ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

Module

ONE

JAVASCRIPT CRASH COURSE

Instructor: Raja Rizwan Saleem

SESSION-7

Functions

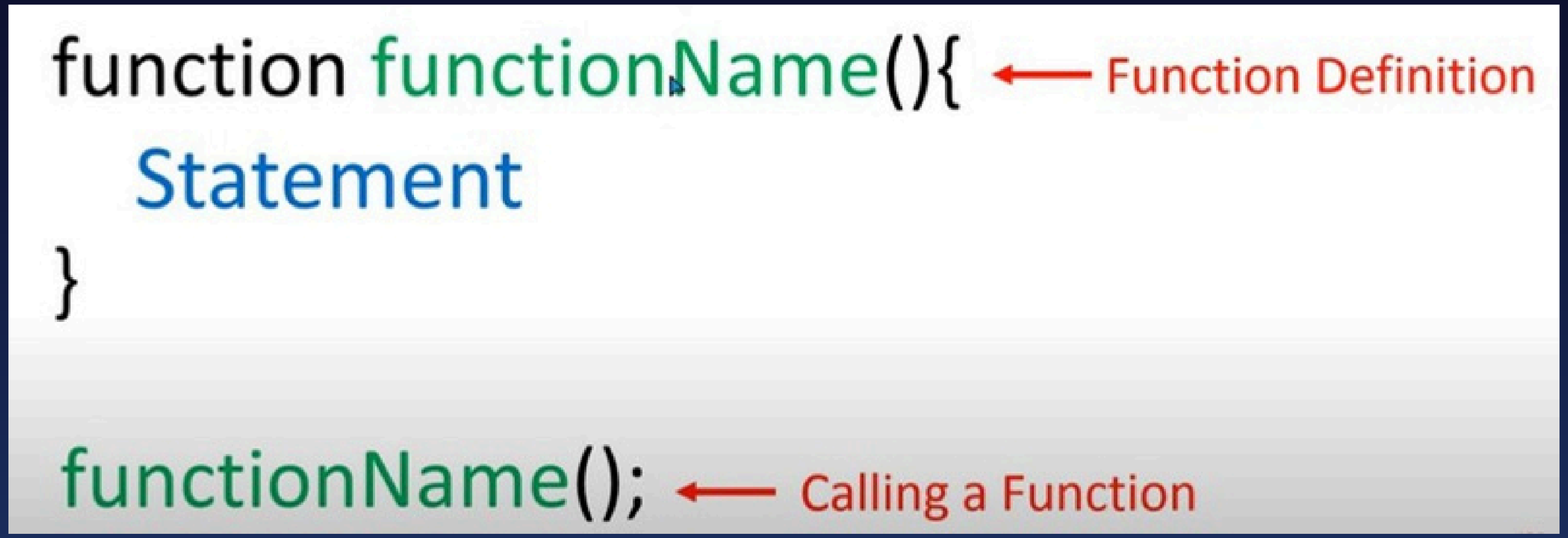
Functions

1. A function is a block of JavaScript that does the same thing again and again.
2. A JavaScript function is executed when "something" invokes it (calls it).
3. It saves you repetitive coding and makes your code easier to understand.
4. You can reuse code: Define the code once, and use it many times

Function Declarations

1. A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
2. Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
3. The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)
4. The code to be executed, by the function, is placed inside curly brackets: {}

What is function in JS



What is function in JS

```
function hello(){
    document.write("Aslam-o-alikum");
}
hello();
```

Function Declarations

```
function sum(a, b){
    return a + b;
}
```

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

What is function in JS

As talked earlier that JavaScript function is a block of code designed to perform a particular task. A function is defined with the function keyword, followed by a name, followed by parentheses ().

```
function addition(){
      var a = 5; var
      b = 6; return
      a + b;
} var c = addition();
console.log(c);
```

Function Declarations

```
function name(parameter1, parameter2) {
    // code to be executed
}
```

What is function in JS

As talked earlier that JavaScript function is a block of code designed to perform a particular task. A function is defined with the function keyword, followed by a name, followed by parentheses ().

```
function sum(a,b){
    var a = 10;
    var b = 6;
    return a + b;
 } var c = sum();
console.log(c);
```

Invoking a Function

1. Functions execute when the function is called.
2. This process is known as invocation.
3. You can invoke a function by referencing the function name, followed by an open and closed parenthesis: ().

Invoking a Function

1. Declarations

```
function showMessage(message){
    console.log(message);
}
```

2. Invoking

```
showMessage("Hello World");
```

Parameters vs. Arguments

1. Parameters:
    a. Function parameters are listed inside the parentheses () in the function definition.
2. Arguments:
    a. Function arguments are the values received by the function when it is invoked.

Parameters vs. Arguments

1. Declarations

```
function showMessage(message){
    console.log(message);
}
```

Parameter

2. Invoking

```
showMessage("Hello World");
```

Argument

Passing Data to Function

1. In order for a function to become a programmable robot rather than a one-job robot, you have to set it up to receive the data you're passing. You can pass any type of
2. data to function depending on requirement

Passing Data to Function (Numeric values)

```
function multiply(num1, num2){
    var num3 = num1 * num2;
    console.log("Num3 ",num3);
    }

    multiply(3,6);
    multiply(4,2);
```

Passing Data to Function (String values)

```
function showMessage(name){
    console.log("Hello "+name);
}

showMessage("Rizwan");
showMessage("Saleem");
```

Parameter Rules

1. JavaScript function definitions do not specify data types for parameters.
2. JavaScript functions do not perform type checking on the passed arguments.
3. JavaScript functions do not check the number of arguments received.
4. If a function is called with missing arguments (less than declared), the missing values are set to: ***undefined***

