对象模型

- 对象模型基础
- 内置数据模型概述
- 数字
- 字符串
- 列表
- 元组
- 字典
- 其他数据类型
- 深拷贝与浅拷贝

1. 学习目标

- 掌握Python对象模型思想
- 掌握Python内置数据模型
- 掌握序列模型

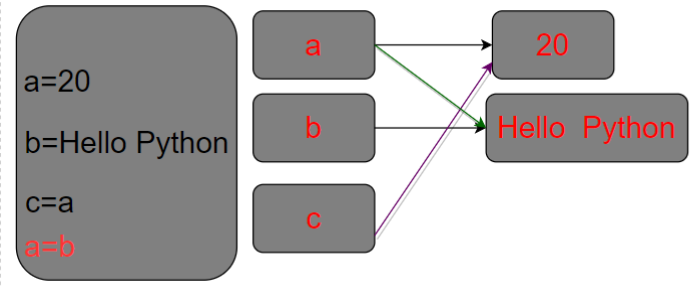
2.对象模型基础

- 对象是Python中可以操作的最小数据单位
- 对象的本质是用于存放数据的内存块
- 对象的属性
  - 标识【ID】
  - 类型
  - 值【数据内容】
- 对象的访问
  - 直接访问
  - 成员访问【.】
  - 方法访问【调用】
  - 下标访问

2.1 对象模型基础--变量、对象、引用

- 赋值语句创建变量、对象、引用
- 变量名在Python缓冲区内独立存放
- 整数、ASCII码、短字符会被Python缓存

举例：



2.2 对象模型基础--赋值语句操作

- 赋值语句会在【变量名】与【对象】之间建立【引用】
- 变量名首次赋值时会被创建
- 有些数据结构对象元素也会在赋值时创建

2.3 对象模型基础--动态类型

- 变量无类型、对象有类型
- 引用与数据【对象】分离
- 通过赋值使变量与对象建立联系
- 引用的本质是指针

示例一：

```
>>> a = 2
>>> b = 2
>>> id(a), id(b)
(140731662762288, 140731662762288)
>>> a,b
(2, 2)
>>> a = 4
>>> a,b
(4, 2)
>>> id(a), id(b)
(140731662762352, 140731662762288)
```

示例二：

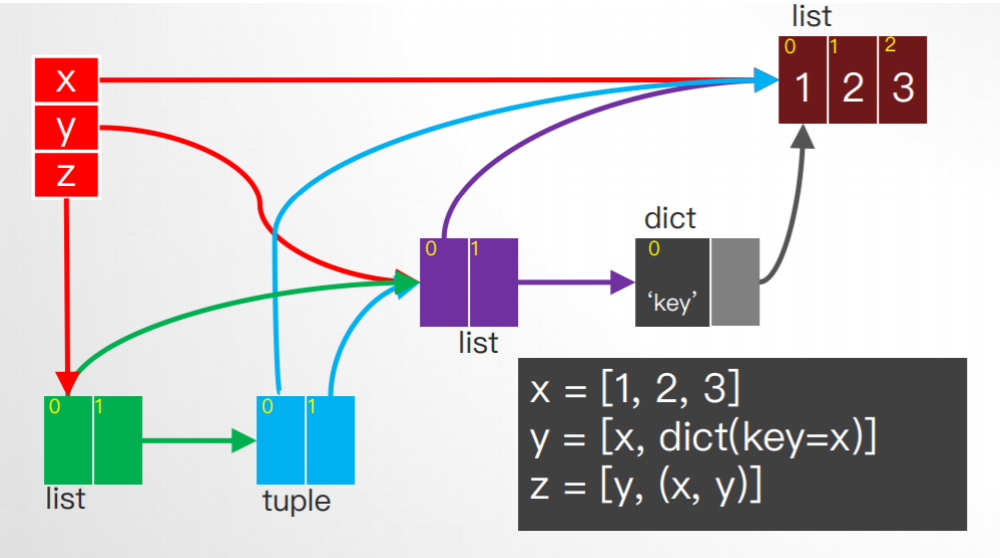
```
>>> a = [1,2,3]
>>> b = a
>>> a,b
([1, 2, 3], [1, 2, 3])
>>> id(a), id(b)
(2320906998792, 2320906998792)
>>> a[0] = 100
>>> a,b
([100, 2, 3], [100, 2, 3])
>>> id(a), id(b)
(2320906998792, 2320906998792)
```

2.4 对象模型基础--引用计数

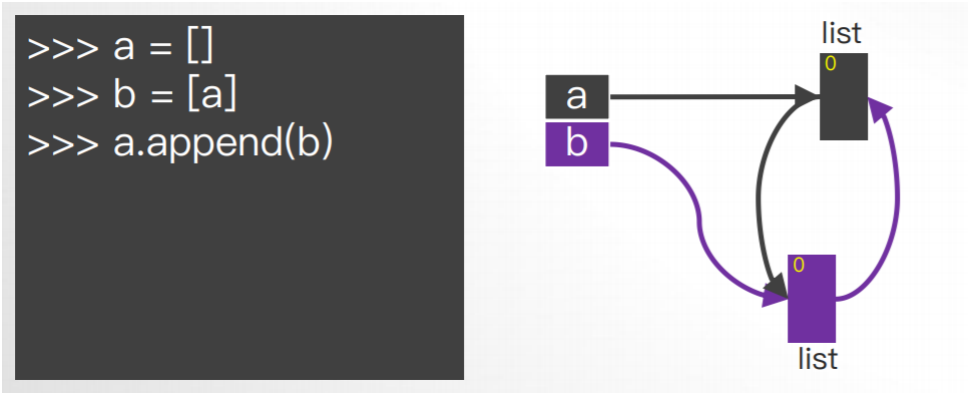
- 对象会保存指向自己的引用总数，即引用计数
- 引用计数器为0时对象会被垃圾回收器回收
- 每增加一个指向对象的引用，计数器+1
- 查看对象引用计数器：

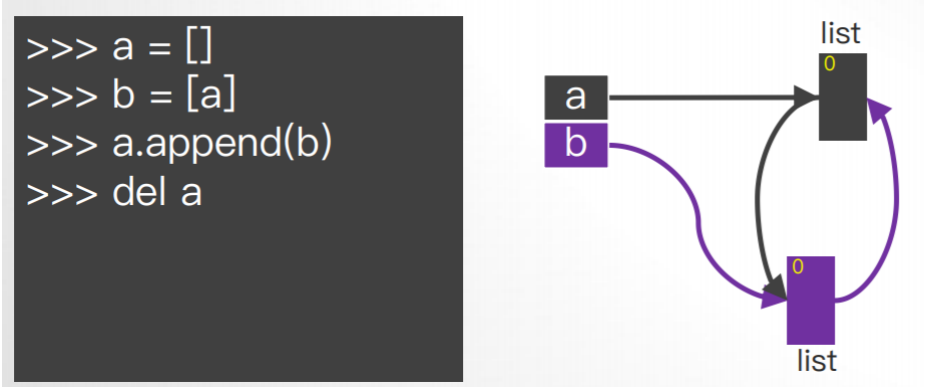
```
>>> from sys import getrefcount
>>> a = [1, 2, 3]
>>> print(getrefcount(a))
2
>>> b = a
>>> print(getrefcount(b))
3
```

