Pakistan Blockchain Institute

MODULE-1

JAVASCRIPT CRASH COURSE

Class-8

Edversity.

Raja Rizwan Saleem
Lead Blockchain Trainer

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

```
function functionName(){
    Statement
}

functionName();
```

← Function Definition

← Calling a Function

What is function in JS

```
function hello(){
    console.log("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.

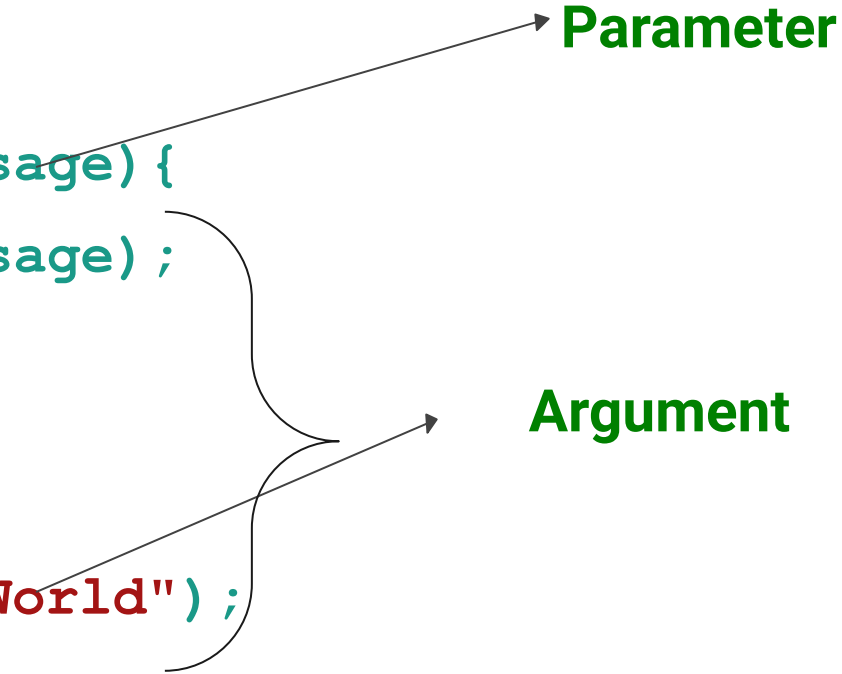
Parameters vs. Arguments

1. Declarations

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

}
```

2. Invoking

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



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.
2. You can pass any type of 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("Edversity");
showMessage("Pakistan Blockchain Institute");
```

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("Zain Shabbir ");      // Hello Zain Shabbir
showMessage(45);                   // Hello 45
showMessage(true);                 // Hello true
showMessage();                     // Hello undefined
```

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

```
function add(a, b) {
    return a + b;
}
console.log(add(2, 3));
```

Function Return (function in variable)

```
const multiply = function(a, b) {
    return a * b;
};
console.log(multiply(2, 3));
// Output: 6
```

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);    // returns 6
console.log(a);           // 6
```

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
```

Call back Function

```
function greet(name, callback) {
    callback(name);
}

function sayHello(name) {
    console.log(`Hello, ${name}!`);
}

greet("Raja Rizwan", sayHello); // Output: Hello, Raja Rizwan!
```

Higher Order Function

```
function makeMultiplier(x) {
    return function(y) {
        return x * y;
    };
}

const multiplyBy2 = makeMultiplier(2);
console.log(multiplyBy2(5)); // Output: 10
```

Function Return closure / counter

```
function makeCounter() {
    let count = 0;
    return function() {
        count++;
        return count;
    };
}

const counter = makeCounter();
console.log(counter()); // Output: 1
console.log(counter()); // Output: 2
```

Function default parameters

```
function greet(name = "Guest") {
    console.log(`Hello, ${name}!`);
}
greet(); // Output: Hello, Guest!
greet("Rizwan"); // Output: Hello, Rizwan!
```

