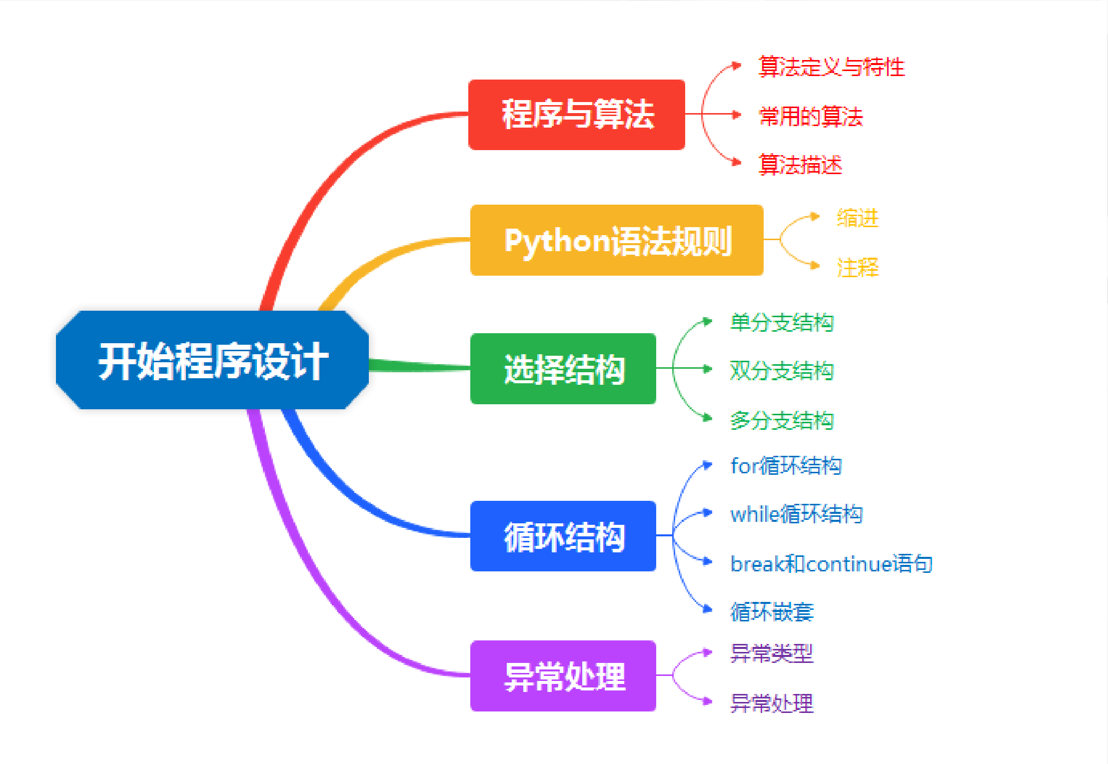
第3章 开始程序设计

史秋实

目录

Contents



开始程序设计
程序与算法
算法定义与特性
常用的算法
算法描述
Python语法规则
缩进
注释
选择结构
单分支结构
双分支结构
多分支结构
循环结构
for循环结构
while循环结构
break和continue语句
循环嵌套
异常处理
异常类型
异常处理

01

程序与算法

1.算法定义与特性

算法就是解决问题的方法和步骤，解决问题的过程就是算法实现的过程。

著名计算机科学家Donald E. Knuth曾把算法的特性归纳为以下5点：

（1）有穷性。任意一个算法在执行有穷个计算步骤后必须终止。

（2）每一个计算步骤必须是精确地定义，无二义性。

（3）可行性。有限多个步骤应该在一个合理的范围内进行。

（4）输入。一般有0个或多个输入。

（5）输出。一般有若干个输出信息，是反映对输入数据加工后的结果。没有输出结果的算法是毫无意义的。

2.常用的算法

（1）枚举

枚举法亦称穷举法或试凑法。它的基本思想是采用搜索的方法，根据题目的部分条件确定答案的大致搜索范围，然后在此范围内对所有可能的情况逐一验证，直到所有情况验证完。若某个情况符合题目的条件，则为本题的一个答案；若全部情况验证完后均不符合题目的条件，则问题无解。枚举法是一种比较耗时的算法，其利用计算机快速运算的特点。枚举的思想可解决许多问题。

