

Python

Итераторы и генераторные выражения



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

-  Введение в работу с файлами. Типы файлов
-  Функция open
-  Режимы работы с файлами
-  Параметр encoding
-  Функции open и close
-  Контекстный менеджер with
-  Чтение из файла и запись в файл
-  Другие популярные режимы
-  Контекстный менеджер с несколькими файлами

План занятия

- Итератор
- Методы `iter` и `next`
- Аналоги магических методов
- Оценка потребления памяти
- Ошибка `StopIteration`
- Как работает цикл `for`
- Итератор и итерируемый объект
- Модуль `itertools`
- Генераторное выражение



ОСНОВНОЙ БЛОК





Итератор



Итерируемый объект

Это объект, который может состоять из множества элементов и предоставляет их по одному



Итератор

Это объект, который позволяет поочерёдно получать элементы коллекции без необходимости загружать их все в память сразу

Итератор содержит



Ссылку на итерируемый объект



Текущую позицию



Логику получения следующего элемента

Особенность итератора



Итератор не изменяет коллекцию, на которую ссылается, а просто управляет процессом последовательного извлечения данных

Использование итератора



Итераторы используются при работе с циклами **for**, ленивыми вычислениями, чтением файлов построчно и в других ситуациях, когда обработка данных должна выполняться эффективно и без лишнего расхода памяти



Методы `iter` и `next`



Магические методы

Итерация в Python основана на магических методах (или **dunder**-методах - сокращение от double underscore) – это специальные методы, которые вызываются автоматически при использовании встроенных механизмов языка, например цикла **for**

Магические методы для работы с итерацией

1

`__iter__()`

создаёт объект итератора из итерируемого объекта

2

`__next__()`

выдаёт следующий элемент итерируемого объекта

Пример использования `__iter__`

```
numbers = [10, 20, 30] # Обычный список

iterator = numbers.__iter__() # Получаем итератор

print(iterator) # Итератор для списка

print(list(iterator)) # Преобразование в список

print(list(iterator)) # Второй раз список будет пустой
```

Особенности `.__iter__`



- `numbers.__iter__()` создаёт итератор, связанный со списком.
- Второй раз список будет пустой, так как итератор "потрачен"

Пример: последовательное получение элементов

```
numbers = [10, 20, 30]

iterator = numbers.__iter__() # Создаём итератор

print(iterator.__next__()) # Получаем нулевой элемент
print(iterator.__next__()) # Получаем первый элемент
print("-" * 30) # Можно прерваться на другие действия
print(iterator.__next__()) # Получаем второй элемент
```

Особенности `.__next__`



- `iterator.__next__()` возвращает следующий элемент
- При каждом вызове итератор запоминает текущую позицию



Аналоги магических методов

Одноименные встроенные функции

1

`iter(obj)`

вызывает `obj.__iter__()`, создавая итератор из итерируемого объекта

2

`next(iterator)`

вызывает `iterator.__next__()`, возвращая следующий элемент

Пример использования встроенных функций

```
numbers = [1, 2, 3] # Итерируемый объект

iterator = iter(numbers) # Вызывает numbers.__iter__()

print(next(iterator)) # Вызывает iterator.__next__()

print(next(iterator)) # Вызывает iterator.__next__()

print(next(iterator)) # Вызывает iterator.__next__()
```



ВОПРОСЫ





Оценка потребления памяти



Функция `sys.getsizeof()`

Эта функция позволяет измерять, сколько памяти занимает объект в байтах. Это полезно для сравнения объектов

Пример: сравнение списка и итератора

```
import sys

# Список из 1 000 000 чисел
numbers_list = [x for x in range(1_000_000)]

print("Размер списка:", sys.getsizeof(numbers_list), "байт")

# Итератор, из списка
numbers_iterator = numbers_list.__iter__()

print("Размер итератора:", sys.getsizeof(numbers_iterator), "байт")
```



Ошибка StopIteration



Ошибка StopIteration

Такая ошибка возникает когда итератор исчерпал все доступные элементы, и метод `__next__()` не может вернуть следующий

