

## 第一部分 基础知识

第1章总结到此结束!!! 🐱

## 第2章 变量和简单的数据类型

之后会采用pycharm来进行实例运行 因为用习惯了😁

### 变量

🦉 非常的浅显易懂

```
message = "Hello Python world!"
print(message)
message = "Hello Python Crash Course world!"
print(message)
#在程序中，可随时修改变量的值，而Python将始终记录变量的最新值
```

```
D:\Anaconda\envs\streamlit\python.exe "E:\Python编程 从入门到实践\example\2. 变量和简单的数据类型.py"
Hello Python world!
Hello Python Crash Course world!
```

message只是一个名字 而赋给它的值定义了它的类型 如果是5那就是整型 如果是""那就是字符串

Python是动态类型但**强类型语言**

```
10 + "a" # ❌ 直接报错
```

因此Python的变量不是“容器”，而是“标签” => Python用来做AI是非常合适的 因为AI需要随时调参讲究的是一个动态的变化 模型结构本身就是运行期对象，**语言必须允许结构动态变化**

#### 💡 Tip

变量的命名规范

1.变量名只能包含字母、数字和下划线。变量名能以字母或下划线开头，但不能以数字开头。

例： message\_1 ✅ 1\_message ❌

2.变量名不能包含空格

3.不要将Python关键字和函数名用作变量名 例如： print、input等等

⚠️（剩余是工程上的规范 一是变量名应既简短又具有描述性，二是慎用小写字母l和大写字母O，因为它们可能被人错看成数字1和0）

#### #2.2.2 使用变量时命名错误

```
message = "Hello Python Crash Course reader!"
print(mesage)
```

```
D:\Anaconda\envs\streamlit\python.exe "E:\Python编程 从入门到实践\example\2. 变量和简单的数据类型.py"
```

```
Traceback (most recent call last):
```

```
File "E:\Python编程 从入门到实践\example\2. 变量和简单的数据类型.py", line 9, in <module>
```

```
    print(mesage)
```

```
    ^^^^^
```

```
NameError: name 'mesage' is not defined. Did you mean: 'message'?
```

#### ⚠️ Caution

上面的代码块我只截取了相关的部分 具体的整体章节代码在example文件夹下的2.变量和简单的数据类型.py中

通过Python解释器提供的报错信息可以发现是代码的第九行 一般变量xxx未定义的报错都是Python无法识别你提供的变量名 **根本在于使用变量前没有给它赋值** 这里的**message**可以当作一个**新变量**

## 字符串

**字符串** (string) 就是一系列字符。在Python 中，用引号引起的都是字符串（可以是单引号也可以是双引号）

### 比较复杂的字符串

```
text = '''老师说: "今天的作业主题是'Python' 的字符串处理", 请大家认真完成, 并在作业中写明: '我已经理解了双引号"和单引号'的区别'。" '''
print(text)
```

#这里采用了三引号 能够实现以下两个功能（一般用不上） **PS:** 中文的“和英文的”是有区别的

#1. 可以跨行

#2. 里面可以随便放'和"

#如果不想用'''三引号却又想在字符串里用"如何解决?

#采用转义字符

```
message = "The language \"Python\" is named after Monty Python, not the snake."
```

```
print(message)
```

输出: The language "Python" is named after Monty Python, not the snake.

#这里的\"就是告诉解释器不需要扫描匹配"字符直接输出=>也就是\"在解释器看来就是直接输出"

### Important

#### 字符串相关操作函数

##### 1.title()函数

```
# title() 将每个单词的首字母转换为大写
name = "tian lin ying"
print(name.title())
```

#输出: Tian Lin Ying

##### 2.upper()/lower()函数

```
# upper()/lower() 将每个单词大写/小写
name = "Tian Lin Ying"
print(name.upper())
print(name.lower())
```

#输出: TIAN LIN YING

tian lin ying

## 在字符串中使用变量

```
first_name = "lin ying"
last_name = "tian"
fullname = f"{first_name} {last_name}"
print(fullname)
```

#输出: lin ying tian

### 💡 Tip

要在字符串中插入变量的值，可先在左引号前加上字母f，再将要插入的变量放在花括号内。

这种字符串称为**f字符串**（format-设置格式）Python通过把花括号内的变量替换为其值来设置字符串的格式

```
first_name = "lin ying"
last_name = "tian"
fullname = f"{first_name} {last_name}"
print(f"Hello, {fullname.title()}!")
```

#输出: Hello, Lin Ying Tian!

也可以使用**f字符串**来创建消息

```
first_name = "lin ying"
last_name = "tian"
fullname = f"{first_name} {last_name}"
message = f"Hello, {fullname.title()}!"
print(message)
```

#输出: Hello, Lin Ying Tian!

