11 if 语句

11.1 前言

if 语句是一种控制语句，它允许程序在特定条件为真时执行一些代码。当条件为真时，程序执行 if 语句中的代码块，否则程序跳过它。在日常编写程序种十分常见。本章将详细地进行学习。

11.2 什么是 if 语句?

我们来看下面这段代码：

```
age = 90
if age >= 90:
print("you are old man!")
else:
print("you are not old enough. ")
```

输出结果：

```
you are old man!
```

这里定义了一个 age 变量，当 age 大于或等于 90，输出 you are old man!，否则输出 you are not old enough. 。

if 语句的核心是判断条件是满足(True)或不满足(False)，如果条件值为 True，Python 就会执行 if 语句下面的代码块，否则不执行。

如果你对 bool 值的用法不熟悉的话，接下来我们学习一下 bool 值。

11.3 bool 值学习

11.3.1 判断是否相等

示例代码：

```
people = "a good man"
print(people == "a good man")
```

输出结果：

```
True
```

判断是否等于用==来表示。如果 people 的值不是 a good man，将会返回 False。

示例代码：

```
people = "a good man"
print(people == "a bad man")
```

输出结果：

```
False
```

11.3.2 判断是否不相等

示例代码：

```
people = "a good man"
print(people!= "a bad man")
```

输出结果：

```
True
```

判断是否不等于用！=来表示。

11.3.3 多条件判断

```
age = 15
print(age >= 15 and age < 30)
```

输出结果：

```
True
```

有时你需要判断变量是否满足多个条件，然后再给出相应的对策。这是就需要上述代码了。

这里的判断条件就是 age 同时满足大于 15 小于 30 两个条件。同时满足才能输出 True。

示例代码：

```
age = 15
print(age >= 15 or age < 30)
```

输出结果：

```
True
```

这里的判断条件就是 age 只需满足大于 15 或小于 30 两个条件中的一个条件，就可以输出 true。

11.3.4 判断节点是否在列表里

示例代码：

```
ls = [i for i in range(0,5)]
print(15 in ls)
```

输出结果：

```
False
```

这里代码判断 15 是否在 0 到 5 的范围内，返回结果为 False。

11.3.5 判断节点是否不在列表里

示例代码：

```
ls = [i for i in range(0,5)]
print(15 not in ls)
```

输出结果：

```
True
```

11.4 简单 if 语句

最简单的 if 语句是一个条件判断加上一个操作。

if 代码格式：

```
if condition:
do something
```

如果满足 if 语句条件，则会执行操作，反之则不会执行。图 11-1 为 if 语句流程图。

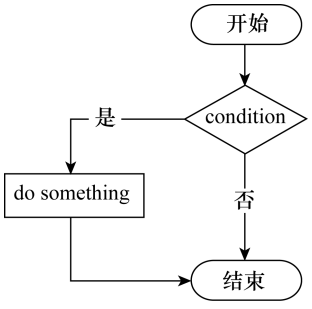

图 11-1 if 语句流程图

Python 是靠代码缩进区分代码块的，所以你可以在操作部分写很多操作。

示例代码：

```
age = 5
if age < 10:
print('you are so young. ')
print('you are so good!')
```

输出结果：

```
you are so young.
you are so good!
```

11.5 多重判断

使用 if-else 语句检查多个条件。

if-else 代码格式：

```
if condition:
do something
else:
do something
```

使用 if-else 语句，如果满足条件，则执行 if 下面代码块中的代码，否则执行 else 下面的代码块中的代码。图 11-2 为 if-else 流程图。

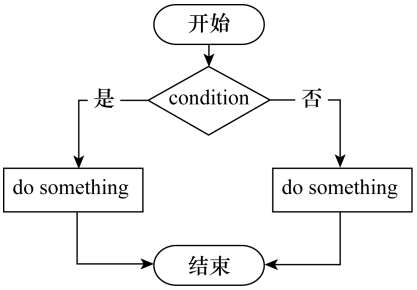

图 11-2 if-else 流程图

示例代码：

```
x = 5
if x > 10:
    print("x 比 10 大")
else:
    print("x 比 10 小")
```

