3.字符串、列表、元组、字典

3.1 字符串

Python的核心数据类型

String（字符串）

- Python中的字符串可以使用单引号 、双引号和三引号（三个单引号或三个双引号）括起来，使用反斜杠 \ 转义特殊字符
- Python3源码文件默认以UTF-8编码，所有字符串都是unicode字符串
- 支持字符串拼接、截取等多种运算

```
>>> a = "Hello"
>>> b = "Python"
>>> print("a + b 输出结果：", a + b)
a + b 输出结果： HelloPython
>>> print("a[1:4] 输出结果：", a[1:4])
a[1:4] 输出结果： ell
```

示例：

```
word = '字符串'
sentence = "这是一个句子。"
paragraph = """这是一个段落，
可以由多行组成"""
```

3.1.1 单引号和双引号如何选择?

1、包含单引号的字符串

假如你想定义一个字符串my_str，其值为： I'm a student，则可以采用如下方式，通过转义字符 \ 进行定义。

```
my_str = 'I\'m a student'
```

也可以不使用转义字符，利用双引号直接进行定义。

```
my_str = "I'm a student"
```

2、包含双引号的字符串

假如你想定义一个字符串my_str，其值为： Jason said "I like you"，则可以采用如下方式，通过转义字符 \ 进行定义。

```
my_str = "Jason said \"I like you\""
```

也可以不使用转义字符，利用单引号直接进行定义。

```
my_str = 'Jason said "I like you"'
```

3、不想让转移字符生效，可以使用r字符放在字符串前面

```
path = "c:\\temp\\1.mp3"          #如果路径太长不方便
path = r"c:\temp\1.mp3"           #不想让转义字符生效
print(path)
```

3.1.2 转义字符

转义字符	描述
(在行尾时)	续行符
****	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，**yy** 代表的字符，例如：**\o12** 代表换行，其中 o 是字母，不是数字 0。
\xyy	十六进制数，yy代表的字符，例如：\x0a代表换行
\other	其它的字符以普通格式输出

3.1.3 字符串的截取和连接

3.1.3.1 字符串截取

```
str1 = "itsishu"
#正向索引：
#       0123456
#反向索引：
#     -7...-3,-2 ,-1
```

```
print(len(str1))                                    # 内置函数 len() 获取字符串长度
print(str1)                                         # 打印字符串
print(f"{str1}[2]:",str1[2])                        # 获取字符串中的第二个字符
print(f"{str1}[0:2]:",str1[0:2])                    # 截取字符串索引值为0~1的字符，不包括索引值为2的字符  [0,2)  左闭右开区间。
print(f"{str1}[2:5]:",str1[2:5])                    # 截取字符串索引值为2~4的字符，不包括索引值为5的字符  [2,5)  左闭右开区间。
print(f"{str1}[2:-1]:",str1[2:-1])                   # 截取字符串重索引值为2开始直到字符串结尾的前一个，-1的索引值表示最后一个
print(f"{str1}[2:len(str1)]:",str1[2:len(str1)])    # 截取字符串索引值2~8，最后一个字符的索引值为7，所以刚刚好能截取到字符串末尾

# 截取在列表中索引值为0-4的数据，冒号前面不设置参数，默认从0开始
print(f"{str1}[:4]:",str1[:4])
# 截取在列表中索引值为2-末尾的数据，冒号后面不设置参数，默认截取到最后一位数据
print(f"{str1}[2:]:",str1[2:])

print(f"{str1}[0:7:2]:",str1[0:7:2])                # [起始位置：结束位置：间隔值]

print(f"{str1}[-1:-6:-1]:",str1[-1:-6:-1])          #反向索引，倒序打印
```

3.1.3.2 字符串拼接

```
str1 = "IT私塾。"
str1 += "学以致用"
print(str1 + "高效学习")

print('-' * 30)

str2 = "这是一个很长的字符串，以至于一行都写不完" \
       "那就第二行接着写" \
       "可以一直写下去，都算一个字符串"
print(str2)
```

3.1.4 字符串的常见操作

序号	方法及描述
1	capitalize() 将字符串的第一个字符转换为大写
2	center(width, fillchar) 返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格。
3	count(str, beg= 0,end=len(string)) 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数
4	**bytes.decode(encoding="utf-8", errors="strict")** Python3 中没有 decode 方法，但我们可以使用 bytes 对象的 decode() 方法来解码给定的 bytes 对象，这个 bytes 对象可以由 str.encode() 来编码返回。
5	**encode(encoding='UTF-8',errors='strict')** 以 encoding 指定的编码格式编码字符串，如果出错默认报一个ValueError 的异常，除非 errors 指定的是'ignore'或者'replace'
6	endswith(suffix, beg=0, end=len(string)) 检查字符串是否以 obj 结束，如果beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False.
7	expandtabs(tabsize=8) 把字符串 string 中的 tab 符号转为空格，tab 符号默认的空格数是 8 。
8	find(str, beg=0, end=len(string)) 检测 str 是否包含在字符串中，如果指定范围 beg 和 end ，则检查是否包含在指定范围内，如果包含返回开始的索引值，否则返回-1
9	index(str, beg=0, end=len(string)) 跟find()方法一样，只不过如果str不在字符串中会报一个异常.
10	**isalnum()** 如果字符串至少有一个字符并且所有字符都是字母或数字则返 回 True,否则返回 False
11	**isalpha()** 如果字符串至少有一个字符并且所有字符都是字母则返回 True, 否则返回 False
12	**isdigit()** 如果字符串只包含数字则返回 True 否则返回 False..
13	islower() 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False
14	**isnumeric()** 如果字符串中只包含数字字符，则返回 True，否则返回 False
15	isspace() 如果字符串中只包含空白，则返回 True，否则返回 False.
16	istitle() 如果字符串是标题化的(见 title())则返回 True，否则返回 False
17	isupper() 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False
18	**join(seq)** 以指定字符串作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串
19	**len(string)** 返回字符串长度
20	[ljust(width, fillchar]) 返回一个原字符串左对齐,并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格。
21	lower() 转换字符串中所有大写字符为小写.

