第五章：条件与循环

Table of Contents

[1. 序言 2](#_Toc50472781)

[1.1 章节简介 2](#_Toc50472782)

[2. 条件语句 2](#_Toc50472783)

[3. 循环与迭代 4](#_Toc50472784)

[3.1 while循环 5](#_Toc50472785)

[3.2 for循环 5](#_Toc50472786)

[3.3 break 6](#_Toc50472787)

[3.4 continue 7](#_Toc50472788)

[4. with与上下文管理器 8](#_Toc50472789)

[4.2 装饰器contextmanager 9](#_Toc50472790)

[4 小结 10](#_Toc50472791)

# 序言

## 章节简介

本章节继续学习Python的基础，为大家详细介绍程序结构与控制流。具体内容为：

a. 条件语句

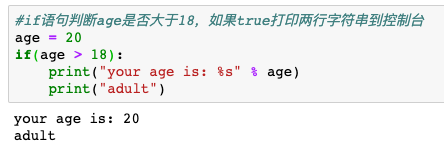
b. 循环与迭代

c. with上下文

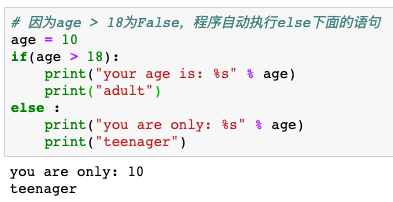
# 条件语句

我们处理现实生活中的问题时会做出决定，就像决定买哪种相机或者今天该穿什么样的衣服。同样我们写计算机程序的时候也要做相同的事情。我们通过 if-elif-else语句来做决定，我们使用它来控制程序运行的流程。

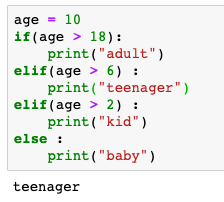
比如，我们根据用户的年龄，在控制台输出不同的内容。和大多数编程语言一样，在Python中我们用if语句实现，根据Python的缩进规则，如果if语句括号中的表达式值为True，那么程序会执行冒号下面，以一个tab空格的所有语句。下面的代码片段里，因为age>18这一判断条件为True，所以程序会自动执行冒号下面，每行用tab缩进的所有语句，程序会执行两行print语句，将字符串输出到控制台。如果if语句括号中的表达式值为False,程序什么也不执行。



也可以给if添加一个else语句，二者组成一个条件语句块，意思是，如果if判断是False，不要执行if的内容，去把else中的代码执行，例如下面的代码，注意冒号(:)是必须的不能省略。



上面的程序仅仅有两种分支(条件True时执行一段代码，False执行另一段)，我们还可以使用if…elif…else来处理多种情况的条件判断，举例代码如下。在这个代码片段里，if、两个elif和else组成了一个完整的条件判断。if语句的特点是从上往下依次判断每个表达式，如果在某个表达式上的值为True，**就执行某个判断里的语句而忽略掉剩下的elif和else**。这里因为满足了age>6为True，只会执行print(“teenager”)，等到print(“teenager”)执行完后忽略后面的条件判断，整个语句执行完毕。



在if…elif…else语句里一般在最后一个分支else后面不添加条件判断 ，如果所有的条件判断都不满足的话，默认就执行最后一个分支即else部分的代码。



# 循环与迭代

我们来假设有这样一种情况，现在让我们在控制台输出100条“hello CGFT”，本质上我们写100次print(“hello CGFT”)就可以了，但是如果一千条一万条该如何处理？这就要用到循环语句。

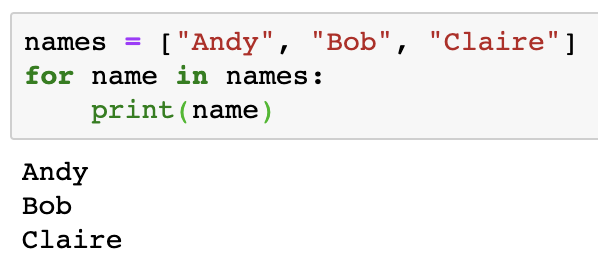
## while循环

while循环的语法是 *while(条件)： 执行代码块*。例如输出10次“hello CGFT”。代码首先判断i<10,如果为True则执行while语句中的代码块(即打印字符串”hello CGFT”并将变量i的值加1)；当代码块执行完一次后会继续进行条件判断i<10,直到条件为False时跳出while循环执行结束。

