



7 Surprising Features of The Go Playground

1 // Matt Dale, 9 Jul 2024

2

3

4

5

6

7

Why Are We Talking About the Go Playground?

- Share ideas using Go code.
- Interact with Go code examples.
- It has lots of surprisingly useful features.

The Go Playground

Go 1.22 Run Format Share Hello, World! ▾

Press Esc to move out of the editor.

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 func main() {
9     j, err := json.Marshal(map[string]any{
10         "msg": "Share this with people.",
11     })
12     if err != nil {
13         panic(err)
14     }
15 }
16
17
18
19
20
21
22
23
24
```

<https://go.dev/play/p/2noPn2YOt6P>

Why Are We Talking About the Go Playground?

- Share ideas using Go code.
- Interact with Go code examples.
- It has lots of surprisingly useful features.
- It can run Conway's Game of Life.

The Go Playground

Go 1.22

Run

Format

Share

Hello, World!

```
1 // You can edit this code!
2 // Click here and start typing.
3 package main
4
5 import "fmt"
6
7 func main() {
8     fmt.Println("Hello, 世界")
9 }
10
11
12
13
14
```

<https://go.dev/play/p/jiblOKi7wqW>

A Local Network

- Connect to a local network.
- Communicate between goroutines.
- No Internet connection.

The Go Playground

Go 1.22

Run

Format

Share

Hello, World!

```
4     "fmt"
5     "net"
6 )
7
8 func main() {
9     go func() {
10         conn, err := net.Dial("udp", "127.0.0.1:1234")
11         if err != nil {
12             panic(err)
13         }
14         conn.Write([]byte("Hello, GopherCon!"))
15     }()
16
17     pconn, err := net.ListenPacket("udp", "127.0.0.1:1234")
18     if err != nil {
19         panic(err)
20     }
21
22     p := make([]byte, 128)
23     n, addr, err := pconn.ReadFrom(p)
24     fmt.Printf("%v sent %q\n", addr, p[:n])
25     if err != nil {
26         panic(err)
27     }
28 }
29 |
```

<https://go.dev/play/p/eP8KlyLLAqh>

External Modules

- Support for pulling in external modules was added in 2019.
- Can be quicker than creating a new local project.

The Go Playground

Go 1.22

Run

Format

Share

Hello, World!

```
1 package main
2
3 import (
4     "fmt"
5
6     "go.mongodb.org/mongo-driver/bson"
7 )
8
9 type doc struct {
10     Text string
11     ID    string
12 }
13
14 func main() {
15     b, err := bson.Marshal(doc{
16         Text: "my text",
17         ID:   "abcd1234",
18     })
19     if err != nil {
20         panic(err)
21     }
22     fmt.Printf("%q\n", b)
23 }
24
25
26
```

<https://go.dev/play/p/KbWrRIdGYIp>

Multiple Files

- Define new files using the
-- <file path> --
syntax.
- Define files in different directories.
- Modify module versions by defining a
go.mod file.

The Go Playground

Go 1.22

Run

Format

Share

Hello, World!

```
3 import (
4     "bytes"
5     "fmt"
6     "strings"
7
8     "golang.org/x/net/html"
9 )
10
11 func main() {
12     doc := `<html><head></head><body><svg><style>&</style></svg></body></html>`
13     n, err := html.Parse(strings.NewReader(doc))
14     if err != nil {
15         panic(err)
16     }
17     b := bytes.NewBuffer(nil)
18     if err := html.Render(b, n); err != nil {
19         panic(err)
20     }
21     fmt.Println("got: ", b.String())
22     fmt.Println("want:", `<html><head></head><body><svg><style>&lt;/style></svg></body></html>`)
23 }
24 -- go.mod --
25 module play.ground
26
27 require golang.org/x/net v0.12.0
28
```

https://go.dev/play/p/x_VWq2ADJYe

Multiple Files

- Define new files using the
 -- <file path> --
syntax.
- Define files in different directories.
- Modify module versions by defining a
go.mod file.

The Go Playground

Go 1.22

Run

Format

Share

Hello, World!

Press Esc to move out of the editor.

```
1 package main
2
3 import (
4     _ "embed"
5 )
6
7 //go:embed prog.go
8 var s string
9
10 func main() {
11     print(s)
12 }
13
14
```

<https://go.dev/play/p/SveZFUtKx7P>

A Local Filesystem

- Read and write files using the playground's in-memory filesystem.
- Listen for filesystem events.
- Run executables like **grep** and **tr**.

The Go Playground

Go 1.22 Run Format Share Hello, World! ▾

```
7     "github.com/fsnotify/fsnotify"
8 )
9
10 func main() {
11     watcher, err := fsnotify.NewWatcher()
12     if err != nil {
13         panic(err)
14     }
15     defer watcher.Close()
16
17     err = watcher.Add("/tmp")
18     if err != nil {
19         panic(err)
20     }
21
22     go func() {
23         f, err := os.Create("/tmp/my-new-file.txt")
24         if err != nil {
25             panic(err)
26         }
27         defer f.Close()
28     }()
29
30     fmt.Println("Filesystem event:", <-watcher.Events)
31 }
32
```

<https://go.dev/play/p/30uwHil-AsW>

Go Tests

- Write function signatures like Go tests instead of a **main** function.
- The Go Playground runs **go test** instead of **go run**.

The Go Playground

Go 1.22

Run

Format

Share

Hello, World!

