The Move Book

A module is the base unit of code organization in Move. Modules are used to group and isolate code, and all members of the module are private to the module by default. In this section you will learn how to define a module, declare its members, and access it from other modules.

Modules are declared using the module keyword followed by the package address, module name, semicolon, and the module body. The module name should be in snake_case - all lowercase letters with underscores between words. Modules names must be unique in the package.

Usually, a single file in the sources/ folder contains a single module. The file name should match the module name - for example, a donut_shop module should be stored in the donut_shop.move file. You can read more about coding conventions in the Coding Conventions section.

If you need to declare more than one module in a file, you must use Module Block syntax.

Structs, functions, constants and imports all part of the module:

The module address can be specified as both: an address literal (does not require the @ prefix) or a named address specified in the Package Manifest . In the example below, both are identical because there's a book = "0x0" record in the [addresses] section of the Move.toml .

Addresses section in the Move.toml:

Module members are declared inside the module body. To illustrate that, let's define a simple module with a struct, a function and a constant:

The pre-2024 edition of Move required the body of the module to be a module block - the contents of the module needed to be surrounded by curly braces {} . The main reason to use block syntax and not label
syntax is if you need to define more than one module in a file. However, using module blocks is not recommended practice.

Module declaration

Modules are declared using the module keyword followed by the package address, module name, semicolon, and the module body. The module name should be in snake_case - all lowercase letters with underscores between words. Modules names must be unique in the package.

Usually, a single file in the sources/ folder contains a single module. The file name should match the module name - for example, a donut_shop module should be stored in the donut_shop.move file. You can read more about coding conventions in the Coding Conventions section.

If you need to declare more than one module in a file, you must use Module Block syntax.

```bash // Module label. module book::my_module;

// module body ```

Structs, functions, constants and imports all part of the module:

The module address can be specified as both: an address literal (does not require the @ prefix) or a named address specified in the Package Manifest . In the example below, both are identical because there's a book = "0x0" record in the [addresses] section of the Move.toml .

```
bash module 0x0::address_literal { /* ... */ } module book::named_address { /* ... */ }
```

Addresses section in the Move.toml:

```bash

# Move.toml

[addresses] book = "0x0" ```
```
```

Module members are declared inside the module body. To illustrate that, let's define a simple module with a struct, a function and a constant:

```bash module book::my_module_with_members;

// import use book::my_module;

// a constant const CONST: u8 = 0;

// a struct public struct Struct {}

// method alias public use fun function as Struct.struct_fun;

// function fun function(_: &Struct) { /*function body*/ } ```

The pre-2024 edition of Move required the body of the module to be a module block - the contents of the module needed to be surrounded by curly braces {} . The main reason to use block syntax and not label
syntax is if you need to define more than one module in a file. However, using module blocks is not recommended practice.

```bash module book::my_block_module_with_members { // import use book::my_module;

```
// a constant
const CONST: u8 = 0;

// a struct
public struct Struct {}

// method alias
public use fun function as Struct.struct_fun;

// function
fun function(_: &Struct) { /* function body */ }
```

}

// module block allows multiple module definitions in the // same file but this is not a recommended practice module book::another_module_in_the_file { // ... } ```

Address / Named address

The module address can be specified as both: an address literal (does not require the @ prefix) or a named address specified in the Package Manifest . In the example below, both are identical because there's a book = "0x0" record in the [addresses] section of the Move.toml .

```
bash module 0x0::address_literal { /* ... */ } module book::named_address { /* ... */ }
```

Addresses section in the Move.toml:

```bash

# Move.toml

[addresses] book = "0x0" ```

Module members are declared inside the module body. To illustrate that, let's define a simple module with a struct, a function and a constant:

```bash module book::my_module_with_members;

// import use book::my_module;

// a constant const CONST: u8 = 0;

// a struct public struct Struct {}

// method alias public use fun function as Struct.struct_fun;

// function fun function(_: &Struct) { /*function body*/ } ```

The pre-2024 edition of Move required the body of the module to be a module block - the contents of the module needed to be surrounded by curly braces {} . The main reason to use block syntax and not label
syntax is if you need to define more than one module in a file. However, using module blocks is not recommended practice.

```bash module book::my_block_module_with_members { // import use book::my_module;

```
// a constant
const CONST: u8 = 0;

// a struct
public struct Struct {}

// method alias
public use fun function as Struct.struct_fun;

// function
fun function(_: &Struct) { /* function body */ }
```

}

// module block allows multiple module definitions in the // same file but this is not a recommended practice module book::another_module_in_the_file { // ... } ```

## Module members

Module members are declared inside the module body. To illustrate that, let's define a simple module with a struct, a function and a constant:

```bash module book::my_module_with_members;

// import use book::my_module;

// a constant const CONST: u8 = 0;

// a struct public struct Struct {}

// method alias public use fun function as Struct.struct_fun;

// function fun function(_: &Struct) { /*function body*/ } ```

The pre-2024 edition of Move required the body of the module to be a module block - the contents of the module needed to be surrounded by curly braces {} . The main reason to use block syntax and not label
syntax is if you need to define more than one module in a file. However, using module blocks is not recommended practice.

```bash module book::my_block_module_with_members { // import use book::my_module;

```
// a constant
const CONST: u8 = 0;

// a struct
public struct Struct {}

// method alias
public use fun function as Struct.struct_fun;

// function
fun function(_: &Struct) { /* function body */ }
```

}

// module block allows multiple module definitions in the // same file but this is not a recommended practice module book::another_module_in_the_file { // ... } ```

Module block

The pre-2024 edition of Move required the body of the module to be a module block - the contents of the module needed to be
```

surrounded by curly braces {} . The main reason to use block syntax and not label
syntax is if you need to define more than one module in a file. However, using module blocks is not recommended practice.

```bash module book::my_block_module_with_members { // import use book::my_module;

// a constant
const CONST: u8 = 0;

// a struct
public struct Struct {}

// method alias
public use fun function as Struct.struct_fun;

// function
fun function(_: &Struct) { /* function body */ }

}

// module block allows multiple module definitions in the // same file but this is not a recommended practice module
book::another_module_in_the_file { // ... } ```

## Further reading
```