IT CAREER HUB

Python

Генераторы

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

- 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

План занятия

- Генератор
- Функция с `yield`
- Ключевое слово `yield`
- Генераторные функции с параметрами
- Использование генератора в `for`
- Бесконечные генераторы
- Метод `close`
- Конструкция `yield from`

ICH IT CAREER HUB

ОСНОВНОЙ БЛОК

IT CAREER HUB

Генератор

Полезно знать

ICH
IT CAREER HUB

Генератор

Это специальный вид итератора, который создаёт элементы по запросу, вместо того чтобы загружать всю последовательность в память

Важно

Все **генераторы** являются **итераторами**, но не все **итераторы** являются **генераторами**

Что используется для создания генераторов

1

Генераторные функции

С использованием ключевого слова `yield`

2

Генераторные выражения

Создание генераторов в круглых скобках `( ... )`

Основные особенности генераторов

- Ленивые вычисления
- Не занимают много памяти
- Сохраняют состояние
- Используют `yield` вместо `return`

Использование генератора

Генераторы удобны для **обработки больших файлов**, **создания бесконечных последовательностей** и **работы с потоками данных**

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Функция с yield

Полезно знать

IT CAREER HUB

Функция с yield

Это функция, которая возвращает объект генератора

Особенности функции с yield

При вызове `next()`

Выполнение функции **приостанавливается на** `yield`

При следующем вызове

Выполнение функции **продолжается с того же места**

ICH IT CAREER HUB

Ключевое слово yield

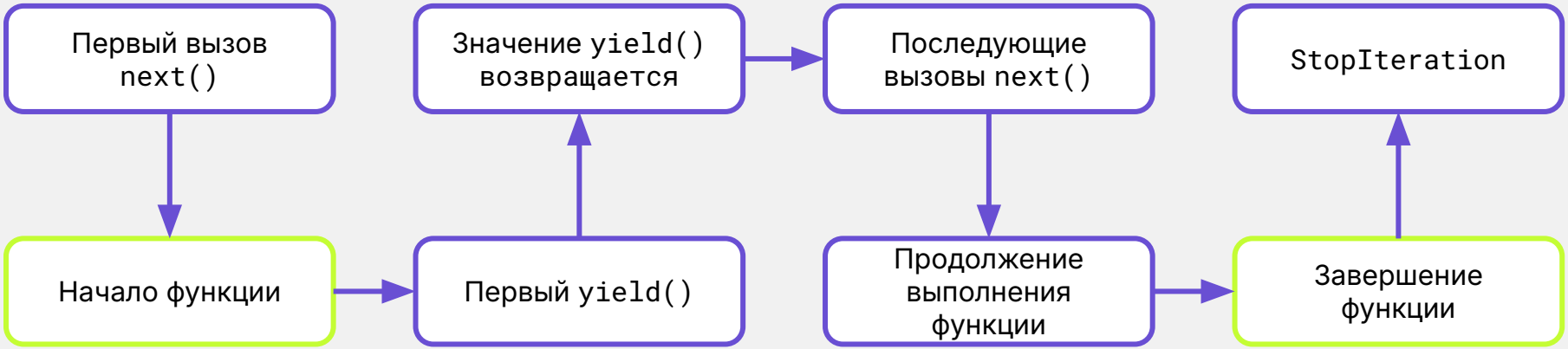
Полезно знать

ICH
IT CAREER HUB

yield

Это ключевое слово, которое используется для приостановки выполнения функции и возврата значения без завершения её работы (в отличие от `return`)

Как работает yield



Как работает yield

Синтаксис

```
def generator():
    yield value
```

Пояснение

- `def generator():` – объявление функции, которая вернет генератор
- `yield` – ключевое слово, приостанавливающее выполнение функции и возвращающее значение
- `value` – результат, который будет передан при вызове `next()`

