计数，但是总是被信息打扰

通用循环构造

average interactive.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\average interactive.py (3.6.4)

File Edit Format Run Options Window Help

```
#average,交互式循环，询问是否数据输入并根据输入数据求平均值
sum=0.0
count=0 #多少个数
moredata="yes"#
while moredata[0]=="y":
    x=eval(input("Enter a number>>"))
    sum=sum+x
    count=count+1
    moredata=input("Do you have more numbers(yes or no)?")
print("\nThe average is",sum/count)
```

通用循环构造

- 哨兵循环，执行循环直到遇到特定的值，循环语句才终止执行的循环结构设计方法
- 哨兵循环是求平均数的更好方案
  - ✓ 设定一个哨兵值作为循环终止的标志
  - ✓ 任何值都可以做哨兵，但要与实际数据有所区别
  - ✓ 在求考试分数平均数的程序中，可以设定负数为哨兵
- 伪代码
  - ✓ 接收第一个数据
  - ✓ while这个数据不是哨兵
  - ✓ 程序执行相关语句
  - ✓ 接收下一个数据项
  - ✓ #average guard，计算平均成绩，输入负数停止计算

通用循环构造

average guard.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\average guard.py (3.6.4)

File Edit Format Run Options Window Help

```
#average,哨兵循环，计算平均成绩遇到负数时候停止执行
sum=0.0
count=0
x=eval(input("Enter a positive number>>"))
while x>=0:
    sum=sum+x
    count=count+1
    x=eval(input("Enter a positive number>>"))
print("\nThe average is",sum/count)
```

通用循环构造

- 哨兵循环，没有yes/no的干扰，执行结果更加清晰
- 不能包含负数的平均数计算，为了更加通用化需要引入字符串

通用循环构造

- 哨兵循环2，利用非数字字符串表示输入结束
- 所有其他字符串将被转换成数字作为数据处理
- 空字符串以“”（引号中间没有空格）代表，可以作为哨兵，用户输入回车Python就返回空字符串

通用循环构造

- 伪代码

  初始化count为0

  接受输入的字符串数据，xStr

  while xStr非空

  将xStr转换为数字x

  将x加入sum

  count值加1

  接受下个字符串数据，xStr

  输出sum/count

  ✓ #average gurad2

通用循环构造

average guard2.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\average guard2.py (3.6.4)

File Edit Format Run Options Window Help

```
#average,哨兵循环，计算平均数,遇到空字符(回车)停止执行
sum=0.0
count=0
xStr=input("Enter a number(<Enter> to quit)>>")
while xStr!="":
    x=eval(xStr)
    sum=sum+x
    count=count+1
    xStr=input("Enter a number(<Enter> to quit)>>")
print("\nThe average is",sum/count)
```

通用循环构造

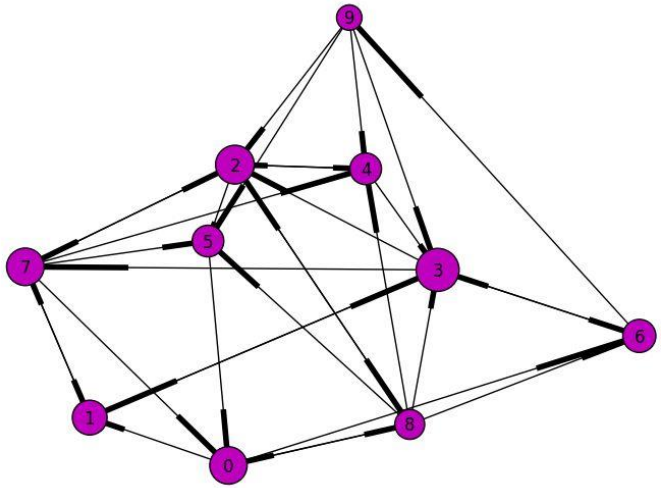
- 嵌套循环，循环体内嵌入其他的循环体，如在while循环中可以嵌入for循环，在for循环中嵌入while循环
  - ✓ #输出所有两位数

输出所有两位数.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\输出所有两位数.py (3.6.

File Edit Format Run Options Window Help

```
#嵌套循环，输出所有2位数
for i in range(1,10):
    for j in range(0,10):
        print (i*10+j)
```

死循环



死循环

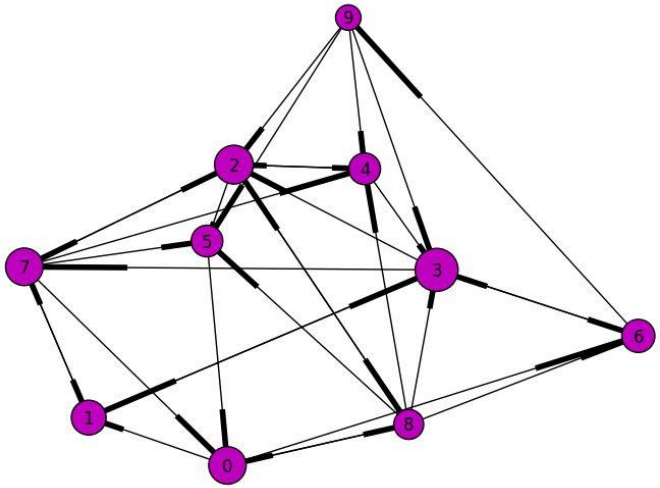
- 死循环并非一无是处，c语言中死循环while true或 while 1是单片机编程的普遍用法，死循环一直运行等待中断程序发生，然后去处理中断程序
- 在Python 中我们也可以利用死循环完成特定功能
- #死循环，输入正数时一直循环，输入负数时停止

