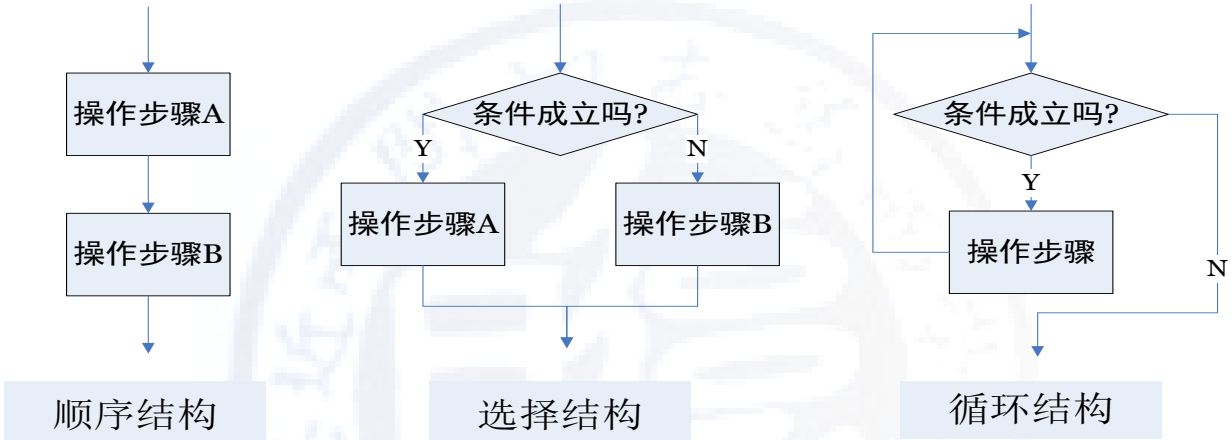
第2章 选择与循环

刘 卉

huiliu@fudan.edu.cn

结构化控制结构



- 三种控制结构可以互相、反复嵌套.
- 能够描述所有可计算问题.

主要内容

- 2.1 条件表达式
- 2.2 选择结构
- 2.3 循环结构
- 2.4 break和continue语句
- 2.5 案例精选

2.1 条件表达式

- 算术运算符: +、-、*、/、//、%、**
- 关系运算符: >、<、==、<=、>=、!=
- 逻辑运算符: and/or/not, 短路特性
- 测试运算符: is/is not

```
>>> a=3
>>> b=4
>>> a is b
False
>>> a is not b
True
```

- 条件表达式的运算结果(值): True/False
- 在选择和循环结构中, 条件表达式的值只要不是0或空, Python解释器均认为与True等价.
- 几乎所有的Python合法表达式都可作为条件表达式, 包括单个对象或函数调用.

```
>>> if 3:    # 使用整数作为条件表达式
    print(5)
5
```

逻辑运算符“and”和“or”的短路特性

- 惰性求值：只计算必须计算的表达式的值.
- "Expr1 and Expr2"：若Expr1的值为“False”或其它等价值，则不论Expr2的值是什么，整个表达式的值都是“False”⇨Expr2将不会被计算.
- "Expr1 or Expr2"：若Expr1的值为“True”或其它等价值，则不论Expr2的值是什么，整个表达式的值都是“True”⇨Expr2不会被计算.
- 在设计条件表达式时，巧妙利用“and”和“or”的短路特性，减少不必要的计算与判断，从而提高程序的运行效率.

- 条件表达式中不允许使用赋值运算符"="
  - 避免了其它语言中误将关系运算符"=="写作赋值运算符"="带来的麻烦.

e.g. 在条件表达式中使用"="将抛出异常, 提示语法错误.

```
>>> if a = 3:
SyntaxError: invalid syntax

>>> if (a = 3) and (b = 4):
SyntaxError: invalid syntax
```

