函数

函数的说明文档

语法如下：

```
def func(x,y):
    """
    函数说明
    :param x:形参x的说明
    :param y:形参y的说明
    :return:返回值的说明
    """
    函数体
    return 返回值
```

pycharm软件中，鼠标悬停在调用的函数上，会显示函数的说明文档

函数的嵌套调用

就是指**一个函数里面又调用了另外一个函数**

```
"""
嵌套函数的使用
"""

# 定义函数func_b
def func_b():
    print("----2")
# 定义函数func_a,并在内部调用func_b
def func_a():
    print("----1")
    # 嵌套调用func_b
    func_b() #执行到此步时，会执行func_b完后继续向下执行

    print("----3")

# 调用函数func_a
func_a()
```

变量的作用域

变量的作用域指的是变量的作用范围（变量在哪里可用，在哪里不可用）

主要分为：局部变量和全局变量

所谓局部变量就是**定义在函数体内部的变量，即只在函数体的内部生效**

局部变量的作用：在函数体内部，临时保存数据，即当函数调用完成之后，则销毁变量。

所谓全局变量就是**在函数体内部和外部都能生效的变量。**（当然在函数外定义）

global关键字：使用global关键字可以**在函数内部声明变量为全局变量**

```
# 在函数内修改全局变量
num = 200

def test_a():
    print(f"test_a:{num}")

def test_b():
    num = 500 # 在这里，num是个局部变量,是个新定义的变量
    print(f"test_b:{num}")

test_a()
test_b()
print(num)

"""
输出结果：
test_a():200
test_b():500
200
"""
```

```
num = 100

def test_A():
    print(num)

def test_B():
    # global关键字声明num是全局变量
    global num
    num = 200 # 这时，num是同一个num
    print(num)

test_A()
test_B()
print(num)
"""
输出结果：
100
200
200
"""
```

综合案例：

```
"""
函数综合案例
"""
```

```python
# 定义全局变量money name
money = 500000
name = None
# 要求用户输入姓名
name = input("请输入你的姓名：")
# 定义查询函数
def query(show):
    if show:
        print("-------------查询余额--------------")
    print(f"{name}，您好,您的余额剩余：{money}元")

# 定义存款金额
def saving(num):
    global money
    money += num
    print("------------存款-------------")
    print(f"您存入了{num}元，当前余额:{money}元") #这里也可以调用查询函数 query(false)

# 定义取款函数
def getmoney(num):
    global money
    money -= num
    print("------------取款------------")
    print(f"您支取了{num}元，当前余额:{money}元") #这里也可以调用查询函数 query(false)

# 定义主菜单函数
def  main():
    print("----------------主菜单----------------")
    print(f"{name}，你好，请选择操作")
    print("查询余额\t【输入1】") # \t 水平制表符
    print("存款\t\t【输入2】")
    print("取款\t\t【输入3】")
    print("退出\t\t【输入4】")
    return input("请输入你的选择：")

# 设置无限循环，确保函数不退出
while True:
    key = main()
    if key == "1":
        query()
        continue
    elif key == "2":
        num = int(input("您要存入多少钱？请输入："))
        saving(num)
        continue
    elif key == "3":
        num = int(input("您要支取多少钱？请输入："))
        getmoney(num)
        continue
    else:
        print("程序退出")
        break
```

数据容器

python中的数据容器：一种可以容纳多份数据的数据类型，容纳的每一份数据称之为1个元素，每一个元素，可以是任意类型的数据，如字符串，数字，布尔等。

数据容器根据特点的不同，如：

- 是否支持重复元素
- 是否可以修改
- 是否有序，等

分为5类，分别是：

列表（list）、元组（tuple）、字符串（str）、集合（set）、字典（dict）

List（列表）：

基本语法:

```
# 字面量
[元素1,元素2,元素3,元素4...]

# 定义变量
变量名称 = [元素1,元素2,元素3,元素4...]

# 定义空列表
变量名称 = []
变量名称 = list()
```

列表内的每一个数据，称为元素

- 以 [] 作为标识
- 列表内每一个元素之间用逗号隔开

```
name_list = ['itheima','itcast','python']
print(name_list)
print(type(name_list))

"""
输出结果:
['itheima','itcast','python']
<class 'list'>
"""
```

```
name_list = ['itheima',666,True]
print(name_list)
print(type(name_list))

"""
输出结果:
['itheima',666,True]
<class 'list'>
"""
```

