Основы языка программирования Go

Шифр: PR.GO.001.20240218

Цель: исследование назначения и способов установки *Go*, исследование типов данных, констант и арифметических операции языка программирования *Go*.

Ход работы

Go представляет компилируемый статически типизированный язык программирования от компании Google. Язык Go предназначен для создания различного рода приложений, но прежде всего это веб-сервисы и клиент-серверные приложения. Хотя также язык обладает возможностями по работе с графикой, низкоуровневыми возможностями и т.д.

Работа над языком Go началась в 2007 в недрах компании Google. Одним из авторов является Кен Томпсон, который, к слову, является и одним из авторов языка Си (наряду с Денисом Ритчи). 10 ноября 2009 года язык был анонсирован, а в марте 2012 года вышла версия 1.0. При этом язык продолжает развиваться. Текущей версией является версия 1.14, которая вышла в феврале 2020 года.

Язык Go развивается как open source, то есть представляет проект с открытым исходным кодом, и все его коды и компилятор можно найти и использовать бесплатно. Официальный сайт проекта - https://golang.org, где можно найти много полезной информации о языке.

Go является кроссплатформенным, он позволяет создавать программы под различные операционные системы - Windows, Mac OS, Linux, FreeBSD, Android и т.д. Код обладает переносимостью: программы, написанные для одной из этих операционных систем, могут быть легко с перекомпиляцией перенесены на другую ОС.

Зачем и почему изучать язык Go?

Если кратко то:

- Go перспективный язык, который быстро набирает популярность среди компаний любого рода

- Go не очень сложный, логичный и понятный язык
- Go нацелен на программистов и компьютеры XXI века
- Go быстрый язык, компиляция больших программ обычно быстрее чем у других языков

Подробнее:

Go проектировался с прицелом на эффективное масштабирование, благодаря чему его можно использовать для создания очень больших приложений и компиляции даже очень больших программ за секунды на единственном компьютере. Молниеносная скорость компиляции обеспечивается отчасти простотой синтаксического анализа программ на этом языке, но главным образом благодаря особенностям управления зависимостями.

Благодаря высокой скорости компиляции программ на языке Go появляется возможность использовать этот язык в областях, где обычно применяются языки сценариев.

Язык Go имеет очень простой и понятный синтаксис, в котором отсутствуют сложные и замысловатые конструкции, характерные для более старых языков, таких как C++ (появившегося в 1983 году) или Java (появившегося в 1995 году). И относится к категории языков со строгой статической типизацией, что многими программистами считается важным условием для разработки крупных программ. Однако система типов данных в языке Go не слишком обременительна благодаря поддержке синтаксиса объявления переменных одновременно с их инициализацией (когда компилятор определяет тип автоматически, избавляя от необходимости явно указывать его) и наличию мощного и удобного механизма динамической типизации.

Все сложности, связанные с учетом ресурсов в языке Go, берут на себя компилятор и среда выполнения. Для управления памятью в Go имеется механизм сборки мусора, что избавляет от необходимости использовать «интеллектуальные» указатели или освобождать память вручную. А поддержка параллелизма в языке Go реализована в форме механизма взаимодействующих последовательных процессов (Communicating Sequential Processes, CSP), основанного на идеях специалиста в области теории вычислительных машин и систем Чарльза Энтони Ричарда Хоара (C. A. R. Hoare) , благодаря которому во многих многопоточных программах на языке Go вообще отпадает необходимость блокировать доступ к ресурсам. Кроме того, в языке Go имеются так называемые go-подпрограммы (goroutines) – очень легковесные процессы, которых можно создать великое множество. Выполнение этих процессов автоматически будет распределяться по доступным процессорам и ядрам, что обеспечивает

возможность более тонкого деления программ на параллельно выполняющиеся задачи, чем это позволяют другие языки программирования, основанные на потоках выполнения. Фактически поддержка параллелизма в языке Go реализована настолько просто и естественно, что при переносе однопоточных программ на язык Go часто обнаруживается возможность параллельного выполнения нескольких задач, ведущая к увеличению скорости выполнения и более оптимальному использованию машинных ресурсов.

Go – практичный язык, где во главу угла поставлены эффективность программ и удобство программиста. Например, встроенные и определяемые пользователем типы данных в языке Go существенно отличаются – операции с первыми из них могут быть значительно оптимизированы, что невозможно для последних. В Go имеются также два встроенных фундаментальных типа коллекций: срезы (slices) (фактически ссылки на массивы переменной длины) и отображения (maps) (словари, или хеши пар ключ/значение). Коллекции этих типов высокооптимизированы и с успехом могут использоваться для решения самых разных задач. В языке Go также поддерживаются указатели (это действительно компилирующий язык программирования – в нем отсутствует какая-либо виртуальная машина, снижающая производительность), что позволяет с непринужденностью создавать собственные, весьма сложные типы данных, такие как сбалансированные двоичные деревья.

В то время как C поддерживает только процедурное программирование, а Java вынуждает программистов писать все программы в объектно-ориентированном стиле, Go позволяет использовать парадигму, наиболее подходящую для конкретной задачи. Go можно использовать как исключительно процедурный язык программирования, но он также обладает поддержкой объектно-ориентированного стиля программирования. **Однако**, как будет показано далее в курсе, реализация объектно-ориентированной парадигмы в Go радикально отличается от реализации этой же парадигмы в таких языках, как C++, Java или Python. Она намного проще в использовании и значительно гибче.

