

[笔记][黑马 Python 之 Python 基础 - 5]

Python

[笔记][黑马 Python 之 Python 基础 - 5]

- 154. 模块-01-概念介绍
- 155. 模块-02-使用模块演练
- 156. 模块-03-模块名也是一个标识符
- 157. 模块-04-[科普]pyc文件可以提高程序执行速度
- 158. 高级变量-01-学习目标确定
- 159. 列表-01-作用和定义方式
- 160. 列表-02-存储数据的方式以及从列表中取值
- 161. 列表-03-确认列表常用操作的方法
- 162. 列表-04-从列表中取值和取索引
- 163. 列表-05-修改指定位置的数据
- 164. 列表-06-向列表增加数据
- 165. 列表-07-从列表删除数据
- 166. 列表-08-使用del关键字从列表删除数据
- 167. 列表-09-列表统计及删除方法扩展
- 168. 列表-10-列表的排序和反转
- 169. 列表-11-关键字、函数和方法的特点和区别
- 170. 列表-12-迭代遍历
- 171. 列表-13-列表的应用场景
- 172. 元组-01-特点以及和列表的区别
- 173. 元组-02-元组变量的定义
- 174. 元组-03-元组变量的常用操作
- 175. 元组-04-元组变量的循环遍历
- 176. 元组-05-元组的应用场景
- 177. 元组-06-元组和格式化字符串
- 178. 元组-07-元组和列表之间的转换
- 179. 字典-01-字典的定义格式以及键值对的概念
- 180. 字典-02-变量的定义以及应用场景
- 181. 字典-03-字典的增删改查常用操作
- 182. 字典-04-字典的统计、合并、清空操作
- 183. 字典-05-字典的循环遍历
- 184. 字典-06-字典和列表组合的应用场景
- 185. 字符串-01-定义和基本使用
- 186. 字符串-02-长度、计数、位置方法演练
- 187. 字符串-03-常用方法总览和分类
 - 1) 判断类型 - 9

- 2) 查找和替换 - 7
- 3) 大小写转换 - 5
- 4) 文本对齐 - 3
- 5) 去除空白字符 - 3
- 6) 拆分和连接 - 5

188. 字符串-04-判断空白字符以及学习方法分享

189. 字符串-05-判断数字的三个方法

154. 模块-01-概念介绍

模块是 Python 程序架构的一个核心概念

- **模块** 就好比是 **工具包**，要想使用这个工具包中的工具，就需要 **导入 import** 这个模块
- 每一个以扩展名 **.py** 结尾的 **Python** 源代码文件都是一个 **模块**
- 在模块中定义的 **全局变量**、**函数** 都是模块能够提供给外界直接使用的工具

155. 模块-02-使用模块演练

步骤

- 新建 **hm_10_分隔线模块.py**
 - 复制 **hm_09_打印多条分隔线.py** 中的内容，最后一行 **print** 代码除外
 - 增加一个字符串变量

```
name = "黑马程序员"
```

- 新建 **hm_10_体验模块.py** 文件，并且编写以下代码：

```
import hm_10_分隔线模块

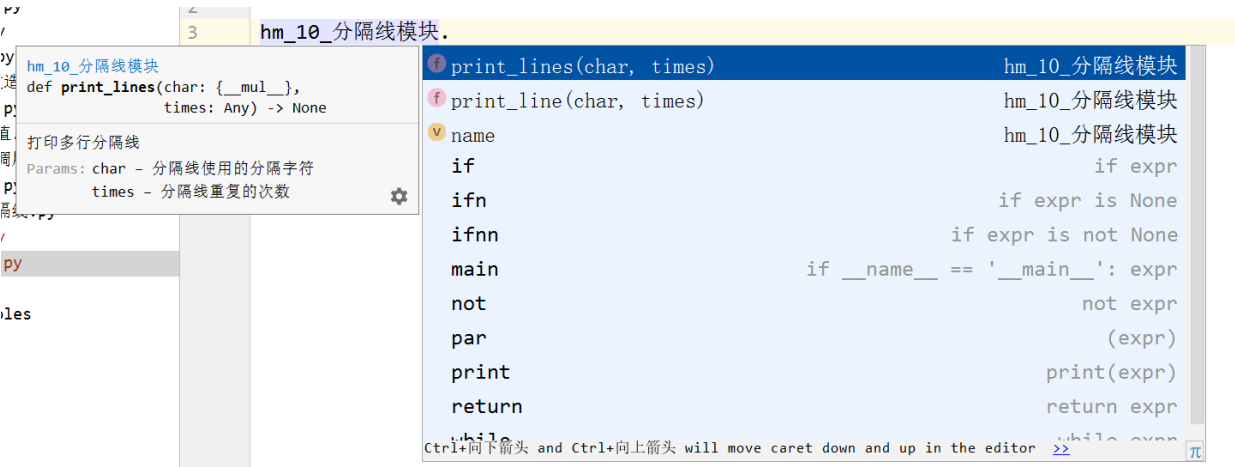
hm_10_分隔线模块.print_line("-", 80)
print(hm_10_分隔线模块.name)
```

