2.3 流程控制

2023 年 9 月 8 日

流程控制

流程是指程序运行时语句的执行次序。Python 包含了三种基本流程控制结构：顺序结构、分支结构、循环结构。
顺序结构是从上往下的顺序逐剧执行的结构，先执行位置在前的语句、后执行位置在后的语句。

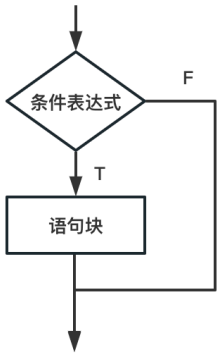
一、分支结构

分支结构又称选择结构，它是一种在两条或更多条执行路径中**选择一条执行**的语句控制结构。

（一）单分支结构

if 语句单分支结构的语法形式为：

```
if(条件表达式):
    语句块
```



若条件表达式的逻辑值为 True 则执行语句块，否则不执行语句块。**if 语句的条件表达式之后需要添加冒号（:），语句块内的每行语句均需要通过缩进表示同属一个语句块**。

运行下述例子：

[58]:
```
grade = int(input('输入成绩后，按【回车键】确认')) #Python 键盘输入默认为字符串，这里做类型转换，把字符串转成整数
if(grade < 60):                                    # 如果输入小数，也就是浮点数，使用 float() 来转换
    print('很不幸，你挂科了！ ')
```

输入成绩后，按【回车键】确认 59.5

很不幸，你挂科了！

练习 if 语句

如果输入成绩在 60 分和 80 分之间，请打印“良好”

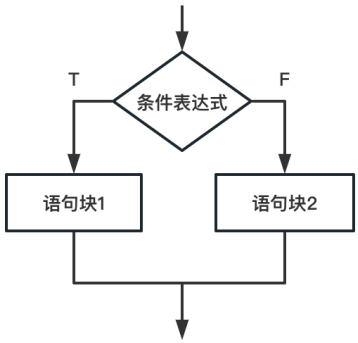
[ ]:
```

```

（二）双分支结构

if 语句双分支结构的语法形式为：

```
if(条件表达式):
    语句块1
else:
    语句块2
```



若条件表达式的逻辑值为 True，则执行语句块 1；否则执行语句块 2。

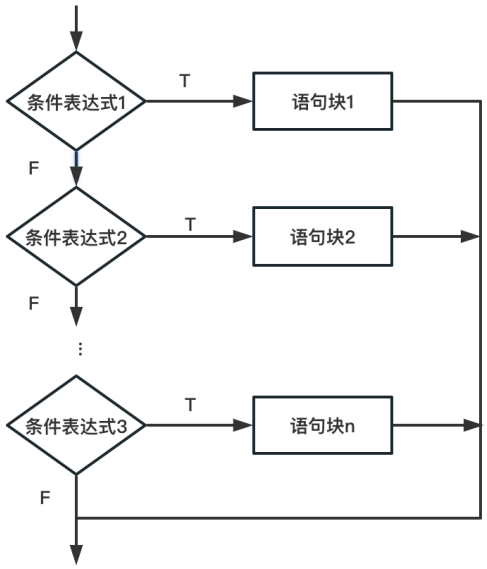
[ ]:
```
grade = int(input('输入成绩后，按【回车键】确认'))
if(grade < 60):
    print('很不幸，你挂科了！ ')
else:                                          # 注意 else 与 if 对齐
```

```
    print('恭喜！通过考试')
```

（三）多分支结构

if 语句多分支结构的语法如下：

```
if(条件表达式1):
    语句块1
elif(条件表达式2):
    语句块2
    ...
elif(条件表达式 n):
    语句块 n
else:
    语句块 n+1
```



[68]:
```
grade = int(input('输入成绩后，按【回车键】确认'))
if(grade < 60):
    print('很不幸，你挂科了！')
elif(grade >= 60 and grade < 80):                    # 注意 elif 与 if 对齐
    print('你及格了！')
elif(grade >= 80 and grade < 90):                    # 注意 elif 与 if 对齐
```

```
    print('表现不错！')
elif(grade >= 90 and grade<=100):                    # 注意 elif 与 if 对齐
    print('优秀！')
else:
    print('您输入的成绩有误！')
```

输入成绩后，按【回车键】确认 90

优秀!