Совет

Искать информацию "гуглить" по Go нужно, вводя его полное название - Golang, чтобы видеть только нужную информацию, например: "как в golang создать переменную?" Запомните это, иначе вам будет сложно находить ответы на ваши проблемы и ошибки. А если вы более-менее знаете английский, то рекомендую гуглить сразу на нём: англоязычное сообщество Go все-таки больше ;)

В настоящее время Go находит широкое применение в различных сферах. В частности, среди известных компаний, которые применяют Go, можно найти следующие: Google, Яндекс, PayPal, American Express, Mail.ru Group, Dropbox, Netflix, Twitch, Soundcloud, Uber, CloudFlare и множество других. Поэтому этот прекрасный язык очень перспективен для изучения.

Что нужно для работы с Go?

Прежде всего необходим текстовый редактор для написания кода и компилятор для преобразования кода в исполняемый файл. Для начала скачайте и установите компилятор с официального сайта: https://golang.org/dl/

Некоторые полезные статьи по установке Go в зависимости от вашей операционной системы:

- **Как установить и настроить Go на Windows**
- **Установка Go 1.14.2 на MacOS**
- **Как установить Go на Ubuntu 18.04**
- **Как установить Go на Debian 10**
- **Как установить Golang на CentOS 7**

Для написания кода можно использовать специальные интегрированные среды разработки (IDE), которые поддерживают Go.

Самая удобная и многофункциональная IDE это **GoLand** (хотя новичкам и не обязательно использовать такие студии), маленькая проблема то что она платная, но есть несколько несложных способов использовать её бесплатно:

- Получить **бесплатно** лицензию для студентов, школьников и преподавателей: https://www.jetbrains.com/student/
- Получить **бесплатно** если вы решите более 3 задач по программированию на нашем курсе в течении 7 дней
- Использовать как есть так как 30 дней **бесплатно**

Другие инструменты:

- Visual Studio Code + плагин для Go
- Atom + плагин для Go
- Sublime Text + плагин для Go
- Lite IDE

Существуют плагины для **Go** для других IDE и редакторов: **IntelliJ IDEA**, Netbeans, Vim и других.

Также можно программировать прямо из браузера (но в этом случае вы будете ограничены возможностями по функционалу):

- Repl.it - Самый удобный с поддержкой терминала и полноценного ввода
- Поддерживается ввод данных: https://www.tutorialspoint.com/execute_golang_online.php
- Поддерживается ввод данных: https://ideone.com
- Нет ввода данных: https://play.golang.org

Так же можно программировать с помощью любого текстового редактора (в том числе в блокноте). Просто создайте файл с расширением *.go (например, main.go) и редактируйте файл как текст. Запускать можно через терминал/командную строку:

```
go run main.go
```

Команда выше только выполнит программу. Если же вы хотите скомпилировать программу, то-есть получить готовый бинарник для запуска на машинах без Go (для windows это будет .exe) - выполните следующую команду:

```
go build main.go
```

Первая программа

Традиционно первая программа, с которой начинается изучение любого языка программирования, называется «Hello World» — эта программа просто выводит в консоль строку `Hello World`. Давайте немного её изменим и напишем с помощью Go.

Наша первая программа будет выводить на консоль "Hello, Go!":

```
package main


import "fmt"

```

```
func main() {
    fmt.Println("Hello, Go!")
}
```

Теперь давайте разберёмся, что мы написали.

package main - объявили новый пакет, в любом проекте должен быть обязательно пакет main. Запуск программы начинается именно с этого пакета. Подробнее о пакетах мы поговорим в другом модуле.

Далее следует пустая строка. Go не обращает на подобные строки внимания, но мы используем их, чтобы облегчить себе чтение программы (вы можете удалить эту строку и убедиться, что программа ведет себя в точности как раньше).

import "fmt" - импортировали пакет ввода/вывода.

func main(){} - объявили функцию с названием "main". Имя `main` является особенным, эта функция будет вызываться сама при запуске программы. Эта функция тоже обязательная в программе, с неё начинает работать ваш код.

fmt.Println("Hello, Go!") - здесь мы выводим уже то, что нам нужно, используя встроенную функцию **Println()** из пакета **fmt**.

Если вам что-то непонятно на данном этапе - ничего страшного. Далее мы подробнее всё это рассмотрим, а сейчас вы просто смотрите, как выглядит простейшая программа на Go.

Типы переменных

Go — это язык программирования со статической типизацией. Это означает, что переменные всегда имеют определенный тип и этот тип нельзя изменить. Статическая типизация, на первый взгляд, может показаться неудобной. Вы потратите кучу времени только на попытки исправить ошибки, не позволяющие программе скомпилироваться. Однако знание типов данных заранее дает нам возможность понять, почему именно сломалось программа еще на этапе компиляции, и помогают избежать распространённых ошибок заранее.

В Go есть несколько встроенных типов данных, с которыми мы сейчас познакомимся.

Числа

В Go есть несколько различных типов для представления чисел. Вообще, мы разделим числа на два различных класса: целые числа и числа с плавающей точкой.

Целые числа

Целые числа, точно так же, как их математические коллеги, — это числа без дробной части. В отличие от десятичного представления чисел, которое используем мы, компьютеры используют двоичное представление.