#将消息赋给了一个变量 让最后的函数调用print()简单得多

### 实际案例:

```
out_name = out_nameitems[attribute_box.currentIndex()]
sql_e1 = textwrap.dedent(f'''
    SELECT
        '当前表' as 表名,
        COUNT(*) as 记录数,
        SUM(a.{attribute_box.currentText()}) as {out_name}总数
    FROM {textbox_30.text()} a
    WHERE a.init_date between 20251001 and 20251031
    UNION ALL
    SELECT
        '历史表' as 表名,
        COUNT(*) as 记录数,
        SUM(a.{attribute_box.currentText()}) as {out_name}总数
    FROM {textbox_20.text()}@uf20_his a
    WHERE a.init_date between 20251001 and 20251031
''')
```

```
out_sql.setPlainText(sql_e1)
```

这里的代码有一点复杂 因为涉及到了`pyqt`的知识 可以把**花括号里的内容当作变量** 因为很多部分基本上都差不多的 因此脚本只需要替换变的地方即可=>用**变量**来实现，最后用**f字符串**进行拼接

## 使用制表符或换行符来添加空白

```
#制表符和换行符
print("Python")
print("\tPython")

#输出：
#Python
#    Python
#\t缩进4个字符

print("Languages:\nPython\nC\nJavaScript")

#输出：
#Languages:
#Python
#C
#JavaScript
#\n换行

print("Languages:\n\tPython\n\tC\n\tJavaScript")

#输出：
#Languages:
#    Python
#    C
#    JavaScript
```

## Important

### 字符串相关函数

#### 1.rstrip()函数

```
#rstrip() 删除字符串右端空白
favorite_language = ' python '
print(favorite_language.rstrip())
```

#输出：（此处为空格）python

#### 2.lstrip()函数

```
#lstrip() 删除字符串左端空白
favorite_language = ' python '
print(favorite_language.lstrip())
```

#输出：python（此处为空格）

#### 3.strip()函数

```
#strip() 删除字符串两端空白
favorite_language = ' python '
print(favorite_language.strip())
```

#输出: python

#### 4.removeprefix()函数

```
#removeprefix() 删除前缀
nostarch_url = 'https://nostarch.com'
simple_url = nostarch_url.removeprefix('https://')
print(simple_url)
```

#输出: nostarch.com

#### 5.removesuffix()函数

```
#removesuffix() 删除后缀
filename = "python_notes.txt"
print(filename.removesuffix('.txt'))
```

#输出: python\_notes

## 数

### Tip

#### 整数

加减乘除

\*\*表示乘方运算

#### 浮点数

```
print(0.2 + 0.1)
#输出: 0.30000000000000004
```

#结果包含的小数位数可能是不确定的 所有编程语言都存在这种问题，没有什么可担心的 后续会讲处理多余小数位的方式

#### 整数和浮点数

- 1.将任意两个数相除，结果总是浮点数
- 2.只要有操作数是浮点数，默认得到的就总是浮点数，即便结果原本为整数

#### 数中的下划线

在书写很大的数时，可使用下划线将其中的位分组，使其更清晰易读

```
universe_age = 14_000_000_000
print(universe_age)
```

#输出: 14000000000  
#在存储这种数时, Python会忽略其中的下划线

### 同时给多个变量赋值

```
x, y, z = 0, 0, 0
```

用逗号将变量名分开; 对于要赋给变量的值, 也需要做同样的处理。Python将按顺序将每个值赋给对应的变量。只要变量数和值的个数相同, Python就能正确地将变量和值关联起来。

### 常量

Python没有内置的常量类型, 但Python程序员会使用全大写字母来指出应将某个变量视为常量

```
MAX_CONNECTIONS = 5000
```

## 注释

在Python中, 注释用井号(#)标识。井号后面的内容都会被Python解释器忽略

## 第3章 列表

### 3.1 列表介绍

列表(list)由一系列按特定顺序排列的元素组成

在Python中, 用方括号([])表示列表, 用逗号分隔其中的元素。

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

#输出: ['trek', 'cannondale', 'redline', 'specialized']  
#Python将打印列表的内部表示, 包括方括号

#### Tip

#### 1. 访问列表元素

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])
```

#输出: trek  
#就当C语言的数组用! 直接访问下标 不同点是C语言的数组是单一数据类型而列表能放很多种数据类型  
#例如lists = ['trek', 'cannondale', 'redline', 'specialized', 1]也可行!

#小测试 🤖 (思考一下这个代码输出的结果是啥):  
print(bicycles[0].title())

## 2.索引从0而不是1开始

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[1])
print(bicycles[3])
```

#输出:

#cannondale

#specialized

#bingo!也是跟C语言一模一样

### 特殊语法

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[-1])
```

#输出:

#specialized

# **100** 可以实现现在不知道列表长度的情况下访问最后的元素 这种语法也适用于其他负数索引

#索引-2返回倒数第二个列表元素, 索引-3返回倒数第三个列表元素, 依此类推

## 3.使用列表中的各个值

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
message = f"My first bicycle was a {bicycles[0].title()}."
print(message)
```

#输出: My first bicycle was a Trek.

## 3.2 修改、添加和删除元素

创建的大多数列表将是动态的, 这意味着列表创建后, 将随着程序的运行增删元素

### Tip

#### 1.修改列表元素

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles[0] = 'ducati'
print(motorcycles)
```

#输出:

#[ 'honda', 'yamaha', 'suzuki' ]

#[ 'ducati', 'yamaha', 'suzuki' ]

😄 相信学过C语言的你能理解

#你可以修改任意列表元素的值, 而不只是第一个元素的值

#### 2.在列表中添加元素

#在列表末尾添加元素



```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles.append('ducati')
print(motorcycles)
```

#输出:

```
#[ 'honda', 'yamaha', 'suzuki']
#[ 'honda', 'yamaha', 'suzuki', 'ducati']
```

#append()方法将元素'ducati'添加到列表末尾

#append()方法让动态地创建列表易如反掌

```
motorcycles = []
motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')
print(motorcycles)
```

#输出: ['honda', 'yamaha', 'suzuki']

#在列表中插入元素

#使用insert()方法可在列表的任意位置添加新元素

#需要指定新元素的索引和值

