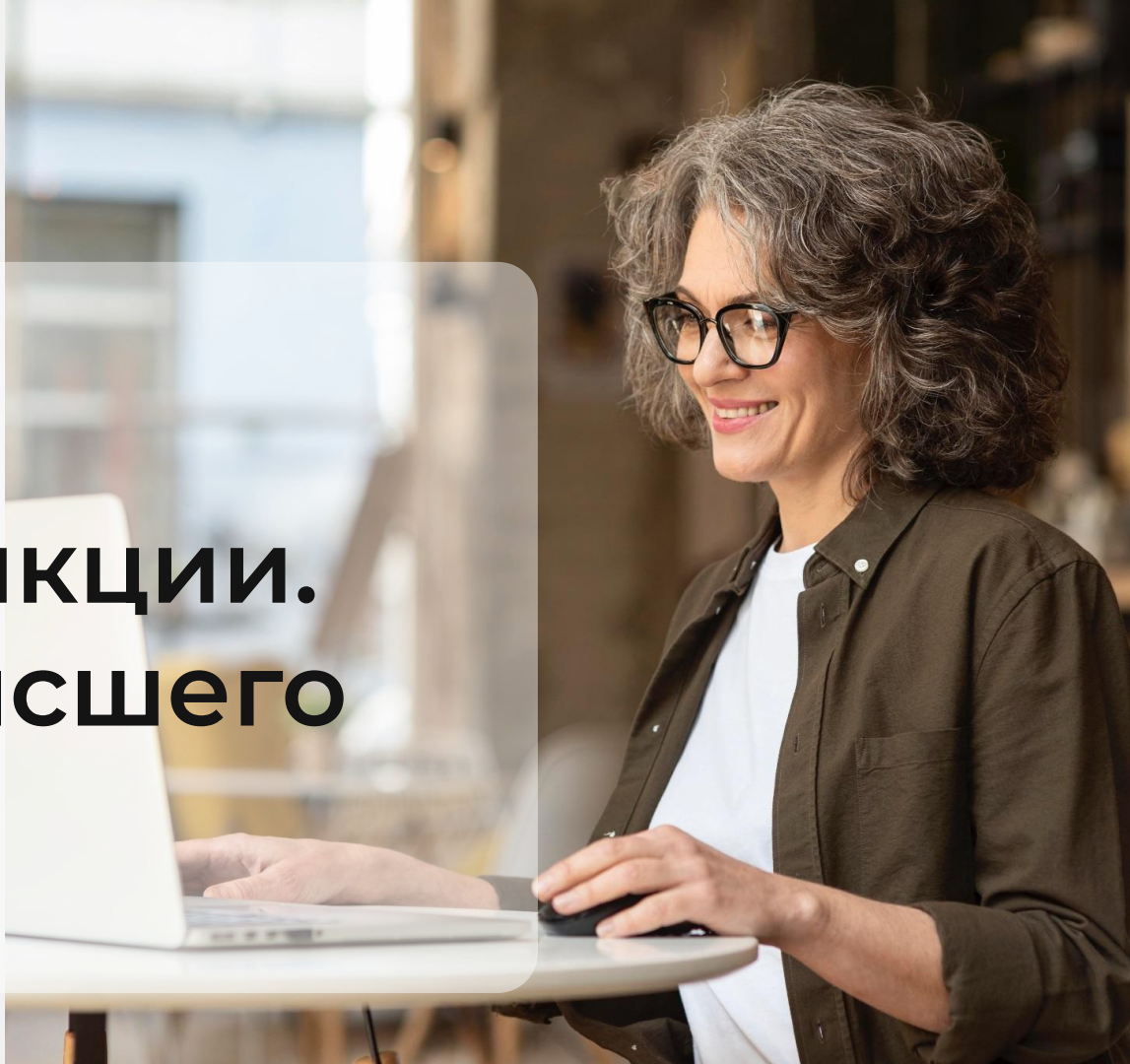


Python

# Lambda-функции. Функции высшего порядка



# 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

-  Модуль time
-  Модуль collections
-  Метод popitem()
-  Класс defaultdict
-  Класс Counter
-  Методы класса Counter
-  Операции между объектами Counter

# План занятия

- Передача функций в качестве аргументов
- Lambda-функции
- Парадигмы программирования
- Функциональное программирование
- Функции высшего порядка
- Функции map, filter, reduce
- Функции any и all
- Функция как ключ в sorted(), min(), max()



# ОСНОВНОЙ БЛОК





# Передача функций в качестве аргументов

# Передача функций в качестве аргументов



Функции в Python являются объектами, которые можно присваивать переменным, хранить в коллекциях и передавать в другие функции.

# Как это работает?



Не нужно указывать ( ) при передаче, иначе передастся результат её выполнения.



Функция может быть вызвана внутри другой функции по переданной ссылке с помощью ( ).

