IT CAREER HUB

Python

Методы списков

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

- [x] Кортеж (tuple)
- [x] Цикл for с кортежами
- [x] Методы кортежа

План занятия

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

ICH IT CAREER HUB

ОСНОВНОЙ БЛОК

IT CAREER HUB

Методы списков

Методы списков

Для работы со списками доступен широкий набор встроенных методов:

- анализа содержимого
- изменения элементов списка
- добавлять новых элементов
- удаления существующих элементов
- сортировки данных

Добавление элементов

Для добавления новых элементов в списки предусмотрены несколько методов, которые позволяют добавлять новые элементы изменяя исходную коллекцию, а не создавая новую.

Добавление элементов

Метод	Описание	Пример использования
`append(item)`	Добавляет один элемент в конец списка	`fruits.append("cherry")`
`extend(iterable)`	Добавляет элементы из другой коллекции по отдельности в конец списка	`numbers.extend([4, 5, 6])`
`insert(index, item)`	Вставляет элемент в список по указанному индексу	`fruits.insert(1, "blueberry")`

Полезно знать

Метод append()

Этот метод добавляет один элемент в конец списка. Один объект (например, строка, число или даже другой список) будет помещён в конец списка как единое целое, даже если это другая последовательность.

Метод append()

Пояснения

Этот метод изменяет исходный список и увеличивает его длину на один.

`item` — элемент, который добавляется в конец списка.

Синтаксис

```
list.append(item)
```

Примеры

```python
# добавление числа
numbers = [1, 2, 3]
numbers.append(4)
print(numbers)

# добавление строки
fruits = ["apple", "banana"]
fruits.append("cherry")
print(fruits)
```

```python
# добавление другой коллекции в список
nested_list = [1, 2, 3]
nested_list.append([4, 5])
print(nested_list)

# добавление нескольких элементов
nested_list = [1, 2, 3]
nested_list.append(4, 5)  # вызовет
# TypeError, так как ожидается один элемент
```

Полезно знать

ICH
IT CAREER HUB

Метод extend()

Этот метод расширяет список, добавляя в него элементы из другой коллекции (например, из списка, кортежа, строки и т.д.). В отличие от метода `append()`, который добавляет один элемент, `extend()` добавляет все элементы итерируемого объекта по отдельности в конец списка.

Метод extend()

Пояснение

Метод изменяет исходный список и добавляет в него элементы один за другим.

`iterable` — любой итерируемый объект (например, список, строка, кортеж, объект `range` и т.д.), элементы которой добавляются в конец списка.

Синтаксис

```
list.extend(iterable)
```

Примеры

```python
# расширение списка другой коллецией
numbers = [1, 2, 3]
numbers.extend([4, 5, 6])
print(numbers)

# расширение списком строк
fruits = ["apple", "banana"]
fruits.extend(["cherry"])
print(fruits)
```

```python
# расширение списка строкой (символы строки добавляются по отдельности)
fruits = ["apple", "banana"]
fruits.extend("cherry")
print(fruits)
# расширение списка другим итерируемым объектом (например, range)
numbers = [0]
numbers.extend(range(1, 5))
print(numbers)
```

Примеры

```
# расширение списка несколькими элементами
numbers = [1, 2, 3]
numbers.extend(4, 5)  # вызовет TypeError, так как ожидается один элемент

# расширение списка неитерируемым объектом
numbers = [1, 2, 3]
numbers.extend(4)  # вызовет TypeError, так как int не является итерируемым объектом
```

Полезно знать

IT CAREER HUB

Метод insert()

Этот метод вставляет элемент в список по указанному индексу. Элементы, находящиеся на позициях справа от вставки, смещаются вправо.

Метод insert()

Пояснение

В отличие от методов `append()` и `extend()`, insert() позволяет добавлять элемент в любое место списка, а не только в конец.

`index` — позиция, в которую будет вставлен элемент.

`item` — элемент, который нужно вставить.

