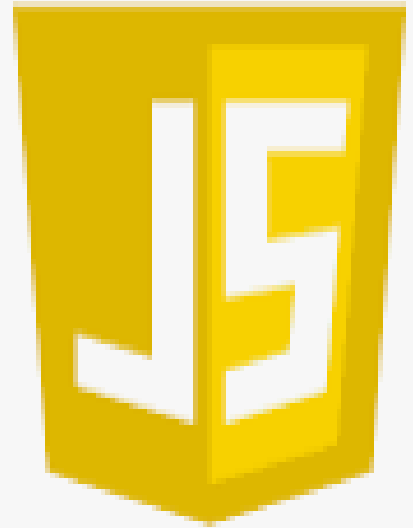


Chapter Three

Web application Requirement engineering

JS



Group presentation

- Where Do Requirements Come From?
- Revision on Requirements Engineering Activities
- Software Requirements with Relevance to the Web
- Principles for RE of Web Applications
- Adapting RE Methods to Web Application Development.

Where Do Requirements Come From?

- Where Do Requirements Come From?
- Revision on Requirements Engineering Activities
- Software Requirements with Relevance to the Web
- Principles for RE of Web Applications
- Adapting RE Methods to Web Application Development.

Revision on Requirements Engineering Activities

- Software Requirements with Relevance to the Web
- Principles for RE of Web Applications
- Adapting RE Methods to Web Application Development.

Software Requirements with Relevance to the Web

- Software Requirements with Relevance to the Web
- Principles for RE of Web Applications
- Adapting RE Methods to Web Application Development.

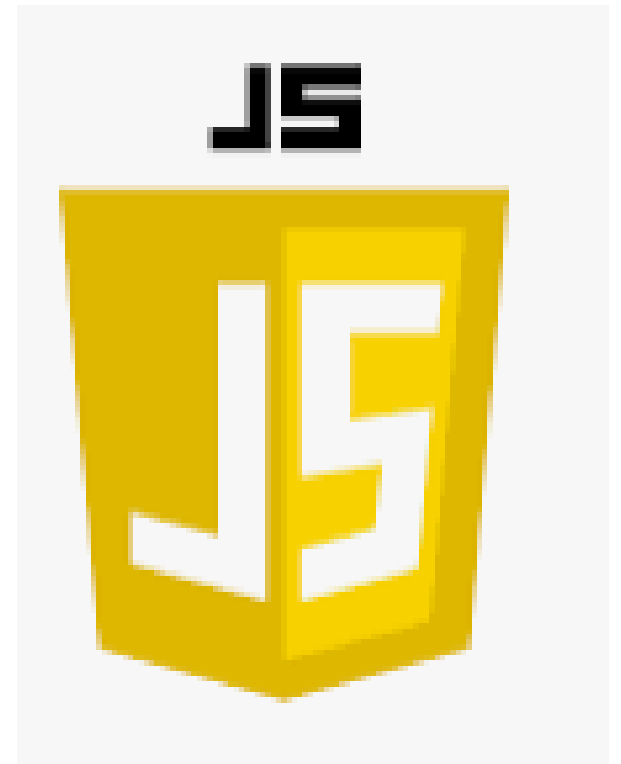
Principles for RE of Web Applications

- Software Requirements with Relevance to the Web
- Adapting RE Methods to Web Application Development.

Adapting RE Methods to Web Application Development.

- Software Requirements with Relevance to the Web

JAVASCRIPT



Introduction

- **JavaScript** is a lightweight, interpreted **programming** language.
- Very easy to implement because it is integrated with HTML.
- It is open and cross-platform.
- Great for developing front-end as well as back-end softwares using different JavaScript based frameworks like jQuery, React, Express, Node.JS etc.
- It comes installed on every modern web browser
 - Chrome, Firefox, Safari, ...
 - you really do not need any special environment setup.
- JavaScript engines were originally used only in web browsers
- now extended to mobile app development, desktop app development, and game development.

