Глава 1
СИНТАКСИС, ВСТРОЕННЫЕ ТИПЫ ДАННЫХ И УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ GO

В этой главе мы рассмотрим историю создания языка программирования Go и его основные особенности. Вы узнаете, как установить Go на ваш компьютер под управлением ОС Windows и настроить рабочее окружение. Кроме того, вы познакомитесь с базовыми типами данных, правилами именования переменных, основными операторами и управляющими конструкциями Go.

1.1. Краткая история и основные особенности Go

Golang (Go) — кросс-платформенный компилируемый язык программирования со строгой статической типизацией, открытым исходным кодом и поддержкой обобщенного программирования (в версии 1.18 добавлены шаблоны/дженерики). Работа над ним началась в 2007 году, в 2009-м состоялся первый релиз, а в марте 2012-го вышла первая стабильная версия. Авторами этого языка программирования принято считать Роберта Гризмера, Роба Пайка и Кена Томпсона. Основные требования, которые ставились ими при разработке Go, можно сформулировать следующим образом:

- простой в использовании синтаксис языка;
- компилируемый язык с быстрой компиляцией;
- легкость в изучении;
- встроенный, простой в использовании параллелизм;
- надежная стандартная библиотека;
- отсутствие неявных преобразований;
- сборка мусора;
- разделение интерфейсов и реализации;
- система пакетов, обеспечивающая быструю сборку приложений и с явным указанием зависимостей.

По синтаксису Go очень похож на язык программирования C (Си). Он не перегружен огромным количеством функций, что позволяет писать читаемый и более

легкий в сопровождении код. Этого удалось достичь в том числе и благодаря тому, что в Go всего 25 ключевых слов (в C11 — 32, в Java — 50, а в C++ — более 50):

```
break      default       func      interface     select
case       defer         go        map           struct
chan       else          goto      package       switch
const      fallthrough   if        range         type
continue   for           import    return        var
```

Исходный код вашей программы будет собираться в один исполняемый файл, уже содержащий все необходимые зависимости проекта. Этим Go тоже отличается от других языков: например, в Java или Python, чтобы создать самодостаточный исполняемый файл, необходимо упаковать довольно большое количество зависимостей или убедиться, что в системе уже установлено нужное программное обеспечение, а возможно, и библиотеки с прописанными к ним путями в переменных окружения.

Программы, написанные на Go, по скорости выполнения сопоставимы с аналогичными, реализованными средствами Java, но уступают C/C++. При этом писать на Go конкурентный/параллельный код в разы проще, чем на других языках.

По замыслу Go ближе всего к языку программирования C (Си), поэтому в нем нет классов, из-за чего отсутствует и такое понятие, как наследование. Вместо классов используются *структуры*, а вместо наследования — *композиция*. Еще одна особенность Go — обрезанные возможности полиморфизма и отсутствие перегрузки функций во время компиляции. Таким образом, его нельзя назвать языком, полностью поддерживающим концепцию объектно-ориентированного программирования (ООП). Конечно, тем, кто уже знаком с любым ООП-языком, такие нюансы могут показаться странными и вызвать сомнения в необходимости изучения Go, но, поверьте, в них нет ничего страшного! Go — очень изящный, минималистичный и мощный инструмент для написания как системных утилит, так и серверного программного обеспечения (вне зависимости от архитектуры — монолитной или микросервисной).

Код проекта на Go собирается достаточно быстро, что позволяет проводить тестирование или отладку более быстро, чем, например, для проекта на C++. Возможно, для вас это не является чем-то особенным, но, поверьте моему опыту, когда объем кода на C++ превышает несколько сотен тысяч строк — выполнять его сборку то еще «удовольствие». Одно дело, когда это финальная сборка проекта; но если необходимо провести ручную отладку, то сборка может длиться не один десяток минут. Конечно, существуют подходы для ускорения этого процесса (один из них — предкомпилированные заголовки), но всегда найдется ситуация, где они бессильны.

Вместо определения «менеджер пакетов» в Go принято использовать определение «менеджер зависимостей». Он работает с модулями, которые можно подключить к вашему проекту в процессе его разработки, тем самым сделав его зависимым от чужого кода. Есть также сайт, на котором можно найти огромное количество модулей с различной функциональностью, — https://pkg.go.dev/. Каждый из них (если это не пакет из стандартной библиотеки) можно подключить к проекту, указав путь до репозитория, где хранится код. Таким образом, в качестве зависимости может использоваться не только модуль, найденный на приведенном сайте, но и вообще любой пакет, код которого находится в открытом репозитории.