Синтаксис

```
list.insert(index, item)
```

Примеры

```python
# вставка элемента в начало
fruits = ["apple", "banana"]
fruits.insert(0, "blueberry")
print(fruits)

# вставка элемента между другими
numbers = [1, 2, 3]
numbers.insert(1, 4)
print(numbers)
```

```python
# вставка в конец списка с индексом,
превышающего длину списка
animals = ["cat", "dog"]
animals.insert(10, "rabbit")
print(animals)

# вставка другой коллекции
nested_list = [1, 2, 3]
nested_list.insert(2, [4, 5])
print(nested_list)
```

Примеры

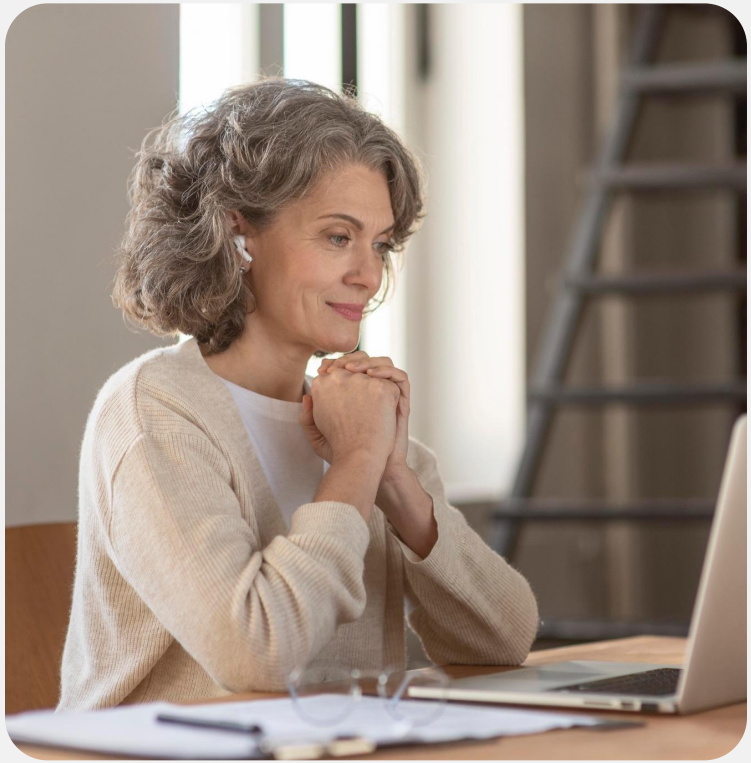
```
# использование отрицательного индекса для вставки
letters = ['a', 'b', 'c']
letters.insert(-1, 'x')
print(letters)

# вставка элемента без указания индекса
numbers = [1, 2, 3]
numbers.insert(5)
# вызовет TypeError, так как ожидается два аргумента: индекс и элемент
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЕ

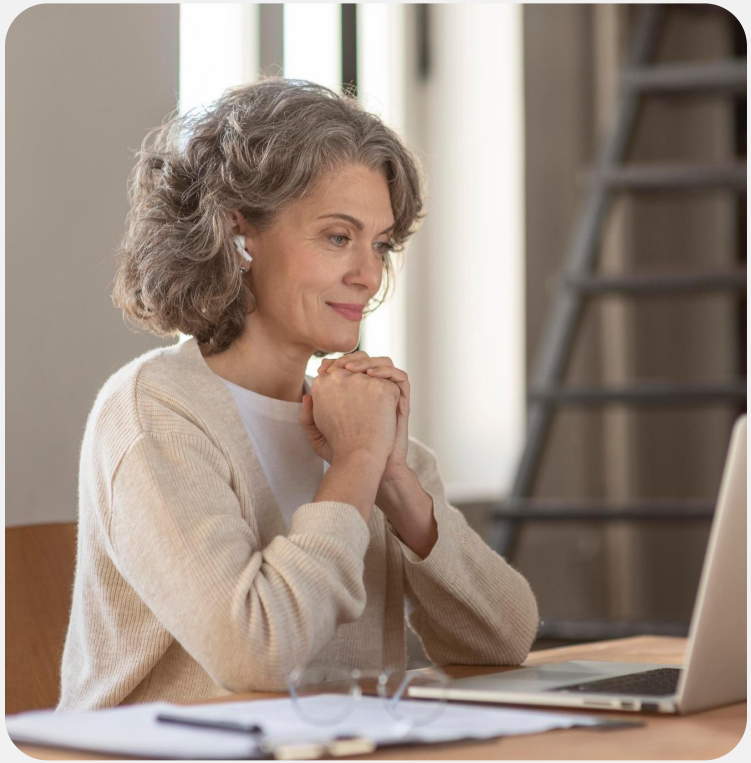


Выберите правильный вариант ответа

Какой результат выведет следующий код?