# Передача функции без вызова



```
def square(x):
    return x * x

def cube(x):
    return x * x * x

def apply_function(func, value):
    return func(value) # Вызываем переданную функцию внутри другой функции

result_square = apply_function(square, 5) # Передаём функцию square без вызова (без скобок)
result_cube = apply_function(cube, 5) # Передаём функцию cube без вызова (без скобок)

print(result_square)

print(result_cube)
```

# Ошибка при передаче вызванной функции



```
def square(x):  
    return x * x
```

```
def apply_function(func, value):  
    return func(value)
```

```
# Передается результат вызова функции, а не ссылка на функцию  
result = apply_function(square(5), 5) # Ошибка!
```

# Хранение функций в коллекциях



```
def add(x, y):
    return x + y

def multiply(x, y):
    return x * y

# Функции можно хранить в списках, словарях и передавать их динамически
operations = {
    "+": add,
    "*": multiply
}

choice = input("Выберите операцию (+, *): ")

# Из словаря получена функция и скобки с аргументами запускают её
print(operations[choice](10, 5))
```

# Передача встроенной функции



```
def process_data(func, data):  
    return func(data)
```

```
# Можно передавать не только пользовательские функции, но и встроенные  
result = process_data(abs, -10)  
print(result)
```



# ВОПРОСЫ





# Lambda-функции



## Lambda-функция

Это небольшая, одноразовая функция, которая не требует явного объявления с `def`. Она используется для краткой записи простых операций и может быть передана как аргумент в другие функции.

# Lambda-функции



## Пояснение

- `lambda` — ключевое слово для создания анонимной функции.
- `arguments` — параметры, которые принимает функция.
- `expression` — выражение, результат которого возвращается (без `return`).

## Синтаксис

`lambda arguments: expression`

# Lambda с одним аргументом



# Функция принимает число x и возвращает его квадрат

```
square = lambda x: x ** 2
```

```
print(square(4))
```

```
print(square(5))
```

# Аналог с def

```
def square(x):
```

```
    return x ** 2
```

```
print(square(4))
```

```
print(square(5))
```

# Lambda с несколькими аргументами



# Функция принимает два аргумента и возвращает их сумму

```
add = lambda x, y: x + y
```

```
print(add(3, 5))
```

```
print(add(8, 9))
```

# Аналог с def

```
def add(x, y):
```

```
    return x + y
```

```
print(add(3, 5))
```

```
print(add(8, 9))
```

# Lambda как аргументы других функций



Lambda-функции можно передавать как аргументы в другие функции, не создавая отдельные именованные функции.

# Lambda как аргументы других функций



```
def apply_func(func, numbers):  
    return [func(num) for num in numbers]  
  
result = apply_func(lambda x: x + 10, [5, 8, 3])  
print(result)
```

# Особенности lambda-функций



- Lambda-функция всегда возвращает результат выражения
- Lambda может содержать только одно выражение
- Нет многострочных блоков кода (if, for и т. д.)



