WSA | Forward looking IT finishing school

Types and Statements

JavaScript

Table of Content

- Reserve Keywords
- Data Types
- Statements

Reserve Keywords

(JavaScript)

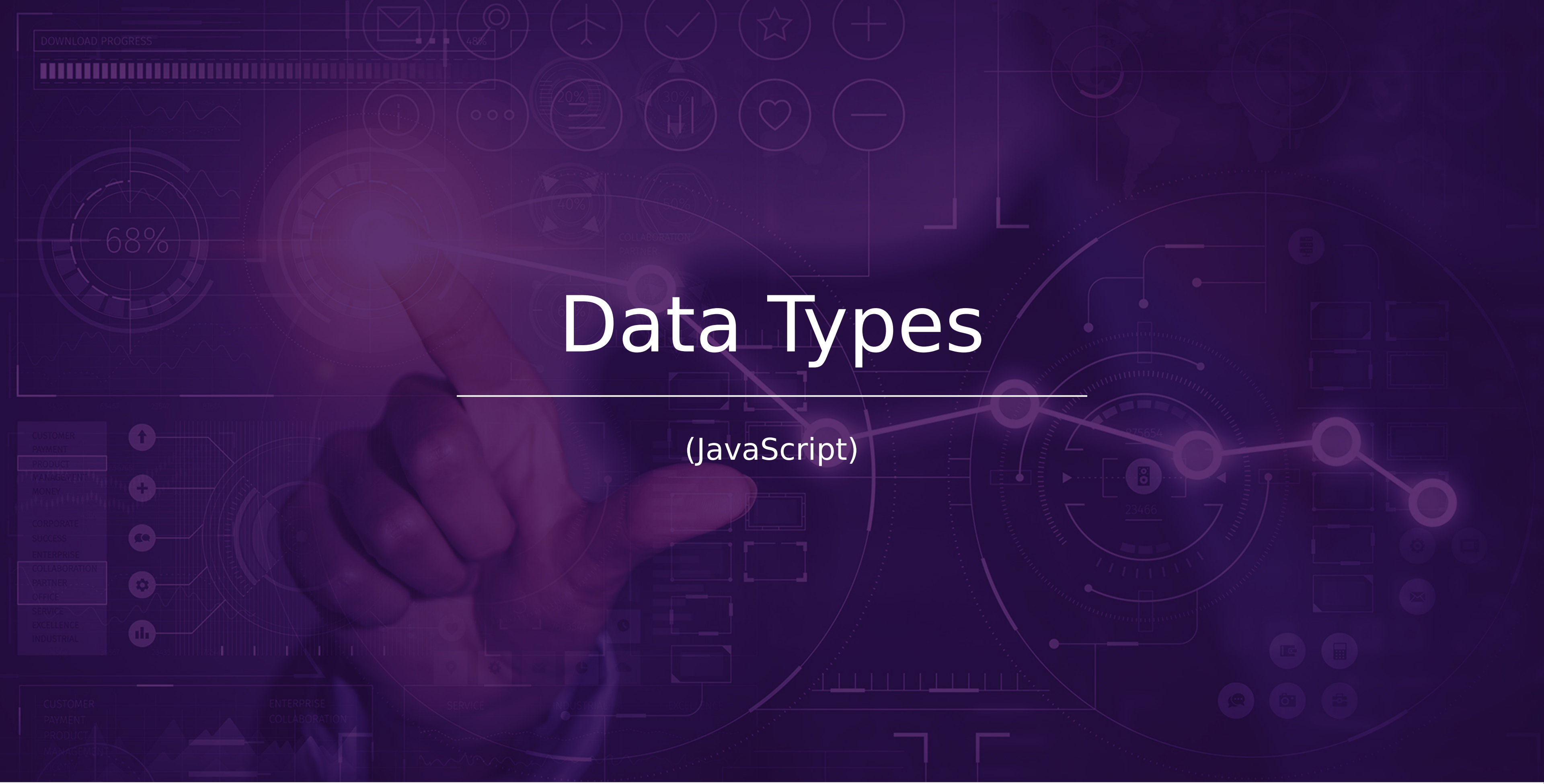
JS - Reserved Words

- Any language has a set of words called vocabulary
- JavaScript also has a set of keywords with special purpose
- These special purpose keywords words are used to construct statements as per language syntax and cannot be used as JavaScript variables, functions, methods or object names
- Therefore, also called reserve keywords
- The limited set of keywords help us to write unlimited statements