序号	方法及描述
22	**lstrip()** 截掉字符串左边的空格或指定字符。
23	maketrans() 创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串表示转换的目标。
24	max(str) 返回字符串 str 中最大的字母。
25	min(str) 返回字符串 str 中最小的字母。
26	[replace(old, new , max]) 把 将字符串中的 str1 替换成 str2,如果 max 指定，则替换不超过 max 次。
27	rfind(str, beg=0,end=len(string)) 类似于 find()函数，不过是从右边开始查找.
28	rindex(str, beg=0, end=len(string)) 类似于 index()，不过是从右边开始.
29	[rjust(width,, fillchar]) 返回一个原字符串右对齐,并使用fillchar(默认空格）填充至长度 width 的新字符串
30	**rstrip()** 删除字符串字符串末尾的空格.
31	**split(str="", num=string.count(str)) num=string.count(str))** 以 str 为分隔符截取字符串，如果 num 有指定值，则仅截取 num+1 个子字符串
32	[splitlines(keepends]) 按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。
33	startswith(substr, beg=0,end=len(string)) 检查字符串是否是以指定子字符串 substr 开头，是则返回 True，否则返回 False。如果beg 和 end 指定值，则在指定范围内检查。
34	[strip(chars]) 在字符串上执行 lstrip()和 rstrip()
35	swapcase() 将字符串中大写转换为小写，小写转换为大写
36	title() 返回"标题化"的字符串,就是说所有单词都是以大写开始，其余字母均为小写(见 istitle())
37	translate(table, deletechars="") 根据 str 给出的表(包含 256 个字符)转换 string 的字符, 要过滤掉的字符放到 deletechars 参数中
38	upper() 转换字符串中的小写字母为大写
39	zfill (width) 返回长度为 width 的字符串，原字符串右对齐，前面填充0
40	isdecimal() 检查字符串是否只包含十进制字符，如果是返回 true，否则返回 false。

3.1.4.1 字母转换

```
# capitalize 字符串首字母大写
str1 = "itsishu"
res = str1.capitalize()
print(res)                                  #输出:Itsishu

# title 每个单词的首字母大写 (非字母隔开的单词)
```

```
str1 = "It si shu"
res = str1.title()
print(res)                                  #输出:It Si Shu

res = str1.istitle()                        #判断每个单词的首字母是否大写
print(res)                                  #输出:False

#upper 将所有字母变成大写
str1 = "It Si Shu"
res = str1.upper()
print(res)                                  #输出:IT SI SHU

# lower 将所有字母变成小写
res = str1.lower()
print(res)                                  #输出:it si shu

# swapcase 大小写互换
res = str1.swapcase()
print(res)                                  #输出:iI sI sHU
```

3.1.4.2 统计与查找

```
#count 统计字符串中某个元素的数量
str1 = "it si shu"
res = str1.count("s")                #s为2；it为1；IT为0
print(res)


#find 查找某个字符串第一次出现的索引位置
#find('字符串',开始位置,结束位置) 结束位置取不到,取到之前的一个值
str1 = "IT私塾：高效学习，学以致用"
res = str1.find("学")                 # 7，标点符号也算字符
res = str1.find("学",8)               # 10
res = str1.find("学",1,8)             #-1表示没有匹配到  [1,7) 左闭右开区间
res = str1.find("学习")               # 7
print(res)

#index 与 find 功能相同 find找不到返回-1,index找不到数据直接报错
res = str1.index("it")               # 推荐使用find
print(res)
```

3.1.4.3 判断开头、结尾

```
# startswith 判断是否以某个字符或字符串为开头
# startswith('字符串',开始位置,结束位置) 返回True或False

str1 = "IT私塾：高效学习，学以致用"
res= str1.startswith("IT")                  #True
res = str1.startswith("学",10)              #True
res = str1.startswith("学以",10,12)         #True
print(res)
```

```
# endswith 判断是否以某个字符或字符串结尾
# endswith('字符串',开始位置,结束位置) 返回True或False
res = str1.endswith("用")                    #True
res = str1.endswith("学习",-9,-5)            #True
print(str1[-9:-5])                           #高效学习
print(res)
```

3.1.4.4 分隔与拼接

```
str1 = "IT私塾 高效学习 学以致用"
res = str1.split()
print(res)                                   #['IT私塾', '高效学习', '学以致用']

str1 = "IT_私塾_高效_学习_学以_致用"
res = str1.split("_",2)                      # 第二个参数是分割几次 (从左向右)
print(res)                                   #输出：['IT', '私塾', '高效_学习_学以_致用']
#rsplit 从右向左分割
res = str1.rsplit("_",1) # (从右向左)
print(res)                                   # 返回列表，输出：['IT_私塾_高效_学习_学以',
'致用']
#join  按某字符将列表拼接成字符串(容器类型都可)
listvar  =  ['IT私塾',"高效学习","学以致用"]
res = "*".join(listvar)
print(res)                                   # 输出：IT私塾*高效学习*学以致用
```

3.1.4.5 去掉与替换

```
# replace 替换字符串(第三个参数选择替换的次数)
str1 = "IT私塾 高效学习 学以致用"
res  = str1.replace(" ","/")
print(res)                                   #输出：IT私塾/高效学习/学以致用
# 第三个参数为替换的次数
res = str1.replace("学","练",1)
print(res)                                   #输出：IT私塾 高效练习 学以致用

# strip  默认去掉首尾两边的空白符
str1 = "\r IT私塾   \t \n"
print(str1)                                  #输出：  IT私塾
res = str1.strip()
print(res)
print("end")                                 #输出：IT私塾
                                             #     end

#lstrip(),只去掉左侧的空白符
res = str1.lstrip()                          #输出：IT私塾
print(res)
print("end")                                 #     end

# rstrip(),只去掉右侧的空白符
res = str1.rstrip()
print(res)                                   #输出：  IT私塾
print("end")                                 #       end
```

3.1.4.6 判断类型

```
#isspace() 字符串中只包含空白，则返回 True，否则返回 False
res = "       "                          #True
res = "\t  \r\n"                         #True
print(res.isspace())


# isalpha 字符串至少有一个字符并且所有字符都是字母则返回 True, 否则返回 False
res = "abc"                              #True
# res = "123"                            #False
# res = "123.0"                          #False
res = "$123"                             #False
print(res.isalpha())



# isalnum,字符串至少有一个字符并且所有字符都是字母或数字则返回 True,否则返回 False
res = "abc"                              #True
res = "abc123"                           #True
res = "123.0"                            #False
res = "$123"                             #False
print(res.isalnum())




# isdecimal 检测字符串是否以数字组成  必须是纯数字
# isdigit() 字符串只包含数字则返回 True 否则返回 False
# isnumeric() 字符串中只包含数字字符，则返回 True，否则返回 False
res = "1234"                                #True
# res = "1234.0"                            #False
# res = "$1234"                             #False
print(res.isdecimal())


num = "1"  #unicode
print(f"{num} isdigit:",num.isdigit())      # True
print(f"{num} isdecimal:",num.isdecimal()) # True
print(f"{num} isnumeric:",num.isnumeric()) # True

#切换全角：win10 在输入法的”设置“中，”按键“，选择”全/半角切换”中的“Shift+空格
num = "１" # 全角
print(f"{num} isdigit:",num.isdigit())      # True
print(f"{num} isdecimal:",num.isdecimal()) # True
print(f"{num} isnumeric:",num.isnumeric()) # True

num = b"1" # byte
print(f"{num} isdigit:",num.isdigit())      # True
#print(f"{num} isdecimal:",num.isdecimal()) # AttributeError 'bytes' object has
no attribute 'isdecimal'
#print(f"{num} isnumeric:",num.isnumeric()) # AttributeError 'bytes' object has
no attribute 'isnumeric'

num = "Ⅳ" # 罗马数字4
```

```
print(f"{num} isdigit:",num.isdigit())      # False
print(f"{num} isdecimal:",num.isdecimal()) # False
print(f"{num} isnumeric:",num.isnumeric()) # True

num = "四" # 汉字4
print(f"{num} isdigit:",num.isdigit())      # False
print(f"{num} isdecimal:",num.isdecimal()) # False
print(f"{num} isnumeric:",num.isnumeric()) # True
```