# ВОПРОСЫ





**ЗАДАНИЕ**



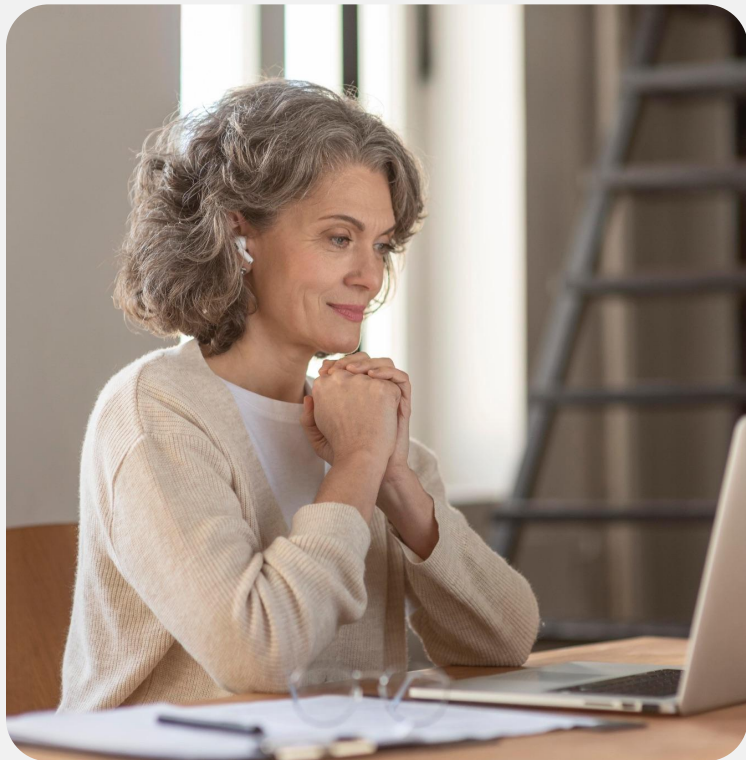


## Исправьте ошибку в коде

```
operations = {
    "sum": lambda x, y: x + y,
    "mul": lambda x, y: x * y
}
```

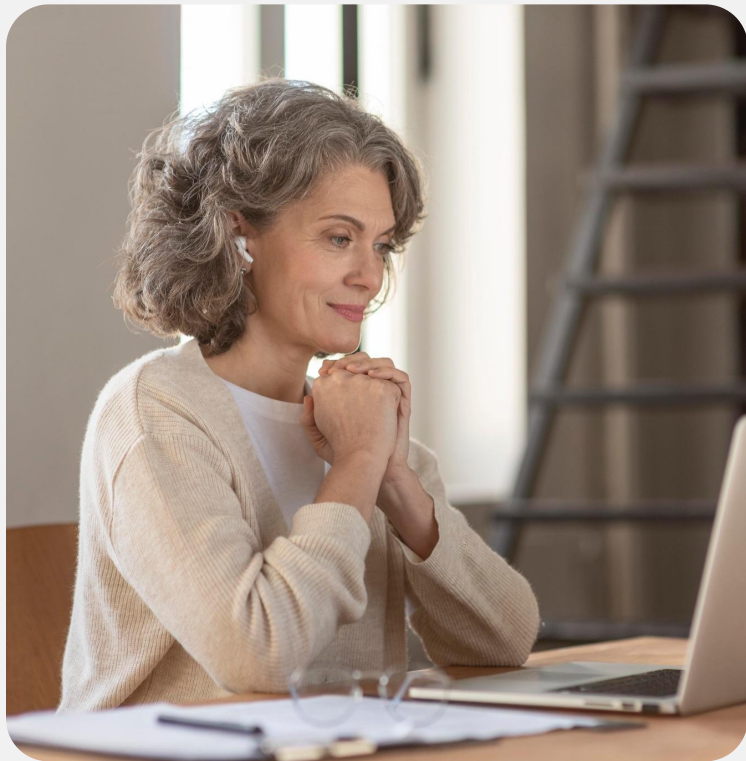
```
print(operations("sum")(2, 3))
```

```
print(operations("mul")(2, 3))
```



## Ответ

Исправьте круглые скобки после operations на квадратные.

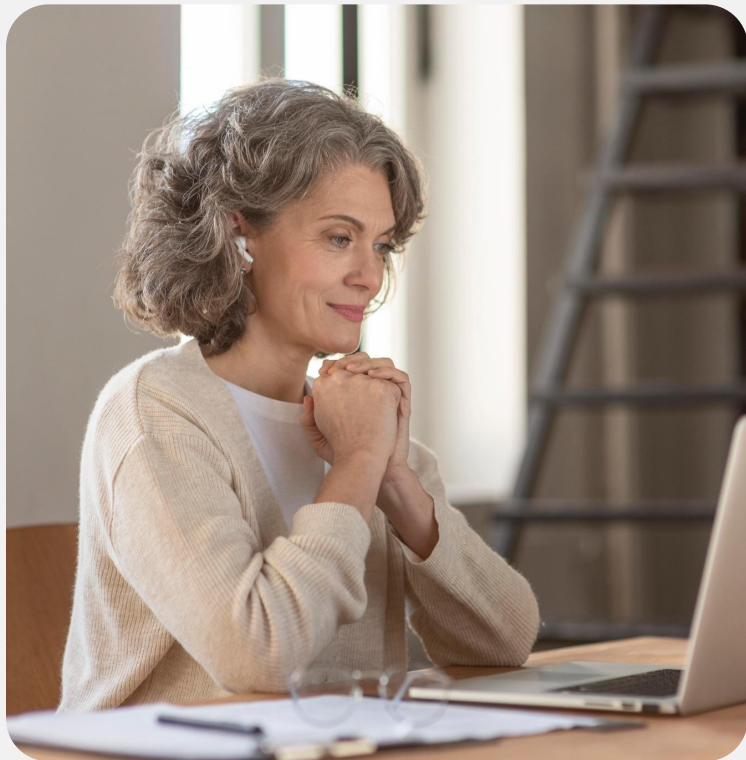


## Объясните

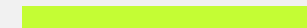
Что происходит при выполнении следующего кода. Произойдет ли ошибка?

```
def add(x, y):  
    return x + y
```

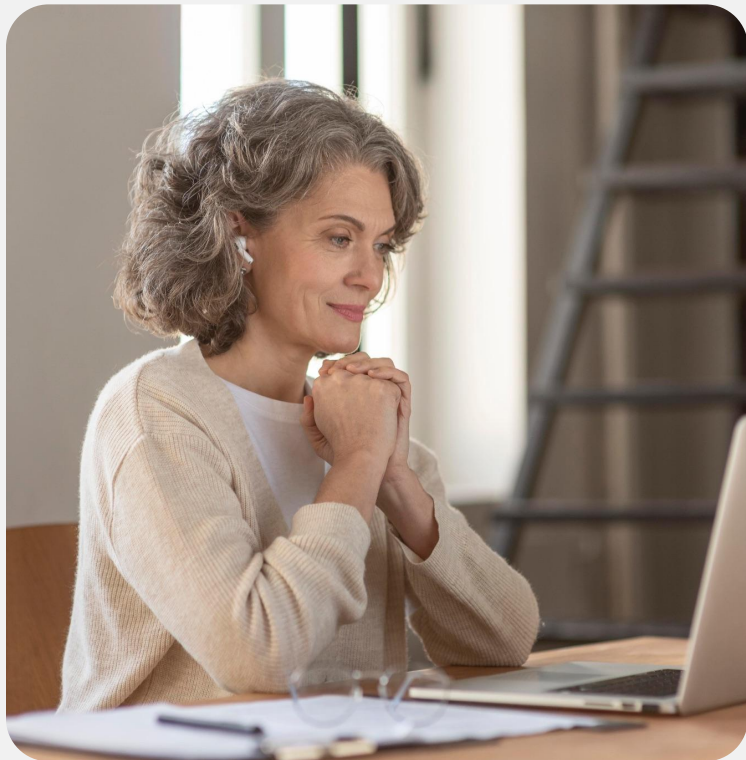
```
print((lambda f, a, b: f(a, b))(add, 3,  
4))
```



