Shadowing in Nested Blocks

```
package main
import "fmt"

func main() {
    x := 10
    if x > 5 {
        x := 20
        fmt.Println(x)
    }
    fmt.Println(x)
}
```

Shadowing with Short Variable Declaration

```go
package main
import "fmt"

func main() {
    x := 10
    if x > 5 {
        x, y := 20, 30
        fmt.Println(x, y)
    }
    fmt.Println(x)
}
```

Dangerous Shadowing with :=

```go
package main
import "fmt"

func main() {
    x := 10
    if x > 5 {
        x := x + 5
        fmt.Println(x)
    }
    fmt.Println(x)
}
```

Universe Block Shadowing

```
package main
import "fmt"

func main() {
    true := false
    fmt.Println(true)
}
```

If with Init Statement

```
package main
import "fmt"

func main() {
    if n := 10; n > 5 {
        fmt.Println(n)
    }
    fmt.Println(n) // What happens?
}
```

If-Else Scope

```go
package main
import "fmt"

func main() {
    if n := 10; n > 20 {
        fmt.Println("big")
    } else {
        fmt.Println(n)
    }
    // Can we access n here?
}
```

For Loop Variable Scope

```go
package main
import "fmt"

func main() {
    for i := 0; i < 3; i++ {
        fmt.Println(i)
    }
    fmt.Println(i) // What happens?
}
```

For-Range Shadowing Gotcha

```go
package main
import "fmt"

func main() {
    x := 10
    nums := []int{1, 2, 3}
    for i, x := range nums {
        fmt.Println(i, x)
    }
    fmt.Println(x)
}
```

Switch with Init Statement

```go
package main
import "fmt"

func main() {
    switch n := 5; n {
    case 5:
        fmt.Println(n)
        n = 10
        fmt.Println(n)
    case 10:
        fmt.Println("ten")
    }
    // Can we access n here?
}
```

Explicit Fallthrough

```
package main
import "fmt"

func main() {
    x := 2
    switch x {
    case 1:
        fmt.Println("one")
    case 2:
        fmt.Println("two")
        fallthrough
    case 3:
        fmt.Println("three")
    case 4:
        fmt.Println("four")
    }
}
```

Switch with Multiple Conditions

```go
package main
import "fmt"

func main() {
    x := 3
    switch x {
    case 1, 3, 5:
        fmt.Println("odd")
    case 2, 4, 6:
        fmt.Println("even")
    default:
        fmt.Println("other")
    }
}
```

Blank Switch (Switch True)

```go
package main
import "fmt"

func main() {
    x := 10
    switch {
    case x < 0:
        fmt.Println("negative")
    case x == 0:
        fmt.Println("zero")
    case x > 0:
        fmt.Println("positive")
    }
}
```

For Loop Continue

```
package main
import "fmt"

func main() {
    for i := 0; i < 5; i++ {
        if i%2 == 0 {
            continue
        }
        fmt.Println(i)
    }
}
```

For Loop Break

```go
package main
import "fmt"

func main() {
    for i := 0; i < 10; i++ {
        if i == 5 {
            break
        }
        fmt.Println(i)
    }
    fmt.Println("done")
}
```

Labeled Break in Nested Loops

```go
package main
import "fmt"

func main() {
outer:
    for i := 0; i < 3; i++ {
        for j := 0; j < 3; j++ {
            if i == 1 && j == 1 {
                break outer
            }
            fmt.Println(i, j)
        }
    }
    fmt.Println("done")
}
```

Labeled Continue in Nested Loops

```go
package main
import "fmt"

func main() {
outer:
    for i := 0; i < 3; i++ {
        for j := 0; j < 3; j++ {
            if j == 1 {
                continue outer
            }
            fmt.Println(i, j)
        }
    }
}
```

For-Range with String

```
package main
import "fmt"

func main() {
    s := "Go!"
    for i, v := range s {
        fmt.Printf("%d: %c (%T)\n", i, v, v)
    }
}
```

For-Range Modifying Slice

```go
package main
import "fmt"

func main() {
    nums := []int{1, 2, 3}
    for _, v := range nums {
        v = v * 2
    }
    fmt.Println(nums)
}
```

For-Range with Map Order

```
package main
import "fmt"

func main() {
    m := map[string]int{"a": 1, "b": 2, "c": 3}
    for k, v := range m {
        fmt.Println(k, v)
    }
}
```

Goto Statement (Rarely Used)

```go
package main
import "fmt"

func main() {
    i := 0
start:
    fmt.Println(i)
    i++
    if i < 3 {
        goto start
    }
    fmt.Println("done")
}
```

Block Scope without Control Structure

```
package main
import "fmt"

func main() {
    x := 10
    {
        x := 20
        fmt.Println(x)
    }
    fmt.Println(x)
}
```

Shadowing Package Names

```go
package main
import "fmt"

func main() {
    fmt := "hello"
    fmt.Println(fmt) // What happens?
}
```

Early Loop Termination Pattern

```go
package main
import "fmt"

func main() {
    nums := []int{1, 2, 3, 4, 5, 6}
    found := false
    for _, n := range nums {
        if n == 4 {
            found = true
            break
        }
    }
    fmt.Println(found)
}
```

Infinite Loop with Condition

```
package main
import "fmt"

func main() {
    i := 0
    for {
        if i >= 3 {
            break
        }
        fmt.Println(i)
        i++
    }
}
```

🎯 BONUS CHALLENGE

```go
package main
import "fmt"

func main() {
    x := 10
    y := 20

    if x < y {
        x, y := 30, 40
        fmt.Println(x, y)

        if true {
            x := 50
            fmt.Println(x, y)
        }
        fmt.Println(x, y)
    }
    fmt.Println(x, y)
}
```