Reserved Words

abstract	debugger	final	instanceof	protected	throws
boolean	default	finally	int	public	transient
break	delete	float	interface	return	true
byte	do	for	let	short	try
case	double	function	long	static	typeof
catch	else	goto	native	super	var
char	enum	if	new	switch	void
class	export	implements	null	synchronized	volatile
const	extend	import	package	this	while
continue	false	in	private	throw	with

***keywords in red color are removed from ECMA script 5/6**



Data Types

(JavaScript)

JS - Variables

- Variables are container to store values
- Name must start with
  - A letter (a to z or A to Z)
  - Underscore(_)
  - Or dollar($) sign
- After first letter we can use digits (0 to 9)
  - Example: x1, y2, ball25, a2b

JS - Variables

- JavaScript variables are case sensitive, for example ‘sum’ and ‘Sum’ and ‘SUM’ are different variables

Example :

```
var x = 6;
var y = 7;
var z = x+y;
```

JS - Data Types

- There are two types of Data Types in JavaScript
  - Primitive data type
  - Non-primitive (reference) data type

Note : JavaScript is weakly typed. Every JavaScript variable has a data type , that type can change dynamically

Primitive data type

- String
- Number
- Boolean
- Null
- Undefined

Primitive data type
(String)

- String - A series of characters enclosed in quotation marks either single quotation marks (') or double quotation marks (")

```
Example :
var name = “Webstack Academy”;
var name = 'Webstack Academy';
```

Primitive data type (Number)

- All numbers are represented in IEEE 754-1985 double precision floating point format (64 bit)
- All integers can be represented in -2^{53} to $+2^{53}$ range
- Largest floating point magnitude can be ± 1.7976×10^{308}
- Smallest floating point magnitude can be ± 2.2250×10^{-308}
- If number exceeds the floating point range, its value will be infinite

Primitive data type (Number)

- Floating point number is a formulaic representation which approximates a real number
- The most popular code for representing real numbers is called the IEEE Floating-Point Standard

	Sign	Exponent	Mantissa
Float (32 bits) Single Precision	1 bit	8 bits	23 bits
Double (64 bits) Double Precision	1 bit	11 bits	52 bits

Float : $V = (-1)^{s} * 2^{(E-127)} * 1.F$

Double : $V = (-1)^{s} * 2^{(E-1023)} * 1.F$

Primitive data type
(Number formats)

- The integers can be represented in decimal, hexadecimal, octal or binary

```
Example :

var a = 0x10;   // Hexadecimal
var b = 010;    // Octal number
var c = 0b10;   // Binary
var num1 = 5; // Decimal
var num2 = -4.56;
```

Primitive data type
(Number conversion)

- Converting from string

```
Example :
 var num1 = 0, num2 = 0;

 // converting string to number
 num1 = Number("35");
 num2 = Number.parseInt(“237”);
```

Primitive data type
(Number conversion)

- Converting to string

Example :

```
var str = “”; // Empty string
var num1 = 125;

// converting number to string
str = num1.toString();
```

Primitive data type
(Number – special values)

Special Value	Cause	Comparison
Infinity, -Infinity	Number too large or too small to represent	All infinity values compare equal to each other
NaN (not-a-number)	Undefined operation	NaN never compare equal to anything (even itself)

Primitive data type

(Number – special value checking)

Method	Description
isNaN(number)	To test if number is NaN
isFinite(number)	To test if number is finite

