

Python语法基础

课程内容：

- 知识点回顾
- 容器变量
- 列表生成式

知识点回顾(5): 课后作业选讲

编程计算: $1+2+\dots+n$, n 为一个从键盘输入的数字。

```
n = input("请输入数字n:>>")
```

调用input方法, 接收从键盘输入的字符串 (回车结束输入)

```
n = int(n)
```

由于input的返回结果是str, 所以需要调用python内置的int方法, 把字符串变量转化成int变量 (容易犯错, 不做类型转换而直接使用, 会报错)

```
total = 0  
for i in range(1, n+1):  
    total += i
```

调用for... in range()方法, 还有+=运算符, 将 $1+2+\dots n$ 的计算结果保存到total, 注意range顾头不顾尾, 要做1到n的连加, 所以是range(1, n+1)。

```
print(f"1+2+...{n} = {total}")
```

调用print方法, 打印输出结果。

等价于:

```
str_out = f"1+2+...{n} = {total}" # 先调用f语法, 构造拟输出字符串  
print(str_out) # 再调用print方法, 输出字符串到屏幕
```

最终的输出结果

```
请输入数字n:>>12  
1+2+...12 = 78
```

知识点回顾(2): 深入理解变量的定义过程

c/c++过程

强类型语言

内存与变量固定绑定

变量与类型固定绑定

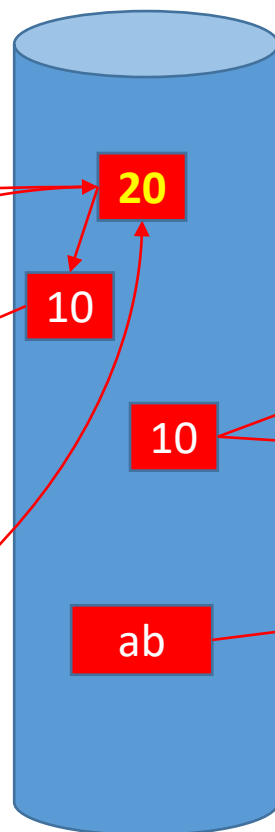
```
int a = 10 ;
```

```
int b = a;
```

```
a = 'ab'; 报错
```

```
a = 20;
```

内存



python

弱类型语言

内存与类型固定绑定

变量与内存动态绑定

```
a = 10
```

——> 数据类型绑定位置不同

```
b = a
```

——> 与c不同，没有新开辟内存

```
a = 'ab'
```

——> 与c不同，开辟了内存

课程内容：

- 知识点回顾
- 容器变量
- 列表生成式
- 课后作业

通过计算机编程，“把大象装进冰箱”，可以分成三步：打开冰箱门、驱使大象进入冰箱、关上冰箱门。



■ 实操时面临的问题：

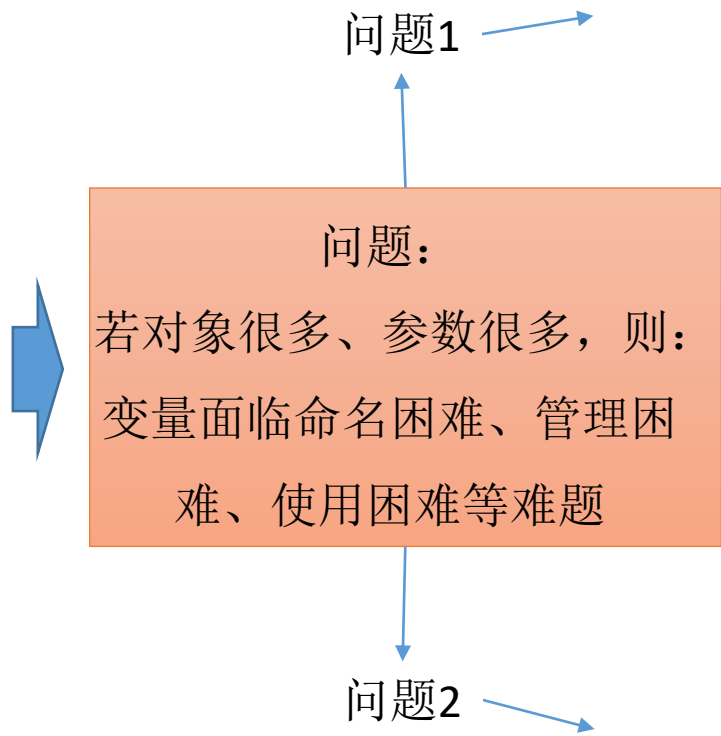
- 大象会有很多，被关进冰箱的是哪一头大象呢？程序需要知道该大象的名字（`name, string`）或身份ID（`id, int`）
- 冰箱容量有限，能否容下该大象？程序需要知道该大象的尺寸，长(`length, float`)、宽(`width, float`)、高（`height, float`）

■ 因此，程序要实现与某物理实体进行交互，得根据实际需求，抽象出物理实体的多维度信息，并在内存中进行一一结构化建模，每个物理实体都将对应多个参数变量。

引入容器变量的原因：

变量：物理对象数字化建模

物理对象	参数	变量	类型
人	姓名	name	字符串
	年龄	age	整型数
	身高	height	浮点数
	收入	salary	浮点数
	性别	sex	整型数 布尔型



问题1：

如何表达不同人的名称, 怎么办？
一个name变量不够用了，name1, name2, name3? ? ?

自明性太差。

问题2：

各散装式变量之间相互独立，如何访问某一物理实体的某个参数变量？

做不到！！

引入容器变量的原因：

变量：物理对象数字化建模

物理对象	参数	变量	类型
人	姓名	name	字符串
	年龄	age	整型数
	身高	height	浮点数
	收入	salary	浮点数
	性别	sex	整型数 布尔型



问题：
若对象很多、参数很多，
则：变量面临命名困难、
管理困难、使用困难等
难题

需求1

诉求1：
能否用一个变量来
描述（或存储）一
个人的多个变量信
息？



需求2

诉求2：
能否用一个变量来
存储多个人的所有
信息？

容器变量



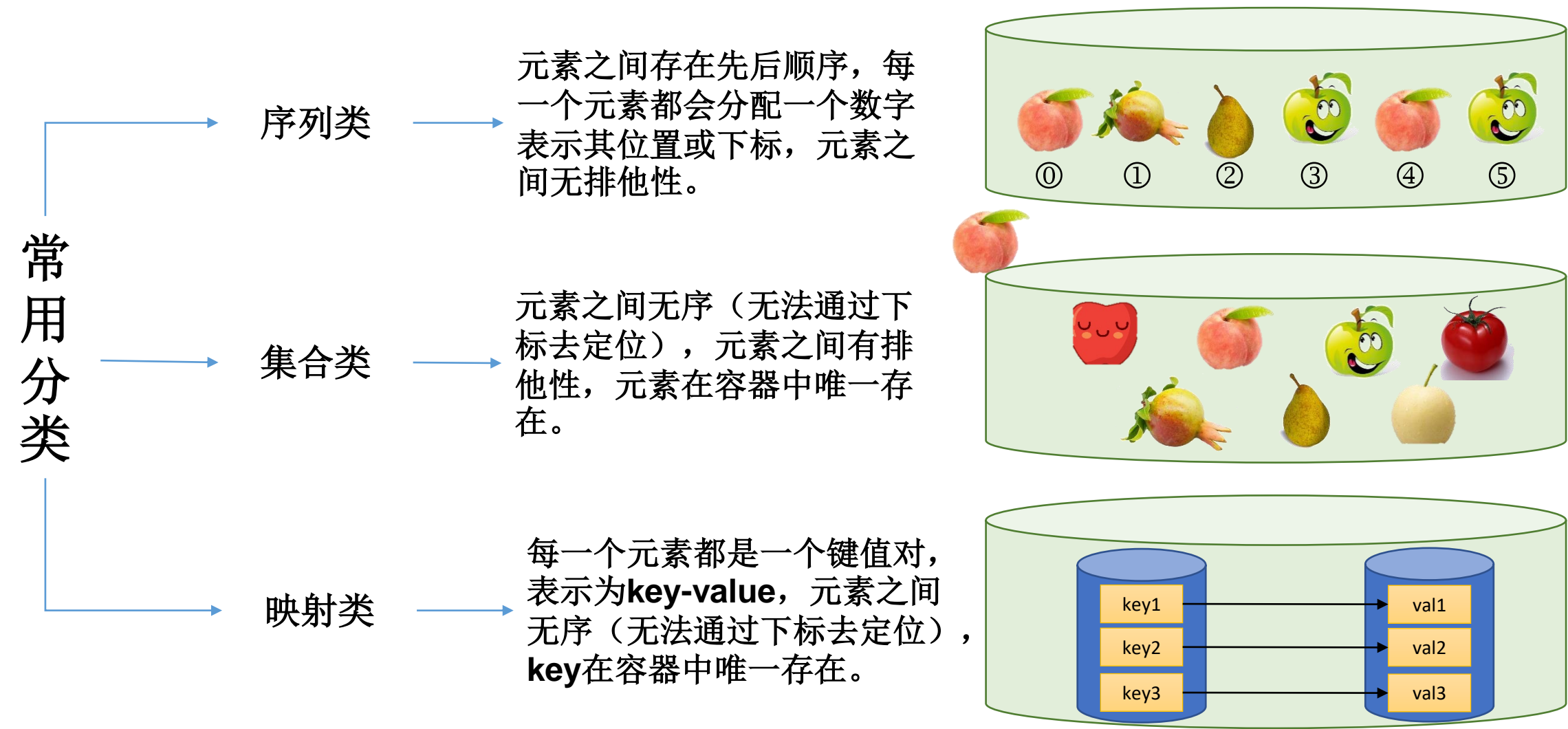
课程目的

1. 常用容器变量的基本概念与增删改查操作：
 - 线性类： 列表(**list**)/元组(**tuple**)/字符串(**str**)
 - 集合类： 集合(**set**)
 - 映射类： 字典(**dict**)
2. Python语法糖： 列表生成式(**List generation**)
 - 基本概念
 - 使用方法

掌握更多的、更实用的语法糖，让你的程序更优雅，
让你的代码更**pythonic**!