2.2 选择结构

- 2.2.1 单分支选择结构
- 2.2.2 双分支选择结构
- 2.2.3 多分支选择结构
- 2.2.4 选择结构的嵌套
- 2.2.5 选择结构应用案例

2.2.1 单分支选择结构

```
if 条件表达式：
    语句块    # ≥1条语句
```

- 若条件表达式的运算结果为True, 则运行语句块; 为False, 则什么也不做.
- Occasionally, it is useful to have a body with no statements (usually as a place keeper for code you haven't written yet). In that case, you can use the `pass` statement, which does nothing.

```
if x < 0:
    pass  # TODO: need to handle negative values!
```

2.2.2 双分支选择结构

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

```
if x % 2 == 0:
    print('x is even')
else:
    print('x is odd')
```

- Python还支持如下形式的表达式:

```
expr1 if condition else expr2
```

  - 当条件表达式condition的值与True等价时，表达式的值为expr1的运算结果, 否则为expr2的运算结果.
  - expr1和expr2可以是复杂表达式, 包括函数调用.

```
>>> a = int(input())
-5
>>> b = a if a > 0 else -a
>>> b
5
>>> print(a if a > 0 else -a)
5
>>> print(a) if a > 0 else print(-a)
5
```

- 该结构的表达式也具有短路特性

```
>>> x = math.sqrt(9) if 5>3 else random.randint(1, 100) #此时尚未导入math模块

NameError: name 'math' is not defined

>>> import math

>>> x = math.sqrt(9) if 5>3 else random.randint(1, 100) #此时尚未导入random模块,
但由于条件表达式5>3的值为True, 所以正常运行.

>>> x = math.sqrt(9) if 2>3 else random.randint(1, 100) #此时尚未导入random模块,
由于2>3的值为False, 需要计算第二个表达式的值, 因此出错

NameError: name 'random' is not defined

>>> import random
>>> x = math.sqrt(9) if 2>3 else random.randint(1, 100)
```

2.2.3 多分支选择结构

```
if 表达式1:
    语句块1
elif 表达式2:  # 关键字elif是else if的缩写
    语句块2
elif 表达式3:  # 可有多个elif分支
    语句块3
else:
    语句块4
```

各个分支相互排斥，仅有一个分支语句块被执行。

[例] 利用多分支选择结构将成绩从百分制转换为等级制.

```
if score > 100:
    print('wrong score. Score must <= 100.')
elif score >= 90:  # elif 90 <= score <= 100:
    print('A')
elif score >= 80:  # [80, 90)
    print('B')
elif score >= 70:
    print('C')
elif score >= 60:
    print('D')
elif score >= 0:
    print('F')
else:
    print('wrong score. Score must >= 0')
```

各个分支相互排斥

multi_branch.py

Copy&Run

2.2.4 选择结构的嵌套

```
if 表达式1:
    语句块1
    if 表达式2:
        语句块2
    else:
        语句块3
else:
    if 表达式4:
        语句块4
```

注意：各个层次代码的缩进必须正确且一致。

```
if x == y:
   print('x and y are equal')
else:
   if x < y:
      print('x is less than y')
   else:
      print('x is greater than y')
```

- Although the indentation of the statements makes the structure apparent, **nested conditionals** become difficult to read very quickly. It is a good idea to avoid them when you can.

2.2.5 选择结构的应用

例1: 输入三角形的边长a, b, c, 求面积area(小数点后保留2位).

```
import math
a, b, c = eval(input('Enter 3 numbers(separated with comma): '))
if a+b > c and b+c > a and c+a > b:
    s = (a + b + c) / 2
    area = math.sqrt(s * (s - a) * (s - b) * (s - c))
    print('area = %.2f' % area)
else:
    print("Can't make a triangle.")
    area = 0
```