isdigit()、isdecimal()、isnumeric()对比

isdigit()
True: Unicode数字，byte数字（单字节），全角数字（双字节）
False: 汉字数字
Error: 无

isdecimal()
True: Unicode数字，，全角数字（双字节）
False: 罗马数字，汉字数字
Error: byte数字（单字节）

isnumeric()
True: Unicode数字，全角数字（双字节），罗马数字，汉字数字
False: 无
Error: byte数字（单字节）

3.1.4.7 长度、填充

```
# len 计算容器类型长度
res = len("IT私塾")
print(res)                                      #输出：4

# center 填充字符串,原字符居中 (默认填充空格)
str1 = "IT私塾"
res = str1.center(10,"*")                       #center(填充的个数,填充的字符)
print(res)                                      #输出：***IT私塾***
```

3.1.4.8 编码和解码

```
# max 返回字符串中编码最大的字母
# min 返回字符串中编码最小的字母
str1 = "IT私塾"
print(max(str1))                            #输出：私
print(min(str1))                            #输出：I

# ord,内置函数，返回汉字或字母的Unicode编码
print("'I'的编码",ord("I"))                  #输出：'I'的编码 73
print("'T'的编码",ord("T"))                  #输出：'T'的编码 84
print("'私'的编码",ord("私"))                 #输出：'私'的编码 31169
print("'塾'的编码",ord("塾"))                 #输出：'塾'的编码 22654
```

```
#python3 默认是unicode，16位的编码
print(chr(22654))  #编码为65的字符          #输出：塾

str = "IT私塾"
str_utf8 = str.encode("UTF-8")
str_gbk = str.encode("GBK")
#b表示字节    2字符 = 2字节    1字节 = 8bit  比特位  比特 0或1
print(str)

print("UTF-8 编码：", str_utf8,type(str_utf8))
                          #输出：UTF-8 编码：  b'IT\xe7\xa7\x81\xe5\xa1\xbe' <class
'bytes'>
print("GBK 编码：", str_gbk,type(str_gbk))
                          #输出：GBK 编码：  b'IT\xcb\xbd\xdb\xd3' <class 'bytes'>
print("UTF-8 解码：", str_utf8.decode('UTF-8','strict'))    #输出：UTF-8 解码：  IT私
塾
print("GBK 解码：", str_gbk.decode('GBK','strict'))         #输出：GBK 解码：  IT私塾
```

字符集：

在计算机内部，所有的信息最终都表示为一个二进制的字符串。

字符集简单理解，就是将不同语言的文字用一个二进制的编码对应起来的编码库。

扩展：

什么是Unicode，与UTF-8，GBK，GB2312，GB18030有什么区别?

参考：https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6%E9%9B%86/946585

字符集的概念理解：

参考：https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6%E7%BC%96%E7%A0%81/8446880

参考：https://blog.csdn.net/qq_28098067/article/details/53486032

ASCII（American Standard Code for Information Interchange，美国信息互换标准编码）

ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符										ASCII 打印字符												
		0000					0001					0010		0011		0100		0101		0110		0111		
		0					1					2		3		4		5		6		7		
		+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	►	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH	头标开始	17	◄	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX	正文结束	19	‼	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	◆	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK	确认	22	▬	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ	^Back space

注：表中的ASCII字符可以用:ALT + "小键盘上的数字键" 输入

、

ASCII 字符代码表 二

高四位 / 低四位		扩充ASCII码字符集															
		1000		1001		1010		1011		1100		1101		1110		1111	
		8		9		A/10		B/16		C/32		D/48		E/64		F/80	
		+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符
0000	0	128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
0001	1	129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
0010	2	130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
0011	3	131	â	147	ô	163	ú	179	|	195	├	211	╙	227	π	243	≤
0100	4	132	ä	148	ö	164	ñ	180	┤	196	─	212	Ô	228	Σ	244	⌠
0101	5	133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
0110	6	134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	μ	246	÷
0111	7	135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
1000	8	136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	º
1001	9	137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	•
1010	A	138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
1011	B	139	ï	155	¢	171	½	187	╗	203	╦	219	▉	235	δ	251	√
1100	C	140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	n
1101	D	141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
1110	E	142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
1111	F	143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	BLANK FF

注：表中的ASCII字符可以用 ALT + "小键盘上的数字键" 输入

3.2 列表

Python的核心数据类型

List（列表）

- 列表可以完成大多数集合类的数据结构实现。列表中元素的类型可以不相同，它支持数字，字符串甚至可以包含列表（所谓嵌套）。
- 列表是写在方括号 [] 之间、用逗号分隔开的元素列表。
- 列表索引值以 0 为开始值，-1 为从末尾的开始位置。
- 列表可以使用+操作符进行拼接，使用*表示重复。

```
>>> list = ['abcd', 786 , 2.23, 'runoob', 70.2]
>>> print(list[1:3])
[786, 2.23]
>>> tinylist = [123, 'runoob']
>>> print(list + tinylist)
['abcd', 786, 2.23, 'runoob', 70.2, 123, 'runoob']
```

3.2.1 列表的定义与访问

3.2.1.1 列表的定义与创建

- **列表的格式**

  变量A的类型为列表

```
namelist = []                          #定义一个空的列表
print(namelist,type(namelist))         #输出：[] <class 'list'>
namelist = ["小张","小王","小李"]
```

比C语言的数组强大的地方在于列表中的元素可以是不同类型的

```
testList = [1, 'a', True, 2.3, 1]   #元素内容可以重复
```

通过list()创建列表

```
a = list()   #创建一个空的列表对象
print(a)
a = list(range(10))  #将range所定义的数据作为列表元素  [0,9)
a = list(range(2,-3,-1))
print(a)
a = list("IT私塾")    #将字符串的每个字符作为列表元素
```

通过推导式创建列表（了解）

```
a = [x*2 for x in range(5)]     #循环创建多个元素
```

3.2.1.2 列表元素的访问

----------操作名称----------	操作方法	举例
访问列表中的元素	通过下标直接访问	print(list1[0])
列表的切片	使用[: :]	list1[2:5:2]
遍历列表	通过for循环	for i in list1: print(i)

- **打印列表**

示例:

```
    namesList = ['xiaoWang','xiaoZhang','xiaoHua']
    print(namesList[0])
    print(namesList[1])
    print(namesList[2])
```

- **列表的循环遍历**

  为了更有效率的输出列表的每个数据，可以使用循环来完成

示例:

```
    namelist = ["小张","小王","小李"]
    for name in namesList:
        print(name)
```

3.2.2 常用操作

3.2.2.1 常用操作： len()、 max()、 min()、sum()