```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.insert(0, 'ducati')
print(motorcycles)
#输出: ['ducati', 'honda', 'yamaha', 'suzuki']
```

### 3.从列表中删除元素

#使用del语句删除元素

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
del motorcycles[0]
print(motorcycles)
```

#输出:

```
#[ 'honda', 'yamaha', 'suzuki']
#[ 'yamaha', 'suzuki']
```

 #使用del可删除任意位置的列表元素，只需要知道其索引即可

```
#使用pop()方法删除元素
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
popped_motorcycle = motorcycles.pop()
print(motorcycles)
print(popped_motorcycle)
```

```
#输出: ['honda', 'yamaha', 'suzuki']
#['honda', 'yamaha']
#suzuki
```

#pop()默认删除列表末尾的元素

```
#删除列表中任意位置的元素
#使用pop()删除列表中任意位置的元素，只需要在括号中指定要删除的元素的索引即可
motorcycles = ['honda', 'yamaha', 'suzuki']
first_owned = motorcycles.pop(0)
print(f"The first motorcycle I owned was a {first_owned.title()}.")
```

#输出: The first motorcycle I owned was a Honda.

#🔔注意注意:

###每当你使用pop()时，被弹出的元素就不再在列表中了!!!

PS:如何区分使用del还是pop()

需要使用删除的值就用pop();其余两者问题都不大

#### 4.根据值删除元素

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
motorcycles.remove('ducati')
print(motorcycles)
```

#输出:

```
#['honda', 'yamaha', 'suzuki', 'ducati']
#['honda', 'yamaha', 'suzuki']
```

PS:

remove()方法只删除第一个指定的值。如果要删除的值可能在列表中出现多次，就需要使用循环  
当然remove()函数中的内容可以用变量代替

```
too_expensive = 'ducati'
motorcycles.remove(too_expensive)
```

### 3.3 管理列表

#### ① Note

##### 1.使用sort()方法对列表进行永久排序

`#sort()`方法能永久地修改列表元素的排列顺序

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort() #这里其实默认了reverse=False 等同于用cars.sort(reverse=False)
print(cars)
```

#输出: ['audi', 'bmw', 'subaru', 'toyota']

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort(reverse=True) #与字母顺序相反的顺序排列列表元素
print(cars)
```

#输出: ['toyota', 'subaru', 'bmw', 'audi']

## 2.使用sorted()函数对列表进行临时排序

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print("Here is the original list:")
print(cars)
print("\nHere is the sorted list:")
print(sorted(cars))
print("\nHere is the original list again:")
print(cars)
```

#输出:

```
#Here is the original list:
#['bmw', 'audi', 'toyota', 'subaru']
```

```
#Here is the sorted list:
#['audi', 'bmw', 'subaru', 'toyota']
```

```
#Here is the original list again:
#['bmw', 'audi', 'toyota', 'subaru']
```

PS: 通过这个案例可以发现`sorted()`函数是不会对列表发生改变的

## 3.反向打印列表

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)
```

#输出:

```
#[ 'bmw', 'audi', 'toyota', 'subaru']
#[ 'subaru', 'toyota', 'audi', 'bmw']
```

PS: `reverse()`方法会永久地修改列表元素的排列顺序,但可随时恢复到原来的排列顺序,只需对列表再次调用`reverse()`即可  
且`reverse()`不是按与字母顺序相反的顺序排列列表元素,只是反转列表元素的排列顺序

## 4.确定列表的长度

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(len(cars))
```

#输出: 4

## 第4章 操作列表

### 4.1 遍历整个列表

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician)
```

#输出:  
#alice  
#david  
#carolina

#### 💡 Tip

刚开始使用循环时请牢记，不管列表包含多少个元素，每个元素都将被执行循环指定的步骤。如果列表包含100万个元素，Python就将重复执行指定的步骤100万次，而且通常速度非常快。

```
#在for循环中执行更多的操作
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
```

#输出:  
#Alice, that was a great trick!  
#David, that was a great trick!  
#Carolina, that was a great trick!

### 4.2 避免缩进错误

#### ⚠ Caution

注意循环语句的缩进问题

```
#1. 忘记缩进
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician)
```

```
D:\Anaconda\envs\streamlit\python.exe "E:\Python编程 从入门到实践\example\4.操作列表.py"
File "E:\Python编程 从入门到实践\example\4.操作列表.py", line 14
    print(magician)
    ^
IndentationError: expected an indented block after 'for' statement on line 13
```

### #2. 不必要的缩进

```
message = "Hello Python world!"
print(message)
```

```
D:\Anaconda\envs\streamlit\python.exe "E:\Python编程 从入门到实践\example\4.操作列表.py"
File "E:\Python编程 从入门到实践\example\4.操作列表.py", line 18
    print(message)
IndentationError: unexpected indent
```

### #3. 遗漏冒号

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians
    print(magician)
```

##这里需要注意的是Python的循环或者条件语句基本上都会用到：**get**到这个点就好了！

```
D:\Anaconda\envs\streamlit\python.exe "E:\Python编程 从入门到实践\example\4.操作列表.py"
File "E:\Python编程 从入门到实践\example\4.操作列表.py", line 22
    for magician in magicians
                                ^
SyntaxError: expected ':'
```

## 4.3 创建数值列表

### 📌 Important

#### 1. 使用range()函数

```
for value in range(1, 5):
    print(value)
```

#输出:

```
#1
#2
#3
#4
```

#看到这个结果可能有些许疑惑

#range() 函数让Python从指定的第一个值开始数，并在到达指定的第二个值时停止

PS: 可以理解为高中数学集合的左闭右开 😊

#要打印数1~5，需要使用range(1,6)

```
for value in range(1, 6):
    print(value)
```

#### 2. 使用range()创建数值列表

```
numbers = list(range(1, 6))
print(numbers)
```

#输出: [1, 2, 3, 4, 5]