体验小结

- 可以在一个 Python 文件中定义变量或者函数
- 然后在另外一个文件中 使用 **import** 导入这个模块
- 导入之后，就可以使用 **模块名.变量** / **模块名.函数** 的方式，使用这个模块中定义的变量或者函数

模块可以让 **曾经编写过的代码** 方便的被 **复用**！

注意：模块名。了之后，出现提示，移动上下箭头，按 **Ctrl+Q** 可以展示文档注释。



156. 模块-03-模块名也是一个标识符

- 标示符可以由 **字母**、**下划线** 和 **数字** 组成
- **不能以数字开头**
- **不能与关键字重名**

注意：如果在给 Python 文件起名时，**以数字开头**是无法在 PyCharm 中通过导入这个模块的

157. 模块-04-[科普]pyc文件可以提高程序执行速度

pyc 文件

c 是 compiled 编译过的意思

操作步骤

1. 浏览程序目录会发现一个 `__pycache__` 的目录
2. 目录下会有一个 `hm_10_分隔线模块.cpython-36.pyc` 文件，`cpython-36` 表示 Python 解释器的版本
3. 这个 `pyc` 文件是由 Python 解释器将 **模块的源码** 转换为 **字节码**
 - `Python` 这样保存 **字节码** 是作为一种启动 **速度的优化**

字节码

- `Python` 在解释源程序时是分成两个步骤的
 1. 首先处理源代码，**编译** 生成一个二进制 **字节码**
 2. 再对 **字节码** 进行处理，才会生成 CPU 能够识别的 **机器码**
- 有了模块的字节码文件之后，下一次运行程序时，如果在 **上次保存字节码之后** 没有修改过源代码，`Python` 将会加载 `.pyc` 文件并跳过编译这个步骤
- 当 `Python` 重编译时，它会自动检查源文件和字节码文件的时间戳
- 如果你又修改了源代码，下次程序运行时，字节码将自动重新创建

提示：有关模块以及模块的其他导入方式，后续课程还会逐渐展开！

模块是 Python 程序架构的一个核心概念

如果使用 `import` 导入了模块（模块一般开发测试完毕，代码很少修改），`Python` 解释器会先把它编译成字节码文件。以后如果没有修改，就会直接加载 `.pyc` 文件，跳过编译的步骤。

158. 高级变量-01-学习目标确定

- 列表
- 元组
- 字典
- 字符串
- 公共方法
- 变量高级

【!】黑马的 `Python` 讲的不错，尤其是关于 `PyCharm` 使用方面。
但是还是有一些基本概念的错误，比如这一节中把字典当做一个序列
以下是更正的笔记内容
另一个问题是，没有讲集合

列表、元组和字符串

1. 都是一个 **序列** `sequence`，也可以理解为 **容器**
2. **取值** `[]`
3. **遍历** `for in`
4. **计算长度、最大/最小值、比较、删除**
5. **链接** `+` 和 **重复** `*`
6. **切片**

159. 列表-01-作用和定义方式

列表的定义

- `List`（列表）是 `Python` 中使用 **最频繁** 的数据类型，在其他语言中通常叫做 **数组**
- 专门用于存储 **一串信息**
- 列表用 `[]` 定义，**数据** 之间使用 `,` 分隔
- 列表的 **索引** 从 `0` 开始
 - **索引** 就是数据在 **列表** 中的位置编号，**索引** 又可以被称为 **下标**

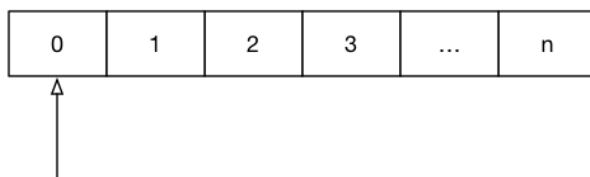
注意：从列表中取值时，如果 **超出索引范围**，程序会报错

```
name_list = ["zhangsan", "lisi", "wangwu"]
```

160. 列表-02-存储数据的方式以及从列表中取值

列表的索引值是从 `0` 开始的

`len(列表)` 获取列表的**长度** `n + 1`
`列表.count(数据)` 数据在列表中出现的**次数**



`列表.sort()` 升序排序
`列表.sort(reverse=True)` 降序排序
`列表.reverse()` 反转/逆序

`列表[索引]` 从列表中取值
`列表.index(数据)` 获得数据第一次出现的索引

`del 列表[索引]` 删除指定索引的数据
`列表.remove[数据]` 删除第一个出现的指定数据
`列表.pop` 删除末尾数据
`列表.pop(索引)` 删除指定索引的数据

`列表.insert(索引, 数据)` 在指定位置插入数据
`列表.append(数据)` 在末尾追加数据
`列表.extend(列表2)` 将列表 2 的数据追加到列表 1

注意，`remove` 方法有误，应该写成 `remove(数据)`

161. 列表-03-确认列表常用操作的方法

列表常用操作

- 在 `ipython3` 中定义一个 **列表**，例如：`name_list = []`
- 输入 `name_list.` 按下 `TAB` 键，`ipython` 会提示 **列表** 能够使用的 **方法** 如下：

```
In [1]: name_list.  
name_list.append      name_list.count      name_list.insert      name_list.revers  
e  
name_list.clear       name_list.extend      name_list.pop          name_list.sort  
name_list.copy        name_list.index       name_list.remove
```

