

- Basics
  - Installation
    - Check go installation
    - Setup environment variable
  - First program
  - Launch without a binary file
  - Build a binary file
  - Install additional tools
  - Example with linter usage
  - Troubleshooting
    - Tabs and spaces
  - Variables
- Data types
  - Literals
  - Conditional
  - Cycles
    - For
    - Range
  - Slices
  - Functions
  - Structures
  - Method with structure
  - Visibility
  - Pointers
  - go routines
  - channels
    - buffer channel
  - Interfaces

# Basics

---

## Installation

Check go installation

```
go version
```

```
1@DESKTOP-8B6DSJ8 MINGW64 ~  
$ go version  
go version go1.18.2 windows/amd64
```

Setup environment variable

for linux

---

```
export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin
```

For windows

```
setx GOPATH %USERPROFILE%\go
setx path "%path%;%GOPATH%\bin"
```

## First program

```
package main

import "fmt"

func main(){

    fmt.Println("Hello, world!")

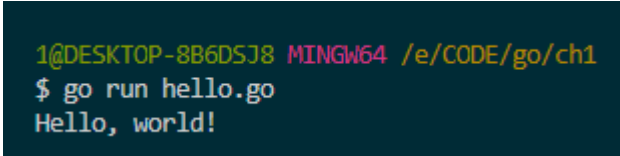
}
```

- Save file as `hello.go`

## Launch without a binary file

- Launch

```
go run hello.go
```



```
1@DESKTOP-8B6DSJ8 MINGW64 /e/CODE/go/ch1
$ go run hello.go
Hello, world!
```

- While launching binary file was created in temporary directory and deleted after program was finished

## Build a binary file

```
go build -o hello_world hello.go
```

```
1@DESKTOP-8B6DSJ8 MINGW64 /e/CODE/go/ch1
$ go build hello.go

1@DESKTOP-8B6DSJ8 MINGW64 /e/CODE/go/ch1
$ ll
total 1849
-rwxr-xr-x 1 1 197121 1892352 May 28 22:01 hello.exe*
-rw-r--r-- 1 1 197121      81 May 26 10:53 hello.go
```

## Install additional tols

You can install additional tools via `go install`

For example install aggregate linter (include many popular linters)

```
go install github.com/golangci/golangci-lint/cmd/golangci-lint@v1.46.2
```

## Example with linter usage

- create module for an application

```
go mod init ch1
```

- create makefile for build. It will apply linter and create binary file

```
.DEFAULT_GOAL:= build

fmt:
    go fmt ./...
.PHONY: fmt

lint:fmt
    golint ./...
.PHONY: lint

vet:fmt
    go vet ./...
.PHONY: vet

build:vet
    go build hello.go
.PHONY: build
```

After `:` defined previous `target`, link to previous task which has to been completed before current task

- launch build

```
make
```

```
1@DESKTOP-8B6DSJ8 MINGW64 /e/CODE/go/ch1
$ make
go fmt ./...
go vet ./...
go build hello.go
```

if you don't have `make` on Windows, you can install `choko` manager and then install `make`

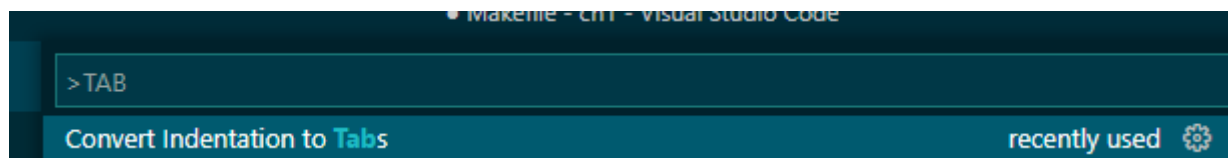
```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-
Object
System.Net.WebClient).DownloadString('[https://community.chocolatey.org/ins
tall.ps1](https://www.google.com/url?q=https://community.chocolatey.org/install.ps1&sa=D&source=editors&ust=1675
178682544290&usg=A0vVaw1VvL_ZL3FJM_aIM1uyGrzj)'))
```

```
choco install make
```

