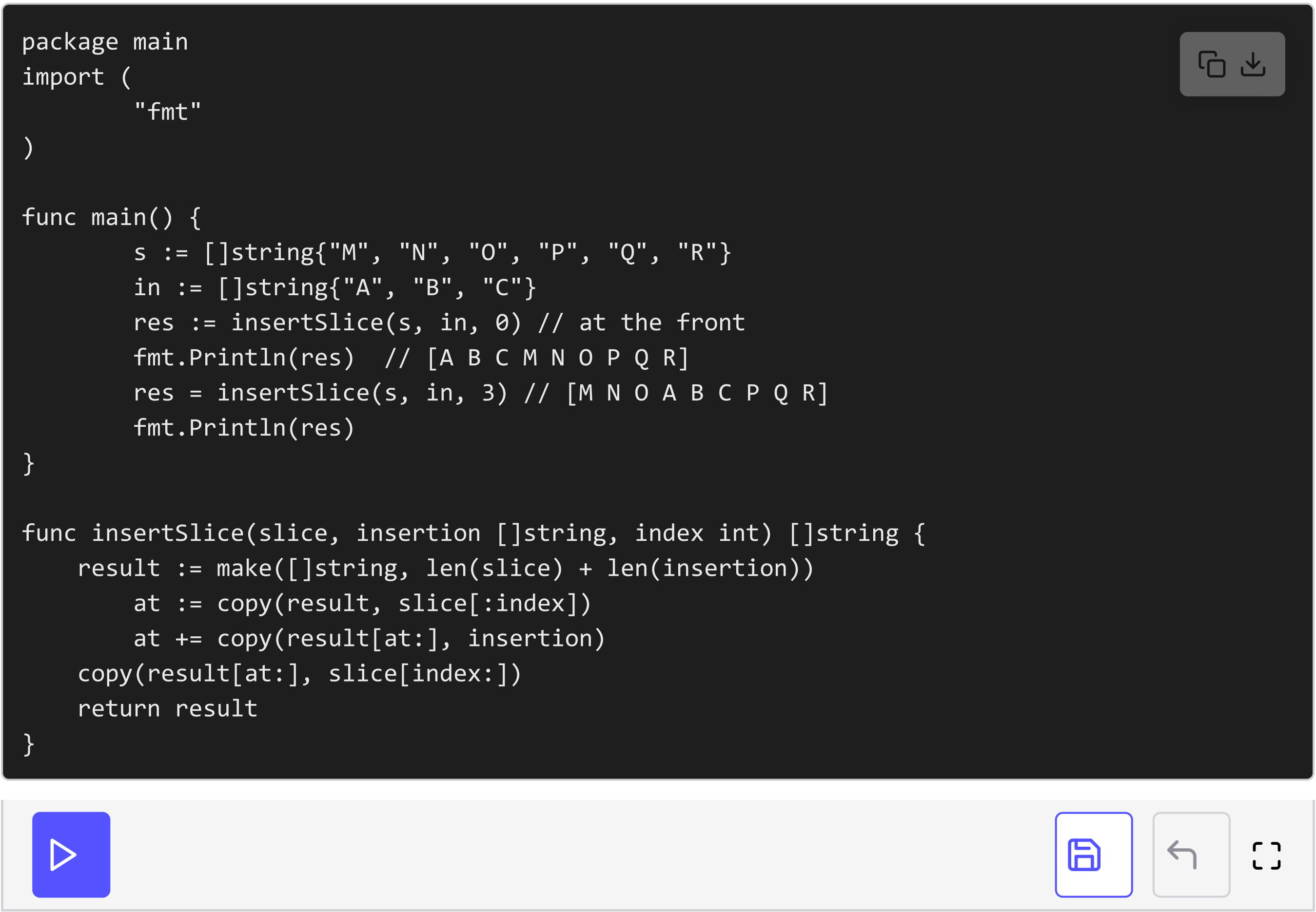
Solution Review: Inserting Slice in a Slice

