Art2Dec

Pure Go language Patterns for mshell Workflow — Complete Reference Guide (p1–p24)

Pure Go language Edition — Art2Dec SoftLab, March 23rd 2026

Pure Edition Go-Only · No Python, C, C++, Rust, Lua or Bash — only `Golang`, `mshell` directives, and LLM blocks. All 24 patterns verified individually in mshell.

What is mshell?

mshell is a polyglot UNIX shell environment for AI and data workflows. It integrates multiple programming languages and LLM models into a single unified execution pipeline. Go code blocks are compiled and executed automatically via `go run`. Variables flow between blocks via named files. LLM directives inject model responses as first-class variables that any subsequent block can consume.

Variable System

Reading a variable:

```
import "os"
data, _ := os.ReadFile(os.Getenv("MSH_VAR_varname"))
value := strings.TrimSpace(string(data))
```

Writing a variable (multiple >outvar block):

```
// stdout is NOT captured when multiple >outvar are declared — write directly
os.WriteFile(os.Getenv("MSH_VAR_varname"), []byte("value\n"), 0644)
```

Slurp helper (read multiple variables cleanly):

```
func slurp(key string) string {
    data, _ := os.ReadFile(os.Getenv(key))
    return strings.TrimSpace(string(data))
}
```

Compiling and running LLM-generated Go code:

```
tmp, _ := os.CreateTemp("", "msh_gen_*.go")
tmp.Write(code); tmp.Close()
```

```
out, err := exec.Command("go", "run", tmp.Name()).CombinedOutput()
os.Remove(tmp.Name())
```

Async await barrier (use bash, not go):

```
for p in "$(printenv MSH_VAR_ans1)" "$(printenv MSH_VAR_ans2)"; do
    i=0; while [ ! -s "$p" ] && [ $i -lt 180 ]; do sleep 1; i=$((i+1)); done
done
```

WHILE loop counter update:

```
path := os.Getenv("MSH_VAR_counter")
data, _ := os.ReadFile(path)
val, _ := strconv.Atoi(strings.TrimSpace(string(data)))
os.WriteFile(path, []byte(fmt.Sprintf("%d\n", val+1)), 0644)
```

FOREACH list (no trailing newline):

```
fmt.Print("item1\nitem2\nitem3")  // Print not Println — no trailing newline
```

Final status display (use bash, not go — avoids getwd issue):

```
echo "=== Final status: $(cat $(printenv MSH_VAR_result) | tr -d '\n') ==="
```

Critical Rules

1. **`bash await=`** — Use `bash` blocks for async barriers, not go. Go blocks recompile each time and can hit race conditions.
2. **Multiple `>outvar`** — Write directly via `os.WriteFile(os.Getenv("MSH_VAR_*"), ...)`. Stdout is NOT captured.
3. **WHILE counter** — Read/write `MSH_VAR_*` files directly with `os.ReadFile`/`os.WriteFile`.
4. **FOREACH list** — Use `fmt.Print` (no trailing newline). One item per line.
5. **TRY/CATCH** — Trigger CATCH with `os.Exit(1)`. CATCH block prints literal `"try_block_failed"` — do NOT use `<errvar`.
6. **LLM prompts** — Add `You MUST respond` to prevent empty responses.
7. **`go run` temp files** — Always use `os.CreateTemp("", "*.go")` — the `.go` extension is required for `go run`.
8. **Final display after LOOP** — Use `bash` not go for the post-loop display block to avoid `getwd: no such file` errors.
9. **Safe reads** — Use `strings.TrimSpace(string(data))` to strip trailing newlines from variable values.
10. **CONFIG block** — Documentation only — always pair with `go >varname` blocks that `fmt.Println` the actual values.

Pattern Summary Table

#	Pattern	Nodes	Models	Key Technique
1	Linear Data Pipeline	—	—	5-stage sequential Go pipeline
2	LLM in the Middle	—	@1	Go → LLM → Go
3	Fan-Out	—	@1	One var, multiple consumers
4	LLM Code Gen + Execute	—	@1	go run with temp file
5	Two-LLM Review Chain	—	@1, @2	Generate → Review → Improve
6	Parallel 3-Model Query	—	@1–@3	Sequential LLM directives
7	Evaluator-Optimizer Loop	LOOP	@1, @2	ACCEPTED/REJECTED gate
8	Multi-Stage + Multi-Model	—	@1, @2	Go stats + LLM analysis
9	Routing	—	@1	if=route:VALUE conditional
10	Full Pipeline	AWAIT	@1–@3	All patterns combined
11	MShell Native AI	—	@1, @2	ollama1/ollama2 inline
12	Async 3 Models + Synthesis	AWAIT	@1–@3	bash await barrier
13	WHILE Counter	WHILE	@1	os.WriteFile MSH_VAR_status
14	FOREACH List	FOREACH	@1	fmt.Print no trailing newline
15	TRY/CATCH	TRY/CATCH	—	os.Exit(1) triggers CATCH
16	SPLIT + MERGE	SPLIT, MERGE	@1, @2	bash await + synthesis
17	CONFIG Pipeline	CONFIG	@1, @2	CONFIG docs, go sets values
18	FOREACH + Async	FOREACH, AWAIT	@1, @2	bash await per iteration
19	WHILE Quality Gate	WHILE	@1, @2	os.WriteFile status on threshold
20	Map-Reduce	SPLIT, MERGE, AWAIT	@1, @2	3 async map + reduce
21	TRY/CATCH + LOOP	TRY/CATCH, LOOP	@1	go run temp file, bash final
22	Multi-Variable Output	—	@1,	os.WriteFile per MSH_VAR_*

#	Pattern	Nodes	Models	Key Technique
			@2	
23	CONFIG + WHILE + 3M	CONFIG, WHILE	@1–@3	Adaptive quality pipeline
24	FOREACH + TRY/CATCH	FOREACH	@1	encoding/json fault isolation

---

Part I — Patterns 1–12: Core Patterns

Pattern 1 — Linear Data Pipeline (Go)

What it does: Five sequential Go stages transform a single integer: raw number → squaring/cubing → primality check → FizzBuzz classification → formatted report. Pure Go, no LLM.

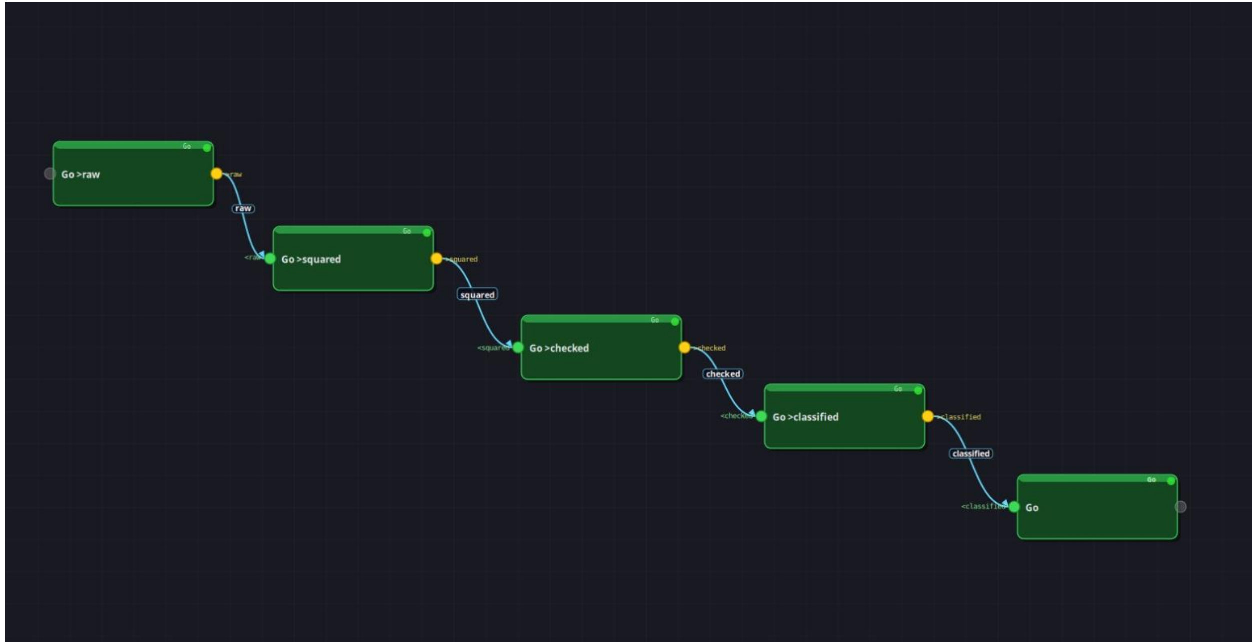
```
package main
import "fmt"
func main() {
    fmt.Println("42")
}
```

```
package main

import (
    "fmt"
    "os"
    "strconv"
    "strings"
)

func main() {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_raw"))
    n, _ := strconv.Atoi(strings.TrimSpace(string(data)))
    fmt.Printf("value=%d squared=%d cubed=%d\n", n, n*n, n*n*n)
}
```

```
package main

import (
    "fmt"
    "os"
    "strings"
)

func isPrime(n int) bool {
    if n < 2 {
        return false
```

```go
    }
    for i := 2; i*i <= n; i++ {
        if n%i == 0 {
            return false
        }
    }
    return true
}

func main() {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_squared"))
    line := strings.TrimSpace(string(data))
    // extract original value
    var val int
    fmt.Sscanf(line, "value=%d", &val)
    prime := "not prime"
    if isPrime(val) {
        prime = "prime"
    }
    fmt.Printf("%s | primality=%s\n", line, prime)
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_checked"))
    line := strings.TrimSpace(string(data))
    var val int
    fmt.Sscanf(line, "value=%d", &val)
    fb := "none"
    switch {
    case val%15 == 0:
        fb = "FizzBuzz"
    case val%3 == 0:
        fb = "Fizz"
    case val%5 == 0:
        fb = "Buzz"
    }
    fmt.Printf("%s | fizzbuzz=%s\n", line, fb)
}

package main

import (
```

```
    "fmt"
    "os"
    "strings"
)

func main() {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_classified"))
    fmt.Println("=== Pipeline Report ===")
    fmt.Println(strings.TrimSpace(string(data)))
}
```

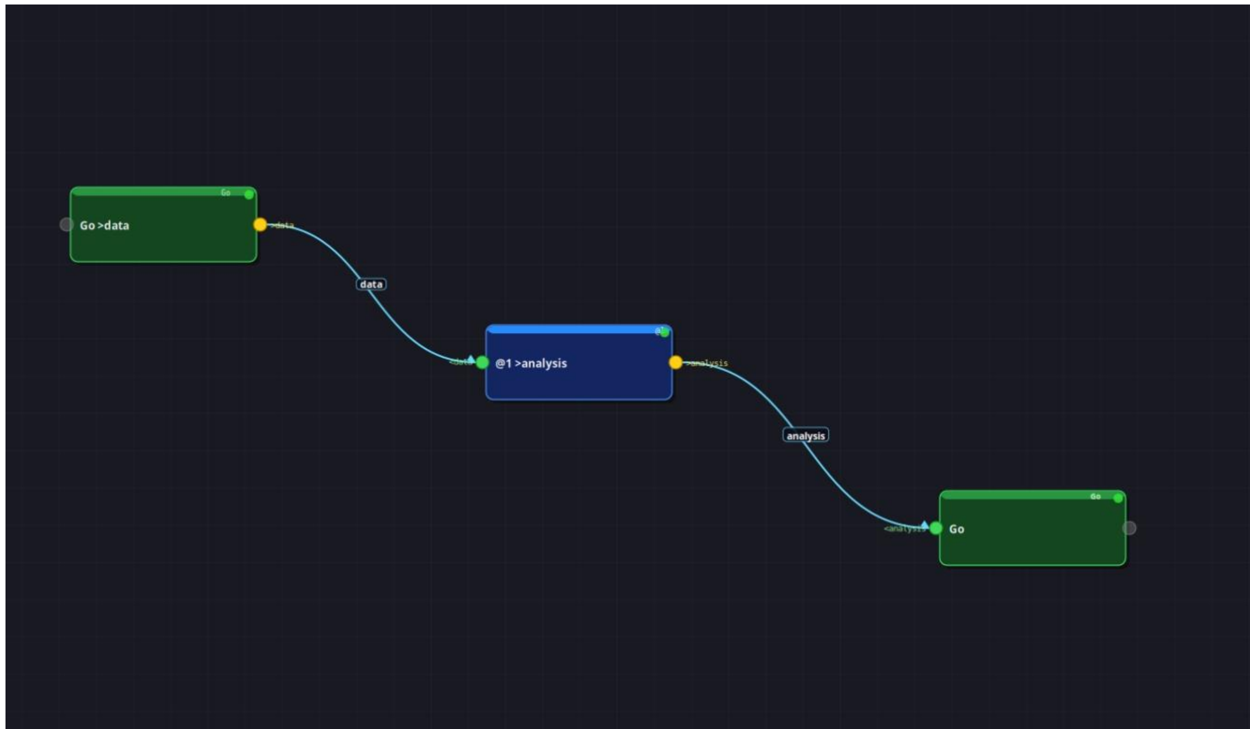


Pattern 2 — LLM in the Middle

What it does: Go generates a synthetic server log. LLM @1 writes an incident summary. A second Go block extracts word and sentence statistics from the response.

```
package main

import "fmt"

func main() {
    logs := []string{
        "2026-03-11 10:00:01 GET /api/users 200 45ms",
        "2026-03-11 10:00:03 POST /api/login 401 12ms",
        "2026-03-11 10:00:07 GET /api/data 500 2301ms",
        "2026-03-11 10:00:09 GET /api/users 200 48ms",
        "2026-03-11 10:00:15 DELETE /api/item 403 9ms",
        "2026-03-11 10:00:22 GET /api/data 500 3100ms",
    }
    for _, l := range logs {
```

```
        fmt.Println(l)
    }
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    text, _ := os.ReadFile(os.Getenv("MSH_VAR_analysis"))
    content := strings.TrimSpace(string(text))

    // Fallback if LLM returned empty response
    if content == "" {
        fmt.Println("LLM analysis received: [empty response \u2014 model
returned no content]")
        fmt.Println("HTTP codes mentioned: [unknown]")
        fmt.Println("--- Analysis ---")
        fmt.Println("[No analysis available]")
        return
    }

    words := strings.Fields(content)
    sentences := strings.Count(content, ".") + strings.Count(content, "!") +
strings.Count(content, "?")

    codes := []string{"200", "401", "403", "404", "500", "503"}
    mentioned := []string{}
    for _, code := range codes {
        if strings.Contains(content, code) {
            mentioned = append(mentioned, code)
        }
    }

    fmt.Printf("LLM analysis received: %d words, ~%d sentences\n",
len(words), sentences)
    fmt.Printf("HTTP codes mentioned: %s\n", strings.Join(mentioned, ", "))
    fmt.Println("--- Analysis ---")
    fmt.Println(content)
}
```



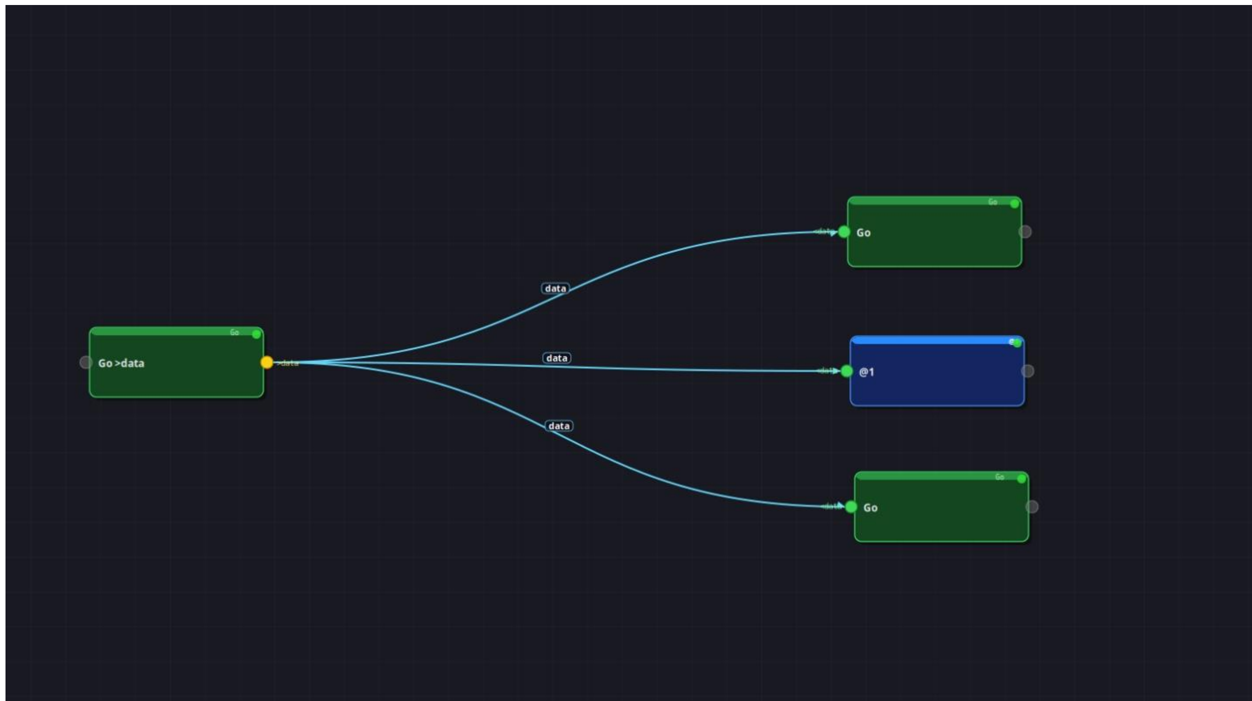
Pattern 3 — Fan-Out: One Variable → Many Consumers

What it does: One Go block generates a dataset. Three independent consumers read the same variable: Go computes stats, Go finds primes in the set, and LLM @1 provides a natural-language interpretation.

```
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    r := rand.New(rand.NewSource(2026))
    nums := make([]int, 12)
    for i := range nums {
        nums[i] = r.Intn(100) + 1
    }
    for i, n := range nums {
        if i > 0 {
            fmt.Print(",")
        }
        fmt.Print(n)
    }
    fmt.Println()
}
```

```go
package main

import (
	"fmt"
	"os"
	"strconv"
	"strings"
)

func main() {
	raw, _ := os.ReadFile(os.Getenv("MSH_VAR_data"))
	parts := strings.Split(strings.TrimSpace(string(raw)), ",")
	nums := make([]int, 0, len(parts))
	for _, p := range parts {
		n, err := strconv.Atoi(strings.TrimSpace(p))
		if err == nil {
			nums = append(nums, n)
		}
	}
	min, max, sum := nums[0], nums[0], 0
	for _, n := range nums {
		sum += n
		if n < min { min = n }
		if n > max { max = n }
	}
	mean := float64(sum) / float64(len(nums))
	fmt.Printf("Go stats: count=%d sum=%d min=%d max=%d mean=%.2f\n",
		len(nums), sum, min, max, mean)
}

package main

import (
	"fmt"
	"os"
	"sort"
	"strconv"
	"strings"
)

func main() {
	raw, _ := os.ReadFile(os.Getenv("MSH_VAR_data"))
	parts := strings.Split(strings.TrimSpace(string(raw)), ",")
	nums := make([]int, 0, len(parts))
	for _, p := range parts {
		n, err := strconv.Atoi(strings.TrimSpace(p))
		if err == nil {
			nums = append(nums, n)
		}
	}
```

```
    sorted := make([]int, len(nums))
    copy(sorted, nums)
    sort.Ints(sorted)
    n := len(sorted)
    var median float64
    if n%2 == 0 {
        median = float64(sorted[n/2-1]+sorted[n/2]) / 2.0
    } else {
        median = float64(sorted[n/2])
    }
    sum := 0
    for _, v := range nums { sum += v }
    mean := float64(sum) / float64(n)
    skew := "symmetric"
    if mean > median+2 { skew = "right-skewed" }
    if mean < median-2 { skew = "left-skewed" }
    fmt.Printf("Go median: %.1f | distribution: %s\n", median, skew)
}
```

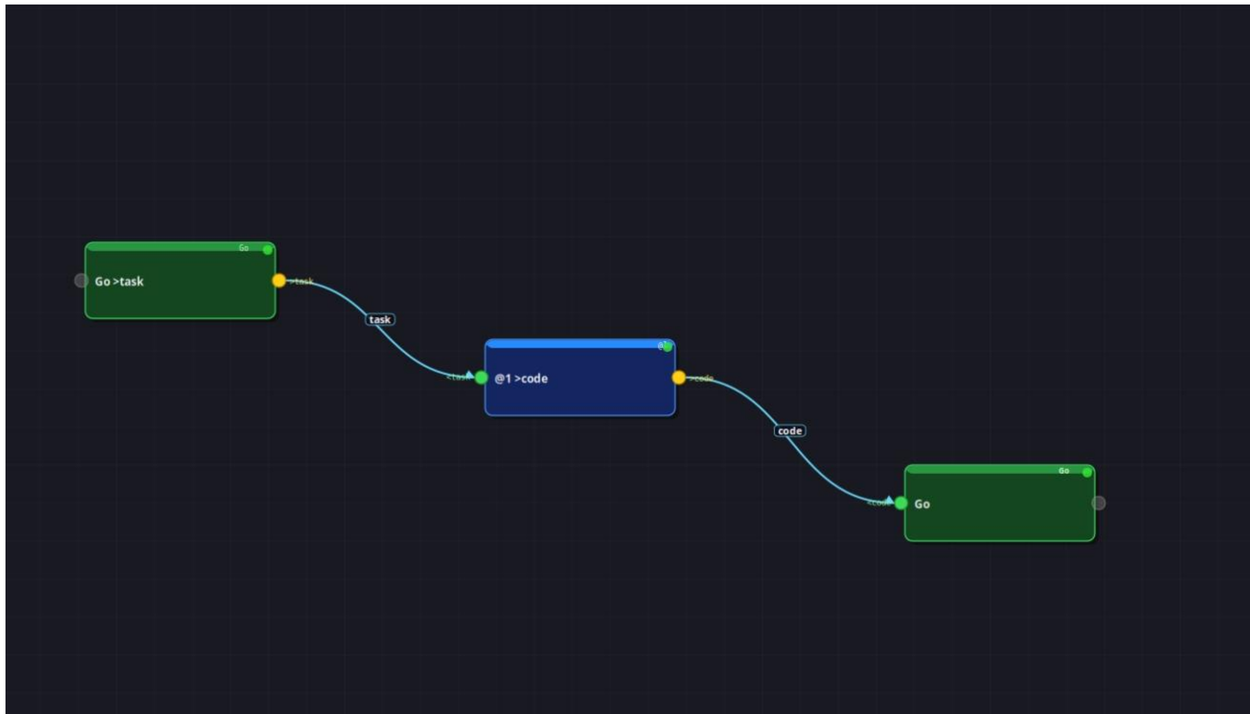


Pattern 4 — LLM Code Generation → Compile & Execute

What it does: Go writes a task description. LLM @1 generates a complete Go program. A second Go block reads the generated code path, compiles and runs it with `go run`.

```
package main

import "fmt"

func main() {
```

```
    fmt.Println("Write a Go program that generates the first 10 Fibonacci
numbers and prints their sum and product.")
}

package main

import (
    "fmt"
    "os"
    "os/exec"
)

func main() {
    codePath := os.Getenv("MSH_VAR_code")
    codeBytes, err := os.ReadFile(codePath)
    if err != nil {
        fmt.Fprintf(os.Stderr, "cannot read code: %v\n", err)
        os.Exit(1)
    }

    fmt.Println("=== Generated Go code ===")
    fmt.Println(string(codeBytes))
    fmt.Println("=== Executing ===")

    tmpFile, err := os.CreateTemp("", "msh_gen_*.go")
    if err != nil {
        fmt.Fprintf(os.Stderr, "cannot create temp file: %v\n", err)
        os.Exit(1)
    }
    defer os.Remove(tmpFile.Name())

    tmpFile.Write(codeBytes)
    tmpFile.Close()

    out, err := exec.Command("go", "run", tmpFile.Name()).CombinedOutput()
    if err != nil {
        fmt.Fprintf(os.Stderr, "execution error: %v\n%s\n", err, out)
        os.Exit(1)
    }
    fmt.Print(string(out))
}
```