操作名称	操作方法	举例
获取列表长度	len（）	
获取列表元素最大值	max（）	
获取列表元素最小值	min（）	
其它类型对象转换成列表	list（）	

示例:

```
# （1）len()  获取列表元素个数
namelist = ["小张","小王","小李","小赵"]
print(len(namelist))    #4

i = 0
while i < len(namelist):
    print(namelist[i])
    i += 1
```

```
#（2）max() 和min() 获取列表中最大和最小的元素
list1 = [3,2,1]
list1= ["aa","bb","cc"]
#list1 = [False, 2.0, 30, "aa"]      #max和min函数调用时会报错，字符串和数值无法比较
list1 = [0, False,  2.0, 30]
print(max(list1))
print(min(list1))

#(3) sum() 对全部为数值型元素的列表求和
list1 = [10,20,30,2.0,True]
print(sum(list1))                    #输出：63.0
```

3.2.2.2 增/删/改/查

----------操作名称----------	操作方法	举例
【增】新增数据到列表尾部	使用append方法	list1.append(5)
【增】列表的追加	使用extend方法	list1.extend(list2)
【增】列表的加法操作	+	list3 = list1 + list2
【增】列表数据插入	insert方法	list1.insert(1, 3)
【删】列表的删除	del ： 我们通过索引删除指定位置的元素。 remove：移除列表中指定值的第一个匹配值。如果没找到的话，会抛异常。	del list1[0] list1.remove(1) 注意两种方法的区别
【删】弹出列表尾部元素	使用pop方法	list1.pop()
【改】更新列表中的数据	通过下标原地修改	list1[0] = 8
【查】列表成员关系	in 、not in	2 in list1
【查】指定区间某元素的索引	index方法	list.index("a",1,4)
【查】列表中元素出现次数	count方法	list.count("c")
【排】列表的排序	sort方法	list1.sort()
【排】列表的反转	reverse方法	list1.reverse()

示例:

【增】:

```
#增：    【append】追加

namelist = ["小张","小王","小李","小赵"]

print("-------增加前，名单列表的数据-------")
for name in  namelist:
    print(name,end=",")

nametemp = input("\n请输入学生姓名：")
namelist.append(nametemp)    #在末尾追加一个元素

print("-------增加后，名单列表的数据-------")
for name in  namelist:
    print(name,end=",")

#增：    【extend】扩展

a = [1,2]
b = [3,4]
# a.append(b)                #将列表当作一个元素，加入到a列表中
# print(a)

a.extend(b)                  #将b列表中的每个元素，逐一追加到a列表中
print(a)

#增：  【+】连接

list1 = [10,20,30]
list2 = [40,50]
res = list1 + list2
print(res,id(res))           #[10, 20, 30, 40, 50]
print(list1,id(list1))

list1.extend(list2)
print(list1,id(list1))

#增：【*】重复 （了解）

list1 = [1,2,3]
list2 = list1 * 3
print(list2)


#增：【insert】插入
a = [0,1,2]
a.insert(1,3)                #第一个变量表示下标，第二个表示元素（对象）
print(a)                     #指定下表位置插入元素
```

【删】：

```
movieName = ["加勒比海盗","骇客帝国","第一滴血","指环王","速度与激情","指环王"]

print("-----删除前，电影列表的数据-----")
for name in movieName:
    print(name,end=",")
```

```
print()
#删：  【del】删除
del movieName[1]                    #在指定位置删除一个元素，没有返回值

#删：  【pop】弹出
movie = movieName.pop()             #默认弹出末尾最后一个元素，有返回值（被弹出的那个元素）
print("被pop弹出的元素：",movie)

movie = movieName.pop(-2)           #弹出指定位置的元素，有返回值（被弹出的那个元素）
print("被pop弹出的元素：",movie)

#删：  【remove】移除
movieName.remove("指环王")           #直接删除指定“内容”的元素，但只删除匹配到的第一个
movieName.remove("指环王2")          #删除不存在的元素，报错。

print("-----删除后，电影列表的数据-----")
for name in movieName:
    print(name,end=",")
```

【改】：

```
namelist = ["小张","小王","小李","小赵"]
print("-----修改前，名单列表的数据-----")
for name in namelist:
    print(name)

namelist[2] = "小红"

print("-----修改后，名单列表的数据-----")
for name in namelist:
    print(name)
```

【查】：

```
#查：【in】 ，【not in】
findName = input("请输入你要查找的学生姓名：")
if findName in namelist:
    print("在学员名单中找到了相同的名字")
else:
    print("没有找到")

#查：【index】
mylist = ["a","b","c","a","b"]
print(mylist.index("a",1,4))          #可以查找指定下标范围的元素，并返回找到对应数据的下标
print(mylist.index("a",1,3))          # 范围区间，左闭右开   [1,3)
                                      #找不到会报错
#查 ：【count】
print(mylist.count("a"))              #统计某个元素出现几次

#排序和反转 【sort】、【reverse】
a = [1,4,2,3]
print(a,id(a))
```

```
a.reverse()                             # 将列表所有元素反转
print(a,id(a))                          #a的地址没有改变

a.sort()                                #升序
print(a,id(a))                          #a的地址没有改变

a.sort(reverse=True)                    #降序
print(a,id(a))
```

3.2.3 列表的嵌套

- 列表嵌套

  类似while循环的嵌套，列表也是支持嵌套的

  一个列表中的元素又是一个列表，那么这就是列表的嵌套

```
    schoolNames = [['北京大学','清华大学'],
                   ['南开大学','天津大学','天津师范大学'],    #每个子列表元素个数可以不同
                   ['山东大学','中国海洋大学']]
```

示例：

```
namelist = []   #一维空列表

namelist = [[],[],[]]                   # 有3个元素的空列表，每个元素都是一个空列表
print(len(namelist))

zhangsan = ["张三",18,"男"]
lisi = ["李四",19,"女"]
wangwu  = ["王五",20,"男"]

namelist = [zhangsan,lisi,wangwu]   #二维列表
print(namelist)

# 二维数据，类似表格的存储形式
#[['张三', 18, '男'],  #[0][0],[0][1],[0][2]
# ['李四', 19, '女'],  #[1][0],[1][1],[1][2]
# ['王五', 20, '男']]  #[2][0],[2][1],[2][2]

print(namelist[0])                      #通过索引，获取到的第一个元素是一个列表
print(namelist[0][0])                   #通过第二个中括号中的索引，获取的是第一个列表的第一个
元素

for person in namelist:
    # print(person)
    for i in person:
        print(i,end=",")
    print()
```

