ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

www.edversity.com.pk

Module

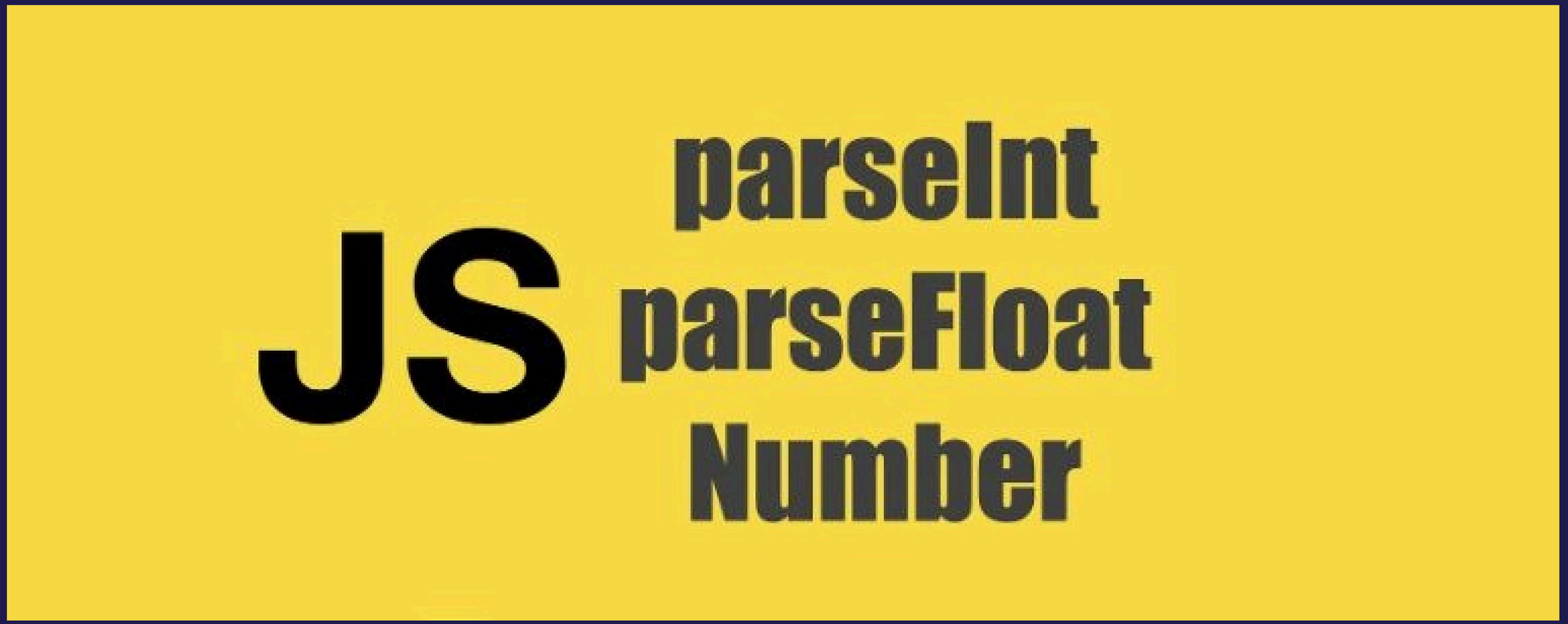
ONE

JAVASCRIPT CRASH COURSE

Instructor: Raja Rizwan Saleem

SESSION-4

Convert string to integer



Convert string to integer

`parseInt` is a built-in JavaScript function that converts a string to an integer. It is useful for converting user input or data from other sources into a numeric format.

Convert string to integer

parseFloat is a function that is used to parse a string and extract a floating-point number(a number with decimal places). It will read from the beginning of the string until it encounters a character that is not a valid part of a floating-point number. parseFloat will always return a floating-point number, even if the input string contains integer values. This is a great choice when you need to parse strings that might contain decimals that you want to preserve

Convert string to integer

parseInt() stops parsing as soon as it encounters a non-numeric character and returns the parsed integer value up to that point. **parseFloat()** continues parsing until it reaches the end of the string or encounters an invalid character, and it returns the parsed floating-point value up to that point

Convert string to integer

Financial Analysts: Those working in the finance sector, such as financial analysts and accountants who require precise calculations involving currency or financial data. This ensures the accuracy when dealing with monetary values.