## Ответ

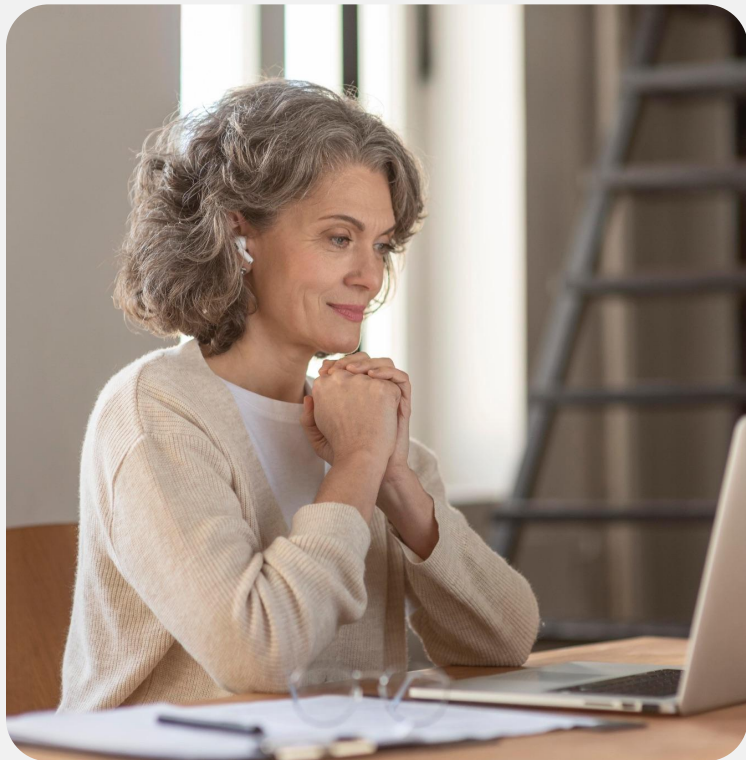


Нет, код корректен.



## Объясните

Можно ли передавать встроенные функции Python в качестве аргументов другим функциям?



## Ответ

Да, можно. Например, функции `abs`, `len`, `sum` и другие можно передавать как аргумент.



# ВОПРОСЫ





# Парадигмы программирования



## Парадигма программирования

Это стиль написания и организации кода, который определяет способы решения задач в программировании.

# Основные парадигмы программирования

Императивное

Процедурное

Объектно-  
ориентированное

Функциональное

Декларативное

# Императивное программирование



Описывает последовательность команд, которые изменяют состояние программы.



Код состоит из инструкций, выполняемых шаг за шагом.



**Пример языков:** Python, C, Java.

# Процедурное программирование



Код организован в функции (процедуры), каждая из которых выполняет определённую задачу.



Используется разделение программы на логические блоки для повторного использования.



**Пример:** Python, Pascal, C.

# Объектно-ориентированное программирование



Основной концепцией является объект, который объединяет данные и методы для их обработки.



Код организуется в классы и объекты, позволяя моделировать реальные сущности.



**Пример:** Python, Java, C++.

# Функциональное программирование



Основано на функциях высшего порядка, чистых функциях и отсутствии изменения состояния.



Предпочитает использование неизменяемых данных и рекурсии вместо циклов.



**Пример:** Haskell, Lisp, Python.

# Декларативное программирование



Описывает **что** должно быть сделано, а не **как**.

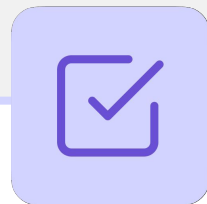


Включает функциональное программирование, а также языки разметки (SQL, HTML).



**Пример:** SQL, Prolog, Haskell.

# Современные языки программирования



Такие как Python, поддерживают несколько парадигм, позволяя использовать гибридные подходы. Например, в Python можно писать как в процедурном стиле, так и использовать ООП и функциональные концепции. Выбор парадигмы зависит от задачи, удобства и требований проекта.



# ВОПРОСЫ





# Функциональное программирование



## Функциональное программирование

Это парадигма программирования, в которой основной единицей организации кода являются функции. Программы строятся из чистых функций, которые принимают аргументы и возвращают результат, не изменяя состояние программы.

# Основные принципы функционального программирования



Чистые функции



Неизменяемость данных



Функции как объекты



Функции высшего порядка



Рекурсия вместо циклов



# ВОПРОСЫ





# Функции высшего порядка



## Функции высшего порядка

Это функции, которые могут принимать другие функции в качестве аргументов и/или возвращать функции в качестве результата

# Признаки функций высшего порядка



Принимают функции в качестве аргументов



Возвращают функции как результата

# Часто используемые функции высшего порядка



Встроенные функции высшего порядка: `map()`, `filter()`, `reduce()`



Использование `sorted()` с ключом сортировки



# ВОПРОСЫ





Функции **map, filter,**  
**reduce**

# Функции map, filter и reduce



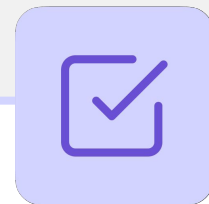
Это функции высшего порядка, которые принимают другую функцию в качестве аргумента и применяют её к элементам переданного итерируемого объекта



## Функция `map()`

Эта функция применяет переданную функцию к каждому элементу одного или нескольких итерируемых объектов и возвращает итератор с результатами

# Особенности функции map



- Используется для преобразования данных
- Результат нужно преобразовать в список или другой итерируемый объект, чтобы увидеть значения

# Функция map()



## Синтаксис

```
map(function, iterable)
```

## Пояснения

- `function` — функция, применяемая к каждому элементу
- `iterable` — итерируемый объект (список, кортеж, строка и т.д.)

