JavaScript

JavaScript

Is Nothing to do with JAVA

It is used to make webpages interactive and provide online programs.

JavaScript

- JavaScript is a scripting language.
- Designed to add interactivity to HTML pages and create dynamic web sites.
  - i.e. Change contents of document, provide forms and controls, animation, control web browser window, etc.
- JavaScript statements embedded in an HTML page can recognize and respond to User Events.
- JavaScript is very simple and flexible.
- Powerful, beautiful and full-fledged dynamic Programming Language

Languages Types

Markup Language

Scripting Language

Programming Language

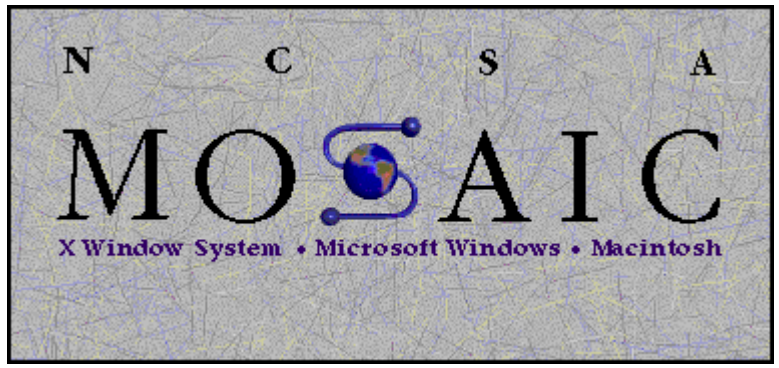
Scripting vs. Programming vs. Markup Language

- Scripting Language
  - Interpreted command by command and remain in their original form.
- Programming Language
  - Compiled, converted permanently into binary executable files (i.e., zeros and ones) before they are run.
- Markup Language
  - A text-formatting language designed to transform raw text into structured documents, by inserting procedural and descriptive markup into the raw text.

Founder

Brendan Eich

Late 1995



1993 introduce images

1995 Font Element And Text Decoration

Brendan Eich

James H. Clark

Marc Andreessen

Java

Scheme

SELF

LiveScript

2 weeks

1995

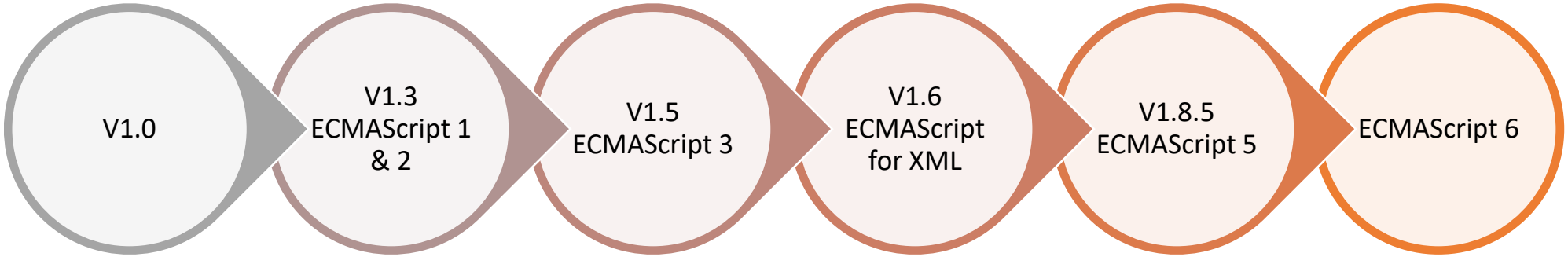
JavaScript

1996

JScript

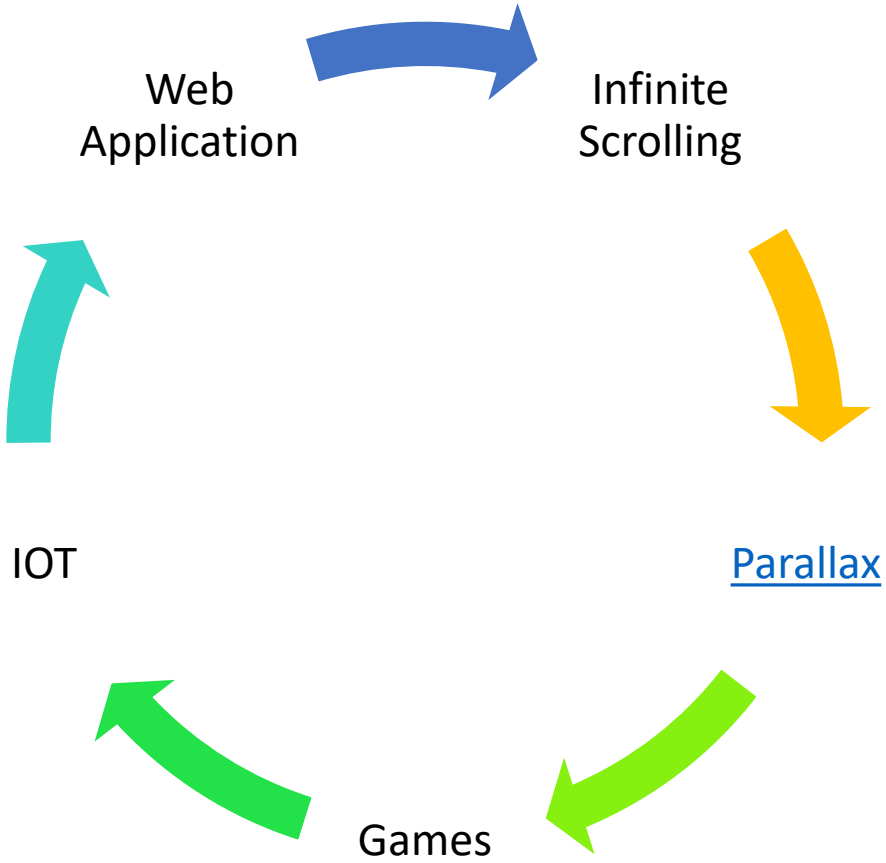
1997

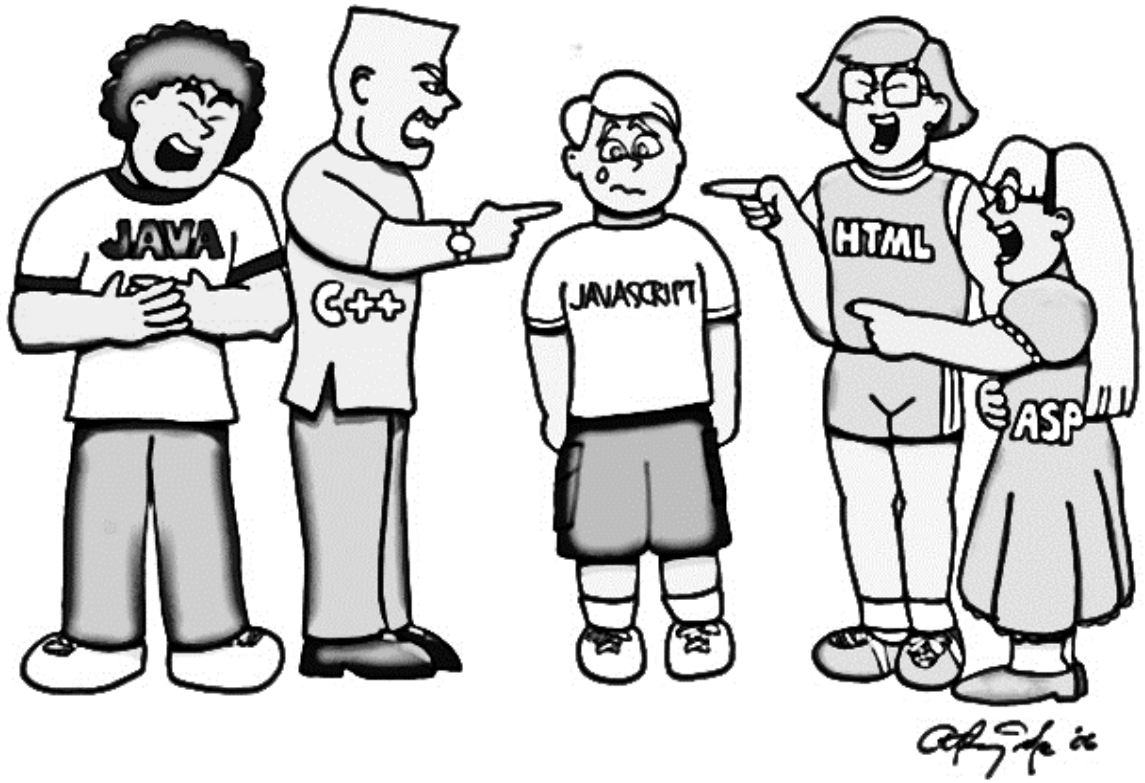
Script


V1.0
V1.3
ECMAScript 1
& 2
V1.5
ECMAScript 3
V1.6
ECMAScript
for XML
V1.8.5
ECMAScript 5
ECMAScript 6