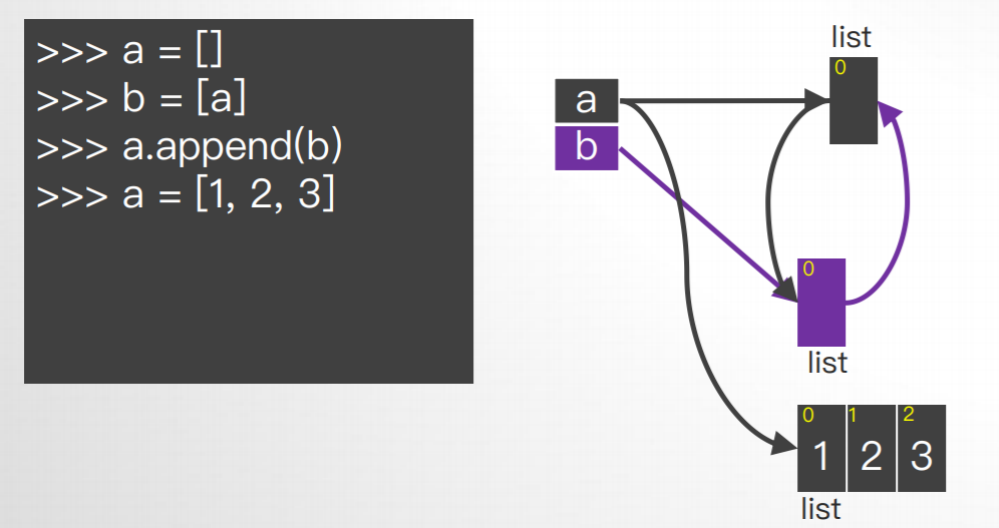
引用计数：



引用计数删除：







2.5 对象模型基础--垃圾回收

- 系统在适当的时候会启动垃圾回收器
- 回收原则基于引用计数
- 对象的引用计数为0时，内存空间被回收
- 垃圾回收代价较大，采用'定时'回收机制

```
>>> import gc
>>> print(gc.get_threshold())  #自动回收触发阈值
(700, 10, 10)
>>> gc.collect() #手动回收
0
>>> gc.set_threshold(800, 10, 5)
```

2.6 对象模型基础--类型

- 类型的作用
  - 对【数据】进行封装
  - 明确数据的含义
  - 防止语法歧义
- 自定义类型
  - 函数
  - 类
- 内建类型
  - Python内置数据结构

3. 内置数据模型概述

- 数字（Number）
  - 布尔类型（Bool）
  - 复数（Complex ）
  - 浮点数（Float）
  - 整数（Integer）
- 字符（String）

- 列表（List）
- 元组（Tuple）
- 集合（Set）
- 字典（Dict）
- 文件（File）

3.1 内置数据模型概述-运算

操作	操作符
算术运算	+, -, * , /, //, %, ** ,
逻辑运算	and, or, not
比较运算	==, !=, >, =, <=
位运算	&, |, ~, ^ , <>
成员运算	in, not in
身份运算	is, is not
赋值运算	=, +=, -=, *=, /=, %=, //=, **=

3.2 内置数据模型概述-常用内建函数

函数	功能
type()	返回对象类型
isinstance()	基类、派生类/相同对象判断
cmp(a, b)	a>b返回1, a<b返回-1, a==b返回0
str()/repr()	返回对象的字符串形式
dir()	返回对象的字符串形式

- cmp 在3.4+被opreator替代

```
import operator
operator.gt(1,2)
operator.le(1,2)
```

- str、repr、eval
  - str 是将其他数据类型转化成str类型，一般用于给用户显示查看
  - repr返回对象的字符串表现形式，一般开发人员调试或实现特殊的功能
  - 差别：当作用于字符串时，repr比str候多一个引号
  - eval() 函数用来执行一个字符串表达式 `1<2`，并返回表达式的值。

4. 数字类型

- 整数
- 浮点数
- 布尔类型
- 复数

4.1 数字--常量表示

常量	案例
整数常量	1234， -1， 0， 9999999999999
浮点数	3.14e-10， 1.23
十六进制	0x1122， 0xABCD
八进制	0O127， 0O777
二进制	0B1101， 0B0000
复数	Z=a+bj；z.real表示实部；z.imag表示虚部

4.2 数字--布尔类型

- True
  - 数字非零
  - 对象非空
- False
  - 数字零
  - 空对象
  - None

4.3 数字--数字运算

- 混合类型计算时按最大精度类型计算
  - 精度：整数<浮点数<复数
- 运算类型
  - 赋值运算
  - 算术运算
  - 比较运算
  - 逻辑运算：布尔类型
  - 位运算
- 类型转换

操作	说明
int(x[,base])	将x转换为一个整数
long(x[,base])	将x转换为一个长整数
float(x)	将x转换到一个浮点数
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为Unicode字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

- 注意 long/unichr不再支持
- 8位 2^8=255

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	_
48	0	64	@	80	P	96	`	112	p		

4.4 数字--内置函数模块

内建函数	描述
abc()	abs() 函数返回数字的绝对值。
pow()	pow() 方法返回 xy（x 的 y 次方） 的值。
max()	max() 方法返回给定参数的最大值，参数可以为序列。
min()	min() 方法返回给定参数的最小值，参数可以为序列。
sum()	sum() 方法对序列进行求和计算。
math模块	**描述**
math.sqrt()	sqrt() 方法返回数字x的平方根。
math.pi	圆周率数值
math.sin()	sin() 返回的x弧度的正弦值。
random模块	**描述**
random.randint(1, 10)	生成1到9的一个随机数
random.choice([1, 2, 3, 4, 5])	随机返回列表中的一个数

5. 字符串类型

- 字符串基础
- BIF
- 字符串操作
- 格式化
- 数据类型划分

5.1 字符串--基础

- 定义：由一系列字符组成的不可变序列容器，存储的是字符的编码值。
- 基础知识
  - 字节byte：计算机最小存储单位，等于8 位bit。
  - 字符：单个的数字、文字与符号。

    - 字符集(码表)：存储字符与二进制序列的对应关系。
    - 编码：将字符转换为对应的二进制序列的过程。
    - 解码：将二进制序列转换为对应的字符的过程。
- 编码方式
    - ASCII编码：包含英文、数字等字符，每个字符1个字节。
    - GBK编码：兼容ASCII编码，包含21003个中文；英文1个字节，汉字2个字节。
    - Unicode字符集：国际统一编码，旧字符集每个字符2字节，新字符集4字节。
    - UTF-8编码：Unicode的存储与传输方式，英文1字节，中文3字节。
- 表示方法
    - 注释：
        - `#`: 表示单行注释
        - 三引号：表示多行注释

```
'This is a Python string.'

"This is a Python string."

'''
This is a Python String.
Life is short you need Python.
'''

"""
This is a Python String.
Life is short you need Python.
"""
```