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



```
package main
import (
        "fmt"
)

func main() {
        s := []string{"M", "N", "O", "P", "Q", "R"}
        in := []string{"A", "B", "C"}
        res := insertSlice(s, in, 0) // at the front
        fmt.Println(res)  // [A B C M N O P Q R]
        res = insertSlice(s, in, 3) // [M N O A B C P Q R]
        fmt.Println(res)
}

func insertSlice(slice, insertion []string, index int) []string {
    result := make([]string, len(slice) + len(insertion))
        at := copy(result, slice[:index])
        at += copy(result[at:], insertion)
    copy(result[at:], slice[index:])
    return result
}
```

Insert Slice in a Slice

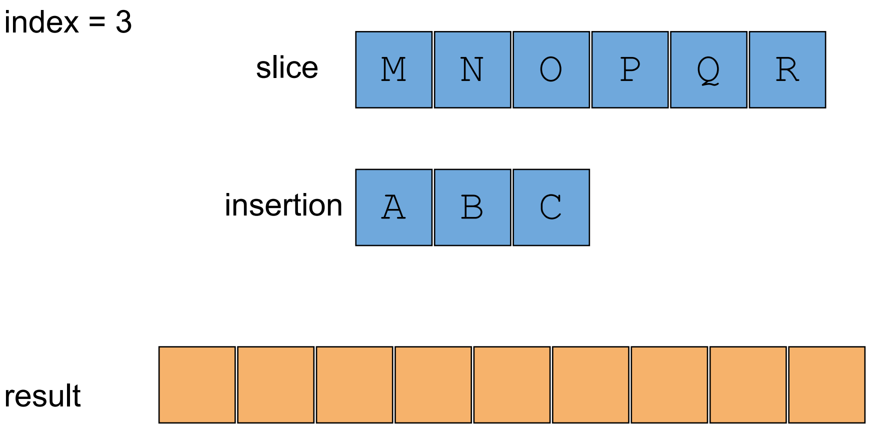
In the code above, look at the header for the function `insertSlice` at **line 15**: `insertSlice(slice, insertion []string, index int) []string`. This function takes two slices `slice` (the slice in which another slice will be inserted) and `insertion` (the slice that is to be inserted in slice) and an integer parameter `index` that defines the point of insertion. The function returns an updated slice after insertion.

We make a slice called `result` at **line 16** with the `make` function. The length of the `result` should be a sum of `len(slice)` (the length of the original `slice`) and `len(insertion)` (the length of `insertion`). Inserting `insertion` at an `index` means that elements of `slice` from and after `index` will move `len(insertion)` indexes forward.

`len(insertion)` indexes forward.

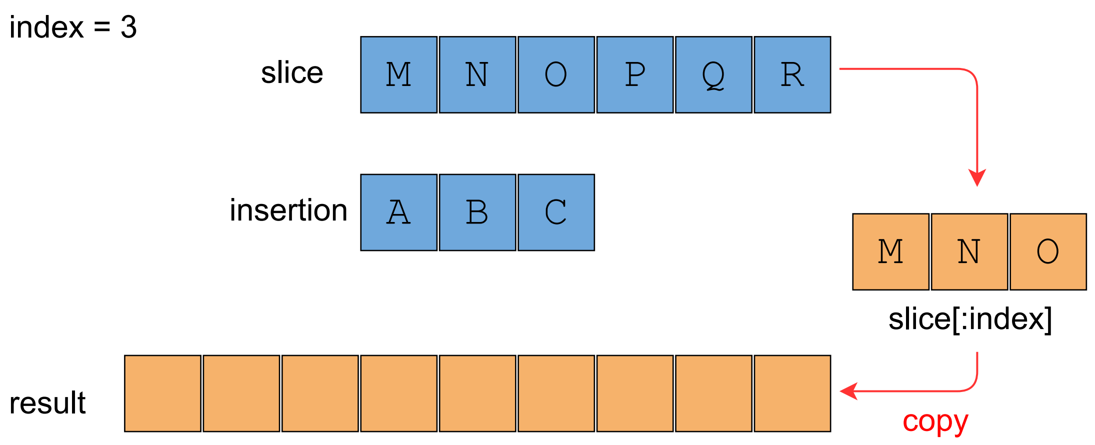
At **line 17**, we first copy all the contents from the **0** index until the index before `index` to `result` and store the number of elements copied in `at`. Then, on the next line (**line 18**), we are copying `insertion` into `result`. The contents from slice `insertion` must be copied in the `result` after the contents from `slice`. To handle this, we make the variable `at`. The statement `at += copy(result[at:], insertion)` will place the `insertion` after the elements of `slice` in `result`, and add the copied elements in `at`.

