The Move Book

Testing is a crucial aspect of software development, especially in blockchain applications where security and correctness are paramount. In this section, we will cover the fundamentals of testing in Move, including how to write and organize tests effectively.

Tests in Move are functions marked with the #[test] attribute. This attribute tells the compiler that the function is a test function and should be run when tests are executed. Test functions are regular functions, but they must take no arguments and have no return value. They are excluded from the bytecode and are never published.

To run tests, you can use the sui move test command. This command will first build the package in test mode and then run all tests found in the package. In test mode, modules from both sources/ and tests/ directories are processed and their tests executed.

Tests for fail cases can be marked with #[expected_failure] . This attribute, when added to a #[test] function, tells the compiler that the test is expected to fail. This is useful when you want to test that a function fails when a certain condition is met.

Note: This attribute can only be added to a #[test] function.

The attribute can take an argument specifying the expected abort code that should be returned if the test fails. If the test returns an abort code different from the one specified in the argument, it will fail. Likewise, if execution does not result in an abort, the test will also fail.

The abort_code argument can use constants defined in the tests module as well as imported from other modules. This is the only case where constants can be used and "accessed" in other modules.

In some cases, it is helpful to give the test environment access to some internal functions or features. This simplifies the testing process and allows for more thorough testing. However, it is important to remember that these functions should not be included in the final package. This is where the #[test_only] attribute comes in handy.

Functions marked with the #[test_only] will be available to the test environment, and to the other modules if their visibility is set to public .

The

Tests in Move are functions marked with the #[test] attribute. This attribute tells the compiler that the function is a test function and should be run when tests are executed. Test functions are regular functions, but they must take no arguments and have no return value. They are excluded from the bytecode and are never published.

```bash module book::testing;

// The test attribute is placed before the `fun` keyword (can be both above or // right before the `fun` keyword, as in `#[test] fun my_test() { ... }`) // The name of the test in this case would be `book::testing::simple_test`.

# [test]

fun simple_test() { let sum = 2 + 2; assert!(sum == 4); }

// The name of this test would be `book::testing::more_advanced_test`.

# [test] fun more_advanced_test() {

```
let sum = 2 + 2 + 2;
assert!(sum == 4);
~~~

} ```

To run tests, you can use the sui move test command. This command will first build the package in test mode and then run all tests found in the package. In test mode, modules from both sources/ and tests/ directories are processed and their tests executed.

```bash $ sui move test

UPDATING GIT DEPENDENCY https://github.com/MystenLabs/sui.git INCLUDING DEPENDENCY Bridge
~~~

INCLUDING DEPENDENCY DeepBook INCLUDING DEPENDENCY SuiSystem INCLUDING DEPENDENCY Sui INCLUDING DEPENDENCY MoveStdlib BUILDING book Running Move unit tests ... ```

Tests for fail cases can be marked with #[expected_failure] . This attribute, when added to a #[test] function, tells the compiler that the test is expected to fail. This is useful when you want to test that a function fails when a certain condition is met.

Note: This attribute can only be added to a #[test] function.

The attribute can take an argument specifying the expected abort code that should be returned if the test fails. If the test returns an abort code different from the one specified in the argument, it will fail. Likewise, if execution does not result in an abort, the test will also fail.

```bash module book::testing_failure;

const EInvalidArgument: u64 = 1;

# [test]

# [expected_failure(abort_code = 0)]

fun test_fail() { abort 0 // aborts with 0 }

// attributes can be grouped together

# [test, expected_failure(abort_code = EInvalidArgument)]

fun test_fail_1() { abort 1 // aborts with 1 } ```

The abort_code argument can use constants defined in the tests module as well as imported from other modules. This is the only case where constants can be used and "accessed" in other modules.

In some cases, it is helpful to give the test environment access to some internal functions or features. This simplifies the testing process and allows for more thorough testing. However, it is important to remember that these functions should not be included in the final package. This is where the #[test_only] attribute comes in handy.

```bash module book::testing;

// Public function which uses the `secret` function. public fun multiply_by_secret(x: u64): u64 { x * secret() }

/// Private function which is not available to the public. fun secret(): u64 { 100 }

# [test_only]

/// This function is only available for testing purposes in tests and other /// test-only functions. Mind the visibility - for `#[test_only]` it is /// common to use `public` visibility. public fun secret_for_testing(): u64 { secret() }

# [test]

// In the test environment we have access to the `secret_for_testing` function. fun test_multiply_by_secret() { let expected = secret_for_testing() * 2; assert!(multiply_by_secret(2) == expected); } ```

Functions marked with the #[test_only] will be available to the test environment, and to the other modules if their visibility is set to public .

## Running Tests

To run tests, you can use the sui move test command. This command will first build the package in test mode and then run all tests found in the package. In test mode, modules from both sources/ and tests/ directories are processed and their tests executed.