```
nested_list = [1, 2, 3]

nested_list.insert(1, "new")

print(nested_list)
```

a. [1, "new", 2, 3]
b. [1, 2, "new", 3]
c. ["new", 2, 3]
d. [1, "new", 3]

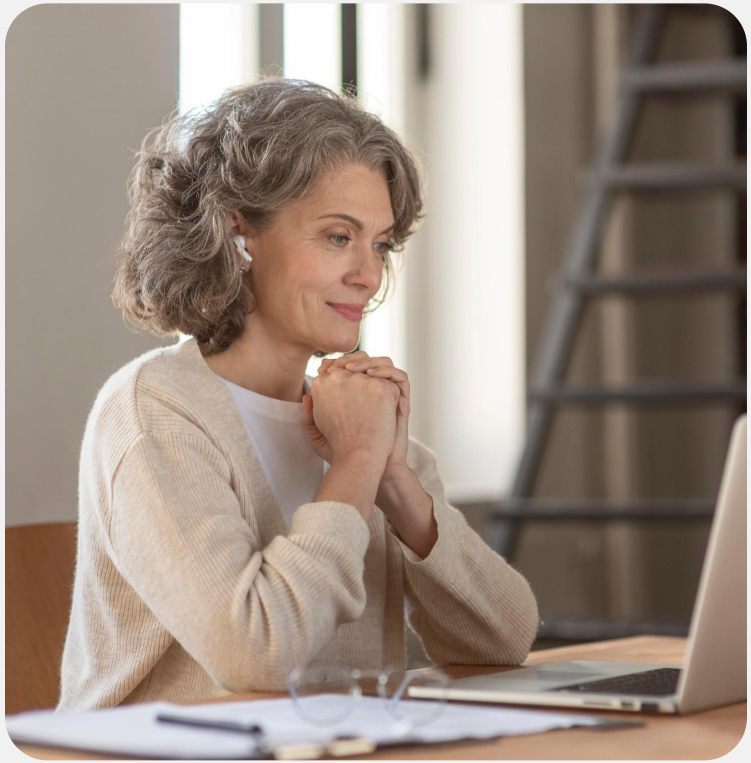


Выберите правильный вариант ответа

Какой результат выведет следующий код?

```
nested_list = [1, 2, 3]

nested_list.insert(1, "new")

print(nested_list)
```

a. [1, "new", 2, 3]
b. [1, 2, "new", 3]
c. ["new", 2, 3]
d. [1, "new", 3]

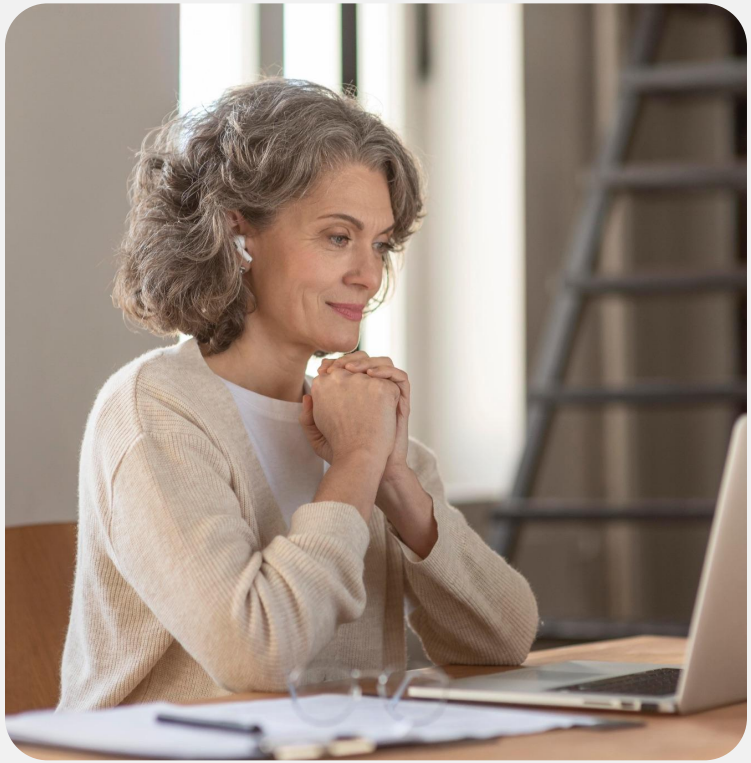


Выберите правильный вариант ответа

Какой результат выведет следующий код?

```
fruits = ["apple", "banana"]

fruits.extend("grape")

print(fruits)
```

a. ["apple", "banana", "grape"]
b. ["apple", "banana", "g", "r", "a", "p", "e"]
c. Ошибка
d. ["apple", "banana", ["grape"]]



Выберите правильный вариант ответа

Какой результат выведет следующий код?

