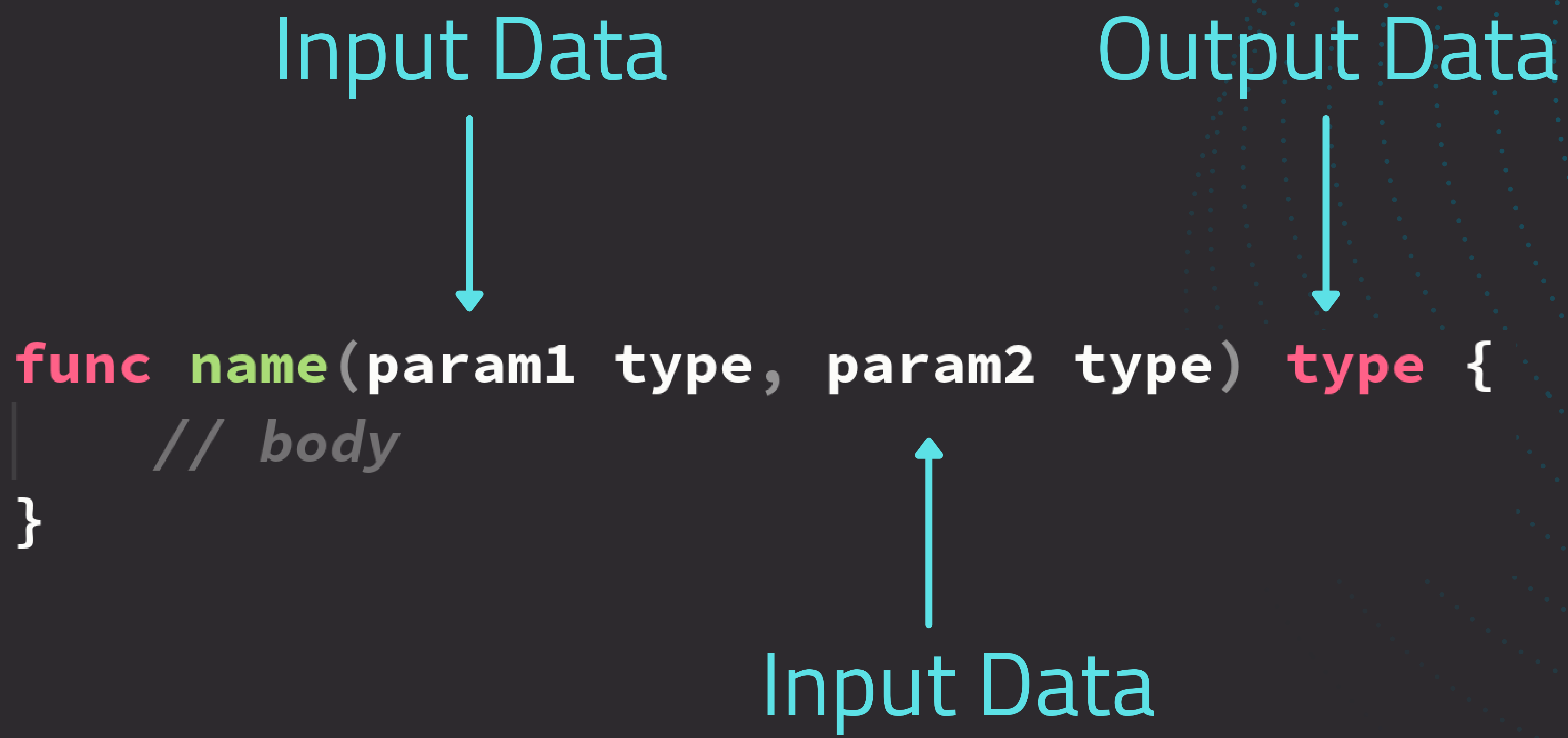
Functions

About Functions

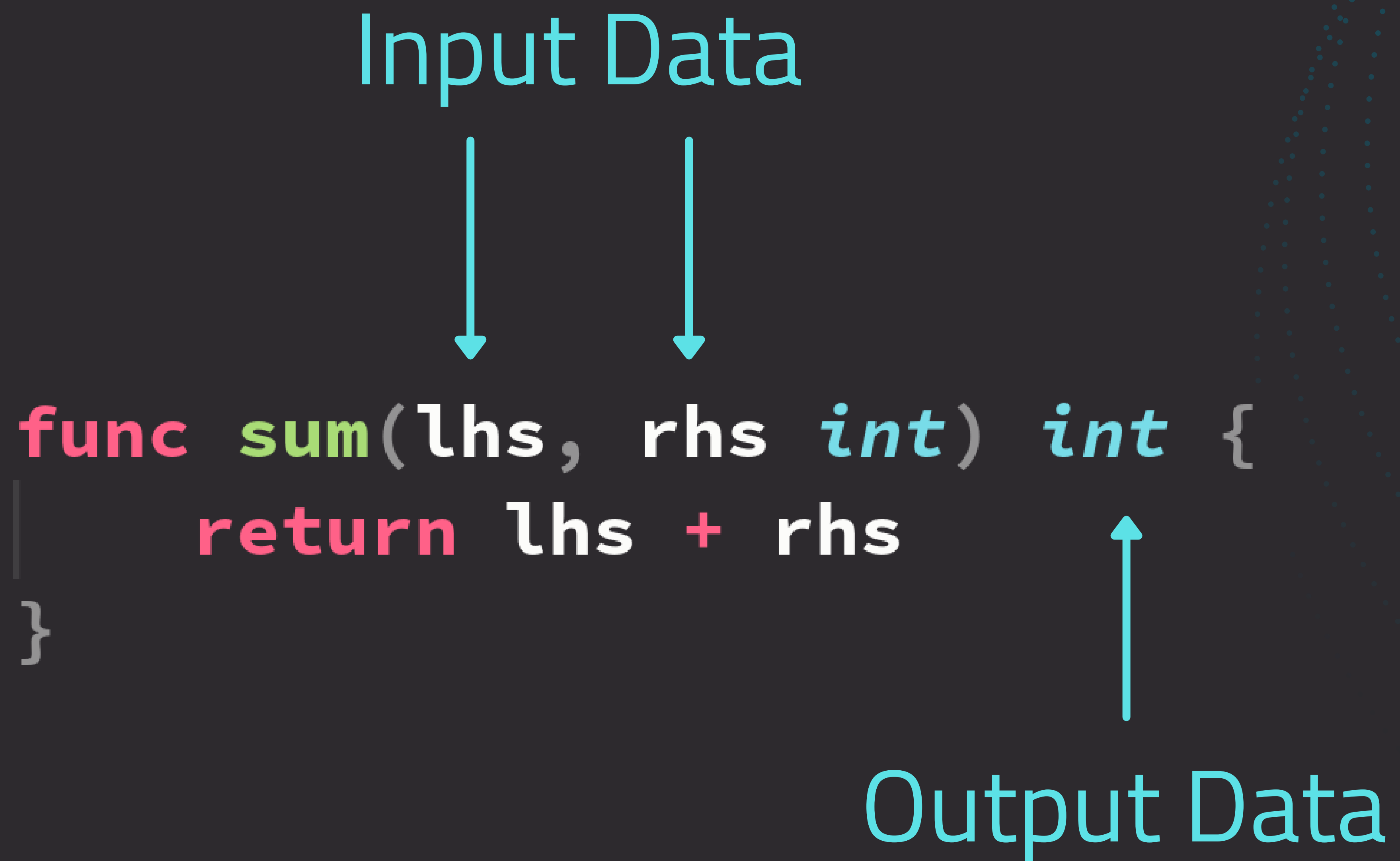
- Most basic building block of Go programs
- Allows functionality to be isolated, which makes programs easier to:
  - Test, debug, extend, modify, read, write, document
- Functions are simple: they take data as input and return data as output
  - Input and output data is optional

Creating Functions



```
func name(param1 type, param2 type) type {
    // body
}
```

Example Function



```
func sum(lhs, rhs int) int {
    return lhs + rhs
}
```

Using Functions

- Functions can be used by **calling** them
- The **caller** provides **arguments** to be utilized by the **function parameters**
  - Arguments are the input data to the function

Using Functions: Example

```go
func sum(lhs, rhs int) int {
    return lhs + rhs
}

result := sum(2, 2)
```

Multiple Return Values

```go
func multiReturn() (int, int, int) {
    return 1, 2, 3
}

a, b, _ := multiReturn()
```

Ignored

Tips

- Go function naming convention is camelCase

```go
func myReallyLongFunctionName() {}
```

- Just like variables, use names that convey the purpose of the function:

```go
// Good
func encode(data Stream, codec Codec) {}

// Bad
func compute(a, b, c float64) float64 {}
```

Recap

- Functions encapsulate program functionality which leads to more maintainable code
- Functions have **parameters** which define the input data
- Functions are used by **calling** the function and supplying **arguments**
- Functions can return multiple values
  - An underscore can be used to ignore a return value