注意：列表可以**一次存储多个数据**，且**可以为不同的数据类型**，支持**嵌套**

```
my_list = [[1,'q',false],[4,6],'python']
```

列表的索引

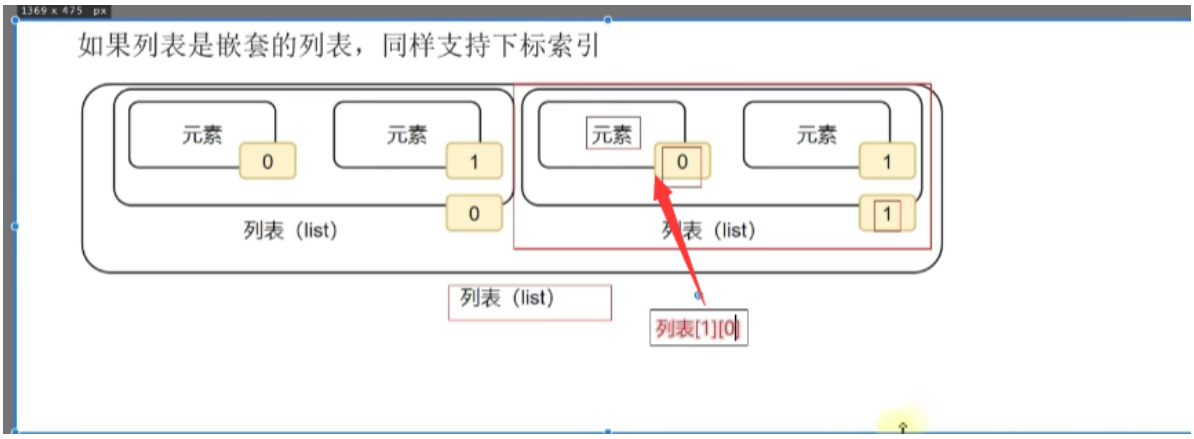
```
# 语法：列表[下标索引]，从前向后下标索引从0开始，依次递增
# 超出下标索引范围会报错

name_list = ['tom','lily','rose']
print(name_list[0]) # 结果：tom
print(name_list[1]) # 结果：lily
print(name_list[2]) # 结果：rose
```

```
# 也可以反向索引，从后向前下标从-1开始，依次递减

name_list = ['tom','lily','rose']
print(name_list[-1]) # 结果：rose
print(name_list[-2]) # 结果：lily
print(name_list[-3]) # 结果：tom
```

嵌套列表的索引




```
my_list = [['li','wang'],[34,69]]
print(my_list[1])
print(my_list[0][1])
print(my_list[-1][0])

"""
输出结果：
[34,69]
wang
34
"""
```

列表的常用操作（方法）

列表除了可以：

- 定义
- 使用下标索引获取值

此外，列表还提供了一系列功能：

- 插入元素
- 删除元素
- 清空列表
- 修改元素
- 统计元素个数

等等功能，我们称之为：**列表的方法**

回忆：函数是一个封装的代码单元，可以提供特定功能。

在python中，如果将函数定义为class（类）的成员，那么函数会被称之为：方法

方法和函数功能一样，有传入参数，有返回值，只是方法的使用格式不同：

函数的使用：`num = add(1,2)`

方法的使用：`student = Student()`

`num = student.add(1,2)`

查询功能（方法）

- 查找某元素的下标

  功能：查找指定元素在列表的下标（从前向后的下标），如果找不到，报错 ValueError

  语法：列表.index(元素)

index 是列表内置的方法

```
my_list = ['tom','lily','rose']

index =my_list.index("tom")
print(f"tom在列表中的下标索引值是：{index}")

"""
输出结果：
tom在列表中的下标索引值是：0
"""
```

修改功能（方法）

- 修改特定位置（索引）的元素值：

  语法：**列表[下标] = 值**

```
my_list[1] = "你好"
```

- 插入元素：

语法：**列表.insert(下标，元素)**，在指定的下标位置，插入指定的元素

```
my_list = [1,2,3]
my_list.insert(1,"tom")
print(my_list)  # 结果：[1,"tom",2,3]
```

