Петербургский государственный университет путей сообщения Императора Александра I

Введение в Python

Ведет: *аспирант 2 года* **Волков Егор Алексеевич**
gole00201@gmail.com
ауд. 11 - 304

Санкт-Петербург 2024

Работа ссылок в Python

- Python использует ссылки (references) для работы с объектами. Когда переменной присваивается объект, она просто указывает на этот объект в памяти.

```
a = [1, 2, 3]
b = a  # Теперь обе переменные ссылаются на один и тот же список
b.append(4)
print(a)  # Выведет: [1, 2, 3, 4] — изменения в b отражаются на a
```

Как это работает?

- Изменяемые объекты: такие как списки, словари и множества — могут изменяться напрямую через ссылки. Если несколько переменных ссылаются на один и тот же объект, изменения в одной переменной отразятся на всех остальных.
- Неизменяемые объекты: такие как строки, числа и кортежи — не могут быть изменены. Любая операция, изменяющая неизменяемый объект, создаст новый объект в памяти.

```
x = 10
y = x
y += 5
print(x)  # Выведет: 10 — x не изменился
print(y)  # Выведет: 15 — y теперь ссылается на другой объект
```

Глубокое и поверхностное копирование

Поверхностное копирование

- Метод copy() создаёт копию объекта, но копирует только сам объект, без вложенных структур. Вложенные объекты продолжают ссылаться на исходные данные.

```python
import copy

a = [1, [2, 3], 4]
b = copy.copy(a)
b[1].append(5)
print(a)  # Выведет: [1, [2, 3, 5], 4] — вложенные объекты изменились
```

Глубокое копирование

- Глубокое копирование (метод deepcopy()) создаёт полную копию объекта, включая все вложенные объекты.

```
b = copy.deepcopy(a)
b[1].append(6)
print(a)  # Выведет: [1, [2, 3, 5], 4] — исходный объект не изменился
print(b)  # Выведет: [1, [2, 3, 5, 6], 4]
```

Как применять на практике?

- Чтобы избежать изменения списков (или других изменяемых объектов) в Python, важно понимать, как работают ссылки на объекты. Когда переменная присваивается списку, она лишь указывает на этот список в памяти, а не создаёт его копию. Поэтому изменение списка через одну переменную будет отражаться на всех переменных, ссылающихся на тот же объект.

Использовать метод copy

- Метод copy() создаёт поверхностную копию списка, то есть копируются только сами элементы верхнего уровня, но если элементы внутри списка являются изменяемыми объектами (например, вложенные списки), они всё равно будут ссылаться на исходные объекты.

```
original_list = [1, 2, 3, [4, 5]]
copied_list = original_list.copy()

copied_list[0] = 100
copied_list[3].append(6)

print(original_list)  # Выведет: [1, 2, 3, [4, 5, 6]] — вложенный список изменился
print(copied_list)    # Выведет: [100, 2, 3, [4, 5, 6]] — первый элемент изменён
```

Использовать метод deepcopy

- Если в вашем списке есть вложенные списки (или другие изменяемые объекты), и вы хотите избежать изменений как на верхнем уровне, так и внутри вложенных объектов, нужно использовать глубокое копирование с помощью функции deepcopy() из модуля copy.

```python
import copy

original_list = [1, 2, 3, [4, 5]]
deep_copied_list = copy.deepcopy(original_list)

deep_copied_list[3].append(6)

print(original_list)      # Выведет: [1, 2, 3, [4, 5]] — исходный список не изменён
print(deep_copied_list)   # Выведет: [1, 2, 3, [4, 5, 6]]
```

Копирование списка с помощью срезов

- **Срезы в Python** — это мощный инструмент, позволяющий получать части последовательностей, таких как строки, списки или кортежи. Они дают возможность выбирать подмножество элементов, а также изменять или копировать части последовательности.

Синтаксис

```
sequence[start:stop:step]
```

- start — начальный индекс (включительно). Если не указан, срез начинается с начала последовательности.
- stop — конечный индекс (не включается в результат). Если не указан, срез продолжается до конца последовательности.
- step — шаг, с которым будут извлекаться элементы (по умолчанию равен 1).

Примеры

- Срез последовательности от начала до конца с шагом 1:

```
my_list = [0, 1, 2, 3, 4, 5]
print(my_list[1:4])  # Выведет: [1, 2, 3]
```

- Пропуск значений с шагом:

```
my_list = [0, 1, 2, 3, 4, 5]
print(my_list[::2])  # Выведет: [0, 2, 4]
```

- Обратный порядок:

```
my_list = [0, 1, 2, 3, 4, 5]
print(my_list[::-1])  # Выведет: [5, 4, 3, 2, 1, 0]
```

Использование срезов для копирования

```
my_list = [1, 2, 3]
copied_list = my_list[:]  # Создаст поверхностную копию
copied_list[0] = 100
print(my_list)  # Выведет: [1, 2, 3]
print(copied_list)  # Выведет: [100, 2, 3]
```

Задачи

1. Циклический сдвиг массива

- Дан список чисел. Требуется выполнить циклический сдвиг на заданное число позиций. Элементы, смещаемые за пределы массива, должны "переходить" в начало.

```
# Вход:
array = [1, 2, 3, 4, 5]
shift = 2
# Выход:
[4, 5, 1, 2, 3]
```

2. Перестановка вложенных списков

- Дан вложенный список, состоящий из нескольких списков одинаковой длины. Нужно выполнить циклический сдвиг элементов внутри каждого из вложенных списков.

```
# Вход:
matrix = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]
shift = 1
# Выход:[[3, 1, 2],
         [6, 4, 5],
         [9, 7, 8]]
```

3.Разворот блоков в списке

- Дан список чисел и размер блока. Нужно развернуть элементы в каждом блоке по отдельности. Если последний блок меньше по размеру, его нужно оставить без изменений.

```
# Вход:
array = [1, 2, 3, 4, 5, 6, 7]
block_size = 3

# Выход:
[3, 2, 1, 6, 5, 4, 7]
```

4. Поиск максимальной суммы подмассива фиксированной длины

- Дан список целых чисел и длина подмассива k. Нужно найти подмассив длины k, сумма элементов которого будет максимальной.

```
# Вход:
array = [1, -2, 3, 4, -1, 2, 1, -5, 4]
k = 3
# Выход:
[3, 4, -1]  # Сумма 6
```