CS 50 Web Design

APCSP Module 2: Internet

Unit 3: JavaScript

LECTURE 7: DATA TYPES

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

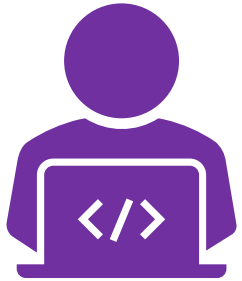
- Learn the basic JavaScript programming environments: Code Pen, Visual Studio Code, Google Chrome Console.
- Basic program structure, I/O
- Basic Data Types

Introduction

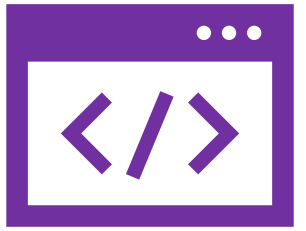
SECTION 1

What Is JavaScript?

- JavaScript was initially created to **“make web pages alive”.**
- The programs in this language are called **scripts**. They can be written right in a web page’s HTML and run automatically as the page loads.
- Scripts are provided and executed as **plain text**. They don’t need special preparation or compilation to run.
- In this aspect, JavaScript is very different from another language called Java.



When JavaScript was created, it initially had another name: “LiveScript”. But Java was very popular at that time, so it was decided that positioning a new language as a “younger brother” of Java would help.



But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.

Why is it called JavaScript?

What makes JavaScript unique?

- Full integration with HTML/CSS.
- Simple things are done simply.
- Supported by all major browsers and enabled by default.

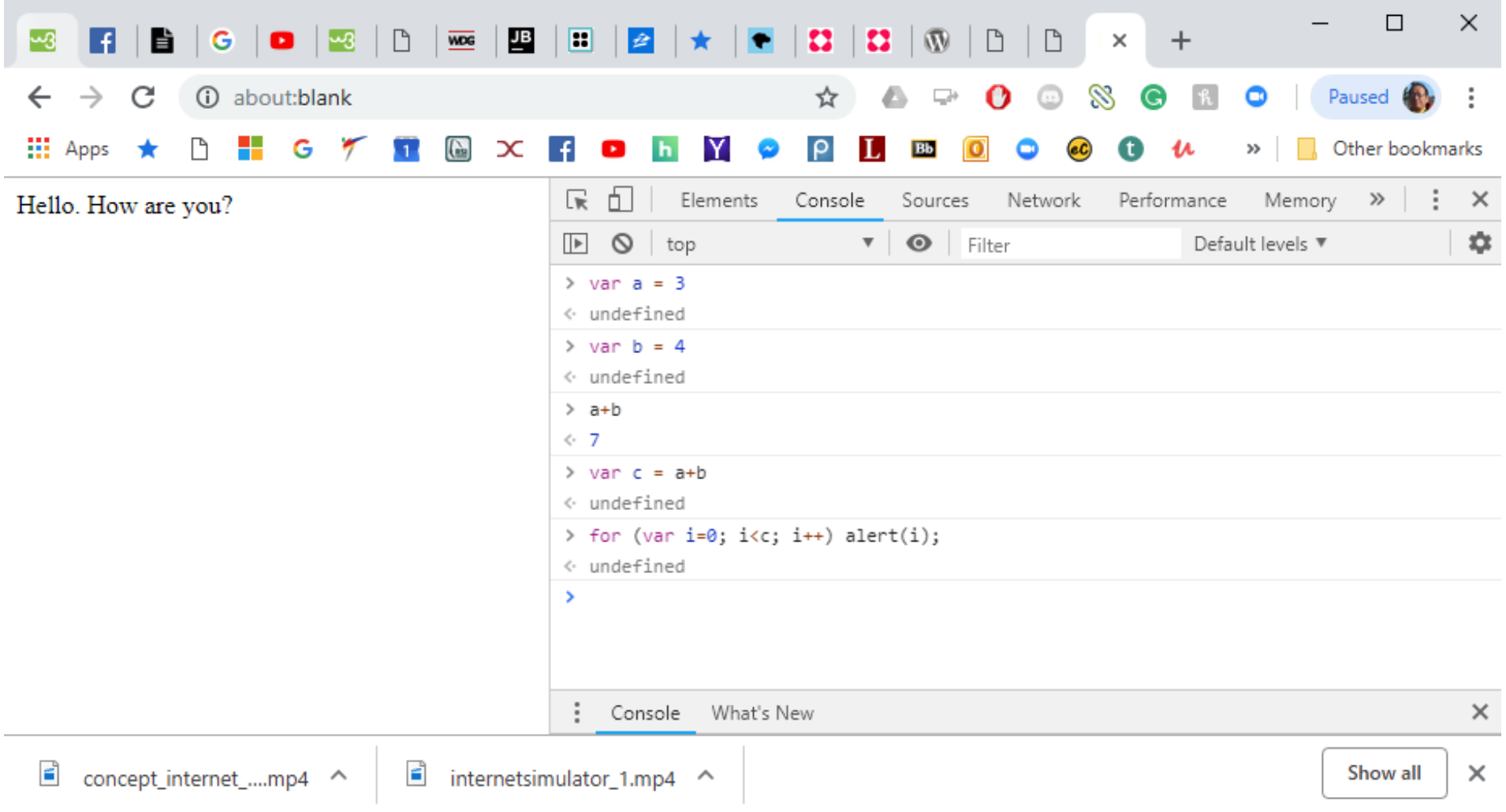
Google Chrome Console

SECTION 2

Writing Some JavaScript

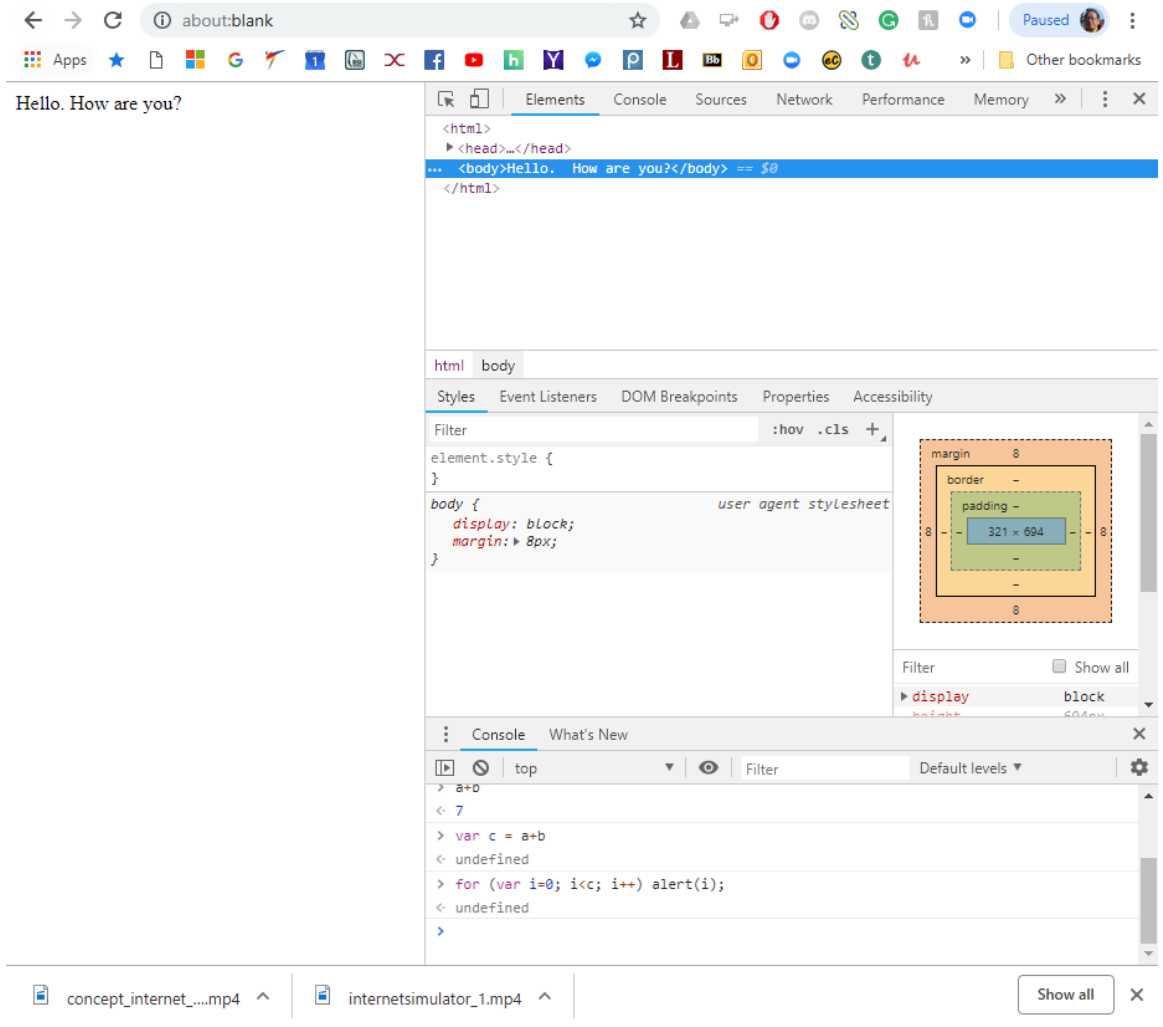
Console Mode

- Install Chrome on your computer (if it's not already installed), and then open it and type **about:blank** in the address bar. Now press ENTER and you'll see a blank page.
- We'll begin by coding in Chrome's JavaScript console, which is a secret way programmers can test out short JavaScript programs. On Microsoft Windows or Linux, hold down the CTRL and SHIFT keys and press J. On Mac OS, hold down the COMMAND and OPTION keys and press J.
- If you've done everything correctly, you should see a blank web page and, beneath that, a blinking cursor (|) next to a right angle bracket (>). That's where you'll write JavaScript!



Google Chrome JavaScript Console Mode

about:blank page as console mode and html editor



Google Chrome JavaScript Console Mode

about:blank page as console mode and html editor

Visual Studio Code Docs Updates Blog API Extensions FAQ Learn Search Docs Download

Version 1.70 is now available! Read about the new features and fixes from July.

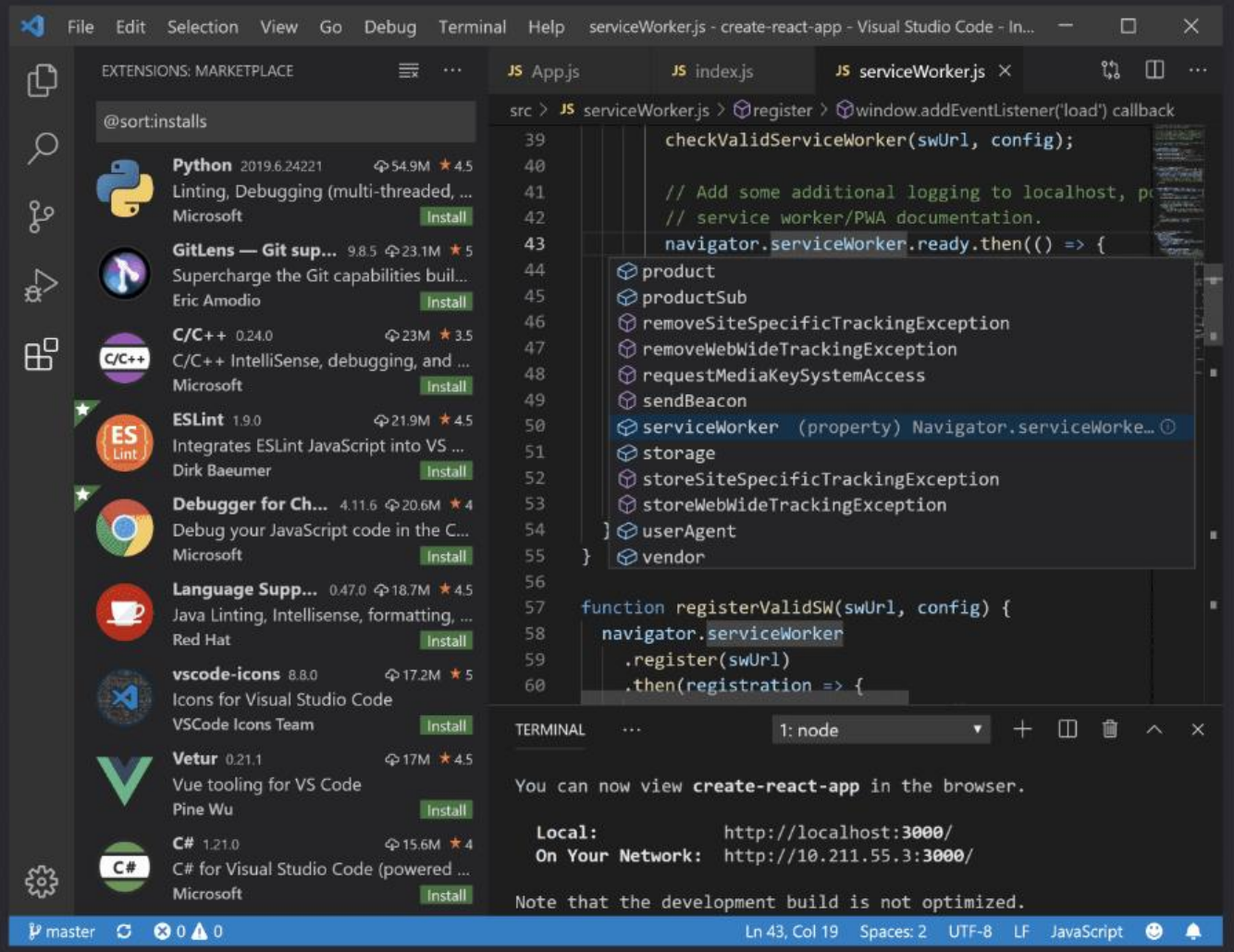
Code editing. Redefined.

Free. Built on open source. Runs everywhere.

Download for Windows
Stable Build

Web, Insiders edition, or other platforms

By using VS Code, you agree to its license and privacy statement.



CODEPEN

Search CodePen...

Sign Up

Log In

TRY OUR ONLINE EDITOR

Start Coding

Search Pens

Challenges

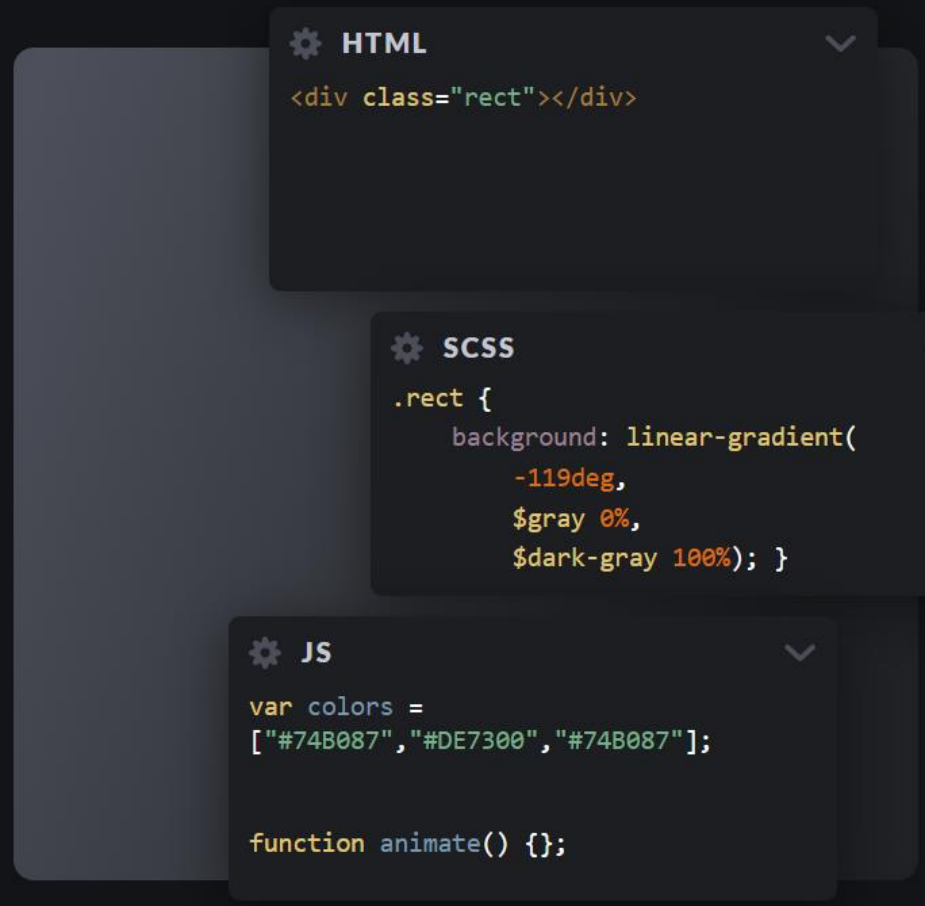
Spark

CodePen PRO

The best place to build, test, and discover front-end code.

