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

5.2 集合的创建与操作

北京石油化工学院 人工智能研究院

刘 强

集合（set）

集合（set）是Python中用于存储不重复元素的数据结构。

集合具有自动去重的特性，并且支持高效的成员检测和数学集合运算，是处理唯一性数据和集合关系的理想选择。

集合（set）

集合的三个核心特性：

1. 元素唯一性：集合会自动去除重复元素，每个元素只能出现一次
2. 无序性：集合中的元素没有固定的顺序，不支持索引访问
3. 高效检测：检查元素是否存在的速度非常快，不受集合大小影响

5.2.1 集合的创建方法

使用set()函数

set()函数是创建集合最常用的方法，特别适合从其他数据结构（如列表、字符串）创建集合。需要注意的是，空集合只能用set()创建，不能用{}。

```
## 创建空集合（注意：{}创建的是空字典，不是空集合）
empty_set = set()
print(empty_set)  # 输出：set()
print(type(empty_set))  # 输出：<class 'set'>
```

5.2.1 集合的创建方法

使用set()函数

set()函数是创建集合最常用的方法，特别适合从其他数据结构（如列表、字符串）创建集合。需要注意的是，空集合只能用set()创建，不能用{}。

```
## 从列表创建集合（自动去重）
numbers = [1, 2, 3, 3, 4, 4, 5]
unique_numbers = set(numbers)
print(unique_numbers)  # 输出：{1, 2, 3, 4, 5}
## 从字符串创建集合
letters = set("hello")
print(letters)  # 输出：{'h', 'e', 'l', 'o'}（注意：'l'只出现一次）
```

5.2.1 集合的创建方法

使用花括号创建

当需要创建包含已知元素的集合时，使用花括号{}是最直观的方式。这种方法类似于字典的创建，但只包含值而不包含键值对。

```
## 直接创建包含元素的集合
fruits = {"apple", "banana", "orange", "apple"}
print(fruits)  # 输出：{'banana', 'orange', 'apple'}（自动去重）
## 创建数字集合
numbers = {1, 2, 3, 4, 5}
print(numbers)  # 输出：{1, 2, 3, 4, 5}
```

5.2.1 集合的创建方法

集合推导式

集合推导式是一种简洁的集合创建方式，类似于列表推导式和字典推导式。它可以根据条件快速生成满足要求的集合。

```
## 创建平方数集合
squares = {x**2 for x in range(1, 6)}
print(squares)  # 输出：{1, 4, 9, 16, 25}
## 创建偶数集合
evens = {x for x in range(1, 11) if x % 2 == 0}
print(evens)  # 输出：{2, 4, 6, 8, 10}
```

5.2.2 集合的基本操作

添加元素

集合提供了add()和update()方法来添加元素。

add()用于添加单个元素，update()用于添加多个元素。

由于集合的唯一性，重复添加相同元素不会改变集合。

```
fruits = {"apple", "banana"}
## 添加单个元素
fruits.add("orange")
print(fruits)  # 输出：{'apple', 'banana', 'orange'}
```

5.2.2 集合的基本操作

添加元素

集合提供了add()和update()方法来添加元素。

add()用于添加单个元素，update()用于添加多个元素。

由于集合的唯一性，重复添加相同元素不会改变集合。

```
## 添加已存在的元素（无效果）
fruits.add("apple")
print(fruits)  # 输出：{'apple', 'banana', 'orange'}

## 添加多个元素
fruits.update(["grape", "kiwi", "apple"])
print(fruits)  # 输出：{'apple', 'banana', 'orange', 'grape', 'kiwi'}
```

5.2.2 集合的基本操作

删除元素

集合提供了多种删除元素的方法，每种方法在处理不存在的元素时行为不同。选择合适的删除方法可以让程序更加健壮。

```
fruits = {"apple", "banana", "orange", "grape"}
## 使用remove()删除元素（元素不存在会报错）
fruits.remove("banana")
print(fruits)
```

5.2.2 集合的基本操作

删除元素

集合提供了多种删除元素的方法，每种方法在处理不存在的元素时行为不同。选择合适的删除方法可以让程序更加健壮。

```
## 使用discard()删除元素（元素不存在不会报错）
fruits.discard("kiwi")  # kiwi不存在，但不报错
fruits.discard("apple")
print(fruits)
```

5.2.2 集合的基本操作

删除元素

集合提供了多种删除元素的方法，每种方法在处理不存在的元素时行为不同。选择合适的删除方法可以让程序更加健壮。

```
## 使用pop()随机删除一个元素
removed = fruits.pop()
print(f"删除的元素：{removed}")
print(fruits)
```

5.2.2 集合的基本操作

删除元素

集合提供了多种删除元素的方法，每种方法在处理不存在的元素时行为不同。选择合适的删除方法可以让程序更加健壮。

```
## 清空集合
fruits.clear()
print(fruits)  # 输出：set()
```

5.2.2 集合的基本操作

成员检测

集合的成员检测操作非常高效，时间复杂度为O(1)（即常数时间，不管集合有多大，查找速度都一样快）。这是集合相比列表的重要优势之一，特别适合需要频繁检查元素是否存在的场景。

```
fruits = {"apple", "banana", "orange"}
## 检查元素是否存在
print("apple" in fruits)     # 输出：True
print("grape" in fruits)     # 输出：False
print("kiwi" not in fruits)  # 输出：True
```

5.2.2 集合的基本操作

获取集合长度

使用len()函数可以获取集合中元素的个数。由于集合会自动去重，len()返回的是集合中唯一元素的数量。

```
fruits = {"apple", "banana", "orange"}
print(len(fruits))  # 输出：3

## 注意：重复元素不会影响长度
numbers = {1, 2, 3, 3, 4, 4, 5}
print(len(numbers))  # 输出：5（3和4只计算一次）
```