Обязательное требование к запускаемому приложению — наличие функции верхнего уровня `main`, выступающей в роли точки входа (запуска). При этом, как и в Python, не нужно каждую команду в коде завершать символом точки с запятой (`;`).

1.2. Установка и настройка рабочего окружения

Для разработки на Go могут использоваться как IDE (Goland), так и редакторы кода (Visual Studio Code, Vim, Emacs) после установки в них соответствующих плагинов/расширений. Кроме того, писать и выполнять код Go прямо в браузере можно с помощью ознакомительного веб-сервиса («песочницы») The Go Playground:

https://go.dev/play/

Один из часто используемых редакторов кода для разработки на G — Visual Studio Code. Его можно скачать по следующей ссылке:

https://code.visualstudio.com/download

Далее необходимо скачать установщик Go под ту операционную систему, которую вы обычно используете. В моем случае это Windows 11, а версия языка программирования — go1.24 (рис. 1.1):

https://go.dev/dl/

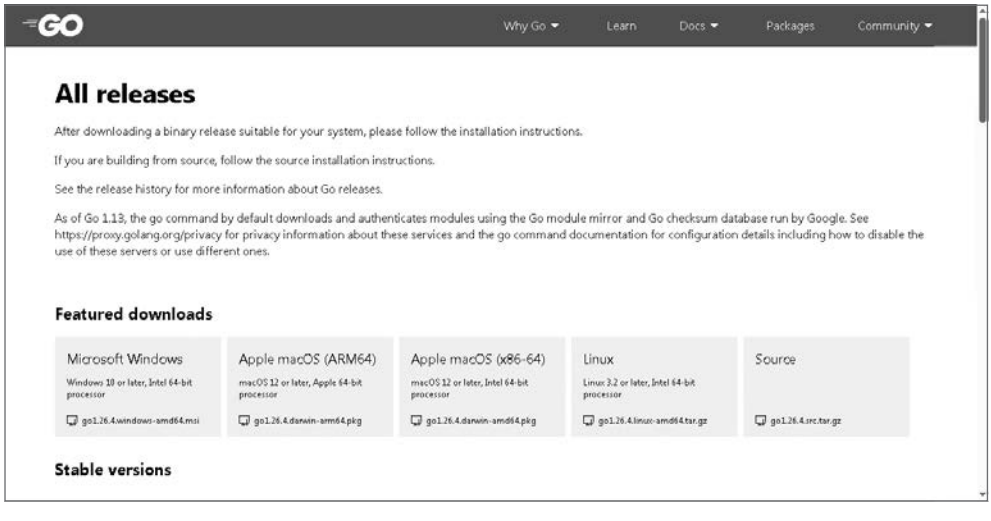

Рис. 1.1. Скачиваем установщик с текущей стабильной версией Go

Затем запустите установщик и укажите путь для распаковки Go в удобную для вас директорию. Обычно ею выступает корневой каталог диска (C, D, F и т. д.). Например, `C:\Go\`.

Чтобы убедиться в успешной установке, можно запустить терминал и ввести в него команду `go version` (рис. 1.2).

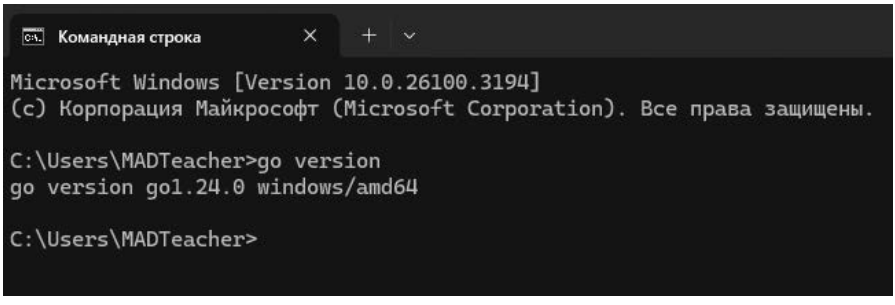

Рис. 1.2. Проверка корректной установки Go

Далее запустите Visual Studio Code и перейдите на панель Extensions (Расширения), нажав сочетание клавиш Ctrl+Shift+X. В появившемся поле поиска введите **Go** и установите расширение (рис. 1.3).

