Python快速入门指北

~~友情提示：专供可怜的中国高中生~~

1.1 编者前言

这并不是一篇文章，而是一篇入门文档，重点针对的是高中信息会考，会有许多疏漏之处，请谅解。

Python是一门极其容易的语言，相信在未来的所有专业中必将都有计算机的一席之地，现在学一点基础的内容也不是什么坏事吧。

本篇教程完全基于一个啥也不会的小白，当然由于是笔试而不是机试，所以Python安装等过程就略过了。

右侧存在目录，挑选自己不会的看。如果是啥都不会的，那就按顺序看。

再次强调，本教程由于重点针对考试，对于实际的一些操作还有很多未介绍

章节上有星号的不一定要掌握，但是考试中有的时候还是涉及了这些基本概念和算法

本指南中所有代码均在**Python3.6**版本中测试通过。

参考文档：Learn Python with Penjee

The Python Tutorial — Python 3.6.15 documentation

walter201230/Python: 最良心的 Python 教程(github.com)

1.2 目录

1.3 基础知识

1.3.1 程序段

看这一段程序：

```
a = 1
b = 2
if a>b:
    a,b = b,a
    print(a)
else:
    print(b)
```

Python的代码的易读性非常强，在同一段区块的程序都是一段，比如在这段程序中`a,b=b,a`和`print(a)`就是一段顺序执行的程序。

1.3.2 变量

就如同你做代数运算一样，当你算a+b的数值时就一定要知道a和b的具体数值。计算机在进行运算时也需要“记住”这些数值。

```
a = 1
b = 1
c = a + b
```

这段代码的意思就是告诉计算机"a"里面存放的是数值1，“b”里面存放的是数值1，“c”里面存放的是a + b的数值，那么就是1+1=2。

> TIPS：1.变量名称由**数字、字母(包括大写字母和小写字母)、下划线**组成。
>
> 2.变量名不能以数字开头
>
> 3.变量名不能用python关键字（如print，if，else之类的）
>
> 4.变量命名严格区分大小写

1.3.3 变量类型

- `int` 类型：存放整数（可正可负）
- `float` 类型：存放浮点数（即普通小数，如1.1234）
- `string` 类型：存放字符串

> TIPS：1.int和float类型都可以进行加减乘除。
>
> 2.string操作在下文。

1.3.4 算术运算符

以下假设变量： **a=10，b=20**：

运算符	描述	实例
+	加 - 两个对象相加	a + b 输出结果 30
-	减 - 得到负数或是一个数减去另一个数	a - b 输出结果 -10
*	乘 - 两个数相乘或是返回一个被重复若干次的字符串	a * b 输出结果 200
/	除 - x除以y	b / a 输出结果 2
%	取模 - 返回除法的余数	b % a 输出结果 0
**	幂 - 返回x的y次幂	a**b 为10的20次方， 输出结果 100000000000000000000
//	取整除 - 返回商的整数部分（**向下取整**）	`9//2=4`

1.3.5 运算优先级

即运算的先后顺序

下表从大到小排列：

运算符	描述
()	小括号
**	乘方
*、/、//、%	乘除
+、-	加减

1.3.6 比较运算符

先说明一个概念：`True`或`False`。很好理解，这个判断条件成立那么就是True，如果不成立就是False。

> TIPS:在if语句中，如果判断表达式的值为true，那么就执行条件下的语句，如果判断表达式的值为false，那么就执行else。

以下假设变量a为10，变量b为20：

运算符	描述	实例
==	等于 - 比较对象是否相等	(a == b) 返回 False。
!=	不等于 - 比较两个对象是否不相等	(a != b) 返回 true.
>	大于 - 返回x是否大于y	(a > b) 返回 False。
<	小于 - 返回x是否小于y。	(a < b) 返回 true。
>=	大于等于 - 返回x是否大于等于y。	(a >= b) 返回 False。
<=	小于等于 - 返回x是否小于等于y。	(a <= b) 返回 true。

1.3.7 赋值运算符

运算符	描述	实例
=	赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	`c **= a 等效于 c = c ** a`
//=	取整除赋值运算符	c //= a 等效于 c = c // a

1.4 输入输出

在Python中获取键盘输入输出的函数非常简单，总结起来就是`input`和`print`

1.4.1 input

input是从键盘中获取输入，直到你按下回车键为止，它的返回值是一段字符串，例如：

```
var = input()
```

此时你可以输入Hello World、1等等文本，最后var变量里存放的就是这么一段字符串。

当我们想要得到数字或浮点类型时，只需要在input外面套一层`int()`或`float()`，例如：

```
var1 = int(input())
var2 = float(input())
```

那么变量var1里存放的就是int类型的整数，var2里存放的就是float类型的浮点数。

当然你也可以在input()里面加上一段话，作为输入的提示：

```
var1 = input("Please Enter a word:")
```

> TIPS：有关数据类型请看上一章。

1.4.2 print

print是往屏幕上输出变量，通常情况下以回车结尾，例如：

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

那么输出的就是一个“Hello World!”

