IT CAREER HUB

Python

Рекурсия

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

План занятия

- Примеры рекурсивных функций
- Хвостовая рекурсия
- Рекурсия или итерация?
- Функция isinstance
- Проверяем, что копия независима от оригинала

ICH IT CAREER HUB

ОСНОВНОЙ БЛОК

IT CAREER HUB

Примеры рекурсивных функций

Факториал числа

Факториал числа n!n! – это произведение всех чисел от 1 до nn.

Рекурсивная реализация

```python
def factorial(n: int) -> int:
    if n == 0 or n == 1:  # Базовый случай
        return 1
    return n * factorial(n - 1)  # Рекурсивный случай


print(factorial(5))
```

Бинарный поиск

Это эффективный алгоритм поиска элемента в отсортированном списке.

Рекурсивная реализация

```
def binary_search(arr: list[int], target: int, left: int, right: int) -> int:
    if left > right:  # Базовый случай: элемент не найден
        return -1
    mid = (left + right) // 2
    if arr[mid] == target:
        return mid
    elif arr[mid] < target:
        return binary_search(arr, target, mid + 1, right)  # Поиск в правой части
    else:
        return binary_search(arr, target, left, mid - 1)  # Поиск в левой части
array = [1, 3, 5, 7, 9, 11, 13]
print(binary_search(array, 5, 0, len(array) - 1))
print(binary_search(array, 13, 0, len(array) - 1))
print(binary_search(array, 8, 0, len(array) - 1))
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Хвостовая рекурсия

Хвостовая рекурсия

Это особый вид рекурсии, в котором рекурсивный вызов является последней операцией функции перед возвратом результата.

Обычная рекурсия

```python
def factorial(n: int) -> int:
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)


print(factorial(5))
```

Хвостовая рекурсия

```python
def factorial_tail(n: int, accumulator: int = 1) -> int:
    if n == 0 or n == 1:
        return accumulator
    return factorial_tail(n - 1, n * accumulator)


print(factorial_tail(5))
```

Разница между обычной и хвостовой рекурсией

В Python нет встроенной оптимизации хвостовой рекурсии, которая заменяет хвостовой рекурсивный вызов на повторное использование текущего кадра стека, что позволяет избежать переполнения стека. Поэтому рекомендуется заменять хвостовую рекурсию на итерацию.

Итеративный вариант факториала

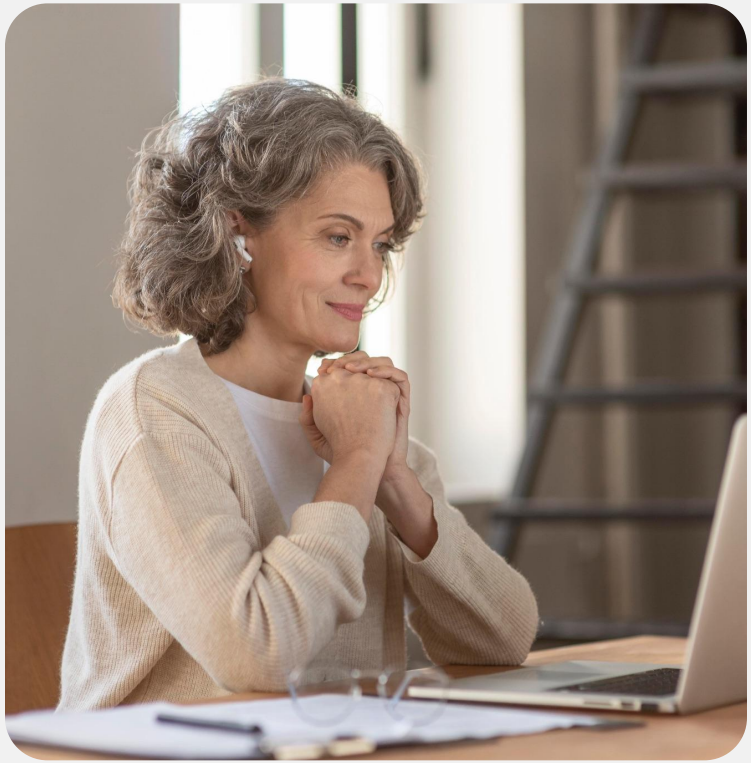
```python
def factorial_iterative(n: int) -> int:
    accumulator = 1
    while n > 1:
        accumulator *= n
        n -= 1
    return accumulator


print(factorial_iterative(5))
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЕ

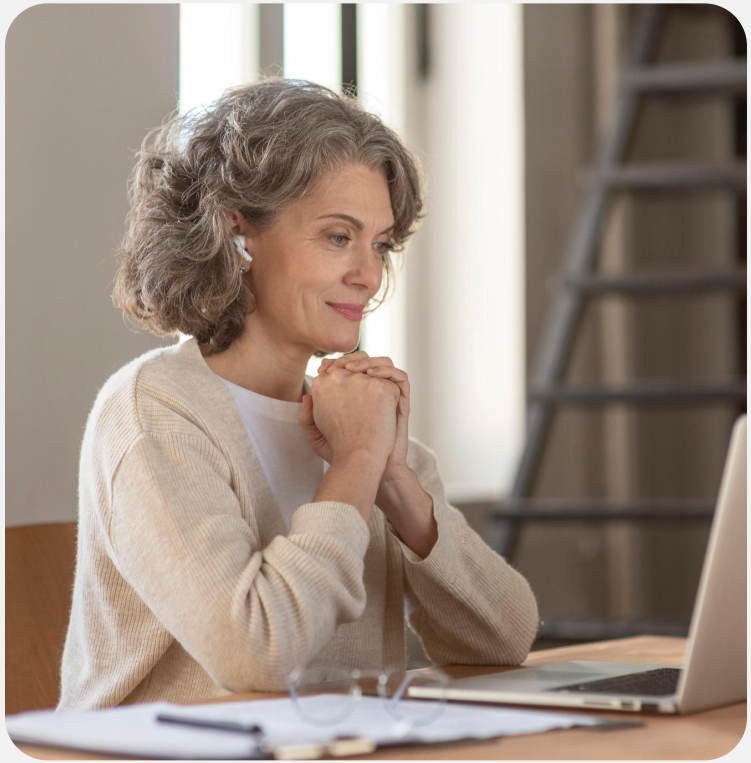


Выберите верный вариант ответа

Что произойдёт при выполнении следующего кода?

```
def infinite():

    return infinite()



infinite()
```

a. Бесконечный цикл
b. Ошибка RecursionError
c. Программа завершится без ошибок
d. Ошибка SyntaxError

