目录

07 不只有一条路—分支和循环

更新时间：2019-08-28 10:04:03

> 不经一翻彻骨寒，怎得梅花扑鼻香。
>
> ——宋帆

前面的章节中我们学习了数据类型、变量、赋值、数值运算，并且用这些知识编写程序来做一些简单的运算，比如「计算一年有多少秒」。像这样的程序，执行流程是完全固定的，每个步骤事先确定好，运行时一步一步地线性地向下执行。

但是很多时候程序的功能会比较复杂，单一的执行流程并无法满足要求，程序在运行时可能需要对一些条件作出判断，然后选择执行不同的流程。这时就需要**分支**和**循环**语句了。

input()、print() 和 int() 函数

在开始学习分支和循环前，为了可以让程序与我们交互，先来学习三个函数。至于什么是函数，我们暂且把它看作是程序中具有某种功能的组件，下一小节中将会详细介绍函数的概念。

input() 函数

如果想要通过命令行与程序交互，可以使用 `input()` 函数。

`input()` 函数可以在代码执行到此处时输出显示一段提示文本，然后等待我们的输入。在输入内容并按下回车键后，程序将读取输入内容并继续向下执行。读取到的输入内容可赋值给变量，供后续使用。写法如下：

```
读取到的输入 = input('提示文本')
```

```
>>> age = input(‘请输入你的年龄：’)
请输入你的年龄：30
```

这行代码会在命令行中显示「请输入你的年龄：」，然后等待输入，读取到输入内容后赋值给 `age` 变量。

`input()` 返回的结果是字符串类型，如 `'30'` 。如果我们需要整数型，可以使用 `int()` 函数进行转换。

int() 函数

`int()` 函数可以将字符串、浮点型转换整数型。写法为：

```
int(字符串或浮点数)
```

将字符串类型 ‘1000’ 转换为整数型 1000：

```
>>> int(‘1000’)
1000
```

将浮点数 3.14 转化为整数：

```
>>> int(3.14)
3
```

print() 函数

`print()` 函数可以将指定的内容输出到命令行中。写法如下：

```
print('要输出的内容')
```

```
>>> print(‘Hello World!’)
Hello World!
>>> print(‘你的年龄是’，20)
你的年龄是 20
```

要输出的内容放在括号中，多项内容时用逗号分隔，显示时每项以空格分隔。

input()、print() 示例

我们可以把 `input()` 和 `print()` 结合起来。如下面这两行代码将在命令行中提示「请输入你的年龄：」，然后等待输入，手动输入年龄后按下回车键，将显示「你的年龄是 x」。

我们把代码保存到文件中，文件命名为 age.py ， 然后执行下：

```
➜ ~ python3 age.py
请输入你的年龄：18
你的年龄是 18
```

分支

上一个例子很简单，接收一个输入内容然后把该内容显示出来。现在难度升级。在刚才代码的基础上，如果所输入的年龄小于 18 岁，那么在最后再显示一句勉励语——「好好学习，天天向上」。如何来实现？

if 语句

如果想要表达「如果……」或者「当……」这种情形，需要用到 if 语句。它的写法是：

```
if 条件:
    代码块
```

它的执行规则是，若「条件」满足，则执行 if 下的「代码块」，若「条件」不满足则不执行。

条件满足指的是，条件的结果为布尔值 True ，或非零数字，或非空字符串，或非空列表。

代码块就是一段代码（可以是一行或多行），这段代码作为一个整体以缩进的形式嵌套在 if 下面。按照通常的规范，缩进以 4 个空格表示。

回到我们之前的需求上，「当年龄小于 18 岁」就可以通过 if 语句来实现。完整代码如下：

```
age = int(input('请输入你的年龄：')) # 注意此处用 `int()` 将 `input()` 的结果由字符串转换为整数型
print('你的年龄是', age)

if age < 18:
    print('好好学习，天天向上')
```

保存在文件中，执行一下看看：

```
➜ ~ python3 age.py
请输入你的年龄：17
你的年龄是 17
好好学习，天天向上
```

```
➜ ~ python3 age.py
请输入你的年龄：30
```

可以看到，当所输入的年龄小于 18 时，程序在最后输出了「好好学习，天天向上」，而输入年龄大于 18 时则没有。

else 语句

又在上面的基础上，如果输入的年龄大于等于 18 岁，输出「革命尚未成功，同志任需努力」。该如何实现？

我们可以在 if 语句之后紧接着使用 else 语句，当 if 的条件不满足时，将直接执行 else 的代码块。写法如下：

```
if 条件:
    代码块 1
else:
    代码块 2
```

