IT CAREER HUB

Python

Списки. Работа с памятью

Преподаватель

Портрет

Имя Фамилия

Текущая должность

Количество лет опыта

Какой у Вас опыт - ключевые кейсы

Самые яркие проекты

Дополнительная информация по вашему усмотрению

Корпоративный e-mail

Социальные сети (по желанию)

Важно

- Камера должна быть включена на протяжении всего занятия
- В течение занятия вопросы задавать в чате или когда преподаватель спрашивает, есть ли у Вас вопросы
- Вести себя уважительно и этично по отношению к остальным участникам занятия
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя

Повторение

- Методы списков
- Удаление элементов
- Поиск и подсчет элементов
- Метод reverse
- Функции min, max, sum

План занятия

- Изменяемые типы данных
- Примеры вложенных коллекций
- Изменение элементов вложенных списков
- Итерация по вложенным коллекциям
- Оператор del
- Поверхностное и глубокое копирование
- Глубокое копирование

ICH IT CAREER HUB

ОСНОВНОЙ БЛОК

ICH IT CAREER HUB

Изменяемые типы данных

Полезно знать

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** – позволяет модифицировать содержимое без создания нового объекта.
- **Ссылочная природа** – передача в функцию или переменную создаёт ссылку на объект, а не его копию.
- **Эффективность** – уменьшение нагрузки на память за счёт работы с одним объектом.
- **Методы изменения** – возможность добавлять, удалять и изменять элементы внутри структуры.

Полезно знать

ICH
IT CAREER HUB

Куча

Это область динамической памяти, в которой хранятся все объекты в Python. Используется для управления памятью и позволяет программам создавать, хранить, изменять и удалять объекты.

Основные аспекты работы с кучей

- **Выделение памяти** – автоматически выполняется интерпретатором Python.
- **Доступ по ссылке** – переменные хранят ссылки, а не сами объекты.
- **Изменяемые и неизменяемые объекты** – неизменяемые (`int`, `str`, `tuple`) создают новые объекты при изменении, изменяемые (`list`, `dict`, `set`) модифицируются на месте.
- **Сборщик мусора** – Python автоматически очищает неиспользуемые объекты.

Полезно знать

Функция id

Возвращает уникальный идентификатор объекта, который представляет его местоположение в памяти.

Функция id

Синтаксис

```
id(object)
```

Пояснение

- `object` — объект, для которого нужно узнать его идентификатор

Пример использования id()

```
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
```

Особенности id

- Идентификатор, возвращаемый функцией `id()`, не может измениться для объекта, пока он существует.
- Если два объекта имеют одинаковый идентификатор, это означает, что они ссылаются на один и тот же объект в памяти.
- После того как объект перестаёт существовать, его идентификатор может быть присвоен другому объекту.

Важно

Функция `id()` полезна для проверки, ссылаются ли две переменные на один и тот же объект в памяти.

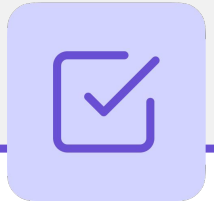
IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

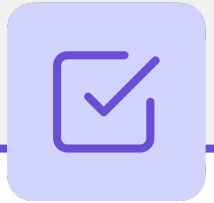
Как работает присваивание

Присваивание



Когда вы присваиваете одну переменную другой, **обе переменные** начинают ссылаться на **один и тот же объект** в памяти. Изменения, внесённые через одну переменную, будут отражены и в другой, так как обе они ссылаются на один и тот же объект.

Присваивание неизменяемых объектов

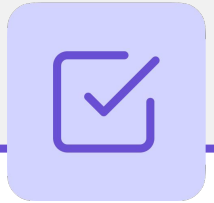


Когда неизменяемый объект передаётся в другую переменную или функцию, Python **копирует по ссылке**. Это значит, что создаётся **новая ссылка на тот же объект в памяти.** Любое "изменение" приведёт к созданию нового объекта в памяти.

Пример со строкой

```
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**. Обе переменные будут указывать на **один и тот же объект в памяти**. Любое изменение отразится на **обеих переменных**

Пример со списком

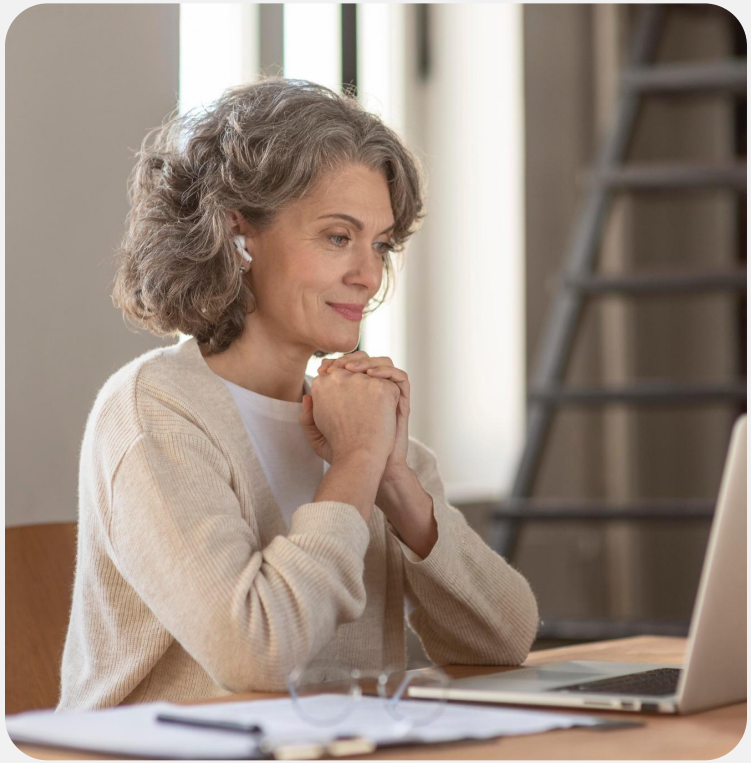
```
list_a = [1, 2, 3]
list_b = list_a  # list_b ссылается на тот же объект, что и list_a
# Добавление нового элемента
list_b.append(4)  # Изменение объекта по ссылке
print(list_a)
print(list_b)
# Изменение элемента по индексу
list_b[0] = 'new'
print(list_a)
print(list_b)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