Primitive data type
(Boolean)

- Boolean data type is a logical true or false

```
Example :
var ans = true;
```

Primitive data type
(Null)

- In JavaScript the data type of null is an object
- The null means empty value or nothing

```
Example :
var num = null; // value is null but still type is an object.
```

Primitive data type
(Undefined)

- A variable without a value is undefined
- The type is object

Example :

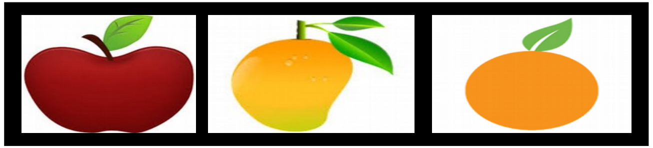
```
var num;    // undefined
```

Non-primitive data types

- Array
- Object

Arrays

- Array represents group of similar values
- Array items are separated by commas
- Array can be declared as :
  - var fruits = [“Apple” , “Orange”, “Mango”];



Objects

- An Object is logically a collection of properties
- Objects represents instance through which we can access members
- Object properties are written as name:value pairs separated by comma

```
Example :

var student={ Name:“Mac”, City:“Banglore”, State:“Karnataka”};
```

Statements

(JavaScript)

JS – Simple Statements

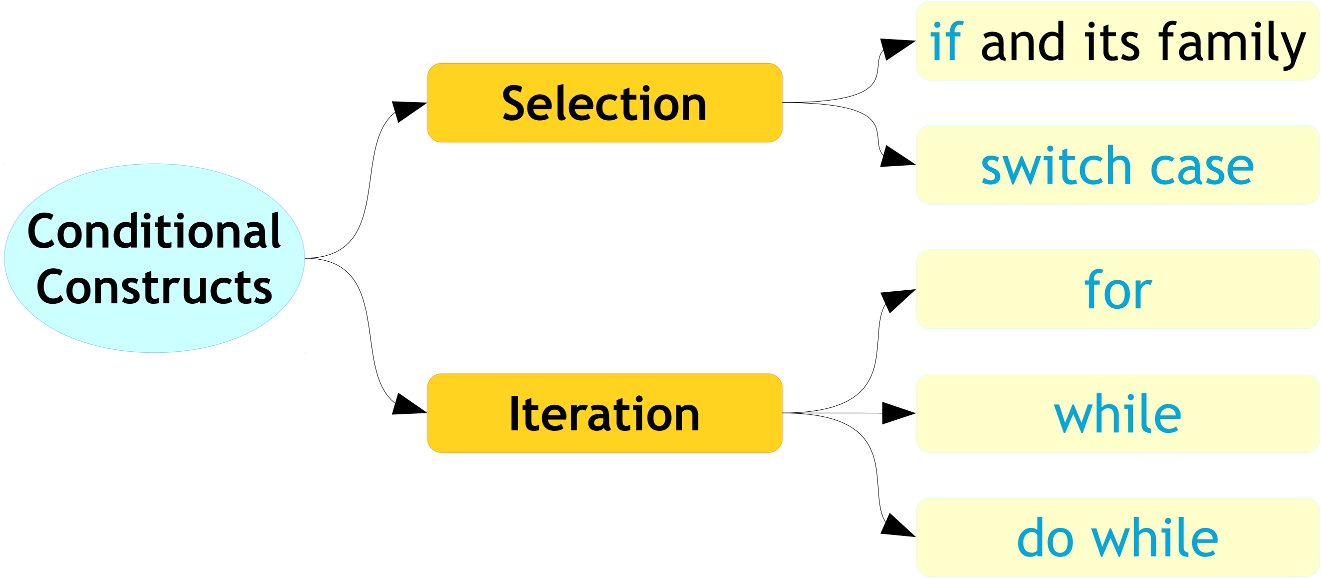
- In JavaScript statements are instructions to be executed by web browser

```
Example :

    <script>
        var y = 4, z = 7; // statement
        var x = y + z;    // statement

        document.write("x = " + x);
    </script>
```