练习：计算小明的体重

小明身高 1.75 米，体重 80.5kg。请根据 BMI 公式（体重除以身高的平方）帮小明计算他的 BMI 指数，并根据 BMI 指数：

- 低于 18.5：过轻
- 18.5-25：正常
- 25-28：过重
- 28-32：肥胖
- 高于 32：严重肥胖

Tips：用 `if-elif` 判断并打印结果

[ ]:

二、循环结构

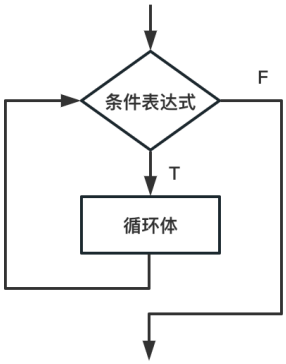
循环结构是在满足一定条件下**反复执行某一段语句块**的流程控制结构。反复被执行的语句块被称为**循环体**。

（一）while 循环

while 循环语句的语法形式为：

```
while(条件表达式):
    循环体
```

while 语句的执行过程是先判断条件表达式的值，若值为 True，则执行循环体，否则将跳过循环体，执行 while 语句后面的语句。每次循环体执行完毕后，再转到条件表达式判断真假，以决定是否再次执行循环体。



这个例子是计算 1 到 100 的自然数的和。

[51]:
```
n = 1                            # n 负责存储自然数，用于遍历自然数 1，2，3，4，5.。。。
100
s = 0                            # s 负责存储求和的结果，初始为 0
while(n<=100):
    s = s + n                    # 求和，并保存求和的结果到 s
    print('n:', n, 's:', s)
    n = n + 1                    # 让自然数 +1，并更新自然数 n

print('The sum from 1 to 100 is', s)
```

```
n: 1 s: 1
n: 2 s: 3
n: 3 s: 6
n: 4 s: 10
n: 5 s: 15
n: 6 s: 21
n: 7 s: 28
n: 8 s: 36
n: 9 s: 45
n: 10 s: 55
n: 11 s: 66
n: 12 s: 78
n: 13 s: 91
n: 14 s: 105
n: 15 s: 120
n: 16 s: 136
```

```
n: 17 s: 153
n: 18 s: 171
n: 19 s: 190
n: 20 s: 210
n: 21 s: 231
n: 22 s: 253
n: 23 s: 276
n: 24 s: 300
n: 25 s: 325
n: 26 s: 351
n: 27 s: 378
n: 28 s: 406
n: 29 s: 435
n: 30 s: 465
n: 31 s: 496
n: 32 s: 528
n: 33 s: 561
n: 34 s: 595
n: 35 s: 630
n: 36 s: 666
n: 37 s: 703
n: 38 s: 741
n: 39 s: 780
n: 40 s: 820
n: 41 s: 861
n: 42 s: 903
n: 43 s: 946
n: 44 s: 990
n: 45 s: 1035
n: 46 s: 1081
n: 47 s: 1128
n: 48 s: 1176
n: 49 s: 1225
n: 50 s: 1275
n: 51 s: 1326
n: 52 s: 1378
n: 53 s: 1431
```