Engineers: Those working in the engineering sector often work with data that includes real-world measurements. They may use parseFloat to ensure accuracy with coordinates, angles, or simulations.

Convert string to integer

1. If you are sure that string has number, then you have to convert it into number to perform addition
2. We need to use **parseInt** function for conversion

```
var age = prompt("What is your age");
var num = 4;
var sum = parseInt(age) + num ;

console.log(sum);
```

Convert string to integer `parseInt`

1. If value is not number then it will return NaN

```
var age = prompt("What is your age");//input abc
var num = 4;
var sum = parseInt(ag)   + num;

console.log(sum); // result, NaN
```

Convert string to integer `parseInt = Example`

```
var usdollar=parseInt(prompt("Plz enter an amount in dollars"));
var riyal=parseInt(prompt("Plz enter amount in saudi riyal"));
document.write("Amount of in PKR is : ",
((usdollar*277)+(riyal*74)))
```

Convert string to decimal **(parseFloat)**

1. If your string is number with decimal places then you can use parseFloat for conversion

```
var age = prompt("What is your age");//input 5.5
var num = 4;
var sum = parseFloat(age) + num;
console.log(sum); // result, 9.5
```

KeyPoints and Differences (parseInt vs parseFloat)

Common Features:

- Both parseInt and parseFloat parse strings and convert them to numbers.
- Both functions ignore leading whitespace.
- Parsing stops at the first non-numeric character.
- If the string cannot be converted to a number, both functions return NaN.

Differences:

- parseInt parses integers and can take an optional radix to specify the number base.
- parseFloat parses floating-point numbers and does not take a radix parameter

Convert string to Number [Number()]

1. The Number() function converts the object argument to a number that represents the object's value.
2. If the value cannot be converted to a legal number, NaN is returned.

```
var age = prompt("What is your age");//input 5.5
var num = 4;
var sum = age + Number(num);
console.log(sum); // result, 5.54
```

Convert string to Number (Practice)

1. In case of Number() function you don’t have to use separate parseInt or parseFloat function for conversion

```
var a = Number(true);          //returns 1
var b = Number(false);         //returns 0
var c = Number("999");         //returns 999
var d = Number("999 888");     //returns NAN
var e = Number("Hello");       //returns NAN
```

Convert Number to string toFixed()

The toFixed() method converts a number into a string, rounding to a specified number of decimals.

Note: *if the desired number of decimals are higher than the actual*

```
var a = 5.56789;
var num = a.toFixed();
document.write(num);
```

Format a number to a specified length toPrecision()

The toPrecision() method formats a number to a specified length. A decimal point and nulls are added (if needed), to create the specified length.

```
let num = 25.3852;
let a = num.toPrecision();
let b = num.toPrecision(2);
let c = num.toPrecision(3);
let d = num.toPrecision(10);
let n = a + "<br>" + b + "<br>" + c + "<br>" + d;
document.write(n);
```

Comparison Operators

Comparison operators

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

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type

Operator	Description
>	greater than
<	less than
>=	greater than and equal to
<=	less than and equal to

Comparison operators

```
var a = 3;
console.log(a == 3);          //return false
console.log(a === 6);         //return false
console.log(a != 6);          //return true
console.log(a !== 6);         //return true
console.log(a > 6);           //return false
console.log(a < 6);           //return true
console.log(a >= 6);          //return false
console.log(a <= 6);          //return true
```

Comparison operators

1. == and === are used for equality check
2. == does not consider data type of values being compared and
3. == tries to convert one of the value and compare based on that
4. === also check datatype of the values and datatype of value on both side should be same otherwise it will return false

Comparison operators

```
var a = 3;
console.log(a == 3);          //return true
console.log(a == "3");        //return true
console.log(a == 6);          //return false

console.log(a === 3);         //return true
console.log(a === "3");       //return false
console.log(a === 6);         //return false
```

Comparison operators

1. Be careful JavaScript is dynamic language and many decision taken at start are still causing confusion
2. Comparing different type of values will result in answer which is not easily understandable
3. E.g: When comparing a string with a number, JavaScript will convert the string to a number while doing the comparison. An empty string converts to 0. A non-numeric string converts to NaN which is always false.

