INT201 Client-side

JavaScript History

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

ECMAScript

- The formal specification of the JavaScript language specified in the document ECMA-262
  องค์กรมาตรฐาน
- ES1, ES2, ES3,...ESX are a different version of the ECMAScript specification มาตรฐานในการลงรับ script
  ES

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

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

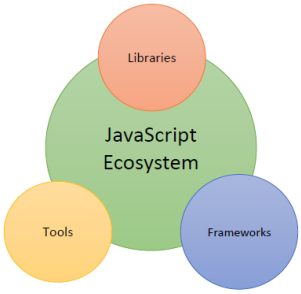
- Classes
- Modules
- Iterators
- Generators
- Promises
- Arrow functions

ES5 & ES6 เสริมให้ JS ทำงานหลายๆอย่างซับซ้อนขึ้นได้
รองรับการเขียนแบบ functional programming
ควบคุมงานหลายๆอย่างได้ในเวลาเดียวกันได้แบบโดยอัตโนมัติ

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 ไม่ต้องมี browser ก็ run 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.

พัฒนาจาก ECMAScript

JavaScript

ECMAScript	DOM	BOM
Scripting lan. คำสั่งที่ตกลงตัวภาษา	can model document (html), สร้าง model tree ได้ ไม่ใช่ของ JS	

run ที่ browser

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

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

Web Browser

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.

Chrome V8

ใช้ JS ตัวเดียวกัน

open source JavaScript engine project

JavaScript Engine

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.**

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 เหมือนเรา run ใน browser ประมวลผล high lv. lan.

- **Node.js**: a JavaScript runtime built on Chrome's V8 JavaScript engine.
- พัฒนาต่อ **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

```
<!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.

"terminal"
node
num
let num ประกาศตัวแปร
num = 3
num = 'INT201'
typeof (num) 'string'
num = 5
typeof (num) 'number'
num = {id: 1, name: 'Pornalee'}
num.email = 'pornalee.fern@mail.kmutt.ac.th'

: คั่นระหว่าง key กับ value
{ } = obj.

JavaScript Language Features

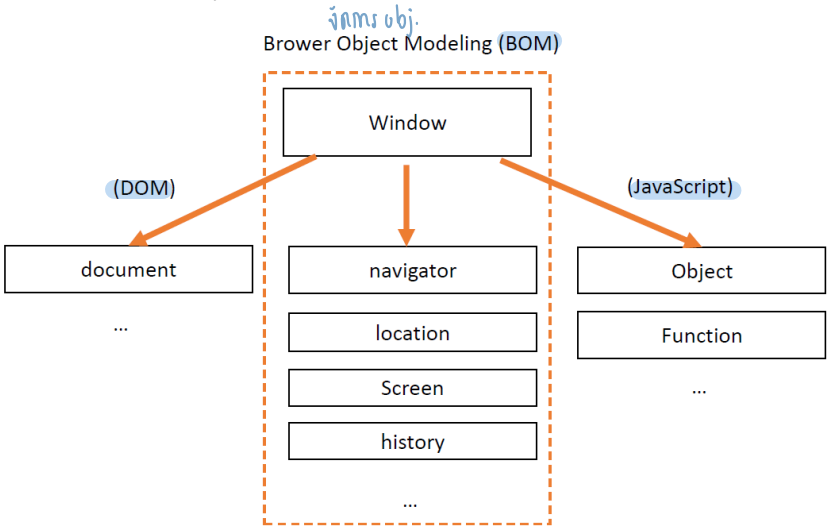
แปลทีละบรรทัด if มี error จะหยุด run ทันที

- Interpreted Language ใช้ translate ตัวเดียว
- Single Threaded, do one operation at one time ทำได้ทีละ 1 ค่าสิ่ง
- Dynamically and weakly typed language //02_TypesValuesVariables/script1.js
- Support Object Oriented Programming (Prototyped based)

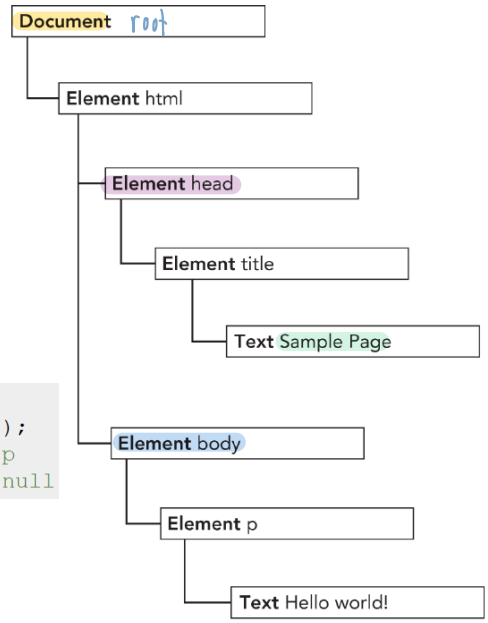
เวลาเปลี่ยนแปลงค่า type ของตัวแปรได้อิสระ

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

จัดการ obj.



tree structure ทำให้เห็น node ต่างๆ ได้ง่าย



DOM: The Document Object Model

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

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

//01_BasicJS/script3.js

Asynchronous vs. Synchronous Programming

ทำ task ให้เสร็จก่อนค่อยไป

- **Synchronous** tasks are performed one at a time and 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.

ยังทำไม่เสร็จก็ไปทำอันต่อไปก่อน

Asynchronous Callback Functions

Js ให้สิทธิ์ function เทียบเท่ากับ Data

function ที่เป็น parameter

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. รับ function เป็น parameter

// 01_BasicJS/script4.js

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

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

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

anonimus function
callback function

