Отчёт технического интервью

Кандидат: Candidate_1

Задач в сессии: 3 (Python)

1. История выполнения задач

Попытка #1

Задача: py-easy-5

Время выполнения: 38 ms

stdout: Сумма списка [1, 2, 3]: 6 Сумма списка [10, 20, 5, -5]: 30

stderr:

Попытка #2

Задача: py-med-1

Время выполнения: 49 ms

stdout:

stderr: File "C:\Users\woonze\AppData\Local\Temp\tmp0bwfo_n3.py", line 1 1. Итеративная функция (рекомендуется) ^^^^^^^^^^ SyntaxError: invalid syntax

Попытка #3

Задача: py-med-1

Время выполнения: 35 ms

stdout: fib_iterative(10): 55 fib_iterative(5): 5

stderr:

Попытка #4

Задача: py-hard-9

Время выполнения: 41 ms

stdout: Ошибка вычисления: Недопустимый символ в выражении. 3 * (4 + 5) / 2 = None Ошибка вычисления: Недопустимый символ в выражении. 10 / 2 + 3 = None Ошибка вычисления: Недопустимый символ в выражении. (1 + 2) * 3 = None

stderr:

2. Решения кандидата по задачам

Задача py-easy-5 — Сумма списка

Реализуйте функцию sum_list(nums), возвращающую сумму элементов.

```
def sum_list(nums: list) -> int:
    """
```

```
    Возвращает сумму элементов списка чисел.
    """
    total = 0
    # Проходим по каждому элементу в списке и добавляем его к общей сумме
    for num in nums:
        total += num
    return total
# Пример использования:
numbers1 = [1, 2, 3]
print(f"Сумма списка {numbers1}: {sum_list(numbers1)}") # Вывод: 6
numbers2 = [10, 20, 5, -5]
print(f"Сумма списка {numbers2}: {sum_list(numbers2)}") # Вывод: 30
```

Задача py-med-1 — Фибоначчи

Реализуйте рекурсивную или итеративную функцию fib(n).

```
def fib_iterative(n: int) -> int:
    """
    Вычисляет n-ное число Фибоначчи итеративным методом.
    Ряд Фибоначчи: 0, 1, 1, 2, 3, 5... (F(0)=0, F(1)=1)
    """
    if n < 0:
        raise ValueError("n должно быть неотрицательным целым числом")
    if n == 0:
        return 0
    # Инициализируем первые два числа в последовательности
    a, b = 0, 1
    # Проходим циклом от 2 до n
    for _ in range(2, n + 1):
        # Следующее число - это сумма двух предыдущих
        a, b = b, a + b
    return b
# Пример использования итеративной функции:
print(f"fib_iterative(10): {fib_iterative(10)}") # Вывод: 55
print(f"fib_iterative(5): {fib_iterative(5)}")   # Вывод: 5
```

Задача py-hard-9 — Парсер выражений

Реализуйте простой калькулятор: + - * / и скобки.

```
def calculate_eval(expression: str) -> float:
    try:
        allowed_chars = "0123456789.+-*/()"
        for char in expression:
            if char not in allowed_chars:
                raise ValueError("Недопустимый символ в выражении.")
        result = eval(expression)
        return float(result)
    except (SyntaxError, ZeroDivisionError, ValueError) as e:
        print(f"Ошибка вычисления: {e}")
        return None
print(f"3 * (4 + 5) / 2 = {calculate_eval('3 * (4 + 5) / 2')}") # 13.5
print(f"10 / 2 + 3 = {calculate_eval('10 / 2 + 3')}")          # 8.0
print(f"(1 + 2) * 3 = {calculate_eval('(1 + 2) * 3')}")        # 9.0
```

3. Анализ решений по задачам (LLM)

Задача py-easy-5 — оценка: 92.0/100

Комментарий: Итоговая оценка за задачу: 92.0/100 Оценка кода: 95/100 Код корректно реализует функцию суммирования элементов списка. Использован простой и понятный подход с циклом for. Есть документация и примеры использования. Единственное улучшение - можно добавить проверку на пустой список или тип данных, но это не критично. Оценка коммуникации: 85/100 Решение корректно использует встроенную функцию sum() для вычисления суммы элементов списка. Код простой, понятный и выполняет поставленную задачу. Однако отсутствует обработка возможных ошибок (например, если на вход подается не список или список содержит не числа), что снижает robustness решения.