Version

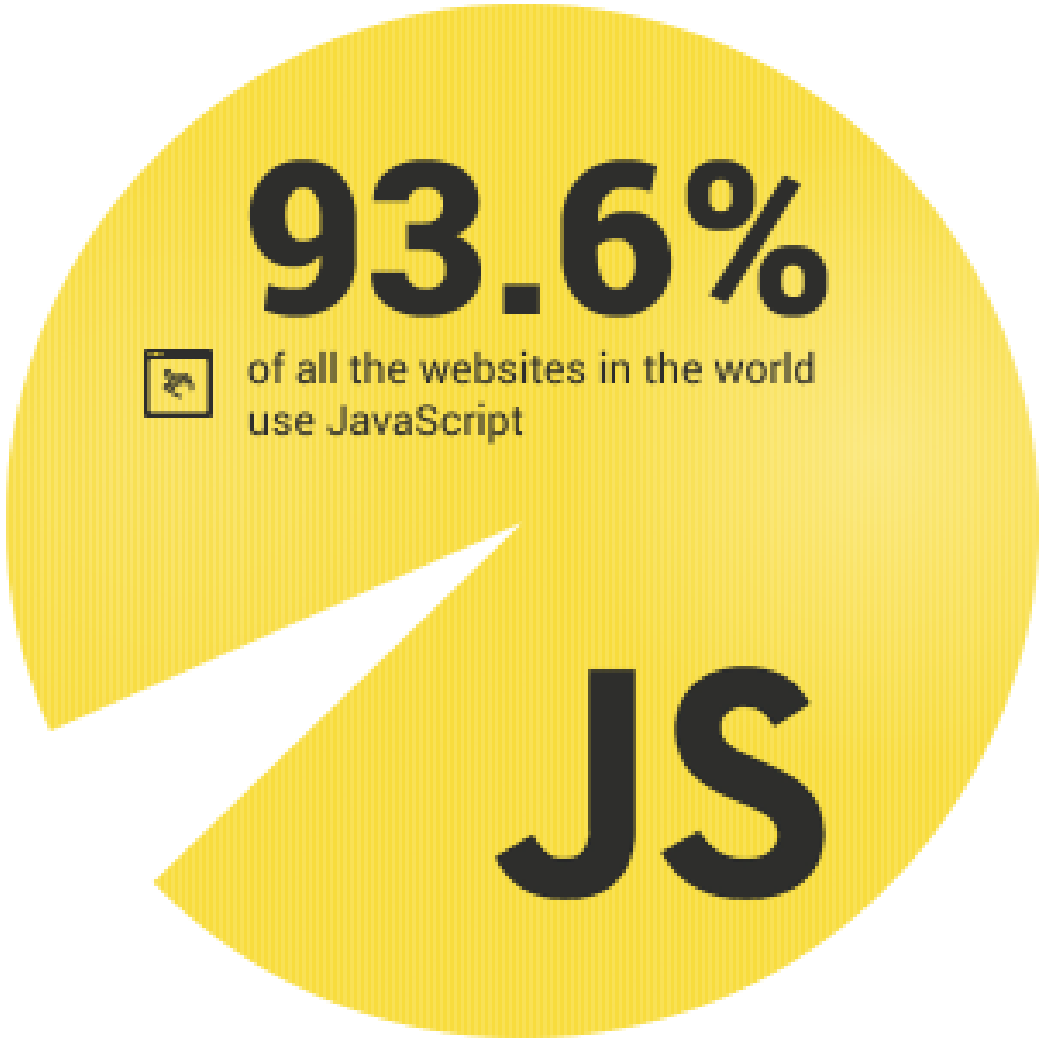
What we can do with JavaScript





Before

After


93.6%
of all the websites in the world
use JavaScript
JS

JavaScript Characteristics

- Case sensitive.
- Object-based.
- Event-Driven.
- Browser-Dependent.
- Interpreted language.
- Dynamic.

Where we can write JavaScript Code

HTML File

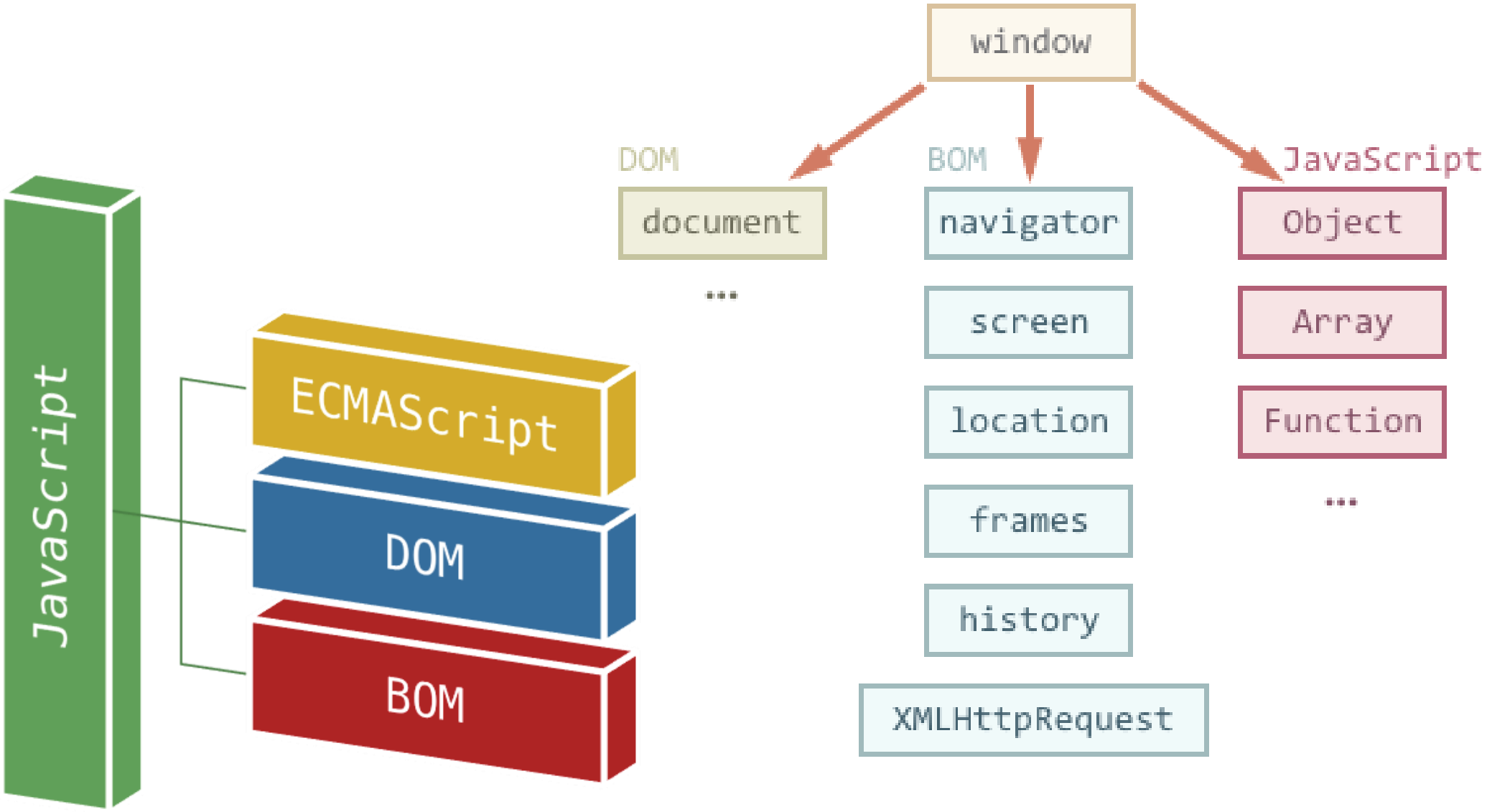
Script Tag

- Head or Body

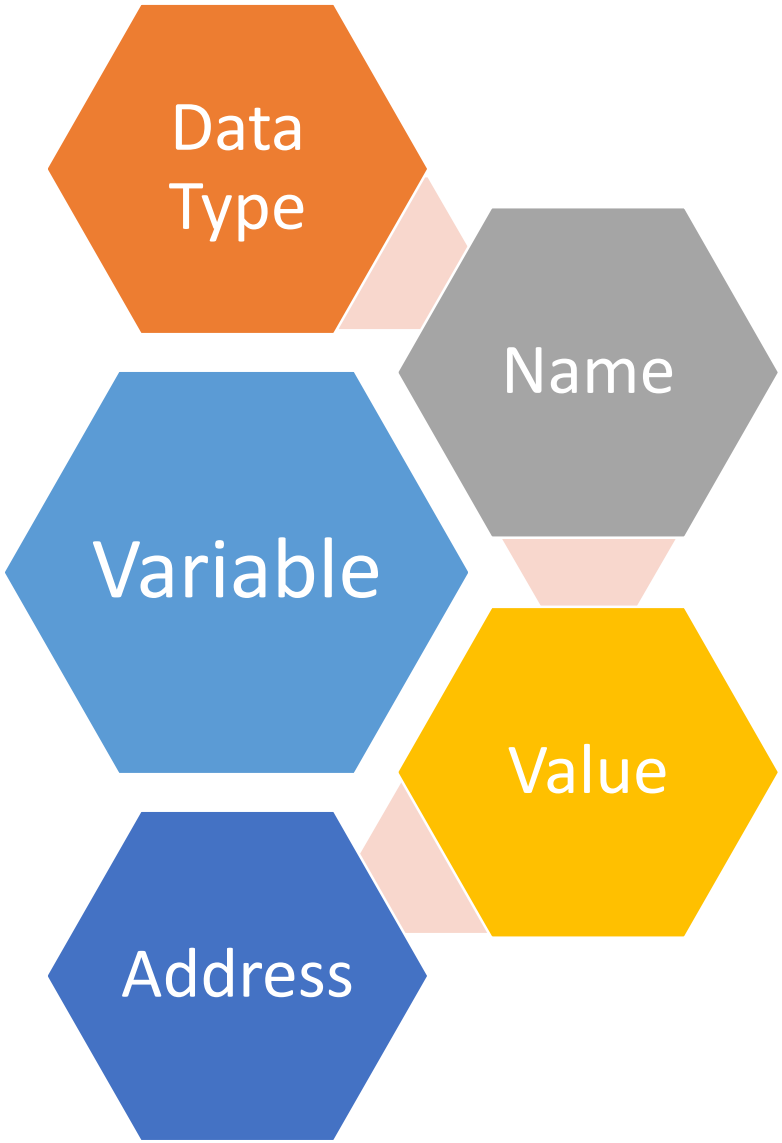
Event Handling

External File

Filename.js + Script tag


JavaScript
ECMAScript
DOM
BOM
window
DOM
document
...
BOM
navigator
screen
location
frames
history
XMLHttpRequest
JavaScript
Object
Array
Function
...

