Golang для начинающих

Клестов Дима,
Golang-разработчик ООО «Инносети»

https://t.me/MaJloe_3Jlo
https://github.com/MaJloe3Jlo

Результаты занятия 2

✓ Тестовые вопросы +

✗ Задача: Напишите программу...
i = i+5
i = i*3 - это не решение задачи!
Правильно: полный листинг программы или ссылка на play.golang.org

✗ Задача: Напишите программу, которая выводит последнюю цифру введенного 3хзначного числа
Вывести цифру, а не символ строки. Конвертация (пакет strconv) и использование типа string это неправильно.
Правильно: использование int и остаток от деления (%) на 10

✗ Задача с суммой 3хпервых и последних чисел — те же ошибки со string

Плюсы:
✓ Знание и использование стандартной библиотеки
✓ Использование циклов
✓ Использование функций
✓ Указатели

https://play.golang.org/

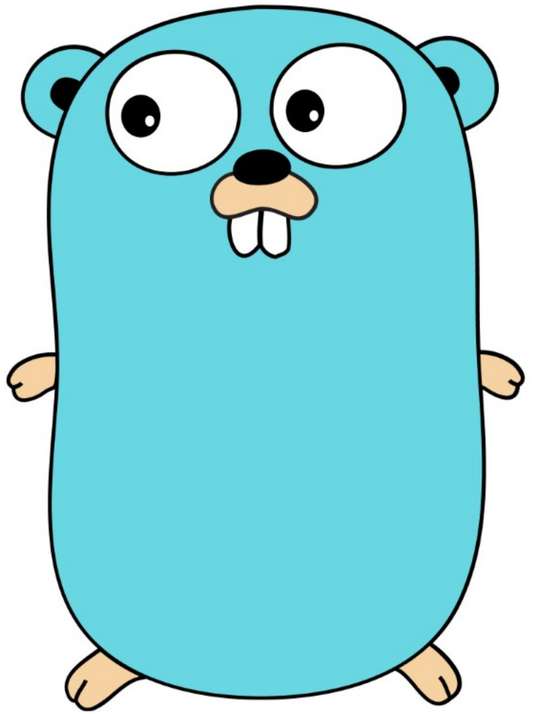
План занятия

На прошлом занятии:
1) Типы данных в Go
2) Переменные, константы и операции с ними
3) Области видимости в Go
4) Домашняя работа

План занятия на сегодня:
1) Управляющие конструкции
2) Коллекции данных
3) Домашняя работа

Управляющие конструкции

- ✔ If
- ✔ switch
- ✔ for



Управляющая конструкция if

Оператор if принимает выражение и проверяет его на истинность или ложность, и на основе этого выполняет те или иные действия.

Общий вид:

```
if  <условие>  {
    <блок действий, если условие верно>
}
```

Пример:

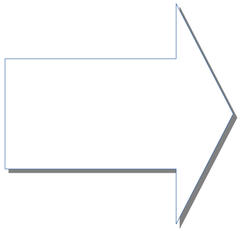
```
if  number % 2 == 0  {
    fmt.Println(“Четное“)
}
```

Условные выражения		
==	Равно	2 == 2
>	Больше чем	2 > 1
<	Меньше чем	2 < 3
<=	Меньше или равно	V % 2 <=0
>=	Больше или равно	V % 2 >=0
!=	Не равно	3 != 2

Управляющая конструкция if else

Пример:

```
if   number % 2 == 0  {
    fmt.Println(“Четное“)
}
if   number % 2 != 0 {
    fmt.Println(“Нечетное“)
}
```



Пример:

```
if   number % 2 == 0  {
    fmt.Println(“Четное“)
} else {
    fmt.Println(“Нечетное“)
}
```

Общий вид:

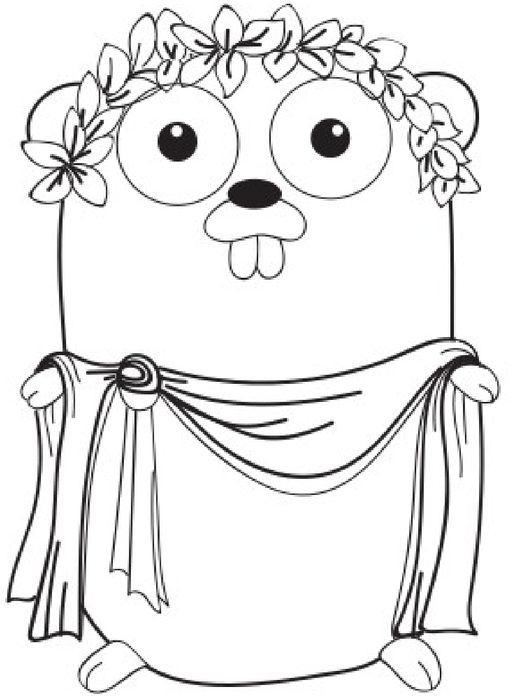
```
if   <условие>   {
    <блок действий, если  условие верно>
} else {
    <блок действий, если  условие неверно>
}
```