Now, we have to copy the remaining elements from `slice` if any to `result`. We'll copy the elements starting from `index` from `slice` to the `result` starting from `at` index. At last, we'll return the `result`. Let suppose the slice is **{"M", "N", "O", "P", "Q", "R"}**, the insertion is **{"A","B","C"}**, and the index is **3**. Visualize the concept with these slides."



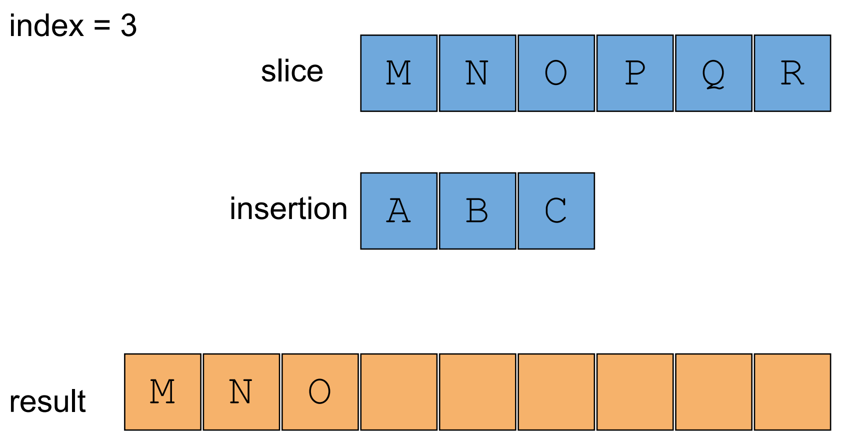
Inserting Slice in a Slice

1 of 7



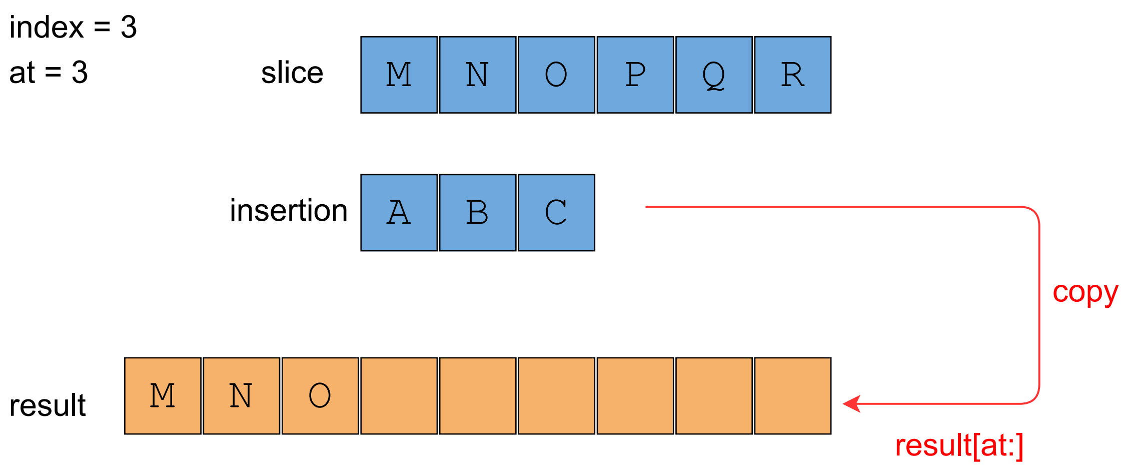
Inserting Slice in a Slice

2 of 7



Inserting Slice in a Slice

3 of 7



index = 3
at = 3

slice M N O P Q R

insertion A B C

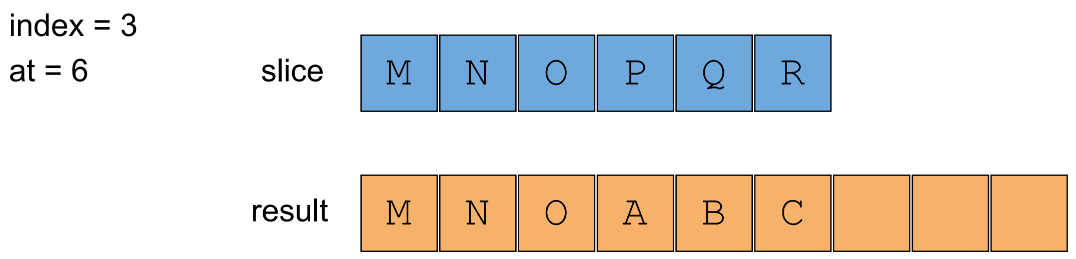
copy

result M N O

result[at:]

Inserting Slice in a Slice

4 of 7



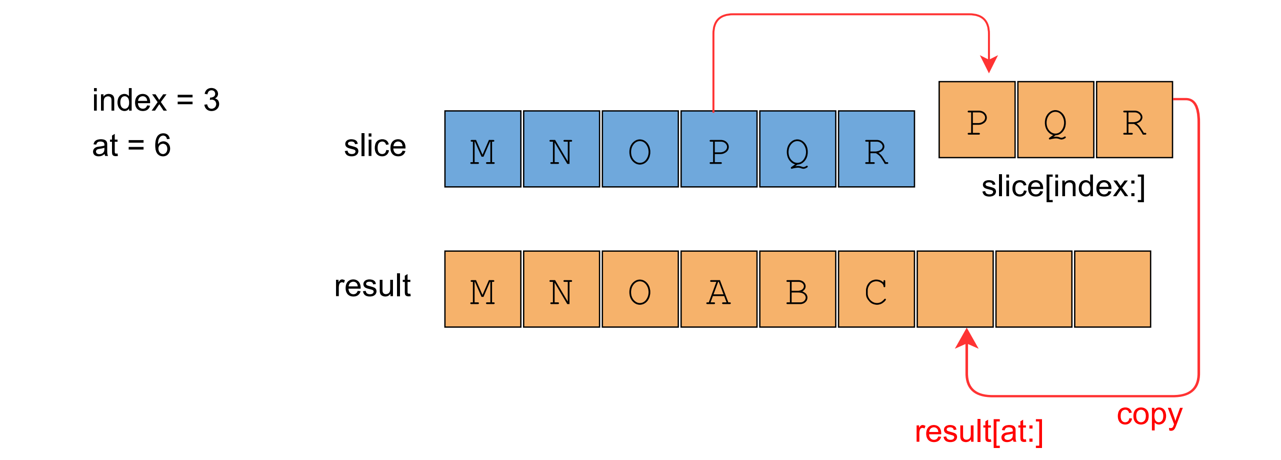
index = 3
at = 6

