**相关知识**

* abs(x)：返回整数的绝对值，如abs(-10)返回10。
* ceil(x)：返回数字的向上取整，如math.ceil(4.1)返回5。
* exp(x)：返回e的x次幂，如math.exp(1)返回2.718281828459045。
* fabs(x)：返回浮点数的绝对值，如math.fabs(-10) 返回10.0。
* floor(x)：返回数字的向下取整，如math.floor(4.9)返回4。
* log(x,base)：如math.log(math.e,math.e)返回1.0，math.log(100,10)返回2.0。
* log10(x)：返回以10为基数的x的对数，如math.log10(100)返回2.0。
* max(x1,x2,...)：返回给定参数的最大值，参数可以为序列。
* min(x1,x2,...)：返回给定参数的最小值，参数可以为序列。
* modf(x)：以元组的形式返回，（小数部分,整数部分）。两部分的数值符号与x相同，整数部分以浮点型表示。
* pow(x, y)：*xy* 运算后的值。
* round(x [,n])：返回浮点数x的四舍五入值，如给出n值，则代表舍入到小数点后的位数。
* sqrt(x)：返回数字x的平方根，返回类型为实数，如math.sqrt(4)返回2.0。
* acos(x)：返回x的反余弦弧度值。
* asin(x)：返回x的反正弦弧度值。
* atan(x)：返回x的反正切弧度值。
* atan2(y, x)：返回给定的X及Y坐标值的反正切值。
* cos(x)：返回x的弧度的余弦值。
* hypot(x, y)：返回欧几里德范数*sqrt*(*x*2+*y*2)。
* sin(x)：返回x弧度的正弦值。
* tan(x)：返回x弧度的正切值。
* degrees(x)：将弧度转换为角度，如degrees(math.pi/2) ， 返回90.0。
* radians(x)：将角度转换为弧度
* 除了上述常用的数学函数，math库中还定义了两个常用的数学常量：
* pi——圆周率，一般以π来表示。
* e——自然常数。

除了数字，Python 也可以操作字符串。字符串有多种形式，可以使用单引号（'……'），双引号（"……"）都可以获得同样的结果 2。反斜杠 \ 可以用来转义。  
字符串可以用 + 进行连接（粘到一起），也可以用 \* 进行重复:

1. >>> 3 \* 'un' + 'ium'
2. 'unununium'

字符串是可以被 索引 （下标访问）的，第一个字符索引是 0。索引也可以用负数，这种会从右边开始数。

1. >>> word = 'Python'
2. >>> word[0]
3. 'P'
4. >>> word[5]
5. 'n'
6. >>> word[-1]
7. 'n'
8. >>> word[-2]
9. 'o'
10. >>> word[-6]
11. 'P'

除了索引，字符串还支持 切片。索引可以得到单个字符，而 切片 可以获取子字符串:

1. >>> word[0:2]
2. 'Py'
3. >>> word[2:5]
4. 'tho'

切片的索引有默认值；省略开始索引时默认为0，省略结束索引时默认为到字符串的结束:

1. >>> word[:2]
2. 'Py'
3. >>> word[4:]
4. 'on'
5. >>> word[-2:]
6. 'on'

内建函数 len() 返回一个字符串的长度:

1. >>> s = 'supercalifragilisticexpialidocious'
2. >>> len(s)
3. 34

字符串实现了所有 一般 序列的操作，还额外提供了以下列出的一些附加方法。

字符串还支持两种字符串格式化样式，一种提供了很大程度的灵活性和可定制性 (参阅 str.format(), 格式字符串语法 和 自定义字符串格式化) 而另一种是基于 C printf 样式的格式化，它可处理的类型范围较窄，并且更难以正确使用，但对于它可处理的情况往往会更为快速 (printf 风格的字符串格式化)。

str.count(sub[s])  
反回子字符串 sub 在 [start, end] 范围内非重叠出现的次数。 可选参数 start 与 end 会被解读为切片表示法。

str.find(sub[, start[, end]])  
返回子字符串 sub 在 s[start:end] 切片内被找到的最小索引。 可选参数 start 与 end 会被解读为切片表示法。 如果 sub 未被找到则返回 -1。  
注解 find() 方法应该只在你需要知道 sub 所在位置时使用。 要检查 sub 是否为子字符串，请使用 in 操作符:

1. >>> 'Py' in 'Python'
2. True

str.index(sub[, start[, end]])  
类似于 find()，但在找不到子类时会引发 ValueError。

str.format(\*args, \*\*kwargs)  
执行字符串格式化操作。 调用此方法的字符串可以包含字符串字面值或者以花括号 {} 括起来的替换域。 每个替换域可以包含一个位置参数的数字索引，或者一个关键字参数的名称。 返回的字符串副本中每个替换域都会被替换为对应参数的字符串值。

1. >>> "The sum of 1 + 2 is {0}".format(1+2)
2. 'The sum of 1 + 2 is 3'

str.lower()  
返回原字符串的副本，其所有区分大小写的字符均转换为小写。

str.replace(old, new[, count])  
返回字符串的副本，其中出现的所有子字符串 old 都将被替换为 new。 如果给出了可选参数 count，则只替换前 count 次出现。

str.split(sep=None, maxsplit=-1)  
返回一个由字符串内单词组成的列表，使用 sep 作为分隔字符串。 如果给出了 maxsplit，则最多进行 maxsplit 次拆分（因此，列表最多会有 maxsplit+1 个元素）。 如果 maxsplit 未指定或为 -1，则不限制拆分次数（进行所有可能的拆分）。

如果给出了 sep，则连续的分隔符不会被组合在一起而是被视为分隔空字符串 (例如 '1,,2'.split(',') 将返回 ['1', '', '2'])。 sep 参数可能由多个字符组成 (例如 '1<>2<>3'.split('<>') 将返回 ['1', '2', '3'])。 使用指定的分隔符拆分空字符串将返回 ['']。

例如:

1. >>> '1,2,3'.split(',')
2. ['1', '2', '3']
3. >>> '1,2,3'.split(',', maxsplit=1)
4. ['1', '2,3']
5. >>> '1,2,,3,'.split(',')
6. ['1', '2', '', '3', '']

如果 sep 未指定或为 None，则会应用另一种拆分算法：连续的空格会被视为单个分隔符，其结果将不包含开头或末尾的空字符串，如果字符串包含前缀或后缀空格的话。 因此，使用 None 拆分空字符串或仅包含空格的字符串将返回 []。

例如:

1. >>> '1 2 3'.split()
2. ['1', '2', '3']
3. >>> '1 2 3'.split(maxsplit=1)
4. ['1', '2 3']
5. >>> ' 1 2 3 '.split()
6. ['1', '2', '3']

str.strip()  
返回原字符串的副本，移除其中的前导和末尾空格字符。

1. >>> ' spacious '.strip()
2. 'spacious'

str.upper()  
返回原字符串的副本，其中所有区分大小写的字符均转换为大写。

tr.title()  
返回原字符串的标题版本，其中每个单词第一个字母为大写，其余字母为小写。

例如:

1. >>> 'Hello world'.title()
2. 'Hello World'

str.isnumeric()  
如果字符串中至少有一个字符且所有字符均为数值字符则返回 True ，否则返回 False 。

str.islower()  
如果字符串中至少有一个区分大小写的字符 4 且此类字符均为小写则返回 True ，否则返回 False 。

str.isalpha()  
如果字符串中的所有字符都是字母，并且至少有一个字符，返回 True ，否则返回 False 。

Python 中可以通过组合一些值得到多种 复合 数据类型。其中最常用的 列表 ，可以通过方括号括起、逗号分隔的一组值得到。一个 列表 可以包含不同类型的元素，但通常使用时各个元素类型相同。  
字符串（以及各种内置的 sequence 类型）一样，列表也支持索引和切片:

1. >>> squares = [1, 4, 9, 16, 25]
2. >>> squares[0] # indexing returns the item
3. 1
4. >>> squares[-1]
5. 25
6. >>> squares[-3:] # slicing returns a new list
7. [9, 16, 25]
8. >>> squares[:]
9. [1, 4, 9, 16, 25]

列表同样支持拼接操作:

1. >>> squares + [36, 49, 64, 81, 100]
2. [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

你也可以在列表结尾，通过 append() 方法 添加新元素：

1. >>> squares.append(200)
2. >>> squares
3. [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 200]

内置函数 len() 也可以作用到列表上:

1. >>> letters = ['a', 'b', 'c', 'd']
2. >>> len(letters)
3. 4

也可以嵌套列表 (创建包含其他列表的列表), 比如说:

1. >>> a = ['a', 'b', 'c']
2. >>> n = [1, 2, 3]
3. >>> x = [a, n]
4. >>> x
5. [['a', 'b', 'c'], [1, 2, 3]]
6. >>> x[0]
7. ['a', 'b', 'c']
8. >>> x[0][1]
9. 'b'

列表数据类型还有很多的方法。这里是列表对象方法的清单：  
list.append(x)  
在列表的末尾添加一个元素。相当于 a[len(a):] = [x] 。

list.extend(iterable)  
使用可迭代对象中的所有元素来扩展列表。相当于 a[len(a):] = iterable 。

list.insert(i, x)  
在给定的位置插入一个元素。第一个参数是要插入的元素的索引，所以 a.insert(0, x) 插入列表头部， a.insert(len(a), x) 等同于 a.append(x) 。

list.remove(x)  
移除列表中第一个值为 x 的元素。如果没有这样的元素，则抛出 ValueError 异常。

list.pop([i])  
删除列表中给定位置的元素并返回它。如果没有给定位置，a.pop() 将会删除并返回列表中的最后一个元素。（ 方法签名中 i 两边的方括号表示这个参数是可选的，而不是要你输入方括号。你会在 Python 参考库中经常看到这种表示方法)。

list.clear()  
删除列表中所有的元素。相当于 del a[:] 。

list.index(x[, start[, end]])  
返回列表中第一个值为 x 的元素的从零开始的索引。如果没有这样的元素将会抛出 ValueError 异常。

可选参数 start 和 end 是切片符号，用于将搜索限制为列表的特定子序列。返回的索引是相对于整个序列的开始计算的，而不是 start 参数。

list.count(x)  
返回元素 x 在列表中出现的次数。

list.sort(key=None, reverse=False)  
对列表中的元素进行排序（参数可用于自定义排序，解释请参见 sorted()）。

list.reverse()  
反转列表中的元素

#### 访问字典中的值

1. cars = {'BMW': 8.5, 'BENS': 8.3, 'AUDI': 7.9}
2. print(cars['AUDI'])

#### 添加键值对

1. cars = {'BMW': 8.5, 'BENS': 8.3, 'AUDI': 7.9}
2. cars['JLR']=8.0
3. print(cars)

#### 修改字典中的值

1. cars = {'BMW': 8.5, 'BENS': 8.3, 'AUDI': 7.9,'JLR':8.0}
2. cars['JLR']=7.0
3. print(cars)

#### 删除键值对

1. cars = {'BMW': 8.5, 'BENS': 8.3, 'AUDI': 7.9,'JLR':8.0}
2. del cars['JLR']
3. print(cars)

#### 判断字符串是否由数字组成

1. str=input()
2. if str.isdigit():
3. str=eval(str)

这样可以确保输入的成绩会以数值形式保存。

#### 从列表创建字典

python 字典 fromkeys() 函数用于创建一个新字典，以序列 seq 中元素做字典的键，value 为字典所有键对应的初始值：  
dict.fromkeys(seq[, value])  
参数  
seq -- 字典键值列表。  
value -- 可选参数, 设置键序列（seq）的值。

#### 遍历数组

遍历字典中的键:

1. cars = {'BMW': 8.5, 'BENS': 8.3, 'AUDI': 7.9}
2. for key in cars.keys():
3. print(key)

遍历字典中的值:

1. cars = {'BMW': 8.5, 'BENS': 8.3, 'AUDI': 7.9}
2. for value in cars.values():
3. print(value)

遍历字典中的键值对:

1. cars = {'BMW': 8.5, 'BENS': 8.3, 'AUDI': 7.9}
2. for key,value in cars.items():
3. print(key,value)

#### 列表嵌套字典

1. employee\_dict1 = {'name':'david', 'dept':'ops', 'salary': 12000}
2. employee\_dict2 = {'name':'brain' , 'dept': 'auto' , 'salary': 13000}
3. employee\_list=[employee\_dict1,employee\_dict2]

#### 字典嵌套列表

1. employee\_dict = {'name':['david','brain'] , 'dept':['ops', 'auto'] , 'salary': [12000,13000]}

#### 定义函数

def <函数名> (<参数列表>):  
<函数体>  
return <返回值列表>