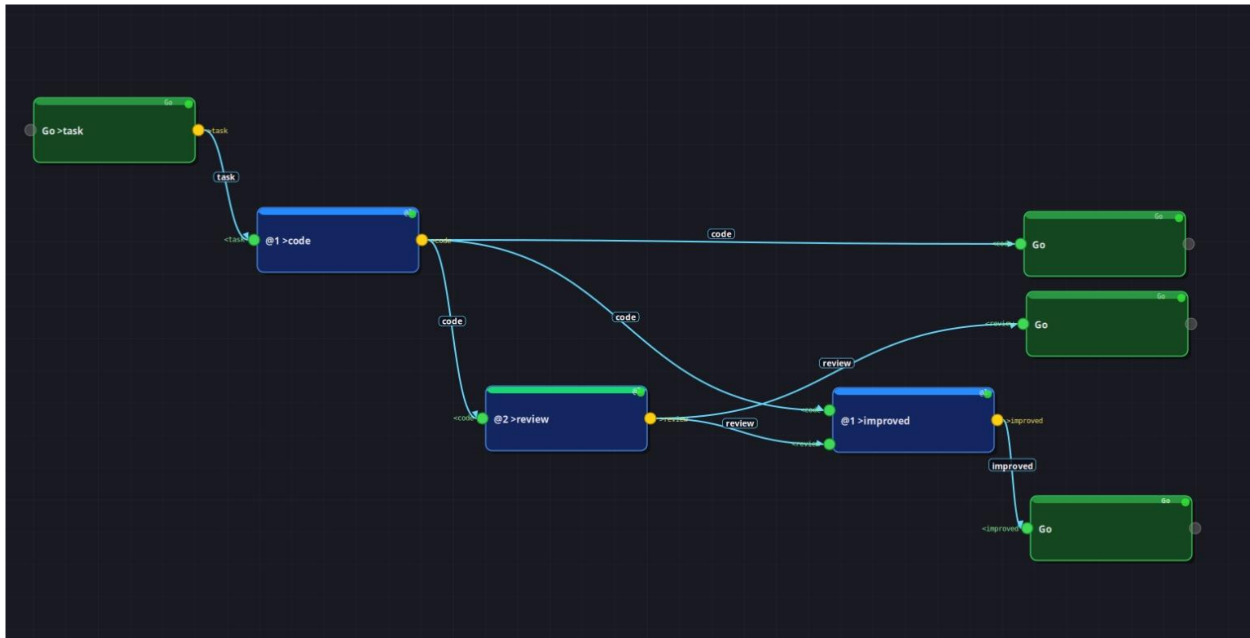
Pattern 5 — Two-LLM Review Chain

What it does: LLM @1 generates a Go function. LLM @2 reviews it for correctness and idioms. LLM @1 improves based on the review. A Go block compiles and runs the final version.

```
package main

import "fmt"

func main() {
    fmt.Println("Write a Go function that implements binary search on a sorted slice of integers. Include a main function that tests it with a sample slice.")
}
```

```
package main

import (
    "fmt"
    "os"
)

func main() {
    codePath := os.Getenv("MSH_VAR_code")
    codeBytes, _ := os.ReadFile(codePath)
    fmt.Println("=== Model 1 generated ===")
```

```go
    fmt.Println(string(codeBytes))
}

package main

import (
    "fmt"
    "os"
)

func main() {
    review, _ := os.ReadFile(os.Getenv("MSH_VAR_review"))
    fmt.Println("=== Model 2 review ===")
    fmt.Println(string(review))
}

package main

import (
    "fmt"
    "os"
    "os/exec"
)

func main() {
    improvedPath := os.Getenv("MSH_VAR_improved")
    codeBytes, _ := os.ReadFile(improvedPath)
    fmt.Println("=== Final improved code ===")
    fmt.Println(string(codeBytes))
    fmt.Println("=== Executing ===")

    tmpFile, _ := os.CreateTemp("", "msh_improved_*.go")
    defer os.Remove(tmpFile.Name())
    tmpFile.Write(codeBytes)
    tmpFile.Close()

    out, err := exec.Command("go", "run", tmpFile.Name()).CombinedOutput()
    if err != nil {
        fmt.Fprintf(os.Stderr, "error: %v\n%s\n", err, out)
        os.Exit(1)
    }
    fmt.Print(string(out))
}
```



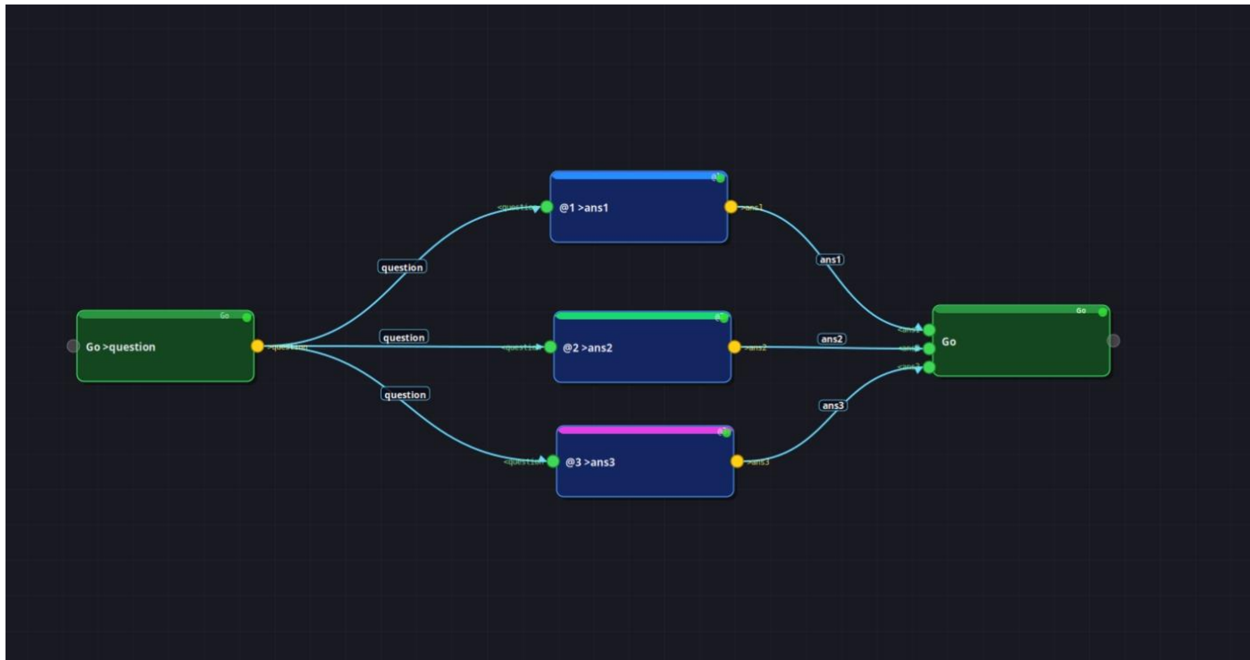
Pattern 6 — Parallel 3-Model Query

What it does: Go formulates a Go concurrency design question. All three LLM models answer independently with different focus areas. Go renders all three responses in a comparison table.

```
package main

import "fmt"

func main() {
	fmt.Println("What is the single most important principle to follow when designing concurrent Go programs? Answer in exactly one sentence.")
}
```

```
package main

import (
	"fmt"
	"os"
	"strings"
)

func wordSet(s string) map[string]bool {
	set := make(map[string]bool)
	for _, w := range strings.Fields(strings.ToLower(s)) {
		// strip punctuation
		w = strings.Trim(w, ".,!?;:")
		if len(w) > 3 {
			set[w] = true
```

```
        }
    }
    return set
}

func overlap(a, b map[string]bool) int {
    count := 0
    for w := range a {
        if b[w] {
            count++
        }
    }
    return count
}

func main() {
    ans1, _ := os.ReadFile(os.Getenv("MSH_VAR_ans1"))
    ans2, _ := os.ReadFile(os.Getenv("MSH_VAR_ans2"))
    ans3, _ := os.ReadFile(os.Getenv("MSH_VAR_ans3"))

    a1 := strings.TrimSpace(string(ans1))
    a2 := strings.TrimSpace(string(ans2))
    a3 := strings.TrimSpace(string(ans3))

    fmt.Println("=== Model 1 ===")
    fmt.Println(a1)
    fmt.Println()
    fmt.Println("=== Model 2 ===")
    fmt.Println(a2)
    fmt.Println()
    fmt.Println("=== Model 3 ===")
    fmt.Println(a3)
    fmt.Println()

    s1, s2, s3 := wordSet(a1), wordSet(a2), wordSet(a3)
    fmt.Printf("Word overlap 1\u21942: %d | 1\u21943: %d | 2\u21943: %d\n",
        overlap(s1, s2), overlap(s1, s3), overlap(s2, s3))
}
```



Pattern 7 — Evaluator-Optimizer Loop

What it does: LLM @1 generates Go code. LLM @2 evaluates it strictly (ACCEPTED / REJECTED). Loop repeats up to 3 times. A Go block compiles and runs the accepted code.

```
package main

import "fmt"

func main() {
    fmt.Println("Write a Go function isPalindrome(s string) bool that returns true if the string is a palindrome (ignoring case and non-alphanumeric characters). Include a main function that tests it with at least 3 examples.")
}
```

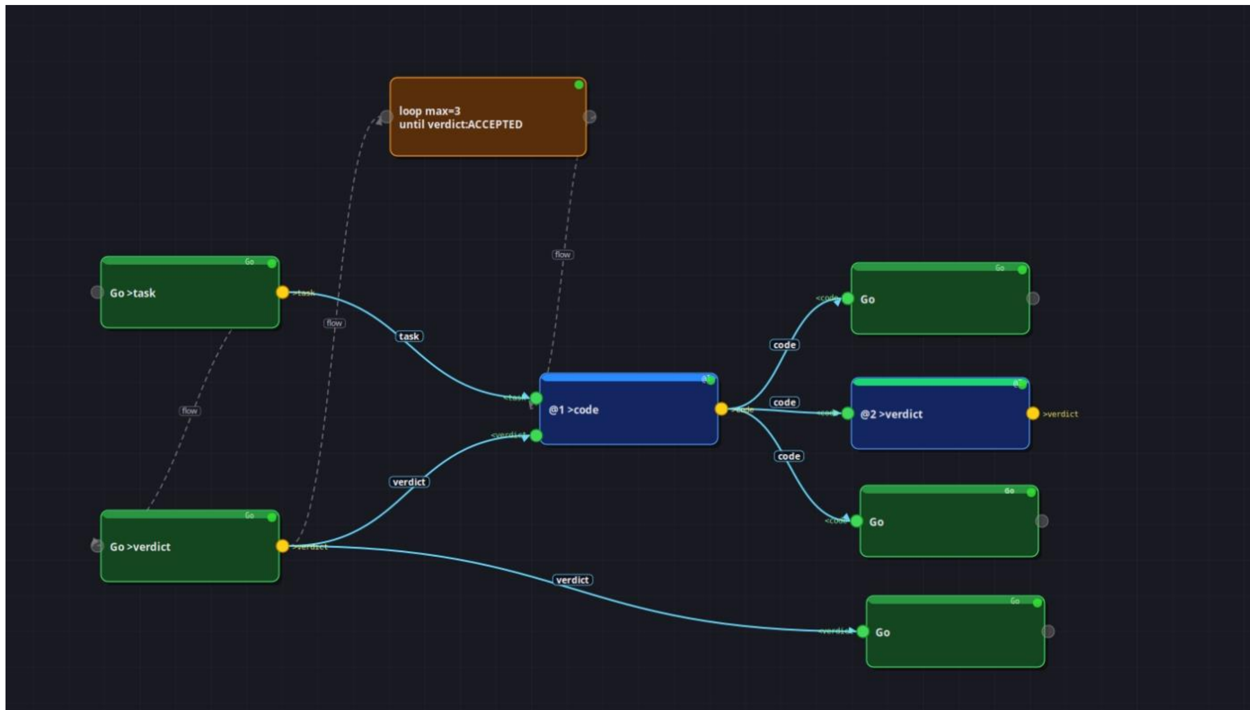
```
package main

import "fmt"

func main() {
    fmt.Println("REJECTED: no code yet")
}
```

```
package main

import (
    "fmt"
    "os"
)
```

```go
func main() {
    codePath := os.Getenv("MSH_VAR_code")
    codeBytes, _ := os.ReadFile(codePath)
    fmt.Println("=== Generated code ===")
    fmt.Println(string(codeBytes))
}
```

```go
package main

import (
    "fmt"
    "os"
)

func main() {
    verdict, _ := os.ReadFile(os.Getenv("MSH_VAR_verdict"))
    fmt.Println("=== Verdict ===")
    fmt.Println(string(verdict))
}
```

```go
package main

import (
    "fmt"
    "os"
    "os/exec"
)

func main() {
    codePath := os.Getenv("MSH_VAR_code")
    codeBytes, _ := os.ReadFile(codePath)
    fmt.Println("=== Final accepted code ===")
    fmt.Println(string(codeBytes))
    fmt.Println("=== Executing ===")

    tmp, _ := os.CreateTemp("", "msh_accepted_*.go")
    defer os.Remove(tmp.Name())
    tmp.Write(codeBytes)
    tmp.Close()

    out, err := exec.Command("go", "run", tmp.Name()).CombinedOutput()
    if err != nil {
        fmt.Fprintf(os.Stderr, "error: %v\n%s\n", err, out)
        os.Exit(1)
    }
    fmt.Print(string(out))
}
```



Pattern 8 — Multi-Stage Go + Multi-Model Pipeline

What it does: Go computes temperature statistics. LLM @1 interprets the data in a paragraph. LLM @2 compresses to a headline. Go formats a structured benchmark report.

```
package main

import (
	"fmt"
	"math/rand"
)

func main() {
	r := rand.New(rand.NewSource(42))
	for i := 0; i < 10; i++ {
		temp := 18.0 + r.Float64()*15.0
		if i > 0 {
			fmt.Print(",")
		}
		fmt.Printf("%.2f", temp)
	}
	fmt.Println()
}
```

```
package main

import (
	"fmt"
```

```
    "math"
    "os"
    "strconv"
    "strings"
)

func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_data"))
    parts := strings.Split(strings.TrimSpace(string(raw)), ",")
    vals := make([]float64, 0, len(parts))
    for _, p := range parts {
        v, err := strconv.ParseFloat(strings.TrimSpace(p), 64)
        if err == nil {
            vals = append(vals, v)
        }
    }
    n := float64(len(vals))
    sum := 0.0
    mn, mx := vals[0], vals[0]
    for _, v := range vals {
        sum += v
        if v < mn { mn = v }
        if v > mx { mx = v }
    }
    mean := sum / n
    variance := 0.0
    for _, v := range vals {
        diff := v - mean
        variance += diff * diff
    }
    stddev := math.Sqrt(variance / n)
    fmt.Printf("count=%d mean=%.2f stddev=%.2f min=%.2f max=%.2f\n",
        len(vals), mean, stddev, mn, mx)
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func readVar(name string) string {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
    s := strings.TrimSpace(string(data))
    if s == "" {
        return "[no data]"
    }
    return s
```

```
}

func main() {

fmt.Println("\u2554\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2557")
    fmt.Println("\u2551   SENSOR MONITORING REPORT   \u2551")

fmt.Println("\u2560\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2563")
    fmt.Printf("\u2551 Readings : %s\n", readVar("raw_data"))
    fmt.Printf("\u2551 Stats    : %s\n", readVar("stats"))
    fmt.Printf("\u2551 Analysis : %s\n", readVar("analysis"))
    fmt.Printf("\u2551 Alert    : %s\n", readVar("alert"))

fmt.Println("\u255a\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255d")
}
```

Pattern 9 — Routing: LLM Classifies → Conditional Go Branch

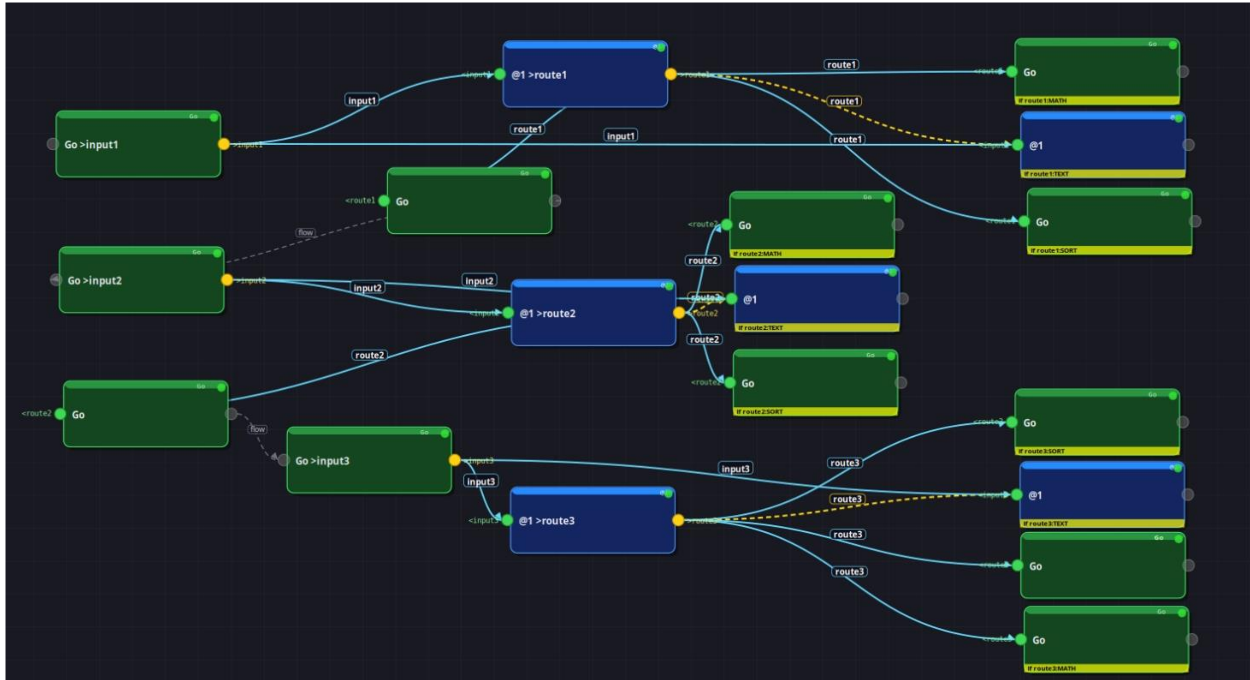
What it does: LLM @1 classifies an input into SORT, MATH, or TEXT. Only the matching conditional Go block runs; others are skipped. Three test cases cover all branches.

```
package main
import "fmt"
func main() { fmt.Println("sort this list of numbers: 7 2 9 1 5 3") }
```

```go
package main

import (
    "fmt"
    "sort"
)

func main() {
    nums := []int{7, 2, 9, 1, 5, 3}
    sort.Ints(nums)
    fmt.Printf("=== SORT branch === result: %v\n", nums)
}

package main

import "fmt"

func main() {
    fmt.Println("=== MATH branch === (unexpected for this input)")
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    r, _ := os.ReadFile(os.Getenv("MSH_VAR_route1"))
    fmt.Printf("Test1 classified as: %s\n\n", strings.TrimSpace(string(r)))
}

package main
import "fmt"
func main() { fmt.Println("what is the prime factorization of 84") }

package main
import "fmt"
func main() { fmt.Println("=== SORT branch === (unexpected)") }

package main

import "fmt"

func primeFactors(n int) []int {
    factors := []int{}
    for d := 2; d*d <= n; d++ {
        for n%d == 0 {
```

```go
            factors = append(factors, d)
            n /= d
        }
    }
    if n > 1 {
        factors = append(factors, n)
    }
    return factors
}

func main() {
    fmt.Printf("=== MATH branch === prime factors of 84: %v\n",
primeFactors(84))
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    r, _ := os.ReadFile(os.Getenv("MSH_VAR_route2"))
    fmt.Printf("Test2 classified as: %s\n\n", strings.TrimSpace(string(r)))
}

package main
import "fmt"
func main() { fmt.Println("explain what a goroutine is") }

package main
import "fmt"
func main() { fmt.Println("=== SORT branch === (unexpected)") }

package main
import "fmt"
func main() { fmt.Println("=== MATH branch === (unexpected)") }

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    r, _ := os.ReadFile(os.Getenv("MSH_VAR_route3"))
```

```
    fmt.Printf("Test3 classified as: %s\n\n", strings.TrimSpace(string(r)))
}
```



Pattern 10 — Full Pipeline: All Patterns Combined

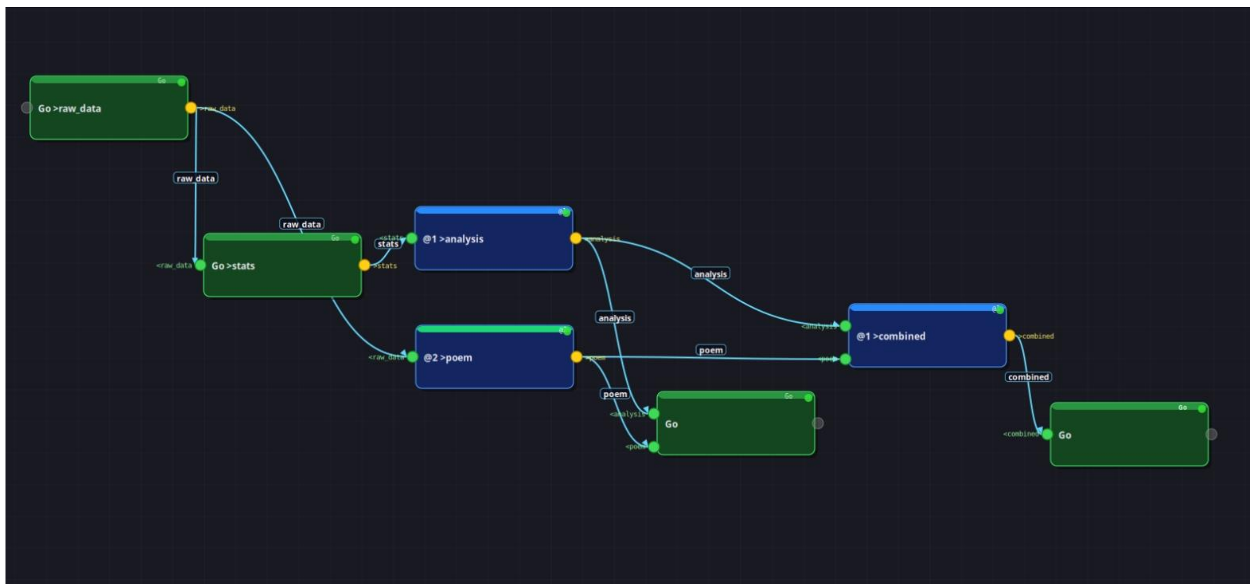
What it does: Go computes primes and derives stats. Two LLMs run asynchronously. A bash await barrier synchronises. LLM @1 synthesises. Go renders the output in a decorative frame.

```
package main

import (
    "fmt"
    "strings"
)

func main() {
    primes := []string{}
    for n := 2; len(primes) < 10; n++ {
        isPrime := true
        for d := 2; d*d <= n; d++ {
            if n%d == 0 {
                isPrime = false
                break
            }
        }
        if isPrime {
            primes = append(primes, fmt.Sprintf("%d", n))
        }
```

```go
    }
    fmt.Println(strings.Join(primes, " "))
}

package main

import (
    "fmt"
    "os"
    "strconv"
    "strings"
)

func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_data"))
    parts := strings.Fields(strings.TrimSpace(string(raw)))
    nums := make([]int, 0, len(parts))
    for _, p := range parts {
        n, _ := strconv.Atoi(p)
        nums = append(nums, n)
    }
    sum := 0
    maxGap := 0
    for i, n := range nums {
        sum += n
        if i > 0 && nums[i]-nums[i-1] > maxGap {
            maxGap = nums[i] - nums[i-1]
        }
    }
    mean := float64(sum) / float64(len(nums))
    fmt.Printf("First 10 primes: %v\nSum=%d Mean=%.1f LargestGap=%d\n",
        nums, sum, mean, maxGap)
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    analysis, _ := os.ReadFile(os.Getenv("MSH_VAR_analysis"))
    poem, _ := os.ReadFile(os.Getenv("MSH_VAR_poem"))
    fmt.Println("=== Analysis ===")
    fmt.Println(strings.TrimSpace(string(analysis)))
    fmt.Println()
    fmt.Println("=== Poem ===")
```

```
    fmt.Println(strings.TrimSpace(string(poem)))
}
```

```
package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_combined"))
    text := strings.TrimSpace(string(data))
    border := strings.Repeat("\u2550", len([]rune(text))+4)
    fmt.Printf("\u2554%s\u2557\n", border)
    fmt.Printf("\u2551  %s  \u2551\n", text)
    fmt.Printf("\u255a%s\u255d\n", border)
}
```



Pattern 11 — MShell Node with Multiple Models

What it does: Go produces `topic` and `constraints` variables. Native mshell blocks call `ollama1`/`ollama2` inline with variable interpolation. Go formats the final report.

```
package main

import "fmt"

func main() {
    fmt.Println("Go channels and goroutines")
}
```

