__Re__

随笔 - 72, 文章 - 37, 评论 - 31, 引用 - 0


**导航**

博客园
新随笔
联系
已订阅
管理



< 2020年7月 >

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 28 | 29 | 30 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |



**公告**

昵称： R_e
园龄： 4年2个月
粉丝： 769
关注： 1
-取消关注



**搜索**

找找看
谷歌搜索

**最新随笔**

1.想要资源的就加群吧，懒得每次发链接了
2.Docker 009 容器的资源限制
3.Docker 008 自建Registry
4.Docker 007 镜像的发布与删除
5.Docker 006 Dockerfile指令
6.docker 004 镜像和仓库
7.Docker 005 构建镜像
8.docker 003 基本操作
9.docker 002 安装docker-ce
10.docker 001 简介

**我的标签**

python(4)
re(1)
编码(1)
翻译(1)
官方文档(1)
简介(1)
迁移(1)
入门(1)
算法(1)
正则表达式(1)
更多


# 开始使用Pyhton

安装

---

第一个Python程序

首先我们打开python 交互式解释器， 执行如下命令：

```
Python 3.5.1+ (default, Mar 30 2016, 22:46:26)
[GCC 5.3.1 20160330] on linux
Type "help", "copyright", "credits" or "license" for more information
>>> print("Hello World") #print ()  用来输出指定的内容
Hello World
```

print之前的三个大于号 叫做提示符。print的是作用是在屏幕显示print之后的内容，这里显示的是"Hello world"

这就是我们的第一个程序。

但是这**和我们平时使用的程序不一样。**

平时都是在文件上双击，然后程序就运行起来了。
交互式解释器退出后程序就没了。交互式解释器是一个临时的程序运行环境，不保存运行程序。
我们写的程序最终是要保存在文件里的。这样我们下次要运行程序的时候，运行我们上次保存的程序文件就可以了。

现在我们写一个最简单的Python程序。

打开记事本 、 将如下内容写入文件：

```
print("Hello World!")
```

另存为 first.py文件，.PY后缀是为了让别人知道这是个python文件。

**变量**

前面我们在使用print()输出内容的时候，如果内容很长，后面要再次输出的时候，就需重新在输入一遍。

如果给输出的内容起个简单的别名。这样我们用简短的别名来代替长内容，下次要输出的时候就直接使用别名来使用原来的长内容。

这个别名就是变量。那如何使用呢?

```
name = "hello world"  name ="alex"
 # 这个别名就是变量。 这样就定义了一个变量。
#  name 是变量名 "hello world"是变量值
```

```
name,age = "alex",12    # 两个变量交换值?
name,age = age,name
```

变量的作用： 用来保存数据，

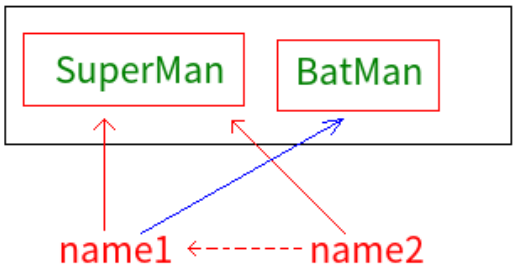
为什么要保存？ 后面要使用。

**Variables** are used to store information to be referenced and manipulated in a computer program. They also provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves. It is helpful to think of variables as containers that hold information. Their sole purpose is to label and store data in memory. This data can then be used throughout your program.

变量： 变量就是用来存储一些信息，供程序以后调用或者操作修改。变量为标记数据提供了一种描述性的名字，以便我们的程序可以被程序的阅读者很清晰的理解。把变量作为一个存储信息的容器会更容易理解变量。它的主要是目的是笔记和存储在内存中的数据，这个数据就可以在你的整个程序中使用。

我们给变量指定值的过程叫做**赋值。**

```
name1 = "SuperMan"
name2 = name1
name1 = "BatMan"
print(name1,name2)   # 这里输出的name1和name2的内容分别是什么?
```