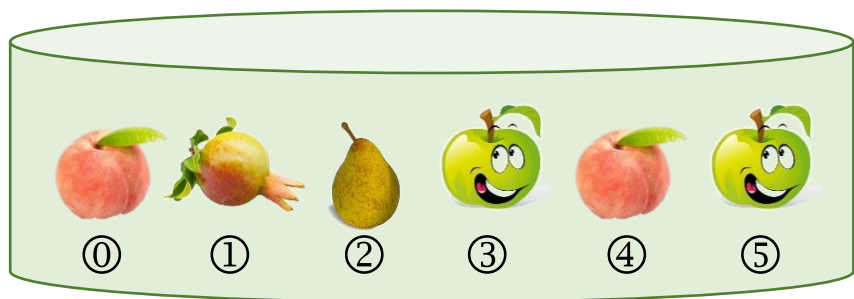
容器的概念:

把多个元素组织起来的数据结构，不同类型的数据结构对应元素不同的组织方式。



容器变量-列表(list)

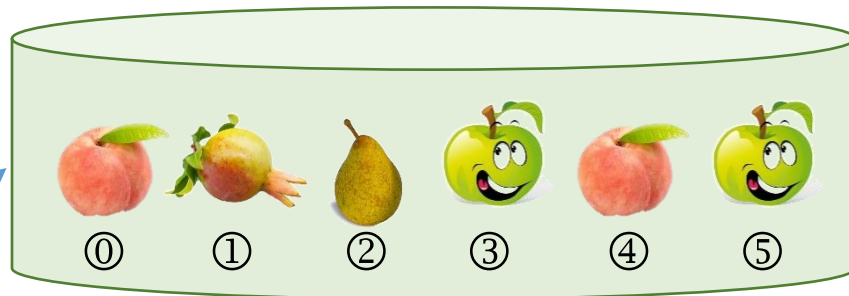
■ Python中最基本的数据结构，序列类结构，类比于C++的数组。



■ 与 C 数组相同的性质：

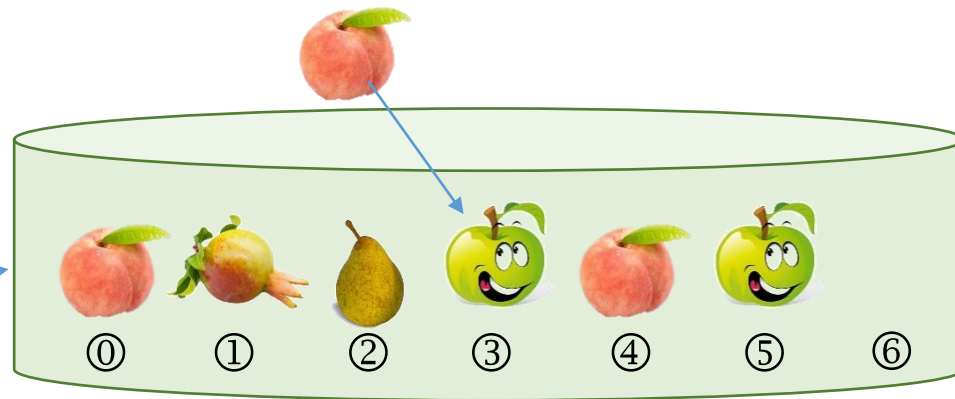
- ✓ 元素之间存在先后顺序
- ✓ 各元素拥有下标(从0开始)
- ✓ 元素之间无排他性
- ✓ 元素之间紧密排列

从列表中
删除元素



删除、左移、长度-1

向列表
中插入元素

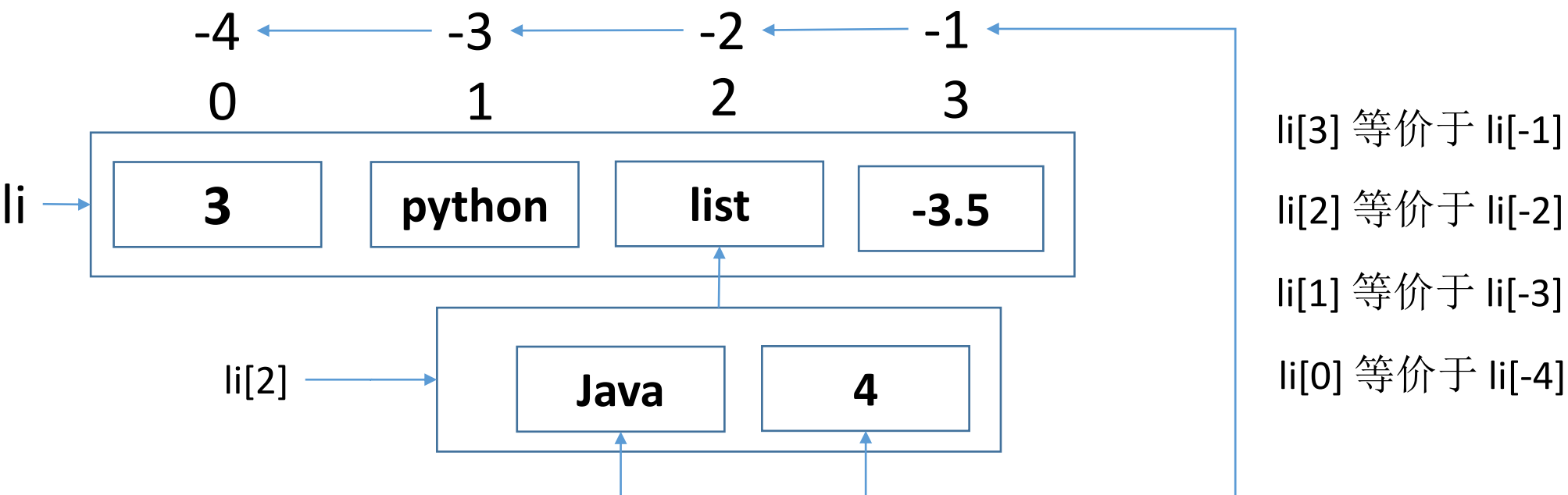


长度+1、右移、插入

容器变量-列表(list)- 创建

■ 列表创建方法：用逗号分隔的不同数据项，并使用方括号括起来，示例：

```
li=[3, 'python', ['Java',4], -3.5]
```



Python 特色功能

元素类型可不相同
支持内嵌操作

类似于2维数组的访问

支持逆向下标，便于从后向前查找元素

容器变量-列表(list)-从内存角度，深入理解列表的构建过程

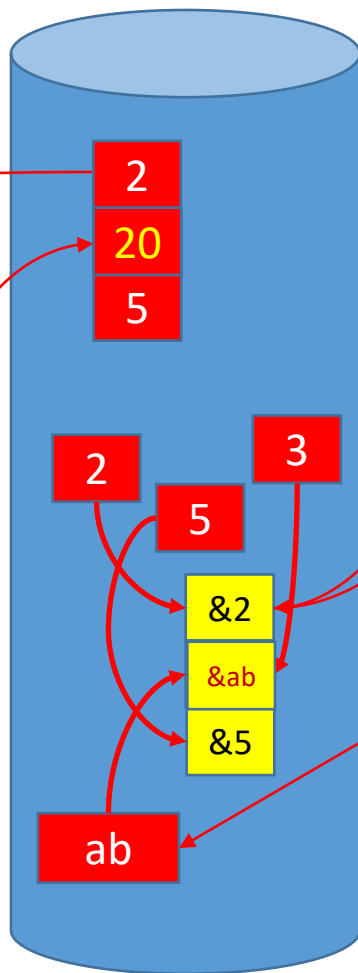
c/c++过程

```
int a[3] = {2,3,5}
```

```
a[1] = 20
```

数组元素就是物理对象
对象地址连续，紧密排列
变量赋值等于修改内存值

内存



python

```
a = [2,3,5] → a里存储的是list对象的地址
```

```
b = a
```

思考: b[1] = ???

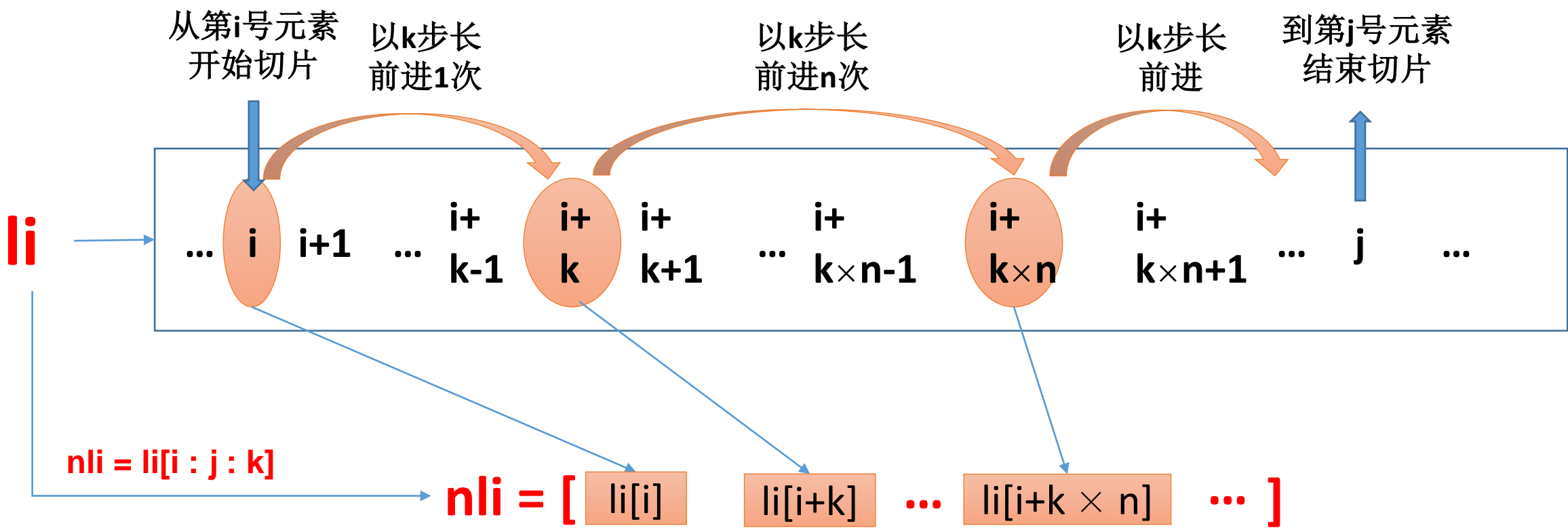
```
a[1] = 'ab' → 开辟了内存，指向新地址
```

列表元素是物理对象的引用
对象地址可以不连续，不需要紧密排列
变量赋值等于变量指向新的内存地址

容器变量-列表(list)-切片

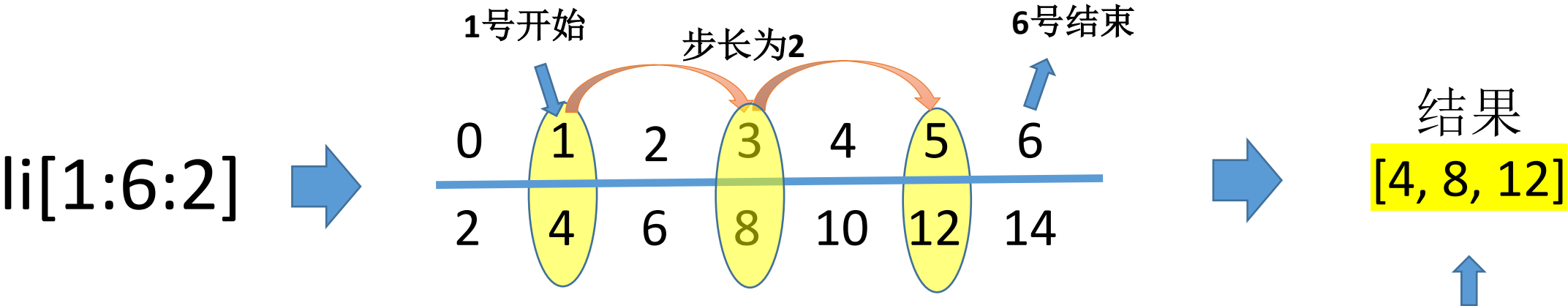
■ **list**提供了一种语法糖：切片， 可以根据规则从原列表中切出一个新列表。

