IT CAREER HUB

Python

List comprehension. Стек и очередь

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

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

План занятия

- List comprehension
- Функция zip
- Стек и очередь
- Устойчивость сортировки

ICH IT CAREER HUB

ОСНОВНОЙ БЛОК

ICH IT CAREER HUB

List comprehension

Полезно знать

ICH
IT CAREER HUB

List comprehension

(Генератор списка или списковое включение)

Это удобный способ создать **новый список,** применяя выражение к каждому элементу существующего итерируемого объекта (например, списка, кортежа, строки) и/или фильтруя элементы по условию

List comprehension

Синтаксис

```
new_list = [expression for item in iterable]
```

Пояснения

- `expression` — выражение (функция, операция или просто элемент), которое будет применено к каждому элементу. Результат будет добавлен в новый список.
- `item` — переменная для каждого элемента в итерируемом объекте.
- `iterable` — итерируемый объект, по которому выполняется перебор.

Создание списка квадратов чисел

```
numbers = [1, 4, 6, 7, 9]

# Возведение каждого элемента numbers в квадрат

squares = [n ** 2 for n in numbers]

print(squares)

print(numbers)  # Изначальный список останется без изменений
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

List comprehension и цикл for

List comprehension и цикл for

Позволяют создавать и заполнять списки, но делают это по-разному. `List comprehension` обеспечивает более компактный и читаемый синтаксис, а цикл `for` предоставляет больше возможностей для сложной логики.

Вычисление квадратов чисел

```python
# Создание аналогичного списка с помощью List comprehension
print([x ** 2 for x in range(5)])

# Эквивалент с циклом for
squares = []
for x in range(5):
    squares.append(x ** 2)
print(squares)
```

Выражение `x ** 2` в `List comprehension` аналогично выражению `squares.append(x ** 2)` в цикле.

Преобразование верхний регистр

```python
# List comprehension
words = ["hello", "world", "python"]
uppercase_words = [word.upper() for word in words]
print(uppercase_words)

# Эквивалент с циклом for
words = ["hello", "world", "python"]
uppercase_words = []
for word in words:
    uppercase_words.append(word.upper())
print(uppercase_words)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

List comprehension с условием if

List comprehension с условием if

С условием `if` позволяет добавлять элементы в новый список только при выполнении определённого условия.

List comprehension с условием if

Синтаксис

```
new_list = [expression for item in
iterable if condition]
```

Пояснения

- `expression` — выражение (функция, операция или просто элемент), которое будет применено к каждому элементу. Результат будет добавлен в новый список.
- `item` — переменная для каждого элемента в итерируемом объекте.
- `iterable` — итерируемый объект, по которому выполняется перебор.
- `condition` — условие, по которому элементы отбираются.

Выбор только чётных чисел

```python
# List comprehension
even_numbers = [x for x in range(10) if x % 2 == 0]
print(even_numbers)

# Эквивалент с циклом for
even_numbers = []
for x in range(10):
    if x % 2 == 0:
        even_numbers.append(x)
print(even_numbers)
```

Выбор слов, содержащих букву 'a'

```python
# List comprehension
words = ["apple", "banana", "cherry", "date"]
words_with_a = [word for word in words if 'a' in word]
print(words_with_a)
# Эквивалент с циклом for
words = ["apple", "banana", "cherry", "date"]
words_with_a = []
for word in words:
    if 'a' in word:
        words_with_a.append(word)
print(words_with_a)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

List comprehension с условием if-else

List comprehension с условием if-else

С условием `if...else` позволяет создавать новый список, где для каждого элемента применяется условие, и добавляется разное значение в зависимости от выполнения этого условия.

List comprehension с условием if-else

Синтаксис

```
new_list = [expression_if_true if condition else expression_if_false for item in iterable]
```

Пояснения

- `expression` — выражение (функция, операция или просто элемент), которое будет применено к каждому элементу. Результат будет добавлен в новый список.
- `item` — переменная для каждого элемента в итерируемом объекте.
- `iterable` — итерируемый объект, по которому выполняется перебор.
- `condition` — условие для проверки.
- `expression_if_true` — выражение, добавляемое в список, если условие истинно.
- `expression_if_false` — выражение, добавляемое в список, если условие ложно.

Замена нечётных чисел на -1

```python
# List comprehension
numbers = [2, 7, 5, 4, 1, 1, 7, 8]
modified_list = [x if x % 2 == 0 else -1 for x in numbers]
print(modified_list)

# Эквивалент с циклом for
numbers = [2, 7, 5, 4, 1, 1, 7, 8]
modified_list = []
for x in numbers:
    if x % 2 == 0:
        modified_list.append(x)
    else:
        modified_list.append(-1)
print(modified_list)
```

Преобразование коротких слов в верхний регистр

```python
# List comprehension
words = ["cat", "elephant", "dog", "bird"]
modified_words = [word if len(word) > 3 else word.capitalize() for word in words]
print(modified_words)

# Эквивалент с циклом for
words = ["cat", "elephant", "dog", "bird"]
modified_words = []
for word in words:
    if len(word) > 3:
        modified_words.append(word)
    else:
        modified_words.append(word.capitalize())
print(modified_words)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

List comprehension с вложенным условием if-else

List comprehension с вложенным условием if-else

Поддерживает вложенные условия, позволяя добавлять несколько уровней проверки в одном выражении.

List comprehension

Пример

Замена слов на основе условий.

Задача

Если длина слова больше 5 символов, оставить его без изменений. Если длина слова от 3 до 5 символов, заменить слово на 'medium'. Если длина слова меньше 3 символов, заменить его на 'short'.

