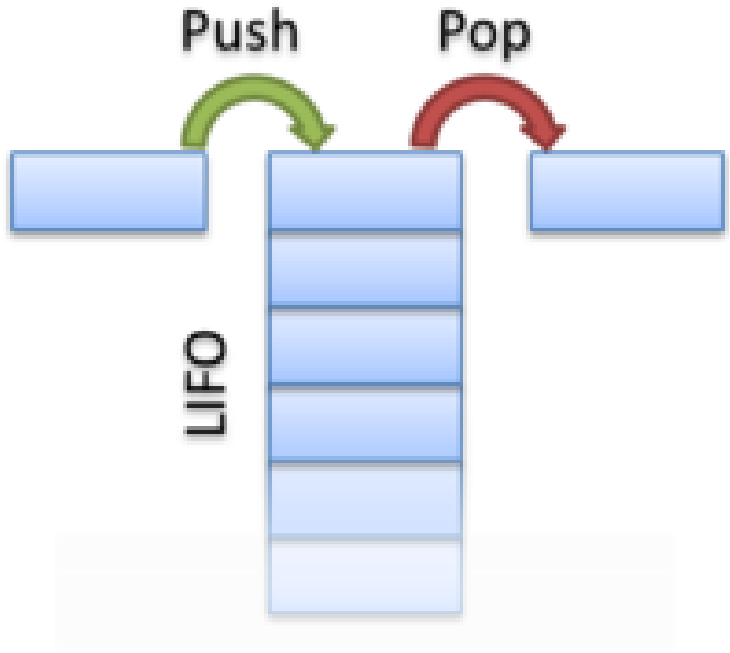
Задачи Стек (Stack)

Стек (Stack) - LIFO

- Стек — это линейная структура данных, работающая по принципу «последним пришёл — первым ушёл» (LIFO - Last In, First Out)
- Добавление новых элементов и удаление существующих происходит только с одного конца, называемого вершиной стека, что можно сравнить со стопкой тарелок или колодой карт.



Основные принципы и операции

- **Принцип LIFO**: Элемент, который был добавлен в стек последним, будет первым из него извлечён.
- **Вершина стека**: Это единственный конец, где можно выполнять операции. Доступ к другим элементам напрямую невозможен.
- **Основные операции**:
  - **Push (Вставка)**: Добавление нового элемента на вершину стека.
  - **Pop (Удаление)**: Извлечение и удаление верхнего элемента из стека.
  - **Peek (Просмотр)**: Возвращение верхнего элемента без его удаления.
  - **IsEmpty (Проверка на пустоту)**: Проверка, пуст ли стек.

Задача: Реализуй стек на Python

Цель: Создать класс Stack, который поддерживает основные операции со стеком:

- Добавление элемента в верхушку стека (**push**)
- Удаление и возврат верхнего элемента (**pop**)
- Просмотр верхнего элемента без удаления (**peek** или **top**)
- Проверка, пуст ли стек (**is_empty**)
- Получение размера стека (**size**)

Дополнение к предыдущей задаче

- К классу стек **Stack** добавьте метод **get_min()** который возвращает минимальный элемент в стеке.

Дополнение к предыдущей задаче

- К классу стек **Stack** добавьте метод **get_min()** который возвращает минимальный элемент в стеке.