- 追加元素：

方式1语法：**列表.append(元素)**，将指定元素，追加到列表的尾部（只能追加一个元素，一个元素可以是一个列表）

```
my_list = [1,2,3]
print(my_list) #结果：[1,2,3]
my_list.append(6) #一次只能追加一个元素
pritn(my_list) #结果：[1,2,3,6]
my_list.append([7,8,9])
pritn(my_list) #结果：[1,2,3,6,[7,8,9]]
```

方式2语法：**列表.extend(其他数据容器)**，将其它数据容器的内容取出，依次追加的列表尾部（追加一批元素，只有一个也是可以的）

```
my_list = [1,2,3]
print(my_list) # 结果：[1,2,3]
my_list.extend([4,5,6])
print(my_ist) # 结果：[1,2,3,4,5,6]
my_list.extend([7]) # 也可以只有一个元素
print(my_ist) # 结果：[1,2,3,4,5,6,7]
my_list2 = [8,9]
my_list.extend(my_list2)
print(my_ist) # 结果：[1,2,3,4,5,6,7,8,9]
```

- 删除元素

语法1：**del 列表[下标]**

语法2：**列表.pop(下标)**

```
my_list = [1,2,3]
print(my_list) # 结果：[1,2,3]
# 方式1
del my_list[0] #del 用中括号，仅仅是删除

print(my_list) # 结果：[2,3]
# 方式2
ele = my_list.pop(0) #pop用小括号，pop是将某个元素取出来，也就是我们可以接收该元素
print(ele) # 结果：2
print(my_list) # 结果：[3]
```

- 删除某元素在列表的第一个匹配项

语法：**列表.remove(元素)**

```
my_list = [1,2,3,2,3]
my_list.remove(2)
print(my_list) # 结果：[1,3,2,3]
```

- 清空列表内容，语法：**列表.clear()**

```
my_list = [1,2,3]
my_list.clear()
print(my_list) # 结果：[]
```

- 统计某元素在列表内的数量，语法：**列表.count(元素)**

```
my_list = [1,2,3,2,2,3,4]
print(my_list.count(2)) # 结果：3
```

- 统计列表中，共有多少元素（就是列表的长度），语法：**len(列表)**

```
my_list = [1,2,"tom",4]
print(len(my_list)) # 结果：4
```

编号	使用方式	作用
1	列表.append(元素)	追加一个元素
2	列表.extend(容器)	将数据容器内的内容依次取出，追加一批元素到列表
3	列表.insert(下标，元素)	指定下标处，插入指定元素
4	del 列表[下标]	删除指定下标元素
5	列表.pop(下标)	取出指定下标元素
6	列表.remove(元素)	删除第一个匹配项
7	列表.clear()	清空列表
8	列表.count(元素)	统计某元素在列表中出现的次数
9	列表.index(元素)	查找指定元素的下标
10	len(列表)	统计容器内有多少个元素

列表的遍历（while、for）

while循环：

```
# while循环
def list_while_func():

    my_list = ["中建八局","欢迎你的到来","朋友！"]
    index = 0
    while index < len(my_list):
        print(my_list[index])
        index += 1
# 调用遍历函数
list_while_func()
```

for循环：

```
# for循环
def list_for_func():
    my_list = [1,2,3,4,5]
    for element in my_list:
        print(element)

# 调用函数
list_for_func()
```

while循环和for循环，都是循环语句，但细节不同：

- 在循环控制上：
  - while循环可以自定循环条件，并自行控制
  - for循环不可以自定循环条件，只能从容器内一个个取出数据
- 在无限循环上：
  - while循环可以通过条件控制做到无限循环
  - for循环理论上不可以，因为容器容量不是无限大的
- 在使用场景上：
  - while循环适用于任何想要循环的场景
  - for循环适用于，遍历数据容器场景或简单的固定次数循环场景

Tuple元组

元组同列表一样，都是可以封装多个、不同类型的元素在内。但最大不同在于：**元组一旦定义完成，就不可修改。**（但是，如果元组内有list列表，那么list内的内容可以修改）

