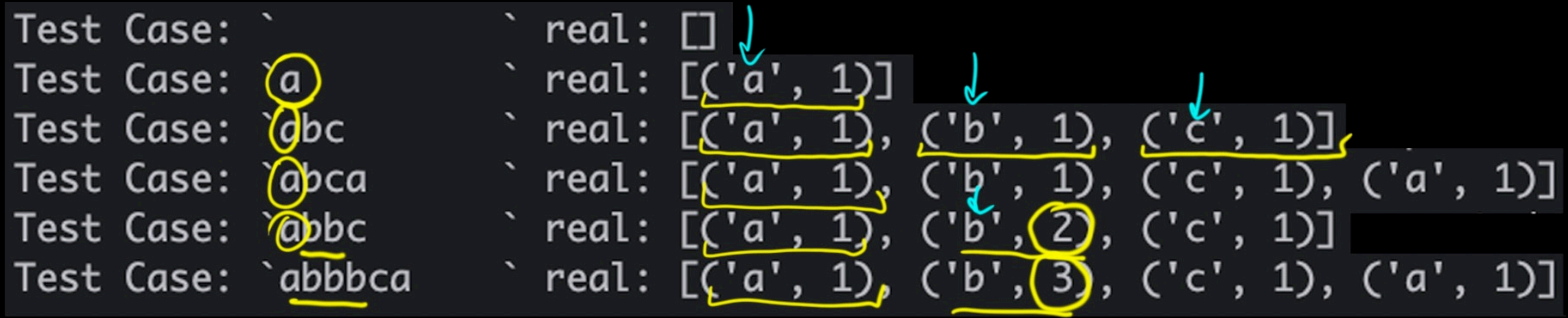
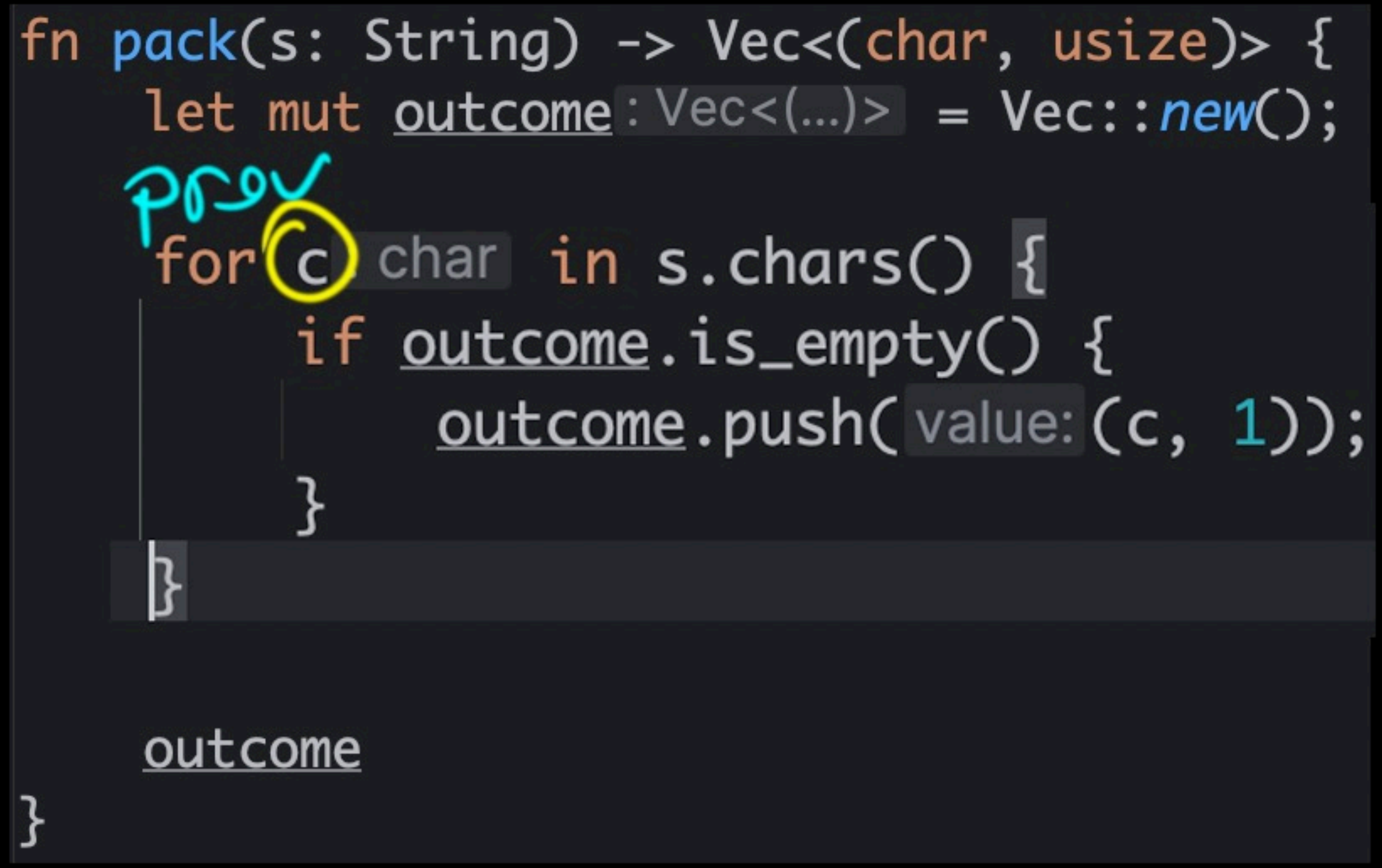


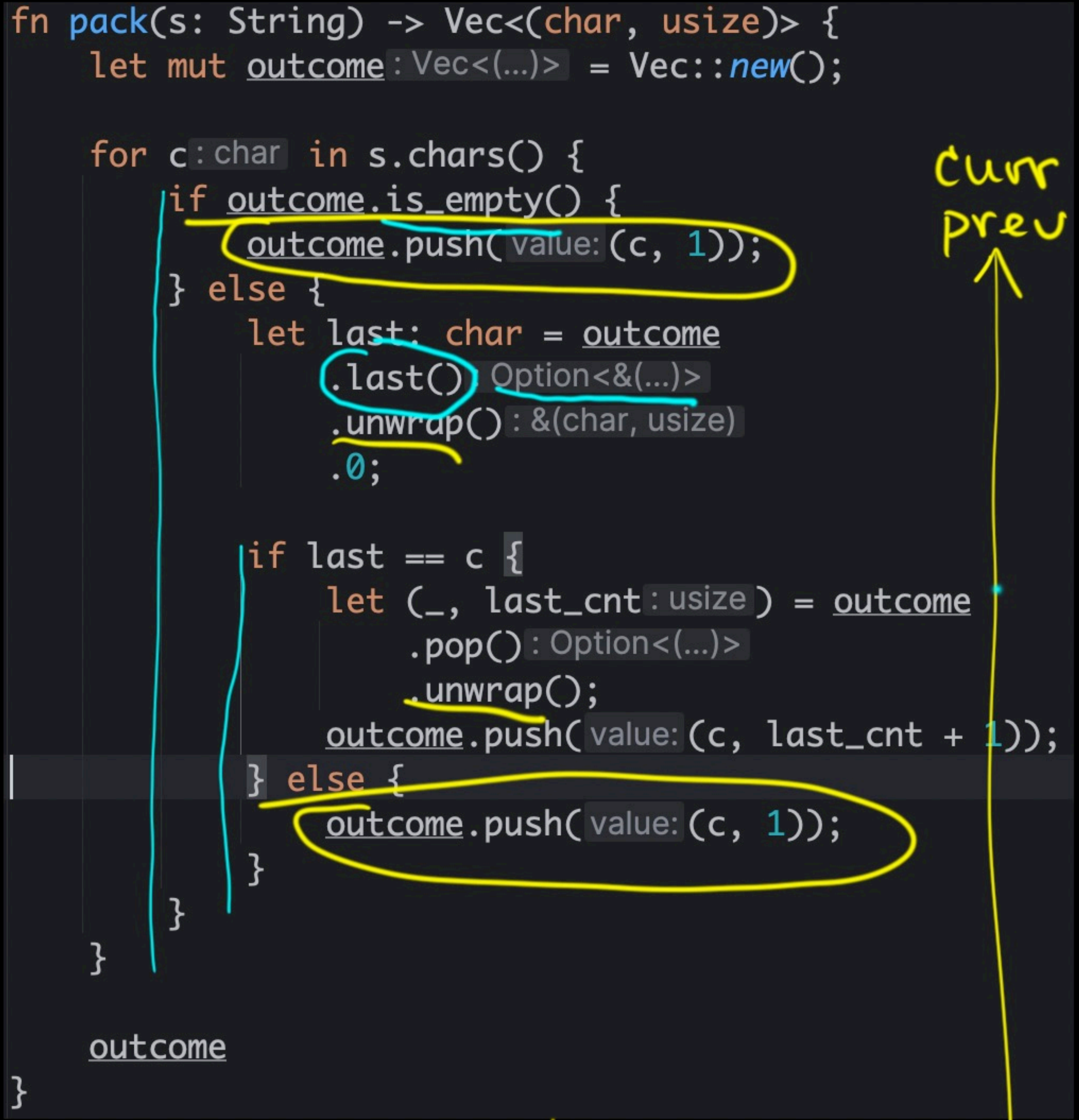
```
Test Case: `      ` real: []
Test Case: `a     ` real: [('a', 1)]
Test Case: `abc   ` real: [('a', 1), ('b', 1), ('c', 1)]
Test Case: `abca  ` real: [('a', 1), ('b', 1), ('c', 1), ('a', 1)]
Test Case: `abbc  ` real: [('a', 1), ('b', 2), ('c', 1)]
Test Case: `abbbca` real: [('a', 1), ('b', 3), ('c', 1), ('a', 1)]
```

pack(s: String) -> Vec<(char, usize)>



prev

```rust
