【Python】常用函数

一、基本输入输出

```
# input():将所有输入都当作字符串,需要类型转换

num_int = int(input('Please input an int'))
num_float = float(input('Please input a float'))
num_complex = complex(input('Please input a complex'))
container_list = eval(input('Please input a list'))
container_tuple = eval(input('Please input a tuple'))
container_dic = eval(input('Please input a dic'))
container_set = eval(input('Please input a set'))

print(num_int)
print(num_float)
print(num_complex)
print(container_list)
print(container_tuple)
print(container_dic)
print(container_set)
```

```
# print()

num_int = 233
num_float = 2.6985
num_complex = 2+6j

print(num_int, num_float)                           # 默认空格分隔,回车结尾
print(num_int, num_float, num_complex, sep=',')     # 逗号分隔
print(num_int, num_float, num_complex, end='end')   # end结尾
print(num_int, num_float)
```

二、数学函数

数学库(math)

函数或常数	功能
e	自然常数
pi	圆周率
log(x[,base])	返回以base为底的对数，缺省为e
pow(x,y)	返回x的y次方
sqrt(x)	返回x的平方根
fabs(x)	返回x的绝对值
round(x[,n])	返回浮点数x的四舍五入值，n代表舍入到小数点后的位数
divmod(x,y)	返回x和y的商和余数

运算符的优先级和结合性

优先级	运算符	描述	结合性
1	+x,-x	正，负	
2	x**y	幂	从右向左
3	x*y, x/y, x%y	乘，除，取模	从左向右
4	x+y, x-y	加，减	从左向右
5	x<y, x<=y, x==y, x!=y, x>=y, x>y	比较	从左向右
6	not x	逻辑否	从左向右
7	x and y	逻辑与	从左向右
8	x or y	逻辑或	从左向右

```python
import math
# 实数比大小
print(math.isclose(0.4-0.3, 0.1))
print(round(0.1+0.2,2) == 0.3)

# 阶乘
print(math.factorial(32))  # 32的阶乘

# 平方根
print(7**0.5)  # 7的平方根

# 复数运算
c = 3+4j
print(c+c)              # 加
print(c**2)             # 幂运算
print(c.real)           # 查看实部
print(c.imag)           # 查看虚部
print(c.conjugate())    # 查看共轭复数
print(abs(c))           # 计算模
```

三、容器操作

```python
# 容器定义
x_list = [1, 2, 3]                      # 列表
x_tuple = (1, 2, 3)                     # 元组
x_dict = {'a': 97, 'b': 98, 'c': 99}    # 字典
x_set = {1, 2, 3}                       # 集合

# 容器访问
print(x_list[0], x_list[-1])            # 列表
print(x_tuple[0], x_tuple[-2])          # 元组
print(x_dict['a'])                      # 字典(不安全)
print(x_dict.get('speed'))              # 字典(安全),返回None
print(x_set)                            # 集合,是无序的，不能索引

# 元素个数
print(len(x_list))
print(len(x_tuple))
print(len(x_dict))
print(len(x_set))
```

```python
# 增加元素
x_list = []
x_tuple = ()
x_dic = {}
x_set = set()

x_list.append('list0')                      # 列表,增加在末尾
x_list.insert(0, 'listx')                   # 列表,增加在任意位置

x_tuple = (200, 50)                         # 元组,定义了就不能修改,除非重新定义

x_dic['key'] = 'value'                      # 字典,添加和修改一样

x_set.add('set0')                           # 集合,无序,无需索引
x_set.update({1, 3})
x_set.update([1, 4], [5, 6])

print("After add:")
print("x_list:", x_list)
print("x_tuple:", x_tuple)
print("x_dic:", x_dic)
print("x_set:", x_set)
```

```python
# 删除元素
x_list = ['list0', 'list1', 'list2', 'list3']
```

```
x_tuple = ('tuple0', 'tuple1', 'tuple2', 'tuple3')
x_dic = {'dic0': 0, 'dic1': 1, 'dic2': 2, 'dic3': 3, 'key': 'value'}
x_set = {'set0', 'set1', 'set2', 'set3', 'set4'}

del x_list[0]                                   # 列表,删除0号位置
x_list.remove('list3')                          # 列表,删除值为list3的元素
popped_element = x_list.pop(1)                  # 列表,弹出1号位置的元素,并赋给变量

# 元组,不可修改

del x_dic['key']                                # 字典,删除名称为key的键值对

x_set.remove('set2')                            # 集合(不安全),删除名为set2的元素
x_set.discard('set2')                           # 集合(安全),不存在不会发生错误
x_set.pop()                                     # 集合(随机删除一个元素)

print("\nAfter delete:")
print("x_list:", x_list)
print("x_tuple:", x_tuple)
print("x_dic:", x_dic)
print("x_set:", x_set)
```

```
# 清空or删除容器
x_list.clear()                                  # 清空列表
del x_list                                      # 删除列表

# 元组不可修改
del x_tuple                                     # 删除元组

x_dic.clear()                                   # 清空字典
del x_dic                                       # 删除字典

x_set.clear()                                   # 清空集合
del x_set                                       # 删除集合
```