元组定义：定义元组使用**小括号**，且使用逗号隔开各个数据，数据可以是不同的数据类型。

```
# 定义元组字面量
(元素,元素,元素,......)
# 定义元组变量
变量名称 = (元素,元素,......)
# 定义空元组
变量名称 = ()        #方式1
变量名称 = tuple()   #方式2
```

注意：**元组只有一个数据时，这个数据后面要添加逗号（不然会被认成string型）**

元组也支持嵌套：

```
# 定义一个嵌套元组
t1 = ((1,2,3),(4,5,6))
print(t1[0][0]) # 结果：1
```

元组的相关操作：

编号	方法	作用
1	index()	查找某个数据，返回对应下标
2	count()	统计某个数据出现的次数
3	len(元组)	统计元组内的元素个数

同样，**元组也支持下标索引。**

元组的遍历（while、for）：

```
t8 = (1,3,"hello",[2,"you",True],False)
# while
index = 0
while index < len(t8):
    print(f"元组的元素有：{t8.[index]}")
    index += 1

# for
for element in t8:
    print(f"元组的元素有：{element}")
```

案例演示：

```
t_stu = ('坤坤',19,['basketball','music'])
# 查询年龄的下标
print(f"年龄的下标位置是：{t_stu.index(19)}") # 年龄的下标位置：1
# 查询学生姓名
print(f"姓名是：{t_stu[0]}") # 姓名是：坤坤
# 删除篮球爱好
del t_stu[2][0]
print(t_stu) # ('坤坤', 19, ['music'])
# 增加coding爱好到list
t_stu[2].append("coding")
print(t_stu) # ('坤坤', 19, ['music', 'coding'])
```

Str字符串

字符串是字符的容器，一个字符串可以存放任意数量的字符。

字符串的下标（索引）

和其他容器：列表，元组一样，字符串也可以通过下标进行访问

- 从前向后，下标从0开始
- 从后向前，下标从-1开始

同元组一样，字符串也是一个：**无法修改的数据容器**

字符串的常用操作

- 查找特定字符串的下标索引值

  语法：字符串.index(字符串)

```
my_str = "itcast and itheima"
print(mystr.index("and")) # 结果：7
```

- 字符串的替换

  语法：字符串.replace(字符串1，字符串2)

  功能：将字符串内的全部：字符串1，替换为字符串2

  注意：不是修改字符串本身，而是获得了一个新字符串

- 字符串的分割

  语法：字符串.split(分隔符字符串)

  功能：按照指定的分隔符字符串，将字符串划分为多个字符串，并存入列表对象中。

  注意：字符串本身不变，而是得到了一个列表对象

- 字符串的规整操作（去前后空格）

  语法：**字符串.strip()**

```
my_str = "  itheima and itcast  "
print(my_str.strip()) # 结果："itheima and itcast"
```

- 字符串的规整操作（去前后指定字符串）

  语法：**字符串.strip(字符串)**

```
my_str = "12itheima and itcast21"
print(my_str.strip("12")) # 结果："itheima and itcast"
```

注意：传入的是”12“其实就是：”1“和”2“都会移除，是按照单个字符

- 统计在字符串内某个字符串出现的次数

  语法：字符串.count(字符串)

- 统计字符串的长度

  语法：len(字符串)

