basics

idiomatic Go

sometimes you have to think differently

think simpler

idiomatic ways

- formatting
- naming
- control structures
- …

see https://golang.org/doc/effective_go.html

Variables

```
// declaration & assignment
var x int
var s string = "foo"

// type inference
b := "bar"

// multiple assignment
var y, z int = 1, 2

// constants
const s string = "constant"
```

types

```
bool

string

int int8 int16 int32 int64 uint uint8 uint16 uint32 uint64 uintptr

byte     // alias for uint8
rune     // alias for int32
         // represents a Unicode code point

float32 float64

complex64 complex128
```

formatting

```
print("hello")
fmt.Print("hello")
fmt.Println("hello")

fmt.Printf("Hello, %s", "Karlsruhe")
fmt.Printf("Hello, %d", 42)
fmt.Printf("Hello, %v %v", "Karlsruhe", 47.11)
fmt.Printf("Hello, %+v", someStruct) // struct with fields
fmt.Printf("Hello, %T", someStruct) // type

fmt.Sprintf("Hello, %T", someStruct) // string
fmt.Fprintf(w, "Hello, %T", someStruct) // writer
```

https://golang.org/pkg/fmt/

if-else

```
x := 42
if x < 0 {
	fmt.Println("Negative")
} else if x == 42 {
	fmt.Println("The answer to life. Also positive.")
} else {
	fmt.Println("Positive")
}

// inline statement
if y := x % 2; y == 0 {
	fmt.Println("Even.")
} else {
	fmt.Println("Odd")
}
```

switch

```
x := 4711
switch x {
case 4711:
	fmt.Println("Cologne")
case 42:
	fmt.Println("The answer to life.")
}
```

Automatic break

break for breaking out of loops

fallthrough for falling through

switch (no expression)

```
t := time.Now()
switch {
case t.Hour() >= 9 && t.Hour() < 12:
    fmt.Println("Good Morning")
case t.Hour() >= 12 && t.Hour() < 18:
    fmt.Println("Good Afternoon")
case t.Hour() >= 18 && t.Hour() < 22:
    fmt.Println("Good Evening")
default: fmt.Println("Good Night!")
}
```

operators

//comparison
== != < <= > >=

//logical
&& || !

//arithmetical
* / % + -

//bitwise
& | ^ &^ << >>

arrays

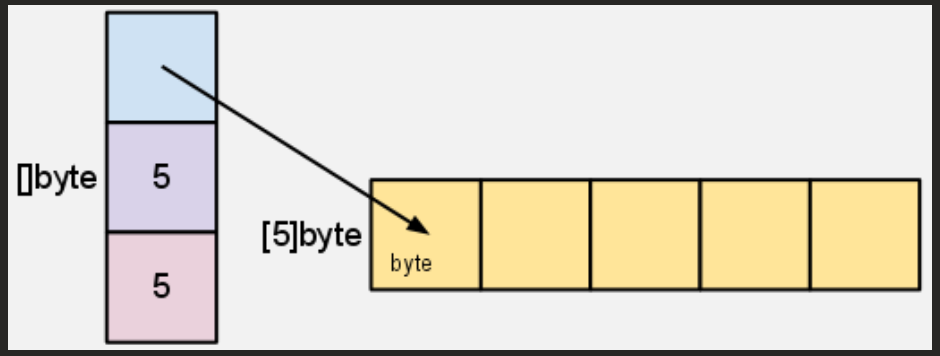
```
var a [10]byte

b := [4]int{1, 2, 3, 4}
b[0] = 100
var x [5][100]int
```

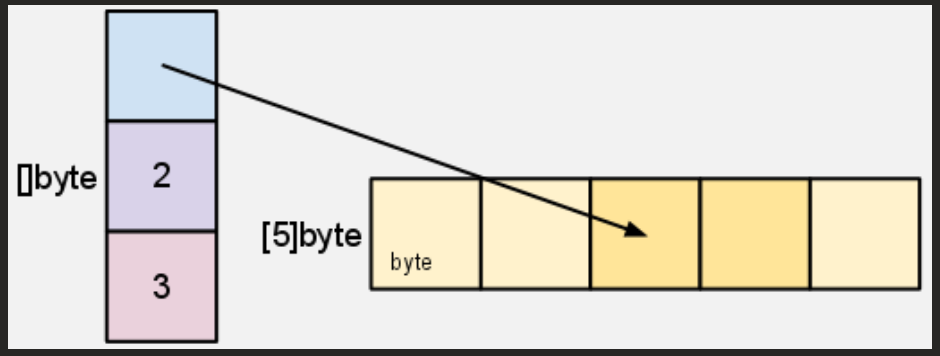
fixed size sequences
rarely used

slices

```
s := make([]string, 5, 5)
```




```
t := s[2:3]
```



https://blog.golang.org/go-slices-usage-and-internals

slices

pointers to (sub)arrays

```
s := []int{3, 4, 2, 7, 5, 6, 10, 1, 9, 8}
s[0] = 100

t := make([]string, 5, 5)
t = append(t, "foo")

x := s[1:] // all starting from second
y := s[:len(s)-2] //all without last
z := s[1:3] // elements 2,3,4
```

maps

```
m := make(map[string]int)
m := map[int]string{42: "answer", 4711: "cologne"}

m1[1] = "bar"
m1[4711] = "baz"
delete(m1, 1)

val, ok := m1[7]
if !ok {
    fmt.Printf("Key 7 not found")
}
```

loops

```
i := 0
for i < 10 {
	fmt.Println(i)
	i++
}

for i := 0; i < 10; i++ {
	fmt.Println(i)
}

for { // for ever
	fmt.Println("go")
}

arr := []string{"a", "b", "c", "d", "e", "f", "g", "h", "i", "j"}
for i, letter := range arr {
	fmt.Println(i, letter)
}
```

Only one type of loop: for

functions

```
func greet(name string) {
    fmt.Printf("Hello %s!\n", name)
}

func add(a int, b int) int {
    return a + b
}

func mult(a, b int) (c int) {
    c = a * b
    return
}


greet("World")
fmt.Println(mult(6, 7))
```

multi-value

```
func div(a, b int) (int, error) {
    if b == 0 {
        return 0, fmt.Errorf("divisor is 0", b)
    }
    return a / b, nil
}

result, err := div(1, 0)

if err != nil {
    fmt.Println(err.Error())
}

if result, err := div(1, 0); err != nil {
    fmt.Println(err.Error())
}

_, err := div(1, 0) // ignore result
```

function values

```
func compute(a, b int, fn func(a, b int) int) int {
        return fn(a, b)
}

myFunc := func(a, b int) int {
        func1 := add
        func2 := mult
        return func2(a, func1(b, 1)) // a*(b+1)
}

fmt.Println(compute(7, 6, myFunc))
```

closures

```
func makeGreeter(greeting string) func(string) string {
  count := 0
  return func(name string) string {
    count++
    return fmt.Sprintf("%s %s [%d x]", greeting, name, count)
  }
}

hello := makeGreeter("Hello")
whazzup := makeGreeter("Whazzup")

fmt.Println(hello("World"))              // => Hello World [1 x]
fmt.Println(whazzup("Karlsruhe"))        // => Whazzup Karlsruhe [1 x]
fmt.Println(whazzup("Hackschool"))       // => Whazzup Hackschool [2 x]
```

session 2 – basics

https://github.com/iigorr/go-workshop
1-hello/README.md
1-hello/CheatSheet.md

2.1 FizzBuzz

2.2 Bubble Sort

2.3 Fibonacci

2.4 Go Find

If you are stuck: please ask, peak into solutions