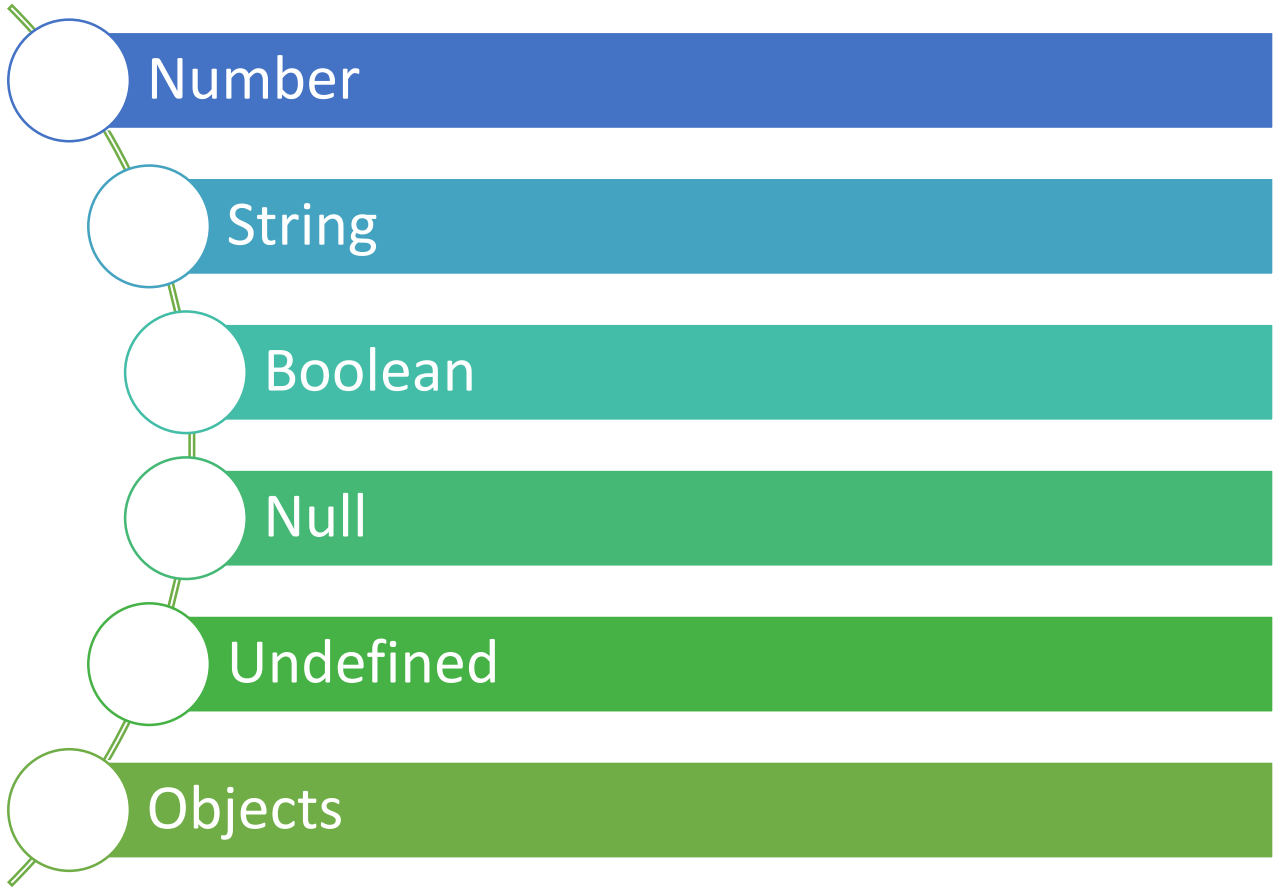
Variables

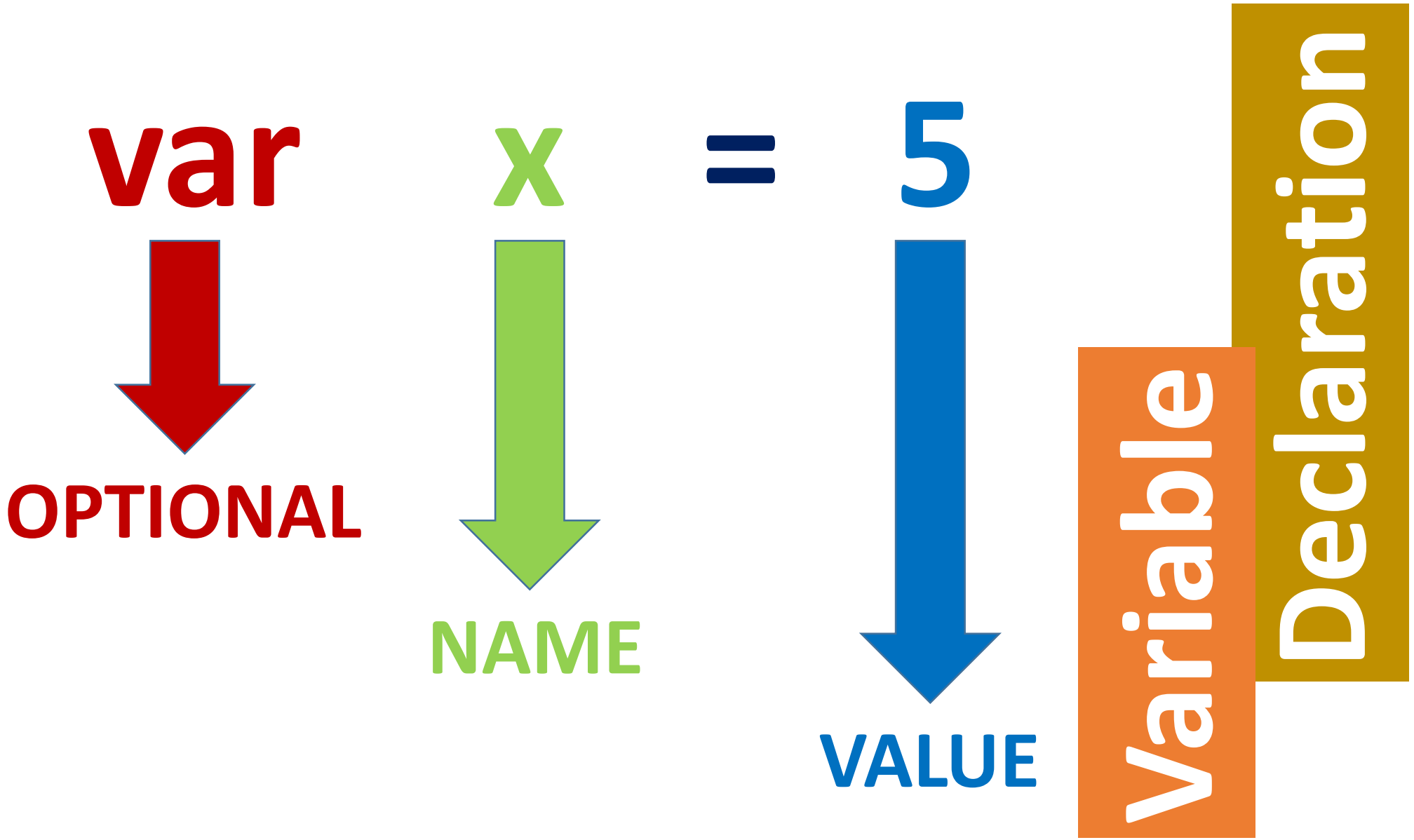


Variables Names

- First char a-z or A-Z or _
- Other char a-z or A-Z or _ or 0-9
- No whitespaces
- Avoid reserved words
- Case-sensitive

Data Types




var
x
=
5
OPTIONAL
NAME
VALUE
Variable
Declaration

Disabled JS

<noscript>any text</noscript>

Comments

```
// single line of text
/* multiline
   of text */
```

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y	x=6
--	Decrement	x=--y	x=4

Assignment Operators

Operator	Example	Description
=	x = y Sets x to the value of y	Assigns the value of the right operand to the left operand
+=	x += y i.e. x = x + y (15)	Adds together the operands and assigns the result to the left operand.
–=	x –= y i.e. x = x – y (5)	Subtracts the right operand from the left operand and assigns the result to the left operand.
*=	x *= y i.e. x = x * y (50)	Multiplies together the operands and assigns the result to the left operand.
/=	x /= y i.e. x = x / y (2)	Divides the left operand by the right operand and assigns the result to the left operand.
%=	x %= y i.e. x = x % y (0)	Divides the left operand by the right operand and assigns the result to the left operand.

Comparison Operators

Operator	Description	Example
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x==="5" is false
!==		
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

Logical Operators

Operator	Definition
&&	Logical "AND" – returns true when both operands are true; otherwise it returns false (Gard Operator)
||	Logical "OR" – returns true if either operand is true. It only returns false when both operands are false (Default Operator)
!	Logical "NOT"—returns true if the operand is false and false if the operand is true. This is a unary operator and precedes the operand

Ternary Operator

(condition) ? true : false

typeof Operator

- typeof Operator
  - A unary operator returns a string that
  - represents the data type.
- The return values of using typeof can be one of the following:
  - "number", "string", "boolean", "undefined","object", or "function".
- Example:
  - var myName = “javascript”;
  - typeof myName; //string

JavaScript Expression

An expression is a part of a statement that is evaluated as a value.

Main types of expressions:

- Left-hand-side “Assignment”
  - a = 25; → assign RHS to variable of LHS
- Arithmetic
  - 10 + 12; →evaluates to sum of 10 and 12
- String
  - “Hello” + ” All !!”; → evaluates to new string
- Logical
  - 25<27 → evaluates to the Boolean value

Coercion

- Coercion is forcing conversion from one data type to another when expression is executed giving a result without causing any error.
- Sometimes gives surprising results from human perspective
- JavaScript engine coerce
  - Number to string 1+”2”➔”12”
  - Boolean to number 3<2<1➔true
- Both undefined and null coerce to false

Operator Precedence

- The operators that you have learned are evaluated in the following order (from highest precedence to lowest):
- 1. Parentheses(())
- 2. Multiply/divide/modulus (*, /, %)
- 3. Addition/Subtraction (+, -)
- 4. Relational (<, <=, >=, >)
- 5. Equality (==, !=)
- 6. Logical and (&&)
- 7. Logical or (||)
- 8. Conditional (?:)
- 9. Assignment operators (=, +=, -=, *=, /=, %=)

