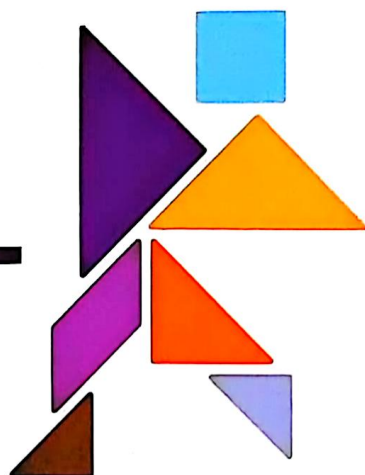




21世纪高等学校计算机
应用技术系列教材

Python 程序设计

(思政版) 微课视频版



◎ 王霞 王书芹 郭小荟 梁银 刘小洋 宋杰鹏 魏思政 编著

《教学课件》

《教学大纲》

640 分钟
视频讲解

《知识导图》

《思政元素》

《程序源码》

《习题答案》

《丰富实例》

《实战任务》

清华大学出版社



目 录

第 1 章 绪论	1
1.1 Python 简介	1
1.1.1 Python 的发展历程及趋势	2
1.1.2 Python 的优缺点	3
1.1.3 Python 的应用领域	3
1.2 Python 的安装和使用	4
1.2.1 Python 的下载	4
1.2.2 Python 的安装	5
1.2.3 Python 的执行	8
1.2.4 Python 文件的执行过程	9
1.3 Python 集成开发环境——PyCharm	10
1.3.1 PyCharm 的下载	10
1.3.2 PyCharm 的安装	11
1.3.3 PyCharm 的简单使用	13
1.3.4 PyCharm 的常用快捷键	16
1.4 Python 发展历程——创新共享	16
1.4.1 Python 的创新	16
1.4.2 Python 的共享	17
1.4.3 对我们的启示	17
1.5 本章小结	18
1.6 巩固训练	18
第 2 章 Python 语法基础	19
2.1 关键字	19
2.2 标识符	20
2.3 变量	21
2.3.1 变量的含义	21

2.3.2	变量名和变量值	22
2.3.3	变量在内存的存储	22
2.4	基本数据类型 	23
2.4.1	数字类型	24
2.4.2	字符串类型	25
2.4.3	数据类型转换	31
2.4.4	常用数学函数和字符串函数	31
2.5	运算符和表达式 	32
2.5.1	运算符五要素	32
2.5.2	算术运算符与算术表达式	32
2.5.3	赋值运算符与赋值表达式	33
2.5.4	比较运算符	34
2.5.5	逻辑运算符	35
2.5.6	位运算符	37
2.5.7	成员运算符	38
2.5.8	身份运算符	39
2.5.9	运算符的优先级和结合性总结	40
2.6	标准输入和输出 	41
2.6.1	标准输入函数 input()	41
2.6.2	标准输出函数 print()	42
2.7	良好的编程习惯 	48
2.7.1	注释	48
2.7.2	代码缩进	48
2.7.3	编码规范	49
2.7.4	Python 之禅	50
2.8	国家荣誉称号——家国情怀 	51
2.8.1	案例背景	51
2.8.2	案例任务,分析和实现	51
2.8.3	总结、启示和拓展	52
2.9	本章小结	53
2.10	巩固训练	53
第3章	流程控制	55
3.1	顺序结构 	55
3.2	选择结构 	56
3.2.1	单分支结构	57

3.2.2	双分支结构	58
3.2.3	多分支结构	59
3.2.4	分支结构的嵌套	62
3.3	循环结构 	63
3.3.1	while 循环	63
3.3.2	for...in 循环	66
3.3.3	循环嵌套	70
3.3.4	break 和 continue	73
3.3.5	穷举与迭代	76
3.4	流程控制综合例子 	78
3.5	光盘行动餐饮系统——勤俭节约 	83
3.5.1	案例背景	83
3.5.2	案例任务	84
3.5.3	案例分析和实现	84
3.5.4	总结和启示	86
3.6	本章小结	86
3.7	巩固训练	87
第4章	高级数据类型	89
4.1	列表 	90
4.1.1	列表的创建	90
4.1.2	列表的访问	90
4.1.3	列表的遍历	91
4.1.4	列表元素的增加	92
4.1.5	列表元素的删除	92
4.1.6	列表元素的修改	93
4.1.7	列表元素的排序	93
4.1.8	列表的其他操作	94
4.2	元组 	95
4.2.1	元组的常用操作	95
4.2.2	序列解包	96
4.2.3	列表和元组实例	96
4.3	字典 	97
4.3.1	字典的创建	98
4.3.2	字典元素的访问	98
4.3.3	字典元素的修改	99
4.3.4	字典的遍历	99