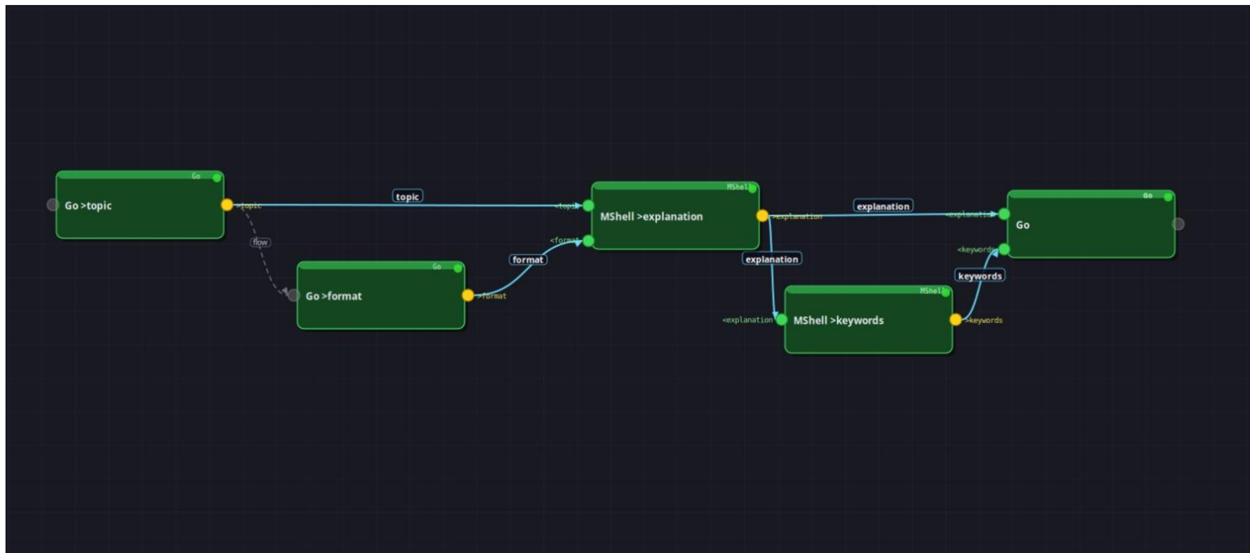
```go
package main

import "fmt"

func main() {
	fmt.Println("three bullet points, each one sentence")
}
```

ollama1 "Explain $topic in the following format: $format. Be concise and accurate."

ollama2 "Extract exactly 4 technical keywords from this text: $explanation. Reply with only the 4 words, comma-separated, nothing else."

```go
package main

import (
	"fmt"
	"os"
	"strings"
)

func avgWordLen(text string) float64 {
	words := strings.Fields(text)
	if len(words) == 0 {
		return 0
	}
	total := 0
	for _, w := range words {
		total += len(strings.Trim(w, ".,!?;:"))
	}
	return float64(total) / float64(len(words))
}

func main() {
	explanation, _ := os.ReadFile(os.Getenv("MSH_VAR_explanation"))
	keywords, _ := os.ReadFile(os.Getenv("MSH_VAR_keywords"))

	expl := strings.TrimSpace(string(explanation))
	kw := strings.TrimSpace(string(keywords))

	wordCount := len(strings.Fields(expl))
	avgLen := avgWordLen(expl)

	fmt.Println("=== Explanation ===")
	fmt.Println(expl)
	fmt.Println()
	fmt.Printf("=== Metrics ===\nWord count: %d | Avg word length: %.1f chars\n", wordCount, avgLen)
	fmt.Println()
```

```
    fmt.Println("=== Keywords ===")
    fmt.Println(kw)
}
```



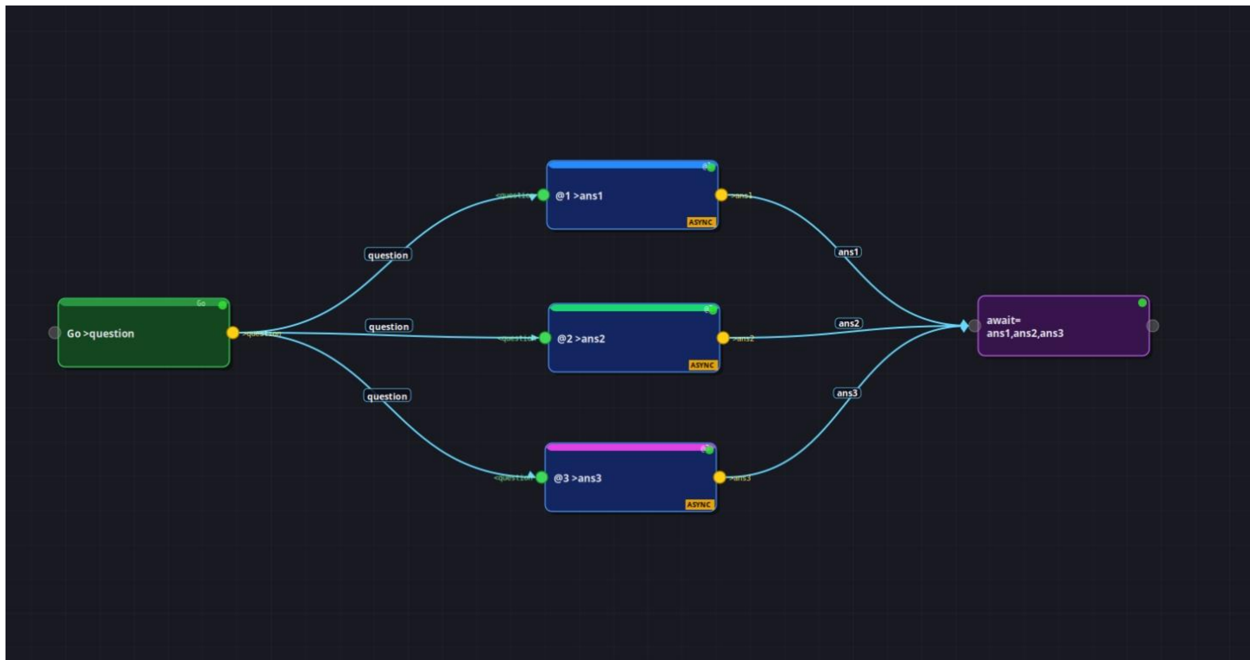
Pattern 12 — Async Parallel 3 Models + Await Barrier + Synthesis

What it does: Three LLMs answer the same question asynchronously with different instructional angles. A bash await barrier synchronises all three. LLM @1 synthesises into one sentence.

```
package main

import "fmt"

func main() {
    fmt.Println("What is the Go memory model and why does it matter for
concurrent programming?")
}
```

```
package main

import (
    "fmt"
    "os"
    "strings"
)

func readVar(name string) string {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
    return strings.TrimSpace(string(data))
}
```

```go
func wordCount(s string) int {
    return len(strings.Fields(s))
}

func main() {
    a1 := readVar("ans1")
    a2 := readVar("ans2")
    a3 := readVar("ans3")
    final := readVar("final")

    fmt.Printf("=== Beginner (%d words) ===\n%s\n\n", wordCount(a1), a1)
    fmt.Printf("=== Analogy (%d words) ===\n%s\n\n", wordCount(a2), a2)
    fmt.Printf("=== Technical (%d words) ===\n%s\n\n", wordCount(a3), a3)
    fmt.Println("=== Synthesized ===")
    fmt.Println(final)
}
```



Part II — Patterns 13–24: Advanced Node Types

New node types: **WHILE · FOREACH · TRY/CATCH · SPLIT · MERGE · CONFIG · Multi-Variable Output**

Pattern 13 — WHILE Loop: Iterative Counter with LLM Commentary

What it does: WHILE loop. Each iteration a Go block increments the counter and writes the new status directly to `MSH_VAR_*` files. LLM @1 provides one math fact per step. Stops at 4.

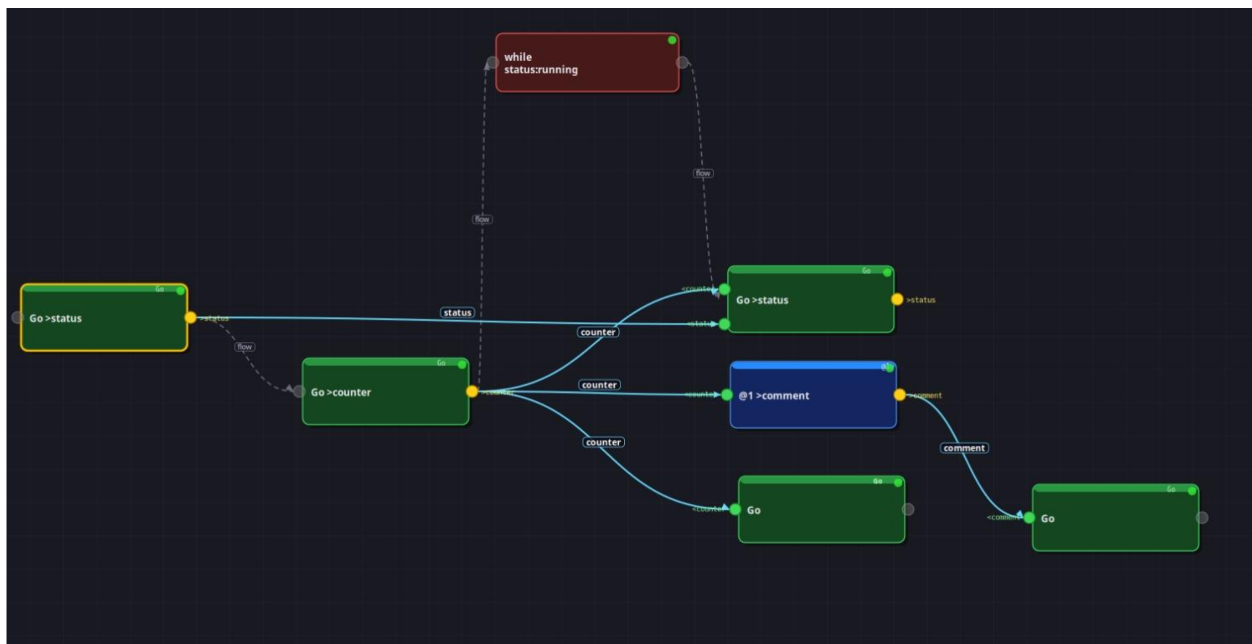
```go
package main

import "fmt"

func main() {
	fmt.Println("running")
}
```

```go
package main

import "fmt"

func main() {
	fmt.Println("0")
}
```

```go
package main

import (
	"fmt"
	"os"
	"strconv"
	"strings"
)

func main() {
	counterPath := os.Getenv("MSH_VAR_counter")
	statusPath := os.Getenv("MSH_VAR_status")

	raw, _ := os.ReadFile(counterPath)
	val, _ := strconv.Atoi(strings.TrimSpace(string(raw)))
	val++

	os.WriteFile(counterPath, []byte(fmt.Sprintf("%d", val)), 0644)

	if val >= 4 {
		os.WriteFile(statusPath, []byte("done"), 0644)
	} else {
		os.WriteFile(statusPath, []byte("running"), 0644)
	}

	fmt.Printf("%d", val)
}
```

```go
package main

import (
	"fmt"
	"os"
```

```
    "strings"
)

func main() {
    counter, _ := os.ReadFile(os.Getenv("MSH_VAR_counter"))
    comment, _ := os.ReadFile(os.Getenv("MSH_VAR_comment"))
    fmt.Printf("[iter %s] %s\n",
        strings.TrimSpace(string(counter)),
        strings.TrimSpace(string(comment)))
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    counter, _ := os.ReadFile(os.Getenv("MSH_VAR_counter"))
    fmt.Printf("=== WHILE done. Final counter = %s ===\n",
        strings.TrimSpace(string(counter)))
}
```



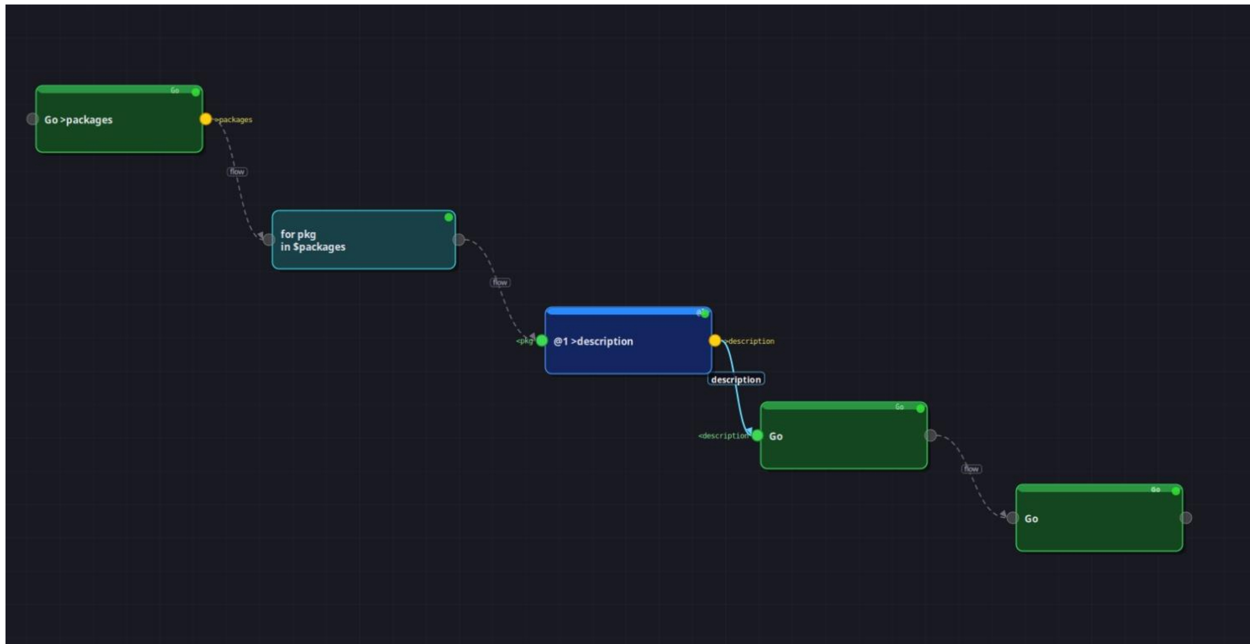
Pattern 14 — FOREACH: LLM Explains Each Go Package

What it does: Go creates a newline-separated list of Go standard library packages. FOREACH iterates line by line. LLM @1 explains each package. Go prints formatted description cards.

```go
package main

import "fmt"

func main() {
    pkgs := []string{"fmt", "strings", "sync", "net/http"}
    for _, p := range pkgs {
        fmt.Println(p)
    }
}
```

```go
package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    pkg, _ := os.ReadFile(os.Getenv("MSH_VAR_pkg"))
    desc, _ := os.ReadFile(os.Getenv("MSH_VAR_description"))
    fmt.Printf("--- %s ---\n%s\n\n",
        strings.TrimSpace(string(pkg)),
        strings.TrimSpace(string(desc)))
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("=== All packages described ===")
}
```



Pattern 15 — TRY/CATCH: Safe Go Execution with Error Capture

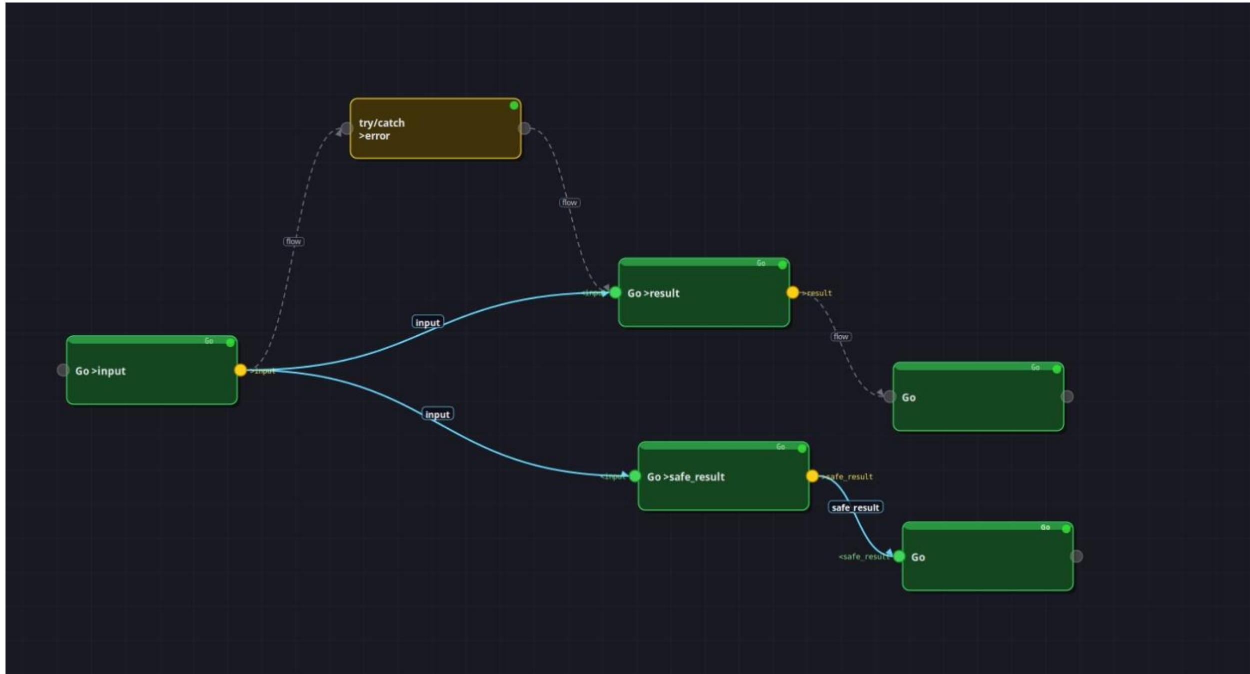
What it does: Go attempts strict RFC3339 timestamp parsing. On failure `os.Exit(1)` triggers CATCH. CATCH prints a hardcoded literal. A safe fallback Go block extracts just the date portion.

```
package main

import "fmt"

func main() {
    fmt.Println("2026-03-11T10:00:00Z")
}
```

```
package main

import (
    "fmt"
    "os"
    "strings"
    "time"
)

func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_input"))
    ts := strings.TrimSpace(string(raw))

    // Intentionally wrong layout \u2014 will fail
    wrongLayout := "02-01-2006"
    t, err := time.Parse(wrongLayout, ts)
```

```go
    if err != nil {
        fmt.Fprintf(os.Stderr, "parse error: %v\n", err)
        os.Exit(1)
    }
    fmt.Println(t)
}

package main

import "fmt"

func main() {
    fmt.Println("=== Caught error: try_block_failed ===")
    fmt.Println("Timestamp parsing failed. Using safe fallback.")
}

package main

import (
    "fmt"
    "os"
    "strings"
    "time"
)

func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_input"))
    ts := strings.TrimSpace(string(raw))

    t, err := time.Parse(time.RFC3339, ts)
    if err != nil {
        fmt.Fprintf(os.Stderr, "fallback parse error: %v\n", err)
        os.Exit(1)
    }
    fmt.Printf("year=%d month=%s day=%d hour=%d UTC\n",
        t.Year(), t.Month(), t.Day(), t.Hour())
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    result, _ := os.ReadFile(os.Getenv("MSH_VAR_safe_result"))
    fmt.Println("=== Safe result ===")
```

```
    fmt.Println(strings.TrimSpace(string(result)))
}
```



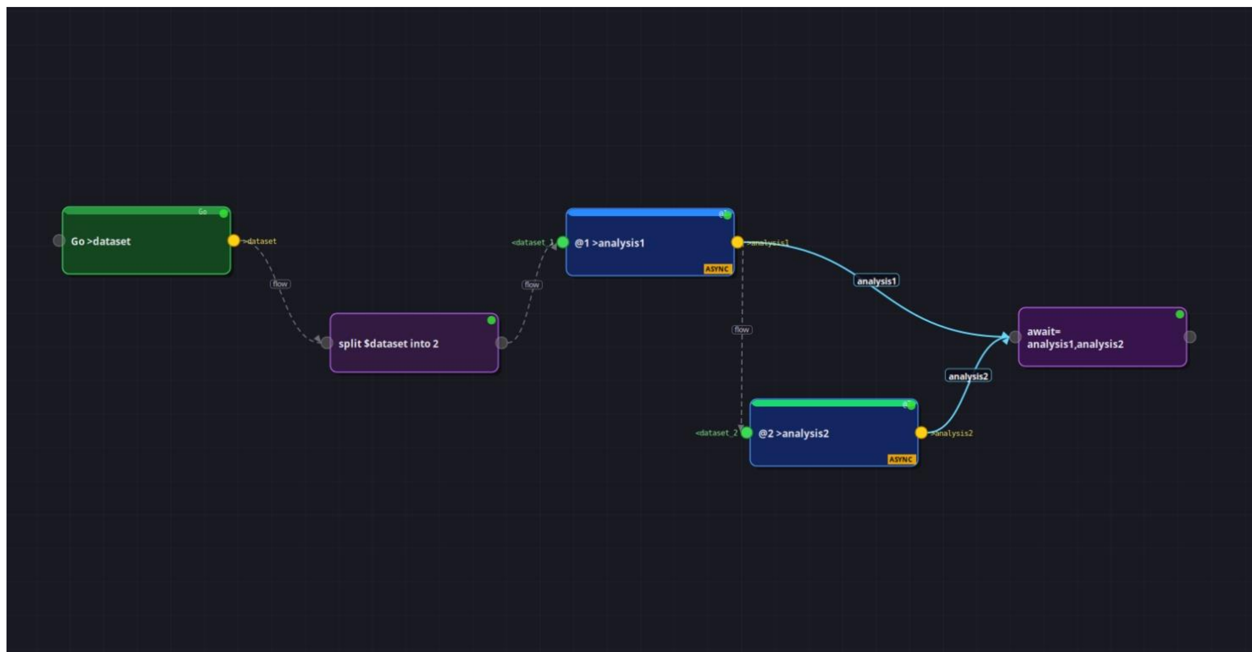
Pattern 16 — SPLIT + MERGE: Divide-and-Conquer Analysis

What it does: Go creates a two-line latency dataset. SPLIT divides by lines. Two async LLMs analyse each half. Bash await synchronises. LLM @1 synthesises a comparative summary.

```go
package main

import "fmt"

func main() {
    fmt.Println("12,45,23,67,34,89,15,52,38,71")
    fmt.Println("8,19,44,11,56,23,77,31,48,62")
}
```

```go
package main

import (
    "fmt"
    "os"
    "strings"
)

func readVar(name string) string {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
```

```
    return strings.TrimSpace(string(data))
}

func main() {
    fmt.Println("=== DC-East ===")
    fmt.Printf("Data     : %s\n", readVar("dataset_1"))
    fmt.Printf("Analysis : %s\n", readVar("analysis1"))
    fmt.Println()
    fmt.Println("=== DC-West ===")
    fmt.Printf("Data     : %s\n", readVar("dataset_2"))
    fmt.Printf("Analysis : %s\n", readVar("analysis2"))
    fmt.Println()
    fmt.Println("=== Comparative Summary ===")
    fmt.Println(readVar("combined"))
}
```



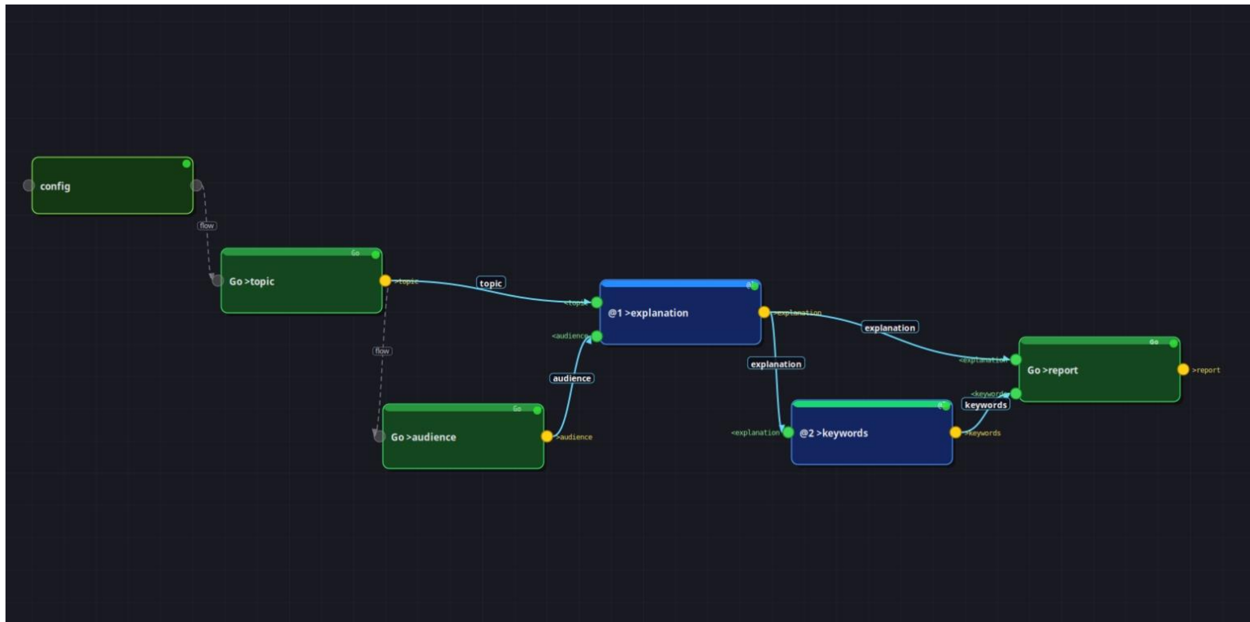
Pattern 17 — CONFIG Node: Parameterized Go Pipeline