Управляющая конструкция if else if

Задача:
Необходимо сравнить 2 числа X и Y, X может быть больше Y, Y может быть больше X, а еще может быть что X и Y равны

Решение:

```
if x < y {
      fmt.Println("x меньше y")
} else if x > y {
      fmt.Println("x больше y")
} else {
    fmt.Println("x равно y")
}
```



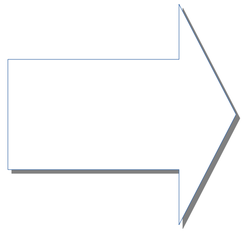
Общий вид:

```
if <условие 1> {
   <блок действие, если условие 1 верно>
}  else if <условие 2> {
   <блок действие, если условие 1 ложно, а условие 2 верно>
} else {
   <блок действие, если условие 2 ложно>
}
```

Управляющая конструкция switch

Пример:

```
if i == 1 {
    fmt.Println("Понедельник")
} else if i == 2 {
    fmt.Println("Вторник")
} else if i == 3 {
    fmt.Println("Среда")
} else if i == 4 {
    fmt.Println("Четверг")
} else if i == 5 {
    fmt.Println("Пятница")
} else if i == 6 {
    fmt.Println("Суббота")
} else if i == 7 {
    fmt.Println("Воскресенье")
} else {
    fmt.Println("Дня недели не
существует")
}
```



Пример:

```
switch i {
    case 1: fmt.Println("Понедельник")
    case 2: fmt.Println("Вторник")
    case 3: fmt.Println("Среда")
    case 4: fmt.Println("Четверг")
    case 5: fmt.Println("Пятница")
    case 6: fmt.Println("Суббота")
    case 7: fmt.Println("Воскресенье")
    default: fmt.Println("Дня недели не
существует")
}
```

break и fallthrough

- Нет необходимости заканчивать каждый блок case служебным словом break
- Если необходимо, чтобы ход программы «провалился» в следующий case, можно использовать ключевое слово fallthrough

Пример:

```
age := 31
switch age {
case 16:
      fmt.Println(16)
       fallthrough
case 31:
       fmt.Println(31)
       fallthrough
case 100:
       fmt.Println(100)
       fallthrough
default:
    fmt.Println("NaN")
}
```

Еще один switch

Пример:

```
var month int
fmt.Scan(&month)
switch {
case 1 <= month && month <= 3:
   fmt.Println(“I квартал года“)
case 4 <= month && month <= 6:
   fmt.Println(“II квартал года“)
case 7 <= month && month <= 9:
   fmt.Println(“III квартал года“)
case 10 <= month && month <= 12:
   fmt.Println(“IV квартал года“)
default:
   fmt.Println(“Такого не существует“)
}
```

Общий вид:

```
switch  {
   case <условий1>: <блок действий 1>
   ...
   case <условие n>: <блок действий n>
   default: <блок действий если ни одно условие не выполняется>
}
```

Цикл for

Общий вид:

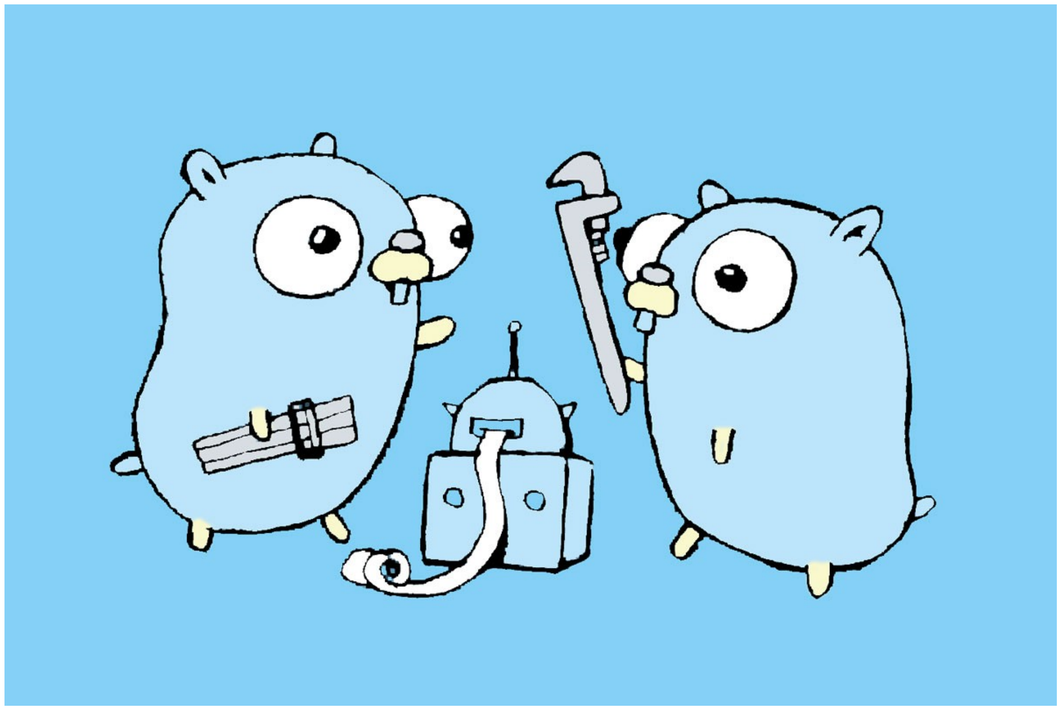
```
for <инициализация счетчика>; <условие>; <изменение счетчика>{
        // действия
   }
```

