学习C/C++之后，自学Python（3.x），刚开始对这门语言不太熟悉，记点笔记。

**最开始最直观的印象**

1. 定义一个变量时，不用int a，或者char c这样，就直接value = 123，不用声明类型，它的赋值机制与c++不太一样，它其实是先开辟一个空间，存放这个数据后，value相当于是一个引用。

还可以x , y = 4 , 8这样同时赋值。

1. score = int(input("请输入分数:"))

if 90 <= score <= 100 : #注意冒号

print('A') #python靠缩进来判断

elif 80 <= score < 90 : #注意冒号

print('B')

elif 60 <= score < 80 : #注意冒号

print('C')

elif 0 <= score < 60 : #注意冒号

print('D')

else : #注意冒号

print("输入错误")

没有分号，没有分号，没有分号

if 、else 、for 、while等后面有冒号，有冒号，有冒号

新增了elif ，就相当于else if

Python 不用｛｝来区分作用域，靠的是缩进，缩进，缩进

1. 除法 1/2 等于0.5 1//2 等于0

幂方运算 2\*\*5 等于32

1. for(i=0;i<4;i++)

{

......

}

在Python里为

for i in range(0,4) :

......

虽然Python是用c语言写的，但是不同之处还是有很多的，Python更注重实用.....

1. **数据类型**
2. 数字
3. 字符串
4. 元组
5. 列表
6. 字典

字符串、列表、元组都为序列类型的数据,序列就可以进行切片操作

1. **数字**

添加了复数类型，j就相当于i

|  |
| --- |
| 1. >>> c=3.14 2. >>> type(c) 3. <class 'float'> 4. >>> d=3.14j 5. >>> type(d) 6. <class 'complex'> 7. >>> c+d 8. (3.14+3.14j) |

没有double,只有float

**2.字符串**

（1）三重引号

|  |
| --- |
| 1. >>> str1='''tom: 2. hello world! ''' 3. >>> str1 4. 'tom:\n hello world! ' 5. >>> print (str1) 6. tom: 7. hello world! |

三重引号也可以用来多行注释

1. 索引

|  |
| --- |
| 1. >>> str1="abcde" 2. >>> str1[0] 3. 'a' 4. >>> str1[4] 5. 'e' 6. >>> str1[5] 7. Traceback (most recent call last): 8. File "<pyshell#3>", line 1, in <module> 9. str1[5] 10. IndexError: string index out of range 11. >>> str1[-1] #-1表示倒数第一个 12. 'e' 13. >>> str1[-2] 14. 'd' 15. >>> str1[0]+str1[-1] 16. 'ae' |

（3）切片操作

|  |
| --- |
| 1. >>> str1[1:4] #序列类型均有的切片操作   'bcd' |

1. 相关方法

|  |  |
| --- | --- |
| 操作 | 含义 |
| + | 连接 |
| \* | 重复 |
| <string>[] | 索引 |
| <string>[ : ] | 切片 |
| len(<string>) | 求长度 |
| <string>.upper() | 将字符串中所有字母大写(但原字符串没有改变) |
| <string>.lower() | 将字符串中所有字母小写 |
| <string>.strip() | 去两边空格及去指定字符 |
| <string>.split() | 按指定字符分割字符串为数组 |
| <string>.join() | 连接两个字符串序列 |
| <string>.find() | 搜索指定字符串 |
| <string>.replace() | 字符串替换 |
| for <var> in <string> | 字符串迭代 |

1. **元组（tuple）**

|  |
| --- |
| >>> t=123,  >>> type(t)  <class 'tuple'>  >>> t=123,456,'abc' #t=（123,456,'abc'）括号可有可无  >>> type(t)  <class 'tuple'>  >>> t[2]  'abc'  >>> tl=789,234,t #一个元组也可以作为另一个元组的一个元素  >>> tl[1]  234  >>> tl[2]  (123, 456, 'abc')  >>> tu=('a',345,(3.14,'b')) #元组作为另一个元组的元素时要加括号避免歧义  >>> tu[2]  (3.14, 'b')  >>> tu[1:]  (345, (3.14, 'b')) #元组属于序列的一种同样有切片操作  >>> tu\*2  ('a', 345, (3.14, 'b'), 'a', 345, (3.14, 'b'))  >>> tu  ('a', 345, (3.14, 'b')) |