2.常用的算法

（2）查找

查找在我们日常生活中经常会遇到，利用计算机快速运算的特点，可方便地实现查找。

查找的方法很多，对不同的数结构有对应的方法。

对无序数据，用顺序查找；

对有序数据，采用二分法查找；

对某些复杂的站构的查找，可用树形查找方法。

顺序查找是根据查找的关键值与数组中的元素逐一比较。

顺序查找对数组中的数不要求有序，查找效率比较低。

2.常用的算法

（3）排序

在日常生活和工作中，许多问题的处理过程都要依赖于数据的有序性，因此，我们需要将数据整理为有序数据，即排序，常用的排序算法有：选择排序和冒泡排序。

选择排序是最为简单且易于理解的算法，基本方法是每次在无序数中找到最小数的下标，然后与第一个位置的数交换。

冒泡排序与选择排序相似，选择排序在每一轮中进行寻找最值小（递增次序）的下标，然后与应放位置的数交换位置。而冒泡排序在每一轮排序时将相邻两个数组元素进行比较，次序不对时立即交换位置，一轮比较结束小数上浮，大数沉底。有n个数则进行n-1轮上述操作。

2.常用的算法

（4） 迭代

迭代法又称递推法，是利用问题本身所具有的某种递推关系求解问题的一种方法。其基本思想是从初值出发，归纳出新值与旧值间直到最后值为止存在的关系，从而把一个复杂的计算过程转化为简单过程的多次重复，每次重复都从旧值的基础上递推出新值，并由新值代替旧值。

除上述四个常用算法之外，还有诸如贪心算法、分治法、回溯法等方法也应用在程序设计中，需要根据不同的情况选择适合的算法来解决问题。

3.算法描述

算法的表示方法有很多，常用的有自然语言、传统的流程图、伪代码和计算机语言等。目前使用较为广泛的是流程图（便于理清程序的处理过程）和计算机语言（用于在计算机上实现程序功能）。

3.算法描述

（1）自然语言

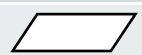
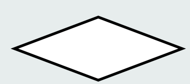
用人们使用的语言，即自然语言描述算法。用自然语言描述算法通俗易懂，但存在以下缺陷：

- 易产生歧义性，往往需要根据上下文才能判别其含义，不太严格。
- 语句比较烦琐、冗长，并且很清楚地表达算法的逻辑流程，尤其对描述含有选择、循环结构的算法，不太方便和直观。

3.算法描述

（2）流程图

流程图是描述算法的常用工具，采用一些图框、线条以及文字说明来形象、直观地描述算法处理过程。美国国家标准化协会（ANSI）规定了一些常用的流程图符号，如表3-1所示。

符号名称	图形	功能
开始与结束框		表示一个过程的开始或结束
输入/输出框		表示数据的输入和输出
处理框		表示算法中的各种处理操作
判断框		表示算法中的条件判断操作
流程线		表示算法的执行方向

3.算法描述

（3）伪代码

由于绘制流程图较费时，自然语言易产生歧义性和难以清楚地表达算法的逻辑流程等缺陷，因而采用伪代码。伪代码产生于20世纪70年代，也是一种描述程序设计逻辑的工具。

伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法。有如下简单约定。

如计算两数之的伪代码如图3-2所示。

```
BEGIN
    x←1
    y←2
    s=x+y
Print t
End
```

3.算法描述

（4）计算机语言

计算机无法识别自然语言、流程图、伪代码。这些方法仅为了帮助人们描述、理解算法，要用计算机解题，就要用计算机语言描述算法。只有用计算机语言编写的程序才能被计算机执行。用计算机语言描述算法必须严格遵循所选择的编程语言的语法规则。

如计算两数之和的Python程序代码如下。

```
x,y=1,2
s=x+y
print(s)
```

02

Python语法规则

1.缩进

Python用缩进来标识代码块，它有着严格的缩进规则。缩进是指代码行前面的空白区域，表示代码之间的层次关系，同一层次的代码块必须有相同的缩进。在编写代码时，一般用Tab键实现缩进。