如果输出多个变量，那么Python会自动以空格隔开，例如：

```
var1 = "Hello"
var2 = "World"
print(var1 ,var2)
```

那么输出的就是`Hello World`，注意到Hello和World之间存在一个空格。

你可以自定义你结尾的字符串，例如：

```
print("Hello",end="")
print("World!")
```

那么在第一个`print()`操作完成之后，结尾不再是换行。

1.5 字符串操作

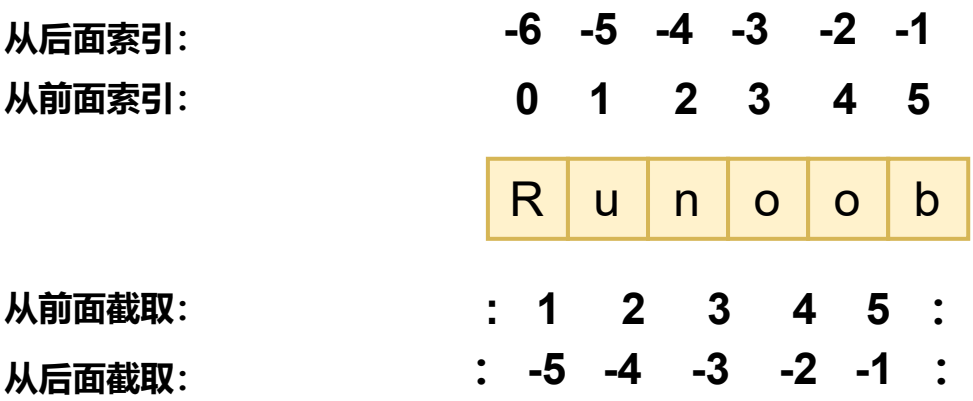
字符串是一种最为基础的数据类型，可以理解为“一段文本”。

1.5.1 创建与访问

创建字符串：`a = "Hello"`，那么变量a就是一个字符串了。

Python 访问子字符串，可以使用方括号 **[]** 来截取字符串，字符串的截取的语法格式如下：

```
变量[头下标:尾下标]
```



需要注意的是第一个字符的位置是0，如果从负数开始那么最后一个字符就是-1开始。

> TIPS：
>
> 在Python中无论是访问数组还是字符串都遵循**左闭右开**的原则。
>
> 例如在上图的字符串中如果标注var[0:3]，那么其实是0、1、2三个位置的字符，并不包含3位置的字符。

例如：

```
var = "Hello World!"
print(var[1]) #输出的是第2个字符
print(var[0:5]) #输出的是从第1个到第5个字符
print(var[-1]) #输出的是最后一个字符
```

输出为：

```
e
Hello
!
```

1.5.2 拼接与复制

在Python中可以使用“+”来连接两个字符串，例如：

```
var1 = "Hello"
var2 = "World"
var3 = var1 + var2
```

那么在上面这段程序中`var3`里面存放的就是“`HelloWorld`”

还可以使用“*”来复制字符串，例如：

```
var1 = "Hello"
var2 = var1 * 2
```

那么变量`var2`中存放的就是“`HelloHello`”

1.5.3 字符串比较

在字符串同样可以进行比较操作，例如：

```
a = "abed"
b = "abce"
if a>b:
    print(a)
else:
    print(b)
```

在字符串比较中，会从0位置开始依次、一一对应地比较。比较方式很简单，按照字母顺序比较（在本质上时ASCII码的比较，有兴趣自己查询），字符串的大小取决于计算机取到的第一个不相同的位置的字符大小。

例如在上面这个例子中，字符串a和字符串b的0、1位置都相同，在第2位置上“e”>"c"，所以字符串a比字符串b大，后续的位置无需比较。

1.5.4 常用函数

- `len()`函数获取字符串长度

  例如：

  ```
  >>>str = "runoob"
  >>> len(str)              # 字符串长度
  6
  >>> l = [1,2,3,4,5]
  >>> len(l)                # 列表元素个数
  5
  ```

