

# Generics

**01**

About

**02**

Syntax

**03**

Approximation

# Generics

- | Allow one function to handle multiple types of data
- | Reduces code duplication
  - | Less code = less bugs chances
- | Generics are defined using interfaces (called **constraints**)
  - | Function parameters / return types are constrained to a specific set of interfaces

# Syntax

Generic type name



"or"



```
func name[T constraint, U constraintA | constraintB](a T, b U) T {  
    // ...  
}
```



Constraint / Interface

# Example

```
func IsEqual[T comparable](a, b T) bool {  
    return a == b  
}
```

```
IsEqual(2, 2)
```

```
IsEqual("foo", "bar")
```

```
IsEqual('a', 'b')
```

```
IsEqual[uint8](4, 4)
```



# Creating a Constraint

```
type Integers32 interface {  
    int32 | uint32  
}  
  
func SumNumbers[T Integers32](arr []T) T {  
    var sum T  
    for i := 0; i < len(arr); i++ {  
        sum += arr[i]  
    }  
    return sum  
}  
  
nums := []int32{1, 2, 3}  
nums2 := []uint32{1, 2, 3}  
total := SumNumbers(nums)  
total2 := SumNumbers(nums2)
```

# Constraints and Type Aliases

```
type Integers32 interface {  
    int32 | uint32  
}  
  
func SumNumbers[T Integers32](arr []T) T {  
    var sum T  
    for i := 0; i < len(arr); i++ {  
        sum += arr[i]  
    }  
    return sum  
}  
  
type MyInt int32  
nums := []MyInt{MyInt(1), MyInt(2), MyInt(3)}
```

Error: MyInt does not implement Integers32

# Approximation

```
type Integers32 interface {  
    ~int32 | ~uint32  
}  
  
func SumNumbers[T Integers32](arr []T) T {  
    var sum T  
    for i := 0; i < len(arr); i++ {  
        sum += arr[i]  
    }  
    return sum  
}  
  
type MyInt int32  
nums := []MyInt{MyInt(1), MyInt(2), MyInt(3)}
```

# Builtin Constraints

**constraints  
Package**

**Constraint**

**Description**

any

Any type

comparable

Anything that can be compared for equality

Unsigned

All unsigned integers

Signed

All signed integers

Ordered

Sortable types (numbers, strings)

Integer

All integers

Float

All floating point numbers

Complex

All complex numbers



# Generic Structure

```
import "golang.org/x/exp/constraints"

type MyArray[T constraints.Ordered] struct {
    inner []T
}

func (m *MyArray[T]) Max() T {
    max := m.inner[0]
    for i := 0; i < len(m.inner); i++ {
        if m.inner[i] > max {
            max = m.inner[i]
        }
    }
    return max
}

arr := MyArray[int]{inner: []int{6, 4, 8, 9, 4, 0}}
fmt.Println(arr.Max())
```



# Recap

- | Generic functions reduce code duplication
- | Generic constraints are interfaces which specify allowable types
- | Use tilde (~) to allow approximate types
  - | Approximation permits checking underlying types
- | The **constraints** package has commonly used constraints
- | The **comparable** constraint is always available