CodePen is a **social development environment** for front-end designers and developers. Build and deploy a website, show off your work, build test cases to learn and debug, and find inspiration.

Sign Up for Free



Basic JavaScript Language

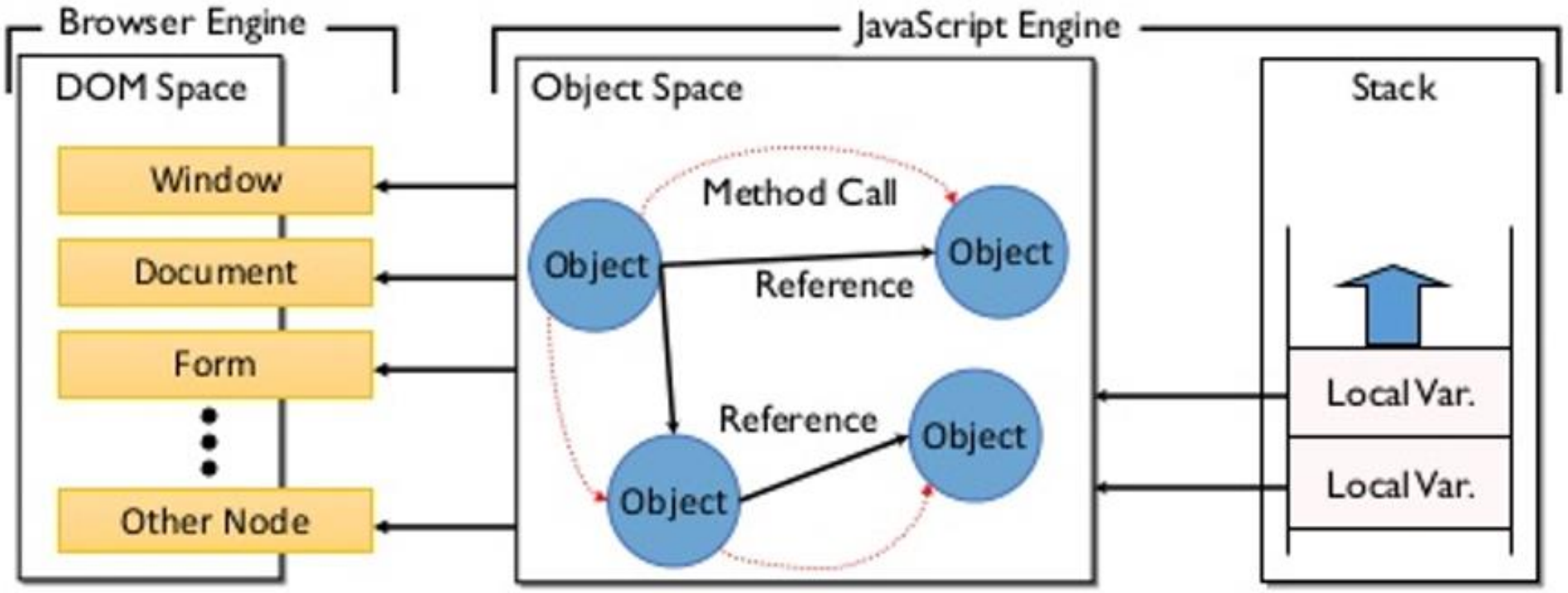
SECTION 3

JavaScript

- Interpreter Based Scripting Language
- Everything is an object.
- No data type (Type Inference)
- Single Threading
- Event-Driven Programming

JavaScript Memory Model

- DOM Space: the space where the Document Object Model representing the HTML's layered structure is represented.
- Object Space: the space where all JavaScript objects are located.
- Stack: short-term memory



What can JavaScript do?

- It is a full programming language
  - API (application programming interface) is specific to working with browsers
- Restrictions:
  - Security-based limitations
    - No networking
    - No access to local file system
  - Limited UI toolkit and graphics
    - (This is changing with HTML5)

What can JavaScript do?

- Benefits:
  - Close integration with the browser
    - Access the webpage and all elements within
    - Adjust or create HTML
    - Open and resize browser windows
    - Run animations, play sounds

The Structure of a JavaScript Program

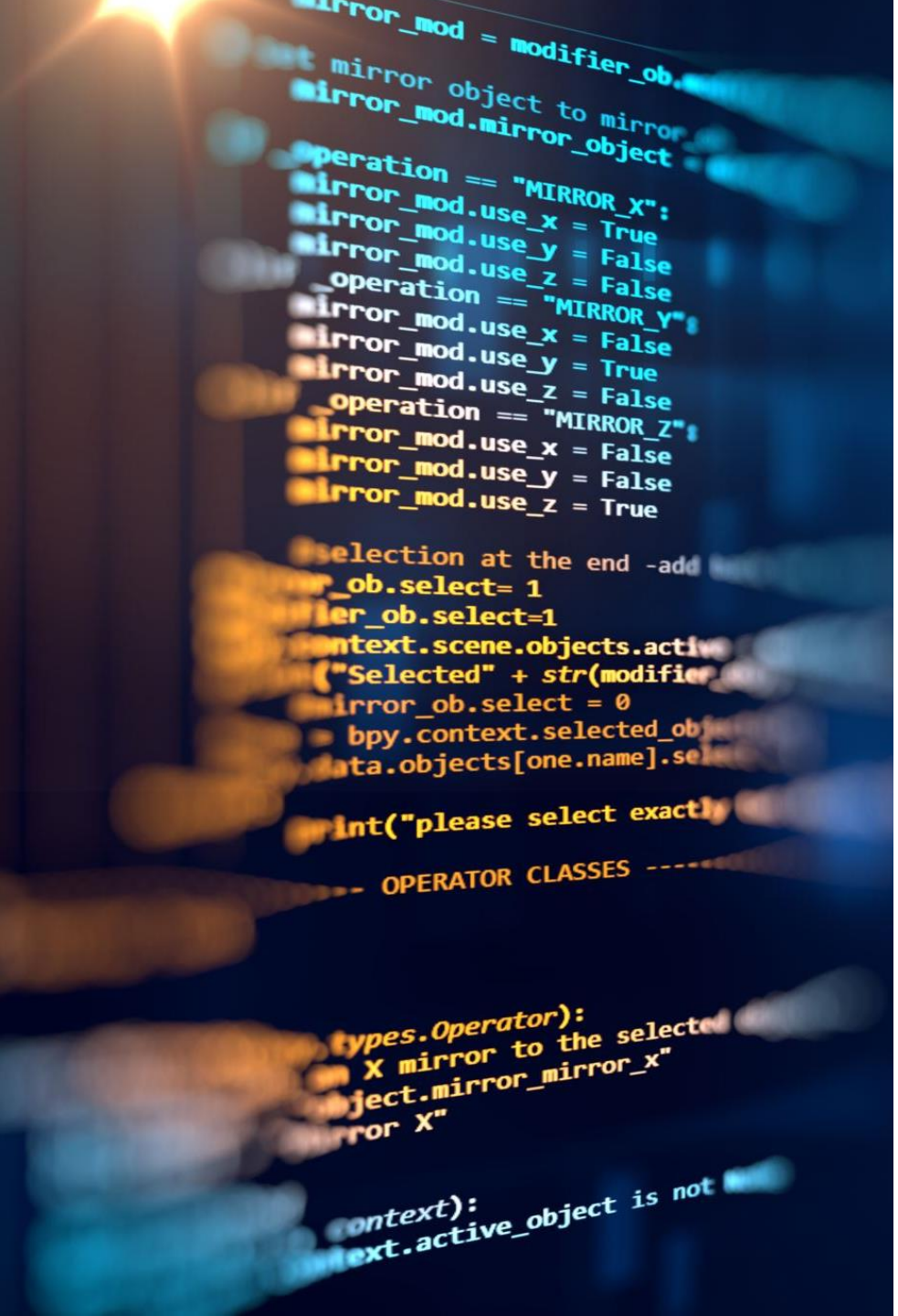
SECTION 4

Where do scripts go?

- In the HMTL page
- Like styles, can be **external**, **internal**, or **inline**
  - Use these for different situations

Body example
Demo Program: bodyexample.html

```
<html>
  <head>
  </head>
  <body>
   <script type="text/javascript">
   document.write("This message written by
   JavaScript");
   </script>
  </body>
</html>
```



Use of Document Write

- Use document as output terminal, you may debug JavaScript
- Until all JavaScript code is code, then, we may optimize the html/css to make the page more attractive.

Internal example

```
<html>
  <head>
   <script type="text/javascript">
    function message()
    {
      alert("This alert was called
    with the onload event");
    }
   </script>
  </head>
  <body onload="message()">
  </body>
</html>
```

Use of Alert

- Use alert as printf.
- Alert can be commented out when the development is finished.

External example

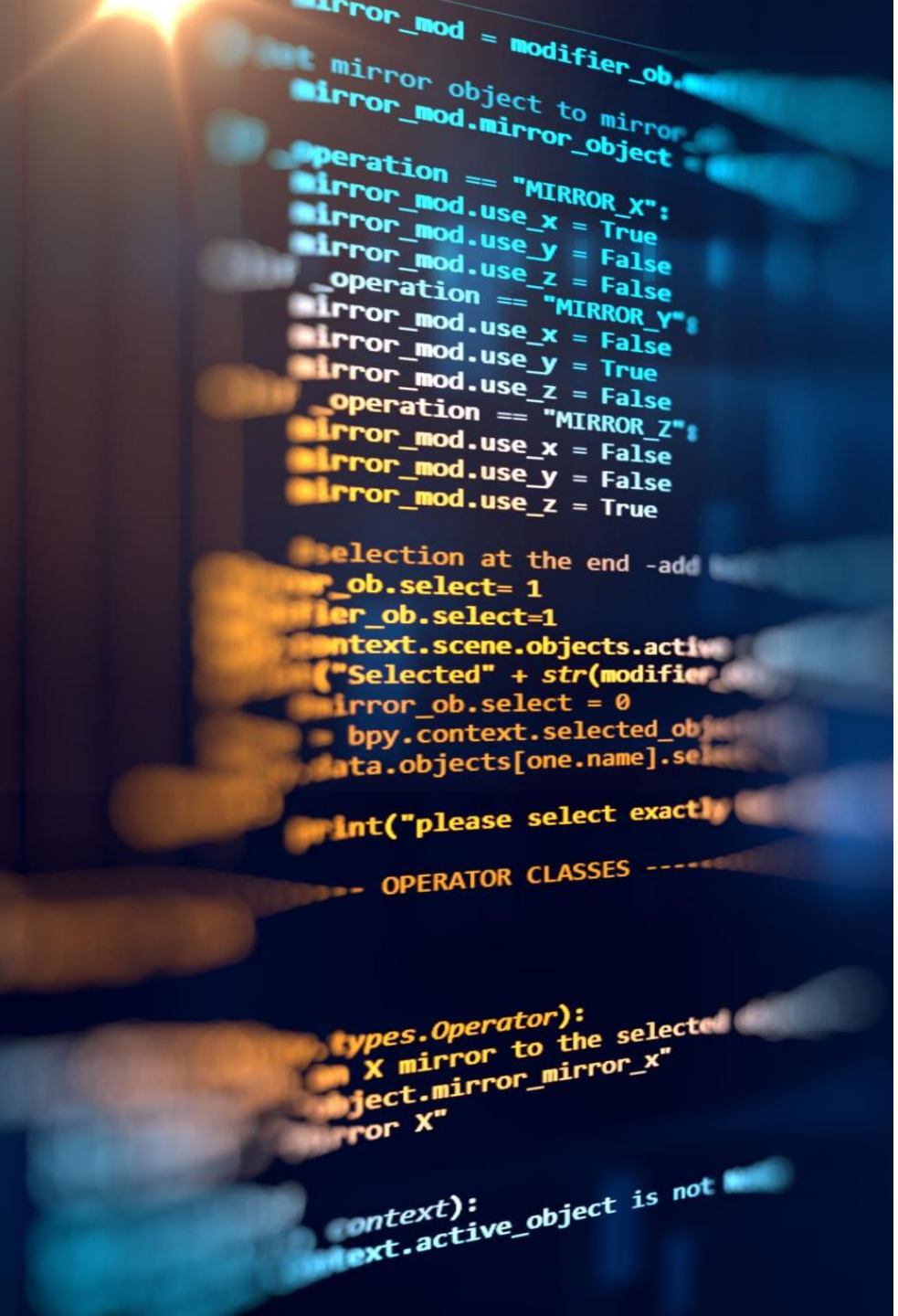
```
<html>
  <head>
    <script type="text/javascript"
            src="xyz.js">
    </script>
  </head>
  <body>
  </body>
</html>
```

Embedded JavaScript

- JavaScript Library
- JavaScript API (processing.js and other API)
- Put the library at proper location.

First Simple JavaScript Program

SECTION 5



Use JavaScript as other programming language:

Just For Computing Purpose

- Use internal <script></script> and put the script on top of body.
- Use only document.write() as printf function to deposit all output to the document.

Write to Page (Dynamic Document)

Demo Program: cats.html and cats2.html

```
<html>
<head>
<script>
// Draw as many cats as you want!
var drawCats = function (howManyTimes) {
for (var i = 0; i < howManyTimes; i++) {
 document.write(i + " =^.^= <br>");
}
};
drawCats(10);
</script>
</head>
<body>

</body>
</html>
```

```
<html>
<head>

</head>
<body>
<script>
// Draw as many cats as you want!
var drawCats = function (howManyTimes) {
for (var i = 0; i < howManyTimes; i++) {
 document.write(i + " =^.^= <br>");
}
};
drawCats(10);
</script>
</body>
</html>
```

Execution Results:

0 =^.^=
1 =^.^=
2 =^.^=
3 =^.^=
4 =^.^=
5 =^.^=
6 =^.^=
7 =^.^=
8 =^.^=
9 =^.^=

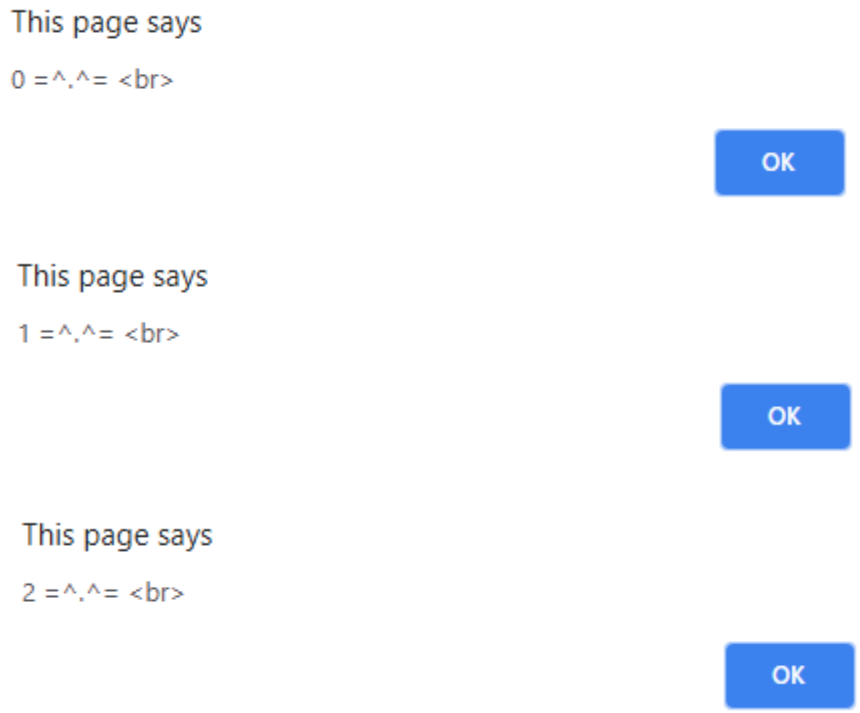
Trouble: Modify the page content

Alert

Demo Program: cats3.html

```
<html>
<head>

</head>
<body>
<script>
// Draw as many cats as you want!
var drawCats = function (howManyTimes) {
for (var i = 0; i < howManyTimes; i++) {
 alert(i + " =^.^= <br>");
}
};
drawCats(3);
</script>
</body>
</html>
```