4.4	集合 	100
4.4.1	集合的创建	100
4.4.2	集合的常用方法及运算符	101
4.4.3	不可变集合	102
4.5	综合例子	102
4.6	法治中国,任重道远,先从排队做起 	105
4.6.1	案例背景	105
4.6.2	案例任务	105
4.6.3	案例分析与实现	106
4.6.4	总结和启示	107
4.7	本章小结	107
4.8	巩固训练	107
第5章	函数	108
5.1	内置函数	109
5.2	自定义函数 	109
5.2.1	自定义函数的定义	109
5.2.2	自定义函数的调用	110
5.2.3	形式参数和实际参数	111
5.2.4	参数传递	113
5.3	函数特殊参数 	114
5.3.1	默认参数	114
5.3.2	关键字参数	115
5.3.3	可变长度参数	116
5.4	lambda 函数 	120
5.5	变量的作用域 	120
5.5.1	局部变量	121
5.5.2	全局变量	122
5.5.3	global 关键字	123
5.6	递归函数 	125
5.7	综合例子	126
5.8	垃圾分类——共创美好家园 	131
5.8.1	案例背景	131
5.8.2	案例任务	131
5.8.3	案例分析与实现	131
5.8.4	总结和启示	132

5.9	本章小结	132
5.10	巩固训练	133
第6章 面向对象程序设计		134
6.1	类与对象 	135
6.1.1	类的定义	135
6.1.2	类的实例化结果——实例对象	136
6.1.3	类成员的可访问范围	138
6.2	属性 	138
6.2.1	实例属性	139
6.2.2	类属性	142
6.2.3	特殊属性	143
6.2.4	动态添加/删除属性	145
6.3	方法 	146
6.3.1	实例方法	146
6.3.2	类方法	148
6.3.3	静态方法	149
6.3.4	特殊方法	150
6.3.5	动态添加/删除方法	152
6.4	运算符重载 	154
6.5	继承 	156
6.5.1	相关概念	156
6.5.2	单继承	156
6.5.3	多继承及 MRO 顺序	158
6.6	多态性 	161
6.7	综合例子 	162
6.8	北斗卫星导航系统——科技强国	164
6.8.1	案例背景	164
6.8.2	案例任务	164
6.8.3	案例分析与实现	164
6.8.4	总结和启示	165
6.9	本章小结	166
6.10	巩固训练	166
第7章 文件和目录操作		167
7.1	文件的概念 	167

7.2	文件的常用操作 	168
7.2.1	文件的打开	168
7.2.2	文件的关闭	169
7.2.3	文件的读写	170
7.3	文本文件操作 	170
7.4	二进制文件操作 	173
7.4.1	struct 模块	174
7.4.2	pickle 模块	176
7.5	csv 格式文件的操作 	178
7.5.1	什么是 csv 文件	178
7.5.2	csv 文件的写入	179
7.5.3	csv 文件的读取	181
7.6	文件与目录操作 	183
7.6.1	os 和 os.path 模块	183
7.6.2	shutil 模块	185
7.7	文件的压缩与解压缩 	186
7.7.1	zipfile 模块	186
7.7.2	tarfile 模块	188
7.8	综合例子 	189
7.9	中国诗词大会——寻文化基因、品生活之美 	191
7.9.1	案例背景	191
7.9.2	案例任务	192
7.9.3	案例分析与实现	192
7.9.4	总结和启示	195
7.10	本章小结	195
7.11	巩固训练	196
第 8 章	异常处理	197
8.1	异常的概念 	197
8.2	Python 异常类 	198
8.3	异常处理 	199
8.3.1	异常处理结构	199
8.3.2	抛出异常语句	200
8.4	自定义异常类 	201