实现过程：

程序是运行在内存中的，我们执行 name1="SuperMan"时，这是在内存中哦嗯开辟了一块空间，并将这块空间的内存地址赋值给name1； 在执行name2=name1的时候，将name1中的内存复制给name2,也就是说，name2中的内存地址也是存储"SuperMan"的内存空间的内存地址； 在执行name1= "BatMan"，此时在内存中另外在开辟一块空间存储"BatMan"，将存储"BatMan"的内存空间的地址赋值给name1,。所以在最后执行 print(name1,name2)的时候，就会出现你看到的结果。

就好像我们中国人有中国人起名字的规则，外国人有外国人起名字的规则，变量也有他的命名规则。

**变量的命名规则：**

变量名只能是 字母、数字或下划线的任意组合
变量名的第一个字符不能是数字
区分大小写
关键字不能声明为变量名

```
from keyword import kwlist
print(kwlist)
```

约定俗成的一些规则： 变量名称应该有意义、不要用中文做变量名、不要使用拼音

---

表达式和运算符

什么是表达式?

1+2*3 就是一个表达式，这里的加号和乘号叫做运算符，1、2、3叫做操作数。1+2*3 经过计算后得到的结果是7，就1+2*3 = 7。我们可以将计算结果保存在一个变量里，ret = 1-2*3 。 所以表达式就是由操作数和运算符组成的一句代码或语句，表达式可以求值，可以放在"="的右边，用来给变量赋值。

算术运算符： + - * / //(取整除) %（取余） **

```
>>> 2+3
5
>>> 3-2
1
>>> 2*3
6
>>> 5/2
2.5
>>> 5//2
2
>>> 5%2
1
>>> 2**3
8
```

赋值运算符： = 、+= -= *= /= %= //= **=

```
>>> num = 2
>>> num += 1    # 等价于 num = num + 1
>>> num -= 1    # 等价于 num = num - 1
>>> num *= 1    # 等价于 num = num * 1
>>> num /= 1    # 等价于 num = num / 1
>>> num //= 1    # 等价于 num = num // 1
>>> num %= 1    # 等价于 num = num % 1
>>> num **= 2    # 等价于 num = num ** 2
```

比较运算符：>、 <、 >=、 <=、 ==、!= True False简单讲一下

8. python3.5 正则表达式(5486)
9. 开始使用Pyhton(5241)
10. openstack cloudinit 遇坑记(4384)

**评论排行榜**

1. 电脑结构和CPU、内存、硬盘三者之间的关系(5)
2. Ubuntu学习——第一篇(5)
3. 开始使用Pyhton(4)
4. 操作系统简史(3)
5. zabbix问题处理(2)
6. 给自己的 MAC 添加一个桌面日历(2)
7. 大独裁者最后演讲台词(2)
8. elasticsearch-dump 迁移es数据 （elasticdump）(1)
9. 北京市图书馆免费入口(1)
10. 面向对象的弊端是什么（转）(1)

**推荐排行榜**

1. Ubuntu学习——第一篇(31)
2. 电脑结构和CPU、内存、硬盘三者之间的关系(7)
3. 电脑简史(6)
4. Openstack入坑指南(4)
5. 开始使用Pyhton(3)

顾名思义，比较运算符是用来做比较的，比较的结果会有两种，分别是成立和不成立，成立的时候，结果是 True，不成立的时候结果是False。 True和False 用来表示比较后的结果。

```
>>> a = 5
>>> b = 3
>>> a > b  # 检查左操作数的值是否大于右操作数的值，如果是，则条件成立。
True
>>> a < b  # 检查左操作数的值是否小于右操作数的值，如果是，则条件成立。
False
>>> a <= b  # 检查左操作数的值是否小于或等于右操作数的值，如果是，则条件成立。
False
>>> a >= b  # 检查左操作数的值是否大于或等于右操作数的值，如果是，则条件成立。
True
>>> a == b  # 检查，两个操作数的值是否相等，如果是则条件变为真。
False
>>> a != b  # 检查两个操作数的值是否相等，如果值不相等，则条件变为真。
True
```