Примеры:

1)
```
for i := 1; i <= 100; i++ {
    fmt.Println(i)
}
```

2)
```
i := 1
for ; i <= 100; i++ {
    fmt.Println(i)
}
```

3)
```
i := 1
for ; i <= 100;  {
    fmt.Println(i)
    i++
}
```



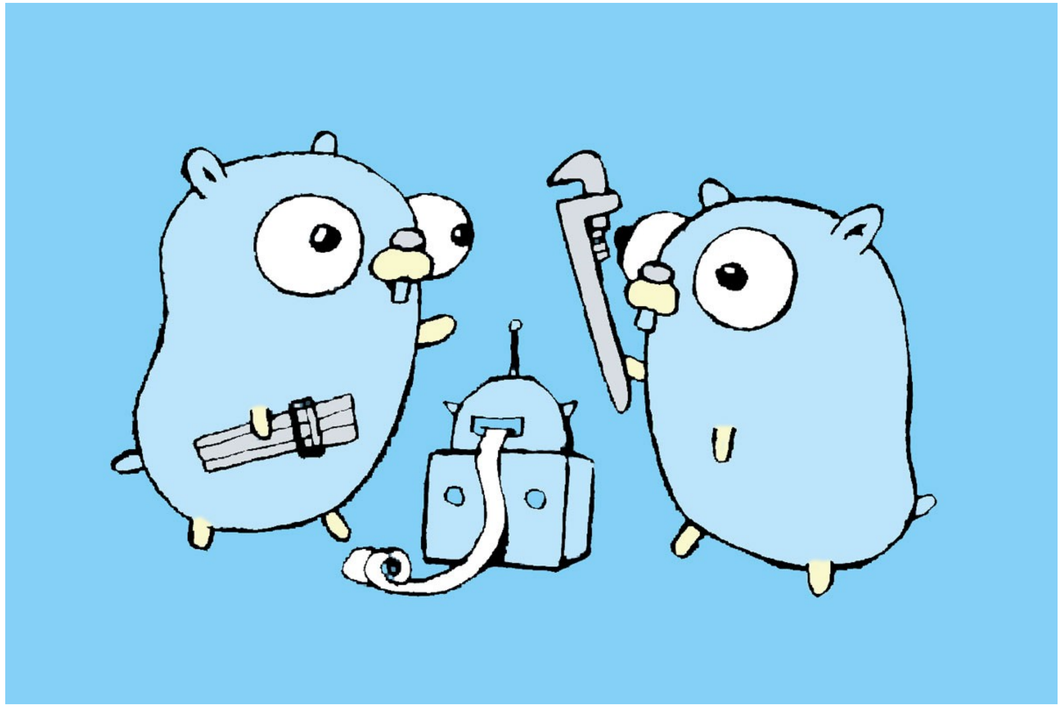
Цикл for бесконечный

Общий вид:

```
for {
    // действия
    // не забудьте добавить break, иначе программа зациклится.
}
```

Пример:

```
i := 10
for {
    i++
    if i == 100 {
        break
    }
}
```



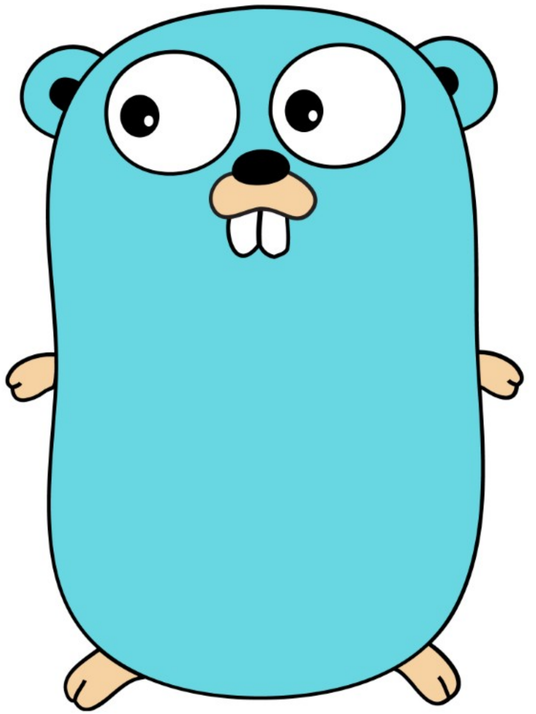
Цикл for для многократного ввода. Пример:

```
var n int
// считываем числа пока число не будет четным
for fmt.Scan(&n); n%2 != 0; fmt.Scan(&n) {
    fmt.Println(n)
}
```

