

Python

Python排行榜

Python之禅

第一个Python程序

使用Python Shell 实现

代码解释：

注释

标识符和关键字

1标识符

2关键字

变量：

python保留字

Python命名规范

python3基本数据类型

数字类型

整数类型

浮点数类型

复数类型

布尔类型

Python中的字符串类型Unicode字符

数字类型的转换

隐式类型转换

显式类型转换

字符串类型

python运算符。

关系、逻辑运算符

关系运算符

逻辑运算符

赋值运算符

其他运算符

同一性测试运算符

成员测试运算符

控制语句

分支语句

if结构

if-else结构

elif结构

if-else结构实例

elif结构实例

循环语句

- while语句
- for语句
- while语句实例：
- for语句实例：
- Python跳转语句
 - Break语句
 - Continue 语句
- While和For中的else语句
- 使用范围
- 本章结束
- Python数据结构！ 结构！
 - 元组
 - 序列
 - 创建元组
 - 访问元组
 - 遍历元组
 - 附加程序
 - 列表
 - 列表创建
 - 追加元素
 - 插入元素
 - 替换元素
 - 删除元素
 - 列表的其他常用方法
 - 列表推导式
- 1.集合
 - 1.1创建可变集合
 - 1.2修改可变集合
 - 1.3遍历集合
- 函数式编程
 - 函数
 - 定义函数
 - 自定义函数
- 函数参数
 - 使用关键字参数调用函数
 - 参数默认值
- 函数返回值
 - 无返回值函数
 - 多返回值函数
- 【Python入门自学笔记专辑】——函数嵌套-Lambda表达式
 - 函数嵌套

前言	
正题	
可能出现的错误	
1	
报错信息:	
原因	
解决办法	
2	
报错信息	
原因	
解决办法	
Lambda表达式	
前言	
正题	
The END	
幕后	
THE END	

Python

自从20世纪90年代初Python语言诞生至今，它已被逐渐广泛应用于系统管理任务的处理和Web编程。

Python的创始人荷兰人吉多·范罗苏姆 [3]（Guido van Rossum）。1989年圣诞节期间，在阿姆斯特丹，Guido为了打发圣诞节的无趣，决心开发一个新的脚本解释程序，作为ABC语言的一种继承。之所以选中Python（大蟒蛇的意思）作为该编程语言的名字，是取自英国20世纪70年代首播的电视喜剧《蒙提·派森的飞行马戏团》（Monty Python's Flying Circus）。

Python排行榜

Python诞生已经有20多年了，到现在仍然是一门非常热门的语言

TIOBE社区发布的2017年3月和2018年3月的编程语言热度榜（部分）

变化	编程语言	评级/%
-	Java	14.639
-	C	7.002
-	C++	4.751
↑	Python	3.548
↓↓	C#	3.457
↑↑↑	Visual Basic .Net	3.391
-	JavaScript	3.071

Python之禅

Python语言有的设计理念和哲学，称为“Python之禅”。Python之禅是Python的灵魂，理解Python之禅能帮助开发人员编写出优秀的Python程序。在Python交互式方式运行工具IDLE（也称为Python shell）中输入import this命令，如图所示，显示的内容就是Python之禅。



```

Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informatio
n.
>>>
>>> print("Hello World!")
Hello World!
>>> 1 + 1
2
>>>
>>> str = "Hello World!"
>>> print(str)
Hello World!
>>> |
Ln: 12 Col: 4

```

```

1 Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC
  v.1916 32 bit (Intel)] on win32
2
3 Type "help", "copyright", "credits" or "license()" for more
  information.
4
5 \>>> import this
6
7 The Zen of Python, by Tim Peters
8
9
10

```

```
11 Beautiful is better than ugly.
12
13 Explicit is better than implicit.
14
15 Simple is better than complex.
16
17 Complex is better than complicated.
18
19 Flat is better than nested.
20
21 Sparse is better than dense.
22
23 Readability counts.
24
25 Special cases aren't special enough to break the rules.
26
27 Although practicality beats purity.
28
29 Errors should never pass silently.
30
31 Unless explicitly silenced.
32
33 In the face of ambiguity, refuse the temptation to guess.
34
35 There should be one-- and preferably only one --obvious way to do
   it.
36
37 Although that way may not be obvious at first unless you're
   Dutch.
38
39 Now is better than never.
40
41 Although never is often better than *right* now.
42
43 If the implementation is hard to explain, it's a bad idea.
44
45 If the implementation is easy to explain, it may be a good idea.
46
47 Namespaces are one honking great idea -- let's do more of those!
48
49 \>>>
```

翻译:

《Python之禅》(The Zen of Python), 蒂姆·彼得斯(Tim Peters)著

美丽总比丑陋好。

显式的比隐式的好。

简单总比复杂好。

复杂总比复杂好。

平铺总比嵌套好。

稀疏总比稠密好。

可读性。

特殊情况并不特别到足以违反规则。

尽管实用性胜过纯洁性。

错误不应该悄无声息地过去。

除非显式地沉默。

面对模棱两可的情况, 拒绝猜测的诱惑。

应该有一种——最好是只有一种——显而易见的方法。

尽管这种方式一开始可能并不明显, 除非你是荷兰人。

现在总比没有好。

尽管“从不”常常比“现在”更好。

如果实现很难解释, 那就是个坏主意。

如果实现很容易解释, 这可能是一个好主意。

名称空间是一个伟大的想法——让我们做更多这样的事情!

第一个Python程序

1.程序运行方法:

程序编写结束就可以运行了, 可以使用快捷键Ctrl+F5运行程序。

在visual studio code里面打开终端就可以找到调试控制台查看运行。

2.编写代码:

首先使用visual studio code或者任何编辑器创建一个文件, 然后将文件保存为*.py的格式,

接着在文件中编写代码。

3.第一个python代码——Hello World!——hello.py

输入如下代码:

```
1  """
2
3  Created on 2020/1/9
4  编写者 : Thomas
5
6  """
7  string = "Hello World"
8  print (string)
9
10 fun main(args : Array <String>){
11     println ("Hello World")
12 }
```

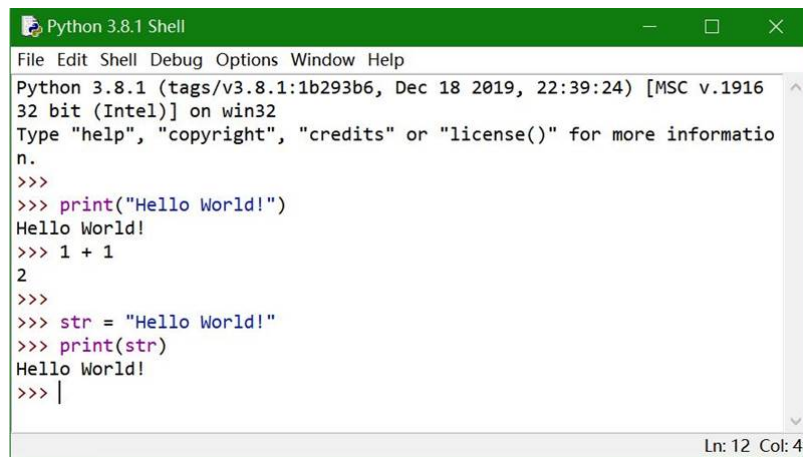
或

```
1  """
2
3  Created on 2020/1/9
4  编写者 : Thomas
5
6  """
7
8  string = "Hello World"
9  print(string)
10 print("Hello World")
```

使用Python Shell 实现

- (1) 点击Python **.Lnk快捷方式启动
- (2) 在windows命令提示符中输入Python（不区分大小写，如果你的电脑里有Python2和Python3，那么启动Python2用“Python”，启动Python 3用“Python3”）
- (3) 通过Python IDLE 启动Python Shell，Python IDLE提供了简单的文本编辑功能，如剪切、复制、粘贴、撤销和重做等，且支持语法高亮显示。

无论采用哪一种方式启动Python Shell，其命令提示符都是“>>>”，在该命令提示符后可以输入Python语句，然后按下Enter键就可以运行Python语句，Python Shell马上输出结果。