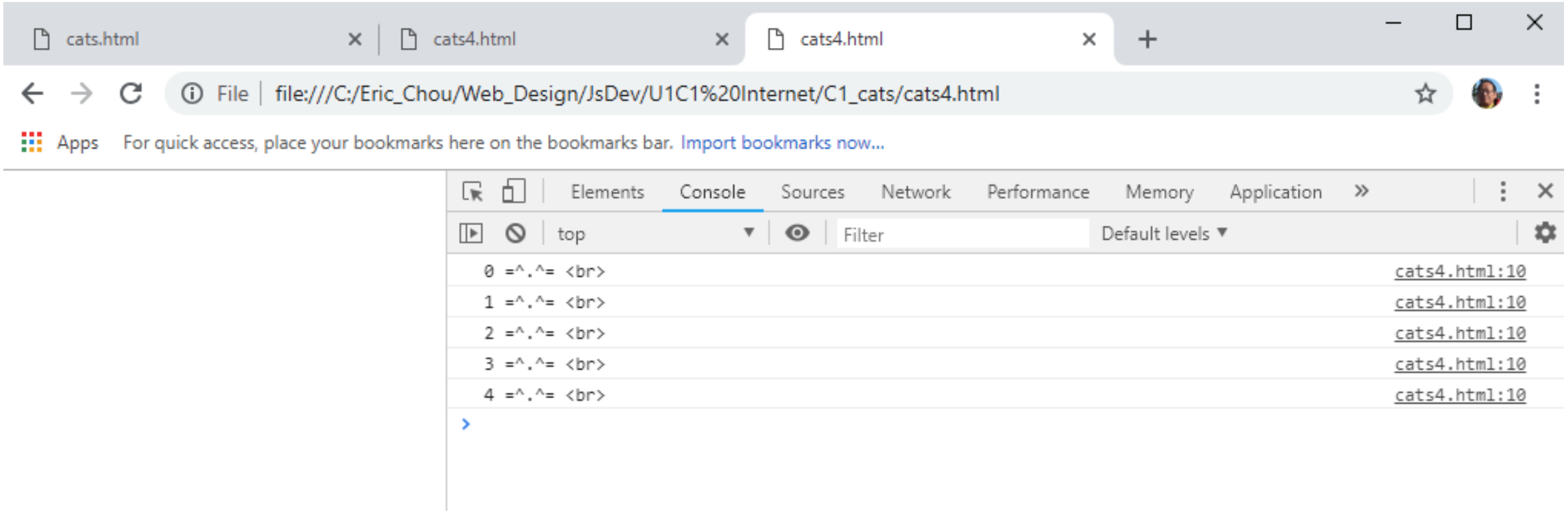


Trouble: Busy alerting

Console Log

Demo Program: cats4.html

```
<html>
<head>
</head>
<body>
<script>
// Draw as many cats as you want!
var drawCats = function (howManyTimes) {
for (var i = 0; i < howManyTimes; i++) {
 console.log(i + " =^.^= <br>");
}
};
drawCats(5);
</script>
</body>
</html>
```



Inspect Element -> Console

Best for Debugging

Syntax

- Our program includes lots of symbols, including parentheses **()**, semicolons **;**, curly brackets **{}**, plus signs **+**, and a few words that might seem mysterious at first (like var and console.log). These are all part of JavaScript's syntax — that is, JavaScript's rules for how to combine symbols and words to create working programs.
- When you're learning a new programming language, one of the trickiest parts is getting used to the rules for how to write different kinds of instructions to the computer. When you're first starting out, it's easy to forget when to include parentheses, or to mix up the order in which you need to include certain values. But as you practice, you'll start to get the hang of it.
- In this course, we'll go slow and steady, introducing new syntax little by little so that you can build increasingly powerful programs.

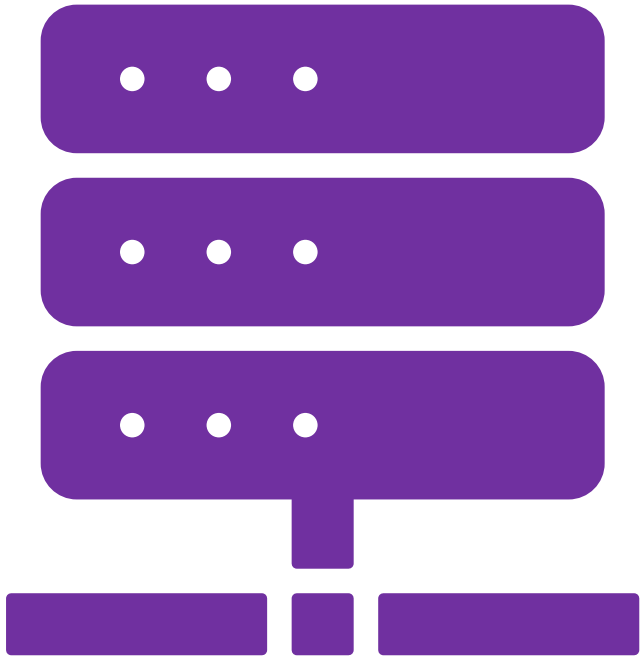
Comments

// comment out the rest of the line

/*comment out a code block */

Data in JavaScript

SECTION 6



Goals

- Programming is all about manipulating data, but what is data? Data is information that we store in our computer programs. For example, your name is a piece of data, and so is your age. The color of your hair, how many siblings you have, where you live, whether you're male or female — these things are all data.
- This chapter is to teach students: data in JavaScript – number, strings, and boolean values.
- The composite data types are: Arrays and Objects (Chapter 3, 4), objects stored in files (JSON, JSONP)

JavaScript Data Types

In JavaScript there are 5 different data types that can contain values:

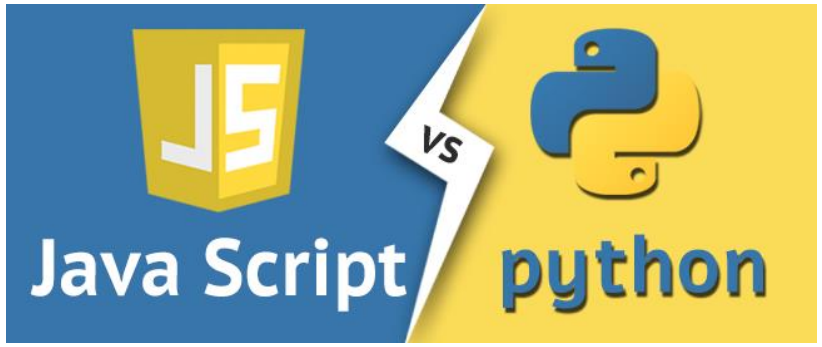
- string
- number
- boolean
- object
- function

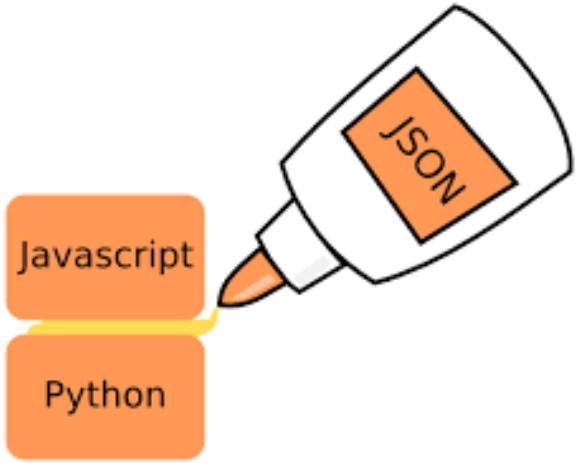
There are 3 types of objects:

- Object
- Date
- Array

And 2 data types that cannot contain values:

- null
- undefined





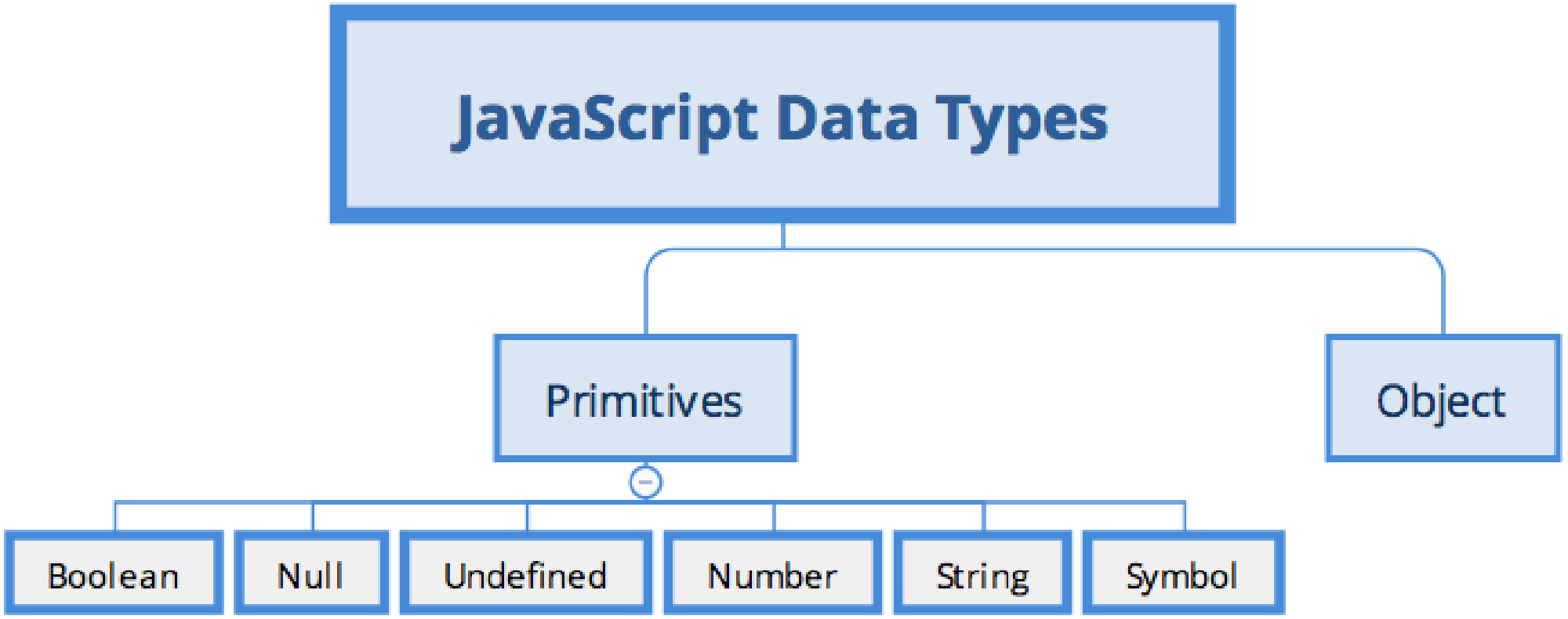
JSON	Python
object	dict
array	list
string	unicode
number (int)	int, long
number (real)	float
TRUE	TRUE
FALSE	FALSE
null	None

Topics

- Number and Operators
- Variables
- Strings
- Booleans
- **undefined** and **null**

JavaScript Data Types

SECTION 7


JavaScript Data Types
Primitives
Object
Boolean
Null
Undefined
Number
String
Symbol

Numbers and Operators

- JavaScript lets you perform basic mathematical operations like addition, subtraction, multiplication, and division. To make these calculations, we use the symbols +, -, *, and /, which are called operators.
- You can use the JavaScript console just like a calculator. We've already seen one example, adding together 3 and 4. Let's try something harder. What's 12,345 plus 56,789?

  12345 + 56789;

  69134

- That's not so easy to work out in your head, but JavaScript calculated it in no time.

JavaScript: Integers literals

Description

An **integer** must have at least one digit (0-9).

- No comma or blanks are allowed within an integer.
- It does not contain any fractional part.
- It can be either positive or negative if no sign precedes it is assumed to be positive.

In JavaScript, integers can be expressed in three different bases.

1. Decimal (base 10)

Decimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and there will be no leading zeros.

Example: 123, -20, 12345

JavaScript: Integers literals

2. Hexadecimal (base 16)

Hexadecimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and letters A, B, C, D, E, F or a, b, c, d, e, f. A leading 0x or 0X indicates the number is hexadecimal.

Example: 7b, -14, 3039

3. Octal (base 8)

Octal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7. A leading 0 indicates the number is octal.

Example: 0173, -024, 030071

JavaScript: Floating number literals

Description

- A floating number has the following parts.
- A decimal integer.
- A decimal point ('.').
- A fraction.
- An exponent.
- The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-").

JavaScript: Floating number literals

Example of some floating numbers :

8.2935

-14.72

12.4e3 [Equivalent to 12.4×10^3]

4E-3 [Equivalent to 4×10^{-3} => .004]

JavaScript: Boolean literals

The Boolean type has two literal values:

- **true**
- **false**

JavaScript: String literals

Description

- JavaScript has its own way to deal with string literals. A string literal is zero or more characters, either enclosed in single quotation (') marks or double **quotation** (") marks. You can also use + operator to join strings.

JavaScript: String literals

The following are the examples of string literals :

- string1 = "w3resource.com"
- string1 = 'w3resource.com'
- string1 = "1000"
- string1 = "google" + ".com"
- In addition to ordinary characters, you can include special characters in strings, as shown in the following table.
- string1 = "First line. \n Second line."

List of special characters used in JavaScript string:

Character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\'	Single quote
\"	Double quote
\\	Backslash character (\)
\XXX	The character with the Latin-1 encoding specified by up to three octal digits XXX between 0 and 377. For example, \100 is the octal sequence for the @ symbol.
\xXX	The character with the Latin-1 encoding specified by the two hexadecimal digits XX between 00 and FF. For example, \x40 is the hexadecimal sequence for the @ symbol.
\uXXXX	The Unicode character specified by the four hexadecimal digits XXXX. For example, \u0040 is the Unicode sequence for the @ symbol.

JavaScript is Case Sensitive

- JavaScript is case sensitive - therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions.
- document.write() IS NOT the same thing as
- Document.write() IS NOT the same thing as
- document.Write() IS NOT the same thing as
- Document.Write()

- *Which will actually work?*

```
1  <html>
2  <head>
3  <script>
4    // Draw as many cats as you want!
5    var newline = "<br>";
6    var helloWorld = "Hello World !"+newline;
7    document.write(helloWorld);
8    document.write(helloWorld);
9    </script>
10   </head>
11   <body>
12   </body>
13   </html>
```

Hello World !
Hello World !