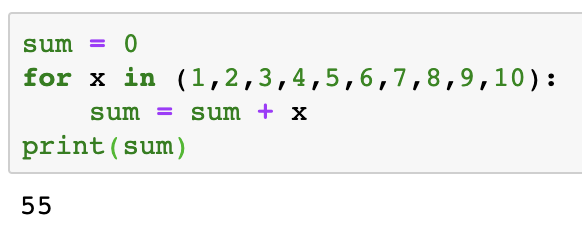


## for循环

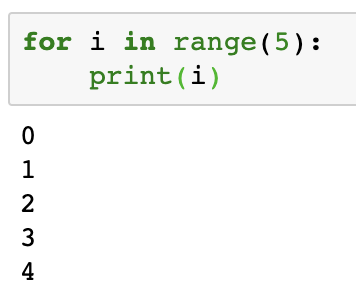
for循环和while循环一样可以进行循环，并且是运用最多的循环方式，而且它有一项非常厉害的功能—遍历。在Python中for循环可以遍历任何序列项目，如字符串列表元组等。遍历列表，就是把列表里的元素全部访问一遍。执行这段代码，会依次打印names的每一个元素。



所以for x in …循环就是把每个元素带入变量x，然后执行缩进块里的语句。再比如我们想计算1-10的整数之和，可以用一个sum变量做累加：

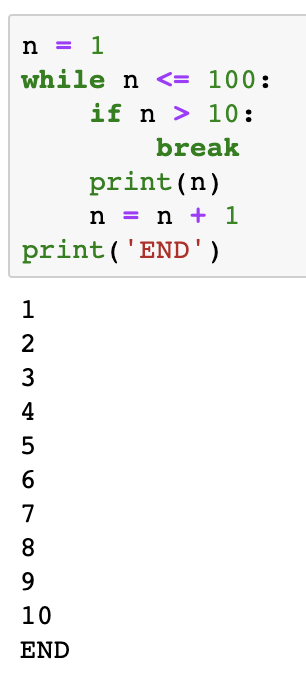


Python还提供了一个range()函数，可以配合我们的for循环使用，例如：



## break

在循环中，break语句可以提前退出循环。比如我们有一个循环打印1~100的数字，如果我们想提前结束循环，可以使用break，实例如下,执行下面的代码可以看到，打印出1-10后，代码就已经退出循环，紧接着打印END，程序结束。



## continue

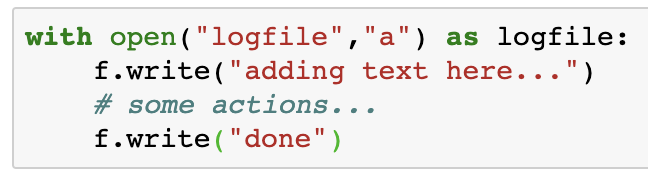
在循环过程中，也可以通过continue语句，跳过当前的这次循环，继续开始下一次循环。比如同样是打印数字，如果我们只想打印奇数，可以用continue跳过某些次的循环，实例如下。执行下面的代码可以看到，打印的不再是1~10，而是1、3、5、7、9。可见continue的作用是提前结束本轮循环，并直接开始下一轮循环。