What it does: CONFIG block documents pipeline parameters. Go blocks set the actual runtime values. LLM @1 explains a Go topic. LLM @2 extracts keywords. Go formats a full report.

```
topic=Go interfaces
audience=junior developer
max_words=60
```

```
package main

import "fmt"
```

```go
func main() {
    fmt.Println("Go interfaces")
}

package main

import "fmt"

func main() {
    fmt.Println("junior developer")
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func readVar(name string) string {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
    return strings.TrimSpace(string(data))
}

func main() {
    lines := []string{
        "=== Workflow Report ===",
        fmt.Sprintf("Topic    : %s", readVar("topic")),
        fmt.Sprintf("Audience : %s", readVar("audience")),
        "",
        "Explanation:",
        readVar("explanation"),
        "",
        fmt.Sprintf("Keywords : %s", readVar("keywords")),
    }
    report := strings.Join(lines, "\n")
    fmt.Println(report)
}
```



Pattern 18 — FOREACH + Async LLM: Parallel Go Pattern Batch

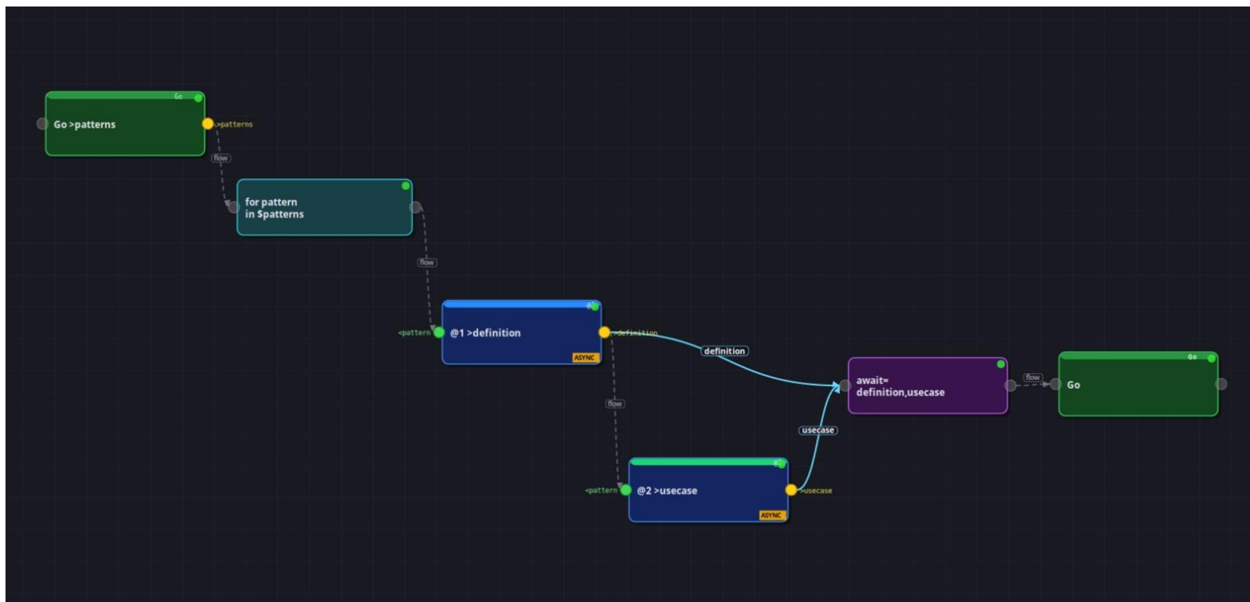
What it does: Go creates a list of Go design patterns. FOREACH iterates. Per item two async LLMs generate explanation and analogy simultaneously. Bash await + Go prints the pair.

```go
package main

import "fmt"

func main() {
    items := []string{
        "Functional Options",
        "Worker Pool",
        "Pipeline",
        "Fan-Out Fan-In",
    }
    for _, item := range items {
        fmt.Println(item)
    }
}
```

```go
package main

import (
    "fmt"
    "os"
    "strings"
)
```

```go
func main() {
    pattern, _ := os.ReadFile(os.Getenv("MSH_VAR_pattern"))
    definition, _ := os.ReadFile(os.Getenv("MSH_VAR_definition"))
    usecase, _ := os.ReadFile(os.Getenv("MSH_VAR_usecase"))

    fmt.Printf("=== %s ===\n", strings.TrimSpace(string(pattern)))
    fmt.Printf("Definition : %s\n", strings.TrimSpace(string(definition)))
    fmt.Printf("Use case   : %s\n\n", strings.TrimSpace(string(usecase)))
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("=== All patterns described ===")
}
```



Pattern 19 — WHILE Quality Gate: Generate Go Code Until Score >= 8

What it does: WHILE loop. Go increments counter and writes status. LLM @1 generates Go code. LLM @2 scores it 1–10. Go checks the threshold and writes done when score >= 8.

```go
package main

import "fmt"

func main() {
    fmt.Println("Write an idiomatic Go function that safely reads a file and
returns its contents as a string, handling all errors properly.")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("running")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("0")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("0")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("")
}
```

```go
package main

import (
    "fmt"
    "os"
    "strconv"
    "strings"
)

func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_iteration"))
    val, _ := strconv.Atoi(strings.TrimSpace(string(raw)))
    val++
    os.WriteFile(os.Getenv("MSH_VAR_iteration"), []byte(fmt.Sprintf("%d",
val)), 0644)
    fmt.Printf("%d", val)
}
```

```go
package main

import (
	"fmt"
	"os"
	"strconv"
	"strings"
)

func main() {
	iterPath := os.Getenv("MSH_VAR_iteration")
	scorePath := os.Getenv("MSH_VAR_score")
	statusPath := os.Getenv("MSH_VAR_status")

	iter, _ := os.ReadFile(iterPath)
	scoreRaw, _ := os.ReadFile(scorePath)
	sc := strings.TrimSpace(string(scoreRaw))

	fmt.Printf("[Iter %s] Score=%s\n", strings.TrimSpace(string(iter)), sc)

	scoreVal, err := strconv.Atoi(sc)
	if err == nil && scoreVal >= 8 {
		os.WriteFile(statusPath, []byte("done"), 0644)
		fmt.Println("done")
	} else {
		os.WriteFile(statusPath, []byte("running"), 0644)
		fmt.Println("running")
	}
}
```

```go
package main

import (
	"fmt"
	"os"
	"strings"
)

func main() {
	snippet, _ := os.ReadFile(os.Getenv("MSH_VAR_snippet"))
	score, _ := os.ReadFile(os.Getenv("MSH_VAR_score"))
	fmt.Printf("=== Accepted snippet (score=%s) ===\n",
		strings.TrimSpace(string(score)))
	fmt.Println(strings.TrimSpace(string(snippet)))
}
```

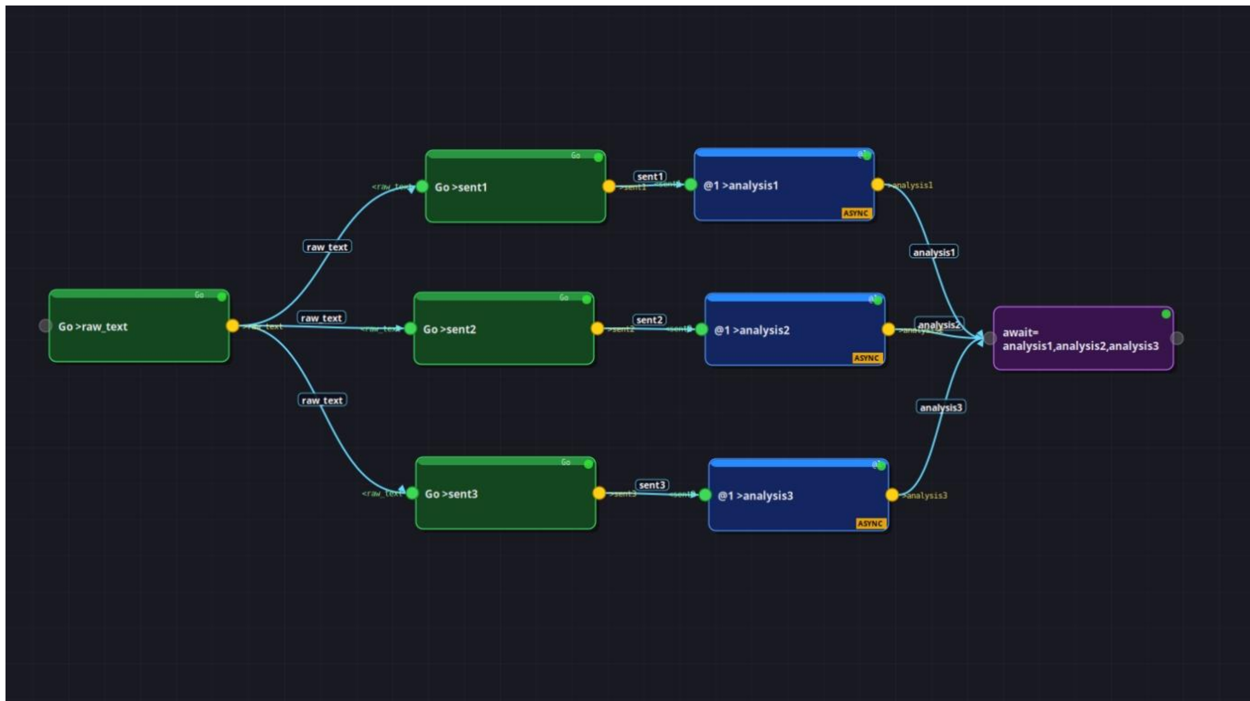
Pattern 20 — SPLIT + Async + MERGE: Map-Reduce Go Source Analysis

What it does: Go splits a Go concurrency text into three sentence chunks. Three async LLMs extract the main concept from each (MAP). Bash await. LLM @2 synthesises a theme (REDUCE).

```
package main

import "fmt"

func main() {
    fmt.Print("Go achieves concurrency through goroutines, which are lightweight threads managed by the runtime. ")
    fmt.Print("Channels provide a safe way for goroutines to communicate and synchronize. ")
    fmt.Print("The select statement allows a goroutine to wait on multiple channel operations. ")
    fmt.Print("The sync package offers primitives like Mutex and WaitGroup for shared state. ")
    fmt.Println("Go's concurrency model follows the principle: do not communicate by sharing memory, share memory by communicating.")
}

package main

import (
    "fmt"
    "os"
    "strings"
)
```

```go
func main() {
	raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_text"))
	sentences := strings.SplitN(strings.TrimSpace(string(raw)), ". ", -1)
	if len(sentences) > 0 {
		fmt.Println(strings.TrimSpace(sentences[0]))
	}
}

package main

import (
	"fmt"
	"os"
	"strings"
)

func main() {
	raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_text"))
	sentences := strings.SplitN(strings.TrimSpace(string(raw)), ". ", -1)
	chunk := []string{}
	for _, s := range sentences[1:3] {
		t := strings.TrimSpace(s)
		if t != "" {
			chunk = append(chunk, t)
		}
	}
	fmt.Println(strings.Join(chunk, ". "))
}

package main

import (
	"fmt"
	"os"
	"strings"
)

func main() {
	raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_text"))
	sentences := strings.SplitN(strings.TrimSpace(string(raw)), ". ", -1)
	chunk := []string{}
	for _, s := range sentences[3:] {
		t := strings.TrimSpace(s)
		if t != "" {
			chunk = append(chunk, t)
		}
	}
	fmt.Println(strings.Join(chunk, ". "))
}
```

```go
package main

import (
    "fmt"
    "os"
    "strings"
)

func readVar(name string) string {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
    return strings.TrimSpace(string(data))
}

func main() {
    fmt.Println("=== Map ===")
    fmt.Printf("Chunk 1: %s\n", readVar("analysis1"))
    fmt.Printf("Chunk 2: %s\n", readVar("analysis2"))
    fmt.Printf("Chunk 3: %s\n", readVar("analysis3"))
    fmt.Println()
    fmt.Println("=== Reduce ===")
    fmt.Println(readVar("summary"))
}
```



Pattern 21 — TRY/CATCH + LOOP: Resilient Go Code Retry

What it does: LOOP max=3. LLM @1 generates Go code. TRY compiles and runs via `go run` temp file. On failure CATCH records the error for the next self-correction iteration. Bash prints final status.

```
package main
import "fmt"
func main() {
    fmt.Println(`Write a Go program that parses the JSON string
{"name":"Alice","age":30} and prints the name field. No external imports
beyond encoding/json and fmt.`)
}
```

```
package main
import "fmt"
func main() { fmt.Println("fail") }
```

```
package main
import "fmt"
func main() { fmt.Println("none") }
```

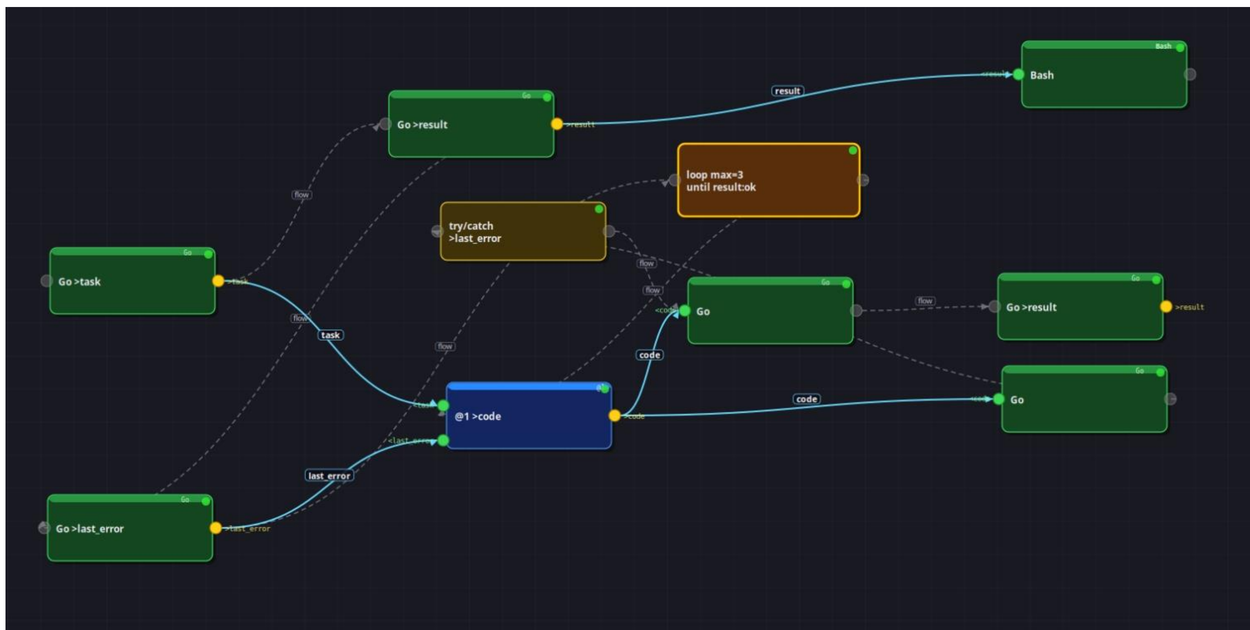
```
package main
import (
    "fmt"
    "os"
)
func main() {
    code, _ := os.ReadFile(os.Getenv("MSH_VAR_code"))
    fmt.Println("=== Generated Code ===")
    fmt.Print(string(code))
}
```

```
package main
import (
    "fmt"
    "os"
    "os/exec"
)
func main() {
    code, _ := os.ReadFile(os.Getenv("MSH_VAR_code"))
    resultPath := os.Getenv("MSH_VAR_result")
    tmp, err := os.CreateTemp("", "msh_p21_*.go")
    if err != nil { os.Exit(1) }
    tmpName := tmp.Name()
    tmp.Write(code)
    tmp.Close()
    out, err := exec.Command("go", "run", tmpName).CombinedOutput()
    os.Remove(tmpName)
    if err != nil {
        fmt.Fprintf(os.Stderr, "run failed: %s\n", out)
```

```
        os.Exit(1)
    }
    fmt.Print(string(out))
    os.WriteFile(resultPath, []byte("ok\n"), 0644)
    fmt.Println("ok")
}
```

```
package main
import (
    "fmt"
    "os"
)
func main() {
    fmt.Println("=== Error: try_block_failed ===")
    os.WriteFile(os.Getenv("MSH_VAR_result"), []byte("fail\n"), 0644)
    fmt.Println("fail")
}
```

```
echo "=== Final status: $(cat $(printenv MSH_VAR_result) | tr -d '\n') ==="
```



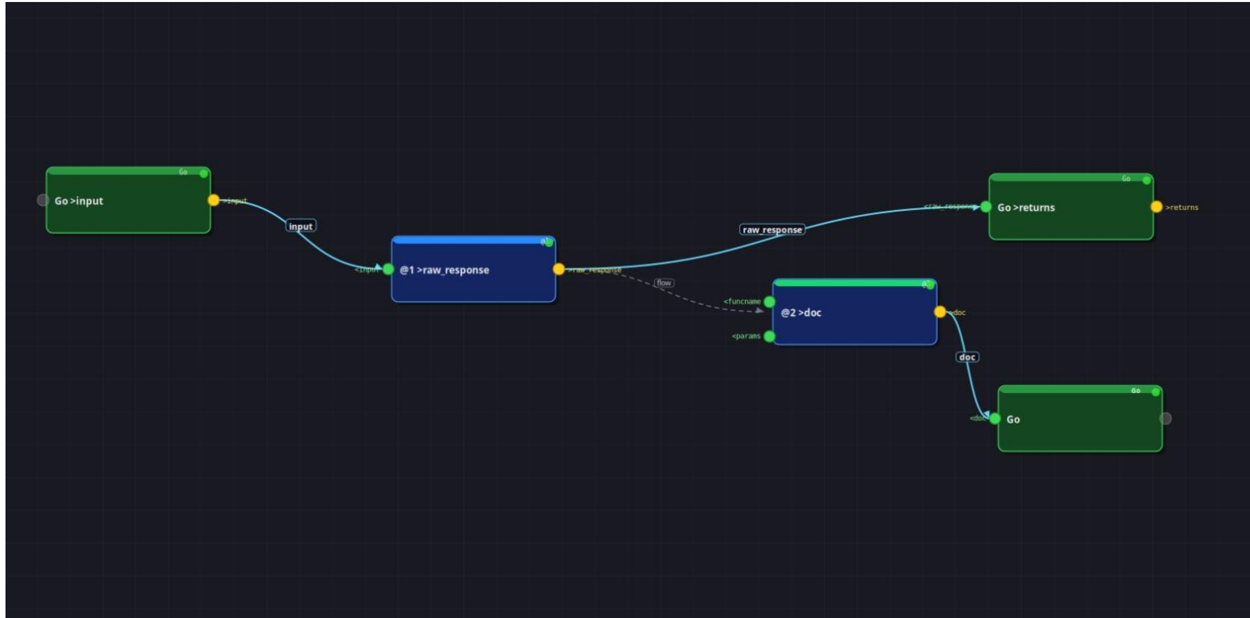
Pattern 22 — Multi-Variable Output: Structured Go Function Extraction

What it does: LLM @1 responds in strict 3-line format (FUNCNAME/SIGNATURE/AUDIENCE). Go with 3 >`outvar` parses with regex and writes each field directly to its `MSH_VAR_*` file. LLM @2 adapts.

```
package main

import "fmt"

func main() {
```

```
    fmt.Println("A function that takes a context, a database connection, a user ID string, and a boolean includeDeleted flag, and returns a pointer to a User struct and an error.")
}

package main

import (
    "fmt"
    "os"
    "regexp"
    "strings"
)

func extract(text, field string) string {
    re := regexp.MustCompile(field + `:\s*(.+)`)
    m := re.FindStringSubmatch(text)
    if len(m) > 1 {
        return strings.TrimSpace(m[1])
    }
    return "n/a"
}

func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_response"))
    text := string(raw)

    funcname := extract(text, "FUNCNAME")
    params := extract(text, "PARAMS")
    returns := extract(text, "RETURNS")

    os.WriteFile(os.Getenv("MSH_VAR_funcname"), []byte(funcname), 0644)
    os.WriteFile(os.Getenv("MSH_VAR_params"), []byte(params), 0644)
    os.WriteFile(os.Getenv("MSH_VAR_returns"), []byte(returns), 0644)

    fmt.Printf("FuncName : %s\n", funcname)
    fmt.Printf("Params   : %s\n", params)
    fmt.Printf("Returns  : %s\n", returns)
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    doc, _ := os.ReadFile(os.Getenv("MSH_VAR_doc"))
```

```
    fmt.Println()
    fmt.Println("=== Generated doc comment ===")
    fmt.Println(strings.TrimSpace(string(doc)))
}
```



Pattern 23 — CONFIG + WHILE + Multi-Model: Adaptive Go Code Pipeline

What it does: CONFIG documents parameters. WHILE loop runs until quality >= 7. Go increments counter. LLM @1 explains. LLM @2 scores. Go checks threshold. LLM @3 polishes the final result.

```
subject=Go generics and type constraints
target_audience=experienced Go developer new to generics
quality_threshold=7
```

```go
package main

import "fmt"

func main() {
    fmt.Println("Go generics and type constraints")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("experienced Go developer new to generics")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("running")
}
```

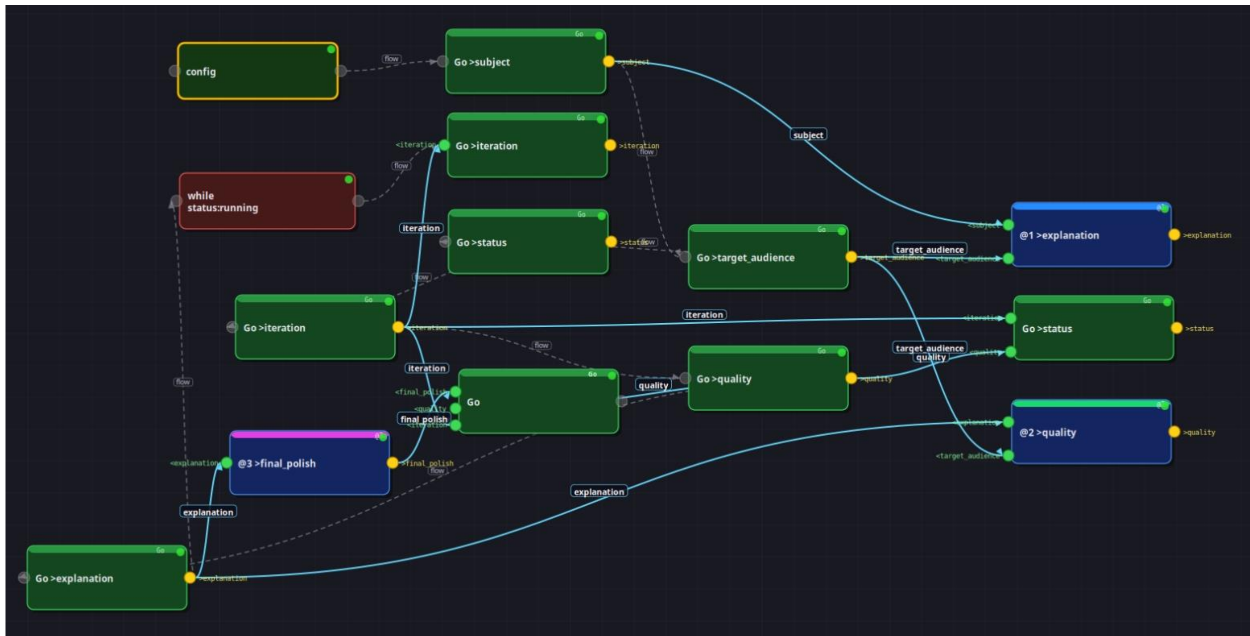
```go
package main

import "fmt"

func main() {
    fmt.Println("0")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("0")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("")
}
```

```go
package main

import (
    "fmt"
    "os"
    "strconv"
    "strings"
)

func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_iteration"))
    val, _ := strconv.Atoi(strings.TrimSpace(string(raw)))
    val++
    os.WriteFile(os.Getenv("MSH_VAR_iteration"), []byte(fmt.Sprintf("%d",
val)), 0644)
    fmt.Printf("%d", val)
}
```

```go
package main

import (
	"fmt"
	"os"
	"strconv"
	"strings"
)

func main() {
	iter, _ := os.ReadFile(os.Getenv("MSH_VAR_iteration"))
	scoreRaw, _ := os.ReadFile(os.Getenv("MSH_VAR_quality"))
	statusPath := os.Getenv("MSH_VAR_status")

	sc := strings.TrimSpace(string(scoreRaw))
	fmt.Printf("[Iter %s] Quality: %s\n", strings.TrimSpace(string(iter)),
sc)

	q, err := strconv.Atoi(sc)
	if err == nil && q >= 7 {
		os.WriteFile(statusPath, []byte("done"), 0644)
		fmt.Println("done")
	} else {
		os.WriteFile(statusPath, []byte("running"), 0644)
		fmt.Println("running")
	}
}

package main

import (
	"fmt"
	"os"
	"strings"
)

func readVar(name string) string {
	data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
	return strings.TrimSpace(string(data))
}

func main() {
	fmt.Printf("=== Final (score=%s, iters=%s) ===\n",
		readVar("quality"), readVar("iteration"))
	fmt.Println(readVar("final_polish"))
}
```