Наша система строится на 10 различных цифрах. Когда мы исчерпываем доступные нам цифры, мы представляем большое число, используя новую цифру 2 (а затем 3, 4, 5, …) числа следуют одно за другим. Например, число, следующее за 9, это 10, число, следующее за 99, это 100 и так далее. Компьютеры делают то же самое, но они имеют только 2 цифры вместо 10. Поэтому, подсчет выглядит так: 0, 1, 10, 11, 100, 101, 110, 111 и так далее. Другое отличие между той системой счисления, что используем мы, и той, что использует компьютер - все типы чисел имеют строго определенный размер. У них есть ограниченное количество цифр. Поэтому четырехразрядное число может выглядеть так: 0000, 0001, 0010, 0011, 0100. В конце концов мы можем выйти за лимит, и большинство компьютеров просто вернутся к самому началу (что может стать причиной очень странного поведения программы).

В Go существуют следующие типы целых чисел: `uint8`, `uint16`, `uint32`, `uint64`, `int8`, `int16`, `int32` и `int64`. 8, 16, 32 и 64 говорит нам, сколько бит использует каждый тип. `uint` означает «unsigned integer» (беззнаковое целое), в то время как `int` означает «signed integer» (знаковое целое). Беззнаковое целое может принимать только положительные значения (или ноль).

uint8	**Беззнаковые 8-битные целые числа**	**от 0 до 255**
uint16	Беззнаковые 16-битные целые числа	от 0 до 65535
uint32	Беззнаковые 32-битные целые числа	от 0 до 4294967295
uint64	Беззнаковые 64-битные целые числа	от 0 до 18446744073709551615
int8	Знаковые 8-битные целые числа	от -128 до 127
int16	Знаковые 16-битные целые числа	от -32768 до 32767
int32	Знаковые 32-битные целые числа	от -2147483648 до 2147483647

uint8	**Беззнаковые 8-битные целые числа**	**от 0 до 255**
int64	Знаковые 64-битные целые числа	от -9223372036854775808 до 9223372036854775807

В дополнение к этому существуют два типа-псевдонима: `byte` (то же самое, что `uint8`) и `rune` (то же самое, что `int32`). Байты — очень распространенная единица измерения в компьютерах (1 байт = 8 бит, 1024 байта = 1 килобайт, 1024 килобайта = 1 мегабайт, …), и именно поэтому тип `byte` в Go часто используется для определения других типов.

Также существует 3 машинно-зависимых целочисленных типа: `uint`, `int` и `uintptr`. Они машинно-зависимы, потому что их размер зависит от архитектуры используемого компьютера:

- `int`: представляет целое число со знаком, которое в зависимости от платформы может занимать либо 4 байта, либо 8 байт. То есть соответствовать либо int32, либо int64.
- `uint`: представляет целое число только без знака, которое, аналогично типу int, в зависимости от платформы может занимать либо 4 байта, либо 8 байт. То есть соответствовать либо uint32, либо uint64.

В общем, если вы работаете с целыми числами — просто используйте тип `int`.

Числа с плавающей точкой

Числа с плавающей точкой — это числа, которые содержат вещественную часть (вещественные числа) (1.234, 123.4, 0.00001234). Их представление в компьютере довольно сложно и не особо необходимо для их использования. Так что мы просто должны помнить:

- Числа с плавающей точкой неточны. Бывают случаи, когда число вообще нельзя представить. Например, результатом вычисления `1.01 - 0.99` будет `0.020000000000000018` - число очень близкое к ожидаемому, но не то же самое.
- Как и целые числа, числа с плавающей точкой имеют определенный размер (32 бита или 64 бита). Использование большего размера увеличивает точность (сколько цифр мы можем использовать для вычисления)
- В дополнение к числам существуют несколько других значений, таких как: (`NaN`, для вещей наподобие `0/0`), а также положительная и отрицательная бесконечность (`+∞` и `-∞`).

В Go есть два вещественных типа: `float32` и `float64` (соответственно часто называемые вещественными числами с одинарной и двойной точностью), а также два дополнительных типа для представления комплексных чисел (чисел с мнимой частью): `complex64` и `complex128`. При работе с вещественными числами достаточно использовать `float32`, однако, если Вы хотите работать с более точными числами, можно использовать и `float64`.

Строки

Строка — это последовательность символов определенной длины, используемая для представления текста. Строки в Go состоят из независимых байтов, обычно по одному на каждый символ (символы из других языков, таких как китайский, представляются несколькими байтами).

Строковые литералы могут быть созданы с помощью двойных кавычек `"Hello World"` или с помощью обратных апострофов `Hello World`. Различие между ними в том, что строки в двойных кавычках не могут содержать новые строки и они позволяют использовать особые управляющие последовательности символов. Например, `\n` будет заменена символом новой строки, а `\t` - символом табуляции.

Распространенные операции над строками включают в себя нахождение количества байт строки `len("Hello World")`, доступ к отдельному символу в строке `"Hello World"[1]` (строки “индексируются” начиная с 0, а не с 1), и конкатенацию двух строк `"Hello " + "World"`.

Так как строки в Go хранятся в виде байтов нужно понимать что такой код не выведет символ:

```
package main

import "fmt"

func main() {
  fmt.Println("Hello Go"[0]) // вывод: 72
}
```

Представим байты в виде строки:

```
package main
```

```
import "fmt"

func main() {
        fmt.Println(string("Hello Go"[0])) // вывод: H
}
```

Логические типы

Булевский тип - **bool** (названный так в честь Джорджа Буля) — это специальный целочисленный тип, используемый для представления истинности и ложности. Переменная такого типа будет занимать только один байт. С этим типом используются три логических оператора:

Литерал	Пояснение
&&	И
||	ИЛИ
!	НЕ

Переменная типа bool может принимать только два значения: true или false.