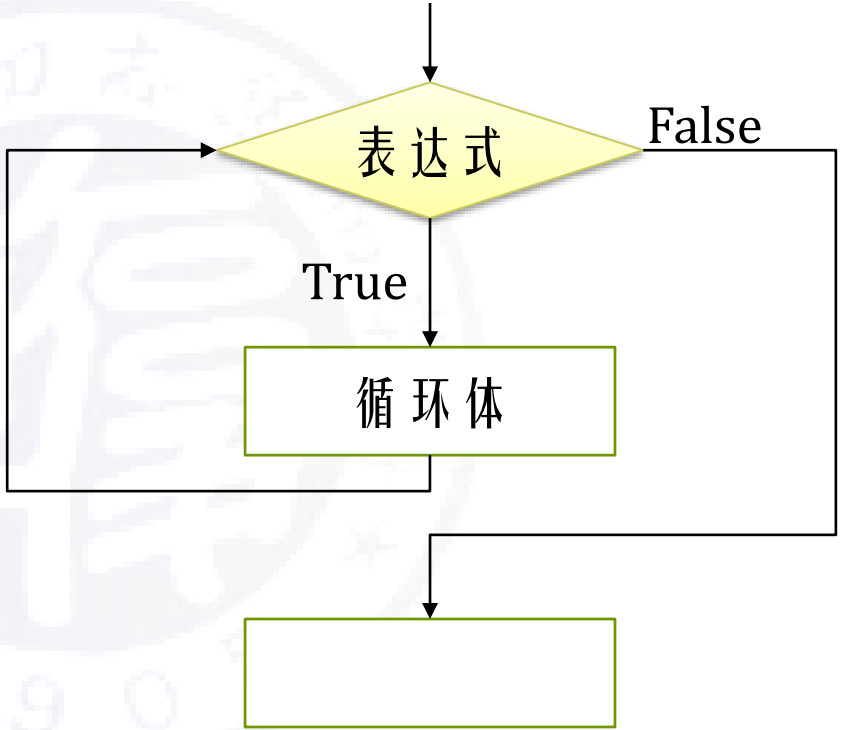
Copy&Run

2.3 循环结构

- 2.3.1 while循环
- 2.3.2 循环结构的优化

while循环语法

```
while 表达式:
    循环体
```



示例1

```
>>>
This is Line 1
This is Line 2
This is Line 3
This is Line 4
This is Line 5
This is Line 6
This is Line 7
This is Line 8
This is Line 9
This is Line 10
>>>
```

```
lineNumber = 1

while lineNumber <= 10:
    print('This is line', lineNumber)
    lineNumber += 1
```

示例2

- 输入一个正整数n，求从1~n各数的平方和，如：输入15，求 $1^2+2^2+3^2+\ldots+14^2+15^2=?$

```
Enter an integer: 15
Sum of squares(1~15): 1240
```

直观解法

Step1: 求1^2, 得到1.

Step2: 求2^2得4, 再加上步骤1的结果1, 得到5.

Step3: 求3^2得9, 再加上步骤2的结果5, 得到14.

Step4: ……

……

Step n: 求n^2得…, 再加上步骤n-1的结果…, 得到…

算法

- 设变量s保存平方和, 设变量i从1变到n, 用循环法求结果

  Step1 : 0⇒s

  Step2 : 1⇒i

  Step3 : 将i×i累加到s中, 即i×i +s⇒s

  Step4 : 使i的值加1, 即i+1⇒i

  Step5 : 如果i≤n, 返回步骤3执行; 否则, 算法结束

  循环

  - 最后得到s的值就是1~n的平方和.

```
n = int(input('Enter an integer: '))

s = 0   # 平方和的初值为0
i = 1   # 整数从1开始
while i <= n:
    s += i * i
    i += 1

print('Sum of squares(1~%d): %d' % (n, s))
```

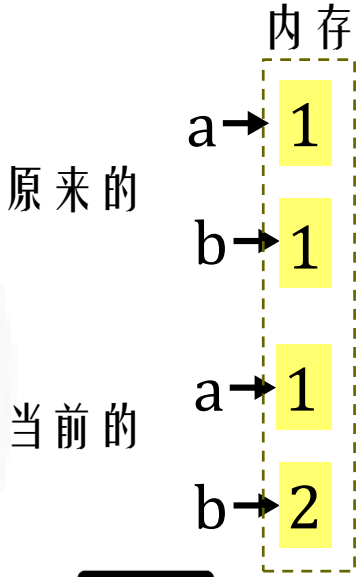
Copy&Run

示例3: 求Fibonacci数列的前n项

```
n = int(input('Enter n: '))

a, b = 0, 1
i = 0
while i < n:
    a, b = b, a+b
    print(a, end=' ')
    i += 1
```

```
a1, b1 = b, a+b
a, b = a1, b1
```




```
Enter n: 14
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Fibonacci.py

Copy&Run

示例4

- 查找一个最小正整数，要求满足条件：被3除余2，被5除余3，被7除余4.
  - 关键：条件表达式的构造

    num % 3 == 2 and num % 5 == 3 and num % 7 == 4

