Solution Review: Generate Panic from Division by 0

This lesson discusses the solution to the challenge given in the previous lesson.

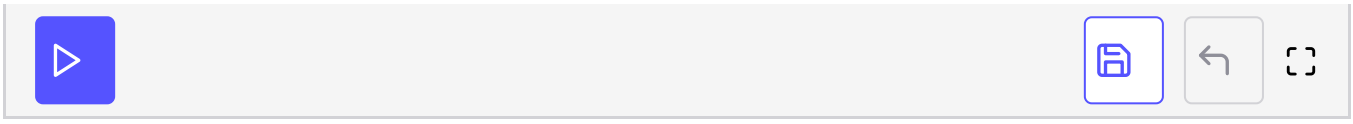
```
package main
import (
        "fmt"
)

func badCall() {
  a, b := 10, 0
  n := a / b   // it will cause panic as it's a division by 0
  fmt.Println(n)
}

func test() {
   defer func() {
    if e := recover(); e != nil {
       fmt.Printf("Panicking %s\r\n", e);
    }

    }()
    badCall()
    fmt.Printf("After bad call\r\n"); // It won't be called, because we just recovered from a
}

func main() {
   fmt.Printf("Calling test\r\n");
   test()   // calling test function
   fmt.Printf("Test completed\r\n");
}
```



Panic from Division by Zero

Following the proposed schema, we call `test()` from the `main()` at **line 25**. The function `test()` first `defers` an anonymous function call at **line 13**, trying to recover from any panic, and printing the error at **line 15**. This code is executed whenever a panic happens, or if not at the end of the function. Then, we execute `badCall()` at **line 19**, which is defined from **line 6** to **line 10**. It divides by 0 at **line 8**, causing a *panic*. The output is:

```
Calling test
```

```
Calling test
Panicking runtime error: integer divide by zero
Test completed
```

We see that because of the panic, **line 20** is never executed, but because we recovered from the error, **line 26** will be executed.

That is it for the solution. In the next lesson, we'll see what it means to start a program externally, when it is needed and how it is done.