`max(str)`	**返回字符串 *str* 中最大的字母。**
`min(str)`	返回字符串 *str* 中最小的字母。

> TIPS：字符串常用函数可看1.10.3章节

1.6 数组(列表)操作

1.6.1 创建与访问

此外，Python已经内置确定序列的长度以及确定最大和最小的元素的方法。

列表是最常用的Python数据类型，它可以作为一个方括号内的逗号分隔值出现。

列表的数据项不需要具有相同的类型

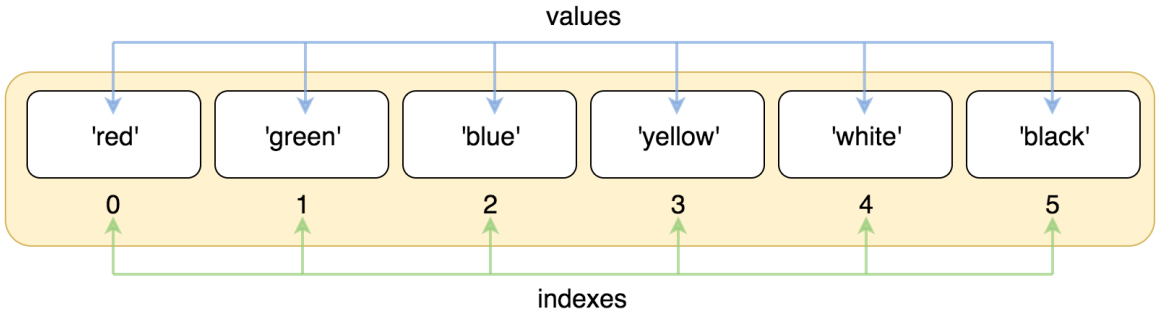
创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。如下所示：

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5 ]
list3 = ["a", "b", "c", "d"]
```

与字符串的索引一样，列表索引从0开始。列表可以进行截取、组合等。

Python 访问列表，可以使用方括号 [] 来选取元素，列表的截取的语法格式如下：

```
变量[头下标:尾下标]
```



需要注意的是第一个字符的位置是0，如果从负数开始那么最后一个字符就是-1开始。

> TIPS：
>
> 在Python中无论是访问数组还是字符串都遵循**左闭右开**的原则。
>
> 例如在上图的数组中如果标注`list2[0:3]`，那么其实是包含第0、1、2三个元素的一个列表，并不包含第3个位置的元素。

例如：

```
var = ["A" ,"B" ,3 ,4]
print(var[1]) #输出的是第2个元素
print(var[0:3]) #输出的是从第1个到第3个元素的列表
print(var[-1]) #输出的是最后一个元素
```

输出为：

```
B
['A', 'B', 3]
4
```

1.6.2 添加修改与删除元素

你可以对列表的数据项进行修改或更新，你也可以使用 `append()` 方法来添加列表项，如下所示：

```
list = ['Google', 'Runoob', 1997, 2000]

print ("第三个元素为 : ", list[2])              #输出列表的第三个元素
list[2] = 2001                                  #修改第三个元素为数据2001
print ("更新后的第三个元素为 : ", list[2])

list1 = ['Google', 'Github', 'Taobao']
list1.append('Baidu')
print ("更新后的列表 : ", list1)
```

> TIPS：`append`可以在列表的末尾添加一个元素，例如上面一段代码

输出结果：

```
第三个元素为 :  1997
更新后的第三个元素为 :  2001
更新后的列表 :  ['Google', 'Github', 'Taobao', 'Baidu']
```

删除元素可以使用del命令：

```
list = ['Google', 'Github', 1997, 2000]
print ("原始列表 : ", list)
del list[2]                          #删除了第三个元素
print ("删除第三个元素 : ", list)
```

以上实例输出结果：

```
原始列表 :  ['Google', 'Github', 1997, 2000]
删除第三个元素 :  ['Google', 'Github', 2000]
```

1.6.3 操作符

列表对 + 和 * 的操作符与字符串相似。+ 号用于组合列表，* 号用于重复列表。

如下所示：

Python 表达式	结果	描述
`len([1, 2, 3])`	`3`	长度
`[1, 2, 3] + [4, 5, 6]`	`[1, 2, 3, 4, 5, 6]`	组合
`['Hi!'] * 4`	`['Hi!', 'Hi!', 'Hi!', 'Hi!']`	重复
`3 in [1, 2, 3]`	`True`	元素是否存在于列表中

1.6.4 常用函数

设`list=["A","B","C","D","E","A"]`

函数	描述	例子
`len()`	列表元素个数	`print(len(list))` 输出5
`max(list)`	返回列表元素最大值	`print(max(list))` 输出E
`min(list)`	返回列表元素最小值	`print(min(list))` 输出A
`list.count()`	统计某个元素在列表中出现的次数	`print(list.count())` 输出2
`list.append()`	在末尾添加元素	见1.6.2

1.7 条件控制

1.7.1 简单条件控制

Python 条件语句是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。