String Output Demo Program:

helloWorld.html

JavaScript Operators

SECTION 8

Try These out on JavaScript Console about:blank

You can add multiple numbers with multiple plus signs:

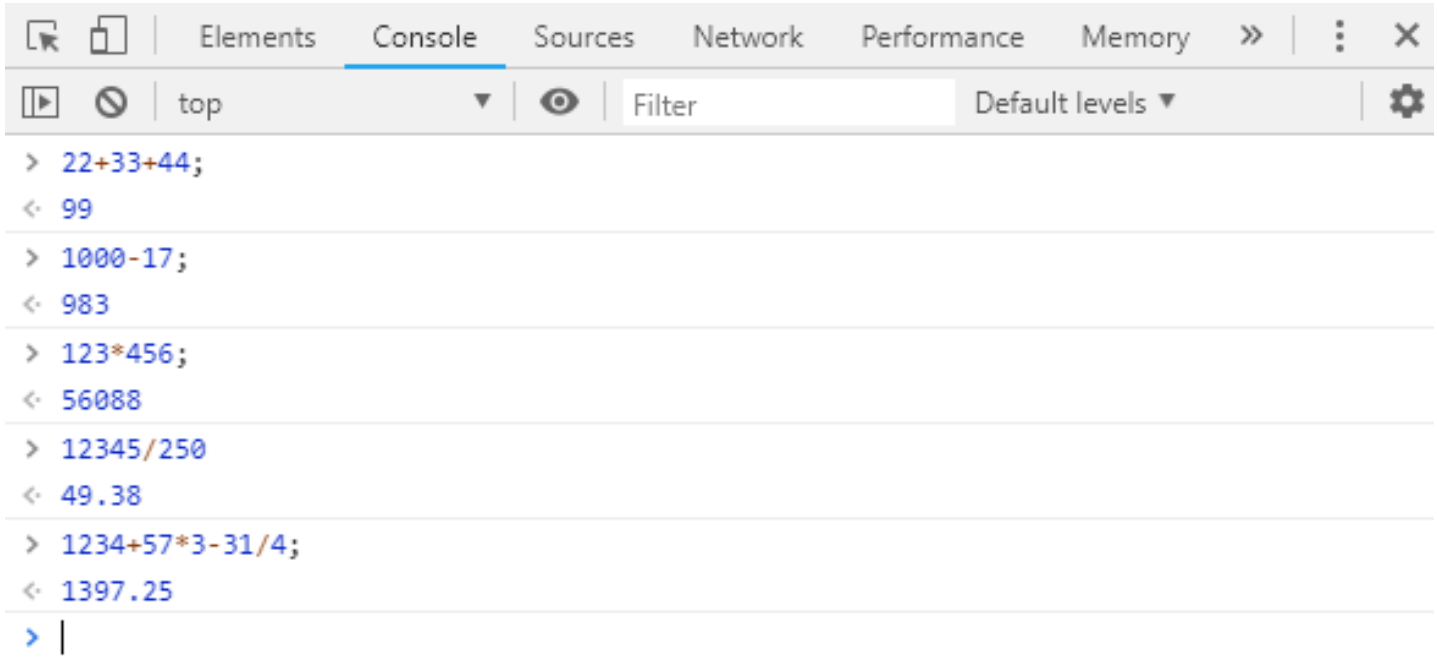
22 + 33 + 44;

99

JavaScript can also do subtraction . . .

1000 - 17;

983



Try These out on JavaScript Console

about:blank

and multiplication, using an asterisk . . .

```
123 * 456;
```

```
56088
```

and division, using a forward slash . . .

```
12345 / 250;
```

```
49.38
```

You can also combine these simple operations to make something more complex, like this:

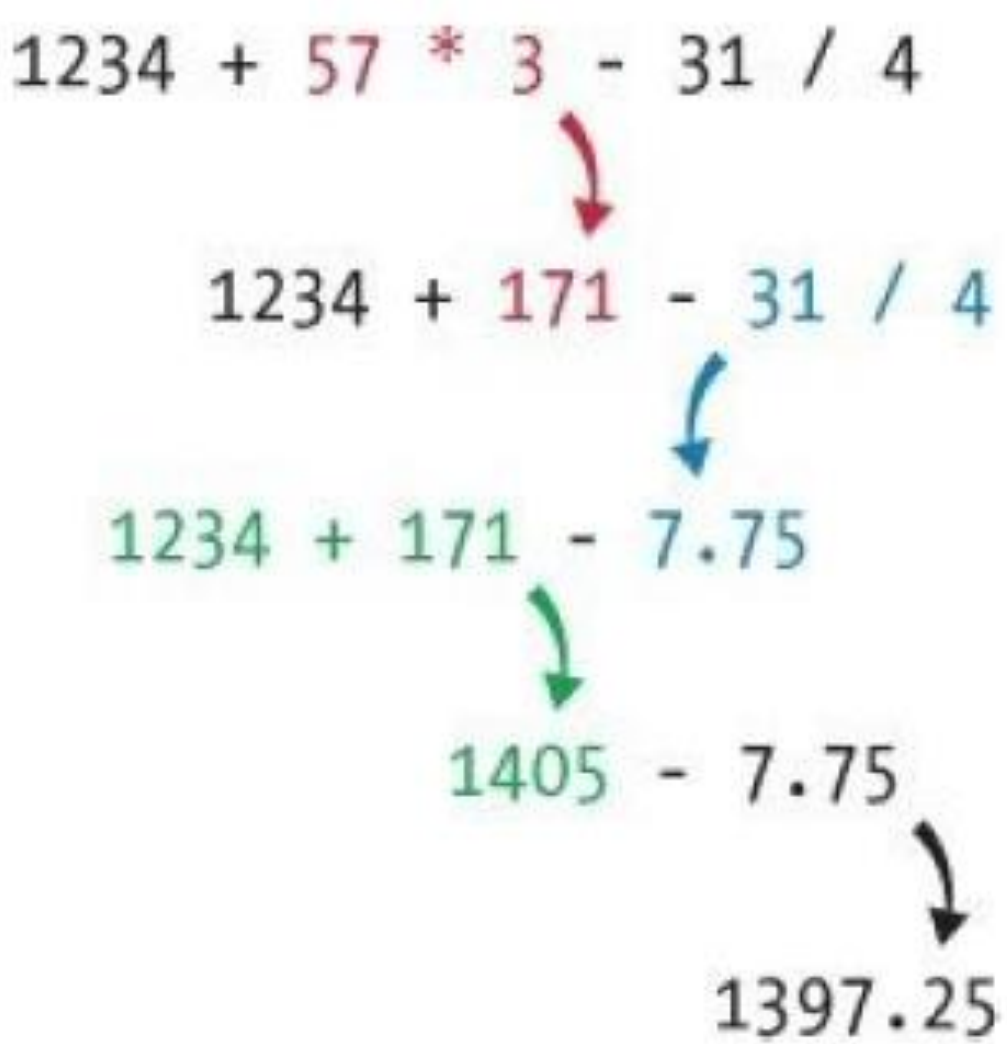
```
1234 + 57 * 3 - 31 / 4;
```

```
1397.25
```

Integer Division

```
var answer = Math.floor(x);
var x = parseInt(455/10);
```

Category	Operator	Name/Description	Example	Result
Arithmetic	+	Addition	3+2	5
	-	Subtraction	3-2	1
	*	Multiplication	3*2	6
	/	Division	10/5	2
	%	Modulus	10%5	0
	++	Increment and then return value	X=3; ++X	4
		Return value and then increment	X=3; X++	3
	--	Decrement and then return value	X=3; --X	2
		Return value and then decrement	X=3; X--	3
Logical	&&	Logical “and” evaluates to true when both operands are true	3>2 && 5>3	False
	||	Logical “or” evaluates to true when either operand is true	3>1 || 2>5	True
	!	Logical “not” evaluates to true if the operand is false	3!=2	True
Comparison	==	Equal	5==9	False
	!=	Not equal	6!=4	True
	<	Less than	3<2	False
	<=	Less than or equal	5<=2	False
	>	Greater than	4>3	True
	>=	Greater than or equal	4>=4	True
String	+	Concatenation(join two strings together)	“A”+”BC”	ABC



Priority of the Operators

- Figure on the right shows the order JavaScript would follow. First, JavaScript multiplies 57 * 3 and gets **171**. Then it divides 31 / 4 to get **7.75**. Next it adds 1234 + 171 to get **1405**. Finally it subtracts 1405 - 7.75 to get 1397.25, which is the final result.
- What if you wanted to do the addition and the subtraction first, before doing the multiplication and division? For example, say you have 1 brother and 3 sisters and 8 candies, and you want to split the candies equally among your 4 siblings? (You've already taken your share!) You would have to divide 8 by your number of siblings.

JavaScript operation	Arithmetic operator	Algebraic expression	JavaScript expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $x \div y$	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Basic Arithmetic Operators

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	1) If the parentheses nested, expression in innermost pair evaluated first. If several pairs of parentheses "on the same level" (not nested), evaluated left to right.
*, / or %	Multiplication, Division, Modulus	2) If more then one, then evaluated left to right.
+ or -	Addition, Subtraction	3) If more than one, then evaluated left to right.

Operators' Precedence

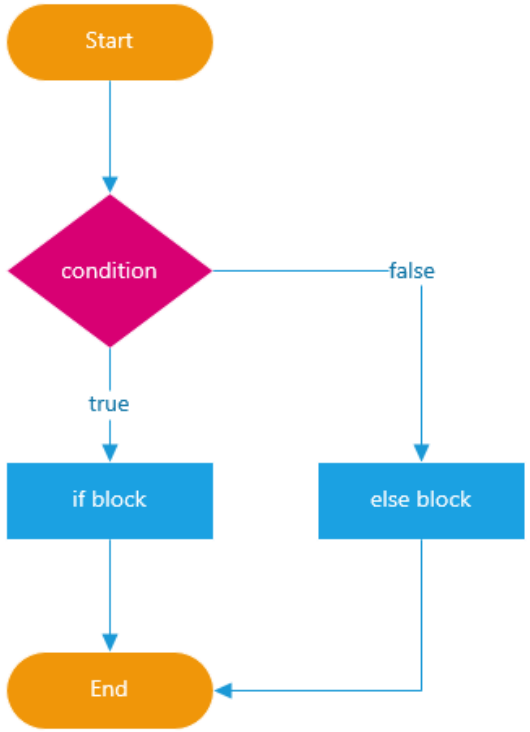
Assignment operator	Initial variable value	Sample expression	Explanation	Assigns
`+=`	`c = 3`	`c += 7`	c = c + 7	10 to c
`-=`	`d = 5`	`d -= 4`	d = d - 4	1 to d
`*=`	`e = 4`	`e *= 5`	e = e * 5	20 to e
`/=`	`f = 6`	`f /= 3`	f = f / 3	2 to f
`%=`	`g = 12`	`g %= 9`	g = g % 9	3 to g

Augmented Assignments

Operator	Called	Sample expression	Explanation
++	preincrement	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	postincrement	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	predecrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	postdecrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Increment and Decrement

```
if (expression 1){
   Statement(s) to be executed if expression 1 is true
}else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}else{
   Statement(s) to be executed if no expression is true
}
```



JavaScript if-else Statements
Preview

Relational Operators

Standard algebraic equality operator or relational operator	JavaScript equality or relational operator	Sample JavaScript condition	Meaning of JavaScript condition
Equality Operators			
=	==	x == y	x is equal to y
Not =	!=	x != y	x is not equal to y
Relational Operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
>=	>=	x >= y	x is greater than or equal to y
<=	<=	x <= y	x is less than or equal to y

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that `x = 6` and `y = 3`, the table below explains the logical operators:

Operator	Description	Example	Try it
&&	and	(x < 10 && y > 1) is true	
||	or	(x == 5 || y == 5) is false	
!	not	!(x == y) is true	

Conditional statements

Implement alternative behaviours based on conditions

```
if (temperature < 20) {
 console.log("It is cold");
} else if (temperature >= 20 && temperature <
            29) {
 console.log("It is warm");
} else {
 console.log("it is hot");
}
```

Comparison operators

Expressions based on comparison operators evaluate to a boolean:

- Equal:

```
3.5 == 2 // (evaluates to false)
```

- Not equal:

```
"aString" != "anotherString" // (true)
```

- Greater than / (or equal):

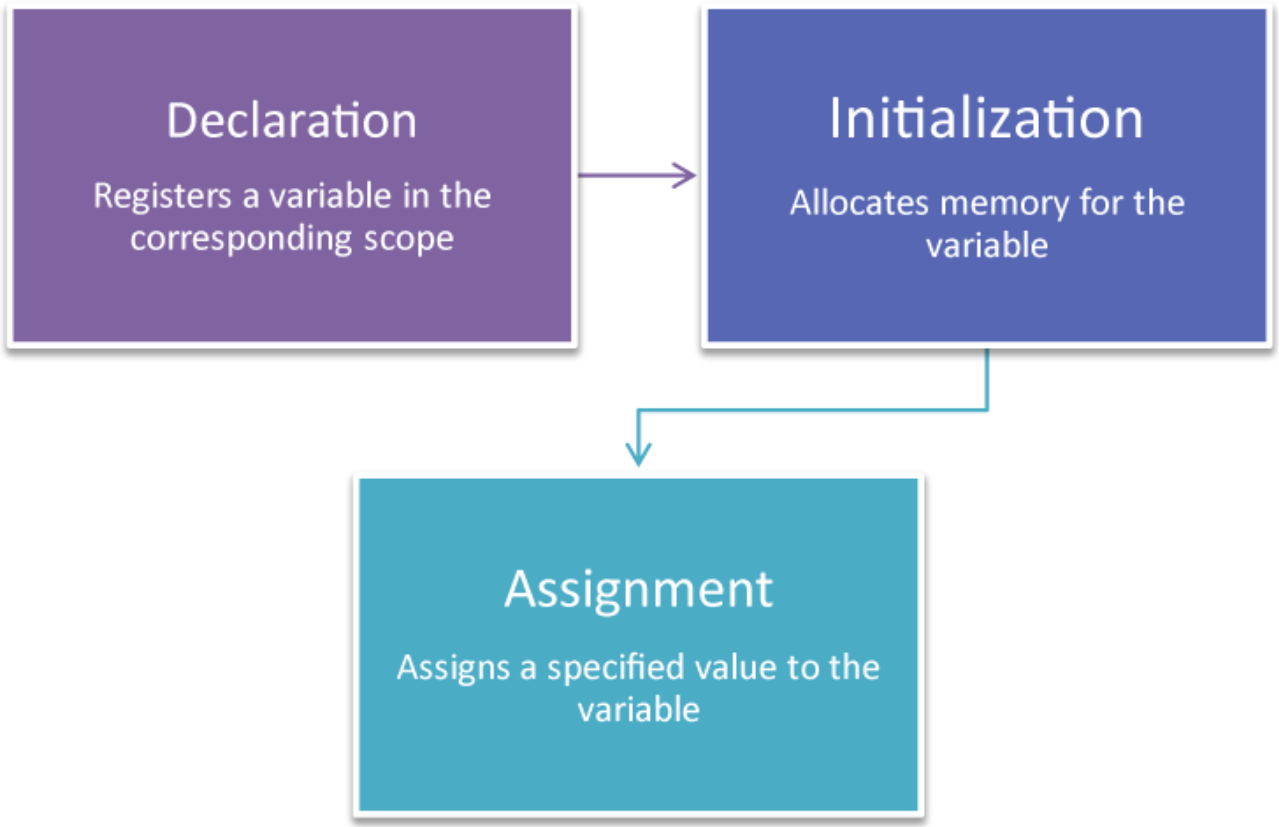
```
6 > 6 // (false)
6 >= 6 // (true)
```

- Less than / (or equal):

```
5 < 3 // (false)
5 <= 3 // (false)
```

JavaScript Variables

SECTION 9



Three Steps to Construct a Variable

JavaScript Variables

- JavaScript variables are containers for storing data values.
- In this example, x, y, and z, are variables:

Example

```
var x = 5;
var y = 6;
var z = x + y;
```

Much Like Algebra

- In this example, price1, price2, and total, are variables:

Example

```
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
```

- In programming, just like in algebra, we use variables (like price1) to hold values.
- In programming, just like in algebra, we use variables in expressions (total = price1 + price2).
- From the example above, you can calculate the total to be 11.
- JavaScript variables are containers for storing data values.

JavaScript Identifiers

- All JavaScript variables must be identified with unique names.
- These unique names are called **identifiers**.
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

JavaScript Identifiers

- The general rules for constructing names for variables (unique identifiers) are:
  - Names can contain **letters**, **digits**, **underscores**, and **dollar signs**.
  - Names must begin with a **letter**
  - Names can also begin with **$** and _ (but we will not use it in this tutorial)
  - Names are case sensitive (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names
- JavaScript identifiers are **case-sensitive**.