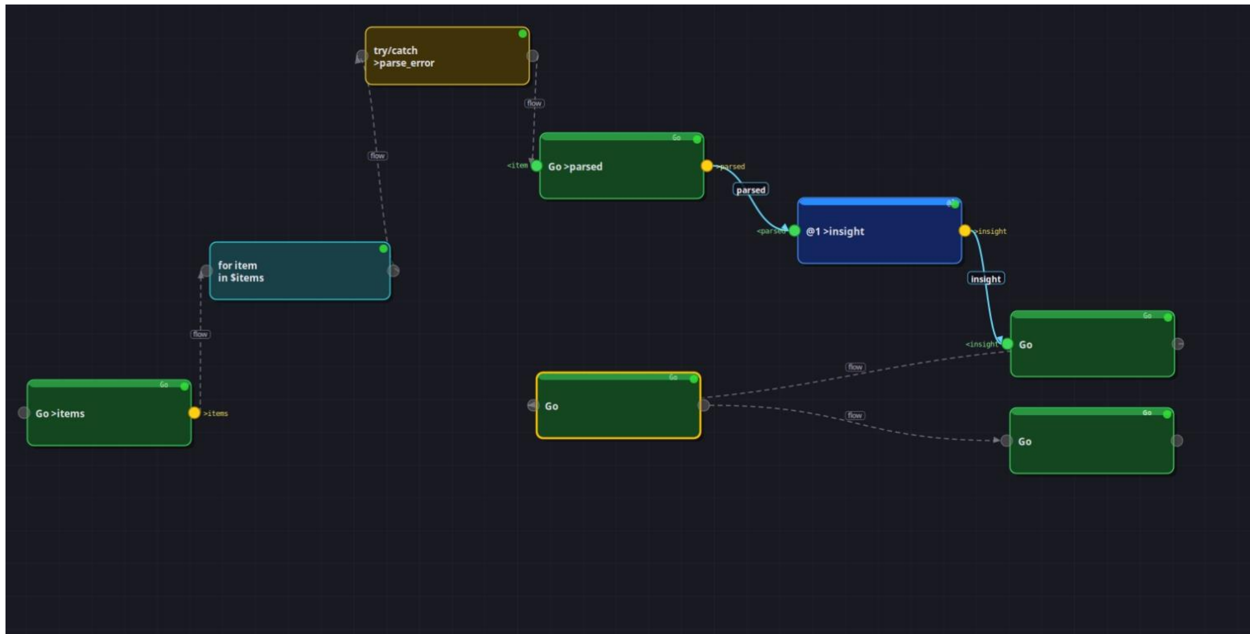
Pattern 24 — FOREACH + TRY/CATCH: Fault-Tolerant Go Batch Processing

What it does: Go creates a list of JSON strings including one intentionally broken. FOREACH iterates. TRY block parses JSON via `encoding/json`. On failure CATCH prints `[ERR]`. Pipeline never stops.

```go
package main

import "fmt"

func main() {
    records := []string{
        `{"name":"Alice","role":"engineer","active":true}`,
        `{name: broken json}`,
        `{"name":"Bob","role":"manager","active":false}`,
        `{"name":"Carol","role":"designer","active":true}`,
    }
    for _, r := range records {
        fmt.Println(r)
    }
}
```

```go
package main

import (
    "encoding/json"
    "fmt"
    "os"
    "strings"
)
```

```go
func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_item"))
    line := strings.TrimSpace(string(raw))

    var obj map[string]interface{}
    if err := json.Unmarshal([]byte(line), &obj); err != nil {
        fmt.Fprintf(os.Stderr, "parse error: %v\n", err)
        os.Exit(1)
    }
    fmt.Printf("Parsed OK: %v\n", obj)
}

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    insight, _ := os.ReadFile(os.Getenv("MSH_VAR_insight"))
    fmt.Printf("[OK] %s\n", strings.TrimSpace(string(insight)))
}

package main

import "fmt"

func main() {
    fmt.Println("[ERR] Parse failed: try_block_failed")
}

package main

import "fmt"

func main() {
    fmt.Println("=== Batch complete. Errors were isolated, pipeline never
stopped. ===")
}
```



Quick Reference: Common Go Patterns

Read entire variable file:

```
data, _ := os.ReadFile(os.Getenv("MSH_VAR_name"))
val := strings.TrimSpace(string(data))
```

Write to multiple output variables:

```
os.WriteFile(os.Getenv("MSH_VAR_field1"), []byte(val1), 0644)
os.WriteFile(os.Getenv("MSH_VAR_field2"), []byte(val2), 0644)
// stdout goes to terminal only — not captured
```

WHILE exit:

```
status := "running"
if score >= threshold { status = "done" }
os.WriteFile(os.Getenv("MSH_VAR_status"), []byte(status+"\n"), 0644)
```

LOOP exit (last stdout line):

```
// Must be the absolute last line printed in the loop body
fmt.Println("ok")   // or "fail"
```

Compile and run LLM-generated code:

```
tmp, _ := os.CreateTemp("", "msh_gen_*.go")
tmpName := tmp.Name()
tmp.Write(code); tmp.Close()
out, err := exec.Command("go", "run", tmpName).CombinedOutput()
os.Remove(tmpName)
```

```go
if err != nil { fmt.Fprintf(os.Stderr, "%s", out); os.Exit(1) }
fmt.Print(string(out))
```

Appendix I: Code examples for each pattern

/home/igor > Sent to mshell (1736 bytes)

Received from GUI editor:

Pattern 1 Linear Data Pipeline (Go)

```go >raw

package main

import "fmt"

func main() {

    fmt.Println("42")

}

```

```go <raw >squared

package main

import (

    "fmt"

    "os"

    "strconv"

    "strings"

)


func main() {

    data, _ := os.ReadFile(os.Getenv("MSH_VAR_raw"))

    n, _ := strconv.Atoi(strings.TrimSpace(string(data)))

---
```

~~~
    fmt.Printf("value=%d squared=%d cubed=%d\n", n, n*n, n*n*n)
}
```
```go <squared >checked
package main
import (
    "fmt"
    "os"
    "strings"
)
func isPrime(n int) bool {
    if n < 2 {
        return false
    }
    for i := 2; i*i <= n; i++ {
        if n%i == 0 {
            return false
        }
    }
    return true
}
func main() {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_squared"))
    line := strings.TrimSpace(string(data))
    // extract original value
    var val int
    fmt.Sscanf(line, "value=%d", &val)
~~~

~~~~
    prime := "not prime"
    if isPrime(val) {
        prime = "prime"
    }
    fmt.Printf("%s | primality=%s\n", line, prime)
}
```
```go <checked >classified
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_checked"))
    line := strings.TrimSpace(string(data))
    var val int
    fmt.Sscanf(line, "value=%d", &val)
    fb := "none"
    switch {
    case val%15 == 0:
        fb = "FizzBuzz"
    case val%3 == 0:
        fb = "Fizz"
    case val%5 == 0:
        fb = "Buzz"
~~~~

    }
    fmt.Printf("%s | fizzbuzz=%s\n", line, fb)
}
```

```go <classified
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_classified"))
    fmt.Println("=== Pipeline Report ===")
    fmt.Println(strings.TrimSpace(string(data)))
}
```

42

value=42 squared=1764 cubed=74088

value=42 squared=1764 cubed=74088 | primality=not prime

value=42 squared=1764 cubed=74088 | primality=not prime | fizzbuzz=Fizz

/home/igor > === Pipeline Report ===

value=42 squared=1764 cubed=74088 | primality=not prime | fizzbuzz=Fizz

----

/home/igor > Sent to mshell (1916 bytes)

Received from GUI editor:
```

Pattern 2 LLM in the Middle

```go >data
package main

import "fmt"

func main() {
    logs := []string{
        "2026-03-11 10:00:01 GET /api/users 200 45ms",
        "2026-03-11 10:00:03 POST /api/login 401 12ms",
        "2026-03-11 10:00:07 GET /api/data 500 2301ms",
        "2026-03-11 10:00:09 GET /api/users 200 48ms",
        "2026-03-11 10:00:15 DELETE /api/item 403 9ms",
        "2026-03-11 10:00:22 GET /api/data 500 3100ms",
    }
    for _, l := range logs {
        fmt.Println(l)
    }
}
```

<!--@1 <data >analysis

The input contains HTTP server log lines. You MUST respond with at least one sentence. Analyze the logs in one paragraph: identify error patterns, slow responses, and any security concerns worth noting. Do not leave the response empty.

-->

```go <analysis
package main
```

```
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    text, _ := os.ReadFile(os.Getenv("MSH_VAR_analysis"))
    content := strings.TrimSpace(string(text))
    // Fallback if LLM returned empty response
    if content == "" {
        fmt.Println("LLM analysis received: [empty response \u2014 model returned no
content]")
        fmt.Println("HTTP codes mentioned: [unknown]")
        fmt.Println("--- Analysis ---")
        fmt.Println("[No analysis available]")
        return
    }
    words := strings.Fields(content)
    sentences := strings.Count(content, ".") + strings.Count(content, "!") +
strings.Count(content, "?")

    codes := []string{"200", "401", "403", "404", "500", "503"}
    mentioned := []string{}
    for _, code := range codes {
        if strings.Contains(content, code) {
            mentioned = append(mentioned, code)
        }
    }
```

```
    fmt.Printf("LLM analysis received: %d words, ~%d sentences\n", len(words), sentences)
    fmt.Printf("HTTP codes mentioned: %s\n", strings.Join(mentioned, ", "))
    fmt.Println("--- Analysis ---")
    fmt.Println(content)
}
```

2026-03-11 10:00:01 GET /api/users 200 45ms

2026-03-11 10:00:03 POST /api/login 401 12ms

2026-03-11 10:00:07 GET /api/data 500 2301ms

2026-03-11 10:00:09 GET /api/users 200 48ms

2026-03-11 10:00:15 DELETE /api/item 403 9ms

2026-03-11 10:00:22 GET /api/data 500 3100ms

The server logs reveal significant operational and security concerns that require immediate attention. The `/api/data` endpoint is experiencing consistent failures with 500 errors and extremely slow response times (2.3-3.1 seconds), indicating a critical backend issue such as database timeouts, memory problems, or resource exhaustion that needs urgent investigation. From a security perspective, there are authentication failures with a 401 on `/api/login` and a 403 forbidden response on `/api/item` deletion, which could suggest either legitimate access control working properly or potential unauthorized access attempts that warrant monitoring. The normal `/api/users` GET requests show healthy ~45ms response times, indicating the server infrastructure is capable of good performance when not affected by the `/api/data` endpoint issues, suggesting the problem is likely isolated to specific backend services or database queries rather than general server overload.

/home/igor > LLM analysis received: 131 words, ~6 sentences

HTTP codes mentioned: 401, 403, 500

--- Analysis ---

The server logs reveal significant operational and security concerns that require immediate attention. The `/api/data` endpoint is experiencing consistent failures with 500 errors and extremely slow response times (2.3-3.1 seconds), indicating a critical backend issue such as database timeouts, memory problems, or resource exhaustion that needs urgent investigation. From a security perspective, there are authentication failures with a 401 on

`/api/login` and a 403 forbidden response on `/api/item` deletion, which could suggest either legitimate access control working properly or potential unauthorized access attempts that warrant monitoring. The normal `/api/users` GET requests show healthy ~45ms response times, indicating the server infrastructure is capable of good performance when not affected by the `/api/data` endpoint issues, suggesting the problem is likely isolated to specific backend services or database queries rather than general server overload.

/home/igor > Sent to mshell (2255 bytes)

Received from GUI editor:

Pattern 3 Fan-Out: One Variable - Many Consumers

````
```go >data

package main

import (

   "fmt"

   "math/rand"

)

func main() {

   r := rand.New(rand.NewSource(2026))

   nums := make([]int, 12)

   for i := range nums {

      nums[i] = r.Intn(100) + 1

   }

   for i, n := range nums {

      if i > 0 {

         fmt.Print(",")

      }

      fmt.Print(n)
````
````

````
  }
  fmt.Println()
}
```
```go <data
package main
import (
  "fmt"
  "os"
  "strconv"
  "strings"
)
func main() {
  raw, _ := os.ReadFile(os.Getenv("MSH_VAR_data"))
  parts := strings.Split(strings.TrimSpace(string(raw)), ",")
  nums := make([]int, 0, len(parts))
  for _, p := range parts {
    n, err := strconv.Atoi(strings.TrimSpace(p))
    if err == nil {
      nums = append(nums, n)
    }
  }
  min, max, sum := nums[0], nums[0], 0
  for _, n := range nums {
    sum += n
    if n < min { min = n }
    if n > max { max = n }
````

```
    }
    mean := float64(sum) / float64(len(nums))
    fmt.Printf("Go stats: count=%d sum=%d min=%d max=%d mean=%.2f\n",
        len(nums), sum, min, max, mean)
}
```

```go <data
package main
import (
    "fmt"
    "os"
    "sort"
    "strconv"
    "strings"
)
func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_data"))
    parts := strings.Split(strings.TrimSpace(string(raw)), ",")
    nums := make([]int, 0, len(parts))
    for _, p := range parts {
        n, err := strconv.Atoi(strings.TrimSpace(p))
        if err == nil {
            nums = append(nums, n)
        }
    }
    sorted := make([]int, len(nums))
    copy(sorted, nums)
```

```
    sort.Ints(sorted)
    n := len(sorted)
    var median float64
    if n%2 == 0 {
        median = float64(sorted[n/2-1]+sorted[n/2]) / 2.0
    } else {
        median = float64(sorted[n/2])
    }
    sum := 0
    for _, v := range nums { sum += v }
    mean := float64(sum) / float64(n)
    skew := "symmetric"
    if mean > median+2 { skew = "right-skewed" }
    if mean < median-2 { skew = "left-skewed" }
    fmt.Printf("Go median: %.1f | distribution: %s\n", median, skew)
}
```

<!--@1 <data

The input is a comma-separated list of integers. In one sentence, describe what is statistically interesting about this dataset.

-->

89,4,8,50,40,73,72,10,28,36,38,97

Go median: 39.0 | distribution: right-skewed

Go median: 39.0 | distribution: right-skewed

This dataset shows a fairly uniform distribution across a wide range (4 to 97) with a mean of approximately 45.4 and no obvious clustering or patterns, suggesting it could represent

random sampling from a larger population rather than measurements with natural groupings or systematic trends.

/home/igor > Sent to mshell (1278 bytes)

Received from GUI editor:

Pattern 4 LLM Code Generation - Compile & Execute

~~~
```go >task

package main

import "fmt"

func main() {

   fmt.Println("Write a Go program that generates the first 10 Fibonacci numbers and prints their sum and product.")

}

```

<!--@1 <task >code

Write only a complete, self-contained Go program that solves the task.

No markdown fences. No explanation. Start directly with: package main

-->

```go <code

package main

import (

   "fmt"

   "os"

   "os/exec"

)

func main() {

   codePath := os.Getenv("MSH_VAR_code")
~~~

```
    codeBytes, err := os.ReadFile(codePath)
    if err != nil {
        fmt.Fprintf(os.Stderr, "cannot read code: %v\n", err)
        os.Exit(1)
    }
    fmt.Println("=== Generated Go code ===")
    fmt.Println(string(codeBytes))
    fmt.Println("=== Executing ===")

    tmpFile, err := os.CreateTemp("", "msh_gen_*.go")
    if err != nil {
        fmt.Fprintf(os.Stderr, "cannot create temp file: %v\n", err)
        os.Exit(1)
    }
    defer os.Remove(tmpFile.Name())
    tmpFile.Write(codeBytes)
    tmpFile.Close()
    out, err := exec.Command("go", "run", tmpFile.Name()).CombinedOutput()
    if err != nil {
        fmt.Fprintf(os.Stderr, "execution error: %v\n%s\n", err, out)
        os.Exit(1)
    }
    fmt.Print(string(out))
}
```

Write a Go program that generates the first 10 Fibonacci numbers and prints their sum and product.

```
package main

import "fmt"

func main() {

    fib := make([]int, 10)

    fib[0], fib[1] = 0, 1

    for i := 2; i < 10; i++ {

        fib[i] = fib[i-1] + fib[i-2]

    }

    sum := 0

    product := 1

    for _, num := range fib {

        sum += num

        product *= num

    }

    fmt.Printf("First 10 Fibonacci numbers: %v\n", fib)

    fmt.Printf("Sum: %d\n", sum)

    fmt.Printf("Product: %d\n", product)

}

/home/igor > === Generated Go code ===

package main

import "fmt"

func main() {

    fib := make([]int, 10)

    fib[0], fib[1] = 0, 1

    for i := 2; i < 10; i++ {
```

~~~
        fib[i] = fib[i-1] + fib[i-2]
    }
    sum := 0
    product := 1
    for _, num := range fib {
        sum += num
        product *= num
    }

    fmt.Printf("First 10 Fibonacci numbers: %v\n", fib)
    fmt.Printf("Sum: %d\n", sum)
    fmt.Printf("Product: %d\n", product)
}
=== Executing ===
First 10 Fibonacci numbers: [0 1 1 2 3 5 8 13 21 34]
Sum: 88
Product: 0
--------------------------------------------------------------------------------
/home/igor > Sent to mshell (1818 bytes)
Received from GUI editor:
----------------------------------------
~~~

Pattern 5 Two-LLM Review Chain

~~~
```go >task
package main
import "fmt"
func main() {
~~~

````
    fmt.Println("Write a Go function that implements binary search on a sorted slice of integers. Include a main function that tests it with a sample slice.")

}

```

<!--@1 <task >code

Write only Go code, no markdown fences, no explanation. Start with: package main

-->


```go <code

package main

import (

    "fmt"

    "os"

)

func main() {

    codePath := os.Getenv("MSH_VAR_code")

    codeBytes, _ := os.ReadFile(codePath)

    fmt.Println("=== Model 1 generated ===")

    fmt.Println(string(codeBytes))

}

```

<!--@2 <code >review

Review this Go code. Check correctness, edge cases (empty slice, element not found,

duplicates), and idiomatic Go style. Reply in 3 sentences maximum.

-->

```go <review

package main
````

````
import (
    "fmt"
    "os"
)
func main() {
    review, _ := os.ReadFile(os.Getenv("MSH_VAR_review"))
    fmt.Println("=== Model 2 review ===")
    fmt.Println(string(review))
}
```
<!--@1 <code <review >improved
Improve the Go code based on the review. Return only the improved Go code, no fences,
no explanation. Start with: package main
-->
```go <improved
package main
import (
    "fmt"
    "os"
    "os/exec"
)
func main() {
    improvedPath := os.Getenv("MSH_VAR_improved")
    codeBytes, _ := os.ReadFile(improvedPath)
    fmt.Println("=== Final improved code ===")
    fmt.Println(string(codeBytes))
    fmt.Println("=== Executing ===")
````

```
    tmpFile, _ := os.CreateTemp("", "msh_improved_*.go")
    defer os.Remove(tmpFile.Name())
    tmpFile.Write(codeBytes)
    tmpFile.Close()

    out, err := exec.Command("go", "run", tmpFile.Name()).CombinedOutput()
    if err != nil {
        fmt.Fprintf(os.Stderr, "error: %v\n%s\n", err, out)
        os.Exit(1)
    }
    fmt.Print(string(out))
}
```

Write a Go function that implements binary search on a sorted slice of integers. Include a main function that tests it with a sample slice.

```
package main

import "fmt"

func binarySearch(arr []int, target int) int {
    left, right := 0, len(arr)-1
    for left <= right {
        mid := left + (right-left)/2
        if arr[mid] == target {
            return mid
        } else if arr[mid] < target {
            left = mid + 1
        } else {
```

```
      right = mid - 1
    }
  }

  return -1
}
func main() {
  arr := []int{2, 5, 8, 12, 16, 23, 38, 45, 67, 78}
  tests := []int{23, 5, 67, 99, 2, 78, 15}
  fmt.Println("Array:", arr)
  fmt.Println("Search results:")
  for _, target := range tests {
    index := binarySearch(arr, target)
    if index != -1 {
      fmt.Printf("Found %d at index %d\n", target, index)
    } else {
      fmt.Printf("%d not found\n", target)
    }
  }
}
=== Model 1 generated ===
package main

import "fmt"
func binarySearch(arr []int, target int) int {
  left, right := 0, len(arr)-1
  for left <= right {
```

```
        mid := left + (right-left)/2


        if arr[mid] == target {
            return mid
        } else if arr[mid] < target {
            left = mid + 1
        } else {
            right = mid - 1
        }
    }
    return -1
}
func main() {
    arr := []int{2, 5, 8, 12, 16, 23, 38, 45, 67, 78}
    tests := []int{23, 5, 67, 99, 2, 78, 15}
    fmt.Println("Array:", arr)
    fmt.Println("Search results:")
    for _, target := range tests {
        index := binarySearch(arr, target)
        if index != -1 {
            fmt.Printf("Found %d at index %d\n", target, index)
        } else {
            fmt.Printf("%d not found\n", target)
        }
    }
}
```

The binarySearch implementation is correct for sorted integer slices: it uses the standard left/right/mid pattern and correctly returns -1 when the element is not found, and it also handles the empty-slice case because right starts at len(arr)-1, so the loop is skipped and -1 is returned. For duplicates, it will find one occurrence but does not guarantee the first or last index, which is acceptable unless a specific duplicate-handling policy is required. The code is idiomatic Go in structure and style (clear naming, use of range in the loop, fmt for output), though you could optionally add a brief comment on the function and consider returning (int, bool) instead of using -1 as a sentinel in more robust APIs.

=== Model 2 review ===

The binarySearch implementation is correct for sorted integer slices: it uses the standard left/right/mid pattern and correctly returns -1 when the element is not found, and it also handles the empty-slice case because right starts at len(arr)-1, so the loop is skipped and -1 is returned. For duplicates, it will find one occurrence but does not guarantee the first or last index, which is acceptable unless a specific duplicate-handling policy is required. The code is idiomatic Go in structure and style (clear naming, use of range in the loop, fmt for output), though you could optionally add a brief comment on the function and consider returning (int, bool) instead of using -1 as a sentinel in more robust APIs.

```
package main

import "fmt"

// binarySearch performs binary search on a sorted slice and returns the index and whether the element was found

func binarySearch(arr []int, target int) (int, bool) {

  left, right := 0, len(arr)-1

  for left <= right {

    mid := left + (right-left)/2

    if arr[mid] == target {

      return mid, true

    } else if arr[mid] < target {

      left = mid + 1

    } else {

      right = mid - 1

    }

  }
```

```
    return -1, false
}
func main() {
    arr := []int{2, 5, 8, 12, 16, 23, 38, 45, 67, 78}
    tests := []int{23, 5, 67, 99, 2, 78, 15}
    fmt.Println("Array:", arr)
    fmt.Println("Search results:")
    for _, target := range tests {
        index, found := binarySearch(arr, target)
        if found {
            fmt.Printf("Found %d at index %d\n", target, index)
        } else {
            fmt.Printf("%d not found\n", target)
        }
    }
}
/home/igor > === Final improved code ===
package main
import "fmt"
// binarySearch performs binary search on a sorted slice and returns the index and
whether the element was found
func binarySearch(arr []int, target int) (int, bool) {
    left, right := 0, len(arr)-1
    for left <= right {
        mid := left + (right-left)/2

        if arr[mid] == target {
```

```
            return mid, true
        } else if arr[mid] < target {
            left = mid + 1
        } else {
            right = mid - 1
        }
    }
    return -1, false
}
func main() {
    arr := []int{2, 5, 8, 12, 16, 23, 38, 45, 67, 78}
    tests := []int{23, 5, 67, 99, 2, 78, 15}
    fmt.Println("Array:", arr)
    fmt.Println("Search results:")
    for _, target := range tests {
        index, found := binarySearch(arr, target)
        if found {
            fmt.Printf("Found %d at index %d\n", target, index)
        } else {
            fmt.Printf("%d not found\n", target)
        }
    }
}
=== Executing ===
Array: [2 5 8 12 16 23 38 45 67 78]
Search results:
Found 23 at index 5
```

Found 5 at index 1

Found 67 at index 8

99 not found

Found 2 at index 0

Found 78 at index 9

15 not found

/home/igor > Sent to mshell (1709 bytes)

Received from GUI editor:

Pattern 6 Parallel 3-Model Query

```go >question
package main

import "fmt"

func main() {
   fmt.Println("What is the single most important principle to follow when designing concurrent Go programs? Answer in exactly one sentence.")
}
```

<!--@1 <question >ans1

Answer the question in exactly one sentence.