Выбор слов, содержащих букву 'a'

```
# List comprehension
words = ["hi", "apple", "banana", "cat", "blueberry", "on"]
modified_words = [word if len(word) > 5 else ('medium' if len(word) >= 3 else 'short')
for word in words]
print(modified_words)
# Эквивалент с циклом for
words = ["hi", "apple", "banana", "cat", "blueberry", "on"]
modified_words = []
for word in words:
    if len(word) > 5:
        modified_words.append(word)
    else:
        if len(word) >= 3:
            modified_words.append('medium')
        else:
            modified_words.append('short')
print(modified_words)
```

List comprehension с вложенным условием if-else

Чаще всего `list comprehension` используются с простыми условиями, чтобы сохранить читаемость и простоту понимания кода.

Полезно знать

ICH
IT CAREER HUB

Матрица

Это двумерная структура данных, представленная в виде вложенного списка (списка списков), где каждая строка содержит одинаковое количество элементов.

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

List comprehension с вложенным циклом

List comprehension с вложенным циклом

`List comprehension` поддерживает вложенные циклы, что позволяет создавать списки на основе более сложных структур данных, например, вложенных списков или матриц.

List comprehension с вложенным циклом

Синтаксис

```
new_list = [expression for item1 in
iterable1 for item2 in iterable2]
```

Пояснения

- `expression` —выражение (функция, операция или просто элемент), которое будет применено к каждому элементу. Результат будет добавлен в новый список.
- `item1`, `item2` — переменные для каждого элемента из итерируемых объектов.
- `iterable1`, `iterable2` — итерируемые объекты, по которым выполняется перебор.

Создание списка пар чисел

```
# List comprehension
pairs = [(x, y) for x in range(3) for y in range(2)]
print(pairs)

# Эквивалент с циклом for
pairs = []
for x in range(3):
    for y in range(2):
        pairs.append((x, y))

print(pairs)
```

Распаковка вложенных списков

```python
# List comprehension
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flattened = [num for row in matrix for num in row]
print(flattened)
# Эквивалент с циклом for
flattened = []
for row in matrix:
    for num in row:
        flattened.append(num)
print(flattened)
```

List comprehension

Преимущества

- Краткость и читаемость: Код выглядит более лаконичным и понятным для простых операций.
- Однострочное создание списка: Удобно для компактного представления и передачи результата в функции.

Недостатки

- Ограниченная читаемость для сложных условий: Если в выражении много логики или условий, его труднее читать.
- Не так гибко, как цикл `for`: Для более сложных операций и многошаговых вычислений цикл for более уместен.

Цикл for

Преимущества

- Гибкость: Подходит для выполнения сложных операций и добавления дополнительных условий и логики.
- Более лёгкая отладка: Код, написанный с использованием цикла `for`, легче понять и модифицировать при необходимости.

Недостатки

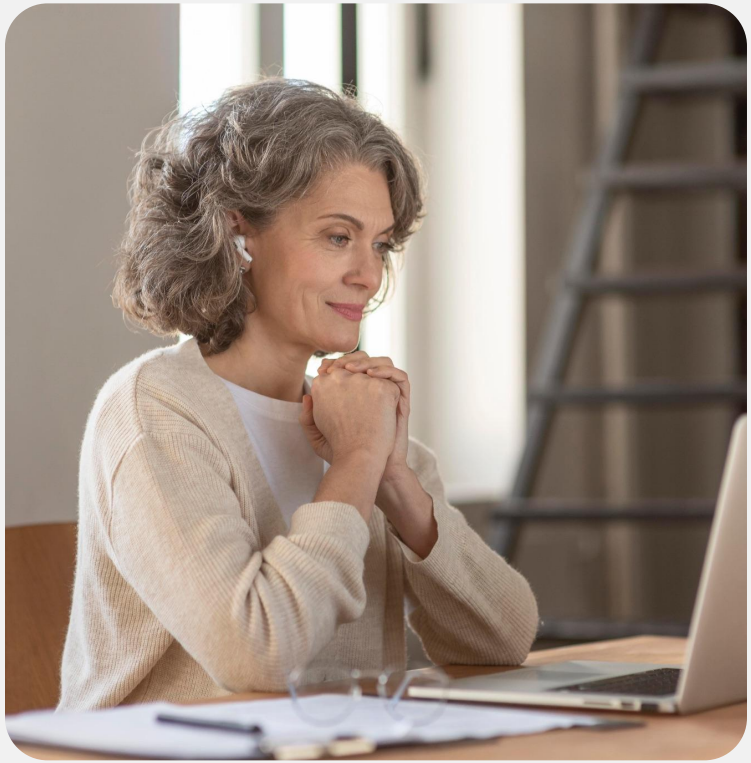
- Многословность: Требуется больше строк кода для выполнения простой операции по сравнению с `list comprehension`.