JavaScript Reserved Words

Reserved Keywords			
break	else	new	var
case	finally	return	void
catch	for	switch	while
continue	function	this	with
default	if	throw	
delete	in	try	
do	instanceof	typeof	

Declaring Variables

SECTION 10

Declaring (Creating) JavaScript Variables

- Creating a variable in JavaScript is called "declaring" a variable.
- You declare a JavaScript variable with the var keyword:

```
var carName;
```

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

```
carName = "Volvo";
```

Declaring (Creating) JavaScript Variables

- You can also assign a value to the variable when you declare it:

```
var carName = "Volvo";
```

Declaring (Creating) JavaScript Variables

- In the example below, we create a variable called **carName** and assign the value "Volvo" to it.
- Then we "output" the value inside an HTML paragraph with id="demo":

Example

```
<p id="demo"></p>

<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

Multiple Assignment

- You can declare many variables in one statement.
- Start the statement with var and separate the variables by comma:

```
var person = "John Doe", carName = "Volvo", price = 200;
```

- A declaration can span multiple lines:

```
var person = "John Doe",
carName = "Volvo",
price = 200;
```

Value = undefined

- In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.
- A variable declared without a value will have the value undefined.
- The variable **carName** will have the value undefined after the execution of this statement:

Example

```
var carName;
```

Value = undefined

'**Undefined**' is a global variable that JavaScript creates at run time. This global variable is assigned to an object in one of the following cases –

1. Declared but not defined –
2. JavaScript assigns 'undefined' to any object that has been declared but not initialized or defined. In other words, in a case where no value has been explicitly assigned to the variable, JavaScript calls it 'undefined'.
3. Array index or object property that does not exist.
4. A function parameter that has not been supplied.
5. The return value of functions that have to but don't return a value.

```
1  var item;
2  console.log(item)
```

Upon execution, the code will print undefined.

Value = null

- **null** is a keyword in JavaScript that signifies ‘no value’ or nonexistence of any value. If you wish to shred a variable off its assigned value, you can simply assign ‘null’ to it. Besides this, like any other object, it is never implicitly assigned to a variable by JavaScript.
- An example of explicit **null** assignment is –

```
1  var some_item= null;
2  console.log(some_item)
```

- Upon execution, the code will print **null.**

Difference between null and undefined

Difference in Type –

‘Null’ is an object with a valid value, no properties, is non-mutable, and only a single instance of the same exists in the system at all times. In case you wish to verify the object nature of ‘null’, you can use the ‘typeof’ operator. The use of the same will give the output as ‘object’. However, if you use the ‘typeof’ operator on an object that belongs to one of the points mentioned in the ‘undefined’ list, you will get the object type as ‘undefined’.

Difference between null and undefined

- A major difference between ‘null’ and ‘undefined’ is in the way they are converted to primitive types. When you perform arithmetic conversion on ‘null’, the value determined is 0. This conversion can be verified using the following code snippet.

```
var v1= 5+ null;
console.log(v1)
```

- Upon execution, this output of this code will print 5.
- However, ‘undefined’ does not carry out any such conversion. If you try to add ‘undefined’ to a numeral, you will get NaN or Not-a-Number. The following code snippet illustrates this facet of ‘undefined’.

```
var v2= 5+ undefined;
console.log(v2)
```

- Upon execution, the code will print **NaN**.

Data Type Conversion

SECTION 11

Type Conversions

- Most of the time, operators and functions automatically convert the values given to them to the right type. This is called “type conversion”.
- For example, **alert** automatically converts any value to a **string** to show it. Mathematical operations convert values to numbers.
- There are also cases when we need to explicitly convert a value to the expected type.

To String

- String conversion happens when we need the string form of a value.
- For example, alert(value) does it to show the value.
- We can also call the String(value) function to convert a value to a string:

```
1  let value = true;
2  alert(typeof value); // boolean
3
4  value = String(value); // now value is a string "true"
5  alert(typeof value); // string
```

- String conversion is mostly obvious. A false becomes "false", null becomes "null", etc.

To Number

- Numeric conversion happens in mathematical functions and expressions automatically.
- For example, when division / is applied to non-numbers:

```
alert( "6" / "2" ); // 3, strings are converted to numbers
```

- We can use the Number(value) function to explicitly convert a value to a number:

```
let str = "123";
alert(typeof str); // string

let num = Number(str); // becomes a number 123

alert(typeof num); // number
```

To Number

- Explicit conversion is usually required when we read a value from a string-based source like a text form but expect a number to be entered.
- If the string is not a valid number, the result of such a conversion is NaN. For instance:

```
1  let age = Number("an arbitrary string instead of a number");
2
3  alert(age); // NaN, conversion failed
```

Numeric Conversion Rules

Value	Becomes...
`undefined`	`NaN`
`null`	`0`
`true and false`	`1` and `0`
`string`	Whitespaces from the start and end are removed. If the remaining string is empty, the result is `0`. Otherwise, the number is "read" from the string. An error gives `NaN`.

Examples:

```
alert( Number("   123   ") ); // 123
alert( Number("123z") );      // NaN (error reading a number at "z")
alert( Number(true) );        // 1
alert( Number(false) );       // 0
```

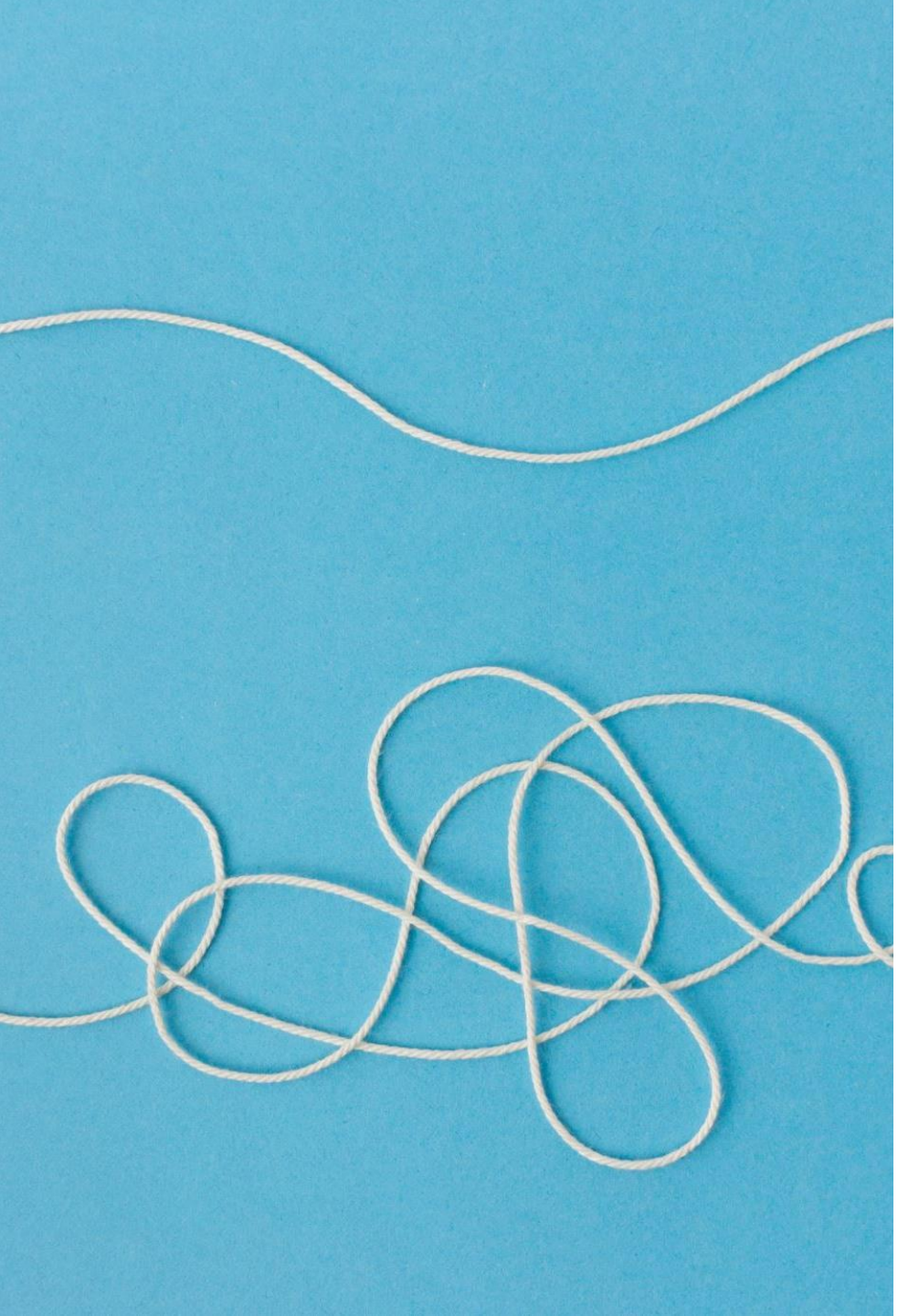
Testing in Chrome 56.0.2924 / Mac OS X 10.12.3		
Test		**Ops/sec**
plus	`var x = +"1000"`	775,574,732 ±1.94% fastest
Number	`var x = Number("1000")`	94,470,129 ±2.02% 88% slower
Math.floor	`var x = Math.floor("1000");`	36,427,122 ±2.03% 95% slower
parseInt	`var x = parseInt("1000", 10);`	135,009,099 ±2.29% 83% slower
mul	`var x = "1000"*1`	777,771,315 ±1.77% fastest
Math.round	`var x = Math.round("1000");`	34,869,214 ±3.74% 96% slower

Addition ‘+’ concatenates strings

- Almost all mathematical operations convert values to numbers. A notable exception is addition +. If one of the added values is a string, the other one is also converted to a string.
- Then, it concatenates (joins) them:

```
1  alert( 1 + '2' ); // '12' (string to the right)
2  alert( '1' + 2 ); // '12' (string to the left)
```

- This only happens when at least one of the arguments is a string. Otherwise, values are converted to numbers.



To Boolean

- Boolean conversion is the simplest one.
- It happens in logical operations (later we'll meet condition tests and other similar things) but can also be performed explicitly with a call to Boolean(value).
- The conversion rule:
- Values that are intuitively "empty", like **0**, an **empty string**, **null**, **undefined**, and **NaN**, become **false**.
- Other values become **true**.

For instance:

```
1  alert( Boolean(1) ); // true
2  alert( Boolean(0) ); // false
3
4  alert( Boolean("hello") ); // true
5  alert( Boolean("") ); // false
```

Please note: the string with zero `"0"` is `true`

Some languages (namely PHP) treat `"0"` as `false`. But in JavaScript, a non-empty string is always `true`.

```
1  alert( Boolean("0") ); // true
2  alert( Boolean(" ") ); // spaces, also true (any non-empty string is true)
```

Monthly Interests

Demo Program: monthinterests.html

```
<html>
<head>
<script>
  var amount = "1000";
  var interest = "0.05";      // Monthly interest rate
  amount = parseInt(amount); // amount is only a reference to number object
  interest = parseFloat(interest);
  document.writeln("Principal Amount: "+amount);
  document.writeln("Monthly Interest Rate: "+interest);
  document.writeln("Monthly Interest: "+ (amount*interest));
  </script>
  </head>
  <body>
  </body>
  </html>
```

Principal Amount: 1000 Monthly Interest Rate: 0.05 Monthly Interest: 50

JavaScript I/O

SECTION 12

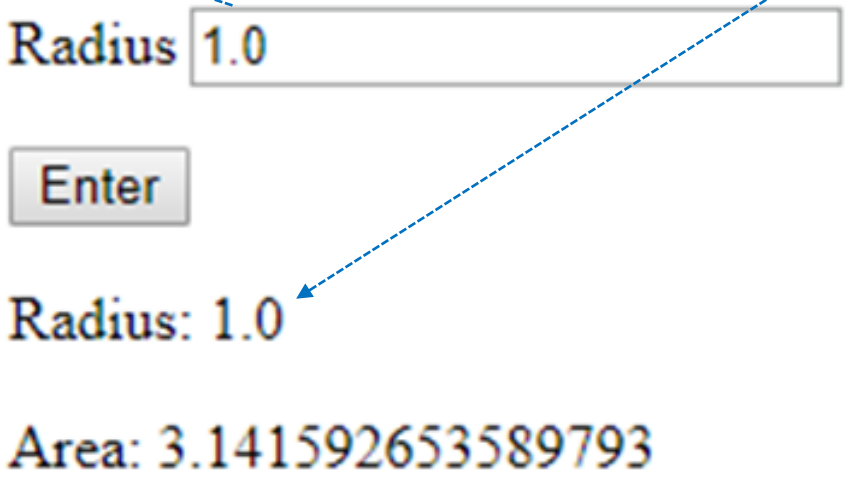
Basic I/O Commands in JavaScript

- Input Box and document update.
- Window prompt and document update.
- Window prompt and alert.

Basic I/O Commands in JavaScript

Index Box and document update

- Input Box with Button (onclick to activate main function in JavaScript)
- var x = document.getElementById(“Input_Box”).value;
- document.getElementById(“Output_Area”).innerHtml(print_msg);




```
1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <!-- Input Section (Input Box) -->
6  <p>Radius <input type="text" id="myText" value="0.0"></p>
7  <button onclick="main()">Enter</button>
8
9  <!-- Output Section (Paragraph) -->
10 <p id="Line1"><br></p>
11 <p id="Line2"><br></p>
12
13 <!-- Processing Section -->
14 <script>
15 function main() {  // single main function
16   var x = document.getElementById("myText").value;
17   document.getElementById("Line1").innerHTML = "Radius: "+x;
18   var radius = Number(x);
19   var area = Math.PI * radius * radius;
20   document.getElementById("Line2").innerHTML = "Area: "+area;
21 }
22 </script>
23
24 </body>
25 </html>
```

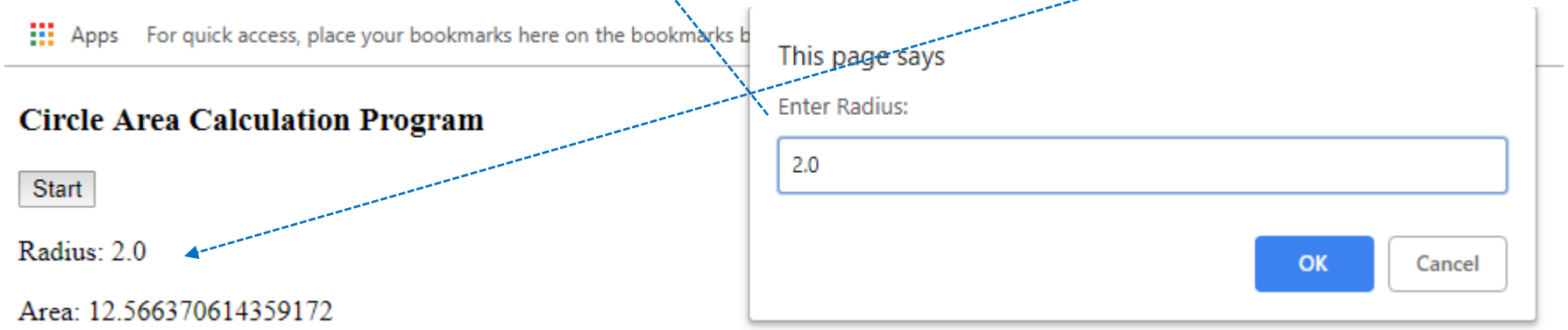
Circle

Demo Program: circle.html

Basic I/O Commands in JavaScript

Prompt Box and document update

```
<h3>Circle Area Calculation Program </h3>
<button onclick="main()">Start</button>
var x = prompt("Enter Radius: "); // in javascript section
document.getElementById("Output_Area").innerHtml(print_msg);
```



```html
<!DOCTYPE html>
<html>
<body>

<!-- Input Section (Input Box) -->
<h3>Circle Area Calculation Program </h3>
<button onclick="main()">Start</button>

<!-- Output Section (Paragraph) -->
<p id="Line1"><br></p>
<p id="Line2"><br></p>

<!-- Processing Section -->
<script>
function main() {  // single main function
  var x = prompt("Enter Radius: ");
    if (x != null){
        document.getElementById("Line1").innerHTML = "Radius: "+x;
        var radius = Number(x);
        var area = Math.PI * radius * radius;
        document.getElementById("Line2").innerHTML = "Area: "+area;
    }
}
</script>