死循环.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\死循环.py (3.6.4)

File Edit Format Run Options Window Help

```
while True:
    number=eval(input("Enter a positive number:"))
    if number<=0:
        break
print("程序终止")
```

异常处理机制



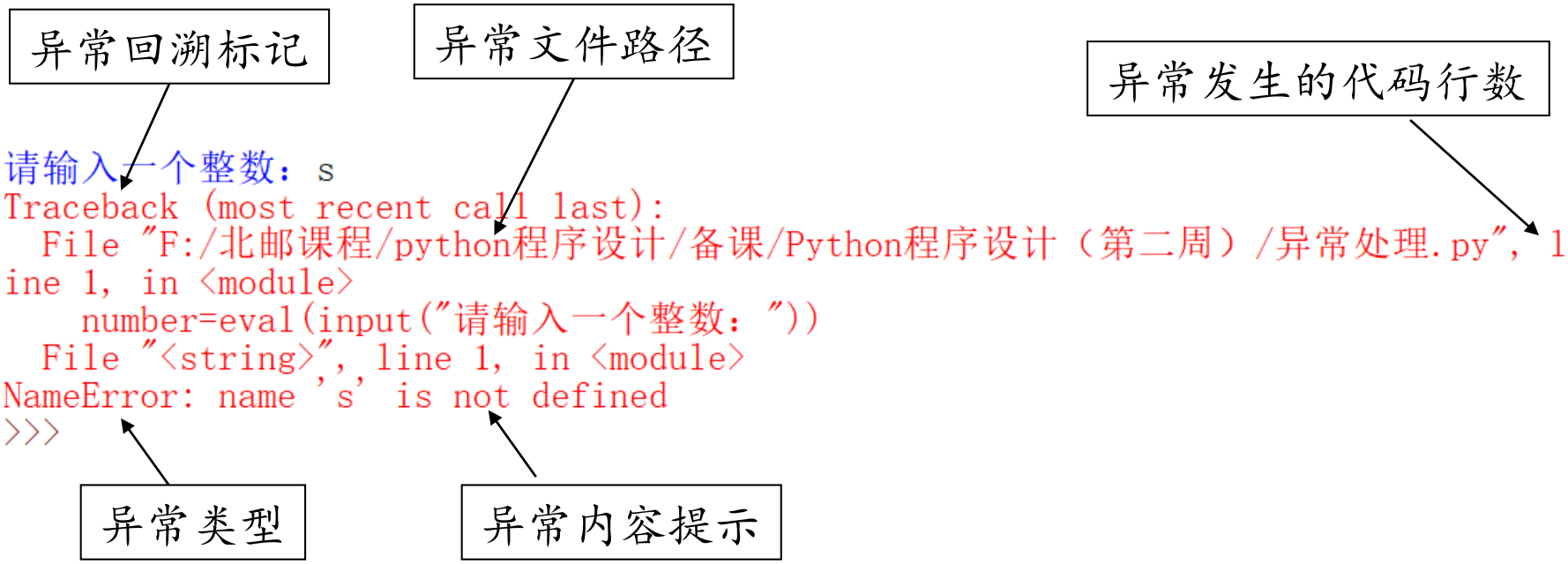
异常处理机制

- 异常处理在任何一门编程语言里都是值得关注的一个话题，良好的异常处理可以让你的程序更加健壮，清晰的错误信息更能帮助你快速修复问题
- 异常即是一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行
- 一般情况下，在Python无法正常处理程序时就会发生一个异常
- 异常是Python的对象，当Python脚本发生异常时我们需要捕获处理它，否则程序会终止执行

异常处理机制

- 观察下面这段程序

```
number=eval(input（‘请输入一个整数：’))
print（number*2）
```



异常处理机制

- Python异常信息中最重要的部分是异常类型，它表明异常发生的原因，也就是程序处理异常的依据
- Python使用try-except来实现异常处理

```
try:
    <语句块1>
except <异常类型>:
    <语句块2>
```

- 语句块1是正常执行的内容，当发生异常时，执行except保留字后面的语句块
- #异常处理 try except

异常处理机制

异常处理_try except.py - F:\北邮课程\python程序设计\备课\Python程序设计（第二周）\异常处理_try except.py (3.6.4)