另外一种方法

```
a = []  
dir(a)
```

也可以查看到列表的所有方法（包括魔法方法）

序号	分类	关键字 / 函数 / 方法	说明
1	增加	列表.insert(索引, 数据)	在指定位置插入数据
		列表.append(数据)	在末尾追加数据
		列表.extend(列表2)	将列表2 的数据追加到列表
2	修改	列表[索引] = 数据	修改指定索引的数据
3	删除	del 列表[索引]	删除指定索引的数据
		列表.remove(数据)	删除第一个出现的指定数据
		列表.pop	删除末尾数据
		列表.pop(索引)	删除指定索引数据
		列表.clear	清空列表
4	统计	len(列表)	列表长度
		列表.count(数据)	数据在列表中出现的次数
5	排序	列表.sort()	升序排序
		列表.sort(reverse=True)	降序排序
		列表.reverse()	逆序、反转

162. 列表-04-从列表中取值和取索引

163. 列表-05-修改指定位置的数据

164. 列表-06-向列表增加数据

append

insert 希望插入后该数据在哪个位置，就写哪个位置的 index

extend

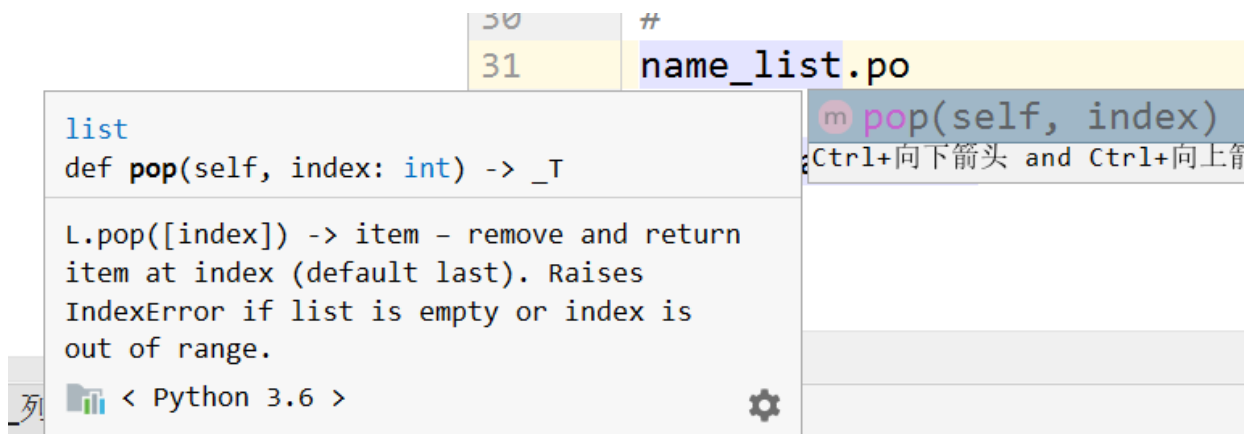
165. 列表-07-从列表删除数据

remove

clear

pop 默认删除列表末尾的元素

注意 Ctrl + Q 相当于起到了帮助的作用。



```
name_list = ['zhangsan', 'lisi', 'wangwu']
```

```
# 1. 取值和索引
```

```
# list index out of range - 列表索引超出范围
```

```
print(name_list[2])
```

```
# 知道数据的内容，想确定数据在列表中的位置
```

```
# 使用 index 方法需要注意，如果传递的数据不在列表中，程序会报错！
```

```
print(name_list.index('wangwu'))
```

```
# 2. 修改
name_list[1] = '李四'
# list assignment index out of range
# 列表指定的索引超出范围，程序会报错！

# 3. 增加
# append 方法可以向列表的末尾追加数据
name_list.append('王小二')
# insert 方法可以在列表的指定索引位置插入数据
name_list.insert(1, '小美眉')
# extend 方法可以把其他列表的完整内容，追加到当前列表的末尾
temp_list = ['孙悟空', '猪二哥', '沙师弟']
name_list.extend(temp_list)

# 4. 删除
# remove 方法可以从列表中删除指定的数据
name_list.remove('wangwu')
# pop 方法默认可以把列表中最后一个元素删除
name_list.pop()
# pop 方法可以指定要删除元素的索引
name_list.pop(3)
# clear 方法可以清空列表
name_list.clear()

print(name_list)
```

166. 列表-08-使用del关键字从列表删除数据

del 关键字 (科普)

- 使用 `del` 关键字(`delete`) 同样可以删除列表中元素
- `del` 关键字本质上是用来 **将一个变量从内存中删除的**
- 如果使用 `del` 关键字将变量从内存中删除，后续的代码就不能再使用这个变量了

```
del name_list[1]
```