输出结果：

```
x 比 10 小
```

if-else 是一对组合，一个 else 语句对应一个 if 语句，如果出现多余的 else 语句，则会报错。

```
if condition:
    do something
else:
    do something
else:
    do something
```

上述代码 Python 会将 if 和第一个 else 配对，第二个 else 由于没有对应的 if 语句，将会报错。

示例代码：

```
if x > 0:
    print("x is positive")
else:
    print("x is negative")
else:
    print("x is zero")
```

输出结果：

```
else:
^^^^
SyntaxError: invalid syntax
```

如果有三个判断条件，对应三个不同的执行代码，该如何操作呢？可以使用 if-elif-else 语句。

代码格式：

```
if condition_1:
    #如果条件 1 为真,执行此代码块
elif condition_2:
    #如果条件 1 为假且条件 2 为真,执行此代码块
else:
    #如果上述所有条件都为假,执行此代码块
```

程序将检查 condition_1 是否为真。如果是，则执行与之对应的代码块。如果不是，程序将检查 condition_2 是否为真。如果是，则执行与之对应的代码块。如果两个条件都不满足，程序将执行 else 代码块中的代码。图 11-3 为具体流程图。

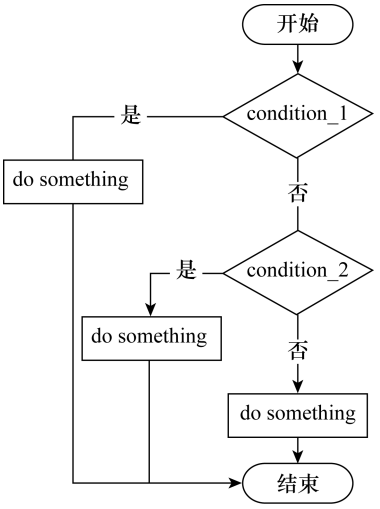

图 11-3　if-elif-else 流程图

代码示例：

```
age = 15
if age < 15:
    print('you are so young. ')
elif age >= 15 and age < 30:
    print('you are in middle age. ')
else:
    print('you are old man. ')
```

输出结果：

```
you are in middle age.
```

这样，if 语句、elif 语句、else 语句分别对应三个判断分支，执行三个不同的操作。

我们经常需要在判断条件是否满足某几项中的一项，满足不同的选项对应着不同的操作。如果你想写更多的条件判断语句，elif 代码块可以写多个。

```
age = 15
if age < 15:
    print('you are so young. ')
elif age >= 15 and age < 30:
    print('you are in middle age. ')
elif age>=30 and age <50:
    print('you are more older. ')
else:
    print('you are old man. ')
```

输出结果：

```
you are in middle age.
```

Python 并没有对 else 代码块有强制要求。也就是说，上述代码如果去掉 else 的代码块程序也是能运行的。

如果你想针对一项任务满足多个不同的条件分别执行不同的任务，同时允许它们都执行的话，可以这么写：

```
ls = [i for i in range(0,5)]
if 1 in ls:
    print("yes!")
if 2 in ls:
    print("yes!")
if 3 in ls:
    print("yes!")
if 4 in ls:
    print("yes!")
```

输出结果：

```
yes!
yes!
yes!
yes!
```

11.6 小结

通过本章，我们学习了条件语句。学会使用条件语句进行不同条件下的不同操作。if 语句是一种判断语句，可以根据某个条件来决定程序执行什么操作。在学习 if 语句之前，我们需要了解 bool 值，它是一种特殊的数据类型，只有 True 和 False 两种值。我们可以使用简单 if 语句来进行单一判断，也可以使用多重判断来进行多种情况的判断。