Пример

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

iterator = iter(numbers) # Создаём итератор

print(next(iterator))

print(next(iterator))

print(next(iterator)) # Последний элемент

print(next(iterator)) # Ошибка StopIteration
```

Как избежать появления ошибки StopIteration



Объяснение

Функция `next()` поддерживает второй аргумент, который возвращается вместо ошибки `StopIteration`, если элементы в итераторе закончились

Пример

```
numbers = [1, 2, 3]
iterator = iter(numbers)

print(next(iterator))
print(next(iterator))
print(next(iterator))
# None вместо StopIteration
print(next(iterator, None))
# 0 вместо StopIteration
print(next(iterator, 0))
```



ВОПРОСЫ





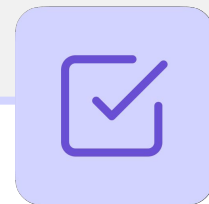
Как работает цикл for



Цикл for

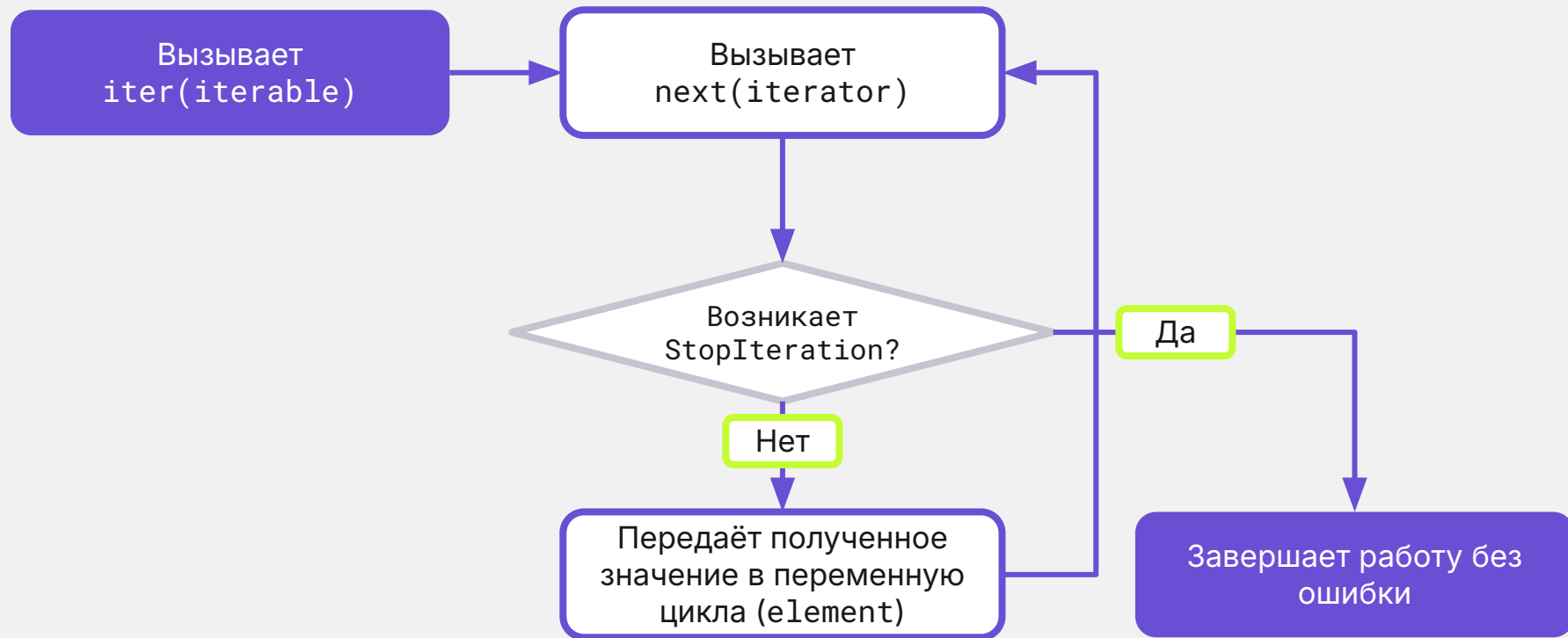
В Python этот цикл используется для перебора итерируемых объектов (списки, строки, итераторы и т.д.)

Важно



Внутри себя `for` автоматически использует итератор, вызывая методы `__iter__()` и `__next__()`

Пошаговая работа цикла for



Пример работы цикла for