四、类型转换


```
# int()

# 实数->整数
print(int(3.5))                 # 获取实数的整数部分

# 字符串->整数
print(int('119'))               # 把整数字符串转换为整数
print(int('  9\n'))             # 自动忽略字符串两个的空白字符

# 进制转换
print(int('1111', 2))           # 把1111看作二进制数，转换为十进制数
print(int('1111', 8))           # 把1111看作八进制数，转换为十进制数
print(int('1111', 16))          # 把1111看作十六进制数，转换为十进制数
```

```
#float()、complex()
```

```

# 字符串->浮点数
print(float('3.1415926'))       # 把字符串转换为实数
print(float('-inf'))            # 负无穷大

# 产生复数
print(complex(3, 4))            # 复数
print(complex(6j))
print(complex('3'))
```

```
# bin()、oct()、hex()

# 进制转换
print(bin(8888))              # 把整数转换为二进制
print(oct(8888))              # 把整数转换为八进制
print(hex(8888))              # 把整数转换为十六进制
```

```
# ord()、chr()、str()

# ASCII or Unicode 字码转换
print(ord('a'))               # 返回字符的ASCII码
print(chr(65))                # 返回指定ASCII码对应的字符
print(ord('董'))              # 返回汉字字符的Unicode编码
print(chr(33891))             # 返回指定Unicode编码对应的汉字

# 容器->字符串
print(str([1, 2, 3, 4]))      # 把列表转换为字符串
print(str({1, 2, 3, 4}))      # 把集合转换为字符串
```

```
# list()、tuple()、dict()、set()

# 集合->列表or元组
s = {3, 2, 1, 4}
print(list(s))
print(tuple(s))

# 列表->元组or集合
lst = [1, 1, 2, 2, 3, 4]
print(tuple(lst))
print(set(lst))  # 自动去重

# 字符串->列表
print(str(lst))  # 包括[],逐字符插入
print(list(str(lst)))
print(tuple(str(lst)))
print(set(str(lst)))

# 键值对->字典
print(dict(name='Dong', sex='Male', age=41))
```

五、字符串操作

【Python】字符串，数，运算符_fftx_00的博客-CSDN博客

六、max(),min()

```
"""
比较单个字符
"""
# 表中最大单个字符
max(data, key=str)
# 表中最大单个字符(小写)
max(data, key=str.lower)
# 表中出现次数最多的元素
max(set(data), key=data.count)
```

```
# 表中最大元素的位置，列表方法__getitem__()用于获取指定位置的值
max(range(len(data)), key=data.__getitem__)
# 与原顺序相同的唯一元素
max(set(data),key=data.index)

"""
比较字符串
"""
# 表中最长字符串
max(data, key=len)

"""
如果在类中定义了__getitem__()方法，那么他的实例对象（假设为P）就可以这样P[key]取值。
"""
"""
所有结果最后拼接成字符串：
''.join()
"""
```

```
data = [111, 22, 556, 8494131, 1354]
max(data, key=str)
max(data, key=lambda x: str(x))

data =['12364654', '131', '12313']
max(data, key=str.lower)
max(data, key=lambda x: x.lower())

"""
lambda用于取容器中的元素，并施加函数
【如取list和string中的元素】
对于整体似乎用不上
"""
```

七、len(),sum()

len(*s*)

返回对象的长度（元素个数）。实参可以是序列（如 string、bytes、tuple、list 或 range 等）或集合（如 dictionary、set 或 frozen set 等）。

CPython implementation detail: `len` 对于大于 `sys.maxsize` 的长度如 `range(2 ** 100)` 会引发 `OverflowError`。

sum(*iterable*, /, *start=0*)

