INT201 Client-side

JavaScript History

- 1995 – JavaScript is a programming language that was created by Brendan Eich who was working for Netscape. *คนพัฒนา*   *ชื่อของ browser ตอนนี้เป็น mozilla*
- 1997 – JavaScript 1.1 proposal was submitted to the European Computer Manufacturers Association (ECMA). *เข้าสู่มาตรฐาน*

ECMAScript

- The formal specification of the JavaScript language specified in the document ECMA-262 *อันมาตรฐาน*
- ES1, ES2, ES3,…ESX are a different version of the ECMAScript specification *ES* *มาตราฐานในการลงรับ script*

* Started from ES6, version of the ECMAScript start naming the versions based on the year of published specification, for example, ES2015 (ES6),ES2016 (ES7), … *เริ่ม v. ใส่ปี ปจบ. ES2020*

JavaScript

ES5 (2009) is fully supported by most modern browser in early 2016

- Higher order iteration functions (map, reduce, filter, foreach);
- JSON support; *JavaScript obj*
- Better reflection and object properties;

**ES6 (ES2015)** provide a greatly improved developer experience

- Classes
- Modules
- Iterators
- Generators
- Promises
- Arrow functions

ES5 & ES6 เริ่มใช้ JS ทำงานหลายๆ อย่างซับซ้อนขึ้นได้
รองรับการเขียนแบบ functional programing
รวม ทำงานหลายๆ อย่างได้ ในเวลาเดียวกันได้แบบ โดยอ้อม

From 2016 to 2019, a new edition of the ECMAScript standard was published each year, but the scope of changes was much smaller than the 5th or 6th editions

ES11 (ES2020), officially known as ECMAScript 2020, was published in June 2020 *minor*

JavaScript Ecosystem
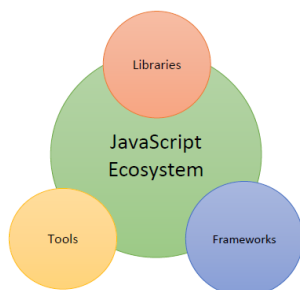
The different aspects of JavaScript

- Front-End: React, Angular, Vue.js, svelte, jQuery
- Back-End: node.js Deno
- Web Framework: Express
- Mobile: React Native, Apache Cordova Ionic
- Desktop: Electron
- Database: MongoDB

Introduction to JavaScript

- JavaScript is the programming language of the web.
- The overwhelming majority of websites use JavaScript, and all modern web browsers on desktops, tablets, and phones

ทำให้ JS มี power มากขึ้น ไม่ต้องผ่าน browser ก็สามารถ ประมวลผลได้

- Over the last decade, Node.js has enabled JavaScript programming outside of web browsers, and the dramatic success of Node means that JavaScript is now also the most used programming language among software developers.

- JavaScript is completely different from the Java programming language.

พัฒนามาจาก ECMAScript

JavaScript

Scripting lan. คำสั่งที่ทำกับตัวภาษา

can model document (html), สร้าง model tree ได้

ECMAScript	DOM ไม่ใช่ของ JS	BOM

run ที่ browser {

DOM: The Document Object Model. Map out an entire page as a hierarchy of nodes   JS can เที่ไปจัดการ obj. ของ เอกสารได้

BOM: The Browser Object Model. Deals with the browser window and frames   can เข้าถึง obj. ต่างๆบน browser ได้   ex. history , location bar etc.

Chromium
open source browser project

เป็น project ที่มี partner หลาย partner มาใช้ open source มาพัฒนา code base ของ browser ร่วมกัน

Chromium-based browser: Google Chrome   Microsoft Edge   Opera

พัฒนา browser เอง {

Safari is a graphical web browser developed by Apple, based on the WebKit engine.

Mozilla Firefox, or simply **Firefox**, is a free and open-source web browser developed by the Mozilla Foundation and its subsidiary, the Mozilla Corporation. Firefox uses the Gecko layout engine to render web pages.

Web Browser

Chrome V8   ใช้ JS ตัวเดียวกัน
open source JavaScript engine project

Chrome V8: Google Chrome   Microsoft Edge   Opera

JavaScriptCore: A JavaScript interpreter and JIT originally derived from KJS. It is used in the WebKit project and applications such as **Safari**.