```
numbers = [10, 20, 30] # Итерируемый объект  
  
# Цикл for по итерируемому объекту  
for num in numbers:  
    print(num)  
  
print()  
  
iterator = iter(numbers) # Итератор  
  
# Цикл for по итератору  
for num in iterator:  
    print(num)
```

Как Python выполняет этот цикл внутри

```

numbers = [10, 20, 30] # Итерируемый объект
iterator = iter(numbers) # Создание итератора

while True:
    try:
        num = next(iterator) # Получаем следующий элемент
        print(num)
    except StopIteration:
        break # Завершаем цикл при окончании элементов
    
```



ВОПРОСЫ





Итератор и итерируемый объект

Итератор и итерируемый объект



Все итераторы являются итерируемыми объектами, но не все итерируемые объекты являются итераторами

Характеристика	Итерируемый объект (iterable)	Итератор (iterator)
Методы	Реализует <code>__iter__()</code> , который возвращает итератор	Реализует <code>__iter__()</code> и <code>__next__()</code>
Создание	Списки, кортежи, множества, строки, словари	Создаётся с помощью <code>iter(iterable)</code>
Работа с <code>next()</code>	<code>next()</code> вызывать нельзя, вызовет ошибку	<code>next()</code> выдаёт следующий элемент, запоминая позицию
Хранение данных	Хранит все элементы сразу	Не хранит все элементы, а выдаёт их по одному
Перебор	Можно перебирать много раз	Можно перебирать только один раз
Конец данных	Не зависит от <code>StopIteration</code>	При завершении элементов вызывает <code>StopIteration</code>

Особенности итераторов



Экономия памяти



Последовательный доступ



Неизменяемость



Однократное использование



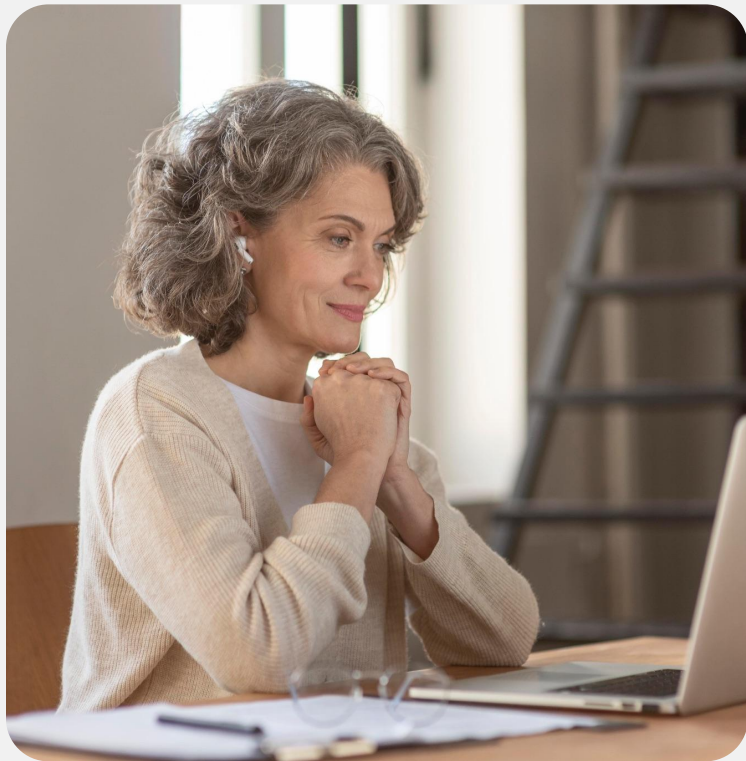
ВОПРОСЫ