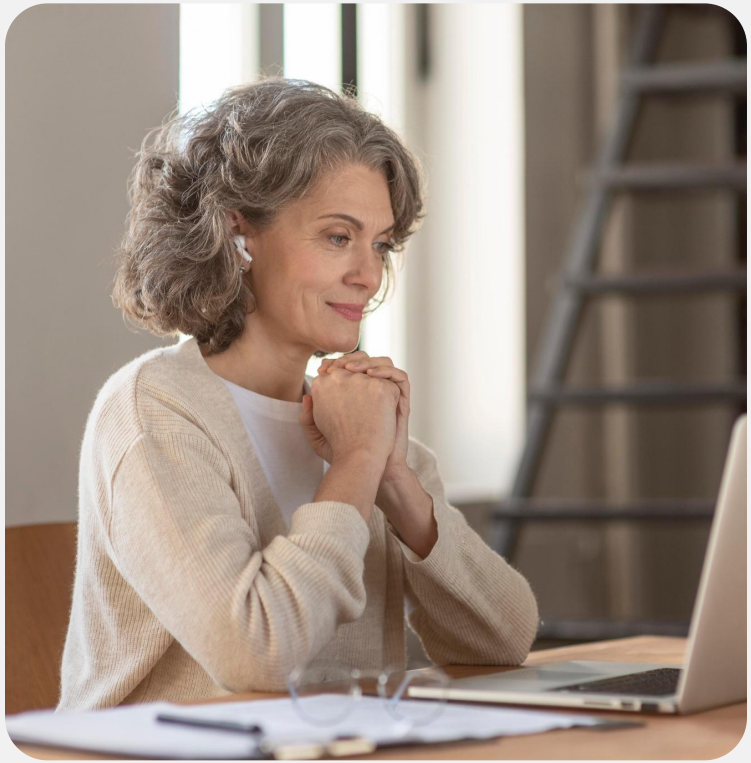
ЗАДАНИЕ



Выберите правильный вариант ответа

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

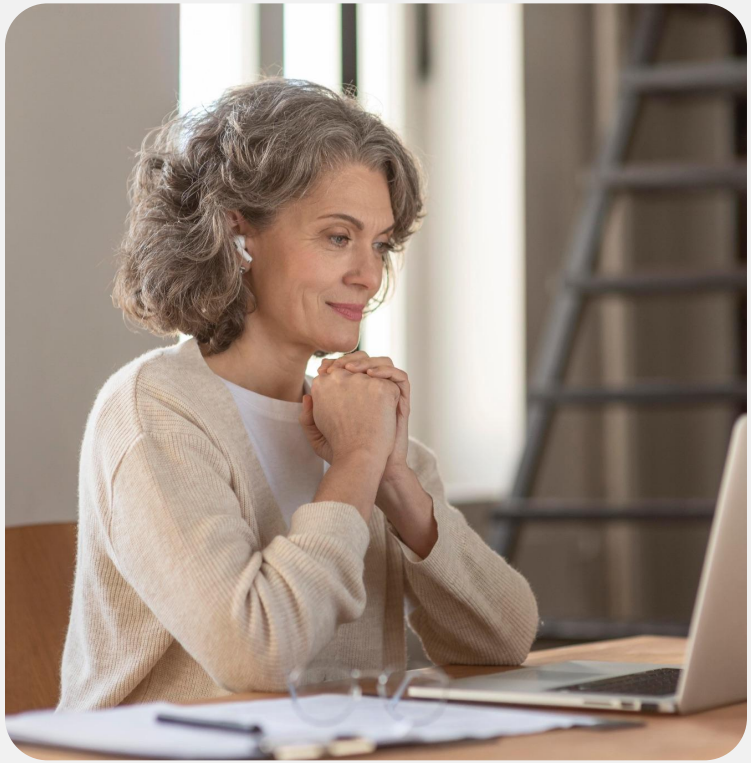
a. Содержимое объекта останется неизменным.
b. Изменится содержимое только одной переменной.
c. Изменится содержимое обеих переменных, так как они ссылаются на один объект.
d. Python создаст новую копию объекта для каждой переменной.



Выберите правильный вариант ответа

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.