## Troubleshooting

### Tabs and spaces

In the process writing VS code change tab to spaces. Then I find an option



You can check indentation with help of `cat`

```
cat -e -t -v Makefile
```

```
1@DESKTOP-8B6DSJ8 MINGW64 /e/CODE/go/ch1
$ cat -e -t -v Makefile
.DEFAULT_GOAL := build$
$
fmt:$
^Igo fmt ./...$
.PHONY:fmt$
$
lint: fmt$
^Igolint ./...$
.PHONY:lint$
$
vet: fmt$
^Igo vet ./...$
.PHONY:vet$
$
build: vet$
^Igo build hello.go$
.PHONY:build
```

Before it was like this

```
.PHONY:build
1@DESKTOP-8B6DSJ8 MINGW64 /e/CODE/go/ch1
$ cat -e -t -v Makefile
.DEFAULT_GOAL := build$
$
fmt:$
    go fmt ./...$
.PHONY:fmt$
$
lint: fmt$
    golint ./...$
.PHONY:lint$
$
vet: fmt$
    go vet ./...$
.PHONY:vet$
$
build: vet$
    go build hello.go$
.PHONY:build
```

## Variables

keyword var then name of variable then type of variable

```
var age int
```

```
package main

import "fmt"

func main() {
```

```
var name string = "John"
fmt.Println(name)
}
```

## Data types

---

`int`, `int8`, `int16`, `int32`, `int64`

`float32`, `float64` - floating-point

`bool` - boolean data type

## Literals

In integral literal you can write underscores `_`

## Conditional

```
if condition {
} else {
}
```

comparison operators: `==`, `!=`, `<`, `>`, `<=`, `>=`.

```
package main

import "fmt"

func main() {

    const age = 20

    if age >= 18 {
        fmt.Println("Этот человек совершеннолетний")
    } else {
        fmt.Println("Этот человек несовершеннолетний")
    }

}
```

## Cycles

For

```
package main

import "fmt"

func main() {
    for i := 0; i < 5; i++ {
        fmt.Println(i)
    }
}
```

## Range

```
package main

import "fmt"

func main() {
    names := []string{"Ivan", "Petr", "Johan"}

    for index, name := range names {
        fmt.Println(index, name)
    }
}
```

```
package main

import "fmt"

func main() {
    numbers := []int{0, 2, 3, 4}
    for index, value := range numbers {
        fmt.Println(index, value)
    }
}
```

## ignoring index

```
for _, value := range slice {
    fmt.Println(value)
}
```

## ignoring value

```
for index, _ := range slice {  
    fmt.Println(index)  
}
```

## Slices

```
package main  
  
import "fmt"  
  
func main() {  
  
    numbers := []int{1, 2, 3, 4, 5}  
    numbers = append(numbers, 6)  
    subset := numbers[2:4]  
    fmt.Println("numbers:", numbers)  
    fmt.Println("subset:", subset)  
}
```

## Functions

```
func add(x int, y int) int {  
    return x + y  
}
```

```
package main  
  
import "fmt"  
  
func main() {  
  
    fmt.Println(isEven(0))  
    fmt.Println(isEven(2))  
    fmt.Println(isEven(145))  
    fmt.Println(isEven(3))  
    fmt.Println(isEven(10))  
}  
  
func isEven(number int) bool {  
    return number%2 == 0  
}
```

## Structures



```
type Person struct {  
    name string  
    age  int  
}
```

```
var p Person  
p.name = "John"  
p.age = 30
```

```
person := Person{name: "Alice", age: 30}
```

```
package main  
  
import "fmt"  
  
func main() {  
  
    var field Rectangle  
    field.height = 10  
    field.width = 20  
    fmt.Println(field.height, field.width)  
}  
  
type Rectangle struct {  
    width  int  
    height int  
}
```

## Method with structure

method is outside of structure. The first part after func is receiver. So called that connected to structure.

```
package main  
  
import "fmt"  
  
func main() {  
  
    var field Rectangle  
    field.height = 10  
    field.width = 20  
    fmt.Println(field.perimeter())  
}
```

```
type Rectangle struct {  
    width  int  
    height int  
}  
  
func (rectangle Rectangle) perimeter() int {  
    return rectangle.height + rectangle.width  
}
```

