

# 第4章

## Python序列数据

### 本章重点内容：

- 字符串类型
- 字符串函数
- 列表类型
- 元组类型
- 字典类型
- 字典与函数
- 实践项目：我的英文字典。
- 练习4。



微课 4-1  
字符串类型

字符串类型

PPT

## 4.1 字符串类型

### 4.1.1 教学目标

字符串是程序中常用的一种数据类型，字符串可以包含中文与英文等任何字符，在内存中用 Unicode 编码存储，但是存储到磁盘中时往往采用 GBK 或者 UTF-8 等别的编码形式，本节目标是掌握字符串的操作。

### 4.1.2 字符串类型的使用

字符数组可以用来存储字符串，字符串在内存中的存放形式也就是字符数组的形式，字符串可以看成是字符的数组，例如：

```
s="Hello";
```

其内存分布如图 4-1-1 所示。

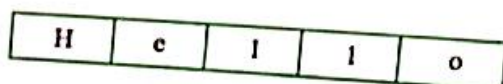


图 4-1-1

#### 1. 获取字符串长度函数 len

实际上字符串  $s$  的长度为  $\text{len}(s)$ ，例如：

```
len("abc")    #3
len("我们 abc") #5
```

注意空字符串  $s=""$  是连续两个引号，中间没有任何东西，空串的长度为 0， $\text{len}(s)=0$ ，但是  $s=" "$  包含一个空格， $s$  不是空串，长度为 1。

#### 2. 读出字符串各个字符

要得到其中第  $i$  个字符，可以像数组访问数组元素那样用  $s[i]$  得到，其中  $s[0]$  是第 1 个字符， $s[1]$  是第 2 个字符，……， $s[\text{len}(s)-1]$  是最后一个字符。例如：

```
s="a 我们"
n=len(s)
for i in range(n):
    print(s[i])
```





微课 4-1  
字符串类型

字符串类型

PPT

## 4.1 字符串类型

### 4.1.1 教学目标

字符串是程序中常用的一种数据类型，字符串可以包含中文与英文等任何字符，在内存中用 Unicode 编码存储，但是存储到磁盘中时往往采用 GB 或者 UTF-8 等别的编码形式，本节目标是掌握字符串的操作。

### 4.1.2 字符串类型的使用

字符数组可以用来存储字符串，字符串在内存中的存放形式也就是字符数组的形式，字符串可以看成是字符的数组，例如：

```
s="Hello";
```

其内存分布如图 4-1-1 所示。

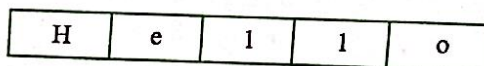


图 4-1-1

#### 1. 获取字符串长度函数 len

实际上字符串  $s$  的长度为  $\text{len}(s)$ ，例如：

```
len("abc")    #3
len("我们 abc") #5
```

注意空字符串  $s=""$  是连续两个引号，中间没有任何东西，空串的长度为 0， $\text{len}(s)=0$ ，但是  $s=" "$  包含一个空格， $s$  不是空串，长度为 1。

#### 2. 读出字符串各个字符

要得到其中第  $i$  个字符，可以像数组访问数组元素那样用  $s[i]$  得到，其中  $s[0]$  是第 1 个字符， $s[1]$  是第 2 个字符，……， $s[\text{len}(s)-1]$  是最后一个字符。例如：

```
s="a 我们"
n=len(s)
for i in range(n):
    print(s[i])
```



结果:

```
a
我
们
```

注意字符串中的字符是不可以改变的, 因此不能对某个字符 `s[i]` 赋值, 如 `s[0]='h'` 是错误的。

### 3. 字符在内存中的编码

计算机只能识别二进制数, 字符在计算机中实际上是用二进制数存储的, 该编码称为 Unicode, 每个英文字符用两个字节存储。要想知道某个字符的编码, 使用函数 `ord(字符)` 就可以了, 例如:

```
s="Hi,你好"
n=len(s)
for i in range(n):
    print(s[i],ord(s[i]))
```

结果:

```
H 72
i 105
, 65292
你 20320
好 22909
```

可以看到 “H” 的 Unicode 码是 72, “你” 的是 20320。可以用程序测试出 “A” ~ “Z”, “a” ~ “z” 字母的 Unicode 码:

```
S="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
s="abcdefghijklmnopqrstuvwxyz"
n=len(s)
for i in range(n):
    print(s[i],"---",ord(s[i]),S[i],"---",ord(S[i]))
```

同样可以测试 “0” ~ “9” 的 Unicode 码:

```
s="0123456789"
n=len(s)
for i in range(n):
    print(s[i],"---",ord(s[i]))
```



汉字的编码也是用两个字节才能表示, Unicode 码包含所有的符号。它表示英文字符时有一个字节是 0, 这样表示虽然浪费一个字节, 但是它把所有符号都统一成一样的, 因此还是划算的。

#### 4. 编码转为字符

如果知道一个符号的编码为  $n$ , 那么可以用 `chr(n)` 函数把它转为一个符号, 例如:

```
a=chr(25105)
b=chr(20204)
c=chr(119)
d=chr(101)
print(a,b,c,d)
```

结果:

我们 we

#### 5. 字符串的大小比较

两个字符串  $a$ 、 $b$  可以比较大小, 比较规则是按各个对应字符的 Unicode 编码, 编码大的一个为大。

比较  $a[0]$  与  $b[0]$ , 如果  $a[0]>b[0]$  则  $a>b$ , 如果  $a[0]<b[0]$  则  $a<b$ , 如果  $a[0]=b[0]$  则继续比较  $a[1]$  与  $b[1]$ 。

比较  $a[1]$  与  $b[1]$ , 如果  $a[1]>b[1]$  则  $a>b$ , 如果  $a[1]<b[1]$  则  $a<b$ , 如果  $a[1]=b[1]$  则继续比较  $a[2]$  与  $b[2]$ 。

.....

这个过程一直进行下去, 直到比较出大小, 如果比较完毕两个字符串的每个字符都一样, 那么如果两个字符串一样长  $\text{len}(a)=\text{len}(b)$ , 那么  $a=b$ ; 如果  $\text{len}(a)>\text{len}(b)$  则  $a>b$ ; 如果  $\text{len}(a)<\text{len}(b)$  则  $a<b$ 。

编写一个比较函数 `compare(a,b)` 比较  $a$ 、 $b$  大小, 如果  $a>b$  返回 1, 如果  $a<b$  返回 -1, 如果  $a=b$  返回 0, 那么 `compare` 是这样工作的:

```
def compare(a,b):
    m=len(a)
    n=len(b)
    if m<n:
        k=m
    else:
        k=n
```



```

for i in range(k):
    if a[i]>b[i]:
        return 1
    elif a[i]<b[i]:
        return -1
if m==n:
    return 0
elif m>n:
    return 1
else:
    return -1

```

在实际应用中可以简单使用 ==、>=、<=、>、<、!= 等符号来判断两个字符串的关系。

根据字符的编码存在下列关系：

"0"<"1"<...<"9"<"A"<"B"<...<"Z"<"a"<"b"<...<"z"<汉字

特别注意的是，大写字母比小写字母小！

例 4-1-1 输入一个字符串，统计它包含的大写字母的个数。

```

s=input("Enter a string: ")
count=0
for i in range(len(s)):
    if s[i]>="A" and s[i]<="Z":
        count=count+1
print("count=",count)

```

例 4-1-2 输入一个字符串，统计它包含的大写字母、小写字母、数字的个数。

```

s=input("Enter a string: ")
upper=0
lower=0
digit=0
for i in range(len(s)):
    if s[i]>="A" and s[i]<="Z":
        upper=upper+1
    elif s[i]>="a" and s[i]<="z":