元组定义后不能更改，也不能删除

与字符串一样，元组之间可以使用+号和\*进行运算

**4.列表（list）**

列表与元组类似，但是列表可以随时修改

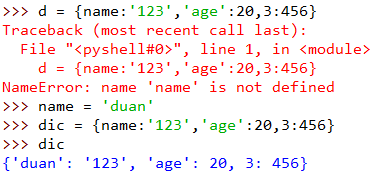
（1）列表相关方法

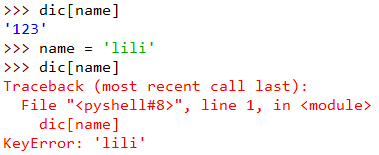
|  |  |
| --- | --- |
| 操作 | 含义 |
| <list>.append(x) | 将元素增加到列表的最后 |
| <list>.sort() | 将列表元素排序 |
| <list>.reverse() | 将列表元素反转 |
| <list>.index(x) | 返回第一次出现元素x的索引值 |
| <list>.insert(i,x) | 在位置i处插入新元素x |
| <list>.count(x) | 返回元素x在列表中的数量 |
| <list>.remove(x) | 删除列表中第一次出现的元素x |
| <list>.pop(i) | 取出列表中位置i的元素，并删除它 |

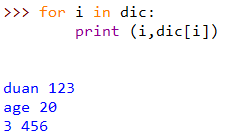
1. **字典 (dict)**

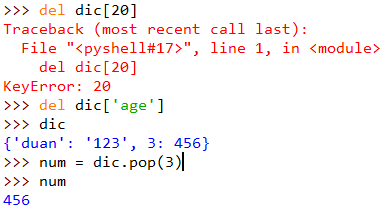
dict **= {**key1:value1,key2:value2.......}

key应为一个常量



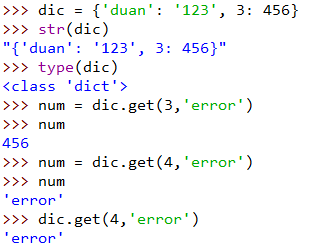






11

12



1. **函数**

基本形式

|  |
| --- |
| 1. def add(x, y): 2. """Add two numbers""" 3. a = x + y 4. return a |

因为python不用定义数据类型，所以函数最开始也不用写返回值类型，直接以def开头

与c++的不同之处：

1. 可以用关键词传参数

|  |
| --- |
| **def fun(**x**=**0, y**=**1, z**=**2**):** print**(**x, y, z**)** fun**(**y**=**3**)**  0 3 2  Process finished with exit code 0 |

2.传入元组

|  |
| --- |
| **def fun(**x**=**0, y**=**1, z**=**2**):** print**(**x, y, z**)** t **= (**4, 5, 6**)** fun**(\***t**)**  4 5 6  Process finished with exit code 0 |

注意星号不能少，否则报错，且只有一个星号

2.传入字典

|  |
| --- |
| **def fun(**x**=**0, y**=**1, z**=**2**):** print**(**x, y, z**)** d **= {'y':** 7, **'z':** 8**}** fun**(\*\***d**)**  0 7 8  Process finished with exit code 0 |

传字典需两个星号，且注意为‘y’，‘z’不能是y，z，因为之前说过key只能是常量，不能是变量

1. 返回多个值

|  |
| --- |
| **def fun(**x, y**):  return** y, x  num1 **=** 3 num2 **=** 5 num1, num2 **=** fun**(**num1, num2**)** #实现交换 print**(**num1, num2**)**  5 3  Process finished with exit code 0 |

1. 接受不定参数

|  |
| --- |
| **def fun(**x, **\***t**):** print**(**x**)** print**(**t**)** fun**(**1, 2**)** fun**(**3, 4, 5, 6**)**  1  (2,)  3  (4, 5, 6)  Process finished with exit code 0 |

多余参数相当于存放在元组t中

|  |
| --- |
| **def fun(**x, **\*\***d**):** print**(**x**)** print**(**d**)** fun**(**1, a**=**2, b**=**3**)**  1  {'a': 2, 'b': 3}  Process finished with exit code 0 |

