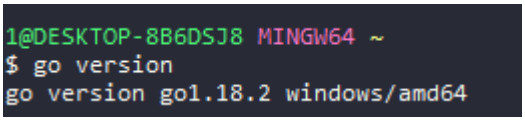
Basics

Installation

Check go installation

```
go version
```



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`

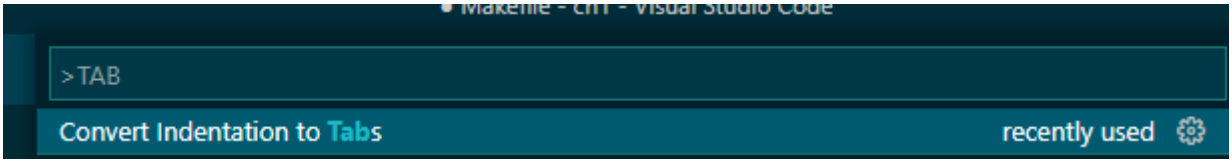
```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-
Object
System.Net.WebClient).DownloadString('[https://community.chocolatey.org/ins
tall.ps1](https://www.google.com/url?
q=https://community.chocolatey.org/install.ps1&sa=D&source=editors&ust=1675
178682544290&usg=AOvVaw1Vvl_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 halp 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

```
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)
    }
}
```

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
```

```
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 reciever. 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
}
```