Function - Rest parameters

```
function sum(...numbers) {
    return numbers.reduce((total, num) => total + num, 0);
}
console.log(sum(1, 2, 3, 4)); // Output: 10
```

Function - Spread Operators

```
function joinStrings(...strings) {
    return strings.join(' ');
}
const words = ['Hello', 'world', 'from', 'JavaScript'];
console.log(joinStrings(...words));
// Output: Hello world from JavaScript
```

Local vs Global Variables

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

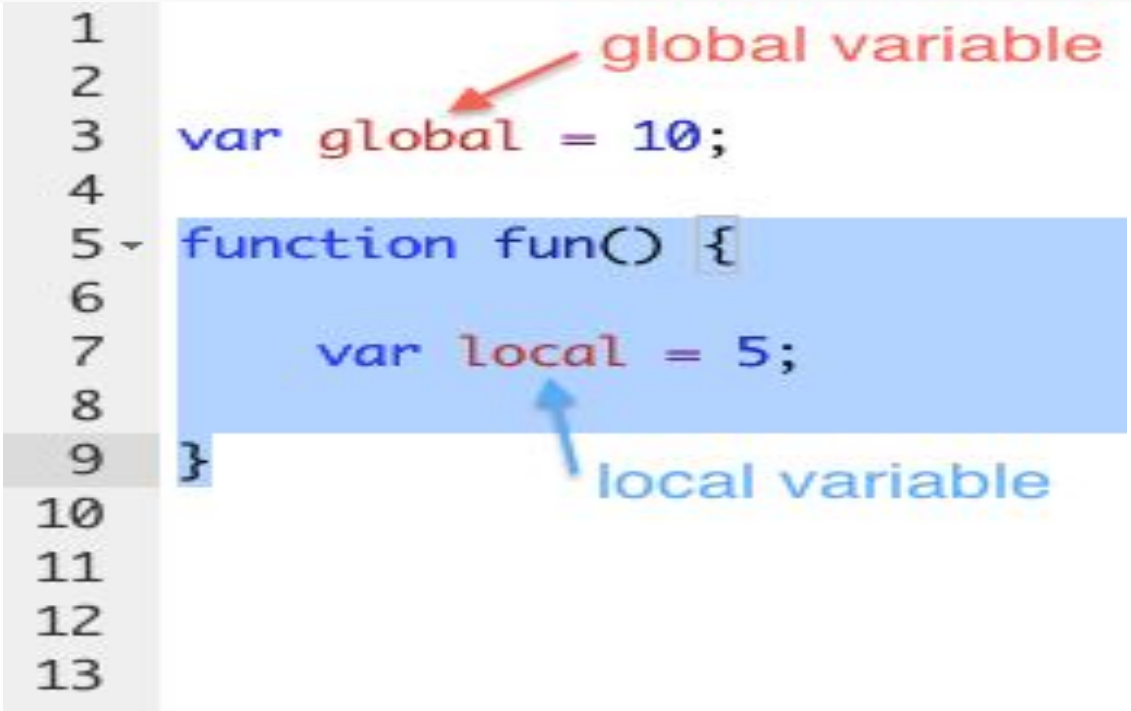
5. Global variable is meaningful in every section of your code, whether that code is in the main body or in any of the functions.
6. Local variable is one that's meaningful only within the function that declares it.

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



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 global variable
}
sum();
console.log("A = "+a);   // 11, value of a updated
```

Local vs Global Variables

```
var a = 7;                          // Global Variable
function sum(){
    var b =                        // Local Variable
    6;  a = b
    + 5;
}
sum();
// error: b is not available outside sum function
console.log(b);
```

Local vs Global Variables

```
var a = 7;                          // Global Variable
function sum(){
  var a = 6;      // Local Variable a same name as global
  a = 3 + 2;      // Local a variable will be affected
  console.log("A "+a);//5, access local variable
}
sum();
console.log("A "+a);//7, access global variable
```

Global Variables without var keyword