在日常开发中，要从列表删除数据，建议 **使用列表提供的方法**

```
name_list = ['张三', '李四', '王五']
```

```
# 使用 del 关键字删除列表元素
```



```
# 提示：在日常开发中，要从列表中删除数据，建议使用列表提供的方法
del name_list[1]

# del 关键字本质上是用来将一个变量从内存中删除的
name = '小明'

del name

# 注意：如果使用 del 关键字将变量从内存中删除
# 后续的代码就不能再使用这个变量了

print(name_list)
```

167. 列表-09-列表统计及删除方法扩展

```
name_list = ['张三', '李四', '王五', '王小二', '张三']

# len 函数可以统计列表中元素的个数
list_len = len(name_list)
print('列表中包含 %d 个元素' % list_len)

# count 方法可以统计列表中某一个数据出现的次数
count = name_list.count('张三')
print('张三出现了 %d 次' % count)

# 从列表中删除第一次出现的数据，如果数据不存在，程序会报错
name_list.remove('张三')

print(name_list)
```


把光标放在方法名， `Ctrl + Q` 叫出帮助

```
# 从列表中删除数据
```

```
name_list.remove('张三')
```

```
list
def remove(self, object: _T) -> None

L.remove(value) -> None - remove first
occurrence of value. Raises ValueError if
the value is not present.
```

 < Python 3.6 >



168. 列表-10-列表的排序和反转

升序

```
name_list.sort()
```


降序

逆序（反转）

```
print(name_list)
print(num_list)
```

```
list
def sort(self,
           *,
           key: Optional[(_T) -> Any],
           reverse: bool) -> None
```

L.sort(key=None, reverse=False) -> None -
stable sort *IN PLACE*

 < Python 3.6 >



```
name_list = ['zhangsan', 'lisi', 'wangwu', 'wangxiaoer']
num_list = [6, 8, 4, 1, 10]
```

升序

```
# name_list.sort()
```

```
# num_list.sort()
```

降序

```
# name_list.sort(reverse=True)
```

```
# num_list.sort(reverse=True)
```

逆序（反转）

```
name_list.reverse()
```

```
num_list.reverse()
```

```
print(name_list)
```

```
print(num_list)
```

169. 列表-11-关键字、函数和方法的特点和区别

关键字、函数和方法

- **关键字** 是 Python 内置的、具有特殊意义的标识符

```
In [1]: import keyword
In [2]: print(keyword.kwlist)
In [3]: print(len(keyword.kwlist))
```

关键字后面不需要使用括号
一共有 33 个关键字

- **函数** 封装了独立功能，可以直接调用

函数名(参数)

函数需要死记硬背

- **方法** 和函数类似，同样是封装了独立的功能
- **方法** 需要通过 **对象** 来调用，表示针对这个 **对象** 要做的操作

对象.方法名(参数)