# Пример с одним объектом



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

# Каждый элемент списка возводится в квадрат

squared = map(lambda x: x ** 2, numbers)

print(list(squared)) # [1, 4, 9, 16]
```

# Пример с несколькими объектами



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

```
b = [4, 5, 6]
```

```
# Каждая пара элементов списков суммируется
```

```
result = map(lambda x, y: x + y, a, b)
```

```
print(list(result)) # [5, 7, 9]
```

# Пример со встроенными функциями



```
group_numbers = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

```
# К каждому кортежу применяется функция sum
```

```
result = map(sum, group_numbers)
```

```
print(list(result)) # [6, 15, 24]
```



## Функция filter()

Эта функция используется для фильтрации элементов итерируемого объекта, возвращая только те, для которых переданная функция-предикат возвращает True

# Особенности функции filter



- Используется для отбора нужных элементов, соответствующих условию
- Если вместо функции передать None, будут выбраны только элементы, которые оцениваются как True
- Результат нужно преобразовать в список или другой итерируемый объект, чтобы увидеть значения

# Функция filter()



## Синтаксис

```
filter(function, iterable)
```

## Пояснения

- `function` — функция-предикат, определяющая условие для фильтрации
- `iterable` — итерируемый объект

# Пример с функцией-предикатом



```
numbers = [1, 2, 4, 5, 7, 9, 10, 11]

# Из списка выбираются только чётные числа

even_numbers = filter(lambda x: x % 2 == 0, numbers)

print(list(even_numbers)) # [2, 4, 10]
```

# Пример с None



```
data = [0, 1, False, True, '', 'Python', [], [1, 2, 3]]  
# Из списка выбираются только те элементы, которые оцениваются как True  
even_numbers = filter(None, data)  
print(even_numbers)  
print(list(even_numbers))
```



## Функция `reduce()`

Эта функция последовательно применяет переданную функцию к элементам итерируемого объекта с накоплением результата, сводя его к одному значению. Она находится в модуле `functools`

# Особенности функции reduce



- Используется для агрегации данных (например, суммы или произведения).
- Работает с парой элементов, начиная с первых двух, и продолжает с результатом и каждым следующим значением

# Функция reduce()



## Синтаксис

```
from functools import reduce  
reduce(function, iterable, initializer)
```

## Пояснения

- `function` — функция, принимающая два аргумента и возвращающая один результат
- `iterable` — итерируемый объект
- `initializer` (необязательно) — начальное значение для накопления результата

# Пример



```
from functools import reduce
```

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

```
# Умножение всех элементов списка последовательно
```

```
result = reduce(lambda x, y: x * y, numbers)
```

```
print(result) # 24
```

# Пример с initializer



```
from functools import reduce

numbers = [1, 2, 3, 4]

# Умножение всех элементов списка, начиная с 10
result = reduce(lambda x, y: x * y, numbers, 10)

print(result) # 240
```

# Сравнение функций map, filter, reduce

| Функция | Описание   | Возвращает                         |
|---------|--|------------------------------------|
| map     | Применяет функцию ко всем элементам итерируемого объекта | Итератор преобразованных элементов |
| filter  | Фильтрует элементы на основе условия                     | Итератор отфильтрованных элементов |
| reduce  | Последовательно применяет функцию, накапливая результат  | Итоговое значение                  |



