**Solidity Control Structures**

**Table of Contents**

* **If statement**
* **If/else statement**
* **Nested if/else statements**
* **Solidity Loops**
* **For loop**
* **While loop**
* **Do-while loop**
* **Break statement**
* **Continue statement**
* **Return keyword**

**If statement**

When writing a program, there may be a situation in which you must choose one of a number of possible options from a given set of options. It is necessary to utilise conditional statements in such situations to ensure that your program makes the correct decisions and performs the appropriate actions.

Conditional statements are supported in Solidity, and they can be used to conduct multiple actions depending on distinct conditions. In this section, we will go through the if statement.

| **pragma solidity ^0.8.0;  contract IfStatement{   function globalVariables(uint a) public returns(string memory){  if (a<10){  return "Welcome to Ineuron";  }  } }** |
| --- |

**If Else statement**

****

| **// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;  contract IfElse {  function Ineuron(uint x) public returns (string memory) {  if (x < 10) {  return "Ineuron";  } else if (x < 20) {  return "One Neuron";  } else {  return "kids neuron";  }  } }** |
| --- |

**Nested If Else Statement**

| **// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;  contract IfElse {  function Ineuron(uint x) public returns (string memory) {  if (x < 10) {  return "Ineuron";  } else if (x < 20) {  if (x<15 && x>10){  return "krish naik";  }  else if (x>15 && x<20){  return "sudh kumar";  }    } else {  return "kids neuron";  }  } }** |
| --- |

**Loops**

When writing a smart contract, you may come across a situation in which you must repeat a process over and over again. This is known as repetitive processing. You would need to write loop statements in such scenarios in order to limit the amount of lines of code.

Solidity provides support for all of the necessary loops to reduce the load of program development.

Solidity Loops

|  | **For loop** | **While loop** | **Do while loop** |
| --- | --- | --- | --- |
| **1.** | **Syntax: For(initialization; condition;updating), { . Statements; }** | **Syntax: While(condition), { . Statements; . }** | **Syntax: Do { . Statements; } While(condition);** |
| **2.** | **It is known as entry controlled loop** | **It is known as entry controlled loop.** | **It is known as exit controlled loop.** |
| **3.** | **If the condition is not true first time than control will never enter in a loop** | **If the condition is not true first time than control will never enter in a loop.** | **Even if the condition is not true for the first time the control will enter in a loop.** |
| **4.** | **There is no semicolon; after the condition in the syntax of the for loop.** | **There is no semicolon; after the condition in the syntax of the while loop.** | **There is semicolon; after the condition in the syntax of the do while loop.** |
| **5.** | **Initialization and updating is the part of the syntax.** | **Initialization and updating is not the part of the syntax.** | **Initialization and updating is not the part of the syntax** |

**For Loop**

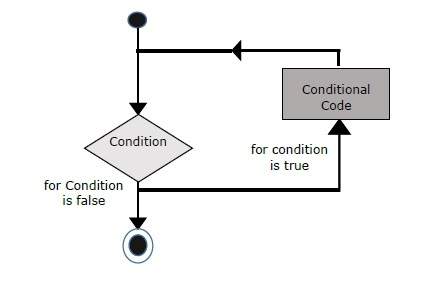
The for loop is the most compact kind of loop available. It is comprised of the three essential components listed below:

Our counter is initially set to a starting value during the loop startup process. The initialization statement is executed prior to the start of the loop's execution.

The test statement is used to determine whether a particular condition is true or false. If the condition is met, the code contained within the loop will be performed; if it is not, the control will be released from the loop.

The iteration statement is where you can change the value of your counter by increasing or decreasing it.

You can put all three pieces on a single line and split them with a semicolon if you want.



Solidity For loop syntax

| for (initialization; test condition; iteration statement) {  Statement(s) to be executed if test condition is true } |
| --- |

For loop examples

Example 1

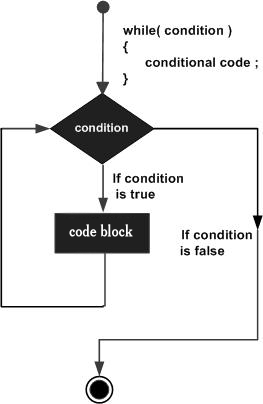
| pragma solidity ^0.8.0;    // Creating a contract contract Types {     // Declaring a dynamic array   uint[] data;     // Defining a function   // to demonstrate 'For loop'  function loop(  ) public returns(uint[] memory){  for(uint i=0; i<5; i++){  data.push(i);  }  return data;  } } |
| --- |