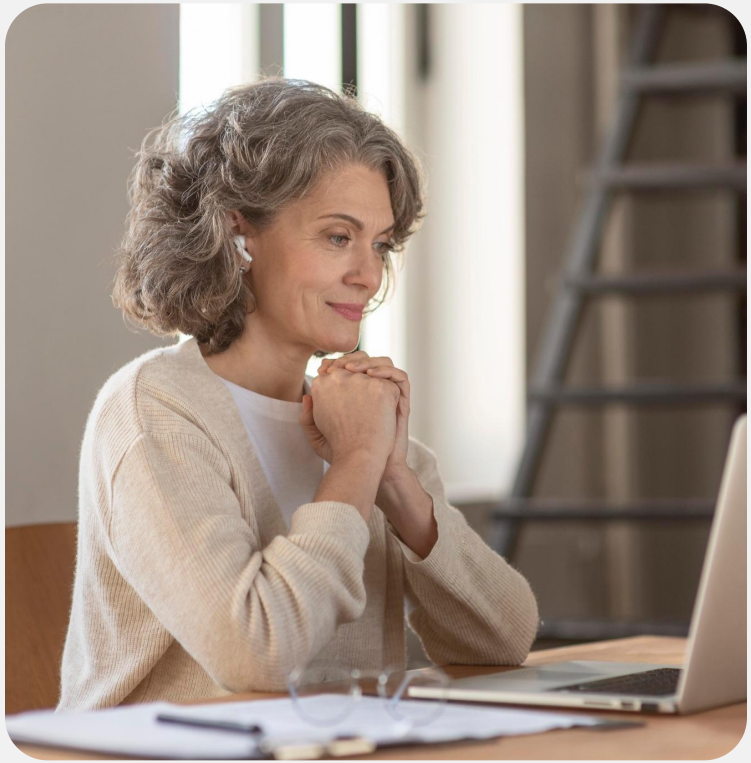


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
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]

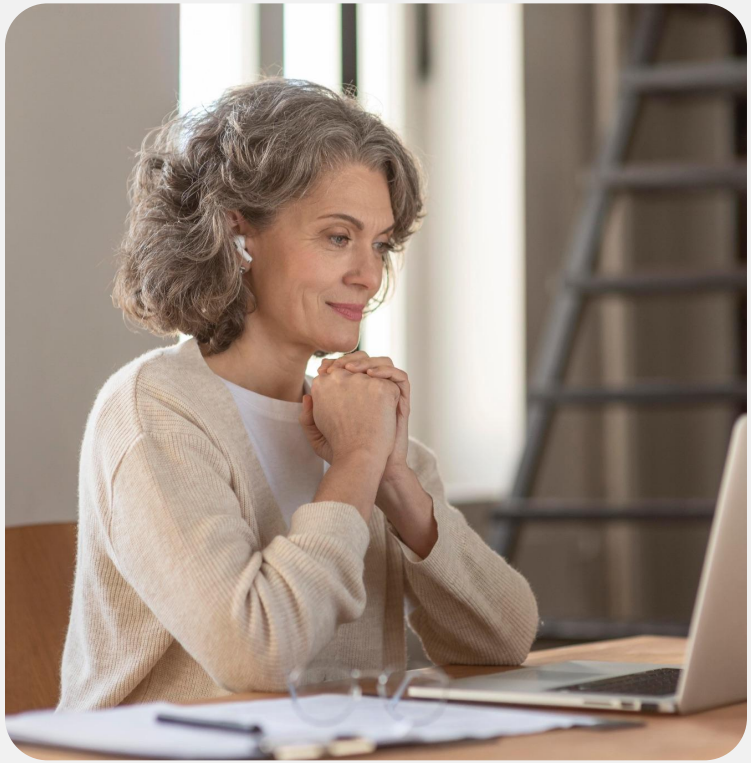


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
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]