PS: 这里的list()函数相当于直接将range(1,6)强制转换成列表对象 可以类比成C语言中的强制类型转换(int)'123'

```
!!! 补充!!!  
r = range(1,6)  
print(r)  
print(type(r))
```

#输出:

#range(1, 6)

#<class 'range'>

这里可以发现`range`也是一个数据类型 `range`对象采用惰性求值的策略，它只存储生成序列的起始值、终止值和步长，以及计算下一个值的方法。只有当你真正需要用到序列中的数字时（例如在`for`循环中迭代），它才会临时计算并提供给你。

📌 所以最上面那个例子的`numbers`需要进行`list()`函数进行类型转换才能逐个显示1-5的数

#在使用`range()`函数时，还可指定步长 给这个函数指定第三个参数，`python`将根据这个步长来生成数  
`even_numbers = list(range(2, 11, 2))`

`print(even_numbers)`

#输出: [2, 4, 6, 8, 10]

在这个示例中，`range()`函数从2开始数，然后不断地加2，直到达到或超过终值（11）

#小案例

#生成前10个整数的平方

`squares = []`

`for value in range(1, 11):`

`square = value ** 2`

`squares.append(square)`

`print(squares)`

#输出: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

### 3.对数值列表执行简单的统计计算

```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]  
print(min(digits))  
print(max(digits))  
print(sum(digits))
```

#输出:

#0

#9

#45

### 4.列表推导式

```
squares = [value**2 for value in range(1, 11)]  
print(squares)
```

#输出: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

#列表推导式（`list comprehension`）将`for`循环和创建新元素的代码合并成一行，并自动追加新元素。

要使用这种语法，首先指定一个描述性的列表名，如`squares`。然后指定一个左方括号，并定义一个表达式，用于生成要存储到列表中的值。在这个示例中，表达式为`value**2`，它计算平方值。接下来，编写一个`for`循环，用于给表达式提供值，再加上右方括号。在这个示例中，`for`循环为`for value in range(1,11)`，它将值1~10提供给表达式`value**2`。注意，这里的`for`语句末尾没有冒号。

## 📌 Note

🐼 接下来考考你(重点看看Text6和Text7)

🌱 答案在这: [0基础入门Python-小测试](#)

### Test1

使用一个for循环打印数1~20 (含)

### Test2

创建一个包含数1~1000000的列表，再使用一个for循环将这些数打印出来。（如果输出的时间太长，按 Ctrl+C停止输出，或关闭输出窗口。）

### Text3

创建一个包含数1~1000000的列表，再使用min()和max()核实该列表确实是从1开始、到1000000结束的。另外，对这个列表调用函数 sum()，看看Python将100万个数相加需要多长时间。

### Text4

通过给range()函数指定第三个参数来创建一个列表，其中包含1~20的奇数；再使用一个for循环将这些数打印出来。

### Text5

创建一个列表，其中包含3~30内能被3整除的数，再使用一个for循环将这个列表中的数打印出来。

### Text6

将同一个数乘三次称为立方。例如，在Python中，2的立方用 2\*\*3 表示。创建一个列表，其中包含前10个整数（1~10）的立方，再使用一个for循环将这些立方数打印出来。

### Text7

使用列表推导式生成一个列表，其中包含前10个整数的立方。

## 4.4 使用列表的一部分

### 切片

要创建切片，可指定要使用的第一个元素和最后一个元素的索引。与range()函数一样，Python在到达指定的第二个索引之前的元素时停止。（同样可以理解为左闭右开）

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[0:3])
#输出: ['charles', 'martina', 'michael']

players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[1:4])
```

```
#输出: ['martina', 'michael', 'florence']
```

如果没有指定第一个索引, 自动从列表开头开始

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']  
print(players[:4])
```

```
#输出: ['charles', 'martina', 'michael', 'florence']
```

要让切片终止于列表末尾, 也可使用类似的语法

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']  
print(players[2:])
```

```
#输出: ['michael', 'florence', 'eli']
```

负数索引返回与列表末尾有相应距离的元素, 因此可以输出列表末尾的任意切片

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']  
print(players[-3:])
```

```
#输出: ['michael', 'florence', 'eli']
```

## 遍历切片

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']  
print("Here are the first three players on my team:")  
for player in players[:3]:  
    print(player.title())
```

```
#输出:
```

```
Here are the first three players on my team:
```

```
Charles
```

```
Martina
```

```
Michael
```

👉 !!!只需要记住列表的切片还是列表 只不过内容是列表的部分

## 复制列表

### 💡 Tip

```
my_foods = ['pizza', 'falafel', 'carrot cake']  
friend_foods = my_foods[:]
```

```
print("My favorite foods are:")  
print(my_foods)
```

```
print("\nMy friend's favorite foods are:")  
print(friend_foods)
```

```
#输出:
```

```
My favorite foods are:
```

```
['pizza', 'falafel', 'carrot cake']
```

```
My friend's favorite foods are:
```

```
['pizza', 'falafel', 'carrot cake']
```

#为了核实确实有两个列表, 下面在每个列表中都添加一种食品, 并确认每个列表都记录了相应的人喜欢的食品

```
my_foods.append('cannoli')
```



```
friend_foods.append('ice cream')

print("My favorite foods are:")
print(my_foods)

print("\nMy friend's favorite foods are:")
print(friend_foods)