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЕ

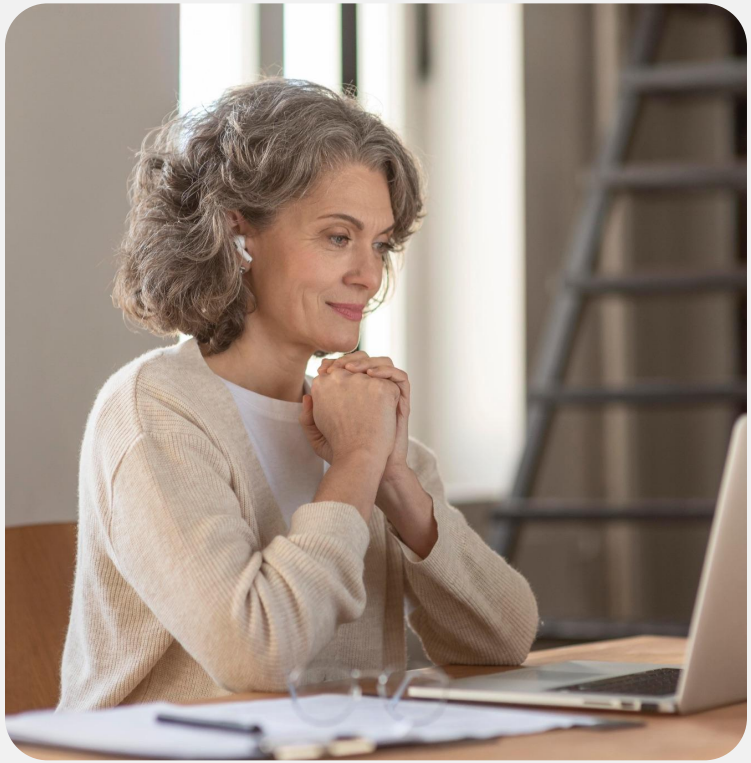


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
ages = [12, 17, 24, 18, 30]
adults = [age for age in ages if age >=
18]
print(adults)
```

a. [12, 17, 18, 24, 30]
b. [24, 18, 30]
c. [12, 17, 24, 18, 30]
d. [12, 17]

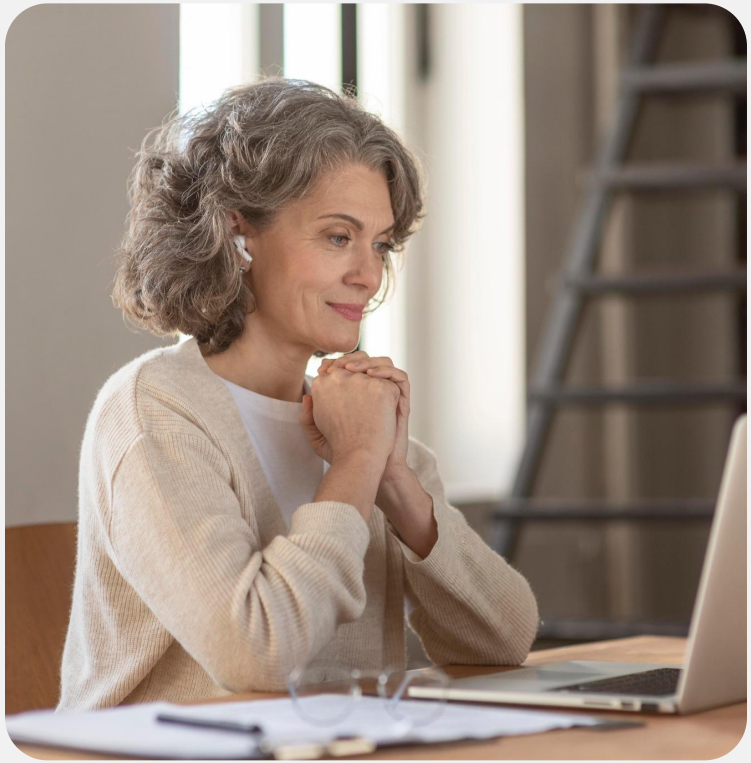


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
ages = [12, 17, 24, 18, 30]
adults = [age for age in ages if age >=
18]
print(adults)
```

a. [12, 17, 18, 24, 30]
b. [24, 18, 30]
c. [12, 17, 24, 18, 30]
d. [12, 17]



Выберите правильный вариант ответа

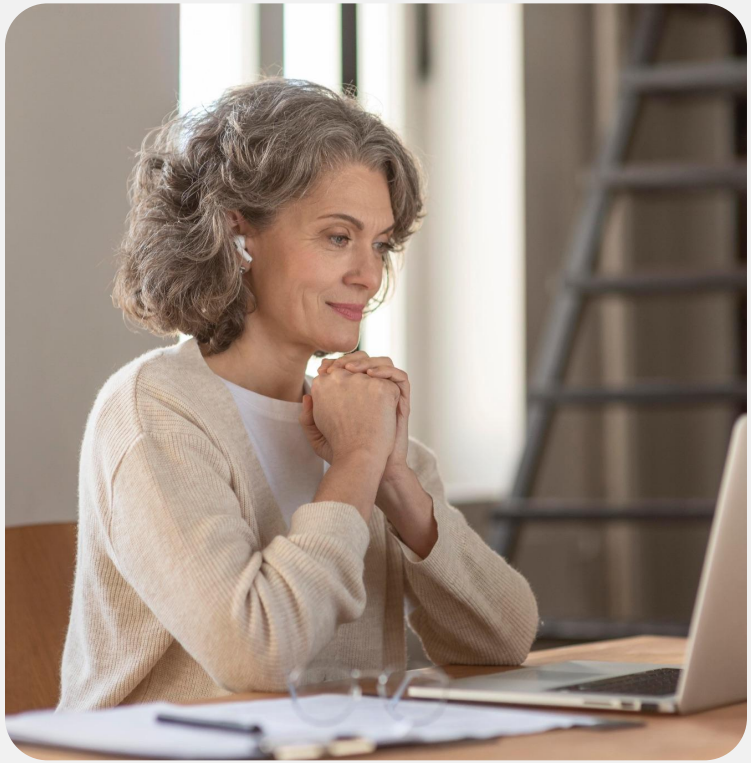
Какой результат будет выведен при выполнении следующего кода?

```
names = ["John", "Anna", "Zoe", "Mark"]

formatted_names = [name.lower() if len(name) > 3
else name.upper() for name in names]

print(formatted_names)
```

a. `['john', 'anna', 'zoe', 'mark']`
b. `['JOHN', 'anna', 'ZOE', 'MARK']`
c. `['john', 'anna', 'ZOE', 'mark']`
d. `['JOHN', 'ANNA', 'zoe', 'MARK']`