A screenshot of a Python 3.8.1 Shell window. The window has a green title bar with the text 'Python 3.8.1 Shell' and standard window controls. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following content:

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informatio
n.
>>>
>>> print("Hello World!")
Hello World!
>>> 1 + 1
2
>>>
>>> str = "Hello World!"
>>> print(str)
Hello World!
>>> |
```

The status bar at the bottom right indicates 'Ln: 12 Col: 4'.

代码解释：

到现在只介绍了如何编写程序和运行Helloworld，并没有进行解释

```
1  """                                ①
2
3  Created on 2020.1.9
4  制作人：Thomas
5
6  """                                ②
7
8  string = "Hello World"             ③
9  print(string)                       ④
```

从代码中可见，Python实现HelloWorld的方式比java、C和C++等语言要简单得多，而且没有main主函数。

代码①到②使用一对三重双引号包裹起来，起到做注释的作用。三重双引号也可以替换成三重单引号。

代码③是声明字符串变量string(定义)，并使用"Hello World"为他赋值。

代码④是通过print函数将字符串输出到控制台，类似于C中的printf函数

print(*objects, sep = ' ', end = '\n', file = sys.stdout, flush = False)

print语句有5个参数：

- 1.*object是可变长度的对象参数
- 2.sep是分隔符参数，默认值是一个空格
- 3.end是输出字符串之后的结束符号，默认值是换号符
- 4.file是输出文件参数，默认值sys.stdout是标准输出，即控制台
- 5.flush为是否刷新文件输出流缓冲区，如果刷新字符串会马上打印输出默认值不刷新

实例详见书P33页

注释

注释:

Python注释使用井号"#", 使用时"#"位于注释行的开头, #后面

有一个空格, 接着写注释内容。

在第三章中介绍过文档字符串, 它也是一种注释, 只是用来注释文档的, 可以多行注释。

```
1 print("hello world")
2 #print("hello world")
3 '''
4 这是Python文档注释
5 '''
```

编译器不处理注释

标识符和关键字

1标识符

标识符就是变量、常量、函数、属性、类、模块、包等由程序员指定的名字。构成标识符的字符均有一定的规范, Python语言中标识符的命名规则如下:

- (1) 区分大小写, Myname和myname是两个不同的标识符
- (2) 首字母可以试试下划线“_”或字母, 但不能是数字
- (3) 除首字符外其他字符, 可以是下划线、字母和数字
- (4) 关键字不能作为标识符
- (5) 不能使用Python内置函数作为自己的标识符

2关键字

关键字是类似于标识符的字符序列, 由语言本身定义好。Python语言中有33个关键字, 只有False、None、True首字母大写, 其他的全部小写。具体内容见表。

Python关键字			
False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

变量：

在Python中声明变量时不需要指定它的类型，只是给一个标识符赋值就声明了变量，实例代码如下：

```
1
2  \# 代码文件：chapter4/src/ch4.2.1.kt
3
4  _hello = "Hello World"
5
6  score_for_student = 0.0
7
8  y = 20
9
10 x = True
11
12 b = False
13
14 b = 20
15
16 print(_hello, score_for_student, y, x, b)
```

变量声明不需要指定数据类型，你赋给它什么数值，它就是该类型的变量了

注意b这个变量，虽然已经赋值False(bool)，但是它也可以接受其他类型

python保留字

保留字即关键字，我们不能把它们用作任何标识符名称。Python 的标准库提供了一个 keyword 模块，可以输出当前版本的所有关键字：

```
1 Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC
  v.1916 32 bit (Intel)] on win32
2 Type "help", "copyright", "credits" or "license()" for more
  information.
3 >>> import keyword
4 >>> keyword.kwlist
5 ['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
  'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
  'except', 'finally', 'for', 'from', 'global', 'if', 'import',
  'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
  'return', 'try', 'while', 'with', 'yield']
6 >>>
```

Python命名规范

程序代码中到处都是标识符，因此取一个一致并且符合规范的名字非常重要。Python中命名规范采用多种不同方式。不同的代码元素命名不同，下面将分类说明。

- 包名：全部小写字母，中间可以由点分隔开，不推荐使用下划线。作为命名空间，包名应该具有唯一性，推荐采用公司或组织域名的倒置，如 com.apple.quicktime.v2
- 模块名：全部小写字母，如果是多个单词构成，可以用下划线隔开，如 dummy_threading。
- 类名：采用大驼峰法命名，如 SplitViewController。
- 异常名：异常属于类，应该使用类名，但是要以 Error 为后缀
- 变量名：全部小写字母，如果由多个单词构成，可以使用下划线隔开。如果变量应用于模块或函数内部，则变量名可以由单下划线开头。
- 函数名和方法名：命名如变量名，如 balance_account、_push_cm_exit。
- 常量名：全部大写字母，其他如同变量名

驼峰命名法：

1.大驼峰，每一个单词的首字母都大写，例如：AnimalZoo，JavaScript中构造函数用的是大驼峰式写法。

2.小驼峰，第一个单词的首字母小写，后面的单词的首字母全部大写，例如：fontSize、backgroundColor。

python3基本数据类型

Python 中的变量不需要声明。每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。

在 Python 中，变量就是变量，它没有类型，我们所说的"类型"是变量所指的内存中对象的类型。

等号 (=) 用来给变量赋值。

等号 (=) 运算符左边是一个变量名,等号 (=) 运算符右边是存储在变量中的值。

```
1  #!/usr/bin/python3
2
3  counter = 100          # 整型变量
4  miles   = 1000.0       # 浮点型变量
5  name    = "nowcoder"   # 字符串
6
7  print (counter)
8  print (miles)
9  print (name)
```

输出

```
1  100
2  1000.0
3  nowcoder
```

数字类型

Python数字类型有4种：整数类型、浮点数类型、复数类型和布尔类型。

整数类型

Python整数类型称为int，整数类型的范围可以很大（Python2里有long类型，Python3中没有long类型），只受所在计算机硬件的限制。

整数类型默认为十进制数，但是也可以表示十六进制、八进制和二进制，表示方法如下。

- 二进制数：以0b或0B为前缀，注意0是阿拉伯数字，不要误以为是字母‘o’
- 八进制数：以0o或0O为前缀，注意是O，注意！
- 十六进制：以0x或0X为前缀

例子（在Python Shell中运行）

```
1  >>>28
2  28
3  >>>0b11100
4  28
5  >>>0o34
6  28
7  >>>0034
8  28
9  >>>0x1c
10 28
```

浮点数类型

Python的浮点类型为float，float类型用于存储小数类型，只支持双精度浮点数。大小写e可以表示10的指数。e2表示 10^2 。

Python Shell 实例：

```
1  >>>1.0
2  1.0
3  >>>0.0
4  0.0
5  >>>3.36e2
6  336.0
7  >>>1.56e-2
8  0.0156
```

复数类型

很多计算机语言都不支持复数类型，但是Python语言支持复数。

Python中复数类型为complex。例如1+2j表示实部为1、虚部为2的复数。

Python Shell中运行如下

```
1 >>>(1+2j)
2 (1+2j)
3 >>>(1+2j) + (1+2j)
4 (2+4j)
```

布尔类型

布尔值为bool，bool是int的子类，它只有两个值：True和False（第一个字母必须大写）

实例：

```
1 >>>bool(0)
2 False
3 >>>bool(2)
4 True
5 >>>bool(1)
6 True
7 >>>bool('')
8 False
9 >>>bool(' ')
10 True
```

Python中的字符串类型Unicode字符

```
1 'Hello World'
2 "Hello World"
3 '\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u0064'
4 "\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u0064"
```

Python中的字符不同于c++字符，Python使用Unicode编码，所以字符串可以包含中文等亚洲字符。

代码第①行和第②行的字符串使用Unicode编码表示的字符串，事实上它表示的也是Hello World字符串，

可通过print（）函数将Unicode编码表示的字符串输出到控制台上，就会看到Hello World字符串。

Python Shell运行实例：