若条件满足，则执行代码块 1，若不满足则执行代码块 2。所以之前的需求我们可以这样实现：

```
age = int(input('请输入你的年龄：'))
print('你的年龄是', age)

if age < 18:
    print('好好学习，天天向上')
else:
    print('革命尚未成功，同志仍需努力')
```

执行下看看：

```
➜ ~ python3 age.py
请输入你的年龄：18
你的年龄是 18
革命尚未成功，同志任需努力
```

```
➜ ~ python3 age.py
请输入你的年龄：17
你的年龄是 17
好好学习，天天向上
```

elif 语句

我们可以看到， if 和 else 表达的是「如果……否则……」这样的二元对立的条件，非此即彼。但有时我们还需要表达「如果……或者……或者……否则……」这样多个条件间的选择。

举个例子，下表是年龄和其对应的人生阶段。

年龄	人生阶段

7-17 岁	少年
18-40 岁	青年
41-65 岁	中年
65 岁之后	老年

当我们在程序中输入一个年龄时，输出对应的人生阶段。该如何实现？我们先用汉语来描述一下代码逻辑（这种自然语言描述的代码逻辑，也叫作**伪代码**）：

```
如果年龄小于等于 6:
    输出童年
如果年龄介于 7 到 17:
    输出少年
如果年龄介于 18 到 40:
    输出青年
如果年龄介于 41 到 65:
    输出中年
否则:
    输出老年
```

可以看到，我们需要依次进行多个条件判断。要实现它就要用到 `elif` 语句了，字面上它是 else if 的简写。

`elif` 置于 `if` 和 `else` 之间，可以有任意个：

```
if 条件 1:
    代码块 1
elif 条件 2:
    代码块 2
else
    代码块 3
```

之前根据年龄输出人生阶段的需求，可以这样实现：

```
age = int(input('请输入年龄：'))

if age <= 6:
    print('童年')
elif 7 <= age <=17:
    print('少年')
elif 18 <= age <= 40:
    print('青年')
elif 41 <= age <= 65:
    print('中年')
else:
    print('老年')
```

```
➜ ~ python3 age.py
请输入年龄：3
童年
➜ ~ python3 age.py
```

```
➜ ~ python3 age.py
请输入年龄：30
青年
➜ ~ python3 age.py
请输入年龄：65
中年
➜ ~ python3 age.py
请输入年龄：100
老年
```

分支语句小结

如上所述， if 可以配合 elif 和 else 一起使用。代码执行时，将会从第一个条件开始依次验证判断，若其中某个条件满足，则执行对应的代码块，此时后续条件将直接跳过不再验证。

一个 if - elif - else 组合中， elif 可出现任意次数， else 可出现 0 或 1 次。

while 循环

之前介绍的 if 语句，是根据条件来选择执行还是不执行代码块。我们还有一种很重要的场景——根据条件来判断代码块该不该被重复执行，也就是**循环**。

在 Python 中可以使用 while 语句来执行循环操作，写法如下：

```
while 条件:
    代码块
```

它的执行流程是，从 while 条件 这句出发，判断条件是否满足，若满足则执行代码块，然后再次回到 while 条件 ，判断条件是否满足……循环往复，直到条件不满足。

可以看到，如果这里的条件一直满足且固定不变，那么循环将无穷无尽地执行下去，这称之为**死循环**。一般情况下我们很少会刻意使用死循环，更多的是让条件处于变化中，在循环的某一时刻条件不被满足然后退出循环。

循环示例

举个例子，如何输出 100 次「你很棒」？

显然我们可以利用循环来节省代码，对循环条件做一个设计，让它刚好执行 100 次后结束。

```
count = 0

while count < 100:
    print('你很棒')
    count = count + 1
```

利用一个计数器 count 让它保存循环的次数，当 count 小于 100 就执行循环，代码块每执行一次就给 count 加 1 。我们在大脑中试着来模拟这个流程，用大脑来调试（Debug）。

将代码写入文件 loop.py，执行下看看：

> 你很棒
> 你很棒
> 你很棒
> ...