以关键词形式传入参数并存放在字典中

|  |
| --- |
| **def fun(**x, **\***args, **\*\***kwargs**):** print**(**x, args, kwargs**)** fun**(**1, a**=**2, b**=**3, 4, 5**)**  File "F:/Python/fun", line 4  fun(1, a=2, b=3, 4, 5) ^  SyntaxError: positional argument follows keyword argument  Process finished with exit code 1 |

传入数据顺序应先按顺序赋值，多余的放在元组中，最后才能用关键字的形式传值，顺序不能乱。

|  |
| --- |
| **def fun(**x, **\***args, **\*\***kwargs**):** print**(**x, args, kwargs**)** fun**(**1, 4, 5, a**=**2, b**=**3, x**=**8**)**  Traceback (most recent call last):  File "F:/Python/fun", line 4, in <module>  fun(1, 4, 5, a=2, b=3, x=8)  TypeError: fun() got multiple values for argument 'x'  Process finished with exit code 1 |

注意以关键字传参时不要和顺序传参的参数名重复

1. lambda函数（匿名函数）

|  |
| --- |
| f **= lambda** x**:** x**\*\***3 # 相当于def f(x):  # return x\*\*3  print**(**f**(**2**))**  8  Process finished with exit code 0 |

1. **模块和包**
2. 模块（module）

任何一个.py文件都可以作为一个模块，并使用import加载使用它（有点相当于c++中的#include一个文件，include之后，这个文件里的函数以及数据成员都可以使用了），python在寻找import文件时先从当前工作目录下寻找，没有的话再在安装python时自带的许多.py文件中寻找，所以这也要求我们给文件命名时最好不要和自带的文件重名（如random等），否则会导致用不了官方提供的模块。

#module1.py

|  |
| --- |
| **def Sum(**lst**):** #列表所有元素求和  sum **=** 0  **for** i **in** lst**:** sum **=** sum **+** i  **return** sum   **def isPrime(**num**):  if** num **==** 1**:** print**("该数不是素数")  else:  for** i **in** range**(**2, num**):  if** num **%** i **==** 0**:** print**("该数不是素数")  break  else:** # 当循环被 break 结束时, else 不执行;当循环正常结束时,即没有被break, else 被执行  print**("该数是素数")  if** \_\_name\_\_ **== '\_\_main\_\_':** isPrime**(**3**)**  该数是素数  Process finished with exit code 0 |

注意name属性，if \_\_name\_\_ == '\_\_main\_\_': 这一句代表只有当这个文件当作主文件被执行时才会执行isPrime(3)

使用module1（方法一）

|  |
| --- |
| **import** module1 lst1 **= [**1, 2, 3, 4**]** module1.Sum**(**lst1**)**  10  Process finished with exit code 0 |

注意使用时要用module1.Sum(lst1)，不能直接Sum（lst1），并且注意这里并没有执行isPrime(3)，因为这里是将module导入作为模块使用，并不是主文件

使用module1（方法二）

|  |
| --- |
| **from** module1 **import** Sum lst1 **= [**1, 2, 3, 4**]** Sum**(**lst1**)** module1.isPrime**(**4**)**  10  Traceback (most recent call last):  File "F:/Python/usemodule1.py", line 4, in <module>  module1.isPrime(4)  NameError: name 'module1' is not defined  Process finished with exit code 1 |

注意这里from module1 import Sum，相当于只导入了module1中的Sum函数，就算写module1.isPrime(4)还是会报错，直接写isPrime(4)照样会报错

|  |
| --- |
| **from** module1 **import \*** lst1 **= [**1, 2, 3, 4**]** Sum**(**lst1**)** isPrime**(**4**)**  10  该数不是素数  Process finished with exit code 0 |

这样写是将module1中的所有函数都导入了，且不用加module1. ，

|  |
| --- |
| **from** module1 **import \*  def Sum(**x**):** print**(**x**)** lst1 **= [**1, 2, 3, 4**]** Sum**(**lst1**)** isPrime**(**4**)**  [1, 2, 3, 4]  该数不是素数  Process finished with exit code 0 |

