序列

序列

- 序列通用操作
- 列表
- 元组
- 字典
- 集合
- 字符串

序列通用操作

序列的通用操作

- 索引
- 切片
- 序列相加
- 序列乘法
- 序列是否包含元素
- 序列长度、最大值、最小值

序列索引

◆ 通过下标访问内容

```
numbers = [10,11,12,13,14]
print(numbers[0])
print(numbers[-1])
```

```
10
14
```

序列切片

◆ 通过一串下标访问一串内容

```
numbers = [10,11,12,13,14]
print(numbers[1:3])
print(numbers[2:])
print(numbers[:2])
print(numbers[:-2])
print(numbers[0:4:2])
```

```
[11, 12]
[12, 13, 14]
[10, 11]
[10, 11, 12]
[10, 12]
```

序列相加

◆ 两个序列前后拼在一起

```
numbers = [1,2,3,4,5]
data = ["a", "b", 3, 4.0, 5]
result = numbers + data
print(result)
```

```
[1, 2, 3, 4, 5, 'a', 'b', 3, 4.0, 5]
```

序列乘法

◆ 序列中的内容重复出现多次

```
numbers = [1, 2, 3]
result = numbers * 3
print(result)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

序列是否包含元素

◆ 查看元素是否在序列中

```
numbers = [1,2,3]
if 1 in numbers:
    print("1 在 numbers 里面")
else:
    print("1 不在 numbers 里面")
```

```
1 在 numbers 里面
```

序列长度、最大值、最小值

◆ 对序列做简单的统计

```
numbers = [1,2,3, 4, 5]
print(len(numbers))
print(max(numbers))
print(min(numbers))
```

```
5
5
1
```

列表

创建列表

- 列表是最常用的序列

```
list_empty = []
list_a = [1,2,3]
list_b = list(range(10))

print(list_empty)
print(list_a)
print(list_b)
```

```
[]
[1, 2, 3]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

访问列表元素

◆ 列表属于序列，可以用索引访问

```
list_a = [1,2,3]
print(list_a[1])
```

```
2
```

遍历列表

◆ 对列表遍历的两种方式

```
data_list = ['a', 'b', 'c', 'd', 'e']
for data_i in data_list:
    print(data_i)
```

```
a
b
c
d
e
```

```
data_list = ['a', 'b', 'c', 'd', 'e']
for index, data_i in enumerate(data_list):
    print(index, data_i)
```

```
0 a
1 b
2 c
3 d
4 e
```

添加、修改、删除列表元素

◆ 列表可以进行增删改查

```
1 list_a = [1,2,3,4,5]
2 print(list_a)
3 list_a.append(6)
4 print(list_a)
5 list_a[0] = 0
6 print(list_a)
7 list_a.remove(4)
8 print(list_a)
```

```
1 [1, 2, 3, 4, 5]
2 [1, 2, 3, 4, 5, 6]
3 [0, 2, 3, 4, 5, 6]
4 [0, 2, 3, 5, 6]
```

列表元素统计计算

◆ 列表的简单统计

```
1 list_a = [1,2,3,4,5]
2 result = sum(list_a)
3 print(result)
```

```
1 15
```

列表排序

◆ 对列表进行排序

```
score = [50,60,20,40,30,80,90,55,100]
print("原列表：",score)
score.sort()
print("升序后：",score)
score.sort(reverse=True)
print("降序后：",score)
```

```
原列表： [50, 60, 20, 40, 30, 80, 90, 55, 100]
升序后： [20, 30, 40, 50, 55, 60, 80, 90, 100]
降序后： [100, 90, 80, 60, 55, 50, 40, 30, 20]
```

列表推导式

◆ 可以快速创建一个有序列表

```
x_list = [i for i in range(10)]
print(x_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

列表案例

- 关于列表的小案例

```python
# 初始的模型准确率
accuracies = [0.85, 0.90, 0.88, 0.92]

# 添加新的准确率
accuracies.append(0.95)

# 计算平均准确率
average_accuracy = sum(accuracies) / len(accuracies)
print(f"Average Accuracy: {average_accuracy:.2f}")
```

```
Average Accuracy: 0.90
```

元组

创建元组

◆ 列表是方括号，元组是小括号

```
tuple_1 = ()
tuple_2 = tuple(range(10, 20, 2))
print(tuple_1)
print(tuple_2)
```