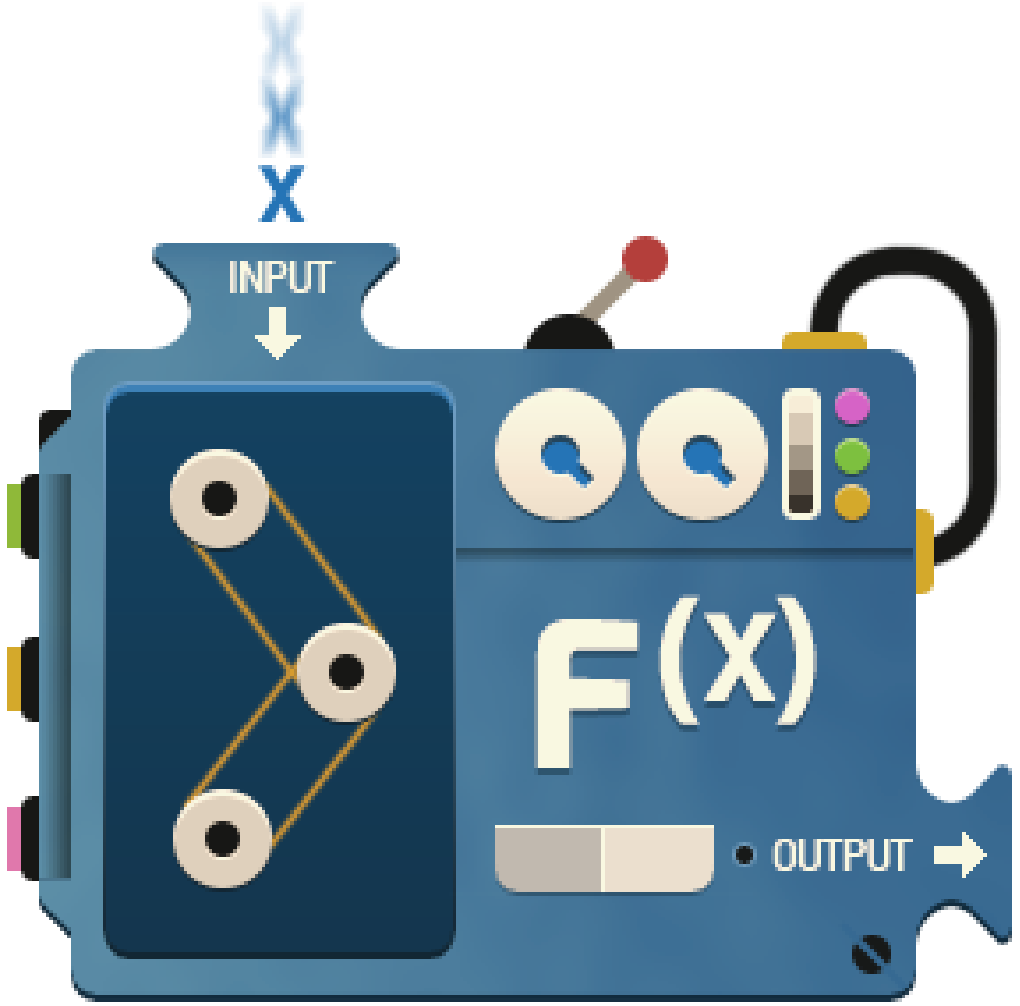
Control Statements

Conditional Statements

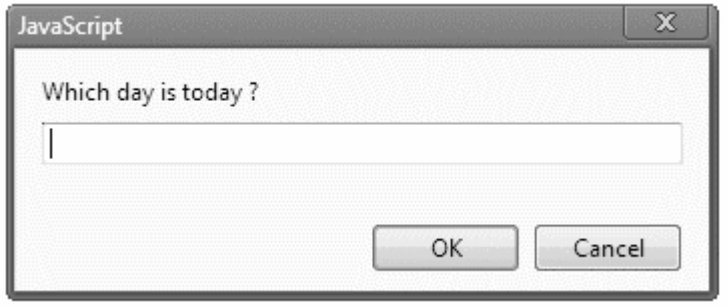
- if
- switch

Loop Statements

- for
- while
- do-while
- for in



Input



Output

Console

```
console.log(any data)
console.warn(any data)
console.error(any data)
```

Communicating with the User

- Four ways of communication:
  - one that displays a text message in a pop-up window
  - one that asks for information in a pop-up window
  - one that asks a question in a pop-up window
  - and one that displays a text message in the browser window.

Dialogues and Alerts

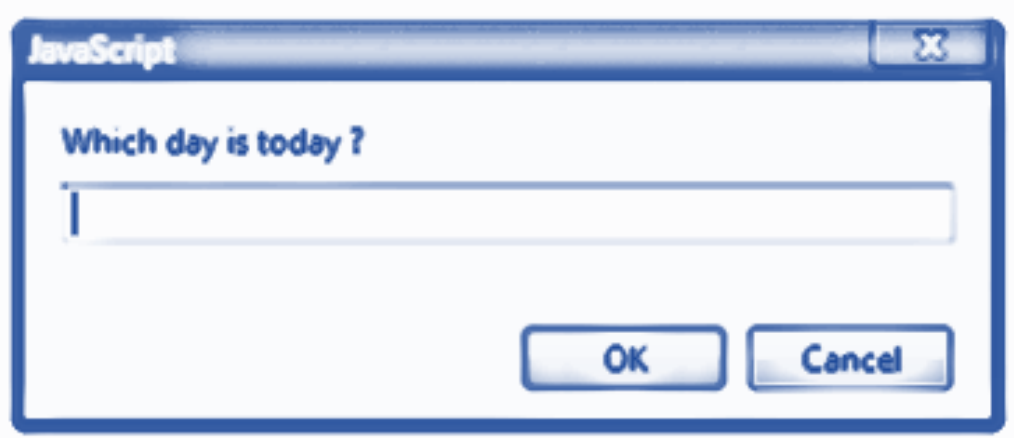
There are 3 types of dialogues:

- **alert()** → Show a message box with one button, has no return.
- **confirm()** → Show a message box with 2 buttons (OK and Cancel).
  - Returns true → OK pressed.
  - Returns false → Cancel pressed.
- **prompt()** → Show a message box with 2 buttons (OK and Cancel) and a textbox
  - Returns the text in textbox →OK pressed
  - Returns null → Cancel pressed

Popup Boxes


JavaScript Alert
This is an alert box
OK


JavaScript
You have to confirm !!
OK
Cancel


JavaScript
Which day is today ?
OK
Cancel

Outputting text with JavaScript (on the current window)

- You can write out plain text or you can mix HTML tags in with the text being written using document.write()
- to return text to the browser screen.
  - document.write(“ ”)
- Example: document.write(”Hello There!“)

JavaScript Functions

A function is an organized block of reusable code (a set of statements) that handles and performs actions generated by user events.

- Functions categorized into
  - built-in functions improve your program's efficiency and readability.
  - user defined functions , created by developer to make your programs scalable.
- Function executes when it is called.
  - from another function
  - from a user event, called by an event or
  - from a separate <script> block.

Global Properties & functions

Function	Description
eval()	Evaluates a string and executes it as if it was script code
isFinite()	Determines whether a value is a finite, legal number
isNaN()	Determines whether a value is an illegal number
Number()	Converts an object's value to a number
parseFloat()	Parses a string and returns a floating-point number
parseInt()	Parses a string and returns an integer
String()	Converts an object's value to a string

Object Categories

Host Objects
BOM
DOM

Built-in objects
Object
Math
String
Boolean
Array
Date
Number

Author Objects
All other objects

JavaScript Objects fall into 4 categories

1. Custom Objects (User-defined)
   - Objects that you, as a JavaScript developer, create and use.
2. Built – in Objects (Native)
   - Objects that are provided with JavaScript to make your life as a JavaScript developer easier.
3. BOM Objects “Browser Object Model” (Host)
   - It is a collection of objects that are accessible through the global objects window. The browser objects deal with the characteristic and properties of the web browser.
4. DOM Objects “Document Object Model”
   - Objects provide the foundation for creating dynamic web pages. The DOM provides the ability for a JavaScript script to access, manipulate, and extend the content of a web page dynamically.

String

Number

Boolean

Array

Date

Math

RegExp

Error

Function

Object

object.
Constructor

Properties

Functions

Object

Properties

Functions

Strings are useful for holding data that can be represented in text form.

String Object

- Enables us to work with and manipulate strings of text.
- String Objects have:
  - Property
    - length : gives the length of the String.
  - Methods that fall into three categories:
    - Manipulate the contents of the String
    - Manipulate the appearance of the String
- To create a String Object
  - var str = new String(‘hello’);

String Object Methods

Method Name	Description
charAt(n)	Return the character at a given position
indexOf(substr,[start])	Searches for first occurrence for a substring.
lastIndexOf(substr,[start])	Searches for last occurrence for a substring.
toUpperCase( )	Returns with all characters converted to uppercase.
trim	removes whitespace from both ends of a string
substring(from, to)	method returns the part of the string between the start and end indexes, or to the end of the string.
slice(start , end)	Extracts a substring of a string. **What's the difference then ?**
split(delimiter, limit)	Return array of strings, from splitting *string* into substrings at the boundaries specified by *delimiter*, constrained by *limit* number of elements.

RegExp

object.

Regular expression object for matching text with a pattern.

RegExp Object

- Regular expressions provide a powerful way to search and manipulate text.
- A Regular Expression is a way of representing a pattern you are looking for in a string.
- A Regular Expression lets you build patterns using a set of special characters. Depending on whether there's a match, appropriate action can be taken.
- People often use regular expressions for validation purposes.
- In the validation process; you don't know what exact values the user will enter, but you do know the format they need to use.

RegExp Object

➔RegExp Method **test()**

- returns a boolean (true when there's a match, false otherwise)
- Example:

```
/j.*t/.test("Javascript")

false                    (case sensitive)
```

➔How to write regExp (self study)

Number Wrapper object

Number object is a wrapper object allowing you to work with numerical values.

Number Object

To create a Number Object

→ var n = new Number(101);

OR

→ n = new Number(); // if not assigned a value initially n =0

→ n=10; // value changed to n=10

Number has a set of Constant values & object methods.

Boolean *Wrapper* object.

Boolean object is an object wrapper for a boolean value.

Boolean Object

The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).

- Everything in the language is either “truthy” or “falsy”
- The rules for truthiness:

  0 , ”” , NaN , null , and undefined → falsy

  Everything else → truthy
- You can convert any value to it’s boolean equivalent by applying “!!” preceding the value
- Example:

  !!”” ➔ false

  !!123 ➔true
- To create Boolean Object

  var b = new Boolean(); → false // typeof is Object

  B = false → false // typeof “boolean”

Boolean Object

➔All the following lines of code create Boolean objects withan initial value of false:

```
var myBoolean=new Boolean()

var myBoolean=new Boolean(0)

var myBoolean=new Boolean(null)

var myBoolean=new Boolean(undefined)

var myBoolean=new Boolean("")

var myBoolean=new Boolean(false)

var myBoolean=new Boolean(NaN)
```

Boolean Object

➔And all the following lines of code create Boolean objects with an initial value of true:

```
var myBoolean=new Boolean(true)

var myBoolean=new Boolean(1)

var myBoolean=new Boolean("false“)

var myBoolean=new Boolean(“anyThing")
```

Math object.

Math is a built-in object that has properties and methods for mathematical constants and functions.

Math Object

- Allows you to perform common mathematical tasks.
- The Math object is a static object.
- Math is a little different from other built-in objects because it cannot be used as a constructor to create objects.
- Its just a collection of functions and constants

Math Object

- Constant Properties:

Constants	value
Math.PI	3.141592653589793
Math.E	2.718281828459045
Math.LN2	0.6931471805599453

Math Object

- Methods

Methods	
Math.abs()	Returns the absolute unsigned value
Math.ceil()	Return rounded number up to nearest interger
Math.cos()	Returns cosine of number
Math.floor()	Returns number down to nearest integer
Math.pow()	Returns the number raised to a power
Math.random()	Returns a number between 0 and 1.0
Math.sqrt()	Returns square root of number
Math.round()	Returns number rounded to closest integer

Date object.