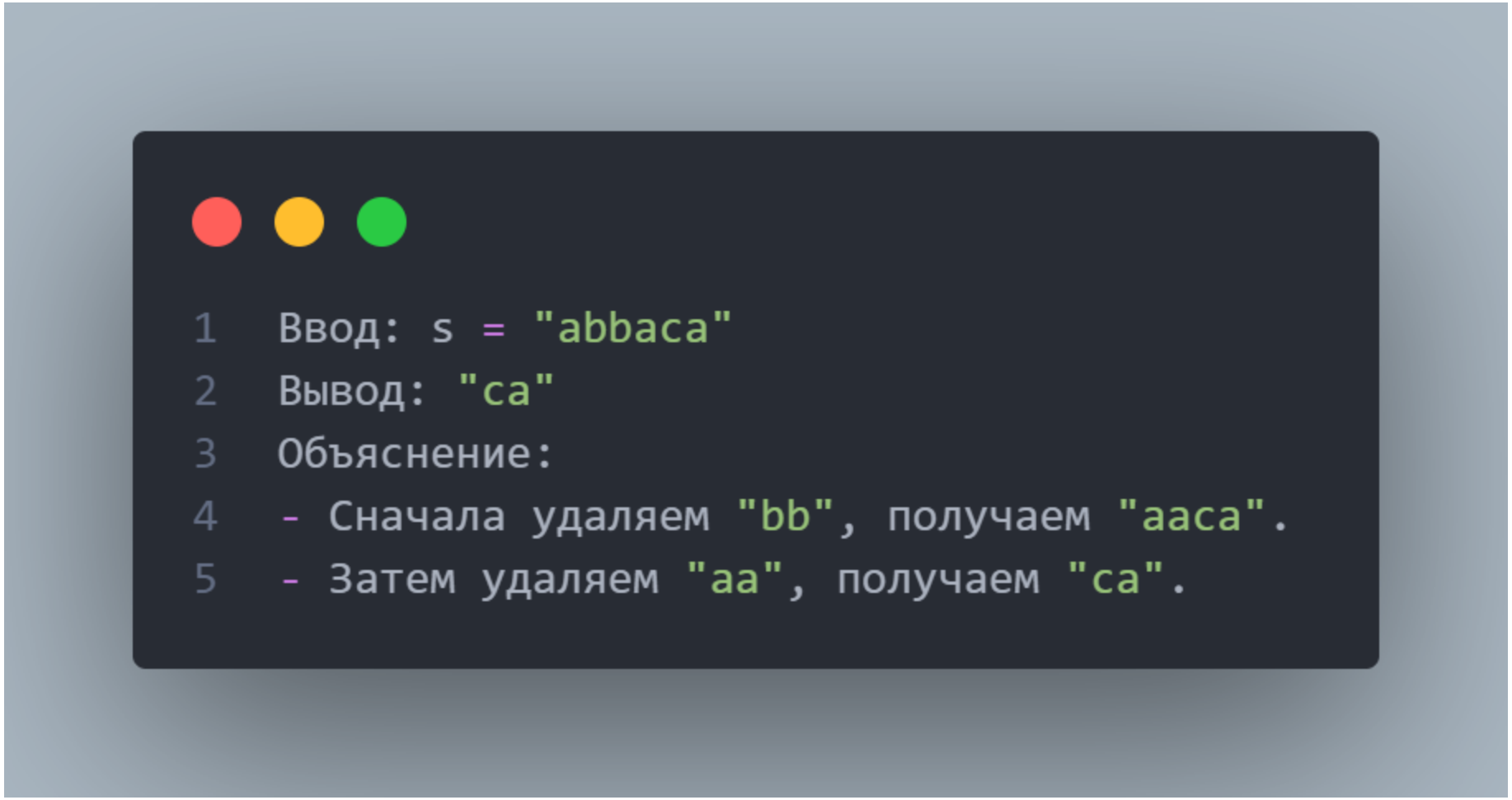
  Примечание: реализуйте метод **get_min()** так чтобы работал в **O(1)** времени

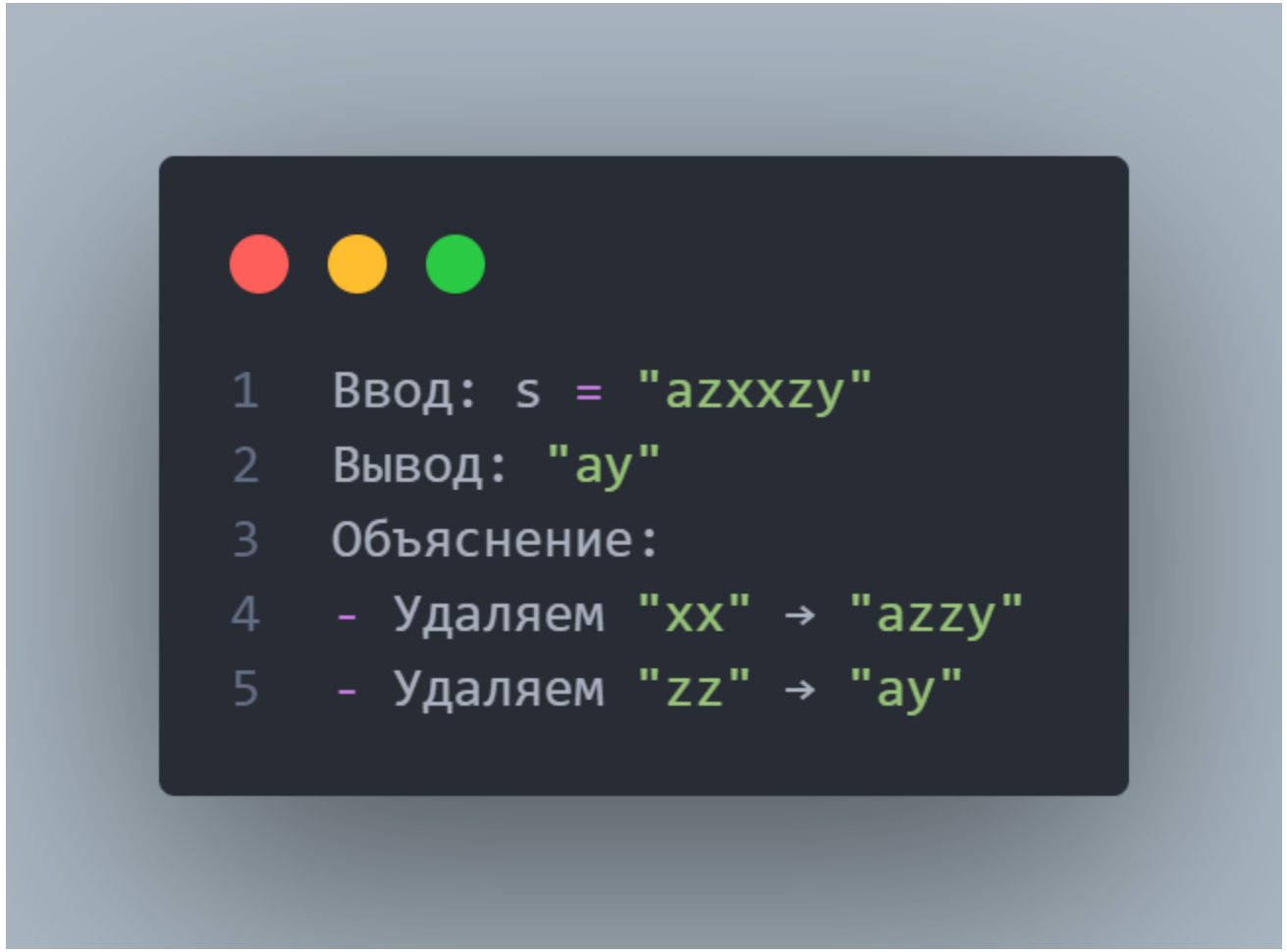
Задача 2: Удаление всех смежных дубликатов в строке

