IT CAREER HUB

Урок 15.

Списки. Работа с памятью

IT CAREER HUB

Изменяемые типы данных

Изменяемые типы данных — это такие структуры данных, которые позволяют изменять своё содержимое после создания объекта, не создавая при этом новый объект.

Особенности изменяемых типов данных

- **Изменение в месте**: Значения внутри изменяемого объекта могут изменяться, не создавая нового объекта.
- **Ссылочная природа**: При передаче изменяемых объектов в функции передаётся не копия объекта, а ссылка на него.
- **Эффективность**: Экономия памяти, так как изменяется существующий объект, а не создаётся новый.
- **Методы изменения**: Добавление, удаление, замена или сортировка элементов.

Куча

Куча (Heap) — это область динамической памяти, в которой хранятся все объекты в Python.

Основные аспекты работы с кучей:

- **Выделение памяти**: Python автоматически управляет памятью.
- **Доступ к объектам по ссылке**: Переменные хранят ссылки на объекты в куче.
- **Изменяемые и неизменяемые объекты**: Изменяемые объекты могут обновляться на месте.
- **Сборщик мусора**: Удаляет неиспользуемые объекты для освобождения памяти.

Функция id

Функция `id()` возвращает уникальный идентификатор объекта, который представляет его местоположение в памяти.

Пример использования `id()`

IT CAREER HUB

```python
a = [1, 2, 3]
b = a
print(id(a))  # Идентификатор объекта a
print(id(b))  # Тот же идентификатор

c = [1, 2, 3]
print(id(c))  # Другой идентификатор, так как это другой объект
```

Как работает присваивание

Присваивание неизменяемых объектов

Когда неизменяемый объект передаётся в другую переменную или функцию, Python копирует по ссылке.

Пример со строкой (неизменяемый объект)

```python
text1 = "hello"
text2 = text1  # text2 ссылается на тот же объект
text1 + " Python"  # Создаётся новый объект
print(text1)

text2 += " world"  # Создаётся новый объект "hello world"
print(text1)
print(text2)
```

Присваивание изменяемых объектов

Когда изменяемый объект присваивается другой переменной, копирование происходит **по ссылке**.

Пример со списком (изменяемый объект)

```python
list_a = [1, 2, 3]
list_b = list_a  # list_b ссылается на тот же объект, что и list_a

# Добавление нового элемента
list_b.append(4)  # Изменение объекта по ссылке
print(list_a)
print(list_b)

# Изменение элемента по индексу
```

```
list_b[0] = 'new'
print(list_a)
print(list_b)
```

IT CAREER HUB

Задания для закрепления 1

1. Что произойдёт, если две переменные ссылаются на один и тот же изменяемый объект, и его содержимое изменится через одну из переменных?

a. Содержимое объекта останется неизменным.
b. Изменится содержимое только одной переменной.
c. Изменится содержимое обеих переменных, так как они ссылаются на один объект.
d. Python создаст новую копию объекта для каждой переменной.

Посмотреть ответ

2. Какой результат будет выведен при выполнении следующего кода?

```python
a = [10, 20, 30]
b = a
b.append(40)
a.append(30)
print(a)
```

a. `[10, 20, 30]`
b. `[10, 20, 30, 30]`
c. `[10, 20, 30, 30, 40]`
d. `[10, 20, 30, 40, 30]`

Посмотреть ответ

3. Что возвращает функция `id()` в Python?

a. Тип объекта.
b. Длину объекта.
c. Уникальный идентификатор объекта в памяти.
d. Значение объекта.

Посмотреть ответ

4. Какой результат будет выведен при выполнении следующего кода?

```python
x = [1, 2, 3]
y = [1, 2, 3]
print(id(x) == id(y))
```

a. True
b. False
c. Ошибка
d. Зависит от типа данных

Посмотреть ответ

IT CAREER HUB

Вложенные коллекции

Вложенные коллекции — это коллекции, которые содержат другие коллекции в качестве элементов. Это позволяет строить более сложные структуры данных, такие как многомерные массивы, таблицы и другие.

Примеры вложенных коллекций

```python
# Список с числами и списками
list_elements = [[1, 2, 3], [4, 5], 6, [7], [8, 9]]
print(list_elements)

# Кортеж со списками
lists_tuple = ([1, 2], [3, 4], [5, 6])
print(lists_tuple)

# Список со списками
lists = [[1, 2], [3, 4]]
print(lists)
```

Доступ к вложенным элементам

Для доступа к элементам вложенного списка используется несколько уровней индексации.