Переменные

Для хранения данных в программе применяются переменные. Переменная представляет именованный участок в памяти, который может хранить некоторое значение. Для определения переменной применяется ключевое слово `var`, после которого идет имя переменной, а затем указывается ее тип:

```
var имя_переменной тип_данных
```

Имя переменной представляет произвольный идентификатор, который состоит из алфавитных и цифровых символов и символа подчеркивания. При этом первым символом должен быть либо алфавитный символ, либо символ подчеркивания. При этом имена не должны представлять одно из ключевых слов: `break`, `case`, `chan`, `const`, `continue`, `default`, `defer`, `else`, `fallthrough`, `for`, `func`, `go`, `goto`, `if`, `import`, `interface`, `map`, `package`, `range`, `return`, `select`, `struct`, `switch`, `type`, `var`.

Например, простейшее определение переменной:

```
var hello string
```

Данная переменная называется hello и она представляет тип string, то есть некоторую строку.

Можно одновременно объявить сразу несколько переменных через запятую:

```
var a, b, c string
```

Важно учитывать, что Go - регистрозависимый язык, то есть переменные с именами **hello** и **Hello** будут представлять собой разные переменные.

Еще примеры объявления переменных:

```
var x int
var y float64
var max, min int
```

Мы можем одновременно с объявлением переменной задать ей некоторое значение. Например:

```
var x int = 10
var c string = "Hello World!"
var z float64 = 1.045
```

Также допустимо присваивать значение переменной в момент ее объявления следующим образом:

```
var a = 12
var hello = "Hello"
```

В этом случае компилятор сможет сам определить тип присваиваемого значения.

Теперь мы можем объявить переменную типа string или int, присвоить ей любое значение, а затем вывести.

```
package main

import "fmt"

func main() {

    var hello string

    hello = "Hello Go!"

    var a int = 2019

    fmt.Println(hello)
    fmt.Println(a)
}
```

Для хранения символов можно использовать int32/rune. Здесь используются одинарные кавычки. Компилятор определяет код буквы в unicode и присваивает его переменной symbol. То есть мы не храним никакую 'c', а храним лишь число 99. Функция string() из переданного в него числа 99 делает строку 'c'.

```
var symbol int32 = 'c'
fmt.Println(string(symbol))
```

Значения по умолчанию

Когда объявляется переменная, она автоматически содержит значение по умолчанию для своего типа: `0` для `int`, `0.0` для `float`, `false` для `bool`, пустая строка для `string`, `nil` для указателя и т. д.

Также существует более краткий способ объявить переменную (краткое объявление доступно только внутри функций, но о них мы поговорим во 2 модуле):

```
a := 5
```

Это то же самое, что:

```
var a int = 5
```

Так же можно объявить вот так:

```
var a = 5
```

Также можно объявить сразу несколько переменных в одном блоке **var**:

```
package main

import "fmt"

func main() {
    var (
        name string = "Dima"
        age int = 23
    )

    fmt.Println(name)
    fmt.Println(age)
}
```

Вывод программы будет такой:

```
Dima
23
```

Арифметические операции

У переменных есть разные операции, как в алгебре.

`+` сложение

`-` вычитание

`*` умножение

`/` деление

Примеры:
(если вы не понимаете что значит // - это комментарии, вы можете прочитать про них в уроке 1.6)

```
a := 100
b := 10

c := a + b // c = 110

c = a * b  // c = 1000

c = a - b // c = 90

c = a / b  // c = 10
```

При делении стоит быть внимательным, так как если в операции участвуют два целых числа, то результат деления будет округляться до целого числа:

```
var a int = 10 / 6
------------------
Вывод: 1
```

Чтобы получить в результате деления вещественное число, как минимум первый операнд также должен представлять собой вещественное число и результат мы должны при этом тоже сохранять в переменную вещественного типа:

```
var m float32 = 10.0 / 6
----------------------
Вывод: 1.6666666
```

> ⓘ **Тут действует принцип:**
>
> Арифметические операторы применяются к числовым значениям и дают результат **того же типа, что и первый операнд.**

% Возвращает остаток от деления (в этой операции могут принимать участие **только целые числа**):

```
var c int = 10 % 3
---------------
Вывод: 1
```

Постфиксный инкремент (x++). Увеличивает значение переменной на единицу:

```
var a int = 1
a++
fmt.Println(a)
---------------
Вывод: 2
```

Постфиксный декремент (x--). Уменьшает значение переменной на единицу:

```
var a int = 10
a--
fmt.Println(a)
---------------
Вывод: 9
```

Чтение данных с консоли

Конечно же, работать с переменными достаточно интересно, но гораздо интереснее с ними работать, если их значения вводятся с консоли. Для того чтобы это сделать, нам нужно воспользоваться методом **fmt.Scan(&a)**, где **&a** - ссылка (более точно - адрес) на переменную a. Если проще, то введённое число запишется из консоли прямиком в эту переменную и там будет храниться, пока не понадобится её куда-нибудь пристроить/поменять.

Теперь давайте прочтём, например, имя и возраст:

```
package main

import "fmt"

func main() {
    var name string
    var age int
    fmt.Print("Введите имя: ")
    fmt.Scan(&name)
    fmt.Print("Введите возраст: ")
    fmt.Scan(&age)

    fmt.Println(name, age)
}
```

Программа сначала прочтёт имя, а затем запишет его в переменную name. Аналогично, введённый возраст запишется в переменную age. В конце программа

выведет эти переменные через пробел.