在变量后面输入 `.`，然后选择针对这个变量要执行的操作，记忆起来比函数要简单很多

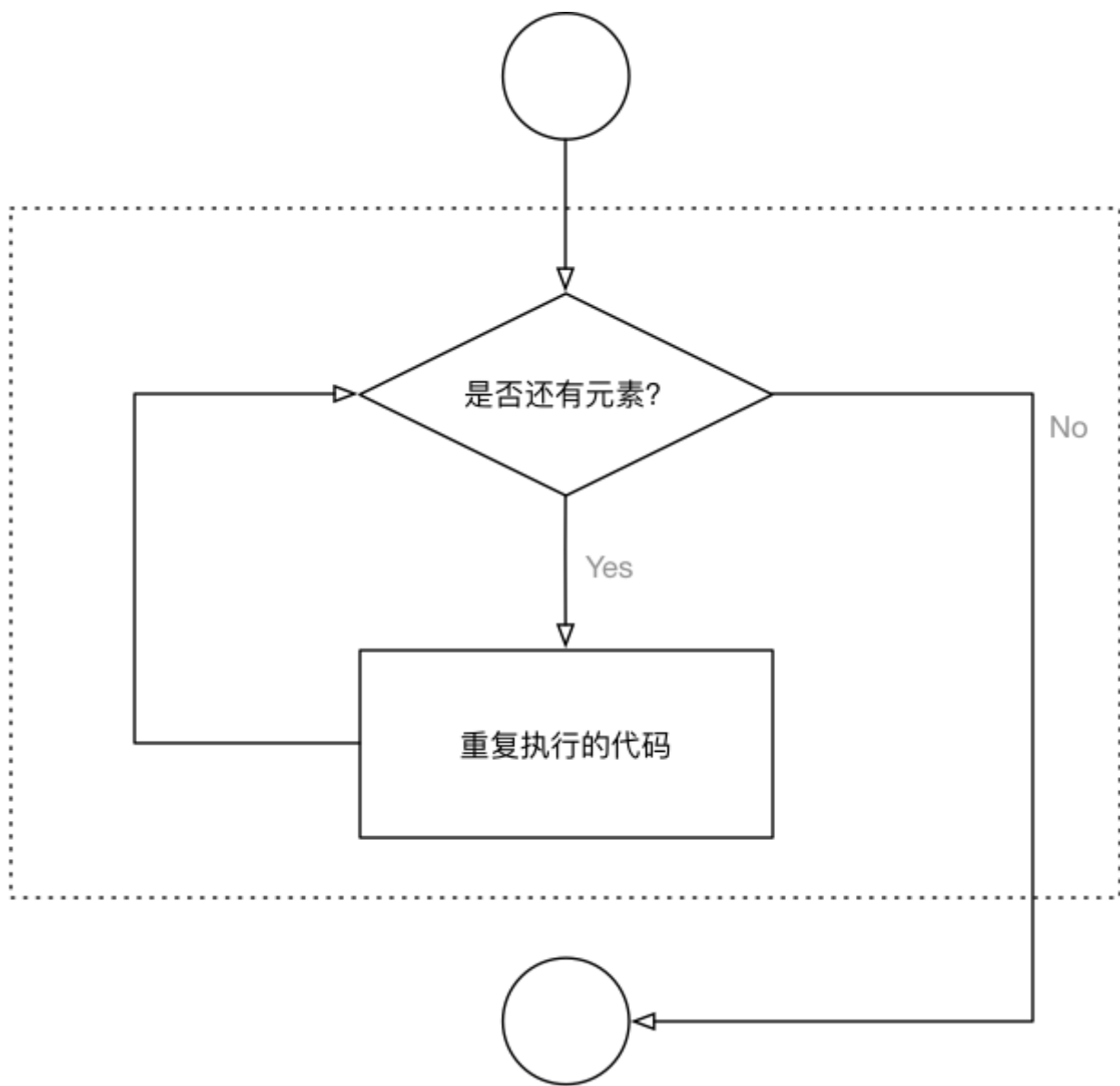
170. 列表-12-迭代遍历

循环遍历

- **遍历** 就是 **从头到尾 依次** 从 **列表** 中获取数据
 - 在 **循环体内部** 针对 **每一个元素**，执行相同的操作
- 在 **Python** 中为了提高列表的遍历效率，专门提供的 **迭代 iteration 遍历**
- 使用 **for** 就能够实现迭代遍历

```
# for 循环内部使用的变量 in 列表
for name in name_list:
```

```
    循环内部针对列表元素进行操作
    print(name)
```



171. 列表-13-列表的应用场景

- 尽管 **Python** 的 **列表** 中可以 **存储不同类型的数据**
- 但是在开发中，更多的应用场景是
 1. **列表** 存储相同类型的数据
 2. 通过 **迭代遍历**，在循环体内部，针对列表中的每一项元素，执行相同的操作

172. 元组-01-特点以及和列表的区别

元组的定义

- **Tuple**（元组）与列表类似，不同之处在于元组的 **元素不能修改**
 - **元组** 表示多个元素组成的序列
 - **元组** 在 **Python** 开发中，有特定的应用场景

- 用于存储 **一串信息**，**数据** 之间使用 **,** 分隔
- 元组用 **()** 定义
- 元组的 **索引** 从 **0** 开始
 - **索引** 就是数据在 **元组** 中的位置编号

173. 元组-02-元组变量的定义

创建元组

```
info_tuple = ("zhangsan", 18, 1.75)
```

创建空元组

```
info_tuple = ()
```

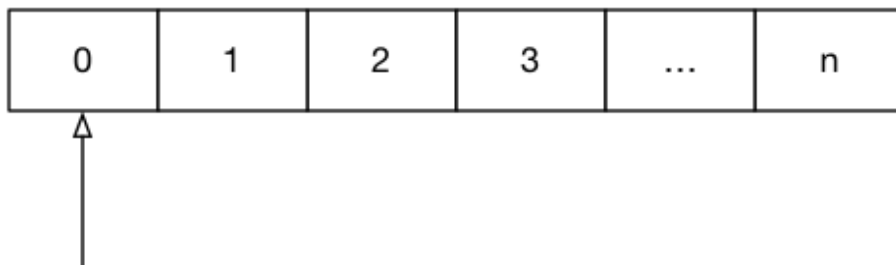
元组中 **只包含一个元素** 时，需要 **在元素后面添加逗号**

```
info_tuple = (50, )
```

元组的索引值是从 **0** 开始的

len(元组) 获取元组的长度 $n + 1$

元组.count(数据) 数据在元组中出现的次数



元组[索引] 从列表中取值

元组.index(数据) 获得数据第一次出现的索引

174. 元组-03-元组变量的常用操作

- 在 `ipython3` 中定义一个 **元组**，例如：`info = ()`
- 输入 `info.` 按下 `TAB` 键，`ipython` 会提示 **元组** 能够使用的函数如下：

```
info.count info.index
```

有关 **元组** 的 **常用操作** 可以参照上图练习

```
info_tuple = ('zhangsan', 18, 1.75, 'zhangsan')
```

```
# 1. 取值和取索引
```

```
print(info_tuple[0])
```

```
# 已经知道数据的内容，希望知道该数据在元组中的索引
```

```
print(info_tuple.index('zhangsan'))
```

```
# 2. 统计计数
```

```
print(info_tuple.count('zhangsan'))
```

```
# 统计元组中包含元素的个数
```

```
print(len(info_tuple))
```