```
fruits = ["apple", "banana"]

fruits.extend("grape")

print(fruits)
```

a. ["apple", "banana", "grape"]
b. ["apple", "banana", "g", "r", "a", "p", "e"]
c. Ошибка
d. ["apple", "banana", ["grape"]]

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Удаление элементов

Полезно знать

IT CAREER HUB

Метод remove()

Этот метод удаляет первый найденный элемент, который равен переданному аргументу.

Метод remove()

Пояснения

- Если такого элемента нет, возникает ошибка `ValueError.`
- Этот метод изменяет исходный список, удаляя только первое вхождение указанного значения.

`item` — значение, которое нужно удалить из списка.

Синтаксис

```
list.remove(item)
```

Примеры

```
# удаление первого вхождения элемента
fruits = ["apple", "banana", "cherry", "banana"]
fruits.remove("banana")
print(fruits)

# повторное удаление первого вхождения элемента
fruits.remove("banana")
print(fruits)

# попытка удалить несуществующий элемент ()
numbers = [1, 2, 3, 4, 5]
numbers.remove(10)  # вызовет ValueError
```

Полезно знать

IT CAREER HUB

Метод pop()

Этот метод удаляет и возвращает элемент с указанного индекса. Если индекс не указан, по умолчанию удаляется последний элемент списка.

Метод pop()

Пояснение

- Метод изменяет список, возвращая удалённый элемент.
- Если указанный индекс выходит за пределы списка, возникает ошибка `IndexError`.

`index` (опционально) — индекс элемента, который нужно удалить.

Синтаксис

```
list.pop(index)
```

Примеры

```
# удаление и возврат последнего элемента
numbers = [10, 20, 30, 40]
last_item = numbers.pop()  # возвращенный элемент можно присвоить
переменной
print(numbers)
print(last_item)

# удаление и возврат элемента по индексу
numbers = [10, 20, 30, 40]
second_item = numbers.pop(1)
print(numbers)
print(second_item)

# попытка удалить элемент по несуществующему индексу
numbers.pop(10)  # вызовет ValueError
```

Полезно знать

IT CAREER HUB

Метод clear()

Этот метод полностью очищает список, удаляя все элементы, но оставляя сам объект списка в памяти.

Метод clear()

Пояснения

Метод изменяет список, делая его пустым.

Синтаксис

```
list.clear()
```

Пример

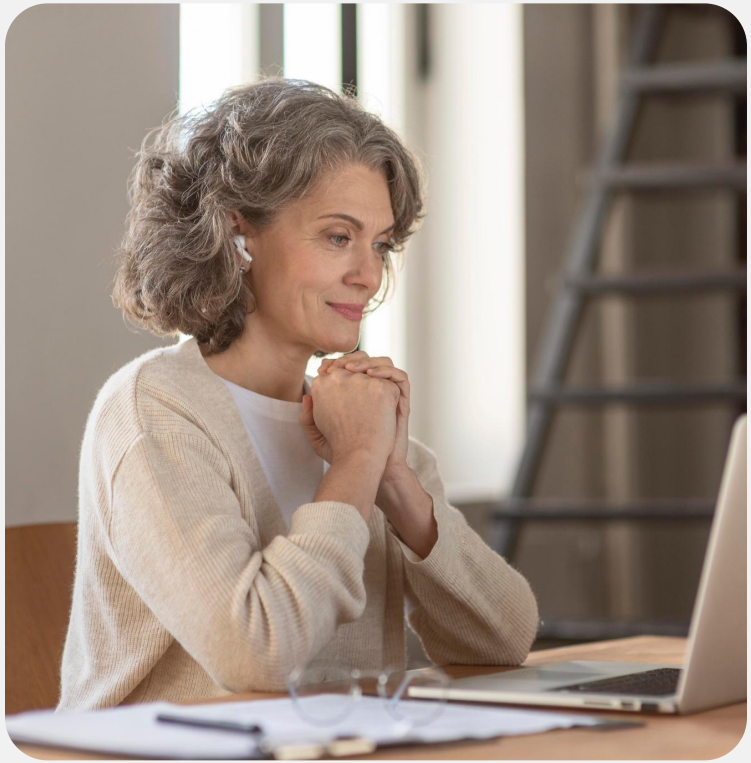
```
# очистка всего списка

fruits = ["apple", "banana", "cherry"]

fruits.clear()