Parameter Rules

```
function showMessage(name){
    console.log("Hello "+name);
}
showMessage("Rizwan");
showMessage(45);
showMessage(true);
showMessage();
```

Function Return

1. Function can returns data back to caller
2. After executing logic in function if you want to return result to the caller of function then you use ***return*** keyword
3. When JavaScript reaches a ***return*** statement, the function will stop executing and return value is "returned" back to the "caller"
4. Every function in JavaScript returns ***undefined*** unless otherwise specified

Function Return

```
function test(){


}

var a = test(); // return undefined
console.log(a); // undefined
```

Function Return

In this example we explicitly tell the function to return 45

```
function test(){

    return 45;
}
var a = test();          //  return 45
console.log(a);          //  45
```

Function Return

```
 function multiply(num1,  num2){
     return num1 * num2;
}
var a = multiply(3,6);          //returns    18
var b = multiply(4,2);          //returns    8
console.log(a);                 //18
console.log(b);                 //8
console.log(multiply(2,5)); // 10
```

Function Return

```
function multiply(num1, num2){
    return num2; // function execution ends here
    return num1 * num2;

}
var a = multiply(3,6);
console.log(a);
```

Function in Expressions

1. JavaScript functions can be used in expressions
2. Just like we use variables in calculation we can use function and output function will be included in calculation

```
function multiply(num1, num2){
      return num1 * num2 ;
} var a = multiply(3,4) + 5
console.log(a);
```

Function in Expressions

```
function multiply(num1, num2){
    return num1 * num2;

}
function sum(a, b){
    // Result of multiply sum with value of b
    return multiply(a,b) + b;//16

}
var total = sum(3,4) + 6; // result 22
```

Function in Expressions

```
function multiply(num1, num2){
    return num1 * num2;

}
function sum(a, b){
    return a + b;

}
// Call multiply first and result passed to sum
var total = sum(multiply(3,4), 2) + 6; // result 20
```

Local vs Global Variables

1. Variables can have local or global scope
2. A global variable is one that's declared in the main body of your code, not inside a function.
3. A local variable is one that's declared inside a function.
4. A local variable can be either a parameter of the function, which is declared implicitly by being named as a parameter, or a variable declared explicitly in the function with the var keyword.

Local vs Global Variables

5. Global variable is meaningful in every section of your code, whether that code is in the main body or in any of the functions.
6. Local variable is one that's meaningful only within the function that declares it.

Local vs Global Variables

7. There are two differences between global and local variables—where they're declared, and where they're known and can be used.

Global Variables	Local Variables
Declared in the main code	Declared in a function
Known everywhere, useable everywhere	Known only inside the function, usable only inside the function

Local vs Global Variables

```
var a = 7;                          // Global Variable
function sum(){

    var b = 6;                      // Local Variable
    var c = a + b;                  // 13, Accessing global
    console.log("C "+c);
}
sum();
console.log("A = "+a);// 7
```

Local vs Global Variables

```
var a = 7;                          // Global Variable
function sum(){

    var b = 6;                      //Local Variable
    a = b + 5;
    console.log("A  "+a);//Accessing globalvariable
}
sum();
console.log("A = "+a);      // 11, value of a updated
```

Local vs Global Variables

```
var a = 7;                          // Global Variable
function sum(){

    var b = 6;                      // Local Variable
    a = b + 5;

}
sum();
console.log("B = "+b);
// error, b is not available outside sum function
```

Local vs Global Variables

```
var a = 7;                          // Global Variable
function sum(){

    var b = 6;                      // Local Variable
    a = b + 5;
    console.log("A "+a);// Accessing global variable
}
sum();
console.log("A = "+a);   // 11, value of a updated
```

Global Variables without var keyword

```
var a = 7;                          // Global Variable
function sum(){
    var b = 6;                      // Local Variable
    a = b + 5;

}
sum();
console.log("B = "+b);
// error, b is not available outside sum function
```

Global Variables without var keyword

```
a = 7;    // Without var still Global Variable
function sum(){
    b = 6;    // Global variable because its without var
    a = b + 5;
    console.log("A "+a);// Accessing global variable
}
sum();
console.log("B "+b);//b available outside of function
```

Function Expressions

1. A JavaScript function can also be defined using an expression.
2. A function expression can be stored in a variable

```
var sum = function (a, b){
    return a + b;

};
var c = sum(4,5);
console.log(c);
```

```
// function as expression
```

Function Expressions

1. After a function expression has been stored in a variable, the variable can be used as a function
2. The function in expression is actually an anonymous function (a function without a name).
3. Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

Function Expressions

```
var square = function(num) {
return num * num;
};
var b = square(4); // 16
```

Notice that function above ends with semicolon because it is part of executable statement

Function Hoisting

1. Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope.
2. Hoisting applies to variable declarations and to function declarations.
3. Because of this, JavaScript functions can be called before they are declared

Function Hoisting

```
var total = sum(5,6); //Calling before declaration
console.log("Sum = "+total);

function sum(a, b){
    return a + b;
}
```

Arguments Passed by Value

1. Primitive data is passed by value: The function only gets to know the values, not the argument's locations.
2. If a function changes an argument's value, it does not change the parameter's original value.
3. Changes to arguments are not visible (reflected) outside the function.

Arguments Passed by Value

```
var num = 5;
function changeValue(a){
      a = 7;// change to a will not affect num
}

changeValue(num);
console.log(num);//5, num will be updated
```