Q1:A Plus Abs B

operator 模块

import operator

常用的 operator 函数:

- **算术运算符**:
  - `add(x, y)`：返回 `x + y`
  - `sub(x, y)`：返回 `x - y`
  - `mul(x, y)`：返回 `x * y`
  - `truediv(x, y)`：返回 `x / y`（浮点除法）
  - `floordiv(x, y)`：返回 `x // y`（整数除法）
  - `mod(x, y)`：返回 `x % y`（模）
  - `pow(x, y)`：返回 `x ** y`（幂）
- **比较运算符**:
  - `lt(x, y)`：返回 `x < y`
  - `le(x, y)`：返回 `x <= y`
  - `eq(x, y)`：返回 `x == y`
  - `ne(x, y)`：返回 `x != y`
  - `gt(x, y)`：返回 `x > y`
  - `ge(x, y)`：返回 `x >= y`
- **逻辑运算符**:
  - `and_()`：返回 `x and y`
  - `or_()`：返回 `x or y`
  - `not_()`：返回 `not x`
- **其他常用操作**:
  - `concat(x, y)`：返回 `x + y`（用于连接字符串）
  - `indexof(x, y)`：返回 `x.index(y)`（在 `x` 中查找 `y` 的索引）

为什么使用operator模块?

- 将函数作为参数

```
sorted_numbers = sorted(numbers, key=abs)
```

- 可阅读性

python自带其他模块（python内建库）

所有模块

- `sys`：与 Python 解释器和操作系统交互。
- `os`：文件和目录操作、进程管理等。
- `time`：时间和日期操作。
- `random`：生成随机数。
- `collections`：扩展数据结构（如 `Counter`、`deque`）。
- `itertools`：高效的迭代器工具。
- `json`：处理 JSON 数据。
- `math`：高级数学计算。
- `operator`：运算符函数。

题目相关

```
from operator import add, sub

def a_plus_abs_b(a, b):
    """Return a+abs(b), but without calling abs.//不能调用operator中的abs函数

    >>> a_plus_abs_b(2, 3)
    5
    >>> a_plus_abs_b(2, -3)
    5
    >>> # a check that you didn't change the return statement!
    >>> import inspect, re
    >>> re.findall(r'^\s*(return .*)', inspect.getsource(a_plus_abs_b), re.M)
    ['return f(a, b)']
    """
    if b < 0:
        f = sub
    else:
        f = add
    return f(a, b)
```

可以直接使用f=函数名对一个变量赋予函数的功能（函数的引用）

Q2：Two of Three

问题描述：编写一个函数，接受三个正数作为参数，返回其中两个最小数的平方和。

有关序列，迭代，索引，切片

1. 序列（Sequence）

序列是指一种可以按**顺序访问元素的数据结构**。在 Python 中，**序列**类型包括**列表、元组、字符串、范围（`range`）、字典（通过键访问）等**。序列是一个包含多个元素的集合，这些元素有一个固定的顺序，并且可以通过索引（或其他方式）来访问。

常见的序列类型：

- **列表（List）**：有序且可变的**集合**，如 `[1, 2, 3]`。
- **元组（Tuple）**：有序但不可变的**集合**，如 `(1, 2, 3)`。
- **字符串（String）**：有序且不可变的字符**序列**，如 `"hello"`。
- **范围（Range）**：生成一系列整数，如 `range(5)` 产生 `0, 1, 2, 3, 4`。
- **字典（Dictionary）**：无序的键值对集合，虽然字典本身不算传统意义上的序列，但可以通过键来访问数据。

序列的特点：

- **有序**：序列中的元素有一个特定的顺序。
- **可迭代**：你可以遍历序列中的所有元素。

所有的序列都具有可迭代性，都是可迭代对象；可迭代对象不一定是序列（不一定有顺序，比如集合）

2. 迭代（Iteration）:可迭代对象

迭代是指逐个访问**序列**或其他**可迭代对象**中的元素的过程。Python 提供了多种方法来迭代序列，常见的方式有 `for` 循环、`while` 循环，以及使用迭代器和生成器等。

- **可迭代对象（Iterable）**：任何支持迭代的对象都叫做可迭代对象（如列表、元组、字符串、字典、集合、**生成器**等）。
- **迭代器（Iterator）**：一个实现了 `__iter__()` 方法和 `__next__()` 方法的对象。你可以通过迭代器逐一获取序列中的元素。

迭代的例子：

```
##使用 for 循环迭代列表
my_list = [1, 2, 3, 4, 5]
for elem in my_list:
    print(elem)
# 输出：1 2 3 4 5
```