8.5 断言 (👤)	202
8.6 大国工匠——匠心筑梦 (👤)	204
8.6.1 案例背景	204
8.6.2 案例任务	204
8.6.3 案例分析与实现	205
8.6.4 总结和启示	206
8.7 本章小结	206
8.8 巩固训练	206
第9章 Python 综合应用实例	207
9.1 实例引入	208
9.2 新冠肺炎疫情数据的获取 (👤)	208
9.2.1 网络爬虫的基本流程	209
9.2.2 所需库的安装	209
9.2.3 爬取海外新冠肺炎疫情历史数据	213
9.2.4 爬取国内新冠肺炎疫情历史数据	216
9.2.5 爬取海外新冠肺炎疫情实时数据	220
9.2.6 爬取国内新冠肺炎疫情实时数据	222
9.3 新冠肺炎疫情数据的可视化展示 (👤)	225
9.3.1 读取新冠肺炎疫情数据	225
9.3.2 数据可视化 matplotlib 库	226
9.3.3 绘制确诊人数趋势曲线图	227
9.3.4 绘制确诊人数词云图	230
9.3.5 绘制国内确诊人数南丁格尔玫瑰图	234
9.4 本章小结	239
参考文献	240

```

6         y = 2 * x - 1
7     else:
8         y = 3 * x - 11
9     print("x = " + str(x) + ", f(x) = " + str(y))

```

运行结果：

input x: 0.5 x = 0.5, f(x) = 0.5	input x: 5 x = 5.0, f(x) = 9.0	input x: 20 x = 20.0, f(x) = 49.0
-------------------------------------	-----------------------------------	--------------------------------------

当然,实例 3.8 也可以不使用嵌套语句实现,而使用多分支结构实现。

```

1 x = float(input("input x: "))
2 if x <= 1:
3     y = x
4 elif x < 10:
5     y = 2 * x - 1
6 else:
7     y = 3 * x - 11
8 print("x = " + str(x) + ", f(x) = " + str(y))

```

很明显,多分支结构比分支嵌套可读性更强,Python 之禅中有一句话:“Flat is better than nested.”,扁平化总比嵌套好,所以能扁平化时尽量不要用嵌套。

3.3 循环结构

如果需要重复执行某条或者某些指令,例如“中国诗词大赛”中的“飞花令”,选手要根据给定的关键字,在给定的时间内轮流背诵含有关键字的诗句,直至时间结束。重复执行类似动作就是循环结构。Python 提供两种循环结构语句: while 循环和 for...in 循环。前者根据条件返回值的情况决定是否执行循环体,后者采用遍历的形式指定循环范围。要更加灵活地操纵循环语句的流向,还需要使用 break、continue 和 pass 等语句。

3.3.1 while 循环

while 循环也称为无限循环,是由条件控制的循环运行方式,一般用于循环次数难以提前确定的情况。while 循环的语法格式为:

```

while 条件表达式:
    循环体
[else:
    语句块]

```

其中,“条件表达式”可以是任何非空或者非零的表达式,“循环体”可以是单条语句或语句块,方括号内的 else 子句可以省略。

流程图如图 3.5 所示。

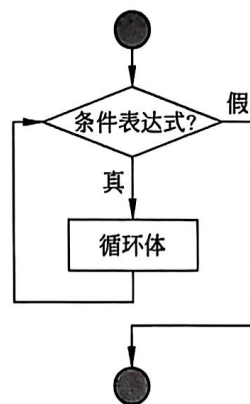


图 3.5 while 循环结构流程图



视频讲解

63

第 3 章

流程控制

执行过程为先判断条件表达式,如果结果为真,则执行循环体,继续进行条件判断;否则循环结束。

【实例 3.9】 求 $\sum_{i=1}^{100} i$ 。

算法分析:设计循环算法需要考虑循环三要素:循环初值、结束条件以及增量(步长)。本例中,循环变量为 i ,初值为 1,结束条件或者终值为 100,步长为 1。另外还需要一个变量存储累加和,其初值为 0。对应的结构流程图如图 3.6 所示。

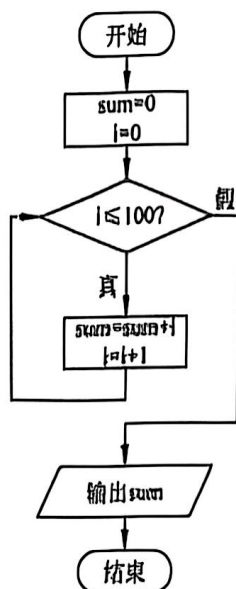


图 3.6 求累加和结构流程图

源代码:

```

1 sum, i = 0, 0
2 while i <= 100:
3     sum += i
4     i = i + 1
5 print("sum = " + str(sum))
  