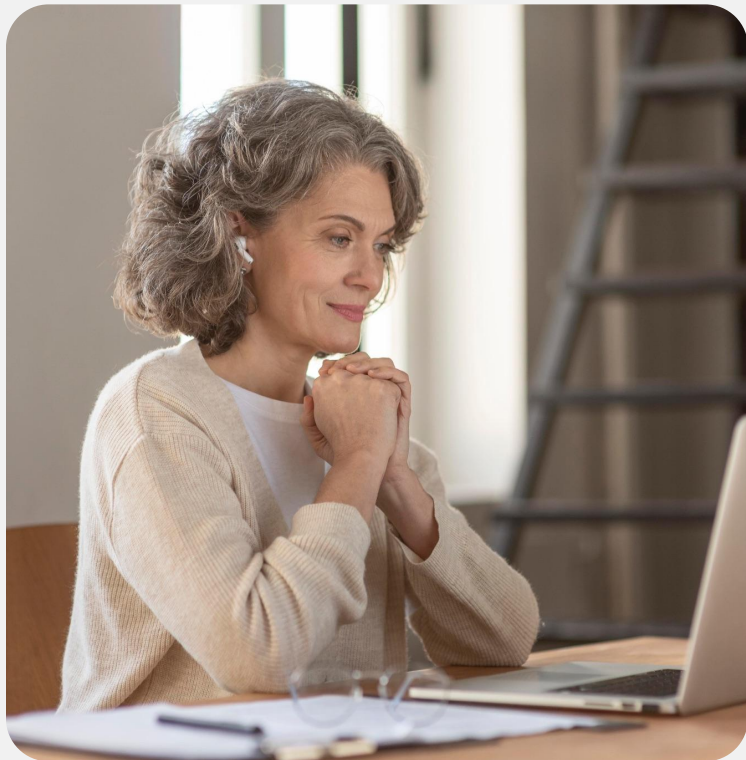
# ВОПРОСЫ





# ЗАДАНИЕ





## Выберите правильный вариант ответа

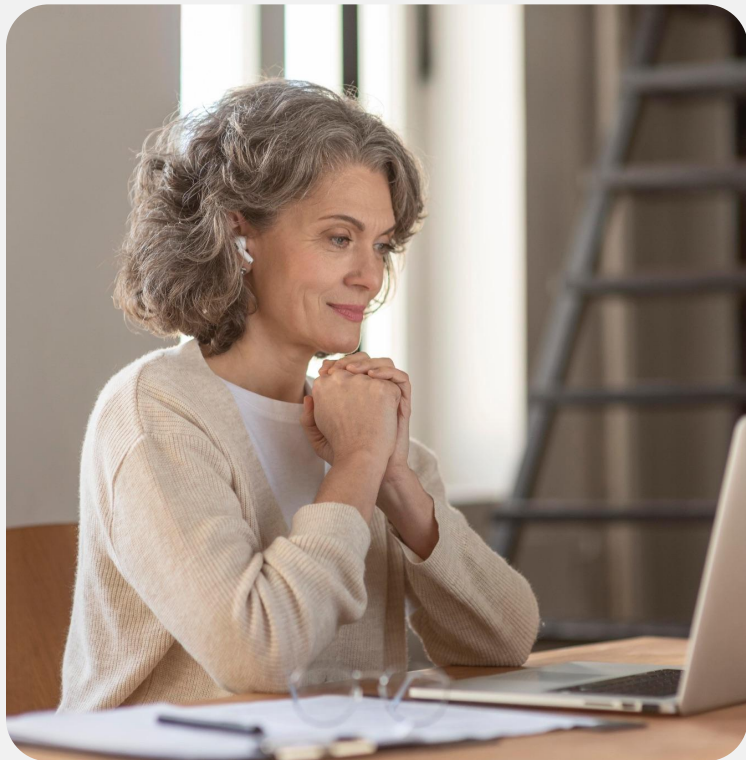
Какой результат будет выведен при выполнении следующего кода?

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

```
lengths = map(len, words)
```

```
print(list(lengths))
```

- a. [5, 6, 6]
- b. [('apple', 5), ('banana', 6), ('cherry', 6)]
- c. ['apple', 'banana', 'cherry']
- d. Ошибка



## Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

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

```
lengths = map(len, words)
```

```
print(list(lengths))
```

- a. [5, 6, 6]
- b. [('apple', 5), ('banana', 6), ('cherry', 6)]
- c. ['apple', 'banana', 'cherry']
- d. Ошибка



## Выберите правильный вариант ответа

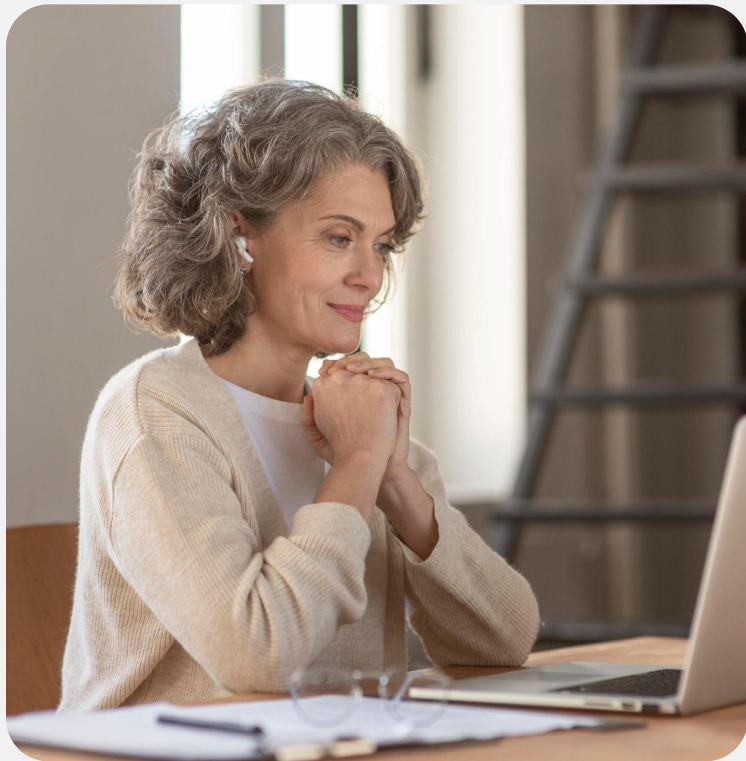
Какой результат будет выведен при выполнении следующего кода?