175. 元组-04-元组变量的循环遍历

- **取值** 就是从 **元组** 中获取存储在指定位置的数据
- **遍历** 就是 **从头到尾 依次** 从 **元组** 中获取数据

```
# for 循环内部使用的变量 in 元组
```

```
for item in info:
```

```
    循环内部针对元组元素进行操作
```

```
    print(item)
```

- 在 `Python` 中，可以使用 `for` 循环遍历所有非数字型类型的变量：**列表**、**元组**、**字典** 以及 **字符串**
- 提示：在实际开发中，除非 **能够确认元组中的数据类型**，否则针对元组的循环遍历需求并不是很多

176. 元组-05-元组的应用场景

- 尽管可以使用 `for in` 遍历 元组
- 但是在开发中，更多的应用场景是：
 - 函数的 参数 和 返回值，一个函数可以接收 任意多个参数，或者 一次返回多个数据
 - 有关 函数的参数 和 返回值，在后续 函数高级 给大家介绍
 - 格式字符串，格式化字符串后面的 `()` 本质上就是一个元组
 - 让列表不可以被修改，以保护数据安全

```
info = ("zhangsan", 18)

print("%s 的年龄是 %d" % info)
```

177. 元组-06-元组和格式化字符串

- 格式化字符串后面的是一个元组
- 使用格式化字符串和元组可以拼接新的字符串

```
info_tuple = ('小明', 18, 1.75)

# 格式化字符串后面的 () 本质上就是元组
print('%s 年龄是 %d 身高是 %.2f' % info_tuple)

# 使用格式化字符串和元组来拼接一个新的字符串
info_str = '%s 年龄是 %d 身高是 %.2f' % info_tuple

print(info_str)
```

178. 元组-07-元组和列表之间的转换

- 使用 `list` 函数可以把元组转换成列表

```
list(元组)
```

- 使用 `tuple` 函数可以把列表转换成元组

179. 字典-01-字典的定义格式以及键值对的概念

字典的定义

- `dictionary`（字典）是除列表以外 `Python` 之中最灵活的数据类型
- 字典同样可以用来存储多个数据
 - 通常用于存储描述一个 `物体` 的相关信息
- 和列表的区别
 - **列表** 是 **有序** 的对象集合
 - **字典** 是 **无序** 的对象集合
- 字典用 `{}` 定义
- 字典使用 **键值对** 存储数据，键值对之间使用 `,` 分隔
 - **键** `key` 是索引
 - **值** `value` 是数据
 - **键** 和 **值** 之间使用 `:` 分隔
 - **键必须是唯一的**
 - **值** 可以取任何数据类型，但 **键** 只能使用 **字符串**、**数字**或 **元组**

【!】键必须是不可变的

```
xiaoming = {"name": "小明",  
            "age": 18,  
            "gender": True,  
            "height": 1.75}
```


`len(字典)` 获取字典的 键值对数量

key	value
name	小明
age	18
gender	True
height	1.75

`字典.keys()` 所有 key 列表

`字典.values()` 所有 value 列表

`字典.items()` 所有 (key, value) 元组列表

`字典[key]` 可以从字典中取值, key 不存在会报错

`字典.get(key)` 可以从字典中取值, key 不存在不会报错

`del 字典[key]` 删除指定键值对, key 不存在会报错

`字典.pop(key)` 删除指定键值对, key 不存在会报错

`字典.popitem()` 随机删除一个键值对

`字典.clear()` 清空字典

`字典[key] = value`

如果 key 存在, 修改数据

如果 key 不存, 新建键值对

`字典.setdefault(key, value)`

如果 key 存在, 不会修改数据

如果 key 不存在, 新建键值对

`字典.update(字典2)` 将字典 2 的数据合并到字典 1

180. 字典-02-变量的定义以及应用场景

```
# 字典是一个无序的数据集合
# 使用 print 函数输出字典时
# 通常输出的顺序和定义的顺序是不一致的!
xiaoming = {'name': '小明',
            'age': 18,
            'gender': True,
            'height': 1.75,
            'weight': 75.5}

print(xiaoming)
```

字典里面都是用来描述一个物体 (小明) 的信息。

建议定义字典的时候, 每个键值对占用一行代码。

181. 字典-03-字典的增删改查常用操作

- 在 `ipython3` 中定义一个 **字典**，例如：`xiaoming = {}`
- 输入 `xiaoming.`，按下 `TAB` 键，`ipython` 会提示 **字典** 能够使用的函数如下：

```
In [1]: xiaoming.  
xiaoming.clear          xiaoming.items          xiaoming.setdefault  
xiaoming.copy           xiaoming.keys           xiaoming.update  
xiaoming.fromkeys       xiaoming.pop            xiaoming.values  
xiaoming.get            xiaoming.popitem
```

有关 **字典** 的 **常用操作** 可以参照上图练习

```
xiaoming_dict = {'name': '小明'}  
  
# 1. 取值  
print(xiaoming_dict['name'])  
# 在取值的时候，如果指定的 key 不存在，程序会报错！  
# print(xiaoming_dict['name123'])  
  
# 2. 增加/修改  
# 如果 key 不存在，会新增键值对  
# 如果 key 存在，会修改已经存在的键值对  
xiaoming_dict['age'] = 18  
xiaoming_dict['name'] = '小小明'  
  
# 3. 删除  
# 在删除指定键值对的时候，如果指定的key不存在，程序会报错！  
# xiaoming_dict.pop('name123')  
xiaoming_dict.pop('name')  
  
print(xiaoming_dict)
```

