Range Clauses

Spec: https://go.dev/ref/spec#For_statements

Summary

A range clause provides a way to iterate over an array, slice, string, map, or channel.

Example

```
for k, v := range myMap {
    log.Printf("key=%v, value=%v", k, v)
}

for v := range myChannel {
    log.Printf("value=%v", v)
}

for i, v := range myArray {
    log.Printf("array value at [%d]=%v", i, v)
}
```

Reference

If only one value is used on the left of a range expression, it is the 1st value in this table.

Range expression	1st value	2nd value (optional)	notes
array or slice a `[n]E`, `*[n]E`, or `[]E`	index `i` `int`	`a[i]` E	
string s string type	index `i` `int`	rune `int`	range iterates over Unicode code points, not bytes
map m `map[K]V`	key `k` `K`	value `m[k]` V	
channel c chan E	element `e` `E`	*none*	

Gotchas

When iterating over a slice or map of values, one might try this:

```
items := make([]map[int]int, 10)
for _, item := range items {
    item = make(map[int]int, 1) // Oops! item is only a copy of the slice element.
    item[1] = 2                 // This 'item' will be lost on the next iteration.
}
```

The `make` and assignment look like they might work, but the value property of `range` (stored here as `item`) is a *copy* of the value from `items`, not a pointer to the value in `items`. The following will work:

```
items := make([]map[int]int, 10)
for i := range items {
    items[i] = make(map[int]int, 1)
    items[i][1] = 2
}
```