# Лекция 11

## Работа с коллекциями и массивами объектов

### 1. Введение в коллекции и массивы объектов

При разработке программного обеспечения часто возникает необходимость работы с большим количеством данных, представленных в виде объектов. Коллекции и массивы позволяют хранить, организовывать и управлять этими объектами, обеспечивая возможность их эффективного добавления, удаления, поиска и модификации. В Python для работы с коллекциями и массивами существует множество встроенных структур данных, таких как списки, кортежи, множества и словари. Эти структуры делают код более организованным, управляемым и способствуют эффективной обработке данных.

Цель этой лекции — изучить основные типы коллекций и массивов в Python, методы работы с ними, их особенности и применимость к различным задачам.

### 2. Основные коллекции в Python

Python предоставляет несколько встроенных коллекций, которые можно использовать для хранения и управления объектами: список (list), кортеж (tuple), множество (set) и словарь (dict). Каждая из этих коллекций имеет свои особенности и области применения.

### 2.1 Список (list)

Списки — это изменяемая структура данных, которая позволяет хранить упорядоченные элементы, в том числе и объекты. Списки поддерживают добавление, удаление и изменение элементов, а также обеспечивают доступ к элементам по индексу.

Пример работы со списком:

python
Копировать код

```
# Создание списка объектов
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

people = [Person("Alice", 30), Person("Bob", 25), Person("Charlie", 35)]
```

```
# Добавление нового объекта в список
people.append(Person("Diana", 28))

# Обход списка
for person in people:
    print(person.name, person.age)
```

Списки подходят для хранения изменяемых данных, к которым необходимо получать доступ по индексу или выполнять сортировку.

### 2.2 Кортеж (tuple)

Кортежи — это неизменяемые коллекции, которые используются для хранения фиксированного набора объектов. Они полезны, когда требуется гарантировать неизменяемость данных.

Пример работы с кортежем:

python
Копировать код
```
# Создание кортежа объектов
person1 = Person("Alice", 30)
person2 = Person("Bob", 25)
people_tuple = (person1, person2)

# Обход кортежа
for person in people_tuple:
    print(person.name, person.age)
```

### 2.3 Множество (set)

Множества — это неупорядоченные коллекции уникальных элементов. Они не позволяют хранить дубликаты и поддерживают быстрый поиск, что делает их идеальными для операций объединения, пересечения и разности.

Пример работы с множеством:

python
Копировать код
```
# Создание множества
person1 = Person("Alice", 30)
person2 = Person("Bob", 25)
people_set = {person1, person2}

# Добавление элемента в множество
```

people_set.add(Person("Charlie", 35))

### 2.4 Словарь (dict)

Словари — это коллекции пар «ключ-значение». Каждый элемент в словаре имеет уникальный ключ, связанный с определенным значением, что делает словари идеальными для хранения данных, которые необходимо организовать и к которым нужен быстрый доступ по ключу.

Пример работы со словарем:

python
Копировать код
```
# Создание словаря объектов
people_dict = {
    "Alice": Person("Alice", 30),
    "Bob": Person("Bob", 25)
}

# Доступ к объекту по ключу
print(people_dict["Alice"].age)

# Добавление нового элемента в словарь
people_dict["Charlie"] = Person("Charlie", 35)
```

## 3. Коллекции объектов и массивы

Коллекции в Python, такие как списки и словари, могут быть использованы как массивы для хранения объектов. Хотя в Python нет статического массива, как в некоторых других языках, списки выполняют аналогичную функцию динамических массивов и предоставляют возможность управлять набором объектов.

### 3.1 Работа со списками объектов

Список может использоваться для хранения объектов одного или нескольких типов. Это позволяет работать с группой данных как с единым целым.

Пример работы со списком объектов:

python
Копировать код
```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade
```

```
students = [
    Student("Alice", "A"),
    Student("Bob", "B"),
    Student("Charlie", "C")
]

# Сортировка списка объектов по оценке
students.sort(key=lambda x: x.grade)

# Вывод списка студентов
for student in students:
    print(student.name, student.grade)
```

### 3.2 Списки списков и вложенные структуры

Вложенные списки позволяют создавать двумерные и многомерные структуры для работы с массивами объектов. Это полезно для представления таблиц данных или сеток.

Пример работы со списком списков:

```
python
Копировать код
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Доступ к элементу
print(matrix[1][2])  # Выводит 6
```

## 4. Функции и методы для работы с коллекциями

Python предоставляет широкий набор встроенных функций и методов для работы с коллекциями, которые помогают эффективно управлять и обрабатывать данные.