## Visibility

Visibility rules:

- Capitalization: If the name of a method, function, variable, or structure begins with a capital letter, then the identifier is exportable, meaning it can be accessed from other packages. This is similar to public in other programming languages.
- Lowercase Letter: If the name begins with a lowercase letter, then this identifier is non-exportable and can only be accessed within its package. This is similar to private in other programming languages.

```
package mypackage  
  
type Car struct { // Exportable structure  
    Make string // private field  
    model string // public field  
}  
  
func (c Car) Describe() string { // public method  
    return fmt.Sprintf("%s %s", c.Make, c.model)  
}  
  
func secretFunction() { // private method  
    fmt.Println("This is a secret")  
}
```

## Pointers

```
var ptr *int
```

```
var x int = 10  
ptr := &x  
fmt.Println(*ptr)
```

```

package main

import "fmt"

func main() {

    var variable int = 10
    ptr := &variable

    *ptr = 20

    fmt.Println(variable)

}

```

```

package main

import "fmt"

func increment(x *int) {
    *x += 1
}

func main() {
    var a int = 5
    increment(&a)
    fmt.Println(a) // Выведет 6
}

```

```

package main

import "fmt"

func newInt() *int {
    var dummy int = 10
    return &dummy
}

func main() {
    numPtr := newInt()
    fmt.Println(*numPtr) // Выведет 10
}

```

if we will change to `fmt.Println(numPtr)` then we just print an address to memory as hex digit.  
`0xc000014088`

## go routines

it is lightweight threads.

```
package main

import (
    "fmt"
    "time"
)

func say(s string) {
    for i := 0; i < 5; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(s)
    }
}

func main() {
    go say("world")
    say("hello")
}
```

## channels

a way to exchange data between go routines without races. It can be described as a stream of data.

```
package main

import "fmt"

func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum // put int to the channel
}

func main() {
    s := []int{7, 2, 8, -9, 4, 0}

    c := make(chan int)
    go sum(s[:len(s)/2], c)
    go sum(s[len(s)/2:], c)
    x, y := <-c, <-c // blocking operation before results are ready

    fmt.Println(x, y, x+y)
}
```

buffer channel

```
resultChan := make(chan int, 2)
```

A buffered pipe has an internal buffer that allows it to store a certain number of elements without having to read them immediately. When you create a channel using `make(chan Type, size)`, you specify the maximum number of elements that can be stored in the channel buffer.

Sending to a buffered channel:

- If a buffered channel has free buffer space, sending to the channel occurs without blocking—the sender does not wait for the receiver to start reading.
- If the channel's buffer is full, the sender blocks and waits until the buffer becomes free (when another goroutine reads from the channel).

Receiving from a buffered channel:

- If there is data in the channel, the reception occurs without blocking the recipient immediately receives the data.
- If the channel is empty, the receiver blocks and waits until data is sent to the channel.

```
package main

import (
    "fmt"
)

func calculateSum(values []int, resultChan chan int) {
    sum := 0
    for _, value := range values {
        sum += value
    }
    resultChan <- sum
}

func main() {
    numbers := []int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    resultChan := make(chan int, 2)

    mid := len(numbers) / 2
    go calculateSum(numbers[:mid], resultChan)
    go calculateSum(numbers[mid:], resultChan)

    sum1, sum2 := <-resultChan, <-resultChan

    fmt.Println("Total Sum:", sum1 + sum2)
}
```

## Interfaces

```
type Describer interface {  
    Describe() string  
}
```

both of the next structures implemented interface

```
func (p Person) Describe() string {  
    return fmt.Sprintf("%s is %d years old", p.Name, p.Age)  
}  
  
func (c Car) Describe() string {  
    return fmt.Sprintf("%s %s, made in %d", c.Make, c.Model, c.Year)  
}
```

```
func printDescription(d Describer) {  
    fmt.Println(d.Describe())  
}
```