```

运行结果:

sum = 5050

拓展: $1+3+\dots+99$ 、 $\prod_{i=1}^{100} i$ 、 $\sum_{i=1}^n i$ 、 $\sum_{i=m}^n i$ 等类似累加和或者累乘积的计算。

【实例 3.10】 求若干个学生某门课程的平均成绩。

算法分析:循环变量为学生人数,初值为 0,终值为学生个数 n ,步长为 1。循环体累加每个学生的成绩,循环结束后求成绩的平均值。其结构流程图如图 3.7 所示。

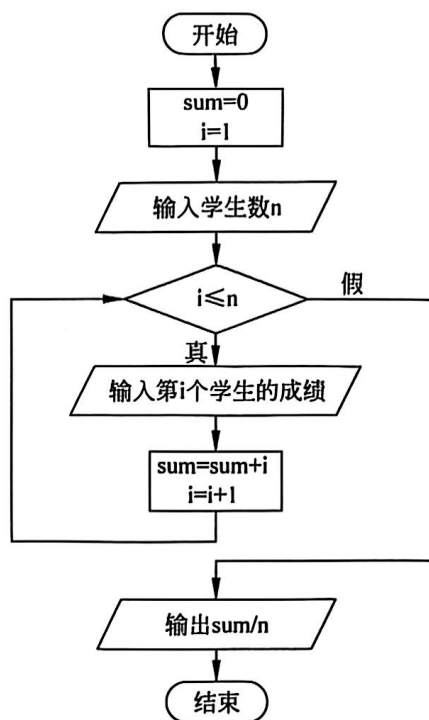


图 3.7 求平均成绩结构流程图

源代码：

```

1 sum,i = 0,1
2 n = int(input("请输入学生人数: "))
3 while i <= n:
4     score = float(input("NO " + str(i) + ": "))
5     sum += score
6     i = i + 1
7 print("平均成绩为 " + str(sum / n))
  
```

运行结果：

```

请输入学生人数: 5
NO 1: 100
NO 2: 85.5
NO 3: 98.7
NO 4: 95
NO 5: 65
平均成绩为 88.84
  
```

循环结构中也可以使用 else 子句,用于表示不满足循环条件时程序的执行流程。

【实例 3.11】 循环结构使用 else 子句。

源代码：

```

1 count = 0
2 while count < 5:
3     print(str(count) + " is less than 5.")
  
```

```

4     count = count + 1
5     else:
6         print(str(count) + " is not less than 5.")

```

运行结果:

```

0 is less than 5.
1 is less than 5.
2 is less than 5.
3 is less than 5.
4 is less than 5.
5 is not less than 5.

```

可见,当满足循环条件“ $\text{count} < 5$ ”时,执行循环体,当不满足循环条件时,执行“ $\text{print}(\text{str}(\text{count}) + \text{" is not less than 5."})$ ”。



视频讲解

3.3.2 for...in 循环

Python 提供的另一种循环结构是 for...in 循环。Python 提供的 for 循环语句与 Java、C++ 等编程语言提供的 for 语句不同,更像是 shell 或是脚本语言中的 for 循环,可以遍历如列表、元组、字符串等序列成员,也可以用在列表解析和生成器表达式中。

1. 使用序列项迭代序列对象

通过 for...in 循环可以迭代序列对象的所有成员,并在迭代结束后,自动结束循环,其语法如下:

```

for iterating_var in list:
    循环体