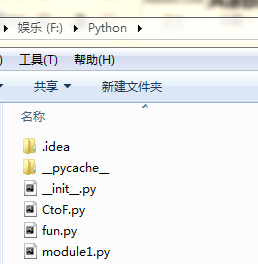
但如果和module1中的函数重名的话，module1中的函数会被覆盖

使用module1（方法三）

|  |
| --- |
| **import** module1 **as** m  lst1 **= [**1, 2, 3, 4**]** m.Sum**(**lst1**)** m.isPrime**(**4**)**  10  该数不是素数  Process finished with exit code 0 |

1. 包

创建一个文件夹，在该文件夹下包含一个\_\_init\_\_.py文件（这个文件内容可以为空），那么这个文件夹就为一个包了，该文件夹的名字即为包的名字。



所以该文件夹就为一个包了，文件夹下的所有.py文件都可以作为一个模块导入，不仅仅局限于module1

但当我们用时不能在该文件夹下创建.py文件再import Pyhton.module1 这样会报错，必须在这个文件夹**平级**的目录下才行

|  |
| --- |
| **import** Python.module1  lst1 **= [**2, 3, 4, 5**]** Python.module1.Sum**(**lst1**)** Python.module1.isPrime**(**5**)**  14  该数是素数  Process finished with exit code 0 |

注意还是要加Python.module1. ，不然会报错

1. **正则表达式**

正则表达式是由字符和操作符构成

4.1 正则表达式常用操作符

|  |  |  |
| --- | --- | --- |
| 操作符 | 说明 | 实例 |
| . | 表示任何单个字符（除换行符外） |  |
| [ ] | 字符集，对单个字符给出取值范围 | [abc]表示a、b、c，[a-z]表示a到z单个字符 |
| [^ ] | 非字符集，对单个字符给出排除范围 | [^abc]表示非a或b或c的单个字符 |
| \* | 前一个字符0次或无限次扩展 | abc\*表示ab、abc、abcc、abccc等 |
| + | 前一个字符1次或无限次扩展 | abc+表示abc、abcc、abccc等 |
| ? | 前一个字符0次或1次扩展 | abc?表示ab、abc |
| | | 左右表达式任意一个 | abc|def表示abc、def |
| {m} | 扩展前一个字符m次 | ab{2}表示abbc |
| {m,n} | 扩展前一个字符m至n次 | ab{1,2}表示abc、abbc |
| {:n} | 扩展前一个字符0至n次 | a{:3}b表示b、ab、aab、aaab |
| ^ | 匹配字符串开头 | ^abc表示abc且在一个字符串的开头 |
| $ | 匹配字符串结尾 | abc$表示abc且在一个字符串的结尾 |
| () | 分组标记，内部只能使用|操作符 | (abc)表示abc，(abc|def)表示abc、def |
| \d | 数字，等价于[0-9] |  |
| \w | 单词字符，等价于[A-Za-z0-9] |  |

4.2 经典正则表达式实例

|  |  |
| --- | --- |
| ^[A-Za-z]+$ | 表示26个字母组成的字符串（不含数字或其他） |
| ^[A-Za-z0-9]+$ | 表示26个字母和数字组成的字符串 |
| ^-?\d+$ | 表示整数形式的字符串 |
| ^[0-9]\*[1-9][0-9]\*$ | 表示正整数形式的字符串（排除了00......0） |
| [1-9]\d{5} | 表示中国境内邮政编码形式，6位（首位不能为0） |
| [\u4e00-\u9fa5] | 匹配中文字符（utf-8编码） |
| \d{3}-\d{8}|\d{4}-\d{7} | 表示国内电话号码形式，如010-12345678 |
| [1-9]?\d | 表示0-99 |
| 1\d{2} | 表示100-199 |

ip地址如何表示？（ip地址分4段，每段0-255）

(([1-9]?\d|1\d{2}|2[0-4]\d|25[0-5]).){3}([1-9]?\d|1\d{2}|2[0-4]\d|25[0-5])