Примеры на практике

Коллекции данных

- ✓Массивы
- ✓Срезы (слайсы)
- ✓Карты (мапы, хеш-таблицы)



Массивы в Go

Массив — это последовательность элементов одного типа данных, фиксированной длины.

Пример объявления пустого массива:

```
var numbers [5]int
...
fmt.Println(numbers) // Выведет: [0 0 0 0 0]
```

Пример объявления со значениями:

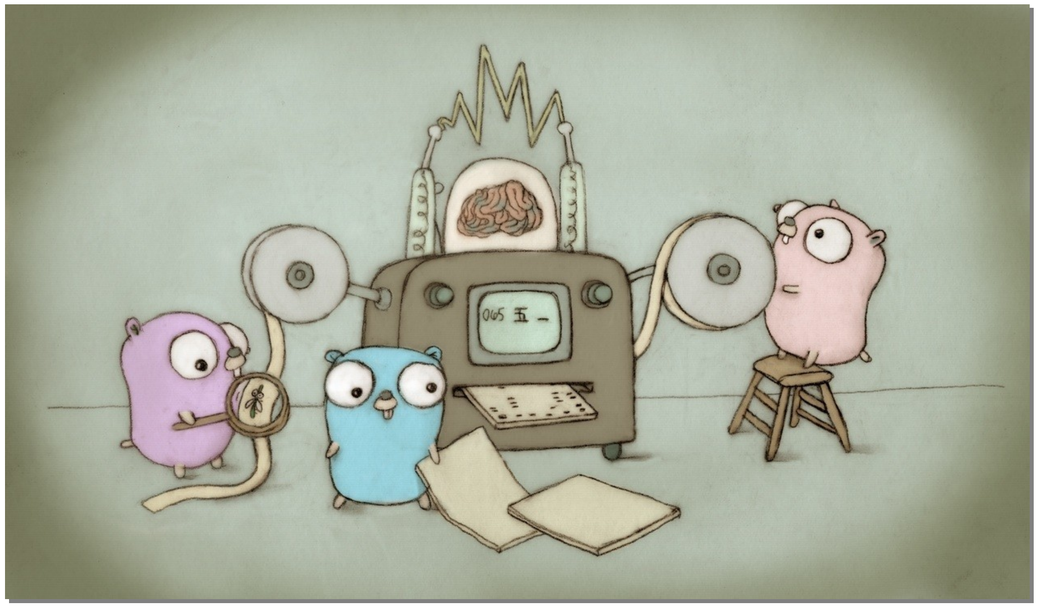
```
var numbers [6]int = [6]int{1, 2, 3, 4, 5, 6}
                или
numbers := [6]int{1,2,3,4,5,6}
```

Обращение к элементам массива. Индексы

- Индексы - номера элементов.
- При этом нумерация элементов массива начинается с нуля (первый элемент - индекс 0).
- Индекс указывается в квадратных скобках.

Пример обращения по индексам:

```
var numbers [5]int = [5]int{1,2,3,4,5}
...
fmt.Println(numbers[0]) // 1
fmt.Println(numbers[4])  // 5
numbers[0] = 87
fmt.Println(numbers[0]) // 87
```



Перебор массива с помощью for

- len — возвращает длину массива

Пример перебора по индексам:

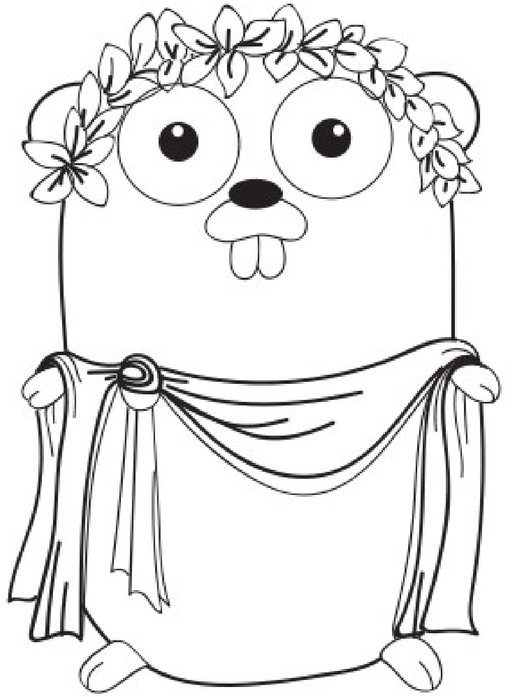
```
var numbers [5]int = [5]int{1,2,3,4,5}
for i := 0; i < len(numbers); i++ {
	fmt.Println(numbers[i])
}
```