- 转义字符

`str1 = ‘a\nb\tc’`

转义字符	描述	转义字符	描述
\ (行尾)	续航符	\\	反斜杠
\’	单引号	\”	双引号
\a	响铃	\b	退格
\e	转义	\000	空
\n	换行	\v	纵向制表符
\t	横向制表符	\r	回车
\f	换页	\0yy	八进制数
\xyy	十六进制数	\other	其他字符以普通格式输出

- 反转义操作

```
>>> fileName = 'F:\briup\text.dat'
>>> print(fileName)
Friup    ext.dat
>>> fileName = 'F:\\briup\text.dat'
>>> print(fileName)
F:\briup        ext.dat
>>> fileName = r'F:\briup\text.dat'
>>> print(fileName)
F:\briup\text.dat
```

5.2 字符串--BIF（常用内置方法）

BIF	功能描述
find()	返回子字符串在字符串中的索引值，不存在返回-1
join()	将指定字符串插入到某序列的条目中间
len()	参数放一个字符串，返回一个字符串的长度
count()	返回字字符串出现的次数
split()	按照指定字符分割字符串，返回一个字符串列表
repleace()	内容替换，返回替换后的字符串，原数据不变
dir()	查看属性
help()	帮助文档

5.3 字符串--操作

- 字符转换

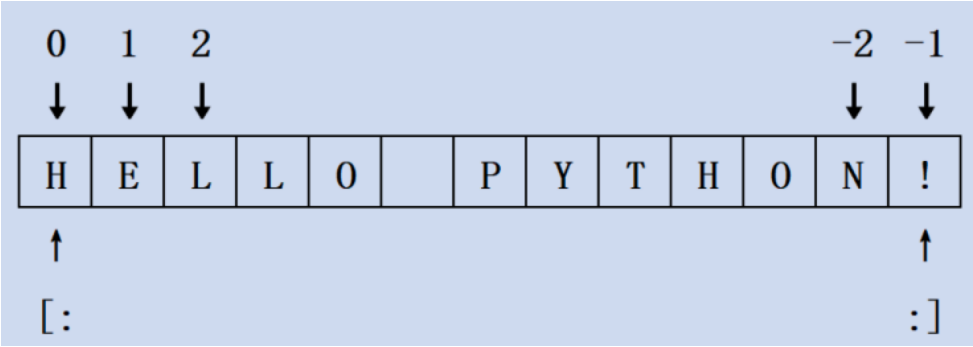
```
str(obj)/repr(obj)   #将对象转化为字符串的类型
ord('a')             #将'a'转化为整数
chr(65)              #将65转化为字符串类型
```

示例：

```
>>> a = [1, 2, 3]
>>> str(a)
'[1, 2, 3]'
>>> str(123)
'123'
>>> repr(3**9)
'19683'
```

- 索引、分片

```
str = 'HELLO PYTHON!'
str[ i ]
str[ i : j : k ]     #  序列[开始位置下标:结束位置下标:步长]
```



练一练：
字符串： content = "你是猴子请来的救兵吗？"
(1) 打印第一个字符、打印最后一个字符、打印中间字符
(2) 打印字前三个符、打印后三个字符
(3) 命题：猴子在字符串content中
(4) 命题：孙悟空不在字符串content中
(5) 通过切片打印“猴子请来的”
(6) 通过切片打印“兵救的来请子猴”
(7) 通过切片打印“你子的吗”
(8) 倒序打印字符

```
content = "你是猴子请来的救兵吗？"
length = len(content)
print(content[0])
print(content[length - 1])
print(content[length // 2])
print(content[:3])
print(content[length - 3::])
print("猴子" in content)
print("孙悟空" not in content)
print(content[2:7])
print(content[-3:1:-1])
print(content[::3])
print(content[::-1])
```

5.4 字符串--格式化

(1) 定义：生成一定格式的字符串。

(2) 语法：字符串%(变量)

```
"我的名字是%s,年龄是%s" % (name)
```

(3) 类型码：

%s 字符串 %d整数 %f 浮点数

```
print("%.2d：%.2d"%(2,3)) # 02：03
print("治愈比例为%d%%" % 5) # 治愈比例为5%
print("价格%.2f元" % (5 / 3)) # 价格1.67元
```

练习：根据下列文字，提取变量，使用字符串格式化打印信息
湖北确诊67802人，治愈63326人，治愈率0.99
70秒是01分零10秒

```python
region = "湖北"
confirm = 67802
cure = 63326
cure_rate = 0.99
print("%s确诊%d人，治愈%d人，治愈率%0.2f" %(region, confirm, cure, cure_rate))
print(f"{region}确诊{confirm}人，治愈{cure}人，治愈率{cure_rate}")

second = 70
print("%d秒是%d分零%d秒" %(second, second//60, second % 60))
print(f"{second}秒是{second//60}分零{second % 60}秒")
```

- format

```
format：格式化方法
      {fieldname!conversionflag:formatspec}
```

字段	描述
fieldname	指定的数字或关键字，位置参数、关键字参数、下标/属性
confersionflag	r/a/s，对应参数的repr、ascii和str内置函数的返回值，默认repr
formatspec	参数的格式描述，见下表

示例：

```python
"My name is Bob, I'm 18 years old."
"My name is {0}, I'm {1} years old.".format('Bob', 18)
"My name is {name}, I'm {age} years old.".format(age=18, name='Bob')
 i =['Bob', 20]
"My name is {info[0]}, I'm {info[1]} years old.".format(info=i)
```

```
>>> "My name is Bob, I' m 18 years old."
'My name is Bob, I' m 18 years old.'
>>>
>>> "My name is {0}, I' m {1} years old.".format('Bob', 18)
'My name is Bob, I' m 18 years old.'
>>>
>>> "My name is {name}, I' m {age} years old.".format(age=18, name='Bob')
'My name is Bob, I' m 18 years old.'
>>>
>>> i =['Bob', 20]
>>> "My name is {info[0]}, I' m {info[1]} years old.".format(info=i)
'My name is Bob, I' m 20 years old.'
```

```
formatspec：格式化方法
      [[fill]align][sign][width][.precision][typecode]
```

字段	描述
fill	可选，填充“{0:-^10}.format(‘a’)=‘----a-----’
align	与fill同时出现，填充的对齐方式<, >, ^ 左/右/居中
sign	可选，数字的符号表正负，参数为字符串时不可用
width	可选，参数显示的宽度，字符为单位
.precision	可选，浮点数显示精度，四舍五入
typecode	可选，同%表达式

示例：

```
info = "My name is {name!s:^10s}, I'm {age:5.3f} years old."
info.format(name='Lucky',age=18.0)
```

```
>>> info = "My name is {name!s:^10s}, I' m {age:5.3f} years old."
>>> info.format(name='Lucky',age=18.0)
'My name is   Lucky    , I' m 18.000 years old.'
```

5.5 字符串--字符编码

- python2的默认编码方式：ASCII
- python3的默认编码方式：utf-8
- 常见编码格式：ASCII、GBK、unicode、UTF-8
    - ASCII：计算机诞生初期使用的编码方式
    - GB2312 是对 ASCII 的中文扩展。
    - GBK对GB2312进行了扩展
    - GBK再次扩展成为了GB18030
    - unicode：为了解决世界各地编码方式的统一问题，ISO发行的国际通用标准的编码方式
    - UTF-8：unicode编码的变种和扩展。