Comparison operators (Class Practice)

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
===	equal value and equal type	x === "5"	false
		x === 5	true
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== "5"	true
		x !== 5	false
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	*true*

Logical Operators

Short-circuit Logical Operators

1. Logical operators are used to determine the logic between variables or values.
2. Logical operators required boolean operands on both side of operator
3. However, the && and || operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they will return a non-Boolean value.

Logical Operators

1. There are three logical operators in JavaScript:
    a. && (AND)
    b. || (OR)
    c. ! (NOT)

&& Logical Operator

1. 1. 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.

```
alert( true && true );   // true
alert( false && true );  // false
alert( true && false );  // false
alert( false && false ); // false
```

&& Logical Operator

```
var a = 60;
var b = a > 50 && a < 70;

alert(b); // return true



var c = 80;
var d = c > 50 && c < 70;

alert(d); // return false
```

|| Logical Operator

1. 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.

```
alert( true || true );   // true
alert( false || true );  // true
alert( true || false );  // true
alert( false || false ); // false
```

|| Logical Operator

```
var a = 60;
var b = a < 50 || a > 70;

alert(b); // return false



var c = 80;
var d = c < 50 || c > 70;

alert(d); // return true
```

! Logical NOT

1. 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:

```
alert( !true ); // false
alert( !false ); // true
```

! Logical NOT

```
var   a  =  60;
var   b  =  !(a  <  50);
alert(b);                  //   return   true


var   c  =  80;
var   d  =  !(c  >  50);
alert(d);                  //   return   false
```

! Logical NOT

1. Using the ! operator in front of a other value will convert it to a Boolean and return an opposite value.

```
alert( !1 ); // false
alert( !0); // true
```

2. Here 1 will be converted to boolean first and it will be true and because of ! operator opposite will return
3. Same for 0, it will be converted to false and ! will negate it to true

!! Double NOT

1. Using the ! operator twice in front of a value will convert it to a Boolean and negate it twice, useful when you want to convert value to boolean

```
alert( !!1 ); //
true   alert( !!0);
// true
```

2. Here 1 will be converted to boolean first and it will be true and because of first ! operator it will convert to false and then because of second ! operator it will be

Why they are called short-circuit

1. && and || operator stops evaluation of expression once they find desired value
2. && stops evaluation as soon as it finds false and returns false
3. && returns true if all values are true
4. || stops evaluation as soon as it finds true and returns true
5. || returns false if all values are false

Why they are called short-circuit

```
//returns false, evaluation stops at first value
var a = false && true && false;


//returns false, evaluation stops at last value
var b = true && true && false;

//returns false, evaluation stops at second
value  var c = true && false && true;
```

Why they are called short-circuit

```
//returns true, evaluation stops at second value
var d = false || true || false;

//returns true, evaluation                    at first value
stops  var e = true || true ||
false;

//returns true, evaluation stops at last value
var f = false || false || true
```

Conditional Statements in JavaScript

Conditions

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.

Conditions

1. In JavaScript 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
    d. Use ***switch*** to specify many alternative blocks of code to be executed

Conditions: if

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

```
if(condition) {
    //block of code to be executed if the condition
is true

}
```

Conditions: if

```
var age = 12;

if( age > 9 ) {
    console.log("Age = "+age);
}
```

Conditions: if

```
let temperature = 30;

if (temperature > 25) {
    console.log("It's hot outside!");
}
```

Conditions: if

```
let temperature = 30;

if (temperature > 25) {
    console.log("It's hot outside!");
}
```

Conditions: if

```
let age = 20;
let hasID = true;
let isMember = false;
if (age >= 18 && hasID || isMember) {
  console.log("Access granted.");
} else {
  console.log("Access denied.");
}
```

Conditions: else

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

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

Conditions: else

```
var age =   15;


if( age > = 18 ) {
    console.log("Qualifies for driving");
} else {
    console.log("Does not qualify for driving");
}
```

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 (practice)

```
var score = 80;

if( score > 80   ) {

    console.log("Grade A");
}   else if( score > 70 ) {   console.log("Grade B");

}   else if( score > 60 ) {   console.log("Grade C");

}   else {

    console.log("Failed");

}
```

Conditions: else if (practice)

```
let score = 85;
if (score >= 90) {
  console.log("Grade: A");
} else if (score >= 80) {
  console.log("Grade: B");
} else if (score >= 70) {
  console.log("Grade: C");
} else if (score >= 60) {
  console.log("Grade: D");
} else {
  console.log("Grade: F");
}
```

For Loop

For Loop

Loops

- A loop is a block of code that allows you to repeat a section of code a certain number of times, perhaps changing certain variable values each time the code is executed.