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

```
filtered = filter(lambda x: x % 2 == 1, numbers)
```

```
print(list(filtered))
```

- a. [5, 3, 1, 5]
- b. [1, 3, 5, 5]
- c. [1, 3, 5]
- d. [True, True, False, True, True, False]



## Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

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

```
filtered = filter(lambda x: x % 2 == 1, numbers)
```

```
print(list(filtered))
```

- a. [5, 3, 1, 5]
- b. [1, 3, 5, 5]
- c. [1, 3, 5]
- d. [True, True, False, True, True, False]



## Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
from functools import reduce

numbers = [1, 2, 3, 4]

result = reduce(lambda x, y: x + y, numbers)

print(result)
```

- a. [1, 2, 3, 4]
- b. [1, 3, 5, 7]
- c. [1, 3, 6, 10]
- d. 10



## Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
from functools import reduce

numbers = [1, 2, 3, 4]

result = reduce(lambda x, y: x + y, numbers)

print(result)
```

- a. [1, 2, 3, 4]
- b. [1, 3, 5, 7]
- c. [1, 3, 6, 10]
- d. 10



# ВОПРОСЫ





# Функции `any` и `all`

# Функции any и all



## Ложные значения

0, None, False, пустые коллекции ([], {}, ()), set())

## Истинные значения

Всё, что не считается ложным



## Функция `any()`

Эта функция проверяет, содержит ли итерируемый объект хотя бы один истинный элемент

# Функция any()



## Синтаксис

`any(iterable)`

## Пояснения

- `iterable` — итерируемый объект (список, кортеж, строка, множество, словарь и т. д.), элементы которого нужно проверить

# Особенности функции any



- Возвращает True, если хотя бы один элемент в итерируемом объекте является истинным.
- Если все элементы ложные, возвращает False.
- Если объект пустой, возвращает False.
- Выполняет проверку поэлементно и останавливается, как только находит истинное значение (**ленивое вычисление**)

# Пример использования



```
data = [0, None, False, 1]
```

```
print(any(data)) # True, так как есть хотя бы одно истинное значение (1)
```

```
data = [0, None, False]
```

```
print(any(data)) # False, так как все элементы ложные
```

```
data = []
```

```
print(any(data)) # False, так как объект пустой
```



## Функция all()

Эта функция проверяет, являются ли все элементы итерируемого объекта истинными

# Функция all()



## Синтаксис

```
all(iterable)
```

## Пояснения

- `iterable` — итерируемый объект, элементы которого нужно проверить

# Особенности функции all



- Возвращает True, если **все** элементы в итерируемом объекте истинные
- Если хотя бы один элемент ложный, возвращается False
- Возвращает True для **пустого объекта**, так как нет элементов, которые можно было бы считать ложными
- Выполняет проверку поэлементно и останавливается, как только находит ложное значение

# Пример использования



```
data = [1, 2, 3]
print(all(data)) # True, так как все элементы истинные
```

```
data = [1, 0, 3]
print(all(data)) # False, так как 0 – ложное значение
```

```
data = []
print(all(data)) # True, так как объект пустой
```

# Сравнение функций any и all

| Характеристика     | any  | all                                      |
|--------------------|--|--|
| Результат          | True, если хотя бы один элемент истинный.  | True, если все элементы истинные.        |
| Пустой объект      | Возвращает False.                          | Возвращает True.                         |
| Остановка проверки | При нахождении первого истинного значения. | При нахождении первого ложного значения. |
| Применение         | Проверка наличия истинных значений.        | Проверка, что все значения истинные.     |

# Пример использования



# Проверка, что хотя бы один объект соответствует условию

```
conditions = [x > 10 for x in [5, 20, 8]]
```

```
print(any(conditions)) # True
```

# Проверка, что все объекты соответствуют условию

```
conditions = [x > 0 for x in [5, 20, 8]]
```

```
print(all(conditions)) # True
```



# ВОПРОСЫ





# ЗАДАНИЕ





## Выберите правильный вариант ответа

Что произойдёт, если передать в функцию `any` пустой список?

```
data = []
```

```
print(any(data))
```

- a. Возвращается True
- b. Возвращается False
- c. Возникает ошибка
- d. Возвращается None



## Выберите правильный вариант ответа

Что произойдёт, если передать в функцию `any` пустой список?

```
data = []
```

```
print(any(data))
```

- a. Возвращается True
- b. Возвращается False**
- c. Возникает ошибка
- d. Возвращается None



## Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
data = [1, 2, 3, "None"]
```

```
print(all(data))
```

- a. True
- b. False
- c. Ошибка
- d. None



## Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
data = [1, 2, 3, "None"]
```

```
print(all(data))
```

- a. True
- b. False
- c. Ошибка
- d. None



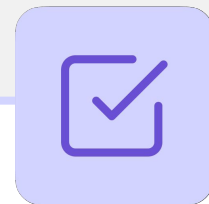
# ВОПРОСЫ





Функция как ключ в  
`sorted()`, `min()`, `max()`

# Параметр key



В Python функции `sorted()`, `min()` и `max()` принимают необязательный параметр **key**.

Он определяет, по какому критерию выполнять сортировку или поиск минимального/максимального значения.

В параметр `key` можно передавать встроенные и пользовательские функции

# Функция sorted



## Пояснения

Эта функция сортирует элементы по значению, которое возвращает переданная функция

## Синтаксис

```
sorted(iterable, key=function,  
reverse=False)
```

# Пример: использование встроенной функции



## Код

```
words = ['mango', 'grape', 'apple',  
'strawberry', 'banana', 'pineapple',  
'kiwi', 'blueberry']
```

# Сортировка по длине слов

```
result = sorted(words, key=len)
```

```
print(result)
```

## Вывод

```
['kiwi', 'apple', 'mango', 'grape',  
'banana', 'orange', 'pineapple',  
'strawberry']
```

# Пример: пользовательская функция



```
def last_char_len(s):  
    return s[-1], len(s)  
  
words = ['mango', 'grape', 'apple', 'strawberry', 'banana', 'pineapple', 'kiwi',  
         'blueberry']  
  
# Сортировка по последнему символу и длине слова  
result = sorted(words, key=last_char_len)  
  
print(result)
```

