Delivering Value. Scientifically.

JavaScript

Presented by: Alka Jhanwar

Agenda

- Introduction
- How JavaScript works
- JavaScript-Core Concepts
- JavaScript-Advanced Concepts

Introduction

JavaScript was developed by **Brendan Eich in 1995**.

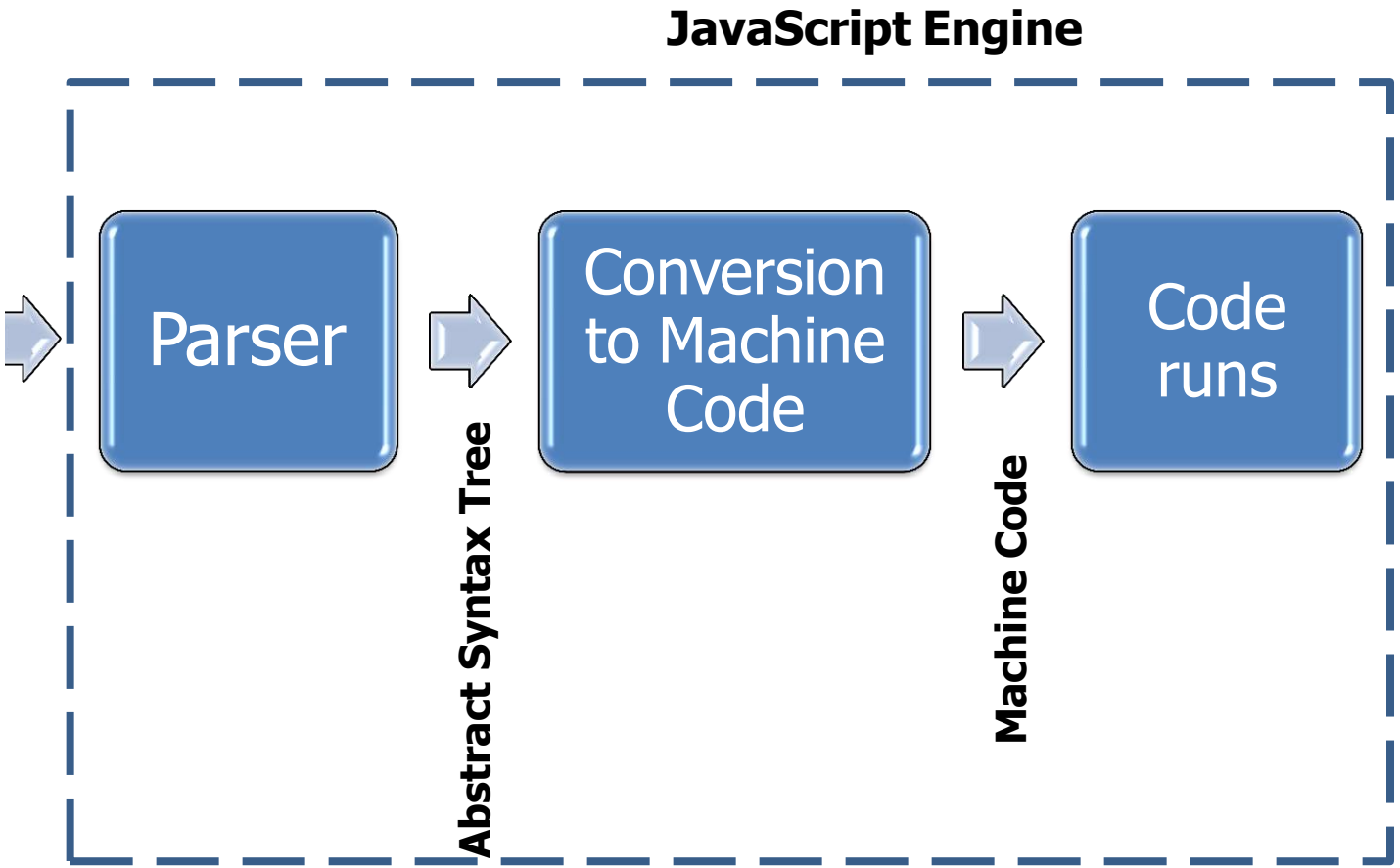
JavaScript is most commonly used as a client side scripting language.

- Makes webpages alive
- JavaScript engines
- How engine works?
  - Reads the script
  - Converts script to machine language
  - Machine code executes

How JavaScript Works

JavaScript code

```
var john = {
    firstName: 'John',
    lastName: 'Smith',
    birthYear: 1992,
    family: ['Jane', 'Mark', 'Bob', 'Emily'],
    job: 'teacher',
    isMarried: false,
    //function definition
    calcAge: function() {
        this.age = 2018 - this.birthYear;
    }
};
//function call
john.calcAge();
```



JavaScript-Core Concepts

- Variable
- Datatypes
- Operators
- Operator Precedance
- Program Flow
- Functions
- Dialog Boxes
- Anonymous,Arrow and Callback Functions
- Arrays
- Strings
- Objects And Properties
- Objects And Methods
- Object Creation
- Classes.