Пример генератора с yield

```
def generate_values():
    print("Начало работы")
    yield 1  # Приостанавливаем выполнение и возвращаем 1
    print("Продолжение работы")
    yield 2  # Приостанавливаем выполнение и возвращаем 2
    print("Завершение работы")

gen = generate_values()  # Создаём генератор, но код внутри функции пока не выполняется

print(next(gen))  # Начало работы → 1
print(next(gen))  # Продолжение работы → 2
print(next(gen))  # Завершение работы → StopIteration, так как нет третьего yield
```

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Генераторные функции с параметрами

Полезно знать

ICH
IT CAREER HUB

Функция-генератор

Такая функция может принимать аргументы, которые передаются однократно при создании генератора и используются для генерации значений

Функция-генератор

Синтаксис

```
def generator(param1, param2):
    yield value
```

Пояснение

`param1, param2` – аргументы, передаваемые при создании генератора

Пример: генератор чисел "n" раз

```
def count_up_to(n):
    count = 1
    while count <= n:
            yield count   # Возвращаем текущее значение и "замораживаем" выполнение
        count += 1   # После следующего вызова next() продолжится отсюда

gen = count_up_to(5)

print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
# print(next(gen))  # StopIteration
```

IT CAREER HUB

Использование генератора в for

Важно

Генератор можно **перебирать в цикле** `for`, как и любой итератор. В этом случае `for` **автоматически вызывает** `next()`, пока генератор **не исчерпает все значения**

Пример: генератор и for

```
def count_up_to(n):
    count = 1
    while count <= n:
        yield count
        count += 1

gen = count_up_to(5)

for number in gen:
    print(number)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЯ

Выберите верный вариант ответа

Какой результат выдаст следующий код?

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

gen = countdown(3)

print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
```

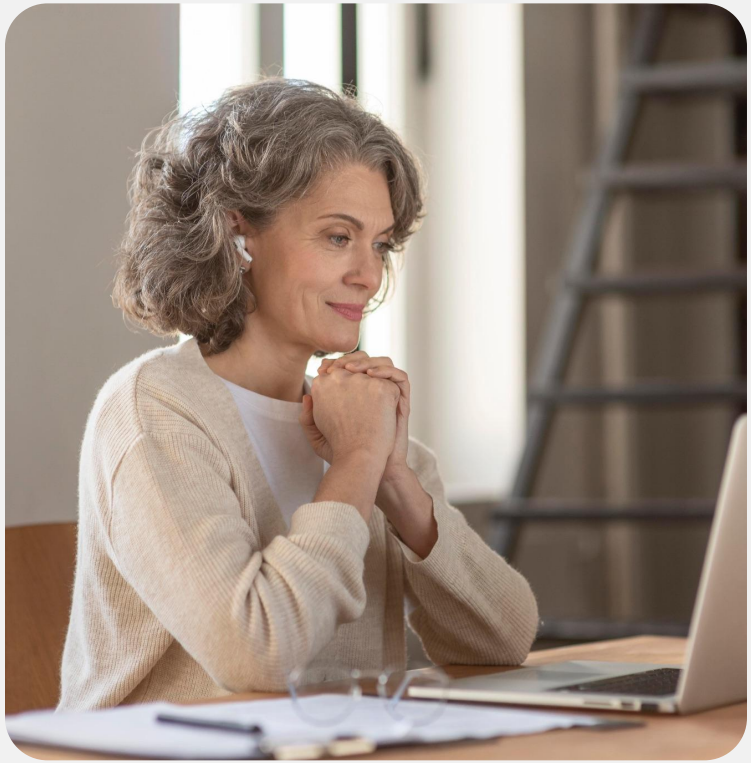
a. 3, 2, 1, 1
b. 3, 2, 1, 0
c. 3, 2, 1, 3
d. 3, 2, 1, StopIteration

Выберите верный вариант ответа

Какой результат выдаст следующий код?

