Multiple return values

- Deklaracja funkcji może zawierac kilka zwracanych wartości.
- Wartości mogą być różnego typu.
- playground

```
func addSubDiv(a, b int) (int, int, float64) {
    add := a + b
    sub := a - b
    div := float64(a) / float64(b)
    return add, sub, div // (int, int, float64)
}

func main() {
    add, sub, div := addSubDiv(11, 5)

    fmt.Printf("Add: %d, Sub: %d, Div: %.2f\n", add, sub, div)
}
```

Named return values

- Zwracane wartości mogą mieć swoje nazwy.

```
func addSubDiv(a, b int) (add int, sub int, div float64) {
    add = a + b
    sub = a - b
    div = float64(a) / float64(b)

    return
}

func main() {
    add, sub, div := addSubDiv(11, 5)

    fmt.Printf("Add: %d, Sub: %d, Div: %.2f\n", add, sub, div)
}
```

Pointer receiver

- Metoda, jako działanie na rzecz struktury
- playground

```
type Adder struct {
    Sum int
    Sub int
}

func (a *Adder) add(x, y int) {
    a.Sum = x + y
}

func sub(a *Adder, x, y int) {
    a.Sub = x - y
}

func main() {
    adder := Adder{}

    adder.add(2, 1)
    sub(&adder, 2, 1)

    fmt.Printf("Sum: %d, Sub: %d\n", adder.Sum, adder.Sub)
}
```

Value receiver

- Receiver jako “wartość”.
- playground

```
type Adder struct {
    Sum int
    Sub int
}

func (a Adder) add(x, y int) {
    a.Sum = x + y
}

func sub(a Adder, x, y int) Adder {
    a.Sub = x - y

    return a
}

func main() {
    adder := Adder{}

    adder.add(2, 1)
    adder = sub(adder, 2, 1)

    fmt.Printf("Sum: %d, Sub: %d\n", adder.Sum, adder.Sub)
}
```

New keyword

- Tworzenie struktur za pomocą słowa kluczowego `new`

```
package main

import "fmt"

type Foo struct {
    Bar string
}

func main() {
    foo := new(Foo)
    foo.Bar = "Baz"
    fmt.Printf("foo: %+v = %+v\n", foo, *foo)
    // output: foo: &{Bar:Baz} = {Bar:Baz}
}
```

Make keyword

- Tworzenie tablic, kanałów lub map przy pomocy `make`

```
package main

import "fmt"

func main() {

    a := make([]int, 10) // stwórz tablicę o pojemności 10
    fmt.Printf("len: %d, cap: %d, val: %v\n", len(a), cap(a), a)
    // output: len: 10, cap: 10, val: [0 0 0 0 0 0 0 0 0 0]

    ch := make(chan int, 4) // swtórz kanał o pojemności 4
    ch <- 1
    fmt.Printf("len: %d, cap: %d, val: %v\n", len(ch), cap(ch), ch)
    // output: len: 1, cap: 4, val: 0xc000110000

    m := make(map[int]int) // swtórz mapę o pojemności ?
    fmt.Printf("len: %d, val: %v\n", len(m), m)
    // output: len: 0, val: map[]
}
```