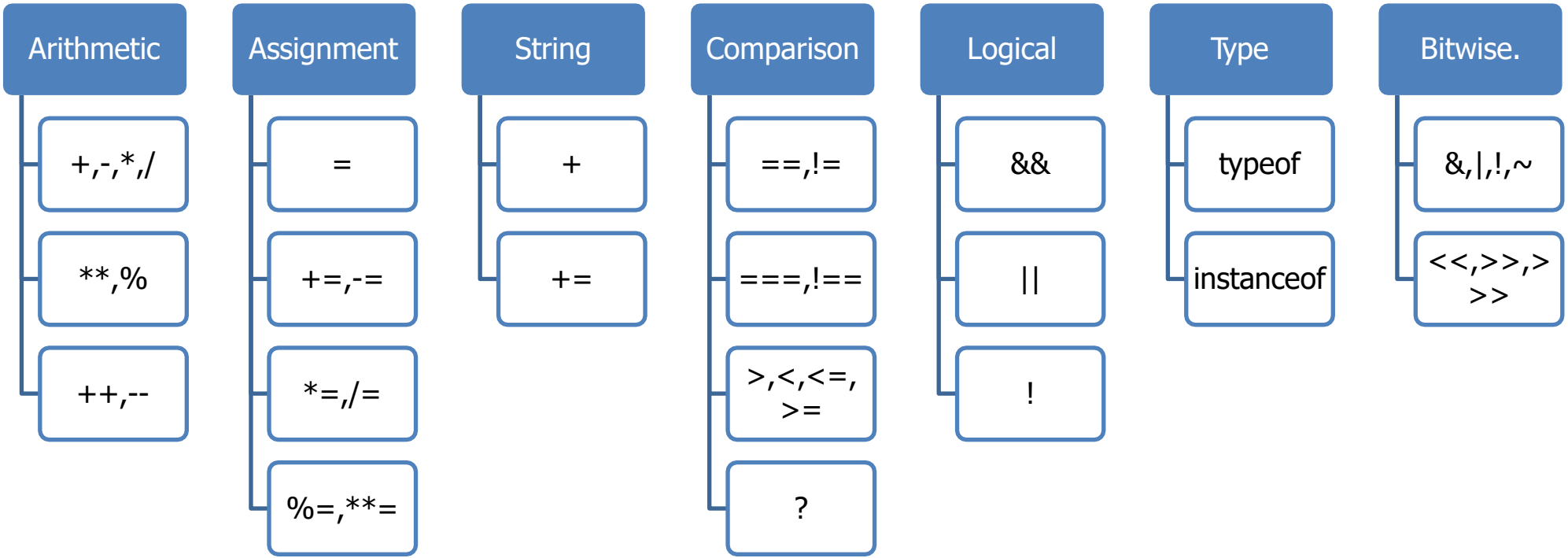
Variables

- Variables are containers for storing data (values)
- **Var**
  - Function scoped/global scoped
  - Can be redeclared and updated within scope
- **Let**
  - Block scoped
  - Cannot be redeclared but can be updated
- **Const**
  - Block scoped
  - Cannot be redeclared or updated.

Datatypes

Number	• Any kind of numbers :integer/floating point
String	• One/more characters
Boolean	• True/false
Null	• Unknown values
Undefined	• Unassigned values
Object	• Complex data structures

Operators



Operator Precedence

Operator type	Individual operators
member	`. []`
call / create instance	`() new`
negation/increment	`! ~ - + ++ -- typeof void delete`
multiply/divide	`* / %`
addition/subtraction	`+ -`
bitwise shift	`<< >> >>>`
relational	`< <= > >= in instanceof`
equality	`== != === !==`
bitwise-and	`&`
bitwise-xor	`^`
bitwise-or	`|`
logical-and	`&&`
logical-or	`||`
conditional	`?:`
assignment	`= += -= *= /= %= <<= >>= >>>= &= ^= |= &&= ||= ??=`
comma	`,`

Program Flow

- if…else
- Ternary operator (?:)
- switch
- while
- do…while
- for loop
- break
- continue.

Dialog Boxes

- 3 types of Dialog boxes
  - Alert
    - Give warning message to users
    - Has only 'OK' button
  - Confirmation
    - Used to take user's consent
    - Has 'OK' and 'Cancel' buttons
  - Prompt
    - Pop-up for user input
    - Has 'OK' and 'Cancel' buttons
    - Label for what to display in textbox
    - Default string to display in textbox.

