References & Borrowing:

what's wrong with this code!?

```
fn main() {
    let s1 = String::from("hello");
    let (s2, len) = calculate_length(s1);
    println!("The length of '{}' is {}:" s2.len);
}

fn calculate_length(s:String) -> (String, usize){
    let length = s.len();
    (s, length)
}
```

Here, we have to return string 's' to the calling function. so we can still use the String after call to calculate_length because string was moved to calculate_length & becomes invalid after that.

⇒ We can provide reference to string value instead of returning string itself:

```
let len = calculate_length(&s1)
```

& inside calculate_length simply return `s.len()`

references allows us to refer values without taking ownership of it

s

name	value
ptr	

s1

name	value
ptr	
len	
capacity	

index	value
0	h
1	e
2	l
3	l
4	o

& String s pointing at String s1

⇒ The opposite of referencing by using '&' is deferencing. Its accomplished with deference operator '*'.

```
let s1 = String::from("hello")
let len = calculate_length(&s1);
```

it lets us create a reference that refers to the value of s1 but does not own it.

```
fn calculate_length(s: &String) -> usize{ here s is a reference to a String.
    s.len()
} // s goes out of scope. But it does not have ownership of what it refers to. it is not dropped.
```

Rule of references:

1. At any given time, you can have either one mutable reference or any number of immutable references
2. References must always be valid