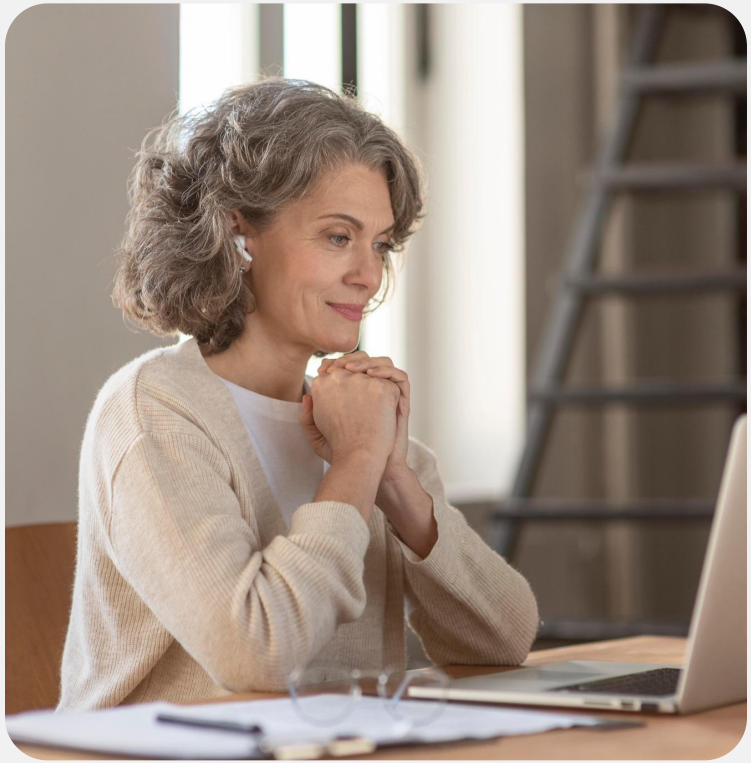


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

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

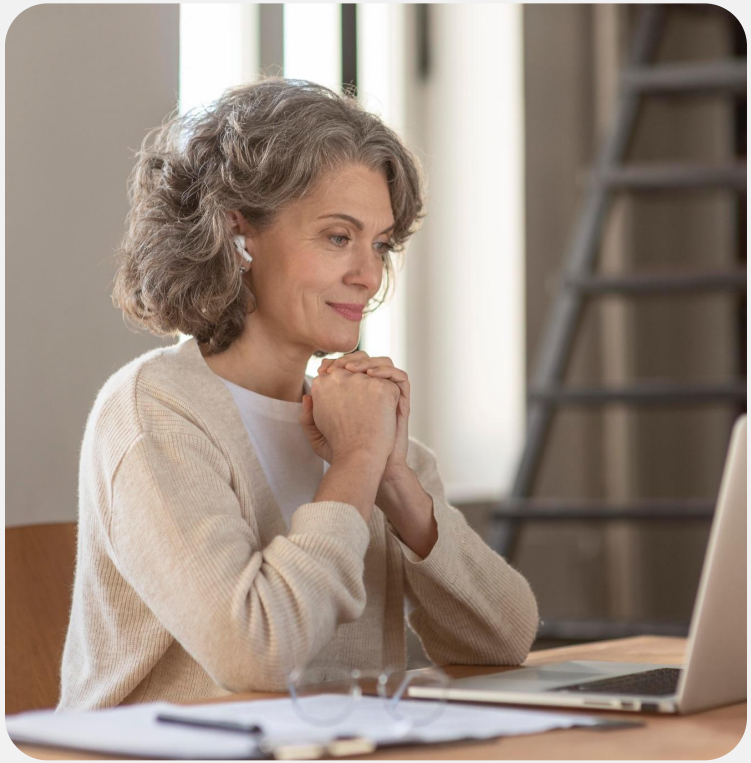
a. [10, 20, 30]
b. [10, 20, 30, 30]
c. [10, 20, 30, 30, 40]
d. [10, 20, 30, 40, 30]



Выберите правильный вариант ответа

Что возвращает функция id() в Python?

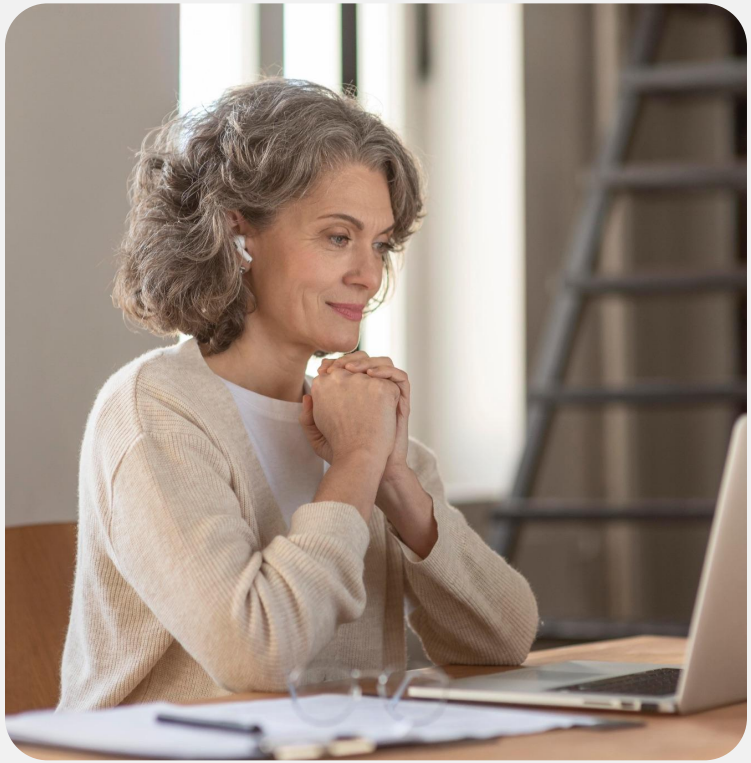
a. Тип объекта.
b. Длину объекта.
c. Уникальный идентификатор объекта в памяти.
d. Значение объекта.



Выберите правильный вариант ответа

Что возвращает функция id() в Python?

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

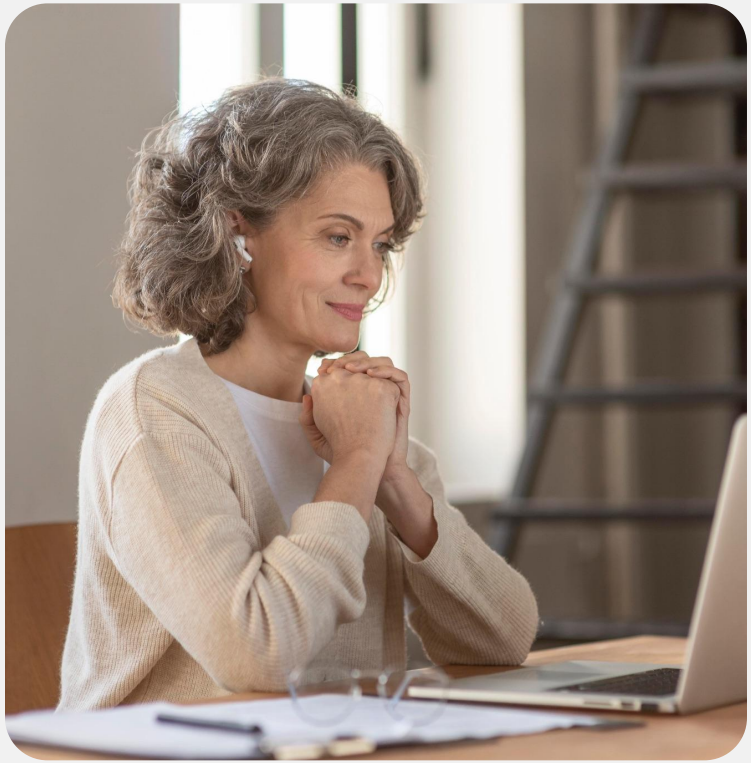


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

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

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



Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

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

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

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Вложенные коллекции

Полезно знать

ICH
IT CAREER HUB

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

```
# 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)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Доступ к вложенным элементам

Доступ к вложенным элементам

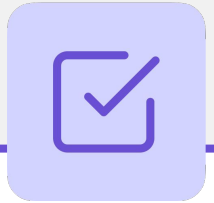
Пояснение

Для доступа к элементам вложенного списка используется несколько уровней индексации.

Пример списка

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

Доступ к вложенным элементам



Для доступа к элементам вложенного списка используется несколько уровней индексации.

Доступ к элементам первого уровня