编号	操作	作用
1	字符串[下标]	根据下标索引相应字符串
2	字符串.index(字符串)	查找第一个匹配项的下标
3	字符串.replace(字符串1，字符串2)	将字符串内的全部字符串1替换为字符串2。不会修改原字符串，而是得到一个新的字符串
4	字符串.split(字符串)	按给定字符串进行分割。不会修改原字符串，而是得到一个新的列表
5	字符串.strip() 字符串.strip(字符串)	移除首尾的空格和换行符或指定字符串
6	字符串.count(字符串)	统计字符串内某字符串出现的次数
7	len(字符串)	统计字符串的字符个数

```
"""
演示字符串
"""
my_str = "My name is Tony!"
# 通过下标索引
value1 = my_str[2] # 可以发现，是空格。空格同样计入下标
value2 = my_str[-8]
print(f"从字符串{my_str}中取下标为2的元素，值是{value1};取下标为-8的元素，值是{value2}")
# 从字符串My name is Tony!中取下标为2的元素，值是 ;取下标为-8的元素，值是i

# index 方法
value = my_str.index("name")
print(f"在字符串{my_str}中查找name，其起始下标是{value}")
# 在字符串My name is Tony!中查找name，其起始下标是3

# replace 方法
new_my_str = my_str.replace("Tony","Kali") # 注意，如果某一字符串出现多次，则都将会进行替换
print(f"将字符串{my_str}进行替换后得到：{new_my_str}")
# 将字符串My name is Tony!进行替换后得到：My name is Kali!

# split 方法
my_str = "hello python iter"
my_str_list = my_str.split(" ") # 按空格切分
print(my_str_list) # ['hello', 'python', 'iter']

# strip 方法
my_str = "  hello python iter  "
new_my_str = my_str.strip() # 不传入参数，去除首尾空格
print(new_my_str) # "hello python iter"
my_str = "12hello python iter21"
new_my_str = my_str.strip("12") # 去除1和2
print(new_my_str) # "hello python iter"

# 统计字符串中某个字符出现的次数，count
```

```
my_str = "hello itpython iter"
count = my_str.count("it")
print(count) # 结果：2

# 统计字符串长度
my_str = "hello python iter"
length = len(my_str)
print(length) # 结果：17
```

字符串的遍历：

同列表、元组一样，字符串也支持while循环和for循环进行遍历。

序列切片（list、tuple、str）

序列是指：内容连续、有序、可使用下标索引的一类数据容器。

列表、元组、字符串，均可以视为序列。

序列的常用操作——切片

切片：从一个序列中，取出一个子序列。

语法：序列[起始下标：结束下标：步长]

表示从序列中，从指定位置开始，依次取出元素，到指定位置结束，得到一个新序列：

- 起始下标可以为空，留空视作从头开始
- 结束下标也可以为空，留空视作截取到结尾
- 步长表示，依次取元素的间隔
    - 步长1表示，一个个取元素
    - 步长2表示，每次跳过1个元素取
    - 步长N表示，每次跳过N-1个元素取
    - 步长为负数表示，反向取（注意，此时起始下标和结束下标也要反向标记）

  注意：切片操作不会影响序列本身，而是得到了一个新的序列（列表、元组、字符串）。原因：元组和字符串都是不支持修改的。

```
"""
对序列切片操作
"""

# 对list进行切片，从1开始，4结束，步长1
my_list = [0,1,2,3,4,5,6]
result1 = my_list[1:4] # 步长默认是1，可以省略不写
print(f"结果1：{result1}")  #结果1：[1,2,3]

# 对tuple进行切片，从头开始，到最后结束，步长1
my_tuple = (0,1,2,3,4,5,6)
result2 = my_tuple[:]
print(f"结果2：{result2}") #结果2：(0,1,2,3,4,5,6)

# 对str进行切片，从头开始，到最后结束，步长为2
my_str = "0123456"
```

```
result3 = my_str[::2]
print(f"结果3：{result3}") #结果3：0246

# 对str进行切片，从头开始，到最后结束，步长为-1
my_str = "01234567"
result4 = my_str[::-1]       #等同于将序列反转
print(f"结果4：{result4}") #结果4：76543210

# 对列表进行切片，从3开始，到1结束，步长-1
my_list = [0,1,2,3,4,5,6]
result5 = my_list[3:1:-1] #注意，这里起始下标和结束下标已经反转了
print(f"结果5：{result5}") #结果5：[3,2]

# 对元组进行切片，从头开始，到尾结束，步长-2
my_tuple = (0,1,2,3,4,5,6)
result6 = my_tuple[::-2]
print(f"结果6：{result6}") #结果6：（6，4，2，0）
print(my_tuple) # (0,1,2,3,4,5,6) 可见原序列并未改变
```

案例：

```
"""
有字符串 "万过薪月,员序程马黑来,nohtyp学"
请得到 "黑马程序员"
"""
# 方法1，倒序字符串，切片取出
str1 = "万过薪月,员序程马黑来,nohtyp学"
result = my_str[::-1][9:14]
print(result)# 黑马程序员
# 方法2
my_str = str1[9:4:-1] # 一定不要忘记下标从0开始的
print(my_str) # 黑马程序员
# 方法3，split分隔"," replace替换"来"为空，倒序字符串
my_str = str1.split(",")[1]
print(my_str) # 员序程马黑来
my_str = my_str.replace("来","") # 员序程马黑
print(my_str[::-1]) # 黑马程序员
```