$nli = li[i : j : k]$

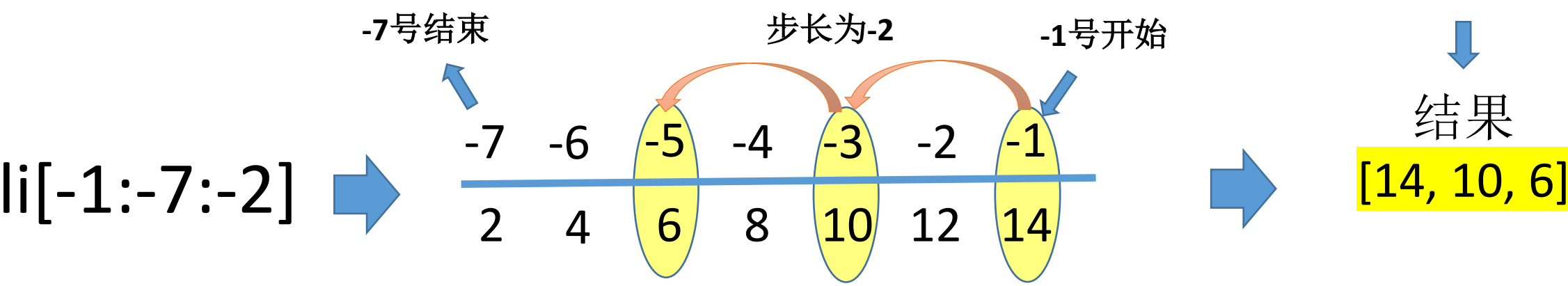


容器变量-列表(list)-切片

■ 以li=[2,4,6,8,10, 12, 14]为例，切片操作的含义与结果。



■ 当步长为负时， 代表逆向操作:



条件: $li = [0, 1, 2, 4, 6, 7, 9]$

问题: $li[-1:0:-2] = ?$

- ☐ A $[0, 2, 4, 7]$
- ☐ B $[9, 6, 2, 0]$
- ☒ C $[9, 6, 2]$
- ☐ D 不知道, 有点烧脑

提交

容器变量-列表(list)-切片

■ **$nli = li[i:j:k]$** ， k 默认为1，当 $k > 0$ 时， i 默认为0， j 默认为`list`长度， k 默认是

■ ¹因此，以`li=[0,2,4,6,8,10]`为例，长度 6，以下操作等价：

✓ `li[1:4:1]` 等价于 `li[1:4]`

✓ `li[0:5:2]` 等价于 `li[:5:2]`

✓ `li[2:6:2]` 等价于 `li[2::2]`

✓ `li[0:4:1]` 等价于 `li[:4:]`

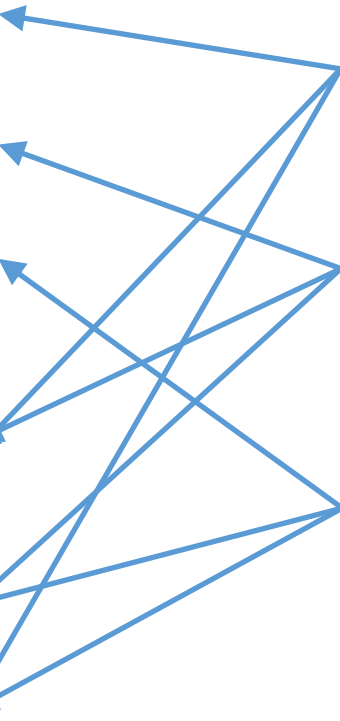
✓ `li[0:6:2]` 等价于 `li[::2]`

✓ `li[2:6:1]` 等价于 `li[2::]`

■ 步长(k)默认取值为1

■ 开始(i)默认取值为0

■ 结束(j)默认取值为6



容器变量-列表(list)-切片

■ **$nli = li[i:j:k]$** ，若 $k < 0$ 时， i 默认取值为-1， j 默认为 $-(li\text{长度})$ 。

以 $li=[0,2,4,6,8,10]$ 为例，长度6，以下操作等价：

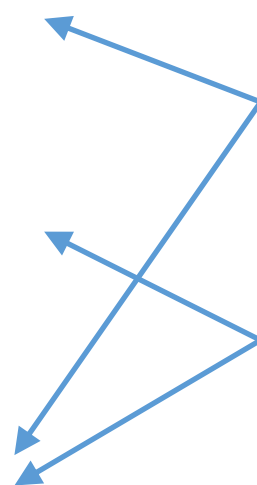
✓ $li[-1:-4:-1]$ 等价于 $li[:-4:-1]$

✓ $li[-2:-7:-2]$ 等价于 $li[-2::-2]$

✓ $li[-1:-7:-2]$ 等价于 $li[::-2]$

■ 开始(i)默认取值为-1

■ 结束(j)默认取值为-6，



容器变量-列表(list)-增删改查

■ 除了切片之外，**list** 提供了一系列内置函数，以实现在列表中添加元素：

增：li = [1,2,4]

接口	使用方法	含义	示例与结果
append	li.append(ele)	在li之后添加新元素ele	li.append(3) 结果：li=[1,2,4,3]
insert	li.insert(pos, ele)	在li的第pos个位置添加元素ele， pos 之后元素向后顺移	li.insert(1, 3) 结果：li=[1,3,2,4]
extend	li.extend(ins_li)	遍历ins_li元素，逐次插入到列表之后	li.extend([4,5]) 结果：li=[1,2,4,4,5]

注意： **li.append([4,5])** 与 **li.extend([4,5])**的区别。

容器变量-列表(list)-增删改查

■ 除了切片之外，list 提供了一系列内置函数，以实现删除列表元素的操作：

删：li = [1,2,4]

接口	使用方法	含义	示例与结果
clear	li.clear()	清空列表内容	li.clear() 结果：li=[]
pop	ele = li.pop()	将列表最后一个元素从列表中删除，并将最后一个元素赋值给ele	ele = li.pop() 结果：li = [1,2], ele = 4
remove	li.remove(ele)	从列表中删除 值为ele 的元素（若存在多个相同元素，只删除第一个查询到的元素）	li.remove(2) 结果：li=[1,4]
del	del li[slice]	根据切片选择列表元素，并删掉被选中元素	del li[1::] 结果：li=[1]

容器变量-列表(list)-增删改查

■ 除了用下标修改列表元素外，list还提供了一系列内置函数，以实现修改操作：

以li = [1,2,4] 为例：

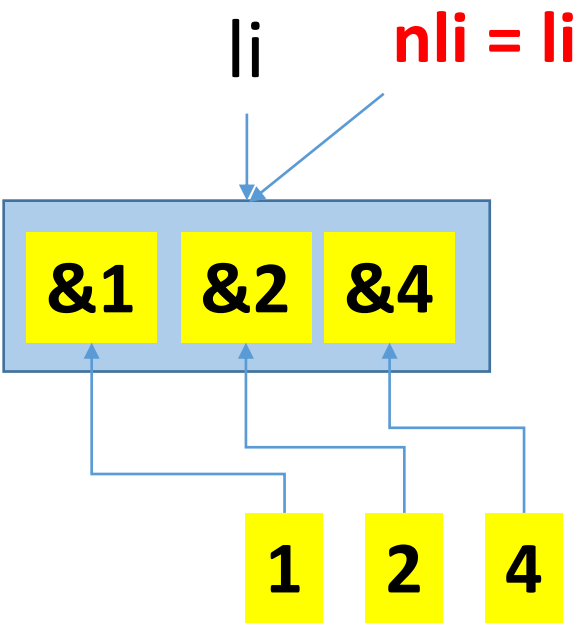
接口	使用方法	含义	示例与结果
reverse	li.reverse()	元素逆序排列，等价于 li = li[::-1]操作。	li.reverse() 结果: li = [4,2,1]
sort	li.sort(reverse=False)	按照值做从小到大排列, 若 reverse=True, 则按照从大到小排列	li.sort(reverse=True) 结果: [4,2,1]
copy	nli = li.copy()	列表元素复制一份，生成一个新的列表内存，将新的内存引用赋给nli	nli = li.copy() 结果: [1,2,4]

注意： nli = li 与 nli = li.copy() 的区别：

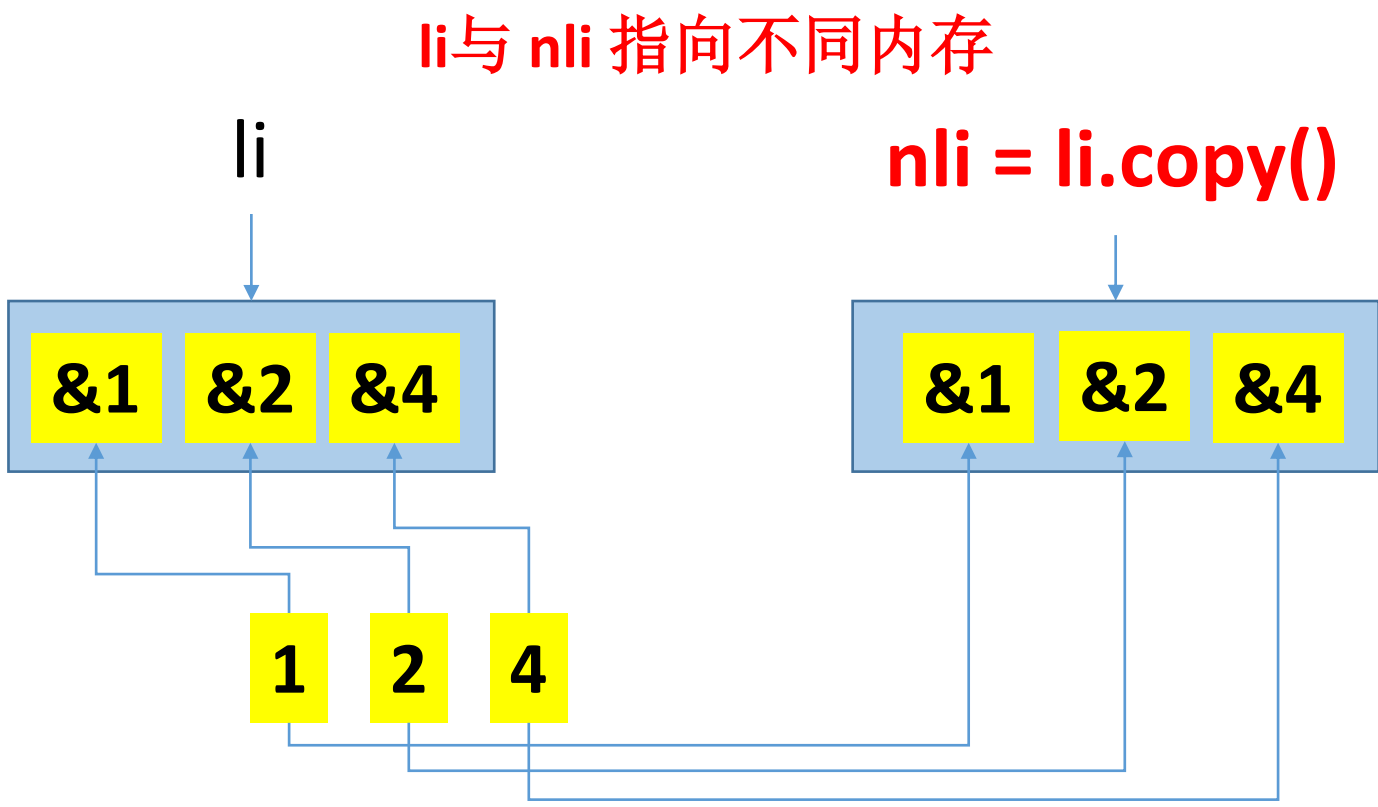
容器变量-列表(list)-增删改查

■ 注意: `nli = li` 与 `nli = li.copy()` 的区别:

以 `li = [1,2,4]` 为例:



没有开辟新的内存



li[i]与nli[i]指向同一内存, 引用复制

前提条件: $li = [1, 2, 3, 4]$, $nli = li$

问 题: 若 $nli[1] = 10$, 则 $li[1] = ?$

- ☐ A 1
- ☒ B 2
- ☐ C 10
- ☐ D 不知道, 有点烧脑。

提交

前提条件: $li = [1, 2, 3, 4]$, $nli = li.copy()$

问 题: 若 $nli[1] = 10$, 则 $li[1] = ?$

- ☐ A 1
- ☐ B 2
- ☒ C 10
- ☐ D 不知道, 有点烧脑。

提交

前提条件: `li = [1, [1, 2, 4, 5], 3, 4]`, `nli = li.copy()`

问 题: 若 `li[1][1] = 10`, 则 `nli[1] = ?`

- ☐ A [1, 2, 4, 5]
- ☒ B [1, 10, 4, 5]
- ☐ C [10, 2, 4, 5]
- ☐ D 不知道, 更烧脑了。

想实现深度copy, 办法:
`import copy`
`nli = copy.deepcopy(li)`

提交

容器变量-列表(list)-增删改查

■ 除了直接用下表对列表进行查询操作外，list还提供了一系列内置的查询方法：

查：以li = [1,2,4] 为例：

接口	使用方法	含义	示例与结果
index	idx = li.index(ele)	从左到右遍历，返回ele元素在该列表中的下标，若不存在，则运行报错	idx = li.index(1) 结果: idx =0
count	num = li.count(ele)	返回元素值在该列表中的数量，若不存在，则返回0	num =li.count(5) 结果: num = 0
in /not in	flag = ele in li flag = ele not in li	判断某元素是否在该列表中	flag = 6 not in li flag = True
len/max/min	len(li)/max(li) /min(li)	返回列表的长度，最大值、最小值	结果： 3、 4、 1

容器变量-列表(list)-遍历

■ 遍历列表，python 提供的方法是 `for ... in`:

■ 方案1，直接遍历元素

```
for ele in li:  
    print(ele)
```

```
>>> li=[1,2,4]  
>>> for ele in li:  
...     print(ele)  
...  
1  
2  
4
```

此方法缺点：
找不到元素下标。

■ 方案2，调用**enumerate**函数，同时对元素下标和元素值进行遍历：

```
for idx, ele in enumerate(li):  
    print(idx,ele)
```

idx: 元素下标, **ele**: 元素值

```
>>> li=[1,2,4]  
>>> for idx, ele in enumerate(li):  
...     print(idx, ele)  
...  
0 1  
1 2  
2 4
```

容器变量-列表(list)-常见操作的运行错误:

对一个空列表执行pop操作

```
>>> li = []
>>> li.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: pop from empty list
>>>
```

对不存在的元素执行index操作

```
>>> li = [2]
>>> li.index(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 1 is not in list
>>>
```

访问下标越界

```
>>> li=[2]
>>> li[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

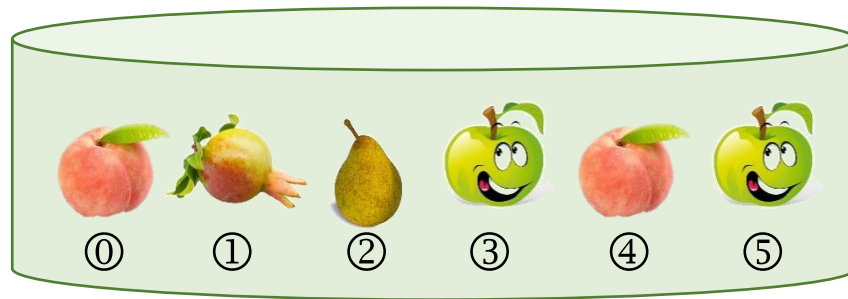
调用不存在的内置方法

```
>>> li.find(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'list' object has no attribute 'find'
>>>
```

提示: 熟练掌握根据python的报错提示, 去定位问题。

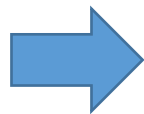
容器变量-元组(tuple)

■ **list** 是 **Python**中最基本的数据结构，类比于**c**语言中的数组。



■ 问题：我定义一个列表变量，可以给别人使用时，但不想让别人去修改。也就是说，在变量传递过程中，我想锁定变量，只读，怎么半？

C语言：const修饰符



python:

1. `nli = li.copy()`

每次都开辟内存，效率太低

2. 使用元组变量 **tuple**

容器变量-元组(tuple)

■ 相当于一种特殊的list，元素不可变。

■ 构造方式：

➤ `tp=(1,2,3)`

➤ `tp=1,2,3`

■ 与list的相关转换：

➤ `tp=tuple(li)`

➤ `li=list(tp)`

```
>>> tp = (1, 2,3)
>>> tp.append(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
```

```
>>> tp = (1, 2,3)
>>> tp[0] = 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

```
>>> tp = (1, 2,3)
>>> tp.clear()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'clear'
```

容器变量-元组(tuple)

- 类似于列表操作，元组可以访问整个元组中的元素，也可以打印下标所对应的元素。
但因为元素不可变，所有列表所提供的增/删/改均不可用。
- **Python**内部对元组进行了大量的优化，访问和处理速度都比列表快。
 - 丰富的内置函数。。。
 - ~~■ 增: `append/insert/extend`~~
 - ~~■ 删: `del a[i:] / pop / remove / clear`~~
 - ~~■ 改: `copy / sort / reverse`~~
 - 查: `index / in / not in / count / len / max / min`
 - 遍历: `for ... in / for ... in enumerate(li)`

容器变量-元组(tuple)-语法糖:

■ 用元组去解释逗号赋值成立的语法解释:

```
>>> a, b = 1, 2
>>> a
1
>>> b
2
```

```
>>> a, b, c = 1, 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack (expected 3, got 2)
>>> a, b = 1, 2, 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

■ 1-打包:

```
>>> tp = 1, 2
>>> tp
(1, 2)
```

赋值语句

■ 2-解包:

```
>>> a, b = tp
>>> a
1
>>> b
2
```

取值语句

```
>>> a, b, c = tp
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack (expected 3, got 2)
>>> tp = (1, 2, 3)
>>> a, b = tp
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

错误示范（左边变量数与右边不相等）

容器变量-元组(tuple)

■ 一些特殊注意事项:

当要创建的元组中只包含一个元素时，必须带逗号。否则，会将左右括号默认视为运算符。

```
>>> tp=(1)
>>> tp
1
>>> tp=(1,)
>>> tp
(1,)
```

支持切片操作，切完片之后，依然是元组。

```
>>> tp=(1,2,3,4,5)
>>> tp[1::2]
(2, 4)
>>>
```

元组里面包含可变数据类型，可以间接修改元组的内容

```
>>> tp=(1,[2,3,4],5)
>>> tp
(1, [2, 3, 4], 5)
>>> tp[1] = [2,10,4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

```
>>> tp[1][1]=10
>>> tp
(1, [2, 10, 4], 5)
>>>
```

思考：是否违背了元组不可变的设计原则？

容器变量-字符串(str)

- 字符串，可以理解成一类特殊的元组，元素不可变。
- 创建过程(4种例子，注意 ‘、’”、’””’的合理使用)：

```
>>> info = "abcdefg"
>>> info
'abcdefg'
>>> info = 'abcdefg'
>>> info
'abcdefg'
```

```
>>> info = 'abc"de"fg'
>>> info
'abc"de"fg'
>>> info = "abc'de'fg"
>>> info
"abc'de'fg"
```

```
>>> a = 1
>>> b = 2
>>> info = f"{a} plus {b} is {a+b}"
>>> info
'1 plus 2 is 3'
```

```
>>> info="\"aaa
... bbb
... cc
... dd"
... \"\"\"
>>> info
'aaa\nbbb\ncc\ndd"\n\n'
>>> print(info)
aaa
bbb
cc
dd"
>>>
```

容器变量-字符串(str)

- 通用方法：参考tuple

- 一些特殊的使用语法：

- 判断：startswith / endswith / isalpha / isdigit / isspace

- 查找：find / rfind

- 修改：replace / lower / upper / capitalize / title

- 格式化：ljust / rjust / center / strip /rstrip / lstrip

- 拆分：split / splitlines / partition / rpartition

- 组织：join （可迭代对象） -> 固定间隔的字符串

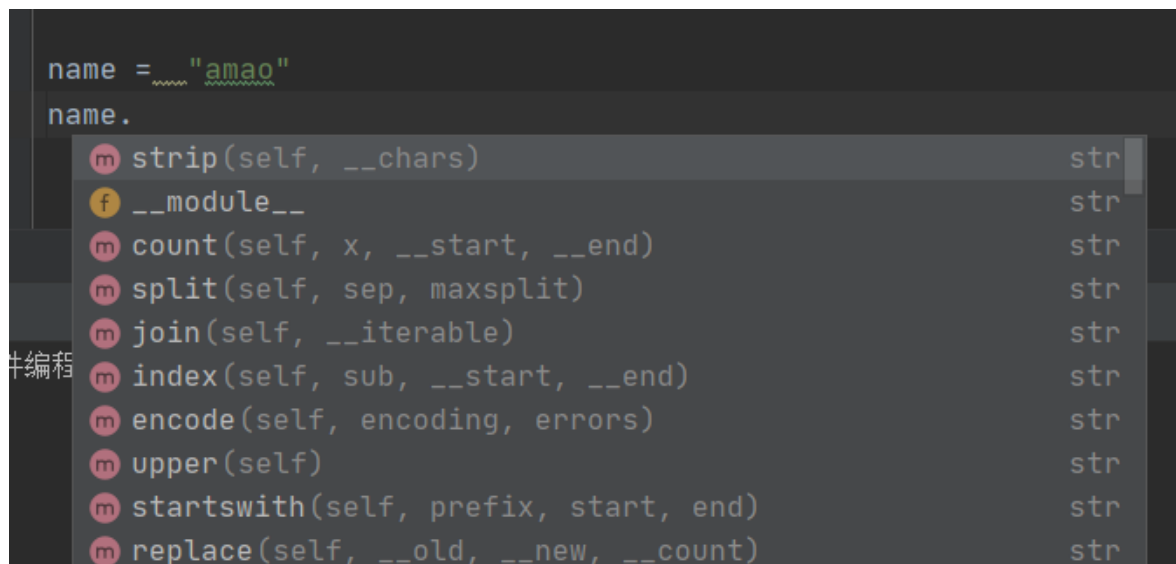
- 赋值：""" """

容器变量-字符串(str)

■ 使用技巧:

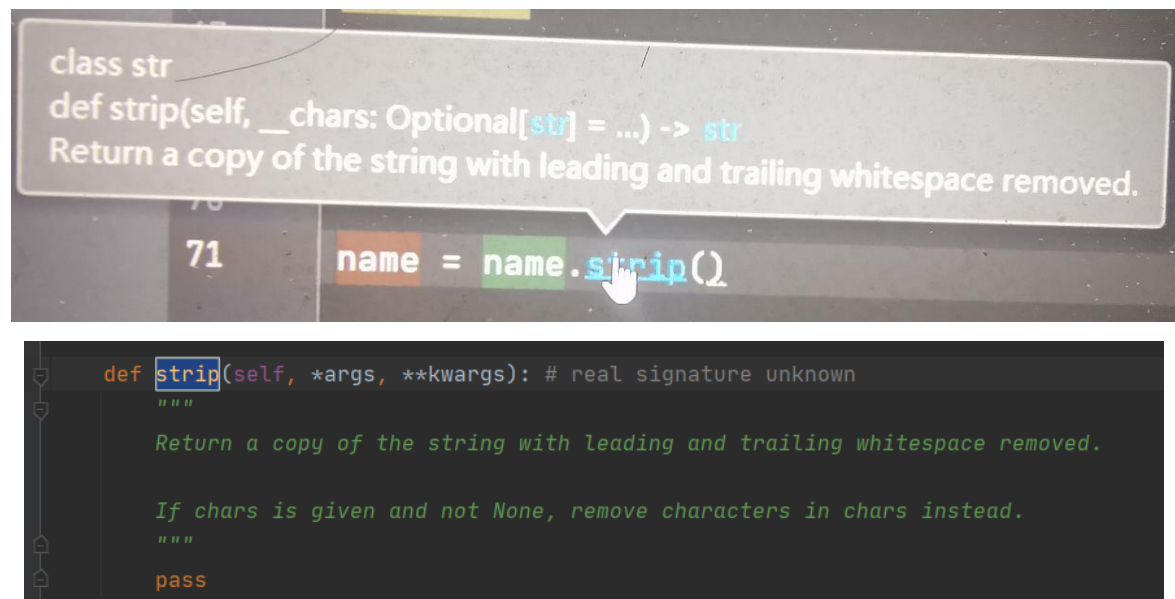
- ✓ 利用pycharm提供的自动补齐功能, 尽量不手敲方法名称, 避免手误。
- ✓ 利用pycharm提供的**builtin**方法查找功能, 提高代码开发效率。

操作1: 通过pycharm查询该变量具有哪些方法。



