人工智能程序设计


python
import turtle
turtle.setup(650,350,200,200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
turtle.circle(40, 80)
turtle.circle(-40, 80)
turtle.fd(40)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)



人工智能程序设计

第5章 字典与集合

北京石油化工学院 人工智能研究院

刘 强

第5章 字典与集合

我们学习了Python的基本语法、变量、数据类型、列表与元组，以及程序的控制结构

字典是Python中最重要的数据结构之一，它以键值对的形式存储数据，提供了快速的数据查找和更新能力。无论是存储用户信息、配置数据，还是建立数据之间的映射关系，字典都是不可或缺的工具。

集合则专门用于存储不重复的元素，在数据去重、成员检测和集合运算方面有着独特的优势。

有序数据结构与映射数据结构的差异

列表和元组等有序数据结构通过数字索引访问元素，适合存储有顺序关系的数据。但在需要通过某个特征（如姓名、学号）快速查找数据时，需要遍历整个列表，效率较低。面对数据去重、关系匹配等需求时，实现起来也较为复杂。

字典和集合采用哈希表实现，提供了O(1)时间复杂度的查找效率。字典通过键值对建立映射关系，可以用有意义的键（如姓名、ID）直接访问数据。集合自动去重且提供数学集合运算，在数据分析和关系处理中非常高效。

有序数据结构与映射数据结构的差异

Python在字典和集合设计上的优势在于：简洁的语法让创建和使用非常直观，强大的内置方法满足各种操作需求，与其他数据结构良好集成，高效的哈希算法保证了性能。

5.1 字典的创建与操作

字典（dictionary）是Python中最重要的数据结构之一，**它以键值对（key-value pair）**的形式存储数据。

字典提供了快速的数据查找能力，是处理映射关系数据的理想选择。

字典（dictionary）

字典的四个核心特性：

1. 键值对结构：每个元素都由键（key）和值（value）组成，键用于查找对应的值

2. 键的唯一性：字典中的键必须是唯一的，重复的键会被后面的值覆盖

3. 快速查找：通过键查找值的速度非常快，时间复杂度为O(1)，不受字典大小影响

4. 可变性：字典是可变的数据结构，可以随时添加、修改或删除键值对

5.1.1 字典的创建方法

使用花括号创建

最常用的字典创建方式是使用花括号{}。字典中的每个元素都是键值对，键和值之间用冒号分隔，不同的键值对之间用逗号分隔。

```
## 创建空字典
empty_dict = {}
print(empty_dict)  # 输出：{}
```

```
## 创建包含数据的字典
student = {
    "name": "张三",
    "age": 20,
    "grade": 85,
    "city": "北京"
}
print(student)
```

5.1.1 字典的创建方法

使用dict()函数创建

除了直接使用花括号，还可以使用dict()构造函数（用于创建对象的特殊函数）来创建字典。这种方式在动态创建字典或从其他数据结构转换时特别有用。

```
## 使用关键字参数创建字典
student = dict(name="李四", age=21, grade=92, city="上海")
print(student)
## 从键值对列表创建字典
pairs = [("name", "王五"), ("age", 19), ("grade", 88)]
student = dict(pairs)
print(student)
```

5.1.1 字典的创建方法

字典推导式

字典推导式是一种简洁的字典创建方式，类似于列表推导式。它可以根据某种规则快速生成字典，特别适用于数据转换和处理。

```
## 创建平方数字典
squares = {x: x**2 for x in range(1, 6)}
print(squares)  # 输出：{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
## 从列表创建字典
names = ["Alice", "Bob", "Charlie"]
name_lengths = {name: len(name) for name in names}
print(name_lengths)  # 输出：{'Alice': 5, 'Bob': 3, 'Charlie': 7}
```

5.1.2 字典的访问与修改

键值访问

字典通过键来访问对应的值，这是字典最基本也是最重要的操作。需要注意的是，如果访问不存在的键会引发KeyError异常。

```
student = {"name": "张三", "age": 20, "grade": 85}
## 使用方括号访问
print(student["name"])  # 输出：张三
print(student["age"])   # 输出：20
## 访问不存在的键会报错
## print(student["height"])  # KeyError: 'height'
```

5.1.2 字典的访问与修改

get()方法安全访问

为了避免KeyError异常，推荐使用get()方法来安全地访问字典。get()方法在键不存在时返回None或指定的默认值，而不会报错。

```
student = {"name": "张三", "age": 20, "grade": 85}
## 使用get()方法安全访问
print(student.get("name"))     # 输出：张三
print(student.get("height"))   # 输出：None
print(student.get("height", "未知"))  # 输出：未知
```