```
# Пример списка с вложенными элементами

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

# Доступ к элементам первого уровня осуществляется через один уровень индекса.

print(list_elements[0])  # Вывод: [1, 2, 3] (первый вложенный список)
print(list_elements[1])  # Вывод: [4, 5] (второй вложенный список)
print(list_elements[2])  # Вывод: 6 (отдельный элемент)
print(list_elements[3])  # Вывод: [7, [8, [9], 10]] (вложенный список с дополнительной
# вложенностью)
```

Доступ к элементам второго уровня

```python
# Пример списка с вложенными элементами

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

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

Чтобы получить доступ к элементам вложенных списков, нужно добавить **новый уровень индексации** на **каждый уровень вложенности**.

Сохранение во временные переменные

```
# Пример списка с вложенными элементами
list_elements = [[1, 2, 3], [4, 5], 6, [7, [8, [9], 10]]]

# Сохраняем первый вложенный список во временную переменную
first_nested_list = list_elements[0]
# Доступ ко второму элементу через временную переменную
second_element = first_nested_list[1]
# В результате будет получен тот же результат, что и при обращении по нескольким индексам
print(second_element)
```

Для того чтобы сохранить промежуточный результат на каждом уровне вложенности, можно разделить доступ к элементам на шаги, сохраняя каждый вложенный список в отдельную переменную.

Примеры глубокой вложенности

```
# Пример списка с вложенными элементами
list_elements = [[1, 2, 3], [4, 5], 6, [7, [8, [9], 10]]]

# Получение числа 10
print(list_elements[3][1][2])  # Доступ к элементу третьего уровня вложенности
# Получение числа 9
print(list_elements[3][1][1][0])  # Доступ к элементу четвертого уровня вложенности
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Изменение элементов вложенных списков

Замена элемента в списке

```
# Пример коллекции
list_elements = [[1, 2], [3, 4], [5, 6]]

# Заменим число 2
list_elements[0][1] = "two"
print(list_elements)
# Заменим список [5, 6]
list_elements[2] = "new"
print(list_elements)
```

Изменяемые объекты, такие как списки, позволяют изменять свои элементы на любом уровне вложенности. Для этого нужно обратиться к нужному элементу через индексы, чтобы получить ссылку на него.

Пример с изменением строки

```
# Пример коллекции
list_elements = [[1, 2], [3, 4], [5, 6]]

# Так так элемент является строкой, к нему можно применять строковые методы

list_elements[0][1] = list_elements[0][1].upper() # Преобразуем строку в верхний регистр

print(list_elements)
```

Изменение элемента списка

Если он сам является изменяемым объектом

```
# Изменим первый элемент вложенного списка, добавив новое значение
list_elements[0].append('new value')  # Добавляем новый элемент в первый вложенный список
print(list_elements)
```

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

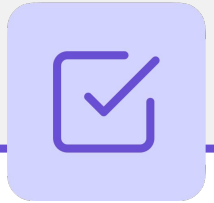
IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Итерация по вложенным коллекциям

Итерация по вложенным коллекциям



Когда вы работаете с вложенными коллекциями, такими как списки, вам часто нужно перебирать все элементы, в том числе те, которые находятся внутри вложенных структур. Для этого можно воспользоваться вложенными циклами `for`.

Итерация по элементам первого уровня вложенности

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

Это обычная итерация по верхнему уровню коллекции

Итерация по элементам нескольких уровней вложенности

```
# Внешний цикл перебирает каждый вложенный список
for sublist in list_elements:
    # Внутренний цикл проходит по каждому элементу внутри этих списков
    for item in sublist:
        print(item, end=" ")
```

Чтобы получить доступ ко всем элементам, включая элементы во вложенных списках, можно использовать вложенные циклы `for`.

Итерация с изменением элементов

```
# Увеличение каждого элемента в коллекциях на 1
for i, sublist in enumerate(list_elements):
    for j, item in enumerate(sublist):
        list_elements[i][j] = item + 1
print(list_elements)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Оператор del

Полезно знать

Оператор del

Этот оператор используется для удаления элементов коллекции по индексу, срезу, а также для удаления целых переменных. Он является мощным инструментом для управления памятью, так как может полностью удалять объекты, освобождая ресурсы.

Особенности оператора del

- 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`.

Оператор del

Синтаксис

```
del list[index]

del list[start:end]

del variable
```

Пояснение

- `list[index]` — удаление одного элемента по индексу
- `list[start:end]` — удаление среза элементов
- `variable` — удаление переменной

