ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

MODULE-2

BLOCKCHAIN AND SMART CONTRACT BASICS

Raja Rizwan Saleem
Lead Blockchain Trainer

Class-08

Structure of a contract

A contract consists of the following multiple constructs

1. State variables
2. Function definitions
3. **Enumeration definitions**
4. Structure definitions
5. Modifier definitions
6. Event declarations

3. Enumeration Continued..

Enumeration

```
pragma solidity ^0.8.0;

contract Colors {
    // Define an enumeration called Color with three possible values: RED, GREEN, and BLUE.
    enum Color {
        RED,
        GREEN,
        BLUE
    }

    function getColor() public pure returns (Color) {
        return Color.RED;  // Return the first value in our enum, which is "RED"
    }
}
```

Enumeration

```
// SPDX-License-Identifier: MIT
pragma solidity ^ 0.8.26;

contract test {
    enum FreshJuiceSize{ SMALL, MEDIUM, LARGE }
    FreshJuiceSize choice;
    FreshJuiceSize constant defaultChoice = FreshJuiceSize.MEDIUM;

    function setLarge() public {
        choice = FreshJuiceSize.LARGE;
    }
    function getChoice() public view returns (FreshJuiceSize) {
        return choice;
    }
    function getDefaultChoice() public pure returns (uint) {
        return uint(defaultChoice);
    }
}
```

Enumeration

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract EnumExample {
    enum Status { Pending, Shipped, Delivered, Canceled }

    Status public orderStatus;

    function setStatus(Status _status) public {
        orderStatus = _status;
    }

    function getStatus() public view returns (Status) {
        return orderStatus;
    }

    function cancelOrder() public {
        orderStatus = Status.Canceled;
    }
}
```

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values. They will result in **true** or **false** only

Comparison Operators

Operator	Meaning	Sample example
==	Equals	`myVar == 10`
!=	Not equals	`myVar != 10`
>	Greater than	`myVar > 10`
<	Less than	`myVar < 10`
>=	Greater than or equal to	`myVar >= 10`
<=	Less than or equal to	`myVar <= 10`

Logical Operators

Logical operators are used to determine the logic between variables or values.

Logical operators required boolean operands on both side of operator

Logical Operators

Operator	Meaning	Sample example
&&	AND	(myVar > 10) && (myVar < 10)
||	OR	(myVar > 10) || (myVar < 10)
!	NOT	!myVar

&& Logical Operator

This logical operator is used with two or more values (operands), and only evaluates to true if all the operands are true. It will return the false value if at least one value is false.

|| Logical Operator

The logical operator || (OR) also is used with two or more values, but it evaluates to true if any of the operands (values) are true, so only evaluates to false if both operands are false.

! Logical NOT

Using the ! operator in front of a boolean will convert it to opposite value. It means that a true value will return false, and a false will return true. This method is known as negation

The If decision control

The if decision control

1. Up until now, all the code in our programs has been executed chronologically
2. Very often when you write code, you want to perform different actions for different decisions.
3. You can use conditional statements in your code to do this.
4. Solidity provides conditional code execution with the help of the if...else instructions

The if decision control

1. In Solidity we have the following conditional statements:
   a. Use ***if*** to specify a block of code to be executed, if a specified condition is true
   b. Use ***else*** to specify a block of code to be executed, if the same condition is false
   c. Use ***else if*** to specify a new condition to test, if the first condition is false

Conditions: if

1. The ***if*** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

```
if(condition) {
    //Code to be executed if the condition is true
}
```

Conditions: if

```
int age = 12;
if( age > 9 ) {
    // Execute code
}
```

Conditions: else

1. Use the ***else*** statement to specify a block of code to be executed if the condition is false.