5.6 字符串--其它常用操作方法

- 查找
    - find()：检测某个子串是否包含在这个字符串中，如果在返回这个子串开始的位置下标，否则返回-1。
        - 语法

```
字符串序列.find(子串，开始位置下标，结束位置下标)
```

> 注意：开始和结束位置下标可以省略，表示在整个字符串序列中查找。

        - 示例

```python
mystr = "hello world and briup and briup and Python"

print(mystr.find('and'))  # 12
print(mystr.find('and', 15, 30))  # 22
print(mystr.find('ands'))  # -1
```

- index()：检测某个子串是否包含在这个字符串中，如果在返回这个子串开始的位置下标，否则则报异常。
  - 语法

```
字符串序列.index(子串，开始位置下标，结束位置下标)
```

> 注意：开始和结束位置下标可以省略，表示在整个字符串序列中查找。

  - 示例

```python
mystr = "hello world and briup and briup and Python"

print(mystr.index('and'))  # 12
print(mystr.index('and', 15, 30))  # 22
print(mystr.index('ands'))  # 报错  ValueError: substring not found
```

- rfind()： 和find()功能相同，但查找方向为**右侧**开始。
- rindex()：和index()功能相同，但查找方向为**右侧**开始。
- count()：返回某个子串在字符串中出现的次数
  - 语法

```
字符串序列.count(子串，开始位置下标，结束位置下标)
```

> 注意：开始和结束位置下标可以省略，表示在整个字符串序列中查找。

  - 示例

```python
mystr = "hello world and briup and briup and Python"

print(mystr.count('and'))  # 3
print(mystr.count('ands'))  # 0
print(mystr.count('and', 0, 20))  # 1
```

- 修改
  - replace()：替换
    - 语法

```
字符串序列.replace(旧子串，新子串，替换次数)
```

> 注意：替换次数如果超出子串出现次数，则替换次数为该子串出现次数。

- 示例

```
mystr = "hello world and briup and briup and Python"

print(mystr.replace('and', 'he'))
# 结果：hello world he briup he briup he Python

print(mystr.replace('and', 'he', 10))
# 结果：hello world he briup he briup he Python

print(mystr)
# 结果：hello world and briup and briup and Python
```

> 注意：数据按照是否能直接修改分为**可变类型**和**不可变类型**两种。字符串类型的数据修改的时候不能改变原有字符串，属于不能直接修改数据的类型即是不可变类型。

- split()：按照指定字符分割字符串。
  - 语法

```
字符串序列.split(分割字符, num)
```

> 注意：num表示的是分割字符出现的次数，即 将来返回数据个数为num+1个。

  - 示例

```
mystr = "hello world and briup and briup and Python"

print(mystr.split('and'))
# 结果：['hello world ', ' briup ', ' briup ', ' Python']

print(mystr.split('and', 2))
# 结果: ['hello world ', ' briup ', ' briup and Python']

print(mystr.split(' '))
# 结果：['hello', 'world', 'and', 'briup', 'and', 'briup',
'and', 'Python']

print(mystr.split(' ', 2))
# 结果：['hello', 'world', 'and briup and briup and Python']
```

> 注意：如果分割字符是原有字符串中的子串，分割后则丢失该子串。

- join()：用一个字符或子串合并字符串，即是将多个字符串合并为一个新的字符串。
  - 语法

```
字符或子串.join(多字符串组成的序列)
```

  - 示例

```python
list1 = ['jie', 'pu', 'ruan', 'jian']
t1 = ('b', 'r', 'i', 'u', 'p')

print('_'.join(list1))
# 结果：jie_pu_ruan_jian

print('...'.join(t1))
# 结果：b...r...i...u...p
```

- capitalize()：将字符串第一个字符转换成大写。

```python
mystr = "hello world and briup and briup and Python"
print(mystr.capitalize())
# 结果:Hello world and briup and briup and python
```

> 注意：capitalize()函数转换后，只字符串第一个字符大写，其他的字符全都小写。

- title()：将字符串每个单词首字母转换成大写。

```python
mystr = "hello world and briup and briup and Python"
print(mystr.title())
# 结果：Hello World And Briup And Briup And Python
```

- lower()：将字符串中大写转小写。

```python
mystr = "hello world and briup and briup and Python"
print(mystr.lower())
# 结果：hello world and briup and briup and python
```

- upper()：将字符串中小写转大写。

```python
mystr = "hello world and briup and briup and Python"
print(mystr.upper())
# 结果：HELLO WORLD AND BRIUP AND BRIUP AND PYTHON
```

- lstrip()：删除字符串左侧空白字符。

```python
mystr = "      hello world and briup and briup and Python     "
print(mystr.lstrip())
# 结果：'hello world and briup and briup and Python     '
```

- rstrip()：删除字符串右侧空白字符。

```python
mystr = "      hello world and briup and briup and Python     "
print(mystr.rstrip())
# 结果：'      hello world and briup and briup and Python'
```

- 判断
  - startswith()：检查字符串是否是以指定子串开头，是则返回 True，否则返回 False。如果设置开始和结束位置下标，则在指定范围内检查。

- 语法

```
字符串序列.startswith(子串, 开始位置下标, 结束位置下标)
```

- 示例

```
mystr = "hello world and briup and briup and Python   "

print(mystr.startswith('hello'))
# 结果：True

print(mystr.startswith('hello', 5, 20))
# 结果False
```

- endswith()：：检查字符串是否是以指定子串结尾，是则返回 True，否则返回 False。如果设置开始和结束位置下标，则在指定范围内检查。
  - 语法

```
字符串序列.endswith(子串, 开始位置下标, 结束位置下标)
```

  - 示例

```
mystr = "hello world and briup and briup and Python"

print(mystr.endswith('Python'))
# 结果：True

print(mystr.endswith('python'))
# 结果：False

print(mystr.endswith('Python', 2, 20))
# 结果：False
```

- isdigit()：如果字符串只包含数字则返回 True 否则返回 False。
- 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。

6. 列表

6.1 列表--基础

- 普通变量只能存储一个元素，列表可以存储多个元素
- [元素，元素，元素，元素，。。。。]
- 可以存储任何对象
- 支持所有序列操作
- 支持原处修改，可变对象
- 长度可变、异构、可任意嵌套
- 本质是对象的引用组成的数组

6.2 列表--定义

```
L1 = []
L2 = list(range(10))
L3 = [‘1’, ‘2’, ‘3’]
L4 = list(‘123’)
```

6.3 列表--操作

- 增加
  - append()：列表结尾追加数据。
    - 语法

      ```
      列表序列.append(数据)
      ```

    - 示例

      ```
      place_list = ['suzhou', 'beijing', 'shanghai']
      place_list.append('chongqing')
      print(place_list)
      # 结果：['suzhou', 'beijing', 'shanghai', 'chongqing']
      ```

      > 列表追加数据的时候，直接在原列表里面追加了指定数据，即修改了原列表，故列表为可变类型数据。
      >
      > 注意：如果append()追加的数据是一个序列，则追加整个序列到列表