逻辑运算符： not 、and、 or

逻辑运算符是用来做逻辑计算的。像我们上面用到的比较运算符，每一次比较其实就是一次条件判断，都会相应的得到一个为True或False的值。而逻辑运算符的的操作数就是一个用来做条件判断的表达式或者变量。

```
>>> a > b and  a < b  # 如果两个操作数都是True，那么结果为True，否则结果为Fa
False
>>> a > b or  a < b  # 如果有两个操作数至少有一个为True，那么条件变为True，否
True
>>> not a > b  # 反转操作的状态，操作数为True，则结果为False，反之则为True
False
```

结果为True的时候，我们一般称 结果为 真， 逻辑运算符会有一个真值表。

| and | True | False |
|---|---|---|
| True | True | False |
| False | False | False |

| or | True | False |
|---|---|---|
| True | True | True |
| False | True | False |

| or | True | False |
|---|---|---|
| | False | True |

and 真值表

or 真值表

or 真值表

成员运算符： not in 、in （判断某个单词里是不是有某个字母）

成员运算符用来判断一个元素是否是另一个元素的成员。 比如说我们可以判断“hello” 中是否有 “h”， 得到的结果也是True 或者 False。

```
>>> "h" in "hello"  # 这里的意思是 "h" 在"Hello" 中，判断后结果为True
True
>>> "h" not in "hello"  # 这里的意思是 "h" 不在"Hello" 中，判断后结果为Fal
False
```

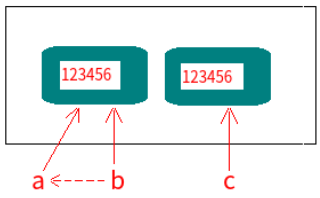
身份运算符： is、is not（讲数据类型时讲解，一般用来判断变量的数据类型）

用来判断身份。

```
>>> a = 123456
>>> b = a
>>> b is a   #判断  a 和 b 是不是同一个 123456
True
>>> c = 123456
>>> c is a  #判断  c 和 a 是不是同一个 123456
False
>>> c is not a   #判断  c 和 a 是不是不是同一个 123456
True
```

这里我们首先将123456赋值给a，后有将a赋值给b， 这样其实是 a和b 的值都是123456, 但是后面c的值也是123456,为什么 第一次a is b 的结果为True ， c和 a 的结果为False 呢?

原因是这样的： 我们知道程序是运行在内存里的，第一次 我们将123456赋值给a的时候，其实是在内存里开辟了一块空间，将123456放在这块空间里，为了找到这里的123456， 会有一个指向这块空间的地址，这个地址叫做内存地址，是123456存储在内存中的地址。a其实指向的就是存储123456的内存空间的地址。执行了 b=a，就是让b指向的地址和a一样。之后我们执行了 c = 123456 ，这里就会再开辟一块内存空间，并将指向该空间的内存地址赋值给c ，这样的话 ， a和b 指向的是同一个123456, c 指向的是另外一个123456 。




位运算符：

先了解一个概念：

我们平时用到的数字在计算机中是以二进制表示的，这个二进制数叫做这个数的机器数。机器数是带符号的，在计算机用一个数的最高位存放符号，正数为0，负数为1.

比如：十进制中的数 +7 ，计算机字长为8位，转换成二进制就是00000111。如果是 -7 ，就是 10000111 。那么，这里的 00000111 和 10000111 就是机器数。

**原码**就是符号位加上真值的绝对值，即用第一位表示符号，其余位表示值. 比如如果是8位二进制:

$[+1]_{原}$ = 0000 0001

$[-1]_{原}$ = 1000 0001

第一位是符号位. 因为第一位是符号位, 所以8位二进制数的取值范围就是:

11111111 到 01111111 即 -127 到 127

**反码**的表示方法是:

正数的反码是其本身

负数的反码是在其原码的基础上, 符号位不变，其余各个位取反.