```
()
(10, 12, 14, 16, 18)
```

访问元组元素

◆ 元组的数据可以索引和切片

```
tuple_1 = (1, 2, 3, 4, 5)
print(tuple_1[2])
print(tuple_1[-1])
```

```
3
5
```

元组推导式

- 像创建列表那样创建元组

```
1 tuple_a = tuple(i for i in range(10))
2 print(tuple_a)
```

```
1 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

列表和元组的差异

特性	列表	元组
定义	使用方括号 [] 或 list()	使用圆括号 () 或 tuple()
可变性	可变，可以修改内容	不可变，一旦创建不能修改
方法	拥有多种方法，如 append 等	方法较少
性能	相对较慢	相对较快
适用场景	需要经常修改内容	不需要修改内容
占用空间	相对较多	相对较少

元组案例

◆ 关于元组的小案例

```python
# 模型的配置（层数，每层的单元数，激活函数）
model_config = (3, 128, "relu")

# 解包元组
layers, units, activation = model_config
print(f"Layers: {layers}, Units: {units}, Activation: {activation}")

```

```
Layers: 3, Units: 128, Activation: relu
```

字典

创建字典

- 字典存放的信息以键值对形式出现

```python
info_xiaoming = {'name':'小明',
                 'age':14,
                 'score':60}

info_zhangsan = {'name':'张三',
                 'age':15,
                 'score':79}
print(info_xiaoming)
print(info_xiaoming['age'])
print(info_zhangsan['score'])
```

```
{'name': '小明', 'age': 14, 'score': 60}
14
79
```

创建字典

- 字典存放的信息以键值对形式出现

```python
# 创建一个字典来存储神经网络的配置参数
neural_network_config = {
    "layer_1": {"units": 64, "activation": "relu"},
    "layer_2": {"units": 128, "activation": "relu"},
    "output_layer": {"units": 10, "activation": "softmax"}
}
print(neural_network_config)
```

```
{'layer_1': {'units': 64, 'activation': 'relu'}, 'layer_2': {'units': 128, 'activation': 'relu'},
'output_layer': {'units': 10, 'activation': 'softmax'}}
```

通过键值对访问字典

◆ 可以通过键访问到值

```
# 创建一个字典来存储神经网络的配置参数
neural_network_config = {
    "layer_1": {"units": 64, "activation": "relu"},
    "layer_2": {"units": 128, "activation": "relu"},
    "output_layer": {"units": 10, "activation": "softmax"}
}
# 访问字典中的特定键值对
layer_1_units = neural_network_config["layer_1"]["units"]
print(f"Number of units in layer 1: {layer_1_units}")

```

```
Number of units in layer 1: 64
```

遍历字典

◆ 对字典遍历的方式

```
info_xiaoming = {'name':'小明',
                 'age':14,
                 'score':60}
# 遍历字典
print("以下为 xiaoming 的信息：")
for key, value in info_xiaoming.items():
    print(f"{key} 为 {value}")
```

```
以下为 xiaoming 的信息：
name 为 小明
age 为 14
score 为 60
```

遍历字典

◆ 对字典遍历的方式

```
neural_network_config = {
    "layer_1": {"units": 64, "activation": "relu"},
    "layer_2": {"units": 128, "activation": "relu"},
    "output_layer": {"units": 10, "activation": "softmax"}
}

# 遍历字典，打印每一层的配置信息
for layer, config in neural_network_config.items():
    print(f"{layer}: {config['units']} units, activation = {config['activation']}")
```

```
layer_1: 64 units, activation = relu
layer_2: 128 units, activation = relu
output_layer: 10 units, activation = softmax
```

添加、修改和删除字典元素

◆ 可以对字典元素进行增删改查

```
neural_network_config = {
    "layer_1": {"units": 64, "activation": "relu"},
    "layer_2": {"units": 128, "activation": "relu"},
    "output_layer": {"units": 10, "activation": "softmax"}
}

# 添加一个新的层到字典
neural_network_config["layer_3"] = {"units": 256, "activation": "relu"}

# 修改第一层的单元数
neural_network_config["layer_1"]["units"] = 128

# 删除输出层的激活函数键值对
del neural_network_config["output_layer"]["activation"]

# 输出修改后的字典
print(neural_network_config)
```

```
{'layer_1': {'units': 128, 'activation': 'relu'}, 'layer_2': {'units': 128, 'activation': 'relu'}, 'output_layer': {'units': 10}, 'layer_3': {'units': 256, 'activation': 'relu'}}
```