Также можно читать с консоли сразу несколько переменных:

```
fmt.Scan(&a, &b, &c)
```

Вывод данных на консоль

Для вывода данных на консоль мы на данном этапе будем пользоваться двумя методами, которые присутствуют в пакете `fmt`. Это Print() и Println().

Первый метод при выводе нескольких объектов вставляет между ними пробелы, если среди них нет строк.

Второй всегда ставит пробелы между выводимыми объектами, плюс добавляет новую строку. То есть он пригодится, если нам необходимо будет сделать вывод на нескольких строках.

Например:

```
fmt.Print("hello, world")
fmt.Print("hello, world")
// вывод будет в одну строку:
// hello, worldhello, world
```

Но если мы сделаем вот так:

```
fmt.Println("hello, world")
fmt.Print("hello, world")
// вывод будет в две строки:
// hello, world
// hello, world
```

Теперь как выводятся несколько объектов:

```
fmt.Print("Ivan", 27) // Ivan27

fmt.Println("Ivan", 27) // Ivan 27
```

```
fmt.Print(33, 27) // 33 27
```

В первом случае один из объектов строка, поэтому пробел между объектами не ставится. Во втором случае мы используем метод `Println()` поэтому пробел ставится в любом случае. В третьем у нас нет строк - поэтому метод `Print()` вставляет пробел между выводимыми объектами.

Еще пример вывода, используя строки и переменные:

```
package main

import "fmt"

func main() {
    name := "Ivan"
    age := 27
    fmt.Println("My name is", name, "and I am", age, "years old.")
}

// вывод:
// My name is Ivan and I am 27 years old.
```

Комментарии

Программа может иметь комментарии. Комментарии служат для описания действий, которые производит программа или какие-то ее части. При компиляции комментарии не учитываются и не оказывают никакого влияния на работу приложения. Комментарии бывают однострочными и многострочными.

Однострочный комментарий располагается в одну строку после двойного слеша (//). Все, что идет после этих символов, воспринимается компилятором как комментарий. Многострочный комментарий заключается между символами / *и* / и может занимать несколько строк:

```
/*
    Первая программа
    на языке Go
*/

package main    // определение пакета для текущего файла
```

```
import "fmt"     // подключение пакета fmt

// определение функции main

func main() {
     fmt.Println("Hello Go!")     // вывод строки на консоль
}
```

Константы

Константы, как и переменные, хранят некоторые данные, но, в отличие от переменных, значения констант нельзя изменить, они устанавливаются один раз. Вычисление констант производится во время компиляции. Благодаря этому уменьшается количество работы, которую необходимо произвести во время выполнения, упрощается поиск ошибок, связанных с константами (так как некоторые из них можно обнаружить на момент компиляции).

Для определения констант применяется ключевое слово `const`:

```
const pi float64 = 3.1415
```

Мы **не** можем менять значение константы:

```
const pi float64 = 3.1415
pi = 2.7182              // ! Ошибка
```

Константы, как и обычные переменные, можно объявлять в блоке:

```
const (
        a int = 45
        b float32 = 3.3
)
```

Также можно не указывать значение следующей константы по порядку (значение будет скопировано):

```
package main

import (
```

```
        "fmt"
)

const(
        A int = 45
        B
        C float32 = 3.3
        D
)
func main() {
        fmt.Println(A, B, C, D)  // Вывод: 45 45 3.3 3.3
}
```

iota

Предположим, что нам нужно использовать дни недели с их номерами:

```
const (
        Sunday    = 0
        Monday    = 1
        Tuesday   = 2
        Wednesday = 3
        Thursday  = 4
        Friday    = 5
        Saturday  = 6
)
fmt.Println(Sunday)    // вывод 0
fmt.Println(Saturday)  // вывод 6
```

iota идентификатор Go используется в объявлениях констант для упрощения определений увеличивающихся чисел.
Сделаем дни недели с использованием **iota -** теперь это выглядит проще (особенно если много данных):

```
const (
        Sunday = iota
        Monday
        Tuesday
        Wednesday
        Thursday
        Friday
        Saturday
)
```

```
func main() {
        fmt.Println(Sunday)   // вывод 0
        fmt.Println(Saturday) // вывод 6
}
```

В объявлении константы *предварительно объявленный идентификатор* **iota** представляет последовательные не типизированные целочисленные константы. Его значение является индексом соответствующего ConstSpec в объявлении константы, начиная с нуля. Поскольку он может использоваться в выражениях, он обеспечивает общность, выходящую за рамки простых перечислений. Его можно использовать для построения набора связанных констант, ознакомьтесь с примерами:

```
const (
  c0 = iota  // c0 == 0
  c1 = iota  // c1 == 1
  c2 = iota  // c2 == 2
)
fmt.Println(c0, c1, c2) // вывод: 0 1 2


const (
        Sunday = iota
        Monday
        Tuesday
        Wednesday
        Thursday
        Friday
        Saturday
        _  // пропускаем 7
        Add
)

fmt.Println(Sunday)   // вывод: 0
fmt.Println(Saturday) // вывод: 6
fmt.Println(Add) // вывод: 8



const (
        u         = iota * 42 // u == 0 (индекс - 0, поэтому 0 * 42 = 0)
        v float64 = iota * 42 // v == 42.0 (индекс - 1, поэтому 1.0 * 42 =
42.0)
        w         = iota * 42 // w == 84  (индекс - 2, поэтому 2 * 42 = 84)
)

// переменные ни в одноме блоке const, поэтому индекс не увеличился
```

```
const x = iota  // x == 0
const y = iota  // y == 0
```