```
if ( condition1 ):
    statement1
else:
    statement2
```

condition即为一个判断表达式（运用比较运算符，详见1.3章节）。

如果最终值为真(True)，那么就执行statement1；如果最终值为假(False)，那么就执行statement2。

以下是一个条件控制的实例：

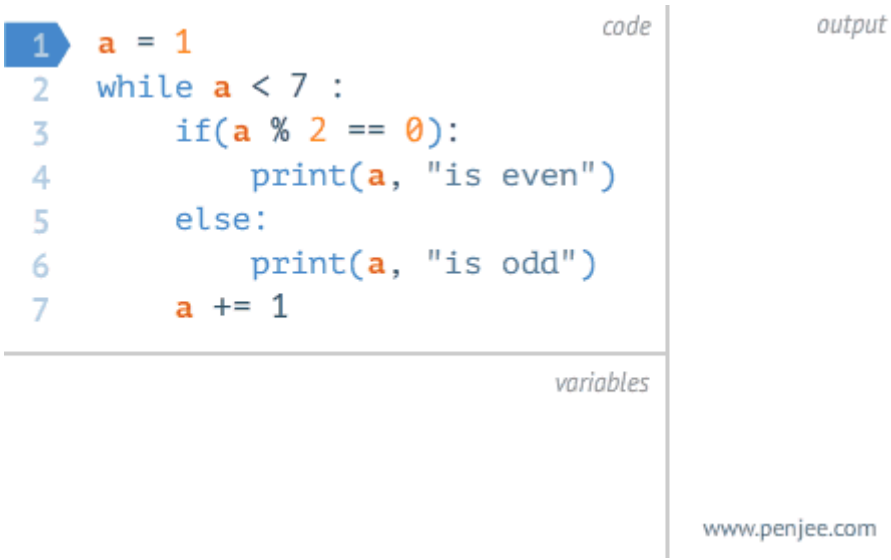
```
var1 = 1
var2 = 2
if (var1 > var2):
    print("var1 is bigger")
else:
    print("var2 is bigger")
```

在以上这个实例中变量`var1`存放数据1，变量`var2`存放数据2。

在第三行上由于`1 > 2`是一个伪命题，所以返回假(False)。

最终执行`else`语句块中的`print`语句。

以下这个GIF图形象地展示了条件语句的执行过程：



1.7.2 条件嵌套

在Python中有一个特殊的关键词`elif`，看字面意思就知道是`if`与`else`的结合体，可以大大简化我们的代码：

```
if ( condition1 ):
    statement1
else:
    if ( condition2 ):
        statement2
    else:
        statement3
```

上面这个是一个最基本的条件嵌套，然而在python中有更方便的写法。

等价于：

```
if ( condition1 ):
    statement1
elif ( condition2 ):
    statement2
else:
    statement3
```

1.8 循环语句

循环是程序设计语言中反复执行某些代码的一种计算机处理过程，常见的有按照次数循环和按照条件循环。

在Python中次数循环即为for循环，条件循环即为while循环。

在学习循环之前，要先了解一个特殊且重要的函数。

1.8.1 Range()函数

这是Python中一个特有的函数。

> `range()` 返回的是一个可迭代对象（类型是对象），而不是列表类型， 所以打印的时候不会打印列表。
>
> 但是在理解中可以认为他是返回了一个列表。

```
range(stop)
range(start, stop, step)
```

参数说明：

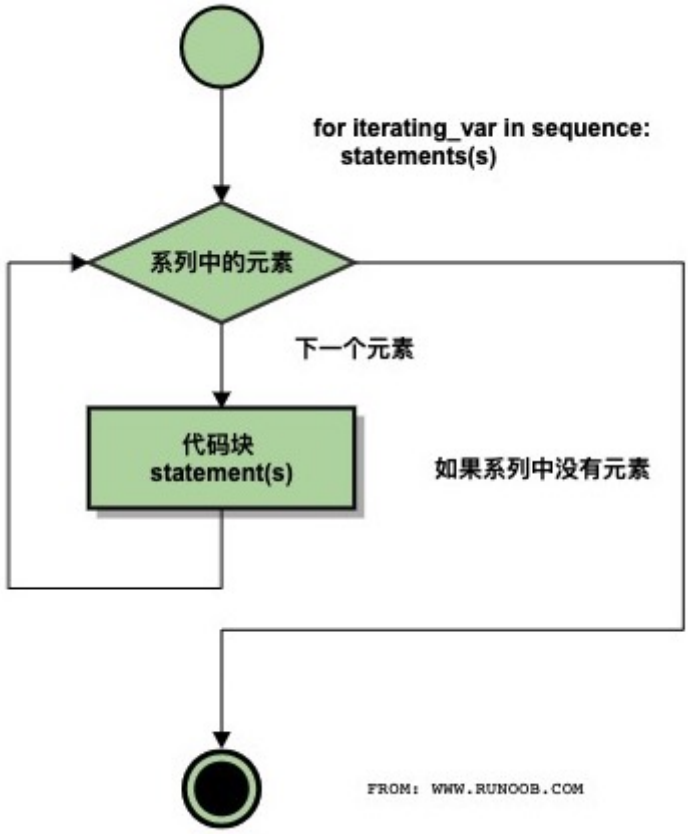
- start:计数从 `start` 开始。默认是从 0 开始。例如 `range（5）` 等价于 `range（0， 5）`
- stop:计数到 `stop` 结束，但**不包括 stop**。例如：`range（0， 5）` 是 `[0, 1, 2, 3, 4]` 没有5
- step:步长，**默认为1**。例如：`range（0， 5）` 等价于 `range(0, 5, 1)`

例如：

```
range(5)        #代表[0,1,2,3,4]
range(1,5)      #代表[1,2,3,4]
range(1,5,2)    #代表[1,3]
```

1.8.2 for循环

Python for 循环可以遍历任何可迭代对象，如一个列表或者一个字符串。



例如：