print(fruits)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЕ

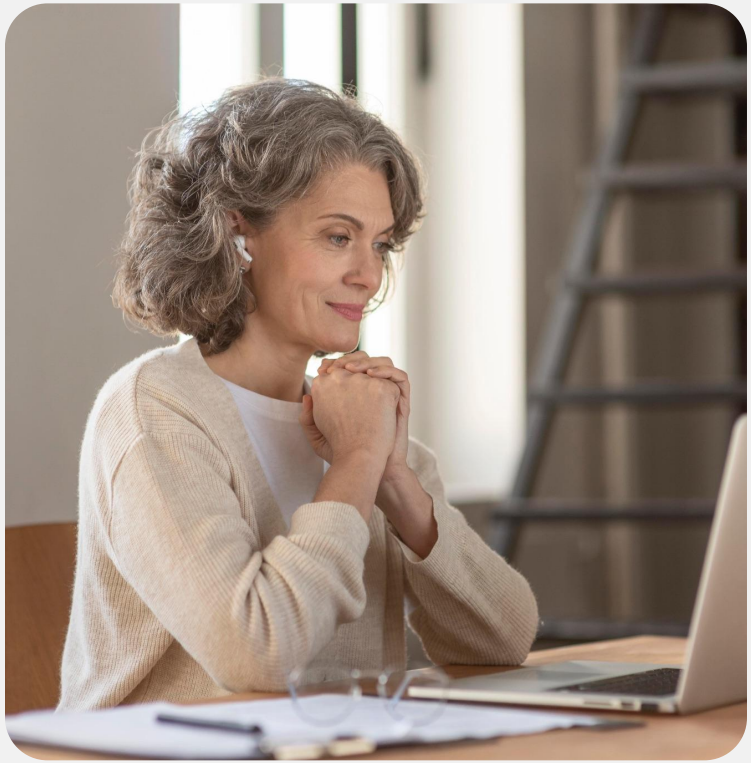


Выберите правильный вариант ответа

Какой результат выведет следующий код?

```
fruits = ["apple", "banana", "cherry",
"banana"]

fruits[1:].remove("banana")

print(fruits)
```

a. ["apple", "cherry", "banana"]
b. ["apple", "banana", "cherry"]
c. ['apple', 'banana', 'cherry', 'banana']
d. ["apple", "cherry"]

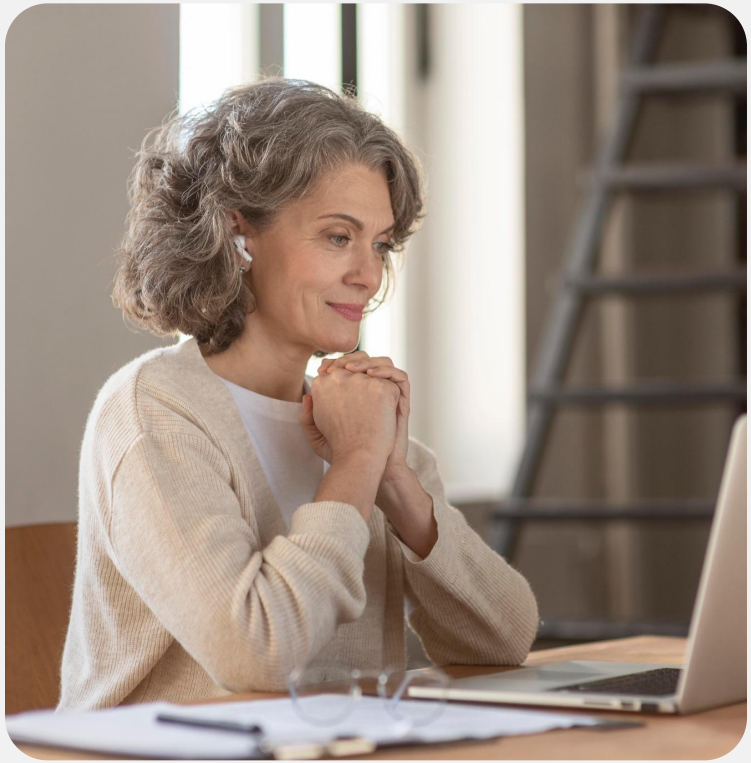


Выберите правильный вариант ответа

Какой результат выведет следующий код?

```
fruits = ["apple", "banana", "cherry",
"banana"]

fruits[1:].remove("banana")

print(fruits)
```

a. ["apple", "cherry", "banana"]
b. ["apple", "banana", "cherry"]
c. ['apple', 'banana', 'cherry', 'banana']
d. ["apple", "cherry"]