Пример списка с вложенными элементами

```python
list_elements = [[1, 2, 3], [4, 5], 6, [7, [8, [9], 10]]]
```

Доступ к элементам первого уровня:

```python
print(list_elements[0])  # [1, 2, 3]
print(list_elements[1])  # [4, 5]
print(list_elements[2])  # 6
print(list_elements[3])  # [7, [8, [9], 10]]
```

Доступ к элементам внутри вложенных списков:

```python

print(list_elements[0][1])  # 2
```

Использование временных переменных для доступа:

```python
first_nested_list = list_elements[0]
second_element = first_nested_list[1]
print(second_element)  # 2
```

IT CAREER HUB

Примеры глубокой вложенности

```python
print(list_elements[3][1][2])  # 10
print(list_elements[3][1][1][0])  # 9
```

IT CAREER HUB

Изменение элементов вложенных списков

Изменяемые объекты, такие как списки, позволяют изменять свои элементы на любом уровне вложенности.

Пример изменения элементов во вложенном списке

```python
list_elements = [[1, 2], [3, 4], [5, 6]]
```

Замена элемента в списке:

```python
list_elements[0][1] = "two"
print(list_elements)  # [[1, 'two'], [3, 4], [5, 6]]
```

Замена вложенного списка:

```python
list_elements[2] = "new"
print(list_elements)  # [[1, 'two'], [3, 4], 'new']
```

Пример с изменением строки внутри списка

```python
list_elements[0][1] = list_elements[0][1].upper()
print(list_elements)  # [[1, 'TWO'], [3, 4], 'new']
```

IT CAREER HUB

Изменение элемента списка (если он сам является изменяемым объектом)

Если элемент внутри вложенного списка является изменяемым объектом (например, другой список), можно изменить его содержимое.

Пример (изменение списка внутри списка)

```python
list_elements[0].append('new value')
print(list_elements)  # [[1, 'TWO', 'new value'], [3, 4], 'new']
```

IT CAREER HUB

Итерация по вложенным коллекциям

При работе с вложенными коллекциями, такими как списки, можно перебирать все элементы, включая вложенные структуры.

Данные

```python
list_elements = [[1, 2], [3, 4], [5, 6]]
```

1. Итерация по элементам первого уровня вложенности

Простая итерация по верхнему уровню коллекции.

```python
for sublist in list_elements:
    print(sublist)
```

2. Итерация по элементам нескольких уровней вложенности

Используем вложенные циклы `for`.

```python
for sublist in list_elements:
    for item in sublist:
        print(item, end=" ")
```

3. Итерация с изменением элементов

Изменяем элементы коллекции во время итерации.

```python
for i, sublist in enumerate(list_elements):
    for j, item in enumerate(sublist):
        list_elements[i][j] = item + 1

print(list_elements)
```

ICH IT CAREER HUB

Оператор del

Оператор `del` используется для удаления элементов коллекции по индексу, срезу, а также для удаления целых переменных.

Особенности:

- Может удалять отдельные элементы, срезы или целые объекты.
- Не возвращает значение удалённого элемента.
- Освобождает память путём удаления переменных.
- Попытка обращения к удалённому объекту вызывает `NameError`.

Синтаксис

```python
del list[index]  # Удаление одного элемента по индексу
del list[start:end]  # Удаление среза элементов
del variable  # Удаление переменной
```

Примеры

```python
# Удаление элемента по индексу
numbers = [10, 20, 30, 40]
del numbers[2]
print(numbers)  # [10, 20, 40]

# Удаление первого элемента
fruits = ["apple", "banana", "cherry"]
del fruits[0]
print(fruits)  # ['banana', 'cherry']

# Удаление среза
numbers = [10, 20, 30, 40, 50]
del numbers[1:3]
print(numbers)  # [10, 40, 50]
```

```python
# Удаление всех элементов (аналог `clear`)
numbers = [10, 20, 30, 40]
del numbers[:]
print(numbers)  # []

# Удаление переменной
old_numbers = [1, 2, 3]
new_numbers = old_numbers
del old_numbers
print(new_numbers)  # [1, 2, 3]
```

IT CAREER HUB

Задания для закрепления 2

1. Какой результат будет выведен при выполнении следующего кода?

```python
nested_list = [10, [20, 30], [40, [50, 60]]]
nested_list[1][1] = "new"
print(nested_list)
```

a. `[10, [20, "new", 30], [40, [50, 60]]]`
b. `[10, [20, "new"], [40, [50, 60]]]`
c. `[10, [20, 30], [40, [50, "new"]]]`
d. Ошибка

Посмотреть ответ

2. Какой результат будет выведен при выполнении следующего кода?

```python
a = [1, 2, 3]
b = a
del a
print(b)
```

a. `Ошибка, так как список удалён`
b. `[1, 2, 3]`
c. `[]`
d. Переменная b тоже будет удалена

Посмотреть ответ

Копирование списка

При копировании списков важно понимать, что присваивание списка другой переменной не создаёт новую копию. Вместо этого создаётся ссылка на тот же объект.

