SOLIDITY Function Definition

Before we use a function, we need to define it. The most common way to define a function in Solidity is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
function function-name(parameter-list) scope returns() {
  //statements
}
```

Example

Try the following example. It defines a function called getResult that takes no parameters −

```
pragma solidity ^0.5.0;

contract Test {
  function getResult() public view returns(uint){
    uint a = 1; // local variable
    uint b = 2;
    uint result = a + b;
    return result;
  }}
```

Calling a Function

To invoke a function somewhere later in the Contract, you would simply need to write the name of that function as shown in the following code.

Try the following code to understand how the string works in Solidity.

```
pragma solidity ^0.5.0;

contract SolidityTest {
```

```
constructor() public{
}
function getResult() public view returns(string memory){
   uint a = 1;
   uint b = 2;
   uint result = a + b;
   return integerToString(result);
}
function integerToString(uint _i) internal pure
   returns (string memory) {

   if (_i == 0) {
      return "0";
   }
   uint j = _i;
   uint len;

   while (j != 0) {
      len++;
      j /= 10;
   }
   bytes memory bstr = new bytes(len);
   uint k = len - 1;

   while (_i != 0) {
      bstr[k--] = byte(uint8(48 + _i % 10));
      _i /= 10;
   }
   return string(bstr);//access local variable
}}
```

Run the above program using steps provided in Solidity First Application chapter.

Output

```
0: string: 3
```

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

Try the following example. We have used a **uint2str** function here. It takes one parameter.

```
pragma solidity ^0.5.0;

contract SolidityTest {
  constructor() public{
  }
  function getResult() public view returns(string memory){
    uint a = 1;
    uint b = 2;
    uint result = a + b;
    return integerToString(result);
  }
  function integerToString(uint _i) internal pure
    returns (string memory) {

    if (_i == 0) {
      return "0";
    }
    uint j = _i;
    uint len;

    while (j != 0) {
      len++;
      j /= 10;
    }
```

```
    bytes memory bstr = new bytes(len);
    uint k = len - 1;

    while (_i != 0) {
      bstr[k--] = byte(uint8(48 + _i % 10));
      _i /= 10;
    }
    return string(bstr);//access local variable
  }}
```

Run the above program using steps provided in Solidity First Application chapter.

Output

```
0: string: 3
```

The return Statement

A Solidity function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

As in above example, we are using uint2str function to return a string.

In Solidity, a function can return multiple values as well. See the example below −

```
pragma solidity ^0.5.0;

contract Test {
  function getResult() public view returns(uint product, uint sum){
    uint a = 1; // local variable
    uint b = 2;
    product = a * b;
    sum = a + b;

    //alternative return statement to return
    //multiple values
    //return(a*b, a+b);
```

```
  }}
```

Run the above program using steps provided in Solidity First Application chapter.

Output

```
0: uint256: product 2
1: uint256: sum 3
```

Solidity-View Functions

View functions ensure that they will not modify the state. A function can be declared as **view**. The following statements if present in the function are considered modifying the state and compiler will throw warning in such cases.

Modifying state variables.

Emitting events.

Creating other contracts.

Using selfdestruct.

Sending Ether via calls.

Calling any function which is not marked view or pure.

Using low-level calls.

Using inline assembly containing certain opcodes.

Getter method are by default view functions.

Example

```
pragma solidity ^0.5.0;

contract Test {
  function getResult() public view returns(uint product, uint sum){
    uint a = 1; // local variable
    uint b = 2;
    product = a * b;
    sum = a + b;
  }}
```

Run the above program using steps provided in Solidity First Application chapter.

Output

```
0: uint256: product 2
1: uint256: sum 3
```