fn pack(s: String) -> Vec<(char, usize)> {
    let mut outcome: Vec<(...)> = Vec::new();
    for c: char in s.chars() {
        if outcome.is_empty() {
            outcome.push(value: (c, 1));
        }
    }

    outcome
}
```

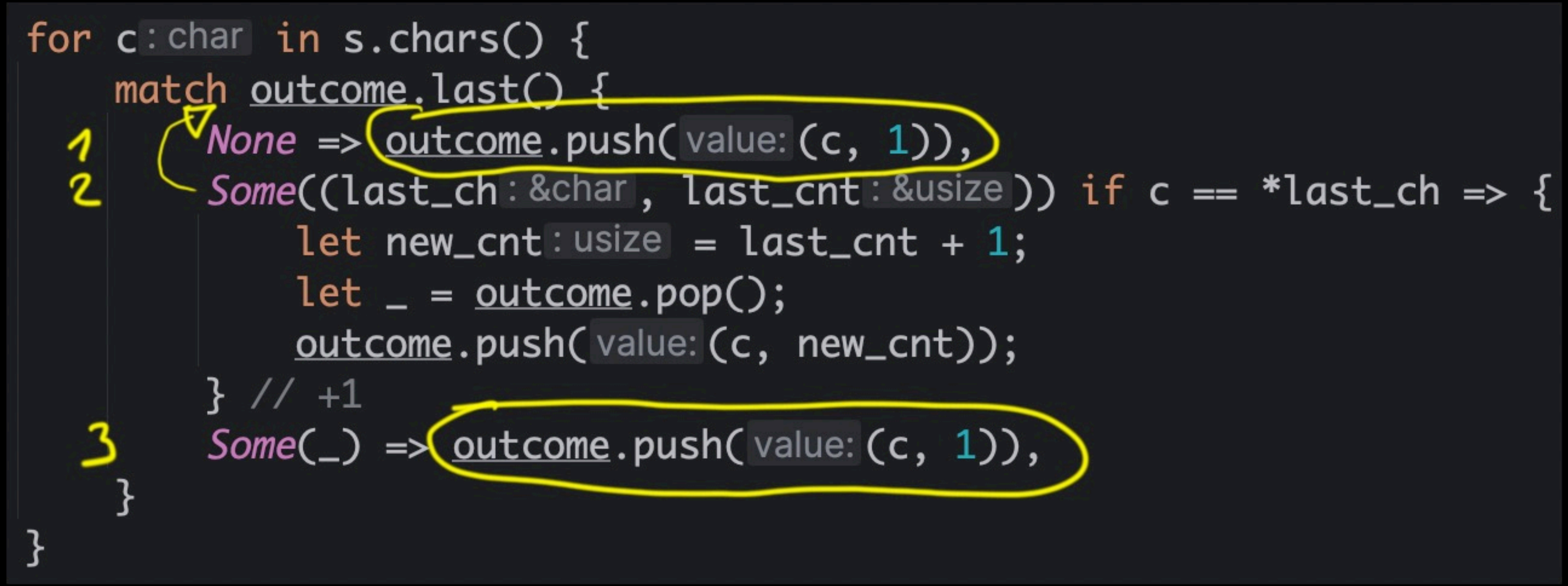


```
fn pack(s: String) -> Vec<(char, usize)> {
    let mut outcome: Vec<(...)> = Vec::new();

    for c: char in s.chars() {
        if outcome.is_empty() {
            outcome.push(value: (c, 1));
        } else {