ЗАДАНИЯ





Выберите верный вариант ответа

Какие утверждения о итераторах верны?

- a. Итератор можно использовать многократно
- b. Итератор хранит все элементы в памяти
- c. Итератор использует методы `__iter__()` и `__next__()`
- d. Итератор можно создать с помощью `iter(iterable)`



Выберите верный вариант ответа

Какие утверждения о итераторах верны?

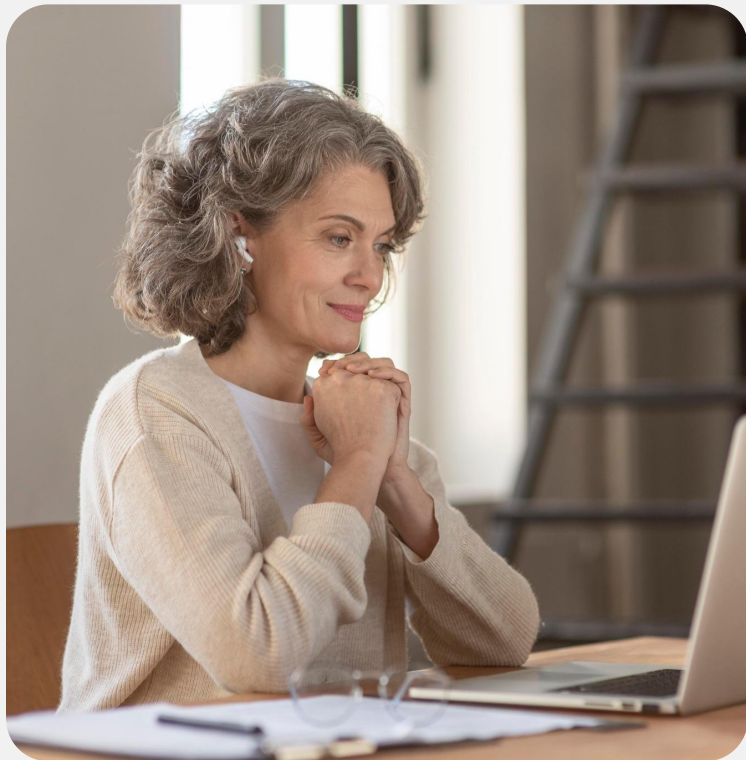
- a. Итератор можно использовать многократно
- b. Итератор хранит все элементы в памяти
- c. Итератор использует методы `__iter__()` и `__next__()`
- d. Итератор можно создать с помощью `iter(iterable)`



Выберите верный вариант ответа

Какое из утверждений о `StopIteration` верное?

- a. `StopIteration` вызывается, когда итератор не может вернуть новый элемент
- b. `StopIteration` вызывается при каждом вызове `next()`
- c. `StopIteration` можно перехватить с помощью `try-except`
- d. `StopIteration` никогда не вызывается автоматически



Выберите верный вариант ответа

Какое из утверждений о `StopIteration` верное?

- a. `StopIteration` вызывается, когда итератор не может вернуть новый элемент
- b. `StopIteration` вызывается при каждом вызове `next()`
- c. `StopIteration` можно перехватить с помощью `try-except`
- d. `StopIteration` никогда не вызывается автоматически



Выберите верный вариант ответа

Какой результат будет у следующего кода?

```
numbers = [5, 10, 15]
iterator = iter(numbers)
for num in iterator:
    print(num)
for num in iterator:
    print(num)
```

- a. 5, 10, 15, 5, 10, 15
- b. 5, 10, 15
- c. 5, 10
- d. 5, 10, 15, Ошибка StopIteration



Выберите верный вариант ответа

Какой результат будет у следующего кода?