- 应用

一个学校，有3个办公室，现在有8位老师等待工位的分配，请编写程序，完成随机的分配

```
import random

# 定义一个列表用来保存3个办公室
offices = [[],[],[]]      #有3个元素的空列表，每个元素都是一个空列表
#未来可能形成的结果：[["A","D","H"],["B","C"],["E","F","G"]]

# 定义一个列表用来存储8位老师的名字
names = ['A','B','C','D','E','F','G','H']

#1.拿出每个字符串（names列表中）
#2.随机生成offices中的下标，找到对应的辨公室
#3.将老师的名字，放到（追加到）对应的办公室列表中

for name in names:
    index = random.randint(0,2)
    offices[index].append(name)

i = 0
for office in offices:
    print('办公室%d的人数为:%d'%(i,len(office)))
    i += 1
    for name in office:
        print(name,end="\t")
    print()
    print("-"*20)
```

3.2.4 浅拷贝和深拷贝（计算机专业必会）

变量赋值的底层逻辑

内存，就像储物柜

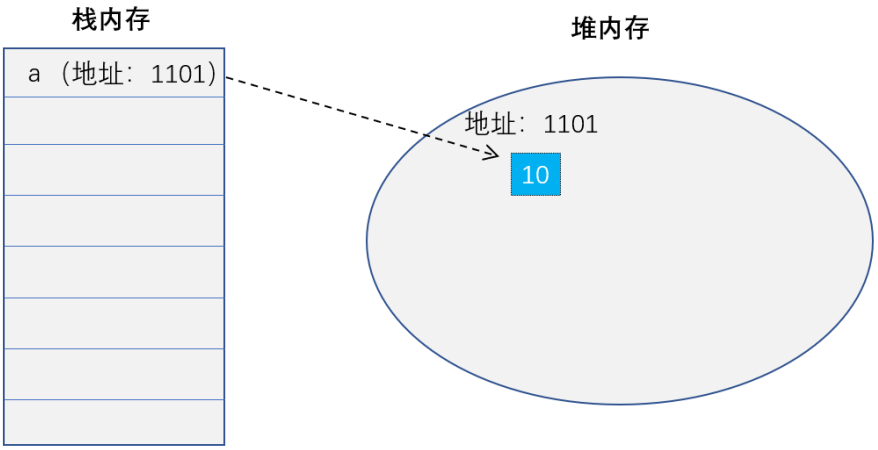
a = 10

变量a ，
就像小票，
存储的是箱子的地址（位置）

数字10 ，
就像寄存的包裹，
存储在某个箱子中

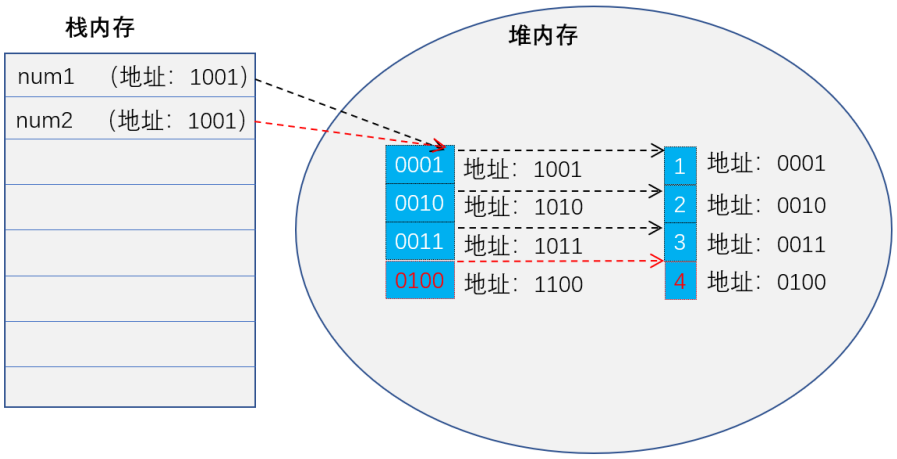
变量赋值

表达式：a = 10



变量赋值

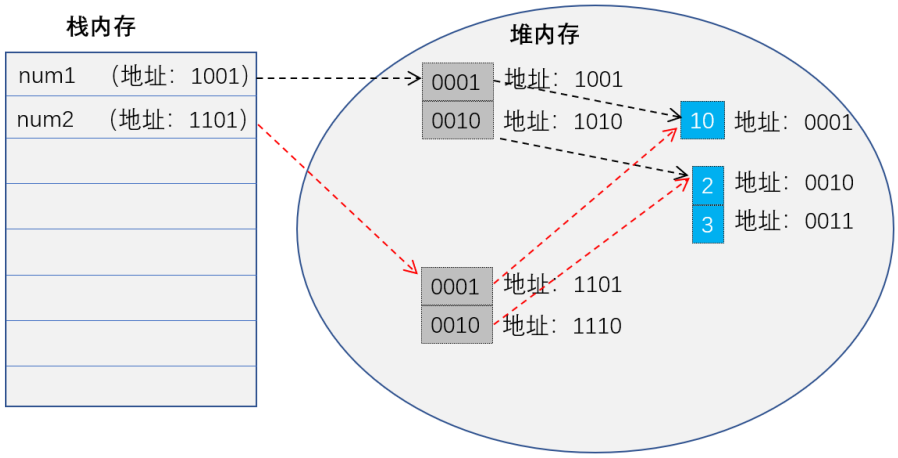
表达式：
num1 = [1, 2, 3]
num2 = num1
num1.append(4)
print(num2)




```
num1 = [1,2,3]
num2 = num1                #num1将地址值复制给了num2  （参考将寄存箱小票指向的位置，复制给了num2）
num2.append(4)             #所以num2修改了箱子里面的内容，num1取出来的内容也就变了。
print("num1:",num1,id(num1))
print("num2:",num2,id(num2))
```

浅拷贝

表达式：
num1 = [10, [2, 3]]
num2 = num1.copy()

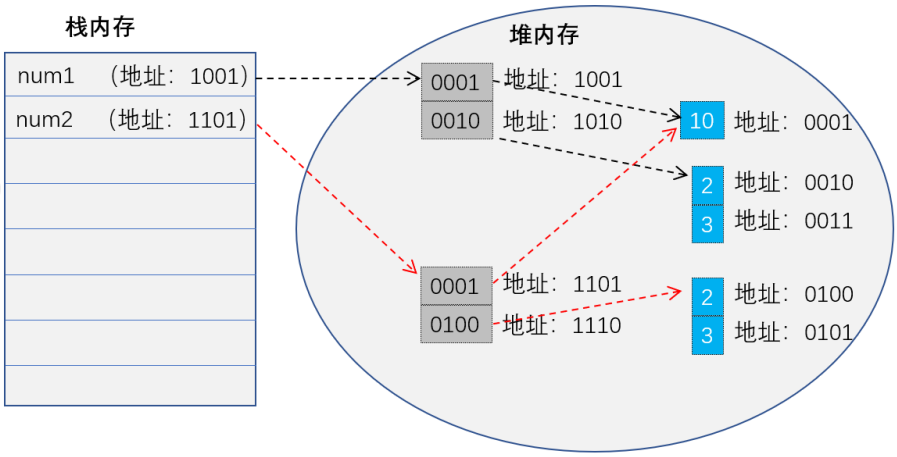


```
num1 = [10,[2,3]]
num2 = num1.copy()                  #将num1所指向的数据内容（地址值），复制了一份给num2
# num2.append(4)                    #所以，修改num2,num1不受影响。
print("num1:",num1,id(num1))        #num1和num2的地址值不同
print("num2:",num2,id(num2))
#num1[0] = 8                        #修改num1，相当于修改了地址指向，num2不受影响
print("num1[0]:",num1[0],id(num1[0]))
print("num2[0]:",num2[0],id(num2[0]))
print("num1[1]:",num1[1],id(num1[1]))
print("num2[1]:",num2[1],id(num2[1]))
```