  - insert()：指定位置新增数据。
    - 语法

      ```
      列表序列.insert(位置下标, 数据)
      ```

    - 示例

      ```
      place_list = ['suzhou', 'beijing', 'shanghai']
      place_list.insert(1,'chongqing')
      print(place_list)
      # 结果：['suzhou', 'chongqing', 'beijing', 'shanghai']
      ```

  - extend()：使用新的列表扩展旧的列表的元素
    - 语法

      ```
      列表序列.extend(数据)
      ```

    - 示例

```
place_list = ['suzhou', 'beijing', 'shanghai']
place_list.extend('chongqing')
print(place_list)
# 结果：['suzhou', 'beijing', 'shanghai', 'c', 'h', 'o', 'n',
'g', 'q', 'i', 'n', 'g']
```

> 如果数据是一个序列，则将这个序列的数据逐一添加到列表。

- +：合并两个列表的元素

- 删除
  - remove()：移除列表中某个数据的第一个匹配项。
    - 语法

```
列表序列.remove(数据)
```

    - 示例

```
place_list = ['suzhou', 'beijing', 'shanghai']
place_list.remove('suzhou')
print(place_list)
# 结果：['beijing', 'shanghai']
```

  - pop()：删除指定下标的数据(默认为最后一个)，并返回该数据。
    - 语法

```
列表序列.pop(下标)
```

    - 示例

```
place_list = ['suzhou', 'beijing', 'shanghai']
del_place = place_list.pop(0)
print(del_place)
# 结果：suzhou
print(place_list)
# 结果：['beijing', 'shanghai']
```

  - del：删除一个元素
    - 语法

```
del 目标
```

    - 示例

```python
place_list = ['suzhou', 'beijing', 'shanghai']
del place_list[0]
print(place_list)
# 结果：['beijing', 'shanghai']

place_list = ['suzhou', 'beijing', 'shanghai']
del place_list
print(place_list)
# 报错：NameError: name 'place_list' is not defined
```

- clear()：清空列表

```python
place_list = ['suzhou', 'beijing', 'shanghai']
place_list.clear()
print(place_list) # 结果： []
```

- 修改
  - 修改指定下标数据

```python
place_list = ['suzhou', 'beijing', 'shanghai']
place_list[0] = 'hangzhou'
print(place_list)
# 结果：['hangzhou', 'beijing', 'shanghai']
```

  - 逆置：reverse()

```python
num_list = [1, 5, 2, 3, 6, 8]
num_list.reverse()
# 结果：[8, 6, 3, 2, 5, 1]
print(num_list)
```

  - 排序：sort()
    - 语法

```
列表序列.sort(reverse=False)
```

> 注意：reverse表示排序规则，**reverse = True** 降序， **reverse = False** 升序（默认）

    - 示例

```python
num_list = [1, 5, 2, 3, 6, 8]
num_list.sort()
print(num_list)
# 结果：[1, 2, 3, 5, 6, 8]
num_list.sort(reverse=True)
print(num_list)
# 结果：[8, 6, 5, 3, 2, 1]
```

- 查询
  - 下标

```
place_list = ['suzhou', 'beijing', 'shanghai']

print(place_list[0])  # suzhou
print(place_list[1])  # beijing
print(place_list[2])  # shanghai
```

- index()：返回指定数据所在位置的下标 。
  - 语法

    ```
    index()：返回指定数据所在位置的下标 。
    ```

  - 示例

    ```
    place_list = ['suzhou', 'beijing', 'shanghai']
    print(place_list.index('beijing', 0, 2))  # 1
    ```

  > 注意：如果查找的数据不存在则报错。

- count()：统计指定数据在当前列表中出现的次数。

```
place_list = ['suzhou', 'beijing', 'shanghai']
print(place_list.count('suzhou'))  # 1
```

- len()：访问列表长度，即列表中数据的个数。

```
place_list = ['suzhou', 'beijing', 'shanghai']
print(len(place_list))  # 3
```

- in：判断指定数据在某个列表序列，如果在返回True，否则返回False

```
place_list = ['suzhou', 'beijing', 'shanghai']
print('suzhou' in place_list)
# 结果：True

print('hangzhou' in place_list)
# 结果：False
```

- not in：判断指定数据不在某个列表序列，如果不在返回True，否则返回False

- 复制
  - copy()

```
place_list = ['suzhou', 'beijing', 'shanghai']
place_list2 = place_list.copy()
print(place_list2)
# 结果：['suzhou', 'beijing', 'shanghai']
```

练习1：

创建地区列表、新增列表、现有列表，至少存储3行信息

地区	新增	现有	累计	治愈	死亡
香港	7	171	11531	11155	205
云南	2	68	301	231	2
广东	1	40	2290	2242	8
上海	2	37	1904	1860	7
台湾	2	36	1050	1004	10
四川	2	20	954	931	3

练习2：

向以上三个列表追加第4行数据

在第1个位置插入第5行数据

练习3：

打印香港疫情信息(xx地区新增xx人现存xx人)

将地区列表后2个元素修改为 ["GD","SH"]

打印地区列表元素(一行一个)

倒序打印新增列表元素(一行一个)

练习4：

在地区列表中删除“云南”

在新增列表中删除第1个元素

在现有列表中删除前2个元素

练习5：

八大行星："水星" "金星" "地球" "火星" "木星" "土星" "天王星" "海王星"

-- 创建列表存储4个行星："水星" "金星" "火星" "木星"

-- 插入"地球"、追加"土星" "天王星" "海王星"

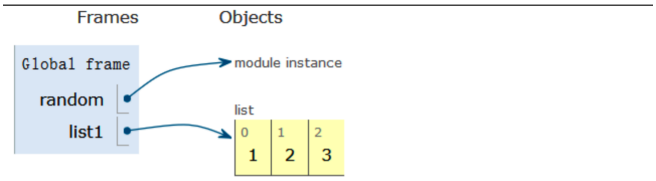
-- 打印距离太阳最近、最远的行星(第一个和最后一个元素)

-- 打印太阳到地球之间的行星(前两个行星)

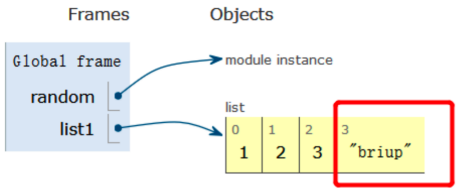
-- 删除"海王星",删除第四个行星

-- 倒序打印所有行星(一行一个)