Loop

Why loops are useful?

- Loops are useful because they allow you to repeat lines of code without retyping them or using cut and paste in your text editor.
- They save time and trouble of repeatedly typing the same lines of code, but also avoids typing errors in repeated lines.
- You are also able to change one or more variable values each time the browser passes through the loop.

For Loop

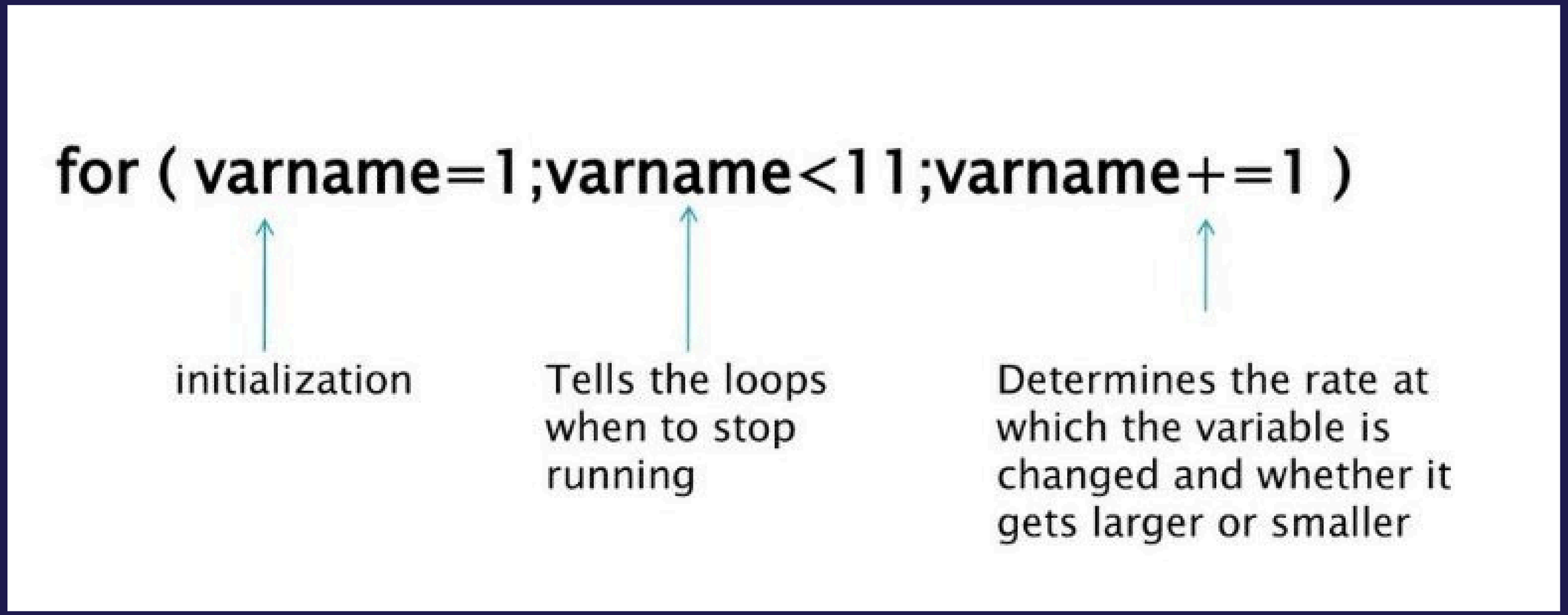
1. Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.
2. The for statement creates a loop that is executed as long as a condition is true.
3. It will only stop when the condition becomes false.