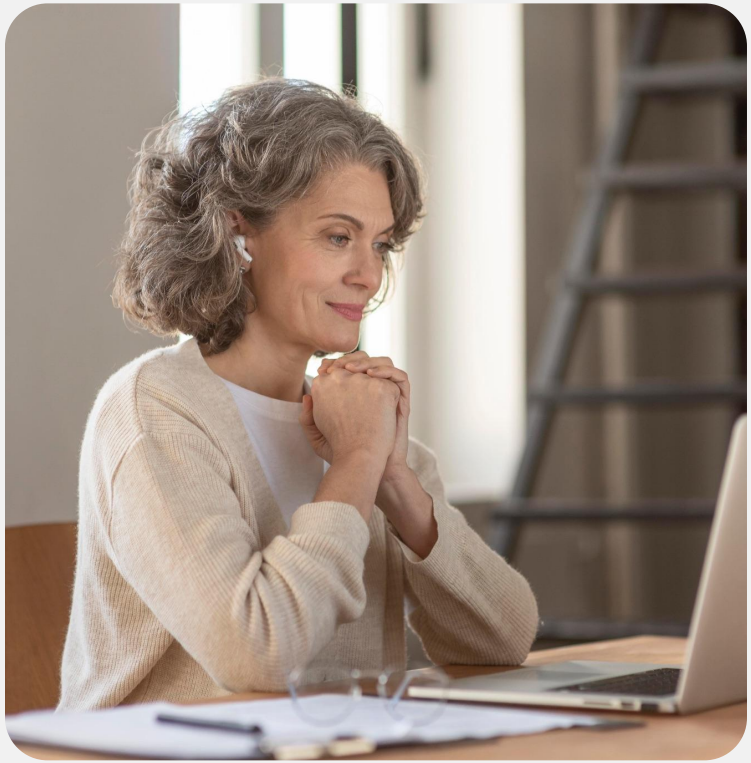


Выберите правильный вариант ответа

Какой результат выведет следующий код?

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

numbers.remove(10)
```

a. [1, 2, 3, 4, 5]
b. Ошибка
c. [1, 2, 3, 4]
d. [1, 2, 3, 5]



Выберите правильный вариант ответа

Какой результат выведет следующий код?

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

numbers.remove(10)
```

a. [1, 2, 3, 4, 5]
b. Ошибка
c. [1, 2, 3, 4]
d. [1, 2, 3, 5]

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Поиск и подсчет элементов

Метод index()

Определение

Метод `index()` возвращает индекс первого вхождения указанного элемента `item` в списке.

Особенности

- Возвращает индекс первого вхождения элемента в список.
- Если элемент не найден в указанном диапазоне (или в списке), будет вызвано исключение `ValueError`.
- Дополнительные параметры `start` и `stop` позволяют ограничить диапазон поиска.

Метод index()

Пояснения

- `item` — элемент, который нужно найти в списке.
- `start` (опционально) — индекс, с которого начинается поиск.
- `stop` (опционально) — индекс, до которого выполняется поиск.

Синтаксис

```
list.index(item, start, stop)
```

Примеры

```
# поиск первого вхождения элемента "banana"
fruits = ["apple", "banana", "cherry", "banana", "cherry"]
banana_index = fruits.index("banana")
print(banana_index)
# поиск первого вхождения "banana", начиная с индекса 2
fruits = ["apple", "banana", "cherry", "banana", "cherry"]
banana_index = fruits.index("banana", 2)
print(banana_index)
# поиск "cherry" в диапазоне индексов от 1 до 4 (не включая 4)
fruits = ["cherry", "banana", "cherry", "banana", "cherry"]
banana_index = fruits.index("cherry", 1, 4)
print(banana_index)
# поиск индекса элемента, которого нет в списке
fruits = ["apple", "banana", "cherry", "banana"]
index = fruits.index("orange")  # вызовет ValueError
```

Метод count()

Определение

Метод `count()` возвращает количество вхождений указанного элемента `item` в списке.

Особенности

Если элементы не найдены, метод возвращает 0.

Метод count()

Пояснения

- `item` — элемент, который нужно посчитать.

Синтаксис

```
list.count(item)
```

Примеры

```
# подсчёт вхождений элемента
fruits = ["apple", "banana", "cherry", "banana"]
banana_count = fruits.count("banana")
print(banana_count)

# подсчёт элементов, которых нет в списке
orange_count = fruits.count("orange")
print(orange_count)
```

Полезно знать

Метод sort()

Этот метод используется для сортировки элементов исходного списка. Его элементы будут упорядочены по возрастанию или убыванию. Метод поддерживает различные параметры, такие как выбор направления сортировки и использование функций для выбора логики сортировки.