程序将如预期输出 100 行「你很棒」。

> 扩展： count = count + 1 可以简写为 count += 1

条件的与、或、取反

if 语句和 while 语句中的条件可以由多个语句组合表达。

and 关键字

要表达多个条件同时满足的情况，可以使用 and **关键字**。使用 and 关键字时，在所有并列的条件均满足的情况下结果为 True 。至少一个条件不满足时结果为 False 。如：

```
>>> 2 > 1 and ‘abc’ == ‘abc’ and True
True
>>> 1 > 0 and 0 != 0
False
```

在 if 语句中可以这样使用 and 关键字 ：

```
if 条件1 and 条件2 and 条件N:
    代码块
```

上述 if 语句只有在所有的条件均满足的情况下，代码块才会被执行。

例如我们假设把年龄大于 30 并且为男性的人称为大叔，「年龄大于 30 」和「男性」是两个判断条件，并且需要同时满足，这种情况就可以用 and 关键字来表达。如：

```
if age > 30 and sex == 'male':
    print('大叔')
```

or 关键字

要表达多个条件中至少一个满足即可的情况，可以使用 or **关键字**。使用 or 关键字时，并列的条件中至少有一个满足时，结果为 True 。全部不满足时结果为 False 。

在 if 语句中可以这样使用 or 关键字 ：

```
if 条件1 or 条件2 or 条件N:
    代码块
```

not 关键字

not 关键字可以将一个布尔值取反。如：

```
>>> not True
False
>>>
>>> not 1 > 0
False
```

用在 if 语句和 while 语句的条件上时，条件的结果被反转。

在 if 语句中可以这样使用 not 关键字：

```
if not 条件:
    代码块
```

上述 if 语句在条件不满足时执行代码块，条件满足时反而不执行，因为 not 关键字对结果取了反。

for 循环

前面介绍了 while 循环，在 Python 中还有一种循环方式——for 循环。

for 循环更多的是用于从头到尾地去扫描列表、字符串这类数据结构中的每一个项，这种方式叫做**遍历或迭代**。

for 循环写法为：

```
for 项 in 序列:
    代码块
```

其执行过程是，反复执行 for 项 in 序列 语句和其代码块，项 的值依次用序列的各个数据项替换，直到序列的所有项被遍历一遍。

比如，有个列表为 ['apple', 'banana', 'cherry', 'durian']，我们想依次输出它的每个列表项，就可以用 for 循环。

```
fruit_list = ['apple', 'banana', 'cherry', 'durian']

for fruit in fruit_list:
    print(fruit)
```

将代码写入 for.py，执行下：

```
➜ ~ python3 for.py
apple
```

> durian

可以看到，

1. 第一次循环时， fruit 的值为 apple
2. 第二次循环时， fruit 的值为 banana
3. 第三次循环时， fruit 的值为 cherry
4. 第四次循环时， fruit 的值为 durian

每次循环时 fruit 都自动被赋予新的值，直到 fruit_list 的所有列表项遍历完，循环退出。

总结

input() 函数可以在程序运行到此处时输出一段提示文本，然后停留在此等待我们的输入，输入内容后按下回车键，程序将读取输入内容并向下执行。写法为：

```
age = input('请输入你的年龄：')
```

print() 函数可以将内容输出到命令行中，内容放到括号中，多项内容时可用逗号分隔。写法为：

```
print('你的年龄是', 20)
```

int() 函数可以将字符串、浮点型转换整数型。写法为：

```
int(字符串或浮点数)
```

if ， elif ， else 组合使用，根据条件来选择对应的执行路径。写法为：

```
if 条件 1:
    代码块 1
elif 条件 2:
    代码块 2
else
    代码块 3
```

while 语句来用执行循环操作，根据条件来判断代码块该不该被重复执行。写法为：

```
while 条件:
    代码块
```

for 循环通常用于执行遍历操作。写法为：

```
for 项 in 序列:
    代码块
```

> 多语言比较：

```
if (条件1) {
 代码块1
} else if (条件2) {
 代码块2
} else {
 代码块3
}
```

C/C++ 中的分支语句：

```
if (条件1) {
 代码块1
} else if (条件2) {
 代码块2
} else {
 代码块3
}
```

Go 中的分支语句：

```
if 条件1 {
 代码块1
} else if 条件2 {
 代码块2
} else {
 代码块3
}
```

Java 中的循环：

```
for (int i=0; i < 100; i++) {
 代码块
}

// 或 for each 形式
for (int number: numbers) {
 代码块
}
```

C/C++ 中的循环：

```
for (int i=0; i < 100; i++) {
 代码块
}
```

Go 中的循环：

```
for i := 0; a < 100; i++ {
 代码块
}

// 相当于 while
for 条件 {
 代码块
}
```

```
}
```

← 06 一串数据怎么存—列表和字符串

08 将代码放进盒子—函数 →

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论

千学不如一看，千看不如一练