[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)/[Interchain Developer Academy](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)



Search

[Interchain Developer Academy](https://ida.interchain.io/)[Interchain Developer Academy](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

Search



Filters

Interchain Developer Academy

[](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 0 - Getting Started](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Getting Started](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Blockchain 101](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Blockchain History](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Public and Managed Blockchains](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Consensus in Distributed Networks](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Cryptography](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Self-Assessment Quiz](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Go Introduction - First Steps](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Go Basics](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Go Interfaces](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Control Structures in Go](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Arrays and Slices in Go](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Standard Packages in Go](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Concurrency in Go](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Good-To-Know Dev Terms](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Docker Introduction](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Introduction to the Interchain](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Blockchain Technology and the Interchain](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[The Interchain Ecosystem](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Getting ATOM and Staking It](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[A Blockchain App Architecture](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Accounts](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Transactions](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Messages](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Modules](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Protobuf](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Multistore and Keepers](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[BaseApp](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Queries](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Events](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Context](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Testing](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Relaying with IBC](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Interchain Security](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Bridges](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Migrations](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 1 Quiz](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 2 - First Steps](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[First Steps](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Setup Your Work Environment](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Run a Node, API, and CLI](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Ignite CLI](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Exercise - Make a Checkers Blockchain](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Store Object](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Create Custom Messages](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Create and Save a Game Properly](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Add a Way to Make a Move](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Emit Game Information](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Record the Game Winner](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 2 Exercise](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Introduction to IBC and CosmJS](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[What is IBC?](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[IBC/TAO - Connections (OPTIONAL)](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[IBC/TAO - Channels (OPTIONAL)](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[IBC/TAO - Clients (OPTIONAL)](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[IBC Token Transfer](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Interchain Accounts (OPTIONAL)](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[IBC Middleware (OPTIONAL)](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[IBC Tooling](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[What is CosmJS?](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Your First CosmJS Actions](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Compose Complex Transactions](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Learn to Integrate Keplr](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Create Custom CosmJS Interfaces](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 4 - Ignite CLI and IBC Advanced](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Ignite CLI and IBC Advanced](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Put Your Games in Order](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Auto-Expiring Games](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Let Players Set a Wager](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Handle wager payments](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Integration tests](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Incentivize Players](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Help Find a Correct Move](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Play With Cross-Chain Tokens](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Understand IBC Denoms](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Go Relayer](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Hermes Relayer](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 5 - CosmJS Advanced](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[CosmJS Advanced](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Create Custom Objects](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Create Custom Messages](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Get an External GUI](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Integrate CosmJS and Keplr](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Backend Script for Game Indexing](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 6 - IBC Deep Dive](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[IBC Deep Dive](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[IBC Application Developer Introduction](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Make a Module IBC-Enabled](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Adding Packet and Acknowledgment Data](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Create a Leaderboard Chain](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Run in Production](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Prepare the Software to Run](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Prepare a Validator and Keys](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Prepare Where the Node Starts](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Prepare and Connect to Other Nodes](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Prepare and Do Migrations](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Simulate Production in Docker](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Tally Player Info After Production](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Add a Leaderboard as a Module](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Simulate a Migration in Docker](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Final Exam](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[What's Next?](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

[Continue Your Interchain Journey](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html)

Docs Version Switcher

On this page

[Numbers](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#numbers)

[Strings](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#strings)

[Booleans](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#booleans)

[Type declaration](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#type-declaration)

[String formatting](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#string-formatting)

[Functions](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#functions)

[Methods](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#methods)

[Pointer](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#pointer)

[#Copy link](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#go-basics) **Go Basics**

This section introduces basic types and string formatting. After that, you will dive into functions and methods in Golang.

[#Copy link](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#numbers) Numbers

Integer types are:



Copy

int int8 int16 int32 int64

uint uint8 uint16 uint32 uint64 uintptr

* **int** will be 32 or 64 bits long depending on the OS. However, one can specify precisely how many bits are used with 8, 16, 32, and 64.
* **uint** defines the unsigned integers, which are simply positive integers.

There are two aliases:

* **byte** for **uint8**
* **run** for **int32**
* **uintptr** is an integer ["to hold the bit pattern of any pointer" (opens new window)↗](https://golang.org/pkg/builtin/#uintptr).

The types for floating-point arithmetic are float32 and float64. These are only an *approximation* for real numbers because of the [finite precision (opens new window)↗](https://en.wikipedia.org/wiki/Floating-point_arithmetic#Accuracy_problems).

complex64 and complex128 represent complex numbers. These are useful in geospatial coordinate systems and scientific applications, among others. They have "real" and "imaginary" parts that are always floats. When the real and imaginary parts are float32, the complex number is a complex64. Likewise, when the real and imaginary parts are float64, the complex number is a complex128.

[#Copy link](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#strings) Strings

In Go, a string is a read-only sequence of bytes. Therefore strings are immutable. They're encoded in UTF8 by default.