```
num = 1 # 从1开始检查
while not num % 3 == 2 and num % 5 == 3 and num % 7 == 4:
    num += 1
print(num)
```

改用标志来控制循环

```
num = 1
found = False  # 标志变量：是否已经找到
while not found:
    if num % 3 == 2 and num % 5 == 3 and num % 7 == 4:
        found = True
    else:
        num += 1
print(num)
```

示例5

- 查找满足以下条件的前n个正整数: 被3除余2, 被5除余3, 被7除余4. n由用户指定.

```
Enter an integer: 6
53 158 263 368 473 578
```

```
total = int(input('Enter an integer: '))

num = 1
count = 0

while count < total:
    if num % 3 == 2 and num % 5 == 3 and num % 7 == 4:
        print(num, end=' ')
        count += 1
    num += 1
```

示例6

- 用户输入一个数, 程序输出其平方数; 重复上述过程, 当用户输入0时, 程序退出.

```
Enter a number: 2.5
(2.5)^2 = 6.25
Enter a number: -2.4
(-2.4)^2 = 5.76
Enter a number: 12846375137563256 21
(12846375137563256 21)^2 = 165029354175003369913126959098 1035641
Enter a number: 0
Good bye!
```

```
x = eval(input('Enter a number: '))

while x:      # x非零就循环
    print('{}^2 = {}'.format(x, x*x))
    x = eval(input('Enter a number: '))

print('Good bye!')
```

Copy&Run

2.4 break/continue语句

在循环结构中使用

- break语句: 一般放在if选择结构(嵌套于循环)中, 一旦break语句被执行, 循环提前结束.
- continue语句: 终止本轮循环, 并忽略continue之后的语句, 回到循环顶端, 提前进入下一轮循环.
- 除非break/continue语句让代码更简单/清晰, 否则不要轻易使用.

break 和 continue

```
while -------- :
      ----------------------
      ----------------------
      if -------- :
          --------
          break    # if块内最后一条语句
       ----------------------
 ----------------------
```

跳出循环

```
while -------- :
      ----------------------
      ----------------------
      if -------- :
          --------
          continue    # if块内最后一条语句
      ----------------------
      ----------------------
```

开始下一轮循环

while循环语法(含break语句)

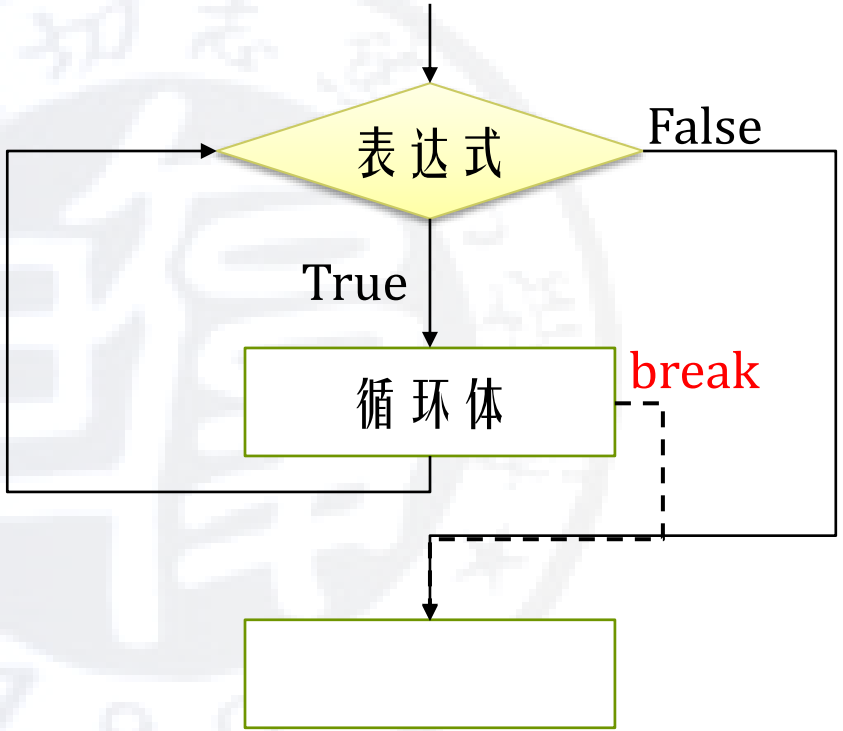
```
while 表达式:
    循环体
```