$[+1] = [00000001]_{原} = [00000001]_{反}$

$[-1] = [10000001]_{原} = [11111110]_{反}$

**补码**的表示方法是:

正数的补码就是其本身

负数的补码是在其原码的基础上, 符号位不变, 其余各位取反, 最后+1. (即在反码的基础上+1)

$[+1] = [00000001]_{原} = [00000001]_{反} = [00000001]_{补}$

$[-1] = [10000001]_{原} = [11111110]_{反} = [11111111]_{补}$

我们设置a=234 (二进制为 11101010)， b=44 （二进制为 101100）

& 按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0

```
    a = 11101010 = 234
    b = 00101100 = 44
result: 00101000 = 40
```

| 按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1。

```
    a = 11101010 = 234
    b = 00101100 = 44
result: 11101110 = 238
```

^ 按位异或运算符：当两对应的二进位相异时，结果为1

```
    a = 11101010 = 234
    b = 00101100 = 44
result: 11000110 = 198
```

~ 按位取反运算符：对数据的每个二进制位取反,即把1变为0,把0变为1

a = 10000000 = 128

~a

*result: 01111111 = 127*

<< 左移动运算符：运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补0。

a = 10110011 = 179

a << 2

result: 1011001100

>> 右移动运算符：把">>"左边的运算数的各二进位全部右移若干位，">>"右边的数指定移动的位数

a = 10110011 = 179

a >> 2

result: 00101100 = 44

位运算符一般用于二进制操作，一般用于底层，我们很少用，知道就可以了。

优先级

| 运算符 | 描述 |
| --- | --- |
| ** | 指数 (最高优先级) |
| ~ + - | 按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@) |
| * / % // | 乘，除，取模和取整除 |
| + - | 加法减法 |
| >> << | 右移，左移运算符 |
| & | 位 'AND' |
| ^ \| | 位运算符 |
| <= < > >= | 比较运算符 |
| <> == != | 等于运算符 |
| = %= /= //= -= += *= **= | 赋值运算符 |
| is is not | 身份运算符 |
| in not in | 成员运算符 |
| not or and | 逻辑运算符 |

运算符那么多，优先级记不住怎么办？ 使用小括号。通过使用小括号，我们就可以很方便的进行优先级的区分。

---

注释

程序很长的时候，不弄明白代码就不知道代码是什么的，这个时候怎么办?

我们看书的时候，有不懂的地方，我们一般都会标注一下。
我们写程序也一样，我在代码旁边标注一下是不是就很方便了。

注释有两种方式：

单行注释 #
多行注释 """ 内容"""

作用：
1. 避免自己忘了写的代码是做什么的——
2. 写给人看的
3. 不要去注释你代码做了什么，而要去 注释 我的代码为什要这么做。

Linux/Unix用户需要注意的内容：

特殊的注释：

在某些Python文件中我们会看到文件的第一行是

```
#!/usr/bin/env python
```

这一行是一个特殊的注释，他有特殊的作用，被称为Shebang，一般在linux/Unix中出现。

Shebang是一个由“#”和“!”构成的字符串行（#!），她出现在文件的第一行。当文件中出现Shebang 时，Linux/Unix 操作系统的的程序载入器会分析Shebang的内容，将之后的内容作为解释器命令，并调用该执行，将载有Shebang的文件路径作为解释器的参数。

在这里 #! 先用于帮助内核找到Python解释器，但是在导入模块的时候将会被忽略，因此，只有在直接执行的文件才有必要加入#! 。

---

如何获取用户的输入

我们前面写的程序，并没有和用户交互，程序运行后，就等待结果的输出。而有些程序是需要用户输入才能继续向下执行。

Python中获取用户输入的语句——input()

```
var = input()
print(var)

var = input("请输入：")
```

---

流程控制 之if语句

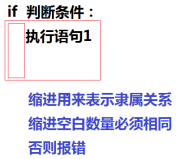
我们知道了如何获取用户输入，现在要写一个猜数字的游戏，我们想一下，首先我们的程序运行起来，然后让用户输入数字，之后程序判断用户输入的数字是否正确，并返回判断结果。

这里就需要用到if语句来进行判断。if语句的结构是：

```
if 判断条件:
    执行语句……

var = input("Enter:")
if var == "A":
    print("True")
```