5.1.2 字典的访问与修改

修改和添加

字典是可变的数据结构，可以随时修改现有键的值或添加新的键值对。

```
student = {"name": "张三", "age": 20, "grade": 85}

## 修改现有键的值
student["age"] = 21
print(student)  # age变为21

## 添加新的键值对
student[“city”] = “北京”
print(student)  # 新增了city键

## 使用update()方法批量更新
student.update({“grade”: 90, “major”: “智能科学"})
print(student)
```

5.1.3 字典的常用方法

keys()、values()、items()

这三个方法是字典操作中最常用的方法，分别用于获取字典的所有键、所有值和所有键值对。它们返回的是视图对象，可以转换为列表进行进一步操作。

```
student = {"name": "张三", "age": 20, "grade": 85, "city": "北京"}

## 获取所有键
keys = student.keys()
print(list(keys))  # 输出：['name', 'age', 'grade', 'city']
```

5.1.3 字典的常用方法

keys()、values()、items()

这三个方法是字典操作中最常用的方法，分别用于获取字典的所有键、所有值和所有键值对。它们返回的是视图对象，可以转换为列表进行进一步操作。

```
student = {"name": "张三", "age": 20, "grade": 85, "city": "北京"}
## 获取所有值
values = student.values()
print(list(values))  # 输出：['张三', 20, 85, '北京']
## 获取所有键值对
items = student.items()
print(list(items))  # 输出：[('name', '张三'), ('age', 20), ('grade', 85), ('city', '北京')]
```

5.1.3 字典的常用方法

删除操作

字典提供了多种删除键值对的方法，每种方法都有不同的特点和使用场景。选择合适的删除方法可以让代码更加安全和高效。

```
student = {"name": "张三", "age": 20, "grade": 85, "city": "北京"}
## 使用del删除指定键
del student["city"]
print(student)
## 使用pop()删除并返回值
age = student.pop("age")
print(f"删除的年龄：{age}")
print(student)
```

5.1.3 字典的常用方法

删除操作

字典提供了多种删除键值对的方法，每种方法都有不同的特点和使用场景。选择合适的删除方法可以让代码更加安全和高效。

```
student = {"name": "张三", "age": 20, "grade": 85, "city": "北京"}
## 使用pop()删除不存在的键（提供默认值）
height = student.pop("height", "未知")
print(f"身高：{height}")
## 使用clear()清空字典
student.clear()
print(student)  # 输出：{}
```

5.1.3 字典的常用方法

其他常用方法

除了基本的增删改查操作，字典还提供了一些实用的辅助方法，这些方法在日常编程中经常用到。

```
student = {"name": "张三", "age": 20, "grade": 85}

## 检查键是否存在
print("name" in student)     # 输出：True
print("height" in student)   # 输出：False

## 获取字典长度
print(len(student))  # 输出：3

## 复制字典
student_copy = student.copy()
print(student_copy)
```

5.1.4 字典的遍历

遍历键

字典的遍历是数据处理中的常见操作。Python提供了多种遍历字典的方式，可以根据需要选择遍历键、值或键值对。

```
student = {"name": "张三", "age": 20, "grade": 85, "city": "北京"}
## 遍历键
print("遍历键：")
for key in student:
    print(key)
## 或者显式使用keys()
for key in student.keys():
    print(f"{key}: {student[key]}")
```

5.1.4 字典的遍历

遍历值

当只需要处理字典中的值而不关心键时，可以直接遍历values()。

```
## 遍历值
print("遍历值：")
for value in student.values():
    print(value)
```

5.1.4 字典的遍历

遍历键值对

当需要同时使用键和值时，遍历items()是最高效的方式。

```
## 遍历键值对
print("遍历键值对：")
for key, value in student.items():
    print(f"{key}: {value}")
```

示例 5.1.1：用户偏好统计系统

展示字典在数据分析中，综合运用字典的创建、访问、修改和遍历等操作。

```
## 统计用户偏好数据
preferences = ["咖啡", "茶", "果汁", "咖啡", "水", "茶", "咖啡", "果汁"]

preference_count = {}
for item in preferences:
    preference_count[item] = preference_count.get(item, 0) + 1

print("偏好统计：")
for item, count in preference_count.items():
    print(f"{item}: {count}次")
```

实践练习

练习 5.1.1：员工档案管理

创建一个员工档案管理系统，实现以下功能：

1. 存储多个员工的信息（姓名、部门、工资、入职时间）

2. 根据姓名查询员工信息

3. 修改员工的工资

4. 统计所有员工的平均工资

实践练习

练习 5.1.2：商品价格查询

创建一个商品价格字典，实现以下功能：

1. 添加新商品和价格
2. 查询商品价格（使用安全访问方法）
3. 更新商品价格
4. 删除下架商品
5. 显示所有商品和价格

实践练习

练习 5.1.3：网站访问统计

编写程序分析网站访问日志：

1. 统计每个页面的访问次数

2. 找出访问量最高的前5个页面

3. 统计总访问量和不同页面数量