```



```

        lower=lower+1
    elif s[i]>="0" and s[i]<="9":
        digit=digit+1
    print("Upper chars: ",upper)
    print("Lower chars: ",lower)
    print("Digit chars: ",digit)

```

**例 4-1-3** 输入一个字符串, 把它反向显示。

函数 reverseA 与 reverseB 都可以反向显示字符串:

```

def reverseA(s):
    t=""
    for i in range(len(s)-1,-1,-1):
        t=t+s[i]
    return t

def reverseB(s):
    t=""
    for i in range(0,len(s)):
        t=s[i]+t
    return t

print(reverseA("reverse"))
print(reverseB("reverse"))

```

**例 4-1-4** 输入一个字符串, 去掉它左右多余的空格, 如 " a bc " 返回 "a bc"。

```

def trim(s):
    t=""
    i=0
    j=len(s)-1
    while i<=j and s[i]==" ":
        i=i+1
    while i<=j and s[j]==" ":
        j=j-1
    for k in range(i,j+1):
        t=t+s[k]
    return t

```



```
s=input("Enter a string: ")
print(s,"length=",len(s))
t=trim(s)
print(t,"length=",len(t))
```

## 6. 英文字母的大小写转换

如果  $c$  是一个大写英文字母, 那么  $\text{ord}(c)$  是它的编码,  $\text{ord}(c) - \text{ord}("A")$  是它相对 "A" 的偏移量,  $\text{ord}("a")$  是 "a" 的编码, 显然  $\text{ord}("a") + \text{ord}(c) - \text{ord}("A")$  是  $c$  对应的小写字母的编码, 因此  $\text{chr}(\text{ord}("a") + \text{ord}(c) - \text{ord}("A"))$  就是  $c$  对应的小写字母。例如:

```
c="P"
d=chr(ord("a")+ord(c)-ord("A"))
print(d)
```

那么  $d$  是小写 "p"。

同样如果  $c$  是一个小写的英文字母, 那么  $\text{chr}(\text{ord}("A") + \text{ord}(c) - \text{ord}("a"))$  是它对应的大写字母。

**例 4-1-5** 编写把一个字符串中所有小写字母变成大写字母的函数。

```
def myToUpper(s):
    t=""
    for i in range(len(s)):
        if s[i]>="a" and s[i]<="z":
            t=t+chr(ord("A")+ord(s[i])-ord("a"))
        else:
            t=t+s[i]
    return t

def myToLower(s):
    t=""
    for i in range(len(s)):
        if s[i]>="A" and s[i]<="Z":
            t=t+chr(ord("a")+ord(s[i])-ord("A"))
        else:
            t=t+s[i]
    return t
```



```
s=input("Enter a string: ")
print("myToUpper: ",myToUpper(s))
print("myToLower: ",myToLower(s))
```

## 7. 字符串中的子串

设置  $s$  为字符串变量, 在  $s$  中取出它的一截, 即一个子串, 这是常用的一个操作。

例 4-1-6 设计函数 `subString(s, start, length)` 表示从  $s$  的 `start` 位置开始, 取出长度为 `length` 的一个子串。

```
def subString(s, start, length):
    m=len(s)
    if start>=length:
        return ""
    t=""
    i=start
    while i<start+length and i<m:
        t=t+s[i]
        i=i+1
    return t

s="abcdefghijk"
print(subString(s,2,4))
print(subString(s,2,24))
```

### 4.1.3 【案例】字符串的对称

#### 1. 案例描述

设计程序判断一个字符串是否对称。

#### 2. 案例分析

方法 1: 编写一个函数 `reverse(s)` 把字符串  $s$  反向, 然后把反向的结果与原来的字符串比较, 如果一样就说明是对称的。

方法 2: 有一种判别对称的方法, 用  $i$ 、 $j$  表示左右的下标, 逐步比较  $(s[0], s[\text{len}(s)-1])$ ,  $(s[1], s[\text{len}(s)-2])$ ,  $\dots$ , 如果有不相等的则一定不对称, 如果全部比较完毕都相等则对称。



### 3. 案例代码

方法1:

```
def reverse(s):  
    t=""  
    for i in range(len(s)-1,-1,-1):  
        t=t+s[i]  
    return t
```

```
def isSymmetry(s):  
    t=reverse(s)  
    if s==t:  
        return 1  
    else:  
        return 0
```

```
s=input("Enter a string: ")  
if isSymmetry(s)==1:  
    print("对称")  
else:  
    print("不对称")
```

方法2:

```
def isSymmetry(s):  
    i=0  
    j=len(s)-1  
    while i<=j:  
        if s[i]!=s[j]:  
            return 0  
        i=i+1  
        j=j-1  
    return 1
```

```
s=input("Enter a string: ")  
if isSymmetry(s)==1:  
    print("对称")  
else:  
    print("不对称")
```



## 4.2 字符串函数



微课 4-2  
字符串函数

PPT 字符串函数

PPT

### 4.2.1 教学目标

字符串的操作有很多函数,如查找一个字符串  $s$  是否包含另外一个字符串,是字符串运算与查找中经常使用的。本节目标是掌握这些常用的字符串函数。

### 4.2.2 字符串函数的使用

#### 1. 字符串的子串 $string[start:end:step]$

字符串中的子串规则与列表中的切片规则完全一样,只是字符串切片后返回一个新的字符串,原来字符串不变。

$start$ 、 $end$ 、 $step$  可选,冒号必须要有,基本含义是从  $start$  开始 (包括  $string[start]$ ),以  $step$  为步长,获取到  $end$  的一段元素 (注意不包括  $string[end]$ )。

如果  $step=1$ ,那么就是  $string[start], string[start+1], \dots, string[end-2], string[end-1]$ ,如果  $step>1$ ,那么第一为  $string[start]$ ,第二为  $string[start+step]$ ,第三为  $string[start+2*step]$ ,...以此类推,最后一个为  $string[m]$ ,其中  $m<end$ ,但是  $m+step \geq end$ 。即索引的变化是从  $start$  开始,按  $step$  跳跃变化,不断增大,但是不等于  $end$ ,也不超过  $end$ 。

如果  $end$  超过了最后一个元素的索引,那么最多取到最后一个元素。

$start$  不指定则默认为 0,  $end$  不指定则默认为序列尾,  $step$  不指定则默认为 1。

$step$  为正数则索引是增加的,索引沿正方向变化;如果  $step<0$ ,那么索引是减少的,按负方向变化。

不能使用  $step=0$ ,否则索引就原地踏步不变了。

如果  $start$ 、 $end$  为负数,表示倒数的索引,例如  $start=-1$ ,则表示  $len(string)-1$ ,  $start=-2$ ,表示  $len(string)-2$ 。

例如:

```
s = "abcdefghijk"
print("s---", s)
print("s[0:2]---", s[0:2])
print("s[:2]---", s[:2])
print("s[2:]---", s[2:])
print("s[2,6]---", s[2:6])
print("s[:]---", s[:])
```



```

print("s[:,2]---",s[:,2])
print("s[0:7:2]---",s[0:7:2])
print("s[8:14]---",s[8:14])
print("s[1:5:2]---",s[1:5:2])
print("s[1:4:2]---",s[1:4:2])

```

结果:

```

s--- abcdefghijk
s[0:2]--- ab
s[:2]--- ab
s[2:]--- cdefghijk
s[2,6]--- cdef
s[:]--- abcdefghijk
s[:,2]--- acegik
s[0:7:2]--- aceg
s[8:14]--- ijk
s[1:5:2]--- bd
s[1:4:2]--- bd