JS – Compound Statements

```
Example :
    <script>
        ...
    if (num1 > num2) {
        if (num1 > num3) {
            document.write("Hello");
        }
        else {
            document.write("World");
        }
    }
    ...
    </script>
```

JS – Conditional Construct


Conditional Constructs

Selection

Iteration

if and its family

switch case

for

while

do while

JS – Statements
(conditional)

```
Syntax :

if (condition) {

    statement(s);

}
```

```
Example :

<script>

var num = 2;
if (num < 5) {
    document.write("num < 5");
}

</script>
```

JavaScript - Input

Method	Description
prompt()	It will asks the visitor to input Some information and stores the information in a variable
confirm()	Displays dialog box with two buttons ok and cancel

Example - confirm()

```
Example :
    <script>
    var x = confirm('Do you want to continue?');
    if (x == true) {
        alert("You have clicked on Ok Button.");
    }
    else {
        alert("You have clicked on Cancel Button.");
    }
</script>
```

Example - prompt()

```
Example :

<script>

var person = prompt("Please enter your name", "");
if (person != null) {
    document.write("Hello " + person +
                   "! How are you today?");
}

</script>
```

Exercise

- Write a JavaScript program to input Name, Address, Phone Number and city using prompt() and display them
- Write a JavaScript program to find area and perimeter of rectangle
- Write a JavaScript program to find simple interest
  - Total Amount = P (1 + rt)

JS – Statements

(conditional)

```
Syntax :
if (condition) {
    statement(s);
}
else {
    statement(s);
}
```

JS – Statements
(conditional)

```
Example :
<script>
var num = 2;
if (num < 5) {
    document.write("num is smaller than 5");
}
else {
    document.write("num is greater than 5");
}
</script>
```

JS – Statements

(conditional)

```
Syntax :
if (condition1) {
    statement(s);
}
else if (condition2) {
    statement(s);
}
else {
    statement(s);
}
```

JS – Statements

(conditional)

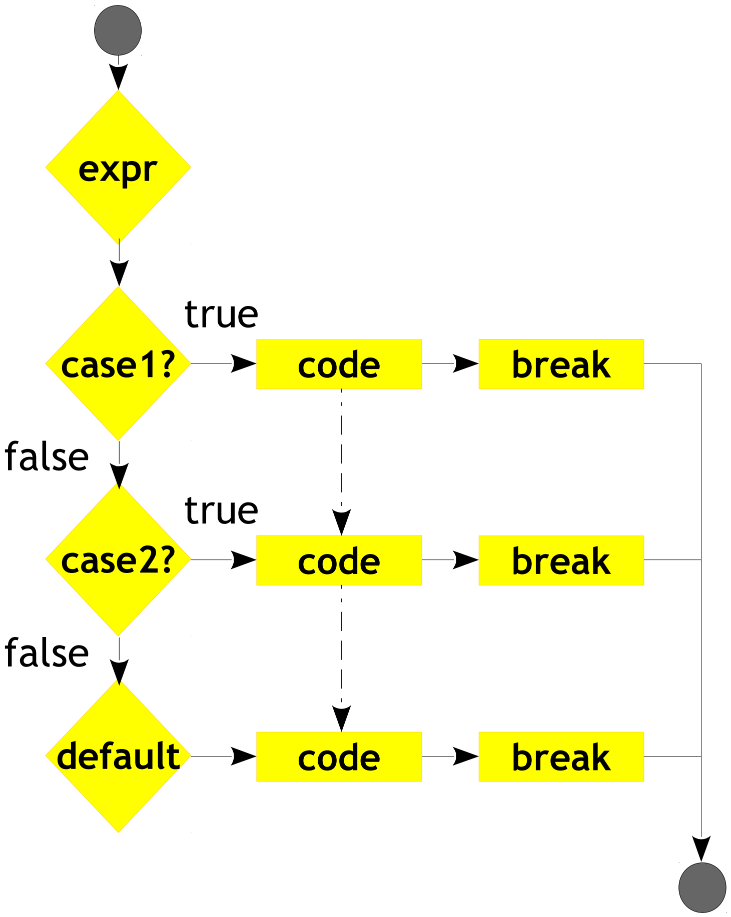
```
Example :
<script>
var num = 2;
if (num < 5) {
    document.write("num is smaller than 5");
}
else if (num > 5) {
    document.write("num is greater than 5");
}
else {
    document.write("num is equal to 5");
}
</script>
```