# with与上下文管理器

正确地管理各种系统资源(如文件、锁定和连接)，在涉及异常时通常是一棘手的问题。例如，引发的一个异常可能导致控制流跳过负责释放关键资源(如锁定)的语句。

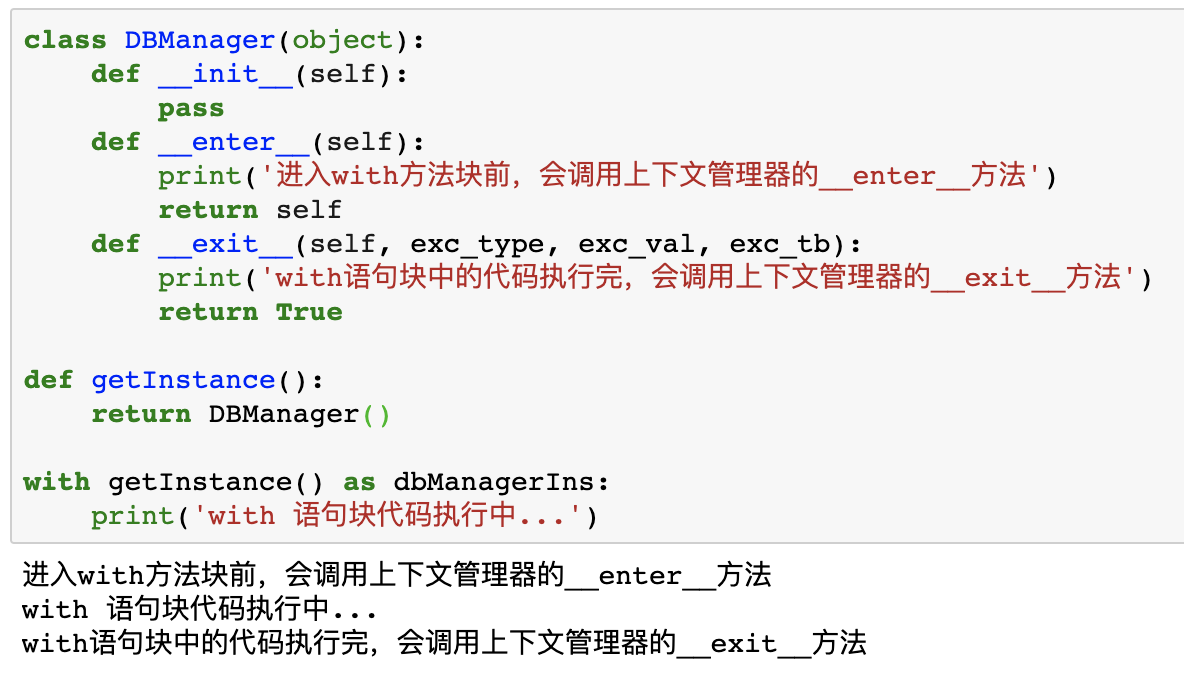
with语句支持在由上下文管理器对象控制的运行时上下文中执行一系列语句，例如下面的代码里，我们用open()方法打开了一个文件并对该文件进行写的操作，当程序执行结束离开with语句块时，with语句将自动关闭已打开的文件释放资源。



with obj语句在程序进入和离开with代码块时，允许对象obj管理所发生的的事情。执行with obj语句时，他执行方法obj.\_\_enter\_\_()方法来指示正在进入一个with语句块(关于对象和方法会在对应章节详细介绍)。当控制流离开with上下文时，则会执行obj.\_\_exit\_\_(type, value, trackback)。如果没有引发异常，exit()方法的三个参数均被设置为None。否则，它们将包含与代码执行期间发生的异常相关的类型、值和跟踪信息。exit()方法返回True或者False, 出现异常时，如果 exit(type, value, traceback) 返回 False，则会重新抛出异常，让with 之外的语句逻辑来处理异常，这也是通用做法；如果返回 True，则忽略异常，不再对异常进行处理。

with obj接受一个可选的as var说明符。如果指定了该说明符，obj.\_\_enter\_\_()方法的返回对象将保存在var中以供with语句块中的代码使用。

with语句只对支持上下文管理协议的对象有效，一个的类只要实现了.\_\_enter\_\_()和.\_\_exit\_\_()方法，我们就说它支持上下文管理协议，可以在with语句里使用。我们定义的类可以实现这两个方法，从而定义它们自己的自定义上下文管理。如下所示：



## 4.2 装饰器contextmanager

contextmanager 用于对生成器函数进行装饰，生成器函数被装饰以后，返回的是一个上下文管理器，其 enter() 和 exit() 方法由 contextmanager 负责提供，而不再是之前的迭代子。被装饰的生成器函数只能产生一个值，否则会导致异常 RuntimeError；产生的值会赋值给 as 子句中的 target，如果使用了 as 子句的话。下面看一个简单的例子。



可以看到，生成器函数中 yield 之前的语句在 enter() 方法中执行，yield 之后的语句在 exit() 中执行，而 yield 产生的值赋给了 as 子句中的 value 变量。需要注意的是，contextmanager 只是省略了 enter() / exit() 的编写，但并不负责实现资源的”获取”和”清理”工作；”获取”操作需要定义在 yield 语句之前，”清理”操作需要定义 yield 语句之后，这样 with 语句在执行 enter() / exit() 方法时会执行这些语句以获取/释放资源，即生成器函数中需要实现必要的逻辑控制，包括资源访问出现错误时抛出适当的异常。

# 小结

本章我们初步介绍了条件和循环的使用。关于循环我们只是初步学习和了解，在下一章节里我们会继续深入学习跟循环相关的一些高级话题，对这一功能会有更深刻的了解。