```
name = "amao"
name.
  strip(self, __chars) str
  __module__ str
  count(self, x, __start, __end) str
  split(self, sep, maxsplit) str
  join(self, __iterable) str
  index(self, sub, __start, __end) str
  encode(self, encoding, errors) str
  upper(self) str
  startswith(self, prefix, start, end) str
  replace(self, __old, __new, __count) str
```

操作2: 通过pycharm, 切入查看该变量的**builtin**实现。



```
class str
def strip(self, __chars: Optional[str] = ...) -> str
Return a copy of the string with leading and trailing whitespace removed.

71 name = name.strip()

def strip(self, *args, **kwargs): # real signature unknown
    """
    Return a copy of the string with leading and trailing whitespace removed.

    If chars is given and not None, remove characters in chars instead.
    """
    pass
```

容器变量-字符串(str)

■ 字符串转整形/浮点型

```
>>> int("10")
10
>>> int("10.1")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '10.1'
>>> int("abc")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'abc'
```

```
>>> float("10")
10.0
>>> float("10.0")
10.0
>>> float("10.1")
10.1
>>> float("abc")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'abc'
```

■ 整形/浮点型转字符串

```
>>> str(10)
'10'
>>> str(10.1)
'10.1'
```

■ 字符串的相加与乘法

```
>>> 'a'*10
'aaaaaaaaaa'
>>> 'aa'+'bb'
'aabb'
>>> 'ab' * 5
'ababababab'
```

容器变量-字符串(str)

■ 是否纯数字？

```
>>> "1234".isdigit()
True
>>> "abc".isdigit()
False
>>> "0.12".isdigit()
False
```

■ 是否均为空白字符：

```
>>> "\n \t".isspace()
True
>>> "\n a \t".isspace()
False
>>>
```

■ 是否均为纯字母？

```
>>> "abc".isalpha()
True
>>> "Abc".isalpha()
True
>>> "Abc12".isalpha()
False
>>> "Abc.12".isalpha()
False
```

■ 是否均为纯字母+数字？

```
>>> "\n a \t".isalnum()
False
>>> "abc123".isalnum()
True
>>> "abc.123".isalnum()
False
>>>
```

容器变量-字符串(str)

■ 字符串判断是否存在某字符或字符串

```
>>> info = 'a b aa bb cc dd ee ff'
>>> "bb" in info
True
>>> "tt" in info
False
```

```
>>> info = 'a b aa bb cc dd ee ff'
>>> info.find("bb")
7
>>> info.find("tt")
-1
```

■ 字符串替换

```
>>> ninfo = info.replace("ee", "tt")
>>> info
'a b aa bb cc dd ee ff'
>>> ninfo
'a b aa bb cc dd tt ff'
```

■ 字符串大小写切换

```
>>> info
'aa bb cc dd'
>>> info.upper()
'AA BB CC DD'
>>> info.lower()
'aa bb cc dd'
```

注意： **info**本身没有变化，会生成新的字符串

容器变量-字符串(str)

■ 字符串去除空格（生成新字符串）

```
>>> info = ' a b aa bb cc dd ee ff '
>>> info
' a b aa bb cc dd ee ff '
>>> ninfo = info.strip()
>>> ninfo
'a b aa bb cc dd ee ff'
>>> info
' a b aa bb cc dd ee ff '
```

■ 字符串对齐（生成新字符串）

```
>>> info
'aa bb cc dd'
>>> info.center(20)
'      aa bb cc dd      '
>>> info.ljust(20)
'aa bb cc dd           '
>>> info.rjust(20)
'           aa bb cc dd'
```

■ 字符串切分，str 转 li

```
>>> info
'a b aa bb cc dd ee ff'
>>> li = info.split(" ", 1)
>>> li
['a', 'b aa bb cc dd ee ff']
>>> li = info.split(" ")
>>> li
['a', 'b', 'aa', 'bb', 'cc', 'dd', 'ee', 'ff']
```

■ 字符串合并，li 转 str

```
>>> li = ['aa', 'bb', 'cc', 'dd']
>>> li
['aa', 'bb', 'cc', 'dd']
>>> info = ".".join(li)
>>> info
'aa.bb.cc.dd'
```


容器变量-字符串(str)-应用示例: IP地址转换

■ IP地址就是用来编码在计算机网络上的一个地址，我们在电脑上网在使用手机上网，都是因为我们有IP地址，如果没有IP地址那就无法上网，就像没有家庭住址无法收取快递一样。

```
C:\Users\wangbin>ipconfig
```

```
Windows IP 配置
```

```
以太网适配器 以太网:
```

```
媒体状态 . . . . . : 媒体已断开连接  
连接特定的 DNS 后缀 . . . . . :
```

```
以太网适配器 以太网 3:
```

```
媒体状态 . . . . . : 媒体已断开连接  
连接特定的 DNS 后缀 . . . . . :
```

```
以太网适配器 以太网 2:
```

```
连接特定的 DNS 后缀 . . . . . :  
IPv4 地址 . . . . . : 172.13.41.34  
子网掩码 . . . . . : 255.255.255.255  
默认网关 . . . . . : 172.0.0.1
```

■ 在win下的cmd窗口中，执行 ipconfig 命令查询自己的IP地址

■ 该网卡的IPv4地址, 172.13.41.34

网际协议版本4（Internet Protocol version 4，IPv4），又称互联网通信协议第四版，使用32位（4字节）地址。每个字节取值范围：0-255。

为了提高可读性，转化成字符串去显示。

容器变量-字符串(str)-应用示例： IP地址转换

■ 问题1：从 IP地址字符串 “172.13.41.34” 解析出 4个整形IP地址构成的列表

[172, 13, 41, 34]

■ 第一步，根据IP地址的组成规律（用.号合并4个数字），调用split命令分割字符串：

```
>>> ip = "172.13.41.34"
>>> ip_list = ip.split(".")
>>> ip_list
['172', '13', '41', '34']
>>>
```

→ ■ 得到列表，但每个元素都是字符串

■ 第二步，遍历列表，调用int()，将字符串元素转成整形元素

```
>>> for idx, ele in enumerate(ip_list):
...     ip_list[idx] = int(ele)
...
>>> ip_list
[172, 13, 41, 34]
>>>
```

→ ■ 得到4个整形数构成的列表，done

容器变量-字符串(str)-应用示例: IP地址转换

■ 问题2: 根据出4个整形数构成列表[172, 13, 41, 34], 组织字符串格式的IP地址:

■ 方法1, f语法

```
>>> ip_list
[172, 13, 41, 34]
>>> ip = f"{ip_list[0]}.{ip_list[1]}.{ip_list[2]}.{ip_list[3]}"
>>> ip
'172.13.41.34'
```

■ 方法2, join语法

```
>>> ip_list
[172, 13, 41, 34]
>>> ".".join(ip_list)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sequence item 0: expected str instance, int found
>>>
```

报错, 只支持字符串的join

容器变量-字符串(str)-应用示例： IP地址转换

■ 问题2： 根据出4个整形数构成列表[172.13, 41, 34]，组织字符串格式的IP地址：

■ 方法2，join语法

```
>>> for idx, ele in enumerate(ip_list):  
...     ip_list[idx] = str(ele)  
...  
>>> ip_list  
['172', '13', '41', '34']  
>>> ip = ".".join(ip_list)  
>>> ip  
'172.13.41.34'  
>>>
```

■ 方法3，列表生成式（推荐使用，下节课会介绍到）

```
>>> ".".join([str(ele) for ele in ip_list])  
'172.13.41.34'  
>>>
```

课堂练习（1）

列表操作：

有如下列表，li=['wangbin', 'jessie', 'eric']，按照要求实现每一个功能：

- 计算列表长度并输出 `len(li)`
- 列表中追加元素“seven”，并输出添加后的列表 `li.append("seven")`
- 请在列表的第1个位置插入元素“Tony”，并输出添加后的列表 `li.insert(1, "Tony")`
- 请修改列表第2个位置的元素为“Kelly”，并输出修改后的列表 `li[2] = 'Kelly'`
- 请删除列表中的元素“eric”，并输出修改后的列表 `li.remove("eric")`
- 请删除列表中的第3个元素，并输出删除元素后的列表 `ele = li.pop(3)`
- 请删除列表中的第2至4个元素，并输出删除元素后的列表 `del li[2:5:]`
- 请将列表所有的元素反转，并输出反转后的列表 `li.reverse()`
- 请使用for、len、range输出列表的索引 `for idx in range(len(li))`
- 请使用enumerate输出列表元素和序号（序号从100开始） `for idx, val in enumerate(li) print(idx+100, val)`
- 请使用for循环输出列表的所有元素 `for val in li:`

课堂练习（1）

元组操作：

写代码，有如下元组，请按照功能要求实现每一个功能