Date instance that represents a single moment in time.

Date Object

- Object to manipulate date and time based on the local machine time.
- Examples:
- var d=new Date() //returns date of local machine now
- Note: The argument month Index is 0-based. This means that January = 0 and December = 11.
- var d = new Date(2008, 1, 1); //31/1/2008
- var weekday = d.getDay( ); //0 ➔ Sunday

Date Types

Type	Example
ISO Date	"2015-03-25" (The International Standard)
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" or "25 Mar 2015"
Full Date	"Wednesday March 25 2015"

Date Object Number Conventions

Date Attribute	Numeric Range
seconds, minutes	0 - 59
hours	0 - 23
day	0 - 6 (0 = Sunday, 1 = Monday, and so on)
date	1 - 31
month	0 - 11 (0 = January, 1 = February, and so on)
year	0 + number of years since 1900

Array object

Array object is a global object that is used in the construction of arrays.

Array Object

- Array is a special type of object
- It has length property:
  - gives the length of the array It is one more than the highest index in the array
- Declaring an array
  - var myarr=[]; //declaring an array literal
  - var myarr=[1 , 2 , ‘three’,’four’,false];
  - var myarr= new Array() //using array constructor.
  - var myarr=new Array(2) //an array with 2 undefined elements
  - var myarr=new Array(1,2,’r’,true); //an array with 4 elements

Array Methods

Method	Description
concat()	Concatenates elements from one array to another array.
join()	Joins the elements of an array by a separator to form a string.
pop()	Removes and returns the last element of an array.
push()	Adds elements to the end of an array.
reverse()	Reverses the order of the elements in an array.
shift()	Removes and returns the first element of an array.
slice([*begin*[, *end*]]**)**	Creates a new array from elements of an existing array.
sort()	Sorts an array alphabetically or numerically.
Splice(*start[, deleteCount[, item1[, item2[, ...]]]]***)**	Removes and/or replaces elements of an array.
toString()	Returns a string representation of the array.
unshift()	Adds elements to the beginning of an array.

function object.

Every JavaScript function is actually a Function object.

User-defined functions

Function blocks begin with the keyword function followed by the function name and () then {} its building block declaration.

Syntax:

```
function functionName (argument1, argument2, ...)
{
    //statement(s) here
    //return statement;
}
```

return can be used anywhere in the function to stop the function's work. Its better placed at the end.

Functions

Defining function

```
function myFunction(x,l)
{
    var y = x+l;
    return y;
}
```

Calling function

```
var z = myFunction(5,6)
```

User-defined functions

- Function can be called before its declaration block.
- Functions are not required to return a value
  - Functions will return undefined implicitly if it is to set explicitly
- When calling function it is not obligatory to specify all its arguments
  - The function has access to all of its passed parameters via arguments collection
  - We can specify default value using || operator or ES6 default function parameters

Variable

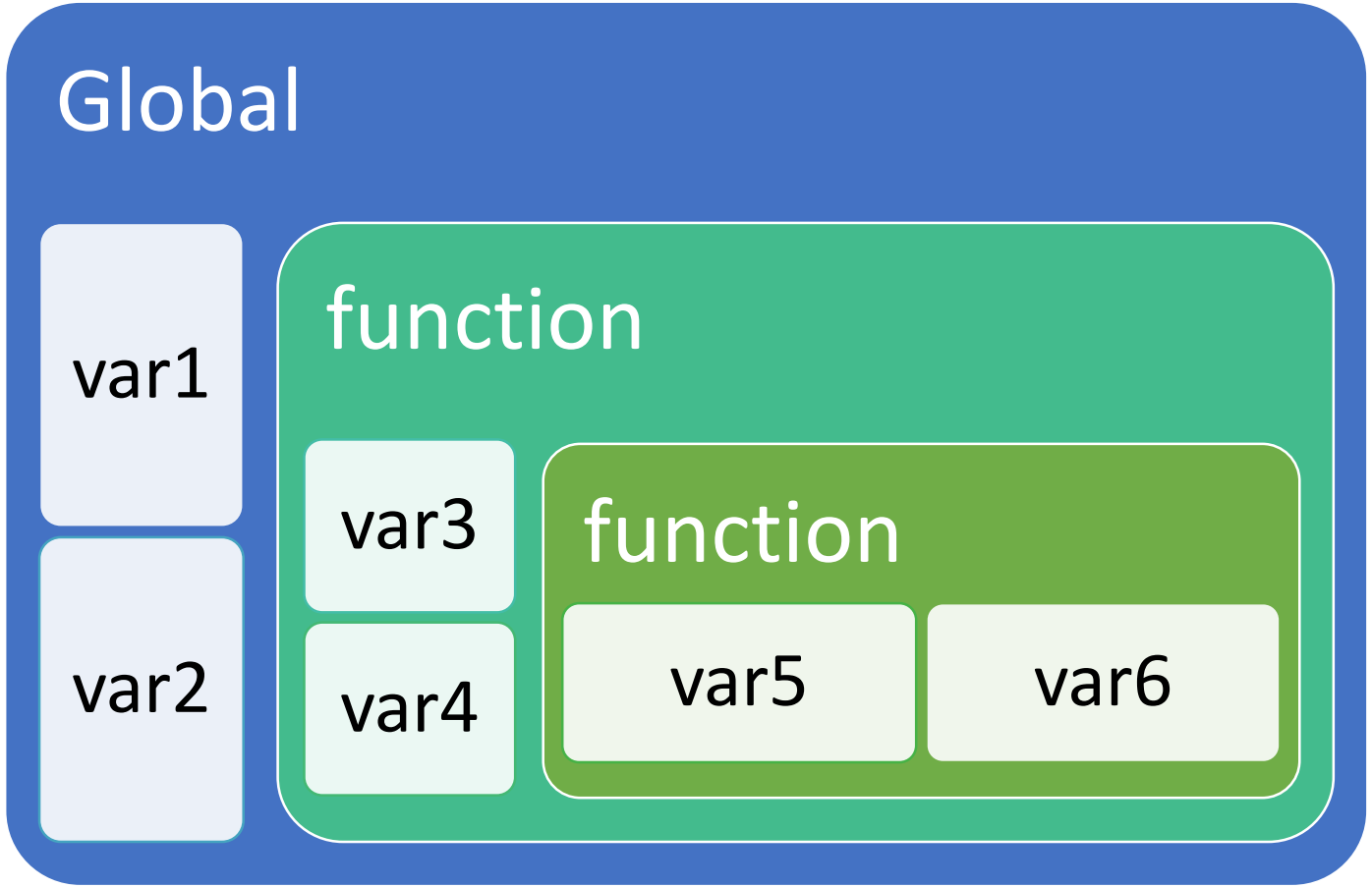
Scope

Life Time

JavaScript Variables Lifetime

- **Global Scope**
  - A variable declared outside a function and is accessible in any part of your program
- **Local Scope**
  - A variable inside a function and stops existing when the function ends.

Understanding Scopes



Variable Scope

- It is where var is available in code
- All properties and methods are in the public global scope
  - They are properties of the global object “window”
- There was no Block Scope before ES6, only function scope
- Variables declared
  - Inside a function are local variable
  - outside any function are global variable
    - i.e. available to any other code in the current document

Variable Hoisting

Hoisting

- Hoisting takes place before code execution
- Variables
  - Any variable declared with var is hoisted up to the top of its scope
  - Hoisted variables are of undefined value.
  - We can refer to a variable declared later without getting any exception or error
- Functions
  - Function statements are hoisted too.
  - Functions are available even before its declaration

What is the output of the execution?

```
function b() {
    var myvar = 1;
    console.log(myvar);
}
console.log(x);
a();
b();

function a() {
    var myvar = 2;
    console.log(myvar);
}
var x = 5;
```

Lexical Environments

- In JavaScript environment there are 2 main types of Execution Context.
- First is Global Execution Context, when your code is initially run even if it's spread across to a page using a <script /> tag,
- Second is the Function Execution Context from the word itself it was created when you invoked the function that you define.

Lexical Environments

- When the javascript code runs, JavaScript engine create one Global Execution Context and push it to Execution Context Stack.
- Execution Context Stack — is a placed where our Global and Function Execution Context was handled and manipulated.

What is the output of the execution?

```
function f1(x,y)
        {
            if(x>3)
            {
                var z=3;
            }
            else
            {
                var zz=3;
            }
console.log(z,zz)
            zzz=300;
        }
        f1(1,3);
        alert(zzz);
```

What is the output of the execution?

```
function f1(x,y)
        {
            if(x>3)
            {
                var z=3;
            }
            else
            {
                var zz=3;
            }
console.log(z,zz)
            zzz=300;
        }
        //f1(1,3);
        alert(zzz);
```

What is the output of the execution?

```
function b() {
    var myvar;
    console.log(myvar);
}

function a() {
    var myvar = 2;
    console.log(myvar);
    b();
}

var myvar = 1;
console.log(myvar);
a();
console.log(myvar);
```