-->

<!--@2 <question >ans2

Answer the question in exactly one sentence.

-->

<!--@3 <question >ans3

Answer the question in exactly one sentence.

-->

```go <ans1 <ans2 <ans3
package main

import (
    "fmt"
    "os"
    "strings"
)

func wordSet(s string) map[string]bool {
    set := make(map[string]bool)
    for _, w := range strings.Fields(strings.ToLower(s)) {
        // strip punctuation
        w = strings.Trim(w, ".,!?;:")
        if len(w) > 3 {
            set[w] = true
        }
    }
    return set
}

func overlap(a, b map[string]bool) int {
    count := 0
    for w := range a {
        if b[w] {
            count++
        }
```

```
    }
    return count
}
func main() {
    ans1, _ := os.ReadFile(os.Getenv("MSH_VAR_ans1"))
    ans2, _ := os.ReadFile(os.Getenv("MSH_VAR_ans2"))
    ans3, _ := os.ReadFile(os.Getenv("MSH_VAR_ans3"))
    a1 := strings.TrimSpace(string(ans1))
    a2 := strings.TrimSpace(string(ans2))
    a3 := strings.TrimSpace(string(ans3))
    fmt.Println("=== Model 1 ===")
    fmt.Println(a1)
    fmt.Println()
    fmt.Println("=== Model 2 ===")
    fmt.Println(a2)
    fmt.Println()
    fmt.Println("=== Model 3 ===")
    fmt.Println(a3)
    fmt.Println()
    s1, s2, s3 := wordSet(a1), wordSet(a2), wordSet(a3)
    fmt.Printf("Word overlap 1\u21942: %d | 1\u21943: %d | 2\u21943: %d\n",
        overlap(s1, s2), overlap(s1, s3), overlap(s2, s3))
}
```

What is the single most important principle to follow when designing concurrent Go programs? Answer in exactly one sentence.

The single most important principle when designing concurrent Go programs is to follow the Go proverb "Don't communicate by sharing memory; share memory by communicating" by using channels to coordinate goroutines rather than relying on shared variables and locks.

Design concurrent Go programs so that goroutines communicate by passing data over channels rather than sharing mutable memory directly.

Prioritize communication over shared memory by using channels to synchronize and pass data between goroutines, minimizing the need for explicit locking and reducing race conditions.

=== Model 1 ===

The single most important principle when designing concurrent Go programs is to follow the Go proverb "Don't communicate by sharing memory; share memory by communicating" by using channels to coordinate goroutines rather than relying on shared variables and locks.

=== Model 2 ===

Design concurrent Go programs so that goroutines communicate by passing data over channels rather than sharing mutable memory directly.

=== Model 3 ===

Prioritize communication over shared memory by using channels to synchronize and pass data between goroutines, minimizing the need for explicit locking and reducing race conditions.

Word overlap 1↔2: 9 | 1↔3: 5 | 2↔3: 5

/home/igor > Sent to mshell (2091 bytes)

Received from GUI editor:

Pattern 7 Evaluator-Optimizer Loop

```go >task

package main


import "fmt"

func main() {
```

    fmt.Println("Write a Go function isPalindrome(s string) bool that returns true if the string is a palindrome (ignoring case and non-alphanumeric characters). Include a main function that tests it with at least 3 examples.")

}

```

```go >verdict

package main

import "fmt"

func main() {

    fmt.Println("REJECTED: no code yet")

}

```

<!--@loop max=3 until=verdict:ACCEPTED-->

<!--@1 <task <verdict >code

The first input is a coding task. The second input is the evaluator's previous verdict

(or the initial state). If the verdict starts with REJECTED, fix the issue described.

Return only Go code, no fences, no explanation. Start with: package main

-->

```go <code

package main

import (

    "fmt"

    "os"

)


func main() {

    codePath := os.Getenv("MSH_VAR_code")

    codeBytes, _ := os.ReadFile(codePath)

```
    fmt.Println("=== Generated code ===")
    fmt.Println(string(codeBytes))
}
```

<!--@2 <code >verdict

Review this Go code strictly.

If it is completely correct and handles all edge cases, reply with exactly: ACCEPTED

If there is any bug or missing case, reply with: REJECTED: <one sentence describing the fix needed>

-->

```go <verdict
package main
import (
    "fmt"
    "os"
)
func main() {
    verdict, _ := os.ReadFile(os.Getenv("MSH_VAR_verdict"))
    fmt.Println("=== Verdict ===")
    fmt.Println(string(verdict))
}
```

<!--@end_loop-->

```go <code
package main
import (
```

```
    "fmt"

    "os"

    "os/exec"

)

func main() {

    codePath := os.Getenv("MSH_VAR_code")

    codeBytes, _ := os.ReadFile(codePath)

    fmt.Println("=== Final accepted code ===")

    fmt.Println(string(codeBytes))

    fmt.Println("=== Executing ===")

    tmp, _ := os.CreateTemp("", "msh_accepted_*.go")

    defer os.Remove(tmp.Name())

    tmp.Write(codeBytes)

    tmp.Close()

    out, err := exec.Command("go", "run", tmp.Name()).CombinedOutput()

    if err != nil {

        fmt.Fprintf(os.Stderr, "error: %v\n%s\n", err, out)

        os.Exit(1)

    }

    fmt.Print(string(out))

}
```

Write a Go function isPalindrome(s string) bool that returns true if the string is a palindrome (ignoring case and non-alphanumeric characters). Include a main function that tests it with at least 3 examples.

REJECTED: no code yet

[loop] Starting loop: max=3 until=verdict:ACCEPTED

```
package main
import (
   "fmt"
   "strings"
   "unicode"
)
func isPalindrome(s string) bool {
   // Convert to lowercase and keep only alphanumeric characters
   var cleaned strings.Builder
   for _, r := range strings.ToLower(s) {
      if unicode.IsLetter(r) || unicode.IsDigit(r) {
         cleaned.WriteRune(r)
      }
   }
   str := cleaned.String()
   left, right := 0, len(str)-1
   for left < right {
      if str[left] != str[right] {
         return false
      }
      left++
      right--
   }
   return true
}
func main() {
   tests := []string{
```

```
        "A man, a plan, a canal: Panama",
        "race a car",
        "Was it a car or a cat I saw?",
        "Madam",
        "hello",
    }
    for _, test := range tests {
        result := isPalindrome(test)
        fmt.Printf("'%s' is palindrome: %t\n", test, result)
    }
}
=== Generated code ===
package main
import (
    "fmt"
    "strings"
    "unicode"
)

func isPalindrome(s string) bool {
    // Convert to lowercase and keep only alphanumeric characters
    var cleaned strings.Builder
    for _, r := range strings.ToLower(s) {
        if unicode.IsLetter(r) || unicode.IsDigit(r) {
            cleaned.WriteRune(r)
        }
    }
```

```
    str := cleaned.String()
    left, right := 0, len(str)-1
    for left < right {
        if str[left] != str[right] {
            return false
        }
        left++
        right--
    }
    return true
}
func main() {
    tests := []string{
        "A man, a plan, a canal: Panama",
        "race a car",
        "Was it a car or a cat I saw?",
        "Madam",
        "hello",
    }
    for _, test := range tests {
        result := isPalindrome(test)
        fmt.Printf("'%s' is palindrome: %t\n", test, result)
    }
}
```

REJECTED: Comparing bytes of a UTF-8 string risks breaking for multi-byte runes; convert `str` to a rune slice and index that for correct palindrome checking with Unicode.

[loop] until check: last_line='REJECTED: Comparing bytes of a UTF-8 string risks breaking for multi-byte runes; convert `str` to a rune slice and index that for correct palindrome checking with Unicode.' expected='ACCEPTED'

[loop] Iteration 1/3 \u2014 condition not met, looping back

=== Verdict ===

REJECTED: Comparing bytes of a UTF-8 string risks breaking for multi-byte runes; convert `str` to a rune slice and index that for correct palindrome checking with Unicode.

```
package main

import (

    "fmt"

    "strings"

    "unicode"

)

func isPalindrome(s string) bool {

    // Convert to lowercase and keep only alphanumeric characters

    var cleaned []rune

    for _, r := range strings.ToLower(s) {

        if unicode.IsLetter(r) || unicode.IsDigit(r) {

            cleaned = append(cleaned, r)

        }

    }

    left, right := 0, len(cleaned)-1

    for left < right {

        if cleaned[left] != cleaned[right] {

            return false

        }

        left++

        right--
```

```
    }
    return true
}
func main() {
    tests := []string{
        "A man, a plan, a canal: Panama",
        "race a car",
        "Was it a car or a cat I saw?",
        "Madam",
        "hello",
    }
    for _, test := range tests {
        result := isPalindrome(test)
        fmt.Printf("'%s' is palindrome: %t\n", test, result)
    }
}
=== Generated code ===
package main
import (
    "fmt"
    "strings"
    "unicode"
)
func isPalindrome(s string) bool {
    // Convert to lowercase and keep only alphanumeric characters
    var cleaned []rune
    for _, r := range strings.ToLower(s) {
```

```
        if unicode.IsLetter(r) || unicode.IsDigit(r) {
            cleaned = append(cleaned, r)
        }
    }
    left, right := 0, len(cleaned)-1
    for left < right {
        if cleaned[left] != cleaned[right] {
            return false
        }
        left++
        right--
    }

    return true
}
func main() {
    tests := []string{
        "A man, a plan, a canal: Panama",
        "race a car",
        "Was it a car or a cat I saw?",
        "Madam",
        "hello",
    }
    for _, test := range tests {
        result := isPalindrome(test)
        fmt.Printf("'%s' is palindrome: %t\n", test, result)
    }
```

```
}
ACCEPTED
[loop] Exiting loop after 2 iteration(s). reason: until condition met
/home/igor > === Final accepted code ===
package main
import (
    "fmt"
    "strings"
    "unicode"
)

func isPalindrome(s string) bool {
    // Convert to lowercase and keep only alphanumeric characters
    var cleaned []rune
    for _, r := range strings.ToLower(s) {
        if unicode.IsLetter(r) || unicode.IsDigit(r) {
            cleaned = append(cleaned, r)
        }
    }
    left, right := 0, len(cleaned)-1
    for left < right {
        if cleaned[left] != cleaned[right] {
            return false
        }
        left++
        right--
    }
```

```
    return true
}
func main() {
    tests := []string{
        "A man, a plan, a canal: Panama",
        "race a car",
        "Was it a car or a cat I saw?",
        "Madam",
        "hello",
    }
    for _, test := range tests {
        result := isPalindrome(test)
        fmt.Printf("'%s' is palindrome: %t\n", test, result)
    }
}
=== Executing ===
=== Final accepted code ===
package main
import (
    "fmt"
    "strings"
    "unicode"
)
func isPalindrome(s string) bool {
    // Convert to lowercase and keep only alphanumeric characters
    var cleaned []rune
    for _, r := range strings.ToLower(s) {
```

```
        if unicode.IsLetter(r) || unicode.IsDigit(r) {
            cleaned = append(cleaned, r)
        }
    }
    left, right := 0, len(cleaned)-1

    for left < right {
        if cleaned[left] != cleaned[right] {
            return false
        }
        left++
        right--
    }
    return true
}
func main() {
    tests := []string{
        "A man, a plan, a canal: Panama",
        "race a car",
        "Was it a car or a cat I saw?",
        "Madam",
        "hello",
    }
    for _, test := range tests {
        result := isPalindrome(test)
        fmt.Printf("'%s' is palindrome: %t\n", test, result)
    }
```

}

=== Executing ===

'A man, a plan, a canal: Panama' is palindrome: true

'race a car' is palindrome: false

'Was it a car or a cat I saw?' is palindrome: true

'Madam' is palindrome: true

'hello' is palindrome: false

'A man, a plan, a canal: Panama' is palindrome: true

'race a car' is palindrome: false

'Was it a car or a cat I saw?' is palindrome: true

'Madam' is palindrome: true

'hello' is palindrome: false

/home/igor > Sent to mshell (3068 bytes)

Received from GUI editor:

Pattern 8 Multi-Stage Go + Multi-Model Pipeline

```go >raw_data
package main
import (
   "fmt"
   "math/rand"
)
func main() {
   r := rand.New(rand.NewSource(42))
   for i := 0; i < 10; i++ {
      temp := 18.0 + r.Float64()*15.0
```

````
        if i > 0 {
            fmt.Print(",")
        }
        fmt.Printf("%.2f", temp)
    }
    fmt.Println()
}
```
```go <raw_data >stats
package main
import (
    "fmt"
    "math"
    "os"
    "strconv"
    "strings"
)
func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_data"))
    parts := strings.Split(strings.TrimSpace(string(raw)), ",")
    vals := make([]float64, 0, len(parts))
    for _, p := range parts {
        v, err := strconv.ParseFloat(strings.TrimSpace(p), 64)
        if err == nil {
            vals = append(vals, v)
        }
    }
````

```
    n := float64(len(vals))
    sum := 0.0
    mn, mx := vals[0], vals[0]
    for _, v := range vals {
        sum += v
        if v < mn { mn = v }
        if v > mx { mx = v }
    }
    mean := sum / n
    variance := 0.0
    for _, v := range vals {
        diff := v - mean
        variance += diff * diff
    }
    stddev := math.Sqrt(variance / n)
    fmt.Printf("count=%d mean=%.2f stddev=%.2f min=%.2f max=%.2f\n",
        len(vals), mean, stddev, mn, mx)
}
```

<!--@1 <stats >analysis

The input contains statistics for 10 temperature sensor readings (°C).

You MUST respond with at least one sentence \u2014 do not leave the response empty.

In one sentence, describe what these statistics suggest about the monitored environment.

-->

<!--@2 <analysis >alert

The input is a one-sentence sensor analysis.

You MUST respond \u2014 do not leave the response empty.

Compress it into a 6-word alert label suitable for a monitoring dashboard.

Reply with ONLY the label, nothing else.

-->

```go <raw_data <stats <analysis <alert
package main

import (
    "fmt"
    "os"
    "strings"
)

func readVar(name string) string {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
    s := strings.TrimSpace(string(data))
    if s == "" {
        return "[no data]"
    }
    return s
}

func main() {

fmt.Println("\u2554\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2557")
    fmt.Println("\u2551   SENSOR MONITORING REPORT   \u2551")

fmt.Println("\u2560\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2563")
```

```
    fmt.Printf("\u2551 Readings : %s\n", readVar("raw_data"))

    fmt.Printf("\u2551 Stats    : %s\n", readVar("stats"))

    fmt.Printf("\u2551 Analysis : %s\n", readVar("analysis"))

    fmt.Printf("\u2551 Alert    : %s\n", readVar("alert"))

fmt.Println("\u255a\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255d")

}
```

23.60,18.99,27.06,21.13,18.66,23.75,30.19,23.77,23.75,27.70

count=10 mean=23.86 stddev=3.51 min=18.66 max=30.19

The temperature sensor readings suggest a stable, well-controlled indoor environment with a mean of 23.86°C (near room temperature) and relatively low variability (standard deviation of 3.51°C), indicating consistent climate control with temperatures ranging comfortably between 18.66°C and 30.19°C.

Stable indoor temperature, low variance detected

\u2554\u2550\u2557

\u2551   SENSOR MONITORING REPORT   \u2551

\u2560\u2550\u2563

\u2551 Readings : 23.60,18.99,27.06,21.13,18.66,23.75,30.19,23.77,23.75,27.70

\u2551 Stats   : count=10 mean=23.86 stddev=3.51 min=18.66 max=30.19

\u2551 Analysis : The temperature sensor readings suggest a stable, well-controlled indoor environment with a mean of 23.86°C (near room temperature) and relatively low variability (standard deviation of 3.51°C), indicating consistent climate control with temperatures ranging comfortably between 18.66°C and 30.19°C.

\u2551 Alert   : Stable indoor temperature, low variance detected

\u255a\u2550\u255d

/home/igor > Sent to mshell (2926 bytes)

Received from GUI editor:

Pattern 9 Routing: LLM Classifies - Conditional Go Branch

```go >input1
package main

import "fmt"

func main() { fmt.Println("sort this list of numbers: 7 2 9 1 5 3") }
```

<!--@1 <input1 >route1

Classify this request into exactly one word: SORT, MATH, or TEXT.

Reply with only that single word, nothing else.

-->

```go <route1 if=route1:SORT
package main

import (
   "fmt"
   "sort"
)


func main() {
   nums := []int{7, 2, 9, 1, 5, 3}
   sort.Ints(nums)
   fmt.Printf("=== SORT branch === result: %v\n", nums)
```

````
}
```
```go <route1 if=route1:MATH
package main
import "fmt"
func main() {
    fmt.Println("=== MATH branch === (unexpected for this input)")
}
```
<!--@1 <input1 if=route1:TEXT
Answer this question directly in plain language, no code.
-->
```go <route1
package main
import (
    "fmt"
    "os"
    "strings"
)

func main() {
    r, _ := os.ReadFile(os.Getenv("MSH_VAR_route1"))
    fmt.Printf("Test1 classified as: %s\n\n", strings.TrimSpace(string(r)))
}
```
```go >input2
package main
````

import "fmt"

func main() { fmt.Println("what is the prime factorization of 84") }

```

&lt;!--@1 &lt;input2 &gt;route2

Classify this request into exactly one word: SORT, MATH, or TEXT.

Reply with only that single word, nothing else.

--&gt;

```go &lt;route2 if=route2:SORT

package main

import "fmt"

func main() { fmt.Println("=== SORT branch === (unexpected)") }

```

```go <route2 if=route2:MATH

package main

import "fmt"

func primeFactors(n int) []int {

  factors := []int{}

  for d := 2; d*d <= n; d++ {

    for n%d == 0 {

      factors = append(factors, d)

      n /= d

    }

  }

  if n > 1 {

    factors = append(factors, n)

  }

  return factors

````
}
func main() {
    fmt.Printf("=== MATH branch === prime factors of 84: %v\n", primeFactors(84))
}
```
<!--@1 <input2 if=route2:TEXT
Answer this question directly in plain language, no code.
-->
```go <route2
package main
import (
    "fmt"
    "os"
    "strings"
)

func main() {
    r, _ := os.ReadFile(os.Getenv("MSH_VAR_route2"))
    fmt.Printf("Test2 classified as: %s\n\n", strings.TrimSpace(string(r)))
}
```
```go >input3
package main
import "fmt"
func main() { fmt.Println("explain what a goroutine is") }
```
<!--@1 <input3 >route3
````
````

~~~~
Classify this request into exactly one word: SORT, MATH, or TEXT.

Reply with only that single word, nothing else.

-->

```go <route3 if=route3:SORT

package main

import "fmt"

func main() { fmt.Println("=== SORT branch === (unexpected)") }

```

```go <route3 if=route3:MATH

package main

import "fmt"

func main() { fmt.Println("=== MATH branch === (unexpected)") }

```


<!--@1 <input3 if=route3:TEXT

Answer this question directly in plain English. No code.

-->

```go <route3

package main

import (

    "fmt"

    "os"

    "strings"

)

func main() {

    r, _ := os.ReadFile(os.Getenv("MSH_VAR_route3"))

    fmt.Printf("Test3 classified as: %s\n\n", strings.TrimSpace(string(r)))
~~~~

}

```

---------------------------------------

sort this list of numbers: 7 2 9 1 5 3

SORT

what is the prime factorization of 84

what is the prime factorization of 84

what is the prime factorization of 84

MATH

explain what a goroutine is

explain what a goroutine is

explain what a goroutine is

TEXT

A goroutine is Go's lightweight thread that allows functions to run concurrently with other goroutines within the same program. Unlike traditional operating system threads, goroutines are managed by Go's runtime and are extremely cheap to create - you can have thousands or even millions of them running simultaneously. They start with a very small stack (just a few kilobytes) that grows and shrinks as needed, making them much more memory-efficient than regular threads. Goroutines are created by simply putting the word "go" before a function call, and they communicate with each other through channels rather than shared memory, which helps prevent common concurrency bugs like race conditions.

/home/igor > Test3 classified as: TEXT

--------------------------------------------------------------------------------------

/home/igor > Sent to mshell (2383 bytes)

Received from GUI editor:

---------------------------------------

# Pattern 10  Full Pipeline: All Patterns Combined

```go >raw_data

package main

import (

````
    "fmt"
    "strings"
)
func main() {
    primes := []string{}
    for n := 2; len(primes) < 10; n++ {
        isPrime := true
        for d := 2; d*d <= n; d++ {
            if n%d == 0 {
                isPrime = false
                break
            }
        }
        if isPrime {
            primes = append(primes, fmt.Sprintf("%d", n))
        }
    }
    fmt.Println(strings.Join(primes, " "))
}
```
```go <raw_data >stats
package main
import (
    "fmt"
    "os"
    "strconv"
    "strings"
````

```
)

func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_data"))
    parts := strings.Fields(strings.TrimSpace(string(raw)))
    nums := make([]int, 0, len(parts))
    for _, p := range parts {
        n, _ := strconv.Atoi(p)
        nums = append(nums, n)
    }
    sum := 0
    maxGap := 0
    for i, n := range nums {
        sum += n
        if i > 0 && nums[i]-nums[i-1] > maxGap {
            maxGap = nums[i] - nums[i-1]
        }
    }
    mean := float64(sum) / float64(len(nums))
    fmt.Printf("First 10 primes: %v\nSum=%d Mean=%.1f LargestGap=%d\n",
        nums, sum, mean, maxGap)
}
```

<!--@1 <stats >analysis

In one sentence, describe what is mathematically interesting about these prime number statistics.

-->

<!--@2 <raw_data >poem

Write a 2-line rhyming poem about prime numbers. Use the actual numbers from the input.

-->

```go <analysis <poem
package main

import (
   "fmt"
   "os"
   "strings"
)

func main() {
   analysis, _ := os.ReadFile(os.Getenv("MSH_VAR_analysis"))
   poem, _ := os.ReadFile(os.Getenv("MSH_VAR_poem"))
   fmt.Println("=== Analysis ===")
   fmt.Println(strings.TrimSpace(string(analysis)))
   fmt.Println()
   fmt.Println("=== Poem ===")
   fmt.Println(strings.TrimSpace(string(poem)))
}
```

<!--@1 <analysis <poem >combined

Combine the mathematical analysis and the poetic perspective into one single elegant sentence.

-->

```go <combined
package main

import (
   "fmt"
```

```
    "os"

    "strings"

)

func main() {

    data, _ := os.ReadFile(os.Getenv("MSH_VAR_combined"))

    text := strings.TrimSpace(string(data))

    border := strings.Repeat("\u2550", len([]rune(text))+4)

    fmt.Printf("\u2554%s\u2557\n", border)

    fmt.Printf("\u2551  %s  \u2551\n", text)

    fmt.Printf("\u255a%s\u255d\n", border)

}
```

----

2 3 5 7 11 13 17 19 23 29

First 10 primes: [2 3 5 7 11 13 17 19 23 29]

Sum=129 Mean=12.9 LargestGap=6

The largest gap of 6 (between 23 and 29) represents the first occurrence of a composite run of this length among the initial primes, while the mean of 12.9 illustrates how prime density decreases as numbers grow larger, with gaps becoming more irregular and substantial compared to the early consecutive primes like 2, 3, 5, 7.

2, 3, 5, 7, 11 in line, they glitter like stars in a numeric design,

13, 17, 19, 23, 29, prime-hearted beats in a perfect rhyme.

=== Analysis ===

The largest gap of 6 (between 23 and 29) represents the first occurrence of a composite run of this length among the initial primes, while the mean of 12.9 illustrates how prime density decreases as numbers grow larger, with gaps becoming more irregular and substantial compared to the early consecutive primes like 2, 3, 5, 7.

=== Poem ===

2, 3, 5, 7, 11 in line, they glitter like stars in a numeric design,

13, 17, 19, 23, 29, prime-hearted beats in a perfect rhyme.

The first ten primes\u20142, 3, 5, 7, 11, 13, 17, 19, 23, 29\u2014glitter like stars in perfect numeric design, their prime-hearted beats creating an elegant rhythm that reveals how density decreases and gaps widen as we ascend the infinite ladder, with the largest gap of 6 between 23 and 29 marking the first substantial pause in this celestial mathematical rhyme.

\u2554\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u2557

\u2551 The first ten primes\u20142, 3, 5, 7, 11, 13, 17, 19, 23, 29\u2014glitter like stars in perfect numeric design, their prime-hearted beats creating an elegant rhythm that reveals how density decreases and gaps widen as we ascend the infinite ladder, with the largest gap of 6 between 23 and 29 marking the first substantial pause in this celestial mathematical rhyme. \u2551

\u255a\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255
0\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u25
50\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2
550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u
2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\
u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550
\u2550\u2550\u2550\u2550\u2550\u2550\u255d

/home/igor > Sent to mshell (1519 bytes)

Received from GUI editor:

Pattern 11 MShell Node with Multiple Models

```go >topic

package main

import "fmt"
```

~~~~
func main() {
   fmt.Println("Go channels and goroutines")
}
```
```go >format
package main


import "fmt"
func main() {
   fmt.Println("three bullet points, each one sentence")
}
```
```mshell <topic <format >explanation
ollama1 "Explain $topic in the following format: $format. Be concise and accurate."
```
```mshell <explanation >keywords
ollama2 "Extract exactly 4 technical keywords from this text: $explanation. Reply with only the 4 words, comma-separated, nothing else."
```
```go <explanation <keywords
package main
import (
   "fmt"
   "os"
   "strings"
)
func avgWordLen(text string) float64 {
~~~~

```go
    words := strings.Fields(text)
    if len(words) == 0 {
        return 0
    }
    total := 0
    for _, w := range words {
        total += len(strings.Trim(w, ".,!?;:"))
    }
    return float64(total) / float64(len(words))
}
func main() {
    explanation, _ := os.ReadFile(os.Getenv("MSH_VAR_explanation"))
    keywords, _ := os.ReadFile(os.Getenv("MSH_VAR_keywords"))
    expl := strings.TrimSpace(string(explanation))
    kw := strings.TrimSpace(string(keywords))
    wordCount := len(strings.Fields(expl))
    avgLen := avgWordLen(expl)
    fmt.Println("=== Explanation ===")
    fmt.Println(expl)
    fmt.Println()
    fmt.Printf("=== Metrics ===\nWord count: %d | Avg word length: %.1f chars\n", wordCount, avgLen)
    fmt.Println()
    fmt.Println("=== Keywords ===")
    fmt.Println(kw)
}
```

Go channels and goroutines

three bullet points, each one sentence

Goroutines are lightweight, user-space threads managed by Go's runtime that allow functions to run concurrently with minimal memory overhead and can be created by simply prefixing a function call with the `go` keyword.

Channels are typed conduits that enable safe communication between goroutines by allowing them to send and receive values, eliminating the need for explicit locks or shared memory access.

Together, goroutines and channels implement Go's concurrency philosophy of "don't communicate by sharing memory; share memory by communicating," providing a safer and more intuitive approach to concurrent programming than traditional threading models.goroutines,channels,concurrency,runtime/home/igor > === Explanation ===

Goroutines are lightweight, user-space threads managed by Go's runtime that allow functions to run concurrently with minimal memory overhead and can be created by simply prefixing a function call with the `go` keyword.

Channels are typed conduits that enable safe communication between goroutines by allowing them to send and receive values, eliminating the need for explicit locks or shared memory access.

Together, goroutines and channels implement Go's concurrency philosophy of "don't communicate by sharing memory; share memory by communicating," providing a safer and more intuitive approach to concurrent programming than traditional threading models.

=== Metrics ===

Word count: 96 | Avg word length: 5.9 chars

=== Keywords ===

goroutines,channels,concurrency,runtime

/home/igor > Sent to mshell (1520 bytes)

Received from GUI editor:

Pattern 12 Async Parallel 3 Models + Await Barrier + Synthesis

```go >question
```

````
package main

import "fmt"


func main() {

    fmt.Println("What is the Go memory model and why does it matter for concurrent programming?")