Set集合

通过特性来分析：

- 列表可修改，支持重复元素且有序
- 元组、字符串不可修改，支持重复元素且有序

局限性：他们都支持重复元素

而**集合**，最主要的特点就是：**不支持重复元素（自带去重功能）、并且内容无序。**

基本语法：

```
# 定义集合字面量
{元素，元素，.....，元素}
# 定义集合变量
变量名称 = {元素，元素，.....，元素}
# 定义空集合
变量名称 = set()
```

回忆一下：

- 列表使用：[]
- 元组使用：()
- 字符串使用：""
- 集合使用：{}

集合的常用操作—修改

首先，因为集合是无序的，所以集合不支持：下标索引访问；但是集合和列表是一样，是允许修改的。

- 添加新元素

语法：集合.add(元素) 将指定元素添加到集合内

结果：集合本身被修改，添加了新元素。

- 移除元素

语法：集合.remove(元素) 将指定元素，从集合内移除

结果：集合本身被修改，移除了元素。

- 从集合中随机取出元素

语法：集合.pop()，从集合中**随机地**取出一个元素

结果：会得到一个元素的结果，同时集合本身被修改，元素被移除。

- 清空集合

语法：集合.clear()

结果：清空集合，集合变为空集合。

```
# 定义集合
my_set = {25,"dihsd",True,25,00,"eghnaqi"}
print(my_set) # 会自动去重，并且每一次的输出结果的顺序是无序的，随机的

# 添加新元素
my_set.add("ppyy")
my_set.add(25) # 注意，25已经存在，所以会自动去重
print(my_set)

# 移除元素
my_set.remove(0)
print(my_set)

# 随机取出一个元素
elememt = my_set.pop()
print(f"取出了{elememt}，还剩下{my_set}")
```

```
# 清空集合
my_set.clear()
```

- 取出2个集合的差集

语法：集合1.difference(集合2)

功能：取出集合1和集合2的差集（集合1有而集合2没有的）

结果：得到一个新集合，集合1和集合2不变。

- 消除2个集合的差集

语法：集合1.difference_update(集合2)

功能：在集合1内，删除和集合2相同的元素

结果：集合1被修改，集合2不变

- 2个集合合并

语法：集合1.union(集合2)

功能：将集合1和集合2组合成新集合

结果：得到新集合，集合1和集合2不变

- 统计集合元素数量

语法：len(集合)

```
# 取出两个集合差集
set1 = {1,2,3}
set2 = {1,5,6}
set3 = set1.difference(set2)
print(set3) # 得到的差集，结果：{2,3}
print(set1) # 结果：{1,2,3} 不变
print(set2) # 结果：{1,5,6} 不变

# 消除两个元素的差集
set1 = {1,2,3}
set2 = {1,5,6}
set1.difference_update(set2)
print(set1) # 结果：{2, 3}
print(set2) # 结果：{1, 5, 6}

#并集
set1 = {1,2,3}
set2 = {1,5,6}
set3 = set1.union(set2)
print(set3) # {1, 2, 3, 5, 6} 注意：顺序仍然不确定的
print(set1) # {1, 2, 3}
print(set2) # {1, 5, 6}

# 统计集合元素数量
set1 = {1,2,3,4,1,2,3,4}
num = len(set1)
print(num) # 结果：4  会自动去重
```

集合的遍历（for）

集合不支持下标索引，所以不能用while循环，可以用for循环。

```
# 集合的遍历
set = {"ndjvk",233,5830,"hviffsnbkj"}
for ele in set:
    print(f"集合内的元素有：{ele}")

"""
输出：也是无序的
集合内的元素有：233
集合内的元素有：ndjvk
集合内的元素有：5830
集合内的元素有：hviffsnbkj
"""
```