#输出:
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'cannoli']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'ice cream']
```

### !!!容易犯的误区

```
my_foods = ['pizza', 'falafel', 'carrot cake']

#这是行不通的:
friend_foods = my_foods

my_foods.append('cannoli')
friend_foods.append('ice cream')
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)

#输出:
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'cannoli', 'ice cream']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'cannoli', 'ice cream']
```

这里将`my_foods`赋给`friend_foods`，而不是将`my_foods`的副本赋给`friend_foods`。这种语法实际上是让 `python`将新变量`friend_foods`关联到已与`my_foods`相关联的列表，因此这两个变量指向同一个列表。

因此创建副本需要`friend_foods = my_foods[:]`

#### Note

#### 小小的练习一下

使用切片 来打印列表中间的三个元素

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
```

```
#我一猜你会用players[1:4]
```

```
#想想如何不用指定的下标访问呢?如果列表很长的话难不成要一个一个数过去?
```

```
#bingo!想到用len()函数来获取列表长度
```

```
n = len(players)
```

```
print(players[int((n - 1) / 2) - 1:int((n - 1) / 2) + 2])
```

PS: 这里用了int()强制类型转换 想想为什么?

第3章数的部分提到过将任意两个数相除,结果总是浮点数 那么我们切片的索引应该是整数吧=>所以需要  
进行类型转换

还有要注意我们切片的范围,左闭右开噢!

## 4.5 元组

Python将不能修改的值称为**不可变的**,而不可变的列表称为**元组** (tuple)

### 定义元组

元组看起来很像列表,但使用圆括号而不是方括号来标识。定义元组后,就可使用索引来访问其元素,就像访问列表元素一样。

```
#如果有一个大小不应改变的矩形,可将其长度和宽度存储在一个元组中,从而确保它们是不能修改的
```

```
dimensions = (200, 50)
```

```
print(dimensions[0])
```

```
print(dimensions[1])
```

```
#输出:
```

```
200
```

```
50
```

尝试修改元组dimensions的一个元素,看看结果如何:

```
dimensions = (200, 50)
```

```
dimensions[0] = 250
```

```
D:\Anaconda\envs\streamlit\python.exe "E:\Python编程 从入门到实践\example\4.操作列表.py"
```

```
Traceback (most recent call last):
```

```
File "E:\Python编程 从入门到实践\example\4.操作列表.py", line 108, in <module>
```

```
    dimensions[0] = 250
```

```
    ~~~~~^
```

```
TypeError: 'tuple' object does not support item assignment
```

在代码试图修改矩形的尺寸时, Python会报错。

### ⚠ Caution

严格地说,元组是由逗号标识的,圆括号只是让元组看起来更整洁、更清晰。如果你要定义只包含一个元素的元组,必须在这个元素后面加上逗号

```
my_t = (3)
for item in my_t:
    print(item)
```

D:\Anaconda\envs\streamlit\python.exe "E:\Python编程 从入门到实践\example\4.操作列表.py"

Traceback (most recent call last):

File "E:\Python编程 从入门到实践\example\4.操作列表.py", line 112, in <module>

for item in my\_t:

TypeError: 'int' object is not iterable

解释器会把不加逗号的元组当成**单个元素它本身的数据类型**来看 而不是当**元组** 因此无法进行遍历

```
my_t = (3,)
for item in my_t:
    print(item)
print(type(my_t))
```

#输出:

#3

#<class 'tuple'>

加了逗号就会当作元组来看

## 遍历元组

```
dimensions = (200, 50)
for dimension in dimensions:
    print(dimension)
```

#输出:

200

50

## 修改元组变量

```
dimensions = (200, 50)
print("Original dimensions:")
for dimension in dimensions:
    print(dimension)

dimensions = (400, 100)
print("\nModified dimensions:")
for dimension in dimensions:
    print(dimension)
```

#输出:

Original dimensions:

200

50

Modified dimensions:

400

100

相当于将一个新元组关联到变量`dimensions` 并没有改变原先元组中的元素 因此合法

## 总结:

相比于列表，元组是更简单的数据结构。如果需要存储一组在程序的整个生命周期内都**不变的值**，就可以使用元组=>**不可改变的列表+定义采用的是圆括号**

### 📌 Important

#### 补充一个知识点方便写代码

如果用pycharm的话它自带自动缩进：Ctrl+Alt+L

列表的内容到此结束咯! 🌸

## 第5章 if语句

### 📌 Note

很多例子是书上的 我觉得有一丝繁琐 能get到点即可

### 5.1 一个简单的示例

```
# 一个简单的示例
cars = ['audi', 'bmw', 'subaru', 'toyota']
for car in cars:
    if car == 'bmw':
        print(car.upper())
    else:
        print(car.title())
```

#输出:  
Audi  
BMW  
Subaru  
Toyota

### 5.2 条件测试

#### 检查是否相等

```
car = 'bmw'
print(car == 'bmw')
```

#输出: True

#### 如何在检查是否相等时忽略大小写

#不管car里是什么 调用lower()全部小写进行比较 而且lower()方法并没有影响存储在变量car中的值

```
car = 'Audi'  
print(car.lower() == 'audi')
```

#输出: True

## 检查是否不等

```
requested_topping = 'mushrooms'  
if requested_topping != 'anchovies':  
    print("Hold the anchovies!")
```

#输出: Hold the anchovies!

## 数值比较

```
answer = 17  
if answer != 42:  
    print("That is not the correct answer. Please try again!")
```

#输出: That is not the correct answer. Please try again!