```

其中,iterating_var 为迭代变量,list 为序列(字符串、列表、元组、字典、集合)。执行时,迭代变量依次取序列中元素的值,直至取完,循环退出。

对应的结构流程图如图 3.8 所示。

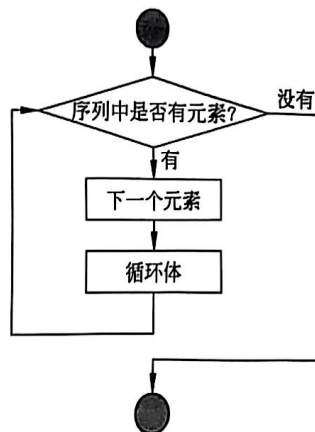


图 3.8 迭代序列 for...in 循环结构流程图

【实例 3.12】 统计字符串中各类字符的个数。

算法分析：使用迭代变量遍历序列(字符串)中的每一个元素,分别判断所属类型,并将对应个数加 1,直至遍历结束。结构流程图如图 3.9 所示。

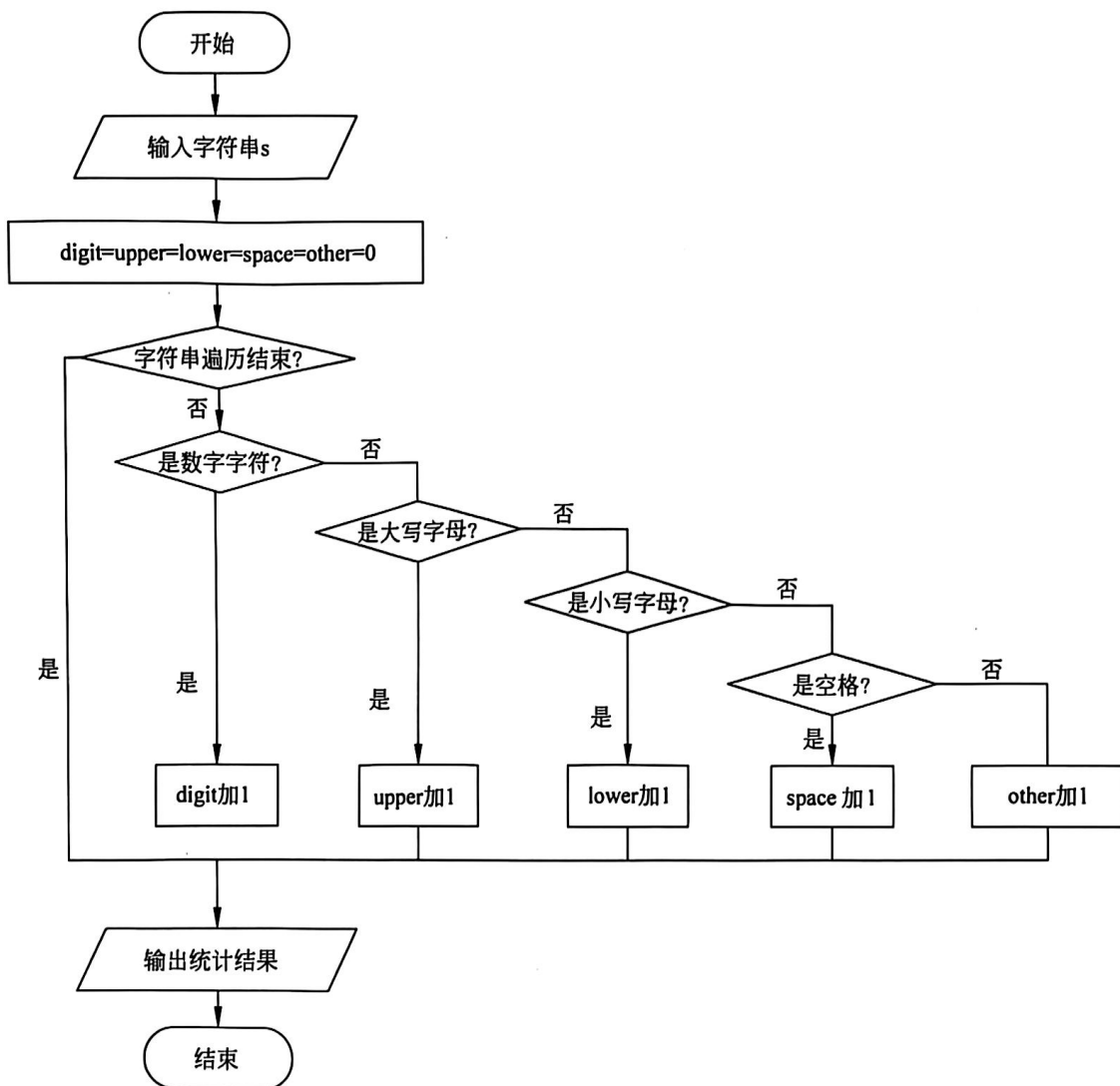


图 3.9 统计字符串中各类字符个数的结构流程图

源代码：

```
1 s = input("请输入一个字符串: ")
2 digit, upper, lower, space, other = 0, 0, 0, 0, 0 # 数字字符, 大写字符, 小写字符, 空格字符,
# 其他字符的个数
3 for i in s:
4     if i >= '0' and i <= '9': # 判断 i 是否为数字字符
5         digit = digit + 1
6     elif i >= 'A' and i <= 'Z': # 判断 i 是否为大写字符
7         upper = upper + 1
8     elif i >= 'a' and i <= 'z': # 判断 i 是否为小写字母
```

```

9         lower = lower + 1
10        elif i == ' ': # 判断 i 是否为空格字符
11            space = space + 1
12        else: # 其他字符
13            other = other + 1
14    print("数字字符: %d\n大写字母: %d\n小写字母: %d\n空格字符: %d\n其他字符: %d\n" % (digit, upper, lower, space, other))
15

```

运行结果:

```

请输入一个字符串: Life is short, we need Python!
数字字符: 0
大写字母: 2
小写字母: 21
空格字符: 5
其他字符: 2

```

其中,实现字符分类的循环体也可用内置函数代替,如:

```

1    if i.isdigit(): # 判断 i 是否为数字字符
2        digit = digit + 1
3    elif i.isupper(): # 判断 i 是否为大写字母
4        upper = upper + 1
5    elif i.islower(): # 判断 i 是否为小写字母
6        lower = lower + 1
7    elif i.isspace(): # 判断 i 是否为空格字符
8        space = space + 1
9    else:
10        other = other + 1

```

内置函数 `i.isdigit()`、`i.isupper()`、`i.islower()`、`i.isspace()` 分别用来判断 `i` 是否为数字字符、大写字母、小写字母和空格字符。

2. 使用序列索引迭代序列对象

