

iota

01

About

02

Custom Values

03

Enumeration Pattern



About

- | **const** is like a variable, but unchanging
- | Common to make groups of constants
- | **iota** keyword can be used to automatically assign values

```
const (
    Online = 0
    Offline = 1
    Maintenance = 2
    Retired = 3
)
```

```
const (
    Online = iota
    Offline
    Maintenance
    Retired
)
```

iota

// Long-form:

```
const (
    L0 = iota // 0
    L1 = iota // 1
    L2 = iota // 2
    L3 = iota // 3
    L4 = iota // 4
)
```

// Short-form:

```
const (
    s0 = iota // 0
    s1           // 1
    s2           // 2
    s3           // 3
    s4           // 4
)
```

Skipping Values

```
// Skip a value
const (
    s0 = iota // 0
    -           // 1 (skip)
    -           // 2 (skip)
    s3           // 3
    s4           // 4
)
```

```
// Start at 3
const (
    i3 = iota + 3 // 3 = iota + 3
    i4           // 4
    i5           // 5
)
```

iota Enumeration Pattern

```
type Direction byte
const (
    North Direction = iota
    East
    South
    West
)
north := North
fmt.Println(north)
// prints "North"
```

```
func (d Direction) String() string {
    switch d {
    case North:
        return fmt.Sprintf("North")
    case South:
        return fmt.Sprintf("South")
    case East:
        return fmt.Sprintf("East")
    case West:
        return fmt.Sprintf("West")
    default:
        return "other direction"
}
```

iota Enumeration Pattern

```
type Direction byte

const (
    North Direction = iota
    East
    South
    West
)

func (d Direction) String() string {
    return []string{"North", "East", "South", "West"}[d]
}

north := North
fmt.Println(north)
// prints "North"
```

Recap

- | The **iota** keyword can be used to assign integers to constants
- | Replicates C-style enums
- | Values can be skipped by using an underscore (_)
- | **iota** values can be expressions (**iota + 5**)
- | Use a receiver function to more easily work with constants and **iota**