## 检查多个条件

```
#使用and检查多个条件  
age_0 = 22  
age_1 = 18  
print(age_0 >= 21 and age_1 >= 21)
```

#输出: False

### 💡 Tip

为了改善可读性, 可将每个条件测试都分别放在一对括号内, 但并非必须这样做。如果使用括号, 条件测试将类似于下面这样:

```
(age_0 >= 21) and (age_1 >= 21)
```

```
#使用or检查多个条件  
age_0 = 22  
age_1 = 18  
print(age_0 >= 21 or age_1 >= 21)
```

#输出: True

## 检查特定的值是否在列表中

```
requested_toppings = ['mushrooms', 'onions', 'pineapple']
print('mushrooms' in requested_toppings)
print('pepperoni' in requested_toppings)
```

#输出:  
True  
False

### 检查特定的值是否不在列表中

```
banned_users = ['andrew', 'carolina', 'david']
user = 'marie'
if user not in banned_users:
    print(f"{user.title()}, you can post a response if you wish.")
```

#输出: Marie, you can post a response if you wish.

### 布尔表达式

布尔表达式不过是条件测试的**别名**罢了。与条件表达式一样，布尔表达式的结果要么为**True**，要么为**False**。

布尔值通常用于**记录条件** 如游戏是否正在运行或用户是否可以编辑网站的特定内容

```
game_active = True
can_edit = False
```

## 5.3 if语句

### 简单的if语句

```
age = 19
if age >= 18:
    print("You are old enough to vote!")
```

#输出: You are old enough to vote!

### if-else语句

```
age = 17
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote as soon as you turn 18!")
```

#输出:  
#Sorry, you are too young to vote.  
#Please register to vote as soon as you turn 18!

### if-elif-else语句

```
age = 12
if age < 4:
    print("Your admission cost is $0.")
elif age < 18:
    print("Your admission cost is $25.")
else:
    print("Your admission cost is $40.")

#输出: Your admission cost is $25.

#简洁版:
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
else:
    price = 40
print(f"Your admission cost is ${price}.")
```

### 使用多个elif代码块

```
age = 12

if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
else:
    price = 20

print(f"Your admission cost is ${price}.")

#输出: Your admission cost is $25.
```

### 省略else代码块

```
age = 12

if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
elif age >= 65:
    price = 20

print(f"Your admission cost is ${price}.")
```

### 测试多个条件

```

requested_toppings = ['mushrooms', 'extra cheese']
if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
if 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")
if 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")
print("\nFinished making your pizza!")

#输出:
#Adding mushrooms.
#Adding extra cheese.

#Finished making your pizza!

```

如果像下面这样转而使用if-elif-else语句，代码将不能正确运行，因为只要有一个条件测试通过，就会跳过余下的条件测试

```

requested_toppings = ['mushrooms', 'extra cheese']

if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
elif 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")
elif 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")

print("\nFinished making your pizza!")

#输出:
#Adding mushrooms.

#Finished making your pizza!

```

总之，如果只想运行一个代码块，就使用if-elif-else语句

如果要运行多个代码块，就使用一系列独立的if语句

## 5.4 使用if语句处理列表

### 检查特殊元素

```

#循环语句+条件判断

requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']

for requested_topping in requested_toppings:
    if requested_topping == 'green peppers':
        print("Sorry, we are out of green peppers right now.")
    else:
        print(f"Adding {requested_topping}.")

print("\nFinished making your pizza!")

```



```
#输出:
#Adding mushrooms.
#Sorry, we are out of green peppers right now.
#Adding extra cheese.

#Finished making your pizza!
```

## 确定列表非空

```
if requested_toppings:
    for requested_topping in requested_toppings:
        print(f"Adding {requested_topping}.")
    print("\nFinished making your pizza!")
else:
    print("Are you sure you want a plain pizza?")
```

```
#输出: Are you sure you want a plain pizza?
```

## 使用多个列表

```
available_toppings = ['mushrooms', 'olives', 'green peppers',
                      'pepperoni', 'pineapple', 'extra cheese']
requested_toppings = ['mushrooms', 'french fries', 'extra cheese']
for requested_topping in requested_toppings:
    if requested_topping in available_toppings:
        print(f"Adding {requested_topping}.")
    else:
        print(f"Sorry, we don't have {requested_topping}.")

print("\nFinished making your pizza!")
```

```
#输出:
#Adding mushrooms.
#Sorry, we don't have french fries.
#Adding extra cheese.
#Finished making your pizza!
```

### Note

#### 小补充 虽然可能用不上

```
#or和and的短路问题
is_admin = True

if(is_admin or print("显示管理员面板")==None):
    print(1)
```

```
#输出: 1
```

这里的is\_admin已经是True了 就不会再执行后面的条件 这就是经典的or的短路问题

```
is_admin = False
if(is_admin and print("显示管理员面板")==None):
```

```
print(1)
```

#无输出

这里的`is_admin = False` 整个`and`语句就不可能再会是`True` 就不会再执行后面的条件 直接跳过`if`语句块 这就是经典的`and`的短路问题

第5章到此结束! 🌸

## 第6章 字典

理解字典后，你就能够更准确地为各种真实物体建模。你可以创建一个表示人的字典，然后在其中存储你想存储的任何信息：姓名、年龄、地址，以及可以描述这个人的任何其他方面。

### 6.1 一个简单的字典

```
alien_0 = {'color': 'green', 'points': 5}
```

```
print(alien_0['color'])
print(alien_0['points'])
```

#输出:

green

5

### 6.2 使用字典

在Python中，字典（dictionary）是一系列**键值对**。每个**键**都与一个**值**关联，可以使用键来访问与之关联的值。

键值对包含两个相互关联的值。当你指定键时，Python将返回与之关联的值。键和值之间用冒号分隔，而键值对之间用逗号分隔。

#### 访问字典中的值

```
alien_0 = {'color': 'green'}
print(alien_0['color'])
```

#输出: green

```
alien_0 = {'color': 'green', 'points': 5}
```

```
new_points = alien_0['points']
print(f"You just earned {new_points} points!")
```

#输出: You just earned 5 points!