4.3 re库主要功能函数

|  |  |
| --- | --- |
| 函数 | 说明 |
| re.search() | 在一个字符串中搜索匹配正则表达式的第一个位置，返回match对象 |
| re.match() | 从一个字符串的开始位置起匹配正则表达式，返回match对象 |
| re.findall() | 搜索字符串以列表类型返回全部能匹配的子串 |
| re.split() | 将一个字符串按照正则表达式匹配结果进行分割，返回列表类型 |
| re.finditer() | 搜索字符串，返回一个匹配结果的迭代类型，每个迭代元素是match对象 |
| re.sub() | 在一个字符串中替换所有匹配正则表达式的子串，返回替换后的字符串 |
| re.compile() | 将正则表达式的字符串形式编译成正则表达式对象 |

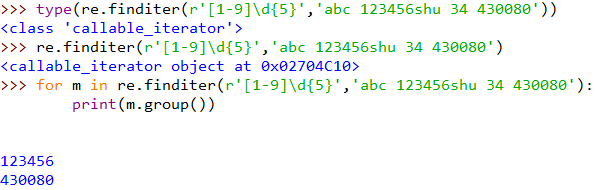
|  |
| --- |
| **import** re r **= r'\d{3}-\d{8}'** match **=** re.search**(**r, **'123,010-12345678,027-12345678,')** print**(**type**(**match**)) if** match**:** print**(**match.group**()) else:** print**("not match")**   1. <class '\_sre.SRE\_Match'> 2. 010-12345678 3. Process finished with exit code 0 |

从给定字符串中寻找第一个匹配正则表达式的位置，返回match对象，用match.group()取得数据

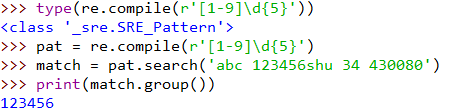
|  |
| --- |
| **import** re r **= r'\d{3}-\d{8}'** match **=** re.match**(**r, **'123,010-12345678,027-12345678,')** print**(**type**(**match**)) if** match**:** print**(**match.group**()) else:** print**("not match")**   1. <class 'NoneType'> 2. not match 3. Process finished with exit code 0 |

re.match()从字符串的第一个位置开始匹配（就是从数字1开始），因为之后的不匹配，所以返回的是NoneType

1



3

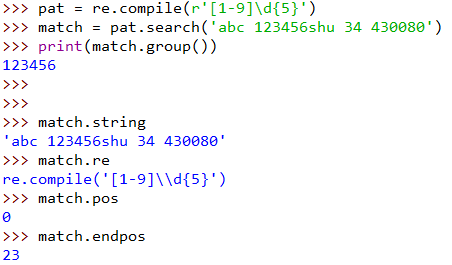


re.complie(r’[1-9]\d{5}’)是将里面的字符串编译成一个正则表达式对象（即Pattern对象），然后这个Pattern对象可以调用上述的6个函数

4.4 Match对象

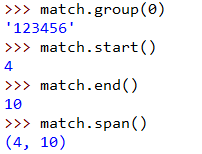
4.4.1 Match对象的属性

|  |  |
| --- | --- |
| 属性 | 说明 |
| .string | 待匹配的文本 |
| .re | 匹配时使用的patter对象（正则表达式） |
| .pos | 正则表达式搜索文本的开始位置 |
| .endpos | 正则表达式搜索文本的结束位置 |



4.4.2 Match对象的方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| .group(0) | 获得匹配后的字符串 |
| .start() | 匹配字符串在原是字符串的开始位置 |
| .end() | 匹配字符串在原是字符串的结束位置 |
| .span() | 返回(.start(),.span()) |



4.5 match的贪婪匹配和最小匹配

7

按正则表达式来看‘PYAN’、‘PYANBN’、‘PYANBNCN’、‘PYANBNCNDN’都可以，但是输出的是‘PYANBNCNDN’，说明re默认的是贪婪匹配，也就是会输出最长的那个。

那么如何实现最小匹配呢？

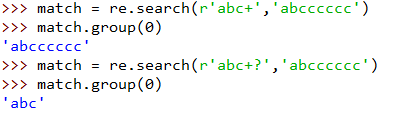
可以在加上?