}
```

<!--@1 <question >ans1 async

Explain the answer in one sentence for a complete beginner with no prior concurrency knowledge.

-->

<!--@2 <question >ans2 async

Explain the answer in one sentence using a real-world everyday analogy.

-->

<!--@3 <question >ans3 async

Explain the answer in one sentence using precise technical terminology.

-->

```bash await=ans1,ans2,ans3

```

<!--@1 <ans1 <ans2 <ans3 >final

Synthesize these three explanations (beginner, analogy, technical) into one perfect sentence

that remains accessible to all audiences while being technically accurate.

-->

```go <ans1 <ans2 <ans3 <final

package main

import (
````

```
    "fmt"
    "os"
    "strings"
)
func readVar(name string) string {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
    return strings.TrimSpace(string(data))
}
func wordCount(s string) int {
    return len(strings.Fields(s))
}
func main() {
    a1 := readVar("ans1")
    a2 := readVar("ans2")
    a3 := readVar("ans3")
    final := readVar("final")
    fmt.Printf("=== Beginner (%d words) ===\n%s\n\n", wordCount(a1), a1)
    fmt.Printf("=== Analogy (%d words) ===\n%s\n\n", wordCount(a2), a2)
    fmt.Printf("=== Technical (%d words) ===\n%s\n\n", wordCount(a3), a3)
    fmt.Println("=== Synthesized ===")
    fmt.Println(final)
}
```

What is the Go memory model and why does it matter for concurrent programming?

[async llm] Launched PID 33311 \u2192 var=ans1 (model @1)

[async llm] Launched PID 33312 \u2192 var=ans2 (model @2)

[async llm] Launched PID 33314 \u2192 var=ans3 (model @3)

[async] await= barrier: waiting for vars: ans1,ans2,ans3

[async] Waiting for PID 33311 (var=ans1)...

[async] PID 33311 done (var=ans1)

The Go memory model is a set of rules that guarantees when changes made by one goroutine (lightweight thread) will be visible to other goroutines, which matters because without these guarantees, concurrent programs could see inconsistent or outdated data, leading to unpredictable bugs.

The Go memory model defines the traffic rules for when goroutines can safely see each other's data changes, and it matters because following these rules\u2014primarily by using channels to communicate and synchronize rather than sharing memory directly\u2014prevents data races and ensures your concurrent program behaves predictably without subtle bugs.

/home/igor > === Beginner (43 words) ===

The Go memory model is a set of rules that guarantees when changes made by one goroutine (lightweight thread) will be visible to other goroutines, which matters because without these guarantees, concurrent programs could see inconsistent or outdated data, leading to unpredictable bugs.

=== Analogy (51 words) ===

The Go memory model is like the traffic rules for how and when drivers (goroutines) can safely see each other\u2019s lane changes (shared data updates), and it matters because following these rules with signals and intersections (synchronization like channels and mutexes) prevents crashes (data races and subtle bugs) in concurrent programs.

=== Technical (25 words) ===

Prioritize communication over shared memory by using channels to synchronize and pass data between goroutines, minimizing the need for explicit locking and reducing race conditions.

=== Synthesized ===

The Go memory model defines the traffic rules for when goroutines can safely see each other's data changes, and it matters because following these rules\u2014primarily by using channels to communicate and synchronize rather than sharing memory directly\u2014prevents data races and ensures your concurrent program behaves predictably without subtle bugs.

/home/igor > Sent to mshell (1631 bytes)

Received from GUI editor:

Pattern 13 WHILE Loop: Iterative Counter with LLM Commentary

```go &gt;status

package main

import "fmt"

func main() {

    fmt.Println("running")

}

```

```go &gt;counter

package main

import "fmt"

func main() {

    fmt.Println("0")

}

```

<!--@while status:running-->

```go <counter <status >counter >status

package main

import (

    "fmt"

    "os"

    "strconv"

    "strings"

)

func main() {
```

~~~~
    counterPath := os.Getenv("MSH_VAR_counter")
    statusPath := os.Getenv("MSH_VAR_status")
    raw, _ := os.ReadFile(counterPath)
    val, _ := strconv.Atoi(strings.TrimSpace(string(raw)))
    val++
    os.WriteFile(counterPath, []byte(fmt.Sprintf("%d", val)), 0644)
    if val >= 4 {
        os.WriteFile(statusPath, []byte("done"), 0644)
    } else {
        os.WriteFile(statusPath, []byte("running"), 0644)
    }
    fmt.Printf("%d", val)
}
```
<!--@1 <counter >comment
The input is a single integer. State one interesting or surprising mathematical fact about
this specific number. One sentence only.
-->
```go <comment
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    counter, _ := os.ReadFile(os.Getenv("MSH_VAR_counter"))
~~~~

```
    comment, _ := os.ReadFile(os.Getenv("MSH_VAR_comment"))
    fmt.Printf("[iter %s] %s\n",
        strings.TrimSpace(string(counter)),
        strings.TrimSpace(string(comment)))
}
```

<!--@end_while-->

```go <counter
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    counter, _ := os.ReadFile(os.Getenv("MSH_VAR_counter"))
    fmt.Printf("=== WHILE done. Final counter = %s ===\n",
        strings.TrimSpace(string(counter)))
}
```

running

0

[while] iteration 1 condition met, executing body

1The number 1 is the multiplicative identity and the only positive integer that is neither prime nor composite, making it unique in the classification of natural numbers.

[iter 1] The number 1 is the multiplicative identity and the only positive integer that is neither prime nor composite, making it unique in the classification of natural numbers.

[while] iteration 2 condition met, executing body

2The number 2 is the only even prime number, making it unique among all primes since every other even number is divisible by 2 and therefore composite.

[iter 2] The number 2 is the only even prime number, making it unique among all primes since every other even number is divisible by 2 and therefore composite.

[while] iteration 3 condition met, executing body

3The number 3 is the smallest odd prime and the only prime number that is one less than a perfect square ($4 = 2^2$).

[iter 3] The number 3 is the smallest odd prime and the only prime number that is one less than a perfect square ($4 = 2^2$).

[while] iteration 4 condition met, executing body

4The number 4 is the smallest composite number and the only number that equals both 2^2 and 2+2, making it unique as a perfect square that can also be expressed as the sum of two identical primes.

[iter 4] The number 4 is the smallest composite number and the only number that equals both 2^2 and 2+2, making it unique as a perfect square that can also be expressed as the sum of two identical primes.

[while] condition 'status:running' not met, exiting after 4 iter

/home/igor > === WHILE done. Final counter = 4 ===

/home/igor > Sent to mshell (884 bytes)

Received from GUI editor:

Pattern 14 FOREACH: LLM Explains Each Go Package

```go >packages

package main

import "fmt"

func main() {

    pkgs := []string{"fmt", "strings", "sync", "net/http"}

    for _, p := range pkgs {
```

~~~~
        fmt.Println(p)
    }
}
```
<!--@foreach pkg in packages-->
<!--@1 <pkg >description
The input is the name of a Go standard library package. In one sentence, describe its
primary purpose and most common use case.
-->
```go <description
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    pkg, _ := os.ReadFile(os.Getenv("MSH_VAR_pkg"))
    desc, _ := os.ReadFile(os.Getenv("MSH_VAR_description"))
    fmt.Printf("--- %s ---\n%s\n\n",
        strings.TrimSpace(string(pkg)),
        strings.TrimSpace(string(desc)))
}
```
<!--@end_foreach-->
```go
package main
~~~~

```
import "fmt"

func main() {

   fmt.Println("=== All packages described ===")

}
```

----

fmt

strings

sync

net/http

[foreach] iter 1: pkg=fmt

The fmt package provides formatted I/O functions for printing and scanning text, with its most common use case being `fmt.Println()` and `fmt.Printf()` for outputting formatted strings to the console during development and debugging.

--- fmt ---

The fmt package provides formatted I/O functions for printing and scanning text, with its most common use case being `fmt.Println()` and `fmt.Printf()` for outputting formatted strings to the console during development and debugging.

[foreach] iter 2: pkg=strings

The strings package provides functions for manipulating UTF-8 encoded strings, with its most common use cases being `strings.Split()` for parsing text, `strings.Contains()` for searching, and `strings.Join()` for concatenating string slices.

--- strings ---

The strings package provides functions for manipulating UTF-8 encoded strings, with its most common use cases being `strings.Split()` for parsing text, `strings.Contains()` for searching, and `strings.Join()` for concatenating string slices.

[foreach] iter 3: pkg=sync

The sync package provides synchronization primitives for coordinating access to shared resources between goroutines, with its most common use case being `sync.Mutex` for

protecting critical sections and `sync.WaitGroup` for waiting for multiple goroutines to complete.

--- sync ---

The sync package provides synchronization primitives for coordinating access to shared resources between goroutines, with its most common use case being `sync.Mutex` for protecting critical sections and `sync.WaitGroup` for waiting for multiple goroutines to complete.

[foreach] iter 4: pkg=net/http

The net/http package provides HTTP client and server implementations, with its most common use cases being `http.Get()` for making HTTP requests and `http.HandleFunc()` with `http.ListenAndServe()` for building web servers and REST APIs.

--- net/http ---

The net/http package provides HTTP client and server implementations, with its most common use cases being `http.Get()` for making HTTP requests and `http.HandleFunc()` with `http.ListenAndServe()` for building web servers and REST APIs.

/home/igor > === All packages described ===

/home/igor > Sent to mshell (1575 bytes)

Received from GUI editor:

Pattern 15 TRY/CATCH: Safe Go Execution with Error Capture

```go >input
package main

import "fmt"

func main() {

   fmt.Println("2026-03-11T10:00:00Z")

}
```

<!--@try-->

````
```go <input >result
package main
import (
    "fmt"
    "os"
    "strings"
    "time"
)
func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_input"))
    ts := strings.TrimSpace(string(raw))
    // Intentionally wrong layout \u2014 will fail
    wrongLayout := "02-01-2006"
    t, err := time.Parse(wrongLayout, ts)
    if err != nil {
        fmt.Fprintf(os.Stderr, "parse error: %v\n", err)
        os.Exit(1)
    }
    fmt.Println(t)
}
```
<!--@catch >error-->
```go
package main
import "fmt"
func main() {
    fmt.Println("=== Caught error: try_block_failed ===")
````
````

~~~~
    fmt.Println("Timestamp parsing failed. Using safe fallback.")
}
```
<!--@end_try-->
```go <input >safe_result
package main
import (
    "fmt"
    "os"
    "strings"
    "time"
)
func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_input"))
    ts := strings.TrimSpace(string(raw))

    t, err := time.Parse(time.RFC3339, ts)
    if err != nil {
        fmt.Fprintf(os.Stderr, "fallback parse error: %v\n", err)
        os.Exit(1)
    }
    fmt.Printf("year=%d month=%s day=%d hour=%d UTC\n",
        t.Year(), t.Month(), t.Day(), t.Hour())
}
```
```go <safe_result
package main
~~~~

```
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    result, _ := os.ReadFile(os.Getenv("MSH_VAR_safe_result"))
    fmt.Println("=== Safe result ===")
    fmt.Println(strings.TrimSpace(string(result)))
}
```

----------------------------------------

2026-03-11T10:00:00Z

[try] executing try block

parse error: parsing time "2026-03-11T10:00:00Z" as "02-01-2006": cannot parse "26-03-11T10:00:00Z" as "-"

[try] try block failed, executing catch block

=== Caught error: try_block_failed ===

Timestamp parsing failed. Using safe fallback.

year=2026 month=March day=11 hour=10 UTC

/home/igor > === Safe result ===

year=2026 month=March day=11 hour=10 UTC

--------------------------------------------------------------------------------

/home/igor > Sent to mshell (1683 bytes)

Received from GUI editor:

----------------------------------------

# Pattern 16 SPLIT + MERGE: Divide-and-Conquer Analysis

````
```go >dataset
package main
import "fmt"
func main() {
  fmt.Println("12,45,23,67,34,89,15,52,38,71")
  fmt.Println("8,19,44,11,56,23,77,31,48,62")
}
```
<!--@split dataset into 2-->
<!--@1 <dataset_1 >analysis1 async
The input contains comma-separated network latency values in milliseconds for DC-East.
In one sentence, characterize the latency profile: describe average range, consistency,
and any outliers.
-->
<!--@2 <dataset_2 >analysis2 async
The input contains comma-separated network latency values in milliseconds for DC-West.
In one sentence, characterize the latency profile: describe average range, consistency,
and any outliers.
-->
```bash await=analysis1,analysis2
```
<!--@merge-->
<!--@1 <analysis1 <analysis2 >combined
The inputs contain latency analyses for two data centers.
Combine them into a two-sentence comparative summary that recommends which DC has
better performance characteristics.
-->
````

```go <dataset_1 <dataset_2 <analysis1 <analysis2 <combined
package main

import (
    "fmt"
    "os"
    "strings"
)

func readVar(name string) string {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
    return strings.TrimSpace(string(data))
}

func main() {
    fmt.Println("=== DC-East ===")
    fmt.Printf("Data     : %s\n", readVar("dataset_1"))
    fmt.Printf("Analysis : %s\n", readVar("analysis1"))
    fmt.Println()
    fmt.Println("=== DC-West ===")
    fmt.Printf("Data     : %s\n", readVar("dataset_2"))
    fmt.Printf("Analysis : %s\n", readVar("analysis2"))
    fmt.Println()
    fmt.Println("=== Comparative Summary ===")
    fmt.Println(readVar("combined"))
}
```

----------------------------------------

12,45,23,67,34,89,15,52,38,71

8,19,44,11,56,23,77,31,48,62

[split] dataset_1 = 12,45,23,67,34,89,15,52,38,71

[split] dataset_2 = 8,19,44,11,56,23,77,31,48,62

[async llm] Launched PID 36285 \u2192 var=analysis1 (model @1)

[async llm] Launched PID 36286 \u2192 var=analysis2 (model @2)

[async] await= barrier: waiting for vars: analysis1,analysis2

[async] Waiting for PID 36285 (var=analysis1)...

[async] PID 36285 done (var=analysis1)

The DC-East network latency shows moderate performance with values ranging from 15ms to 89ms (average around 44ms), exhibiting high variability and inconsistent performance with 89ms standing out as a significant outlier that's nearly 6 times higher than the best case of 15ms.

DC-West demonstrates superior performance characteristics with lower baseline latency (low teens to ~70ms) compared to DC-East's broader range (15-89ms), and exhibits better consistency with moderate spikes reaching only 77ms versus DC-East's extreme outlier of 89ms. Based on these metrics, DC-West is the recommended choice due to its lower average latency, tighter performance distribution, and fewer severe outliers that could impact user experience.

=== DC-East ===

Data     : 12,45,23,67,34,89,15,52,38,71

Analysis : The DC-East network latency shows moderate performance with values ranging from 15ms to 89ms (average around 44ms), exhibiting high variability and inconsistent performance with 89ms standing out as a significant outlier that's nearly 6 times higher than the best case of 15ms.

=== DC-West ===

Data     : 8,19,44,11,56,23,77,31,48,62

Analysis : DC-West latency ranges roughly from low teens to around 70 ms, is moderately consistent overall, but shows higher spikes at 56\u201377 ms that indicate occasional outliers.

=== Comparative Summary ===

DC-West demonstrates superior performance characteristics with lower baseline latency (low teens to ~70ms) compared to DC-East's broader range (15-89ms), and exhibits better consistency with moderate spikes reaching only 77ms versus DC-East's extreme outlier of 89ms. Based on these metrics, DC-West is the recommended choice due to its lower

average latency, tighter performance distribution, and fewer severe outliers that could impact user experience.

-----------------------------------------------------------------------------

/home/igor > Sent to mshell (1360 bytes)

Received from GUI editor:

---------------------------------------

# Pattern 17 CONFIG Node: Parameterized Go Pipeline

```config
topic=Go interfaces

audience=junior developer

max_words=60
```

```go >topic
package main

import "fmt"

func main() {

  fmt.Println("Go interfaces")

}
```

```go >audience
package main

import "fmt"

func main() {

  fmt.Println("junior developer")

}
```

<!--@1 <topic <audience >explanation

~~~~
The first input is a technical topic. The second input is the target audience.

Explain the topic to that audience in at most 60 words. Use plain language and a short example.

-->

<!--@2 <explanation >keywords

The input is a short technical explanation. Extract exactly 5 keywords from it.

Reply with only a comma-separated list of 5 words, nothing else.

-->

```go <explanation <keywords >report

package main

import (

    "fmt"

    "os"

    "strings"

)

func readVar(name string) string {

    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))

    return strings.TrimSpace(string(data))

}

func main() {

    lines := []string{

        "=== Workflow Report ===",

        fmt.Sprintf("Topic    : %s", readVar("topic")),

        fmt.Sprintf("Audience : %s", readVar("audience")),

        "",

        "Explanation:",

        readVar("explanation"),
~~~~

        "",

        fmt.Sprintf("Keywords : %s", readVar("keywords")),

    }

    report := strings.Join(lines, "\n")

    fmt.Println(report)

}

```

[config] topic=Go interfaces

[config] audience=junior developer

[config] max_words=60

Go interfaces

junior developer

Go interfaces define what methods a type must have, without specifying how. Any type that has those methods automatically satisfies the interface.

```go

type Writer interface {

    Write([]byte) (int, error)

}

```

Both `os.File` and `bytes.Buffer` satisfy `Writer` because they have a `Write` method. This lets you write functions that work with any Writer.

interfaces,type,methods,Writer,polymorphism

=== Workflow Report ===

Topic    : Go interfaces

Audience : junior developer

Explanation:

Go interfaces define what methods a type must have, without specifying how. Any type that has those methods automatically satisfies the interface.
```

```go
type Writer interface {
    Write([]byte) (int, error)
}
```

Both `os.File` and `bytes.Buffer` satisfy `Writer` because they have a `Write` method. This lets you write functions that work with any Writer.

Keywords : interfaces,type,methods,Writer,polymorphism

--------------------------------------------------------------------------------

/home/igor > Sent to mshell (1341 bytes)

Received from GUI editor:

----------------------------------------

# Pattern 18  FOREACH + Async LLM: Parallel Go Pattern Batch

```go >patterns
package main

import "fmt"

func main() {
    items := []string{
        "Functional Options",
        "Worker Pool",
        "Pipeline",
        "Fan-Out Fan-In",
    }
    for _, item := range items {
        fmt.Println(item)
    }
```

~~~~
}
```
<!--@foreach pattern in patterns-->


<!--@1 <pattern >definition async
The input is the name of a Go design pattern. In one sentence, define what this pattern is.
-->
<!--@2 <pattern >usecase async
The input is the name of a Go design pattern. In one sentence, describe a concrete
real-world Go scenario where this pattern is most useful.
-->
```bash await=definition,usecase
```
```go <definition <usecase
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    pattern, _ := os.ReadFile(os.Getenv("MSH_VAR_pattern"))
    definition, _ := os.ReadFile(os.Getenv("MSH_VAR_definition"))
    usecase, _ := os.ReadFile(os.Getenv("MSH_VAR_usecase"))
    fmt.Printf("=== %s ===\n", strings.TrimSpace(string(pattern)))
    fmt.Printf("Definition : %s\n", strings.TrimSpace(string(definition)))
    fmt.Printf("Use case   : %s\n\n", strings.TrimSpace(string(usecase)))
~~~~

}

```

&lt;!--@end_foreach--&gt;

```go
package main

import "fmt"

func main() {
    fmt.Println("=== All patterns described ===")
}
```

----------------------------------------

Functional Options

Worker Pool

Pipeline

Fan-Out Fan-In

[foreach] iter 1: pattern=Functional Options

[async llm] Launched PID 36941 \u2192 var=definition (model @1)

[async llm] Launched PID 36942 \u2192 var=usecase (model @2)

[async] await= barrier: waiting for vars: definition,usecase

[async] Waiting for PID 36941 (var=definition)...

[async] PID 36941 done (var=definition)

The Functional Options pattern uses functions as parameters to configure struct initialization, allowing flexible and readable construction with optional settings while maintaining backward compatibility.

=== Functional Options ===
```

Definition : The Functional Options pattern uses functions as parameters to configure struct initialization, allowing flexible and readable construction with optional settings while maintaining backward compatibility.

Use case : When constructing an HTTP server with many optional settings (timeouts, TLS config, logging, middleware), the Functional Options pattern lets you provide a clean `NewServer(...Option)` API where callers pass only the configuration options they care about.

[foreach] iter 2: pattern=Worker Pool

[async llm] Launched PID 37077 \u2192 var=definition (model @1)

[async llm] Launched PID 37078 \u2192 var=usecase (model @2)

[async] await= barrier: waiting for vars: definition,usecase

[async] Waiting for PID 37077 (var=definition)...

[async] PID 37077 done (var=definition)

Worker Pool is a concurrency pattern where a fixed number of goroutines (workers) continuously process tasks from a shared channel, allowing controlled parallelism and resource management for handling multiple jobs concurrently.

=== Worker Pool ===

Definition : Worker Pool is a concurrency pattern where a fixed number of goroutines (workers) continuously process tasks from a shared channel, allowing controlled parallelism and resource management for handling multiple jobs concurrently.

Use case : A worker pool is ideal in a Go service that must process a high volume of independent HTTP requests\u2014such as resizing uploaded images\u2014by distributing jobs from a channel to a fixed number of goroutines to control concurrency and resource usage.

[foreach] iter 3: pattern=Pipeline

[async llm] Launched PID 37198 \u2192 var=definition (model @1)

[async llm] Launched PID 37199 \u2192 var=usecase (model @2)

[async] await= barrier: waiting for vars: definition,usecase

[async] Waiting for PID 37198 (var=definition)...

[async] PID 37198 done (var=definition)

The Pipeline pattern is a design pattern where data flows through a series of connected stages (typically goroutines with channels), with each stage performing a specific transformation or processing step before passing the result to the next stage.

=== Pipeline ===

Definition : The Pipeline pattern is a design pattern where data flows through a series of connected stages (typically goroutines with channels), with each stage performing a specific transformation or processing step before passing the result to the next stage.

Use case   : Use the Pipeline pattern when processing a large stream of log lines in stages\u2014parsing, filtering, enriching, and finally writing them to storage\u2014by connecting several goroutines with channels so each stage runs concurrently and the system can handle high-throughput logs efficiently.