Выберите правильный вариант ответа

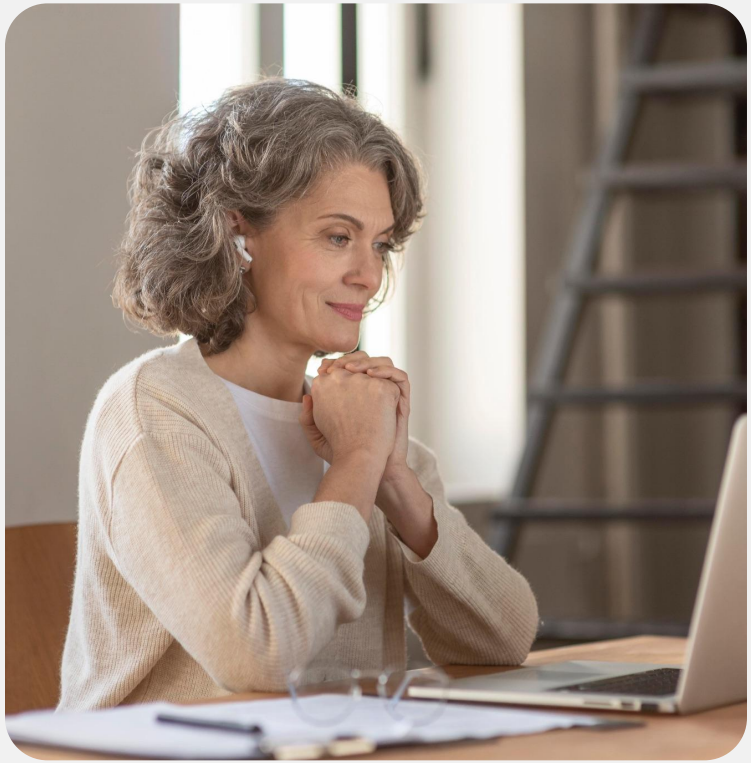
Какой результат будет выведен при выполнении следующего кода?

```
names = ["John", "Anna", "Zoe", "Mark"]

formatted_names = [name.lower() if len(name) > 3
else name.upper() for name in names]

print(formatted_names)
```

a. ['john', 'anna', 'zoe', 'mark']
b. ['JOHN', 'anna', 'ZOE', 'MARK']
c. ['john', 'anna', 'ZOE', 'mark']
d. ['JOHN', 'ANNA', 'zoe', 'MARK']



Выберите правильный вариант ответа

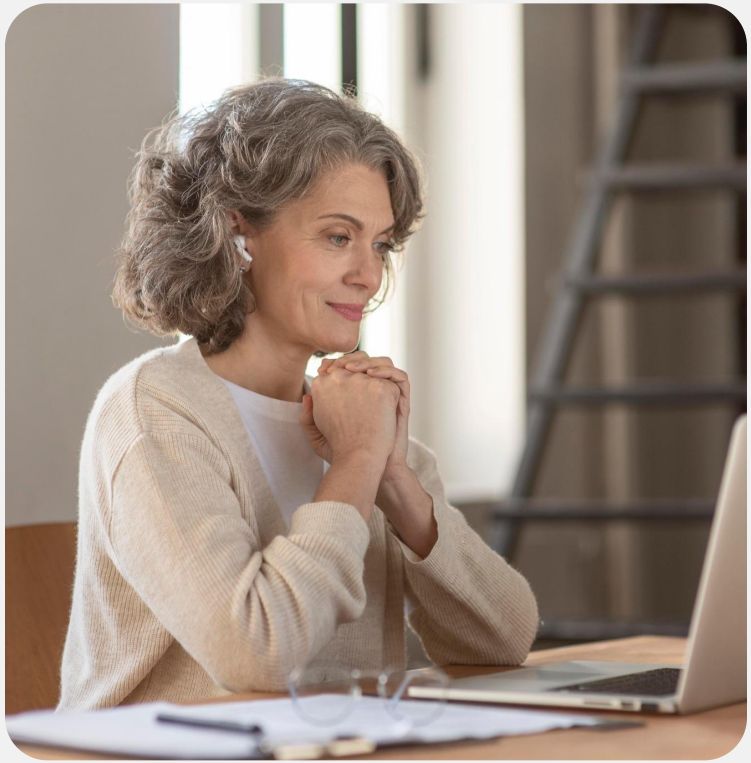
Какой результат будет выведен при выполнении следующего кода?

```
matrix = [[7, 8], [9, 10], [11, 12]]

flattened = [value * 2 for row in matrix
for value in row]

print(flattened)
```

a. [7, 8, 9, 10, 11, 12]
b. [14, 16, 18, 20, 22, 24]
c. [[14, 16], [18, 20], [22, 24]]
d. Ошибка



Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
matrix = [[7, 8], [9, 10], [11, 12]]

flattened = [value * 2 for row in matrix
for value in row]

print(flattened)
```

a. [7, 8, 9, 10, 11, 12]
b. [14, 16, 18, 20, 22, 24]
c. [[14, 16], [18, 20], [22, 24]]
d. Ошибка

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Функция zip

Полезно знать

Функция zip

Позволяет объединять несколько итерируемых объектов (например, списки, кортежи) в один, создавая кортежи из элементов на соответствующих позициях.

List comprehension с вложенным циклом

Синтаксис

```
zip(*iterables)
```

Пояснения

- `*iterables` — любые итерируемые объекты (списки, кортежи, строки и т.д.), которые будут объединены.

Функция возвращает **итерируемый объект** `zip`, который можно преобразовать в коллекцию (например список) или перебрать в цикле.

Примеры использования

```python
# Объединение нескольких итерируемых объектов одинаковой длины
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 35]
cities = ['Hamburg', 'Berlin', 'Munich']
combined = zip(names, ages, cities)
print(combined)
print(list(combined))

# Объединение нескольких итерируемых объектов разной длины
list1 = [1, 2, 3]
list2 = ['a', 'b']
result = zip(list1, list2)
print(list(result))

# Использование в цикле for
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 35]
for name, age in zip(names, ages):
    print(f"{name} is {age} years old.")
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Стек и очередь

Стек и очередь

Это структуры данных, используемые для хранения элементов с определёнными правилами доступа и управления.

Полезно знать

ICH
IT CAREER HUB

Стек (Stack)

Стек работает по принципу **LIFO (Last In, First Out)**, что означает «последним пришёл — первым ушёл». Элементы добавляются и удаляются с одного конца, называемого вершиной стека.

Стек

Примеры использования

- История браузера (возврат на предыдущие страницы).
- Операции отмены (`Ctrl+Z`) в текстовых редакторах.

Основные операции

- Добавление элемента в вершину стека.
- Удаление элемента из вершины стека.

Список list для реализации стека

```
stack = []
# Добавление элементов в стек
stack.append(1)
stack.append(2)
stack.append(3)
stack.append(4)
# Удаление последних элементов из стека
print(stack.pop())
print(stack.pop())
# Текущий стек
print(stack)
```

Полезно знать

Очередь (Queue)

Очередь работает по принципу **FIFO (First In, First Out)**, что означает «первым пришёл — первым ушёл». Элементы добавляются в один конец очереди (в хвост) и удаляются с другого конца (с головы).

Очередь

Примеры использования

- Очередь задач в принтере.
- Обработка запросов в серверных системах.

Основные операции

- Добавление элемента в конец очереди.
- Удаление элемента с начала очереди.

Очередь

Для очереди рекомендуется использовать `deque` из модуля `collections` для повышения производительности.

Пример реализации очереди

```
from collections import deque
queue = deque()
# Добавление элементов в очередь
queue.append(1)
queue.append(2)
queue.append(3)
queue.append(4)
# Удаление первых элементов из очереди
print(queue.popleft())
print(queue.popleft())
# Текущая очередь
print(queue)
```

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Устойчивость сортировки

Полезно знать

ICH
IT CAREER HUB

Устойчивость сортировки

Это свойство алгоритма сортировки сохранять относительный порядок элементов с одинаковыми значениями в исходной последовательности.

Пример устойчивости сортировки

```
# Список строк
words = ["orange", "mango", "apple", "banana", "kiwi", "cherry"]

# Сортировка списка по длине строк
sorted_words = sorted(words, key=len)
for word in sorted_words:
    print(f"{len(word)}: {word}")

#  слова с одинаковой длиной сохранят свой исходный порядок
```

Практическое применение устойчивости

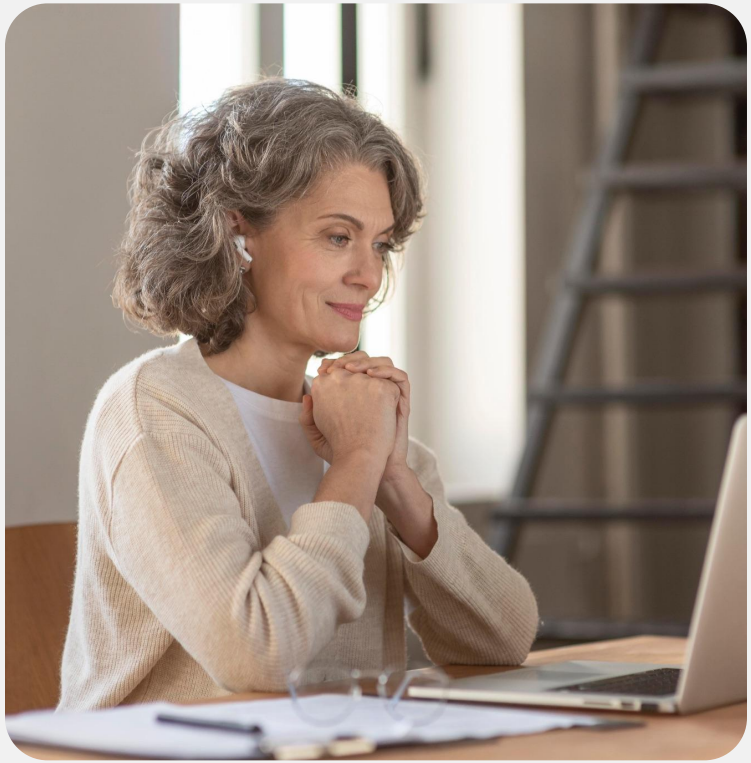
Устойчивость сортировки важна в ситуациях, когда нужно сохранять исходный порядок элементов с одинаковыми значениями для последующей обработки данных.