```

1  >>>s = 'Hello World'
2  >>>print(s)
3  Hello World
4  >>>s = "Hello World"
5  >>>print(s)
6  Hello World
7  >>>s =
    '\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u00
    64'
8  >>>print(s)
9  Hello World
10 >>>s =
    "\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u00
    64"
11 >>>print(s)
12 Hello World

```

数字类型的转换

隐式类型转换

多个数字类型之间可以进行数学计算，由于参与运算的数字类型可能不同，此时会发生隐式类型转换，如表

操作数1类型	操作数2类型	转换后的类型
布尔	整数	整数
布尔、整数	浮点	浮点

Python Shell实例：

```

1  >>>a = 1 + True
2  >>>print(a)
3  2
4  >>> a = 1.0 + 1
5  >>>type(a)
6  <class 'float'>
7  >>>print(a)
8  2.0
9  >>>a = 1.0 + True
10 >>>print(a)
11 2.0
12 >>>a = 1.0 + 1 + True
13 >>>print(a)

```

显式类型转换

再不能隐式类型转换的情况下，就只能用显式类型转换了。除了复数之外，三种数字类型（整数，浮点，布尔）都有自己的转换函数，分别是int（），float（），bool（）

- int（）函数可以将布尔、浮点、字符串数据转化成整数。
- float（）函数可以将布尔、整数、字符串类型转化成浮点数。

Python Shell实例：

```
1  >>>int(False)
2  0
3  >>>int(True)
4  1
5  >>>int (19.6)
6  19
7  >>>float(5)
8  5.0
9  >>>float(False)
10 0.0
11 >>>float(True)
12 1.0
```

字符串类型

Python中字符串类型是str，不是string。Python中有3种字符串表示方法

- 普通字符串：采用一对单引号“'”或一对双引号“””包裹起来
- 原始字符串：在普通字符串前面加r，字符串特殊字符不会发生转义
- 长字符串：字符串中包含了换行符缩进符等排版字符，可以使用三对双引号“"""”或三对单引号“'””包裹起来。

python运算符。

Python语言支持以下类型的运算符：

- 算术运算符
- 比较（关系）运算符
- 赋值运算符
- 逻辑运算符
- 位运算符

- 成员运算符
- 身份运算符
- 运算符优先级

接下来让我们一个个来学习Python的运算符。

以下假设变量a为10，变量b为21:

运算符	描述	实例
+	加-两个对象相加	a+b输出结果31
-	减-得到负数或是一个数减去另一个数	a-b输出结果-11
*	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 210
/	除-x除以y	b/a输出结果2.1
%	取模-返回除法的余数	b%a输出结果 2.1
**	幂-返回x的y次幂	a ** b 为10的21次方
//	取整除-向下取接近除数的整数	9//2输出结果4， -9//2输出结果-5

其他见实验程序.....

```

1  # coding = UTF-8
2  a = 10
3  b = 21
4  print("a + b = ", a + b)
5  print("a - b = ", a - b)
6  print("a * b = ", a * b)
7  print("a / b = ", a / b)
8  print("a % b = ", a % b)
9  print("a ** b = ", a ** b)
10 print("9 // 2 = ", 9 // 2)
11 print("-9 // 2 = ", -9 / 2)

```

关系、逻辑运算符

扩展：‘+’号还可以把两个字符串连接起来，‘*’号可以倍增！ Python Shell实例

```
1 >>> 'hello' + 'world'
2 'helloworld'
3 >>> 'hello' * 2
4 'hellohello'
```

关系运算符

关系运算符无需多说，Python中有6种，和c++一模一样：==、!=、>、<、>=、<=。具体说明见下表

运算符	名称	例子	说明
==	等于	a == b	a等于b时返回True，否则返回False
!=	不等于	a != b	与==相反
>	大于	a > b	a大于b时返回True，否则返回False
<	小于	a < b	a小于b时返回True，否则返回False
>=	大于等于	a >= b	a大于或等于b时返回True，否则返回False
<=	小于等于	a <= b	a小于或等于b时返回True，否则返回False

在Python Shell运行实例：

```
1 """
2 Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC
3 Type "help", "copyright", "credits" or "license()" for more
4 information.
5 """
6 #所以注释
7 >>> a = 1
8 >>> b = 2
9 >>> a > b
10 False
11 >>> a < b
12 True
13 >>> a <= b
14 True
15 >>> a >= b
16 False
17 >>> a == b
18 False
19 >>> a != b
20 True
```

Python中关系运算符可用于比较序列或数字，整数、浮点数都是对象，可以使用关系运算符进行比较。

Python Shell 运行实例：

```

1  >>> a = 'Hello'
2  >>> b = 'Hello'
3  >>> a == b
4  True
5  >>> a = 'World"
6  SyntaxError: EOL while scanning string literal
7  >>> a = 'world'
8  >>> a > b
9  True
10 >>> a < b
11 False
12 >>> a = []
13 >>> b = [1, 2]
14 >>> a == b
15 False
16 >>> a < b
17 True
18 >>> a = [1, 2]
19 >>> a == b
20 True
21 >>>

```

[]是一个列表，列表也可以比较

逻辑运算符

逻辑运算符也不用多说原理，直接开干！

运算符	名称	例子	说明
not	逻辑非	not a	1->0 0->1
and	逻辑与	a and b	a&b true 1 else 0
or	逻辑或	a or b	a&b flase 0 else 1

说明学过c的人都会。

赋值运算符

赋值运算符还是无需多说，只看下面Python赋值运算符内容表

运算符	名称	例子	说明
=	直接赋值	a = b	把b的值直接给a
+=	加赋值	a += b	等价于a = a + b
-=	减赋值	a -= b	等价于a = a - b
*=	乘赋值	a *= b	等价于a = a * b
/=	除赋值	a /= b	等价于a = a / b
%=	取余赋值	a %= b	等价于a = a % b
**=	幂赋值	a **= b	等价于a = a ** b
//=	整除赋值	a //= b	等价于a = a // b

其他运算符

Python还有其他的一些运算符，先介绍两个“测试”运算符，同一性测试运算符和成员测试运算符。

同一性测试运算符

这种运算符就是测试两个对象是否同一个对象，类似 运算符，不同之处在于 是测试两个对象的内容，而同一性测试运算符只测试对象。

同一性测试运算符有两个：is和is not，is是判断同一，is not 是判断不是同一。

成员测试运算符

成员测试运算符可以测试在一个序列对象中是否包含某一个元素（所谓成员），成员测试运算符有两个：in和not in。

```
1  #!/usr/bin/python3
2  #coding = UTF-8
3
4  字符串 = 'hello'
5  string_a = 'Hello'
6  print('e' in 字符串) #True
7  print('ell' not in string_a) #False
8
9  a = [0, 3]
10 print(type(a))
```

```
11
12  print('--')
13
14  list_a = [1, 2]
15  print(2 in list_a)      #True
16  print(1 not in list_a) #False
```

```
1  #!/usr/bin/python3
2  #coding = UTF-8
3
4  class Person:
5      def __init__(self, name, age):
6          self.name = name
7          self.age = age
8
9  p1 = Person('Tony', 18)
10 p2 = Person('Tony', 18)
11
12 print(p1 == p2)      #False
13 print(p1 is p2)      #False
14
15 print(p1 != p2)      #True
16 print(p1 is not p2) #True
```

控制语句

程序设计中的控制语句有三种，即顺序、分支和循环结构，Python中的控制语句有以下几类

- 分支语句：if
- 循环语句：while和for
- 跳转语句：break、continue和return

分支语句

Python中分支语句只有if语句，这个if语句使得程序具有了“判断能力”，能够像人类的大脑一样分析问题。if语句有if结构、if-else结构和elif结构三种。

if结构

例题：输入分数，计算优秀、中等、差。

if语句结构如下：