Class Work

- WAP to find the max of two numbers
- WAP to print the grade for a given percentage
- WAP to find the greatest of given 3 numbers
- WAP to find the middle number (by value) of given 3 numbers

JS – Statements
(switch)

```
Syntax :

switch (expression) {
    case exp1:
        statement(s);
        break;
    case exp2:
        statement(s);
        break;
    default:
        statement(s);
}
```



JS – Statements

(switch)

```
<script>
    var num = Number(prompt("Enter the number!", ""));

    switch(num) {
        case 10 : document.write("You have entered 10");
            break;
        case 20 : document.write("You have entered 20");
            break;
        default : document.write("Try again");
    }

</script>
```

Class work

- Write a simple calculator program
  - Ask user to enter two numbers
  - Ask user to enter operation (+, -, * or /)
  - Perform operation and print result

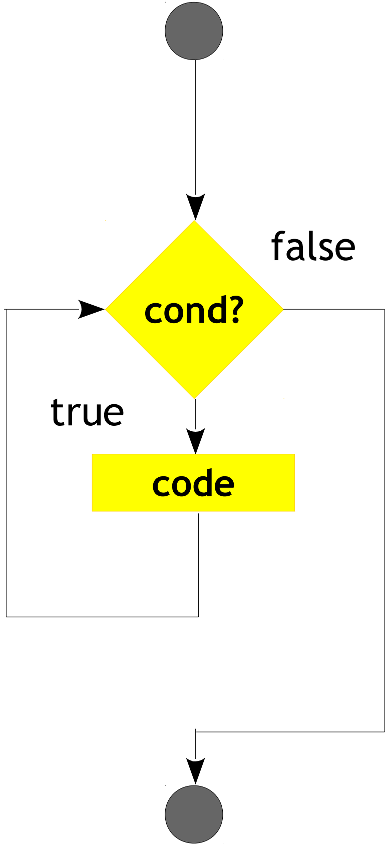
JS – Statements

(while)

Syntax:

```
while (condition)
{
     statement(s);
}
```

- **Controls** the loop.
- Evaluated **before** each execution of loop body



JS – Statements

(while)

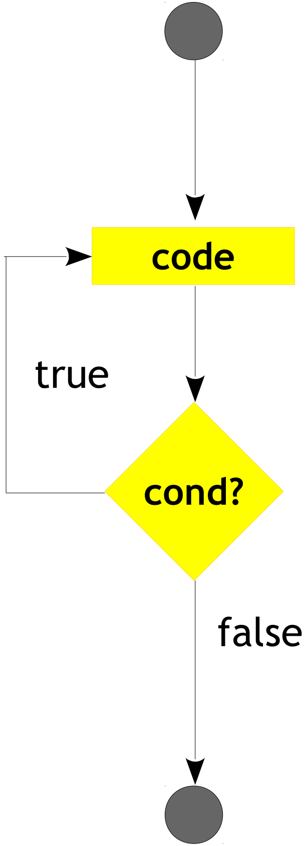
```
Example:

<script>
    var iter = 0;

    while(iter < 5)
    {
        document.write("Looped " + iter + " times <br>");
        iter = iter + 1;
    }
</script>
```

JS – Statements

(do - while)

```
Syntax:

do {
    statement(s);
} while (condition);
```

- **Controls** the loop.
- Evaluated **after** each execution of loop body



JS – Statements

(do-while)

