

# Golang для начинающих



Клестов Дима,  
Golang-разработчик ООО «Инносети»

[https://t.me/MaJloe\\_3Jlo](https://t.me/MaJloe_3Jlo)  
<https://github.com/MaJloe3Jlo>

# Результаты занятия 3

Все молодцы! Но опять строчки.

Хорошие решения:

- 1) [https://play.golang.org/p/7oqiU3m\\_3SJ](https://play.golang.org/p/7oqiU3m_3SJ)
- 2) <https://play.golang.org/p/0GDyEe-uQM6>
- 3) <https://play.golang.org/p/UlrA4aQWsQw>
- 4) <https://play.golang.org/p/R3iTerz-wlD>
- 5) <https://play.golang.org/p/fgDXGKkmobQ>
- 6) <https://play.golang.org/p/X4Tom5kVgVL>

<https://play.golang.org/>



The screenshot shows the 'The Go Playground' interface. At the top, there is a header bar with the title 'The Go Playground' and several buttons: 'Run', 'Format', 'Imports' (with a checkbox), and 'Share'. To the right of these buttons is a text input field containing 'Hello, playground'. Below the header is a large yellow area for code. A red arrow points from the bottom towards the 'Share' button. The code in the editor is as follows:

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     fmt.Println("Hello, playground")
9 }
10
```

# План занятия

На прошлом занятии:

- 1) Управляющие конструкции
- 2) Коллекции данных
- 3) Домашняя работа

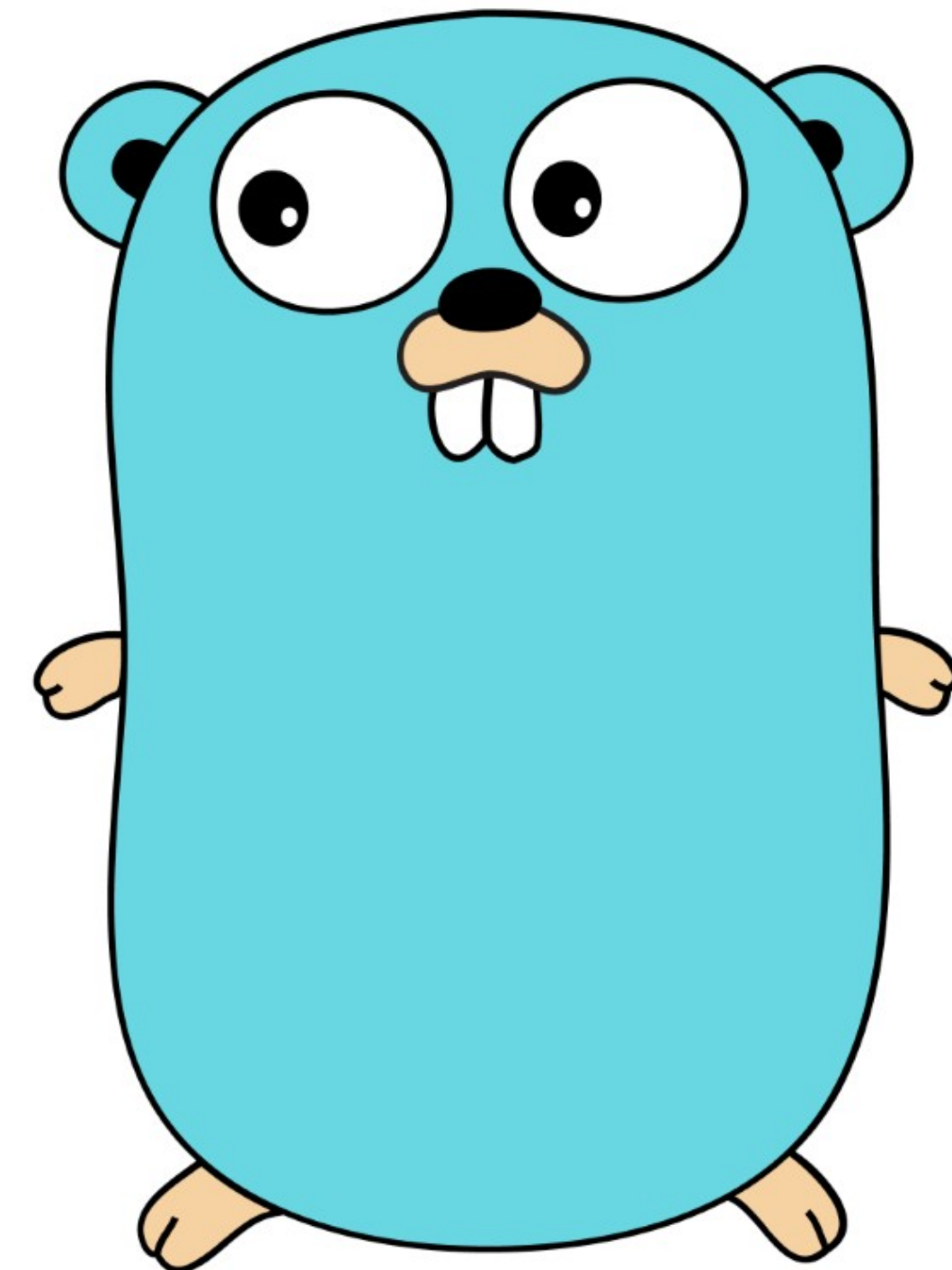


План занятия на сегодня:

- 1) Функции
- 2) Указатели
- 3) Домашняя работа

Служат для:

- ✓ Разбития на логические блоки сложного кода.
- ✓ Сокращения кода.
- ✓ Переиспользование повторяющегося кода.



Объявление функции в общем виде:

```
func <название> (<список параметров>) (<типы возвращаемых значений>) {  
    <выполняемые операторы>  
}
```

Все части в объявлении функций опциональны (могут быть не объявлены).

Пример без параметров и возвращаемых значений:

```
func main() {  
    hello() // Вызов функции.  
}
```

```
func hello() {  
    fmt.Println("Hello world!")  
}  
// Вывод: Hello world!
```

# Параметры функции

- ✓ Через параметры функция получает входные данные.
- ✓ Параметры указываются в круглых скобках() после имени функции.
- ✓ Для каждого параметра указывается тип.
- ✓ Разделение параметров с помощью запятой (,).

Пример:

```
func greeting(name string, age int) {  
    fmt.Printf("My name is %s. My age is %d", name, string)  
}
```

```
func main() {  
    greeting("Dima", 31)  
}
```

// Вывод: My name Dima. My age is 31

- ✓ Параметры одного типа можно объединять по типу. Пример:

```
func multiply(x, y int) {  
    fmt.Println(x*y)  
}
```



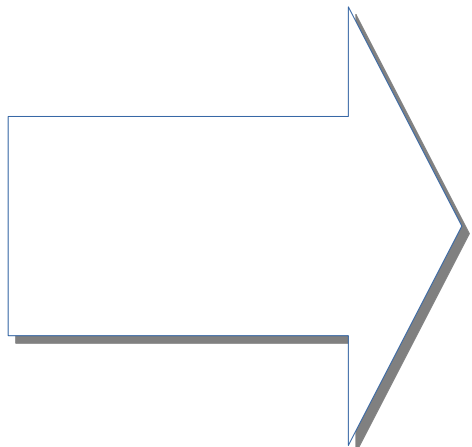
# Возвращение результата из функции

- ✓ Функции могут возвращать результат.
- ✓ После круглых скобок() указывается тип возвращаемого значения.
- ✓ Возврат в теле функции по слову return.

Общий вид:

```
func <имя функции> (<список параметров>) <тип возвращаемого значения> {  
    // выполняемые операторы  
    return <возвращаемое значение>  
}
```

Пример функции суммы двух чисел:

<pre>func sum (x, y int) int {     z := x + y     return z }</pre>		<pre>func sum (x, y int) int {     return x + y }</pre>
--	--	---



# Возвращение нескольких значений

- ✓ Функции могут возвращать несколько значений.
- ✓ После круглых скобок() указывается список типов возвращаемых значений.
- ✓ После return несколько значений через запятую (,).

Пример:

```
func values() (string, int) {  
    // здесь могут еще быть действия  
    return "Dima", 31  
}
```

```
func main() {  
    name, age := values()  
}
```

Также прием любого значения можно игнорировать с помощью \_

```
name, _ := values()
```





Примеры на практике

# Функции с переменным количеством параметров

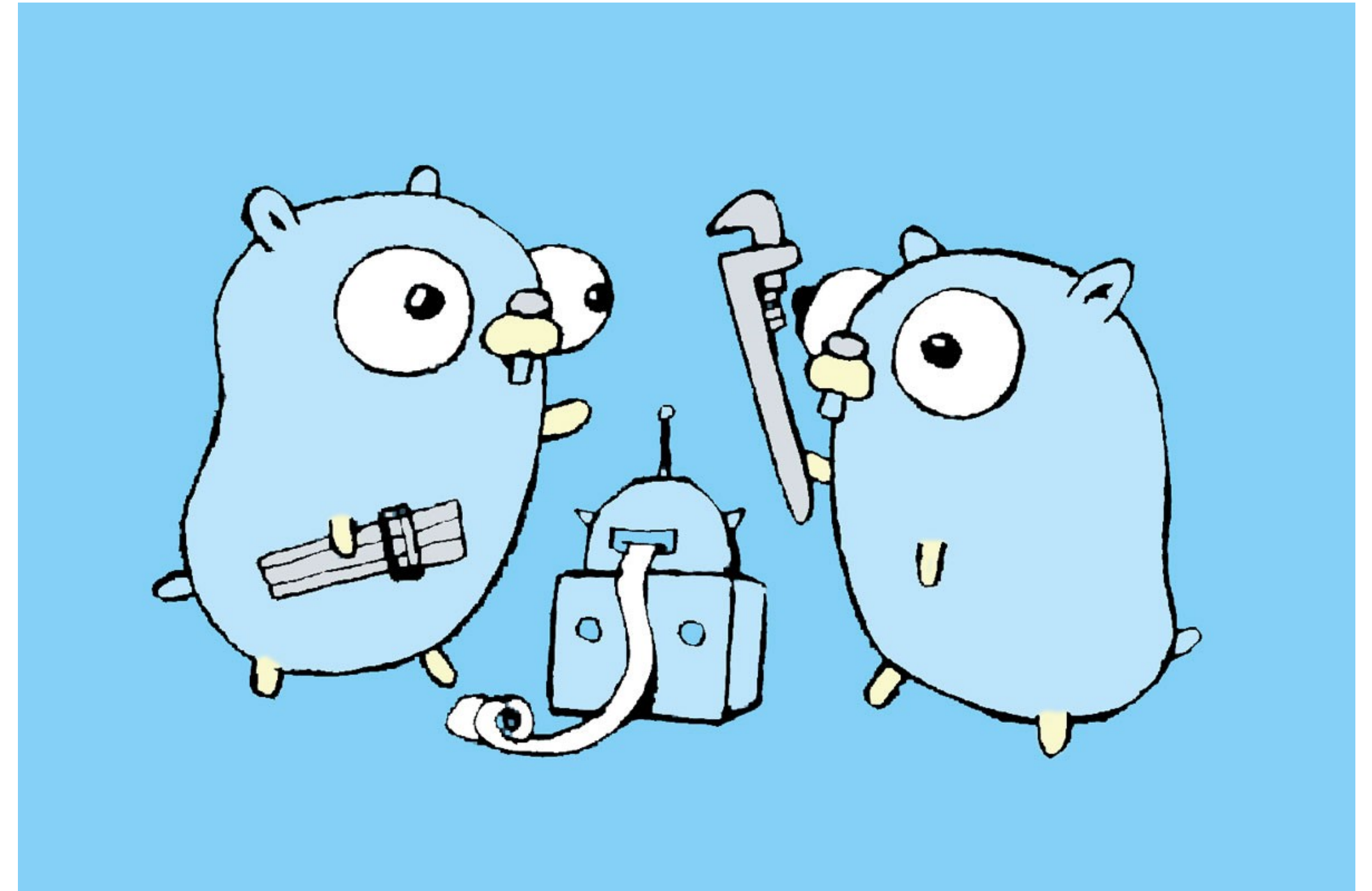
✓ Осуществляется с помощью троеточия (...).

Пример:

```
func sumOfMany(numbers ...int) int {  
    var sum int  
    for _, number := range numbers {  
        sum += number  
    }  
    return sum  
}  
  
func main() {  
    fmt.Println(sumOfMany(1,2)) // Вывод: 2  
    fmt.Println(sumOfMany(1,2,3)) // Вывод: 6  
}
```

Пример из стандартной библиотеки go:

```
func Scanf(format string, a ...interface{}) (n int, err error) {  
    return Fscanf(os.Stdin, format, a...)  
}
```



# Анонимные функции

- ✓ Анонимные функции — функции без имени
- ✓ Могут определяться внутри других функций
- ✓ Позволяют определить действия там, где они применяются

Пример:

```
package main
```

```
import "fmt"
```

```
func main() {  
    sum := func(number1, number2 int) int{ return number1 + number2}  
    fmt.Println(sum(3, 4))    // Вывод: 7  
    fmt.Println(sum(6, 7))    // Вывод:13  
}
```

- ✓ Могут получать значения сразу после своего объявления.

```
func main() {  
    sum := func(number1, number2 int) int {return number1+number2}(3,4)  
    fmt.Println(sum) // Вывод: 7  
}
```

# Анонимные функции, как аргумент функции

```
package main

import "fmt"

func math(number1, number2 int, operation func(int, int) int){
    result := operation(number1, number2)
    fmt.Println(result)
}

func main() {
    math(30, 15, func (x int, y int) int { return x - y }) // Вывод: 15
    math(30, 6, func (x int, y int) int { return x / y }) // Вывод: 5
}
```



# Анонимные функции, как результат функции

```
package main
```

```
import "fmt"
```

```
func superMath(operation string) func(int, int) int {  
    switch operation {  
        case "-": return func(number1, number2 int) int { return number1 - number2 }  
        case "*": return func(number1, number2 int) int { return number1 * number2 }  
        case "/": return func(number1, number2 int) int { return number1 / number2 }  
        default: return func(number1, number2 int) int { return number1 + number2 }  
    }  
}
```