#### 添加键值对

字典是一种动态结构，可随时在其中添加键值对。要添加键值对，可依次指定字典名、用方括号括起来的键和与该键关联的值。

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)

alien_0['x_position'] = 0 #添加键值对操作
alien_0['y_position'] = 25 #添加键值对操作
print(alien_0)

#输出:
{'color': 'green', 'points': 5}
{'color': 'green', 'points': 5, 'x_position': 0, 'y_position': 25}
```

## 从创建一个空字典开始

```
alien_0 = {} #可以类比于创建一个新列表 list=[] 后面再进行list.append()操作进行添加元素

alien_0['color'] = 'green'
alien_0['points'] = 5

print(alien_0)

#输出:
{'color': 'green', 'points': 5}
```

## 修改字典中的值

```
alien_0 = {'color': 'green'}
print(f"The alien is {alien_0['color']}.")

alien_0['color'] = 'yellow'
print(f"The alien is now {alien_0['color']}.")

#输出:
The alien is green.
The alien is now yellow.
```

## 删除键值对

对于字典中不再需要的信息，可使用del语句将相应的键值对彻底删除。在使用del语句时，必须指定字典名和要删除的键。

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)

del alien_0['points']
print(alien_0)

#输出:
{'color': 'green', 'points': 5}
{'color': 'green'}
```

由类似对象组成的字典（这个能看懂代码就行 感觉书中这个小标题很多余）

```

favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python', #这里在最后一个键值对后面也加上逗号，为以后添加键值对做好准备
}

language = favorite_languages['sarah'].title()
print(f"Sarah's favorite language is {language}.")

#输出: Sarah's favorite language is C.

```

## 使用get()来访问值

```

alien_0 = {'color': 'green', 'speed': 'slow'}
print(alien_0['points'])

```

这里可以发现alien\_0并没有points这个键 那么肯定会有问题

```

Traceback (most recent call last):
  File "E:\Python编程 从入门到实践\example\6.字典.py", line 60, in <module>
    print(alien_0['points'])
          ~~~~~^~~~~~
KeyError: 'points'

```

为了避免出现上述的问题 我们可以采用get()的方法来访问值

```

alien_0 = {'color': 'green', 'speed': 'slow'}

point_value = alien_0.get('points', 'No point value assigned.')
print(point_value)

#输出: No point value assigned.

```

如果指定的键有可能不存在，应考虑使用get()方法，而不要使用方括号表示法

### Note

在调用get()时，如果没有指定第二个参数且指定的键不存在，Python将返回值None，这个特殊的值表示没有相应的值 因此用get()比用方括号好！

```

alien_0 = {'color': 'green', 'speed': 'slow'}
point_value = alien_0.get('points')
print(point_value)

#输出: None

```

## 6.3 遍历字典

鉴于字典可能包含大量数据，Python支持对字典进行遍历。字典可用于以各种方式存储信息，因此有多种遍历方式：既可遍历字典的所有键值对，也可只遍历键或值。

### 遍历所有的键值对

```
user_0 = {
    'username': 'efermi',
    'first': 'enrico',
    'last': 'fermi',
}
```

```
for key, value in user_0.items():
    print(f"\nkey: {key}")
    print(f"Value: {value}")
```

#输出:

```
Key: username
Value: efermi
```

```
Key: first
Value: enrico
```

```
Key: last
Value: fermi
```

#上述的循环写法并不是唯一的

```
for a, b in user_0.items():
    print(f"\nkey: {a}")
    print(f"Value: {b}")
```

🍷 运行上述循环的结果也是一样的 `key`和`value`只不过是两个变量罢了!

# 🍷 再举一个例子

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}
```

```
for name, language in favorite_languages.items():
    print(f"{name.title()}'s favorite language is {language.title()}")
```

#输出:

```
Jen's favorite language is Python.
Sarah's favorite language is C.
Edward's favorite language is Rust.
Phil's favorite language is Python.
```

#看完这个例子应该对遍历键值对有了一定的理解

## 遍历字典中的所有键

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}
```

```
for name in favorite_languages.keys(): #这里的keys()就不能被替换成其他的了
    #因为这是字典对象的一个属性!

    print(name.title())
```

#输出:

```
Jen
Sarah
Edward
Phil
```

当然了! 在遍历字典时, 会默认遍历所有的键 如果将上述代码中的

`for name in favorite_languages.keys():` 替换成 `for name in favorite_languages:` 也是可以的!

输出将不变哦!!!

#只不过显式地使用`keys()`方法能让代码的可读性更好一些!

### Tip

#### 再来个书上的例子

#能看懂这段代码在干什么你就掌握了!

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}

friends = ['phil', 'sarah']
for name in favorite_languages.keys():
    print(f"Hi {name.title()}.")

    if name in friends:
        language = favorite_languages[name].title()
        print(f"\t{name.title()}, I see you love {language}!")
```

#输出:

```
Hi Jen.
Hi Sarah.
    Sarah, I see you love C!
Hi Edward.
Hi Phil.
    Phil, I see you love Python!
```

#### 按特定的顺序遍历字典中的所有键

### Note

 看到这里了该考考你了

`sort()`和`sorted()`在列表排序中的区别是什么? 相信聪明的你是知道的呦!