深拷贝

表达式：

```
Import copy
num1 = [10, [2, 3]]
num2 =
    copy.deepcopy(num1)
```




```
import copy
num1 = [10,[2,3]]
num2 = copy.deepcopy(num1)          #将num1所指向的数据元素，复制了一份给num2
# num1[-1].append(8)                #所以，修改num1的列表元素中的内容，num2中对应的列表元素也变了
print("num1中列表的地址：",id(num1[-1]),"\nnum2中列表的地址：",id(num2[-1]))
                                    #说明列表中的”列表元素“没有复制元素内容，仅仅复制的是地址。 print("num1:",num1,id(num1))
print("num2:",num2,id(num2))
print("num1[0]:",num1[0],id(num1[0]))
print("num2[0]:",num2[0],id(num2[0]))
print("num1[1]:",num1[1],id(num1[1]))
print("num2[1]:",num2[1],id(num2[1]))
```

小结：

浅拷贝：只能copy列表的一级元素，复制了嵌套的可变数据类型的**地址**

深拷贝：能够copy列表所有层级的元素，复制了嵌套的可变数据类型**元素**

练习：

结合上面的讲解，尝试用上面的图形表达方式，绘制并解释下列代码显示结果：

```
n1 = [1,2,3,[4,5]]
n2 = n1[:]
n1.append(6)
print(n1)       #[1, 2, 3, [4, 5], 6]
print(n2)       #[1, 2, 3, [4, 5]]
```

作业:

现有商品列表如下:

1. products = [["iphone",6888],["MacPro",14800],["小米6",2499],["Coffee",31],["Book",60],["Nike",699]]，需打印出以下格式:

```
------  商品列表 ------
0  iphone     6888
1  MacPro     14800
2  小米6       2499
3  Coffee     31
4  Book       60
5  Nike       699
```

2. 根据上面的products列表写一个循环，不断询问用户想买什么，用户选择一个商品编号，就把对应的商品添加到购物车里，最终用户输入q退出时，打印购买的商品列表。

3.3 元组

Python的核心数据类型

Tuple（元组）

- tuple与list类似，不同之处在于tuple的元素不能修改。tuple写在小括号里，元素之间用逗号隔开。
- 元组的元素不可变，但可以包含可变对象，如list。

注意：定义一个只有1个元素的tuple，必须加逗号。

```
>>> t = ('abcd', 786 , 2.23, 'runoob', 70.2)
>>> t1 = (1, )
>>> t2 = ('a', 'b', ['A', 'B'])
>>> t[2][0] = 'X'
>>> t
('a', 'b', ['X', 'B'])
```

3.3.1 元组的定义与访问

- 创建空元组

```
tup1 = ()
```

- 元组的定义

```
tup1 = (50)           # 不加逗号，类型为整型
print(type(tup1))     #输出<class 'int'>
```

```
tup1 = (50,)          # 加逗号，类型为元组
print(type(tup1))     #输出<class 'tuple'>
```

另一种写法：

```
tup1 = (50,60,70,80.0,"abc",True,50)          #元组可以存放不同类型的元素
print(tup1,type(tup1))

tup1 = 50,60,70      #小括号可以省略 （了解）
tup1 = 50,           #建议大家还是加上括号，以避免和数值混淆
print(tup1,type(tup1))
```

3.3.2 常用操作：

操作名称	操作方法	举例
访问元组中的元素	通过下标直接访问	print(tuple1[0])
遍历元组	通过for循环	for i in tuple1: print(i)
元组的切片	使用[: :]	tuple1[2:5:2]
元组的加法操作	+	tuple3 = tuple1 + tuple2

增：（连接）

```
tup1 = (12,34,56)
tup2 = ("abc","def")

tup = tup1 + tup2
print(tup,type(tup))
```

删：（删除的是元组对象，而不是元素）

```
tup1 = (12,34,56)
print(tup1)
del tup1
print("删除后：")
print(tup1)              #删除元组后，再次访问会报错
```

改：不能修改

```
tup1 = (12,34,56)
tup1[0] = 1
print(tup1[0])  #报错

#tuple里面的可变元素，元素内的内容可以更改
mylist = ["A","B"]
tup1 = (10,20,30,mylist)
print(tup1)
tup1[-1].append("C")
print(tup1)
```

查：

操作名称	操作方法	举例
元组成员关系	in	2 in list1
得到重复元素数量	count	tuple1.count(1)

操作名称	操作方法	举例
获取元组长度	len（）	
获取元组元素最大值	max（）	
获取元组元素最小值	min（）	
将元组各元素求和	sum（）	
其它类型对象转换成元组	tuple（）	

示例：

```
#切片
tup1 = ("abc","def",2020,2030,333,444,555,666)

print(tup1[0])
print(tup1[-1])                         #访问最后一个元素
print(tup1[1:5])                        #左闭右开，进行切片

tup = (1, 2, 3, 4, 5, 6, 7,3,3,3)       #元组的元素内容可以重复

for i in tup:
    print(i)

#包含  in，not in
if 3 in tup:
    print("True")
else:
    print("False")

#计数  count
print(tup.count(3))
```

```
#最大/最小/求和/个数    max、min、sum、len

print("元组中最大的元素:", max(tup))
print("元组中最小的元素:", min(tup))
print("元组中各元素的和:", sum(tup))
print("元组中元素的个数:", len(tup))

#强制类型转化   tuple()

#强制类型转化：字符串
s = "IT私塾"
tup = tuple(s)
print(tup,type(tup))     #('I', 'T', '私', '塾')

#B强制类型转化：列表
list1 = ["a","b","c"]
tup = tuple(list1)
print(tup,type(tup))


#生成器对象 （了解）
s = (x*2 for x in range(5))
print(tuple(s))
```

3.4 字典

Python的核心数据类型

dict（字典）

- 字典是无序的对象集合，使用键-值（key-value）存储，具有极快的查找速度。
- 键(key)必须使用不可变类型。
- 同一个字典中，键(key)必须是唯一的。

```
>>> d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}
>>> d['Michael']
95
```

3.4.1 字典的定义

变量info为字典类型：

```
info = {}       #定义一个空字典
info = {"name":"彭于晏","age":18,("aa"):[1,2]}
print(info,type(info))
```