Метод sort

Пояснения

- `key` (опциональный параметр) — функция, которая используется для извлечения ключа для каждого элемента.
- `reverse` (опциональный параметр) — если `True`, элементы будут отсортированы в порядке убывания. По умолчанию `reverse=False`

Синтаксис

```
list.sort(key=None, reverse=False)
```

Примеры

```
# Сортировка по возрастанию (по умолчанию)
numbers = [4, 1, 7, 2, 9]
numbers.sort()
print(numbers)

# Сортировка по убыванию с параметром reverse=True
numbers = [4, 1, 7, 2, 9]
numbers.sort(reverse=True)
print(numbers)

# Сортировка строк по алфавиту
fruits = ["banana", "apple", "cherry"]
fruits.sort()  # По умолчанию лексикографическое сравнение
print(fruits)
```

Сортировка с использованием ключа

Параметр `key` позволяет задать функцию, которая извлекает ключ сортировки для каждого элемента.

Нужно передавать незапущенную функцию без скобок `()`. Функция будет запускаться для каждого элемента коллекции.

Примеры

```
# Использование встроенной функции len()**`len()`** для сортировки строк по их длине, а не
# лексикографически.
fruits = ["banana", "apple", "cherry", "blueberry"]
fruits.sort(key=len)
print(fruits)

# Использование встроенной функции max() для сортировки кортежей по максимальному элементу.
tuples = [(3, 6), (1, 7, 9), (12, 5), (1, 3, 7)]
tuples.sort(key=max, reverse=True)
print(tuples)
```

Метод reverse

Определение

Метод `reverse()` используется для разворота списка в обратном порядке и изменяет исходный список.

Синтаксис

```
list.reverse()
```

Примеры

```
numbers = [4, 1, 7, 2, 9]
numbers.reverse()
print(numbers)
fruits = ["apple", "banana", "cherry"]
fruits.reverse()
print(fruits)
```

Функции sorted и reversed

Функции `sorted()` и `reversed()` используются для сортировки и разворота коллекций (например, списков, строк, кортежей), но в отличие от методов `sort()` и `reverse()`, они не изменяют исходную коллекцию.

Вместо этого они возвращают новый отсортированный или развернутый итерируемый объект.

Полезно знать

Функция sorted()

Эта функция возвращает новый отсортированный список на основе итерируемого объекта.

Синтаксис аналогичен методу `sort()`, но первым аргументом передается коллекция для сортировки.

Функция sorted()

Пояснения

- `iterable` — любой итерируемый объект.
- `key` (опциональный) — функция, которая используется для сортировки.
- `reverse` (опциональный) — если `True`, сортировка выполняется в порядке убывания.

Синтаксис

```
sorted(iterable, key=None,
reverse=False)
```

Примеры

```
# Сортировка списка чисел
numbers = [3, 1, 4, 1, 5]
sorted_numbers = sorted(numbers)
print(sorted_numbers)
print(numbers)

# Сортировка строки
letters = "bca"
sorted_letters = sorted(letters)

# Вернет отсортированные символы по отдельности
print(sorted_letters)
print(letters)

# Сортировка кортежа по длине строк в порядке убывания
words = ("apple", "banana", "kiwi")
sorted_words = sorted(words, key=len, reverse=True)
print(sorted_words)
print(words)
```

Полезно знать

ICH
IT CAREER HUB

Функция reversed()

Эта функция возвращает новый итерируемый объект, представляющий элементы исходной коллекции в обратном порядке. Этот объект не является списком — это итерируемый объект, который можно преобразовать в список, строку или кортеж с помощью соответствующих функций.

Функция reversed()

Пояснения

- `iterable` — любой итерируемый объект (например, список, строка, кортеж).

Синтаксис

```
reversed(iterable)
```

Примеры

```
# Разворот списка
numbers = [3, 1, 4, 1, 5]
reversed_numbers = reversed(numbers)  # возвращает не list, а list_reverseiterator
print(reversed_numbers)
list_reversed_numbers = list(reversed_numbers)
# чтобы увидеть элементы, нужно преобразовать в list
print(list_reversed_numbers)

# Разворот строки
word = "hello"
reversed_letters = reversed(word)
print(reversed_letters)
reversed_word = ''.join(reversed_letters)
print(reversed_word)
```

Отличия методов сортировки и реверса

IT CAREER HUB

Характеристика	sort()	sorted()	reverse()	reversed()
Изменяет исходный объект?	Да	Нет	Да	Нет
Возвращаемый результат	None	Новый отсортированный список	None	Новый итерируемый объект
Работает с типами	Списки	Любые итерируемые объекты	Списки	Любые итерируемые объекты

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Функции min, max, sum

Функции min(), max()

Функции `min()` и `max()` возвращают минимальное и максимальное значения среди элементов итерируемого объекта или среди переданных аргументов. Так же как и функции сортировки, они принимают необязательный аргумент `key`, в который передается функция для вычисления критерия сравнения.