[foreach] iter 4: pattern=Fan-Out Fan-In

[async llm] Launched PID 37313 \u2192 var=definition (model @1)

[async llm] Launched PID 37314 \u2192 var=usecase (model @2)

[async] await= barrier: waiting for vars: definition,usecase

[async] Waiting for PID 37313 (var=definition)...

[async] PID 37313 done (var=definition)

Fan-Out Fan-In is a Go concurrency pattern where work is distributed (fanned out) across multiple goroutines for parallel processing, then the results are collected and merged (fanned in) back into a single channel for consolidated output.

=== Fan-Out Fan-In ===

Definition : Fan-Out Fan-In is a Go concurrency pattern where work is distributed (fanned out) across multiple goroutines for parallel processing, then the results are collected and merged (fanned in) back into a single channel for consolidated output.

Use case   : When building a web scraper that launches many goroutines to fetch pages concurrently (fan-out) and then aggregates all parsed results into a single channel or slice for further processing or storage (fan-in), the Fan-Out Fan-In pattern is especially useful.

/home/igor > === All patterns described ===

--------------------------------------------------------------------------------

/home/igor > Sent to mshell (2512 bytes)

Received from GUI editor:

----------------------------------------

# Pattern 19 WHILE Quality Gate: Generate Go Code Until Score >= 8

```go >task

package main

import "fmt"

func main() {

    fmt.Println("Write an idiomatic Go function that safely reads a file and returns its contents as a string, handling all errors properly.")

}

```

```go >status

package main

import "fmt"

func main() {

    fmt.Println("running")

}

```

```go >iteration

package main

import "fmt"

func main() {

    fmt.Println("0")

}

```

```go >score

package main


import "fmt"

func main() {
```

    fmt.Println("0")

}

```

```go >snippet

package main

import "fmt"

func main() {

    fmt.Println("")

}

```

<!--@while status:running-->

```go <iteration >iteration

package main

import (

    "fmt"

    "os"

    "strconv"

    "strings"

)

func main() {

    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_iteration"))

    val, _ := strconv.Atoi(strings.TrimSpace(string(raw)))

    val++

    os.WriteFile(os.Getenv("MSH_VAR_iteration"), []byte(fmt.Sprintf("%d", val)), 0644)

    fmt.Printf("%d", val)

}

```
```

<!--@1 <task >snippet

The input contains a Go coding task. Write a clean, idiomatic Go function that solves it.

Include only the function(s), no main, no package declaration, no explanation.

-->

<!--@2 <snippet >score

The input is a Go code snippet. Rate it 1-10 for correctness, idiomatic Go style, and

conciseness. Reply with ONLY the integer score, nothing else.

-->

````
```go <iteration <score <snippet >status
package main
import (
   "fmt"
   "os"
   "strconv"
   "strings"
)
func main() {
   iterPath := os.Getenv("MSH_VAR_iteration")
   scorePath := os.Getenv("MSH_VAR_score")
   statusPath := os.Getenv("MSH_VAR_status")

   iter, _ := os.ReadFile(iterPath)
   scoreRaw, _ := os.ReadFile(scorePath)
   sc := strings.TrimSpace(string(scoreRaw))
   fmt.Printf("[Iter %s] Score=%s\n", strings.TrimSpace(string(iter)), sc)
   scoreVal, err := strconv.Atoi(sc)
   if err == nil && scoreVal >= 8 {
````
````

```
        os.WriteFile(statusPath, []byte("done"), 0644)
        fmt.Println("done")
    } else {
        os.WriteFile(statusPath, []byte("running"), 0644)
        fmt.Println("running")
    }
}
```

<!--@end_while-->

```go <snippet <score
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    snippet, _ := os.ReadFile(os.Getenv("MSH_VAR_snippet"))
    score, _ := os.ReadFile(os.Getenv("MSH_VAR_score"))
    fmt.Printf("=== Accepted snippet (score=%s) ===\n",
        strings.TrimSpace(string(score)))
    fmt.Println(strings.TrimSpace(string(snippet)))
}
```

Write an idiomatic Go function that safely reads a file and returns its contents as a string, handling all errors properly.

```
running
0
0
[while] iteration 1 \u2014 condition met, executing body
1func readFileAsString(filename string) (string, error) {
    content, err := os.ReadFile(filename)
    if err != nil {
        return "", fmt.Errorf("failed to read file %s: %w", filename, err)
    }
    return string(content), nil
}
10
[Iter 1] Score=10
done
[while] condition 'status:running' not met, exiting after 1 iter
/home/igor > === Accepted snippet (score=10) ===
func readFileAsString(filename string) (string, error) {
    content, err := os.ReadFile(filename)
    if err != nil {
        return "", fmt.Errorf("failed to read file %s: %w", filename, err)
    }
    return string(content), nil
}
--------------------------------------------------------------------------------
/home/igor > Sent to mshell (3100 bytes)
Received from GUI editor:
----------------------------------------
```

Pattern 20 SPLIT + Async + MERGE: Map-Reduce Go Source Analysis

```go >raw_text
package main

import "fmt"

func main() {

   fmt.Print("Go achieves concurrency through goroutines, which are lightweight threads managed by the runtime. ")

   fmt.Print("Channels provide a safe way for goroutines to communicate and synchronize. ")

   fmt.Print("The select statement allows a goroutine to wait on multiple channel operations. ")

   fmt.Print("The sync package offers primitives like Mutex and WaitGroup for shared state. ")

   fmt.Println("Go's concurrency model follows the principle: do not communicate by sharing memory, share memory by communicating.")

}
```

```go <raw_text >sent1
package main


import (

   "fmt"

   "os"

   "strings"

)

func main() {

   raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_text"))

   sentences := strings.SplitN(strings.TrimSpace(string(raw)), ". ", -1)

   if len(sentences) > 0 {
```

````
        fmt.Println(strings.TrimSpace(sentences[0]))
    }
}
```
```go <raw_text >sent2
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_text"))
    sentences := strings.SplitN(strings.TrimSpace(string(raw)), ". ", -1)
    chunk := []string{}
    for _, s := range sentences[1:3] {
        t := strings.TrimSpace(s)
        if t != "" {
            chunk = append(chunk, t)
        }
    }
    fmt.Println(strings.Join(chunk, ". "))
}
```
```go <raw_text >sent3
package main
import (
````

````
    "fmt"
    "os"
    "strings"
)
func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_text"))
    sentences := strings.SplitN(strings.TrimSpace(string(raw)), ". ", -1)
    chunk := []string{}
    for _, s := range sentences[3:] {
        t := strings.TrimSpace(s)
        if t != "" {
            chunk = append(chunk, t)
        }
    }
    fmt.Println(strings.Join(chunk, ". "))
}
```
<!--@1 <sent1 >analysis1 async
The input contains one or two sentences about Go. State the main concept in 3 words max.
-->
<!--@1 <sent2 >analysis2 async
The input contains one or two sentences about Go. State the main concept in 3 words max.
-->
<!--@1 <sent3 >analysis3 async
The input contains one or two sentences about Go. State the main concept in 3 words max.
-->
```bash await=analysis1,analysis2,analysis3
````

````
```

<!--@merge-->

<!--@2 <analysis1 <analysis2 <analysis3 >summary

The input contains three short concept labels from a text about Go's concurrency model.

Synthesize them into one coherent theme sentence that captures the overall message.

-->

```go <analysis1 <analysis2 <analysis3 <summary

package main

import (

    "fmt"

    "os"

    "strings"

)

func readVar(name string) string {

    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))

    return strings.TrimSpace(string(data))

}

func main() {

    fmt.Println("=== Map ===")

    fmt.Printf("Chunk 1: %s\n", readVar("analysis1"))

    fmt.Printf("Chunk 2: %s\n", readVar("analysis2"))

    fmt.Printf("Chunk 3: %s\n", readVar("analysis3"))

    fmt.Println()

    fmt.Println("=== Reduce ===")

    fmt.Println(readVar("summary"))

}

```
````
````

Go achieves concurrency through goroutines, which are lightweight threads managed by the runtime. Channels provide a safe way for goroutines to communicate and synchronize. The select statement allows a goroutine to wait on multiple channel operations. The sync package offers primitives like Mutex and WaitGroup for shared state. Go's concurrency model follows the principle: do not communicate by sharing memory, share memory by communicating.

Go achieves concurrency through goroutines, which are lightweight threads managed by the runtime

Channels provide a safe way for goroutines to communicate and synchronize. The select statement allows a goroutine to wait on multiple channel operations

The sync package offers primitives like Mutex and WaitGroup for shared state. Go's concurrency model follows the principle: do not communicate by sharing memory, share memory by communicating.

[async llm] Launched PID 40032 \u2192 var=analysis1 (model @1)

[async llm] Launched PID 40033 \u2192 var=analysis2 (model @1)

[async llm] Launched PID 40035 \u2192 var=analysis3 (model @1)

[async] await= barrier: waiting for vars: analysis1,analysis2,analysis3

[async] Waiting for PID 40032 (var=analysis1)...

[async] PID 40032 done (var=analysis1)

Goroutine lightweight concurrency

Go\u2019s concurrency model emphasizes lightweight goroutines coordinated through channel-based communication.

/home/igor > === Map ===

Chunk 1: Goroutine lightweight concurrency

Chunk 2: Channel-based concurrency

Chunk 3: Channel-based concurrency

=== Reduce ===

Go concurrency model emphasizes lightweight goroutines coordinated through channel-based communication.

/home/igor > Sent to mshell (2022 bytes)

Received from GUI editor:

Pattern 21 TRY/CATCH + LOOP: Resilient Go Code Retry

```go >task
package main

import "fmt"

func main() {

   fmt.Println(`Write a Go program that parses the JSON string {"name":"Alice","age":30} and prints the name field. No external imports beyond encoding/json and fmt.`)

}
```

```go >result
package main

import "fmt"

func main() { fmt.Println("fail") }
```

```go >last_error
package main

import "fmt"

func main() { fmt.Println("none") }
```

<!--@loop max=3 until=result:ok-->

<!--@1 <task <last_error >code

The first input describes a Go coding task.

The second input contains the error from the previous attempt (or "none" if first attempt).

Return ONLY a complete Go program (package main, with imports), no markdown fences, no explanation.

````
If there was a previous error, fix it.

-->

```go <code

package main

import (

    "fmt"

    "os"

)

func main() {

    code, _ := os.ReadFile(os.Getenv("MSH_VAR_code"))

    fmt.Println("=== Generated Code ===")

    fmt.Print(string(code))

}

```

<!--@try-->

```go <code

package main

import (

    "fmt"

    "os"

    "os/exec"

)

func main() {

    code, _ := os.ReadFile(os.Getenv("MSH_VAR_code"))

    resultPath := os.Getenv("MSH_VAR_result")

    tmp, err := os.CreateTemp("", "msh_p21_*.go")

    if err != nil { os.Exit(1) }
````
````

````
    tmpName := tmp.Name()
    tmp.Write(code)
    tmp.Close()
    out, err := exec.Command("go", "run", tmpName).CombinedOutput()
    os.Remove(tmpName)
    if err != nil {
        fmt.Fprintf(os.Stderr, "run failed: %s\n", out)
        os.Exit(1)
    }
    fmt.Print(string(out))
    os.WriteFile(resultPath, []byte("ok\n"), 0644)
    fmt.Println("ok")
}
```
<!--@catch >last_error-->
```go >result
package main
import (
    "fmt"
    "os"
)
func main() {
    fmt.Println("=== Error: try_block_failed ===")
    os.WriteFile(os.Getenv("MSH_VAR_result"), []byte("fail\n"), 0644)
    fmt.Println("fail")
}
```
````
````

<!--@end_try-->

<!--@end_loop-->

```bash <result
echo "=== Final status: $(cat $(printenv MSH_VAR_result) | tr -d '\n') ==="
```

Write a Go program that parses the JSON string {"name":"Alice","age":30} and prints the name field. No external imports beyond encoding/json and fmt.

fail

none

[loop] Starting loop: max=3 until=result:ok

```
package main

import (
    "encoding/json"
    "fmt"
)

type Person struct {
    Name string `json:"name"`
    Age  int    `json:"age"`
}

func main() {
    jsonStr := `{"name":"Alice","age":30}`
    var person Person
    err := json.Unmarshal([]byte(jsonStr), &person)
    if err != nil {
        fmt.Printf("Error parsing JSON: %v\n", err)
        return
    }
```

fmt.Println(person.Name)

}

[try] executing try block

Alice

ok

Alice

ok

[try] try block succeeded

[loop] Exiting loop after 1 iteration(s). reason: until condition met

=== Final status: ok ===

/home/igor > Sent to mshell (1941 bytes)

Received from GUI editor:

Pattern 22 Multi-Variable Output: Structured Go Function Extraction

```go >input
package main

import "fmt"

func main() {

   fmt.Println("A function that takes a context, a database connection, a user ID string, and a boolean includeDeleted flag, and returns a pointer to a User struct and an error.")

}
```

<!--@1 <input >raw_response

Respond in exactly this format (3 lines, no extra text):

FUNCNAME: one short idiomatic Go function name in camelCase

PARAMS: comma-separated list of parameter names and types

~~~
RETURNS: comma-separated return types
-->

```go <raw_response >funcname >params >returns
package main
import (
    "fmt"
    "os"
    "regexp"
    "strings"
)
func extract(text, field string) string {
    re := regexp.MustCompile(field + `:\s*(.+)`)
    m := re.FindStringSubmatch(text)
    if len(m) > 1 {
        return strings.TrimSpace(m[1])
    }
    return "n/a"
}
func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_raw_response"))
    text := string(raw)
    funcname := extract(text, "FUNCNAME")
    params := extract(text, "PARAMS")
    returns := extract(text, "RETURNS")
    os.WriteFile(os.Getenv("MSH_VAR_funcname"), []byte(funcname), 0644)
    os.WriteFile(os.Getenv("MSH_VAR_params"), []byte(params), 0644)
~~~

````
    os.WriteFile(os.Getenv("MSH_VAR_returns"), []byte(returns), 0644)
    fmt.Printf("FuncName : %s\n", funcname)
    fmt.Printf("Params   : %s\n", params)
    fmt.Printf("Returns  : %s\n", returns)
}
```
<!--@2 <funcname <params >doc
The first input is a Go function name. The second input is its parameter list.
Write a complete Go doc comment for this function (starting with // FuncName ...).
Two sentences maximum.
-->
```go <doc
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    doc, _ := os.ReadFile(os.Getenv("MSH_VAR_doc"))
    fmt.Println()
    fmt.Println("=== Generated doc comment ===")
    fmt.Println(strings.TrimSpace(string(doc)))
}
```
````
````

A function that takes a context, a database connection, a user ID string, and a boolean includeDeleted flag, and returns a pointer to a User struct and an error.

FUNCNAME: getUserByID

PARAMS: ctx context.Context, db *sql.DB, userID string, includeDeleted bool

RETURNS: *User, error

FuncName : getUserByID

Params   : ctx context.Context, db *sql.DB, userID string, includeDeleted bool

Returns  : *User, error

// getUserByID retrieves a user record by its ID from the provided database, honoring the given context for cancellation and timeouts.

// If includeDeleted is true, the function may also return users that have been soft-deleted.

/home/igor >

=== Generated doc comment ===

// getUserByID retrieves a user record by its ID from the provided database, honoring the given context for cancellation and timeouts.

// If includeDeleted is true, the function may also return users that have been soft-deleted.

--------------------------------------------------------------------------------

/home/igor > Sent to mshell (3055 bytes)

Received from GUI editor:

----------------------------------------

# Pattern 23  CONFIG + WHILE + Multi-Model: Adaptive Go Code Pipeline

```config

subject=Go generics and type constraints

target_audience=experienced Go developer new to generics

quality_threshold=7

```

```go >subject
```

````
package main

import "fmt"

func main() {

   fmt.Println("Go generics and type constraints")

}

```

```go >target_audience

package main

import "fmt"

func main() {

   fmt.Println("experienced Go developer new to generics")

}

```

```go >status

package main

import "fmt"

func main() {

   fmt.Println("running")

}

```

```go >iteration

package main

import "fmt"


func main() {

   fmt.Println("0")

}
````

````
```

```go >quality
package main
import "fmt"
func main() {
   fmt.Println("0")
}
```

```go >explanation
package main
import "fmt"
func main() {
   fmt.Println("")
}
```

<!--@while status:running-->

```go <iteration >iteration
package main
import (
   "fmt"
   "os"
   "strconv"
   "strings"
)
func main() {
   raw, _ := os.ReadFile(os.Getenv("MSH_VAR_iteration"))
   val, _ := strconv.Atoi(strings.TrimSpace(string(raw)))
````

````
   val++
   os.WriteFile(os.Getenv("MSH_VAR_iteration"), []byte(fmt.Sprintf("%d", val)), 0644)
   fmt.Printf("%d", val)
}
```
<!--@1 <subject <target_audience >explanation
The first input is a technical subject to explain. The second input is the target audience.
Explain the subject to that audience in exactly 3 sentences. Use no jargon. Be concrete and
give a brief practical example.
-->
<!--@2 <explanation <target_audience >quality
The first input is an explanation text. The second input is the target audience it was written for.
Rate the explanation for that audience on a scale 1-10.
Criteria: clarity, technical accuracy, engagement for that audience.
Reply with ONLY the integer score, nothing else.
-->
```go <iteration <quality >status
package main
import (
   "fmt"
   "os"
   "strconv"
   "strings"
)
func main() {
   iter, _ := os.ReadFile(os.Getenv("MSH_VAR_iteration"))
````

~~~~
    scoreRaw, _ := os.ReadFile(os.Getenv("MSH_VAR_quality"))
    statusPath := os.Getenv("MSH_VAR_status")
    sc := strings.TrimSpace(string(scoreRaw))
    fmt.Printf("[Iter %s] Quality: %s\n", strings.TrimSpace(string(iter)), sc)
    q, err := strconv.Atoi(sc)
    if err == nil && q >= 7 {
        os.WriteFile(statusPath, []byte("done"), 0644)
        fmt.Println("done")
    } else {
        os.WriteFile(statusPath, []byte("running"), 0644)
        fmt.Println("running")
    }
}
```
<!--@end_while-->
<!--@3 <explanation >final_polish
The input contains an explanation text. Polish it slightly for final publication.
Keep exactly 3 sentences. No markdown formatting.
-->


```go <final_polish <quality <iteration
package main
import (
    "fmt"
    "os"
    "strings"
)
~~~~

```go
func readVar(name string) string {
    data, _ := os.ReadFile(os.Getenv("MSH_VAR_" + name))
    return strings.TrimSpace(string(data))
}
func main() {
    fmt.Printf("=== Final (score=%s, iters=%s) ===\n",
        readVar("quality"), readVar("iteration"))
    fmt.Println(readVar("final_polish"))
}
```

[config] subject=Go generics and type constraints

[config] target_audience=experienced Go developer new to generics

[config] quality_threshold=7

Go generics and type constraints

experienced Go developer new to generics

running

0

0

[while] iteration 1 condition met, executing body

1Go generics let you write functions and data structures that work with multiple types while maintaining compile-time type safety, eliminating the need for interface{} and type assertions in many cases. You define type parameters using square brackets and constrain them with interfaces - for example, `func Max[T comparable](a, b T) T` creates a generic function that works with any comparable type like int, string, or float64. Type constraints like `constraints.Ordered` from golang.org/x/exp let you specify exactly what operations your generic code can perform, so `func Sort[T constraints.Ordered](slice []T)` can sort any slice of numbers or strings without duplicating code.

10

[Iter 1] Quality: 10

done

[while] condition 'status:running' not met, exiting after 1 iter

Go generics enable writing type-agnostic functions and data structures with compile-time type safety, reducing reliance on `interface{}` and type assertions while ensuring type correctness. Type parameters are declared in square brackets and constrained by interfaces, as demonstrated by `func Max[T comparable](a, b T) T`, which operates on any comparable type like integers, strings, or floats. Leveraging constraint packages such as `golang.org/x/exp/constraints` allows precise specification of required operations (e.g., `Ordered` for sorting) to create reusable generic algorithms without code duplication.

=== Final (score=10, iters=1) ===

Go generics enable writing type-agnostic functions and data structures with compile-time type safety, reducing reliance on `interface{}` and type assertions while ensuring type correctness. Type parameters are declared in square brackets and constrained by interfaces, as demonstrated by `func Max[T comparable](a, b T) T`, which operates on any comparable type like integers, strings, or floats. Leveraging constraint packages such as `golang.org/x/exp/constraints` allows precise specification of required operations (e.g., `Ordered` for sorting) to create reusable generic algorithms without code duplication.

/home/igor > Sent to mshell (1643 bytes)

Received from GUI editor:

Pattern 24 FOREACH + TRY/CATCH: Fault-Tolerant Go Batch Processing

````
```go >items
package main

import "fmt"

func main() {

    records := []string{

        `{"name":"Alice","role":"engineer","active":true}`,

        `{name: broken json}`,

        `{"name":"Bob","role":"manager","active":false}`,

        `{"name":"Carol","role":"designer","active":true}`,
````
````

```
    }
    for _, r := range records {
        fmt.Println(r)
    }
}
```

<!--@foreach item in items-->

<!--@try-->

```go <item >parsed
package main
import (
    "encoding/json"
    "fmt"
    "os"
    "strings"
)
func main() {
    raw, _ := os.ReadFile(os.Getenv("MSH_VAR_item"))
    line := strings.TrimSpace(string(raw))
    var obj map[string]interface{}
    if err := json.Unmarshal([]byte(line), &obj); err != nil {
        fmt.Fprintf(os.Stderr, "parse error: %v\n", err)
        os.Exit(1)
    }
    fmt.Printf("Parsed OK: %v\n", obj)
}
```

<!--@1 <parsed >insight

The input contains a successfully parsed Go map from JSON. In one sentence, describe what kind of data record this represents based on its fields and values.

-->

```go <insight
package main
import (
    "fmt"
    "os"
    "strings"
)
func main() {
    insight, _ := os.ReadFile(os.Getenv("MSH_VAR_insight"))
    fmt.Printf("[OK] %s\n", strings.TrimSpace(string(insight)))
}
```

<!--@catch >parse_error-->

```go
package main
import "fmt"
func main() {
    fmt.Println("[ERR] Parse failed: try_block_failed")
}
```

<!--@end_try-->

<!--@end_foreach-->

```go
```

```
package main

import "fmt"

func main() {

    fmt.Println("=== Batch complete. Errors were isolated, pipeline never stopped. ===")

}
```

---------------------------------------

{"name":"Alice","role":"engineer","active":true}

{name: broken json}

{"name":"Bob","role":"manager","active":false}

{"name":"Carol","role":"designer","active":true}

[foreach] iter 1: item={"name":"Alice","role":"engineer","active":true}

[try] executing try block

Parsed OK: map[active:true name:Alice role:engineer]

This represents an active employee or user record for Alice who works as an engineer, with a status flag indicating she is currently active in the system.

[OK] This represents an active employee or user record for Alice who works as an engineer, with a status flag indicating she is currently active in the system.

[try] try block succeeded

[foreach] iter 2: item={name: broken json}

[try] executing try block

parse error: invalid character 'n' looking for beginning of object key string

This represents an active employee or user record for Alice who works as an engineer, with a status flag indicating she is currently active in the system.

[try] try block failed, executing catch block

[ERR] Parse failed: try_block_failed

[foreach] iter 3: item={"name":"Bob","role":"manager","active":false}

[try] executing try block

---

Parsed OK: map[active:false name:Bob role:manager]

This represents an inactive employee or user record for Bob who has a manager role, with the status flag indicating he is currently deactivated or no longer active in the system.

[OK] This represents an inactive employee or user record for Bob who has a manager role, with the status flag indicating he is currently deactivated or no longer active in the system.

[try] try block succeeded

[foreach] iter 4: item={"name":"Carol","role":"designer","active":true}

[try] executing try block

Parsed OK: map[active:true name:Carol role:designer]

This represents an active employee or user record for Carol who works as a designer, with a status flag indicating she is currently active in the system.

[OK] This represents an active employee or user record for Carol who works as a designer, with a status flag indicating she is currently active in the system.

[try] try block succeeded

/home/igor > === Batch complete. Errors were isolated, pipeline never stopped. ===

---

References:

*Go language Patterns for mshell Workflow — Complete Reference Guide (P1–P24) Pure Go Edition — Art2Dec SoftLab (Non-profitable SoftLab), 2026 Created by Igor Lukyanov, based on the original mshell Workflow Patterns Reference Guides Part I & Part II*

Resources: - Common examples Part I (P1–P12): https://www.appservgrid.com/paw92/index.php/2026/02/26/mshell-workflow-patterns-reference-guide-part-i-p1-p13/

Resources: - Common examples Part II (P13–P24): https://www.appservgrid.com/paw92/index.php/2026/03/11/mshell-workflow-patterns-reference-guide-part-ii-p13-p24/

- mshell v1.4.1 cheatsheet: https://www.appservgrid.com/paw92/index.php/2026/02/04/mshell-v-1-4-1-cheatsheet-january-26th-2026/