```
numbers = [5, 10, 15]
iterator = iter(numbers)
for num in iterator:
    print(num)
for num in iterator:
    print(num)
```

- a. 5, 10, 15, 5, 10, 15
- b. 5, 10, 15
- c. 5, 10
- d. 5, 10, 15, Ошибка StopIteration



Модуль `itertools`



Модуль itertools

Этот модуль предоставляет набор инструментов для работы с итераторами, позволяя создавать эффективные итерируемые последовательности, объединять, фильтровать и изменять потоки данных без лишнего расхода памяти

Бесконечная последовательность чисел: count()

Примеры использования itertools.

```
import itertools

counter = itertools.count(start=1, step=10)

print(next(counter))
print(next(counter))
print(next(counter))
print(next(counter))
print(next(counter))
```

Бесконечное повторение элементов: cycle()

Примеры использования itertools.

```
import itertools

cyclер = itertools.cycle(["A", "B", "C"])

print(next(cyclер))
print(next(cyclер))
print(next(cyclер))
print(next(cyclер)) # A (повторяется)
print(next(cyclер)) # B (повторяется)
```

Объединение итерируемых объектов: chain()

Примеры использования itertools.

```
import itertools

merged = itertools.chain([1, 2, 3], [100, 200, 300])

print(list(merged))
```

Декартово произведение: product()

Примеры использования itertools.

```
import itertools

pairs = itertools.product([1, 2, 3], ["A", "B", "C"])

print(pairs)

print(list(pairs))
```



Функция `itertools.permutations()`

Эта функция создаёт все возможные упорядоченные перестановки элементов. Можно указать длину перестановки (`r`), или использовать всю последовательность по умолчанию

Перестановки элементов: permutations()

Примеры использования itertools.

```
import itertools

letters = ["A", "B", "C"]

# Все возможные перестановки
perms = itertools.permutations(letters)
print(list(perms))

# Все возможные перестановки длины 2
perms = itertools.permutations(letters, 2)
print(list(perms))
```



ВОПРОСЫ





Генераторное выражение



Генераторное выражение

Это способ создания последовательности значений без предварительного вычисления всех элементов. Вместо хранения данных в памяти оно вычисляет каждый следующий элемент только при необходимости

Особенности генераторного выражения



Генераторное выражение возвращает объект генератора, который является итератором и поддерживает метод `__next__()`, позволяя получать элементы по одному.

Оно похоже на списковое включение, но использует круглые скобки (`...`) вместо квадратных [`...`]

Характеристика	Списковое включение [...]	Генераторное выражение (...)
Создаваемый объект	Список (<code>list</code>)	Генератор (<code>generator</code>)
Создание элементов	Все элементы вычисляются сразу и хранятся в памяти	Каждый элемент создаётся только в момент запроса
Использование памяти	Требует хранения всей последовательности	Экономит память, так как не хранит данные
Доступ к элементам	Можно обращаться по индексу (<code>list[0]</code>)	Доступ возможен только через <code>next()</code> или в <code>for</code>
Подходит для	Малых и средних последовательностей	Больших последовательностей и потоковой обработки данных

Синтаксис генераторного выражения

```
generator = (expression for item in iterable)
```

Пример 1: Создание генераторного выражения

Генерирует квадраты чисел от 0 до 4

```
squares = (x ** 2 for x in range(5))
```

```
print(squares) # Объект генератора
```

```
print(next(squares))
```

```
print(next(squares))
```

```
print(next(squares))
```

```
print(next(squares))
```

```
print(next(squares))
```

Особенности создания генераторного выражения



- Возвращает генератор, а не сразу список значений
- `next()` вычисляет следующий элемент только при запросе

Пример 2: Генераторное выражение в for

```
squares = (x ** 2 for x in range(5))
```

```
for num in squares:
```

```
    print(num)
```

Особенности генераторного выражения в for



- Генератор автоматически перебирается в `for`, без явного вызова `next()`
- После прохода по всем элементам он исчерпывается и повторно использовать его нельзя

Пример 3: Потребление памяти