slice M N O P Q R

result M N O A B C

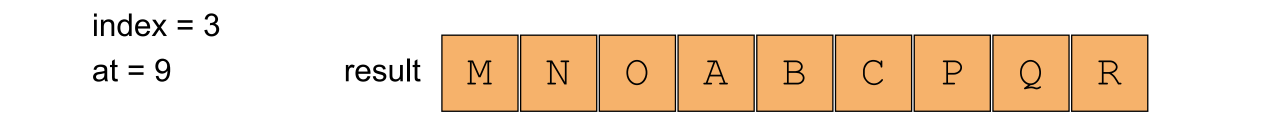
Inserting Slice in a Slice

5 of 7



Inserting Slice in a Slice

6 of 7



Inserting Slice in a Slice

7 of 7

Now, look at the `main` function. We made two *string* slices: `s` (the slice in which another slice will be inserted) and `in` (the slice that is to be inserted in `s`) at **line** 7 and **line 8**, respectively. Then at **line 9**, we call `insertSlice` for `s`, `in` and `index` **0**, and store the result in `res`.The result of `res` is printed in the next line. To test another case, at **line 11**, we call `insertSlice` for `s`, `in` and `index` **3**, and store the result in `res`. The result of `res` is printed in the next line.

That’s it about the solution. In the next lesson, you’ll attempt another challenge.