Пример перебора с помощью range:

```
var numbers [5]int = [5]int{1,2,3,4,5}
  for index, element := range numbers {
	fmt.Printf(“Индекс %d, элемент = %d \n“, index, element)
  }
```

Вывод:

```
Индекс 0, элемент = 1
Индекс 1, элемент = 2
Индекс 2, элемент = 3
Индекс 3, элемент = 4
Индекс 4, элемент = 5
```



Перебор массива с помощью for

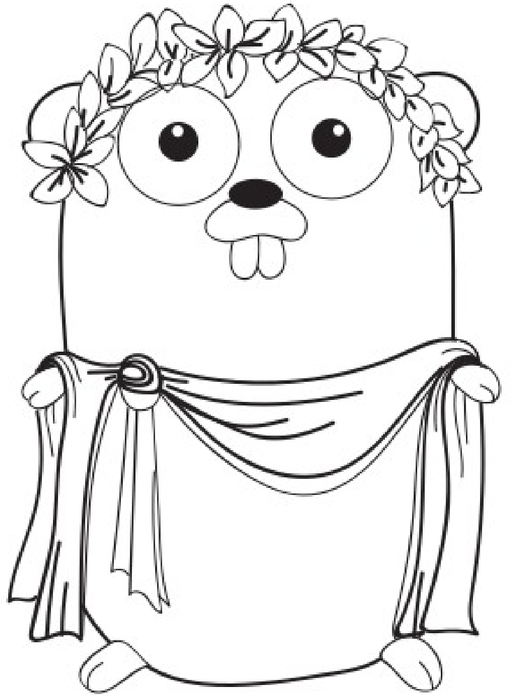
- Вместо любого неиспользуемого имени переменной можно использовать символ _

Примеры:

```
var numbers [5]int = [5]int{1,2,3,4,5}
   for index := range numbers {
      fmt.Printf(“Индекс %d \n“, index)
   }

   for index, _ := range numbers {
      fmt.Printf(“Индекс %d \n“, index)
   }

   for _, element := range numbers {
      fmt.Printf(“Элемент = %d \n“, element)
   }
```

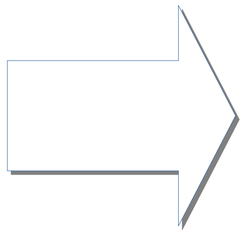


Перебор массива с помощью for

- В качестве второго значения range возвращает копию элемента массива — это важно если хотим изменить значение в массиве

Пример:

```
var numbers [5]int = [5]int{1,2,3,4,5}
for _, element := range numbers {
    element = 200
    fmt.Println(element)
    // Вывод будет
    // 200
    // 200
    // 200
    // 200
    // 200
}
fmt.Println(numbers)
// НО вывод будет [1 2 3 4 5]
```



```
// Для изменения используйте индекс
    for index := range numbers {
        numbers[index] = 200
        fmt.Println(numbers[index])
        // Вывод будет
        // 200
        // 200
        // 200
        // 200
        // 200
    }
    fmt.Println(numbers)
    // Вывод будет [200 200 200 200 200]
```

Срезы

- Срезы (slice) представляют последовательность элементов одного типа переменной длины. В отличие от массивов длина в срезах не фиксирована и динамически может меняться.

- Массивы и срезы тесно связаны. Срез — это коллекция данных, которая предоставляет доступ к подпоследовательности элементов базового массива.

- Срез состоит из трех компонентов: указателя, длины и емкости:
  - Указатель указывает на первый элемент массива, доступный через срез (который не обязательно совпадает с первым элементом массива);
  - Длина (length) — это количество элементов среза; - функция len()
  - Емкость (capacity) - количество элементов между началом среза и концом базового массива. - функция cap()

Срезы

Срез определяется также, как и массив, за тем исключением, что у него не указывается длина.

Пример объявления пустого среза:

```
var names []string
...
fmt.Println(names) // Выведет: []
```

Примеры объявления со значениями:

```
var numbers []int = []int{1,2,3} // Вывод: [1 2 3]
anotherNumbers := []int{4,5,6} // Вывод: [4 5 6]
xNumber := []int{1: 13} // Вывод: [0 13]
```