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

gen = countdown(3)

print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
```

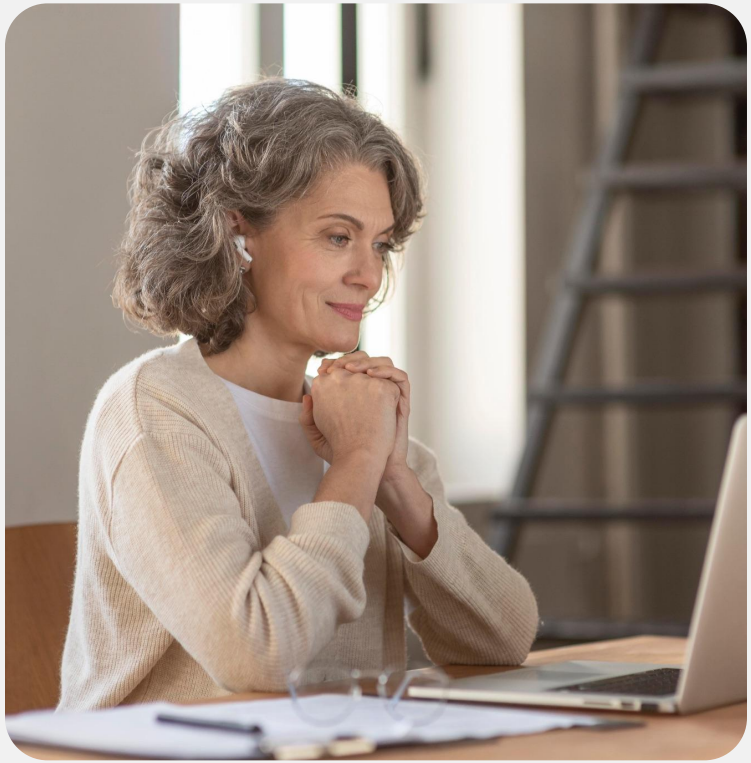
a. 3, 2, 1, 1
b. 3, 2, 1, 0
c. 3, 2, 1, 3
d. 3, 2, 1, StopIteration



Выберите верные варианты ответа

Какие утверждения о `yield` верны?

a. yield приостанавливает выполнение функции и запоминает состояние
b. После yield выполнение начинается заново
c. Генератор можно использовать только один раз
d. yield можно использовать только внутри функции



Выберите верные варианты ответа

Какие утверждения о `yield` верны?

a. yield приостанавливает выполнение функции и запоминает состояние
b. После yield выполнение начинается заново
c. Генератор можно использовать только один раз
d. yield можно использовать только внутри функции

IT CAREER HUB

Бесконечные генераторы

Полезно знать

Бесконечный генератор

Это генератор, который никогда не завершает выполнение и может неограниченно выдавать новые значения

Применение бесконечного генератора

Такие генераторы **особенно полезны**, когда нужно **создавать поток данных** без заранее известного количества элементов

Пример 1: генератор бесконечной последовательности

```python
def infinite_counter():
    count = 1
    while True:  # Бесконечный цикл
        yield count
        count += 1


gen = infinite_counter()

for number in gen:
    if number > 5:  # Условие выхода
        break  # Принудительная остановка генератора
    print(number)

print()

# Вызов next 10 раз
for _ in range(10):
    print(next(gen))
```

Особенности бесконечного генератора

- Генератор **не завершится сам по себе**, так как while True работает бесконечно.
- Нужно **вручную остановить выполнение** (например break в for) или вызвать next() вручную.

Применение бесконечного генератора

Бесконечные генераторы могут применяться **для циклического распределения ресурсов**, например, автоматического назначения задач на сотрудников в команде

Пример 2: генератор распределения задач

```python
def task_assigner(employees):
    while True:
        for employee in employees:
            yield employee


# Список сотрудников
team = ["Alice", "Bob", "Charlie"]


# Создаём генератор распределения задач
assigner = task_assigner(team)


# Назначаем 7 задач
for i in range(7):
    print(f"Task {i + 1} assigned to: {next(assigner)}")
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Метод close

Полезно знать

Метод close()

Этот метод используется для принудительного завершения работы генератора.
Когда вызывается `close()`, генератор завершается и при следующем вызове `next()` вызывает `StopIteration`.