```
3 import (
4     "bytes"
5     "strings"
6     "testing"
7
8     "golang.org/x/net/html"
9 )
10
11 func TestRender(t *testing.T) {
12     doc := `<html><head></head><body><svg><style>&</style></svg></body></html>`
13     n, err := html.Parse(strings.NewReader(doc))
14     if err != nil {
15         panic(err)
16     }
17     b := bytes.NewBuffer(nil)
18     if err := html.Render(b, n); err != nil {
19         panic(err)
20     }
21
22     want := `<html><head></head><body><svg><style>&lt;/style></svg></body></html>`
23     got := b.String()
24     if got != want {
25         t.Errorf("want %q, got %q", want, got)
26     }
27 }
28 |
```

<https://go.dev/play/p/DQyns6lMKGB>

Clear the Terminal

- Clear the terminal by printing “\x0c”.
- Create cool text effects.

The Go Playground

Go 1.22

Run

Format

Share

Hello, World!

```
58             if frame[y][x] != ' ' && rand.Float32() < 0.75 {
59                 frame[y][x] = runes[rand.Intn(len(runes))]
60             }
61         }
62     }
63 }
64
65 func draw() {
66     fmt.Printf("\x0c")
67
68     for y := range frame {
69         for x := range frame[y] {
70             fmt.Print(string(frame[y][x]))
71         }
72         fmt.Println()
73     }
74 }
75
```

<https://go.dev/play/p/KIP3LZz5u4Q>

Display Images

- Print “**IMAGE:**” followed by the base64-encoded image data.
- Works with anything that the browser can display in an **** tag, including GIFs.

The Go Playground

Go 1.22

Run

Format

Share

Hello, World!

```
32     // Save to a PNG file.
33     err = draw2dimg.SaveToPngFile(fn, dest)
34     if err != nil {
35         log.Printf("Saving %q failed: %v", fn, err)
36         return
37     }
38
39     // Read gopher2.png and echo it to the terminal
40     // as a base64-encoded string.
41     dat, err := os.ReadFile("output/samples/gopher2.png")
42     if err != nil {
43         panic(err)
44     }
45     fmt.Println("IMAGE:" + base64.StdEncoding.EncodeToString(dat))
46 }
47 |
```

<https://go.dev/play/p/8NOVwWzZOOp>

goplay.tools

- Syntax highlighting and code completion.
- Compile as WebAssembly and run in-browser.

The screenshot shows the goplay.tools web application. At the top, there's a navigation bar with icons for Open, Run, Share, Download, Settings, About, and a Go 1.21 dropdown. Below the navigation is a code editor window containing a Go script. The code includes comments about WebAssembly features, imports for fmt and syscall/js, and a main function. A warning message in the code states: "Attention: please select 'WebAssembly' or 'Go Interpreter' environment in top". Below the code editor is a dark panel labeled "OUTPUT" which displays the message "Press 'Run' to compile program." At the bottom of the interface, there's a blue footer bar with a "0 Errors" indicator and some small icons.

```
1 // This tutorial introduces Go WebAssembly features.  
2 //  
3 // Go code can be compiled and ran in web browser as WebAssembly module.  
4 // Go allows calling JavaScript functions from browser and also export functions to  
5 package main  
6  
7 ///////////////////////////////////////////////////////////////////  
8 // ⚠ Attention: please select "WebAssembly" or "Go Interpreter" environment in top  
9 ///////////////////////////////////////////////////////////////////  
10  
11 import (  
12     "fmt"  
13     "syscall/js"  
14 )  
15  
16 func main() {  
17 }
```

OUTPUT

Press "Run" to compile program.

0 Errors

<https://goplay.tools/snippet/CmRxNZQXvqB>

goplay.space

- Syntax highlighting.
- Turtle graphics mode.



The Go Play Space

Run ⌘+← Format ⌘+S Share Settings

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func branch(size, ratio, angle
8 float64, iterations int) {
9     // draw the stem
10    fmt.Println("color black")
11    fmt.Printf("width %f\n", size)
12    fmt.Printf("forward %f\n", size)
13    fmt.Println("color off")
14
15    if iterations > 0 {
16        // rotate left and draw the
17        // child stem
18        fmt.Printf("left %f\n",
19        angle)
20        branch(size*ratio, ratio,
21        angle, iterations-1)
22
23        // rotate right and draw the
24        // child stem
25        fmt.Printf("right %f\n",
26        angle*2)
27        branch(size*ratio, ratio,
```

Turtle Graphics Mode

Go Play Space supports the [Turtle graphics](#) mode to help visualize algorithms and make learning experience more fun.

Whenever Go code is executed and produces some console output, this output is parsed, and found control commands are interpreted on a drawing board. You can start with something basic like one `fmt.Println` statement that prints all the commands sequentially, and then build your own functional API and algorithms — in Go — that would produce the desired sequence of control commands.

Try These Examples

1. [Star](#) — an example on how to draw a star
2. [House](#) — an example on drawing a house with multiple colors
3. [Tree](#) — an example on using recursion to draw a tree
4. [Colored squares](#) — an example of using a simple “API” wrapper functions and `for` loops
5. [Colored squares \(Russian\)](#) — the same “Colored squares” example above, but with function/

> Syntax OK

<https://goplay.space/#61SJKVrWwj>

Thank you!

Find links to these examples at:
github.com/matthewdale/gophercon2024