            let last: char = outcome
                .last(): Option<&(...)>
                .unwrap(): &(char, usize)
                .0;

            if last == c {
                let (_, last_cnt: usize) = outcome
                    .pop(): Option<(...)>
                    .unwrap();
                outcome.push(value: (c, last_cnt + 1));
            } else {
                outcome.push(value: (c, 1));
            }
        }
    }

    outcome
}
```

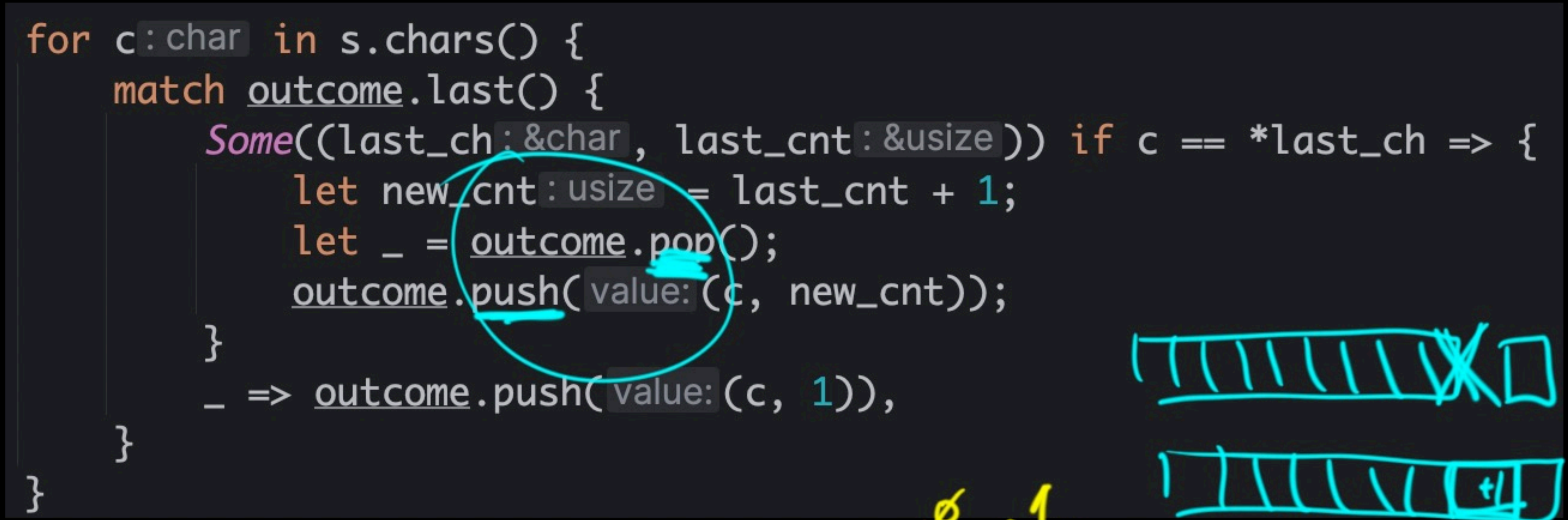
curr
prev