Примеры

```
# 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]
```

Примеры

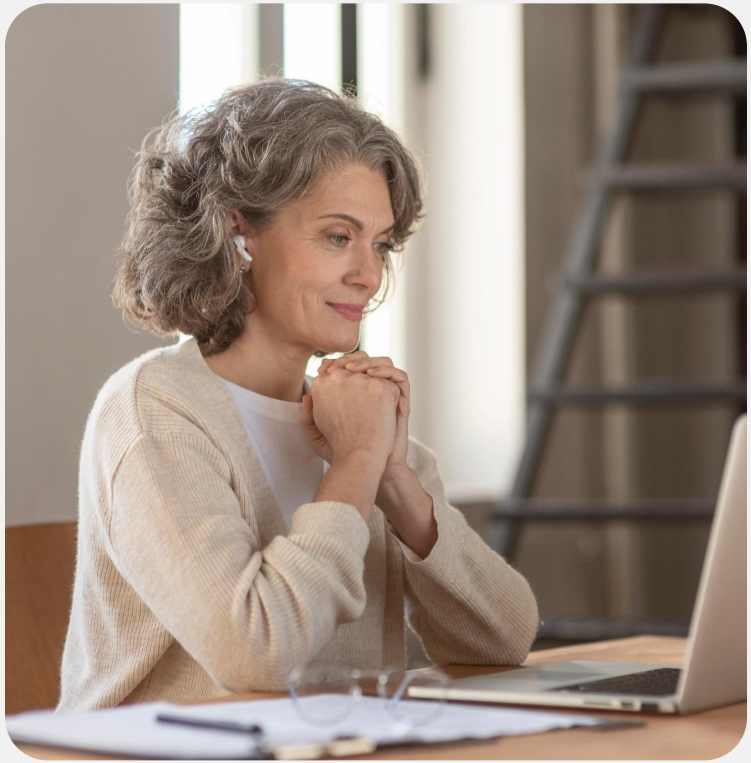
```
# Удаление всех элементов (аналог `clear`)
numbers = [10, 20, 30, 40]
del numbers[:]
print(numbers)  # []

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

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЕ

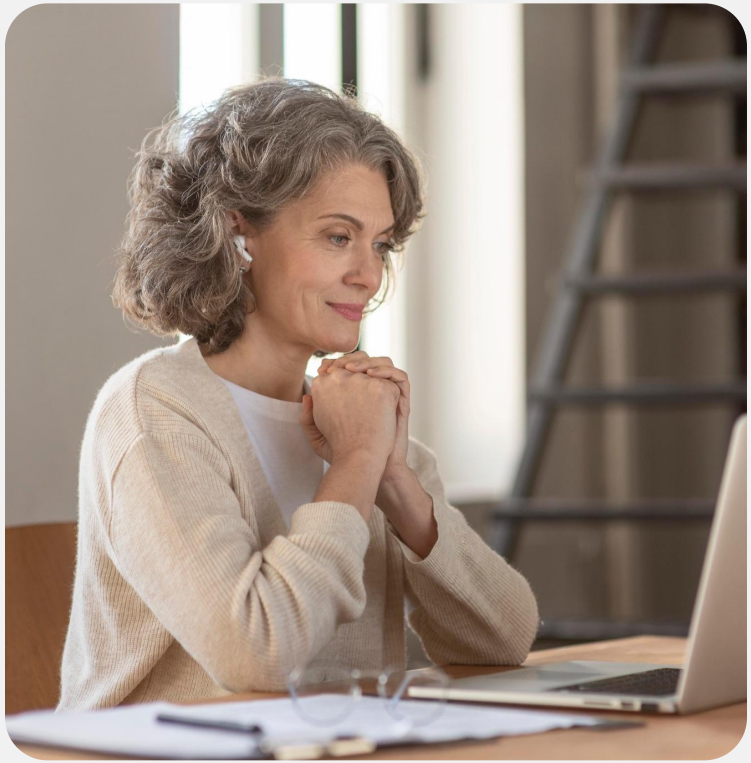


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
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

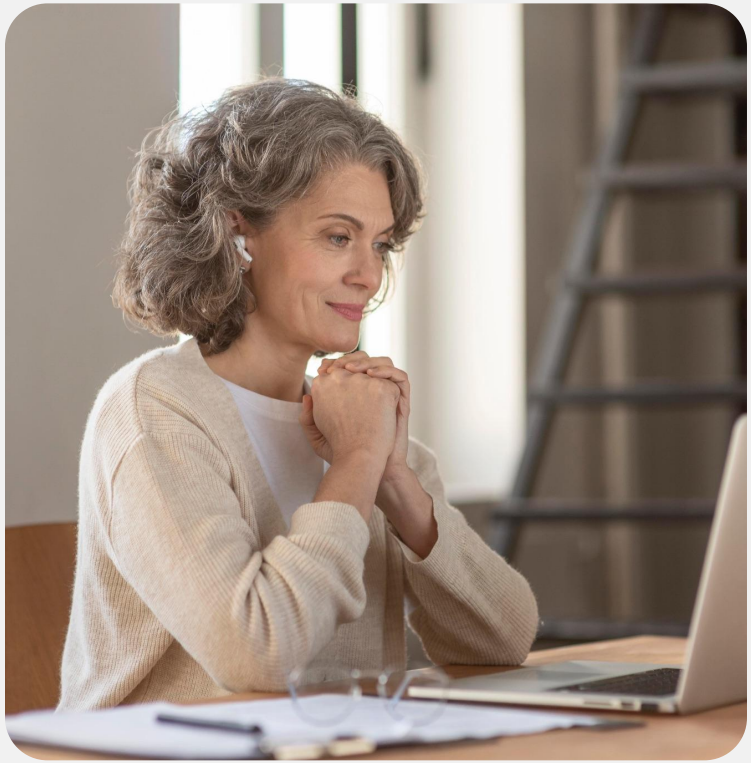


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
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

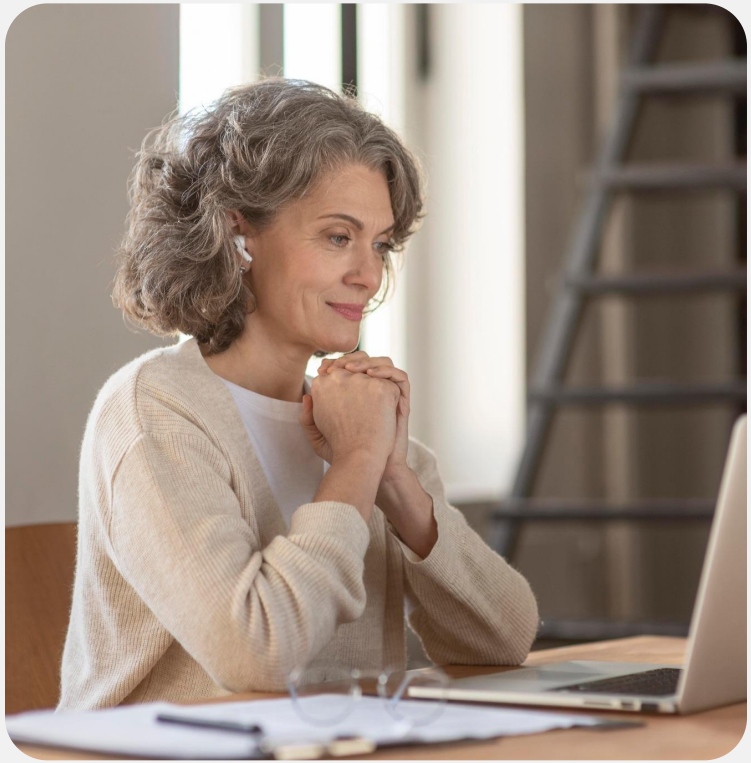


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
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



Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
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

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Копирование списка

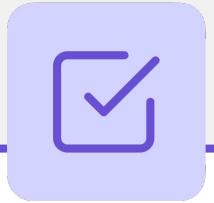
Копирование списка

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

Пример присваивания