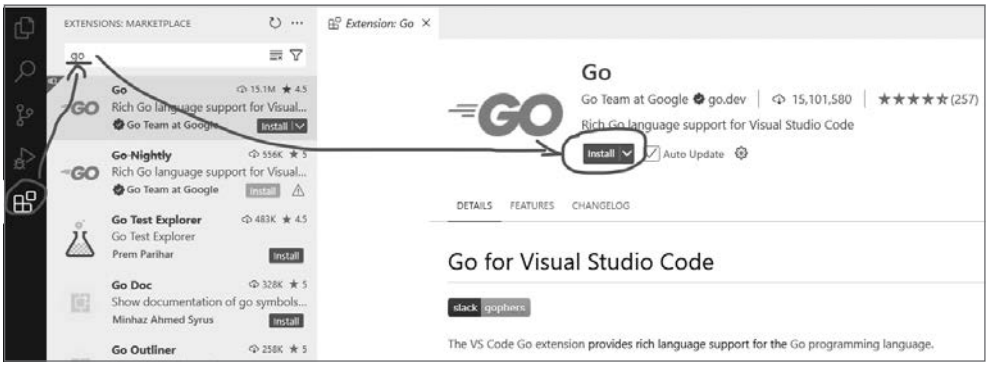

Рис. 1.3. Установка расширения Go для VS Code

Затем необходимо обновить/доустановить несколько его модулей. Для этого нажмите Ctrl+Shift+P или F1, после чего в появившейся строке введите **`> go install`** и выберите Go: Install/Update tools (рис. 1.4).

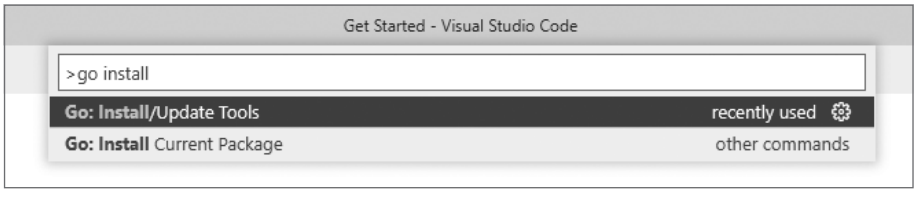

Рис. 1.4. Первый шаг для установки модулей расширения

Эта команда может появиться в списке до того, как вы введете ее полностью. В этом случае просто выберите ее из списка, выделив все модули расширения, и нажмите OK (рис. 1.5).

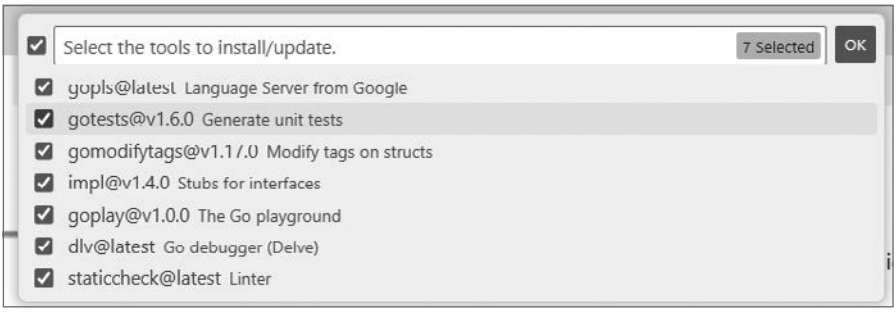

Рис. 1.5. Второй шаг для установки модулей расширения

После того как вы нажмете OK, откроется терминал, в котором будет отображаться ход установки модулей расширения (рис. 1.6).

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

Tools environment: GOPATH=C:\Users\MADTeacher\go
Installing 7 tools at C:\Users\MADTeacher\go\bin in module mode.
  gotests
  gomodifytags
  impl
  goplay
  dlv
  staticcheck
  gopls