```

例如:

```

s = "abcdefghijk"
print("s---",s)
print("s[0:-2]---",s[0:-2])
print("s[:-2]---",s[:-2])
print("s[-2:]---",s[-2:])
print("s[-2,6]---",s[-2:6])
print("s[:]---",s[:])
print("s[:, -2]---",s[:, -2])
print("s[7, -1: -1]---",s[7: -1: -1])
print("s[8:0. -1]---",s[8:0: -1])
print("s[5:1: -2]---",s[5:1: -2])
print("s[4:1. -2]---",s[4:1: -2])

```

结果:

```

s--- abcdefghijk
s[0:-2]--- abcdefghi

```



```

s[:-2]--- abcdefghi
s[-2:]--- jk
s[-2,6]---
s[:]--- abcdefghijk
s[:, -2]--- kigeca
s[7, -1:-1]---
s[8:0, -1])--- ihg fedcb
s[5:1:-2]--- fd
s[4:1, -2]--- ec

```

## 2. 字符串转大小写函数 upper()、lower()

格式: s.upper()

作用: 返回一个字符串, 把 s 中的所有小写字母转为大写字母。

格式: s.lower()

功能: 返回一个字符串, 把 s 中的所有大写字母转为小写字母。

例如:

```

s=" Python(version4.5)is easy"
print(s.upper())
print(s.lower())
print(s)

```

输出:

```

PYTHON(VERSION4.5)IS EASY
python(version4.5)is easy
Python(version4.5)is easy

```

注意 s 自己是不变化的, s.upper() 只是返回另外一个大写的新字符串。

## 3. 字符串查找函数 find(t)

格式: s.find(t)

作用: 返回在字符串 s 中查找 t 子串第一次出现的位置下标, 如不存在就返回 -1。

例如:

```

s="12abcabcab"
i=s.find("ab")
j=s.find("abd")
print(i,j)

```



输出:

2 -1

"ab"在s中出现两次,返回第一次出现的位置2,"abd"在s中不存在。

#### 4. 字符串查找函数 rfind(t)

格式: s.rfind(t)

作用: 返回在字符串s中查找t子串最后一次出现的位置下标,如不存在

就返回-1。

例如:

```
s="12abcbcab"
i=s.rfind("ab")
j=s.rfind("abd")
print(i,j)
```

输出:

8 -1

"ab"在s中出现两次,返回最后一次出现的位置8,"abd"在s中不存在。

rfind函数与find函数类似,只是rfind从右边开始找t,而find是从左边开始找。

#### 5. 字符串查找函数 index(t)

格式: s.index(t)

作用: 返回在字符串s中查找t子串第一次出现的位置下标,如不存在就

报错误。

例如:

```
s="12abcbcab"
i=s.index("ab")
j=s.index("abd")
print(i,j)
```

index函数与find函数功能完全一样,不同的是要找的子串不存在时,index会报错误,find只默默返回-1,find比index好用,建议使用find而不用index。

#### 6. 字符串判断函数 startswith(t)、endswith(t)

格式: s.startswith(t)



作用: 判断字符串  $s$  是否以子串  $t$  开始, 返回逻辑值。

格式:  $s.startswith(t)$

作用: 判断字符串  $s$  是否以子串  $t$  结束, 返回逻辑值。

例如:

```
s="12abcabcbab"
i=s.startswith("12a")
j=s.endswith("ab")
print(i,j)
```

结果:

```
True True
```

显然可以用 `find` 函数来编写功能与 `.startswith(t)` 一样的 `myStartsWith(s,t)` 函数:

```
def myStartsWith(s,t):
    i=s.find(t)
    if i==0:
        return True
    else:
        return False
```

`myStartsWith(s,t)` 函数与 `s.startswith(t)` 最大不同是前者是一般函数, 字符串  $s$  作为变量传入, 而后者 `startswith(t)` 是字符串对象自己的函数, 因此使用方法不同。

同样也可以编写功能与 `endswith(t)` 一样的函数 `myEndsWith(s,t)`:

```
def myEndsWith(s,t):
    i=s.rfind(t)
    if i>=0 and i==len(s)-len(t):
        return True
    else:
        return False
```

## 7. 字符串去掉空格函数 `lstrip()`、`rstrip()`、`strip()`

格式:  $s.lstrip()$

作用: 返回一个字符串, 去掉了  $s$  中左边的空格。



格式: `s.rstrip()`

作用: 返回一个字符串, 去掉了 `s` 中右边的空格。

格式: `s.strip()`

作用: 返回一个字符串, 去掉了 `s` 中左边与右边的空格, 等同 `s.lstrip()`。

`rstrip()`。

例如:

```
s=" ab x yz "
a=s.lstrip()
b=s.rstrip()
c=s.strip()
print(a,len(a))
print(b,len(b))
print(c,len(c))
print(s,len(s))
```

由此可见它们只是去掉左边或者右边的空格, 不去掉字符串中间包含的空格。

#### 8. 字符串分离函数 `split(sep)`

格式: `s.split(sep)`

作用: 用 `sep` 分割字符串 `s`, 分割出的部分组成列表返回。

其中 `sep` 是分隔符, 结果是字符串按 `sep` 字符串分割成多个字符串, 这些字符串组成一个列表, 即函数 `split` 调用后返回一个列表。例如:

```
s="I am learning Python"
w=s.split(" ")
print(w)
```

是把字符串 `s` 按空格分离开生成一个列表, 结果:

```
['I', 'am', 'learning', 'Python']
```

而程序:

```
s="I am learning Python"
w=s.split("ear")
print(w)
```

是按 "ear" 把字符串 `s` 分离开, 结果:

```
['I am l', 'ning Python']
```



程序:

```
s="abcbabcabc"
w=s.split("ab")
print(w)
```

是按" ab" 分离字符串, 结果:

```
['', 'c', 'c', 'c']
```

第一个元素是一个空字符串。

例 4-2-1 编写 myLower(s) 实现 s.lower()。

```
def myLower(s):
    t=""
    for i in range(len(s)):
        if s[i]>="A" and s[i]<="Z":
            t=t+chr(ord("a")+ord(s[i])-ord("A"))
        else:
            t=t+s[i]
    return t

s="aABEBwWFEW"
a=s.lower()
b=myLower(s)
print(a)
print(b)
```

结果发现 myLower(s) 功能与 s.lower() 一样。

例 4-2-2 编写 myStrip(s) 实现 s.strip()。

编写函数 myStrip(s), 在 s 的左边与右边找空格, 跳过这些空格。

```
def myStrip(s):
    i=0
    j=len(s)-1
    #i 是左边下标, 跳过左边空格
    while i<=j and s[i]==" ":
        i=i+1
    #j 是右边下标, 跳过右边空格
```



```

while j >= i and s[j] == " ":
    j = j + 1
return s[i:j+1]

```

```

s = " a b "
a = myStrip(s)
b = s.strip()
print(a, len(a))
print(b, len(b))

```

结果发现 `myStrip(s)` 功能与 `s.strip()` 一样。

#### 例 4-2-3 编写 `mySplit(s, sep)` 实现 `s.split(sep)`。

仔细分析 `split` 的工作过程。`s.split(sep)` 时首先它在字符串 `s` 找到 `sep` 的位置, 例如在 `i` 处, 然后把 `s[0:i]` 部分作为第一个元素 (不包含 `s[i]`), 之后从 `i` 开始跨过 `sep` 取出 `s[i+len(sep)]` 的后半部分, 再次进行同样的操作, 一直到找不到 `sep` 为止。