File Edit Format Run Options Window Help

```
try:
    number=eval(input("请输入一个整数："))
    print(number*2)
except NameError: #未声明/初始化对象（没有属性）
    print("输入错误，请输入一个整数！")
except:#没有指定任何类型，表示对应的语句块可以处理所有其他异常
    print("其他错误")
```

异常处理机制

异常类型

异常名称	描述
BaseException	所有异常的基类
SystemExit	解释器请求退出
KeyboardInterrupt	用户中断执行(通常是输入^C)
Exception	常规错误的基类
StopIteration	迭代器没有更多的值
GeneratorExit	生成器(generator)发生异常来通知退出
StandardError	所有的内建标准异常的基类
ArithmeticError	所有数值计算错误的基类
FloatingPointError	浮点计算错误
OverflowError	数值运算超出最大限制
ZeroDivisionError	除(或取模)零 (所有数据类型)
AssertionError	断言语句失败
AttributeError	对象没有这个属性
EOFError	没有内建输入,到达EOF 标记
EnvironmentError	操作系统错误的基类
OSError	操作系统错误
WindowsError	系统调用失败
ImportError	导入模块/对象失败
LookupError	无效数据查询的基类
IndexError	序列中没有此索引(index)
KeyError	映射中没有这个键
MemoryError	内存溢出错误(对于Python 解释器不是致命的)
NameError	未声明/初始化对象 (没有属性)
UnboundLocalError	访问未初始化的本地变量
ReferenceError	弱引用(Weak reference)试图访问已经垃圾回收了的对象
RuntimeError	一般的运行时错误
NotImplementedError	尚未实现的方法
SyntaxError	Python 语法错误
IndentationError	缩进错误
TabError	Tab 和空格混用
SystemError	一般的解释器系统错误
TypeError	对类型无效的操作

异常处理机制

- 异常和错误的关系，相似但是不同
  - ✓ 异常和错误都可能引起程序执行错误而退出，他们属于程序没有考虑到的例外情况（exception）
  - ✓ 绝大多数不可控因素是可以预见的（除数为0，类型不对，打开不存在的文件等），可以预见的例外情况称为“异常”（checked exception）
  - ✓ 另外一些因为编码逻辑产生的不可预见的情况称为“错误”（unchecked exception），错误发生后程序无法执行，而且程序本不该处理这类可能的例外
  - ✓ 例如，一个包含6个字符的字符串，程序去索引第7个元素
  - ✓ #异常处理_try except unchecked exception

异常处理机制

异常处理_try except unchecked exception.py - F:\北邮课程\python程序设计\备课\

File Edit Format Run Options Window Help

```
str1="Python"
for i in range(7):
    print(str1[i])
```

异常处理机制

- 异常的其他用法

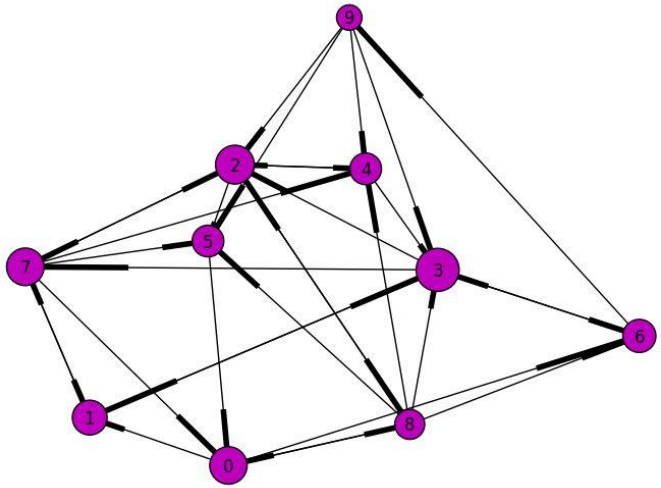
```
try:
<语句>          #运行别的代码
except <名字>:
<语句>          #如果在try部份引发了'name'异常
except <名字>, <数据>:
<语句>          #如果引发了'name'异常，获得附加的数据
else:
<语句>          #如果没有异常发生
```

- 此处的else语句与for循环和while中的else一样，当try语句中的语句块正常执行结束且没有发生异常时，else后的语句块执行

异常处理机制

- Python能识别多种异常类型，但是不建议编写程序时候过度依赖
- try-except异常一般只用来检测极少发生的情况，例如用户输入的合规性或打开文件是否成功等
- 对于面向商业应用的软件，稳定性和可靠性是最重要的衡量指标之一，不会滥用try-except类型语句。采用try-except会影响代码的可读性，增加维护难度

课后思考题



思考题

- 1、利用嵌套循环，输出2-100之间的质数。
- 2、有四个数字：1、2、3、4，能组成多少个互不相同且无重复数字的三位数？各是多少？
- 3、输出9*9乘法口诀表。

```
1*1=1
2*1=2  2*2=4
3*1=3  3*2=6  3*3=9
4*1=4  4*2=8  4*3=12  4*4=16
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25
6*1=6  6*2=12  6*3=18  6*4=24  6*5=30  6*6=36
7*1=7  7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
8*1=8  8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64
9*1=9  9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

谢谢