182. 字典-04-字典的统计、合并、清空操作

```
xiaoming_dict = {'name': '小明',  
                 'age': 18}  
  
# 1. 统计键值对数量  
print(len(xiaoming_dict))  
  
# 2. 合并字典
```

```
temp_dict = {'height': 1.75,
             'age': 20}

# 注意：如果被合并的字典中包含已经存在的键值对，会覆盖原有的键值对
xiaoming_dict.update(temp_dict)

# 3. 清空字典
xiaoming_dict.clear()

print(xiaoming_dict)
```

183. 字典-05-字典的循环遍历

- **遍历** 就是 **依次** 从 **字典** 中获取所有键值对

```
# for 循环内部使用的 `key` 的变量 `in` 字典
for k in xiaoming:

    print("%s: %s" % (k, xiaoming[k]))
```

提示：在实际开发中，由于字典中每一个键值对保存数据的类型是不同的，所以针对字典的循环遍历需求并不是很多

```
xiaoming_dict = {'name': '小明',
                 'qq': '123456',
                 'phone': '10086'}

# 迭代遍历字典
# 变量k是每一次循环中，获取到的键值对的key
for k in xiaoming_dict:

    print('%s - %s' % (k, xiaoming_dict[k]))
```

184-字典-06-字典和列表组合的应用场景

- 尽管可以使用 `for in` 遍历 **字典**
- 但是在开发中，更多的应用场景是：

- 使用 **多个键值对**，存储 **描述一个 物体 的相关信息** —— 描述更复杂的数据信息
- 将 **多个字典** 放在 **一个列表** 中，再进行遍历，在循环体内部针对每一个字典进行 **相同的处理**

```
card_list = [  
    {'name': 'zhangsan',  
     'qq': '12345',  
     'phone': '110'},  
    {'name': 'lisi',  
     'qq': '54321',  
     'phone': '10086'}  
]  
  
for card_info in card_list:  
  
    print(card_info)
```

185. 字符串-01-定义和基本使用

字符串的定义

- **字符串** 就是 **一串字符**，是编程语言中表示文本的数据类型
- 在 Python 中可以使用 **一对双引号** `"` 或者 **一对单引号** `'` 定义一个字符串
 - 虽然可以使用 `\"` 或者 `\'` 做字符串的转义，但是在实际开发中：
 - 如果字符串内部需要使用 `"`，可以使用 `'` 定义字符串
 - 如果字符串内部需要使用 `'`，可以使用 `"` 定义字符串
- 可以使用 **索引** 获取一个字符串中 **指定位置的字符**，索引计数从 **0** 开始
- 也可以使用 **for 循环遍历** 字符串中每一个字符

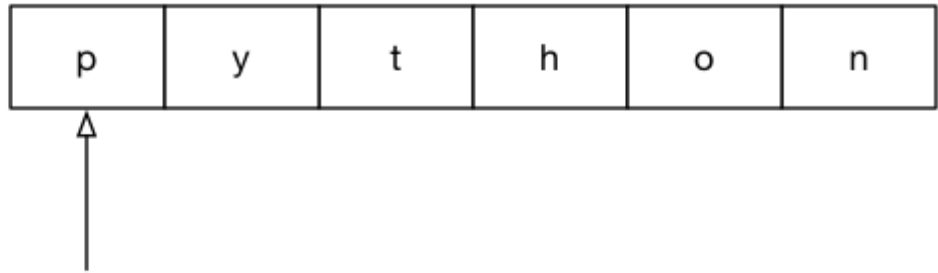
大多数编程语言都是用 `"` 来定义字符串

【!】 其实大多数 **Python** 程序员喜欢用单引号！

字符串的索引值是从 **0** 开始的

`len(字符串)` 获取字符串的**长度**

`字符串.count(字符串)` 小字符串在**大**字符串中出现的次数



`字符串[索引]` 从字符串中取出单个字符

`字符串.index(字符串)` 获得**小字符串**第一次出现的索引

`[]` 像是一个小格子，里面放着东西（数据）

```
str1 = "hello python"
str2 = '我的外号是"大西瓜"'

print(str2)
print(str1[6])

for char in str2:
    print(char)
```

186. 字符串-02-长度、计数、位置方法 演练

```
hello_str = 'hello hello'
```

```
# 1. 统计字符串长度
print(len(hello_str))

# 2. 统计某一个小（子）字符串出现的次数
print(hello_str.count('llo'))
print(hello_str.count('abc'))

# 3. 某一个子字符串出现的位置
print(hello_str.index('llo'))
# 注意：如果使用 index 方法传递的子字符串不存在，程序会报错！
# print(hello_str.index('abc'))
```