```
a = 7;      // Without var still 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

```
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){   // function as expression
    return a + b;

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

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

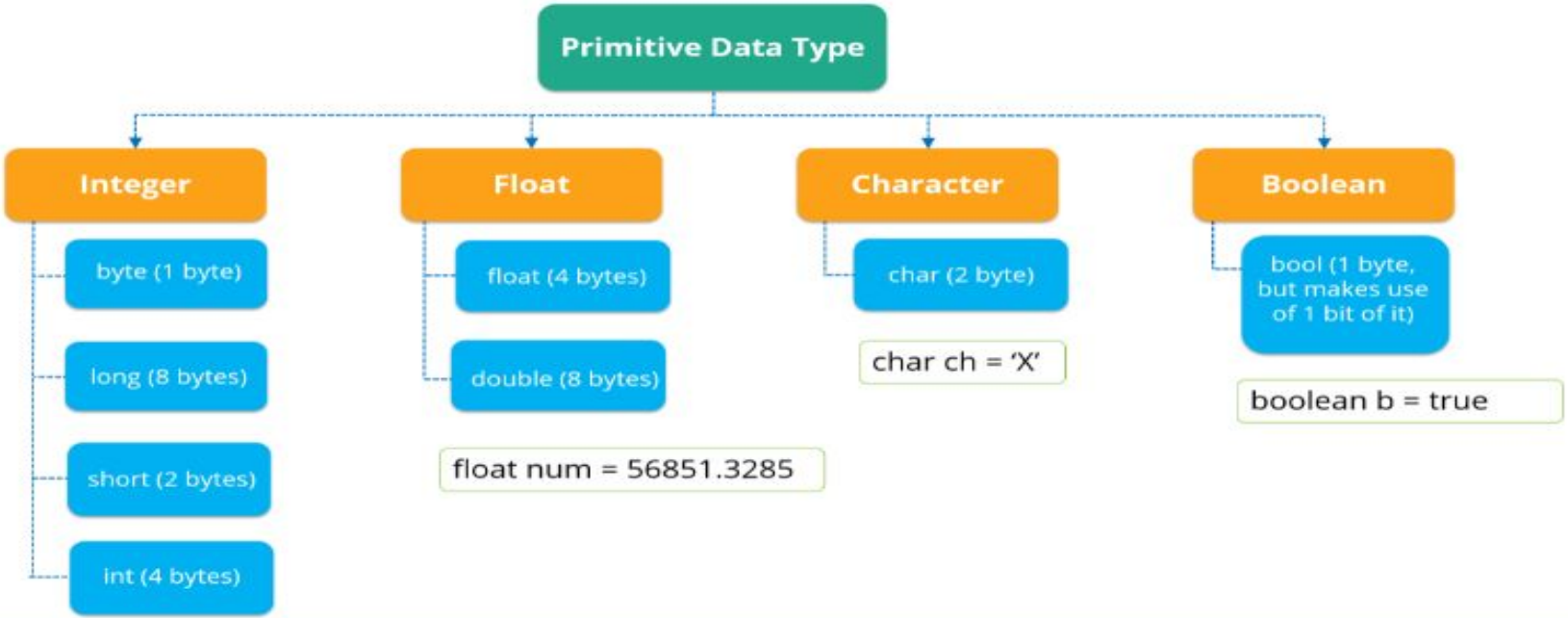
```
var num = 5;
function changeValue(a){
    a = 7;  // change to a will not affect num
}

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

Primitive Data Types

A primitive data type is pre-defined by the programming language. The size and type of variable values are specified, and it has no additional methods.Data types in Javascript are classified into 4 aspects as int, float, character and boolean. But, in general, there are 8 data types.

Primitive Data Types

They are as follows:



Non-Primitive Data Types

These data types are not actually defined by the programming language but are created by the programmer. They are also called “reference variables” or “object references” since they reference a memory location which stores the data.