# Пример: анонимная функция lambda



```
words = ['mango', 'grape', 'apple', 'Strawberry', 'Banana', 'pineapple', 'kiwi',  
         'blueberry']
```

```
# Сортировка по первому символу (игнорируя регистр) и по последнему символу
```

```
result = sorted(words, key=lambda x: (x[0].lower(), x[-1]))
```

```
print(result)
```

# Пример: сортировка списка кортежей



## Код

```
students = [("Alice", 25), ("Bob", 20),  
            ("Charlie", 23)]
```

```
# Сортировка списка кортежей по возрасту  
# (второй элемент)
```

```
sorted_students = sorted(students,  
                          key=lambda x: x[1])
```

```
print(sorted_students)
```

## Вывод

```
[('Bob', 20), ('Charlie', 23), ('Alice',  
25)]
```

# Функции min() и max()



## Пояснения

Функции `min()` и `max()` с параметром `key` позволяют находить минимальный или максимальный элемент на основе вычисленного значения

## Синтаксис

```
min(iterable, key=function)
```

```
max(iterable, key=function)
```

# Пример: поиск самого длинного слова



## Код

```
words = ["apple", "banana", "kiwi",  
"grapefruit"]  
  
longest_word = max(words, key=len)  
  
print(longest_word)
```

## Вывод

```
grapefruit
```

# Пример: поиск города с минимальным населением



## Код

```
cities = [('New York', 8419600), ('Los
Angeles', 3980400), ('Chicago',
2716000)]

smallest_city = min(cities, key=lambda
x: x[1])


print(smallest_city)
```

## Вывод

```
('Chicago', 2716000)
```



# ПРАКТИЧЕСКАЯ РАБОТА





## Список квадратов чисел

Напишите функцию, которая сформирует список квадратов из полученного списка, без использования циклов или списковых включений.

**Данные:**

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

**Пример вывода:**

`[1, 4, 9, 16, 25]`

# Сортировка по возрасту

Отсортируйте список кортежей (имя, возраст) по возрасту.

**Данные:**

```
people = [
    ("Mike", 19), ("Nancy", 35), ("Charlie", 23), ("Oscar", 33), ("Eve", 29),
    ("Frank", 33), ("Bob", 20), ("Grace", 27), ("Isabella", 19), ("Jack", 24),
    ("Alice", 25), ("Kevin", 28), ("Laura", 31), ("Diana", 30), ("Henry", 19)
]
```

**Пример вывода:**

```
[('Mike', 19), ('Isabella', 19), ('Henry', 19), ('Bob', 20), ('Charlie', 23), ('Jack',
24), ('Alice', 25), ('Grace', 27), ('Kevin', 28), ('Eve', 29), ('Diana', 30), ('Laura',
31), ('Oscar', 33), ('Frank', 33), ('Nancy', 35)]
```

# Сортировка по возрасту и имени

Отсортируйте список кортежей (имя, возраст) по убыванию возраста, в рамках одинакового возраста отсортируйте также по имени по алфавиту.

## Данные:

```
people = [
    ("Mike", 19), ("Nancy", 35), ("Charlie", 23), ("Oscar", 33), ("Eve", 29),
    ("Frank", 33), ("Bob", 20), ("Grace", 27), ("Isabella", 19), ("Jack", 24),
    ("Alice", 25), ("Kevin", 28), ("Laura", 31), ("Diana", 30), ("Henry", 19)
]
```

## Пример вывода:

```
[('Nancy', 35), ('Frank', 33), ('Oscar', 33), ('Laura', 31), ('Diana', 30), ('Eve',
29), ('Kevin', 28), ('Grace', 27), ('Alice', 25), ('Jack', 24), ('Charlie', 23),
('Bob', 20), ('Henry', 19), ('Isabella', 19), ('Mike', 19)]
```



# ВОПРОСЫ



# Домашнее задание

## Выбор заказов

У вас есть список заказов. Каждый заказ содержит название продукта и его цену. Напишите функцию, которая:

1. Отбирает заказы дороже 500.
2. Создаёт список названий отобранных продуктов в алфавитном порядке.
3. Возвращает итоговый список названий.

### Данные:

```
orders = [
    {"product": "Laptop", "price": 1200},
    {"product": "Mouse", "price": 50},
    {"product": "Keyboard", "price": 100},
    {"product": "Monitor", "price": 300},
    {"product": "Chair", "price": 800},
    {"product": "Desk", "price": 400}
]
```

### Пример вывода:

```
['Chair', 'Laptop']
```

# Домашнее задание

## Статистика продаж

Дан список продаж в виде кортежей (товар, количество, цена).

Напишите программу, которая:

1. Вычисляет общую выручку для каждого товара.
2. Возвращает словарь с товарами {товар: выручка}, отсортированный по убыванию выручки.

### Данные:

```
sales = [  
    ("Laptop", 5, 1200),  
    ("Mouse", 50, 20),  
    ("Keyboard", 30, 50),  
    ("Monitor", 10, 300),  
    ("Chair", 20, 800)  
]
```

### Пример вывода:

```
{'Chair': 16000, 'Laptop': 6000,  
'Monitor': 3000, 'Keyboard': 1500,  
'Mouse': 1000}
```

## Заключение