```
var1 = ["Baidu","Google","Tecent","Taobao"]
for i in var1:
    print(i)
```

在上面这段代码中，变量`i`将按照顺序依次走过列表`var1`中的每个元素，输出如下：

```
Baidu
Google
Tecent
Taobao
```

你还可以利用1.8.1章节中所学的`range()`函数，进行循环次数限定，例如：

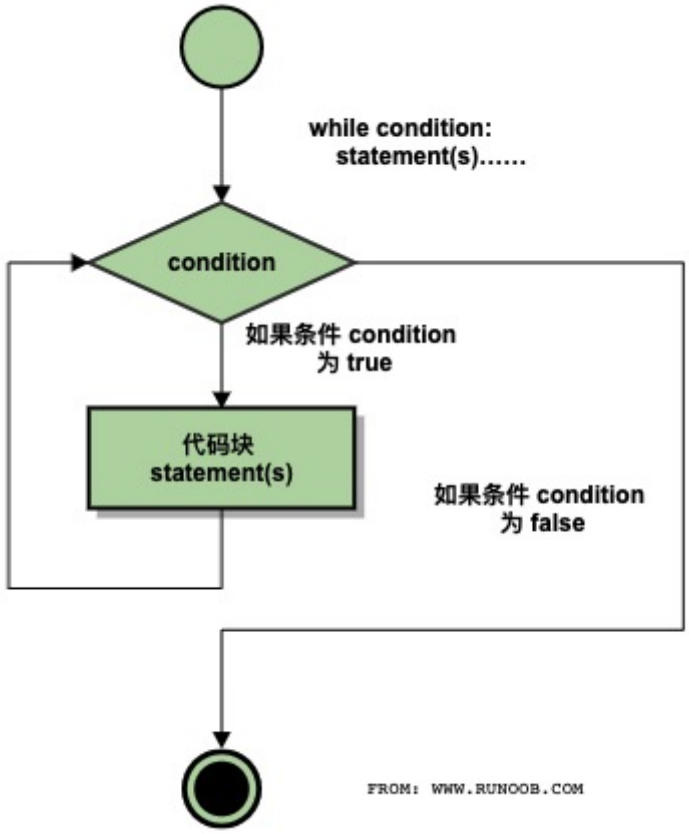
```
for i in range(0,6,2):
    print(i,end=" ")
print()
```

还记得`range()`函数么，这个`range(0,6,2)`等价于`[0,2,4]`,所以最终输出为：

```
0 2 4
```

1.8.3 while循环

while循环是条件循环，在每次进行一次循环时都会判断一次表达式是否成立来决定是否继续执行循环。

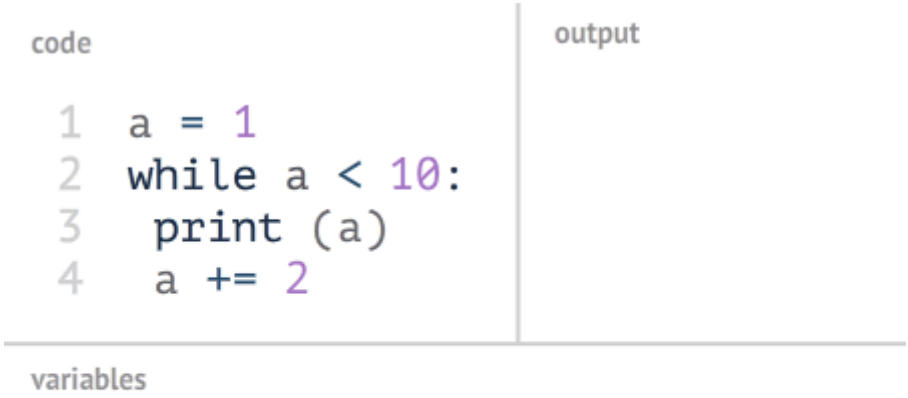


例如：

```
n = 100
sum = 0
counter = 1
while counter <= n:      #每次执行循环时都判断counter<=n
    sum = sum + counter
    counter += 1
print(sum)
```

每次执行循环时都判断counter<=n。如果为真(True)，那么就执行冒号后的代码块，如果为假(False)，那么就直接`print(sum)`。

以下这个GIF动图形象地展示了while循环的执行过程：



> TIPS：聪明的童鞋们可能已经发现了，while循环与for循环本质都是一家，理论上任何while循环和for循环都能互相转化，只不过是于程序员来说的方便程度不同罢了，两者是同等重要的。例如下面这两段就是等价的，童鞋们可以试试。

```
var1 = 0
while (var1 < 6):
 print(var1,end=" ")
 var1 += 2
print()
```

```
for i in range(0,6,2):
 print(i,end=" ")
print()
```

1.8.4 多重循环

字面意思：套娃。一个循环套一个循环，即为多重循环。

本小章并不是一种语法，而是介绍多重循环的用处。

实例如下：