```
1  if 条件:
2      语句组
3      .....
```

代码：

```
1  #coding = utf-8
2  #!/usr/bin/python3
3
4  import sys
5
6  score = int(sys.argv[1])
7
8  if score >= 85:
9      print("您真优秀")
10 if score < 60:
11     print("您需要加倍努力")
12 if score >= 60:
13     print("您的成绩还可以，仍要继续努力！")
```

if-else结构

几乎所有的计算机语言都有这个结构，先判断条件，如果返回值为True，那么执行语句，否则执行else内的语句。

```
1  if-else 结构如下：
2  if 条件:
3      语句组1
4  else:
5      语句组2
```

elif结构

elif结构就是c++语言中的else if结构，elif实际上是if-else的多重嵌套，不用多说

if-else结构实例

```
1  #coding = utf-8
2  #!/usr/bin/python3
3
4  import sys
5
6  score = int(sys.argv[1])
7
8  if score >= 60:
9      print("及格")
10     if score >= 90:
11         print("优秀")
12 else:
13     print("不及格")
```

elif结构实例

```
1  #coding = utf-8
2  #!/usr/bin/python3
3
4  import sys
5
6  score = int(sys.argv[1])
7
8  if score >= 90:
9      grade = 'A'
10 elif score >= 80:
11     grade = 'B'
12 elif score >= 70:
13     grade = 'C'
14 elif score >= 60:
15     grade = 'D'
16 else:
17     grade = 'F'
18
19 print("Grade = " + grade)
```

循环语句

循环语句能够使程序代码重复执行。Python支持while和for两种循环语句。

while语句

while语句是一种先判断后执行的循环语句，格式如下：

```
1  while 循环条件:
2      语句组
3  [else:
4      语句组
5  ]
```

while循环没有初始化语句，循环次数是不可知的，只要循环条件满足，循环就会一直执行循环体。while循环中可以带有else语句，else语句将在后面介绍。

for语句

for语句是应用最广泛、功能最强的一种循环语句。Python语言中没有c语言风格的for语句，它的for语句相等于Java中增强for循环语句，只用于序列，序列包括字符串、列表和元组。

for语句格式如下：

```
1  for 迭代变量 in 序列:
2      语句组
3  [else:
4      语句组]
```

while语句实例：

```
1  #coding = utf-8
2  #!/usr/bin/python3
3
4  i = 0
5
6  while i * i < 100_000:
7      i += 1
8
9  print("i = {0}".format[i])
10 print("i * i = {0}".format(i * i))
```

输出结果如下：

```
1  i = 317
2  i * i = 100489
```

for语句实例：

```
1  #coding - utf-8
```



```

2  #!/usr/bin/python3
3
4  print("-----范围-----")
5  for num in range(1, 10):
6      print("{0} x {0} = {1}".format(num, num * num))
7  print("-----字符串-----")
8  for item in 'Hello':
9      print(item)
10
11  numbers = [43, 32, 53, 54, 75, 7, 10]
12
13  print("-----整数列表-----")
14  for item in numbers:
15      print("Count is : {0}".format(item))

```

输出结果：

```

1  -----范围-----
2  1 x 1 = 1
3  2 x 2 = 4
4  3 x 3 = 9
5  4 x 4 = 16
6  5 x 5 = 25
7  6 x 6 = 36
8  7 x 7 = 49
9  8 x 8 = 64
10 9 x 9 = 81
11 -----字符串-----
12 H
13 e
14 l
15 l
16 o
17 -----整数列表-----
18 Count is : 43
19 Count is : 32
20 Count is : 53
21 Count is : 54
22 Count is : 75
23 Count is : 7
24 Count is : 10

```

for语句的range()函数是创建一个范围对象，它的取值范围是 $1 \leq \text{range}(1, 10) < 10$ ，步长默认为1，总共10个整数

Python跳转语句

跳转语句能够改变程序的执行顺序，可以实现程序的跳转。Python有3种跳转语句：break、continue和return。本章重点介绍break和continue的使用。return将在后面章节介绍。

Break语句

break语句可用于while语句和for语句，它的作用是强行退出循环体，不再执行循环中剩余的语句。

下面是一个实例，代码如下：

```
1  #coding = utf-8
2  #!/usr/bin/python3
3
4  for item in range(10):
5      if item == 3:
6          #跳出循环
7          break
8      print("Count is : {}".format(item))
```

上述代码item变量默认从0开始迭代。运行结果如下：

```
1  Count is : 0
2  Count is : 1
3  Count is : 2
```

Continue 语句

Continue语句用来结束本次循环，跳过循环体中尚未执行的语句，接着进行终止条件的判断，已决定是否继续循环。

实例，代码如下：

```
1  #coding = utf-8
2  #!/usr/bin/python3
3
4  for item in range(10):
5      if item == 3:
6          continue
7      print("Count is : {}".format(item))
```

上述代码中，当条件item==3的时候执行continue语句，continue语句会终止本次循环，循环体中continue之后的语句将不再执行，进行下次循环，所以输出结果中没有3

ans:

```
1  Count is : 0
2  Count is : 1
3  Count is : 2
4  Count is : 4
5  Count is : 5
6  Count is : 6
7  Count is : 7
8  Count is : 8
9  Count is : 9
```

While和For中的else语句

while和for语句的else和if中的else语句不同，这里的else是在循环体正常结束时才运行的代码，当循环被中断时不执行，break、return和异常抛出都会中断循环。

while、for加else实例：(if-break被注释掉)

```
1  #coding = utf-8
2  #!/usr/bin/python3
3
4  i = 0
5  while i * i < 10:
6      i += 1
7      #if == 3:
8      #    break
9      print("{0} * {0} = {1}".format(i, i * i))
10 else:
11     print('While Over!')
12
13 #-----
14
15 for item in range(5):
16     if item == 3:
17         break
18     print("Count is : {0}".format(item))
19 else:
20     print("For Over!")
```

结果:

```
1 1 * 1 == 1
2 2 * 2 == 4
3 3 * 3 == 9
4 4 * 4 == 16
5 While Over!
6 Count is : 0
7 Count is : 1
8 Count is : 2
```

使用范围

for语句在使用时需要用范围函数，范围在Python中是range函数，表示一个整数序列，创建范围对象需要使用range()函数，range()函数语法如下：

```
1 range([start,] stop[,step])
```

range函数也可以使用复数范围，创建一个递减范围，示例如下：

```
1 #coding = utf-8
2 #!/usr/bin/python3
3
4 for item in range(1, 10, 2):
5     print("Count is : {0}".format(item))
6
7 print("-----")
8
9 for item in range(1, -10, -3):
10     print("Count is : {0}".format(item))
```

输出结果如下：

```
1 Count is : 1
2 Count is : 3
3 Count is : 5
4 Count is : 7
5 Count is : 9
6 -----
7 Count is : 0
8 Count is : -3
9 Count is : -6
10 Count is : -9
```

忘了说了，在Python中，整数为了表示清晰可以用下划线隔开，比如100_000和100000是一样的，小数在小数点前面可以有多个0，比如003.1415926等于3.1415926

Python数据结构！ 结构！

当你有很多书的时候，你会考虑买一个书柜，将你的书分门别类地摆放进去。使用了书柜不仅使房间变得整洁，也便于以后使用书时查找。在计算机程序中会有很多数据，这些数据也需要容器将它们管理起来，这就是**数据结构**

常见的有数组（Array）、集合（Set）、列表（list）、队列（queue）、链表（linkedlist）、树（tree）、堆（heap）、栈（stack）和字典（dictionary）等结构。

Python中数据结构主要有序列、集合和字典

注意:Python中并没有数组结构，因为数组要求元素类型是一致的。而Python作为动态类型语言，不强制声明变量的数据类型，也不强制检查元素的数据类型，不能保证元素的数据类型一致，所以Python中没有数组结构。

元组

元组（tuple）是一种序列（sequence）结构

序列

序列包括的结构有列表（list）、字符串（string）、元组、范围（range）和字节序列（bytes）。序列可以进行的操作有索引、分片、加和乘。

1) 索引操作

序列中第一个元素的索引是0，其他元素的索引是第一个元素的偏移量。可以有正偏移量，称为正值索引；也可以有负偏移量，称为负值索引。正值索引的最后一个元素索引是“序列长度-1”，负值索引最后一个元素索引是“-1”。例如Hello字符串，它的正值索引如表

索引	0	1	2	3	4
字符串	H	e	l	l	o

索引	0	-4	-3	-2	-1
字符串	H	e	l	l	o

正值索引和负值索引表[↑]

PythonShell实例:

```
1 Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC
  v.1916 32 bit (Intel)] on win32
2 Type "help", "copyright", "credits" or "license()" for more
  information.
3 >>> a = "Hello"
4 >>> a[0]
5 'H'
6 >>> a[1]
7 'e'
8 >>> a[4]
9 'o'
10 >>> a[-1]
11 'o'
12 >>> a[-2]
13 'l'
14 >>> a[5]
15 Traceback (most recent call last):
16   File "<pyshell#6>", line 1, in <module>
17     a[5]
18 IndexError: string index out of range
19 >>> max(a)
20 'o'
21 >>> min(a)
22 'H'
23 >>> len(a)
24 5
25 >>> #max函数返回最后一个函数
26 >>> #min函数返回第一个元素
27 >>> #len返回长度
```

2) 序列和加和乘

PythonShell实例:

```
1 >>> a = "Hello"
2 >>> a * 3
3 'HelloHelloHello'
4 >>> print(a)
5 Hello
6 >>> a += ' '
7 >>> a += 'World'
8 >>> print(a)
9 Hello World
10 >>>
```

3)序列分片

序列的分片 (slicing) 就是从序列中切出小的子序列。分片使用分片运算符，分片运算符有两种形式。

- [start:end]:start是开始索引，end是结束索引
- [start:end:stop]:step是步长，步长可以为正数，也可以为负数

实例：

```
1 >>>a = 'Hello'
2 >>>a[1:3]
3 'el'
4 >>>a[:3]
5 'Hel'
6 >>>a[0:3]
7 'Hel'
8 >>>a[0:]
9 'Hello'
10 >>>a[0:5]
11 'Hello'
12 >>>a[: ]
13 'Hello'
14 >>>a[1:-1]
15 'ell'
```

附加程序：

创建元组

元组是一种不可变的序列，一旦创建就不能修改。创建元组可以使用 `tuple([iterable])` 函数或直接用逗号“，”将元素分隔。

Python Shell实例代码

```

1  >>>21, 32, 43, 45                                ①
2  (21, 32, 43, 45)
3  >>>(21, 32, 43, 45)                                ②
4  (21, 32, 43, 45)
5  >>>a = (21, 32, 43, 45)
6  >>>print(a)
7  (21, 32, 43, 45)
8  >>>('Hello', 'World')                            ③
9  ('Hello', 'World')
10 >>>('Hello', 'World', 1, 2, 3)                    ④
11 ('Hello', 'World', 1, 2, 3)
12 >>>tuple([21, 32, 43, 45])                        ⑤
13 (21, 32, 43, 45)

```

代码第①行创建了一个有4个元素的元组，创建元组时使用小括号把元素括起来不是必须的；

代码第②行使用小括号将元素括起来，这只是为了提高程序的可读性

代码第③行创建了一个字符型元组

代码第④行创建了一个字符串和整数混合的元组。Python中没有强制声明数据类型，因此元组中的元素可以是任何数据类型

另外，元组还可以使用 `tuple([iterable])` 函数创建，参数iterable可以是任何可迭代对象。代码第⑤行使用了 `tuple` 函数创建元组对象，实参`[21, 32, 43, 45]`是一个列表，列表是可迭代对象，可以作为`tuple()`函数参数创建元组对象。

创建元组还需注意极端情况：⚡

```

1  >>> a = (21)
2  >>> type(a)
3  <class 'int'>
4  >>> a = (21, )
5  >>> type(a)
6  <class 'tuple'>
7  >>> a = ()
8  >>> type(a)
9  <class 'tuple'>

```

访问元组

元组作为序列可以通过下标索引访问其中的元素，也可以对其进行分片

PythonShell实例：


```
1  >>>a = ('Hello', 'World', 1, 2, 3)           ①
2  >>>a[1]
3  'World'
4  >>>a[1:3]
5  ('World', 1)
6  >>>a[2:]
7  (1, 2, 3)
8  >>>a[:2]
9  ('Hello', 'World')
```

上述代码第①行是元组a，a[1]是访问元组的第二个元素，表达式a[1:3]、a[2:]和a[:2]都是分片操作

元组还可以进行拆包(Unpack)操作，就是将元组的元素取出给不同变量

PythonShell实例：

```
1  >>> a = ('Hello', 'World', 1, 2, 3)
2  >>> str1, str2, n1, n2, n3 = a
3  >>> str1
4  'Hello'
5  >>> str2
6  'World'
7  >>> n1
8  1
9  >>> n2
10 2
11 >>> n3
12 3
13 >>> str1, str2, *n = a
14 >>> str1
15 'Hello'
16 >>> str2
17 'World'
18 >>> n
19 [1, 2, 3]
```

**n可以直接获取剩余的元素*

遍历元组

一般用for语句遍历元组，实例代码

```

1  #coding=utf-8
2  #!/usr/bin/python3
3
4
5  a = {21, 32, 43, 45}
6
7  for item in a:
8      print(item)
9
10 print('-----')
11
12 for i, item in enumerate(a):
13     print('{0} - {1}'.format(i, item))

```

运行结果：

```

1  21
2  32
3  43
4  45
5  -----
6  0 - 21
7  1 - 32
8  2 - 43
9  3 - 45

```

其中enumerate(a)函数可以获取元组对象

附加程序

```

1  #coding = utf-8
2
3  a = (20)
4  print(type(a))
5
6  a = (20,)
7  print(type(a))
8  #tuple
9
10 a = (20, 30, 40, 50, 60)
11 print(a)
12 print(a[1])
13 print(a[1:3])
14
15 a = ('Hello', 'World', 1, 2, 3)
16 str1, str2, n1, n2, n3 = a
17 print(str1)

```

```
18 print(str2)
19 print(n1, "", n2, "", n3)
20
21 str1, str2, *n = a
22 print(n)
```

列表

列表 (List) 也是一种序列结构，和元组不一样，列表具有可变性，可以追加、插入、删除和替换列表中的元素。

列表创建

创建列表可以使用list([iterable])函数，或者用中括号[]将元素括起来，元素之间用逗号分隔。在Python Shell中运行实例代码如下：

```
>>>
>>> [20, 10, 50, 40, 30]
[20, 10, 50, 40, 30]
>>> []
[]
>>> ['Hello', 'World', 1, 3, 3]
['Hello', 'World', 1, 3, 3]
>>> a = [10]
>>> type(a)
<class 'list'>
>>> a = [10, ]
>>> type(a)
<class 'list'>
>>> list((20, 10, 50, 40, 30))
[20, 10, 50, 40, 30]
>>>
```

为什么今天的交互要发一张截图呢？原因我发现visual studio2019安装上Python插件之后，点击视图，也有Python交互解释器，还有代码高亮，自动补全，比IDLE好用多了！

```
1  #同一个代码片
2  >>> [20, 10, 50, 40, 30]           ①
3  [20, 10, 50, 40, 30]
4  >>> []
5  []
6  >>> ['Hello', 'World', 1, 3, 3] ②
7  ['Hello', 'World', 1, 3, 3]
8  >>> a = [10]                      ③
9  >>> type(a)
10 <class 'list'>
11 >>> a = [10, ]                    ④
12 >>> type(a)
13 <class 'list'>
14 >>> list((20, 10, 50, 40, 30)) ⑤
15 [20, 10, 50, 40, 30]
16 >>>
```

接着说，代码第①行创建了一个有五个元素的列表，注意和元组不一样，中括号不能省略，如果省略就变成元组了。代码第②行创建了一个字符串和整数混合的列表。代码第③行创建一个只有一个元素的列表，中括号不能省略。

另外，代码第⑤行用list([iterable])函数创建列表。

追加元素

列表中追加单个元素可以使用append()方法。如果想追加另一列表，可以使用+运算符或者extend()方法

append()方法语法：

list.append(x)

其中x参数是要追加的单个元素值

extend()方法语法：

list.extend(t)

其中t参数是追加的另外一个列表

Python Shell实例：