Non-Primitive Data Types

Non-Primitive data types refer to objects and hence they are called **reference types.** Examples of non-primitive types include Strings, Arrays, Classes, Interface, etc. Below image depicts various non-primitive data types.

https://www.edureka.co/blog/data-types-in-java/

Arguments Passed by Reference

1. In JavaScript, object references are values.
2. Non-primitive value such as Array or a user-defined object are passed by reference
3. If function changes the object's properties, that change is visible outside the function

Arguments Passed by Reference

```
var arr = [4, 6, 7, 9];
function updateArray(val){ // array received in val
    val[1] = 57;  // updating val will also update arr
}
console.log(arr[1]);          // 6 before calling function
updateArray(arr);
console.log(arr[1]);          // 57 after calling function
```

Arguments Passed by Reference

```
var obj = { name: "Laila Kashan", age:56 };
function updateObject(val){ // object received in val
    val.age = 40; // updating val will also update arr
}
console.log(obj.age);   // 56 before calling function
updateObject(obj);
console.log(obj.age);   // 40 after calling function
```

Recursive Function

1. A recursive function is a function that calls itself.
2. Recursion is a technique for iterating over an operation by having a function call itself repeatedly until it arrives at a result.
3. In some ways, recursion is analogous to a loop. Both execute the same code multiple times, and both require a condition (to avoid an infinite loop, or rather, infinite recursion in this case)

Recursive Function

1. The classic example of a function where recursion can be applied is the factorial.
2. Factorial of a number n can be defined as product of all positive numbers less than or equal to n.
3. It is the multiplying sequence of numbers in a descending order till 1. It is defined by the symbol of exclamation (!).
4. E.g factorial of 6 is
    a. $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$

Recursive Function (revised)

```
function factorial(n)
    {  if (n <= 1) {
    // Recursion will stop when this condition match
        return 1;
    } else {
        return n * factorial(n - 1); // calling itself
    }
}
```

Switch Statement

1. The switch statement executes a block of code depending on different cases.
2. The switch statement is a part of JavaScript's "Conditional" Statements, which are used to perform different actions based on different conditions.
3. Switch statement works for equality checks only, you can not apply range, greater than or less then checks

Switch Statement

4. This is how it works:
    a. The switch statement evaluates an expression.
    b. The value of the expression is then compared with the values of each case in the structure.
    c. If there is a match, the associated block of code is executed.
5. The switch has one or more case blocks and an optional default.

Switch Syntax

```
switch(expression) {
    case 'value1':   // same as if (expression === 'value1')
        // code block
        break;
    case 'value2':
        // code block
        break;
    default:
        // code block
}
```

Switch Statement

```
var day = 3;
switch (day) {
    case 6:
       console.log("Today is Saturday");  break;
    case 0:
       console.log("Today is Sunday");  break;
    default:
        console.log("Looking forward to the Weekend");
}
```

Switch - Grouping of case

1. Sometimes you will want different cases to use the same code, or fall-through to a common default.
2. If you skip the break and expression match to the case where there is no break then it will also fall-through the next case

Switch - Grouping of case

```
var day = 3;  switch (day)
{
    case 6: // No break  console.log("Today is
        Saturday");
    case 0: // No break in last case, both will execute
       console.log("Today is Sunday");
       break;  default:
        console.log("Looking forward to the Weekend");
}
```

Switch - Grouping of case

```
var day = 3;
switch (day) {
    case 6:
    case 0:
    Case 5:
       console.log("Yaaaa! It's Weekend");
       break;
    default:
        console.log("Looking forward to the Weekend");
}
```

Switch - Strict Comparison

1. Switch cases use strict comparison (===).
2. The values must be of the same type to match.
3. A strict comparison can only be true if the operands are of the same type.

Switch - Strict Comparison

```
var x = "0";  switch
(x) {
    case 0:
        console.log("Off");
        break;
    case 1:
        console.log("On");
         break;
    default:   // this will execute as value did not match
        console.log("No value found");
}
```

Online Exercises

Below link provide you online exercises

http://asmarterwaytolearn.com/js/index-of-exercises.html

Online Exercises

Below link provide you online exercises

https://www.w3schools.com/js/default.asp

Thanks

End of Module-2