```
for i in range(5):
    for j in range(3):
        print("*",end=" ")
    print()                 #每次打印完一行需要换行
```

可以看出，外层循环5次，每次循环中内层再循环3次，最终打印出一个5行3列的矩阵。

输出如下：

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

1.8.5 break和continue

break 语句可以跳出 for 和 while 的循环体。如果你从 for 或 while 循环中终止，任何对应的循环 else 块将不执行。

continue 语句被用来告诉 Python 跳过当前循环块中的剩余语句，然后继续进行下一轮循环。

while 中使用 break：

```
n = 5
while n > 0:
  n -= 1
  if n == 2:
    break    #当检测到n的值为2时，整个循环一并终止
  print(n)
print('循环结束。')
```

输出结果为：

```
4
3
循环结束。
```

while 中使用 continue：

```
n = 5
while n > 0:
  n -= 1
  if n == 2:
    continue     #当检测到n的值为2时，下面的循环中的语句跳过，直接进入下个循环。
  print(n)
print('循环结束。')
```

输出结果为：

```
4
3
1
0
循环结束。
```

1.9 函数与模块

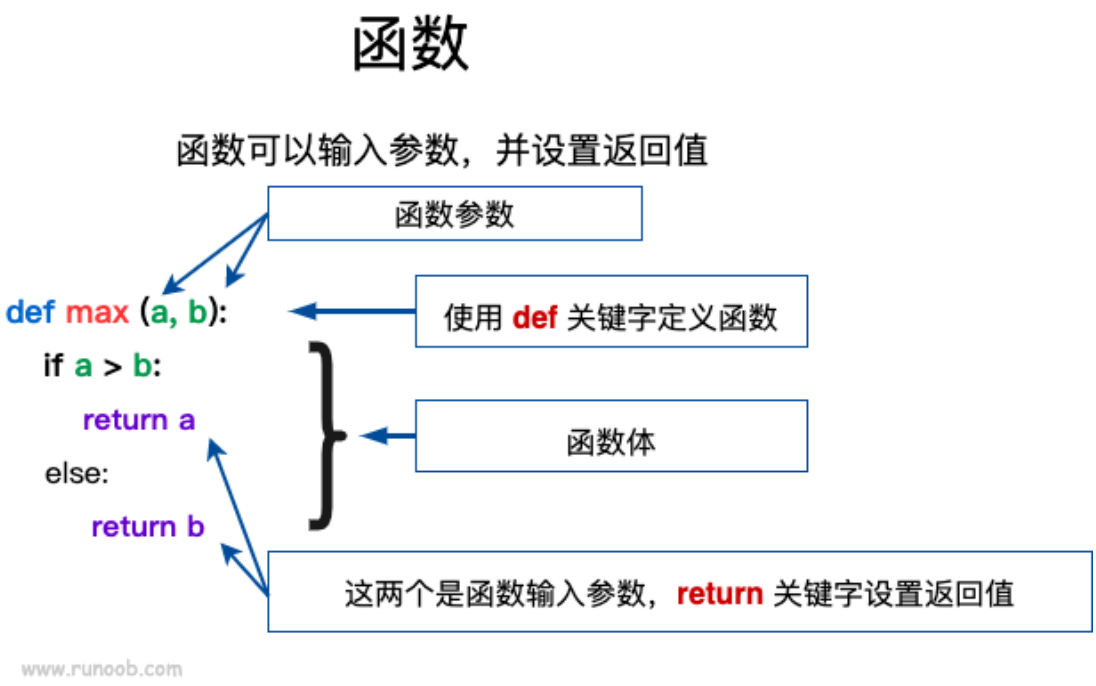
函数和模块本身就是一种可复用的代码形式，函数只能在单个的本地程序中进行复用，模块就是对整个函数进行打包，使得只要安装了这个模块的Python中就可以实现复用代码。

1.9.1 自定义函数

你可以定义一个由自己想要功能的函数，以下是简单的规则：

- 函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号 **()**。
- 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。
- 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
- 函数内容以冒号 **:** 起始，并且缩进。

- **return [表达式]** 结束函数，选择性地返回一个值给调用方，不带表达式的 return 相当于返回 None。



例如：

```
def count(num):                    #定义一个函数名为count的函数，传入参数为num
    sum = 0
    for i in range(1 ,num + 1):
        sum += i
    return sum                     #函数最终返回值时变量sum的值

n = int(input())
print(count(n))                    #调用自定义的count函数计算1+2+...+n的值
```

在上面这个例子中，用户输入一个整数，在`count(n)`中传入这个整数，然后按照定义的count函数，进行计算。

函数的好处显而易见，如果你需要在主程序中反复用同一段代码，就可以把他写成一段函数，大大降低了自己程序的复杂度，提高可读性。

1.9.2 模块

模块是一个包含所有你定义的函数和变量的文件，其后缀名是`.py`。模块可以被别的程序引入，以使用该模块中的函数等功能。这也是使用 python 标准库的方法。

想使用 Python 源文件，只需在另一个源文件里执行 import 语句，例如：