Functions

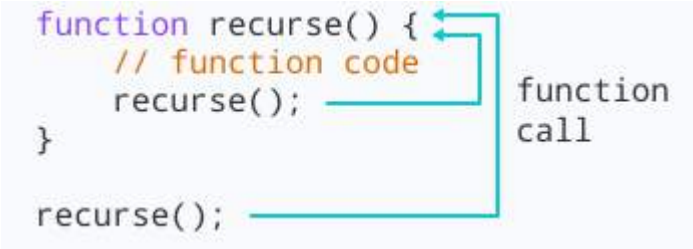
- Containers for code: similar action, multiple places
- Declaration:
  - function name(parameters delimited by comma){

```
/*code*/
}
```

- Local variables
- Outer variables
- Parameters
- Returning value.

Anonymous, Arrow, Callback,Recursive Functions

- Anonymous: function without name
  - var var_name=function(parameters){ /* code */};
- Arrow: accepts arguments, evaluates expression
  - var var_name=(arg1,arg2,…)=> expression
- Callback: function passed to another function as argument
  - var var_name=function1(function2).
- Recursive: A function that calls itself is called a recursive function.



Arrays

- Store multiple values in single variable
- Syntax:
  - var array_name=[element1, element2,…];

    OR
  - var array_name=new Array(element1,element2,…);
- Access elements using index
- Avoid Array() as it adds complexity.

Arrays (contd...)

- Properties

length	• Gives no. of elements
prototype	• Allows to add methods and properties to an object.

Arrays (contd…)

- Methods

To add/remove elements	To search	To transform array	Miscellaneous
•push(items) •pop() •shift() •unshift(item) •splice(pos,deleteCount, items) •slice(start,end) •concat(items)	•indexOf/lastIndexOf (item,pos) •includes(value) •find/filter(func) •findIndex()	•map(func) •sort(func) •reverse() •split/join() •reduce(func,initial)	•forEach(func) •Array.isArray(arr) •arr.some(func)/arr.every (func) •arr.fill(value,start,end) •arr.copyWithin(target,start,end).

Arrays (contd…)

- Looping

Approach	Description
for(var i=0;i<array.length;i++)	Works fastest and old browser compatible
for(var item of array)	Modern syntax to get items
for(var item in array)	Never use as it iterates through all properties and is very slow.

Strings

- Methods

Method	Description
charAt()	Returns character at specified index
charCodeAt()	Returns Unicode value of character at specified index
concat()	Combines two strings
indexOf()	Returns index of first occurrence of a value
lastIndexOf()	Returns index of last occurrence of a value
match()	Matches regular expression against string
replace()	Replace string with other string
search()	Searches regular expression match in string.

Strings (contd…)

- Methods

Method	Description
slice()	Extracts section of string
split()	Splits a string into array of strings
substr()	Returns specified no. of characters starting from specified position
substring()	Returns characters between two indexes
toLowerCase()	Converts string to lower case
toUpperCase()	Converts string to upper case
valueOf()	Returns primitive value of specified object

Objects And Properties

- Key-value pair
- Keys are called ‘properties’
- Access values by using dot(.) with object name.

```
var john = {
    firstName: 'John',
    lastName: 'Smith',
    birthYear: 1990,
    family: ['Jane', 'Mark', 'Bob', 'Emily'],
    job: 'teacher',
    isMarried: false
};
```

Objects And Methods

- Can define functions in the object
- Called using dot(.) with object name.

```
var john = {
    firstName: 'John',
    lastName: 'Smith',
    birthYear: 1992,
    family: ['Jane', 'Mark', 'Bob', 'Emily'],
    job: 'teacher',
    isMarried: false,
    //function definition
    calcAge: function() {
        this.age = 2018 - this.birthYear;
    }
};
//function call
john.calcAge();
```

Object Creation

- Factory function
  - Uses 'return' to return created object

```
//Factory function
function createPerson(name) {
    return {
        name,
        eat: function(){
            console.log("Person eats");
        }
    };
}

const newObj = createPerson("AAA");
```

- Constructor function
  - Uses 'this' and 'new' keywords.

```
//Constructor function
function Person(name) {
    this.name=name,
    this.eat= function(){
    console.log("Person eats");
    }
}

const newObj = new Person("AAA");
```

Classes

- Is kind of a function
- **Syntax:**

```
class ClassName {
   //class methods
   constructor() {... }
   method1() {... }
   method2{... }
...
}
```

- ***new*** is used to create object with listed methods
- ***constructor()*** is called automatically by new.

JavaScript-Advanced Concepts

- Hoisting
- Scoping And Scope-Chain
- Prototype
- HTML DOM
- JavaScript DOM
- Events
- Class Inheritance

Hoisting

- *Variable declarations lifted to the top of function/scope*
- Happens figuratively i.e. without moving code
- Functions declarations are hoisted above variable declarations
- Only variable declaration is hoisted
- Assigned value is not hoisted.