Installing github.com/cweill/gotests/gotests@latest (C:\Users\MADTeacher\go\bin\gotests.exe) SUCCEEDED
Installing github.com/fatih/gomodifytags@latest (C:\Users\MADTeacher\go\bin\gomodifytags.exe) SUCCEEDED
Installing github.com/josharian/impl@latest (C:\Users\MADTeacher\go\bin\impl.exe) SUCCEEDED
Installing github.com/haya14busa/goplay/cmd/goplay@latest (C:\Users\MADTeacher\go\bin\goplay.exe) SUCCEEDED
Installing github.com/go-delve/delve/cmd/dlv@latest (C:\Users\MADTeacher\go\bin\dlv.exe) SUCCEEDED
Installing honnef.co/go/tools/cmd/staticcheck@latest (C:\Users\MADTeacher\go\bin\staticcheck.exe) SUCCEEDED
Installing golang.org/x/tools/gopls@latest (C:\Users\MADTeacher\go\bin\gopls.exe) SUCCEEDED

All tools successfully installed. You are ready to Go. :)
```

Рис. 1.6. Ход установки модулей расширения

Теперь вы готовы приступить к написанию своего первого приложения, сакрального для всех программистов, — `Hello World!`. Для этого создайте в удобном для вас месте директорию `hello-go` и откройте ее средствами VS Code (рис. 1.7).

Директория открыта, но VS Code еще не знает, с каким языком программирования ему предстоит работать. Это связано с тем, что в директории нет ни одного файла с кодом. А в случае с Go ее еще предстоит инициализировать. Для начала создайте файл `main.go` (рис. 1.8).

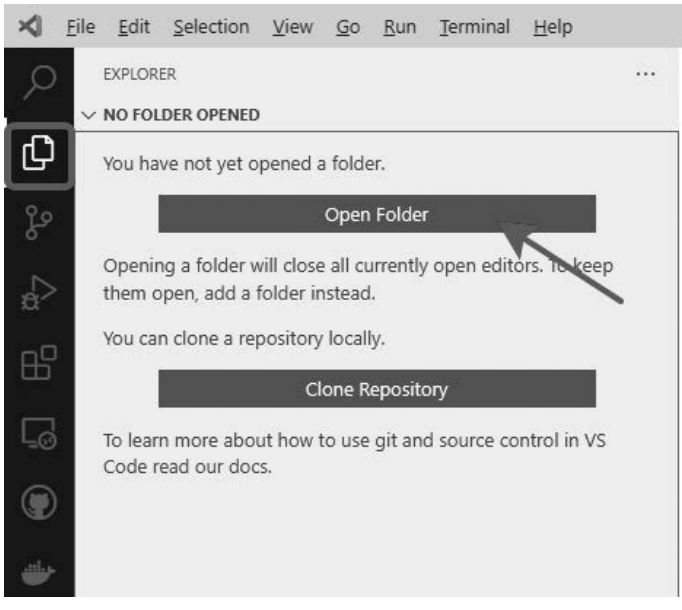

Рис. 1.7. Открытие директории hello-go

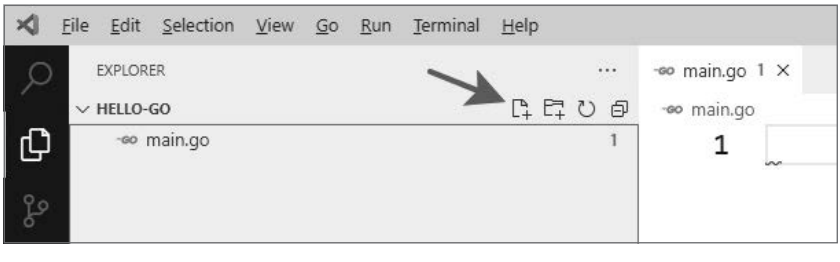

Рис. 1.8. Создание файла main.go

Далее откройте терминал, выбрав Terminal ▸ New Terminal (Терминал ▸ Новый терминал) (или нажмите Ctrl+Shift+`), и запустите команду **`go mod init hello-go`** для инициализации вашего приложения (рис. 1.9).

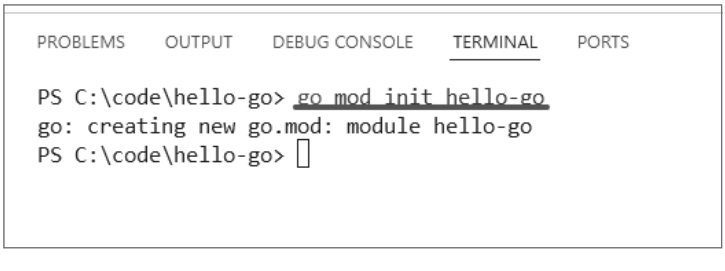

Рис. 1.9. Инициализация проекта

В директории должен появиться файл `go.mod` со следующим содержимым (рис. 1.10).

Рис. 1.10. Содержимое файла go.mod

В этом файле хранятся все зависимости вашего проекта (модуля), которые при первом его запуске или сборке будут скачиваться с их репозиториев, а при последующих — из кэша (директории на компьютере).

Перейдите в файл `main.go` и наберите в нем следующий код:

```
package main

import "fmt"

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

Первая строчка указывает, к какому пакету будет относиться код в этом файле. В третьей строке импортируется пакет `fmt`, который реализует функции форматированного ввода/вывода данных. В нашем случае используется его функция `Println` для вывода в терминал строки `Hello World!`, что представлено на шестой строчке (учитывая пустые). Более подробно пакеты и модули мы рассмотрим в одной из следующих глав. Функция `main` представляет собой точку входа (запуска) разрабатываемого приложения. Именно с нее всегда будет осуществляться сборка и запуск любого приложения, написанного на Go. Но только при условии, что она объявлена в пакете `main`.

Запустить приложение можно одним из следующих вариантов:

- нажмите F5 — для запуска в режиме отладки (debug);
- нажмите сочетание клавиш Ctrl+F5 — для запуска без этого режима;
- введите в терминале команду **`go run main.go`**.

В результате в консоль должен вывестись следующий текст:

```
Hello World!
```

Теперь VS Code можно полноценно использовать для разработки на Go.

1.3. Объявление переменных

Как я уже сказал, Go очень похож на язык программирования C (Си), поэтому каждая его переменная представляет собой область в памяти, где хранится ее значение. У всех переменных есть свой тип данных, который можно вывести автоматически

с помощью компилятора или задать вручную. Мы рассмотрим оба варианта и начнем со второго.

Для объявления переменной используется ключевое слово `var`, за которым следует имя и тип значения:

```
var имя_переменной тип_переменной
```

Если при объявлении переменной ей не присваивается начальное значение, то она инициализируется значением по умолчанию. Для целочисленного типа данных это `0`, для строкового — пустая строка и т. д.:

```
package main

import "fmt"

func main() {
    var value int
    fmt.Println(value) // 0
}
```

Присвоить начальное значение переменной можно так:

```
package main

import "fmt"

func main() {
    var value int = 10
    fmt.Println(value) // 10
}
```

Компилятор способен автоматически вывести тип переменной, но ее обязательно придется инициализировать начальным значением:

```
package main

import (
    "fmt"
    "reflect"
)

func main() {
  var value = 10 // удалите = 10 и посмотрите на возникшую ошибку
  fmt.Println(reflect.TypeOf(value)) // int
}
```

Если необходимо объявить сразу несколько переменных одного типа данных, это можно сделать несколькими способами:

```
package main

import "fmt"

func main() {
  // var value1 int
  // var value2 int
  // var value1, value2 int
  var value1, value2 = 10, 20
  fmt.Println(value1, value2) // 10, 20
}
```

Еще есть возможность использовать только одно ключевое слово var при объявлении переменных различных типов данных:

```
var (
    name     string
    age      int
    location string
)

var (
    name = "Alex"
    age = 22
    location = "SPb"
)

var name, location, age = "Alex", "SPb", 22
var (
  name, location, age = "Alex", "SPb", 22
)
```

Кроме того, Go предоставляет разработчику короткий способ объявления переменных, при котором нет необходимости использовать ключевое слово var:

```
package main

import "fmt"

func main() {
  value := 10
  name, location, age := "Alex", "SPb", 22
  fmt.Println(value)   // 10
  fmt.Println(name, location, age)  // Alex SPb 22
}
```

Чтобы присвоить переменной другое значение, используйте оператор =, а не :=. Повторное использование := приведет к ошибке, из-за чего не получится выполнить сборку и запуск программы (попробуйте, чтобы убедиться в этом).

1.4. Правила именования переменных

К выбору имени переменной нужно подходить с особой осторожностью! Обычно начинающие разработчики любят все сокращать и использовать такие имена, как a, b, i, str и т. д. Если это тестовое приложение или на нем вы обучаетесь языку программирования, то в таких именах нет ничего страшного. Но, когда вы участвуете в разработке реального продукта, пет-проекта, диплома, такие короткие имена лучше не использовать. Это связано с тем, что, помимо вас, этот код могут читать другие люди и им придется прилагать много усилий, чтобы понять, что вы хотели в нем реализовать. К тому же, когда вы вернетесь к проекту через месяц-другой, вам самим будет трудно разобраться в написанном ранее! Думаете, такого не может быть? Еще как может! Даже опытным разработчикам, написавшим часть кода проекта, через какое-то время снова придется читать его, чтобы освежить в памяти его содержимое.