2.注释

注释是程序员在代码中加入的说明性文字，用来对变量、语句、方法等进行功能性说明，提高代码的可读性。程序运行时编译器或者解释器会忽略注释文字，因此注释不影响程序的运行结果。在Python中注释方式有两种写法。单行注释语句用#开始，单行注释可以单独出现在一行，也可以与代码放在同一行。多行注释语句使用连续3个双引号或者单引号对表示。注释的写法如下。

```
#这是单行注释
print("Hello,python!")    #这是单行注释
'''
这是多行注释的第1行
这是多行注释的第2行
这是多行注释的第3行
'''
```

03

选择结构

2.双分支结构

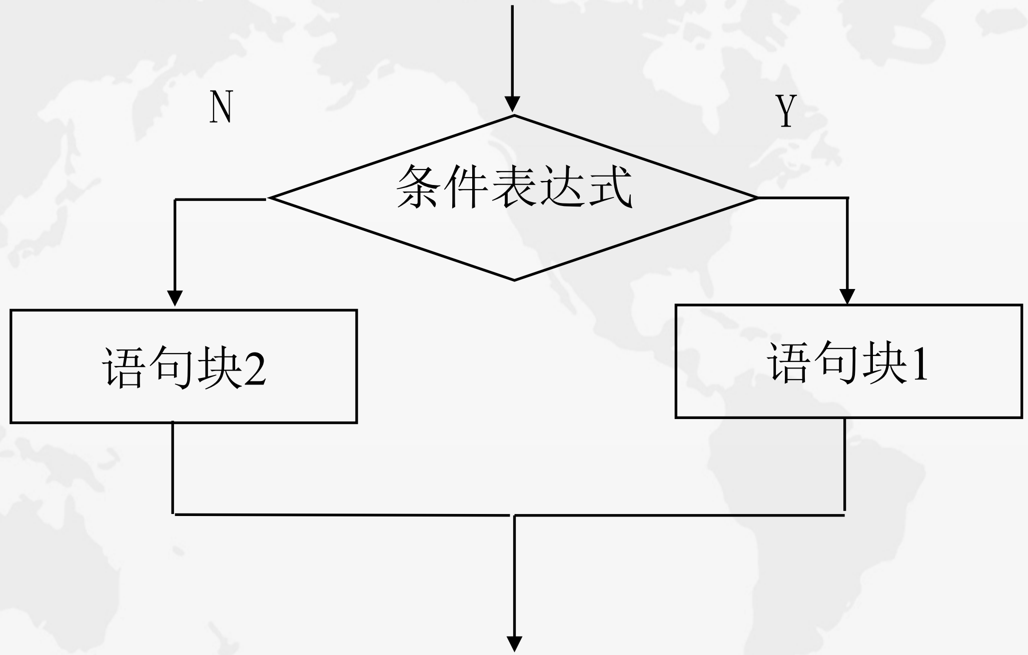
基本语法：

if条件:

　　语句1

else:

　　语句2



3.多分支结构

基本语法：

if 条件1:

    语句1

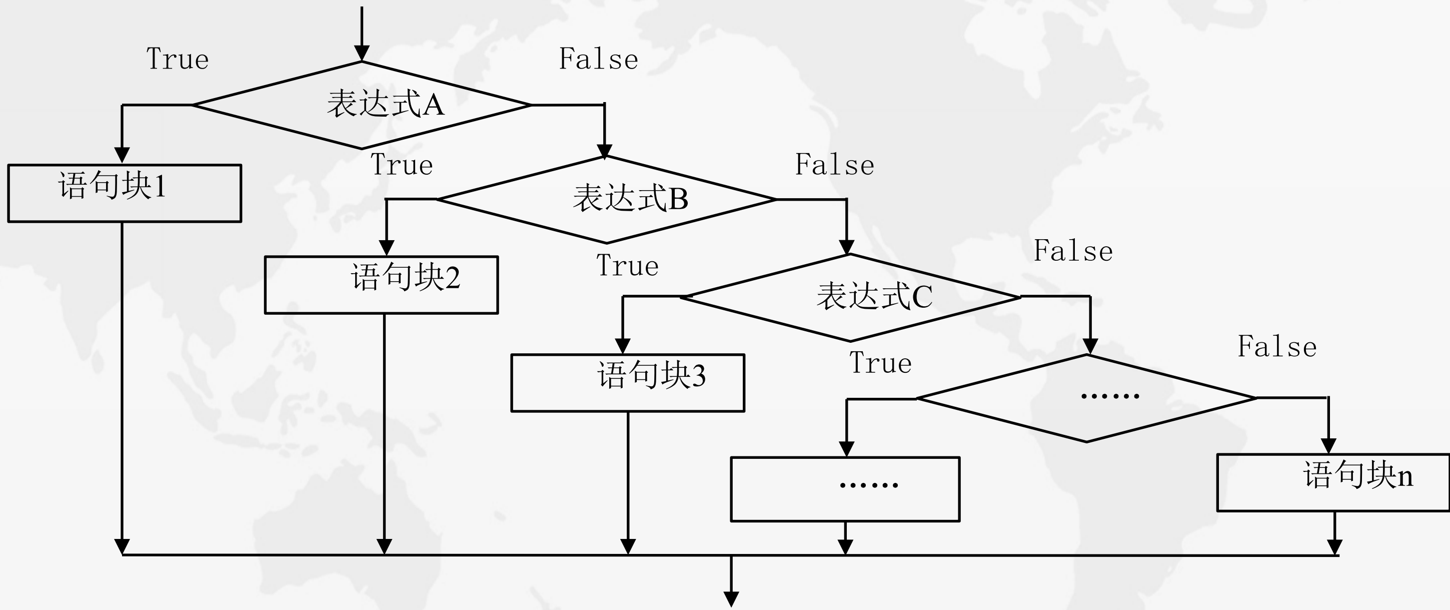
elif 条件2:

    语句2

...

else:

    语句n



04

循环结构

1. for循环结构

for循环表示当满足条件表达式时，重复执行循环语句，否则跳出循环.

其语法格式为：

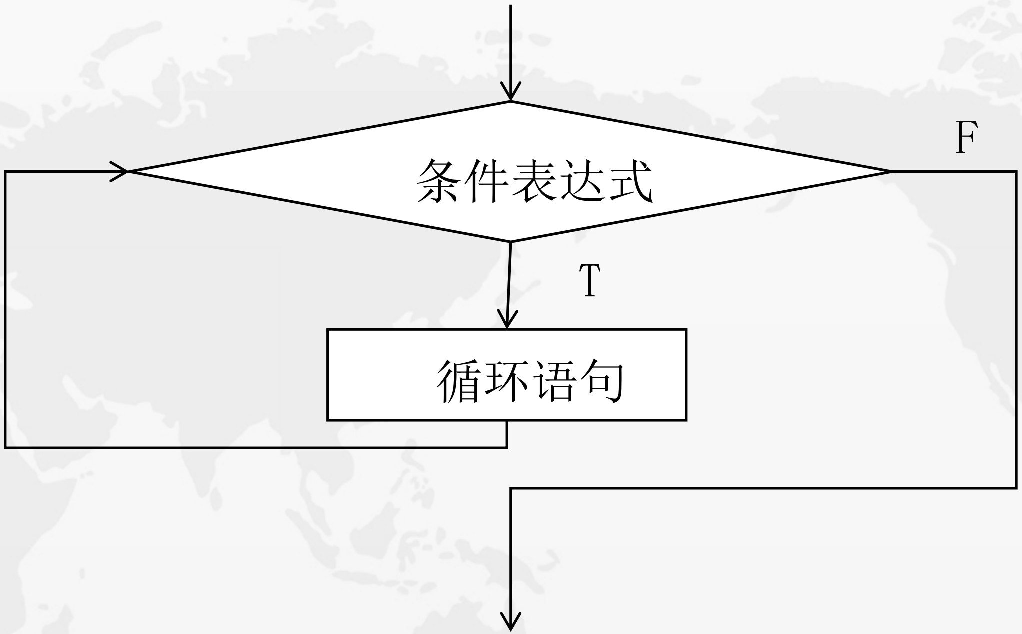
```
for 变量 in range(start, stop[, step]):
    循环语句
```