在一个迭代后面可以用冒号添加表达式用来重复执行某个表达式

```
#使用for循环迭代生成器
for i in range(begin,end,步长)
```

迭代器本身就可以做一个计数器，用来记录并控制某种行为的次数

```
for i in range(6):
    print(...)
```

生成器

range ()

range（开始数字，结束数字（不输出），步长）

yield关键字

```
def mu_gennerator():
    yield 1
    yield 2
    yield 3
gen =my_generator():
for i in gen:
    print(i)
```

生成器表达式

```
gen_exp = (x * x for x in range(5))
for num in gen_exp:
    print(num)
```

itertools 模块

`itertools` 是 Python 标准库中一个提供高效的迭代器生成工具的模块，其中有许多函数返回生成器对象。

常用的 `itertools` 函数：

- `itertools.count()`：生成从某个数开始的无限整数序列。

  ```
  import itertools
  counter = itertools.count(10, 2)  # 从 10 开始，每次增加 2
  for i in range(5):
      print(next(counter))
  ```

  输出：

```
10
12
14
16
18
```

- `itertools.cycle()`：对序列进行循环迭代。

```
import itertools
cyclic = itertools.cycle(['A', 'B', 'C'])
for i in range(6):
    print(next(cyclic))
```

输出：

```
A
B
C
A
B
C
```

- `itertools.repeat()`：重复某个值。

```
import itertools
repeated = itertools.repeat(5, 3)  # 重复值 5，重复 3 次
for val in repeated:
    print(val)
```

输出：

```
5
5
5
```

- `itertools.chain()`：将多个迭代器连接起来。

```
python复制代码import itertools
combined = itertools.chain([1, 2], [3, 4], [5, 6])
for i in combined:
    print(i)
```

输出：

```
1
2
3
4
5
6
```

- `itertools.islice()`：切片迭代器，获取某个序列的指定部分。

```
import itertools
sliced = itertools.islice(range(10), 2, 8)
for i in sliced:
    print(i)
```

输出：

```
2
3
4
5
6
7
```

3. 索引（Indexing）：序列

索引是指通过指定一个数字位置来访问**序列**中的某个元素。序列中的元素是有顺序的，每个元素都有一个唯一的索引，索引通常是从 **0 开始的**。你可以使用 **方括号（`[]`）** 来访问序列中的元素。

- 对于正索引，`0` 对应第一个元素，`1` 对应第二个元素，依此类推。
- 对于负索引，`-1` **对应最后一个元素，**`-2` **对应倒数第二个元素，依此类推。**

索引的例子：

```
my_list = [10, 20, 30, 40, 50]

print(my_list[0])  # 10
print(my_list[2])  # 30
print(my_list[-1]) # 50
print(my_list[-2]) # 40
```

4. 切片（Slicing）：序列

切片是指从序列中提取一个子序列（一个新的子集）。切片允许你通过指定开始索引、结束索引和步长来获取序列的一部分。切片的语法是 `sequence[start:end:step]`，其中：

- `start`：切片的开始位置（**包含**）。
- `end`：切片的结束位置（**不包含**）。
- `step`：切片的步长（默认是 `1`）。

切片的例子：

```
my_list = [10, 20, 30, 40, 50]

print(my_list[1:4])   # [20, 30, 40]  从索引 1 到 3
print(my_list[:3])    # [10, 20, 30]  从开头到索引 2
print(my_list[2:])    # [30, 40, 50]  从索引 2 到结尾
print(my_list[::2])   # [10, 30, 50]  每隔一个元素取一个
print(my_list[::-1])  # [50, 40, 30, 20, 10]  反转列表
```

切片的使用不受限制，可以在任意的序列后面添加，不会影响序列自身的类型

lambda函数

`lambda` 函数是 Python 中的一种简洁的匿名函数，也称为 **匿名函数**，用于定义一个没有名字的小型函数。与常规的 `def` 函数相比，`lambda` 函数通常用于函数体较短、只需要简单操作的情况。

```
lambda 参数：表达式
```

Python一些特殊的运算符

** 运算符：表示幂运算

```
i**2
```

and 运算符：逻辑与

or 运算符：逻辑或

not 运算符：逻辑非

in/not in运算符：检查某一个元素是否在序列中

is/is not运算符：判断是还是不是一个对象

Python 内建函数

python不需要添加任何内部或者外部库也自带的函数

数据类型相关函数

- `type()`：返回对象的类型。

- `len()`：返回对象的长度或元素个数。
- `int()`：将对象转换为整数。
- `float()`：将对象转换为浮点数。
- `str()`：将对象转换为字符串。
- `list()`：将对象转换为列表。
- `tuple()`：将对象转换为元组。
- `set()`：将对象转换为集合。
- `dict()`：创建一个字典。