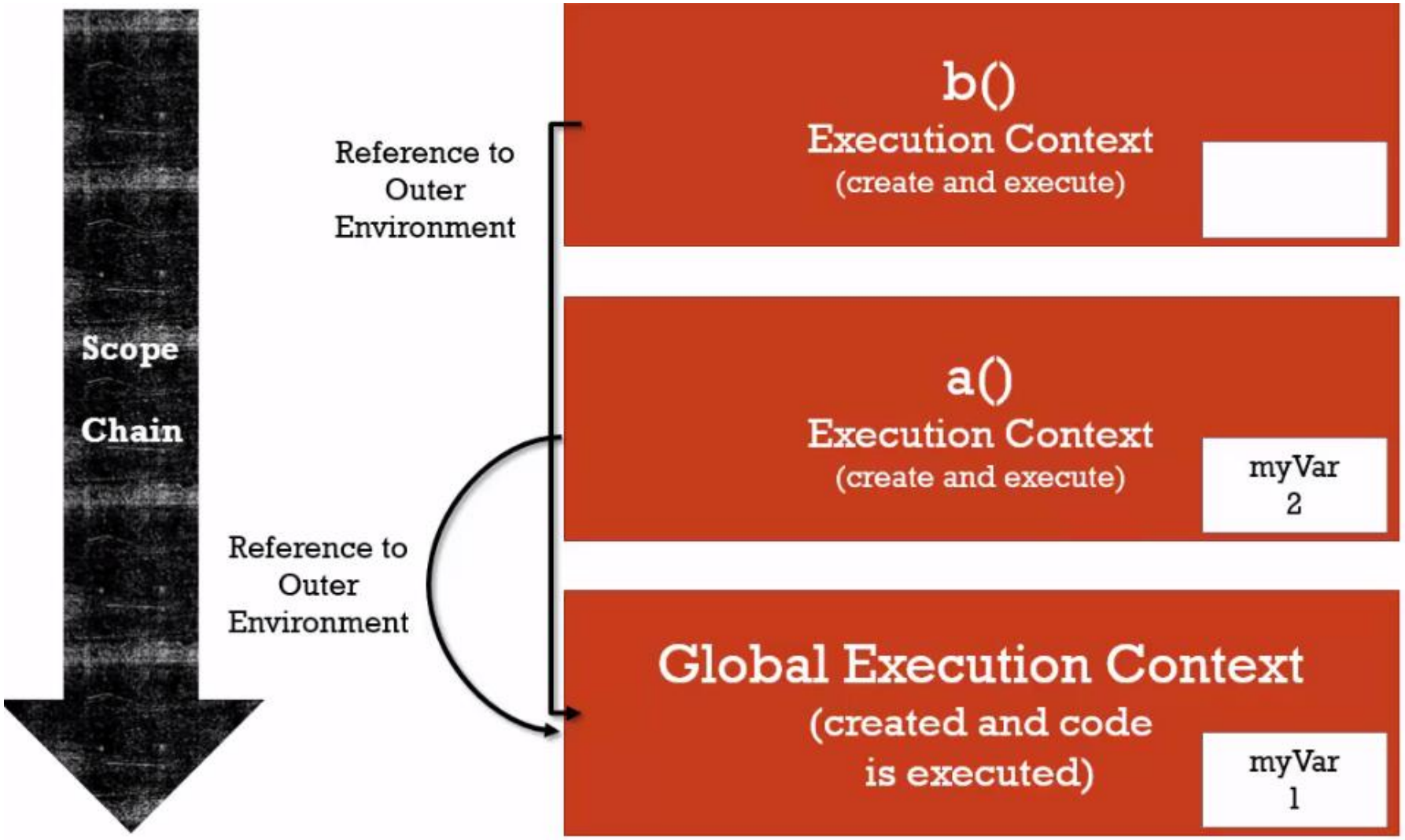
What is the output of the execution?

```
function b() {
    //var myvar;
    console.log(myvar);
}

function a() {
    var myvar = 2;
    console.log(myvar);
    b();
}

var myvar = 1;
console.log(myvar);
a();
console.log(myvar);
```

What is the output of the execution?

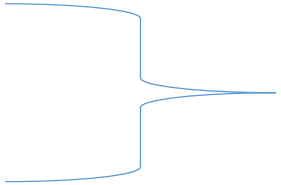
```
function b() {
     console.log(myvar);
}

function a() {
   var myvar = 2;
   console.log(myvar);
   b();
}
a();
console.log(myvar);
```

Scope chain


Scope
Chain
Reference to
Outer
Environment
Reference to
Outer
Environment
b()
Execution Context
(create and execute)
a()
Execution Context
(create and execute)
myVar
2
Global Execution Context
(created and code
is executed)
myVar
1

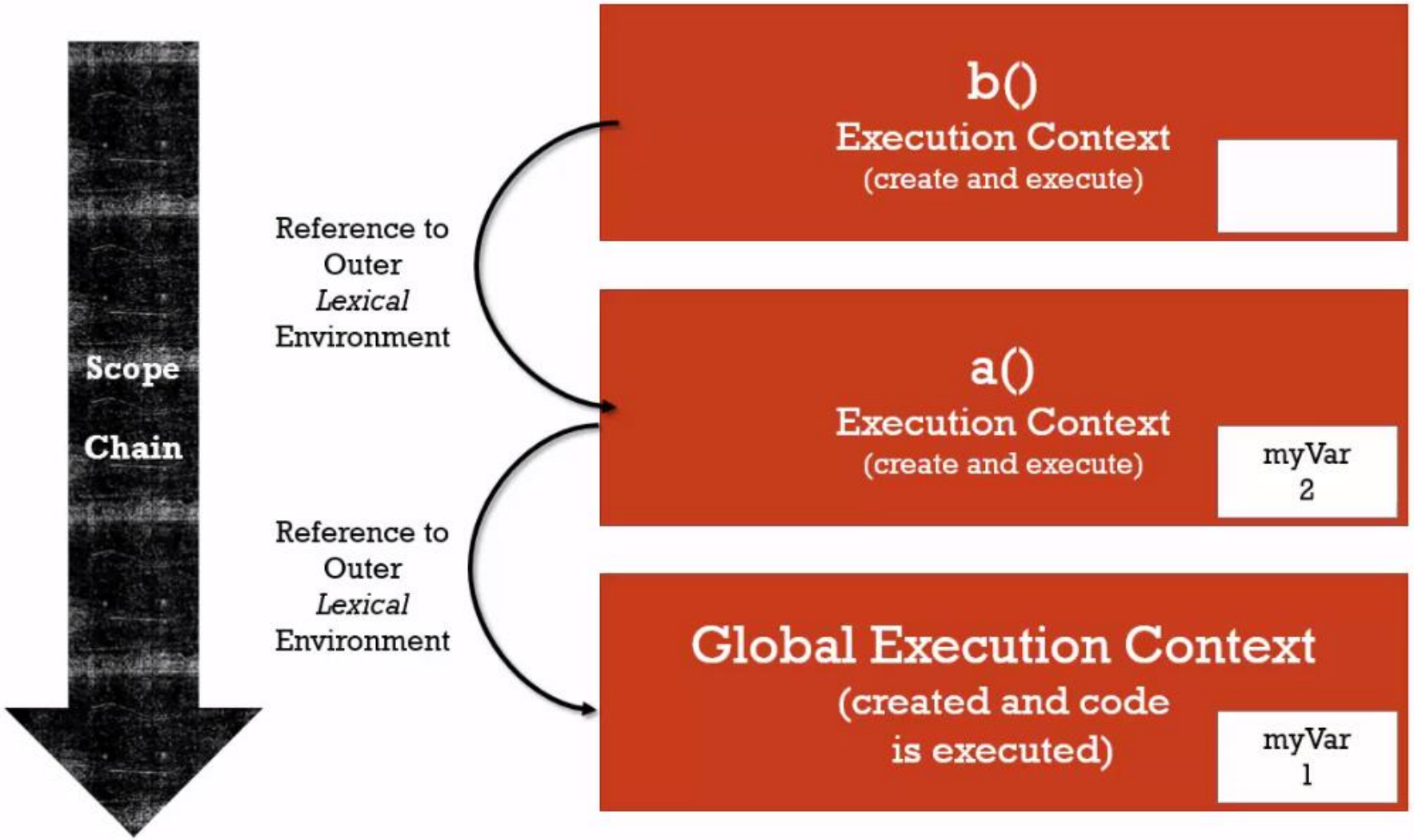
What is the output of the execution now?

```
function a() {
   function b() {
      console.log(myvar);
   }
   var myvar = 2;
   console.log(myvar);
   b();
}
a();
console.log(myvar);
```



Changing the Lexical Environment

Scope chain


Scope
Chain
Reference to
Outer
Lexical
Environment
Reference to
Outer
Lexical
Environment
b()
Execution Context
(create and execute)
a()
Execution Context
(create and execute)
myVar
2
Global Execution Context
(created and code
is executed)
myVar
1

What is the output of the execution?

```
function square(num)
{
        total = num * num;
        return total;
}
var total = 50;
var   number = square(20);
alert(total);
```

Functions (Cont'd)

- Anonymous functions is a function without name
- Example:
  - var f=function(a,b){return a+b}
- To call it,
  - var x=f(1,2) ➔ x=3;
- Self-invoked function: are anonymous functions invoked just after it declaration.
- Anonymous functions are used as:
  - Callback functions
  - Self-invoked functions
- To self-invoke an anonymous function add () to the end of its definition
- Example:
  - var x=(function(a,b){return a+b})(1,2) ➔ x=3

Functions (Cont'd)

- Callback functions:
- Functions sent to other functions.
- Example:
  - function invoke_and_add(a, b)
  {
  return a() + b();
  }
  - function one() {return 1;}
  - function two() {return 2;}
  - invoke_and_add(one, two);
  - invoke_and_add(function(){return 1;}, function(){return 2;})

Function expression hoisting

- Function expressions in JavaScript are not hoisted, unlike function declarations. You can't use function expressions before you create them

```
console.log(s);
s();
var s = function(){
    console.log("func")
}
s();
```

Error Handling

try

catch

throw

Debugger

Window object

a global object, where all global variables are properties of it all methods defined are also properties of it, can be called by name or through window object

Host Objects

- Since the browser is the native host environment for client-side JavaScript programs, there are collections of objects dealing with it.
- The BOM (Browser Object Model)
  - represents objects dealing with what's outside loaded page, through the global object
    - window
- The DOM (Document Object Model)
  - Represents loaded page through object
    - window.document or document

Document object

Document object is the root node of the HTML document and the "owner" of all other nodes

E BUSINESS

DOM

document

elements	attributes	style	events

HTML DOM Nodes

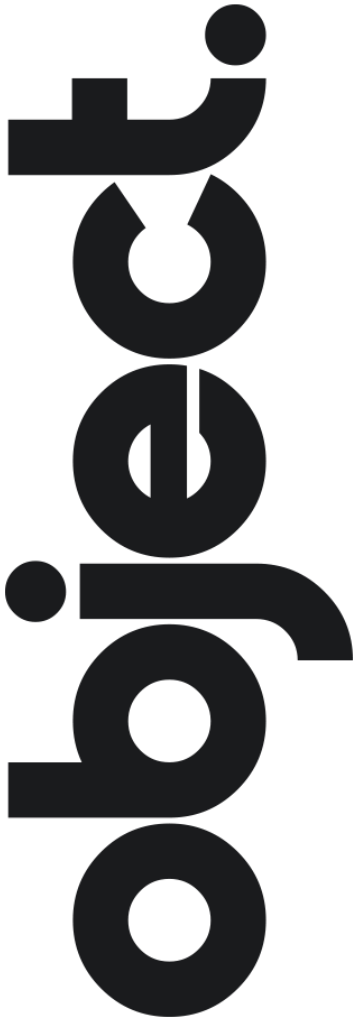
- The document itself is a document node
- All HTML elements are element nodes
- All HTML attributes are attribute nodes
- Text inside HTML elements are text nodes
- Comments are comment nodes

E BUSINESS

DOM

- The Document Object Model represents the document as a node tree (parent, child, and sibling).
- Types of Nodes:
- Element Nodes
- Text Nodes
- Attribute Nodes
- Comment Nodes

Elements



Element object represents an HTML element.
Element objects can have child nodes of type element, text, or comment nodes.