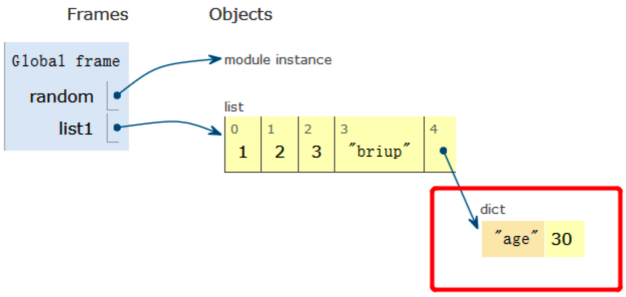
6.4 列表--存储方式



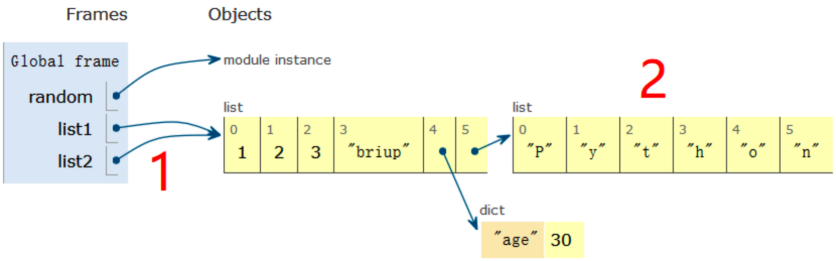
list1 = [1, 2, 3]



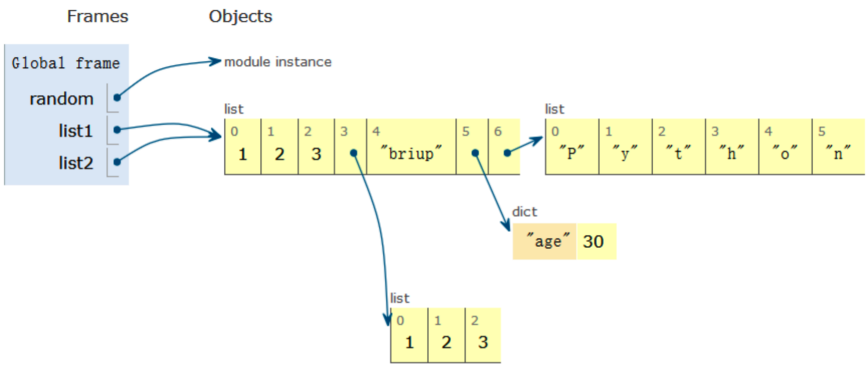
list1.append('briup')



list1.append(dict(age=30))



list2 = list1
list2.append(list('Python'))



list2.insert(3, [1, 2, 3])

6.5 列表--列表推导式

(1) 定义：

使用简易方法，将可迭代对象转换为列表。

(2) 语法：

变量 = [表达式 for 变量 in 可迭代对象]

变量 = [表达式 for 变量 in 可迭代对象 if 条件]

(3) 说明：

如果if真值表达式的布尔值为False，则可迭代对象生成的数据将被丢弃。

```
list01 = [9, 15, 65, 6, 78, 89]
# 需求：在list01中挑出能被3整除的数字存入list02
# list02 = []
# for item in list01:
#     if item % 3 == 0:
#         list02.append(item)
list02 = [item for item in list01 if item % 3 == 0]
print(list02)

# 需求：在list01中所有数字的个位存入list03
# list03 = []
# for item in list01:
#     list03.append(item % 10)
list03 = [item % 10 for item in list01]
print(list03)
```

练习：

生成10--30之间能被3或者5整除的数字

[10, 12, 15, 18, 20, 21, 24, 25, 27]

生成5 -- 20之间的数字平方

[25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361]

(4) 列表推导式嵌套

变量 = [表达式 for 变量1 in 可迭代对象1 for 变量2 in可迭代对象2]

```
#result = []
#for r in ["a", "b", "c"]:
#  for c in ["A", "B", "C"]:
#    result.append(r + c)
result = [r + c for r in list01 for c in list02]
```

6.6 列表与字符串转换

(1) 列表转换为字符串：

result = "连接符".join(列表)

```
list01 = ["a", "b", "c"]
result = "-".join(list01)
print(result)
```

练习：

在终端中，循环录入字符串，如果录入空则停止。

停止录入后打印所有内容(一个字符串)

效果：

请输入内容：香港

请输入内容：上海

请输入内容：新疆

请输入内容：

香港-上海-新疆

(2) 字符串转换为列表：

列表 = "a-b-c-d".split("分隔符")

```
# 使用一个字符串存储多个信息
list_result = "唐僧,孙悟空,八戒".split(",")
print(list_result)
```

练习：将下列英文语句按照单词进行翻转

转换前：To have a government that is of people by people for people

转换后：people for people by people of is that government a have To

7. 元组

7.1 元组--定义及特性

- 理解：不可变的列表，[] -> ()
- （元素1，元素2，元素3，。。。）
- 不可变序列（不可在原处修改）
  - 内容不可变
  - 数据之间的相对位置不可变
- 任意对象的有序集合
- 固定长度、异构、任意嵌套
- 支持所有序列操作，索引、切片

7.2 元组--常见操作

1. 创建空元组：

元组名 = ()

元组名 = tuple()

(2) 创建非空元组：

元组名 = (20,)

元组名 = (1, 2, 3)

元组名 = 100,200,300

元组名 = tuple(可迭代对象)

(3) 获取元素：

变量 = 元组名[索引]

变量 = 元组名[切片] # 赋值给变量的是切片所创建的新元组

(4) 遍历元组：

正向：

for 变量名 in 元组名:

变量名就是元素

反向：

for 索引名 in range(len(元组名)-1,-1,-1):

元组名[索引名]就是元素

```
# 1. 创建
# -- 元组名 = (元素1, 元素2, 元素3)
tuple01 = (10, 20, 30)
# -- 元组名 = tuple( 可迭代对象 )
list01 = ["a", "b", "c"]
tuple02 = tuple(list01)

```

```python
# 2. 定位
# -- 读取(索引 / 切片)
print(tuple01[0]) # 10
print(tuple01[:2]) # (10, 20)

# 3. 遍历
for item in tuple01:
    print(item)

for i in range(len(tuple01) - 1, -1, -1):
    print(tuple01[i])

# 4. 特殊
# 注意1：小括号可以省略
tuple03 = 10, 20, 30
# 注意2：如果元组中只有一个元素,必须有逗号
tuple04 = (10,)
# 拆包：  多个变量  = 容器
# a,b,c = tuple03
# a,b,c = ["A","B","C"]
a,b,c = "孙悟空"
*a,b = "孙悟空"    # *表示该变量接受任意多个值
print(a) # ['孙', '悟']
print(b) # 空
```

元组数据不支持修改，只支持查找，具体如下：

- 按下标查找数据

```python
tuple1 = ('b', 'r', 'i', 'u', 'p')
print(tuple1[0])  # b
```

- index()：查找某个数据，如果数据存在返回对应的下标，否则报错，语法和列表、字符串的index方法相同。

```python
tuple1 = ('b', 'r', 'i', 'u', 'p')
print(tuple1.index('b'))  # 0
```

- count()：统计某个数据在当前元组出现的次数。

```python
tuple1 = ('b', 'r', 'i', 'u', 'p', 'b')
print(tuple1.count('b'))  # 2
```

- len()：统计元组中数据的个数。

```python
tuple1 = ('b', 'r', 'i', 'u', 'p')
print(len(tuple1))  # 5
```