SpiderMonkey: A JavaScript engine in Mozilla Gecko applications, including **Firefox.**

JavaScript Engine

high lv. lan. ต้องไปแปลเป็นภาษาเครื่อง

JavaScript Development Environment

① run ใน

In Web Brower

- Google Chrome
- Microsoft Edge
- Safari
- Firefox
- Opera

② run ใน

Outside Web Browser (based on Chrome V8 JavaScript Engine)

server side scripting เหมือนกับ run ใน brower   ประมวลผล high lv. lan.
Node.js: a JavaScript runtime built on Chrome's V8 JavaScript engine.
ปลอดภัยกว่า
Deno: a simple, modern and secure runtime for JavaScript and TypeScript that uses Chrome's V8 and is built in Rust.

MyFirstScript.js

```
console.log("I am JavaScript.");
```

Demo JavaScript In and Outside Web Browser

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="MyFirstScript.js"></script>
</head>
<body>
    <h1>Hello, This is my HTML page with JavaScript.</h1>
</body>
</html>
```
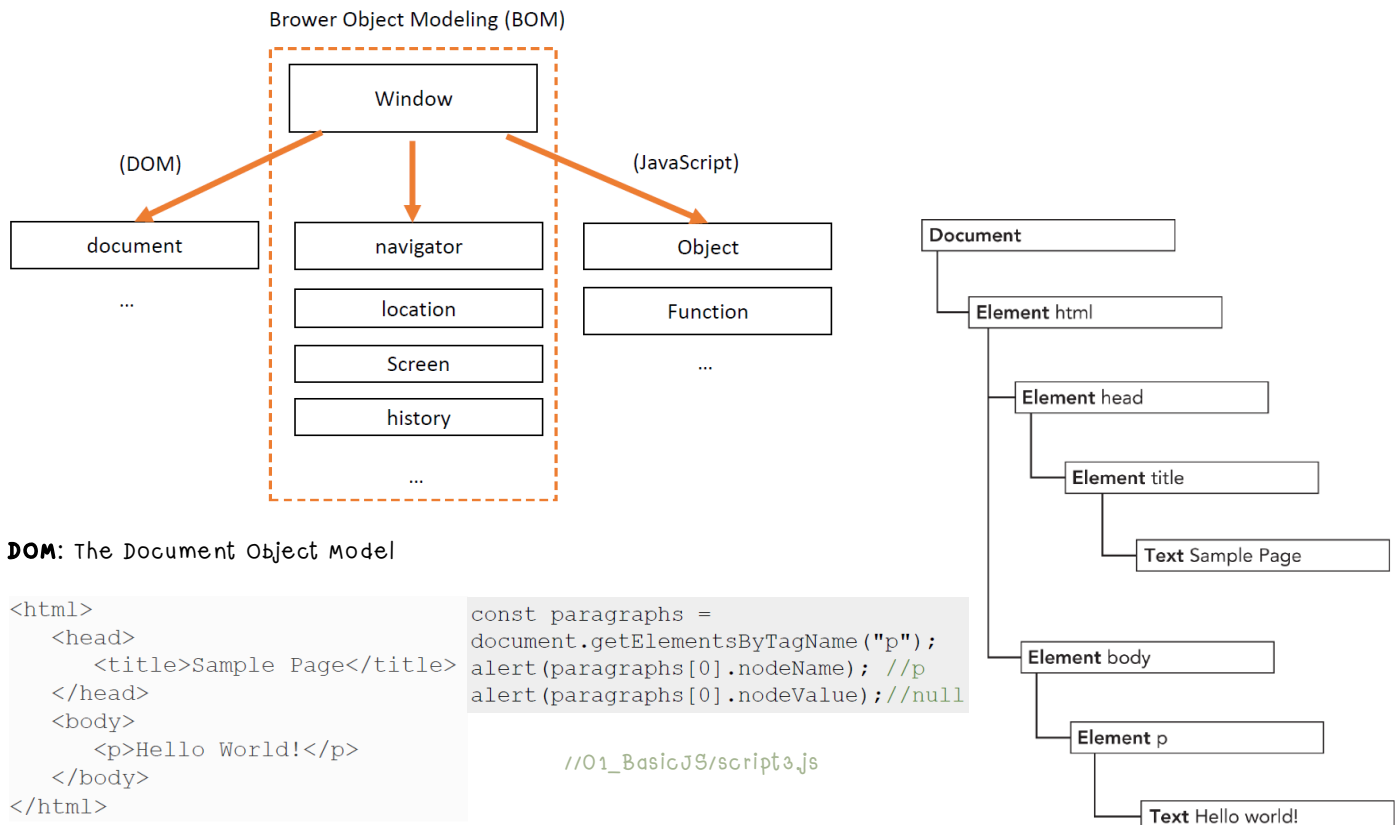
node ... ใน vs คือ run นอก browser but if อ้างถึง DOM & BOM ต้อง run ที่ browser bec. มีการเข้าถึง obj.

JavaScript Language Features

36:39-2:06/S

- Interpreted Language
- Single Threaded, do one operation at one time
- Dynamically and weakly typed language   //02_TypesValuesVariables/script1.js
- Support Object Oriented Programming (Prototyped based)

The Window interface represents a window containing a DOM document. In a tabbed browser, each tab is represented by its own Window object.

Brower Object Modeling (BOM)

DOM: The Document Object Model

