

# 列表

## 目标

- 列表的应用场景
- 列表的格式
- 列表的常用操作
- 列表的循环遍历
- 列表的嵌套使用

## 1、列表的应用场景

思考：有一个人的姓名(正心)怎么书写存储程序？

答：变量。

思考：如果一个班级100位学生，每个人的姓名都要存储，应该如何书写程序？声明100个变量吗？

答：列表即可，列表一次性可以存储多个数据。

## 2、列表的格式

- [数据1, 数据2, 数据3, 数据4.....]

```
# []（中括号）代表列表
# <class 'list'> list 是列表类型
# 列表是一个数据容器，我们可以在里面放置数据(比如我们前面学到的<数字/小数/字符串/布尔类型>)
list1 = [1, 2, 3, 'a', 'b', 'c', True, 3 > 4, print]

print(list1)
print(type(list1))
```

输出结果：

```
[1, 2, 3, 'a', 'b', 'c', True, False, <built-in function print>]
<class 'list'>
```

列表可以一次性存储多个数据，且可以为不同数据类型。

## 3、列表的常用操作

列表的作用是一次性存储多个数据，程序员可以对这些数据进行的操作有：增、删、改、查。

### 3.1 查找

### 3.1.1 索引/下标取值

- 支持正序取值或者逆序取值，超出索引范围会报错
  - 正序：索引从 0 开始
  - 逆序：索引从 -1 开始

```
"""列表是有序的数据容器"""
name_list = ['正心', '丸子', '自游', '木子']

# 正序取值：索引从 0 开始
print(name_list[0])
print(name_list[3])
# print(name_list[6]) # 取值超出索引范围会报错：IndexError: list index out of range

# 逆序取值：索引从 -1 开始
print(name_list[-1])
print(name_list[-4])
```

### 3.1.2 切片取值

Python中符合序列的有序序列都支持切片（slice），切片方法适用于字符串、列表、元组。

格式：

[起始值:结束值:步长]

- start: 起始索引，从0开始，-1表示结束
- stop: 结束索引
- step: 步长，end-start，步长为正时，从左向右取值。步长为负时，反向取值

```
array1 = list(range(10))

# 指定区间切片
print(array1[2:5])
print(array1[2:-2])

# 从头开始切片
print(array1[0:5])

# 切片到末尾
print(array1[0:])

# 省略参数切全部内容
print(array1[:])

# 指定步长切片
print(array1[0:5:1])
print(array1[0:5:2])
```

### 3.1.2 列表查找/统计的方法

- index(): 返回指定数据所在位置的索引。
  - 语法

```
列表序列.index(数据)
```

- 快速体验

```
name_list = ['Tom', 'Lily', 'Rose']

print(name_list.index('Lily', 0, 2)) # 1
```

注意：如果查找的数据不存在则报错

- count(): 统计指定数据在当前列表中出现的次数。

```
name_list = ['Tom', 'Lily', 'Rose']

print(name_list.count('Lily')) # 1
```

- len(): 访问列表长度，即列表中数据的个数。

```
name_list = ['Tom', 'Lily', 'Rose']

print(len(name_list)) # 3
```

### 3.1.3 判断是否存在

- in: 判断指定数据在某个列表序列，如果在返回True，否则返回False

```
name_list = ['Tom', 'Lily', 'Rose']

# 结果: True
print('Lily' in name_list)

# 结果: False
print('Lilys' in name_list)
```

- not in: 判断指定数据不在某个列表序列，如果不在返回True，否则返回False

```
name_list = ['Tom', 'Lily', 'Rose']

# 结果: False
print('Lily' not in name_list)

# 结果: True
print('Lilys' not in name_list)
```

- 体验案例

需求：查找用户输入的名字是否已经存在。

```
name_list = ['正心', '丸子', '自游', '木子']

name = input('请输入您要搜索的名字: ')

if name in name_list:
    print(f'您输入的名字是{name}, 名字已经存在')
else:
    print(f'您输入的名字是{name}, 名字不存在')
```

### 3.2 增加

作用：增加指定数据到列表中。

- `append()`: 列表结尾追加数据。
  - 语法

列表序列.append(数据)

- ### ○ 体验

```
name_list = ['正心', '丸子', '自游', '木子']

name_list.append('清风')

# 结果: ['正心', '丸子', '自游', '木子', '清风']
print(name_list)
```

```
D:\Anaconda3\python.exe "Z:/01 青灯教育课程/01 青灯教育-vip课程/01  
['正心', '丸子', '自游', '木子', '清风']  
  
Process finished with exit code 0
```

列表追加数据的时候，直接在原列表里面追加了指定数据，即修改了原列表，故列表为可变类型数据。

### 3. 注意点

如果append()追加的数据是一个序列，则追加整个序列到列表

```
name_list2 = ['正心', '丸子', '自游', '木子']
name_list2.append(['清风', '丸子'])
print(name_list2)

# 结果: ['正心', '丸子', '自游', '木子', ['清风', '丸子']]
print(name_list2)
```

- `extend()`: 列表结尾追加数据, 如果数据是一个序列, 则将这个序列的数据逐一添加到列表。

## 1. 语法

列表序列.extend(数据)

## 2. 快速体验

### 2.1 单个数据

```
name_list = ['Tom', 'Lily', 'Rose']

name_list.extend('xiaoming')

# 结果: ['Tom', 'Lily', 'Rose', 'x', 'i', 'a', 'o', 'm', 'i', 'n', 'g']
print(name_list)
```