字典案例

◆ 字典的小案例

```python
# 不同模型的信息
models_info = {
    "CNN": {"layers": 3, "units": 128, "activation": "relu"},
    "RNN": {"layers": 2, "units": 64, "activation": "tanh"}
}

# 访问特定模型的信息
cnn_info = models_info["CNN"]
print(f"CNN - Layers: {cnn_info['layers']}, Units: {cnn_info['units']},
Activation: {cnn_info['activation']}")
```

```
CNN - Layers: 3, Units: 128, Activation: relu
```

集合

创建集合

- 集合的标值是大括号，具有去重复作用

```python
set_1 = set()
set_2 = {}
set_3 = {1,2,3,3, 4,5}

print(set_1)
print(set_2)
print(set_3)
```

```
set()
{}
{1, 2, 3, 4, 5}
```

元素添加与删除

- 集合支持增删改查，同时里面的元素会自动去重复

```python
# 初始化一个空集合
my_set = set()

# 添加元素
my_set.add(1)  # {1}
my_set.add(2)  # {1, 2}
my_set.add(3)  # {1, 2, 3}

# 删除元素
my_set.remove(2)  # {1, 3}

print(my_set)
```

```
{1, 3}
```

集合的交集、并集与差集

◆ 使用集合完成数学当中的集合运算

```python
# 定义两个集合
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

# 交集运算
intersection = set1.intersection(set2)
# 或者
# intersection = set1 & set2
print(f"交集: {intersection}")

# 并集运算
union = set1.union(set2)
# 或者
# union = set1 | set2
print(f"并集: {union}")

# 差集运算
difference1 = set1.difference(set2)
# 或者
# difference1 = set1 - set2
print(f"set1 和 set2 的差集: {difference1}")

difference2 = set2.difference(set1)
# 或者
# difference2 = set2 - set1
print(f"set2 和 set1 的差集: {difference2}")
```

```
交集: {3, 4}
并集: {1, 2, 3, 4, 5, 6}
set1 和 set2 的差集: {1, 2}
set2 和 set1 的差集: {5, 6}
```

集合案例

◆ 使用集合的小案例

```python
# 两个实验中使用的激活函数
experiment1 = {"relu", "sigmoid", "tanh"}
experiment2 = {"relu", "softmax"}

# 找出两个实验中都使用过的激活函数
common_activations = experiment1.intersection(experiment2)
print(f"Common Activations: {common_activations}")
```

```
Common Activations: {'relu'}
```

字符串

字符串索引

```
# 示例字符串
string = "this_is_a_file.jpg"

# 获取字符串的第2到第5个字符（索引从0开始）
substring = string[1:5]  # 结果: "his_"
print(substring)

# 获取字符串的第2到最后一个字符
substring = string[1:]  # 结果: "his_is_a_file.jpg"
print(substring)

# 获取字符串的开始到第5个字符
substring = string[:5]  # 结果: "this_"
print(substring)

```

```
# 获取整个字符串
substring = string[:]  # 结果: "this_is_a_file.jpg"
print(substring)

# 获取字符串的最后3个字符
substring = string[-3:]  # 结果: "jpg"
print(substring)

# 获取字符串的第2到倒数第3个字符，每隔2个字符取一个
substring = string[1:-2:2]  # 结果: "hsi__iej"
print(substring)

# 反转字符串
substring = string[::-1]  # 结果: "gpj.elif_a_si_siht"
print(substring)
```

字符串比较

```python
# 定义两个字符串
string1 = "Hello"
string2 = "hello"
string3 = "Hello"

# 使用 == 操作符比较字符串
is_equal = string1 == string2  # 结果: False
print(f"string1 is equal to string2: {is_equal}")

is_equal = string1 == string3  # 结果: True
print(f"string1 is equal to string3: {is_equal}")

# 使用 != 操作符比较字符串
is_not_equal = string1 != string2  # 结果: True
print(f"string1 is not equal to string2: {is_not_equal}")

# 使用 <, > 操作符比较字符串（基于字典顺序）
is_less_than = string1 < string2  # 结果: True (因为大写字母在字典顺序中排在小写字母之前
print(f"string1 is less than string2: {is_less_than}")

# 不区分大小写的字符串比较
is_equal_ignore_case = string1.lower() == string2.lower()  # 结果: True
print(f"string1 is equal to string2 (ignore case): {is_equal_ignore_case}")
```