集合常用功能总结：

编号	操作	作用
1	集合.add(元素)	向集合添加一个元素
2	集合.remove(元素)	移除集合内指定的元素
3	集合.pop()	从集合随机取出一个元素
4	集合.clear()	将集合清空
5	集合1.difference(集合2)	得到一个新集合，集合1 — 集合2（差集）
6	集合1.difference_update(集合2)	在集合1中删除几何2中存在的元素，集合1被修改，集合2不变
7	集合1.union(集合2)	得到一个新集合，集合1，2都不变
8	len(集合)	得到一个整数，记录集合的元素数量

总结集合特点：

- 可以容纳多个数据
- 可以容纳不同类型的数据
- **数据是无序存储的（不支持下标索引）**
- **不允许重复数据的存在**
- 可以修改（增加或删除元素等）
- 支持for循环，不支持while循环

案例：

- 定义一个空集合
- 通过for循环遍历列表
- 在for循环中将列表的元素添加至集合
- 最终得到元素去重后的集合对象，并打印输出

```
my_list = ['黑马程序员',"传智教育",'黑马程序员',"传智教
育",'itheima','itcast','itheima','itcast','best']
my_set = set() # set()定义空集合
for element in my_list:
    my_set.add(element)

print(my_set) # {'itheima', 'itcast', '传智教育', '黑马程序员', 'best'}  无序的
```

Dict（字典、映射）

字典的定义，同样使用：{}，不过存储的元素是一个个的：键值对，语法如下：

```
# 定义字典字面量
{key:value, key:vlue, ......, key:value}
# 定义字典变量
my_dict = {key:value, key:vlue, ......, key:value}
# 定义空字典
my_dict = {}            # 方式1
my_dict = dict()        # 方式2
```

```
# 定义字典
my_dict1 = {"小明":99, "小红":88, "小强":90}
# 定义空字典
my_dict2 = {}
my_dict3 = dict()
```

字典的key不允许重复，虽不会报错，但会覆盖

字典同集合一样，不可以使用下标索引；

但是字典可以通过key值来取得对应的value。

语法：**字典[key]**

```
stu_score = {"小明":99, "小红":88, "小强":90}
print(stu_score["小明"])      #结果：99
print(stu_score["小红"])      #结果：88
print(stu_score["小强"])      #结果：90
```

字典的嵌套

字典的key和value也可以是任意数据类型（key不可为字典）

那么，就表明，字典是可以嵌套的。

例：学生各科的考试成绩

姓名	语文	数学	英语
小明	77	66	33
小红	88	86	55
小强	99	96	66

```
# 定义嵌套字典
stu_score_dict = {
    "小明":{            # value为字典
        "语文":77,
        "数学":66,
        "英语":33
    },
    "小红":{
        "语文":88,
        "数学":86,
        "英语":55

    },
    "小强":{
        "语文":99,
        "数学":96,
        "英语":66
    }
}

print(f"学生的考试信息为：{stu_score_dict}")
# 从嵌套字典中获取数据
# 查小红的语文信息
ch_score = stu_score_dict["小红"]["语文"]
print(f"小红的语文分数是：{ch_score}")
```