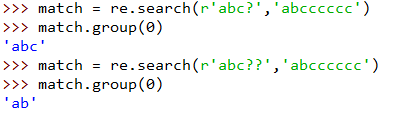
8

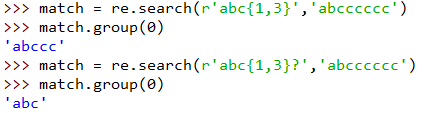
最小匹配操作符

|  |  |
| --- | --- |
| 操作符 | 说明 |
| \*? | 前一个字符0次或无限多次扩展的最小匹配 |
| +? | 前一个字符1次或无限多次扩展的最小匹配 |
| ?? | 前一个字符0次或1次扩展的最小匹配 |
| {m,n}? | 前一个字符m次或n次扩展的最小匹配 |

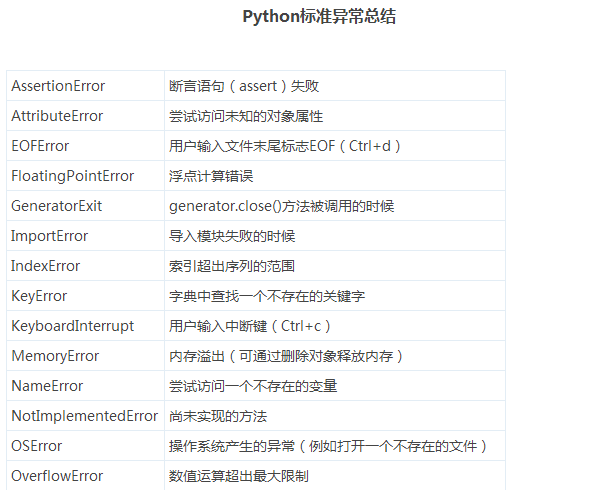
只要长度输出可能不同的，都可以通过在操作符后添加？变成最小匹配



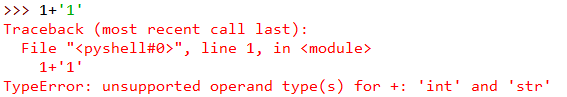


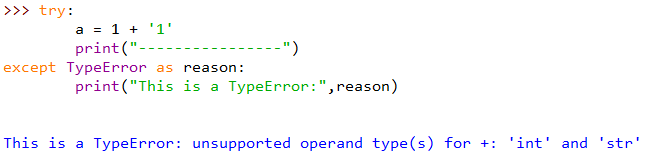


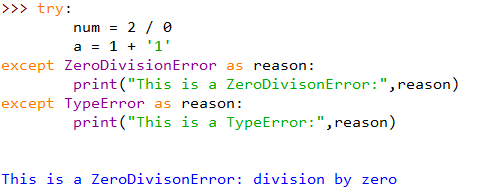
1. 异常处理

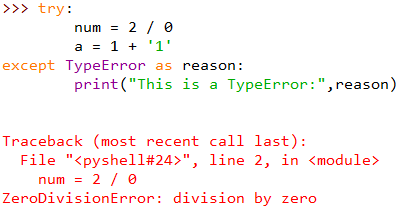


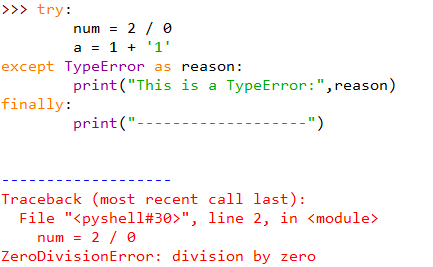


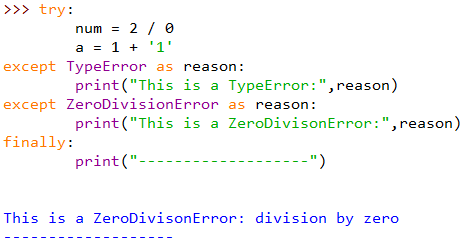


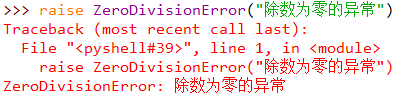


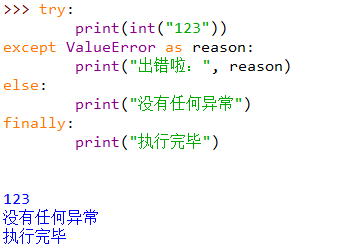


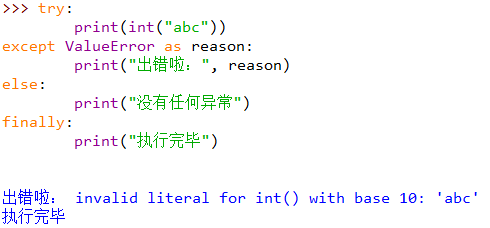




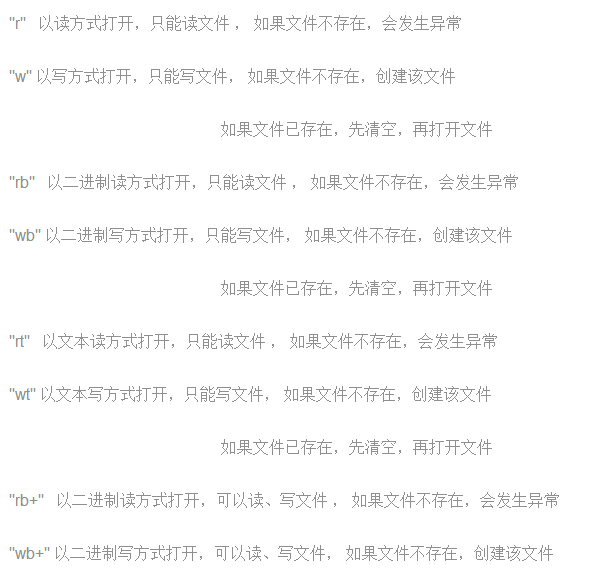






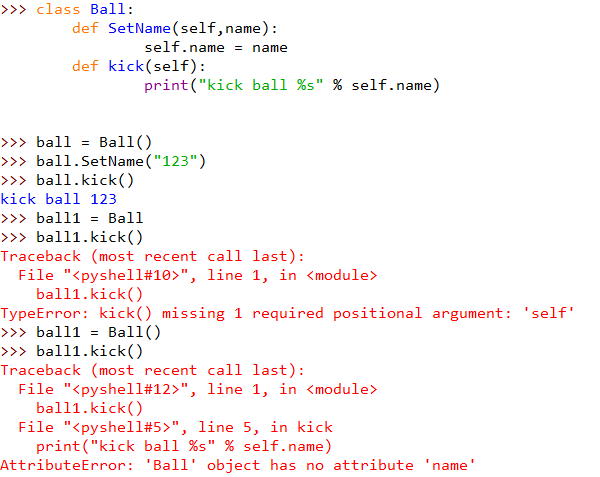


1. 文件操作