For Loop

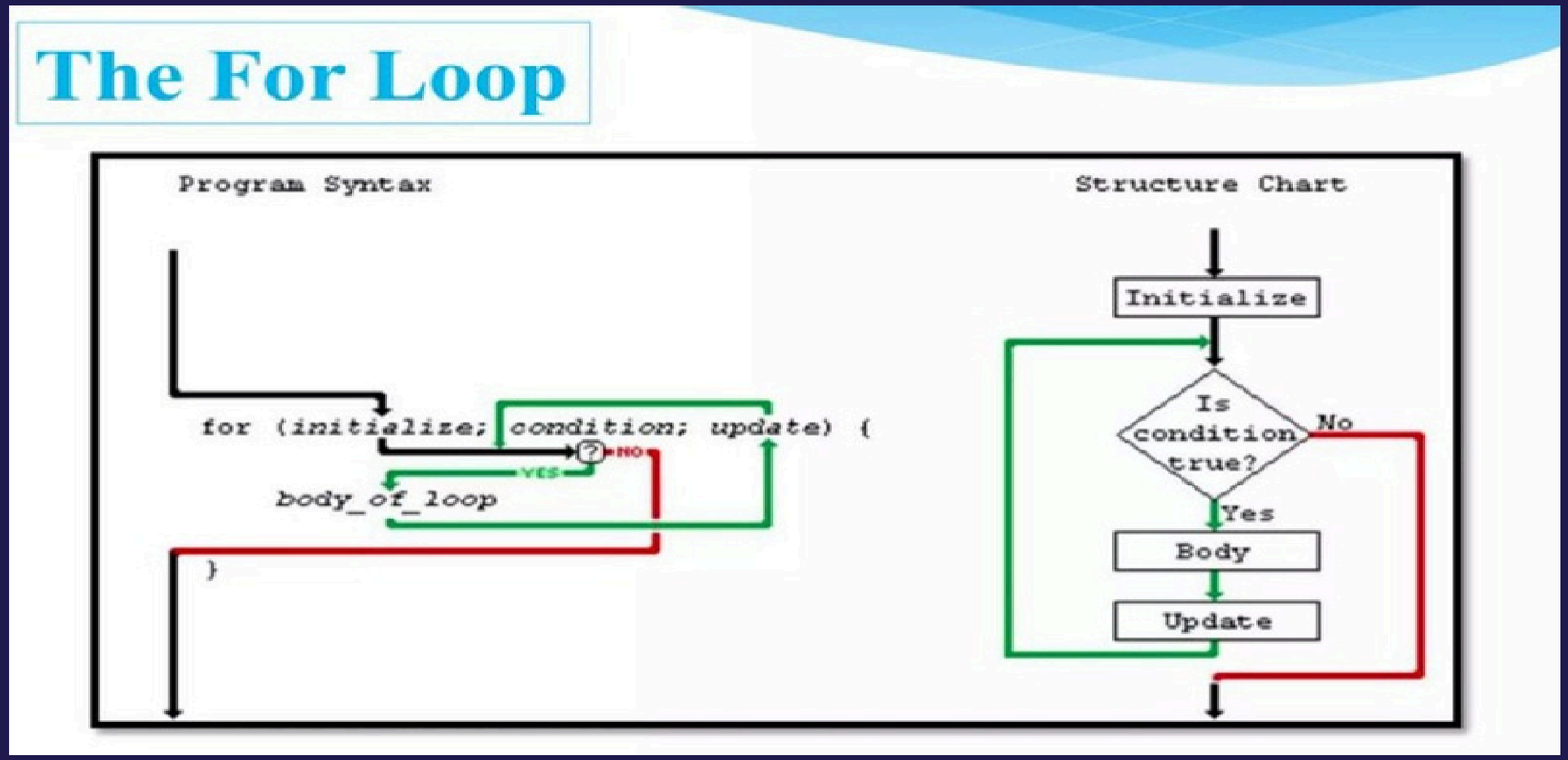
```
for (initialization; condition; expression) {
    // code to be executed
}
```

1. Initialization is done (one time) before the execution of the code block.
2. Condition for executing the code block and exit loop
3. Expression is executed (every time) after the code block has been executed.