```html
<html>
   <head>
      <title>Sample Page</title>
   </head>
   <body>
      <p>Hello World!</p>
   </body>
</html>
```

```
const paragraphs =
document.getElementsByTagName("p");
alert(paragraphs[0].nodeName); //p
alert(paragraphs[0].nodeValue);//null
```

//01_BasicJS/script3.js

Asynchronous vs. Synchronous Programming

- **Synchronous** tasks are performedone at a timeand only **when one is completed, the following is unblocked**. In other words, you need to wait for a task to finish to move to the next one.
- **Asynchronous** software design expands upon the concept by building code that allows a program to ask that a task be performed alongside the original task (or tasks), **without stopping to wait for the task to complete**. When the secondary task is completed, the original task is notified using an agreed-upon mechanism so that it knows the work is done, and that the result, if any, is available.

Asynchronous Callback Functions

// 01_BasicJS/script4.js

In JavaScript, a callback function is a function that is passed into another function as an argument.

This function can then be invoked during the execution of that higher order function.

```
console.log('Hello');

setTimeout(function () {
  console.log('JS');
}, 5000);

console.log('Bye bye');
```

```
//Console

Hello
Bye bye

//until 5 seconds
JS
```

Since, in JavaScript, functions are objects, functions can be passed as arguments.

setTimeout() executes a particular block of code once after a specified time has elapsed.

Higher-Order Functions

A "higher-order function" is a function that accepts functions as parameters and/or returns a function.

JavaScript Functions are **first-class citizens**

- be assigned to variables (and treated as a value)
- be passed as an argument of another function
- be returned as a value from another function

```
//01_BasicJS/script2.js
```

```
//1. store functions in variables

function add(n1, n2) {
  return n1 + n2
}
let sum = add

let addResult1 = add(10, 20)
let addResult2 = sum(10, 20)

console.log(`add result1: ${addResult1}`)
console.log(`add result2: ${addResult2}`)
```

```
//2. returned as a value from another function

function operator(n1, n2, fn) {
  return fn(n1, n2)
}

//3. Passing a function to another function
function multiply(n1, n2) {
  return n1 * n2
}

let addResult3 = operator(5, 3, add)
let multiplyResult = operator(5, 3, multiply)

console.log(`add result3 : ${addResult3}`)
console.log(`multiply result: ${multiplyResult}`)
```

```
console.log('Hello');
setTimeout(function () {
  console.log('JS');
}, 5000);
console.log('Bye bye');
```

```
//Console
Hello
Bye bye
//until 5 seconds
JS
```

```
console.log('Bye bye')
```

with Single thread, JavaScript
Runtime cannot do a setTimeout
while you are doing another code

```
//Call Stack




console.log('Hello')
main()
```

```
//Call Stack




setTimeout(fn, delay)
main()
```

```
//Call Stack




setTimeout(fn, delay)
main()
```

```
//Call Stack



log('Hello')
main()
```

```
//Call Stack



setTimeout(fn, delay)
main()
```

```
//Call Stack



console.log('Bye bye')
main()
```

```
//Call Stack



console.log('JS')
fn
```

Event loop comes in on concurrency, look at the stack and look at the task callback queue. If the stack is empty it takes the first thing on the queue and pushes it on to the stack

callback queue

```
//web APIs pushes the callback on to  the
callback queue when it's done
fn
```

Vanilla JavaScript is just plain or pure JavaScript without any additional libraries or framework

Types, Values, and Variables

Basic JavaScript Statements

- Semicolon in the end of statement is an optional
    - let x=10;

- let y=20
- Statement can take up multiple lines
- Comment
  - //Single Line Comment
  - /* ... */ Single or Multiple Lines Comment
- Console Printing
  - Console.log (*variable*);

Reserved Words

as	const	export	get	null	target	void
async	continue	extends	if	of	this	while
await	debugger	false	import	return	throw	with
break	default	finally	in	set	true	yield
case	delete	for	instanceof	static	try	catch
do	from	let	super	typeof	class	else
function	new	switch	var			

Types

JavaScript types can be divided into **two** categories:

1. primitive types

- number – including integer and floating-point numbers between –253 to 253
- string
- boolean

Primitive value

- number
- string
- boolean
- null (special type)
- undefined (special type)
- symbol (special type)

2. object types

- An object (that is, a member of the type object) is a collection of properties where each property has a name and a value (either a primitive value or another object)
- a special kind of object, known as **an array**, that represents an ordered collection of numbered values

JavaScript Data Types: numbers, string, boolean , undefined, symbol, object

//O2_TypesValuesVariables/script2.js