Application of JavaScript

- Some of the applications of JavaScript
 - **Client side validation**
 - Verifying any user inputs before submitting to server
 - **Manipulating HTML Pages**
 - Adding and deleting any html tags, Change its look and feel..
 - **User Notifications**
 - **Back-end Data Loading**
 - Retrieve data from server using Ajax
 - **Presentations**
 - creating presentations with website look and feel with libraries like RevealJS, BespokeJS
 - **Server Applications**
 - Build fast and scalable network applications including webservers with Node JS

JavaScript Syntax

```
1  <html>
2    <body>
3      <script language = "javascript" type = "text/javascript">
4        |
5        | | document.write("Hello World!")
6        |
7      </script>
8    </body>
9  </html>
```

- You can place the **<script>** tags, anywhere in your page.
 - In **<Head>** is Recommended.
- **Language** attribute specifies what scripting language you are using. **Type** also tells the same thing and recommended in recent version of HTML
- function **document.write** which writes a string or HTML (both) into our HTML document.
- there is no **main()** function/method

JavaScript Syntax

- **spaces, tabs, and newlines** that appear in JavaScript programs are ignored.
- **Semicolons** are **Optional** (if each statement is on **separate** line).

```
5 | var1 = 10 | var1 = 10; var2 = 20;  
6 | var2 = 20
```

- JavaScript is a case-sensitive language.

```
5      let lastname, lastName;
6      lastName = "Doe";
7      lastname = "Peterson";
```

- Comments

```
1 <script language = "javascript" type = "text/javascript">
2     <!--
3         // This is a comment. It is similar to comments in C++
4
5         /*
6         * This is a multi-line comment in JavaScript
7         * It is very similar to comments in C Programming
8         */
9     //-->
10 </script>
```

Datatype

- Datatype - defines type of value and operation.
- JavaScript allows you to work with three primitive data types
 - **Numbers** e.g. 123, 120.50, 123e5, 123e-5 etc.
 - JavaScript does not make a distinction between integer values and floating-point values.
 - All numbers are represented as **64-bit** floating-point values.
 - **Strings** e.g. "hello world" etc.
 - You can use single or double quotes.
 - **Boolean** e.g. true or false.
- JavaScript also defines two trivial data types,
 - **null** and
 - **Undefined** – a variable without a value

```
3 | | let car;    // Value is undefined, type is undefined
```

Datatype

- JavaScript supports a **composite** data type known as **object**.

- written with curly braces { }

```
3  person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

- JavaScript arrays
 - support numbered indexing
 - are written with square brackets.
 - Array items are separated by commas.

```
let students = ['Anteneh', 'daniel', 'teshome'];  
let ages = [12, 13, 18];  
document.write(students[1]); // daneil
```

Identifiers and Reserved Words

- In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign (\$).
- Subsequent characters may be letters, digits, underscores, or dollar signs.
- Reserved words (like JavaScript keywords) cannot be used as names.

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

Variables

- Variables are containers for storing data (values).
- There are 3 ways to declare a JavaScript variable: **var**, **let**, **const**.

```
3  var firstName;  
4  let middleName;  
5  const lastName;
```

- After the declaration, the variable has no value (**undefined**)
- To **assign** a value to the variable, use the equal sign.

```
5  | firstName = "Anteneh";
```

- You can also assign a value to the variable when you declare it.
- **JS is untyped** language. variable can hold a value of any data type

```
5  var firstName = 'Anteneh';  
6  var lastName = 'Daniel';  
7  var year = 3;  
8  var grade = 3.5;
```

Variables

- **var**
 - can be re-declared and updated
 - You can use the variable before it is declared (called Hoisting)
 - can NOT have block scope - Variables declared inside a { } block can be accessed from outside the block
- **let** (introduced in [ES6 \(2015\)](#))
 - can be updated but not re-declared
 - must be Declared before use.
 - have Block Scope - Variables declared inside a { } block cannot be accessed from outside the block
- **const** (also introduced in [ES6 \(2015\)](#))
 - Can not be updated and re-declared
 - have Block Scope - Variables declared inside a { } block cannot be accessed from outside the block

Operators

- Arithmetic Operators: +, -, *, /, %, **, ++, --
- Assignment Operators: =, +=, -=, /=, %=, **=
- Comparison Operators: ==, ===, !=, !==, >, >=, <, <=, ?
- Logical Operators: &&, ||, !
- Type Operators: typeof, instanceof
- Bitwise Operators: &, |, ~, ^, <<, >>, >>>

```
3  let x = 5;
4  let y = 2;
5  let mult = x * y;

7  y *= x
8  y++
9  let power_y = y ** 2

11 document.write(typeof(y));
```

Operators

- **+** operator can also be used to add (concatenate) strings.