其中，start指计数开始值，默认为0。如range（5）等价于range（0，5）。

stop指计数结束值，但不包括stop。如range（0，5）是[0，1，2，3，4]没有5。

step指步长，默认为1。如range(0，5，2)步长是2，则它的结果为[0，2，4]。

1. for循环结构



2. while循环结构

for语句适用于明确变量范围的情景，程序更为简洁。当变量范围不确定时， 则需要使用while语句。其语法格式为：

```
while 条件表达式:
    循环语句（块）
```

3.break和continue语句

（1）break中断语句

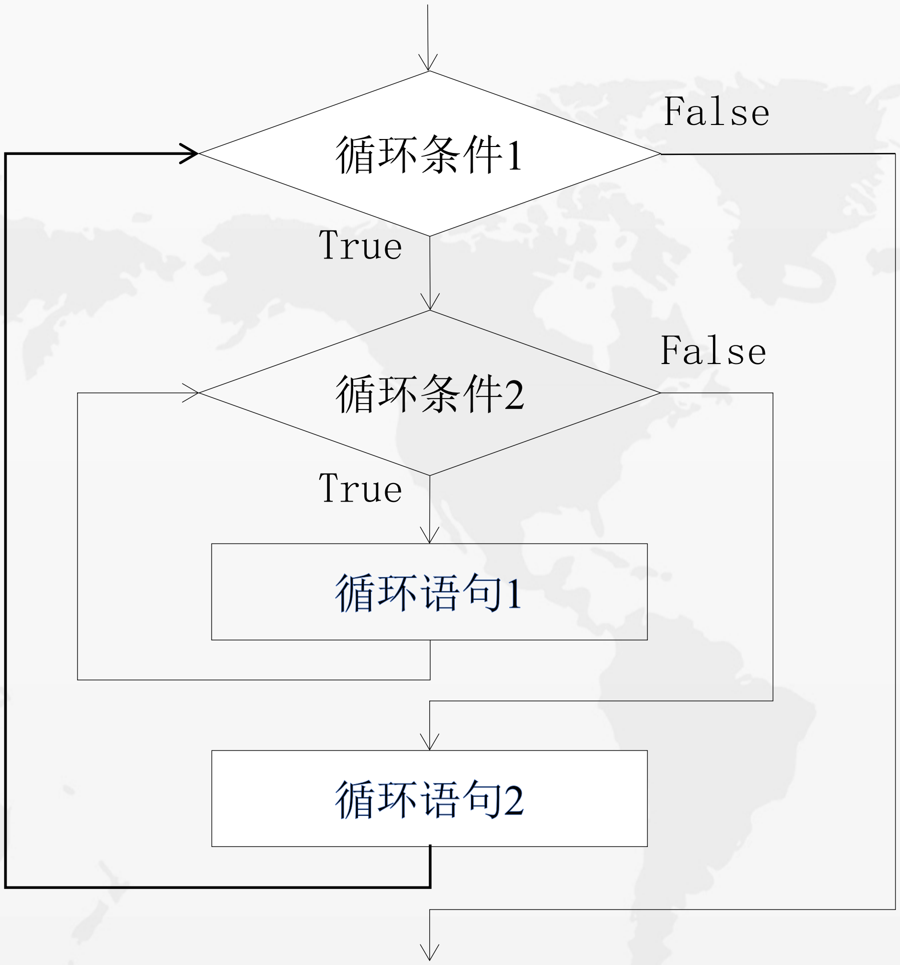
在循环的过程中，如果需要中断循环，则需要使用break语句，可以极大地节省程序的时间复杂度，且便于获取中断时的变量值。