说明：

- 字典和列表一样，也能够存储多个数据
- 列表中找某个元素时，是根据下标进行的
- 字典中找某个元素时，是根据'名字'（就是冒号:前面的那个值，例如上面代码中的'name'、'age'、("aa")
- 字典的每个元素由2部分组成，键:值。例如 'name':'班长' ,'name'为键，'班长'为值

使用列表创建字典 【fromkeys】

```
namelist = ["彭于晏","吴彦祖","金城武"]

#dic1 = {}.fromkeys(namelist)                #以列表中的元素内容作为键，创建字典；值默认为None
dic1 = {}.fromkeys(namelist,"颜值高")         #可以设定统一的默认值
dic1 = {}.fromkeys(namelist,["帅","颜值高","好看"])    #无法逐一设置不同的默认值

print(id(dic1["彭于晏"]),id(dic1["吴彦祖"]),id(dic1["金城武"]))
                                             #dic中的每个键都被指向了同一个地址空间
print(dic1)
```

3.4.2 根据键访问值

```
info = {"name":"彭于晏","age":18}
print(info["name"])              #获取姓名
print(info["age"])               # 获取年龄

# print(info['sex'])             # 获取不存在的key，会发生异常，错误类型位：KeyError
print(info.get('sex'))           # 获取不存在的key，获取到空的内容，不会出现异常

print(info.get("age","20"))      #找到键的情况下，修改键对应的值
print(info.get("gender","m"))    #没找到的时候，可以设定返回的默认值
```

3.4.3 常用操作:

----------操作名称 ----------	操作方法	举例
访问字典中的元素（1）	通过key访问，key不存在会抛出异常	print(dict1[“小明”])
访问字典中的元素（2）	通过get方法，不存在返回None，不抛出异常	print(dict1.get(“”小明))
遍历字典（1）	通过for循环，只能获取key	for key in dict1: print(key, dict1[key])
遍历字典（2）	配合items方法，获取key和val	for key, val in dict1.items(): print(key, val)
直接获取所有key和value	使用keys和values方法	print(dict1.values()) print(dict1.keys())
修改val	直接通过key修改	dict1['小明'] = 2003
新增键值对	直接新增	dict1[‘小李'] = 2005

示例：

```
#增
info = {"name":"彭于晏","age":18}
newID = input("请输入新的学号：")
info["id"] = newID
print(info["id"])
```

操作名称	操作方法	举例
字典元素的删除	通过key删除	del dict1['小李']
字典元素的弹出	通过pop方法	dict1.pop("小李")
判断key是否存在	in	“key” in dict1
合并字典	update	dict1.update(dict2)
把两个列表转为字典	dict+zip方法	dict(zip(list1,list2))
把一个嵌套列表转为字典	dict方法	dict2 = dict([['key1', 'value1']])
清除字典内的元素	clear方法	dict1.clear()

示例：

```
#删：【del】
info = {"name":"彭于晏","age":18}
del info["name"]
print(info)

print(info["name"]) #删除了指定键值对后，再次访问会报错
```

```python
info = {"name":"彭于晏","age":18}
del info                #删除字典对象
print(info)

#删：【clear】  清空
info = {"name":"彭于晏","age":18}
info.clear()
print(info)


#删：  【pop】【popitem】
info = {"name":"彭于晏","age":18,"性别":"男"}
gender = info.pop("性别")       #通过键，删除指定的键值对；会返回删除的键值对的值
gender = info.pop("gender")      #删除一个不存在的键，会报错:KeyError
gender = info.pop("gender","不存在该键")  #删除一个不存在的键，会报错；没有找到指定键，可以返回默认值

gender = info.popitem()      #删除最后一个键值对,返回的是：元组类型
gender = info.popitem()
print(type(gender),info)

#查：遍历
info = {"name":"彭于晏","age":18}

for i in info:  #默认或得到的就是字典的“键”
    print(i)

print(info.keys(),type(info.keys()))      #得到所有的键（可迭代对象）,可以理解为：列表

print(info.values(),type(info.values())) #得到所有的值（可迭代对象），可以理解为：列表

print(info.items(),type(info.items()))    #得到所有的项（可迭代对象），
                                          #可以理解为：每个键值对是一个元组

#遍历所有的键
for key in info.keys():
    print(key)

for value in info.values():
    print(value)

#遍历所有的键值对
for key,value in info.items():
    print(f"key={key},value={value}")


#扩展：使用枚举函数，同时拿到列表中的下标和元素内容
mylist = ["a","b","c","d"]
# print(type(enumerate(mylist)))
for i,x in enumerate(mylist):
    print(i+1,x)
```

操作名称	操作方法	举例
获取字典长度	len（）	
获取最大的key	max（）	
获取最小的key	min（）	
其它类型对象转换成字典	dict（）	dict([(1, 2), (2, 3)])

示例：略

3.5 集合

Python的核心数据类型

set（集合）

- set和dict类似，也是一组key的集合，但不存储value。由于key不能重复，所以，在set中，没有重复的key。
- set是无序的，重复元素在set中自动被过滤。

```
>>> s = set([1, 2, 3])
>>> s
{1, 2, 3}
>>> s = set([1, 1, 2, 2, 3, 3])
>>> s
{1, 2, 3}
```

set可以看成数学意义上的无序和无重复元素的集合，因此，两个set可以做数学意义上的交集（&）、并集（|）、差集（-）等操作。

3.5.1 集合的定义和特性

3.5.1.1集合的定义

```
s1 = set()      #空集合
print(s1,type(s1))

s1 = {1,2,3,4}
print(s1,type(s1))
```

3.5.1.2 集合是无序的

```
s1 = {1,2,3,4}      #{1, 2, 3, 4} <class 'set'>
s1 = {4,3,2,1}      #{1, 2, 3, 4} <class 'set'>  难道是自动排序？
s1 = {"彭于晏","吴彦祖","金城武"}  #{'彭于晏', '吴彦祖', '金城武'} <class 'set'>
s1 = {"彭于晏","金城武","吴彦祖"}  #{'金城武', '彭于晏', '吴彦祖'} <class 'set'>
print(s1,type(s1))      #小结：集合中存放的数据是“无序”的
```

3.5.1.3 集合中的元素是不重复的

```
s1 = {1,2,3,4,4}
print(s1)            #自动去重（去除重复）
```

3.5.1.4 集合：只能存放不可变元素 (key)

```
s1 = {2.0,True,1,"abc"}              #字典也可以存放多种数据类型
s1 = {2.0,True,1,"abc",{"id":1}}     #报错：TypeError: unhashable type: 'dict'
s1 = {2.0,True,1,"abc",[3,4]}        #报错：TypeError: unhashable type: 'list'
s1 = {2.0,True,1,"abc",(3,4)}        #可以
print(s1,type(s1))
```

小结：集合中只能存放不可变的类型，列表（list）和字典（dict）不能放到集合中

3.5.2常用操作：

操作名称	操作方法	举例
遍历集合	通过for循环	for i in set1: print(i)
更新集合	update方法	set1.update(set2)
向集合添加新元素	add方法	set1.add(5)
移除集合中的元素	remove方法	set1.remove(5)
弹出元素	pop方法	val = set1.pop()
清除元素	clear方法	set1.clear()
删除集合	del	del set1