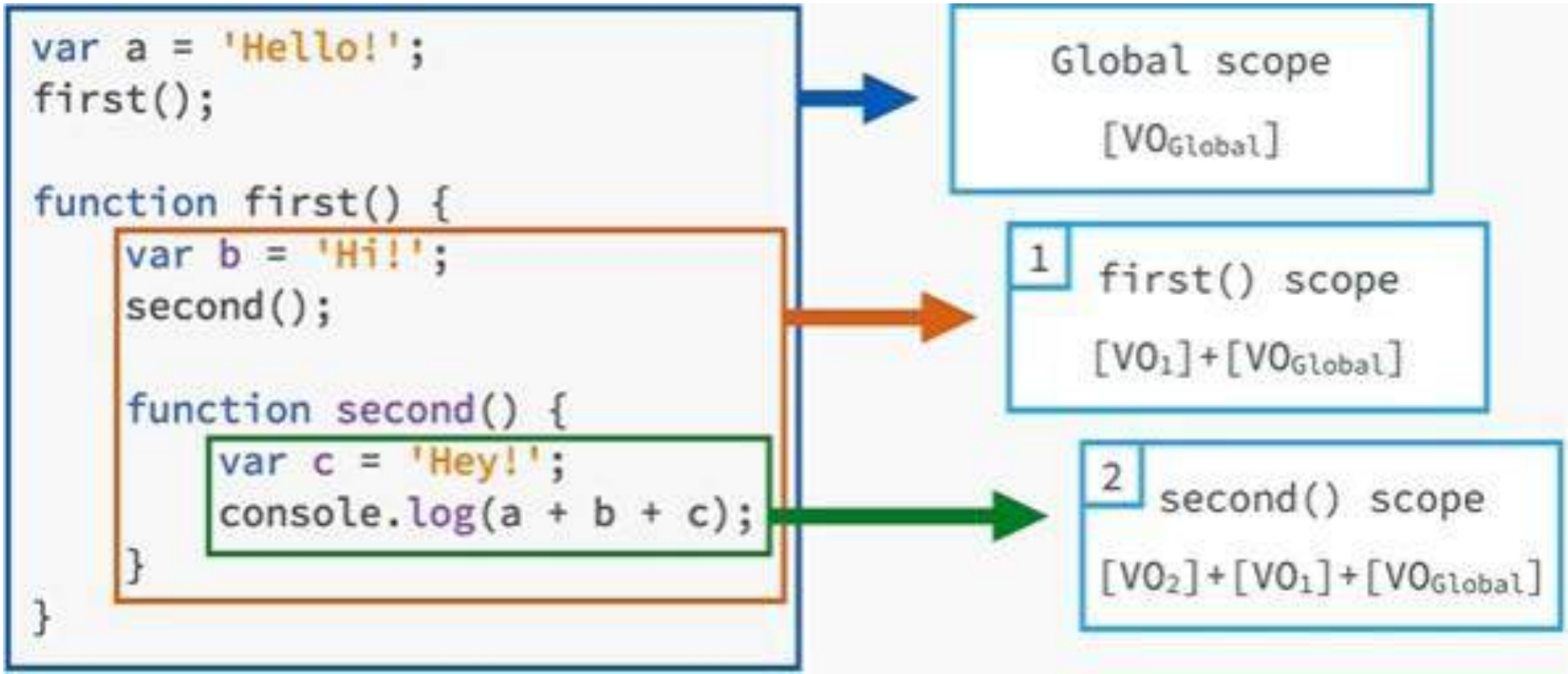
How you write:

```
console.log(myName);
var myName = 'Vrushali';
```

How interpreter/compiler sees:

```
var myName;
console.log(myName);
```

Scoping And Scope-Chain



Scoping And Scope-Chain(contd…)

- Global scope
  - *Variable defined outside function is global*
  - Only one Global scope in entire document
- Local scope
  - *Variable declared inside function*
  - Also called function scope
  - Only accessible within function
- Variable shadowing
  - *Declare a local variable and a global variable with the same name*
  - Local variable takes precedence when inside a function.

Prototype

- 'Prototype' property is an object
- JavaScript engine attaches hidden object '__proto__'
- Lets object access inbuilt methods
- Object.prototype=object.__proto___
- Can be used to modify prototypes.

Prototype

Prototype is an Object that exist on every function and object in JavaScript.

Function's Prototype
A function's prototype is an Object instance that will become the prototype for all objects created using this function as a constructor.

Object Prototype
Its an Object instance from which the object is inherited.