```
import sys      #引入sys模块
import math     #引入math模块

var1 = math.pi

print(math.sqrt(var1))
```

输出：

```
1.7724538509055159
```

在使用引入的某个模块中的函数时，应该使用 `模块名.函数名`，例如上面使用的`math.sqrt()`，`math`就是在第二行中引入的库的名称，`sqrt`就是定义在这个库中的一个函数，用以开方。

sys、math、random等都是python常用的内置模块，无需安装可直接使用。

一些常用的内置模块中用到的函数等将在下一章中总结。

1.10 常用内置函数

本章中介绍的都是常用（考试经常出现的）的Python自带模块或Python内置的函数

Python版本：3.6

1.10.1 数学计算(math库)

`import math`

函数	描述	例子
`math.ceil(x)`	返回 *x* 的上限，即大于或者等于 *x* 的最小整数。	`math.floor(5.19) == 6`
`math.floor(x)`	返回 *x* 的向下取整，小于或等于 *x* 的最大整数。	`math.floor(5.19) == 5`
`math.exp(x)`	返回 *e* 次 *x* 幂，其中 *e* = 2.718281... 是自然对数的基数。	
`math.pow(x, y)`	返回 x 的 y 次幂。	`math.pow(2 ,3) == 8`
`math.log(x, base)`	返回log base底 x	`math.log(8 ,2) == 3`
`math.sqrt(x)`	返回 *x* 的平方根。	`math.sqrt(4) == 2`
`math.pi`	数学常数 *π* = 3.141592...，精确到可用精度。	
三角函数	三角函数(有很多种，不一一列出)	`math.sin( math.pi / 6 ) == 0.5`

1.10.2 随机(random库)

`import random`

函数	描述	例子
`random.random()`	生成一个0到1的随机小数	`n = random.random()`
`random.randint(start, end)`	生成从start到end的随机整数（包含头尾）	`n = random.randint(5,10)`
`random.uniform(start, end)`	生成从start到end的随机小数	`n = random.uniform(5,10)`
`random.choice(list)`	从列表(或其他可迭代对象)中随机选取1个元素	`n = random.choice([1,2,3,4])`
`random.sample(list, n)`	从列表list中随机选取n个元素(返回一个列表)	`n = random.sample([1,2,3,4],2)`

1.10.3 字符串(string库)

`import string`

定义**字符串** `str="abc123ABC*!"`

函数	描述	例子
`str.upper()`	将指定字符串变为大写，可以单独使用	`str.upper("r") == "R"`
`str.lower()`	将指定字符串变为小写，可以单独使用	`str.lower("R") == "r"`
`str.replace(old, new, count)`	将搜索到的字符串改为新字符串，count是可选择输入的参数，代表更改个数。	`str.replace("abc","def",1)`
`str.find(str1, begin, end)`	在字符串str中，从begin开始到end结束查找str1字符串，返回值为第一次出现的头指针位置。	`str.find("23",0,len(str))`
`len(str)`	(这不是模块里的函数)返回字符串长度	`len(str) == 11`

1.10.4 时间(time库)

`import time`

常用的库，但是使用结构体等复杂知识，考试不常考，这里放上官方文档链接，有兴趣自行研究：

[16.3. time — 时间的访问和转换 — Python 3.6.15 文档]

1.10.5 编码函数

这些编码函数不需要引入库，可直接使用

函数	作用
bin(var)	转二进制
oct(var)	转八进制
hex(var)	转十六进制
int(var)	转十进制

1.11 琐碎的细节

输入时需要以空格为分割符输入变量：