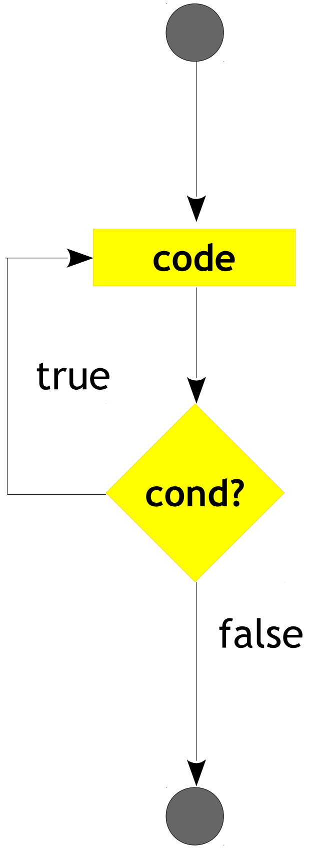
```
Example:

<script>
    var iter = 0;

    do {
        document.write("Looped " + iter + " times <br>");
        iter = iter + 1;
    } while ( iter < 5 );

</script>
```

JS – Statements

(for loop)

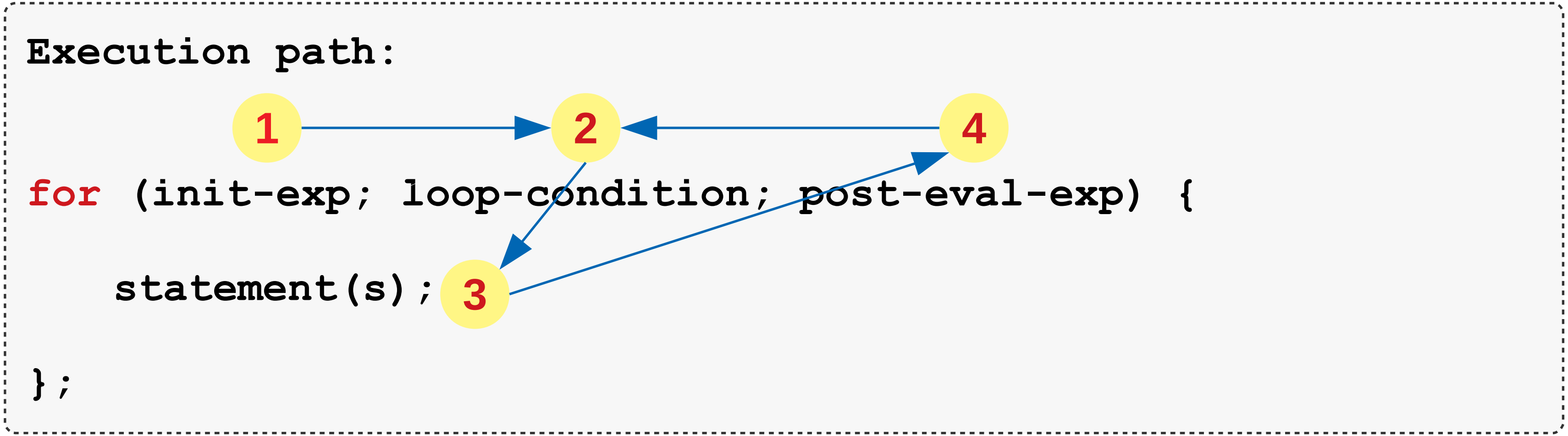
```
Syntax:

for (init-exp; loop-condition; post-eval-exp) {

    statement(s);

};
```

- **Controls** the loop.
- Evaluated **before** each execution of loop body



JS – Statements

(for loop)



```
Execution path:

for (init-exp; loop-condition; post-eval-exp) {

    statement(s);

};
```

JS – Statements

(for loop)

Example:

```
<script>
    for (var iter = 0; iter < 5; iter = iter + 1) {
        document.write("Looped " + iter + " times <br>");
    }
</script>
```