E BUSINESS

attributes

object

Attr object represents an HTML attribute.

E BUSINESS

events object.

Events allow JavaScript to register different event handlers on elements in an HTML document.

E BUSINESS

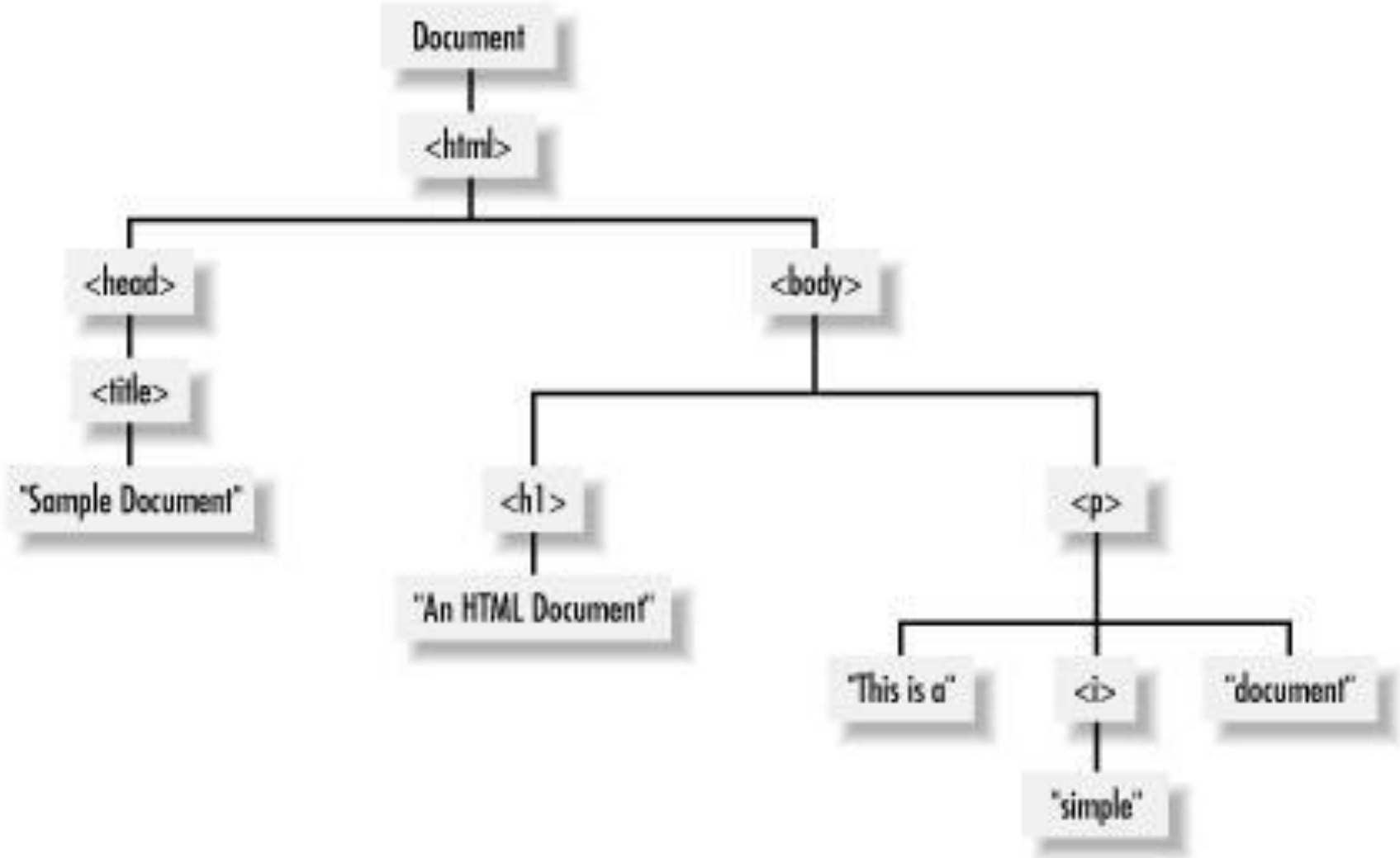
style object.

Style object represents an individual style statement.

E BUSINESS

HTML Example

- <html>
  - <head><title>Sample Document</title></head>
  - <body>
    - <h1>An HTML Document</h1>
    - This is a simpledocument
  - </body>
- </html>



The tree representation of an HTML document

Accessing DOM Nodes

- All nodes have
  - nodeType,
  - nodeName (mostly are tag names),
  - nodeValue (for text nodes; the value is the actual text)
- **window.document:** access to the current document.
  - **documentElement:** the root node *<html>*
  - **childNodes**:
    - In order to tell if a node has any child Nodes you use hasChildNodes(): ➔*true/false*
    - *childNodes[]* ➔ *array to access child nodes of any element, has all array properties.*
    - *a property of Nodes*
  - **children:**
    - *a property of Elements, returns array of all child elements of an element.*
  - **parentNode/parentElement(DOM4):**
    - Provided to child to access its parent

Accessing DOM Nodes

- **window.document (Cont'd):**
  - nextSibling/nextElementSibling, previousSibling/previousElementSibling
  - firstChild/firstElementChild, lastChild/lastElementChild
  - attributes
    - In order to tell if a node has any attributes you use hasAttributes() ➔ *true/false also hasAttribute("attr")*
    - *attributes[]* ➔ *array to access all attribute nodes of any element.*
    - getAttribute(attrname) ➔ to get the value of a certain attribute.
    - setAttribute(attrname,value) ➔ to set value of a certain attribute.
    - Each attribute node has nodeName and nodeValue

A Shortcut to DOM elements

- document.getElementById(id) returns an object representing element or null if not found
- document.getElementsByTagName(tag) returns a collection of objects with the specified tag name or [] an empty array if not found
- document.getElementsByName(name) returns a collection of objects with the specified name attribute or [] an empty array if not found
- document.getElementsByClassName(name) returns a collection of objects with the specified class attribute or [] an empty array if not found

Modifying Style

- Most of the visual element nodes have a style property, which in turn has a property mapped to each CSS property.
- Example:

```
var my = document.getElementById(’mydiv');
my.style.border = "1px solid red";
```

- CSS properties have dashes but in JavaScript map their names to properties by skipping the dash and uppercase the next letter.
- padding-top, paddingTop, margin-left, marginLeft…..

Modifying Style (Cont'd)

- You can change the css class using the className property or setAttribute() function.

Example:

```
var m=document.getElementById('mydiv');

m.className="errorclass";

OR

m.setAttribute('class','errorclass');
```

Creating New Nodes

- Create nodes by createElement() and createTextNode().
- Once you have the new nodes, you add them to the DOM tree with appendChild(). It must be called by the parent object to whose
- children the node is added.
- Example

```
var myp = document.createElement('p');
myp.innerHTML = 'yet another';
myp.style.border = '2px dotted blue’
document.body.appendChild(myp); //here appended to end of body directly
```

Clone Existing Node

- cloneNode()
- The method accepts a boolean parameter (true = deep copy with all the children, false = shallow copy, only this node).
- Get a reference to the element you want to clone:

```
var el = document.getElementsByTagName('p')[1];
```

- Create a shallow clone of el and append it to the body:

```
document.body.appendChild(el.cloneNode(false))
```

- Create a deep copy, the whole DOM subtree

```
document.body.appendChild(el.cloneNode(true))
```

insertBefore

- insertBefore is the same as appendChild(), but accepts an extra parameter, specifying before which element to insert the new node. It must be called through the parent where the element will be added.
- insertBefore(newnode, existingchild)
- Example:
- At the end of the body:

```
document.body.appendChild(document.createTextNode('boo!'));
```

- Add it as the first child of the body:

```
document.body.insertBefore(document.createTextNode('boo!'), document.body.firstChild);
```

Removing/Replacing Nodes

- **removeChild().**
  - *Specify node to be removed, send it to removeChild*

    var removed = document.body.removeChild(myp);
  - The method returns the removed node if you want to use it later.
- **replaceChild()**
  - Removes a node and puts another one in its place.
  - First specify node to remove and node to add, then send them to function

    var replaced = document.body. replaceChild(*newnode*,*oldnode*)
  - It returns a reference to the node that is now out of the tree.

Events

- HTML creates the visual image of controls, but they are mute.
- JavaScript animates it all and makes the page interactive through events, like a button press, a mouse hover, a text changed, a link pressed……etc.
- Every HTML element has several events attached to it. These events need event handlers.
- Event handler is JavaScript code that is executed when an event is triggered

Events (Cont'd)

- Events can be attached to JavaScript code through one of the following ways:
  - Inline HTML Attributes
  - Element Properties (will be discussed later)
  - DOM Event Listeners (will be discussed later)

Inline HTML Attributes

- Using event handler attributes inside the HTML tags.
- We can embed JS code inside it or a call to JS function.
- The attributes start with ‘on’ then the event name
- Examples: onclick, onmouseover……etc.

```
<html>
<head><title>Event</title></head>
<body>
    <form>
        <input type ="button"  value = "Click here"  onClick="alert(Hello!!')” >
    </form>
</body>
</html>
```

Types of Events

- **Mouse events**
  - mouseup, mousedown, click, dblclick, mouseover, mouseout, mousemove.
- **Keyboard events**
  - keydown, keypress, keyup.
- **Loading/window events**
  - load, unload, beforeunload, abort, error, resize, scroll, contextmenu
- **Form events**
  - change, select, reset, submit.

Elements Properties

- assign a function to the on-event property of a DOM node element.
- Example:

```
<div id="my-div">click</div>
<script type="text/javascript">
var myelement = document.getElementById('my-div');
myelement.onclick = function()
{
    alert('Div Clicked!');
}
</script>
```

- You can attach only one function to the event.