例如 `s = "abcabcabc"`, `s.split("ab")`, 设计 `t = []`, 在 `i = 0` 处找到 "ab", `s[0:0]` 是空串, 因此 `t = [""]`; 然后跨过 "ab" 后是 `s[2:]` 即 `s = "cabcbc"`, 再次找 "ab", `i = 1`, `s[0:1]` 为 "c", 因此 `t = ["", "c"]`; `s = s[1+2:]` 即 `s = "cabc"`, 再次找 "ab", `i = 1`, `s[0:1]` 为 "c", `t = ["", "c", "c"]`; `s = s[1+2:]` 即 "c", 在 `s = "c"` 中没有找到 "ab", 把 "c" 加入 `t`, 因此最后 `t = ["", "c", "c", "c"]`。

可以自己编写一个 `mySplit(s, sep)` 函数完成这个工作:

```

def mySplit(s, sep):
    i = s.find(sep)
    t = []
    while i >= 0:
        w = s[0:i]
        t.append(w)
        s = s[i+len(sep):]
        i = s.find(sep)
    t.append(s)
    return t

```

```

s = "abcababcbab"
a = s.split("ab")
b = mySplit(s, "ab")

```



```
print(a)
```

```
print(b)
```

结果发现与系统的 `s.split(sep)` 功能一样。

### 4.2.3 【案例】寻找字符串的子串

#### 1. 案例描述

编写 `myFind(s,t)` 实现 `s.find(t)`。

#### 2. 案例分析

设置 `s` 为字符串变量, 在 `s` 中查找是否包含子字符串 `t` 是一个常用的操作。例如 `s="I am testing"`, `t="am"`, 那么 `s` 包含 `t`; 如果 `t="test"`, `s` 也包含 `t`; 但是 `t="tested"` 时 `s` 不包含 `t`。设计函数 `index(s,t)` 测试 `s` 是否包含 `t`, 如果包含就返回 `t` 在 `s` 的开始下标位置, 否则返回 `-1`。

查找的方法是从 `s[i]` 开始, 把 `t` 与 `s[i]` 对齐, 查看 `s[i]` 与 `t[0]`, `s[i+1]` 与 `t[1]`, ..., `s[i+len(t)-1]` 与 `t[len(t)-1]` 是否相同, 如果都相同那么说明 `s` 包含 `t`, 否则就更换一个 `i` 再次比较。

#### 3. 案例代码

```
def myFind(s,t):
    m=len(s)
    n=len(t)
    if m<n:
        return -1
    i=0
    while i<=m-n:
        j=0
        while j<n:
            if s[i+j]!=t[j]:
                break
            j=j+1
        if j==n:
            return i
        i=i+1
    return -1
```

```
s="ababcabcdl2ab"
```

```
print(myFind(s,"abc"),s.find("abc"))
```

```
print(myFind(s,"ad"),s.find("ad"))
```



结果发现 `myFind(s,t)` 功能与 `s.find(t)` 一样。

## 4.3 列表类型

### 4.3.1 教学目标

字符串或者数值列表是程序中常用的数据类型，例如使用一个字符串列表存储全国的省份名称，使用一个数值列表存储全班学生的成绩等。本节目标是掌握这种列表数据的使用。

### 4.3.2 列表类型的使用

列表是 Python 中最基本的数据结构，列表是常用的 Python 数据类型，列表的数据项不需要具有相同的类型。列表中的每个元素都分配一个数字——它的位置或索引，第 1 个索引是 0，第 2 个索引是 1，依此类推。序列都可以进行的操作包括索引、切片、加、乘、检查成员。此外，Python 已经内置确定序列的长度以及确定最大和最小元素的方法。

#### 1. 创建一个列表

只要把逗号分隔的不同的数据项使用方括号括起来即可，例如：

```
list1 = ['physics', 'chemistry', 'math', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 4, 2]
```

列表的元素可以重复，如 `list2` 中的 2、4 都重复出现，列表中的元素类型不一定要完全一样，如 `list1` 中有字符串也有数值。

列表类型是 Python 中的 `list` 类实例，例如：

```
list=['a','b','c','d']
print(list)
print(type(list))
```

输出结果：

```
['a', 'b', 'c', 'd']
<class 'list'>
```

其中 `type(list)` 返回的类型是一个名称为 `list` 的类。

#### 2. 访问列表中的值

使用下标索引来访问列表中的值，同样也可以使用方括号的形式截取字



微课 4-3  
列表类型

PPT 列表类型

PPT



符, 截取的方法与字符串中截取的类似, 例如:

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7]
print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
```

输出结果:

```
list1[0]: physics
list2[1:5]: [2, 3, 4, 5]
```

### 3. 更新列表

可以对列表的数据项进行修改或更新, 也可以使用 `append()` 方法来添加列表项, 例如:

```
list = ['physics', 'chemistry', 1997, 2000]
print ("Value available at index 2 : ")
print (list[2])
list[2] = 2001
print ("New value available at index 2 : ")
print (list[2])
```

输出结果:

```
Value available at index 2 :
1997
New value available at index 2 :
2001
```

### 4. 删除列表元素

可以使用 `del` 语句来删除列表的元素, 例如:

```
list1 = ['physics', 'chemistry', 1997, 2000, 2017]
print (list1)
del list1[2]
print ("After deleting value at index 2 : ")
print (list1)
```

输出结果:



```
['physics', 'chemistry', 1997, 2000, 2017]
After deleting value at index 2 :
['physics', 'chemistry', 2000, 2017]
```

### 5. 列表操作的联合

可以使用 "+" 来连接多个列表，例如：

```
list1=["a","b"]
list2=["c","a"]
list3=list1+list2
print(list3)
```

输出结果：

```
['a', 'b', 'c', 'a']
```

### 6. 列表的截取 L[start:end:step]

start、end、step 可选，冒号必须要有，基本含义是从 start 开始（包括 L[start]），以 step 为步长，获取到 end 的一段元素（注意不包括 L[end]）。

如果 step=1，那么就是 L[start], L[start+1], ..., L[end-2], L[end-1]，

如果 step>1，那么第一为 L[start]，第二为 L[start+step]，第三为 L[start+2\*step], ..., 以此类推，最后一个为 L[m]，其中 m<end，但是 m+step>=end。即索引的变化是从 start 开始，按 step 跳跃变化，不断增大，但是不等于 end，也不超过 end。

如果 end 超过了最后一个元素的索引，那么最多取到最后一个元素。

start 不指定则默认值为 0，end 不指定则默认值为序列尾，step 不指定则默认值为 1。

step 为正数则索引是增加的，索引沿正方向变化；如果为 step<0，则索引是减少的，沿负方向变化。

不能使用 step=0，否则索引就原地踏步不变了。

如果 start、end 为负数，表示倒数的索引，例如 start=-1；则表示 len(L)-1；start=-2，表示 len(L)-2。

#### 例 4-3-1 列表的截取。

```
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
L[0:2] # [1, 2], 取区间 [i, j), 左闭右开
L[:2] # 同上, 可省略第一位
L[2:] # [3, 4, 5, 6, 7, 8, 9, 10, 11]
```



```

L[2:-1]  #[3, 4, 5, 6, 7, 8, 9, 10]
L[:]     #同 11, 相当于复制一份
L[::2]   #步长 2, [1, 3, 5, 7, 9, 11]
L[0:7:2] #[1, 3, 5, 7]
L[7:0:-2] #[8, 6, 4, 2] 注意步长为负、理解起来相当于从 7 到 1, 倒序步长 2
L[8:14]  #[9, 10, 11] 注意 end 超过最后的索引

```

#### 例 4-3-2 列表的截取。