JavaScript Objects and Prototypes

JavaScript Prototypes and Inheritance: A Graphical Overview of Prototypes

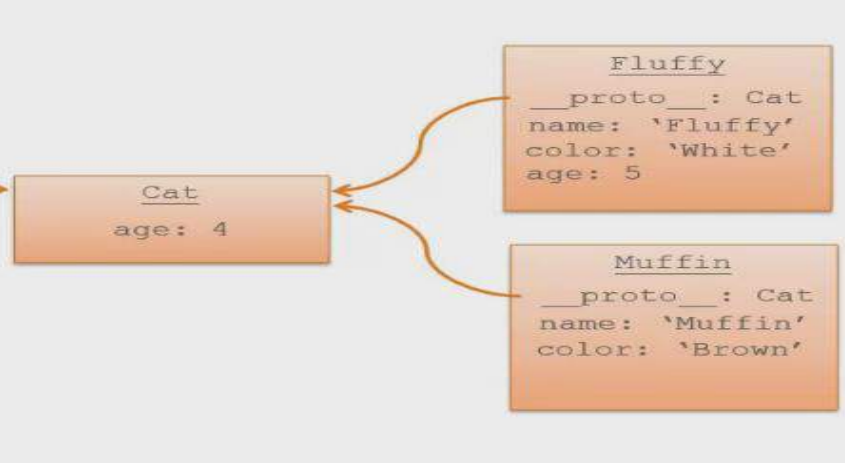
Prototypes Behind the Scenes

```
function Cat(name, color) {
    this.name = name
    this.color = color
}

Cat.prototype.age = 3

var fluffy = new Cat('Fluffy', 'White')
var muffin = new Cat('Muffin', 'Brown')
fluffy.age = 5
Console.log(fluffy.age)  // 5
Cat.prototype.age = 4

Console.log(fluffy.age)  // 5
Console.log(muffin.age)  // 4
```



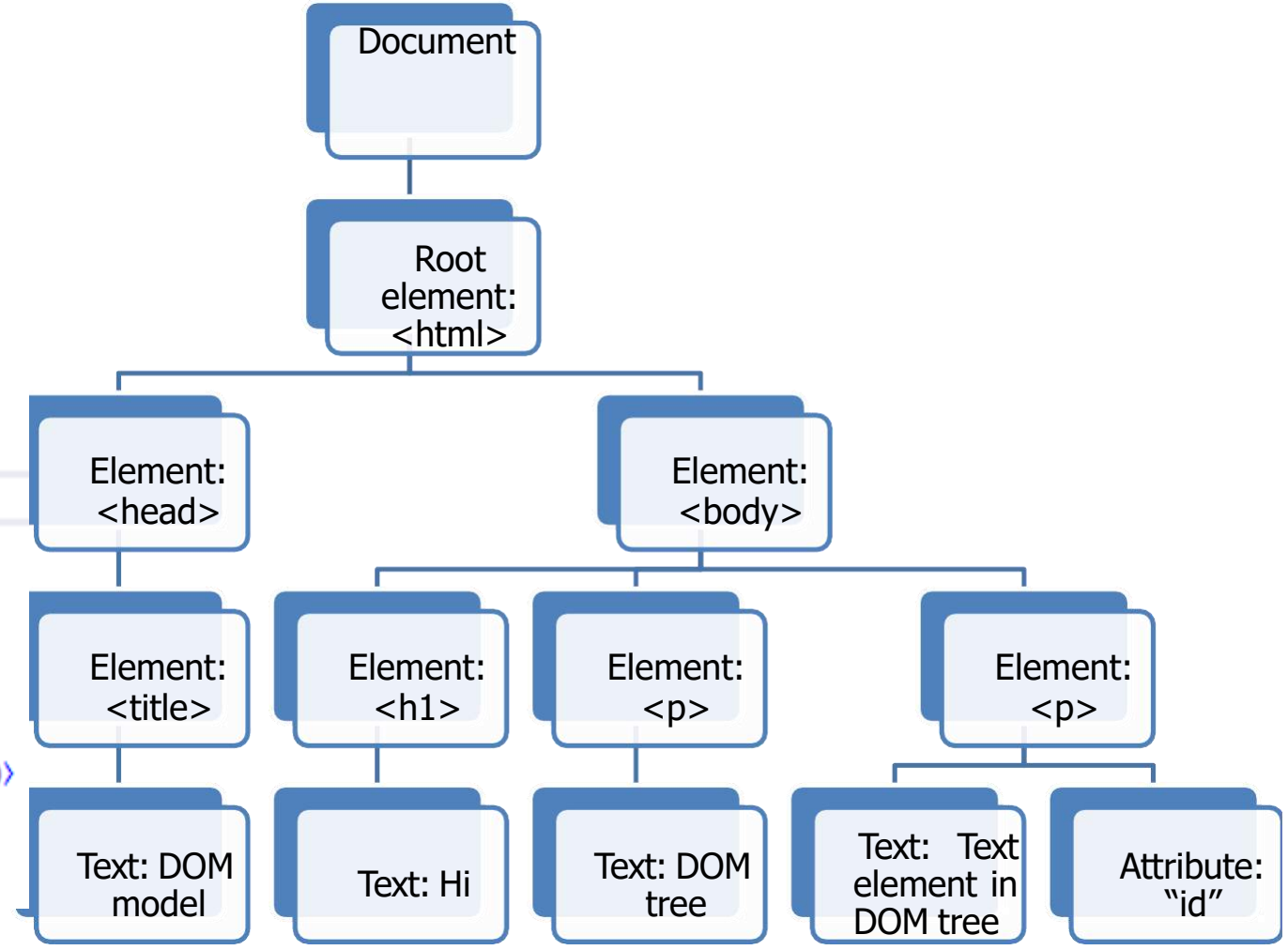
HTML DOM

- DOM (Document Object Model)
- Set of APIs to control HTML
- Uses Document object
- Document is webpage render in browser
- Entry point to webpage’s content
- Programming interface for HTML & XML.

