高级变量类型 第一节

非数字变量(字符串、列表、元组、字典)都支持以下特点：
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、连接 + 和重复 *
6、切片

一、列表
1、列表的定义：
1.1 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**
1.2 专门用于存储 **一串信息**
1.3 专门用 **[ ] 定义**，数据之间使用**，**分隔
1.4 列表的索引从 **0** 开始
1.5 下标范围还有其他的跟Java类似，都有越界之类的问题
1.6 列表中竟然可以存储不同类型的数据类型，不过不常用而已
1.6 列表常用的方法（定义：name = ["zhangsan","lisi","wangwu"] ）

序号	分类	关键字 / 函数 / 方法	说明
1	增加	name.insert(索引，数据)	在指定位置插入数据
		name.append(数据)	在列表末尾增加数据
		name.extend(列表2)	将列表2 的数据全部追加到name中
2	修改	name[索引] = 数据	修改列表中 该索引 的数据
3	删除	del name[索引]	删除指定索引的数据，本质是用来从内存中删除
		name.remove[数据]	删除第一个出现的指定数据
		name.pop()	删除末尾数据，会返回删除的数据
		name.pop(索引)	删除指定索引的数据，会返回删除的数据
		name.clear	清空列表
4	统计	len(name)	列表长度

		name.count(数据)	统计数据在列表中出现的次数
5	排序	name.sort()	升序排序
		name.sort(reverse=True)	降序排序
		name.reverse()	是将原本列表的数据逆序（翻转）

2、迭代遍历

```
name_list = ["张三","李四","王五","王小二"]
# for 临时定义的变量(名字随便起) in name_list(之前定义好的列表):
# 跟Java 中的foreach 类似
for my_name in name_list:
 print("我的名字是 %s" % my_name)
```

二、元组

1、Tuple (元组)与列表类似，不同点

元组的元素不能修改；列表可以修改

元组用（）**定义**，仅仅只是定义；列表用[ ] 定义；但是元组中，想要获取某个下标的数据值，也还是用[ ]

元组存在的目的：元组 专门存储不同类型的数据

2、定义

2.1 定义一个空元组：

```
>>> empty_tuple = ()
>>> type(empty_tuple)
<class 'tuple'>
```

2.2 定义一个只包含一个元素的元组

注意注意：这是错误的

```
>>> single_tuple = (5)
>>> type(single_tuple)
<class 'int'>
```

正确示范：

```
>>> single_tuple = (5,)
>>> type(single_tuple)
<class 'tuple'>
```

3、元组的方法使用

```
info_tuple = ("张三",18,1.75,"张三")

# 1、取值和取索引
print(info_tuple[0])
```

```
# 已经知道数组的内容，希望知道该数据在元组中的索引
print(info_tuple.index("张三"))

# 2、统计次数
print(info_tuple.count("张三"))
 # 2、统计元组中包含元素的个数
 print(len(info_tuple))

 print(info_tuple)
```

4、元组的循环遍历：不常使用

```
info_tuple = ("张三",18,1.75)

for my_info in info_tuple:

 # 使用格式字符串拼接 my_info 这个变量不方便
 # 因为元组中通常保存的数据类型是不同的！
 # 所以元组不常用来保存不同类型的数据类型
 print(my_info)
```

5、元组的使用场景

1、函数的 参数 和 返回值，一个函数可以接收多个任意参数，或者一次返回多个数据

2、格式化字符串，格式化字符串后面的（）本质上就是一个元组 / print("我的个人信息有%s，%d，%f" %(info_tuple[0],info_tuple[1],info_tuple[2]))

```
info_tuple = ("wpp",18,165)

# 格式化字符串后面的（）本质上就是一个元组
print("%s 的年龄是 %d 身高是 %.2f" % info_tuple)
```

3、让列表不可以被修改，以保护数据安全

三、字典：

通常用于存储 **描述一个 物体 的相关信息**

1、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 // 在使用 print 输出一个字典时，通常输出的顺序和定义的顺序是不一致的