从 *start* 开始自左向右对 *iterable* 的项求和并返回总计值。 *iterable* 的项通常为数字，而 start 值则不允许为字符串。

对某些用例来说，存在 `sum()` 的更好替代。 拼接字符串序列的更好更快方式是调用 `''.join(sequence)`。 要以扩展精度对浮点值求和，请参阅 `math.fsum()`。 要拼接一系列可迭代对象，请考虑使用 `itertools.chain()`。

在 3.8 版更改: *start* 形参可用关键字参数形式来指定。


```
# len():容器中元素的个数
# sum():容器中所有元素之和
data = [1, 2, 3, 4]
print(len(data))
print(sum(data))
data = (1, 2, 3)
print(len(data))
print(sum(data))
data = {1, 2, 3}
```

```
print(len(data))
11
print(sum(data))
data = 'Readability counts.'
print(len(data))                    # 字符串长度
data = {1: 'a', 2: 'b', 3: 'c'}
print(len(data))                    # 键个数
print(sum(data))                    # 所有键相加
```

八、sorted(),reversed(),list.sort(),

sorted(*iterable*, *, *key=None*, *reverse=False*)

根据 *iterable* 中的项返回一个新的已排序列表。

具有两个可选参数，它们都必须指定为关键字参数。

key 指定带有单个参数的函数，用于从 *iterable* 的每个元素中提取用于比较的键 (例如 `key=str.lower`)。 默认值为 `None` (直接比较元素)。

reverse 为一个布尔值。 如果设为 `True`，则每个列表元素将按反向顺序比较进行排序。

使用 `functools.cmp_to_key()` 可将老式的 *cmp* 函数转换为 *key* 函数。

内置的 `sorted()` 确保是稳定的。 如果一个排序确保不会改变比较结果相等的元素的相对顺序就称其为稳定的 --- 这有利于进行多重排序（例如先按部门、再按薪级排序）。

有关排序示例和简要排序教程，请参阅 排序指南 。

reversed(*seq*)

返回一个反向的 iterator。 *seq* 必须是一个具有 `__reversed__()` 方法的对象或者是支持该序列协议（具有从 `0` 开始的整数类型参数的 `__len__()` 方法和 `__getitem__()` 方法）。


```
# sorted():  临时升序排序

from random import shuffle
data = list(range(20))
shuffle(data)                   # 随机打乱顺序
print(data)
print(sorted(data))             # 升序排序

print(sorted(data, key=str))                # 按转换成字符串后的大小升序排序
print(sorted(data, key=str, reverse=True))    # 按转换成字符串后的大小降序排序

# reversed():临时逆转容器顺序
# 返回一个反向的 iterator,只能使用一次,不支持使用内置函数reversed()再次翻转
# 惰性求值,不支持使用内置函数len()计算元素个数

from random import shuffle
data = list(range(20))          # 创建列表
shuffle(data)                   # 随机打乱顺序
print(data)

reversedData = reversed(data)  # 生成反向迭代器
print(list(reversedData))      # 根据反向迭代器遍历得列表
print(tuple(reversedData))     # 空元组
```

list.sort()没有返回值，直接修改了原表

为什么 list.sort() 没有返回排序列表?

在性能很重要的情况下，仅仅为了排序而复制一份列表将是一种浪费。因此， `list.sort()` 对列表进行了适当的排序。为了提醒您这一事实，它不会返回已排序的列表。这样，当您需要排序的副本，但也需要保留未排序的版本时，就不会意外地覆盖列表。

如果要返回新列表，请使用内置 `sorted()` 函数。此函数从提供的可迭代列表中创建新列表，对其进行排序并返回。例如，下面是如何迭代遍历字典并按keys排序:

```
for key in sorted(mydict):
    ...   # do whatever with mydict[key]...
```



```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> b = basket.sort(key=str.lower)
>>> b
>>> basket
['apple', 'apple', 'banana', 'orange', 'orange', 'pear']
```


九、range()

Python3 range() 函数返回的是一个**可迭代对象**
【Python】if语句、for语句_fftx_00的博客-CSDN博客

```
# range(start, stop, step)生成数值列表

range1 = range(4)           # 只指定stop为4，start默认为0，step默认为1
range2 = range(5, 8)        # 指定start=5和stop=8，step默认为1
range3 = range(3, 20, 4)    # 指定start=3、stop=20和step=4
range4 = range(20, 0, -3)   # step也可以是负数
```