JavaScript DOM

- Creation of DOM tree
- Manipulation of DOM tree.

```
<html>
<head>
    <title>DOM Model</title>
</head>
<body>
    <h1>Hi</h1>
    <p>DOM Tree</p>
    <p id="text">Text element in the DOM tree.</p>
</body>
</html>
```



JavaScript DOM (contd…)

- Types of nodes

Document Node
- Added at the top of tree
- Represents entire page
- Starting point of DOM tree

Element Node
- HTML elements create element nodes
- Use these to access attributes and text nodes

Attribute Node
- Attributes of HTML tags become attribute nodes
- Not children of element nodes but are part of them

Text Node
- Cannot have child nodes
- Contain text content within that element
- Creates a new branch in DOM tree.

JavaScript DOM(contd…)

Selecting individual element node	Selecting multiple elements
• getElementById('id') – Uses unique value of element's "id" attribute • querySelector('css selector') – Uses CSS selector – Returns first matching element	• getElementByClassName('class') – Selects elements having specified value of class attribute • querySelectorAll('css selector') – Uses CSS selector – Returns all matching elements.

JavaScript DOM (contd…)

- Traversing between element nodes

parentNode

- Selects parent of current element node
- Returns single element

previousSibling/nextSibling

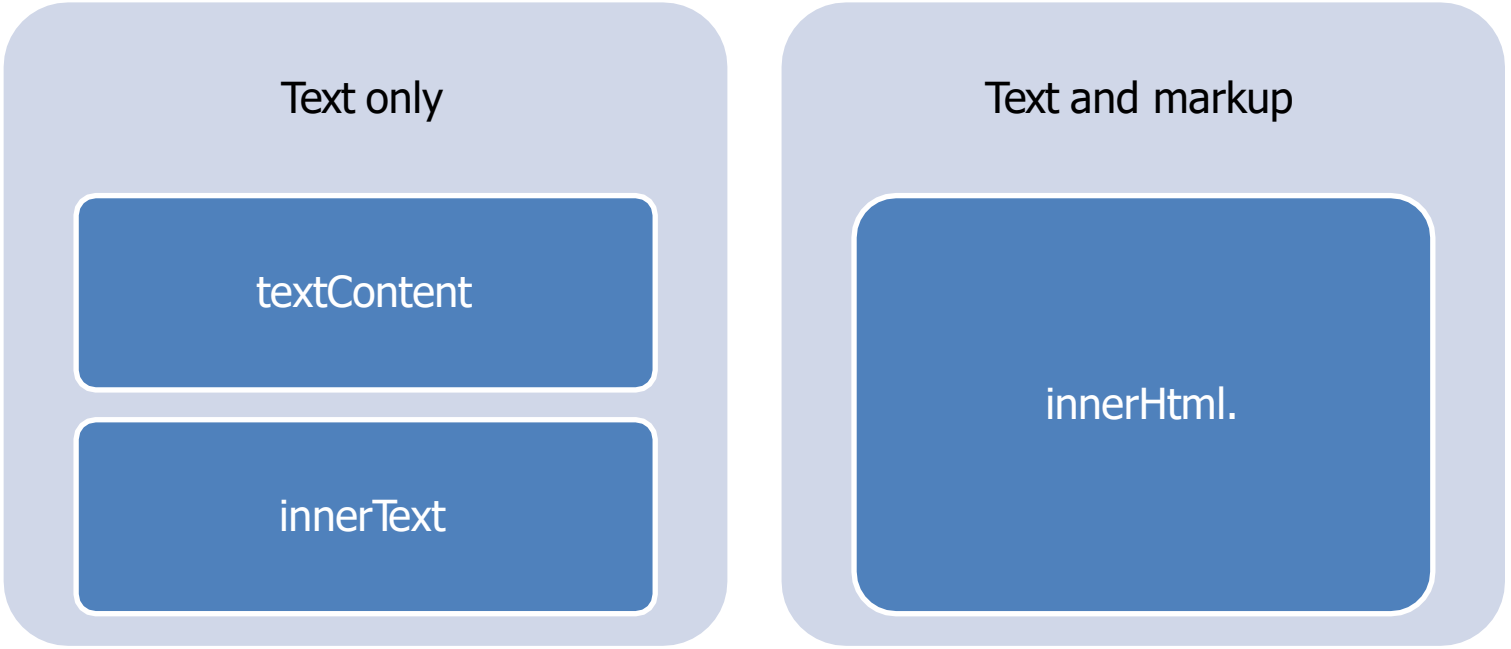
- Selects previous/next sibling from tree

fisrtChild/lastChild

- Selects first/last child of current element.