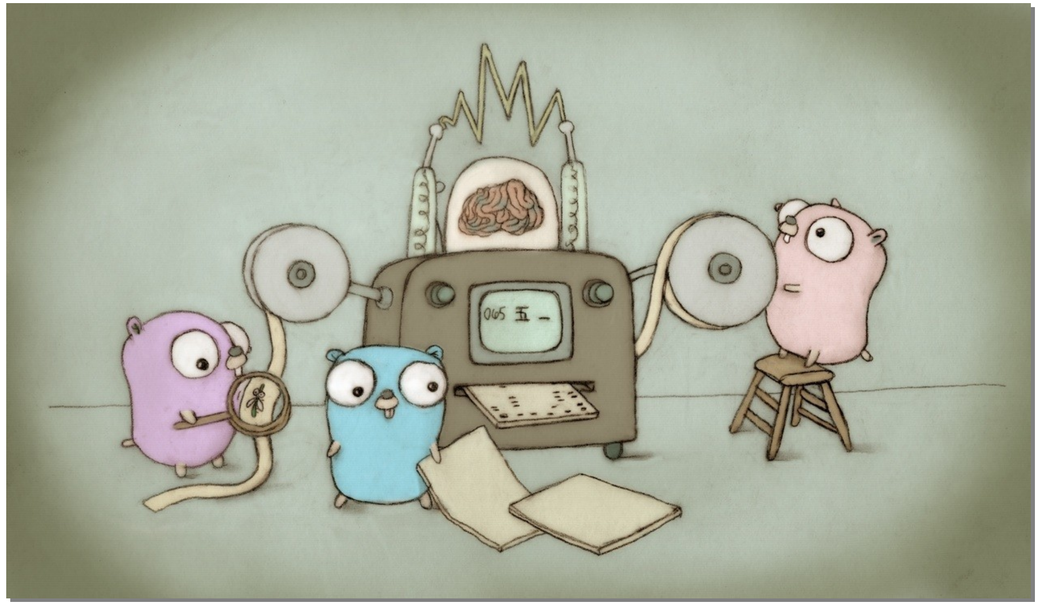
Функция make и оператор среза

✔make — создание пустого среза нужной длины, и вместительности

Общий вид:
make([]T, lenght, capacity)

Пример объявления среза с помощью make:

```
numbers := make([]int, 5, 5)
fmt.Println(numbers)
// Выведет: [0 0 0 0 0]
```



Оператор среза a[i:j] создает из последовательности a новый срез, который содержит элементы последовательности a с i по j-1

```
users := [8]string{"Dima", "Kolya", "Kate", "Julie",
"Tom", "Paul", "Mike", "Robert"}
// базовый массив

users1 := users[2:6] // с 3-го по 6-й
users2 := users[:4] // с 1-го по 4-й
users3 := users[3:] // с 4-го до конца

fmt.Println(users1, len(users1), cap(users1))
// [Kate Julie Tom Paul] 4 6
fmt.Println(users2, len(users2), cap(users2))
// [Dima Kolya Kate Julie] 4 8
fmt.Println(users3, len(users3), cap(users3))
// [Julie Tom Paul Mike Robert] 5 5
```

Функции append и copy

✔append — добавление элементов в срез

Общий вид:
append(slice []Type, elems ...Type)

Пример:
```
a := []int{1, 2, 3}
a = append(a, 4, 5)
fmt.Println(a) // [1 2 3 4 5]
```

Удаление элемента с помощью append:
```
names := []string{“Dima“, “Nick“, “Kate“}
names = append(names[0:1], names[2:]...)
fmt.Println(names) // [Dima Kate]
```

✔copy — копирование элементов в среза

Общий вид:
copy(dst, src)

Пример:
```
a := []int{1, 2, 3}
b := make([]int, 3, 3)
copy(b, a)
fmt.Printf("a = %v \n", a)
// a = [1 2 3]
fmt.Printf("b = %v \n", b)
// b = [1 2 3]
```

- При копировании слайсов длина должна быть одинаковой.

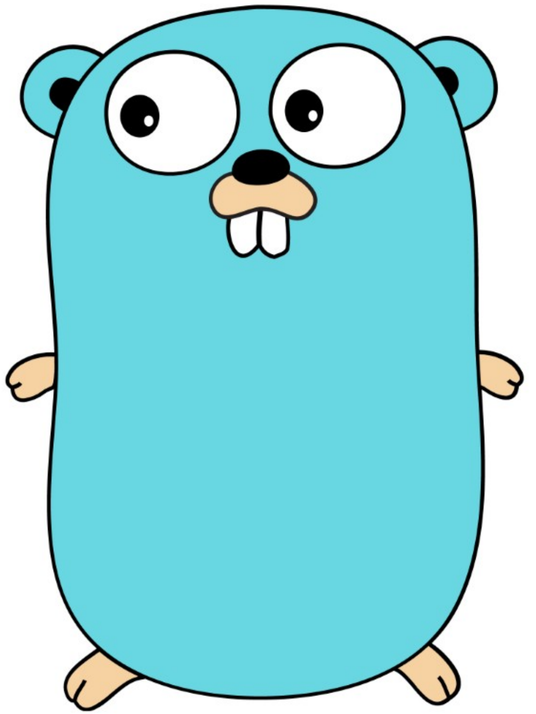
Перебор среза с помощью for

```
var numbers []int = []int{1,2,3,4,5}
for index, element := range numbers {
    fmt.Printf(“Индекс %d, элемент = %d \n“, index, element)
}

for index := range numbers {
    fmt.Printf(“Индекс %d \n“, index)
}

for index, _ := range numbers {
    fmt.Printf(“Индекс %d \n“, index)
}

for _, element := range numbers {
    fmt.Printf(“Элемент = %d \n“, element)
}
```