注意：




缩进——推荐四个空格 （使用2个、3个空格或者tab都是可以得）

不要tab与空格混用

不同软件对空格的显示逻辑总是一样的，但是对于tab却五花八门。

有的软件把Tab展开成空格，有的不会展开。有的Tab宽度是4，有的宽度是8，

这些不一致会使得代码混乱，尤其是靠缩进表示块结构的Python。

我们继续编写我们的猜数字游戏。

前面我们写的猜数字游戏，只有在用户输入正确的时候，才会有提示，那输入错误的时候，是不是也应该提示?

这里就需要用到if-else语句

```
if语句
if 判断条件:
    执行语句……
else:
    执行语句……
```

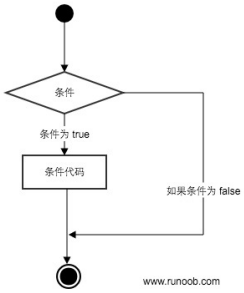
```
num = input("Enter:")
my_num = 12
if num == my_num:
    print("True")
else:
    print("Flase")
```

到现在我们已经完成了一个基本的猜数字游戏，想一下，如果用户在输入数字的时候，如果输入的数字过大，程序就提示数字过大；如果用户输入的数字过小，程序就

提示数字过小。如果这样，咱们的程序就更友好了。

View Code

if 语句的流程图




if语句有三种结构

```
# 第一种
if 条件:
    pass

# 第二种
if 条件:
    pass    # pass语句用来占为，表示什么都不做
else:
    pass

# 第三种
if 条件1:
    pass
elif 条件2:
    pass
elif 条件3:
    pass
else:
    pass
```

---

流程控制——while循环

我们的猜数字游戏，是不是每执行一次，用户就输入一次数字，程序就会判断一个结果，之后程序就结束了。然而这样并不好，程序应该是一直让用户输入数字，一直到用户输入的数字正确。

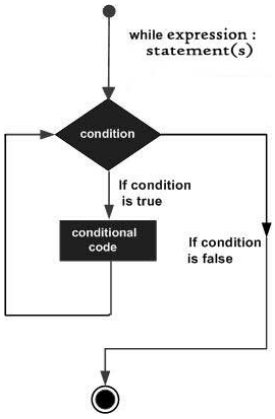
```
# while循环结构
while 判断条件:    # 只有条件不成立时退出循环，如果条件为真，则循环就没有停止的时
    执行语句……
```

```
init_num = 12
num = int(input("Enter:"))
while init_num != num:
    if num >  init_num:
        print("数字过大")
    else:
        print("数字过小")
    num = int(input("Enter:"))

print("猜对了")
```

这样我们就完成了一个简单的小数字游戏。

**while循环流程图：**




现在，我们换一个需求，我们要输出1到10。

```
# 循环输出1-10所有整数
num = 1
while num <11:
    print(num)
    num = num +1
```

如果现在我们要当数字为5的时候结束循环，怎么办?

这里要使用break语句，break语句会终端当前循环。我们看一下效果：

```
# 循环输出1-10所有整数
num = 1
while num <11:
    print("当前数字是",num)
    if num == 5 :
        break
    num = num +1
    print("现在数字变成了：",num)
```

break的作用： 结束循环，在死循环中，也可以通过设置一定的条件来结束循环。

在变一下需求，如果我希望输出1-100之间的所有奇数。

通过continue语句可以是实现。

```
# 输出1-100之间所有奇数
num = 0
while num<100:
    num = num + 1
    if num%2 ==0:
        continue
    print(num)
```

while循环中的else：

```
# while循环结构
while 判断条件:
    执行语句……
else:
    执行语句……
```

while循环中的else语句比较特殊，这里的else语句，只有在循环正常结束的时候才会执行，什么意思呢？意思就是说如果我们的while循环在执行过程中中断了，也就是说执行了break语句，这里的else语句就不会被执行。我们看一下下面的代码：

```
# 循环没有被中断
num = 0
while num<10:
    num = num + 1
    if num%2 ==0:
        continue
    print(num)
else:
    print("else-----")

### 循环被中断
num = 0
while num<10:
    num = num + 1
    if num%2 ==0:
        break
    print(num)
else:
    print("else-----")
```