> 注意：元组内的数据如果直接修改则立即报错
>
> 但是如果元组里面有列表，修改列表里面的数据则是支持的

练习：

根据月日，计算是这一年的第几天。

公式：前几个月总天数 + 当月天数

例如：2020年5月10日

计算：31 29 31 30 + 10

7.3 作用

(1) 元组与列表都可以存储一系列变量，由于列表会预留内存空间，所以可以增加元素。

(2) 元组会按需分配内存，所以如果变量数量固定，建议使用元组，因为占用空间更小。

(3) 应用：

变量交换的本质就是创建元组：x, y = (y, x)

格式化字符串的本质就是创建元祖："姓名：%s, 年龄：%d" % ("briup", 15)

8. 字典

8.1 字典--结构

`{‘name’:‘Lucky’, ‘age’:18, 'phone':'13245678987')`

`{ 键：值， 键：值 .... }`

8.2 字典--特点

- 通过”key”来访问数据
- 任意对象的无序集合
- 可变长、异构、任意嵌套
- 可变对象，支持原处修改

8.3 字典--操作

- 增加

  写法：**字典序列[key] = 值**

  > 注意：如果key存在则修改这个key对应的值；如果key不存在则新增此键值对。

  ```
  dict1 = {'name': 'Tom', 'age': 20, 'gender': '男'}
  dict1['name'] = 'Rose'
  print(dict1)
  # 结果：{'name': 'Rose', 'age': 20, 'gender': '男'}
  dict1['id'] = 110
  print(dict1)
  # {'name': 'Rose', 'age': 20, 'gender': '男', 'id': 110}
  ```

  > 注意：字典为可变类型。

- 删除

- del() / del：删除字典或删除字典中指定键值对。

```
dict1 = {'name': 'Tom', 'age': 20, 'gender': '男'}
del dict1['gender']
print(dict1)
# 结果：{'name': 'Tom', 'age': 20}
```

- clear()：清空字典

```
dict1 = {'name': 'Tom', 'age': 20, 'gender': '男'}
dict1.clear()
print(dict1)  # {}
```

- 修改

  写法：**字典序列[key] = 值**

  > 注意：如果key存在则修改这个key对应的值；如果key不存在则新增此键值对。

- 查询
  - key值查找

```
dict1 = {'name': 'Tom', 'age': 20, 'gender': '男'}
print(dict1['name'])  # Tom
print(dict1['id'])  # 报错
```

> 如果当前查找的key存在，则返回对应的值；否则则报错。

  - get()
    - 语法

```
字典序列.get(key, 默认值)
```

> 注意：如果当前查找的key不存在则返回第二个参数(默认值)，如果省略第二个参数，则返回None。

    - 示例

```
dict1 = {'name': 'Tom', 'age': 20, 'gender': '男'}
print(dict1.get('name'))  # Tom
print(dict1.get('id', 110))  # 110
print(dict1.get('id'))  # None
```

  - keys()

```
dict1 = {'name': 'Tom', 'age': 20, 'gender': '男'}
print(dict1.keys())  # dict_keys(['name', 'age', 'gender'])
```

  - values()

```
dict1 = {'name': 'Tom', 'age': 20, 'gender': '男'}
print(dict1.values())  # dict_values(['Tom', 20, '男'])
```

- items()

```
dict1 = {'name': 'Tom', 'age': 20, 'gender': '男'}
print(dict1.items())  # dict_items([('name', 'Tom'), ('age', 20),
('gender', '男')])
```

8.4 字典--注意事项

- 序列操作对字典无效
- 对不存在的索引赋值会增加新项
- ‘key’必须是[可哈希]的，数字，字符串
- 字典是无序存储的
- 通过哈希算法存储key值，查找速度块
- 查询速度不会因为元素增多而变慢
- 额外存储key值，空间换时间

练习1：

创建字典存储香港信息、字典存储云南信息、字典存储广东信息、字典存储云南信息

地区	新增	现有	累计	治愈	死亡
香港	7	171	11531	11155	205
云南	2	68	301	231	2
广东	1	40	2290	2242	8
上海	2	37	1904	1860	7
台湾	2	36	1050	1004	10
四川	2	20	954	931	3

练习2：

在终端中打印香港的现有人数

在终端中打印上海的新增和现有人数

广东新增人数增加1

练习3：

删除香港现有人数信息

删除广东新增人数信息

删除上海的新增和现有信息

练习4：

在终端中打印香港字典的所有键(一行一个)

在终端中打印上海字典的所有值(一行一个)

在终端中打印广东字典的所有键和值(一行一个)

在云南字典中查找值是68对应的键名称

8.5 字典推导式

(1) 定义：

使用简易方法，将可迭代对象转换为字典。

(2) 语法：

{键:值 for 变量 in 可迭代对象}

{键:值 for 变量 in 可迭代对象 if 条件}

练习1：

将两个列表，合并为一个字典

姓名列表["唐僧","悟空","八戒"]

房间列表[101,102,103]

{101: '唐僧', 102: '悟空', 103: '八戒'}

练习2：

颠倒练习1字典键值

{'张无忌': 101, '赵敏': 102, '周芷若': 103}

9. 集合

9.1 定义

(1) 由一系列不重复的不可变类型变量(元组/数/字符串)组成的可变散列容器。

(2) 相当于只有键没有值的字典(键就是集合的数据)。

9.2 基础操作

(1) 创建空集合：

集合名 = set()

集合名 = set(可迭代对象)

(2) 创建具有默认值集合：

集合名 = {1, 2, 3}

集合名 = set(可迭代对象)

```
# 创建
# -- 集合名 = {元素1,元素2,元素3}
set01 = {"悟空", "唐僧", "八戒"}

list01 = ["唐僧", "悟空", "唐僧", "八戒", "唐僧"]
# -- 集合名 = set(可迭代对象)
set02 = set(list01)
print(set02) # {'八戒', '悟空', '唐僧'}
```

(3) 添加元素：

集合名.add(元素)

```
# 添加：集合名.add(元素)
set02.add("小白龙")
print(set02) #  {'悟空', '八戒', '小白龙', '唐僧'}

# 定位
#(因为无序不能使用索引切片)
#(因为不是键值对不能使用键查找值)

# 遍历
for item in set02:
    print(item)
```

(4) 删除元素：

集合名.discard(元素)

```
# 4. 删除
if "悟空1" in set01:
    set01.remove("悟空1")

# set01.remove("悟空1") # 如果不存在,会报错
set01.discard("悟空1")# 如果不存在,也不报错
```

9.3 运算

(1) 交集&：返回共同元素

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = s1 & s2 # {2, 3}
```

(2) 并集|：返回不重复元素

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = s1 | s2 # {1, 2, 3, 4}
```

(3) 补集-：返回只属于其中之一的元素

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s1 - s2 # {1} 属于s1但不属于s2
```

(4) 补集^：返回不同的的元素

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = s1 ^ s2 # {1, 4}  等同于(s1-s2 | s2-s1)
```

(5) 子集<：判断一个集合的所有元素是否完全在另一个集合中

(6) 超集>：判断一个集合是否具有另一个集合的所有元素

```
s1 = {1, 2, 3}
s2 = {2, 3}
s2 < s1 # True
s1 > s2 # True
```

(7) 相同或不同== !=：判断集合中的所有元素是否和另一个集合相同

```
s1 = {1, 2, 3}
s2 = {3, 2, 1}
s1 == s2 # True
s1 != s2 # False
```

子集或相同，超集或相同 <= >=

练习：一家公司有如下岗位：

"经理"："曹操","刘备","孙权"

"技术"："曹操","刘备","张飞","关羽"

1. 定义数据结构，存储以上信息。
2. 是经理也是技术的都有谁?
3. 是经理不是技术的都有谁?
4. 不是经理是技术的都有谁?
5. 身兼一职的都有谁?
6. 公司总共有多少人数?