```
1  >>> student_list = ['张三', '李四', '王五']
2  >>> student_list.append('董六')
3  >>> student_list
4  ['张三', '李四', '王五', '董六']
```

```

5  >>> student_list += ['刘备', '关羽']
6  >>> student_list
7  ['张三', '李四', '王五', '董六', '刘备', '关羽']
8  >>> student_list.extend(['张飞', '赵云'])
9  Traceback (most recent call last):
10     File "<stdin>", line 1, in <module>
11     NameError: name 'student_list.extend' is not defined
12  >>> #一下子输错了
13  ...
14  >>> student_list.extend(['张飞', '赵云'])
15  >>> student_list
16  ['张三', '李四', '王五', '董六', '刘备', '关羽', '张飞', '赵云']
17  >>>

```

插入元素

插入元素可以用insert()方法。该方法可以指定索引位置插入一个元素

insert()方法语法

list.insert(i, x)

其中参数i是要插入的序列，参数x是要插入的元素数值

Python Shell实例

```

1  >>> student_list = ['张三', '李四', '王五']
2  >>> student_list
3  ['张三', '李四', '王五']
4  >>> student_list.insert(2, '刘备')
5  >>> student_list
6  ['张三', '李四', '刘备', '王五']

```

替换元素

这一点和c++的数组很像，直接修改下标中元素

实例：

```

1  >>> student_list = ['张三', '李四', '王五']
2  >>> student_list[0] = "诸葛亮"
3  >>> student_list
4  ['诸葛亮', '李四', '王五']

```

删除元素

一种放法是remove()方法，另一种是pop()

1) remove

remove方法从左到右查找列表中的元素，如果找到匹配元素则删除，注意如果找到多个匹配元素，只是删除第一个，如果没有找到会抛出错误。

实例：

```
1  >>> student_list = ['张三', '李四', '王五']
2  >>> student_list[0] = "诸葛亮"
3  >>> student_list
4  ['诸葛亮', '李四', '王五']
5  >>>
6  >>>
7  >>>
8  >>> student_list = ['张三', '李四', '王五', '王五']
9  >>> student_list.remove('王五')
10 >>> student_list
11 ['张三', '李四', '王五']
12 >>> student_list.remove('王五')
13 >>> student_list
14 ['张三', '李四']
15 >>> student_list.remove('王五')
16 Traceback (most recent call last):
17   File "<pyshell#26>", line 1, in <module>
18     student_list.remove('王五')
19 ValueError: list.remove(x): x not in list
20 >>> student_list.append('王五')
21 >>> student_list
22 ['张三', '李四', '王五']
23 >>> student_list.pop()
24 '王五'
25 >>> student_list
26 ['张三', '李四']
27 >>>
```

pop是弹出列表最后一个元素

列表的其他常用方法

前面介绍列表的追加、插入和删除时，已经介绍了一些方法。事实上列表还有很多方法，下面再来介绍一些常用方法。

- reverse(): 倒置列表
- copy(): 复制列表
- clear(): 清除列表中的所有元素

- `index(x, i, j)`: 返回查找x第一次出现的索引, i是开始查找索引, j是结束查找索引, 该方法继承自序列, 元组和字符串也可以使用该方法
- `count(x)`: 返回x出现的次数, 该方法继承自序列, 元组与字符串也可以使用该方法

Python Shell中运行实例:

```
1  >>> a = [21, 32, 43, 45]
2  >>> a.reverse()
3  >>> a
4  [45, 43, 32, 21]
5  >>> b = a.copy()
6  >>> b
7  [45, 43, 32, 21]
8  >>> a.clear()
9  >>> a
10 []
11 >>> b
12 [45, 43, 32, 21]
13 >>> a = [45, 43, 32, 21, 32]
14 >>> a.count(32)
15 2
16 >>> student_list = ['张三', '李四', '王五']
17 >>> student_list.index('王五')
18 2
19 >>> student_tuple = ('张三', '李四', '王五')
20 >>> student_tuple.index('王五')
21 2
22 >>> student_tuple.index('李四', 1, 2)
23 1
```

列表推导式

Python中有一种特殊表达式——推导式, 它可以将一种数据结构作为输入, 经过过滤、计算等处理, 最后输出另一种数据结构。根据数据结构的不同可分为列表推导式、集合推导式和字典推导式

如果想获得0~9中偶数的平方数列, 可以通过for循环实现

```
1  n_list = []
2  for x in range(10):
3      if x % 2 == 0:
4          n_list.append(x ** 2)
5
6  print(n_list)
```

输出结果:

```
1 [0, 4, 16, 36, 64]
```

也可以用列表推导式实现，代码如下：

```
1 n_list = [x ** 2 for x in range(10) if x % 2 == 0]
2 print(n_list)
```

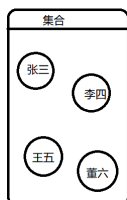
列表推导式格式：

```
n_list = [x ** x 1 for x 2 in range(10) 3 if x % 2 == 0 4]
```

1.集合

集合 (set) 是一种可迭代的、无序的、不能包含重复元素的数据结构。图中是一个班级的集合，其中包含一些学生，这些学生是无序的，不能通过序号访问，而且

不能重复。



提示：与序列比较，序列中的元素是有序的，可以**重复出现**，而且集合中的元素是无序的，且不能有重复的元素。

序列强调的是有序，集合强调的是不重复，而且当没有重复的元素时，序列和集合可以互相替换。

集合又分为**可变集合**和**不可变集合**

1.1创建可变集合

可变集合类型是set，创建可变集合可以使用 `set([iterable])` 函数，或者用大括号{ }将元素括起来，元素之间用逗号分隔

Python Shell实例：

```
1 >>> a = {'张三', '李四', '王五'}
2 >>> a
3 {'张三', '李四', '王五'}
4 >>> a = {'张三', '李四', '王五', '王五'}
5 >>> a
6 {'张三', '李四', '王五'}
7 >>> #集合中如果有重复元素，创建时会自动删除重复元素↑
8 ...
9 >>> len(a)
```



```

10 3
11 >>> b = { }
12 >>> type(a)
13 <class 'set'>
14 >>> type(b)
15 <class 'dict'>

```

空的集合会变成字典，代码中 `b` 不是集合，是字典 `dict`，如果要创建空集合，要使用 `set()` 函数。

1.2 修改可变集合

可变集合类似于列表，可变集合的内容可被修改，可以**插入**和**删除**元素。修改可变集合有几个常用的方法。

- `add(elem)`: 添加元素，如果元素已经存在，则不能添加，不会抛出错误
- `remove(elem)`: 删除元素，如果元素不存在，则会抛出错误
- `discard(elem)`: 删除元素，如果元素不存在，不会抛出错误
- `pop()`: 删除返回集合中任意一个元素，返回值是删除的元素
- `clear()`: 清空

Python Shell 实例:

```

1  >>> student_set = {'张三', '李四', '王五'}
2  >>> student_set.add('董六')           # 随机添加元素，因为集合没有顺序
3  >>> student_set
4  {'张三', '董六', '李四', '王五'}
5  >>> student_set.remove('董六')
6  >>> student_set
7  {'张三', '李四', '王五'}
8  >>> student_set.remove('董六')
9  Traceback (most recent call last):
10     File "<stdin>", line 1, in <module>
11  KeyError: '董六'
12  >>> #报错，没有董六
13  ...
14  >>> student_set.discard('董六')
15  >>> student_set
16  {'张三', '李四', '王五'}
17  >>> #discard()不会抛出错误
18  ...
19  >>> student_set.pop()
20  '张三'
21  >>> student_set
22  {'李四', '王五'}
23  >>> student_set.pop()
24  '李四'

```