```
func main() {  
    math := superMath("+")  
    fmt.Println(math(3, 4)) // Вывод: 7  
    math = superMath("-")  
    fmt.Println(math(30, 4)) // Вывод: 26  
}
```



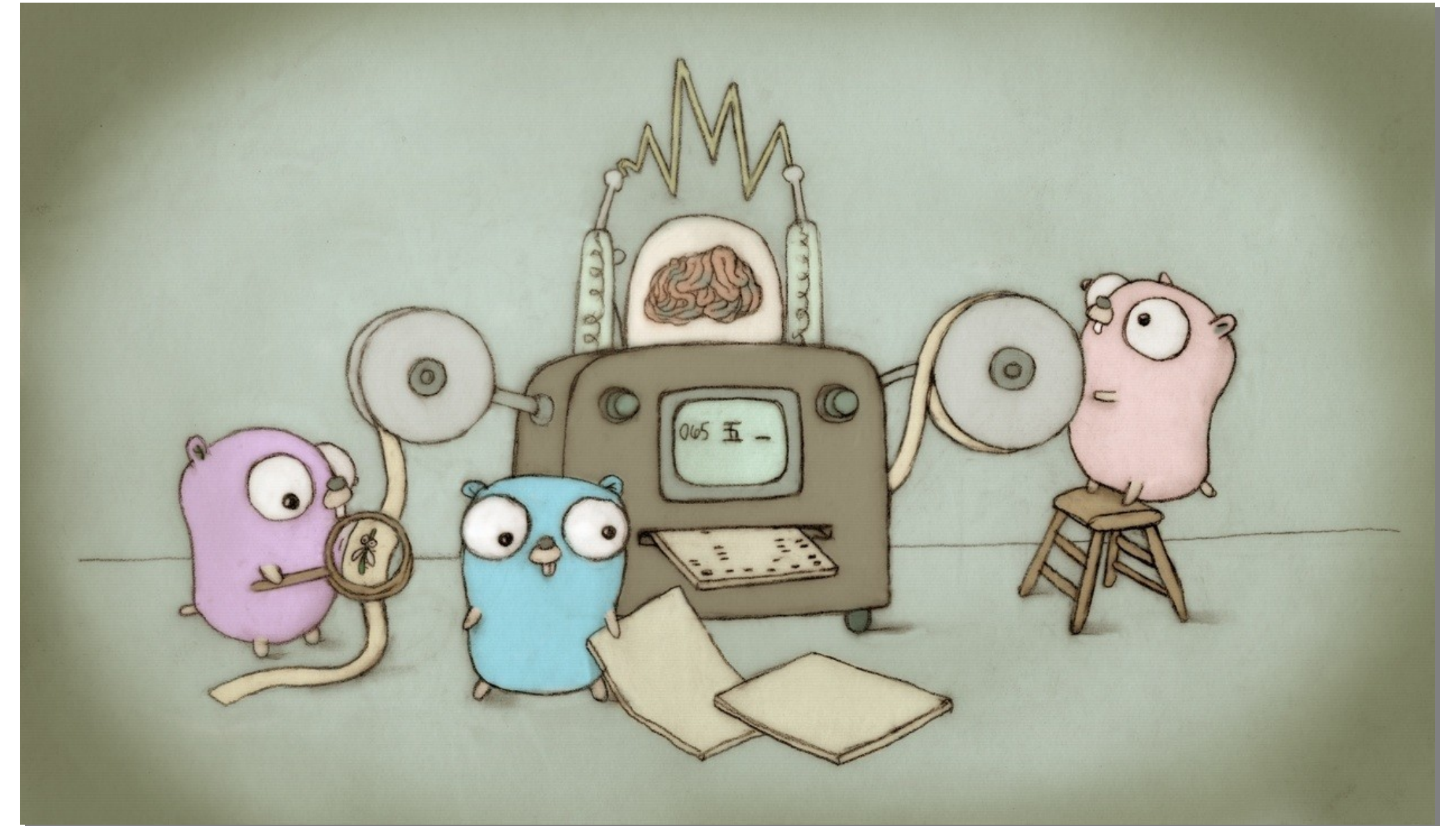
# Рекурсивные функции

```
package main

import "fmt"

func factorial(number int) int {
    if number == 0 {
        return 1
    }
    return number * factorial(number-1)
}

func main() {
    fmt.Println(factorial(14)) // Вывод: 87178291200
}
```



Примеры на практике



Указатель — это объект, который указывает на адрес другой переменной в памяти компьютера.

Для использования указателей используют:

& (амперсанд) — указатель на ячейку памяти и

\* (звездочка) — указатель на значение в ячейке памяти.

Пример с &:

```
number := 3000
```

```
numberPtr := &number
```

```
fmt.Println(numberPtr) // Вывод: 0xc000100010
```

Пример со \*:

```
number := 3000
```

```
numberPtr := &number
```

```
*numberPtr += 1000
```

```
fmt.Println(number)
```

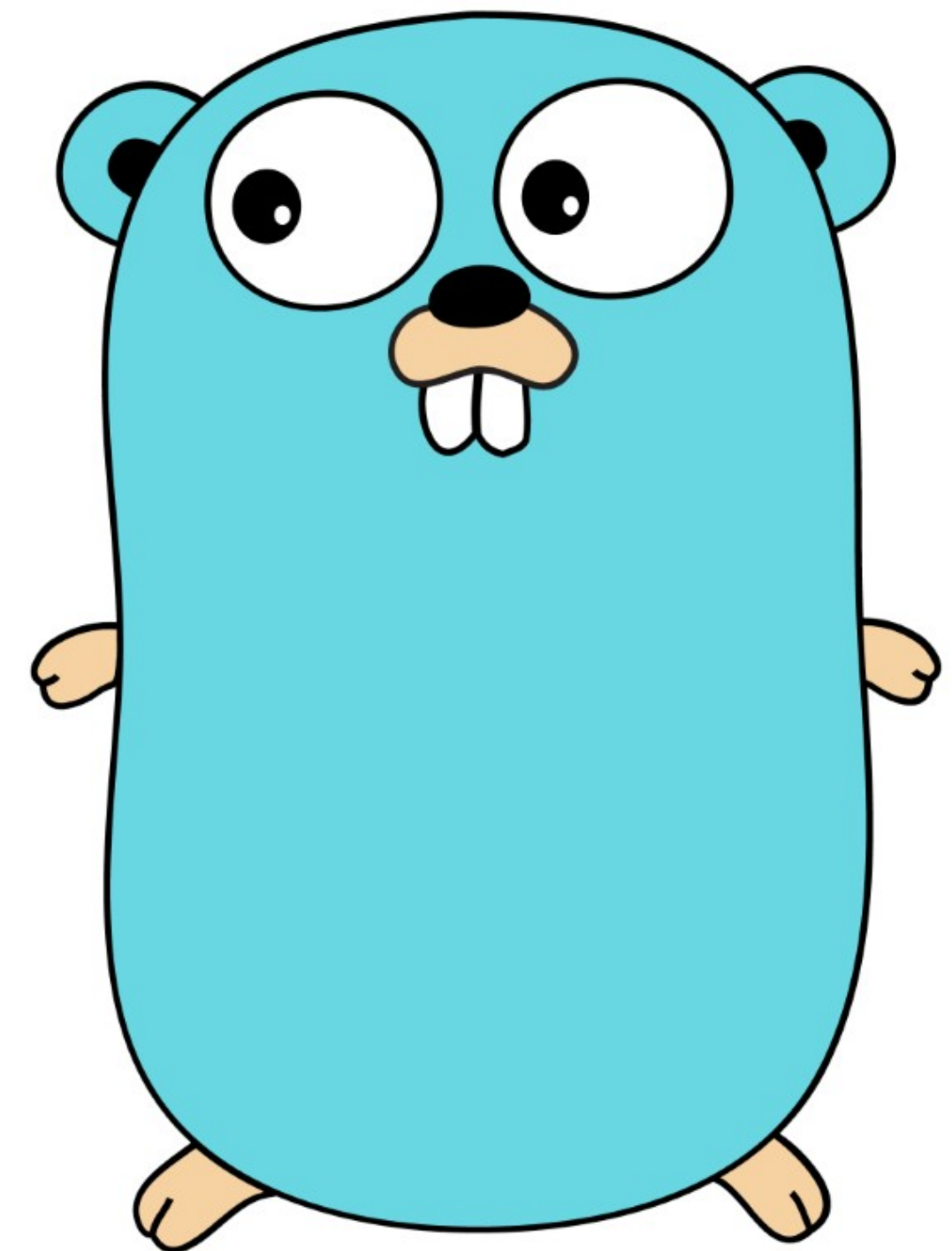
```
fmt.Printf("%T \n", number)
```