```

L = ["a0", "a1", "a2", "a3", "a4", "a5", "a6", "a7", "a8", "a9"]
print("L---", L)
print("L[0:-2]---", L[0:-2])
print("L[:-2]---", L[:-2])
print("L[-2:]---", L[-2:])
print("L[-2,6]---", L[-2:6])
print("L[:]---", L[:])
print("L[:, -2]---", L[:, -2])
print("L[7, -1:-1]---", L[7:-1:-1])
print("L[8:0. -1]---", L[8:0:-1])
print("L[5:1:-2]---", L[5:1:-2])
print("L[4:1. -2]---", L[4:1:-2])

```

输出结果:

```

L--- ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 'a8', 'a9']
L[0:-2]--- ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']
L[:-2]--- ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']
L[-2:]--- ['a8', 'a9']
L[-2,6]--- []
L[:]--- ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 'a8', 'a9']
L[:, -2]--- ['a9', 'a7', 'a5', 'a3', 'a1']
L[7, -1:-1]--- []
L[8:0. -1]--- ['a8', 'a7', 'a6', 'a5', 'a4', 'a3', 'a2', 'a1']
L[5:1:-2]--- ['a5', 'a3']
L[4:1. -2]--- ['a4', 'a2']

```

#### 7. 判断一个元素是否在列表中

使用 in 或者 not in 操作判断一个元素是否在或者不在列表中, 例如:



```
list=['a','b','c','d']  
print('a' in list)  
print('A' in list)  
print('A' not in list)
```

输出结果:

```
True  
False  
True
```

其中 a 在列表中, 但是 A 不在列表中。

### 4.3.3 列表常用操作函数

#### 1. list.append(obj)

作用: 在列表末尾添加新的对象。

以下实例展示了 append() 函数的使用方法:

```
aList = [123, 'xyz', 'zara', 'abc']  
aList.append(2009)  
print("Updated List : ", aList)
```

以上实例输出结果如下:

```
Updated List : [123, 'xyz', 'zara', 'abc', 2009]
```

#### 2. list.count(obj)

作用: 统计某个元素在列表中出现的次数。

以下实例展示了 count() 函数的使用方法:

```
aList = [123, 'xyz', 'zara', 'abc', 123]  
print("Count for 123 : ", aList.count(123))  
print("Count for zara : ", aList.count('zara'))  
print("Count for abc : ", aList.count('abc'))
```

以上实例输出结果如下:

```
Count for 123 : 2  
Count for zara : 1  
Count for abc : 0
```



## 3. list.extend(seq)

作用：在列表末尾一次性追加另一个序列中的多个值（用新列表扩展来的列表）。

以下实例展示了 extend() 函数的使用方法：

```
aList = [123, 'xyz', 'zara', 'abc', 123]
bList = [2009, 'manni']
aList.extend(bList)
print ("Extended List : ", aList)
```

以上实例输出结果如下：

```
Extended List : [123, 'xyz', 'zara', 'abc', 123, 2009, 'manni']
```

## 4. list.index(obj)

作用：从列表中找出某个值第一个匹配项的索引位置。

以下实例展示了 index() 函数的使用方法：

```
aList = [123, 'xyz', 'zara', 'abc']
print ("Index for xyz : ", aList.index('xyz'))
print ("Index for zara : ", aList.index('zara'))
```

以上实例输出结果如下：

```
Index for xyz : 1
Index for zara : 2
```

注意如果元素不在列表中，那么会提示错误：

```
print ("Index for abc: ", aList.index('abc')) #错误!
```

## 5. list.insert(index, obj)

作用：将对象插入列表。

以下实例展示了 insert() 函数的使用方法：

```
aList = [123, 'xyz', 'zara', 'abc']
aList.insert(3, 2009)
print ("Final List : ", aList)
```

以上实例输出结果如下：



```
final List : [123, 'xyz', 'zara', 2009, 'abc']
```

#### 6. list.remove(obj)

作用：移除列表中某个值的第一个匹配项。

以下实例展示了 remove() 函数的使用方法：

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.remove('xyz')
print("List : ", aList)
aList.remove('abc')
print("List : ", aList)
```

以上实例输出结果如下：

```
List : [123, 'zara', 'abc', 'xyz']
List : [123, 'zara', 'xyz']
```

注意如果要删除的元素不在列表中就会提示错误：

```
aList.remove('abcd')    #错误！
```

#### 7. 删除元素 del list[index]

如果要删除某个指定索引 index 的元素，那么可以采用：

```
del list[index]
```

例如：

```
aList = [123, 'xyz', 'zara', 'abc']
del aList[2]
print(aList)
```

输出结果：

```
[123, 'xyz', 'abc']
```

#### 8. 弹出元素 list.pop(index=-1)

弹出元素与删除元素一样，都是从列表中移除一个元素项。如果要弹出某个指定索引 index 的元素，那么可以采用：

```
list.pop(index)
```



index 的默认值是-1, 使用 `list.pop()` 即弹出最后一个元素。

例如:

```
list=['a','b','c','d']
list.pop()
print(list)
list.pop(0)
print(list)
```

输出结果:

```
['a','b','c']
['b','c']
```

### 9. list.reverse()

作用: 反向列表中元素。

注意反向后原来列表的元素顺序改变了, 以下实例展示了 `reverse()` 函数的使用方法:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.reverse()
print ("List : ", aList)
```

以上实例输出结果如下:

```
List :  ['xyz', 'abc', 'zara', 'xyz', 123]
```

### 10. list.sort()

作用: 对原列表进行排序。

注意排序后原来列表的元素顺序改变了, 以下实例展示了 `sort()` 函数的使用方法:

```
aList = ['123', 'xyz', 'zara', 'abc', 'xyz']
aList.sort()
print ("List : ", aList)
```

以上实例输出结果如下:

```
List :  ['123', 'abc', 'xyz', 'xyz', 'zara']
```

注意: 要对列表的元素进行排序, 这些元素必须是同类型的, 如全部为字



字符串，或者全部为数值，保证它们两两能进行大小比较。如果类型是混合的，则不能进行排序，例如：

```
aList = [1,6,3,2,"a"]
aList.sort()
print("List :", aList)
```

结果错误：

```
TypeError: unorderable types: str() < int()
```

#### 4.3.4 列表与函数

列表作为函数参数，如果在函数中改变了列表，那么调用处的列表也同时被改变。也就是说调用处的实际参数与函数的形式参数是同一个变量，这一点与普通的整数、浮点数、字符串变量不同。

例 4-3-3 列表作为函数参数。

```
def fun(mylist,m,s):
    "修改传入的列表"
    mylist.append(1);
    m=1
    s="changed"
    print("函数内取值:", mylist,m,s)
```

#调用 fun 函数

```
mylist = [10,20,30];
m=0
s="try"
fun(mylist,m,s);
print("函数外取值:", mylist,m,s)
```

输出结果：

```
函数内取值: [10, 20, 30, 1] 1 changed
函数外取值: [10, 20, 30, 1] 0 try
```

可以看到 mylist 发生了改变，但是整数 m 与字符串 s 没有变化。

例 4-3-4 函数返回列表。



```
def fun():
    list=[]
    for i in range(10):
        list.append(i)
    return list
```

#调用 fun 函数

```
list=fun()
print(list)
```

输出结果:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

列表是一个变量对象, 函数可以返回一个列表。

### 4.3.5 【案例】列表存储省份与城市

#### 1. 案例描述

使用列表 provinces 存储部分省份名称, 再使用另外一个列表 cities 存储对应省份的城市, 实现省份与城市的查找。

#### 2. 案例分析

provinces 与 cities 设计如下:

```
provinces=["广东","四川","贵州"]
cities=[[ "广州","深圳","惠州","珠海"],[ "成都","内江","乐山"],[ "贵阳","六盘水","遵义"]]
```

这样一个序号 index 下 provinces [index] 为省份, 而 cities [index] 是这个省份的城市, 也是一个列表。

#### 3. 案例代码

##### (1) 输入省份查找城市

```
provinces=["广东","四川","贵州"]
cities=[[ "广州","深圳","惠州","珠海"],[ "成都","内江","乐山"],[ "贵阳","六盘水","遵义"]]
```

```
p=input("输入省份:")
found=False
for i in range(len(provinces)):
    if provinces[i]==p:
```



```

print(provinces[i],end=":")
for j in range(len(cities[i])):
    print(cities[i][j],end=" ")
found=True
break
if not found:
    print("没有这个省份")

```

程序中 cities 是一个二维的列表，即 cities 是一个列表，它的每个元素也是一个列表。

## (2) 输入城市查找省份

```

provinces=["广东","四川","贵州"]
cities=[[ "广州","深圳","惠州","珠海"],["成都","内江","乐山"],["贵阳","六盘水","遵义"]]

def search(c):
    for i in range(len(cities)):
        for x in cities[i]:
            if x==c:
                print(c,"在",provinces[i]+"省")
                return
    print("没有查到")

c = input("输入城市:")
search(c)

```

程序中设计 search(c) 函数查找 c 城市的省份，第 1 个 i 循环遍历所有 cities 中的元素；第 2 个循环中 cities[i] 又是一个列表，在此循环中查找城市，如果查找成功那么 provinces[i] 就是该城市所在的省份。

## 4.4 元组类型

### 4.4.1 教学目标

在程序中求一组值的最大值或者最小值时常用的操作，例如：

```

def max(a,b):
    return a if a>b else b

```



微课 4-4  
元组类型

PPT 元组类型

PPT



可以计算两个值的最大值,但3个参数的最大值就无法计算,能不能设计一个 max 函数计算任意多个数的最大值呢?

教学目标是掌握元组的使用,最后设计一个这样通用的最大值函数,在调用时可以指定任意多个数,都能找出这些数的最大值。

#### 4.4.2 元组类型的使用

元组也是 Python 中常用的一种数据类型,它是 tuple 类的类型,与列表 list 几乎相似,区别如下。

① 元组数据使用圆括号()来表示,如 `t=('a','b','c')`。

② 元组数据的元素不能改变,只能读取。

因此可以简单理解元组就是只读的列表,除了不能改变外其他特性与列表完全一样。

##### 例 4-4-1 元组的使用。

```
s=(1,3,2,3,4,5)
print(s)
print(type(s))
```

结果:

```
(1, 3, 2, 3, 4, 5)
<class 'tuple'>
```

例 4-4-2 建立一个代表星期的元组表,输入一个 0~6 的整数,输出对应的星期名称。

```
week=('日','一','二','三','四','五','六')
print(week)
w=input("Enter an integer: ")
w=int(w)
if w>=0 and w<=6:
    print("星期"+week[w])
else:
    print("错误输入")
```

结果:

```
('日', '一', '二', '三', '四', '五', '六')
Enter an integer: 3
星期三
```



如果在函数参数的末尾使用"\*"参数,那么该参数是可以变化的,一般标注为\*args 参数,在函数中成为一个元组,注意这样的\*args 的参数必须放在函数参数的末尾。

例 4-4-3 元组可变参数的函数。

```
def fun(x,y,*args):
    print(x,y)
    print(args)

fun(1,2)
fun(1,2,3)
fun(1,2,3,4)
```

结果:

```
1 2
()
1 2
(3,)
1 2
(3, 4)
```

其中 args 就是一个可变参数,它根据实际的调用成一个元组,fun(1,2) 时 x=1,y=2,而 args=(),但是 fun(1,2,3)时 x=1,y=2,args=(3,)。

显然不能设计成 def fun(x,\*args,y),或者 def fun(\*args,x,y),不然调用 fun(1,2,3)时不确定到底 args 应该是多少个参数的元组。

### 4.4.3 【案例】通用的最大值函数

#### 1. 案例描述

设计一个通用的最大值函数 max,可以用来计算出任意个数的最大值。

#### 2. 案例分析

函数设计成带任意参数\*args 的形式:

```
def max(*args)
```

那么就可以带任意参数调用了,例如:

```
print(max(1,2))
print(max(1,2,3,4))
```



### 3. 案例代码

```
def max(*args):
    print(args)
    m=args[0]
    for i in range(len(args)):
        if m<args[i]:
            m=args[i]
    return m

print(max(1,2))
print(max(1,2,0,3))
```

结果:

```
(1, 2)
2
(1, 2, 0, 3)
3
```

由此可见, 在调用 `max(1,2)` 时 1、2 都传递给 `args` 参数, `args=(1,2)` 成为一个元组, 同样 `max(1,2,0,3)` 使 `args=(1,2,0,3)` 成为一个元组。

## 4.5 字典类型

### 4.5.1 教学目标

在程序中经常碰到键值对的问题, 即给定一个键值 `key`, 那么它对应的值 `value` 是什么? 例如一个学生的姓名 (`key`) 是什么 (`value`), 性别 (`key`) 是什么 (`value`)。本节目标是掌握这种字典的应用, 实现用列表与字典存储一组学生的信息, 方便查找。

### 4.5.2 字典类型的使用

字典是另一种可变容器模型, 且可存储任意类型对象, 字典的每个键值 (`key=>value`) 对用冒号 (:) 分割, 每个对之间用逗号 (,) 分割, 整个字典包括在花括号 {} 中, 格式如下:

```
d = {key1 : value1, key2 : value2 }
```



微课 4-5  
字典类型

PPT 字典类型

PPT



键必须是唯一的，但值则不必。值可以取任何数据类型，但键必须是不变的，如字符串、数字或元组。一个简单的字典实例：

```
dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
print(type(dict))
```

结果：

```
<class 'dict'>
```

由此可见字典类型是一个类名称为 dict 的对象类型。

### 1. 访问字典里的值

把相应的键放入熟悉的方括弧，如下实例：

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print ("dict['Name']: ", dict['Name'])
print ("dict['Age']: ", dict['Age'])
```

以上实例输出结果：

```
dict['Name']: Zara
dict['Age']: 7
```

如果用字典里没有键的访问数据，会输出错误如下：

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print ("dict['Alice']: ", dict['Alice'])
```

以上实例输出结果：

```
dict['Zara']:
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print "dict['Alice']: ", dict['Alice']
KeyError: 'Alice'
```

### 2. 修改字典

向字典添加新内容的方法是增加新的键/值对，修改或删除已有键/值对。

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
```

如果一个键值已经存在，那么可以修改它的值



```
dict['Age'] = 8
```

如果一个键值不存在,那么可以增加

```
dict['School'] = "DPS School"
print("dict['Age']: ", dict['Age'])
print("dict['School']: ", dict['School'])
```

以上实例输出结果:

```
dict['Age']: 8
dict['School']: DPS School
```

### 3. 删除字典元素

删除一个字典用 del 命令,如下实例:

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name'] #删除键是 Name 的条目
dict.clear()    #清空字典所有条目
del dict        #删除字典
```

### 4. 字典键的特性

字典值可以没有限制地取任何 python 对象,既可以是标准的对象,也可以是用户定义的,但键不行,两个重要的点需要记住:

① 不允许同一个键出现两次,创建时如果同一个键被赋值两次,后一个值会被记住,如下实例:

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print ("dict['Name']: ", dict['Name'])
```