[#Copy link](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#booleans) Booleans

A bool is a special 1-bit integer. It can represent true or false.

[#Copy link](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#type-declaration) Type declaration

In Go, the name comes before the type in the declaration. There are two ways to initialize a variable in Go.

First:



Copy

var s string = "initial"

Second:



Copy

s := "initial"

You can also use **var** to define variables without initialization:



Copy

var (

a, b int

s string

c complex64

)

This is equivalent to:



Copy

var a, b int

var s string

var c complex64

Without initialization, variables have so-called *zero values* which depend on their type.

To define constants, you must use the const keyword instead of var or := keywords.



Constants can be *typed* or *untyped*. For example, an untyped constant:



Copy

const hello = "Hello, World!"

The untyped constant means that the type of hello is not defined yet.

Because of static types in Go, you have more freedom with untyped constants than with typed ones. Compare the following two examples:



Copy

const number = 2

var f float64 = number

This first example **works**: the "number" constant is untyped, so the variable "f" can accept it (despite itself being typed float64).



Copy

const number int = 2

var f float64 = number

This second example **does not work**: the "number" constant and the variable "f" are differently typed (int and float64 respectively).

[#Copy link](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#string-formatting) String formatting

fmt.Printf writes to standard output and returns the number of bytes written and the write error. The string formatting is:



Copy

%v for a value, which will be converted into a string with default options.

%T for the type of a value

%x for the hex encoding

%d for integer

%f for float, %e and %E for scientific notation

%s for string

%p for the pointer address of the variable

Here is an example code:



Copy

package main

import "fmt"

func main() {

a, b := 2, 3

c := float64(a + b)

fmt.Printf("%v + %v = %f = %v, stored as %T", a, b, c, c, c)

}



[Test it online (opens new window)↗](https://go.dev/play/p/7Uvyt1jI-5r).

Compile this to see the output.

[#Copy link](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#functions) Functions

Functions can take zero or more arguments and can return zero or more arguments. The syntax looks like the following:



Copy

func myFunc(v1, v2 type12, v3 type3, v4 type3,....) (ret1 returntype1, ret2 returntype2, ...) {

return

}

If return variable names are given in the declaration, you do not need to explicitly return them.

For example, consider a swap function that switches the values of x and y:



Copy

func swap(x, y string) (string, string) {

return y, x

}

You could also write:



Copy

func swap(x, y string) (r1 string, r2 string) {

r1, r2 = y, x

return

}

Go also offers function closures:



Copy

func fibonacci() func() int {

x, y := 0, 1

return func() int {

x, y = y, x + y

return x

}

}

Let's walk through func fibonacci() in more detail:

1. Go supports *anonymous functions*, which you return.
2. You declare x and y inside fibonacci(), and use them inside the *anonymous function*.



x, y = y, x + y works because the right side is evaluated fully before the left side.

Now write less idiomatic code to highlight some more aspects:



Copy

package main

import "fmt"

func fibonacci() func() int {

x, y := 0, 1

return func() int {

x, y = y, x + y

return x

}

}

func loop(n int, f func() int) {

if n > 0 {

fmt.Println(f())

loop(n - 1, f)

}

}

func main() {

loop(10, fibonacci())

}



[Test it online (opens new window)↗](https://go.dev/play/p/Y9LnhYgirCZ).

This will print the first 10 Fibonacci numbers.



Important here is that fibonacci() returns a function, and this function is passed into loop() as f. On subsequent iterations, loop(n-1,f) passes this anonymous function into itself recursively.

Here you used the control statement if for the first time, to break out of the recursion. Each fibonacci(), stored as f in loop, has its own x and y - this is called a **closure**. So, what happens if you split the loop into 2?



Copy

func main() {

loop(5, fibonacci())

loop(5, fibonacci())

}

This will give the first 5 Fibonacci numbers twice.

To get the first 10, try the following:



Copy

func main() {

f:= fibonacci()

loop(5, f)

loop(5, f)

}

Do you see why that works?

[#Copy link](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#methods) Methods

Methods are defined on types. Go does not have classes. First, define a structure type:



Copy

type Rectangle struct {

a, b int

}

You can use this structure for a variable declaration:



Copy

r1 := Rectangle{2, 3}

You also have access to members through the . operator:



Copy

fmt.Println(r1.a, r1.b)

Now you can declare a method on it:



Copy

func (r Rectangle) Area() int {

return r.a \* r.b

}

Methods are functions, but they have a so-called *receiver* argument (in the previous example r Rectangle). You can use such a method with the . operator:



Copy

fmt.Println(r1.Area())

Do you see how Area() became a method of Rectangle?



You can declare a method with a receiver only in the same package as the type is defined.

The following example is not declared on a struct type:



Copy

package main

import "fmt"

type MyNumber int

func (f MyNumber) Abs() MyNumber {

if f < 0 {

return -f

}

return f

}

func main() {

f := MyNumber(2)

fmt.Println(f.Abs())

}



[Test it online (opens new window)↗](https://go.dev/play/p/6aMJsTmRm2S).