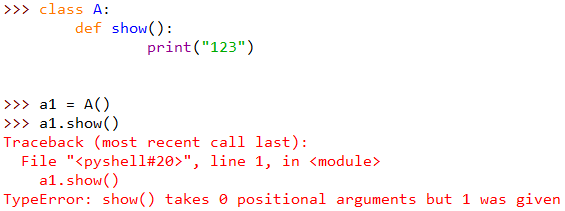


七．面向对象编程

1. 类的定义

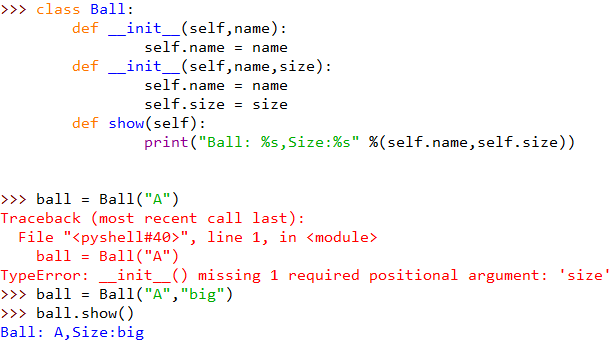


注意：1.定义class里面的函数时，第一个参数一定为self（就相当于c++中的this指针）不加不行，如下



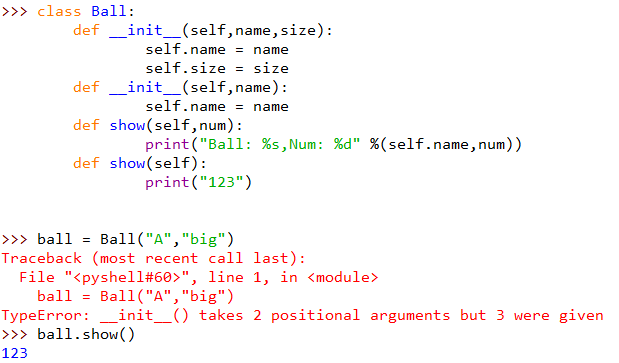
1. 实例化一个对象时，一定要a1 = A()，括号不能掉，但是定义的时候class A：这里是没有括号的

-----------------------------------------------------------------------



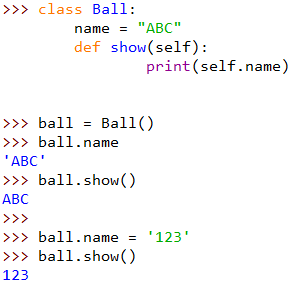
注意：1.当一个对象被创建时，类定义里的左右双下划线的函数是被自动调用的，如\_\_init\_\_就相当于c++中的构造函数

1. 从以上例子可以看出，python 不支持函数重载：



他其实是后面的覆盖前面的，并且我们也不一定要用c++的思维来看待这个问题，python支持键值对的形式赋值，当我们给定默认参数时，可用键值对的形式赋值，也就没有函数重载的必要了

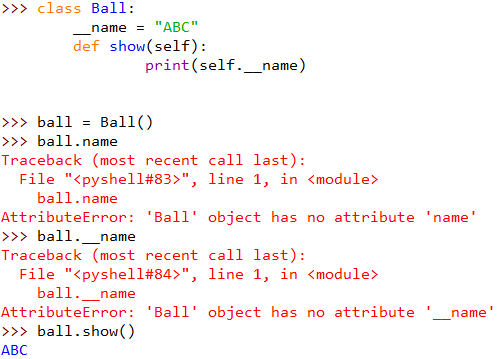
-----------------------------------------------------------------------



ball.name可以直接访问或修改Ball的数据成员name，说明name是共有的，但是python并没有public和private关键字，那么python如何区分公有和私有呢？

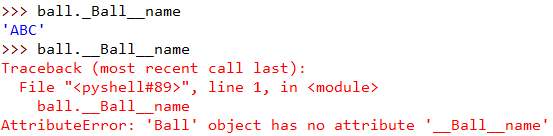
默认的数据成员和函数均为共有的

但当数据成员和函数的名称前有两个下划线，就变为私有的了

