Go 1.22

Release Notes

For loop vars: problem

One the most common mistakes in Go was something like

```
for index := 0; index < len(workers); index++ {
    go func() {
        err := workers[index].Run()
        if err != nil {
            fmt.Println(err)
        }
    }()
}
```

What's wrong with this code? (in Go 1.21)

For loop vars: problem

One the most common mistakes in Go was something like

```
for index := 0; index < len(workers); index++ {
    go func() {
        err := workers[index].Run()
        if err != nil {
            fmt.Println(err)
        }
    }()
}
```

Before Go 1.22, this would always use the vars from the **last iteration**.

For loop vars: solution

The fix was to pass these vars to the func, or create local vars in loop:

```
for index := 0; index < len(workers); index++ {
    index := index
    go func() {
        err := workers[index].Run()
        if err != nil {
            fmt.Println(err)
        }
    }()
}
```

For loop vars: solution

- With Go 1.22, loop variables are created at each iteration

```
package main

import "fmt"

func main() {
    for i := 0; i < 5; i++ {
        fmt.Println(&i)
    }
}
```

Output in Go 1.22: (example)

```
0xc0000a4010
0xc0000a4018
0xc0000a4030
0xc0000a4038
0xc0000a4040
```

Also: `fmt.Printf("%p\n", &i)`

For loop vars: simpler

Let’s make the above code simpler

- Use **range**
- Wrap the func in another func that handles error and returns nothing

```
for _, worker := range workers {
    go worker.RunNoError()
}
```

Range over int

Now you can range over int!

```
for index := range 10 {
    fmt.Printf(“index = %d\n”, index)
}
```

Instead of:

```
for index := 0; index < 10; index++ {
    fmt.Printf(“index = %d\n”, index)
}
```

Iterators

Set env var `GOEXPERIMENT=rangefunc`

```
func Backward[E any](s []E) func(func(int, E) bool) {
    return func(yield func(int, E) bool) {
        for i := len(s)-1; i >= 0; i-- {
            if !yield(i, s[i]) {
                return
            }
        }
    }
}
```

```
s := []string{"hello", "world"}
for i, x := range Backward(s) {
    fmt.Println(i, x)
}
```

More info: https://go.dev/wiki/RangefuncExperiment

math/rand/v2

First “v2” in the standard library: “math/rand/v2”

See my recent commit in “repassgen” as an example:

- https://github.com/ilius/repassgen/commits/master/
- Commit: switch to math/rand/v2

Tools

- “vendor” directory for workspace: `go work vendor`
- No “`go get`” outside `GOPATH`, even with `GO111MODULE=off`
- “`go test -cover`” treats untested packages as %0 coverage for their parent package.
- “`go vet`” comes with new warnings

Enhanced routing patterns

```
package main

import (
    "net/http"
    "log"
)

func main() {
    http.HandleFunc("GET /users/{$}", func(w http.ResponseWriter, r *http.Request) {
        // get list of users
    })
    http.HandleFunc("POST /users/{$}", func(w http.ResponseWriter, r *http.Request) {
        // create a new user
    })
    http.HandleFunc("GET /users/{id}/", func(w http.ResponseWriter, r *http.Request) {
        userId := r.PathValue("id")
        // get this certain user
    })
    log.Fatal(http.ListenAndServe(":80", nil))
}
```

And so much more!

See https://tip.golang.org/doc/go1.22

A few more examples:

- “database/sql”
  - New `Null[T]` type to scan nullable columns for any column types.

```
var s Null[string]
err := db.QueryRow("SELECT name FROM users WHERE id=?", id).Scan(&s)
...
if s.Valid {
   // use s.V
} else {
   // NULL value
}
```