JS – Statements

(for-in loop)

- “for-in” loop is used to iterate over enumerable properties of an object

```
Syntax:
    for (variable in object) {
        statement(s);
    }
```

JS – Statements

(for-in loop)

```
Example:
<script>
    var person = { firstName:”Rajani”, lastName:”Kanth”,
     profession:”Actor” };
     for (var x in person) {
         document.write(person[x] + “ “);
     }
</script>
```

JS – Statements

(for-in loop)

- loop only iterates over enumerable properties
- “for-in” loop iterates over the properties of an object in an arbitrary order
- “for-in” should not be used to iterate over an Array where the index order is important

JS – Statements

(for-of loop)

- “for-of” loop is used to iterate over iterate-able object
- “for-of” loop is not part of ECMA script

```
Syntax:
    for (variable of object) {
        statement(s);
    }
```

JS – Statements

(for-of loop)

```
Example:
<script>
    var array = [1, 2, 3, 4, 5];
    for (var x of array) {
        document.write(x + “ “);
    }
</script>
```

Class Work

- W.A.P to print the power of two series using for loop
  - $2^1, 2^2, 2^3, 2^4, 2^5$ ...
- W.A.P to print the power of N series using Loops
  - N^1, N^2, N^3, N^4, N^5 ...
- W.A.P to multiply 2 numbers without multiplication operator
- W.A.P to check whether a number is palindrome or not

Class Work - Pattern

- Read total (n) number of pattern chars in a line (number should be “odd”)
- Read number (m) of pattern char to be printed in the middle of line (“odd” number)
- Print the line with two different pattern chars
- Example – Let's say two types of pattern chars '$' and '*' to be printed in a line. Total number of chars to be printed in a line are 9. Three '*' to be printed in middle of line.
- Output ==> $$$* * *$$$

Class Work - Pattern

- Based on previous example print following pyramid

```
            *
        *   *   *
    *   *   *   *   *
*   *   *   *   *   *   *
```

Class Work - Pattern

- Based on previous example print following rhombus

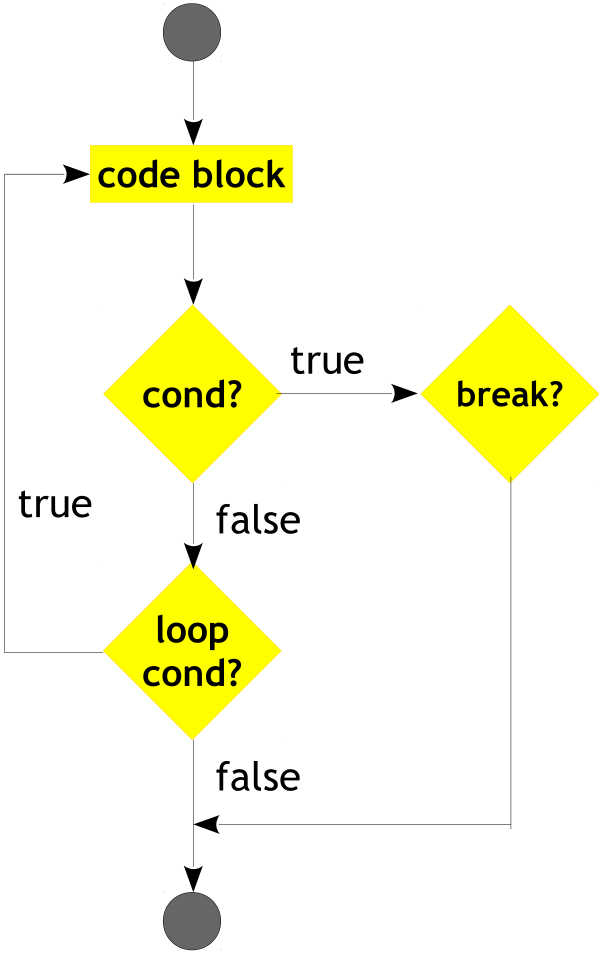
```
      *
    * * *
  * * * * *
* * * * * * *
  * * * * *
    * * *
      *
```