嵌套循环：循环套循环

```
num1 = 0
while num1 <3:
    print(num1,end="++" )
    num1 += 1
    num2 = 0
    while num2<3:
        print(num2,end="  ")
        num2 +=1
    print()
```

练习题：

1. 输出九九乘法表

2. 使用 # 号 输出一个长方形，用户可以指定宽和高，如果长为3 ，高为4 ，则输出一个 横着有3个#号 竖着有4个#号的长方形。

*3.如何输出一个如下的直接三角形，用户指定输出行数； （如果上下反转，又如何实现？）*

*
**
***
****

---

**序列**

我们去银行办理业务都是需要排队的，排队前都是拿一个号，然后去排序，窗口叫到哪个号，哪个号就去窗口办理业务。

我们把排队的人想象成一个排列好的队伍，队伍按照号码来排序，他们是一个有序的队列。每个人排队的人都有名字，这些名字按照顺序排列起来也叫序列。同样，我们把人名替换成数字或字母，那他们也叫序列。

序列就是按照一定的顺序排列起来的一系列元素，每个元素不是在其他元素之前，就是在其他元素之后。

这里需要大家了解一条语句——range()。

range（）语句用来生成一组数字,在Pyhton2.x里可以很明显的看出来，Python3.x则看不出来（后面会解释原因）

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # 就像队伍一样，还是有按照顺序来排列的，每个

>>> range(1,10)  # Range()生成的数字默认从0开始，也可以是指定起始值。
[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(1,10,2)    # 还可以指定步长
[1, 3, 5, 7, 9]
```

```
# rang()的用法:
range(stop)
    #stop为结束位置，列出从0到stop之前的所有整数

range(start, stop[, step])
    #start表示起始数字,stop表示结束数字，stop表示每两个相邻数字之间的差，也叫步
    #列出从start开始，到stop之前所有的数字
```

流程控制——for循环

```
# for循环结构一
for var in sequence:
        statements(s)

# for 循环示例
for i in range(10):
    print(i)
```

和while循环一样，在for循环中也可以使用break和continue语句，两者效果一样。

1. 使用for循环输出1-100的所有偶数/奇数
2. 使用for输出0-10，如果遇到5，结束循环

练习题:

1. 输出九九乘法表，for实现
2. 使用 # 号 输出一个长方形，用户可以指定宽和高，for实现
3. 输出一个如下的直接三角形，用户指定输出行数，for实现

分类: Python



R_e
关注 - 1
粉丝 - 769
我在关注他 取消关注

3 0

« 上一篇: Pyhton的发展历程
» 下一篇: Ubuntu学习——第一篇

posted on 2016-08-16 14:55 R_e 阅读(5241) 评论(4) 编辑 收藏

## 评论

## #1楼 回复 引用


支持(0) 反对(0)

2016-08-24 22:18 | 龙卷风摧毁停车场

## #2楼 [楼主] 回复 引用

@ Vast_lee
显然你知道我是谁的.

支持(0) 反对(0)

2016-08-27 00:20 | R_e

## #3楼 回复 引用

```
for right in range(1,10):
for left in range(1,right+1):
print(left,'*',right,'=',left*right,end = " ")
if left * right < 10:
print (end = ' ')
print ()
```

支持(0) 反对(0)

2017-12-10 11:08 | mysteri0n

## #4楼 回复 引用

```
for right in range(1,10):
    for left in range(1,right+1):
        print(left,'*',right,'=',left*right,end = "  ")
        if left * right < 10:
            print (end = ' ')
    print ()
```

支持(1) 反对(0)

2017-12-10 11:09 | mysteri0n

刷新评论 刷新页面 返回顶部

发表评论

编辑 预览

支持 Markdown

提交评论 退出 订阅评论

[Ctrl+Enter快捷键提交]