Замечаний не зафиксировано.

Задача py-med-1 — оценка: 66.5/100

Комментарий: Итоговая оценка за задачу: 66.5/100 Оценка кода: 95/100 Код корректно реализует итеративный алгоритм вычисления чисел Фибоначчи с правильной обработкой граничных случаев. Ошибок в логике нет, код читаемый и эффективный. Единственное улучшение — добавить проверку на тип входного параметра для большей надежности. Оценка коммуникации: 0/100 Код не представлен, только описание подходов. Ответ не содержит реализации алгоритма, поэтому оценка 0.

Замечаний не зафиксировано.

Задача py-hard-9 — оценка: 0.0/100

Комментарий: Итоговая оценка за задачу: 0.0/100 Оценка кода: 0/100 Код не работает из-за использования функции eval(), которая может быть опасной и не безопасна для выполнения произвольного кода. Также в коде отсутствует обработка ошибок при некорректном выражении, что приводит к возврату None вместо ожидаемого результата. Оценка коммуникации: 0/100 Код не является синтаксически корректным и не может быть выполнен. Отсутствует закрывающая скобка и операторы для выполнения логики.

Замечаний не зафиксировано.

4. Итоговая оценка кандидата

Средняя оценка: 52.83/100. Сильные стороны: Хорошая оптимизация и эффективность решений, Хорошая читаемость и структура кода. Слабые стороны: не выявлены. Рекомендации: развивать алгоритмическое мышление, покрывать код тестами и уделять больше внимания структуре и читаемости решений.

5. Коммуникативные ответы кандидата

Задача: py-easy-5

Вопрос: Какие альтернативные способы решения возможны?

Ответ кандидата: Можно сделать так: def sum_list_built_in(nums: list) -> int: """ Возвращает сумму элементов списка чисел, используя встроенную функцию sum(). """ return sum(nums) # Пример использования встроенного метода: numbers3 = [4, 5, 6] print(f"Сумма списка {numbers3} (built-in): {sum_list_built_in(numbers3)}") # Вывод: 15

Оценка коммуникации: 85/100

Решение корректно использует встроенную функцию sum() для вычисления суммы элементов списка. Код простой, понятный и выполняет поставленную задачу. Однако отсутствует обработка возможных ошибок (например, если на вход подается не список или список содержит не числа), что снижает robustness решения.

Задача: py-med-1

Вопрос: Можешь кратко описать логику своего решения?

Ответ кандидата: 1. Итеративный подход (рекомендуемый) Логика этого подхода основана на накоплении результата с помощью цикла, что эффективно по памяти и скорости: Инициализация: Мы начинаем с двух переменных, a и b, установленных на первые два числа Фибоначчи: 0 и 1.Цикл: Мы запускаем цикл от 2 до n.Обновление: На каждой итерации мы вычисляем следующее число Фибоначчи как сумму a и b, а затем сдвигаем "окно" вперед: новое a становится старым b, а новое b становится только что вычисленной суммой.Результат: После завершения цикла переменная b содержит n-ное число Фибоначчи. 2. Рекурсивный подход Логика этого подхода основана на прямом следовании математическому определению функции \(F(n)=F(n-1)+F(n-2)\): Базовые случаи: Определены условия остановки рекурсии (F(0) = 0 и F(1) = 1). Без них функция зациклится.Рекурсивный вызов: Для любого n > 1 функция просто вызывает саму себя для n-1 и n-2 и складывает результаты. Ключевое отличие: Итеративный метод запоминает предыдущие шаги и выполняет вычисления линейно, а рекурсивный метод пересчитывает одни и те же значения многократно (например, для fib(5) он дважды считает fib(3), трижды fib(2) и т.д.), что делает его медленным. Создание общедоступной ссылки…

Оценка коммуникации: 0/100

Код не представлен, только описание подходов. Ответ не содержит реализации алгоритма, поэтому оценка 0.

Задача: py-hard-9

Вопрос: Можешь кратко описать логику своего решения?

Ответ кандидата: F[f[f[ff[

Оценка коммуникации: 0/100

Код не является синтаксически корректным и не может быть выполнен. Отсутствует закрывающая скобка и операторы для выполнения логики.