在 `for...in` 循环结构中,也可以使用序列索引来遍历列表,语法如下:

```

for index in range(len(list)):
    循环体

```

其中,`index` 为序列的索引项,内置函数 `range()` 为计数函数,`len()` 获取序列长度。

【实例 3.13】 统计字符串中各类字符的个数(range 版)。

```

1    s = input("请输入一个字符串: ")
2    digit = upper = lower = space = other = 0 # 数字字符、大写字母、小写字母、空格字符
                                              # 和其他字符的个数
3    for i in range(len(s)):
4        if s[i].isdigit(): # 判断 i 是否为数字字符
5            digit = digit + 1

```

```

6      elif s[i].isupper( ):    #判断 i 是否为大写字母
7          upper = upper + 1
8      elif s[i].islower( ):    #判断 i 是否为小写字母
9          lower = lower + 1
10     elif s[i].isspace( ):    #判断 i 是否为空格字符
11         space = space + 1
12     else:
13         other = other + 1
14     print("数字字符: %d\n大写字母: %d\n小写字母: %d\n空格字符: %d\n其他字符: %d\n" % (digit, upper, lower, space, other))
15

```

运行结果:

```

请输入一个字符串: I am a student, I am 20 years old!
数字字符: 2
大写字母: 2
小写字母: 20
空格字符: 8
其他字符: 2

```

使用 range() 函数可以得到用来迭代的索引列表,使用索引下标“[]”可以方便快捷地访问序列对象。另外,还可以使用 range() 函数实现类似 Java、C++ 等传统编程语言的 for 循环结构,即从循环三要素角度出发设计循环结构,语法格式为:

```
range([start,] end[, step = 1])
```

其中,range() 函数会返回一个整数序列,可选项 start 为序列初值(循环变量初值),end 为序列终止值(循环变量终值,且不含 end 本身),可选项 step 为步长或增量,默认为 1。

【实例 3.14】 求 $\sum_{i=1}^{100} i$ (range 版)。

```

1  sum = 0
2  for i in range(1, 101, 1):
3      sum = sum + i
4  print("sum = " + str(sum))

```

运行结果:

```
sum = 5050
```

显然,此时的 for...in 循环与 while 循环完全等价。

3. 使用枚举函数迭代序列对象

Python 内置函数 enumerate() 用于将一个可遍历的数据对象(列表、元组或者字符串)组合成一个索引序列,同时列出数据和下标,一般用于 for...in 循环中。语法格式为:

69

第
3
章

流程控制

```
for index, iterating_var in enumerate(list, start_index = 0):
    循环体
```

其中, `index` 返回索引计数, `iterating_var` 为与索引计数相对应的索引对象成员, `list` 为待遍历的序列对象, `start_index` 为返回的起始索引计数, 默认值为 0。

【实例 3.15】 输出学生花名册。

```
1 name_list = ["李白", "孟浩然", "王维", "李绅"] # name_list 的数据类型为列表 list
2 for index, name in enumerate(name_list):
3     print(index, name)
```

运行结果:

```
0 李白
1 孟浩然
2 王维
3 李绅
```



视频讲解

3.3.3 循环嵌套

允许在一个循环结构中嵌入另一个循环结构, 称为循环嵌套。在 Python 中, `for...in` 循环结构和 `while` 循环结构都可以进行循环嵌套。如:

```
while condition_expression 1:
    for index in range(len(list)):
        循环体
```

不仅 `for` 循环结构可以嵌入到 `while` 循环结构中, `while` 循环结构也可以嵌入到 `for...in` 循环结构中, 同样也可以根据自身需要任意嵌套。

【实例 3.16】 输出九九乘法表(下三角形)。

算法分析: 从结构看, 九九乘法表是二维形式, 单重循环无法实现。从内容看, 第一个乘数每行一致, 第二个乘数同行每列依次加 1, 故使用两重循环。外循环控制第一个乘数(迭代变量为 1~9), 内循环控制第二个乘数。因为要求按照下三角形输出, 所以内循环迭代变量只能为 1~i。结构流程图如图 3.10 所示。

源代码:

```
1 for i in range(1, 10):
2     for j in range(1, i+1):
3         print(str(i) + "*" + str(j) + "=" + str(i * j), end=" ")
4     print() # 每行末尾换行
```

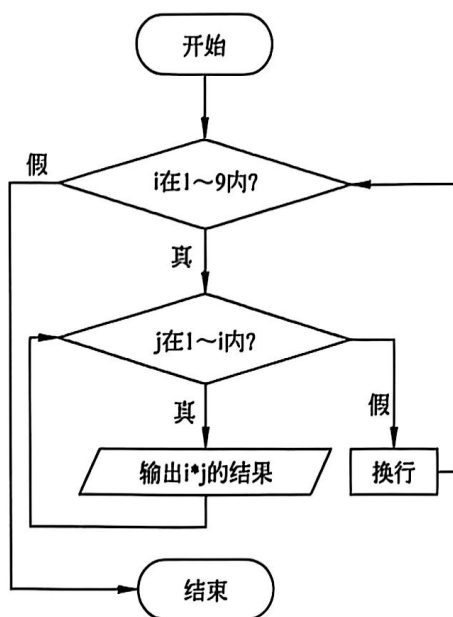


图 3.10 九九乘法表结构流程图

运行结果：

```

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
    
```

拓展：输出上三角九九乘法表、钻石形等图形。

【实例 3.17】 求 $1! + 2! + \dots + 20!$ 。

算法分析：求 $n!$ 需要用循环结构实现，累加和也需要用循环结构实现，故采用双重循环。外循环计算累加和，内循环求 $n!$ 。结构流程图如图 3.11 所示。

源代码：

```

1 sum = 0                # 累加和
2 for i in range(1, 21): # 外循环,用于累加和
3     fact = 1            # 存放 n!
4     for j in range(1, i + 1): # 内循环,用于计算 i!
5         fact = fact * j
6     sum = sum + fact
7 print("sum = " + str(sum))
    
```