n: 54 s: 1485
n: 55 s: 1540
n: 56 s: 1596
n: 57 s: 1653
n: 58 s: 1711
n: 59 s: 1770
n: 60 s: 1830
n: 61 s: 1891
n: 62 s: 1953
n: 63 s: 2016
n: 64 s: 2080
n: 65 s: 2145
n: 66 s: 2211
n: 67 s: 2278
n: 68 s: 2346
n: 69 s: 2415
n: 70 s: 2485
n: 71 s: 2556
n: 72 s: 2628
n: 73 s: 2701
n: 74 s: 2775
n: 75 s: 2850
n: 76 s: 2926
n: 77 s: 3003
n: 78 s: 3081
n: 79 s: 3160
n: 80 s: 3240
n: 81 s: 3321
n: 82 s: 3403
n: 83 s: 3486
n: 84 s: 3570
n: 85 s: 3655
n: 86 s: 3741
n: 87 s: 3828
n: 88 s: 3916
n: 89 s: 4005
n: 90 s: 4095

```
n: 91 s: 4186
n: 92 s: 4278
n: 93 s: 4371
n: 94 s: 4465
n: 95 s: 4560
n: 96 s: 4656
n: 97 s: 4753
n: 98 s: 4851
n: 99 s: 4950
n: 100 s: 5050
The sum from 1 to 100 is 5050
```

练习

如何求 1，3，5，7，。。99，间隔为 2 的等差数列的和。

[ ]:

（二）for 循环

for 语句通常用来遍历字符串、列表、元祖等序列数据类型中的元素。对序列中的每个元素执行一次相关的循环体。for 语句的语法形式如下：

```
for 变量 in 序列:
    循环体
```

下面这个例子使用 for 语句循环输出列表 list1 中的所有元素。

[11]:
```
list1 = [1,2,3,4,5]
for x in list1:
    print(x)
```

```
1
2
3
4
5
```

此外，for 语句经常使用 range/xrange 函数生成序列。

[12]:
```
for i in range(1,5):
    print(i)
```

```
1
2
3
4
```

使用 range 函数生成序列，请注意它的起止范围。函数 range(1,5) 生成序列 [1,2,3,4]。

（三）break 语句

break 语句用于控制程序提前结束循环而执行循环体后面的语句。

[16]:
```
list2 = []                                          # 定义一个空的列表
while(True):
    x = int(input('请输入数据（当输入-1 时结束）:'))
    if x == -1:
        break                                       # 当输入 =-1 时，则跳出循环
    else:
        list2 = list2 + [x]
print(list2)
```

```
请输入数据（当输入-1 时结束）: 1
请输入数据（当输入-1 时结束）: 2
请输入数据（当输入-1 时结束）: 3
请输入数据（当输入-1 时结束）: -1

[1, 2, 3]
```

（四）continue 语句

continue 语句用于结束本次循环，重新判断条件表达式真假，若为 True，则继续循环，否则，结束循环。

[18]:
```
i=0
while(i<=10):
    i=i+1
    if i%2 != 0:        # 如果 i 是奇数，则 continue，跳至 while 语句，不执行后面的
print
```

```
        continue
    print(i)            # 注意此语句的锁进与上面的 if 对齐，表示其仍在 while 内
```

```
2
4
6
8
10
```

练习：高斯小时候的故事

高斯小时候非常淘气，一次数学课上，老师为了让他们安静下来，给他们列了一道很难的算式，让他们一个小时内算出 1+2+3+4+5+6+……+100 的得数。

`range(101)` 就可以生成 0-100 的整数序列，计算如下：

[ ]:

```

```

总结

1. 循环是让计算机做重复任务的有效的方法。
2. `break` 语句可以在循环过程中直接退出循环，而 `continue` 语句可以提前结束本轮循环，并直接开始下一轮循环。这两个语句通常都必须配合 `if` 语句使用。
3. 要特别注意，不要滥用 `break` 和 `continue` 语句。`break` 和 `continue` 会造成代码执行逻辑分叉过多，容易出错。大多数循环并不需要用到 `break` 和 `continue` 语句，上面的两个例子，都可以通过改写循环条件或者修改循环逻辑，去掉 `break` 和 `continue` 语句。
4. 有些时候，如果代码写得有问题，会让程序陷入“死循环”，也就是永远循环下去。在命令行模式下用 `Ctrl+C` 退出程序，或者在 Jupyter 里点击打断内核或者重启内核。