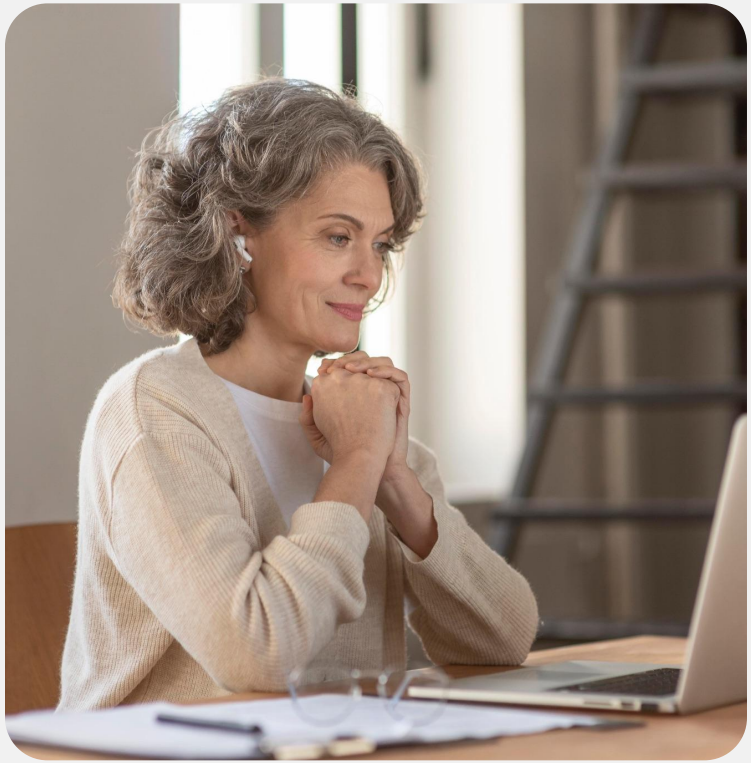


Выберите верный вариант ответа

Что произойдёт при выполнении следующего кода?

```
def infinite():

    return infinite()



infinite()
```

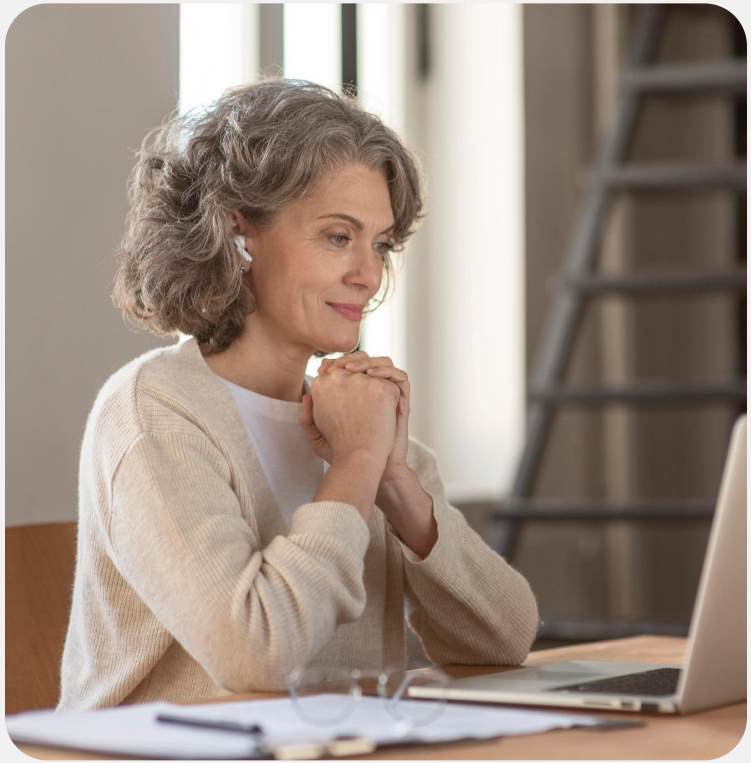
a. Бесконечный цикл
b. Ошибка RecursionError
c. Программа завершится без ошибок
d. Ошибка SyntaxError



Найдите верный ответ

У вас есть две функции. Чем они отличаются и что будет выведено в результате выполнения?

```
def print_numbers(n):
    if n == 0:
        return
    print(n)
    print_numbers(n - 1)
def print_nums(n):
    if n == 0:
        return
    print_nums(n - 1)
    print(n)
```



Найдите верный ответ

Обычная и хвостовая рекурсия. Разный порядок вывода чисел.

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Рекурсия или итерация?

Полезно знать

ICH
IT CAREER HUB

Рекурсия и итерация

Два подхода к решению задач, которые могут быть взаимозаменяемыми в некоторых задачах. Однако выбор между ними зависит от производительности, читабельности кода и ограничений памяти.