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЕ

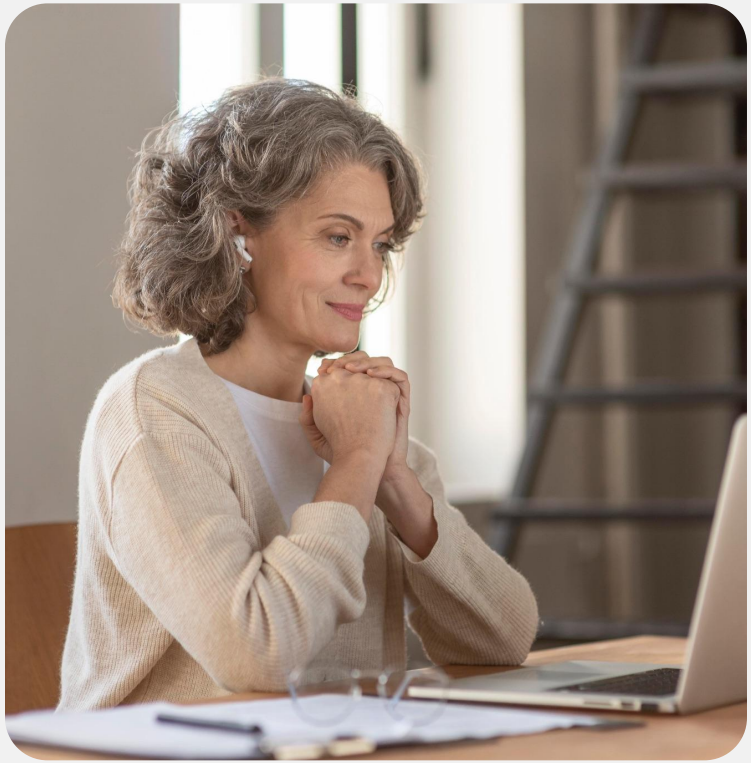


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
list1 = [10, 20, 30]
list2 = [1, 2]
zipped = zip(list1, list2)
print(list(zipped))
```

a. [(10, 1), (20, 2), (30, None)]
b. [(10, 1), (20, 2), (30,)]
c. [(10, 1), (20, 2)]
d. Ошибка

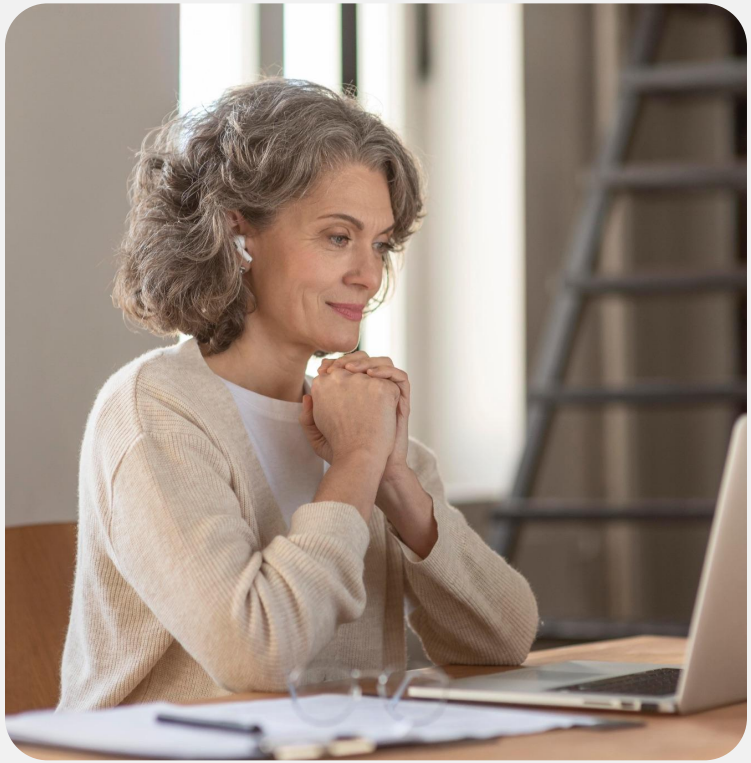


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
list1 = [10, 20, 30]
list2 = [1, 2]
zipped = zip(list1, list2)
print(list(zipped))
```

a. [(10, 1), (20, 2), (30, None)]
b. [(10, 1), (20, 2), (30,)]
c. [(10, 1), (20, 2)]
d. Ошибка