```
//output
type of myNum is number
type of myString is string
type of myBool is boolean
type of myUndefined is undefined
type of mySymbol is symbol
type of myNull is object
```

```javascript
let myNum = 0;
console.log(`type of myNum is ${typeof myNum}`);

let myString = 'Good';
console.log(`type of myString is ${typeof myString}`);

let myBool = true;
console.log(`type of myBool is ${typeof myBool}`);

let myUndefined;
console.log(`type of myUndefined is ${typeof myUndefined}`);

let mySymbol = Symbol();
console.log(`type of mySymbol is ${typeof mySymbol}`);

let myNull = null;
console.log(`type of myNull is ${typeof myNull}`);

let myArr = [1, 2, 3];

console.log(`myArr Length: ${myArr.length}`);
console.log(`type of myArr is ${typeof myArr}`);

let myObj = {id: 1, task: 'grading exam'};

console.log(`${JSON.stringify(myObj)}`);
console.log(`type of myObj is ${typeof myObj}`);
```

Null and undefined

- null is a language keyword that evaluates to a special value.
- null represent normal, expected absence of value and if there is no value, the value of variable can be set to null. If a variable is meant to later hold an object, it is recommended to initialize to null.
- Using the typeof operator on null returns the string "object" indicating that null can be thought of as a special object value that indicates "empty object pointer".
- JavaScript also has a second value that indicates absence of value. The undefined value represents **unexpected absence of value,** a deeper kind of absence.
    - the value of variables that have not been initialized
    - the value you get when you query the value of an object property or array element that does not exist.
    - value of functions that do not explicitly return a value
    - value of function parameters for which no argument is passed
- If you apply the typeof operator to the undefined value, it returns "undefined", indicating that this value is the sole member of a special type.

The following table summarizes the possible return values of typeof

Type	Result
Undefined	`"undefined"`
Null	`"object"` (see below)
Boolean	`"boolean"`
Number	`"number"`
BigInt (new in ECMAScript 2020)	`"bigint"`
String	`"string"`
Symbol (new in ECMAScript 2015)	`"symbol"`
Function object (implements [[Call]] in ECMA-262 terms)	`"function"`
Any other object	`"object"`

Literals

- 15   // The number twelve
- 1.5   // The number one point two
- "Hello World"   // A string of text
- 'Hi'   // Another string
- `"I' am a student", I said` // Another string
- true   // A Boolean value
- false   // The other Boolean value
- null   // Absence of an object

Escape sequences can be used in JavaScript: \n,\t, \\, \b, ...

Identifiers

- **Identifiers** are used to name constants, variables, properties, functions, and classes and to provide labels for certain loops in JavaScript code.

- A JavaScript identifier **must begin with a letter, an underscore (_), or a dollar sign ($).** Subsequent characters can be letters, digits, underscores, or dollar signs. (Digits are not allowed as the first character so that JavaScript can easily distinguish identifiers from numbers.)
- JavaScript is a **case-sensitive language.** This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

let, var, const variables

- One of the features that came with ES 6 is the addition of let and const , which can be used for variable
- var declarations are **globally scoped or function/locally scoped.**
- The scope is global when a var variable is declared outside a function. This means that any variable that is declared with var outside a function block is available for use in the whole window.
- All variables and functions declared globally with var **become properties and methods of the window object.**
- var is function scoped when it is declared within a function. This means that it is available and can be accessed only within that function.

var variables   // 01_BasicJS/ script5.js

```javascript
//greeting is globally scope, it exists outside a function
var greeting = 'Hey';

//var variables can be re-declared and updated
var greeting = 'Ho Ho';

function greeter() {
  //msg is function scoped, we cannot access the variable msg outside of a function
  var msg = 'hello';
}

// console.log(msg); //error: msg is not defined
console.log(greeting);
```

var variables can be re-declared and updated

This means that we can do this within the same scope and won't get an error.