5.2.3 集合运算

集合支持数学中的各种集合运算，包括并集、交集、差集和对称差集等。这些运算使得集合在处理数据关系时非常实用。

这里我们介绍最常用的三种集合运算：并集、交集和差集。

5.2.3 集合运算

并集（Union）

并集运算获取两个集合中所有不重复的元素，相当于把两个集合"合并"起来。Python提供了两种方式实现并集运算。

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

## 使用 | 操作符
union1 = set1 | set2
print(union1)  # 输出：{1, 2, 3, 4, 5, 6}
```

5.2.3 集合运算

并集（Union）

并集运算获取两个集合中所有不重复的元素，相当于把两个集合"合并"起来。

Python提供了两种方式实现并集运算。

```
## 使用union()方法
union2 = set1.union(set2)
print(union2)  # 输出：{1, 2, 3, 4, 5, 6}
```

5.2.3 集合运算

并集（Union）

并集运算获取两个集合中所有不重复的元素，相当于把两个集合"合并"起来。Python提供了两种方式实现并集运算。

实际应用示例

students_math = {"张三", "李四", "王五"}

students_english = {"李四", "王五", "赵六", "孙七"}

获取所有选课学生

all_students = students_math | students_english

print(f"所有选课学生：{all_students}")

5.2.3 集合运算

交集（Intersection）

交集运算获取两个集合中共有的元素，即同时存在于两个集合中的元素。这在查找共同特征时非常有用。

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
## 使用 & 操作符
intersection1 = set1 & set2
print(intersection1)  # 输出：{3, 4}
## 使用intersection()方法
intersection2 = set1.intersection(set2)
print(intersection2)  # 输出：{3, 4}
```

5.2.3 集合运算

交集（Intersection）

交集运算获取两个集合中共有的元素，即同时存在于两个集合中的元素。这在查找共同特征时非常有用。

```
## 实际应用示例：找出同时选修两门课的学生
students_math = {"张三", "李四", "王五"}
students_english = {"李四", "王五", "赵六", "孙七"}

both_courses = students_math & students_english
print(f"同时选修两门课的学生：{both_courses}")  # 输出：{'李四', '王五'}
```

5.2.3 集合运算

差集（Difference）

差集运算获取在第一个集合中但不在第二个集合中的元素，用于找出独有的元素。

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
## 使用 - 操作符
difference1 = set1 - set2
print(difference1)  # 输出：{1, 2}
```

5.2.3 集合运算

差集（Difference）

差集运算获取在第一个集合中但不在第二个集合中的元素，用于找出独有的元素。

```
## 使用difference()方法
difference2 = set1.difference(set2)
print(difference2)  # 输出：{1, 2}
```

5.2.3 集合运算

差集（Difference）

差集运算获取在第一个集合中但不在第二个集合中的元素，用于找出独有的元素。

```
## 实际应用示例：找出只选修数学课的学生
students_math = {"张三", "李四", "王五"}
students_english = {"李四", "王五", "赵六", "孙七"}
only_math = students_math - students_english
print(f"只选修数学课的学生：{only_math}")  # 输出：{'张三'}
```

5.2.3 集合运算

Ask AI：探索更多集合运算

想要了解更多集合运算？向AI助手提出以下问题：

- "对称差集是什么？如何使用 ^ 操作符？"
- "如何判断集合之间的包含关系？子集和超集的概念是什么？"
- "什么是不相交集合？如何判断两个集合是否有交集？"

5.2.4 集合的遍历

集合的遍历与列表类似，但需要注意集合是无序的，因此遍历的顺序是不固定的。这个特性在某些应用场景中需要特别注意。

```
fruits = {"apple", "banana", "orange"}
## 遍历集合元素
print("水果列表：")
for fruit in fruits:
    print(fruit)
```

示例 5.2.1：数据分析与处理

集合在数据处理中有很多实用的应用场景，特别是在需要去重、查找共同元素等操作时表现出色。

```
## 数据去重
data = [1, 2, 3, 2, 4, 3, 5, 1, 4]
unique_data = list(set(data))
print(f"原数据：{data}")
print(f"去重后：{unique_data}")
```

```
## 找出两个列表的共同元素
list1 = ["apple", "banana", "orange", "grape"]
list2 = ["banana", "kiwi", "orange", "mango"]
common = set(list1) & set(list2)
print(f"共同元素：{common}")
## 找出只在第一个列表中的元素
only_in_list1 = set(list1) - set(list2)
print(f"只在第一个列表中：{only_in_list1}")
```

5.3 Ask AI：字典与集合的高级应用

当你想要深入了解字典和集合的更多特性和高级应用时，可以向AI助手提出以下问题：

- "什么时候使用字典而不是列表？"
- "如何用集合解决数据去重和查找问题？”

- "如何使用集合进行数据过滤和清洗？"
- "集合在算法中有哪些常见应用场景？"

实践练习

练习 5.2.1：数据去重

给定一个包含重复元素的数据列表，使用集合完成以下任务：

1. 去除重复元素

2. 统计原始数据和去重后的数据量

3. 找出重复的元素有哪些

实践练习

练习 5.2.2：技能匹配系统

模拟职位技能匹配：

1. 创建几个职位的技能要求集合

2. 创建求职者的技能集合

3. 找出求职者与职位的匹配技能

4. 找出求职者需要补充的技能

实践练习

练习 5.2.3：数据分析统计

分析两组调查数据：

1. 统计两组数据的总体情况（总数、唯一值数量）

2. 找出两组数据的交集、并集、差集

3. 计算数据的重合度（交集大小/并集大小）