# 1.How to work with Error Handling in Solidity? (Require & Assert)

Solidity provides various functions for error handling. Generally when an error occurs, the state is reverted back to its original state. Other checks are to prevent unauthorized code access. Following are some of the important methods used in error handling −

* **assert(bool condition)** − In case condition is not met, this method call causes an invalid opcode and any changes done to state got reverted. This method is to be used for internal errors.
* **require(bool condition)** − In case condition is not met, this method call reverts to original state. - This method is to be used for errors in inputs or external components.
* **require(bool condition, string memory message)** − In case condition is not met, this method call reverts to original state. - This method is to be used for errors in inputs or external components. It provides an option to provide a custom message.
* **revert()** − This method aborts the execution and revert any changes done to the state.
* **revert(string memory reason)** − This method aborts the execution and revert any changes done to the state. It provides an option to provide a custom message.

**Assert**

The assert function should only be used to test for internal errors, and to check invariants. Properly functioning code should never reach a failing assert statement; if this happens there is a bug in your contract which you should fix. Language analysis tools can evaluate your contract to identify the conditions and function calls which will reach a failing assert.

An assert-style exception is generated in the following situations:

* If you access an array or an array slice at a too large or negative index (i.e. x[i] where i >= x.length or i < 0).
* If you access a fixed-length bytesN at a too large or negative index.
* If you divide or modulo by zero (e.g. 5 / 0 or 23 % 0).
* If you shift by a negative amount.
* If you convert a value too big or negative into an enum type.
* If you call a zero-initialized variable of internal function type.
* If you call assert with an argument that evaluates to false.

**Require**

The require function should be used to ensure valid conditions that cannot be detected until execution time. This includes conditions on inputs or return values from calls to external contracts.

A require-style exception is generated in the following situations:

* Calling require with an argument that evaluates to false.
* If you call a function via a message call but it does not finish properly (i.e., it runs out of gas, has no matching function, or throws an exception itself), except when a low level operation call, send, delegatecall, callcode or staticcall is used. The low level operations never throw exceptions but indicate failures by returning false.
* If you create a contract using the new keyword but the contract creation does not finish properly.
* If you perform an external function call targeting a contract that contains no code.
* If your contract receives Ether via a public function without payable modifier (including the constructor and the fallback function).
* If your contract receives Ether via a public getter function.
* If a .transfer() fails.

**Revert**

The revert function is another way to trigger exceptions from within other code blocks to flag an error and revert the current call. The function takes an optional string message containing details about the error that is passed back to the caller.

**Try/Catch**

A failure in an external call can be caught using a try/catch statement