Пример присваивания (копирование ссылки)

```python
original_list = [1, 2, 3]
copied_list = original_list  # Это не создаёт новую копию
copied_list[0] = 99
print(original_list)  # [99, 2, 3]
```

Важно: При изменении `copied_list` изменяется и `original_list`, так как они ссылаются на один и тот же объект в памяти.

IT CAREER HUB

Поверхностное и глубокое копирование

Копирование списков можно осуществлять разными способами. Существуют два подхода к копированию:

- **Поверхностное копирование** — подходит для списков без вложенных структур или если вложенные элементы не будут изменяться.
- **Глубокое копирование** — рекомендуется для списков со вложенными объектами, когда нужно создать независимую копию на всех уровнях.

Поверхностное копирование

Поверхностное копирование создаёт новый список, копируя элементы из исходного списка, то есть ссылки на них. Если в списке есть вложенные списки или другие изменяемые объекты, то изменение этих вложенных объектов в копии также отразится на исходном списке.

Способы выполнения поверхностного копирования:

1. Метод copy( )

```python
original_list = [1, 2, 3]
copied_list = original_list.copy()
copied_list[0] = 99
print(original_list)  # [1, 2, 3]
print(copied_list)  # [99, 2, 3]
```

2. Срез [:]

```python
original_list = [1, 2, 3]
copied_list = original_list[:]
copied_list[0] = 99
print(original_list)  # [1, 2, 3]
print(copied_list)  # [99, 2, 3]
```

3. Функция `list()`

```python
original_list = [1, 2, 3]
copied_list = list(original_list)
copied_list[0] = 99
print(original_list)  # [1, 2, 3]
print(copied_list)  # [99, 2, 3]
```

4. Копирование списка с вложенными объектами:

```python
original_list = [[1, 2], [3, 4]]
shallow_copy = original_list.copy()
shallow_copy[0][0] = 99
print(original_list)  # [[99, 2], [3, 4]]
print(shallow_copy)  # [[99, 2], [3, 4]]
```

Глубокое копирование

Глубокое копирование создаёт новый список и полностью копирует все вложенные структуры, создавая независимые копии каждого вложенного объекта. Для этого используется функция `copy.deepcopy()` из модуля `copy`.

Пример глубокого копирования

```python
import copy
original_list = [[1, 2], [3, 4]]
deep_copy = copy.deepcopy(original_list)
deep_copy[0][0] = 99
print(original_list)  # [[1, 2], [3, 4]]
print(deep_copy)  # [[99, 2], [3, 4]]
```

Поверхностное копирование быстрее, так как оно копирует только верхний уровень, но глубокое копирование создаёт полную копию всех вложенных структур, обеспечивая полную независимость копии от оригинала.

IT CAREER HUB

Задания для закрепления 3

1. Какой результат будет выведен при выполнении следующего кода?

```python
original_list = [[1, 2], [3, 4]]
shallow_copy = original_list.copy()
shallow_copy[1][0] = 0
print(original_list)
```

a. `[[1, 2], [3, 4]]`
b. `[[1, 2], [0, 4]]`
c. `[[1, 0], [3, 4]]`
d. Ошибка

Посмотреть ответ

2. Какой метод используется для создания поверхностной копии списка?

a. `deepcopy()`
b. `copy()`
c. `clone()`
d. `duplicate()`

Посмотреть ответ

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Изменение объекта через одну из переменных	Ответ: c
2. Результат выполнения кода	Ответ: d
3. Функция `id()` в Python	Ответ: c
4. Результат выполнения кода	Ответ: b
Задания на закрепление 2	Вернуться к заданиям
1. Результат выполнения кода	Ответ: b
2. Результат выполнения кода	Ответ: b
Задания на закрепление 3	Вернуться к заданиям
1. Результат выполнения кода	Ответ: b
2. Поверхностная копия списка	Ответ: b

IT CAREER HUB

Практическая работа

Фильтрация элементов в группах

Напишите программу, которая создаёт копию вложенного списка. Затем в копии необходимо удалить элементы, которые меньше среднего значения всех элементов вложенного списка. Убедитесь, что исходный список остался неизменным.

Данные:

```Python
nested_list = [[10, 15, 20], [5, 25, 30], [35, 40, 80]]
```

Пример вывода:

```None
Исходный список: [[10, 15, 20], [5, 25, 30], [35, 40, 80]]
Глубокая копия после изменений: [[15, 20], [25, 30], [80]]
```

Решение:

```Python
import copy

nested_list = [[10, 15, 20], [5, 25, 30], [35, 40, 80]]
deep_copy = copy.deepcopy(nested_list)

for sublist in deep_copy:
    avg = sum(sublist) / len(sublist)
    for i in range(len(sublist) - 1, -1, -1):  # Проход с конца, чтобы избежать смещения
        if sublist[i] < avg:
            del sublist[i]

print("Исходный список:", nested_list)
print("Глубокая копия после изменений:", deep_copy)
```