```
import sys
```

```
list_comp = [x**2 for x in range(10**6)] # Списковое включение
```

```
gen_expr = (x**2 for x in range(10**6)) # Генераторное выражение
```

```
print("Размер списка:", sys.getsizeof(list_comp), "байт")
```

```
print("Размер генератора:", sys.getsizeof(gen_expr), "байт")
```

Особенности потребления памяти генераторным выражением



- Список занимает много памяти, так как хранит все элементы сразу
- Генератор почти не занимает памяти, так как создаёт элементы по мере запроса

Использование генераторных выражений



Генераторные выражения можно передавать в аргументы функций, где ожидается итерируемый объект. Это особенно полезно при обработке больших последовательностей

Пример: Генераторное выражение в any() и all()

```
words = ["apple", "Banana", "cherry", "Apricot"]
```

```
print(any(word[0].isupper() for word in words)) # Есть слово с заглавной  
буквы
```

```
print(all(len(word) > 3 for word in words)) # Все слова длиннее 3 букв
```



ВОПРОСЫ





ЗАДАНИЯ



Выберите верный вариант ответа

Какой результат выдаст следующий код?

```
import itertools

letters = ["A", "B", "C"]
cycled = itertools.cycle(letters)

print(next(cycled))
print(next(cycled))
print(next(cycled))
print(next(cycled))
```

- a. A, B, C, A
- b. A, B, C, C
- c. A, B, C, Ошибка StopIteration

Выберите верный вариант ответа

Какой результат выдаст следующий код?

```
import itertools

letters = ["A", "B", "C"]
cycled = itertools.cycle(letters)

print(next(cycled))
print(next(cycled))
print(next(cycled))
print(next(cycled))
```

- a. A, B, C, A
- b. A, B, C, C
- c. A, B, C, Ошибка StopIteration



Выберите верный вариант ответа

Какие утверждения о генераторных выражениях верны?

- a. Генераторное выражение использует круглые скобки
- b. Генератор хранит все элементы в памяти
- c. Генератор можно использовать только один раз
- d. Генераторное выражение сразу создаёт список



Выберите верный вариант ответа

Какие утверждения о генераторных выражениях верны?

- a. Генераторное выражение использует круглые скобки
- b. Генератор хранит все элементы в памяти
- c. Генератор можно использовать только один раз
- d. Генераторное выражение сразу создаёт список

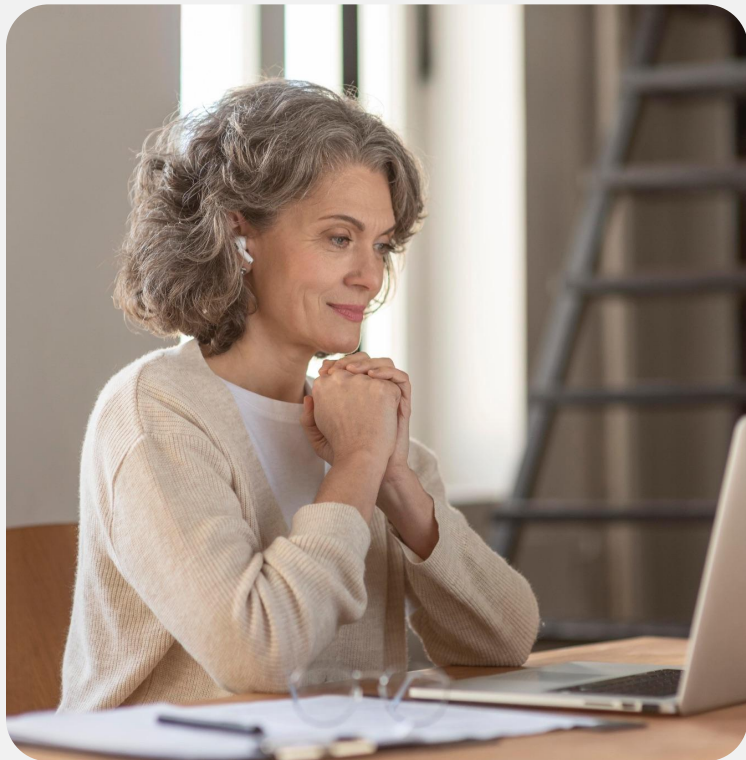


Выберите верный вариант ответа

Что делает следующий код? Перечислите все варианты.