Что касается начинающих специалистов, то при неправильном именовании переменных такое чтение кода может сильно усложниться. Поэтому привыкайте

вкладывать смысл в имя объявляемой переменной, чтобы в последующем уже только по нему можно было понять, для чего она используется, и для этого не приходилось возвращаться в самое начало кода.

Как и любой другой язык программирования, Go имеет правила именования. Для написания многословных имен принято использовать `MixedCaps` или `mixedCaps` (так называемый верблюжий регистр), а не разделять слова подчеркиванием (как в других языках, например в Python). Если идентификатор будет использоваться в других пакетах, то его первый символ должен быть в верхнем регистре, то есть начинаться с заглавной буквы. Если же он предназначен только для внутреннего использования — смело придерживайтесь `mixedCaps`. Это свойство объявления переменных, функций и структур мы более подробно разберем в главе 2. На данный момент просто объявляйте переменные, начиная их имя с прописной буквы. Но при этом следите за тем, чтобы оно не совпадало с любым из ключевых слов языка программирования.

Однобуквенные имена переменных подходят для небольших циклов. Если же цикл занимает десятки строк кода, то лучше выбрать более понятное имя. При использовании аббревиатур все их буквы должны быть заглавными: `productAPI`, а не `productApi`.

1.5. Комментарии

Комментарии — это строки программы, которые содержат описание кода приложения. Компилятором или интерпретатором они игнорируются. Чаще всего комментарии используют для разъяснения того, что делает та или иная часть программы. Но это не значит, что код должен изобиловать комментариями! Комментируя сложение переменных или простое действие, которое и так легко читается по коду программы, вы только усложняете жизнь себе или другим разработчикам. У комментария должна быть цель — донести до читателя концепцию того, почему код написан именно таким образом, а не описывать то, что и так понятно после прочтения самого кода.

В Go используются *однострочные и многострочные комментарии*. Однострочные уже встречались вам в кодах выше. Они начинаются с `//` и действуют только на тот текст или код, который располагается после них:

```
package main

import "fmt"

func main() {
    a, b := 13, 1
    // это комментарий
    // rez := b & a этот код не выполнится
    rez := b ^ a
    fmt.Printf("%d", rez) // это тоже комментарий
}
```

Многострочные комментарии заключаются в `/*` и `*/` . Они могут быть оформлены и как однострочные на каждой строке, и как многострочные для объявления

блока, в теле которого будет помещен комментируемый код или описание того, что происходит в программе:

```
package main

import "fmt"

func main() {
    a, b := 13, 1     /* это многострочный комментарий из одной строки */
    /* это многострочный комментарий
    rez := b & a этот код не выполнится */
    rez := b ^ a
    fmt.Printf("%d", rez) // это однострочный комментарий
}
```

1.6. Расположение фигурных скобок

В C++-сообществе долгое время шел спор, как ставить фигурные скобки в методах, классах и функциях: сразу после их объявления или на следующей строке. К единому мнению разработчики так и не пришли. Именно поэтому, в зависимости от проекта или компании, в Си-подобных языках допустимы оба варианта постановки фигурных скобок.

В Go, напротив, фигурная скобка должна раскрываться сразу после объявления метода, структуры, условного оператора либо функции. Если вы не будете соблюдать эти требования, то компилятор не даст собрать приложение.

Для примера используем следующий код:

```
package main

import "fmt"

func main() {
    a, b := 13, 1
    if a <= b {
        fmt.Println(a)
    }
}
```

Теперь попробуйте сместить фигурную скобку на следующую строку после объявления функции `main` или проверки на неравенство и собрать приложение, нажав `F5`. В терминале будет выведена ошибка времени компиляции (`compile error`), говорящая о том, что так делать нельзя: `unexpected semicolon or newline before { (exit status 2)`.

1.7. Типы данных Go

Типы данных в Go делятся на встроенные (основные) и составные. К *встроенным* относят:

- целые числа (integers) (знаковые (signed) и беззнаковые (unsigned));
- числа с плавающей запятой (floats);
- комплексные числа (complex numbers);