以上实例输出结果:

```
dict['Name']: Manni
```

② 键必须不可变,可以用数字、字符串或元组充当,所以用列表就不行,如下实例:

```
dict = [{'Name': 'Zara', 'Age': 7}]
print ("dict['Name']: ", dict['Name'])
```

以上实例输出结果:



```
Traceback (most recent call last):  
File "test.py", line 3, in <module>  
dict = {'Name': 'Zara', 'Age': 7}  
TypeError: list objects are unhashable
```

5. 函数得到字典的长度 `len(dict)`  
以下实例展示了 `len()` 函数的使用方法:

```
dict = {'Name': 'Zara', 'Age': 7}  
print ("Length : ", len(dict))
```

以上实例输出结果为:

```
Length : 2
```

6. 删除字典 `dict` 的所有元素 `dict.clear()`  
以下实例展示了 `clear()` 函数的使用方法:

```
dict = {'Name': 'Zara', 'Age': 7}  
print ("Start Len : ", len(dict))  
dict.clear()  
print "End Len : ", len(dict))
```

以上实例输出结果为:

```
Start Len : 2  
End Len : 0
```

7. 获取字典的所有键值函数 `dict.keys()`  
Python 字典 `keys()` 函数以列表返回一个字典所有的键, 以下实例展示了 `keys()` 函数的使用方法:

```
dict = {'Name': 'Zara', 'Age': 7}  
print ("keys : ", dict.keys())
```

以上实例输出结果为:

```
keys : ['Age', 'Name']
```

8. `dict.get(key, default=None)`  
Python 字典 `get()` 函数返回指定键的值, 如果值不在字典中返回默认值



None 或者指定的值, 以下实例展示了 get() 函数的使用方法:

```
dict = {'Name': 'Zara', 'Age': 27}
print ("Value : % s" % dict.get('Age'))
print ("Value : % s" % dict.get('Sex', "Never"))
```

以上实例输出结果为:

```
Value : 27
Value : Never
```

### 4.5.3 【案例】字典存储学生信息

#### 1. 案例描述

使用列表与字典存储学生信息, 方便查找, 学生信息包括的姓名、性别、年龄。

#### 2. 案例分析

一个学生的信息是字典对象, 例如:

```
{"Name": "张三", "Gender": "男", "Age": 20}
```

设计一个列表 st=[], 它存储多个学生, 每个列表元素是一个学生字典对象, 例如:

```
st=[{"Name": "张三", "Gender": "男", "Age": 20}, {"Name": "张四", "Gender": "女", "Age": 20}]
```

#### 3. 案例代码

```
st=[]
def getStudents():
    global st
    st=[]
    st.append({"Name": "张三", "Gender": "男", "Age": 20})
    st.append({"Name": "李四", "Gender": "女", "Age": 21})
    st.append({"Name": "王五", "Gender": "男", "Age": 22})

def seekStudent(Name):
    for s in st:
        if s["Name"] == Name:
```



```

print(s["Name"], s["Gender"], s["Age"])
return
print("没有姓名是", Name, "的学生")

getStudents()
seekStudent("张三")
seekStudent("张四")

```

结果:

张三 男 20  
没有姓名是 张四 的学生

## 4.6 字典与函数

### 4.6.1 教学目标

Python 的数据类型是非常灵活的, 字典是非常常用的一种类型, 字典可以作为函数参数, 函数也可以返回一个字典。本节目标是设计一个程序存储省份与其所辖城市的信息, 实现查询功能, 并借此掌握字典在函数中的应用。

### 4.6.2 字典与函数的使用

#### 1. 字典作为函数参数

字典作为函数参数, 如果在函数中改变了字典, 那么调用处的字典也同时被改变。也就是说调用处的实际参数与函数的形式参数是同一个变量, 这一点与普通的整数、浮点数、字符串变量不同。

例 4-6-1 字典作为函数参数。

```

def fun(dict):
    dict["name"] = "aaa"
    print("inside:", dict)

dict = {"name": "xxx", "age": 30}
print("before", dict)
fun(dict)
print("after", dict);

```

PPT 字典与函数

PPT



结果:

```
before {'name': 'xxx', 'age': 30}
inside: {'name': 'aaa', 'age': 30}
after {'name': 'aaa', 'age': 30}
```

由此可见, dict 在函数中变化后, 在主程序中也变化了。

## 2. 函数返回字典

字典可以作为函数返回值返回。

例 4-6-2 字典作为函数返回值。

```
def fun():
    dict={}
    dict["name"]="aaa"
    dict["age"]=20
    dict["gender"]="male"
    return dict

def show(dict):
    keys=dict.keys()
    for key in keys:
        print(key,dict[key])

dict=fun()
print(dict)
show(dict)
```

结果:

```
{'name': 'aaa', 'age': 20, 'gender': 'male'}
name aaa
age 20
gender male
```

## 4.6.3 字典与字典参数

Python 中除了用 "\*" 表示的元组可变参数外, 还有一种是 "\*\*\*" 表示的字典可变参数, 一般标识为 \*\*kargs, 这种 kargs 在函数中是一个字典, 在调用时实际参数按 key=value 的键值对方式提供参数。



## 例4-6-3 具有字典可变参数的函数。

```
def fun(x, y=2, **kargs):
    print(x, y)
    print(kargs)

fun(1, 2)
fun(1, 2, z=3)
fun(1, 2, a=3, b="demo")
fun(x=1, y=2, z=3)
fun(y=1, x=2, z=5, s="demo")
fun(x=1, z=3)
```

结果:

```
1 2
{}
1 2
{'z': 3}
1 2
{'a': 3, 'b': 'demo'}
1 2
{'z': 3}
2 1
{'z': 5, 's': 'demo'}
1 2
{'z': 3}
```

由此可见, 在调用时 `fun(1, 2, a=3, b="demo")` 使得 `kargs = {'a': 3, 'b': 'demo'}` 变成一个字典。

注意如果函数有 `*args` 及 `**kargs` 参数同时存在, 那么 `*args` 必须放在 `**kargs` 参数前面, 即函数最后两个参数是 `*args, **kargs`。

## 例4-6-4 具有元组可变参数与字典可变参数的函数。

```
def fun(x, y=2, *args, **kargs):
    print(x, y)
    print(args)
    print(kargs)
```



```
fun(1,2)
fun(1,2,3,4)
fun(1,2,3,4,z=5,s="demo")
```

结果:

```
1 2
()
{}
1 2
(3, 4)
{}
1 2
(3, 4)
{'z': 5, 's': 'demo'}
```

由于\*args的参数是位置参数,因此有\*args出现时,\*args前面的函数参数在调用时不能以关键字参数的方式出现,只能以位置参数的方式出现,例如下列是错误的调用:

```
fun(x=1,y=2,3,4)
```

#### 4.6.4 【案例】字典存储省份与城市

##### 1. 案例描述

设计一个程序存储省份与其所辖城市的信息,实现查询功能。

##### 2. 案例分析

设计字典 provinces 如下:

```
provinces={"广东":["广州","深圳"],"四川":["成都","内江","乐山"]}
```

字典 provinces 的 keys 是各个省的名称,一个省的值是一个列表,是它下辖的各个城市。

##### 3. 案例代码

```
#provinces 是全局的变量
provinces={}

def append(province,cities):
```



拓展案例



```
global provinces
if province not in provinces.keys():
    provinces[province]=cities
else:
    print(province+"已经存在")

def show():
    for p in provinces.keys():
        print(p,provinces[p])

def seekProvince(province):
    if province in provinces.keys():
        print(province,end=":")
        for c in provinces[province]:
            print(c,end=" ")
        print()
    else:
        print("没有这个省份")

def seekCity(city):
    for p in provinces.keys():
        if city in provinces[p]:
            print(city+"属于"+p+"省")
            return
    print("没有这个城市")

append("广东",["广州","深圳"])
append("四川",["成都","内江","乐山"])
append("贵州",["贵阳","六盘水","兴义"])
show()
seekProvince("四川")
seekCity("六盘水")
```