Дана строка **s**, состоящая из строчных латинских букв.
За одну операцию мы можем удалить два одинаковых соседних символа.

После удаления таких символов строка может образовать новые пары одинаковых соседних символов — их тоже нужно удалять. Процесс продолжается до тех пор, пока больше нет соседних одинаковых символов.

Верни окончательную строку после всех возможных удалений. Гарантируется, что результат уникален.


Ввод: s = "abbaca"
Вывод: "ca"
Объяснение:
- Сначала удаляем "bb", получаем "aaca".
- Затем удаляем "aa", получаем "ca".


Ввод: s = "azxxzy"
Вывод: "ay"
Объяснение:
- Удаляем "xx" → "azzy"
- Удаляем "zz" → "ay"

Задача: Окончательные цены со специальной скидкой в магазине

Дан целочисленный массив prices, где prices[i] — цена i-го товара в магазине.

В магазине действует специальное предложение: для каждого товара ты можешь получить скидку, равную цене ближайшего следующего товара, у которого цена меньше или равна текущей.

Если такого следующего товара нет, то скидка не применяется, и ты платишь полную цену.

Верни массив answer, где answer[i] — окончательная цена на i-й товар после применения скидки.

Пример

```
Ввод: prices = [8,4,6,2,3]
Вывод: [4,2,4,2,3]

Объяснение:
- Для товара 0 (цена 8): ближайший следующий товар со скидкой — цена 4 (индекс 1) → 8 - 4 = 4
- Для товара 1 (цена 4): следующая цена ≤ 4 — это 2 → 4 - 2 = 2
- Для товара 2 (цена 6): следующая цена ≤ 6 — это 2 → 6 - 2 = 4
- Для товара 3 (цена 2): нет следующей цены ≤ 2 → остаётся 2
- Для товара 4 (цена 3): последний товар → остаётся 3
```

Пример

```
1  Ввод: prices = [1,2,3,4,5]
2  Вывод: [1,2,3,4,5]
3  Объяснение: Нет товара с меньшей или равной ценой после любого из них → без скидок.
```

Пример

```
Ввод: prices = [10,1,1,6]
Вывод: [9,0,1,6]
Объяснение:
- 10 - 1 = 9 (берём первый товар со скидкой)
- 1 - 1 = 0 (следующая единица)
- 1 → нет следующей ≤ 1 → остаётся 1
- 6 → остаётся 6
```

Задача Правильные скобки

На вход подаётся одна строка с кодом программы, нужно проверить правильность расстановки парных скобок. Возможные пары скобок: **[ ], { }, ( ).**

- Эту задачу следует решить с использованием структуры данных **stack**.
- Если на Python, то с использованием методов списка **append** и **pop**.

Программа должна возвращать **True** или **False**

Задача Вычисление обратной польской нотации (ОПН)

Вам дан массив строк tokens, представляющий собой арифметическое выражение в обратной польской записи .

Вычислите выражение. Верните целое число, представляющее значение выражения .

Обратите внимание, что:

- Допустимые операторы: '+', '-', '*', и '/'.
- Каждый операнд может быть целым числом или другим выражением.
- Деление двух целых чисел всегда приводит к округлению в сторону нуля .
- Деления на ноль не будет.
- Входные данные представляют собой допустимое арифметическое выражение в обратной польской записи.
- Ответ и все промежуточные вычисления можно представить в виде 32-битного целого числа.

Примеры

```
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

```
Input: tokens = ["4","13","5","/","+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

```
Input: tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
Output: 22
Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
```

Алгоритм

- Создайте пустой стек.
- Читайте выражение (список токенов) слева направо.
- Если токен — это **число**, поместите его в стек.
- Если токен — это **оператор** (+, -, *, /):
- Извлеките из стека два последних числа.
- Выполните над ними операцию.
- Положите результат обратно в стек.
- Когда выражение закончится, единственное число в стеке и будет ответом.