```
tu=('alex','eric','rain')
```

- 计算元组长度并输出： `len()`
- 获取元组的第2个元素，并输出, `tu[2]`
- 获取元组的第1-2个元素，并输出, `tu[1:3]`
- 请使用for输出元组的元素, `for ele in tu: ...`
- 请使用for、len、range输出元组的索引 `for idx in range(len(tu)):`
- 请使用enumerate输出元组元素和序号（序号从10开始） `for idx, val in enumerate(li) print(idx+10, val)`

课堂练习（1）

字符串操作：


1. 用代码将字符串“Tsinghua University”按空格切割， `str.split`
2. 将列表['清','芬','园']中的每一个元素使用‘_’连接为一个字符串，“_”.`join(list)`
3. 利用下划线将列表的每一个元素拼接成字符串 `li=['p1', 'p2', 'p3']`
4. 获取字符串子序列，仅不包含最后一个字符，如：`woaini`则获取`woain` `root`则获取`roo`, `str[:-1]`
5. 将字符串变量对应的值变成小写，并输出结果, `str.lower()`
6. 将字符串变量对应的值中的"o", 替换为"p", 并输出结果, `str.replace('o','p')`
7. 判断字符串变量对应的值是否以"go"开头，并输出结果, `str.startsWith('go')`
8. 移除字符串变量对应值的两边的空格，并输出移除后的内容, `str.strip()`
9. 请输出`name`变量中的值“Q” 的索引的位置, `str.index('Q')`
10. 输入一行字符，并判断这句话是否回文， 比如 `abba`、上海自来水来自海上

上机练习讲解:

输入一行字符，并判断这句话是否回文， 比如 abba、上海自来水来自海上

方法1: 类似于c的写法

```
while True:
    text = input("请输入语句:>>").strip()
    if len(text) == 0:
        continue
    for idx, chr in enumerate(text):
        if chr != text[len(text) - idx - 1]:
            print(f"{text} 不是回文体")
            break
    else:
        print(f"{text} 是回文体!")
```

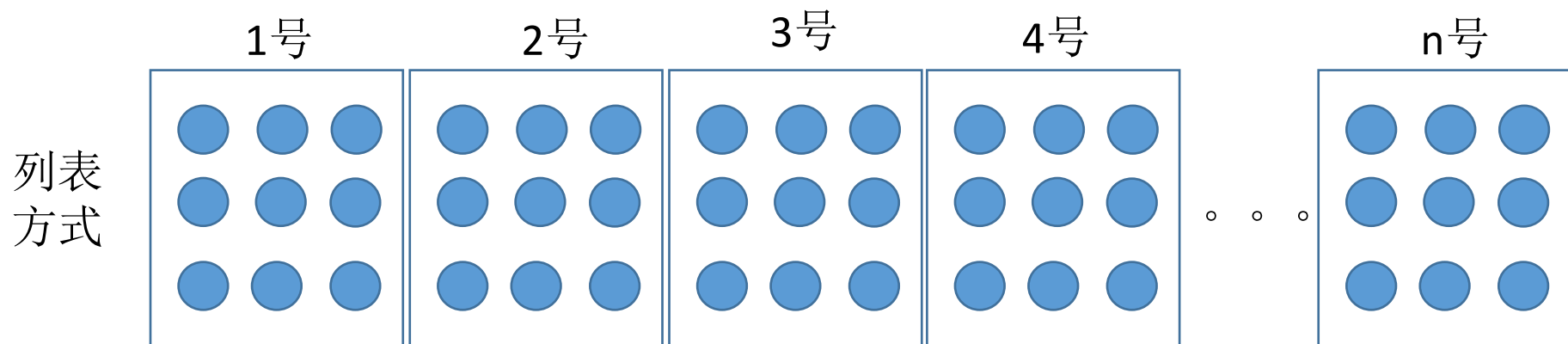


方法2: 直接调用 python 方法 (推荐)

```
while True:
    text = input("请输入语句:>>").strip()
    if len(text) == 0:
        continue
    if text != text[::-1]:
        print(f"{text} 不是回文")
    else:
        print(f"{text} 是回文")
```


容器变量-字典(dict) -引入目的

■ 如何存放人员信息？



list的问题：

■ 检索： $O(n)$ 复杂度

■ 增删：需要做比较多的内存操作。

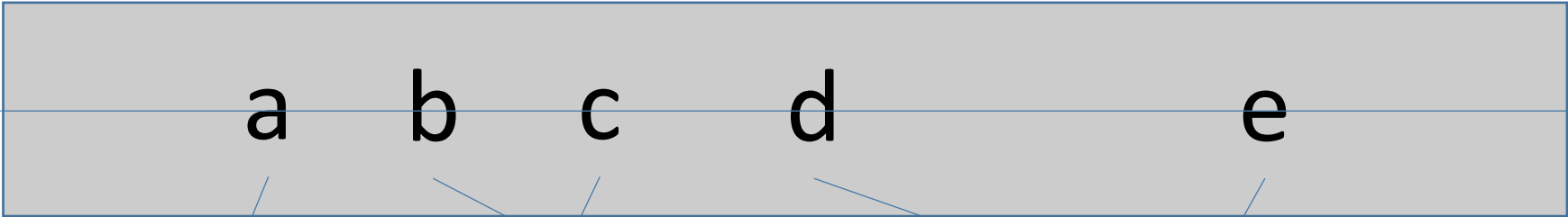
■ 在某位置添加元素时，该位置后面的元素需要向后顺移。

■ 从某位置删除元素时，该位置后面的元素需要向前顺移。

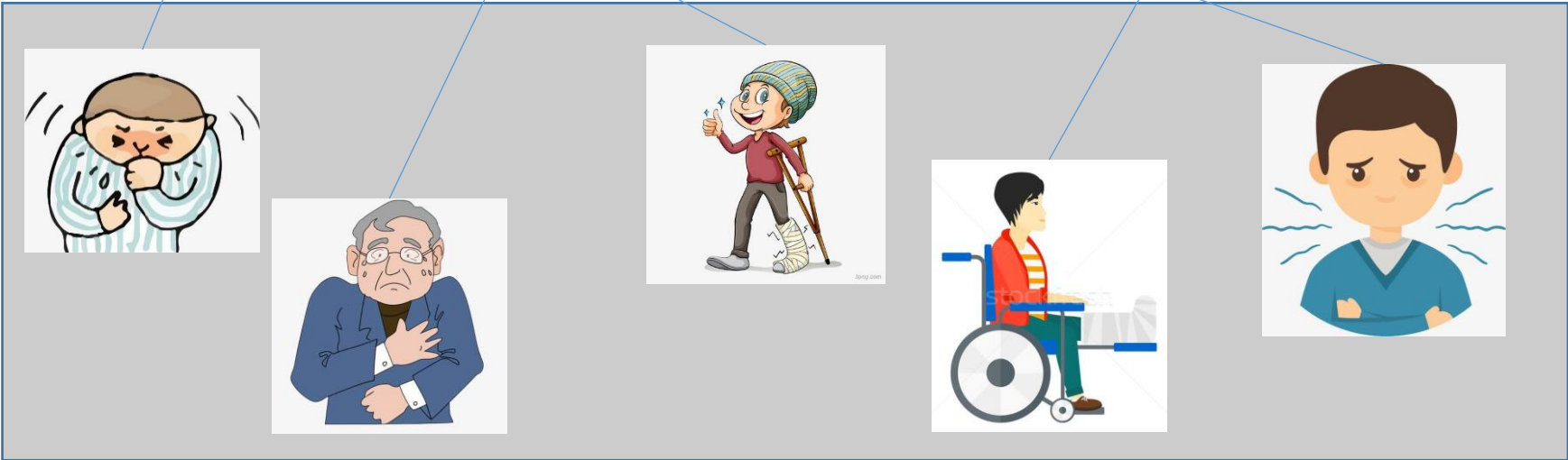
医院就医：
list: 在医生门口人肉排队

容器变量-字典(dict) -引入目的

■ 一个新的做法:



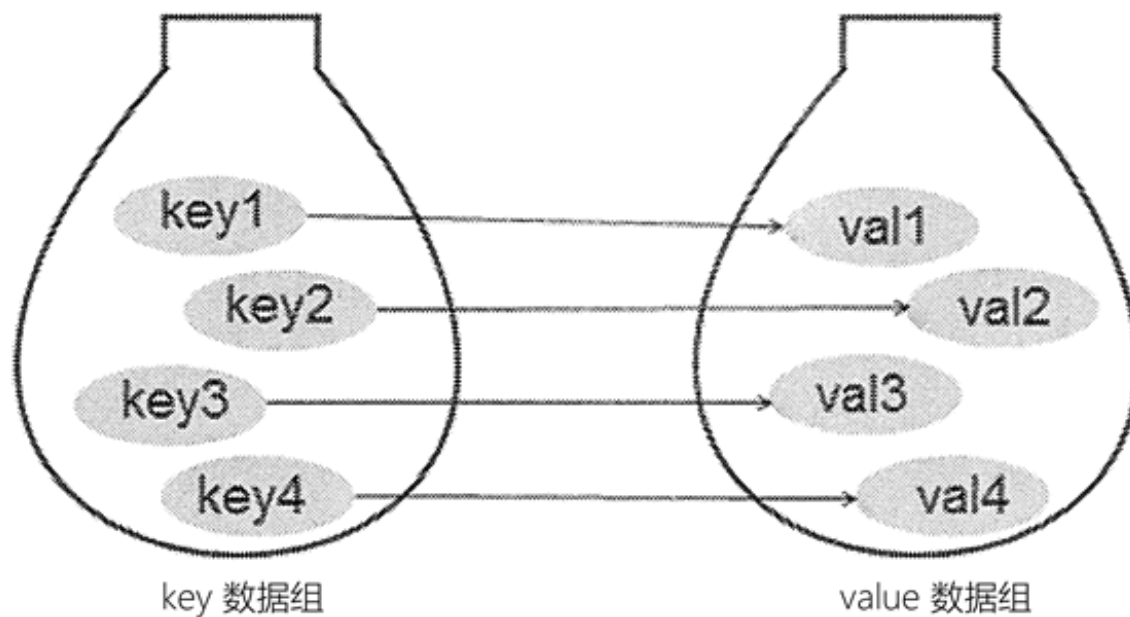
key



value

容器变量-字典(dict) -概念

- 为了保存具有映射关系的数据，Python 提供了一种新的容器变量：字典，字典相当于保存了两组数据，其中一组数据是关键数据，被称为 **key**；另一组数据可通过 **key** 来访问，被称为 **value**。(key: 医生手里的挂号单， value: 病人)



容器变量-字典(dict) -创建方式

■ 方案1：用{}括起来，以冒号(:)分割键-值对，各键值对用

```
>>> dic = {"name": "amao", "age": 28, "sex": "male", "height": 180.0}
>>> dic
{'name': 'amao', 'age': 28, 'sex': 'male', 'height': 180.0}
```

变量：物理对象数字化建模

物理对象	参数	变量	类型
人	姓名	name	字符串
	年龄	age	整型数
	身高	height	浮点数
	收入	salary	浮点数
	性别	sex	整型数 布尔型

■ 方案2：dict(list(tuple)) 方式：

```
>>> tp = [("name", "amao"), ("age", 28), ("sex", "male"), ("height", 180.0)]
>>> dic = dict(tp)
>>> dic
{'name': 'amao', 'age': 28, 'sex': 'male', 'height': 180.0}
```

■ 方案3：dict(key1=value1, key2=valu2, ...) 方式：

```
>>> dic = dict(name="amao", age=28, sex="male", height=180.0)
>>> dic
{'name': 'amao', 'age': 28, 'sex': 'male', 'height': 180.0}
```

容器变量-字典(dict) -创建规律

- 字典中的元素（键值对）没有特定顺序，不能用数字下标去访问。
- 字典中的value可以任何数据类型，包括不限于:字符串(str)、数字(int, float)、元组(tuple)、字典(dict)、列表(list)等。
- 字典中的key是唯一的，必须是不可变数据类型， 比如： 字符串(str)、数字(int, float)、元组(tuple)。