字典的常用操作

- 新增元素

语法：**字典[key] = value**，结果：字典被修改，新增了元素

- 更新元素

语法：**字典[key] = value**，结果：字典被修改，元素被更新

注意：这是对已经存在的key进行的操作

总结：key不存在时，新增元素；key存在时，更新元素

- 删除元素

语法：**字典.pop(key)**，结果：获得指定key的value，同时字典被修改，指定key的数据被删除

- 清空字典

语法：**字典.clear()**，结果：字典被修改，元素被清空

- 获取全部key

语法：**字典.keys()**，结果：得到字典中的全部key

- 字典的遍历

for循环：方式1，方式2

- 统计字典的元素数量

语法：**len(字典)**

```python
my_dict = {"小明":99, "小红":88, "小强":90}
# 新增元素
my_dict["周杰伦"] = 66
print(f"字典经过新增元素后，结果是：{my_dict}")
# 字典经过新增元素后，结果是：{'小明': 99, '小红': 88, '小强': 90, '周杰伦': 66}

# 更新元素
my_dict["小明"] = 50
print(f"字典经过更新后，结果是：{my_dict}")
#字典经过更新后，结果是：{'小明': 50, '小红': 88, '小强': 90, '周杰伦': 66}

# 删除元素
score = my_dict.pop("小明")
print(f"字典移除一个元素后，结果是：{my_dict},小明的考试分数是：{score}")
字典移除一个元素后，结果是：{'小红': 88, '小强': 90, '周杰伦': 66},小明的考试分数是：50

# 清空元素
my_dict.clear()
print(f"字典被清空了，内容是：{my_dict}")
# 字典被清空了，内容是：{}

# 获取全部key
my_dict = {"小明":99, "小红":88, "小强":90}
keys = my_dict.keys()
print(f"字典全部的key是：{keys}")
# 字典全部的key是：dict_keys(['小明', '小红', '小强'])

# 遍历字典
# 方式1，通过全部的key来完成遍历
for key in keys:
    print(f"字典的key是：{key}")
    print(f"字典的value是：{my_dict[key]}")

# 方式2，直接对字典进行for循环，每一次循环都是直接得到key
for key in my_dict:
    print(f"2字典的key是：{key}")
    print(f"2字典的value是：{my_dict[key]}")

# 统计字典中的元素数量
num = len(my_dict)
print(f"字典中的元素数量：{num}个")
```

编号	操作	作用
1	字典[key]	获取指定的key对应的value
2	字典[key] = value	添加或更新键值对
3	字典.pop(key)	取出key对应的value并在字典内删除此key的键值对
4	字典.clear()	清空字典
5	字典.keys()	获取字典全部的key，可用于for循环遍历字典
6	len(字典)	计算字典内的元素数量

总结字典的特点：

- 可以容纳多个数据
- 可以容纳不同类型的数据
- 每一份数据是keyvalue键值对
- 可以通过key获取到value，key不可以重复（重复会覆盖）
- 不支持下标索引
- 可以修改（增加或删除更新元素）
- 支持for循环，不支持while循环

数据容器的总结对比：

- 是都支持下标索引：
  - 支持：列表、元组、字符串 ——序列类型
  - 不支持：集合、字典——非序列类型
- 是否支持重复元素：
  - 支持：列表、元组、字符串——序列类型
  - 不支持：集合、字典——非序列类型
- 是否可以修改：
  - 支持：列表、集合、字典
  - 不支持：元组、字符串

	列表	元组	字符串	集合	字典
元素数量	支持多个	支持多个	支持多个	支持多个	支持多个
元素类型	任意	任意	仅字符	任意	Key：Value Key：除字典外任意类型 Value：任意类型
下标索引	支持	支持	支持	不支持	不支持
重复元素	支持	支持	支持	不支持	不支持
可修改性	支持	不支持	不支持	支持	支持
数据有序	是	是	是	否	否
使用场景	可修改、可重复的一批数据记录场景	不可修改、可重复的一批数据记录场景	一串字符的记录场景	不可重复的数据记录场景	以Key检索Value的数据记录场景

基于各类数据容器的特点，它们应用场景：

- 列表：一批数据，可修改、可重复的存储场景
- 元组：一批数据，不可修改、可重复的存储场景
- 字符串：一串字符串的存储场景
- 集合：一批数据，去重存储场景
- 字典：一批数据，可用Key检索Value的存储场景

数据容器的通用操作：

通用操作遍历：

- 5类数据容器都支持for循环遍历
- 列表、元组、字符串支持while循环，集合、字典不支持（无法索引下标）

除此之外，

- len(容器)：统计容器的元素个数
- max(容器)：统计容器的最大元素
- min(容器)：统计容器的最小元素

通用操作转换：

- list(容器)：将给定容器转换为列表
- str(容器)：将给定容器转换为字符串
- tuple(容器)：将给定容器转换为元组
- set(容器)：将给定容器转换为集合

通用排序功能：

- sorted(容器)：升序，排序后会转成列表，字典排序会丢失value
- sorted(容器，reverse=True)：降序

ASCII码表

在程序中，字符串所用的所有字符如：

- 大小写英文字母
- 数字
- 特殊符号（！、\、|、@、#、空格等）

都有其对应的ASCII码表值

字符串如何比较？

从头开始，一位位进行比较，其中一位大，后面就无需比较了。

单个字符如何比较？

通过ASCII码表。