Функции min(), max()

Пояснения

- `iterable` — итерируемый объект, из которого будет возвращён минимальный элемент.
- `key=func` — (необязательный) функция, которая вычисляет критерий сравнения для элементов.

Синтаксис

```
min(iterable, key=func)

max(iterable, key=func)
```

Примеры

```python
# Поиск минимального/максимального значения в списке
numbers = [4, 1, 7, 2, 9]
print(min(numbers))
print(max(numbers))

# Поиск минимального/максимального значения среди отдельных аргументов
print(min(3, 5, 2, 8))
print(max(3, 5, 2, 8))

# Поиск минимального/максимального символа
chars = "Python"
print(min(chars))
print(max(chars))

# Использование параметра key для поиска по длине строк
words = ["apple", "banana", "kiwi", "grape"]
print(min(words, key=len))
print(max(words, key=len))
```

Полезно знать

ICH
IT CAREER HUB

Функция sum()

Эта функция возвращает сумму всех элементов в итерируемом объекте. Она принимает опциональный аргумент `start`, который добавляется к итоговой сумме (по умолчанию `start=0`).

Функция sum()

Пояснения

- `iterable` — итерируемый объект (например, список, кортеж).
- `start` (опционально) — начальное значение, к которому суммировать остальные (по умолчанию 0).

Синтаксис

```
sum(iterable, start=0)
```

Примеры

```
# Сумма чисел в списке
numbers = [4, 1, 7, 2, 9]
total_sum = sum(numbers)
print(total_sum)

# Сумма чисел в списке с начальным значением
total_sum_with_start = sum(numbers, 1000)
print(total_sum_with_start)

# Сумма чисел в кортеже
numbers_tuple = (1, 2, 3)
total_sum_tuple = sum(numbers_tuple)
print(total_sum_tuple)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ПРАКТИЧЕСКАЯ РАБОТА

1. Начало равно концу

Напишите программу, которая принимает список строк и создает новый список только из слов, начинающихся и заканчиваются одной и той же буквой.

Данные:

```
strings = ["apple", "banana", "level", "radar", "grape"]
```

Пример вывода:

```
Строки, которые начинаются и заканчиваются одной и той же буквой: ['level', 'radar']
```

2. Слова с гласных

Напишите программу, которая обрабатывает список строки создает новый список, где строки, начинающиеся с согласной буквы будут заканчиваться символом "!". Остальные слова добавьте в том же виде.

Данные:

```
strings = ["apple", "banana", "cherry", "grape", "orange"]
```

Пример вывода:

```
Результат: ['apple!', 'banana', 'cherry', 'grape', 'orange!']
```

IT CAREER HUB

ДОМАШНЕЕ ЗАДАНИЕ

Домашнее задание

1. Число в конце

Напишите программу, которая создает новый список. В него необходимо добавить только те строки из исходного списка, в которых цифры находятся только в конце.

Данные:

```
strings = ["apple23", "ban1ana45", "12cherry", "grape3", "blue23berry"]
```

Пример вывода:

```
Строки с цифрами только в конце: ['apple23', 'grape3']
```

Домашнее задание

2. Удаление кратных

Напишите программу, которая удаляет из списка все значения, кратные числу, которое вводится пользователем.

Данные:

```
numbers = [1, 3, 6, 9, 10, 12, 15, 19, 20]
```

Пример вывода:

```
Введите число для удаления кратных ему элементов: 3

Список без кратных значений: [1, 10, 19, 20]
```

Домашнее задание

3. Порядок четных

Напишите программу, которая формирует новый список чисел. Добавьте в него все элементы исходного списка, где:

- нечетные числа занимают те же позиции,
- чётные числа отсортированы между собой обратном порядке.

Данные:

```
numbers = [5, 2, 3, 8, 4, 1, 2, 7]
```

Пример вывода:

```
Список после сортировки: [5, 8, 3, 4, 2, 1, 2, 7]
```

Заключение