```javascript
var year = 'leap';
if (year === 'leap')
    var greeting = 'Hey 366 days'; //re-declared
console.log(greeting);
```

It becomes a problem when you do not realize that a variable greeting has already been defined before.

let variables

- let is now preferred for variable declaration.
- JavaScript block of code is bounded by {}. A block lives in curly braces.Anything within curly braces is a block.
- let is **block scoped**, a variable declared in a block withletis only available for use within that block.
- **Let can be updated but not re-declared.**

let can be updated but not re-declared.   //01 _BasicJS/ script 6 .js

if the same variable is defined in different scopes, there will be no error. This is because both instances are treated as different variables since they have different scopes.

```javascript
let greeting = 'Hey';
greeting = 'Ho Ho';
function greeter() {
    let greeting = 'Good morning';
    console.log(`greeting in function is ${greeting}`);
}

greeter(); console.log(greeting); //Ho Ho
```

```javascript
/*let variables*/
//greeting is block scope
let greeting = 'Hey';
//let variables cannot be re-
declared, only can be updated
greeting = 'Ho Ho';
function greeter() {
  //msg is function scoped, we cannot access the variable msg outside of a function
  let msg = 'hello';
}
// console.log(msg); //error: msg is not defined
console.log(greeting);

let year = 'leap';
if (year === 'leap')
    greeting = 'Hey 366 days';
console.log(greeting);
```

const

- Variables declared with the const maintain constant values.
- const declarations share some similarities with let declarations.
- Like let declarations, const declarations can only be accessed within the block they were declared.
- const **cannot be updated or re-declared**
- Every const declaration, therefore, **must be initialized at the time of declaration**.

```
/*const variables*/
const greeting = 'Hey';
//const variables cannot be re-declared
// const greeting = |'Ho Ho';
//const variables cannot be updated
// greeting = 'Hi Hi';
```
//01 _BasicJS/ script 7 .js

JavaScript String

- The JavaScript type for representing text is the string.
- A string is an immutable ordered sequence of 16-bit values.
- JavaScript's strings (and its arrays) use zero-based indexing: the first 16-bit value is at position 0, the second at position 1, and so on.
- The empty string is the string of length 0.
- JavaScript does not have a special type that represents a single element of a string. To represent a single 16-bit value, simply use a string that has a length of 1.

Template Literals

```
let name = `Umaporn`;

let greeting = `Hello ${ name }.`;
```

- This is more than just another string literal syntax, however, because these template literals can include arbitrary JavaScript expressions.
- Everything between the is interpreted as a JavaScript expression
- Everything outside the curly braces is normal string literal text
- The final value of a string literal in backticks is computed by
    - o evaluating any included expressions,
    - o converting the values of those expressions to strings and
    - o combining those computed strings with the literal characters within the backticks

Explicit Conversions

- Although JavaScript performs many type conversions automatically, you may sometimes need to perform an explicit conversion, or you may prefer to make the conversions explicit to keep your code clearer.
- The simplest way to perform an explicit type conversion is to use the Boolean(), Number(), and String() functions:

```
//Explicit Conversions
Number('3'); //  3
String(false); // "false"
Boolean([]); //  true
```

How values convert from one type to another in JavaScript?

```
1 + ' objects'; //"1 objects":  Number 1 converts to a string
'5' * '4'; //20: both strings convert to numbers
let n = 'y' + 1; // n == NaN; string "y" can't convert to a number
```

JavaScript implicit type conversions

The primitive-to-primitive conversions shown in the table are relatively straightforward but Object-to-primitive conversion is somewhat more complicated
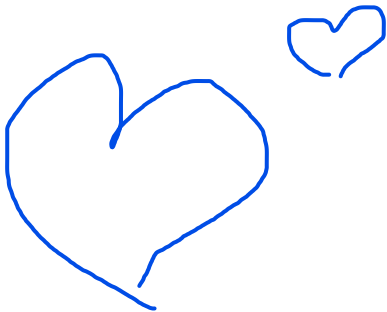
//examples of implicit type conversions

x + ""   // String(x)

+x   // Number(x)

x-0   // Number(x)

!!x   // Boolean(x)

//02_Types Values Variables/script5.js

Value	to String	to Number	to Boolean
undefined	"undefined"	NaN	false
null	"null"	0	false
true	"true"	1	
false	"false"	0	
"" (empty string)		0	**false**
"1.2" (nonempty, numeric)		1.2	true
"one" (nonempty, non-numeric)		NaN	true
0	"0"		**false**
-0	"0"		**false**
1 (finite, non-zero)	"1"		true
Infinity	"Infinity"		true
-Infinity	"-Infinity"		true
NaN	"NaN"		**false**
[] (empty array)	""	0	true

Value	to String	to Number	to Boolean
undefined			