DOM Event Listeners

- assign a function to the on-event property of a DOM node element.
- Example:

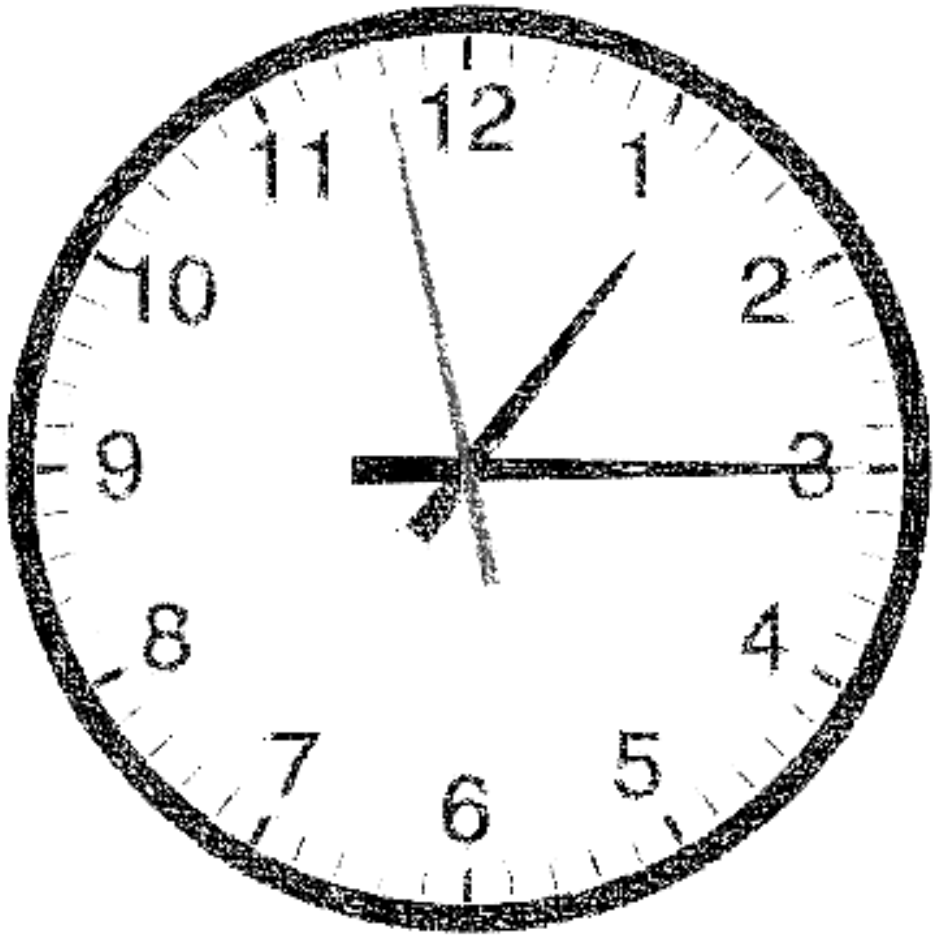
```
<div id="my-div">click</div>
<script type="text/javascript">
var myelement = document.getElementById('my-div');
myelement.onclick = function()
{
    alert('Div Clicked!');
}
</script>
```

- You can attach only one function to the event.

Always keep in mind that HTML is for content, JavaScript for behavior and CSS for formatting, and you should keep these three *SEPARATE* as much as possible.

Event Object

The event object is passed implicit to the event handling function as an argument



Timing Events

setTimeout()

clearTimeout()

setInterval()

clearInterval()

E BUSINESS

BOM

window

A collection of objects access the browser and the computer screen.

Accessible through the global objects window.

There's a window object for every popup, or browser tab.

BOM

window

navigator	location	history	document

E BUSINESS

window.open()/close()/print()

- open() → which open up new browser windows (popups).
  - It returns a reference to the window object of the newly created browser instance.
  - window.open(url, name, target, featurelist)
  - Features can be:
    - resizable: user is able to resize the new window
    - width, height
- close()→ closes the new window.
- print() →to print page.

navigator object

Navigator object contains information about the browser.

E BUSINESS

navigator

- The navigator is an object that has some information about the browser and its capabilities. It is used to identify the browser and provide different versions of the code.
- One of its important properties is navigator.userAgent. It is a long string of browser identification.
- Example: In Firefox
- >>> window.navigator.userAgent
- "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:16.0) Gecko/20100101 Firefox/16.0”

History object

History object contains the URLs visited by the user (within a browser window).

E BUSINESS

history

- It allows limited access to the previously-visited pages in the same browser session.
- history.length ➔ the number of pages visited before this one.
- history.forward() ➔ navigate forward through the user's session (Forward Button)
- history.back() ➔ navigate backward through the user’s session (Backward Button).
- history.go(num) ➔ To go to a certain page in the history list. If num is negative, we go backwards and if positive we go forward.

location object

Location object contains information about the current URL.

E BUSINESS

location

- It is an object that contains information about the URL of the currently loaded page.
- For the following URL : http://search.phpied.com:8080/search?p=javascript#results
- location.href = "http://search.phpied.com:8080/search?p=javascript#results"
- location.hash = "#results"
- location.host = "search.phpied.com:8080"
- location.hostname = "search.phpied.com"
- location.pathname = "/search"
- location.protocol = "http:“
- location.search = "?p=javascript”

location

- Methods:
- reload() ➔To reload a page.
- replace(newurl) ➔ Replaces the current document with a new one

Object Categories

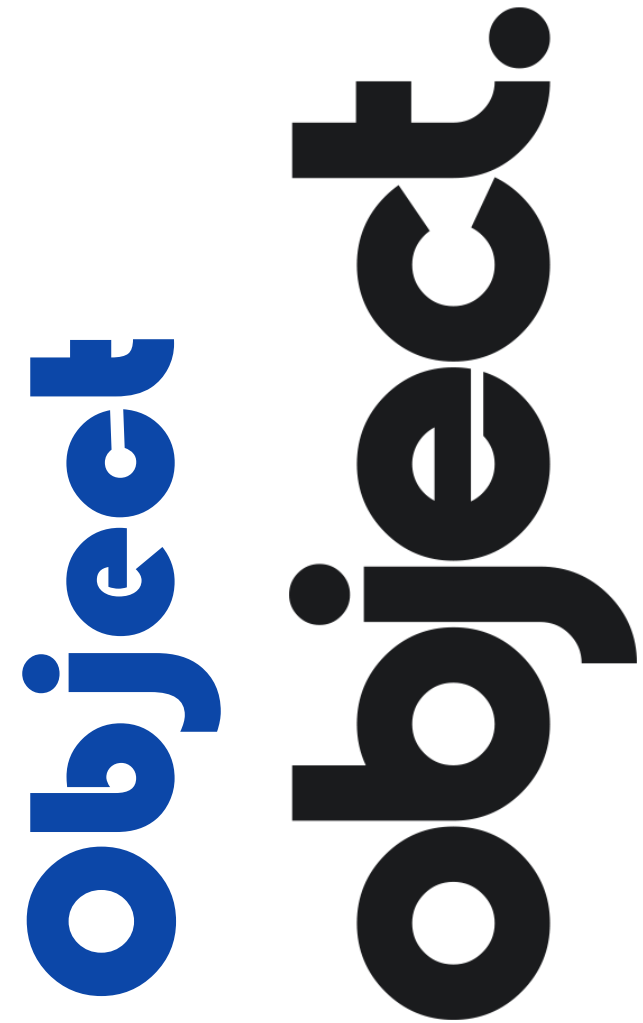
Host Objects
BOM
DOM

Built-in objects
Object
Math
String
Boolean
Array
Date
Number

Author Objects
All other objects

JavaScript Objects fall into 4 categories

1. Custom Objects (User-defined)
   - Objects that you, as a JavaScript developer, create and use.
2. Built – in Objects (Native)
   - Objects that are provided with JavaScript to make your life as a JavaScript developer easier.
3. BOM Objects “Browser Object Model” (Host)
   - It is a collection of objects that are accessible through the global objects window. The browser objects deal with the characteristic and properties of the web browser.
4. DOM Objects “Document Object Model”
   - Objects provide the foundation for creating dynamic web pages. The DOM provides the ability for a JavaScript script to access, manipulate, and extend the content of a web page dynamically.



Object creates an object wrapper or a user defined object.

E BUSINESS

Object

Take a "thing"

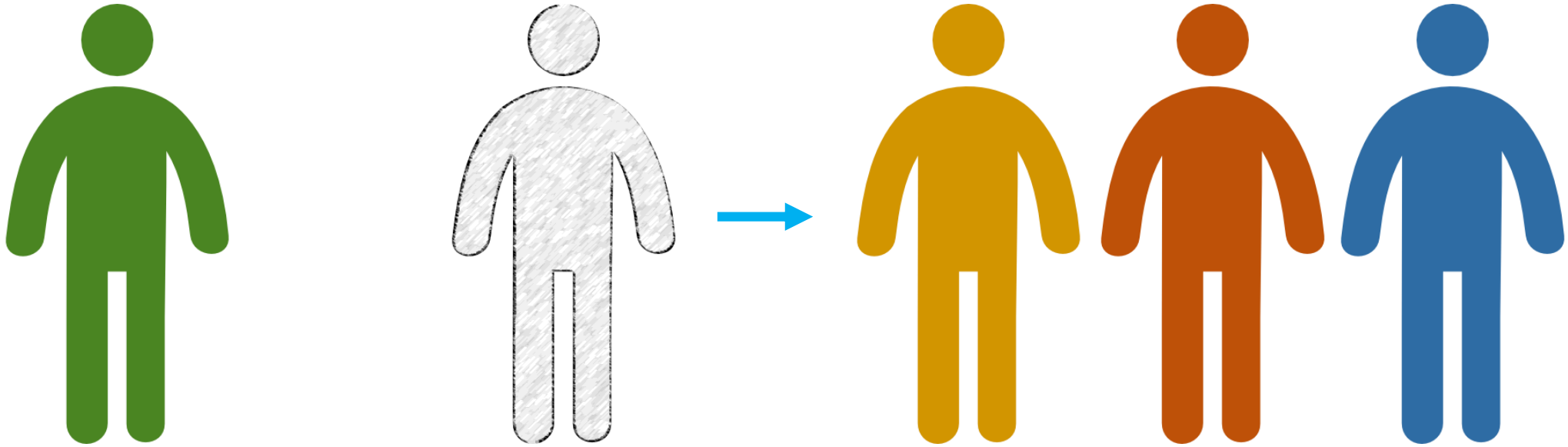
Object

Describe the "thing"

Characteristics	Actions
Width	Speed up
Height	Break
Color	Turn right
⬇	⬇
Properties	Methods

E BUSINESS

Object



Object literal Object Constructor

E BUSINESS

Object

Properties

Methods

Constructors

REFERENCES

w3schools.com

E BUSINESS

Questions ???

E BUSINESS