```
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

ВОПРОСЫ

ICH IT CAREER HUB

Поверхностное и глубокое копирование

Копирование

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

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

- Метод `copy()`
- Срез `[:]`
- Функция `list()`

Метод copy()

```
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]
```

Срез [:]

```
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]
```

Функция list()

```
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]
```

Копирование списка с вложенными объектами

```
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

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]]
```

Важно

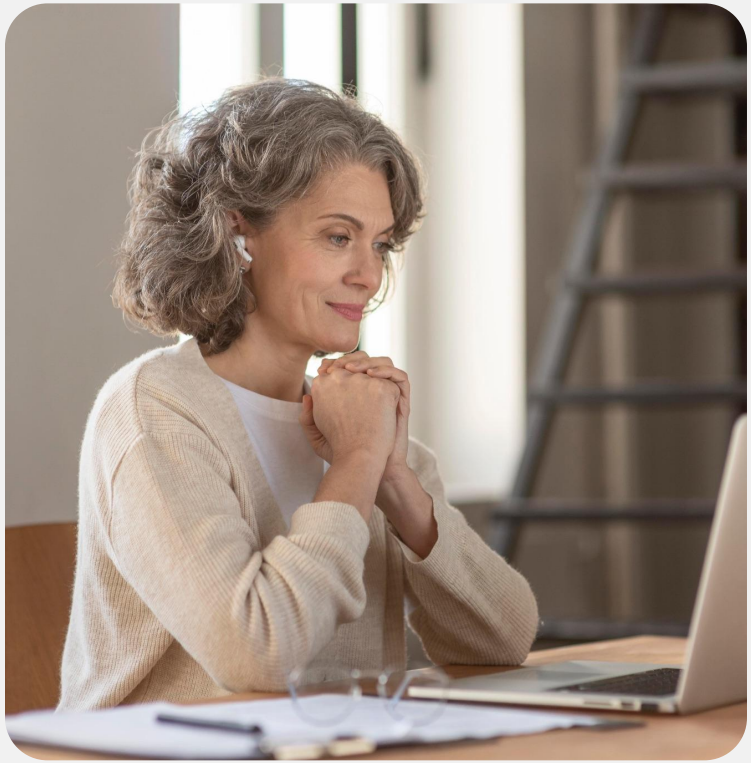
Поверхностное копирование быстрее, так как оно копирует только верхний уровень, но **глубокое копирование** создаёт **полную копию** всех вложенных структур, обеспечивая полную **независимость копии от оригинала**.

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЕ

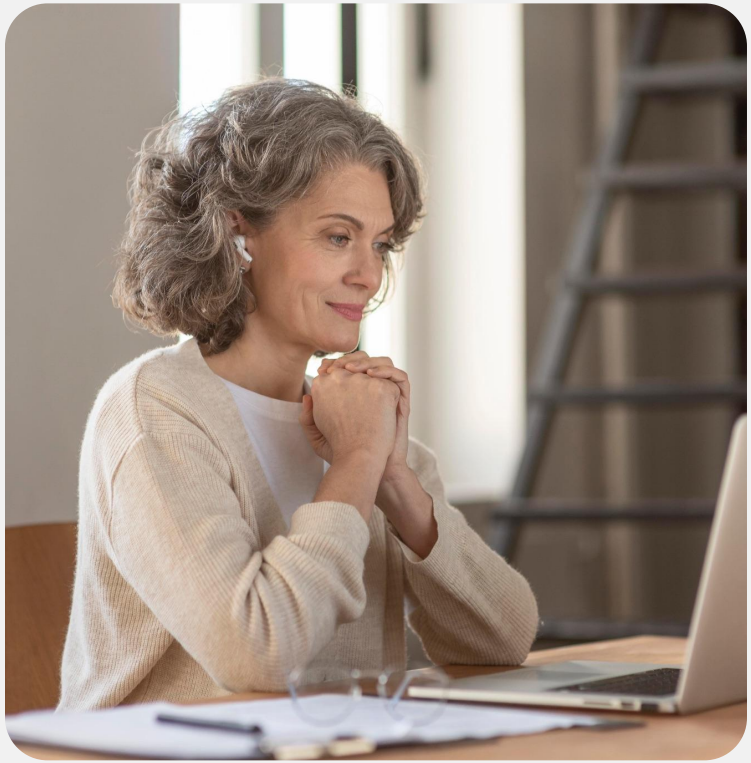


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
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