```
3  let text1 = "Anteneh";
4  let text2 = "Daniel";
5  let text3 = text1 + " " + text2;
```

- Assignment operator **+=** can also be used to concatenate strings

```
7  text1 += " is a Lecturer";
```

- Adding a **number** and **string**, the result will be **string**.

```
3  let result1 = "10" + 5; // 105
4  let result2 = "Hi" + 5; // Hi5
```

- The order of the operation also matters

```
3  let x = 10; let y = 20;
4  let z = "The result is: " + x + y; // The result is: 1020
5  let r = x + y + " is The Result"; // 30 is The Result
```

Operators

- JavaScript will try to convert strings to numbers in all numeric operations.

```
3   var h = "100" * "2"; // 200
4   var y = "100" / "10"; // 10
5   var t = "100" - "20"; // 80
```

- Trying to do arithmetic with a non-numeric string will result in **NaN**

```
7   let x = 100 / "Apple"; // x will be NaN (Not a Number)
```

- Operator Precedence.

Level	Operators	Notes
1	() [] .	call, member (including typeof and void)
2	! ~ - ++ --	negation, increment
3	* / %	multiply/divide
4	+ -	addition/subtraction
5	<< >> >>>	bitwise shift
6	< <= > >=	relational
7	== !=	equality
8	&	bitwise AND
9	^	bitwise XOR
10		bitwise OR
11	&&	logical AND
12		logical OR
13	?:	conditional
14	= += -= *= /= %= <<= >>= >>>= &= ^= =	assignment
15	,	comma

Conditional Statements

- Perform different actions for different decisions using conditional statements
 - If statement
 - Switch statement

If Statements

- JavaScript supports the following forms of **if..else** statement –

- if** statement

```
var age = 20;  
if (age > 18) {  
    document.write("<b>Qualifies for driving</b>");  
}
```

- if...else** statement

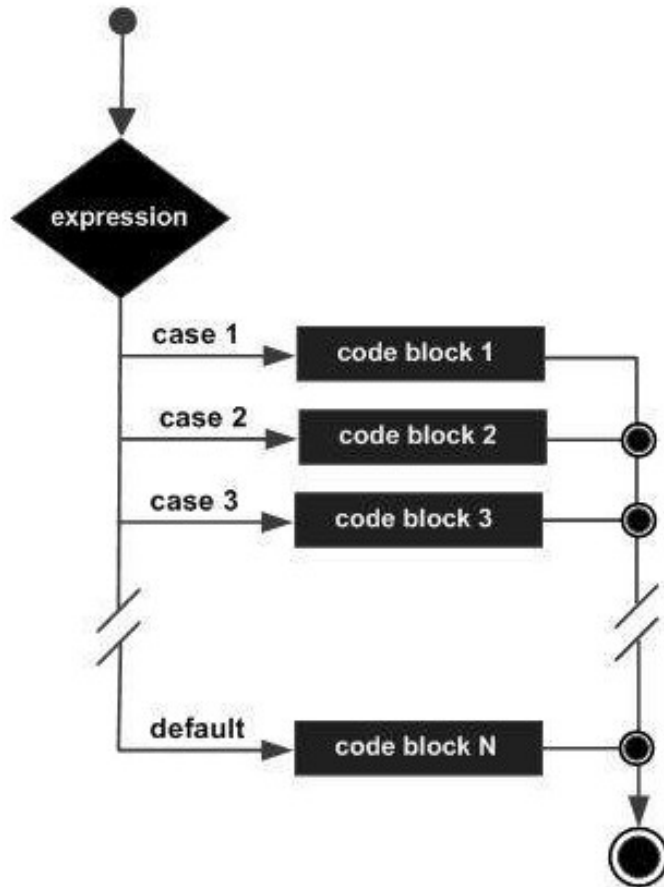
```
var age = 15;  
if (age > 18) {  
    document.write("<b>Qualifies for driving</b>");  
} else {  
    document.write("<b>Does not qualify for driving</b>");  
}
```

- if...else if... statement.**

