Python编程从入门到实践

第一部分 基础知识

第1章 起步

配置Python环境教程（采用conda 创建虚拟环境 防止环境混乱 因为不同的项目需要涉及不同版本的安装包）：

[最新版最详细Anaconda新手安装+配置+环境创建教程_anaconda配置-CSDN博客]

VS Code配置Python环境

[10分钟搞定！VS Code配置Python开发环境指南（2025新版） - 知乎]

可能会出现的问题：

> ⚠ **Warning**
>
> PS E:\Python编程 从入门到实践> (D:\Anaconda\shell\condabin\conda-hook.ps1) ; (conda activate streamlit)
>
> >>>>>>>>>>>>>>>>>>>> ERROR REPORT <<<<<<<<<<<<<<<<<<<<
>
> Traceback (most recent call last): File "D:\Anaconda\Lib\site-packages\conda\exception_handler.py", line 16, in **call** return func(*args, **kwargs) ^^^^^^^ File "D:\Anaconda\Lib\site-packages\conda\cli\main.py", line 111, in main_sourced print(activator.execute(), end="") UnicodeEncodeError: 'gbk' codec can't encode character '\u202a' in position 410: illegal multibyte sequence

因为系统环境变量被Unicode控制字符污染 可能是之前复制网页中的系统路径时无意中带入

成功截图：

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**mesage**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_el = 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_el)
```

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

#💯可以实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romove()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

> **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。

TOP 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！ 🌷