Преимущества и недостатки рекурсии

ICH
IT CAREER HUB

Плюсы	Минусы	Когда использовать рекурсию?	Когда использовать итерацию?
Позволяет разделить сложную задачу на подзадачи.	Может привести к переполнению стека (RecursionError).	Когда задача делится на подзадачи (разбиение на более мелкие части).	Когда рекурсия создаёт избыточную нагрузку на стек вызовов.
Упрощает чтение кода (например, обход деревьев и графов).	Медленнее итеративных решений из-за затрат на вызовы функций.	Для работы с деревьями, графами и вложенными структурами.	Когда можно решить задачу без хранения промежуточных вызовов.
Некоторые алгоритмы легче выразить рекурсивно (например, поиск пути в лабиринте).	Использует дополнительную память (каждый вызов создаёт новый стековый фрейм).	Когда важна выразительность кода (код читается легче, чем итеративная версия).	Когда важна эффективность по памяти и скорости.

Задачи, решаемые рекурсией

Некоторые задачи сложно или невозможно эффективно решить без использования рекурсии. Это связано с необходимостью разбиения задачи на более мелкие подзадачи, а также с работой со сложными вложенными структурами.

Обход дерева (DFS - поиск в глубину)

Дерево – это иерархическая структура данных, где каждый узел может иметь несколько дочерних узлов. Рекурсивный подход позволяет обойти все узлы, последовательно углубляясь в каждый из них.

Обход графа (DFS, рекурсивный поиск путей)

Граф – это структура, состоящая из узлов и рёбер, соединяющих их. Использование рекурсии в глубинном поиске позволяет эффективно находить пути между вершинами.

Разбор выражений и синтаксический анализ

При обработке вложенных математических выражений или языковых конструкций удобно использовать рекурсию, так как каждое подвыражение может рассматриваться как самостоятельная подзадача.

Ханойские башни

Задача, требующая последовательного перемещения дисков между тремя стержнями с сохранением их порядка. Разбивается на две меньшие аналогичные задачи, что делает рекурсию естественным решением.

Разбиение на подмножества и комбинаторика

Задачи, связанные с генерацией всех возможных комбинаций, перестановок и подмножеств множества, решаются с разбиением на подзадачи: выбор одного элемента и рекурсивное построение оставшейся структуры.

Функция deepcopy

Функция `deepcopy()` из модуля `copy` используется для создания глубокой копии объекта. В отличие от обычного `copy()`, который выполняет поверхностное копирование, `deepcopy()` рекурсивно копирует все вложенные объекты, создавая полностью независимую копию.

Преимущества и недостатки рекурсии

IT CAREER HUB

Операция	Что копируется?	Вложенные изменяемые объекты
`copy()`	Поверхностный уровень	Сохраняют ссылки на оригинальные объекты
`deepcopy()`	Полная независимая копия	Создаются новые копии вложенных объектов

Пример с copy() (поверхностная копия):

```python
original_list = [[1, 2], [3, 4]]

# Поверхностная копия, вложенные коллекции скопированы как ссылки

copy_lst = original_list.copy()

# Добавляет в копию, не затрагивает вложенные элементы.

copy_lst.append(99)

# Изменяет копию списка, но затрагивает вложенные элементы.

copy_lst[0][0] = "X"  # Влияет на оригинал!

print("Оригинал:", original_list)

print("Копия:", copy_lst)
```

Пример с deepcopy() (глубокая копия):

```python
from copy import deepcopy

original_list = [[1, 2], [3, 4]]

# Глубокая копия, для вложенных коллекций созданы дубликаты объектов

copy_lst = deepcopy(original_list)

# Добавляет в копию, не затрагивает вложенные элементы.

copy_lst.append(99)

# Изменяет вложенные элементы, которые не связаны с изначальным списком.

copy_lst[0][0] = "X"  # Не влияет на оригинал!

print("Оригинал:", original_list)

print("Копия:", copy_lst)
```

Когда deepcopy() не нужен?

