Solution Review: Sort People with Sorter Interface

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

Environment Variables

Key:	Value:
GOROOT	/usr/local/go
GOPATH	//root/usr/local/go/src
PATH	//root/usr/local/go/src/bin:/usr/local/go...

```
package mysort

type Interface interface {
    Len() int
    Less(i, j int) bool
    Swap(i, j int)
}

func Sort(data Interface) {
    for pass:=1; pass < data.Len(); pass++ {
        for i:=0; i < data.Len() - pass; i++ {
            if data.Less(i+1, i) {
                data.Swap(i, i+1)
            }
        }
    }
}

func IsSorted(data Interface) bool {
    n := data.Len()
    for i := n - 1; i > 0; i-- {
        if data.Less(i, i-1) {
            return false
        }
    }
    return true
}

// Convenience types for common cases
type IntSlice []int

func (p IntSlice) Len() int { return len(p) }

func (p IntSlice) Less(i, j int) bool { return p[i] < p[j] }

func (p IntSlice) Swap(i, j int) { p[i], p[j] = p[j], p[i] }

type StringSlice []string
```

```
type StringSlice []string

func (p StringSlice) Len() int { return len(p) }


func (p StringSlice) Less(i, j int) bool { return p[i] < p[j] }

func (p StringSlice) Swap(i, j int) { p[i], p[j] = p[j], p[i] }

// Convenience wrappers for common cases
func SortInts(a []int) { Sort(IntSlice(a)) }

func SortStrings(a []string) { Sort(StringSlice(a)) }

func IntsAreSorted(a []int) bool { return IsSorted(IntSlice(a)) }

func StringsAreSorted(a []string) bool { return IsSorted(StringSlice(a)) }
```

From **line 8** to **line 11**, we define a struct `Person` with two fields: `firstName` and `LastName`. Then at **line 13** we define a type `Persons` as a `[ ]Person`.

In order to be able to sort `Persons`, we need to implement the `Sorter` interface, which is defined in **mysort.go**- in the **mysort** folder. To do this, we need to define the *three* methods `Len() int`, `Less(I, j int) bool` and `Swap(I, j int)` for a variable `p` of type `Persons`.

Now look at **main.go**.

- See the implementation of `Len` function at **line 15**. `Len()` amounts to the size of the array `p: len(p)`.
- In the `Less` method we have to define how we want to sort a person `p[i]` based on its name. We decide to first concatenate the `firstName` and the `lastName` to the full name (see **line 18** and **line 19**). Then at **line 20**, we compare the two full names and return `in < jn`.
- See the implementation of `Swap` function at **line 24**. It simply swaps the values of `p[i]` and `p[j]`.

> **Note**: The `<` operator works in this case, as `<` is defined for *strings*.

All that is left is to make some `Person` variables in `main()`. We make *three* different `Person` variables from **line 28** to **line 30**, and make an array `arrP` with them at **line 31**. To sort this array on the full names of the persons, we now can call the `Sort` method from package `mysort` on `arrP` at **line 33**:

`mysort.Sort(arrP)`.

That's it about the solution. In the next lesson, you'll see how Go provides support for reading and writing mechanisms.