在下一章节中，我们将学习用户输入和循环。

12 用户输入和循环

12.1 前言

在本章，我们将学习用户输入和循环，学习循环中的中断和跳出操作。通过用户输入和循环的结合，你将会写出交互式程序。

12.2 input 输入

示例代码：

```
message = input('tell me what do you want to say:')
print(message)
```

输出结果：

```
tell me what do you want to say:hello world
hello world
```

input( ) 函数会使程序等待你输入数据，当你在控制台输入 hello world 时，input( ) 函数会将结果返回给变量 message。

input( ) 函数接收一个参数，它会将参数作为提示语在控制台输出。在本例中，提示语就是 tell me what do you want to say。

示例代码：

```
message = input('tell me how old are you:')
print(message)
print(message == 18)
print(message == '18')
```

输出结果：

```
tell me how old are you:18
18
```

```
False
True
```

input( )接收数据后以字符串的形式输出，如果你要判断输入数值是否满足某些条件，就需要对数值进行强制类型转换。

示例代码：

```
message = input('tell me how old are you:')
print(int(message)==18)
```

输出结果：

```
tell me how old are you:18
True
```

12.3 while 循环

示例代码：

```
num = 0
while num < 5:
    print(num)
    num += 1
```

输出结果：

```
0
1
2
3
4
```

while 语句后面的条件和 if 语句一样，当条件满足时，执行代码块中的代码。当代码块中的代码执行完后，程序会跳回到 while 语句进行下一次判断，如果条件仍满足，就会继续执行代码块中的代码，循环往复，直至条件不满足，跳出循环。图 12-1 为 while 循环流程图。

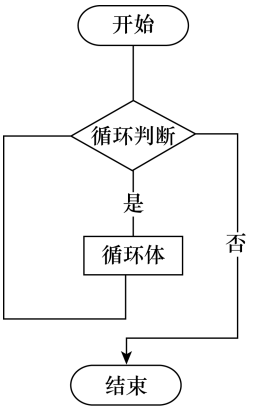

图 12-1 while 循环流程图

如果代码块中没有能够改变 while 语句中判断条件的代码，循环将会永久持续下去。这样循环就成了死循环。例如，上述代码之所以能够结束执行，是因为代码块中的 num 一直在改变，每次循环，num 的值都会加 1，当 num 的值不小于 5 时，程序跳出循环。如果代码块中没有”num+=1”这条语句，循环就会一直进行下去。

死循环在编写程序中是一个致命的错误，它会把电脑内存逐渐消耗殆尽。我们写程序时一定要避免写出死循环代码。

12.3.1 结合 break

当程序满足某些条件时，你想跳出循环，就需要用到 break：

```
while True:
    if input('请输入数据:') == 'quit':
        break
```

输出结果：

```
请输入数据:1
请输入数据:2
请输入数据:quit
Process finished with exit code 0
```

上述代码中，while 语句的判断条件为 True，那么循环将会一直执行下去。可是我们在代码块中加入了判断语句，如果输入为’quit’，执行 break 语句中断循环。这样我们写的循环就不是死循环了。

12.3.2 结合 continue

有时，我们需要在程序满足某个条件时跳过当前循环，后面的循环继续执行。那么使用 continue 语句可以实现这样的操作：

```
num = 0
while num < 5:
    num += 1
    if num == 3:
        continue
    print(num)
```

输出结果：

```
1
2
4
5
```

可以看到，使用 continue 语句，是跳过当前循环，但不中断循环。

12.3.3 结合 pass 语句

写一个程序，当循环体内满足某一条件，执行代码，但不中断整个循环，也不跳过当

前循环该怎么做呢？这时就用上 pass 语句了。

```
num = 0
while num < 5:
    num += 1
    if num==3:
        print(num)
        pass
else:
    print(num)
```