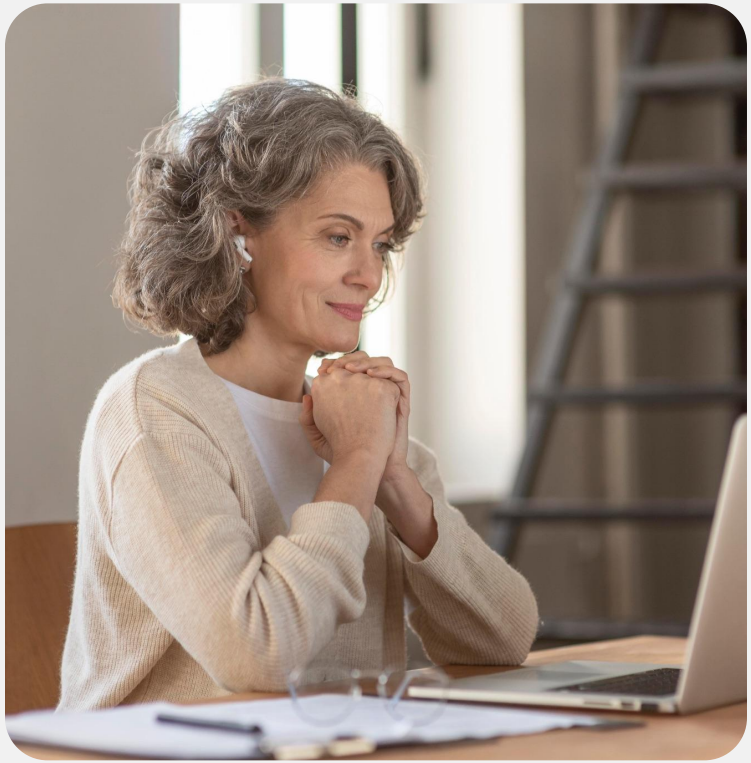


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

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

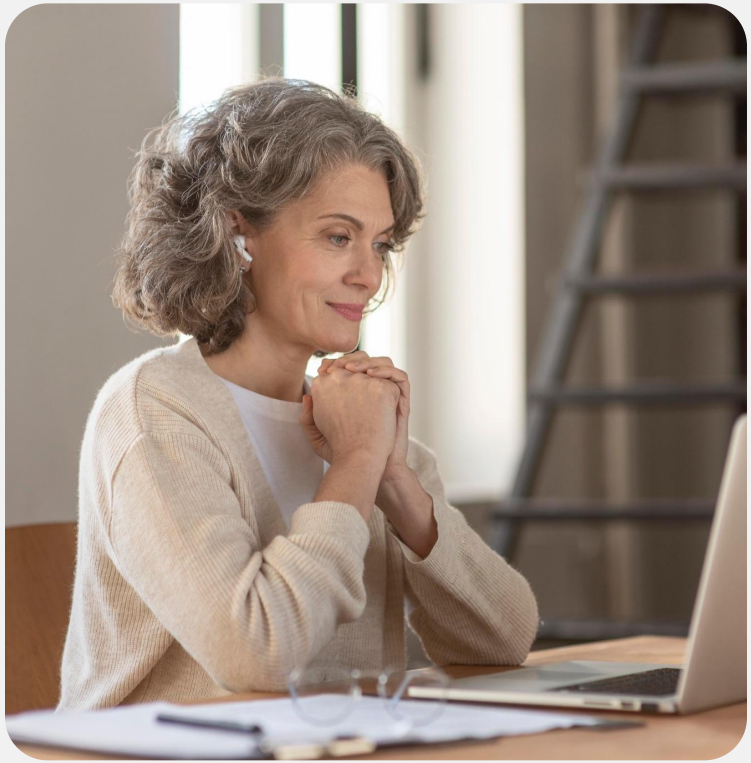
a. `[[1, 2], [3, 4]]`
b. `[[1, 2], [0, 4]]`
c. `[[1, 0], [3, 4]]`
d. Ошибка



Выберите правильный вариант ответа

Какой метод используется для создания поверхностной копии списка?

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



Выберите правильный вариант ответа

Какой метод используется для создания поверхностной копии списка?

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

IT CAREER HUB

ВОПРОСЫ

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:

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

Пример вывода:

```
Исходный список: [[10, 15, 20], [5, 25, 30], [35, 40, 80]]

Глубокая копия после изменений: [[15, 20], [25, 30], [80]]
```

IT CAREER HUB

ВОПРОСЫ

Домашнее задание

1. Одно слово

Напишите программу, которая обрабатывает список из строк и удаляет все строки, содержащие более одного слова, а также преобразует оставшиеся строки к нижнему регистру.

Данные:

```
text_list = ["Hello", "Python Programming", "World", "Advanced Topics", "Simple"]
```

Пример вывода:

```
Обработанный список: ['hello', 'world', 'simple']
```

Домашнее задание

2. Обновление цен товаров

Дан список товаров с ценами. Программа должна применить скидку к каждому товару и добавить в список элемент с новой ценой. В конце вывести таблицу с новой ценой.

Данные:

```
products = [["Laptop", 1200], ["Mouse", 25], ["Keyboard", 75], ["Monitor", 200]]
```

Пример вывода:

Введите скидку (в процентах): 17

```
Товар          Старая цена    Новая цена
Laptop            1200.00$       996.00$
Mouse               25.00$        20.75$
Keyboard            75.00$        62.25$
Monitor            200.00$       166.00$
```

Заключение