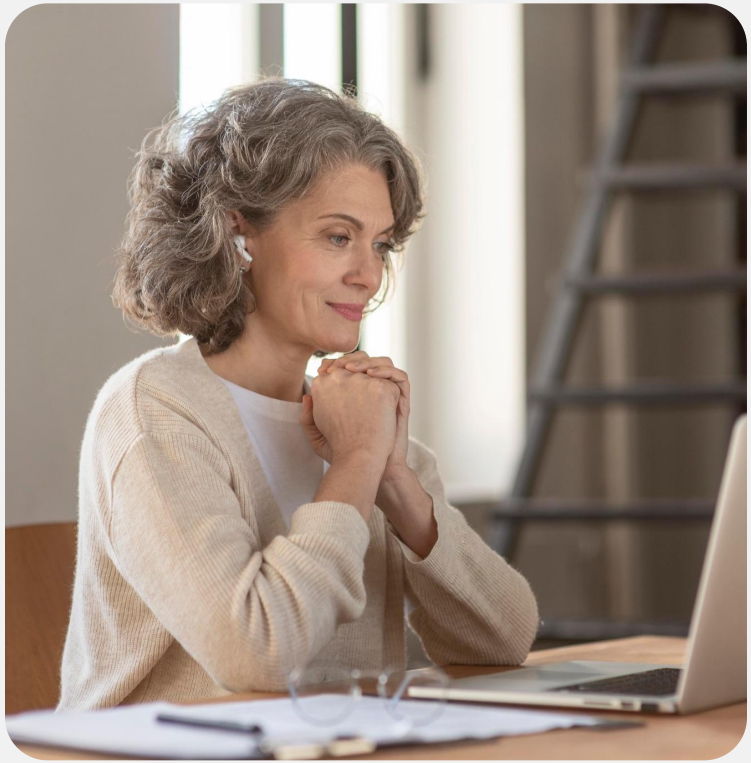
- Если объект не содержит изменяемых вложенных объектов.
- Если производительность критична — `deepcopy()` медленнее, чем `copy()`, а копируемые данные не требуют полного дублирования.

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

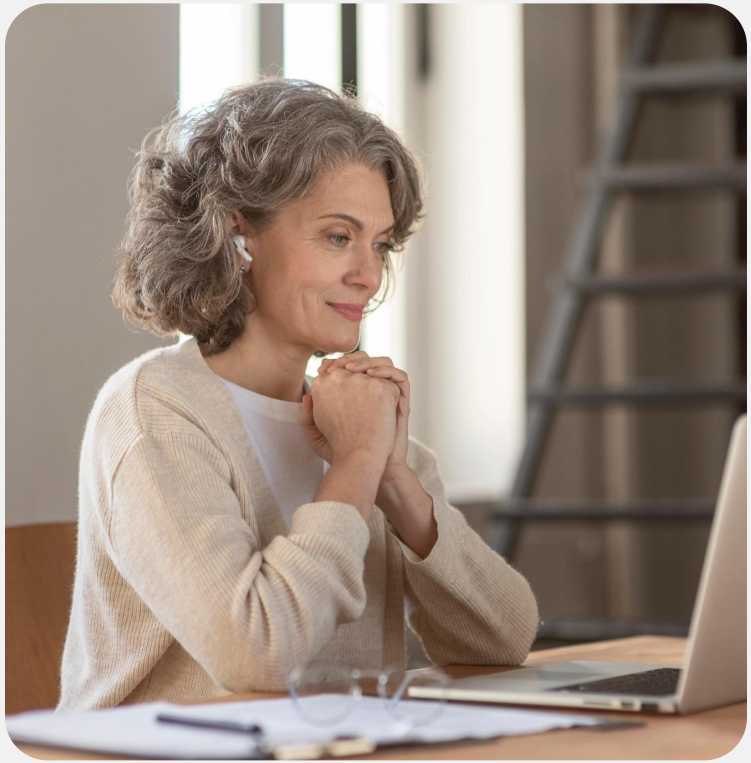
ЗАДАНИЕ



Выберите правильный вариант ответа

Чем `deepcopy()` отличается от `copy()`?

a. `copy()` создаёт полную независимую копию, включая вложенные объекты
b. `copy()` всегда создаёт новый объект без ссылок на оригинал
c. `deepcopy()` создаёт полную независимую копию, включая вложенные объекты
d. `deepcopy()` выполняет копирование быстрее, чем `copy()`



Выберите правильный вариант ответа

Чем `deepcopy()` отличается от `copy()`?