```
while 表达式:
    循环体
else:
    语句块
```

当循环自然结束(不是因为执行了break而结束)时，执行else结构中的语句

while循环流程(1)

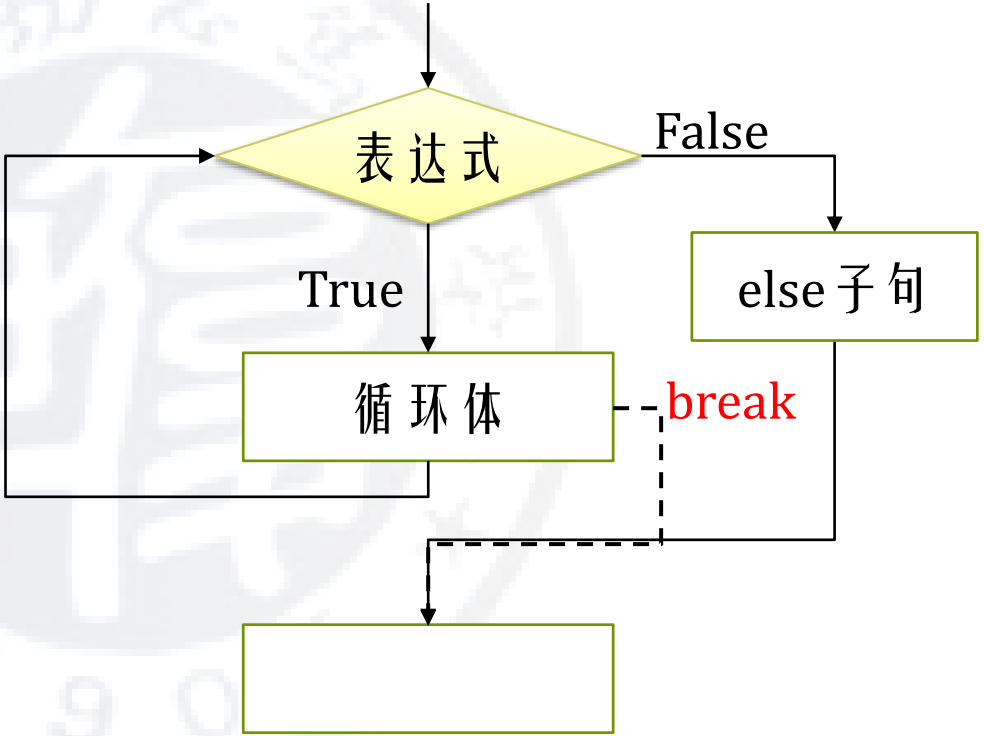
```
while 表达式:
    循环体    #含break
```



while循环流程(2)

```
while 表达式:
        循环体  #含break
else:
        语句块
```

while 可以理解成if的循环版⇨ while-else结构



示例

[例] 查找满足条件的最小正整数：被3除余2，被5除余3，被7除余4．(改用break)

```
num = 1   # 从1开始检查
while not(num % 3 == 2 and num % 5 == 3 and num % 7 == 4):
    num += 1
print(num)
```