</body>
</html>
```

Circle

Demo Program: circle2.html

Basic I/O Commands in JavaScript

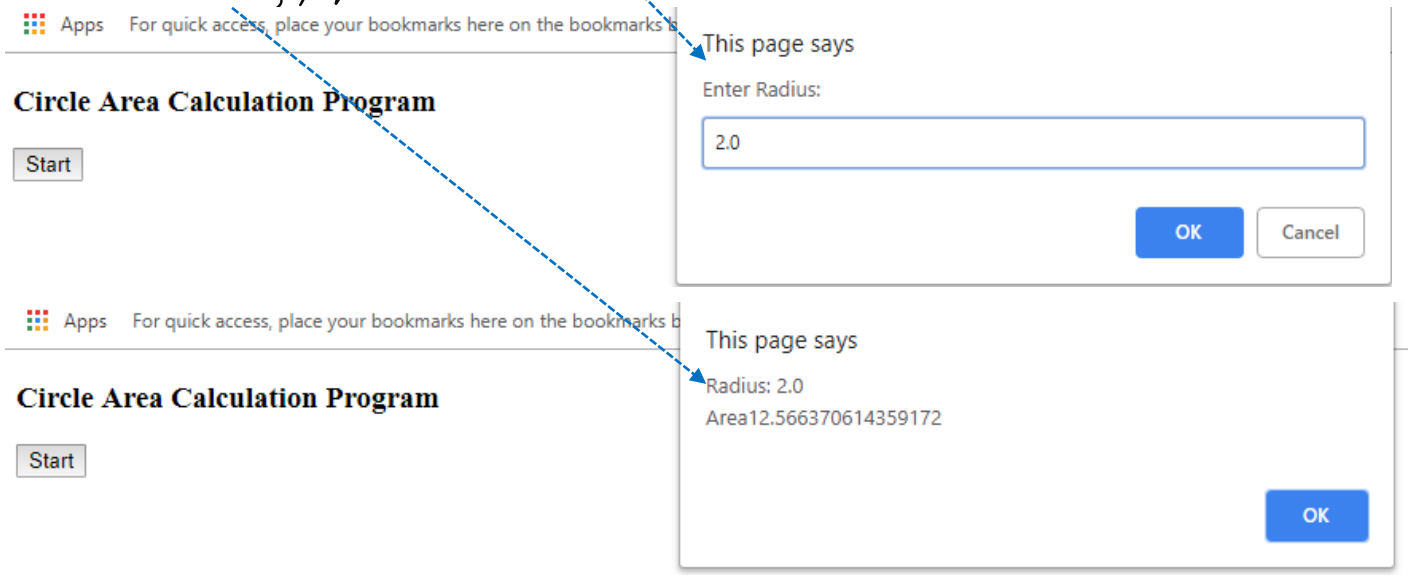
Prompt Box and Alert Box

```
<h3>Circle Area Calculation Program </h3>

<button onclick="main()">Start</button>

var x = prompt("Enter Radius: "); // in javascript section

alert(print msg);
```



```html
<!DOCTYPE html>
<html>
<body>

<!-- Input Section (Input Box) -->
<h3>Circle Area Calculation Program </h3>
<button onclick="main()">Start</button>

<!-- Processing Section -->
<script>
function main() {  // single main function
  var x = prompt("Enter Radius: ");
  var print_msg = "";
    if (x != null){
        print_msg += "Radius: "+x+"\n";
        var radius = Number(x);
        var area = Math.PI * radius * radius;
        print_msg += "Area"+area+"\n";
        alert(print_msg);
    }
}
</script>

</body>
</html>
```

Circle

Demo Program:
circle3.html

JavaScript Output

SECTION 13

JavaScript Output

- Writing into an HTML element, using innerHTML.
  - The element can be any paragraph <p> or header<h>
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
  - Pop-up Window
- Writing into the browser console, using console.log().
  - Not updating document. Good for debugging

Using innerHTML

- To access an HTML element, JavaScript can use the document.getElementById(id) method.
- The id attribute defines the HTML element. The innerHTML property defines the HTML content:

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My First Paragraph.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Using document.write()

- For testing purposes, it is convenient to use document.write():

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<p>Never call document.write after the document has finished loading.
It will overwrite the whole document.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

The document.write() method should only be used for testing.

Using window.alert()

- You can use an alert box to display data:

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

Using console.log()

- For debugging purposes, you can use the console.log() method to display data.
- You will learn more about debugging in a later chapter.

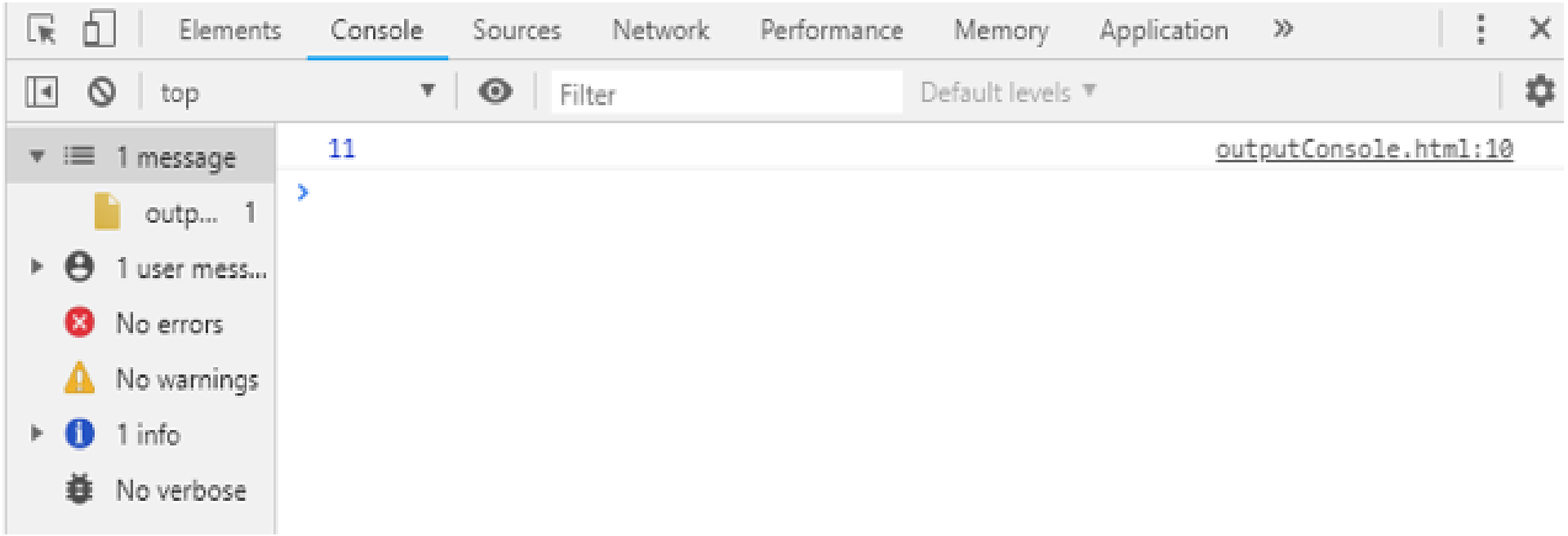
```
<!DOCTYPE html>
<html>
<body>

<h2>Activate debugging with F12</h2>

<p>Select "Console" in the debugger menu. Then click Run again.</p>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```



Activate debugging with F12

Select "Console" in the debugger menu. Then click Run again.

On-load

SECTION 14

```
1  <html>
2  <head>
3  <script>
4  function main(){
5    document.writeln("Hello World!");
6  }
7  </script>
8  </head>
9  <body onload="main()"></body>
10 </html>
```

Hello World!

Demo Program:

onload.html

JavaScript Development Environment [Optional]

SECTION 15

Built-In Libraries

ENV-1

JavaScript Built-In Libraries

Mozilla Developer's Network (MDN):

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

Wikipedia (List of JavaScript Libraries):

- https://en.wikipedia.org/wiki/List_of_JavaScript_libraries

Built-in libraries

Math.PI is a constant (a variable that never changes value) from the built-in Math library. Some additional Math functions:

- `Math.round(4.7)`
- `Math.sqrt(9)`
- `Math.max(1,5)`
- `Math.min(6,7)`
- `Math.floor(5.6)`
- `Math.random()`

`console.log()` is a built-in function (in Chrome) that prints values to the console

Math Object Methods

Method	Description	Examples
abs(x)	ABSOLUTE VALUE OF x	abs(7.2) is 7.2 abs(0.0) is 0.0 abs(-5.6) is 5.6
ceil(x)	ROUNDS x TO THE SMALLEST INTEGER NOT LESS THAN x	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
cos(x)	TRIGONOMETRIC COSINE OF x (x IN RADIANS)	cos(0.0) is 1.0
exp(x)	EXPONENTIAL METHOD e^x	exp(1.0) is 2.71828 exp(2.0) is 7.38906
floor(x)	ROUNDS x TO THE LARGEST INTEGER NOT GREATER THAN x	floor(9.2) is 9.0 floor(-9.8) is -10.0
log(x)	NATURAL LOGARITHM OF x (BASE e)	log(2.718282) is 1.0 log(7.389056) is 2.0
max(x, y)	LARGER VALUE OF x AND y	max(2.3, 12.7) is 12.7 max(-2.3, -12.7) is -2.3
min(x, y)	SMALLER VALUE OF x AND y	min(2.3, 12.7) is 2.3 min(-2.3, -12.7) is -12.7
pow(x, y)	x RAISED TO POWER y (x^y)	pow(2.0, 7.0) is 128.0 pow(9.0, .5) is 3.0
round(x)	ROUNDS x TO THE CLOSEST INTEGER	round(9.75) is 10 round(9.25) is 9
sin(x)	TRIGONOMETRIC SINE OF x (x IN RADIANS)	sin(0.0) is 0.0
sqrt(x)	SQUARE ROOT OF x	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
tan(x)	TRIGONOMETRIC TANGENT OF x (x IN RADIANS)	tan(0.0) is 0.0

Properties of the `Math` object.

Constant	Description	Value
Math.E	Base of a natural logarithm (*B*.	Approximately 2.718
Math.LN2	Natural logarithm of 2	Approximately 0.693
Math.LN10	Natural logarithm of 10	Approximately 2.302
Math.LOG2E	Base 2 logarithm of *E*	Approximately 1.442
Math.LOG10E	Base 10 logarithm of *E*	Approximately 0.434
Math.PI	π- the ratio of a circle's circumference to its diameter	Approximately 3.141592653589793
Math.SQRT1_2	Square root of 0.5	Approximately 0.707
Math.SQRT2	Square root of 2.0	Approximately 1.414

Some `String` object methods

Method	Description
`charAt( index )`	Returns a string containing the character at the specified *index*. If there is no character at the *index*, `charAt` returns an empty string. The first character is located at *index* 0.
`charCodeAt( index )`	Returns the Unicode value of the character at the specified *index*, or `NaN` (not a number) if there is no character at that *index*.
`concat( string )`	Concatenates its argument to the end of the string that invokes the method. The string invoking this method is not modified; instead a new `String` is returned. This method is the same as adding two strings with the string-concatenation operator + (e.g., `s1.concat(s2)` is the same as `s1 + s2`).
`fromCharCode( value1, value2, )`	Converts a list of Unicode values into a string containing the corresponding characters.
`indexOf( substring, index )`	Searches for the first occurrence of *substring* starting from position *index* in the string that invokes the method. The method returns the starting index of *substring* in the source string or -1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from index 0 in the source string.
`lastIndexOf( substring, index )`	Searches for the last occurrence of *substring* starting from position *index* and searching toward the beginning of the string that invokes the method. The method returns the starting index of *substring* in the source string or -1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from the end of the source string.
`replace( searchString, replaceString )`	Searches for the substring *searchString*, and replaces the first occurrence with *replaceString* and returns the modified string, or the original string if no replacement was made.

...Some `String` object methods

`slice( start, end )`	Returns a string containing the portion of the string from index *start* through index *end*. If the *end* index is not specified, the method returns a string from the *start* index to the end of the source string. A negative *end* index specifies an offset from the end of the string, starting from a position one past the end of the last character (so -1 indicates the last character position in the string).
`split( string )`	Splits the source string into an array of strings (tokens), where its *string* argument specifies the delimiter (i.e., the characters that indicate the end of each token in the source string).
`substr( start, length )`	Returns a string containing *length* characters starting from index *start* in the source string. If *length* is not specified, a string containing characters from *start* to the end of the source string is returned.
`substring( start, end )`	Returns a string containing the characters from index *start* up to but not including index *end* in the source string.
`toLowerCase()`	Returns a string in which all uppercase letters are converted to lowercase letters. Nonletter characters are not changed.
`toUpperCase()`	Returns a string in which all lowercase letters are converted to uppercase letters. Nonletter characters are not changed.
Methods that generate XHTML tags	
`anchor( name )`	Wraps the source string in an anchor element (`<a></a>`) with *name* as the anchor name.
`fixed()`	Wraps the source string in a `<tt></tt>` element (same as `<pre></pre>`).
`link( url )`	Wraps the source string in an anchor element (`<a></a>`) with *url* as the hyperlink location.
`strike()`	Wraps the source string in a `<strike></strike>` element.
`sub()`	Wraps the source string in a `<sub></sub>` element.
`sup()`	Wraps the source string in a `<sup></sup>` element.

Important `document` object methods and properties.

Method or property	Description
`getElementById( id )`	Returns the DOM node representing the XHTML element whose `id` attribute matches *ID*.
`write( string )`	Writes the string to the XHTML document as XHTML code.
`writeln( string )`	Writes the string to the XHTML document as XHTML code and adds a newline character at the end.
`cookie`	A string containing the values of all the cookies stored on the user's computer for the current document. See Section 11.9, Using Cookies.
`lastModified`	The date and time that this document was last modified.

...window Object properties

Method or property	Description
open(*url, name, options*)	Creates a new window with the URL of the window set to *URL*, the name set to *NAME* to refer to it in the script, and the visible features set by the string passed in as *OPTION*.
prompt(*prompt, default*)	Displays a dialog box asking the user for input. The text of the dialog is *PROMPT*, and the default value is set to *DEFAULT*.
close()	Closes the current window and deletes its object from memory.
focus()	This method gives focus to the window (i.e., puts the window in the foreground, on top of any other open browser windows).
blur()	This method takes focus away from the window (i.e., puts the window in the background).
window.document	This property contains the document object representing the document currently inside the window.
window.closed	This property contains a boolean value that is set to true if the window is closed, and false if it is not.
window.opener	This property contains the window object of the window that opened the current window, if such a window exists.