```
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 180.0}
>>> dic[(1,2,3,4)] = [3,4,5,6,7]
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 180.0, (1, 2, 3, 4): [3, 4, 5, 6, 7]}
>>> dic[[1,2,3,4]] = [1,2,3,4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

容器变量-字典(dict) -增删改查

- 插入一个元素: `dic[key] = value`

```
>>> dic
{'name': 'amao', 'age': 28, 'sex': 'male', 'height': 180.0}
>>> dic['weight'] = 80.1
>>> dic
{'name': 'amao', 'age': 28, 'sex': 'male', 'height': 180.0, 'weight': 80.1}
```

- 插入一个新的字典: `dic.update(ndic)`

```
>>> dic
{'name': 'amao', 'age': 28, 'sex': 'male', 'height': 180.0, 'weight': 80.1}
>>> ndic={"weight": 70.0, "school": "tsinghua"}
>>> dic.update(ndic)
>>> dic
{'name': 'amao', 'age': 28, 'sex': 'male', 'height': 180.0, 'weight': 70.0, 'school': 'tsinghua'}
```

容器变量-字典(dict) -增删改查

- 清空字典: dic.clear()

```
>>> dic
{'name': 'amao', 'age': 28, 'sex': 'male', 'height': 180.0}
>>> dic.clear()
>>> dic
{}
```

- 删除某一个键值对, del dic[key]

```
>>> dic = {"name": "amao", "age": 28, "sex": "male", "height": 180.0}
>>> del dic['name']
>>> dic
{'age': 28, 'sex': 'male', 'height': 180.0}
```

- 删除某一个键值对, 并返回key对应的value值, value = dic.pop(key)

```
>>> dic = {"name": "amao", "age": 28, "sex": "male", "height": 180.0}
>>> age = dic.pop("age")
>>> age, dic
(28, {'name': 'amao', 'sex': 'male', 'height': 180.0})
```

容器变量-字典(dict) -增删改查

- 可以通过dic[key] = value操作，修改字典中的某一个键值对，如下所示：

```
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 180.0, 'age': 30}
>>> dic['height'] = 185.5
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 185.5, 'age': 30}
```

- 如果 key 不存在，dic[key] = value，会在dic中新添加键值对，如下所示：

```
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 180.0}
>>> dic['age'] = 30
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 180.0, 'age': 30}
```


容器变量-字典(dict) -增删改查

- 通过 key 得到 value: value = dic[key]

```
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 180.0}
>>> dic['name']
'amao'
>>> dic['age']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
```

如果key不存在，则会报错，所以是不安全的。

- 通过get方法得到 value: value = dic.get(key, default_value)

```
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 180.0}
>>> dic.get("name", "def_name")
'amao'
>>> dic.get("age", 20)
20
```

如果key存在，则返回key对应的value值

如果key不存在，则返回预设的默认值 20

容器变量-字典(dict) -增删改查

■ 对字典进行遍历的方案1:

for key in dic, 遍历key

```
>>> for key in dic:
...     print(key, dic[key])
...
name amao
sex male
height 180.0
```

■ 对字典进行遍历的方案2:

for key, value in dic.items(), 遍历键值对

```
>>> for key, value in dic.items():
...     print(key, value)
...
name amao
sex male
height 180.0
```

dic.items(), 返回键值对组成的元组列表

```
>>> dic.items()
dict_items([('name', 'amao'), ('sex', 'male'), ('height', 180.0)])
```

容器变量-字典(dict) -增删改查

- 检测某一个key 是否在字典中 in / not in:

```
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 185.5, 'age': 30}
>>> 'name' in dic
True
>>> 'age' in dic
True
>>> age not in dic
True
```

- 字典复制 copy:

```
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 185.5, 'age': 30}
>>> ndic = dic.copy()
>>> ndic
{'name': 'amao', 'sex': 'male', 'height': 185.5, 'age': 30}
>>> ndic['name'] = 'agou'
>>> ndic
{'name': 'agou', 'sex': 'male', 'height': 185.5, 'age': 30}
>>> dic
{'name': 'amao', 'sex': 'male', 'height': 185.5, 'age': 30}
```

注意: `ndic = li`

与 `ndic = dic.copy()`

的区别。

1个是浅copy, 未开辟新内存;

1个是深copy, 开辟了新内存。

容器变量-字典(dict) -归纳

■ 主要特性:

- 用{}括起来，以冒号(:)分割键-值对，各键值对用逗号(,)分隔开
- 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
- 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：
- 键必须不可变，所以可以用数，字符串或元组充当，所以用列表、字典等做key就不行：

■ 特殊用法:

- 增加/修改: []
- 删除: `del dict[key]` / `dict.pop(key)` / `dict.clear()`
- 查找: [] / `dict.get(key, default)` / `in` / `not in`
- 遍历: `for key in dict:` / `for key, values in dict.items():`
- 复制: `dict.copy()`
- 合并: `dict.update(d2)`
- 长度: `len(dic)`

前提条件: `dic = {'hobby':['swim', 'run', 'ball']}`, `ndic = dic`

问 题: 若 `dic['age'] = 20`, 则 `ndic['age'] = ?`

- ☒ A 20
- ☐ B 报错
- ☐ C 不知道。

提交

前提条件: `dic = {'hobby':['swim', 'run', 'ball']}`, `ndic = dic.copy()`

问 题: 若 `dic['age'] = 20`, 则 `ndic['age'] = ?`

- ☐ A 20
- ☒ B 报错
- ☐ C 不知道。

提交

前提条件: `dic = {'hobby':['swim', 'run', 'ball']}`, `ndic = dic`

问 题: 若 `dic['hobby'].append('sleep')`, 则 `ndic['hobby'] = ?`

- ☐ A `['swim', 'run', 'ball']`
- ☒ B `['swim', 'run', 'ball', 'sleep']`
- ☐ C 不知道。

提交

前提条件: `dic = {'hobby':['swim', 'run', 'ball']}`, `ndic = dic.copy()`

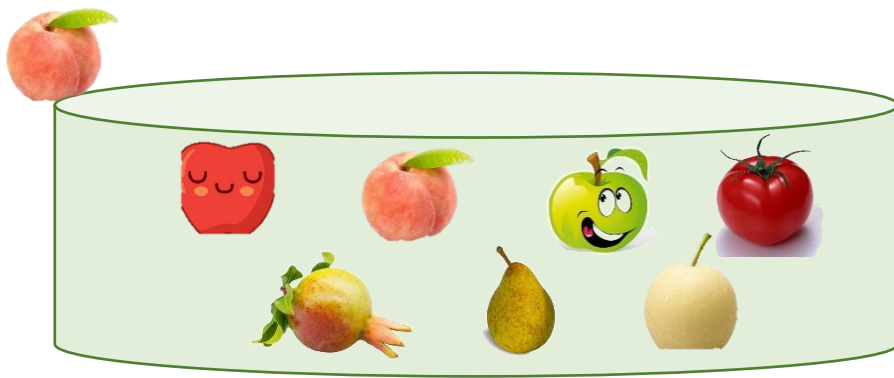
问 题: 若 `dic['hobby'].append('sleep')`, 则 `ndic['hobby'] = ?`

- ☐ A ['swim', 'run', 'ball']
- ☒ B ['swim', 'run', 'ball', 'sleep']
- ☐ C 不知道。

提交

容器变量-集合(set)

■ Python提供的内置数据结构，元素之间无序，元素之间有排他性。



■ 创建过程: {ele1, ele2, ele3....}

```
>>> info = {1, 2, 3, 2, 'a', 'ab', 'a'}
>>> info
{1, 2, 3, 'ab', 'a'}
>>> info[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
>>>
```

■ 无序性，无法下标操作，也无法对集合进行切片操作。

■ 支持元素多类型，天然去重。

■ 元素类型只能是不可变类型（整型、实型、元组、字符串）。

■ 相当于一类特殊的字典，没有value，只有key。

容器变量-集合(set)

■ 作用1: 可完成高效的元素定位:

列表,需要遍历, $O(n)$ 复杂度

```
>>> li = [1, 2, 3, 4, 'a', 'ab', 'a']
>>> 2 in li
True
```



集合,可直接根据hash值定位, $O(1)$ 复杂度

```
>>> st = set(li)
>>> st
{1, 2, 3, 4, 'ab', 'a'}
>>> 2 in st
True
```

■ 作用2: 可完成简单、高效的元素去重:

列表去重, $O(n^2)$ 复杂度

```
list1=[1,2,3,4]
list2=[3,4,5,6]
list3=[]
for i in list1:
    if i in list2:
        list3.append(i)
print(list3)
```



集合去重, $O(n)$ 复杂度

```
my_list = [1, 2, 3, 4, 2, 3, 1, 2, 2]
# 转化为集合
my_set = set(my_list)
print(list(my_set))
```

容器变量-集合(set)-增删改查

■ 不支持对元素进行修改。

增 (add)

```
>>> info
{1, 2, 'ab', 'a'}
>>> info.add('abc')
>>> info
{1, 2, 'ab', 'a', 'abc'}
>>> info.add('abc')
>>> info
{1, 2, 'ab', 'a', 'abc'}
```

查找 (in)

```
>>> info
{1, 2, 'ab', 'a', 'abc'}
>>> len(info)
5
>>> 2 in info
True
>>> 20 in info
False
```

删 (remove)

```
>>> info = {1, 2, 3, 'a', 'ab'}
>>> info.remove(3)
>>> info
{1, 2, 'ab', 'a'}
>>> info.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 3
```

遍历 (for ...in ...)

```
>>> info
{1, 2, 'ab', 'a', 'abc'}
>>> for ele in info:
...     print(ele)
...
1
2
ab
a
abc
```

容器变量-集合(set)

■ 支持交集、并集、补集等关系运算操作，如下所示：

集合运算	代码示意	结果
& 交集	s1={1,2,3} s2={2,3,4} s3 = s1 & s2	s3={2,3} 
并集	s1={1,2,3} s2={2,3,4} s3 = s1 s2	s3={1, 2, 3, 4} 
- 补集	s1={1,2,3} s2={2,3,4} s3 = s1 - s2	s3={1} 
^ 对称补集	s1={1,2,3} s2={2,3,4} s3 = s1 ^ s2	s3={1, 4} 
< 子集	s1={1,2,3} s2={2,3} s2 < s1	#True 判断子集
> 超集	s1={1,2,3} s2={2,3} s1 > s2	#True 判断超集
== 相同, != 不同	s1={1,2,3} s2={2,3, 1}	s2 == s1 # True, s1 != s2 # False

课堂练习（2）

1. `dic = {'k1': "v1", "k2": "v2", "k3": [11,22,33]}`

- 请在字典中添加一个键值对, "k4": "v4", 输出添加后的字典 `dic['k4'] = 'v4'`
- 请在修改字典中“k1”对应的值为“alex”, 输出修改后的字典 `dic['k1'] = 'alex'`
- 请在k3对应的值中追加一个元素44, 输出修改后的字典 `dic['k3'].append(44)`
- 请在k3对应的值的第1个位置插入个元素18, 输出修改后的字典 `dic['k3'].insert(1,18)`