```
var book = "maths";  
if (book == "history") {  
    document.write("<b>History Book</b>");  
} else if (book == "maths") {  
    document.write("<b>Maths Book</b>");  
} else {  
    document.write("<b>Unknown Book</b>");  
}
```

switch Statements

- to perform a multiway branch
- especially when all of the branches depend on the value of a single variable



```
var book = "maths";
switch (book) {
  case "history":
    document.write("<b>History Book</b>");
    break;
  case "maths":
    document.write("<b>Maths Book</b>");
    break;
  default:
    document.write("<b>Unknown Book</b>");
}
```

Loops

- JavaScript supports different kinds of loops:
 - for** - loops through a block of code a number of times

```
for (initialization; test condition; iteration statement) {  
    Statement(s) to be executed if test condition is true  
}
```

```
var count;  
document.write("<ul>");  
  
for (count = 0; count < 10; count++) {  
    | document.write("<li> Current Count : " + count + " </li>");  
}  
document.write("</ul>");
```

- for/in** - loops through the properties of an object

- one property at a time
- this loop ends when all properties of the object are exhausted.

```
for (variablename in object) {  
    statement or block to execute  
}
```

Loop ...

```
let person = {firstName: 'anteneh', age: 12, position: 'Instructor'};

for (var_name in person) {
  document.write(var_name); // display variable name
  document.write(person[var_name]); // display the value of the variable
}
```

```
let students = ['Anteneh', 'daniel', 'teshome'];
for (var_name in students) {
  document.write(students[var_name]); // display value in an array
}
```

- **for/of** - loops through the values of an iterable object (iterable data structures such as **Arrays, Strings, Maps, NodeLists**)

```
let students = ['Anteneh', 'daniel', 'teshome'];
for (var_name of students) {
  document.write(var_name); // display variable name
}
```

Loop ...

- **while** - loops through a block of code while a specified condition is true

```
while (expression) {  
    Statement(s) to be executed if expression is true  
}  
  
var count = 0;  
document.write("<ul>");  
  
while (count < 10) {  
    document.write("<li> Current Count : " + count + " </li>");  
    count++;  
}  
document.write("</ul>");
```

- **do/while** - also loops through a block of code while a specified condition is true

```
do {  
    Statement(s) to be executed;  
} while (expression);
```

```
var count = 0;  
document.write("<ul>");
```

```
do {  
    document.write("<li> Current Count : "  
        + count + " </li>");  
    count++;  
} while (count < 10);  
  
document.write("</ul>");
```

Function

- a block of code designed to perform a particular task
- executed when "something" invokes it or calls it.
- Declaration always begins with keyword **function**,
- no return type

```
function add(num1, num2) {  
    return num1 + num2;  
}  
var sum = add(12, 34)
```

Example Code

```
var thinkingOf;
var guess;

//initialize the computer's number
thinkingOf = Math.ceil(Math.random() * 1000);

// ask the user to guess
guess = prompt("I'm thinking of a number between 1 and 1000. what is it?", "");

while (guess != thinkingOf) {
    // Evaluate the user's guess
    if (guess < thinkingOf) {
        guess = prompt("your guess was too low. guess Again.", "");
    } else {
        guess = prompt("your guess was too high. guess Again.", "");
    }
}

// Game over; congratulate the user
window.alert(guess + " is correct!");
```

JavaScript's where to ?

- JavaScript code is inserted between `<script>` and `</script>` tags.
- JavaScript in
 - `<head>`
 - `<body>`
 - External
- Placing scripts in external files has some advantages:
 - It separates HTML and code
 - It makes HTML and JavaScript easier to read and maintain
 - Cached JavaScript files can speed up page loads