Карта (map), словарь, хеш-таблица

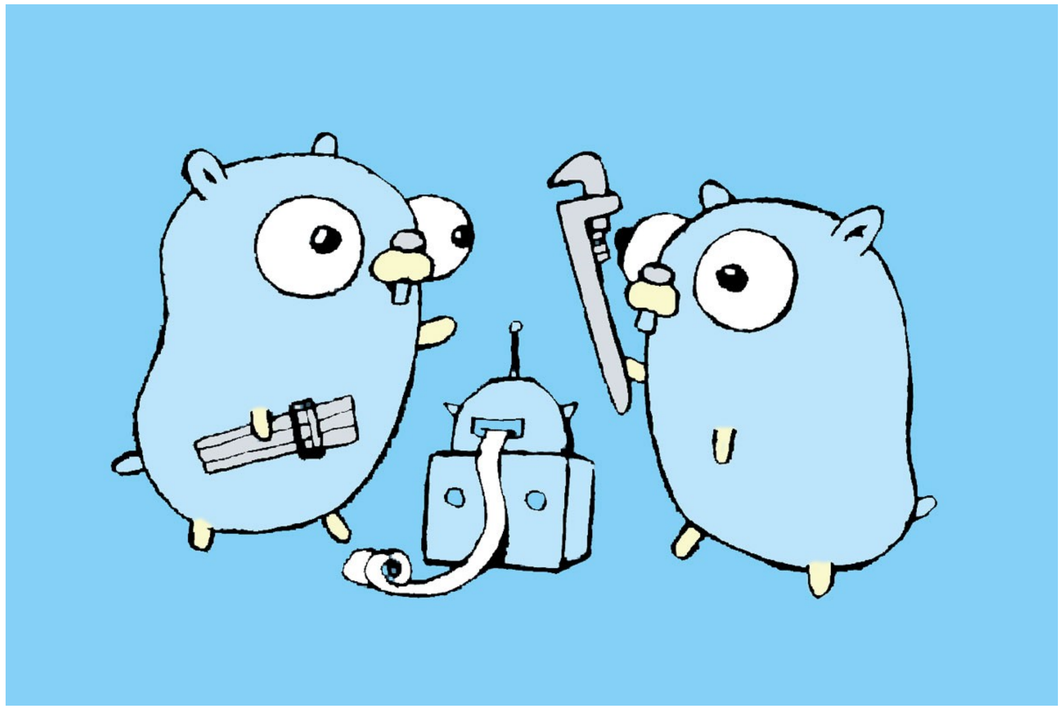
Карта - структура данных, неупорядоченная коллекция пар "ключ-значение", в которой все ключи различны, а значение, связанное с заданным ключом, можно получить, обновить или удалить.

Создание с помощью var, make и короткой формы:

```
var numbers map[int]int = map[int]int{
    1: 1,
}
fmt.Println(numbers[1]) // Вывод: 1

names := make(map[int]string)
names[1] = "Dima"
fmt.Println(names[1]) // Вывод: Dima

dictionary := map[string]string{
    "Greeting": "Hello",
}
fmt.Println(dictionary["Greeting"]) // Вывод: Hello
```