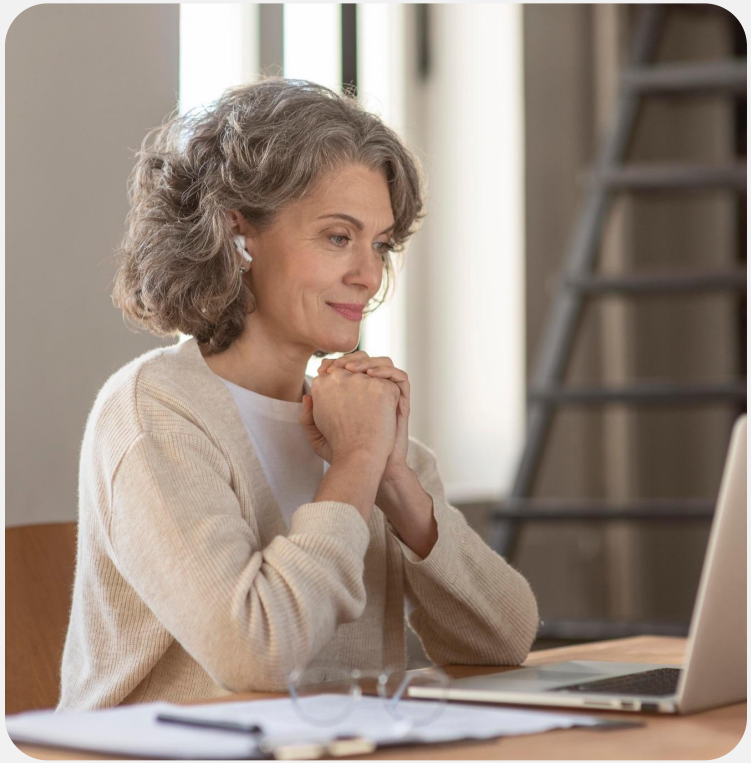


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
from collections import deque
queue = deque()
queue.append(1)
queue.append(2)
queue.popleft()
queue.append(3)
print(queue)
```

a. deque([1, 2, 3])
b. deque([2, 3])
c. deque([1, 3])
d. deque([3, 2])

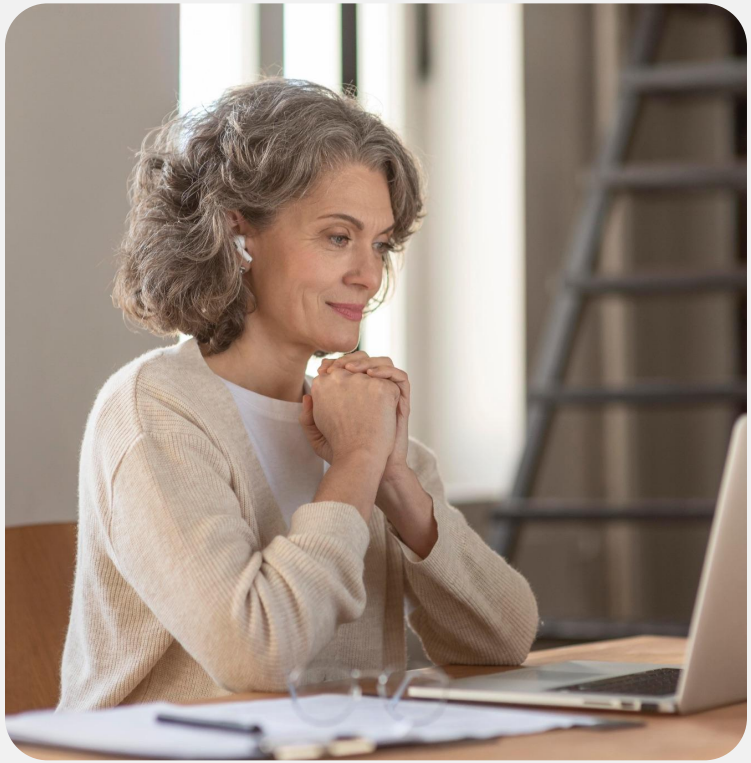


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
from collections import deque
queue = deque()
queue.append(1)
queue.append(2)
queue.popleft()
queue.append(3)
print(queue)
```

a. deque([1, 2, 3])
b. deque([2, 3])
c. deque([1, 3])
d. deque([3, 2])

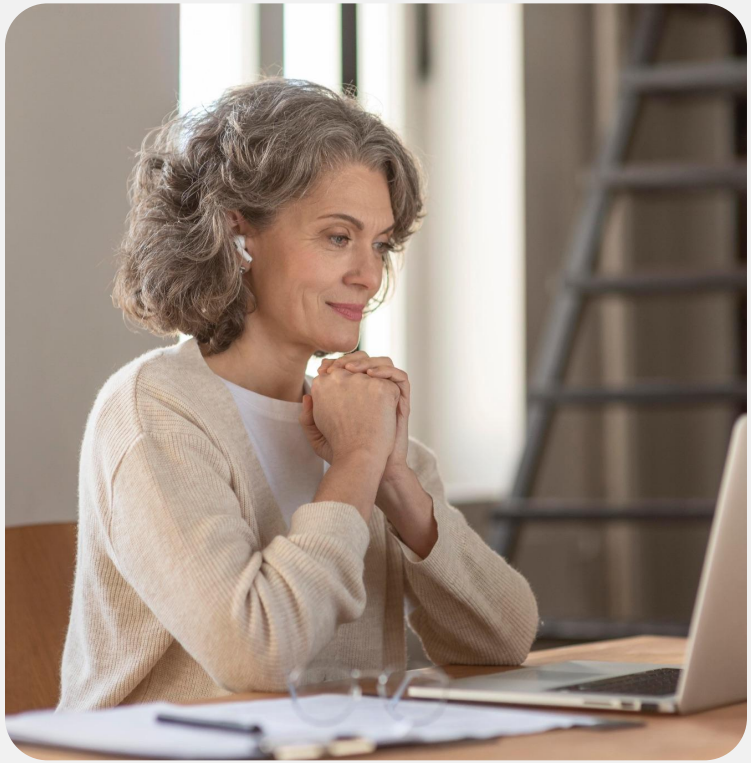


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
words = ["dog", "bat", "cat", "apple"]
sorted_words = sorted(words, key=len)
print(sorted_words)
```

a. ['dog', 'bat', 'cat', 'apple']
b. ['apple', 'bat', 'cat', 'dog']
c. ['bat', 'cat', 'dog', 'apple']
d. Ошибка



Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
words = ["dog", "bat", "cat", "apple"]
sorted_words = sorted(words, key=len)
print(sorted_words)
```