Do you see how Abs() became a method of the new type, MyNumber?

[#Copy link](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#pointer) Pointer

A function argument is copied into the function. If you want to change the argument, you will require pointers. Pointers are addresses of variables. Look at an example:



Copy

func increase(i int) {

i= i + 1

}

The following function will not change i:



Copy

increase(i int)

Instead, try it this way:



Copy

package main

import "fmt"

func increase(i int) {

i = i + 1

}

func main() {

i := 0

fmt.Println(i)

increase(i)

fmt.Println(i)

}



[Test it online (opens new window)↗](https://go.dev/play/p/Wkt9tVnlcun).

The previous attempt will get the same result (0) twice. Nothing happened to i.

Now see what happens if you include a pointer:



Copy

package main

import "fmt"

func increase(i \*int) {

\*i = \*i + 1

}

func main() {

i := 0

fmt.Println(i)

increase(&i)

fmt.Println(i)

}



[Test it online (opens new window)↗](https://go.dev/play/p/nCbLaAbRa49).

Now you see that the value of i changes. What happened is as follows:

1. &i gives the address with type \*int, which is a pointer and expected by the function func increase(i \*int).
2. \*i is the value the pointer points to.

You can also use pointers in methods to modify the receiver:



Copy

package main

import "fmt"

type Rectangle struct {

a, b int

}

func (r \*Rectangle) doubleIt() {

r.a \*= 2

r.b \*= 2

}

func main() {

r := Rectangle{3, 4}

fmt.Println(r.a, r.b)

r.doubleIt()

fmt.Println(r.a, r.b)

}



[Test it online (opens new window)↗](https://go.dev/play/p/khjFSJ0hsAE).



r.b is the same as (\*r).b in this context, but it is easier to read.

Pointers are important.



This video provides a simple demonstration of functional programming in Golang, to clarify how functions interact to produce particular results.



**Further readings**

* [Go Playground (opens new window)↗](https://play.golang.org/)
* [Go by Example: Pointers (opens new window)↗](https://gobyexample.com/pointers)

synopsis

To summarize, this section has explored:

* The basic **types** (including numbers, strings, booleans, and type declarations), **string formatting**, **functions**, and **methods** employed in Golang.
* Where to access online tests to practice implementing some simple coding examples for yourself.

previous

[](https://ida.interchain.io/tutorials/4-golang-intro/1-install.html)

**[Go Introduction - First Steps](https://ida.interchain.io/tutorials/4-golang-intro/1-install.html)**

up next

**[Go Interfaces](https://ida.interchain.io/tutorials/4-golang-intro/3-interfaces.html)**

[[](https://ida.interchain.io/tutorials/4-golang-intro/3-interfaces.html)](https://ida.interchain.io/tutorials/4-golang-intro/3-interfaces.html)

Rate this Page

icon smile

icon meh

icon frown

Would you like to add a message?

Submit

Thank you for your Feedback!

On this page

[Numbers](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#numbers)

[Strings](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#strings)

[Booleans](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#booleans)

[Type declaration](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#type-declaration)

[String formatting](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#string-formatting)

[Functions](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#functions)

[Methods](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#methods)

[Pointer](https://ida.interchain.io/tutorials/4-golang-intro/2-basics.html#pointer)

#### **Get Cosmos updates**

Unsubscribe at any time. [Privacy Policy↗](https://v1.cosmos.network/privacy)

     Next

Documentation

[Cosmos SDK](https://docs.cosmos.network/)[Cosmos Hub](https://hub.cosmos.network/)[CometBFT](https://docs.cometbft.com/)[IBC Protocol](https://ibc.cosmos.network/)

Community

[Interchain blog](https://blog.cosmos.network/)[Forum](https://forum.cosmos.network/)[Discord](https://discord.gg/cosmosnetwork)

Contributing

[Source code on GitHub](https://github.com/cosmos/sdk-tutorials)

[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)

**[](https://blog.cosmos.network/)[](https://twitter.com/cosmos)[](https://discord.gg/cosmosnetwork)[](https://www.linkedin.com/company/interchain-foundation/about/)[](https://reddit.com/r/cosmosnetwork)[](https://t.me/cosmosproject)[](https://www.youtube.com/c/CosmosProject)**



Dark mode

† This website is maintained by the Interchain Foundation (ICF). The contents and opinions of this website are those of the ICF. The ICF provides links to cryptocurrency exchanges as a service to the public. The ICF does not warrant that the information provided by these websites is correct, complete, and up-to-date. The ICF is not responsible for their content and expressly rejects any liability for damages of any kind resulting from the use, reference to, or reliance on any information contained within these websites.

Cosmos is a registered trademark of the [Interchain Foundation.](https://interchain.io/)[Privacy](https://v1.cosmos.network/privacy)