9.4 集合推导式

(1) 定义：

使用简易方法，将可迭代对象转换为集合。

(2) 语法：

{表达式 for 变量 in 可迭代对象}

{表达式 for 变量 in 可迭代对象 if 条件}

```
# 将列表中姓名字数为2个的存入集合，名字不重复
list_person = ["唐僧", "白龙马", "唐僧", "八戒", "唐僧"]
set_result = {name for name in list_person if len(name) == 2}
```

10. 数据类型总结

10.1 按操作划分

- 数字
  包括整数、浮点数、二进制、分数等。支持算术运算、逻辑运算、位运算等。
- 序列
  包括字符串、列表、元组，支持索引、分片、合并、迭代等操作。
- 映射
  主要指字典，以key-value的形式存储和索引。

10.2 按存储方式划分

- 不可变对象
  数字、字符串、元组以及不可变的集合，不支持原处修改，每次修改会创建新的对象。
- 可变对象
  列表、字典、可变集合，支持原处修改。

11. 深拷贝与浅拷贝

- （浅拷贝 拷贝地址）mylist2 = mylist1 两个变量指向同一个对象
- （深拷贝 拷贝数据）mylist2 = mylist1.copy() mylist2指向新的对象，内容是对mylist1对象的拷贝
- Python中默认都是浅拷贝
- 深拷贝：import copy; copy.deepcopy

举例：

浅拷贝

```
mylist1 = ['suzhou', 'hangzhou', 'beijing', 'shanghai']
mylist2 = mylist1
print(mylist2)
#结果为：['suzhou', 'hangzhou', 'beijing', 'shanghai']
mylist1[0] = 'xian'
print(mylist1)
#结果为：['xian', 'hangzhou', 'beijing', 'shanghai']
print(mylist2)
#结果为：['xian', 'hangzhou', 'beijing', 'shanghai']
```

> 像上面这样，把mylist1的值赋值给mylist2, 修改了mylist1后，mylist2也被改变。这被称为浅拷贝。

深拷贝

```python
mylist1 = ['suzhou', 'hangzhou', 'beijing', 'shanghai']
mylist2 = mylist1.copy()
print(mylist2)
# 结果为：['suzhou', 'hangzhou', 'beijing', 'shanghai']
mylist1[0] = 'xian'
print(mylist1)
# 结果为：['xian', 'hangzhou', 'beijing', 'shanghai']
print(mylist2)
# 结果为：['suzhou', 'hangzhou', 'beijing', 'shanghai']
```

> 像上面这样，mylist2复制了mylist1, 修改了mylist1后，mylist2不变。这被称为深拷贝。

```python
"""
拷贝：数据备份,防止数据意外改变
浅拷贝:备份第一层数据
    优点：占用内存较少
    缺点：当深层数据变化时,互相影响
深拷贝:备份所有数据
    优点：绝对互不影响
    缺点：占用内存过多
适用性：
   优先使用浅拷贝,当深层数据有可能被修改时,使用深拷贝
"""
# 直接赋值
list01 = [[10, 20], 30]
list02 = list01
list02[0][0] = 100
list02[1] = 300
print(list01)

# 浅拷贝
list01 = [[10, 20], 30]
list03 = list01[:]
list03[0][0] = 100 # 影响
list03[1] = 300 # 不影响
print(list01)

# 深拷贝
import copy
list01 = [[10, 20], 30]
list04 = copy.deepcopy(list01)
list04[0][0] = 100 # 不影响
list04[1] = 300 # 不影响
print(list01)
```

12. 容器综合训练

练习1：在终端中打印如下图形

$

$$

$$$
$$$$

练习2：二维列表

```
list01 = [
     [1, 2, 3, 4, 5],
      [6, 7, 8, 9, 10],
      [11, 12, 13, 14, 15],
]
```

1. 将第一行从左到右逐行打印

效果：

1

2

3

4

5

2. 将第二行从右到左逐行打印

   效果：

   10

   9

   8

   7

   6

3. 将第三列从上到下逐个打印

   效果：

   3

    8

   13

4. 将第四列从下到上逐个打印

   效果：

   14

   9

   4

5. 将二维列表以表格状打印

   效果：

   1 2 3 4 5

   6 7 8 9 10

   11 12 13 14 15

练习3：多个人的多个爱好

```
dict_hobbies = {
  "于谦": ["抽烟", "喝酒", "烫头"],
  "郭德纲": ["说", "学", "逗", "唱"],
}
```

1. 打印于谦的所有爱好(一行一个)

   效果：

   抽烟

   喝酒

   烫头

2. 计算郭德纲所有爱好数量

   效果：4

3. 打印所有人(一行一个)

   效果：

   于谦

   郭德纲

4. 打印所有爱好(一行一个)

   抽烟

   喝酒

   烫头

   说

   学

   逗

   唱

练习4：

```
dict_travel_info = {
    "北京": {
        "景区": ["长城", "故宫"],
        "美食": ["烤鸭", "豆汁焦圈", "炸酱面"]
  },
    "四川": {
        "景区": ["九寨沟", "峨眉山"],
        "美食": ["火锅", "兔头"]
  }
}
```

1. 打印北京的第一个景区

   效果：

   长城

2. 打印四川的第二个美食

效果:

兔头

3. 所有城市 (一行一个)

   效果:

   北京

   四川

4. 北京所有美食(一行一个)

   效果:

   烤鸭

   豆汁焦圈

   炸酱面

5. 打印所有城市的所有美食(一行一个)

   效果:

   烤鸭

   豆汁焦圈

   炸酱面

   火锅

   兔头

练习5:

对数字列表进行升序排列（小 --> 大）

练习6:

```
# 商品字典
dict_commodity_infos = {
  1001: {"name": "屠龙刀", "price": 10000},
  1002: {"name": "倚天剑", "price": 10000},
  1003: {"name": "金箍棒", "price": 52100},
  1004: {"name": "口罩", "price": 20},
  1005: {"name": "酒精", "price": 30},
}
# 订单列表
list_orders = [
  {"cid": 1001, "count": 1},
  {"cid": 1002, "count": 3},
  {"cid": 1005, "count": 2},
]
```

1. 打印所有商品信息

   格式：商品编号xx，商品名称xx，商品单价xx

2. 打印所有订单中的信息

   格式：商品编号xx，购买数量xx

3. 打印所有订单中的商品信息

   格式：商品名称xx，商品单价xx，数量xx

4. 查找数量最多的订单(使用自定义算法，不使用内置函数)

5. 根据购买数量对订单列表降序(大->小)排列