十、zip()

zip(** iterables , strict = False* **)**

并行迭代多个可迭代对象，从每个可迭代对象中生成一个元素的元组。

示例:

```
>>> for item in zip([1, 2, 3], ['sugar', 'spice', 'everything nice']):
...     print(item)
...
(1, 'sugar')
(2, 'spice')
(3, 'everything nice')
```

更正式地: `zip()` 返回元组的迭代器 其中第*i*个元组包含来自每个参数可迭代对象的第*i*个元素。

另一种思考方式`zip()`是将行转换为列，将列转换为行。这类似于转置矩阵。

`zip()` 懒惰：在迭代可迭代对象之前不会处理元素，例如通过`for`循环或包装在 `list`.

需要考虑的一件事是传递给的可迭代对象`zip()`可能具有不同的长度；有时是设计使然，有时是因为准备这些迭代的代码中的错误。Python 提供了三种不同的方法来处理这个问题：

- 默认情况下，`zip()`当最短的迭代用完时停止。它将忽略较长迭代中的剩余项，将结果截断为最短迭代的长度：

```
>>> list(zip(range(3), ['fee', 'fi', 'fo', 'fum']))
[(0, 'fee'), (1, 'fi'), (2, 'fo')]
```

- `zip()`通常用于假设可迭代对象长度相等的情况。在这种情况下，建议使用该`strict=True`选项。它的输出与常规相同`zip()`：

```
>>> list(zip(('a', 'b', 'c'), (1, 2, 3), strict=True))
[('a', 1), ('b', 2), ('c', 3)]
```

  与默认行为不同，它检查可迭代对象的长度是否相同，`ValueError`如果不相同则引发 a：

```
>>> list(zip(range(3), ['fee', 'fi', 'fo', 'fum'], strict=True))
Traceback (most recent call last):
  ...
ValueError: zip() argument 2 is longer than argument 1
```

  如果没有`strict=True`参数，任何导致不同长度迭代的错误都将被消除，可能表现为程序另一部分中难以找到的错误。

- 较短的可迭代对象可以填充一个常量值，以使所有可迭代对象具有相同的长度。这是由`itertools.zip_longest()`.

边缘情况：使用单个可迭代参数，`zip()`返回 1 元组的迭代器。没有参数，它返回一个空的迭代器。

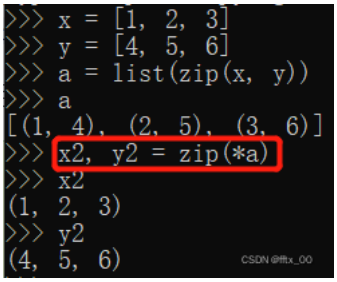
技巧和窍门：

- 可迭代对象的从左到右的评估顺序是有保证的。这使得使用 将数据系列聚类为 n 长度组的习语成为可能。这将重复*相同的*迭代器 次数，以便每个输出元组都具有对迭代器的调用结果。这具有将输入分成 n 个长度的块的效果。`zip(*[iter(s)]*n, strict=True)nn`

- `zip()` 与 `*` 运算符相结合可以用来拆解一个列表：

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> list(zip(x, y))
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*zip(x, y))
>>> x == list(x2) and y == list(y2)
True
```

在 3.10 版更改: Added the `strict` argument.



```
>>>a = [1,2,3]
>>> b = [4,5,6]
>>> c = [4,5,6,7,8]
>>> zipped = zip(a,b)     # 返回一个对象
>>> zipped
<zip object at 0x103abc288>
>>> list(zipped)  # list() 转换为列表
[(1, 4), (2, 5), (3, 6)]
>>> list(zip(a,c))              # 元素个数与最短的列表一致
[(1, 4), (2, 5), (3, 6)]

>>> a1, a2 = zip(*zip(a,b))          # 与 zip 相反，zip(*) 可理解为解压，返回二维矩阵式
>>> list(a1)
[1, 2, 3]
>>> list(a2)
[4, 5, 6]
>>>
```


十一、map()

map() 会根据提供的函数对指定序列做映射。
第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。

map(*function*, *iterable*, ...)

返回一个将 *function* 应用于 *iterable* 中每一项并输出其结果的迭代器。 如果传入了额外的 *iterable* 参数，*function* 必须接受相同个数的实参并被应用于从所有可迭代对象中并行获取的项。 当有多个可迭代对象时，最短的可迭代对象耗尽则整个迭代就将结束。 对于函数的输入已经是参数元组的情况，请参阅 itertools.starmap()。