```
25 >>> student_set
26 {'王五'}
27 >>> student_set.clear()
28 >>> student_set
29 set()
```

1.3遍历集合

集合是无序的，没有索引，不能通过下标访问。但可以遍历集合，访问集合的每一个元素。

实例代码：

```
1 #coding = utf-8
2
3 student_set = {'张三', '李四', '王五'}
4
5 for item in student_set:
6     print(item)
7
8 print('-----')
9 for i, item in enumerate(student_set):
10     print('{0} - {1}'.format(i, item))
```

输出结果

```
1 张三
2 李四
3 王五
4 -----
5 0 - 张三
6 1 - 李四
7 2 - 王五
```

函数式编程

函数

程序中反复执行的代码可以封装到一个代码块中，这个代码块模仿了数学中的函数，具有函数名、参数和返回值，这就是程序中的函数。

Python中的函数很灵活，它可以在模块中、但是在类之外定义，即函数，其作用域是当前模块；也可以在别的函数中定义，即嵌套函数；还可以在类中定义，即方法。

定义函数

在前面的学习过程中用到了一些函数，如 `len()`、`min()` 和 `max()`，这些函数都是由Python官方提供的，称为内置函数(*Built-in Functions, BIF*)

自定义函数

本节介绍自定义函数，自定义函数的语法如下

```
1 def 函数名 ( 参数列表 ):  
2     函数体  
3     return 返回值
```

在Python中定义函数时，关键字是def，函数名需要符合标识符命名规范见[以前博客---Python命名规范](#)。多个参数列表之间可以使用逗号“，”分隔，当然函数也可以没有参数。如果函数有返回数据，就需要在函数体最后使用return语句将数据返回；如果没有返回数据，则函数体中可以使用return None或者省略return语句。

函数定义实例如下：

```
1 # coding = utf-8  
2 #!/usr/bin/python3  
3  
4 def rectangle_area(width, height):  
5     area = width * height  
6     return area  
7  
8 r_area = rectangle_area(320.0, 480.0)  
9 print("320 x 480的长方形的面积:{0:.2f}".format(r_area))
```

函数参数

Python中的函数参数很灵活，具体体现在传递参数有都中形式上。本节介绍几种不同形式的参数和调用方式。

使用关键字参数调用函数

为了提高函数调用的可读性，在函数调用时可以使用关键字参数调用。采用关键字参数调用函数，在函数定义时不需要做额外工作。

实例代码如下：

```

1  -*- coding = utf-8 -*-
2
3  def print_area(width, height):
4      area = width * height
5      print("{0} x {1} 长方形的面积:{2}".format(width, height, area))
6
7  print_area(320.0, 48.0)      # 没有采用关键字参数函数调用
8  print_area(width = 320.0, height = 480.0)      # 采用关键字参数函数调用
9  print_area(320.0, height = 480.0)      # 采用关键字参数函数调用
10 # print_area(width = 32.0, height)      #发生错误
11 #原因--height没有数值,系统就会认为是一个变量,而这个变量没有定义声明
12 print_area(height = 480.0, width = 320.0)      #采用关键字参数函数调用
13

```

代码很简单，就不用了多说了

参数默认值

在定义函数的时候可以为参数设置一个默认值，调用函数时可以忽略该参数。

实例：(下面两段代码是一体的)

```

1  # -*- coding = utf-8 -*-
2
3  def make_coffee(name = "卡布奇诺"):
4      return "制作一杯{0}咖啡".format(name)

```

上述代码定义了 `makeCoffee()` 函数，其中把卡布奇诺☺挺好喝的设置为了默认值。在参数列表中，默认值可以跟在参数类型后面。**在调用的时候，如果调用者没有传递参数，则使用默认值**，调用代码如下：

```

1  coffee1 = make_coffee("拿铁")
2  coffee2 = make_coffee()
3
4  print(coffee1)
5  print(coffee2)

```

其中 `coffee1 = make_coffee("拿铁")` 代码是传递“拿铁”，没有使用默认值。
`coffee2 = make_coffee()` 这一行没有传递参数，因此使用默认值。

StdOut:

```

制作一杯拿铁咖啡
制作一杯卡布奇诺咖啡

```

提示 (新手可以不看)： Java语言中 `make_coffee()` 函数可以采用重载实现多个版本。Python不支持函数重载，而是使用参数默认值的方式提供类似的重载功能。

函数返回值

Python函数的返回值也是比较灵活的（Python的东西都比较灵活），主要有三种形式：无返回值、单一返回值和多返回值。

无返回值函数

有的函数只是为了处理某个过程，此时可以将函数设计为无返回值的。所谓无返回值，事实上是返回`None`，`None`表示没有实际意义的数据。

```
1  -*- coding = utf-8 -*-
2
3  def show_info(sep = ':', **info):    #可变参数,下面讲一下
4      """定义**可变参数函数, dict"""
5      print('-----info-----')
6      for key, value in info.items():
7          print('{0} {2} {1}'.format(key, value, sep))
8          return                      #return None
9
10 result = show_info('->', name = 'Tony', age = 18, sex = True)
11 print(result)
12
13
14 def sum(*numbers, multiple = 1):
15     """定义*可变参数函数, tuple"""
16     if len(numbers) == 0:
17         return
18     total = 0.0
19     for number in numbers:
20         total += number
21     return total * multiple
22
23 print(sum(30.0, 80.0))
24 print(sum(multiple = 2))
```

StdOut:

```
-----info-----  
name -> Tony  
None  
110.0  
None
```

1. 上述代码 `show_info()` 函数中有一个 `**info` 的参数，这是一个可变参数，下面详细解释一下。Python中函数的参数个数可以变化，这种参数称为可变参数。Python中的可变参数有两种，即参数前加 `*` 或 `**` 形式，`*` 可变参数被组装成一个元组，`**` 可变参数在函数中被组装成一个字典。

1. `*` 可变参数

下面看一个实例：（部分代码）

```
1 def sum(*numbers, multiple = 1):  
2     total = 0.0  
3     for number in numbers:  
4         total += number  
5     return total * multiple  
6  
7 print(sum(100.0, 20.0, 30.0))  
8 print(sum(30.0, 80.0))  
9 print(sum(30.0, 80.0, multiple = 2))  
10  
11 double_tuple = (50.0, 60.0, 0.0)  
12 print(sum(30.0, 80.0, *double_tuple))
```

```
1  
2 > StdOut:  
3 >  
4 > > 150.0  
5 > >  
6 > > 110.0  
7 > >  
8 > > 220.0  
9 > >  
10 > > 220.0  
11  
12 不用说都懂了吧:smirk:
```

14 2. `**` 可变参数

15
16 下面看一个实例：（部分代码）

```
17 ```python  
18 def show_info(sep = ':', **info):
```

```

19 print('-----info-----')
20 for key, value in info.items():
21     print('{0} {2} {1}'.format(key, value, sep))
22
23 show_info('->', name = 'Tony', age = 18, sex = True)
24 show_info(student_name = 'Tony', sex = True, sep = '=')

```

都不用说吧？挺简单的

2. 返回值为空的时候可以用 `return` 或 `return None` 都可以。

多返回值函数

有时候需要函数返回多个值，实现返回多个值的方式有很多，简单的方式是使用元组返回多个值，因为元组可以容纳多个数据，另外元组是不可变的，使用起来比较安全。

实例：

```

1  #coding=utf-8
2
3  def position(dt, speed):
4      posx = speed[0] * dt #speed[0] 是X轴上的速度
5      posy = speed[1] * dt #speed[1] 是Y轴上的速度
6      return (posx, posy)
7
8  move = position(60.0, (10, -5)) #move是一个元组，用来存储多个返回值
9
10 print("物体位移: ({0}, {1})".format(move[0], move[1]))

```

【Python入门自学笔记专辑】——函数嵌套-Lambda表达式

函数嵌套

前言

Python的函数有很多地方不同于c++，它的函数可以嵌套！`c++程序员：望尘莫及，太可怕了！` 不过Python主要是基于c语言开发的，c工程师还是可以自豪的，c语言是要自己做功能，而python自带功能。学哪个各有好处。

正题

好了扯远了，继续说Python函数，python的一个函数可以嵌套多个函数，多个函数还可以嵌套。