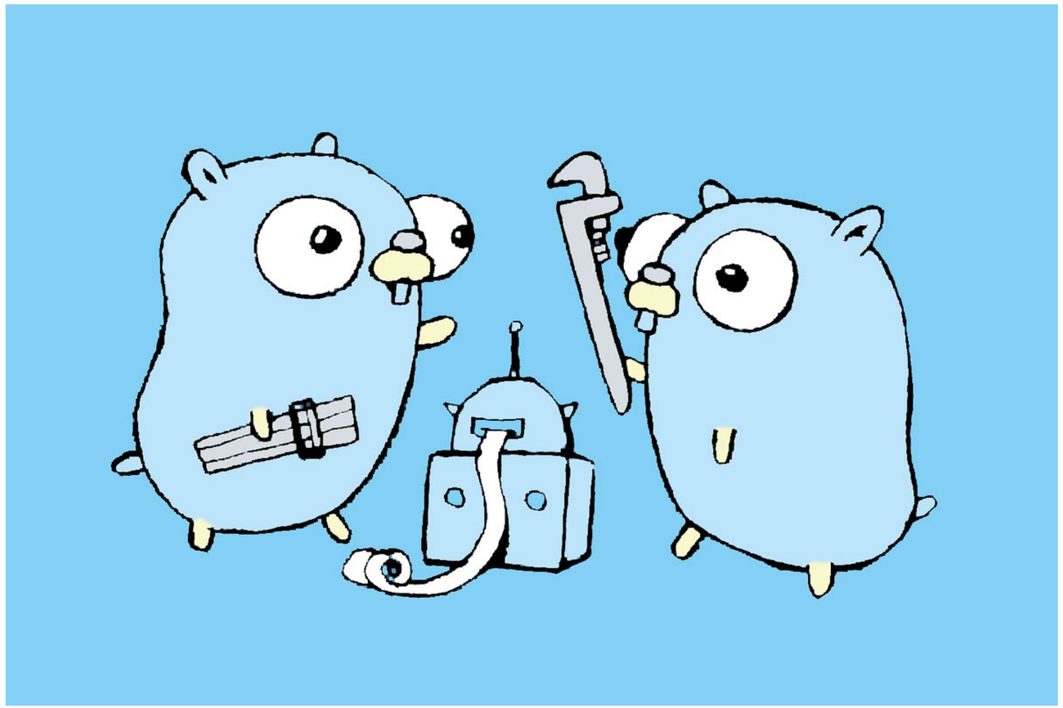
Функция delete и наличие элемента в карте

Функция delete:

```
m := map[int]int{
        1: 2,
        3: 4,
}
delete(m, 3) // Удаление элемента по ключу 3
fmt.Println(m) // map[1:2]
```

Проверка наличия значения:

```
m := map[int]int{
        1: 2,
        3: 4,
}
if value, ok := m[1]; ok {
    fmt.Println(value, ok) // Вывод: 2 true
}
```



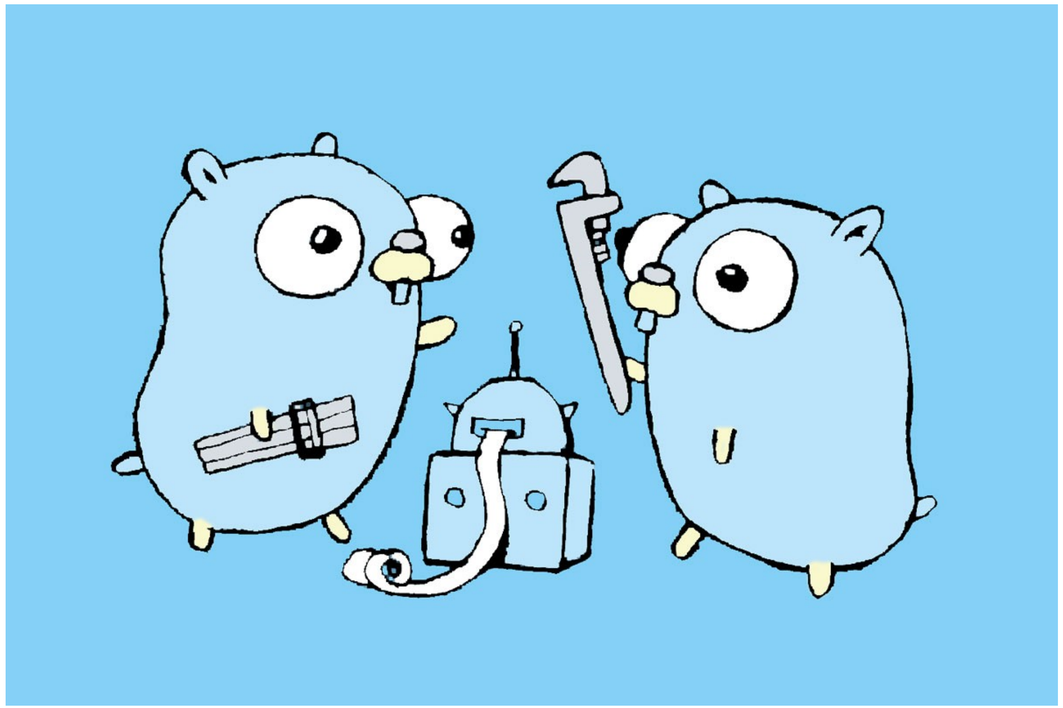
Перебор карты и ее длина

Перебор карты:

```
dictionary := map[string]string{
        "Greeting": "Hello",
        "Farewell": "Adios",
    }
    for key, value := range dictionary {
        fmt.Println(key, value)
    }
// Вывод:
Greeting Hello
Farewell Adios
```

Длина карты len:

```
m := map[int]int{
            1: 2,
            3: 4,
            5: 6,
}
fmt.Println(len(m)) // 3
```



Практическая часть

На следующем занятии:

1) Функции
2) Указатели

Задание здесь:
https://bit.ly/gostudy_3

Благодарю за внимание!!!