```
with open("data.txt", "a",
encoding="utf-8") as file:
    file.write("Новая строка\n")
```

- a. Перезаписывает файл и добавляет "Новая строка"
- b. Добавляет "Новая строка" в конец файла
- c. Очищает файл перед записью
- d. Создаёт новый файл, если его нет



Выберите верный вариант ответа

Что делает следующий код? Перечислите все варианты.


```
with open("data.txt", "a",
encoding="utf-8") as file:
    file.write("Новая строка\n")
```

- a. Перезаписывает файл и добавляет "Новая строка"
- b. Добавляет "Новая строка" в конец файла
- c. Очищает файл перед записью
- d. Создаёт новый файл, если его нет




ВОПРОСЫ





ПРАКТИЧЕСКАЯ РАБОТА



Генерация безопасных паролей

Программа должна сгенерировать все возможные пароли длиной 4 символа, соблюдая следующие условия:

- Пароль должен содержать хотя бы одну заглавную букву, одну строчную букву и одну цифру.
- Символы не должны повторяться.
- Соседние символы не могут быть расположены подряд в таблице символов.
- Все подходящие пароли записываются в файл `valid_passwords.txt`.

Данные:

```
from string import ascii_lowercase,  
ascii_uppercase, digits
```

Пример данных в файле:

```
acA0  
acA1  
acA2  
acA3  
acA4  
acA5  
acA6  
acA7  
acA8  
...
```



ДОМАШНЕЕ ЗАДАНИЕ



Домашнее задание

1. Фильтрация по ключевому слову

Напишите программу, которая помогает планировать дела. Программа должна бесконечно выводить план на следующий день недели, пока пользователь нажимает 'Enter'.

Данные:

```
# Расписание дел на неделю
weekly_schedule = {
    "Monday": ["Gym", "Work", "Read book"],
    "Tuesday": ["Meeting", "Work", "Study Python"],
    "Wednesday": ["Shopping", "Work", "Watch movie"],
    "Thursday": ["Work", "Call parents", "Play
guitar"],
    "Friday": ["Work", "Dinner with friends"],
    "Saturday": ["Hiking", "Rest"],
    "Sunday": ["Family time", "Rest"]
}
```

Пример ввода:

```
Нажмите 'Enter' для получения плана:
Monday: Gym, Work, Read book
Нажмите 'Enter' для получения плана:
Tuesday: Meeting, Work, Study Python
...
Нажмите 'Enter' для получения плана:
Sunday: Family time, Rest
Нажмите 'Enter' для получения плана:
Monday: Gym, Work, Read book
Нажмите 'Enter' для получения плана: q
```

Домашнее задание

2. Объединение списков продуктов

Напишите функцию, которая принимает несколько списков с названиями продуктов и возвращает **генератор**, содержащий все продукты в **нижнем регистре**. Выведите содержимое генератора.

Данные:

```
fruits = ["Apple", "Banana",
"Orange"]
vegetables = ["Carrot", "Tomato",
"Cucumber"]
dairy = ["Milk", "Cheese",
"Yogurt"]
```

Пример вывода:

```
apple
banana
orange
carrot
tomato
cucumber
milk
cheese
yogurt
```

Домашнее задание

3. Комбинации одежды

Напишите функцию, которая принимает списки типов одежды, цветов и размеров, а затем **генерирует все возможные комбинации** в формате "Clothe - Color - Size".

Данные:

```
clothes = ["T-shirt", "Jeans",  
"Jacket"]  
colors = ["Red", "Blue", "Black"]  
sizes = ["S", "M", "L"]
```

Пример вывода:

```
T-shirt - Red - S  
T-shirt - Red - M  
T-shirt - Red - L  
T-shirt - Blue - S  
...  
Jacket - Black - L
```

Заключение