```
a,b,c = map(int, input().split())
print(c,b,a)
```

输入：`1 2 3`

输出：`3 2 1`

可以看出在同一行中输入了三个整数，通过这行代码实现了分别存入了a,b,c三个变量。

也可以把`int`改成`float`以实现浮点数的输入。

还有其他的细节性问题可以给我发邮件wzcwzc0@outlook.com，我会加上的。

1.12 *基本数据结构

1.12.1 栈

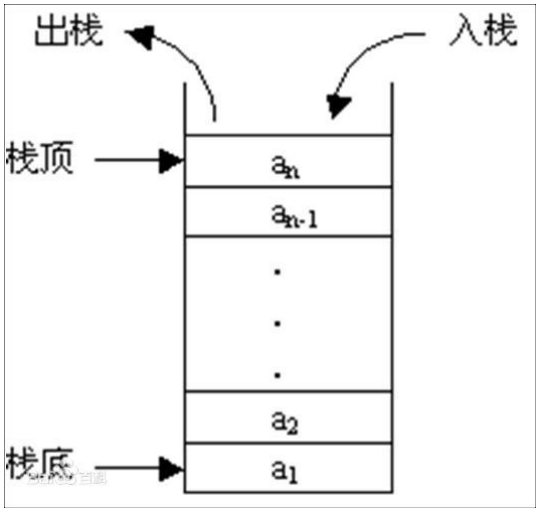
定义：栈是只能在某一端插入和删除的**特殊线性表。栈**也称为**后进先出表**（LIFO表）

表示方法：一个栈可以用定长为n的数组s来表示，用一个指针top指向栈顶。top=0时栈空，top=n时栈满，top<0时为下溢，top>n是为上溢。

进栈时top++，x[top]=n

出栈时top--

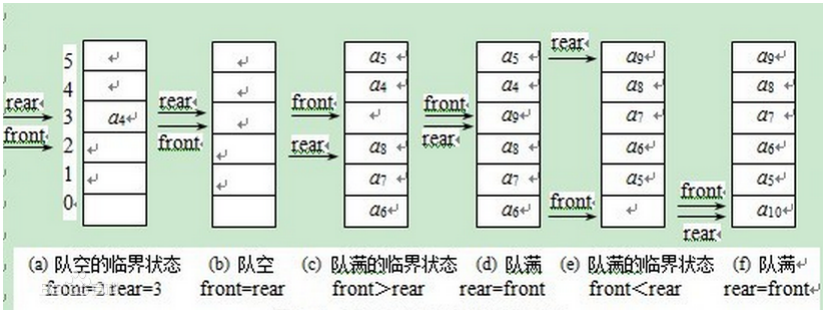
进栈时检查**上溢**，出栈时检查**下溢。**



1.12.2 队列

定义:队列是限定在一端进行插入，另一端进行删除的特殊线性表。**队列**也称为先进先出表（FIFO）

表示方法：一个队列可以用一个数组q来表示，用一个指针front指向**队头**，rear指向**队尾。**



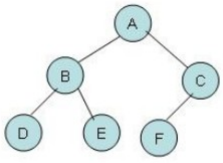
由于入队和出队操作中，头尾指针只增加不减小，致使被删元素的空间永远无法重新利用。当队列中实际的元素个数远远小于向量空间的规模时，也可能由于尾指针已超越向量空间的上界而不能做入队操作。该现象称为"**假上溢**"现象。

对于这种现象可以使用**循环队列。**即当尾指针越过上界时，可以判断底部是否还有剩余空间，从而讲尾指针指向队尾。形象地来看，就像一个循环一样。

PS:对于循环队列中**元素个数**:n=(r-f+n)%n(其中r表示尾指针，f为头指针，n为数组大小)

1.12.3 树

树是一种**非线性的数据结构**



定义：一棵树是由n个元素组成的**有限集合**，其中：

（1）每个元素称为**节点**（node）。

（2）有一个特定的节点，无父节点，称为**根节点**（root）。

（3）除根节点外，其余节点能分成m个互不相交的有限集合，其中每个子集又都是一棵树。

基本概念：

（1）一棵树至少有一个节点，每个节点除根节点外都有一个**前驱结点**（父节点），可以有0个或多个后继节点。

（2）一个节点的子树的个数，称为这个节点的**度。**

（3）定义根节点的**深度**为1，其他节点的深度是其父节点+1。

遍历方式：

（1）先序遍历：先访问根节点，然后从左到右按照先序思想遍历各个子树。如上图：ABDECF（根-左-右）

（2）后序遍历：先从左到右遍历各个子树，然后访问根节点。如上图：DEBFCA（左-右-根）

（3）层次遍历：按深度从小到大逐个访问，同一深度按照从左到右的顺序。如上图：ABCDEF

1.12.4 二叉树

二叉树（binary tree）是一种**特殊的树型结构**，它是度数为2的树。

每个节点的子节点分别称为左孩子、右孩子，两棵子树分别称为左子树、右子树。

性质1：二叉树的第i层上至多有2^{i-1}个节点

性质2：深度为h的二叉树中至多含有2^{h-1}个节点

性质3：若在任意一棵二叉树中，有n0个叶子节点，有n2个度为2的节点，则必有n0=n2+1

性质4：具有n个节点的完全二叉树深为log2x+1（其中x表示不大于n的最大整数）

1.12.5 图

gragh=(V,E)。V是一个非空有限集合，代表**顶点**，E代表**边的集合。**

图的概念：图是描述于一组对象的结构，其中某些对象对在某种意义上是"相关的"。这些对象对应于称为顶点的数学抽象（也称为节点或点），并且每个相关的顶点对都称为边（也称为链接或线）。

图的遍历：深度优先搜索或宽度优先搜索

一笔画问题：如果一个图存在一笔画，则一笔画的路径称为**欧拉路**，如果最后回到起点，那么这个路径称为**欧拉回路。**

（1）存在欧拉路的条件：图是连通的，有且只有2个奇点。（入度+出度为奇数）

（2）存在欧拉回路的条件：图是连通的，有0个奇点。

1.13 *基本算法(方法)

本章本质上并不能称其为算法，涉及一些递归、二分、排序等基本方法。

（未完）

1.14 编者后记

这篇对于我个人来说有点庞大的教程，也终于接近尾声了，我自身在学习Python过程中也存在不少不扎实的地方，有些语法我也是新学的，写这篇教程的同时也帮助我自己再次学习了Python，本篇教程还存在不少不足之处，有错误希望指正改进。