Lesson3 数据结构

一. 数据结构的概念

数据结构（Data Structures）是指存储、组织和管理数据的方式。数据结构通常是由**基本数据类型**组成的。

数据结构和数据类型的关系：

1. 基本数据类型是构建数据结构的基础：例如，一个列表可以包含多个整数、浮点数、字符串等基本数据类型。
2. 数据结构是更复杂的类型：数据结构通常用来组织和管理多种类型的数据，更适合处理复杂的数据操作和存储需求。
3. 操作和用途不同：基本数据类型通常用于存储单一值，而数据结构用于存储和操作多值的集合。

常见的数据结构包括：
--**列表（List）**：有序、可变的元素集合，例如，[1, 2, 3]
--**元组（Tuple）**：有序、不可变的元素集合，例如，(1, 2, 3)
--**集合（Set）**：无序、元素唯一的集合，例如，{1, 2, 3}
--**字典（Dictionary）**：键值对的集合，例如，{"name": "Alice", "age": 25}

二. 列表，元组，集合，字典

--列表：有序、可变的元素集合

列表的基本语法:
my_list = [element1, element2, element3]

In [5]:
```python
# 创建列表
students = ["Jesse", "Andrew"]
print(students)

# 访问列表元素
print(students[0])
print(students[1])

# 添加列表元素
students.append("Jason")
print(students)

# 删除列表元素
students.remove("Jason")
print(students)

# 修改列表元素
students[1] = "Jason"
print(students)
```

```
['Jesse', 'Andrew']
Jesse
Andrew
['Jesse', 'Andrew', 'Jason']
['Jesse', 'Andrew']
['Jesse', 'Jason']
```

--元组：有序、不可变的元素集合

元组的基本语法：
my_tuple = (element1, element2, element3)

In [10]:
```python
# 创建元组
students = ("Jesse", "Andrew")
print(students)

# 访问元组元素
print(students[0])
print(students[1])

# 不可以改变元组元素！
# 遍历元组元素
for student in students:
    print(student)

# 获取元组长度
print(len(students))
```

```
('Jesse', 'Andrew')
Jesse
Andrew
Jesse
Andrew
2
```

--集合：无序、元素唯一的集合

集合的基本语法：
my_set = {element1, element2, element3}

In [15]:
```python
# 创建集合
students = {"Jesse", "Andrew"}
print(students)

# 添加集合元素
students.add("Jason")
print(students)

# 删除集合元素
students.remove("Jason")
print(students)

# 不能修改集合元素！
# 不能访问某一位置的集合元素！
```

```
{'Andrew', 'Jesse'}
{'Andrew', 'Jason', 'Jesse'}
{'Andrew', 'Jesse'}
```

--字典：键值对的集合

字典的基本语法：
my_dict = {key1: value1, key2: value2, key3: value3}

In [22]:
```
# 创建字典
students = {"Jesse": 13, "Andrew": 14}
print(students)

# 访问字典元素
print(students["Jesse"])
print(students["Andrew"])

# 添加或修改键值
students["Jason"] = 15
print(students)
students["Jesse"] = 13.5
print(students)

# 删除键值
del students["Jason"]
print(students)

# 获取所有键，值，键值对
print(students.keys())
print(students.values())
print(students.items())
```

```
{'Jesse': 13, 'Andrew': 14}
13
14
{'Jesse': 13, 'Andrew': 14, 'Jason': 15}
{'Jesse': 13.5, 'Andrew': 14, 'Jason': 15}
{'Jesse': 13.5, 'Andrew': 14}
dict_keys(['Jesse', 'Andrew'])
dict_values([13.5, 14])
dict_items([('Jesse', 13.5), ('Andrew', 14)])
```

三. 数据结构的总结和对比

--列表 (List)

特点：

1. 有序：列表中的元素按添加顺序排列。
2. 可变：可以随时修改列表中的元素。
3. 重复：列表中可以包含重复的元素。
4. 索引访问：可以通过索引访问任意位置的元素。

应用方向：

1. 存储有序的数据集合：适合存储需要保持顺序的数据，例如学生名单、任务列表。
2. 动态数组：当需要频繁添加、删除、修改元素时，列表非常方便。

--元组 (Tuple)

特点：

1. 有序：元组中的元素按添加顺序排列。
2. 不可变：一旦创建，元组中的元素不能修改。
3. 重复：元组中可以包含重复的元素。
4. 索引访问：可以通过索引访问任意位置的元素。

应用方向：

1. 存储固定的数据集合：适合存储不需要修改的数据，例如坐标、配置信息。
2. 函数返回值：常用于函数返回多个值。

--集合 (Set)

特点：

1. 无序：集合中的元素没有特定的顺序。
2. 唯一：集合中的元素不能重复。
3. 可变：可以随时添加或删除元素。
4. 无索引：不能通过索引访问元素。

应用方向：

1. 去重：适合用于存储唯一的元素集合，例如用户ID、标签集合。
2. 集合操作：支持集合的数学运算，如交集、并集、差集等。

--字典 (Dictionary)

特点:

1. 无序：字典中的键值对没有特定的顺序（在Python 3.7及以后版本中，字典保持插入顺序）。
2. 键唯一：字典中的键不能重复。
3. 可变：可以随时修改、添加或删除键值对。
4. 键值对：通过键来访问对应的值。

应用方向:

1. 键值映射：适合用于存储具有映射关系的数据，例如用户信息、配置选项。
2. 快速查找：通过键快速查找对应的值。

作者: Yming
邮箱: yuemingn@student.unimelb.edu.au