ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

Module

ONE

JAVASCRIPT CRASH COURSE

Instructor: Raja Rizwan Saleem

SESSION-7

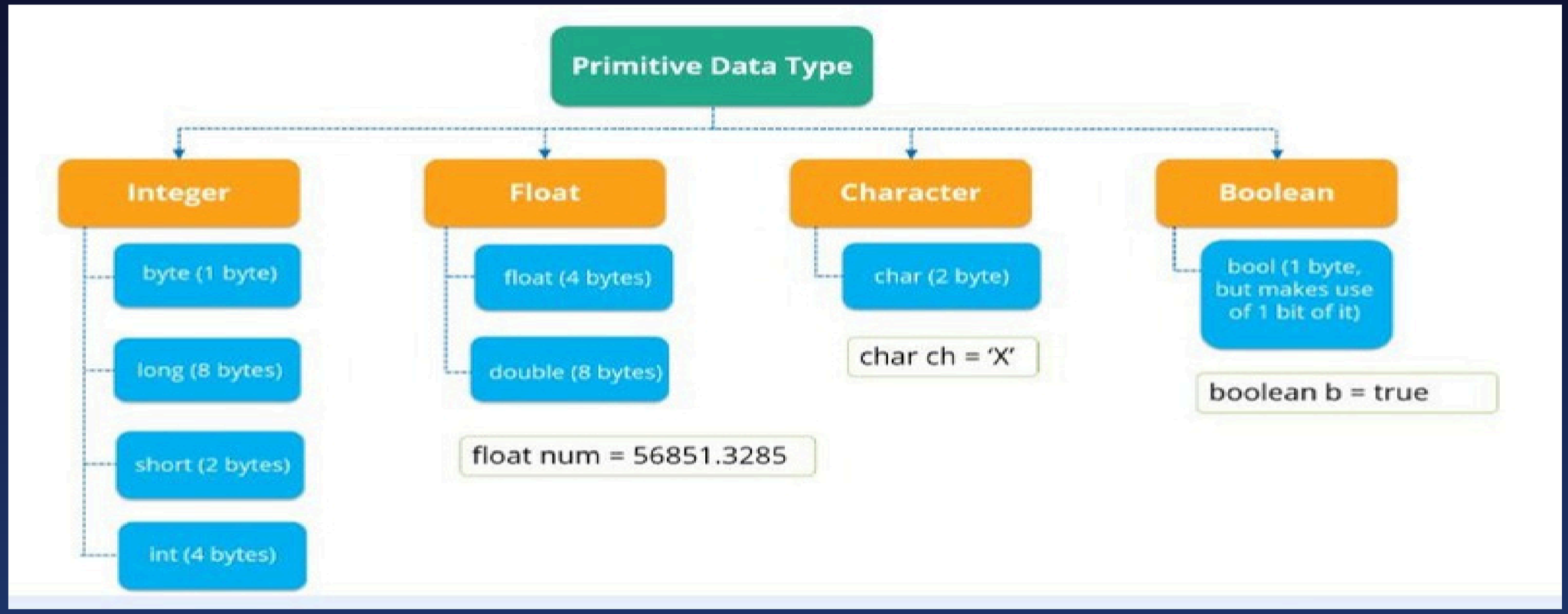
Primitive & Non- Primitive Data Types

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/

Primitive Non-Primitive Data Types

Primitive data structure	Non-primitive data structure
Primitive data structure is a kind of data structure that stores the data of only one type.	Non-primitive data structure is a type of data structure that can store the data of more than one type.
Examples of primitive data structure are integer, character, float.	Examples of non-primitive data structure are Array, Linked list, stack.
Primitive data structure will contain some value, i.e., it cannot be NULL.	Non-primitive data structure can consist of a NULL value.
The size depends on the type of the data structure.	In case of non-primitive data structure, size is not fixed.
It starts with a lowercase character.	It starts with an uppercase character.
Primitive data structure can be used to call the methods.	Non-primitive data structure cannot be used to call the methods.

Primitive Data Types - Number

```
let x = 250;
let y = 40.5;
console.log("Value of x=" + x);
console.log("Value of y=" + y);
```

Primitive Data Types - String

```
let str = 'Hello All';
let str1 = "Welcome to my new house";
console.log("Value of str=" + str);
console.log("Value of str1=" + str1);
```

Primitive Data Types - Undefined

This means that a variable has been declared but has not been assigned a value, or it has been explicitly set to the value `undefined`.

```
let x;
console.log(x); // Outputs: undefined
```

Primitive Data Types - Null

This data type can hold only one possible value that is null.

Example: Below is an example.

```
let x = null;
    console.log("Value of x=" + x);
```

Output

```
Value of x=null
```

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: "Arslan", 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 - Linear sum

```
function sum(number) {
    if (number===0) {
        return 0;
    }
    return number + sum(number-1);
}

sum(10);
```

Recursive Function - Linear sum

1. The function takes an integer `number` as an argument.
2. If `number === 0`, it returns `0`. This is the **base case**, which stops the recursion.
3. Otherwise, it returns `number + sum(number - 1)`, calling itself with a decremented value.
4. This recursive call continues until `number` reaches `0`, at which point the recursion stops and the results are added up.

Recursive Function - Linear sum

```
sum(10) = 10 + sum(9)
sum(9)  =  9 + sum(8)
sum(8)  =  8 + sum(7)
sum(7)  =  7 + sum(6)
sum(6)  =  6 + sum(5)
sum(5)  =  5 + sum(4)
sum(4)  =  4 + sum(3)
sum(3)  =  3 + sum(2)
sum(2)  =  2 + sum(1)
sum(1)  =  1 + sum(0)
sum(0)  =  0  (Base Case)
```

```
sum(1)  =  1 + 0  = 1
sum(2)  =  2 + 1  = 3
sum(3)  =  3 + 3  = 6
sum(4)  =  4 + 6  = 10
sum(5)  =  5 + 10 = 15
sum(6)  =  6 + 15 = 21
sum(7)  =  7 + 21 = 28
sum(8)  =  8 + 28 = 36
sum(9)  =  9 + 36 = 45
sum(10) = 10 + 45 = 55
```

Recursive Function - Linear sum

Final Result

`sum(10) = 55`

This function calculates the sum of the first `n` natural numbers, which follows the formula:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

For `n = 10`:

$$\frac{10(10+1)}{2} = \frac{10(11)}{2} = 55$$

Thus, `sum(10)` correctly returns `55`.

Recursive Function - Linear sum for Even Numbers

```
function sumEven(number) {
    if (number===0) {
        return 0;
    } else if (number%2 !== 0) {
        return sumEven(number-1);
    }
    return number + sumEven(number-1);
}
sumEven(10);
```

Recursive Function - Linear sum for Even Numbers

What is Recursion in JavaScript?

By Benjamin Semah Recursion is a problem-solving technique in programming. In this article, you will learn how to use recursive functions in JavaScript. What is a Recursive Function? A recursive function is a...

freeCodeCamp.org / Nov 14, 2022

Switch Statements

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:
        console.log("Today is Saturday");
        break; // Added break to prevent fall-through
    case 0:
        console.log("Today is Sunday");
        break; // Added break to stop execution after case 0
    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": // Match type correctly
        console.log("Off");
        break;
    case "1": // Match type correctly
        console.log("On");
        break;
    default:
        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

THANK-YOU

End of Module-1