```
fmt.Printf("%T \n", numberPtr)
```

```
// Вывод: 4000
```

```
int
```

```
*int
```



# Функция new

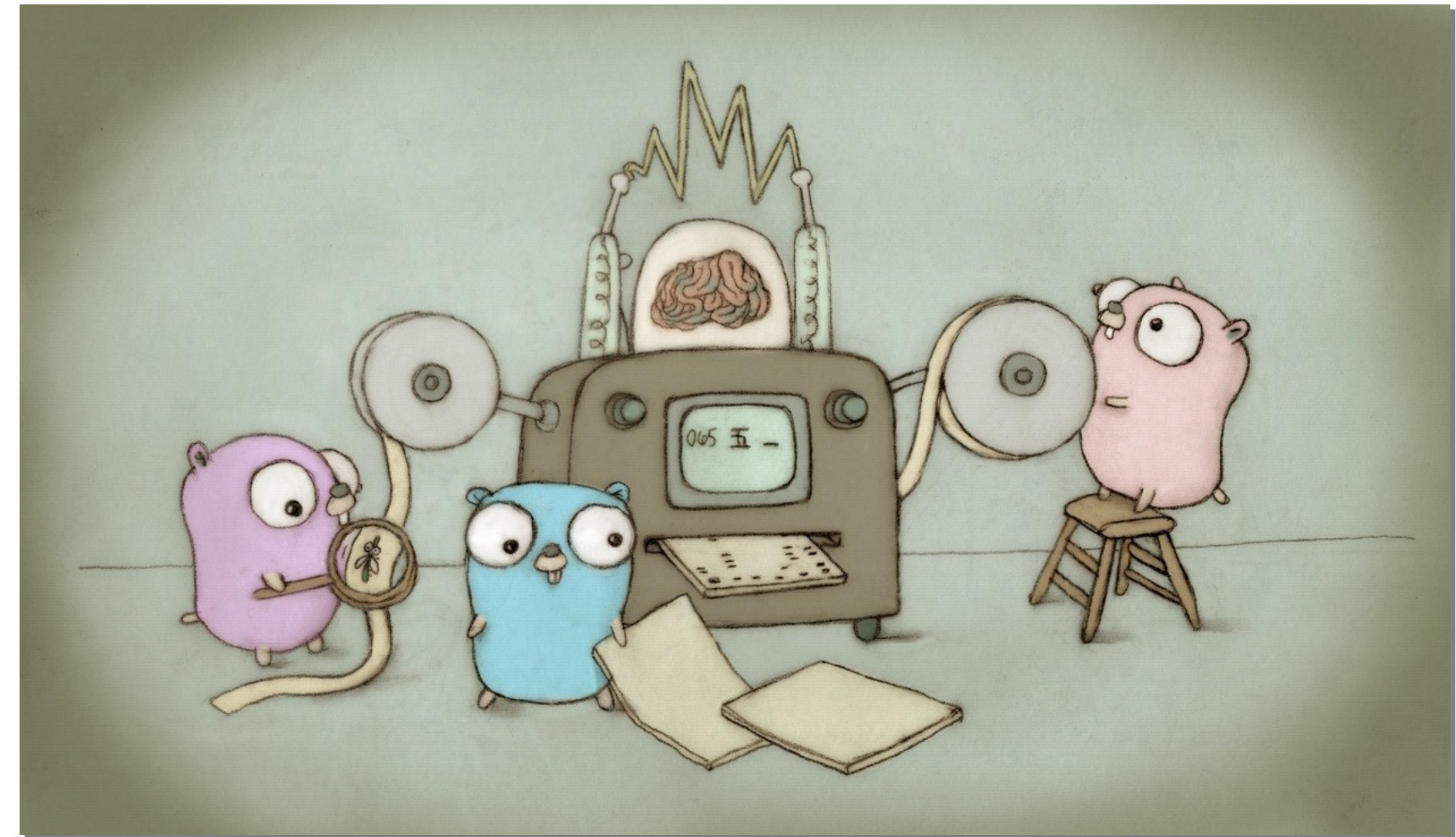
Функция new принимает аргументом тип, выделяет для него память и возвращает указатель на эту память.

Пример:

```
func two(xPtr *int) {  
    *numPtr = 2  
}
```

```
func main() {  
    numPtr := new(int)  
    two(numPtr)  
    fmt.Println(*numPtr) // Вывод: 2  
}
```

- ✓ Нет разницы между & и new
- ✓ Область памяти автоматически очищается
- ✓ Указатели редко используют для базовых типов, чаще и полезнее использовать для структур (следующий урок)



# Практическая часть

На следующем занятии:

- 1) Структуры и интерфейсы
- 2) Конкурирование или многопоточность?



Задание здесь:  
<https://bit.ly/gostudy4>





# Благодарю за внимание!!!