```

favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}

for name in sorted(favorite_languages.keys()):
    print(f"{name.title()}, thank you for taking the poll.")

```

## 遍历字典中的所有值

如果你感兴趣的是字典包含的值，可使用**values()**方法。它会返回一个**值列表**，不包含任何键。

```

favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}

print("The following languages have been mentioned:")
for language in favorite_languages.values():
    print(language.title())

```

**#输出：**

```

The following languages have been mentioned:
Python
C
Rust
Python

```

那么问题来了？如果字典的值包含很多重复项我该如何剔除重复项捏？办法就是采用**集合(set)**

```

favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}

print("The following languages have been mentioned:")
for language in set(favorite_languages.values()):
    print(language.title())

```

**#输出：**

```

The following languages have been mentioned:
Python
C
Rust

```

🌟 `favorite_languages.values()`返回的值列表跟列表还是有一些区别的

```
print(type(favorite_languages.values()))
#输出: <class 'dict_values'>
```

首先数据类型就不是列表list 其次它无法用索引来进行访问，只能允许遍历

```
values = favorite_languages.values()
print(values[0])
```

D:\Anaconda\envs\streamlit\python.exe "E:\Python编程 从入门到实践\example\6.字典.py"

Traceback (most recent call last):

```
File "E:\Python编程 从入门到实践\example\6.字典.py", line 128, in <module>
    print(values[0])
          ~~~~~^^^
```

TypeError: 'dict\_values' object is not subscriptable

### ⚠ Caution

集合和字典很容易混淆，因为它们都是用一对花括号定义的。当花括号内没有键值对时，定义的很可能是集合。不同于列表和字典，集合不会以特定的顺序存储元素。

```
languages = {'python', 'rust', 'python', 'c'}
print(languages)

#输出: {'rust', 'python', 'c'}
```

## 6.4 嵌套

有时候，需要将多个字典存储在列表中或将列表作为值存储在字典中，这称为**嵌套**。

### 字典列表

字典alien\_0包含一个外星人的各种信息，但无法存储第二个外星人的信息，更别说屏幕上全部外星人的信息了。如何管理成群结队的外星人呢？一种办法是创建一个外星人列表，其中每个外星人都是一个字典，包含有关该外星人的各种信息。

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}
```

```
aliens = [alien_0, alien_1, alien_2]
```

```
for alien in alie:
    print(alien)
```

```
#输出:
{'color': 'green', 'points': 5}
{'color': 'yellow', 'points': 10}
{'color': 'red', 'points': 15}
```

更符合现实的情形是，外星人不止三个，而且每个外星人都是用代码自动生成的。在下面的示例中，使用range()生成了30个外星人



#创建一个用于存储外星人的空列表

```
aliens = []
```

#创建30个绿色的外星人

```
for alien_number in range(30): #! !! #循环遍历0-29 总共30次
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
    aliens.append(new_alien)
```

```
for alien in aliens[:5]: #! !! #遍历列表 不指明开始索引的话默认是从0开始的
    #因此是0 1 2 3 4 左闭右开
```

```
    print(alien)
print("...")
```

#显示创建了多少个外星人

```
print(f"Total number of aliens: {len(aliens)}")
```

#输出:

```
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
...
```

```
Total number of aliens: 30
```

#### Note

### 再来一个书上的例子

将前三个外星人修改为黄色、速度中等且值10分

#创建一个用于存储外星人的空列表

```
aliens = []
```

#创建30个绿色的外星人

```
for alien_number in range(30):
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
    aliens.append(new_alien)
```

```
for alien in aliens[:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10
```

#显示前5个外星人

```
for alien in aliens[:5]:
    print(alien)
print("...")
```

```
#输出:
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
...
```

## 在字典中存储列表

```
#存储顾客所点比萨的信息
```

```
pizza = {
    'crust': 'thick',
    'toppings': ['mushrooms', 'extra cheese'],
}
```

```
# 概述顾客点的比萨
```

```
print(f"You ordered a {pizza['crust']}-crust pizza "
      "with the following toppings:")
```

```
for topping in pizza['toppings']:
    print(f"\t{topping}")
```

```
#输出:
```

```
You ordered a thick-crust pizza with the following toppings:
    mushrooms
    extra cheese
```

## 💡 Tip

再来一个书上的例子(能看懂代码即可)

```
favorite_languages = {
    'jen': ['python', 'rust'],
    'sarah': ['c'],
    'edward': ['rust', 'go'],
    'phil': ['python', 'haskell'],
}

for name, languages in favorite_languages.items():
    print(f"\n{name.title()}'s favorite languages are:")
    for language in languages:
        print(f"\t{language.title()}")
```

```
#输出:
```

```
Jen's favorite languages are:
```

```
    Python
    Rust
```

```
Sarah's favorite languages are:
```

```
    C
```

Edward's favorite languages are:

Rust

Go

Phil's favorite languages are:

Python

Haskell

## 在字典中存储字典

```
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
    },

    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
    },
}

for username, user_info in users.items():
    print(f"\nUsername: {username}")
    full_name = f"{user_info['first']} {user_info['last']}"
    location = user_info['location']

    print(f"\tFull name: {full_name.title()}")
    print(f"\tLocation: {location.title()}")
```

第6章到此结束啦! 🍓

---