```bash $ sui move test

> UPDATING GIT DEPENDENCY https://github.com/MystenLabs/sui.git INCLUDING DEPENDENCY Bridge INCLUDING DEPENDENCY DeepBook INCLUDING DEPENDENCY SuiSystem INCLUDING DEPENDENCY Sui INCLUDING DEPENDENCY MoveStdlib BUILDING book Running Move unit tests ... ```

Tests for fail cases can be marked with #[expected_failure] . This attribute, when added to a #[test] function, tells the compiler that the test is expected to fail. This is useful when you want to test that a function fails when a certain condition is met.

Note: This attribute can only be added to a #[test] function.

The attribute can take an argument specifying the expected abort code that should be returned if the test fails. If the test returns an abort code different from the one specified in the argument, it will fail. Likewise, if execution does not result in an abort, the test will also fail.

```bash module book::testing_failure;

const EInvalidArgument: u64 = 1;

# [test]

# [expected_failure(abort_code = 0)]

fun test_fail() { abort 0 // aborts with 0 }

// attributes can be grouped together

# [test, expected_failure(abort_code = EInvalidArgument)]

fun test_fail_1() { abort 1 // aborts with 1 } ```

The abort_code argument can use constants defined in the tests module as well as imported from other modules. This is the only case where constants can be used and "accessed" in other modules.

In some cases, it is helpful to give the test environment access to some internal functions or features. This simplifies the testing process and allows for more thorough testing. However, it is important to remember that these functions should not be included in the final package. This is where the #[test_only] attribute comes in handy.

```bash module book::testing;

// Public function which uses the `secret` function. public fun multiply_by_secret(x: u64): u64 { x * secret() }

/// Private function which is not available to the public. fun secret(): u64 { 100 }

# [test_only]

/// This function is only available for testing purposes in tests and other /// test-only functions. Mind the visibility - for `#[test_only]` it is /// common to use `public` visibility. public fun secret_for_testing(): u64 { secret() }

# [test]

// In the test environment we have access to the `secret_for_testing` function. fun test_multiply_by_secret() { let expected = secret_for_testing() * 2; assert!(multiply_by_secret(2) == expected); } ```

Functions marked with the #[test_only] will be available to the test environment, and to the other modules if their visibility is set to public .

### Test Fail Cases with
```

Tests for fail cases can be marked with #[expected_failure] . This attribute, when added to a #[test] function, tells the compiler that the test is expected to fail. This is useful when you want to test that a function fails when a certain condition is met.

Note: This attribute can only be added to a #[test] function.

The attribute can take an argument specifying the expected abort code that should be returned if the test fails. If the test returns an abort code different from the one specified in the argument, it will fail. Likewise, if execution does not result in an abort, the test will also fail.

```bash module book::testing_failure;

const EInvalidArgument: u64 = 1;

# [test]

# [expected_failure(abort_code = 0)]

fun test_fail() { abort 0 // aborts with 0 }

// attributes can be grouped together

# [test, expected_failure(abort_code = EInvalidArgument)]

fun test_fail_1() { abort 1 // aborts with 1 } ```

The abort_code argument can use constants defined in the tests module as well as imported from other modules. This is the only case where constants can be used and "accessed" in other modules.

In some cases, it is helpful to give the test environment access to some internal functions or features. This simplifies the testing process and allows for more thorough testing. However, it is important to remember that these functions should not be included in the final package. This is where the #[test_only] attribute comes in handy.

```bash module book::testing;

// Public function which uses the `secret` function. public fun multiply_by_secret(x: u64): u64 { x * secret() }

/// Private function which is not available to the public. fun secret(): u64 { 100 }

# [test_only]

/// This function is only available for testing purposes in tests and other /// test-only functions. Mind the visibility - for `#[test_only]` it is /// common to use `public` visibility. public fun secret_for_testing(): u64 { secret() }

# [test]

// In the test environment we have access to the `secret_for_testing` function. fun test_multiply_by_secret() { let expected = secret_for_testing() * 2; assert!(multiply_by_secret(2) == expected); } ```

Functions marked with the #[test_only] will be available to the test environment, and to the other modules if their visibility is set to public .

## Utilities with

In some cases, it is helpful to give the test environment access to some internal functions or features. This simplifies the testing process and allows for more thorough testing. However, it is important to remember that these functions should not be included in the final package. This is where the #[test_only] attribute comes in handy.

```bash module book::testing;
```

// Public function which uses the `secret` function. public fun multiply_by_secret(x: u64): u64 { x * secret() }

/// Private function which is not available to the public. fun secret(): u64 { 100 }

[test_only]

/// This function is only available for testing purposes in tests and other /// test-only functions. Mind the visibility - for `#[test_only]` it is /// common to use `public` visibility. public fun secret_for_testing(): u64 { secret() }

[test]

// In the test environment we have access to the `secret_for_testing` function. fun test_multiply_by_secret() { let expected = secret_for_testing() * 2; assert!(multiply_by_secret(2) == expected); } ```

Functions marked with the #[test_only] will be available to the test environment, and to the other modules if their visibility is set to public .

Further Reading