程序结果:

```
广东 ['广州', '深圳']
四川 ['成都', '内江', '乐山']
贵州 ['贵阳', '六盘水', '兴义']
四川:成都 内江 乐山
六盘水属于贵州省
```



## 4.7 实践项目：我的英文字典

### 4.7.1 项目目标



微课 4-6  
我的英文字典

实现一个简单的英语字典查询与管理程序。一个英文单词包含单词与单词的注释，结构如下：

```
words=[{"word":"about","note":"在附近,关于"}, {"word":"post",
"note":"邮寄、投递"}]
```

所有的单词组成一个列表，每个单词与注释成为一个字典，程序的功能就是管理这样一组单词记录，程序有查找单词、增加单词、更新注释、删除单词、显示单词等功能。

程序运行的效果如下：

1. 显示 2. 查找 3. 增加 4. 更新 5. 删除 6. 退出

请选择(1,2,3,4,5):1

about. :在附近,关于

post :邮寄、投递

### 4.7.2 项目设计

#### 1. 单词存储

数据使用全局变量 `words = []` 存储，它是一个列表，每个元素是一个字典，字典是单词与注释的信息。

#### 2. 单词查找

为了加快查找的速度，人们把单词按字典顺序从小到大排列，查找时采用二分法查找。

二分法查找是一种高效的查找方法，在 `words` 中查找单词 `w`，主要思想如下：

① 设置 `i=0`, `j=len(words)-1`，即 `i`、`j` 是第一与最后一个下标。

② 如果 `i<=j` 就计算 `m=(i+j)//2`，`m` 是中间一个下标，如果 `i>j` 程序结束。

③ 如果 `words[m]["word"] == w["word"]`，那么说明 `words[m]` 就是要找的单词，`m` 就是这个单词在列表中的位置。

④ 如果 `words[m]["word"] > w["word"]`，说明 `word[m]` 这个单词比要找的单词大，由于是从小到大排序的，因此设置 `j=m-1`，构造 `[i,m-1]` 范围。



到②继续查找。

⑤ 如果 `words[m]["word"] < w["word"]`，说明 `word[m]` 这个单词比要查找的单词小，由于是从小到大排序的，因此设置 `i = m + 1`，构造 `[m + 1, j]` 范围回到②继续查找。

⑥ 如果全部查找完毕没有找到单词，那么这个单词是新的单词，它应该放在 `words[i]` 的位置。

查找函数 `seek` 如下：

```
def seek(word):
    i=0
    j=len(words)-1
    while i<=j:
        m=(i+j)//2
        if words[m]["word"] == word:
            print("% -16s : % s" % (word, words[m]["note"]))
            return
        elif words[m]["word"]>word:
            j=m-1
        else:
            i=m+1
    print(word + " ---查找失败")
```

### 3. 插入单词

这是根据二分法查找思想设计的插入函数，把新的单词插入到 `words[i]` 的位置：

```
def insert(w):
    global words
    i=0
    j=len(words)-1
    while i<=j:
        m=(i+j)//2
        if words[m]["word"] == w["word"]:
            print(w["word"]+" ---已经存在")
            return
        elif words[m]["word"]>w["word"]:
            j=m-1
```



```

else:
    i=m+1
words.insert(i,w)
print(w["word"] + " ---增加成功")

```

在单词更新与删除中也采用二分法查找单词。

### 4.7.3 项目实践

```

words=[{"word":"about","note":"在附近,关于"}, {"word":"post",
note":"邮寄、投递"}]

def show():
    for w in words:
        print("%-16s : %s" % (w["word"],w["note"]))
    print()

def enter():
    w={}
    w["word"]=input("单词:")
    w["note"]=input("注释:")
    return w

def seek(word):
    i=0
    j=len(words)-1
    while i<=j:
        m=(i+j)//2
        if words[m]["word"] == word:
            print("%-16s : %s" % (word, words[m]["note"]))
            return
        elif words[m]["word"]>word:
            j=m-1
        else:
            i=m+1
    print(word + " --- 查找失败")

def insert(w):
    global words

```



```
i=0
j=len(words)-1
while i<=j:
    m=(i+j)//2
    if words[m]["word"] == w["word"]:
        print(w["word"]+" --- 已经存在")
        return
    elif words[m]["word"]>w["word"]:
        j=m-1
    else:
        i=m+1
words.insert(i,w)
print(w["word"] + " --- 增加成功")
```

```
def update(w):
    global words
    i=0
    j=len(words)-1
    while i<=j:
        m=(i+j)//2
        if words[m]["word"] == w["word"]:
            words["note"]=w["note"]
            print(w["word"]+" --- 更新成功")
            return
        elif words[m]["word"]>w["word"]:
            j=m-1
        else:
            i=m+1
    print(w["word"] + " --- 查找失败")
```

```
def delete(word):
    global words
    i=0
    j=len(words)-1
    while i<=j:
        m=(i+j)//2
        if words[m]["word"] == word:
            del words[m]
```



```

        print(word+" --- 删除成功")
        return
    elif words[m]["word"]>word:
        j=m-1
    else:
        i=m+1
    print(word + " --- 查找失败")

while True:
    print("1. 显示 2. 查找 3. 增加 4. 更新 5. 删除 6. 退出")
    s=input("请选择(1,2,3,4,5):")
    if s=="1":
        show()
    elif s=="2":
        word = input("单词:")
        seek(word)
    elif s=="3":
        w=enter()
        insert(w)
    elif s=="4":
        w=enter()
        update(w)
    elif s=="5":
        word=input("单词:")
        delete(word)
    elif s=="6":
        break
print("Finished")

```

主程序部分是一个无限循环，只有选择 6 后才退出并结束，选择 1、2、3、4、5 分别执行显示、查找、增加、更新、删除的操作。

## 练习 4

1. 能直接修改字符串的某个字符吗？例如 `s="abc"`，`s[0]="1"` 可以吗？
2. 输入一个字符串，输出它所包含的所有数字，例如输入 "23me3e"，输



出"233"。

3. 设计一个字符串函数 `reverse(s)`，它返回字符串 `s` 的反串，例如 `reverse("abc")` 返回 `"cba"`。

4. 元组与列表有什么不同？

5. 一个列表中的元素类型要求一致吗？例如 `list=[1, "a"]` 是正确的吗？

6. 列表是否还可以嵌套别的列表？列举一个例子说明。

7. 用一个字典描述一个日期，包含年 `year`、月 `month`、日 `day` 的键字。

8. Python 的字典数据类型与 JSON 数据类有很多相似的地方，说明有哪些共同点。

9. 写出下列程序执行的结果：

```
d={"students":[{"name":"A","sex":"M"}, {"name":"B","sex":"C"}]}
for k1 in d.keys():
    for k2 in d[k1]:
        for k3 in k2.keys():
            print(k3, k2[k3])
```

10. 如果使用字典描述一个时间，例如 `t={"hour":12, "minute":23, "second":34}` 表示时间 `"12:23:34"`，设计一个函数 `interval(t1,t2)`，计算时间 `t1` 与 `t2` 的时间差，返回相同结构的一个字典时间。