2、字典是用 **{ }** 定义

3、字典使用 **键值对** 存储数据，键和值之间用 **：** 分隔，键值对之间用 **，** 分隔

4、键必须是**唯一**的，值可以取任何数据类型， 但 **键** 只能用 **字符串**、**数字**和**元组**

5、字典的常用操作

```python
xiaoming_dict = {"name":"小明"}

# 1、取值
print(xiaoming_dict["name"])
# print(xiaoming_dict["123name"])

# 2、增加 / 修改
# 如果 key 不存在，会增加键值对
xiaoming_dict["age"] = 18
 # 如果 key 存在，则修改已存在的键值对
 xiaoming_dict["name"] = "小明明"

 # 3、删除
 xiaoming_dict.pop("name")

 print(xiaoming_dict)
```

```python
xiaoming_dict = {"name":"小明",
 "age":18}

# 1、统计键值对数量
print(len(xiaoming_dict))

# 2、合并键值对
# 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
tmp_dict = {"height":1.75,
  "age":20}
 xiaoming_dict.update(tmp_dict)

 # 3、清空字典
 xiaoming_dict.clear()

 print(xiaoming_dict)
```

6、循环遍历：

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

# 迭代遍历字典
# 变量 k 是每一次循环中，获得到的键值对的 key
```

```
for k in xiaoming_dict:
 print("%s - %s" % (k , xiaoming_dict[k]))
```

7、应用场景

```
# 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

card_list = [
 {"name":"张三","qq":"123456","phone":"110"},
 {"name":"李四","qq":"654321","phone":"119"}
]

for card_info in card_list:
  print(card_info)
```

四、字符串

1、字符串基本操作：查找长度、查找某个子字符串出现的次数、查找某个字符串出现的位置

```
hello_str = "hello hello"

# 1、统计字符串长度
print(len(hello_str))

# 2、统计某一个子字符串在长字符串中出现的次数
print(hello_str.count("llo"))
print(hello_str.count("ii"))

 # 2、某一个字符串出现的位置
 print(hello_str.index("lo"))
 print(hello_str.index("io")) # 报错！！！
```

2、字符串常用到的方法

1）字符串统计：

```
hello_str = "hello hello"

# 1、统计字符串长度
print(len(hello_str))

# 2、统计某一个子字符串在长字符串中出现的次数
print(hello_str.count("llo"))
```

```
print(hello_str.count("ii"))

# 2、某一个字符串出现的位置
print(hello_str.index("lo"))
# print(hello_str.index("io")) # 报错！！！
```

2）字符串判断：

```
# 判断空格（包括空格、\r \n \t）
# 1、判断空格字符
space_str = " \n\t\r"

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
# 2> unicode 字符串
# 3> 中文数字
# num_str = "1"
# num_str = "\u00b2"
num_str = "一千零一"

print(num_str)
print(num_str.isdecimal()) # 只包含全角数字
print(num_str.isdigit()) # 包含全角、(1)、
print(num_str.isnumeric()) # 还有判断中文数字
```

3）字符串的查找和替代：

```
hello_str = "hello world"

# 1、判断是否以指定字符串开始
print(hello_str.startswith("hello"))
print(hello_str.startswith("Hello"))
print("==============================")

# 2、判断是否以指定字符串结束
print(hello_str.endswith("world"))
print("==============================")

# 3、查找指定字符串
# 返回值表示子字符串出现的位置 类同于 index() 方法
# 区别：若不存在子字符串，find()方法返回-1,index() 方法直接报错
# string.find(str,start=0,end=len(string))
```

```
print(hello_str.find("llo")) # 返回值表示子字符串出现的位置 类同于 index()方法
print(hello_str.find("abc"))
print("============================")

# 4、替换字符串
# 执行完成后，不会修改原有字符串的内容，只会返回一个新的字符串
print(hello_str.replace("world","2020"))

print(hello_str)
```

4）字符串切片：

```
>>> num_str = "0123456789"
>>> num_str[2:6]
'2345'
>>> num_str[2]
'2'
>>> num_str[2:]
'23456789'
>>> num[0:6]
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    num[0:6]
NameError: name 'num' is not defined
>>> num_str[0:6]
'012345'
>>> num_str[:6]
'012345'
>>> num[:]
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    num[:]
NameError: name 'num' is not defined
>>> num_str[:]
'0123456789'
>>> num_str[::2]
'02468'
>>> num_str[1::2]
'13579'
>>> num_str[1:-1]
'12345678'
>>> num_str[-1]
'9'
>>> num_str[2:-1]
'2345678'
>>> num_str[-2:]
'89'
>>> num_str[0::-1]
'0'
>>> num_str[-1::-1]
'9876543210'
```