Наиболее часто используемые математические функции пакета `math`

Пакет "math" в языке программирования Go предоставляет богатый набор математических функций для широкого спектра вычислений. Вот описание и примеры более чем 20 функций из этого пакета:

1. **`Abs(x float64) float64`**: Возвращает абсолютное значение числа.

```
result := math.Abs(-5.67)
// result равен 5.67
```

2. **`Ceil(x float64) float64`**: Округляет число вверх до ближайшего целого.

```
result := math.Ceil(5.67)
// result равен 6
```

3. **`Floor(x float64) float64`**: Округляет число вниз до ближайшего целого.

```
result := math.Floor(5.67)
// result равен 5
```

4. **`Sqrt(x float64) float64`**: Возвращает квадратный корень числа.

```
result := math.Sqrt(16)
// result равен 4
```

5. **`Pow(x, y float64) float64`**: Возводит число `x` в степень `y`.

```
result := math.Pow(2, 3)
// result равен 8
```

6. **`Sin(x float64) float64`**, **`Cos(x float64) float64`**, **`Tan(x float64) float64`**: Возвращают синус, косинус и тангенс угла в радианах соответственно.

```
sinValue := math.Sin(math.Pi / 2)
// sinValue равен 1 (синус 90 градусов)
```

7. `Log(x float64) float64`: Возвращает натуральный логарифм числа.

```
result := math.Log(10)
// result равен примерно 2.302585...
```

8. `Max(x, y float64) float64`, `Min(x, y float64) float64`: Возвращают максимальное и минимальное значение из двух чисел соответственно.

```
maxVal := math.Max(3, 7)
// maxVal равен 7

minVal := math.Min(3, 7)
// minVal равен 3
```

9. `Mod(x, y float64) float64`: Возвращает остаток от деления x на y.

```
result := math.Mod(10, 3)
// result равен 1
```

10. `Round(x float64) float64`: Округляет число к ближайшему целому.

```
result := math.Round(5.67)
// result равен 6
```

11. `Trunc(x float64) float64`: Отбрасывает дробную часть числа.

```
result := math.Trunc(5.67)
// result равен 5
```

12. `Inf(sign int) float64`: Возвращает положительную или отрицательную бесконечность в зависимости от знака.

```
posInf := math.Inf(1)
// posInf - положительная бесконечность

negInf := math.Inf(-1)
// negInf - отрицательная бесконечность
```

13. `NaN() float64`: Возвращает "Not a Number" (NaN).

```
nan := math.NaN()
// nan - NaN
```

14. **`Exp(x float64) float64`**: Возвращает экспоненту (e) в степени x.

```
result := math.Exp(2)
// result равен примерно 7.389056...
```

15. **`Exp2(x float64) float64`**: Возвращает 2 в степени x.

```
result := math.Exp2(3)
// result равен 8
```

16. **`Expm1(x float64) float64`**: Возвращает e в степени x минус 1.

```
result := math.Expm1(1)
// result равен примерно 1.718281...
```

17. **`Log10(x float64) float64`**: Возвращает десятичный логарифм числа.

```
result := math.Log10(100)
// result равен 2
```

18. **`Log2(x float64) float64`**: Возвращает двоичный логарифм числа.

```
result := math.Log2(8)
// result равен 3
```

19. **`Log1p(x float64) float64`**: Возвращает натуральный логарифм числа x плюс 1.

```
result := math.Log1p(1)
// result равен примерно 0.693147...
```

20. **`Signbit(x float64) bool`**: Возвращает true, если x отрицательное или отрицательный нуль.

```
isNegative := math.Signbit(-5)
// isNegative равен true
```

Эти функции обеспечивают широкий спектр возможностей для выполнения разнообразных математических операций в языке программирования Go.

Аппаратура и материалы

1. Компьютерный класс общего назначения с конфигурацией ПК не хуже рекомендованной для ОС Windows 10 с подключением к глобальной сети Интернет.
2. Операционная система Windows 10.
3. Система контроля версий Git.
4. Браузер для доступа к web-сервису GitHub (либо иному сервису для удаленного хранения репозиториев Git), рекомендован к использованию Google Chrome.
5. Дистрибутив языка программирования Go.
6. Интегрированная среда разработки Visual Studio Code или подобная.

Указания по технике безопасности

При работе на ЭВМ без разрешения руководителя занятия запрещается:

- подавать (снимать) напряжение на ПЭВМ и электрические розетки с распределительного щита;
- включать и выключать блоки питания ПЭВМ и мониторы;
- извлекать ПЭВМ из защитного кожуха;
- устранять неисправности, возникшие в ходе выполнения лабораторной работы.

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub (или ином сервисе), в котором будет использована лицензия MIT и язык программирования Go.
3. Выполните клонирование созданного репозитория.
4. Дополните файл `.gitignore` необходимыми правилами для работы с используемой IDE (Visual Studio Code и т. д.).
5. Проработайте примеры теоретической части.
6. Решите все задания практической части.
7. Решите индивидуальное задание согласно варианта (номер варианта необходимо уточнить у преподавателя).
8. Используйте для каждой задачи (проекта) отдельную папкую
9. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория. Зафиксируйте изменения.

10. Выполните слияние ветки для разработки с веткой *main*/*master*.
11. Отправьте сделанные изменения на сервер GitHub (или иной используемый сервис).
12. Отправьте адрес репозитория GitHub (или иного используемого сервиса) на электронный адрес преподавателя.