187. 字符串-03-常用方法总览和分类

字符串的常用操作

- 在 `ipython3` 中定义一个 **字符串**，例如：`hello_str = ""`
- 输入 `hello_str.`，按下 `TAB` 键，`ipython` 会提示 **字符串** 能够使用的 **方法** 如下：

```
In [1]: hello_str.
hello_str.capitalize    hello_str.isidentifier  hello_str.rindex
hello_str.casefold       hello_str.islower       hello_str.rjust
hello_str.center         hello_str.isnumeric     hello_str.rpartition
hello_str.count          hello_str.isprintable   hello_str.rsplit
hello_str.encode         hello_str.isspace       hello_str.rstrip
hello_str.endswith       hello_str.istitle       hello_str.split
hello_str.expandtabs     hello_str.isupper       hello_str.splitlines
hello_str.find           hello_str.join          hello_str.startswith
hello_str.format         hello_str.ljust         hello_str.strip
hello_str.format_map     hello_str.lower         hello_str.swapcase
hello_str.index          hello_str.lstrip        hello_str.title
hello_str.isalnum        hello_str.maketrans     hello_str.translate
hello_str.isalpha        hello_str.partition     hello_str.upper
hello_str.isdecimal      hello_str.replace       hello_str.zfill
hello_str.isdigit        hello_str.rfind
```

提示：正是因为 `python` 内置提供的方法足够多，才使得在开发时，能够针对字符串进行更加灵活的操作！应对更多的开发需求！

1) 判断类型 - 9

方法	说明
string.isspace()	如果 string 中只包含 空白 ，则返回 True
string.isalnum()	如果 string 至少有一个字符并且所有字符都是字母或数字则返回 True
string.isalpha()	如果 string 至少有一个字符并且所有字符都是字母则返回 True
string.isdecimal()	如果 string 只包含数字则返回 True， 全角数字
string.isdigit()	如果 string 只包含数字则返回 True， 全角数字 、 (1) 、 \u00b2
string.isnumeric()	如果 string 只包含数字则返回 True， 全角数字 ， 汉字数字
string.istitle()	如果 string 是标题化的(每个单词的首字母大写)则返回 True
string.islower()	如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True
string.isupper()	如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True

2) 查找和替换 - 7

方法	说明
string.startswith(str)	检查字符串是否是以 str 开头，是则返回 True
string.endswith(str)	检查字符串是否是以 str 结束，是则返回 True
string.find(str, start=0, end=len(string))	检测 str 是否包含在 string 中，如果 start 和 end 指定范围，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回 -1
string.rfind(str, start=0, end=len(string))	类似于 find()，不过是从右边开始查找
string.index(str, start=0, end=len(string))	跟 find() 方法类似，不过如果 str 不在 string 会报错
string.rindex(str, start=0, end=len(string))	类似于 index()，不过是从右边开始
string.replace(old_str, new_str, num=string.count(old))	把 string 中的 old_str 替换成 new_str，如果 num 指定，则替换不超过 num 次

3) 大小写转换 - 5

方法	说明
string.capitalize()	把字符串的第一个字符大写
string.title()	把字符串的每个单词首字母大写
string.lower()	转换 string 中所有大写字符为小写
string.upper()	转换 string 中的小写字母为大写
string.swapcase()	翻转 string 中的大小写

4) 文本对齐 - 3

方法	说明
string.ljust(width)	返回一个原字符串左对齐，并使用空格填充至长度 width 的新字符串
string.rjust(width)	返回一个原字符串右对齐，并使用空格填充至长度 width 的新字符串
string.center(width)	返回一个原字符串居中，并使用空格填充至长度 width 的新字符串

5) 去除空白字符 - 3

方法	说明
string.lstrip()	截掉 string 左边（开始）的空白字符
string.rstrip()	截掉 string 右边（末尾）的空白字符
string.strip()	截掉 string 左右两边的空白字符

6) 拆分和连接 - 5

方法	说明
string.partition(str)	把字符串 string 分成一个 3 元素的元组 (str前面, str, str后面)
string.rpartition(str)	类似于 partition() 方法，不过是从右边开始查找
string.split(str="", num)	以 str 为分隔符拆分 string，如果 num 有指定值，则仅分隔 num + 1 个子字符串，str 默认包含 '\r', '\t', '\n' 和空格
string.splitlines()	按照行 ('\r', '\n', '\r\n') 分隔，返回一个包含各行作为元素的列表

string.join(seq)

以 string 作为分隔符，将 seq 中所有的元素（的字符串表示）合并为一个新的字符串

188. 字符串-04-判断空白字符以及学习方法分享

isspace

```
# 1. 判断空白字符
space_str = ' \r\n\t'

print(space_str.isspace())
```

189. 字符串-05-判断数字的三个方法

isdecimal

isdigit

isnumeric

可以识别的范围越来越大，但是开发中尽量使用 isdecimal

```
# 1. 判断空白字符
space_str = ' \r\n\t'

print(space_str.isspace())

# 2. 判断字符串中是否只包含数字
# 1> 都不能判断小数
# num_str = '1.1'

# 2> unicode 字符串
# num_str = '()'
# num_str = '\u00b2'

# 3> 中文数字
num_str = '一千零一'

print(num_str)
print(num_str.isdecimal())
print(num_str.isdigit())
print(num_str.isnumeric())
```

完成于 201809291915