（2）continue语句

continue的作用是用来结束本次循环，紧接着执行下一次的循环。

4.循环嵌套

在一个循环体内又包含另一个完整的循环结构，成为循环的嵌套。当外层循环执行第1遍时，内层循环需要全部执行完方可执行外层循环第2遍。



05

异常处理

1.异常类型

程序设计要考虑各种可能出现的错误，即使程序语法是正确的，运行时也可能会报错，影响整个程序的运行。因此Python设计了强大的异常处理机制，对程序执行过程中出现的异常进行捕获并处理，使程序能够继续执行下去。 Python提供了强大的异常类Exception，可以向用户准确反馈出错信息。

1.异常类型

异常或错误名称	功能描述	异常或错误名称	功能描述
BaseException	所有异常的基类	NameError	未声明/初始化对象 (没有属性)
SystemExit	解释器请求退出	UnboundLocalError	访问未初始化的本地变量
KeyboardInterrupt	用户中断执行(通常是输入Ctrl+C)	ReferenceError	弱引用(Weak reference)试图访问已经被垃圾回收了的对象
Exception	常规错误的基类	RuntimeError	一般的运行时错误
StopIteration	迭代器没有更多的值	NotImplementedError	尚未实现的方法
GeneratorExit	生成器(generator)发生异常来通知退出	SyntaxError	Python 语法错误
StandardError	所有的内建标准异常的基类	IndentationError	缩进错误
ArithmeticError	所有数值计算错误的基类	TabError	Tab 和空格混用
FloatingPointError	浮点计算错误	SystemError	一般的解释器系统错误
OverflowError	数值运算超出最大限制	TypeError	对类型无效的操作
ZeroDivisionError	除(或取模)零 (所有数据类型)	ValueError	传入无效的参数
AssertionError	断言语句失败	UnicodeError	Unicode 相关的错误
AttributeError	对象没有这个属性	UnicodeDecodeError	Unicode 解码时的错误
EOFError	没有内建输入，到达EOF 标记	UnicodeEncodeError	Unicode 编码时错误
EnvironmentError	操作系统错误的基类	UnicodeTranslateError	Unicode 转换时错误
IOError	输入/输出操作失败	Warning	警告的基类
OSError	操作系统错误	DeprecationWarning	关于被弃用的特征的警告
WindowsError	系统调用失败	FutureWarning	关于构造将来语义会有改变的警告
ImportError	导入模块/对象失败	OverflowWarning	旧的关于自动提升为长整型(long)的警告
LookupError	无效数据查询的基类	PendingDeprecationWarning	关于特性将会被废弃的警告
IndexError	序列中没有此索引(index)	RuntimeWarning	可疑的运行时行为(runtime behavior)的警告
KeyError	映射中没有这个键	SyntaxWarning	可疑的语法的警告
UserWarning	用户代码生成的警告	MemoryError	内存溢出错误(对于Python 解释器不是致命的)

2.异常处理

（1） try…except…结构

语法结构为：

```
try:
    代码块1            #此处放置可能引起异常的语句
except Exception [as reason]:
    代码块2    #如果try中代码块不能正常执行，则执行此处代码块进行异常处理
```

2.异常处理

（2）try...except...else...结构

语法结构为：

```
try:
    代码块1                    #此处放置可能引起异常的语句
except Exception [as reason]:
    代码块2   #如果try中代码块不能正常执行，则执行此处代码块进行异常处理
else:
    代码块3 #如果try中的代码没有引发异常，则在执行完try中的代码后，继续执行此处代码块
```

2.异常处理

（3）带有try...多个except...的结构

其语法结构为：

```
try:
    代码块1                #可能引发异常的代码块
except Exception1:             # Exception1是可能出现的一种异常
    代码块2                    #处理Exception1异常的语句
except Exception2:             # Exception2是可能出现的另种异常
    代码块3                    #处理Exception2异常的语句
[else:]                  #可以没有else子句块
    代码块4    #在以上所有Exception异常都未出现时，执行此处代码块
```

2.异常处理

（4）try...except...finally...结构

语法结构为：

```
try:
    代码块1                      #可能引发异常的代码块
except Exception [as reason]:   #此语句也可直接写成except:
    代码块2                      #用来处理异常的代码块
finally
    代码块3      #无论异常是否发生此处代码块都会执行
```