序列操作

- **`sorted()`：返回排序后的列表。**
- **`reversed()`：返回反向迭代器。**
- `sum()`：返回序列元素的总和。
- `max()`：返回最大元素。
- `min()`：返回最小元素。
- `all()`：如果序列中的所有元素为真，返回 `True`。
- `any()`：如果序列中有任意元素为真，返回 `True`。

数学函数

- `abs()`：返回绝对值。
- `round()`：返回四舍五入后的数字。
- `pow()`：返回 x 的 y 次幂。
- `divmod()`：返回商和余数的元组。
- `max()`、`min()`：返回序列中的最大值和最小值。

输入输出

- `print()`：输出到控制台。
- `input()`：从用户获取输入（通常为字符串）。
- `open()`：打开一个文件。

逻辑运算和条件判断

- `all()`：如果序列中的所有元素都为真，返回 `True`。
- `any()`：如果序列中至少有一个元素为真，返回 `True`。
- `not()`：布尔值取反。
- `isinstance()`：检查对象是否是某个类的实例。
- `id()`：返回对象的内存地址。

迭代相关

- `enumerate()`：返回可枚举对象的索引和元素。

- `zip()`：将多个可迭代对象压缩在一起，返回一个元组的迭代器。
- `map()`：对可迭代对象中的每个元素应用一个函数。
- `filter()`：过滤序列中的元素。

函数工具

- `lambda`：用于创建匿名函数。
- `callable()`：检查对象是否是可调用的（如函数或方法）。
- `globals()`：返回当前全局符号表。
- `locals()`：返回当前局部符号表。

对象相关

- `isinstance()`：检查对象是否是指定类型的实例。
- `getattr()`：获取对象的属性值。
- `setattr()`：设置对象的属性值。
- `delattr()`：删除对象的属性。

错误处理

- `raise()`：用于引发异常。

其他

- `id()`：返回对象的唯一标识符。
- `help()`：获取对象的帮助信息。
- `dir()`：列出对象的所有属性和方法。

sum的基本语法：可迭代对象

`sum()` 是 Python 的内建函数，用于对可迭代对象中的所有元素进行求和。

```
sum(iterable, start=0)
```

- `iterable`：这是一个可迭代对象（例如列表、元组、集合等）。
- `start`：这是一个可选的参数，表示开始加和的初始值，默认是 `0`。

`sum()` 函数的主要目的是将可迭代对象中的所有元素加起来，但它也可以接受一些更复杂的参数：

1. 多个数字作为参数

你可以将多个数字传递给 `sum()` 函数，但它们需要先组成一个可迭代对象（例如一个列表或元组）。

```
numbers = [1, 2, 3, 4]
result = sum(numbers)  # 等价于 sum([1, 2, 3, 4])
print(result)  # 输出 10
```

2. 一个表达式加上迭代

`sum()` 也可以接受一个 **生成器表达式** 或 **列表推导式** 作为参数。在这种情况下，`sum()` 会将生成器中的每个值进行求和。

例如：

```
# 计算 1 到 5 的平方和
result = sum(i**2 for i in range(1, 6))
print(result)  # 输出 55，因为 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55
```

在上面的例子中，`i**2 for i in range(1, 6)` 是一个生成器表达式，它会生成从 `1` 到 `5` 每个数字的平方。`sum()` 会对这些平方值进行求和。

3. 多个参数或一个迭代对象

你也可以向 `sum()` 传递多个参数或一个包含多个数字的可迭代对象。

例如：

```
result = sum([1, 2, 3], 10)  # 从 10 开始加
print(result)  # 输出 16，因为 10 + 1 + 2 + 3 = 16
```

4. 迭代器中的表达式

你也可以在 `sum()` 中使用一个复杂的表达式，例如，计算某个数列的平方和，或者是一些条件判断的结果。

例如：

```
# 求 1 到 5 中大于 2 的数字的平方和
result = sum(i**2 for i in range(1, 6) if i > 2)
print(result)  # 输出 50，因为 3^2 + 4^2 + 5^2 = 9 + 16 + 25 = 50
```

sorted的基本语法：可迭代对象

`sorted()` 是 Python 内建的一个非常有用的函数，用于对**可迭代对象**进行排序。它可以对列表、元组、字符串等可迭代对象进行排序，并返回一个新的列表。值得注意的是，`sorted()` **不会修改原始的可迭代对象，而是返回一个新的排序后的列表。**

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

- `iterable`：要排序的可迭代对象，可以是列表、元组、字符串等。
- `key`（可选）：一个函数，用来指定排序的标准。默认是 `None`，表示按照元素本身的值排序。如果提供 `key`，则会根据 `key` 函数的返回值进行排序。
- `reverse`（可选）：一个布尔值，指定排序的顺序。`reverse=False` 表示升序排序，`reverse=True` 表示降序排序，默认是升序。