JavaScript DOM(contd...)

- Get/update element content



JavaScript DOM(contd…)

- DOM manipulation

Adding elements to DOM

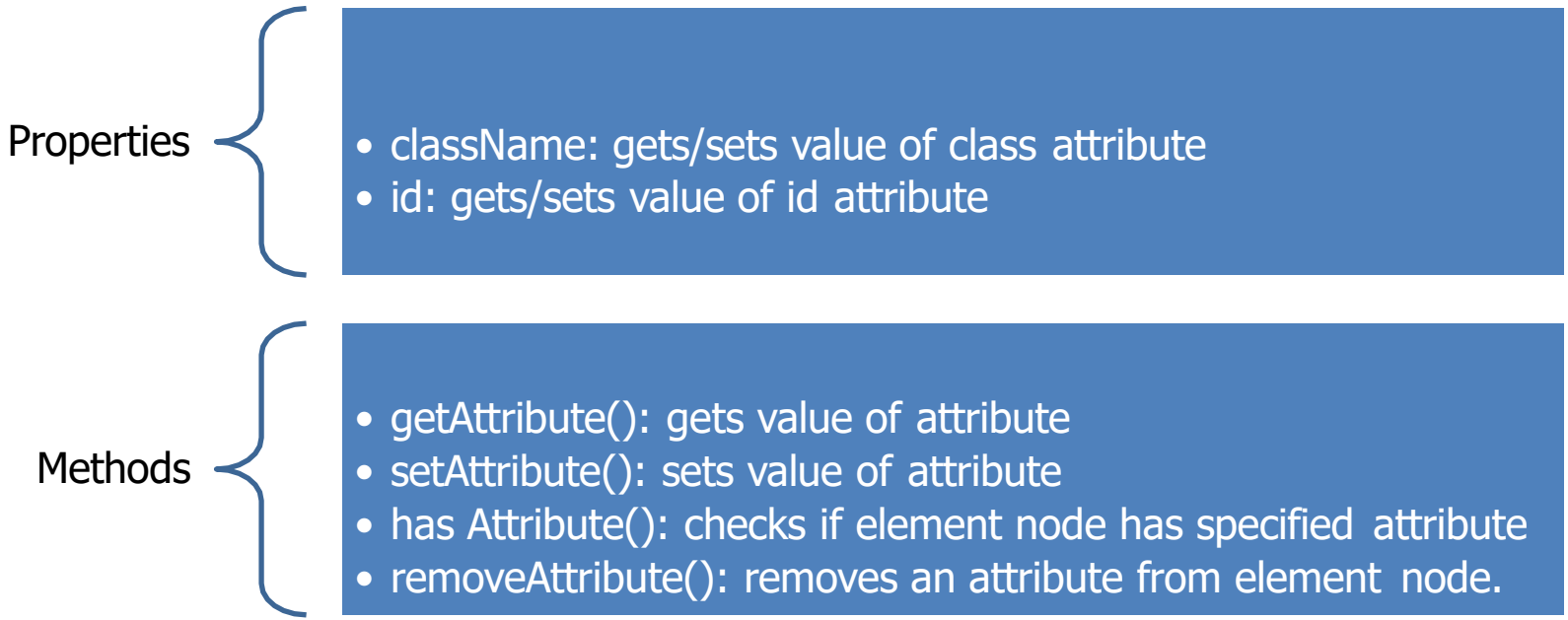
- Create new element using createElement()
- Give content by creating Text node using createTextNode()
- Adds element to DOM tree using appendChild()

Removing elements from DOM

- Store element to remove in a variable
- Store parent of that note in another variable
- Remove element from its containing element using removeChild().