```
>>> a = [2,3,4,5]
>>> b = [1,2,3,4]
>>> map(math.pow,a,b)
<map object at 0x00000266CF5094F0>
>>> x= list(map(math.pow,a,b))
>>> x
[2.0, 9.0, 64.0, 625.0]
```


Python3.x 实例

```
>>> def square(x) :          # 计算平方数
...     return x ** 2
...
>>> map(square, [1,2,3,4,5])    # 计算列表各个元素的平方
<map object at 0x100d3d550>     # 返回迭代器
>>> list(map(square, [1,2,3,4,5]))   # 使用 list() 转换为列表
[1, 4, 9, 16, 25]
>>> list(map(lambda x: x ** 2, [1, 2, 3, 4, 5]))   # 使用 lambda 匿名函数
[1, 4, 9, 16, 25]
>>>
```


参考资料：Python map() 函数_lilong117194的博客-CSDN博客_python中map()

十二、reduce()

reduce() 函数会对参数序列中元素进行累积。
函数将一个数据集合（链表，元组等）中的所有数据进行下列操作：用传给 reduce 中的函数 function（**有两个参数**）先对集合中的第 1、2 个元作，**得到的结果再与第三个数据用 function 函数运算**，最后得到一个结果。

functools. **reduce**(*function*, *iterable*[, *initializer*])

将两个参数的 *function* 从左至右积累地应用到 *iterable* 的条目，以便将该可迭代对象缩减为单一的值。 例如，`reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])` 是计算 `((((1+2)+3)+4)+5)` 的值。 左边的参数 *x* 是积累值而右边的参数 *y* 则是来自 *iterable* 的更新值。 如果存在可选项 *initializer*，它会被放在参与计算的可迭代对象的条目之前，并在可迭代对象为空时作为默认值。 如果没有给出 *initializer* 并且 *iterable* 仅包含一个条目，则将返回第一项。

大致相当于：

```
def reduce(function, iterable, initializer=None):
    it = iter(iterable)
    if initializer is None:
        value = next(it)
    else:
        value = initializer
    for element in it:
        value = function(value, element)
    return value
```

请参阅 `itertools.accumulate()` 了解有关可产生所有中间值的迭代器。

实例(Python 3.x)

```
#!/usr/bin/python
from functools import reduce

def add(x, y) :            # 两数相加
    return x + y
sum1 = reduce(add, [1,2,3,4,5])   # 计算列表和：1+2+3+4+5
sum2 = reduce(lambda x, y: x+y, [1,2,3,4,5])  # 使用 lambda 匿名函数
print(sum1)
print(sum2)
```


以上实例输出结果为：

```
15
15
```

但是此处如果只计算加法，直接
sum1 = sum([1, 2, 3, 4, 5])
更方便

十三、filter()

filter() 函数用于过滤序列，过滤掉不符合条件的元素，返回一个迭代器对象，如果要转换为列表，可以使用 list() 来转换。
该接收两个参数，第一个为函数，第二个为序列，序列的每个元素作为参数传递给函数进行判断，然后返回 True 或 False，最后**将返回 True 的新列表中。**

filter(*function*, *iterable*)

用 *iterable* 中函数 *function* 返回真的那些元素，构建一个新的迭代器。*iterable* 可以是一个序列，一个支持迭代的容器，或一个迭代器。如果 *function* 是 `None` ，则会假设它是一个身份函数，即 *iterable* 中所有返回假的元素会被移除。

请注意， `filter(function, iterable)` 相当于一个生成器表达式，当 function 不是 `None` 的时候为 `(item for item in iterable if function(item))`；function 是 `None` 的时候为 `(item for item in iterable if item)` 。

请参阅 `itertools.filterfalse()` 了解，只有 *function* 返回 false 时才选取 *iterable* 中元素的补充函数。

过滤出1~100中平方根是整数的数：

```
#!/usr/bin/python3

import math
def is_sqr(x):
    return math.sqrt(x) % 1 == 0

tmplist = filter(is_sqr, range(1, 101))
newlist = list(tmplist)
print(newlist)
```

输出结果：

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```