Option<char>

curr=b
prev=Some(a)
abbbca
curr=a
prev=None

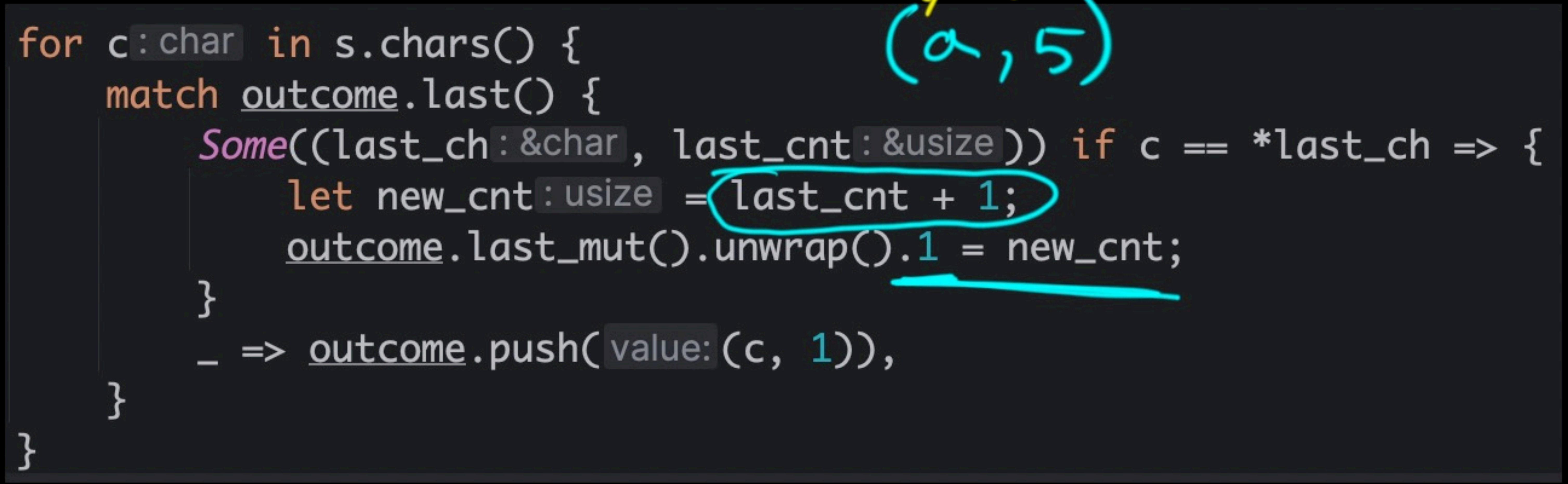


```
for c: char in s.chars() {
    match outcome.last() {
1       None => outcome.push(value: (c, 1)),
2       Some((last_ch: &char, last_cnt: &usize)) if c == *last_ch => {
            let new_cnt: usize = last_cnt + 1;
            let _ = outcome.pop();
            outcome.push(value: (c, new_cnt));
        } // +1
3       Some(_) => outcome.push(value: (c, 1)),
    }
}
```