```
1  def func():
2      def func1():
3          print("func1")
4      def func2():
5          print("func2")
6          def func2_1():
7              print("func2.1")
8  print("hello world")
```

比如上面的这个程序，`func` 函数中嵌套了两个函数——`func1` 和 `func2`，`func2` 又嵌套了 `func2_1`，Python是支持这种情况的。
比如：

```
1  def func():
2      def func1():
3          print("func1")
4      def func2():
5          print("func2")
6          def func2_1():
7              print("func2.1")
8      print("func")
9  func()
```

那么输出结果是：

```
1  func
```

先来一个**温馨提示**：如果要调用函数，必须把函数放在调用的那行**上面**！
再看代码：

```
1  def func():
2      choose = int(input("> "))
3      def func1():
4          print("func1")
5      def func2():
6          print("func2")
7          def func2_1():
8              print("func2.1")
9      if choose == 1:
10         func1()
11     elif choose == 2:
12         func2()
13     print("func")
14 func()
```



```
15 func()
```

运行结果：

```
1 > 1
2 func1
3 func
4 > 2
5 func2
6 func
```

分析：

先调用进 `func` 函数，然后定义两个函数 `func1`、`func2` 然后选择，函数必须在选择的上面，不信大伙可以试试。

可能出现的错误

1

```
1 def func():
2     if choose == 1:
3         func1()
4     elif choose == 2:
5         func2()
6     choose = int(input("> "))
7     def func1():
8         print("func1")
9     def func2():
10        print("func2")
11        def func2_1():
12            print("func2.1")
13    print("func")
14    func()
```

报错信息：

```
1 UnboundLocalError: local variable 'func1' referenced before
  assignment
```

原因

语句在函数上面，无法调用

解决办法

把调用语句和函数换位置

```

1  def func():
2      choose = int(input("> "))
3      def func1():
4          print("func1")
5      def func2():
6          print("func2")
7          def func2_1():
8              print("func2.1")
9      if choose == 1:
10         func1()
11     elif choose == 2:
12         func2()
13     print("func")
14     func1()

```

报错信息

```
1  NameError: name 'func1' is not defined
```

原因

调用函数最多一层，比如在函数外面，不可能跨越两层调用 `func1` 函数，跨级太多😭。

解决办法

先调用进 `func` 函数，再调用 `func1`，更高级的办法我也不知道😁呵呵o(￣▽￣)o

Lambda表达式

前言

理解了函数类型和函数对象😁，学习Lambda就简单了，就是一种函数吧，准确的说，是个解决一两步的方法，小方法。😁

正题

Lambda是一种匿名函数，匿名函数也是函数，有函数类型，也可以创建函数对象。

定义Lambda表达式格式如下：

`lambda 参数列表 : Lambda体`

Lambda是关键字声明，这是一个Lambda表达式，“参数列表”与函数的参数列表是

一样的，但不需要小括号括起来，冒号后面是“Lambda体”，Lambda表达式的主要代码在此处编写，类似于函数体☺。

注意：Lambda体部分不能是一个代码块，不能包含多余语句，**只能有一条语句**，语句会计算一个结果返回给Lambda表达式，但是与函数不同的是，不需要使用 **return** 语句返回。与其他语言中的Lambda表达式相比，Python中提供的Lambda表达式只能处理一些简单的运算。

实例：

```
1  -*- coding = utf-8 -*-
2
3  def calculate_fun(opr):
4      '''
5      #定义相加函数
6      def add(a, b):
7          return a + b
8
9      #定义相减函数
10     def sub(a, b):
11         return a - b
12     '''
13     if opr == '+':
14         # return add
15         return lambda a, b : (a + b)
16     else:
17         # return sub
18         return lambda a, b : (a - b)
19
20     f1 = calculate_fun('+')
21     f2 = calculate_fun('-')
22
23     print(type(f1))
24
25     print("10 + 5 = {}".format(f1(10, 5)))
26     print("10 - 5 = {}".format(f2(10, 5)))
```

运行结果：

```
1  <class 'function'>
2  10 + 5 = 15
3  10 - 5 = 5
```

这个Lambda返回看来是“function”？

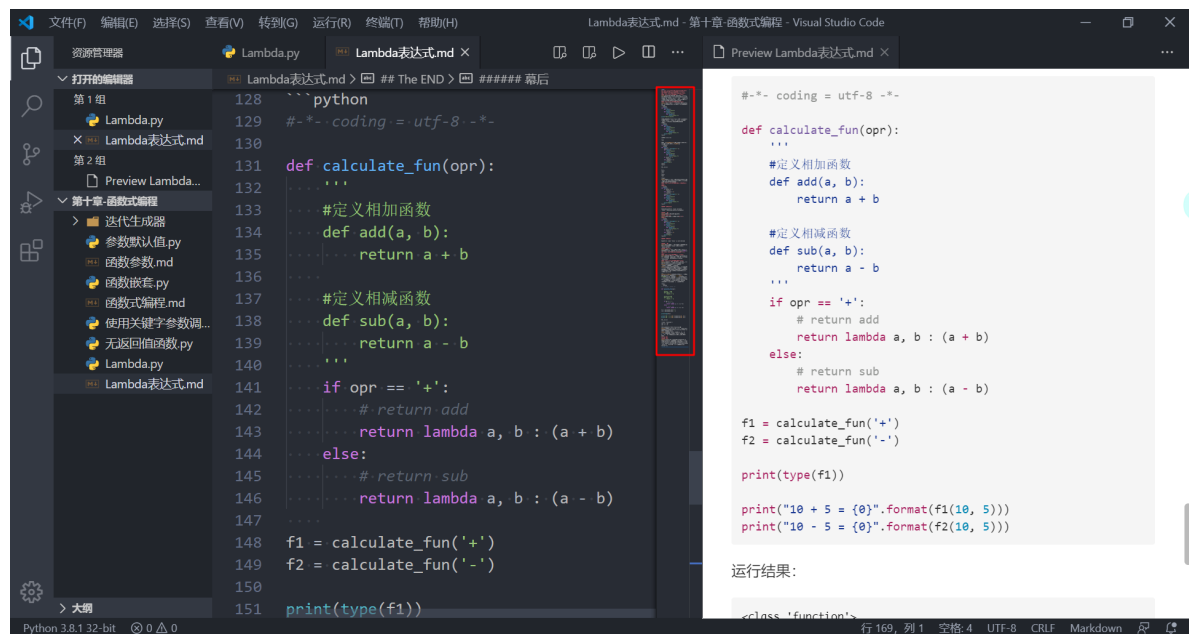
上面的程序注释部分原本是一个旧程序，大家可以把Lambda部分去掉，把注释部分恢复，可以再试试。 😊

上面代码 `return lambda a, b : (a + b)` 替代了 `add()` 函数， `return lambda a, b : (a - b)` 替代了 `sub()` 函数，使得函数更快。

The END

幕后

今天我又学习python了 😊，哈哈，本人努力为大家写了一篇好文，也是给自己这个Python小白写的整理，Lambda这些东东还是有难度的，在VScode上忙碌2小时，弄明白了！ 😊，所以麻烦点一个赞，谢谢！ 😊



```
python
#-*- coding = utf-8 -*-

def calculate_fun(opr):
    ...
    #定义相加函数
    def add(a, b):
        ...
        return a + b
    ...
    #定义相减函数
    def sub(a, b):
        ...
        return a - b
    ...
    if opr == '+':
        ...
        # return add
        return lambda a, b : (a + b)
    else:
        ...
        # return sub
        return lambda a, b : (a - b)
    ...

f1 = calculate_fun('+')
f2 = calculate_fun('-')

print(type(f1))
```

THE END

1. `x ** x` 输出表达式 ↩
2. `x` 元素变量 ↩
3. `range(10)` 输入序列 ↩
4. `if x % 2 == 0` ↩