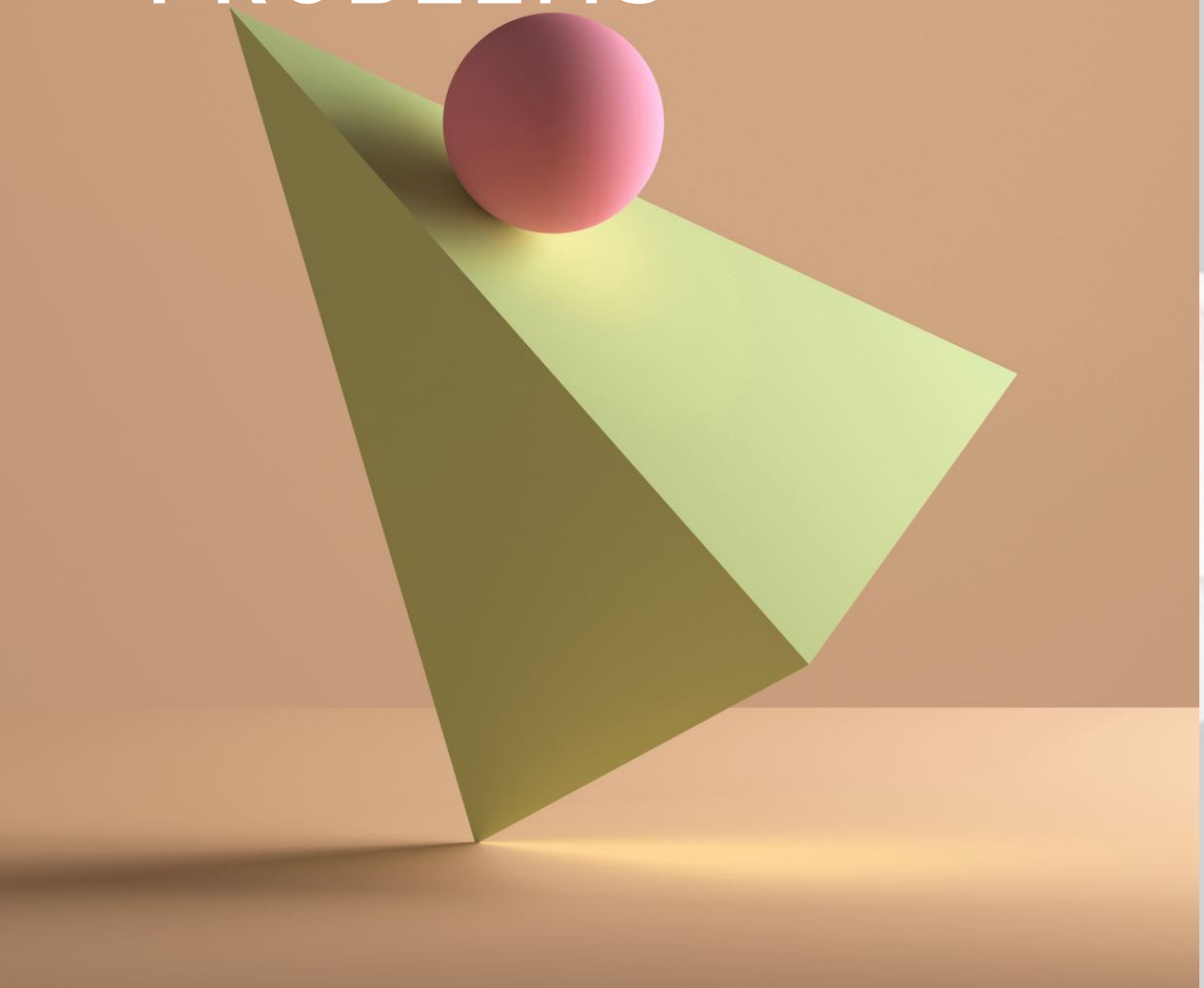
RUST PRACTICE PROBLEMS



PROBLEM

Filter even numbers: Write a function that takes an iterator over integers and returns a new iterator that yields only the even numbers. Test your function with a range of numbers and a vector of integers.

SOLUTION

```rust
fn filter_even_nos<I: Iterator<Item = u32>>(iter: I) -> impl Iterator<Item = u32> {
    iter.filter(|&x| x % 2 == 0)
}
```

TESTING

```
#[test]
fn test_some_range() {
    let iter = 1..=6;
    assert_eq!(filter_even_nos(iter).collect::<Vec<u32>>(), [2, 4,
6]);

#[test]
fn test_some_vec() {
    let v1: Vec<u32> = vec![1, 2, 3, 4, 5, 6];
    assert_eq!(
        filter_even_nos(v1.iter().cloned()).collect::<Vec<u32>>(),
        [2, 4, 6]
    );
}
```

ENTIRE CODE

```rust
//! **Filter even numbers**: Write a function that takes an iterator over integers and
//! returns a new iterator that yields only the even numbers. Test your function with
//! a range of numbers and a vector of integers.

fn filter_even_nos<I: Iterator<Item = u32>>(iter: I) -> impl Iterator<Item = u32> {
    iter.filter(|&x| x % 2 == 0)
}

#[test]
fn test_some_range() {
    let iter = 1..=6;
    assert_eq!(filter_even_nos(iter).collect::<Vec<u32>>(), [2, 4, 6]);
}

#[test]
fn test_some_vec() {
    let v1: Vec<u32> = vec![1, 2, 3, 4, 5, 6];
    assert_eq!(
        filter_even_nos(v1.iter().cloned()).collect::<Vec<u32>>(),
        [2, 4, 6]
    );
}
```

POINT TO PONDER 🤔

Q. Why can't we use this function with return type as Iterator??

The Filter iterator is a struct that wraps the original iterator and applies the filtering predicate.

Since the return type of the filter method is different from the input iterator type,
you cannot use I as the return type.

```
fn filter_even_nos_2<I: Iterator<Item = u32>>(iter: I) -> I
{   iter.filter(|&x| x % 2 == 0)
}
```