示例:

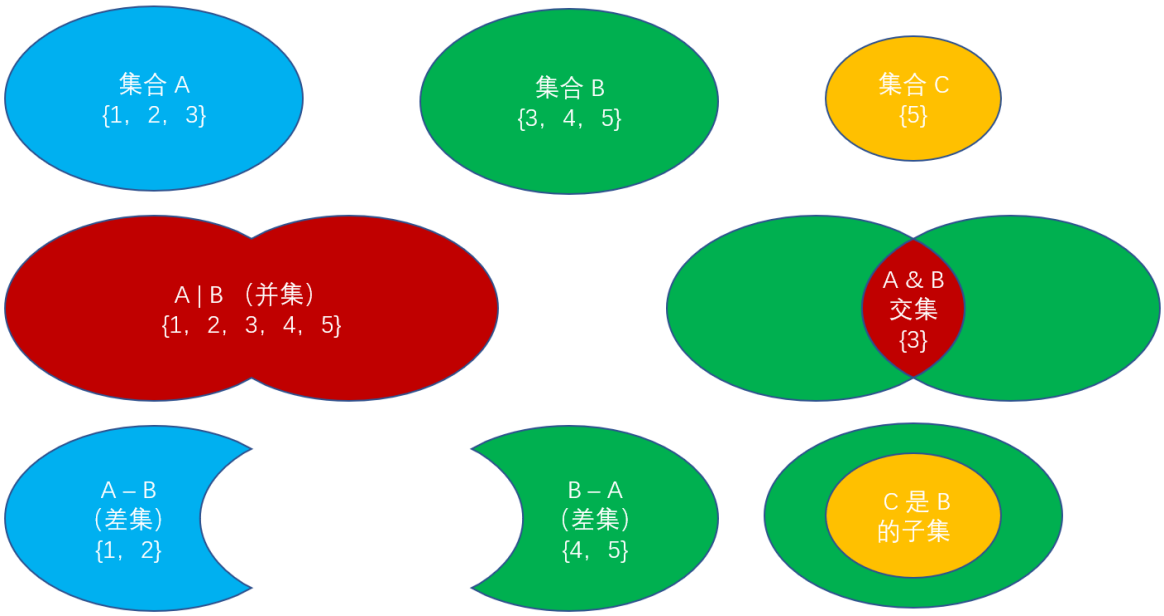
```
#【增】：  add
s1 = {1,2,3,4,5}
print(s1,id(s1))

s1.add(6)
print(s1,id(s1))

#将指定的序列，依次加入到集合中
s1.update("abcd")
print(s1,id(s1))

s1 = {1,2,3,4,5}
s2 = ["aa","bb","cc"]
s1.update(s2)
print(s1,id(s1))
```

```
#【删】 ：clear、remove、pop、discard
s1 = {"a","b","c","d"}

s1.clear()            #清空集合中的元素
x = s1.pop()          #pop随机弹出一个元素，每次执行结果不一定相同
print(x,s1)

s1.remove("e")        #只能通过元素内容来删除,如果没有找到指定元素，报错：KeyError
print(s1)

s1.discard("a")       #如果没有找到指定元素,不做任何操作          英文含义：丢弃、打出（无用的
牌）
print(s1)


#【改】：变相删除，先删后增
s1 = {"a","b","c","d"}
s1.remove("b")
s1.add("e")
print(s1)


#【查】
#遍历
s1 = {"a","b","c","d"}

for i in s1:
    print(i)

#包含 in，not in
s1 = {"a","b","c","d"}
res = "a" not in s1
print(res)
```

操作名称	操作方法	举例
获取集合长度	len（）	
获取最大的元素	max（）	
获取最小的元素	min（）	
其它类型对象转换成集合	set（）	

示例:略

3.5.3集合的运算



集合的运算：交、差、并、反交

在对集合做运算时，不会影响原来的集合，而是返回一个运算结果

示例:

```
#创建两个集合
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

# 交集运算 （& 或者intersection）
result = s1 & s2            # {3, 4, 5} #两个集合中都有的部分
result = s1.intersection(s2)
print("result=",result)


# 并集运算 (| 或 union)
result = s1 | s2            # {1,2,3,4,5,6,7}    #两个集合中出现的全部数据（去重:去掉重复)
result = s1.union(s2)
print('result =',result)

# 差集 (- 或 difference)
result = s1 - s2            # {1, 2}         #保留的是被减数的部分
result = s1.difference(s2)
result = s2 - s1            # {6, 7}
print('result =',result)

# 异或集 或称作 对称差集    (^ 或 symmetric_difference)
result = s1 ^ s2          # {1, 2, 6, 7}  #获取只在一个集合中出现的元素
result = s1.symmetric_difference(s2)
print('result =',result)
```

```python
# 子集或超集
# <= 检查一个集合是否是另一个集合的子集
# 如果a集合中的元素全部都在b集合中出现，那么a集合就是b集合的子集，b集合是a集合超集

a = {1,2,3}
b = {1,2,3,4,5}

result = a <= b
print(a.issubset(b))
print(b.issuperset(a))
# result = {1,2,3} <= {1,2,3}
result = {1,2,3,4,5} <= {1,2,3}
print('result =',result)


# < 检查一个集合是否是另一个集合的真子集
# 如果超集b中含有子集a中所有元素，并且b中还有a中没有的元素，则b就是a的真超集，a是b的真子集

result = {1,2,3} < {1,2,3}      # False
print('result =',result)
result = {1,2,3} < {1,2,3,4,5}  # True
print('result =',result)


# 不相交 isdisjoint
result = a.isdisjoint(b)    #不相交：True；  相交：False
print("result=",result)


#应用：
#列表去重：
num = [1,1,2,2,3,3,4,4]
s = set(num)                        #注意：顺序可能会被打乱
res = list(s)
print(res,type(res))


#冰冻集合 frozenset（了解）
#冰冻集合，可以强制转化为容器类型：冰冻集合
#冰冻集合一旦创建，不能进行任何修改，只能用于集合运算操作

s = frozenset()     #定义一个空的冰冻集合
print(s,type(s))

s1 = [1,2,3,4,5,"abc"]
s1 = "abcde"
s2 = frozenset(s1)
print(s2,type(s2))

#s2.add("f")                        #报错：冰冻集合不能做修改

s3 = frozenset("aef")   #可以进行结合运算
print(s2 & s3)
```

3.6小结

	是否有序	是否可变类型	元素重复与否
列表[ ]	有序	可变类型	可重复
元组( )	有序	不可变，元素不能修改	可重复
字典{ }	无序	可变（键只能增删，不可修改）	key不重复
集合{ }	无序	可变（通过增删实现修改）	不重复（自动去重）

简单理解：因为“有序”，可以通过索引（下标）访问元素，所以可以“重复”

因为“无序”，只能通过内容访问元素，所以“不可重复"