Практическая часть

✔ Задание 1

Задача: Напишите программу, которая выводит "I like Go!"

Пример ввода:

```
```

Соответствующий вывод:

```
I like Go!
```

✔ Задание 2

Задача: Напишите программу, которая выведет "I like Go!" 3 раза.

Пример ввода:

```
```

Соответствующий вывод:

```
I like Go!
I like Go!
I like Go!
```

✔ Задание 3

Задача: Напишите программу, которая последовательно делает следующие операции с введённым числом:

1. Число умножается на 2;
2. Затем к числу прибавляется 100.

После этого должен быть вывод получившегося числа на экран.

Пример ввода:

```
1
```

Соответствующий вывод:

```
102
```

✔ Задание 4

Задача: Петя торопился в школу и неправильно написал программу, которая сначала находит квадраты двух чисел, а затем их суммирует. Исправьте его программу.

Пример ввода:

```
2
2
```

Соответствующий вывод:

```
8
```

```
package main
import "fmt"
func main(){

  var a int
  fmt.Scan(&a) // считаем переменную 'a' с консоли
  fmt.Scan(&b) // считаем переменную 'b' с консоли
```

```
  a = a * a
  b = b * 2
  c = a + b
  fmt.Println(c)
}
```

✔ Задание 5

Задача: По данному целому числу, найдите его квадрат.

Пример ввода:

```
3
```

Соответствующий вывод:

```
9
```

✔ Задание 6

Задача: Дано натуральное число, выведите его последнюю цифру. На вход дается натуральное число N, не превосходящее 10000. Выведите одно целое число - ответ на задачу.

Пример ввода:

```
123
```

Соответствующий вывод:

```
3
```

✔ Задание 7

Задача: Дано неотрицательное целое число. Найдите число десятков (то есть вторую цифру справа). На вход дается натуральное число N, не превосходящее 10000. Выведите одно целое число - число десятков.

Пример ввода:

```
2010
```

Соответствующий вывод:

```
1
```

✔ Задание 8

Задача: Часовая стрелка повернулась с начала суток на d градусов. Определите, сколько сейчас целых часов h и целых минут m. На вход программе подается целое число d (0 < d < 360). Выведите на экран фразу:

```
It is ... hours ... minutes.
```

Вместо многоточий программа должна выводить значения h и m, отделяя их от слов ровно одним пробелом.

Пример ввода:

```
90
```

Соответствующий вывод:

```
It is 3 hours 0 minutes.
```

✔ Задание 9

Задача: Уберите лишние комментарии так, чтобы программа вывела число 100.

Пример ввода:

```

```

Соответствующий вывод:

```
100
```

```
package main
import "fmt"
func main(){
    // a:=44
    /*
    var a2 int = 10
    */
    a2 = a2 * 10
     fmt.Println(a2)
}
```

✔ Задание 10

Задача: Исправьте ошибку в программе ниже:

```
package main

import "fmt"

func main(){
    var a int = 8
    const b int = 10
    a = a + b
    b = b + a
    fmt.Println(a)
}
```

Пример ввода:

```

```

Соответствующий вывод:

```
18
```

✔ Задание 11

Задача: Напишите программу, которая для заданных значений a и b вычисляет площадь поверхности и объем тела, образованного вращением эллипса, заданного уравнением:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1,$$

вокруг оси Ox.

Индивидуальное задание 1

1. **Площадь трапеции:** Задайте переменные для оснований и высоты трапеции. Рассчитайте и выведите площадь трапеции.
2. **Угол между векторами:** Задайте координаты двух векторов в трехмерном пространстве. Рассчитайте и выведите угол между ними.
3. **Объем пирамиды:** Задайте переменные для длины, ширины основания и высоты пирамиды. Рассчитайте и выведите объем пирамиды.
4. **Скорость движения:** Объявите переменные для расстояния и времени движения. Рассчитайте и выведите среднюю скорость.
5. **Объем тетраэдра:** Задайте переменные для длины ребра тетраэдра. Рассчитайте и выведите объем тетраэдра.
6. **Площадь ромба:** Задайте переменные для диагоналей ромба. Рассчитайте и выведите площадь ромба.
7. **Работа и мощность:** Объявите переменные для силы и расстояния. Рассчитайте и выведите работу, а затем мощность, если время выполнения работы равно 10 секундам.
8. **Объем и площадь цилиндрической банки:** Задайте переменные для радиуса и высоты цилиндрической банки. Рассчитайте и выведите объем и полную поверхность цилиндра.
9. **Объем и площадь поверхности куба:** Задайте переменную для длины ребра куба. Рассчитайте и выведите объем и площадь его поверхности.
10. **Длина дуги окружности:** Задайте переменную для угла в радианах и радиуса окружности. Рассчитайте и выведите длину дуги.
11. **Длина отрезка на плоскости:** Задайте координаты двух точек на плоскости. Рассчитайте и выведите расстояние между этими точками.