Standard objects by category (I)

Value properties	Function properties	Fundamental objects
• Infinity • NaN • undefined • null literal	• eval() • uneval() • isFinite() • isNaN() • parseFloat() • parseInt() • decodeURI() • decodeURIComponent() • encodeURI() • encodeURIComponent() • escape() • unescape()	• Object • Function • Boolean • Symbol • Error • EvalError • InternalError • RangeError • ReferenceError • SyntaxError • TypeError • URIError

Standard objects by category (II)

Numbers and dates	Text processing	Indexed collections
• Number • Math • Date	• String • RegExp	• Array • Int8Array • Uint8Array • Uint8ClampedArray • Int16Array • Uint16Array • Int32Array • Uint32Array • Float32Array • Float64Array

Standard objects by category (III)

Keyed collections	Structured data	Control abstraction objects
• Map • Set • WeakMap • WeakSet	• ArrayBuffer • SharedArrayBuffer • Atomics • DataView • JSON	• Promise • Generator • GeneratorFunction • AsyncFunction

Standard objects by category (IV)

Reflection	Internationalization	WebAssembly
• Reflect • Proxy **Other** • arguments	• Intl • Intl.Collator • Intl.DateTimeFormat • Intl.NumberFormat • Intl.RelativeTimeFormat • Intl.ListFormat	• WebAssembly • WebAssembly.Module • WebAssembly.Instance • WebAssembly.Memory • WebAssembly.Table • WebAssembly.CompileError • WebAssembly.LinkError • WebAssembly.RuntimeError

Development Tools

ENV-2

Tools

WEB BROWSER

We will use Google Chrome with built-in developer tools

https://developers.google.com/chrome-developer-tools/

EDITOR

JavaScript (JS) programs are stored in text files with .js file extension or embedded in HTML documents

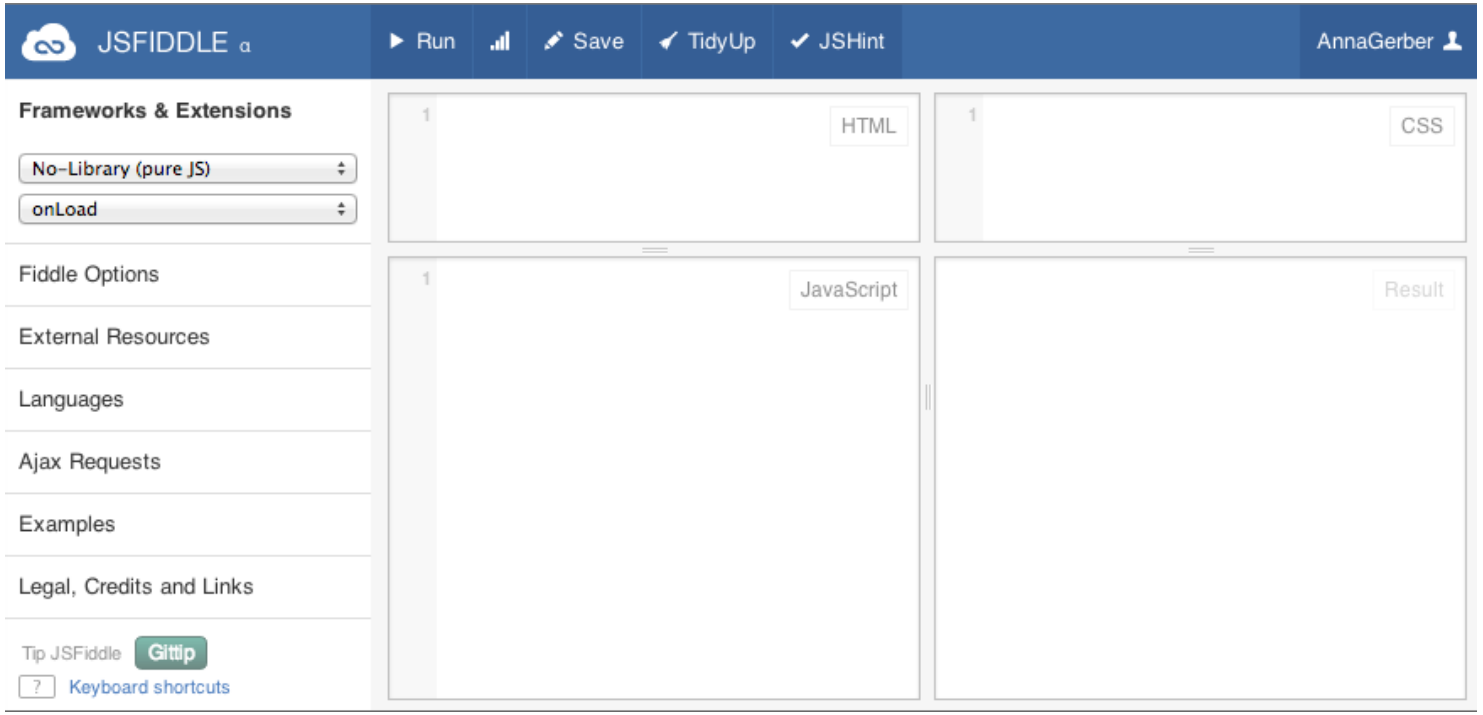
Use your favourite text editor to create and edit JS

We will use Chrome's console and JSFiddle (online editor)
http://jsfiddle.net/

JSFiddle

Online editor and development environment, allows code to be saved and shared

Great for prototyping



Web Authorization Tool

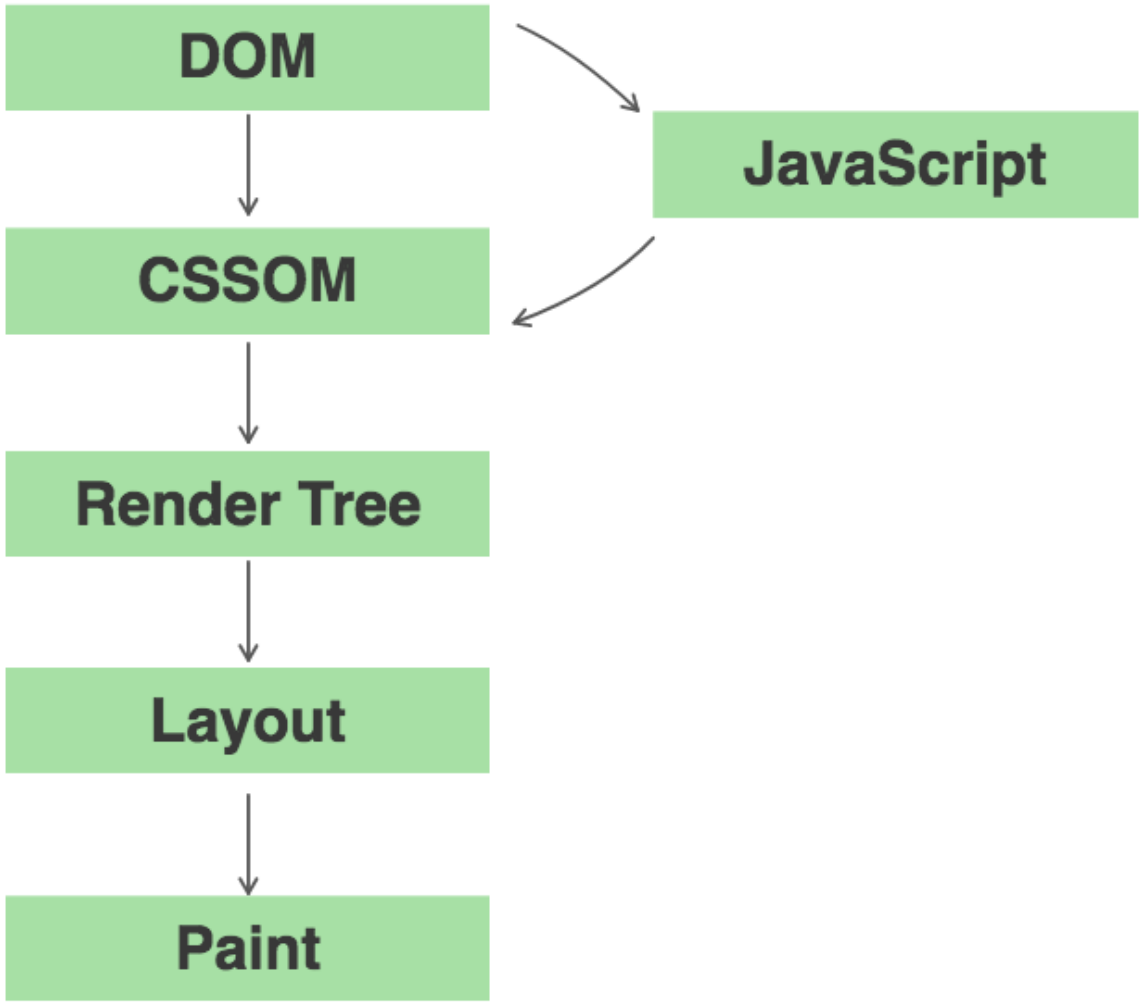
ENV-3

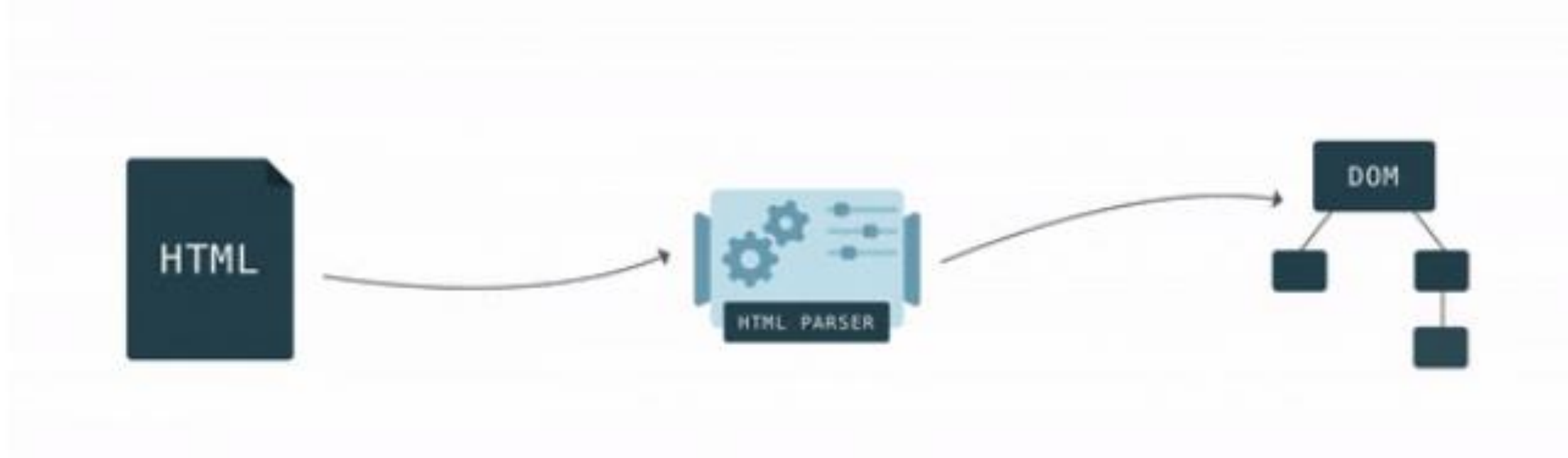
Installation of CoffeeCup Tool

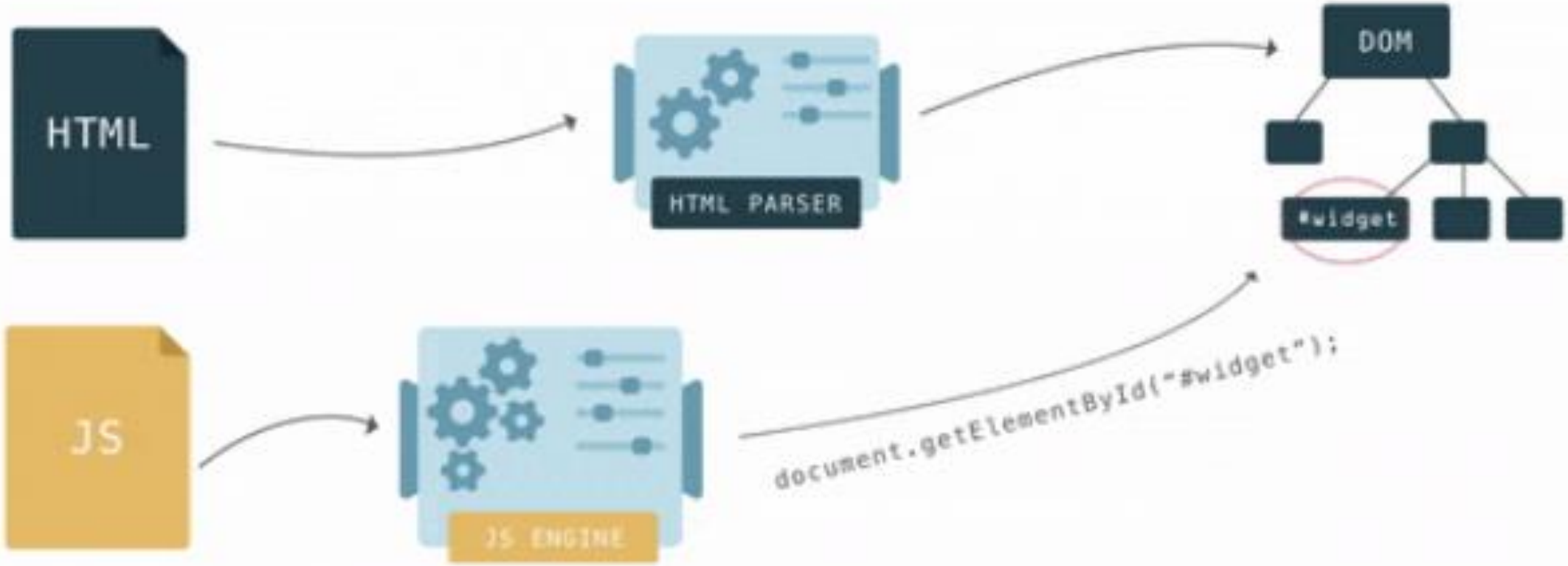
GO COFFEECUP!!!

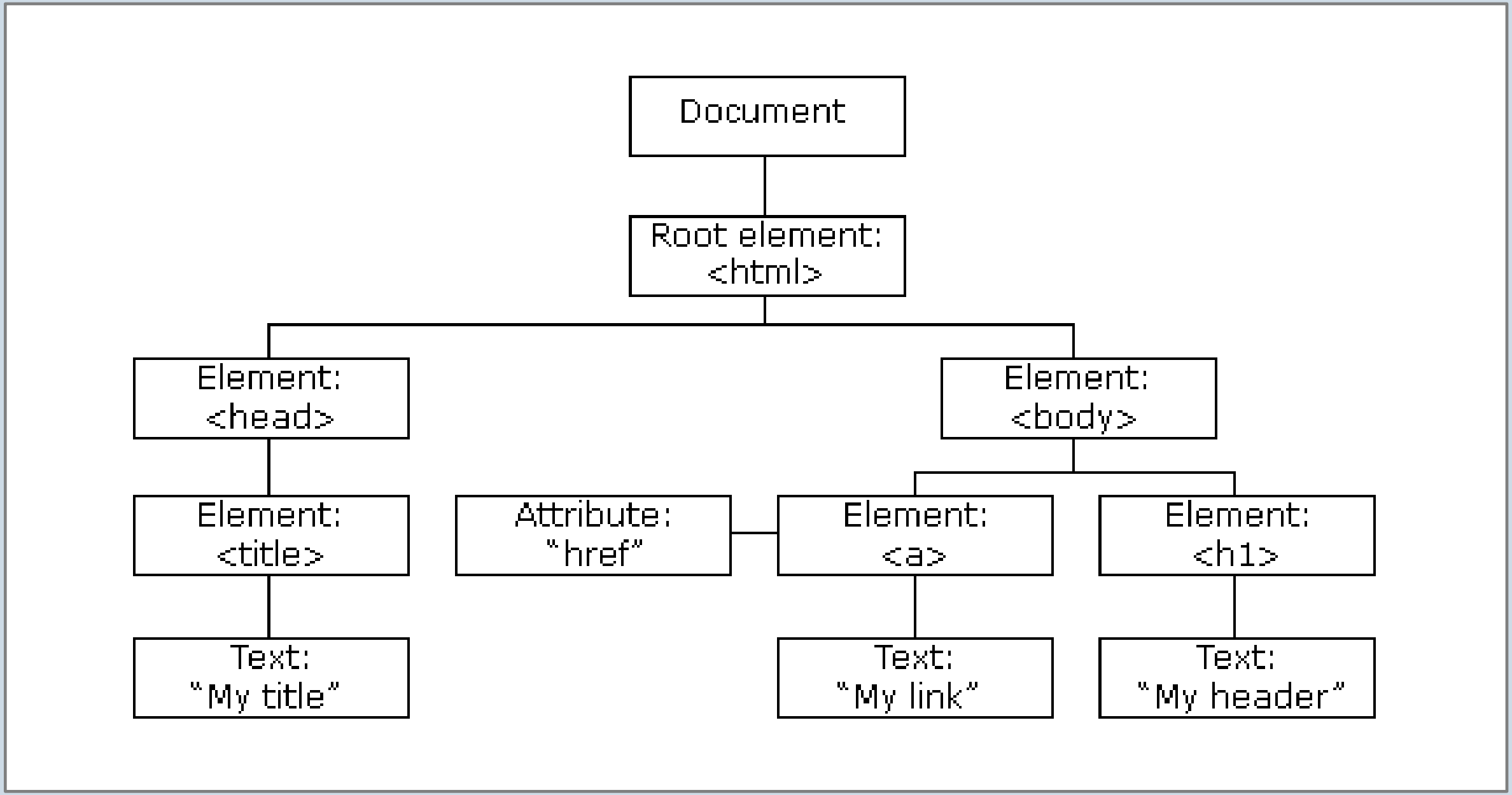
Execution of HTML/CSS/ JavaScript [Optional]

SECTION 16


DOM
JavaScript
CSSOM
Render Tree
Layout
Paint


HTML
HTML PARSER
DOM


HTML
HTML PARSER
DOM
#widget
JS
JS ENGINE
document.getElementById("#widget");


Document
Root element:
<html>
Element:
<head>
Element:
<body>
Element:
<title>
Attribute:
"href"
Element:
<a>
Element:
<h1>
Text:
"My title"
Text:
"My link"
Text:
"My header"