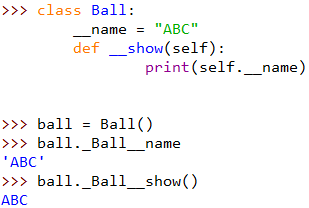


注意不要写成self.name,要与定义的名字一致，为\_\_name

Python使用“名字重整”方法，他其实是把前面带有两个下划线的成员名字进行了改动



Python把前面带有两个下划线的成员名字改成了“\_类名\_\_变量名”的形式（前面为一个下划线），函数也是如此，如下：



八．应用小程序

（1）温度转换

|  |
| --- |
| 1. temperature = input("请输入温度及单位：") 2. if temperature[-1] == ('c' or 'C') : #注意括号一定要打，比较操作符比逻辑操作符优先级高 3. tran = float(temperature[0:-1]) \* 1.8 +32 4. print ("转换成华氏温度为：",tran, 'F') 5. elif temperature[-1] in （'f','F'） : #上面的和这种写法都可以 6. tran = ( float(temperature[0:-1]) - 32 ) / 1.8 7. print ("转换成摄氏温度为:%.2fC"%tran) #注意这里 8. else : 9. print ("输入有误") |

（2）猴子吃桃问题

|  |
| --- |
| 1. num = 1 2. for i in range (1,6) : 3. num = (num+1)\*2 4. print(num) 5. n = 1 6. for i in range (5,0,-1) : 7. n = (n+1)<<1 8. print(n) |

九．附录

1. Python 3.x 保留字列表（34个）

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

1. 字符串处理方法