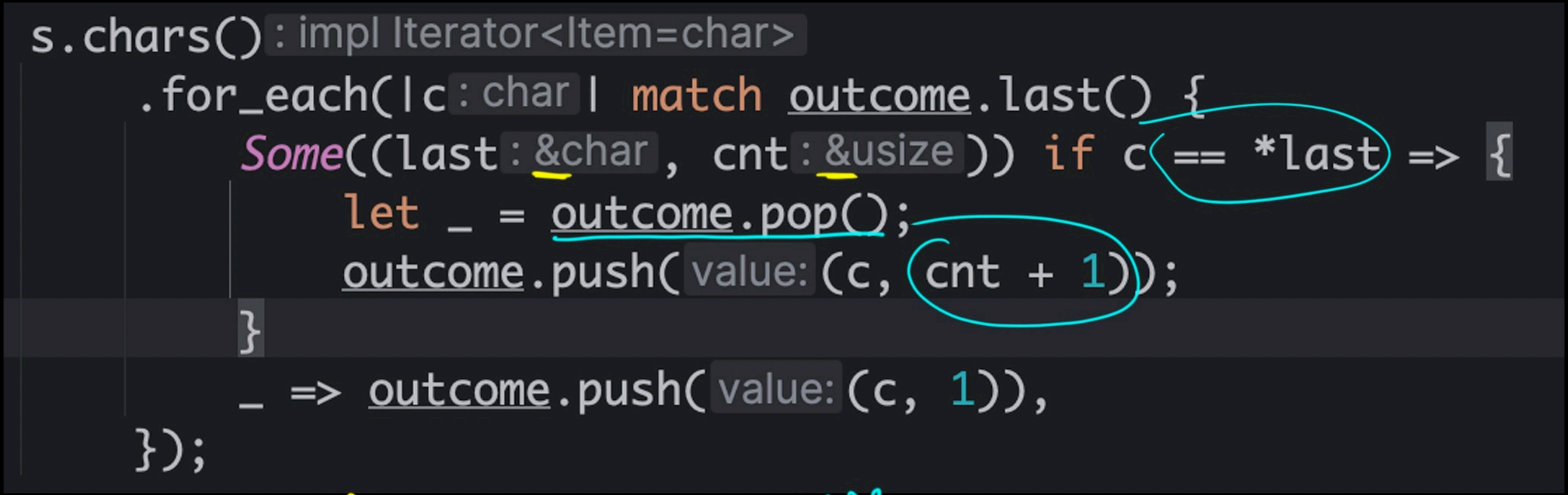


```
for c: char in s.chars() {
    match outcome.last() {
        Some((last_ch: &char, last_cnt: &usize)) if c == *last_ch => {
            let new_cnt: usize = last_cnt + 1;
            let _ = outcome.pop();
            outcome.push(value: (c, new_cnt));
        }
        _ => outcome.push(value: (c, 1)),
    }
}
```

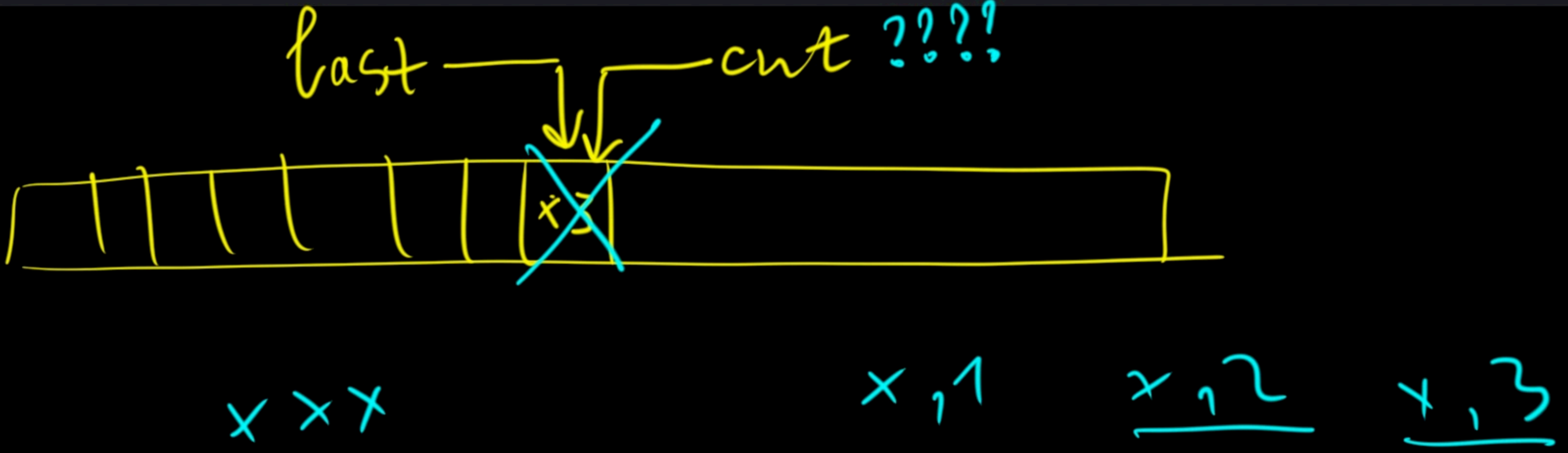
ø .1
(a, 5)

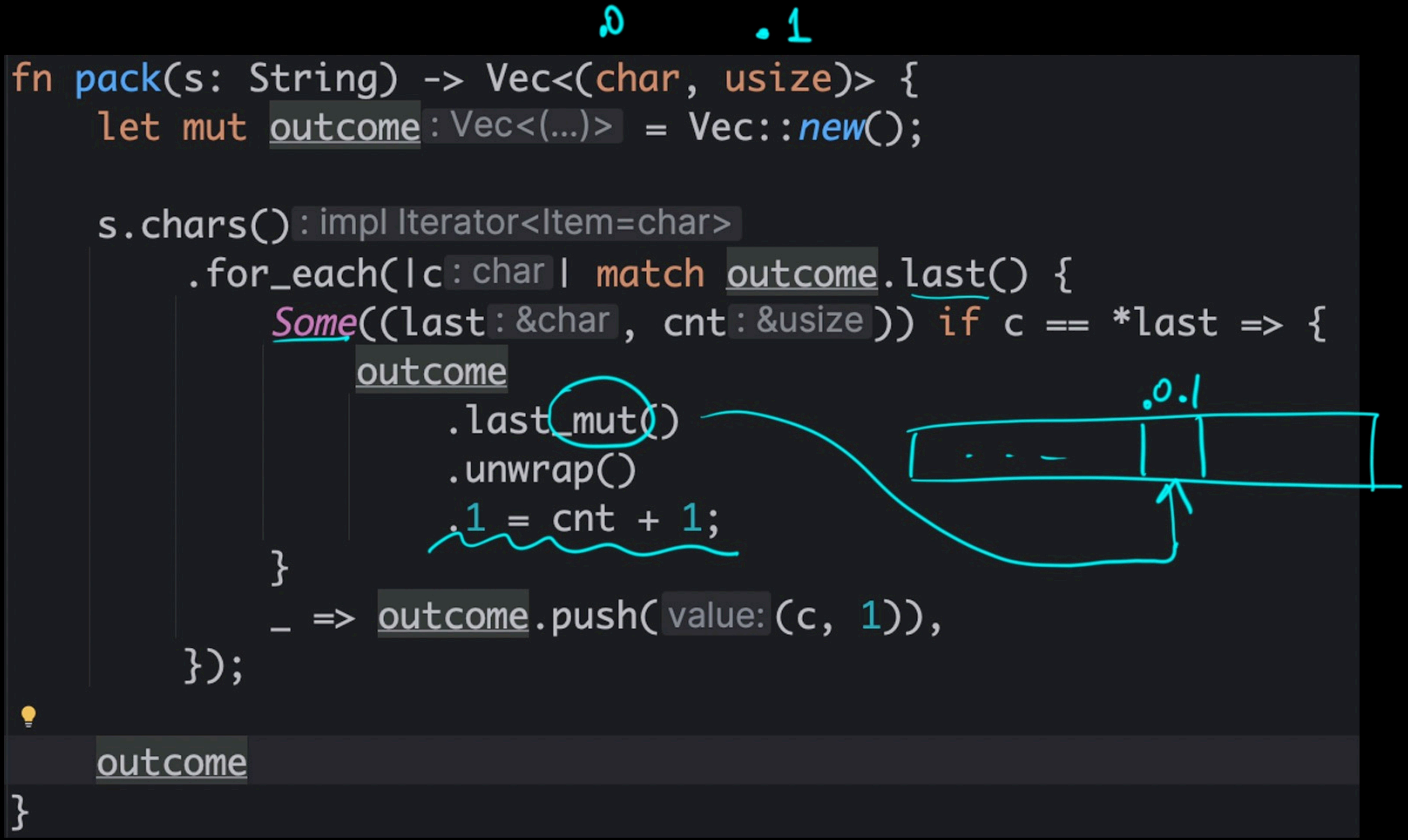


```
for c: char in s.chars() {
    match outcome.last() {
        Some((last_ch: &char, last_cnt: &usize)) if c == *last_ch => {
            let new_cnt: usize = last_cnt + 1;
            outcome.last_mut().unwrap().1 = new_cnt;
        }
        _ => outcome.push(value: (c, 1)),
    }
}
```



```
s.chars() : impl Iterator<Item=char>
    .for_each(|c : char | match outcome.last() {
        Some((last : &char , cnt : &usize )) if c == *last => {
            let _ = outcome.pop();
            outcome.push( value: (c, cnt + 1));
        }
        _ => outcome.push( value: (c, 1)),
    });
```





```
fn pack(s: String) -> Vec<(char, usize)> {
    let mut outcome: Vec<(...)> = Vec::new();

    s.chars(): impl Iterator<Item=char>
        .for_each(|c: char| match outcome.last() {
            Some((last: &char, cnt: &usize)) if c == *last => {
                outcome
                    .last_mut()
                    .unwrap()
                    .1 = cnt + 1;
            }
            _ => outcome.push(value: (c, 1)),
        });

    outcome
}
```