### 4.1 Методы для списков

- append(): Добавляет элемент в конец списка.
- extend(): Добавляет все элементы из переданного списка.
- remove(): Удаляет первое вхождение указанного элемента.
- pop(): Удаляет элемент по индексу и возвращает его.

Пример:

python
Копировать код
```
numbers = [1, 2, 3]
numbers.append(4)
numbers.extend([5, 6])
numbers.remove(2)
print(numbers)  # Вывод: [1, 3, 4, 5, 6]
```

### 4.2 Методы для словарей

- keys(): Возвращает список всех ключей.
- values(): Возвращает список всех значений.
- items(): Возвращает пары ключ-значение.

Пример:

python
Копировать код
```
person_data = {"name": "Alice", "age": 30}
print(person_data.keys())   # Вывод: dict_keys(['name', 'age'])
print(person_data.values()) # Вывод: dict_values(['Alice', 30])
```

### 4.3 Итерации и фильтрация

Python поддерживает методы для итерации и фильтрации элементов коллекций. Например, можно использовать генераторы списков или функции filter и map для обработки данных.

Пример фильтрации:

python
Копировать код
```
# Фильтрация студентов с оценкой "A"
top_students = [student for student in students if student.grade == "A"]
```

## 5. Применение коллекций для работы с объектами

Коллекции широко используются в реальных приложениях, где требуется работа с набором объектов, таких как списки пользователей, товары в корзине, записи базы данных и другие подобные структуры.

### 5.1 Пример: Управление студентами

Предположим, что у нас есть программа для управления студентами и их оценками. Мы можем использовать список объектов для хранения информации о каждом студенте, сортировать и фильтровать данные.

python
Копировать код
```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

# Создание списка студентов
students = [
    Student("Alice", "A"),
    Student("Bob", "B"),
    Student("Charlie", "A"),
    Student("Diana", "C")
]

# Фильтрация по оценке "A"
top_students = [student for student in students if student.grade == "A"]
for student in top_students:
    print(student.name)
```

**5.2 Пример: Управление библиотекой книг**

Для управления библиотекой книг можно использовать словарь, где ключом будет ISBN книги, а значением — объект, представляющий книгу.

python
Копировать код
```
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn

# Создание словаря книг
library = {
    "978-0132350884": Book("Clean Code", "Robert C. Martin", "978-0132350884"),
    "978-1492078005": Book("Python Crash Course", "Eric Matthes", "978-1492078005")
}
```

```
# Доступ к книге по ISBN
isbn = "978-0132350884"
book = library.get(isbn)
if book:
    print(book.title, "-", book.author)
```

### 6. Коллекции из модуля collections

Python предоставляет модуль collections, который включает дополнительные типы коллекций, такие как Counter, defaultdict, deque и OrderedDict.

### 6.1 Counter

Counter — это специальный словарь, который используется для подсчета количества вхождений элементов.

python
Копировать код
```
from collections import Counter

counter = Counter(["apple", "banana", "apple", "orange", "banana", "apple"])
print(counter)  # Вывод: Counter({'apple': 3, 'banana': 2, 'orange': 1})
```

### 6.2 defaultdict

defaultdict автоматически создает значение по умолчанию для несуществующего ключа.

python
Копировать код
```
from collections import defaultdict

dd = defaultdict(int)
dd["apple"] += 1
print(dd["apple"])  # Вывод: 1
```

### 6.3 deque

deque — это двухсторонняя очередь, которая поддерживает добавление и удаление элементов с обоих концов.

python
Копировать код
```
from collections import deque

dq = deque([1, 2, 3])
```

```
dq.appendleft(0)
dq.append(4)
print(dq)  # Вывод: deque([0, 1, 2, 3, 4])
```

**7. Заключение**

Коллекции и массивы объектов играют важную роль в разработке приложений, предоставляя эффективные способы хранения и управления данными. Они позволяют организовать данные, поддерживая их удобный доступ, добавление, удаление и поиск, что особенно важно в задачах обработки большого объема информации. Коллекции, такие как списки, множества и словари, обеспечивают гибкость, а массивы — эффективность в работе с данными однотипного характера. Эти структуры данных способствуют созданию производительного и структурированного кода, облегчая разработку и поддержку приложений, особенно в условиях масштабируемых и динамичных проектов.