Метод close()

Синтаксис

```
generator.close()
```

Пояснение

- `generator` – объект генератора.
- `close()` останавливает выполнение генератора, не дожидаясь его естественного завершения

Использование метода close()

Метод `close()` может использоваться для **прерывания работы генератора в определённой ситуации**, например, когда получено нужное значение

Пример: закрытие генератора при достижении условия

```python
def sensor_data(data):
    for value in data:
        yield value

numbers = [10, 20, 30, 40, 50]
gen = sensor_data(numbers)

for element in gen:
    print("Получено значение:", element)

    # Завершаем генератор, когда получено нужное значение
    if element >= 30:
        print("Значение найдено, закрываем генератор.")
        gen.close()

# next(gen)  # Вызовет ошибку
```

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Конструкция yield from

Полезно знать

ICH
IT CAREER HUB

yield from

Это конструкция в Python, которая делегирует управление другому генератору или итерируемому объекту. Она упрощает вложенные генераторы, позволяя не вызывать yield вручную в цикле

Конструкция yield from

Синтаксис

```
yield from iterable
```

Пояснение

- `iterable` – любой итерируемый объект (список, кортеж, строка, другой генератор)
- `yield from` **поочерёдно передаёт элементы** от указанного источника

Пример 1: итерируемый объект

```
def letters():
    yield from "ABC"

gen = letters()

print(next(gen))  # A
print(next(gen))  # B
print(next(gen))  # C
```