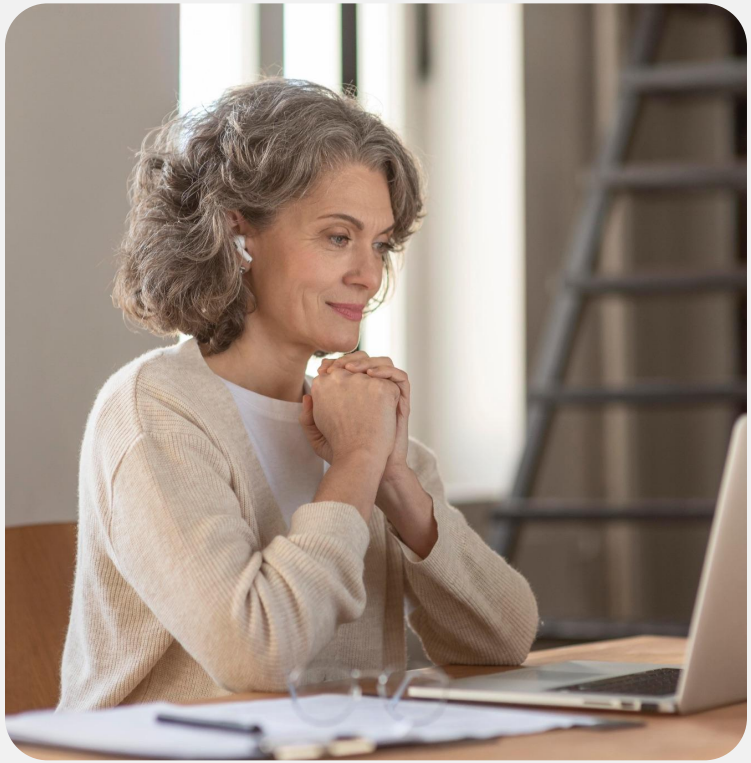
a. ['dog', 'bat', 'cat', 'apple']
b. ['apple', 'bat', 'cat', 'dog']
c. ['bat', 'cat', 'dog', 'apple']
d. Ошибка

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ПРАКТИЧЕСКАЯ РАБОТА



Зеркальные строки больше трех

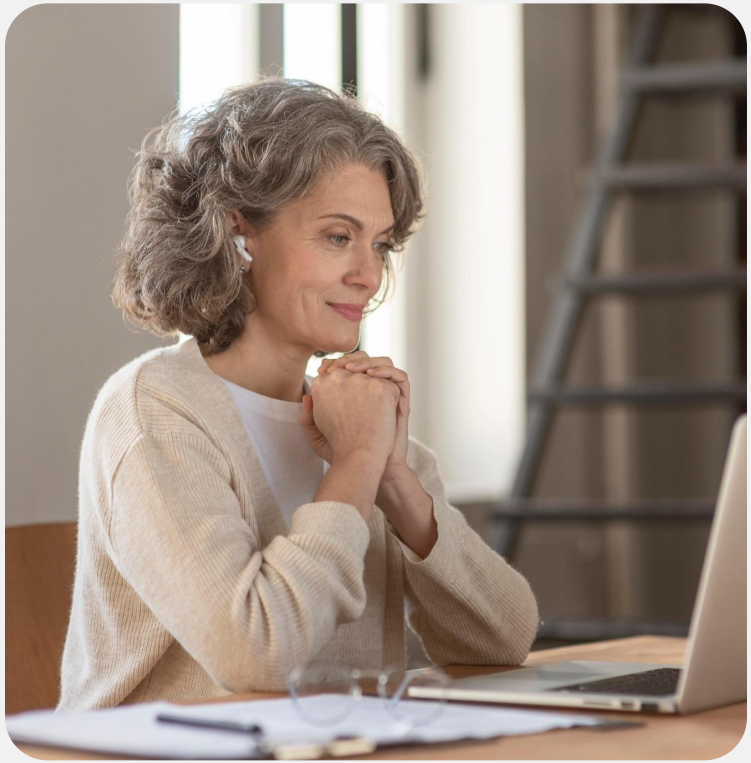
Напишите программу, которая принимает список строк и печатает новый список, в котором содержатся только строки длиной больше 3 символов в перевёрнутом виде.

Данные:

```
words = ["cat", "elephant", "dog", "bird",
"lion", "ant"]
```

Пример вывода:

```
Перевёрнутые слова длиной больше 3 символов:
['tnahpele', 'drib', 'noil']
```



Суммы элементов

Напишите программу, которая принимает двумерный список (матрицу) и создает новый список, содержащий суммы элементов каждой строки.

Данные:

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

Пример вывода:

```
Суммы строк: [6, 15, 24]
```

IT CAREER HUB

ВОПРОСЫ

Домашнее задание

Оценки текстом

Напишите программу, которая преобразует список оценок по системе от 1 до 5 в текстовое представление. Нужно сохранить в списках числовой результат и текстовое представление. Где, 5 — "отлично", 3-4 — "хорошо", а 2 и ниже — "неудовлетворительно".

Данные:

```
grades = [5, 3, 4, 2, 1, 5, 3]
```

Домашнее задание

Правильные скобки

Напишите программу, которая принимает строку, содержащую любые виды скобок — круглые `()`, квадратные `[]` и фигурные `{}`.
Необходимо проверить, правильно ли они расставлены. Скобки считаются правильно расставленными, если:

1. Каждая открывающая скобка имеет соответствующую закрывающую.
2. Скобки закрываются в правильном порядке.

Пример данных:

```
string = "({[}])"
```

Заключение