```
if(condition) {
    //Code to be executed if the condition is true
}   else {
    //Code to be executed if the condition is false
}
```

Conditions: else

```
int age = 15;
if( age > 18) {
    // Execute code
}   else {
    // Execute code
}
```

Conditions: else if

1. Use the ***else if*** statement to specify a new condition if the first condition is false.

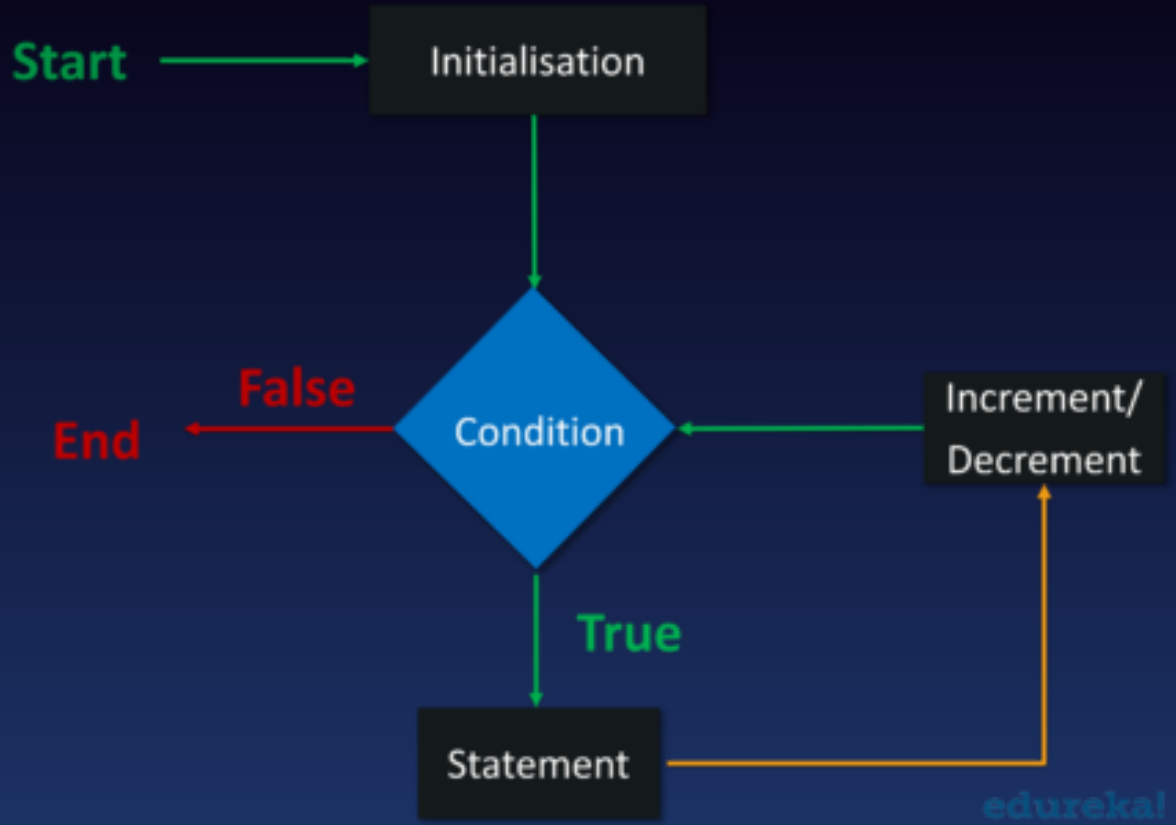
Conditions: else if

```
if (condition1) {
    //block of code to be executed if condition1 is true
} else if (condition2) {
    //block of code to be executed if the condition1 is false
and condition2 is true
} else {
    //block of code to be executed if the condition1 is false
and condition2 is false
}
```

Conditions: else if

```
int score = 80;
if( score > 80 ) {
// Execute code

} else if( score > 70 ) {
    // Execute code
} else {
    // Execute code
}
```


Start
Initialisation
False
End
Condition
Increment/
Decrement
True
Statement
edureka!

Solidity Loops

While writing a contract, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

While, Do-While, and For Loop

https://www.geeksforgeeks.org/solidity-while-do-while-and-for-loop/

THANK-YOU