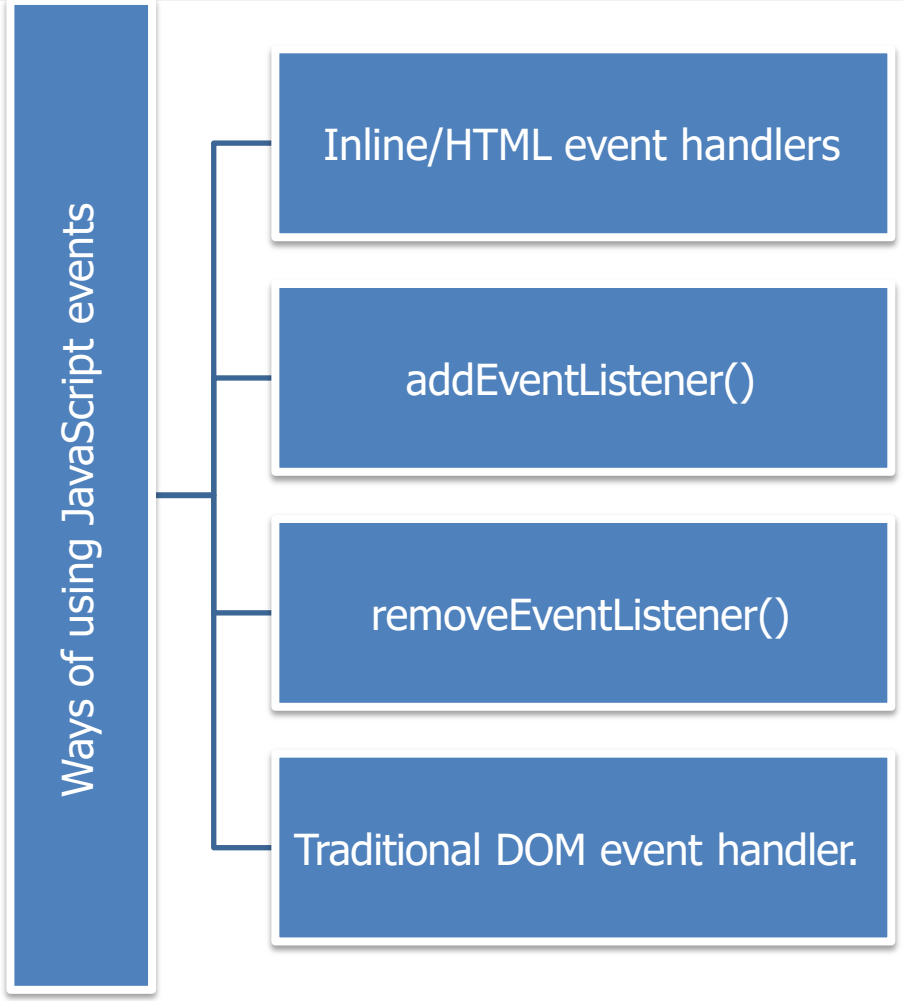
JavaScript DOM(contd…)

- Attribute Nodes



Events

- Any action performed on an instance of browser
- Event handler/listener: block of code runs when event fires
  - Registering event handler: defining block of code
  - Listens for event to happen
  - Runs block of code as response.



Events (contd…)

- Usage:

Inline/HTML event handlers

- Attribute name matches the event names
- **Syntax:** <element attribute = "**functionName**()">

Traditional DOM event handlers

- Separates JavaScript from the HTML
- Drawback: one function per event
- **Syntax:** element.onevent = functionName;

addEventListener()/ removeEventListener()

- Allows a single event to trigger multiple functions
- **Syntax:** element.**addEventListener**("event", functionName [, Boolean]);/ element.**removeEventListener** ("event", functionName [, Boolean]);

Events (contd…)

- Triggering JavaScript from HTML
  - Selecting element node/s to which JavaScript should respond
  - Selecting event that should trigger response
  - Defining code to execute when event occurs.

Events (contd…)

Inline/HTML event handlers

- Attribute name matches the event names
- **Syntax:** *<element attribute = "**functionName**()">*

Traditional DOM event handlers

- Separates JavaScript from the HTML
- Drawback: for any event, you can attach only one function
- **Syntax:** *element.onevent = functionName;*

addEventListener()/removeEventListener()

- Allows a single event to trigger multiple functions
- **Syntax**: *element.**addEventListener**("event", functionName [, Boolean]);*
  *element.**removeEventListener**("event", functionName [, Boolean]);*

Events (contd…)

Event	Description
onchange	Script runs when the element changes
onsubmit	Script runs when the form is submitted
onreset	Script runs when the form is reset
onselect	Script runs when the element is selected
onblur	Script runs when the element loses focus
onfocus	Script runs when the element gets focus
onkeydown	Script runs when key is pressed
onkeypress	Script runs when key is pressed and released
onkeyup	Script runs when key is released
onclick	Script runs when a mouse click
ondblclick	Script runs when a mouse double-click
onmousedown	Script runs when mouse button is pressed
onmousemove	Script runs when mouse pointer moves
onmouseout	Script runs when mouse pointer moves out of an element
onmouseover	Script runs when mouse pointer moves over an element
onmouseup	Script runs when mouse button is released

Closures

- A closure is function having access to parent scope, even after parent function has closed.
- A *closure* is the combination of a function and the lexical environment within which that function was declared.

Import and Export

- To make objects, functions, classes or variables available to the outside world it's as simple as exporting them and then importing them where needed in other files.

- **Export:** You can export a variable using the **export** keyword in front of that variable declaration. You can also export a function and a class by doing the same.

- **Syntax for variable:**
  - `export let variable_name;`
- **Syntax for function:**
  - `export function function_name() { // Statements }`
- **Syntax for class:**
  - `export class Class_Name { constructor() { // Statements }`
    `}`

Import and Export

Import: You can import a variable using **import** keyword. You can specify one of all the members that you want to import from a JavaScript file.

Syntax:

```
import member_to_import from “path_to_js_file”;
```

Example:

```
import Greeting as Greet from "./export.js";
```

Lorem Ipsum Lorem Ipsum

Thank You!