12. **Объем и площадь поверхности шара:** Задайте переменную для радиуса шара. Рассчитайте и выведите объем и площадь его поверхности.
13. **Угловой коэффициент прямой:** Задайте координаты двух точек на прямой. Рассчитайте и выведите угловой коэффициент прямой.
14. **Площадь сектора круга:** Задайте переменные для радиуса и угла сектора круга. Рассчитайте и выведите площадь этого сектора.
15. **Объем и площадь поверхности конуса:** Задайте переменные для радиуса и высоты конуса. Рассчитайте и выведите объем и площадь его поверхности.
16. **Вычисление процента изменения:** Задайте переменные для начального и конечного значения. Рассчитайте процент изменения и выведите результат.
17. **Сферический треугольник:** Задайте переменные для длин трех сторон сферического треугольника. Рассчитайте его углы и выведите результат.
18. **Объем и площадь поверхности биссектриссоида:** Задайте переменные для радиуса и высоты биссектриссоида. Рассчитайте и выведите объем и площадь её поверхности.
19. **Уравнение касательной:** Задайте переменные для координаты точки и углового коэффициента касательной. Рассчитайте уравнение касательной и выведите его.
20. **Теорема Виета:** Задайте коэффициенты квадратного уравнения. Рассчитайте сумму и произведение корней с использованием теоремы Виета.

Индивидуальное задание 2

1. Даны два числа. Найти среднее арифметическое и среднее геометрическое их модулей.
2. Даны стороны прямоугольника. Найти его периметр и длину диагонали.
3. Даны стороны треугольника a, b и c найти площадь треугольника по формуле Герона

   $$S = \sqrt{p(p-a)(p-b)(p-c)},$$

   где p = (a + b + c) / 2.
4. Даны два числа. Найти их сумму, разность, произведение, а также частное от деления первого числа на второе.
5. Даны длины сторон прямоугольного параллелепипеда. Найти его объем и площадь боковой поверхности.
6. Даны координаты на плоскости двух точек. Найти расстояние между этими точками.
7. Даны основания и высота равнобедренной трапеции. Найти периметр трапеции.

8. Даны основания равнобедренной трапеции и угол при большем основании. Найти площадь трапеции.
9. Треугольник задан координатами своих вершин. Найти периметр и площадь треугольника.
10. Выпуклый четырехугольник задан координатами своих вершин. Найти площадь этого четырехугольника как сумму площадей треугольников.
11. Известна стоимость 1 кг конфет, печенья и яблок. Найти стоимость всей покупки, если купили x кг конфет, y кг печенья и z кг яблок.
12. Известна стоимость монитора, системного блока, клавиатуры и мыши. Сколько будут стоить 3 компьютера из этих элементов? N компьютеров?
13. Возраст Тани – X лет, а возраст Мити – Y лет. Найти их средний возраст, а также определить, на сколько отличается возраст каждого ребенка от среднего значения.
14. Два автомобиля едут навстречу друг другу с постоянными скоростями V_1 и V_2 км/ч. Определить, через какое время автомобили встретятся, если расстояние между ними было S км.
15. Два автомобиля едут друг за другом с постоянными скоростями V_1 и V_2 км/ч ($V_1 > V_2$). Определить, какое расстояние будет между ними через 30 мин после того, как первый автомобиль опередил второй на S км.
16. Известно значение температуры по шкале Цельсия. Найти соответствующее значение температуры по шкале:

- Фаренгейта;
- Кельвина. Для пересчета по шкале Фаренгейта необходимо исходное значение температуры умножить на 1,8 и к результату прибавить 32, а по шкале Кельвина абсолютное значение нуля соответствует –273,15 градуса по шкале Цельсия.

17. Напишите программу, в которой вычисляется сумма, разность, произведение, частное и среднее арифметическое двух целых чисел, введенных с клавиатуры. Например, при вводе чисел 2 и 7 должен быть получен ответ вида:

```
2+7=9 2-7=-5 2*7=14 2/7=0.2857142857142857 (2+7)/2=4.5
```

18. Известны объем и масса тела. Определить плотность материала этого тела.
19. Известны количество жителей в государстве и площадь его территории. Определить плотность населения в этом государстве.
20. Составить программу решения линейного уравнения $ax + b = 0$, $(a \neq 0)$.
21. Даны катеты прямоугольного треугольника. Найти его гипотенузу.

22. Найти площадь кольца по заданным внешнему и внутреннему радиусам.
23. Даны катеты прямоугольного треугольника. Найти его периметр.
24. Даны основания и высота равнобедренной трапеции. Найти ее периметр.

Содержание отчета и его форма

Отчет по лабораторной работе оформляется электронно в формате PDF, должен содержать ответы на контрольные вопросы, ссылку на репозиторий с которым выполнялась работа, скриншоты IDE PyCharm, скриншоты результатов работы программ.

Вопросы для защиты работы

1. Как объявить переменную типа int в Go?
2. Какое значение по умолчанию присваивается переменной типа int в Go?
3. Как изменить значение существующей переменной в Go?
4. Что такое множественное объявление переменных в Go?
5. Как объявить константу в Go?
6. Можно ли изменить значение константы после ее объявления в Go?
7. Какие арифметические операторы поддерживаются в Go?
8. Какой оператор используется для выполнения операции остатка в Go?
9. Какой результат выражения `5 / 2` в Go?
10. Как считать строку с консоли в Go?
11. Как считать целое число с консоли в Go?
12. Как обработать ошибку при считывании данных с консоли в Go?
13. Как вывести строку в консоль в Go?
14. Как вывести значение переменной типа int в консоль?
15. Как форматировать вывод числа с плавающей точкой в Go?
16. Как объявить переменную типа `byte` и присвоить ей значение 65?Чем отличается оператор := от оператора = в Go?
17. Какие типы данных можно использовать для представления чисел с плавающей точкой в Go?
18. Как объявить и использовать несколько переменных в Go?