Example 2

| pragma solidity ^0.8.0;  contract SolidityTest {  uint storedData=10;     function getResult() public returns(string memory){  uint a = 10;   uint b = 2;  uint result = a + b;  return integerToString(result);   }   function integerToString(uint \_i)public returns (string memory) {  if (\_i == 0) {  return "0";  }  uint j=0;  uint len;  for (j = \_i; j != 0; j /= 10) { //for loop example  len++;   }  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  } } |
| --- |

**While Loop**

The while loop is the most fundamental loop in Solidity. In programming, the aim of a while loop is to continue the execution of a statement or code block as long as a certain expression is true. The loop comes to an end when the expression is found to be false.



The syntax of while loop in Solidity is as follows −

| while (expression) {  Statement(s) to be executed if expression is true } |
| --- |

Solidity While loop example

| pragma solidity ^0.8.0;    // Creating a contract  contract Types {     // Declaring a dynamic array  uint[] data;     // Declaring state variable  uint8 j = 0;    // Defining a function to   // demonstrate While loop'  function loop(  ) public returns(uint[] memory){  while(j < 5) {  j++;  data.push(j);  }  return data;  } } |
| --- |

**Do-while Loop**

Do-while loop is quite similar to the while loop with the exception that there is a condition check at the conclusion of the loop, which means that the loop will always execute at least once even if the condition is false.



Syntax:

| do  {  block of statements to be executed } while |
| --- |

Do-while loop example

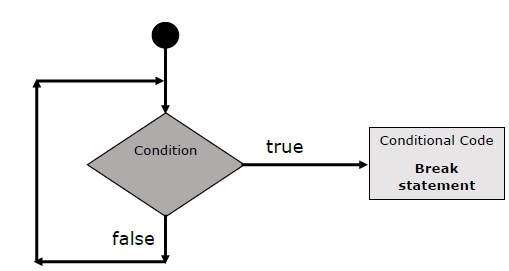
| pragma solidity ^0.8.0;    // Creating a contract  contract Types {     // Declaring a dynamic array  uint[] data;     // Declaring state variable  uint8 j = 0;    // Defining function to demonstrate   // 'Do-While loop'  function loop(  ) public returns(uint[] memory){  do{  j++;  data.push(j);  }while(j < 5) ;  return data;  } } |
| --- |

**Break and Continue Statements**

Solidity gives you complete control over how loops are handled. It is possible to find yourself in a scenario where you need to get out of a loop without reaching its bottom. Additionally, there may be a case in which you wish to bypass a section of your code block and get right into the next iteration of the loop.

Solidity includes break and continue statements to deal with any and all of these situations. These statements are used to either exit a loop immediately or begin the next iteration of a loop, depending on the situation.

**The Break Statement:**

****

**The following example illustrates the use of break statement with a while loop:**

| pragma solidity ^0.8.0;  contract SolidityTest {  uint storedData;   constructor() public{  storedData = 10;   }  function getResult() public returns(string memory){  uint a = 1;   uint b = 2;  uint result = a + b;  return integerToString(result);   }  function integerToString(uint \_i)public  returns (string memory) {    if (\_i == 0) {  return "0";  }  uint j = \_i;  uint len;    while (true) {  len++;  j /= 10;  if(j==0){  break; //using break statement  }  }  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);  } } |
| --- |

**The Continue statement:**

By using the continue statement, you can instruct the interpreter to begin the next iteration of the loop right away and bypass the remaining code block. Whenever a continue statement is met, the programme flow jumps directly to the loop check expression, and if the condition stays true, the programme flow continues to the next iteration; otherwise, the control is released from the loop and the programme ends.

This following example illustrates the use of a continue statement with a while loop:

| pragma solidity ^0.8.0;  contract SolidityTest {  uint storedData;   constructor() public{  storedData = 10;   }  function getResult() public returns(string memory){  uint n = 1;  uint sum = 0;    while( n < 10){  n++;  if(n == 5){  continue; // skip n in sum when it is 5.  }  sum = sum + n;  }  return integerToString(sum);   }  function integerToString(uint \_i) public  returns (string memory) {    if (\_i == 0) {  return "0";  }  uint j = \_i;  uint len;    while (true) {  len++;  j /= 10;  if(j==0){  break; //using break statement  }  }  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);  } } |
| --- |