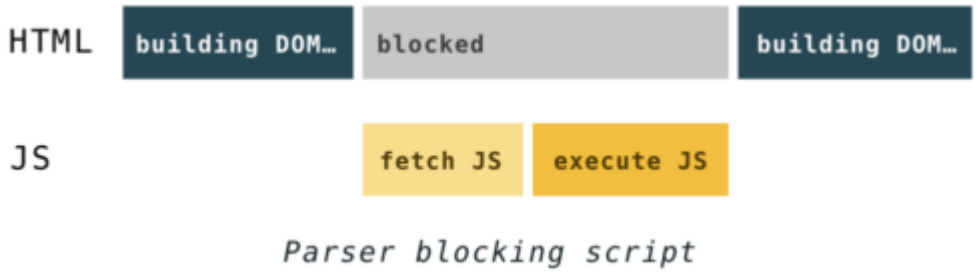
DOM and html

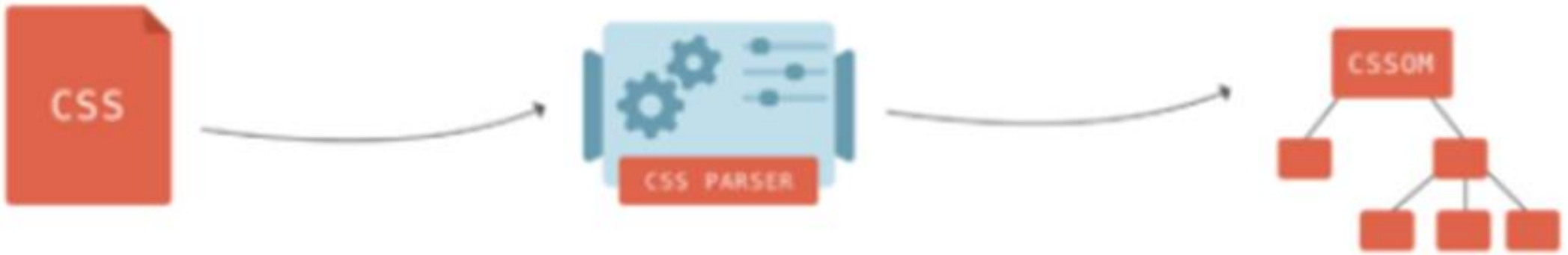
Demo Program: dom.html

```
<!DOCTYPE html>
<html>
<head><title>My Title</title></head>
<body>
<a href="http://www.eCodeHacker.com">My Link</a>
<h1>My Teacher</h1>
</body>
</html>
```

The history of the **<script>** tag

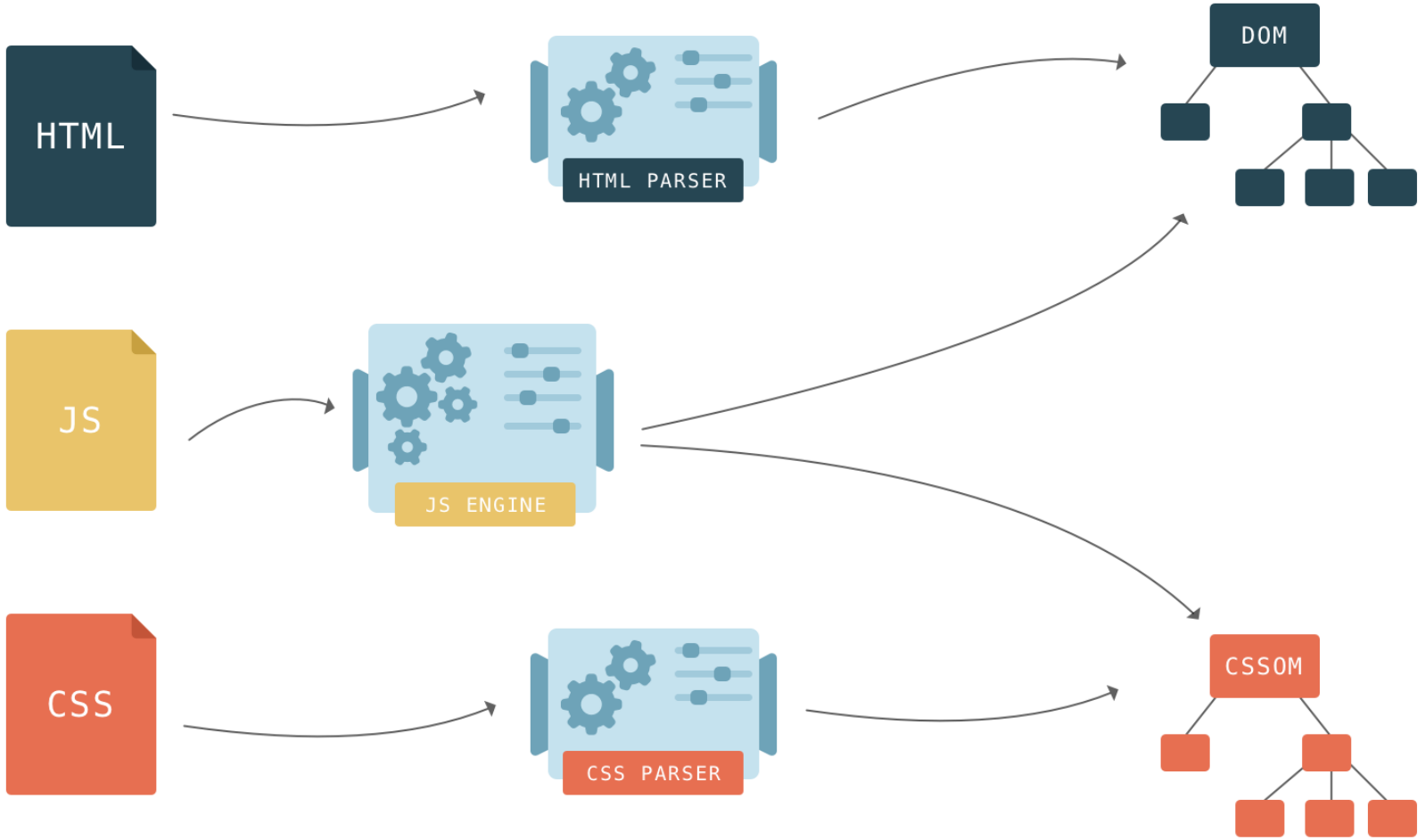
- As the browser is constructing the DOM, if it comes across a **<script>...</script>** tag in the HTML, it must execute it right away. If the script is external, it has to download the script first.
- Back in the old days, in order to execute a script, parsing had to be paused. It would only start up again after the JavaScript engine had executed code from a script.
- Why did the parsing have to stop? Well, scripts can change both the HTML and its product—the DOM. Scripts can change the DOM structure by adding nodes with **document.createElement()**. To change the HTML, scripts can add content with the notorious **document.write()** function. It's notorious because it can change the HTML in ways that can affect further parsing.




CSS
CSS PARSER
CSSOM

What about CSS?

- JavaScript blocks parsing because it can modify the document. CSS can't modify the document, so it seems like there is no reason for it to block parsing, right?
- However, what if a script asks for style information that hasn't been parsed yet? The browser doesn't know what the script is about to execute—it may ask for something like the DOM node's background-color which depends on the style sheet, or it may expect to access the **CSSOM** directly.

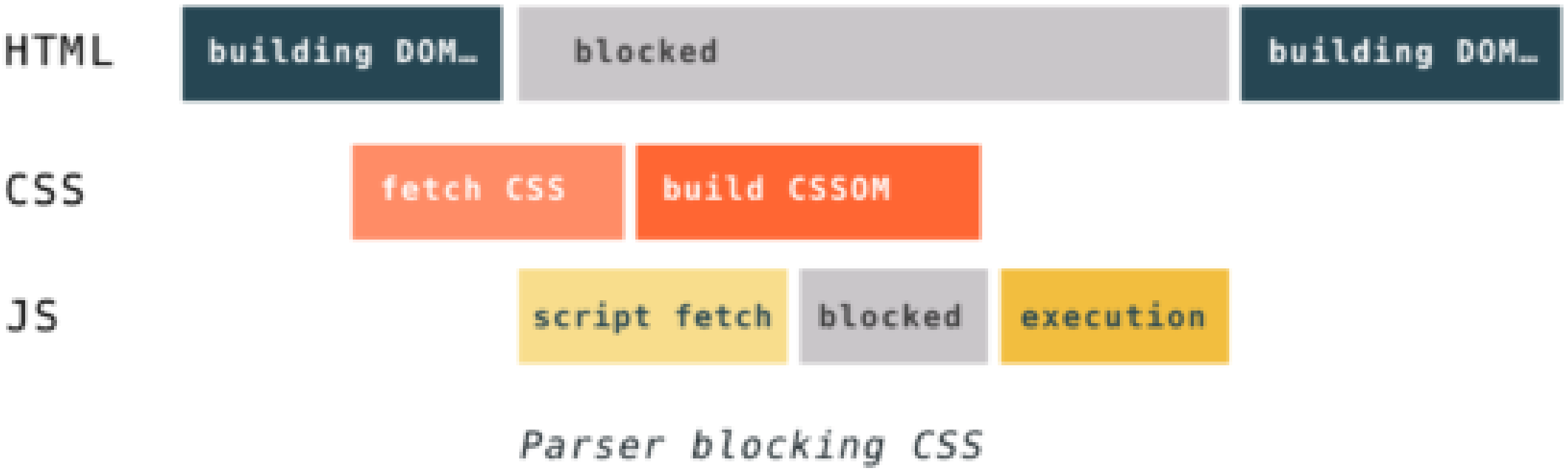

HTML
HTML PARSER
DOM
JS
JS ENGINE
CSS
CSS PARSER
CSSOM

Parsing CSS

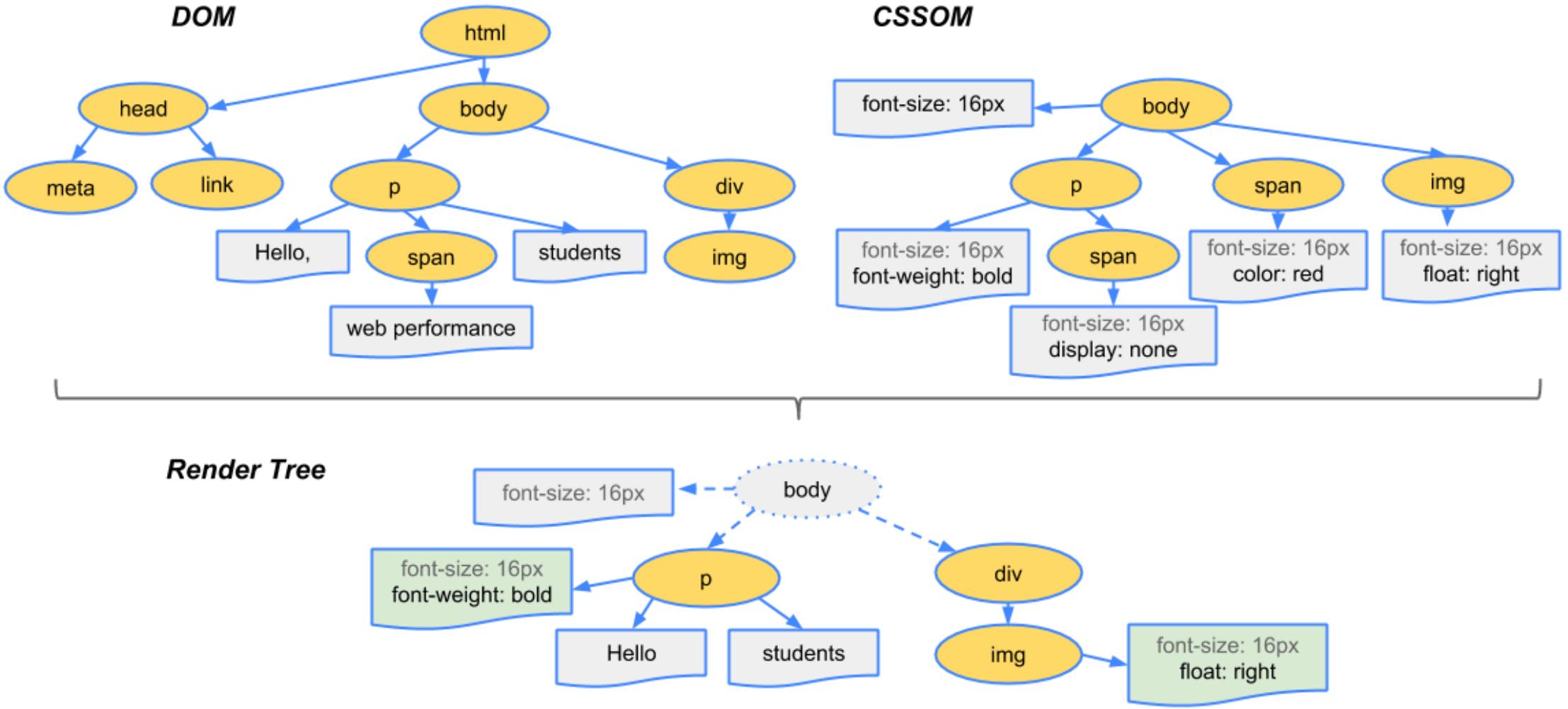
- Because of this, **CSS** may block parsing depending on the order of external style sheets and scripts in the document. If there are external style sheets placed before scripts in the document, the construction of **DOM** and **CSSOM** objects can interfere with each other.
- When the parser gets to a script tag, **DOM** construction cannot proceed until the JavaScript finishes executing, and the JavaScript cannot be executed until the **CSS** is downloaded, parsed, and the **CSSOM** is available.

Parsing CSS

- Another thing to keep in mind is that even if the CSS doesn't block DOM construction, it blocks rendering. The browser won't display anything until it has both the DOM and the CSSOM.
- This is because pages without CSS are often unusable. If a browser showed you a messy page without CSS, then a few moments later snapped into a styled page, the shifting content and sudden visual changes would make a turbulent user experience.



Parser blocking CSS


DOM
html
head
body
meta
link
p
div
Hello,
span
students
img
web performance
CSSOM
font-size: 16px
body
p
span
img
font-size: 16px
font-weight: bold
span
font-size: 16px
color: red
font-size: 16px
float: right
font-size: 16px
display: none
Render Tree
font-size: 16px
body
font-size: 16px
font-weight: bold
p
div
Hello
students
img
font-size: 16px
float: right

Internal CSS <style> tag

- Style tag and placed at many locations.
- Each <style></style> can include css script in the html section <head> or <body>

DOM and html
Demo Program:
dom1.html

```
<!DOCTYPE html>
<html>
<head><title>My Title</title></head>
    <style>
        title{font-size:12px;  font-family:"sans-serif"; }
    </style>
<body>
    <style>
        a{font-size:30px;  font-family:"sans-serif"; }
    </style>
<a href="http://www.eCodeHacker.com">My Link</a>
<h1>My Teacher</h1>
</body>
</html>
```

DOM and html

Demo Program: dom1.html

External Style Sheet

- With an external style sheet, you can change the look of an entire website by changing just one file!
- Each page must include a reference to the external style sheet file inside the **<link>** element. The **<link>** element goes inside the **<head>** section:

Example

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

DOM and html

Demo Program: dom2.html

dom2.html

```
<!DOCTYPE html>
<html>
<head><title>My Title</title></head>
   <link rel="stylesheet" href="mystyle.css">
<body>
<a href="http://www.eCodeHacker.com">My Link</a>
<h1>My Teacher</h1>
</body>
</html>
```

mystyle.css

```
body { background-color: #f2f218; }
```

The HTML Style Attribute

Inline (per-element) Style Properties

- Setting the style of an HTML element, can be done with the style attribute.
- The HTML style attribute has the following syntax:

The HTML `style` attribute has the following **syntax**:

```
<tagname style="property:value;">
```

Example:

```
<h1 style="color: blue; border:2px solid Tomato;">My Teacher</h1>
```

DOM and html

Demo Program: dom3.html

dom3.html

```
<!DOCTYPE html>
<html>
<head><title>My Title</title></head>
<body>
<a href="http://www.eCodeHacker.com">My Link</a>
<h1 style="color: blue; border:2px solid Tomato;">My Teacher</h1>
</body>
</html>
```

My Link

My Teacher