2. 元素分类

有如下值列表[11,22,33,44,55,66,77,88,99,90], 将所有大于66的值保存至字典的第一个key中, 将小于66的值保存至第二个key的值中。

即: {'k1':大于66的所有值列表, 'k2':小于66的所有值列表}

3. 字典应用:

输入一行英文句子, 统计各单词出现的次数:

课堂练习（2）

有两个集合：

- $s1 = \{11, 22, 33\}$
- $s2 = \{22, 33, 44\}$

功能要求：

- 获取内容相同的元素列表
- 获取s1中有，s2中没有的元素列表
- 获取s2中有，s1中没有的元素列表
- 获取s1和s2中内容都不同的元素

课堂练习讲解： 输入一行英文句子，统计各单词出现的次数：

python代码

```
text = input("请输入句子:>>").strip()

words = text.split(" ")
print("分割后形成的列表:", words, sep='\n')

dic = {}
for word in words:
    if word not in dic:
        dic[word] = 1
    else:
        dic[word] += 1

print("统计频次结果:", dic, sep='\n')
```

运算结果

请输入句子:>>aa bb cc aa
分割后形成的列表:
['aa', 'bb', 'cc', 'aa']
统计频次结果:
{'aa': 2, 'bb': 1, 'cc': 1}

容器变量 – 归纳（1）

	特点	定位查找的算法复杂度
列表(list) [1,2,3]	<ul style="list-style-type: none">■ 可存放多个值■ 按照从左到右的顺序定义列表元素，下标从0开始顺序访问，有序■ 可修改指定索引位置对应的值，可变	O(1) O(n)
元组(tuple) (1,2,3)	<div><div><ul style="list-style-type: none">■ 类似于列表■ 元素不可变</div><div>思考题:</div><div><pre>a=([1,2],1,2) # a[0]=[3,2] # a[0][1]=3 print(a)</pre></div><div><pre>b=[1,2] a=(b,1,2) b[0]=3 print(a)</pre></div></div>	
字典(dict) {'a':1, 'b':2}	<ul style="list-style-type: none">■ key-value结构■ key必须为不可变数据类型、必须唯一■ 可存放任意多个value、可修改、可以不唯一■ 无序■ 查询速度快，且不受dict的大小影响。	不支持下标定位 hash后O(1)
集合(set) {'a', 'b'}	<ul style="list-style-type: none">■ 里面的元素不可变■ 天生去重，在集合里没办法存重复的元素■ 无序，不像列表一样通过索引来标记在列表中的位置，元素是无序的，集合中的元素没有先后之分，如集合{3,4,5}和{3,5,4}算作同一个集合	不支持下标定位 hash后O(1)

容器变量 – 归纳（2）

	增	删	改	查	遍历	切片
列表(list) [1,2,3]	append:追加 insert:插入 extend:合并	del list[i] pop clear	list[i] = 'ttt'	in not in index("eva") count("eva")	for val in li: print(val)	a[1:] a[:2] a[1:5:2]
元组(tuple) (1,2,3)	不可增	不可删	元组本身不可变，如果元组中还包含其他可变元素，这些可变元素可以改变		for idx, val in enumerate(li): print(idx, val)	顾头 不顾尾
字典(dict) {'a':1, 'b':2}	dict['a']=2	del dict[i] clear	dict['a']=2	in not in ::get	for k in d: print(k, d[k]) for k,v in d.items(): print(k,v)	不可 切片
集合(set) {'a', 'b'}	add & - ^	remove clear	本身不可变，如果元组中还包含其他可变元素，这些可变元素可以改变	in isdisjoint issubset issuperset	同上	不可 切片

课程内容：

- 知识点回顾
- 容器变量
- 列表生成式

Python语法糖-列表生成式-基本概念

- python内置非常简单却强大的可以用来创建list的生成式，综合运用表达式+循环+条件判断，写出非常简洁的代码。
- 列表生成式的基本结构如下：

[express(ele) for ele in iterator if eval(ele)]

- 在大多数情况下，与for和if循环相比，优先采用列表生成式代码，因为：
 - ✓ 它们比for循环快得多
 - ✓ 它们被认为比循环和映射函数更具python特性
 - ✓ 列表生成式的语法更容易阅读

列表生成式-引出

案例1： 如何生成1-10之间所有偶数形成的list

普通做法

```
li = []
for ele in range(10):
    if ele >= 1 and ele % 2 == 0:
        li.append(ele)
```

Python做法

```
li = [ele for ele in range(10) if ele >= 1 and ele % 2 == 0]
```

[exp(ele) for ele in iterator if eval(ele)]

```
li = []
for ele in range(10):
    if ele >= 1 and ele % 2 == 0:
        li.append(ele)
```

```
li = [ele for ele in range(10) if ele >= 1 and ele % 2 == 0]
```

列表生成式-引出

案例2：如何生成1-10之间所有偶数形成的list，再取平方值

普通做法

```
li = []  
for ele in range(10):  
    if ele >= 1 and ele % 2 == 0:  
        li.append(ele*ele)
```

Python做法

```
li = [ ele * ele for ele in range(10) if ele >= 1 and ele % 2 == 0 ]
```

[exp(ele) for ele in iterator if eval(ele)]

列表生成式-引出

案例3：多重for 循环的列表生成式， 输出9-9乘法表

```
for i in range(1, 10):  
    for j in range(i, 10):  
        print(f"{i}*{j}={i * j}", end=' ' if j < 9 else '\n')
```

```
[print(f"{i}*{j}={i * j}", end=' ' if j < 9 else '\n') for i in range(1, 10) for j in range(i, 10)]
```

生成的列表本身无意义，关键是打印了print语句。

列表生成式-引出

案例4：求解100以内的质数

```
prim_list = []
for num in range(2, 100):
    for num1 in range(2, num//2+1):
        if num % num1 == 0:
            break
    else:
        prim_list.append(num)
```

1. 流程控制转换成列表操作

```
prim_list = []
for num in range(2, 100):
    nli = []
    for num1 in range(2, num//2+1):
        if num % num1 == 0:
            nli.append(num1)
    if len(nli) == 0:
        prim_list.append(num)
```

2. 内层列表生成式

```
prim_list = []
for num in range(2, 100):
    if len([num1 for num1 in range(2, num//2+1) if num % num1 == 0]) == 0:
        prim_list.append(num)
```

最终：

1行搞定

3. 外层列表生成式

```
prim_list = \
    [num for num in range(2, 100) if len([num1 for num1 in range(2, num//2+1) if num % num1 == 0]) == 0]
```

列表生成式-引出

案例5：用列表生成式生成集合(set)对象。

找出list带有a的字符串，并生成集合

```
wd_lst = ['aa', 'bb', 'bc', 'cd', 'da']  
wd_set = set()  
for wd in wd_lst:  
    if wd.count('a'):  
        wd_set.add(wd)  
  
print(wd_set)
```

{ele}语法

```
wd_lst = ['aa', 'bb', 'bc', 'cd', 'da']  
wd_set = {wd for wd in wd_lst if wd.count('a')}  
print(wd_set)
```


列表生成式-引出

案例6：用列表生成器生成字典(dict)对象。

过滤年龄小于20岁的所有用户

```
users = {  
    "张三": {"password": 'zhangsan', "age": 11},  
    "李四": {"password": "lisi", "age": 22},  
    "王五": {"password": "wangwu", "age": 33}  
}
```

```
find_users = {}  
for key in users:  
    if users[key]['age'] < 20:  
        find_users[key] = users[key]  
print(find_users)
```

{key:value}语法

思考问题：能否用列表生成器得到元组对象？

tp = (ele for ele in [1, 2, 3, 4] if ele % 2 == 0)

自己动手，看看是否成功。

```
users = {  
    "张三": {"password": 'zhangsan', "age": 11},  
    "李四": {"password": "lisi", "age": 22},  
    "王五": {"password": "wangwu", "age": 33}  
}  
  
find_users = {key: users[key] for key in users if users[key]['age'] < 20}  
print(find_users)
```

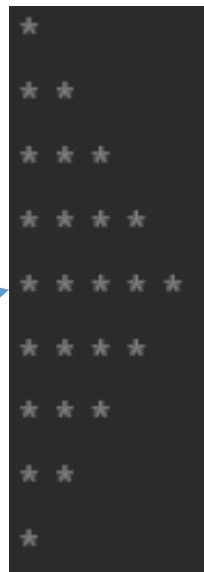
列表生成式-使用提示:

- 列表生成式是一种高效的操作。对于简单的任务，生成式语法很简洁，会提高代码开发和阅读效率（通俗来说，**列表生成式会减少代码数量**）。
- 在复杂的情况下，如果你很难创建或生成式复杂的列表生成式，请尝试使用循环编写（通俗来说，**如果列表生成式的语法过于复杂，而影响了代码可读性，则不应该使用列表生成式，知其技而不用**）。
- 由于列表对象是一次性将所有对象在内存中生成，因此：
 - 对于中小型列表，采用列表生成式或列表是可取的，因为一次性生成，后续访问操作会更快。
 - 对于大型列表（例如元素超过10亿）时，由于内存需求量过大。可能会导致计算机崩溃，此时，应该避免直接使用列表。
 - 建议采用下节课会讲的生成器（generator）。

课堂练习（3）

用两种不同的方式（普通python代码、列表生成式）分别实现如下代码：

- 求100以内不能被3整除的所有数列表，并求出这些数字的总和和平均数。
- 有四个数字：1、2、3、4，能组成多少个互不相同且无重复数字的三位数？各是多少？
- 把列表a=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]，里的每个值加1
- 形成一个列表,列表元素分别为[1 ** 1,2 ** 2,...,9 ** 9]
- 列表中字符串的大写改成小写，不是字符串的元素去掉
- 输出 *号（[*,**,***,****,*****,*****,****,**,*,*]）
- 给定一个用户列表，判断根据用户名称判断该用户是否存在。



```
users = [  
    {"name": "张三", "password": 'zhangsan', 'age': 11},  
    {"name": "李四", "password": 'lisi', 'age': 22},  
    {"name": "王五", "password": 'wangwu', 'age': 33},  
]
```

代码讲解：4. 输出*号



直接在列表生成式中打印一行搞定

```
>>> li = [ i * "*" for i in [1,2,3,4,5,4,3,2,1]]
>>> print("\n".join(li))

*
**
***
****
*****
****
***
**
*
```

```
>>> [ print(i * " ", end= '\n') for i in list(range(1,6)) + list(range(5, 0, -1))]

*
* *
* * *
* * * *
* * * * *
* * * * *
* * * *
* * *
* *
*
```

等价操作

代码讲解：5. 直接判断某用户是否存在

```
users = [  
    {"name": "张三", "password": 'zhangsan', 'age': 11},  
    {"name": "李四", "password": 'lisi', 'age': 22},  
    {"name": "王五", "password": 'wangwu', 'age': 33},  
]
```

普通做法

```
name = 'zhangsan'  
user = None  
for u1 in users:  
    if name == u1['name']:  
        user = u1  
        break  
if user is None:  
    print("您不是已有用户, 请注册!")  
    exit()
```

5行压缩到
2行

Python做法

```
name = 'zhangsan'  
find_users = [u1 for u1 in users if u1['name'] == name]  
user = find_users[0] if len(find_users) > 0 else None  
if user is None:  
    print("找不到用户", name)  
    exit()
```