运行结果：

```
sum = 2561327494111820313
```

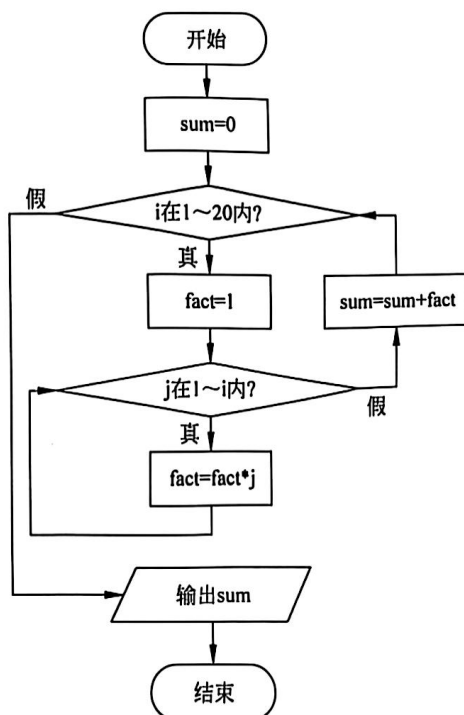


图 3.11 阶乘和结构流程图

观察运行过程可以发现,双重循环计算累加和时,每次都是从1开始计算 $n!$ 。事实上 $n! = (n-1)! * n$,故可以使用单重循环实现。优化后的代码:

```

1  sum = 0          # 累加和
2  fact = 1         # 存放 n!
3  for i in range(1, 21):
4      fact = fact * i # 直接使用 n! = (n-1)! * n 来计算 n!
5      sum = sum + fact
6  print("sum = " + str(sum))
  
```

循环结构中可以嵌套另一个循环结构,也可以嵌套选择结构,反之亦然。

【实例 3.18】 列出 1~200 的所有素数,要求每行输出 10 个数(标志变量版)。

算法分析:素数是一个只能被 1 和本身整除的自然数。判断 n 是否为素数的方法为依次除以 $2 \sim \sqrt{n}$,如果能整除则不是素数。这个过程需要使用单重循环嵌套选择结构实现。而列出 1~200 的所有素数也需要用循环实现,故使用双重循环完成。流程图如图 3.12 所示。

源代码:

```

1  import math      # math 模型库, sqrt() 函数需要使用
2  count = 0        # 累计素数个数
3  for n in range(2, 201):
4      i, flag = 2, True
  
```

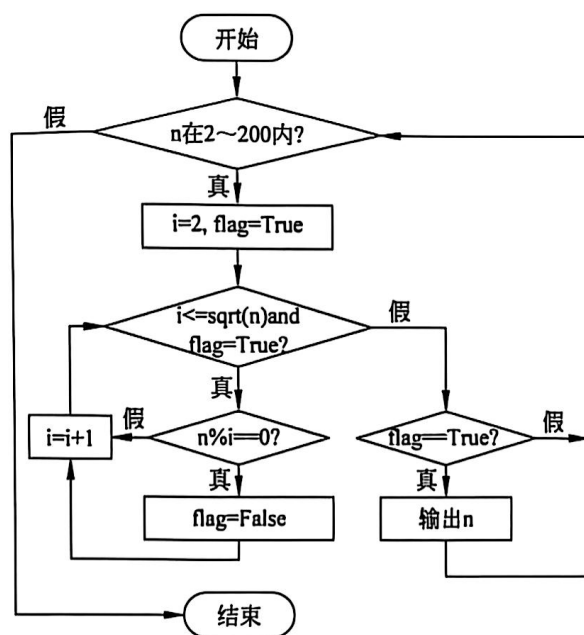


图 3.12 实例 3.18 结构流程图

```

5   while i <= math.sqrt(n) and flag:   # 内循环对每一个数进行是否为素数的判断
6       if n % i == 0:                  # 如果能够整除,则不是素数,flag 置为 False
7           flag = False
8       i = i + 1
9   if flag:                             # 如果是素数,则输出
10      count = count + 1
11      print(n, end="\\t")
12      if count % 10 == 0:              # 控制每行 10 个
13          print()

```

运行结果：

```

2   3   5   7   11  13  17  19  23  29
31  37  41  43  47  53  59  61  67  71
73  79  83  89  97  101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199

```

说明：本实例中外循环使用 for...in 结构，内循环使用 while 结构，并在内循环中嵌入分支结构进行整除判断。引入标志变量 flag 来标志 n 是否为素数（默认值为 True），一旦判断能够整除（即不是素数），则修改 flag 值为 False，内循环条件执行为否，内循环退出。引入计数变量 count 来记录每行输出个数。

3.3.4 break 和 continue

在循环结构中，大多数情况下当循环条件满足时，循环体会一直被执行，直到循环条件不满足。但有时需要在某种条件下使循环提前结束，实现方法有两种：一是使用标志变量，如实例 3.18 中的 flag，通过 flag 值的变化，在循环未正常结束时提前退出循环；二是使用



视频讲解

73

第 3 章

流程控制