输出结果：

```
3
5
```

if 语句在发现满足条件时执行 print( ) 函数，再通过 pass 语句给程序放行。

12.3.4 while-else 语句

示例代码：

```
num = 0
while num < 5:
    num += 1
else:
    print(num)
```

输出结果：

```
5
```

else 语句与 while 语句相对应，当 while 循环正常结束时，就会执行 else 语句的内容。如果 while 循环中出现的 break 语句时，循环中断，这时 else 语句就不会执行。

示例代码：

```
num = 0
while num < 5:
    num += 1
    if num==3:
        print(num)
        break
else:
    print(num)
```

输出结果：

```
3
```

可以看到，程序只执行了 if 语句内的 print( ) 函数。else 语句内的代码块没有执行。

12.4 for 循环

下面给出循环的结构：

```
for 变量 in 可迭代对象：
语句块
```

示例代码：

```
for i in range(5):
    print(i)
```

for 循环能够实现的所有功能，while 循环也能够实现。但是，在某些循环操作中，for 循环的代码结构更加简洁。例如，如果要输出从 1 到 5 的数字，使用 for 循环只需要两行代码就可以完成，而使用 while 循环则会更加复杂。

12.4.1 for 循环遍历

示例代码：

```
ls=['a','b','c','d']
for i in ls:
    print(i)
```

输出结果：

```
a
b
c
d
```

12.4.2 通过索引遍历列表

示例代码：

```
ls=['a','b','c','d']
for i in range(len(ls)):
    print(ls[i])
```

输出结果：

```
a
b
```

```
c
d
```

使用索引遍历列表，可以是实现直接遍历相同的效果。

12.4.3 结合 break 语句

和 while 循环一样，for 循环可以结合 break 语句、continue 语句、else 语句、pass 语句。

示例代码：

```
for i in range(5):
    if i==3:
        break
    print(i)
```

输出结果：

```
0
1
2
```

循环在每次迭代时，检查 i 是否等于 3。如果是，则使用 break 语句终止循环。

12.4.4 结合 continue 语句

示例代码：

```
for i in range(5):
    if i==3:
        continue
    print(i)
```

输出结果：

```
0
1
2
4
```

循环在每次迭代时，检查 i 是否等于 3。如果是，则使用 continue 语句跳过该次迭代，进入下一次循环。

12.4.5 结合 pass 语句

示例代码：

```
for i in range(5):
```

```
        if i==3:
            print(i)
            pass
print(i)
```

输出结果：

```
3
4
```

循环在每次迭代时，检查 i 是否等于 3。如果是，则使用 print 语句将该值打印到屏幕上，并使用 pass 语句指示程序继续执行下一步。

12.4.6 for 循环中的 else

示例代码：

```
for i in range(5):
    if i==3:
        print(i)
        pass
else:
    print('process is over')
```

输出结果：

```
3
process is over
```

循环在每次迭代时，检查 i 是否等于 3。如果是，则使用 print 语句将该值打印到屏幕上，并使用 pass 语句指示程序继续执行下一步。如果 for 循环执行完毕，则使用 else 语句将字符串"process is over"打印到屏幕上。

这里的 else 语句和 whlie 循环中的一样，出现了中断就不会执行。

12.4.7 嵌套循环

嵌套循环在日常使用中非常常见，下面使用嵌套循环打印一个实心的 3 * 3 的正方形：

```
for i in range(0,3):
    for k in range(0,3):
        print (" * ",end = '')
    print ("")
```

输出结果：

```
*   *   *
*   *   *
*   *   *
```

内层 for 循环在每次迭代时，打印一个“ * ”，并使用 end 参数指示不换行。

外层 for 循环在内层 for 循环执行完毕后，使用 print 语句换行。

12.5 小结

在本章，你学会了如何使用循环和输入函数，可以将二者结合起来写出交互式程序。我们学会了如何使用 input( ) 函数获取用户输入，如何使用 while 循环进行重复操作，如何使用 break、continue 和 pass 语句控制循环的流程，以及如何使用 for 循环遍历数据并在嵌套循环中使用多个循环。

在下一章中，我们将学习函数。通过函数，你将学习封装这一重要的基本概念。