For Loop (structure)



For Loop


The For Loop
Program Syntax
for (initialize; condition; update) {
?
NO
YES
body_of_loop
}
Structure Chart
Initialize
Is condition true?
No
Yes
Body
Update

For Loop

```
for (var i = 0; i < 10; i++){
    console.log(i);
}
```

Output: 0

1

2

For Loop

```
for (var i = 5; i <= 8;
    i++){ console.log(i);
}
```

Output:

5 6 7 8

For Loop (descending order)

```
for (var i = 6; i >=1; i--){
    console.log(i);
}
```

Output:

5 4 3 2

Infinite Loop

1. All 3 statement in loop are options, in that case it will be infinite loop
2. Also if you do not provide condition in loop it will make loop infinite

```
for ( ; ; ){
    console.log( "Hello");

}
```

For Loop

1.Printing table of 3 will require hard coded statements

```
console.log("3 x 1 = 3");
console.log("3 x 2 = 6");
console.log("3 x 3 = 9");
console.log("3 x 4 = 12");
console.log("3 x 5 = 15");
```

For Loop to create tables

1. With for loop it will be dynamic

```
var num = 3;
for (var i=1; i<=10; i++){
    console.log(num+" x "+i+" = "+(num*i));
}
```

Output:

3 x 1 = 3

....

3 x 10 = 30

While Loop

1.A "While" Loop is used to repeat a specific block of code an **unknown** number of times, until a condition is met. For example, if we want to ask a user for a number between 1 and 10, we don't know how many times the user may enter a larger number, so we keep asking "while the number is not between 1 and 10". If we (or the computer) knows exactly how many times to execute a section of code (such as shuffling a deck of cards) we use a for loop.

While Loop

While loop

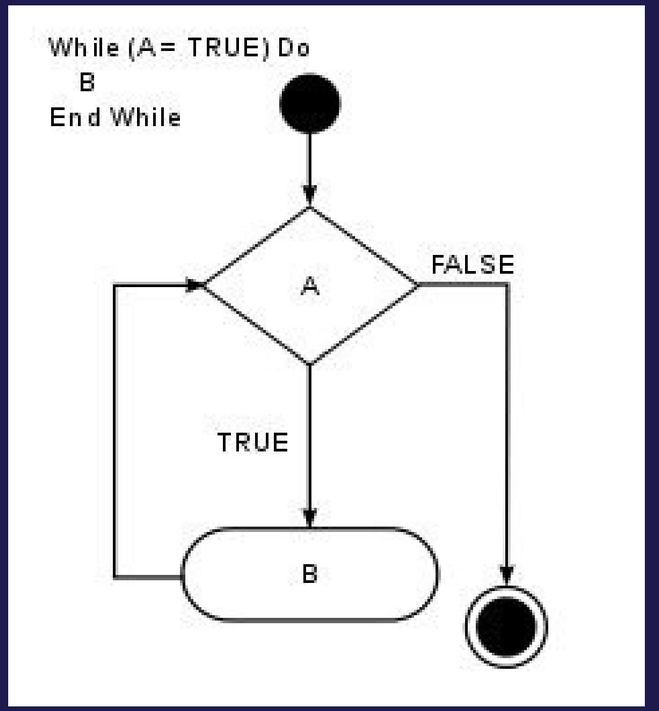
- A while loop just looks at a short comparison and repeats until the comparison is no longer true.

```
while(varname<11)
  {



  }
```

While Loop

```
let j = 0;
while (j < 10) {
console.log(j+1);
j +=1;
}
```

In the above example, the code in the loop will run, over and over again, as long as a variable (j) is less than 10



Break

```
for (var i = 0; i < 8;
    i++){     if(i == 4) {
            break;
    }
    console.log("I = "+i);
}
```

Output

: I = 0

I = 1

I = 2

I = 3

Continue

```
for (var i = 0; i < 8; i++){
    if(i == 4) {
        continue;
    }
    console.log("I = "+i);
}
```

Output

: I = 0

I = 1

I = 2

I = 3

I = 5

I = 6

I = 7

Nested Loops

1. If a loop exists inside the body of another loop, it's called nested loop.

```
for (var i = 0; i < 3; i++){
    for(var j = 0; j < 2; j++) {
        console.log("I = "+i+" J = "+j);
    }
}
```

Output:
I = 0 J = 0
I = 0 J = 1
I = 1 J = 0
I = 1 J = 1
I = 2 J = 0
I = 2 J = 1

Nested Loops

```
let a = ["a", "e", "i", "o", "u", "A", "E", "I", "O", "U"];
let input = prompt("please enter any character");
let check = false;
for (let i = 0; i < a.length; i++) {

if (input === a[i]) {

check = true;
console.log("It is a vowel"); break;
}
} if (check === false) {
console.log("it is a consonant"); }
```

THANK-YOU