### 2.2 序列数据

```
name_list2 = ['正心', '丸子', '自游', '木子']
name_list3 = ['清风', '丸子']

name_list2.extend(name_list3) # 把 name_list3 列表数据合并到 name_list2 尾部
print(name_list2)

# 结果: ['正心', '丸子', '自游', '木子', '清风', '丸子']
```

- insert(): 指定位置新增数据。

- 语法

列表序列.insert(位置下标, 数据)

- 快速体验

```
name_list4 = ['正心', '丸子', '自游', '木子']

name_list4.insert(2, '清风') # 在索引为 2 的位置插入数据
print(name_list4)

# 结果: ['正心', '丸子', '清风', '自游', '木子']
```

## 3.3 删除

- pop(): 删除指定下标的数据(默认为最后一个), 并返回该数据。

- 语法

列表序列.pop(下标)

- remove(): 移除列表中某个数据的第一个匹配项。

- 语法

列表序列.remove(数据)

- 快速体验

```
list1 = ['a', 'b', 'c', 'd', 'e', 'f']

# pop()      默认删除最后一个元素,可以指定索引
# remove()   指定元素进行删除
list1.pop()
list1.pop(0)

list1.remove('e')
```

## 3.4 修改

- 修改指定下标数据

```
name_list = ['Tom', 'Lily', 'Rose']

name_list[0] = 'aaa'

# 结果: ['aaa', 'Lily', 'Rose']
print(name_list)
```

- 逆置: reverse()

```
num_list = [1, 5, 2, 3, 6, 8]

num_list.reverse()

# 结果: [8, 6, 3, 2, 5, 1]
print(num_list)
```

- 排序: sort()
  - 语法

```
列表序列.sort( key=None, reverse=False)
```

注意: reverse表示排序规则, **reverse = True** 降序, **reverse = False** 升序 (默认)

- 快速体验

```
num_list = [1, 5, 2, 3, 6, 8]

num_list.sort()

# 结果: [1, 2, 3, 5, 6, 8]
print(num_list)
```

## 4、列表的循环遍历

需求: 依次打印列表中的各个数据。

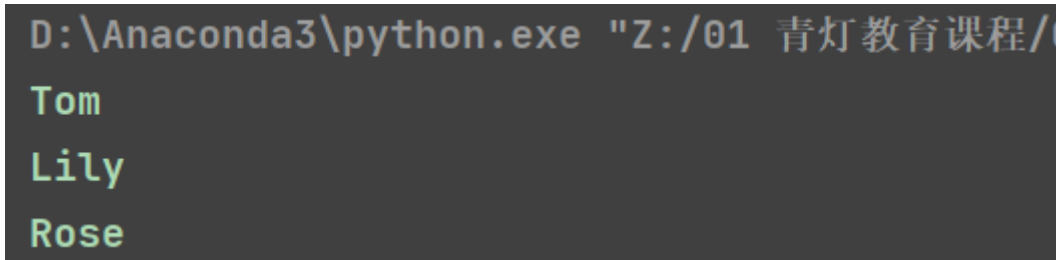
### 4.1 while

- 代码

```
name_list = ['Tom', 'Lily', 'Rose']

i = 0
while i < len(name_list):
    print(name_list[i])
    i += 1
```

- 执行结果



```
D:\Anaconda3\python.exe "Z:/01 青灯教育课程/"
Tom
Lily
Rose
```

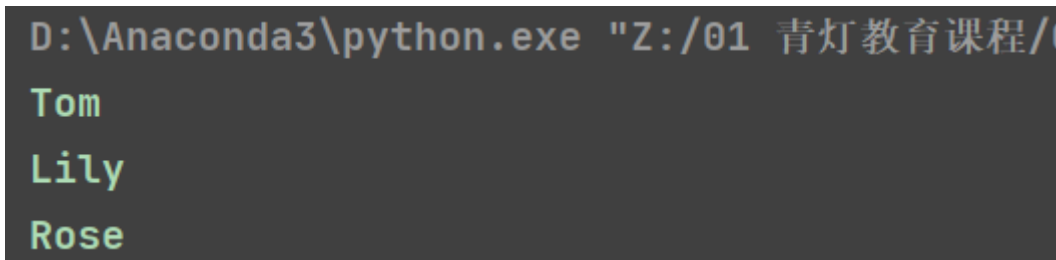
## 4.2 for

- 代码

```
name_list = ['Tom', 'Lily', 'Rose']

for i in name_list:
    print(i)
```

- 执行结果



```
D:\Anaconda3\python.exe "Z:/01 青灯教育课程/"
Tom
Lily
Rose
```

## 5、列表嵌套

所谓列表嵌套指的就是一个列表里面包含了其他的子列表。

应用场景：要存储班级一、二、三三个班级学生姓名，且每个班级的学生姓名在一个列表。

```
name_list = [['小明', '小红', '小绿'], ['Tom', 'Lily', 'Rose'], ['张三', '李四', '王五']]
```

思考：如何查找到数据"李四"？

```
# 第一步：按下标查找到李四所在的列表
print(name_list[2])

# 第二步：从李四所在的列表里面，再按下标找到数据李四
print(name_list[2][1])
```

## 6、综合应用 -- 随机分配办公室

---

需求：有三个办公室，8位老师，8位老师分配到3个办公室

## 7、总结

---

- 列表的格式

```
[数据1, 数据2, 数据3]
```

- 常用操作方法
  - index()
  - len()
  - append()
  - pop()
  - remove()
- 列表嵌套

```
name_list = [['小明', '小红', '小绿'], ['Tom', 'Lily', 'Rose'], ['张三', '李四', '王五']]  
name_list[2][1]
```