Пример 2: Вспомогательный генератор

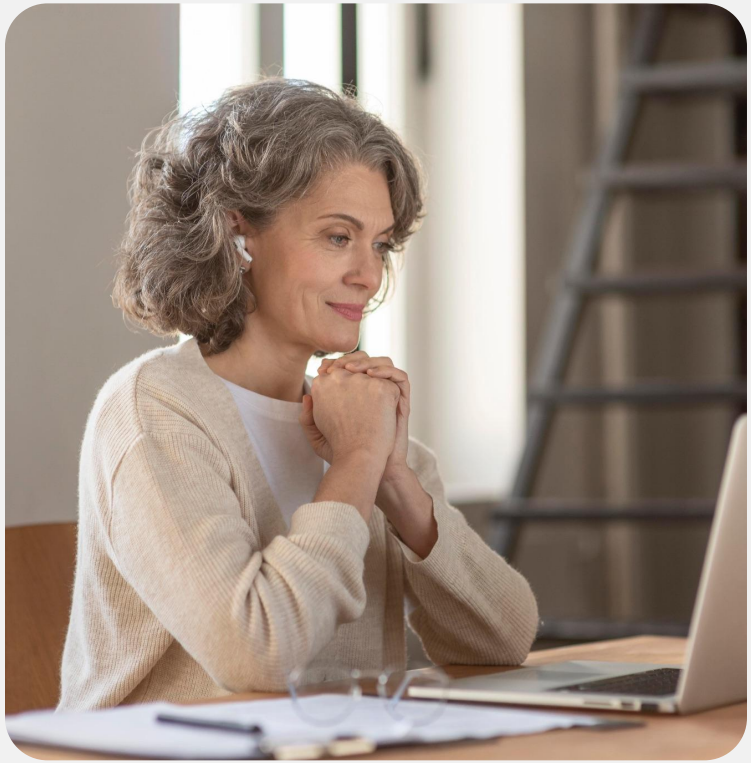
```python
# Умеет работать с одним объектом
def process_values(data):
    for value in data:
        yield value * 2

# Собирает несколько объектов
def main_generator(*sequences):
    # Делегирует обработку каждого объекта вспомогательному генератору
    for seq in sequences:
        yield from process_values(seq)

# Используем несколько источников данных
data1 = [1, 2, 3]
data2 = [10, 15]

for result in main_generator(data1, data2):
    print(result)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

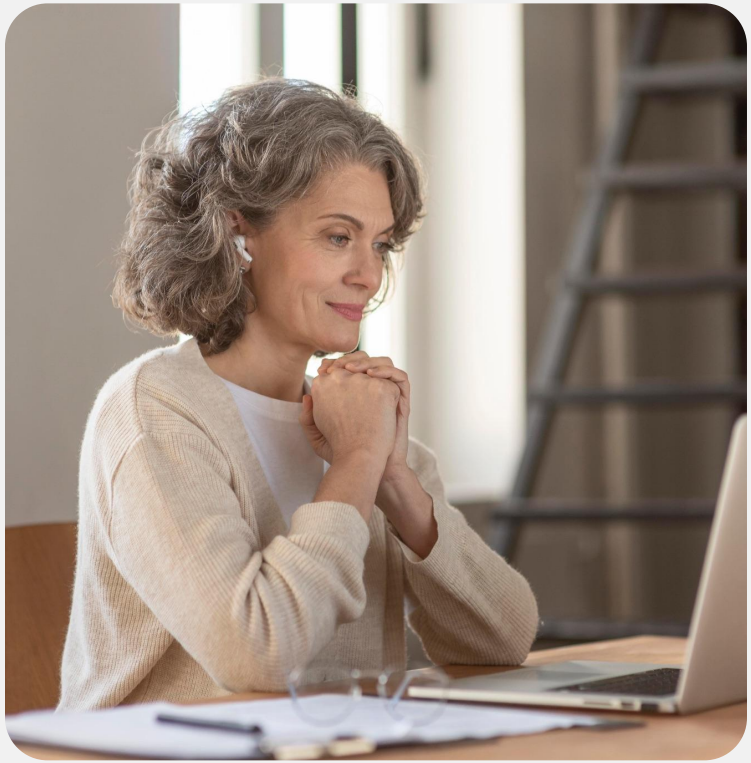
ЗАДАНИЯ



Выберите верный вариант ответа

Какие утверждения о yield from верны?

a. Позволяет делегировать управление другому генератору
b. Используется только с бесконечными генераторами
c. Может передавать элементы из списка
d. Упрощает работу с вложенными генераторами



Выберите верный вариант ответа

Какие утверждения о yield from верны?

a. Позволяет делегировать управление другому генератору
b. Используется только с бесконечными генераторами
c. Может передавать элементы из списка
d. Упрощает работу с вложенными генераторами

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ПРАКТИЧЕСКАЯ РАБОТА

Фильтр чисел

Создайте генератор, который принимает **список чисел** и выдаёт **только числа, кратные 5**.

Данные:

```
numbers = [12, 15, 33, 40, 55, 62, 75, 83, 90]
```

Пример вывода:

```
15
40
55
75
90
```

Квадраты чисел

Создайте **генератор**, который принимает число **n** и **генерирует квадраты чисел** от **1 до n** включительно

Данные:

```
n = 10
```

Пример вывода:

```
1
4
9
16
25
36
49
64
81
100
```

IT CAREER HUB

ДОМАШНЕЕ ЗАДАНИЕ

Домашнее задание

1. Генератор Фибоначчи

Создайте генератор, который генерирует последовательность Фибоначчи бесконечно,

возвращая по одному числу за раз.

Последовательность Фибоначчи — это ряд чисел, где каждое следующее число равно сумме двух предыдущих.

Начинается с 0 и 1.

Пример вывода:

```
0
1
1
2
3
5
8
13
21
34
```

Домашнее задание

2. Генератор уникальных элементов

Создайте генератор, который принимает список элементов и выдаёт только уникальные значения, сохраняя порядок их появления в исходном списке

Данные:

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

Пример вывода:

```
3
1
2
4
5
6
7
8
```

Заключение