```
num = 1
while True:  # 条件永远为真的循环
    if num % 3 == 2 and num % 5 == 3 and num % 7 == 4:
        break  # 找到就终止循环
    num += 1  # 考察下一个整数
print(num)
```

[例] 用户输入一个数，程序输出其平方数；重复上述过程，当用户输入0时，程序退出. (改用break)

```
x = eval(input('Enter a number: '))
while x:      # x非零就循环
    print('({})^2 = {}'.format(x, x*x))
    x = eval(input('Enter a number: '))
print('Good bye!')
```

```
while True:   # 条件永远为真的循环
    x = eval(input('Enter a number: '))
    if not x:
        break   # x为0时终止循环
    print('({})^2 = {}'.format(x, x*x))
print('Good bye!')
```

[例] 求200以内能被17整除的最大正整数.

```
num = 200
while num > 0:
    if num % 17 == 0:
        print(num)
        break    # 找到就终止循环
    num -= 1
```

```
num = 200
while num % 17:
    num -= 1
print(num)
```

[例] 给出一个正整数n(≥3), 判断其是否为素数.

- 所谓素数(prime)，是指除了1和该数本身之外，不能被其它任何整数整除的数.

  例如：13是素数, 因为它不能被2, 3, 4, …, 12整除.

- 判断方法: 将n作为被除数, 将2~n-1 先后作为除数, 如果都不能整除, 则n为素数.

[算法]

S1: 输入n的值

S2: 2⇒i (i作为除数)

循环

S3: n被i除, 得余数r

S4: 如果r为0, 表示n能被i整除, 则输出n不是素数, 算法结束; 否则执行S5

选择

S5: i+1⇒i

S6: 如果i≤n/2, 返回S3; 否则输出n是素数, 结束.

```
n = int(input('Enter an integer(>1): '))
i = 2   # i: 2~n-1,列举所有可能约数
while i < n:
    if n % i == 0:
        print('%d is not a prime.' % n)
        break   # n能被i整除,提前终止循环
    i += 1
else:   # 循环正常结束,依然没有找到约数,说明是素数
    print('%d is a prime.' % n)
```

方法二: 不用break

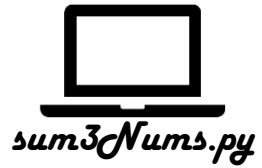
```
n = int(input('Enter an integer(>1): '))
i = 2   # i: 2~n-1,列举所有可能约数
flag = True      # flag为True-没有约数;为False-有约数
while flag and i < n:
    if n % i == 0:
        print('%d is not a prime.' % n)
        flag = False   # n能被i整除,改变flag状态
    i += 1
if flag:    # 直至循环结束,flag状态依然为True,说明n是素数
    print('%d is a prime.' % n)
```

continue示例: 总共循环几次?

```
i = s = 0
while i < 3:
    x = int(input('Enter an integer: '))
    if not x:
        continue  # x为0则回到循环头部
    s += x
    i += 1
print('The sum is:', s)
```

```
7
0
0
2
0
6
The sum is: 15
```

输入0(即x为0)时,while循环并没有增加i的值;输入非零整数时,i的值才增加⇨输入3个非零整数,循环结束.



警惕continue可能带来的问题

```
# 输出10以内奇数
i = 0
while i < 10:
    if i % 2 == 0:
        continue
    print(i, end=' ')
    i += 1
```


```
# 方法1
i = 0
while i < 10:
    i += 1
    if i % 2 == 0:
        continue
    print(i, end=' ')
```

```
# 方法2
i = 0
while i < 10:
    if i % 2 != 0:
        print(i, end=' ')
    i += 1
```

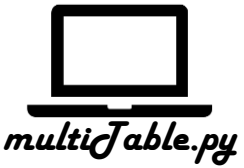
Copy&Run

循环嵌套

```
while -------- :
      ------------------------
      ------------------------
      while -------- :
          --------
          --------
       ------------------------
 ------------------------
```

乘法表(1)

```
i = 1
while i < 10:
    j = 1
    while j < 10:
        print('%d*%d=%d' % (j, i, i*j), end='\t')
        j += 1
    print()
    i += 1
```




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


Copy&Run