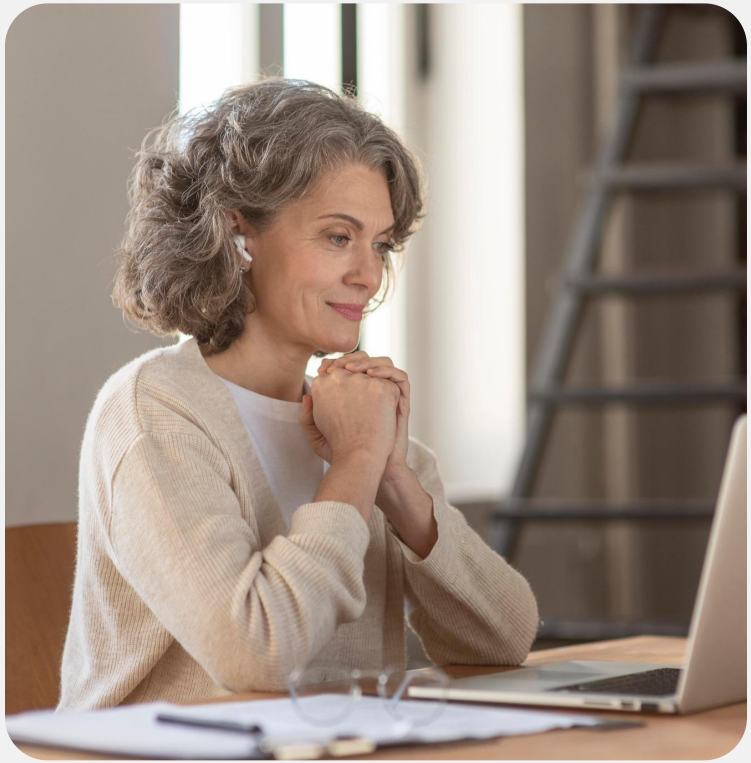
a. `copy()` создаёт полную независимую копию, включая вложенные объекты
b. `copy()` всегда создаёт новый объект без ссылок на оригинал
c. **`deepcopy()` создаёт полную независимую копию, включая вложенные объекты**
d. `deepcopy()` выполняет копирование быстрее, чем `copy()`

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ПРАКТИЧЕСКИЕ ЗАДАНИЯ



Функция deepcopy

Реализуйте аналог `deepcopy()` с помощью рекурсии. Не забудьте проверить, чтобы изменения в копии не затронули оригинал.

Данные:

```
original_data = [

    [1, 2, 3],                  # Вложенный список
    (4, [5, 6], {7, 8}),        # Кортеж с вложенными
структурами
    {"a": 9, "b": [10, 11]},    # Словарь со списком
    "Hello",                    # Строка
    [12, (13, 14)],             # Список с кортежем
    15.5,                       # Число с плавающей точкой
    5                           # Целое число
]
```

Домашнее задание

Сумма цифр числа

Напишите рекурсивную функцию, которая находит сумму всех цифр числа.

Данные:

num = 43197

Пример вывода:

24

Домашнее задание

Сумма вложенных чисел

Напишите рекурсивную функцию, которая суммирует все числа во вложенных списках.

Данные:

nested_numbers = [1, [2, 3], [4, [5, 6]], 7]

Пример вывода:

28

Заключение