JS – Statements
(break)

- A break statement shall appear only in “switch body” or “loop body”
- “break” is used to exit the loop, the statements appearing after break in the loop will be skipped
- “break” without label exits/‘jumps out of’ containing loop
- “break” with label reference jumps out of any block

JS – Statements
(break)

```
Syntax:
while (condition) {
     conditional statement
        break;
}
```



JS – Statements

(break)

```
<script>
for (var iter = 0; iter < 10; iter = iter + 1) {
    if (iter == 5) {
        break;
    }
    document.write("<br>iter = " + iter);
}
</script>
```

JS – Statements
(break with label)

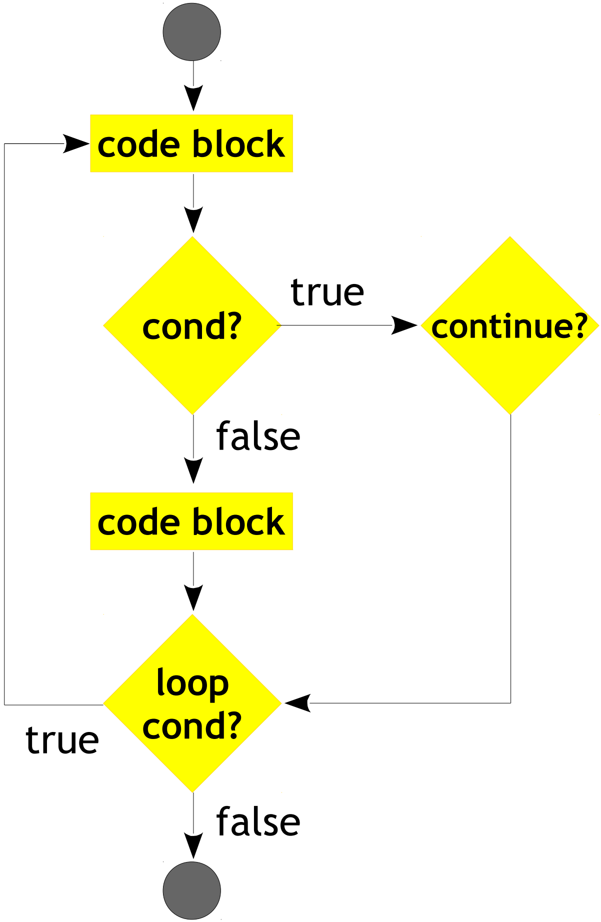
```
Syntax:
outer_loop:
for (condition) {
    inner_loop:
    for(condition) {
      conditional statement
          break outer_loop;  // jump out of outer_loop
    }
}
```

JS – Statements
(continue)

- A continue statement causes a jump to the loop-continuation portion, that is, to the end of the loop body
- The execution of code appearing after the continue will be skipped
- Can be used in any type of multi iteration loop

JS – Statements
(continue)

```
Syntax:
while (condition) {
     conditional statement
        continue;
}
```



JS – Statements

(continue)

```
<script>
for (var iter = 0; iter < 10; iter = iter + 1) {
    if (iter == 5) {
        continue;
    }
    document.write("<br>iter = " + iter);
}
</script>
```

JS – Statements

(continue with label)

```
Syntax:
outer_loop:
for (condition) {
    inner_loop:
    for(condition) {
      conditional statement
          continue outer_loop; // continue from outer_loop
    }
}
```

JS – Statements

(goto)

- This keyword has been removed from ECMA script 5/6
- “goto” keyword is is not recommended to use
- Use break with label if necessary

Thank you

Web Stack Academy (P) Ltd

#83, Farah Towers,
1st floor,MG Road,
Bangalore – 560001

M: +91-80-4128 9576
T: +91-98862 69112
E: info@www.webstackacademy.com