```c
//Console

Hello
Bye bye

//until 5 seconds
JS
```

delay

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

function ที่รับ / คืนค่าเป็น function ได้

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

JavaScript Functions are **first-class citizens** ประโยชน์ reuse ได้ เขียนแค่ครั้งเดียวใช้ได้ตลอด

- 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

```
//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}`)
```

//01_BasicJS/script2.js

```
//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}`)
```

รับ func เป็น para

```
console.log('Hello');  ①
setTimeout(function () {
  console.log('JS');  ③
}, 5000);
console.log('Bye bye');  ②
```

Output

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

console.log('Bye bye')

```
//Call Stack
console.log('Hello')
main()
```

```
//Call Stack
setTimeout(fn, delay)
main()
```

```
//Call Stack
setTimeout(fn, delay)
main()
```

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

```
//Call Stack
①
log('Hello')
main()
```

```
//Call Stack
setTimeout(fn, delay)
main()
```

คืนไว้

```
//Call Stack
②
console.log('Bye bye')
main()
```

```
//Call Stack
③
console.log('JS')
fn
```

Asynchonite

callback queue เอา callback func มาต่อคิว

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

จะไม่เข้าไปแทรกกลางทาง ต้องรอ stack empty

ไม่สนใจอะไรทั้งสิ้น

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; ใส่ / ไม่ใส่ก็ได้

    but แนะนำให้ใส่

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

Reserved Words

ส่วนใหญ่จะเป็น key word ที่เป็นคำสั่ง

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

ได้ทั้ง ทศนิยม & ตัวเลข

- number -including integer and floating-point numbers between -2^{53} to 2^{53}
- string text มัน character, char ใต้ '' / " " ก็ได้
- boolean

Primitive value ค่าพื้นฐาน แตกต่อไปไม่ได้แล้ว

- number
- string
- boolean
- null (special type) แตกต่อไม่ได้แล้ว
- undefined (special type)
- symbol (special type) symbol = unistring (string ไม่ซ้ำ)

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) obj. ซ้อน obj. ได้
- a special kind of object, known as **an array**, เปรียบเทียบ address that represents an ordered collection of numbered values

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

//02_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
```

```
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}`);
```

ตัวไว้เฉยๆ มีค่า ตัวไว้ไม่รู้ว่าจะมี/ไม่มีค่า

Null and undefined

เป็น absent value คือ ไม่มีค่าเกิดขึ้น

- 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 undefinedvalue 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) เช่น unique string ไม่มีค่าซ้ำ	"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
- back tip ` "I' am a student", I said` // Another string ` ให้ค่าควรที่ บางครั้ง ตัวค่าควรที่อาจจะ ปะกอบด้วย ' ' / " " ที่ ไปรบกันกับ ค.นมยครอบ 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

ชื่อใดๆ ที่ตั้งชื่อใน program

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

อนุญาติแค่ _ , $

- 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.** original ของ JS ในการประกาศตัวแปร
  ประกาศตัวแปรนอก func ไม่อยู่ภายใน func ใด
- 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

```
//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);
```

redeclared

greeting = 'H'; → update

อ้างอิงตัวแปรนี้นอก func ไม่ได้แล้ว

สร้างตัวแปรใหม่ — update ตัวเดิม แค่เปลี่ยน

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.

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

ไม่ควรทำ

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

let variables

ลดปัญหาของ var JS ทำได้ทั้งใน & นอก func

- 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.
  ใน fun
- let is **block scoped**, a variable declared in a block withletis only available for use within that block.
  นอก func
- Let **can be updated but not re-declared.**

```
/*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);
```

นอกปีลิตไม่ได้

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.

นักfunction

```
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
```

ไม่ได้ redeclared

if ไม่มี

ตัวแปรถูกสร้างคนละที่

ทำได้ตั้งค่าสุด

const เปลี่ยนค่าไม่ได้

- Variables declared with the const maintain constant values.
- const declarations share some similarities with let declarations. แต่เปลี่ยนค่าไม่ได้
- Like let declarations, const declarations can only be accessed within the block they were declared.
- const **cannot be updated or re-declared** สร้างตัวแปรใหม่
- 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';
```

ให้ค่าอะไรแล้วเป็นค่านั้นตลอด ไม่เปลี่ยนแปลง

//01 _BasicJS/ script 7 .js

ข้ามไป controlStructures file P.10

JavaScript String

ไม่ได้เปลี่ยนค่า value แต่แค่ return ค่า string ตาม function

- The JavaScript type for representing text is the string.
- A string is an immutable ordered sequence of 16-bit values. ไม่เปลี่ยนค่าในตัวแปร ไม่ได้ แต่เปลี่ยนรูปได้ โดยการสร้าง obj. ใหม่
- 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. length ไม่มี () ต่างกัน
- 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.

```
let msg = 'JS'
msg   JS
msg.length   2
msg.charAt(msg.length - 1)   ' '   s
msg.toLowerCase()   js
msg   JS
```

msg.substring(1,2) s

ตัวแปรอยู่ msg → 'JS'

msg.toLowerCast(); → 'js'

msg = msg.loLowerCast(); → JS JS

มีพื้นที่ ที่อยู่ของค่าเปลี่ยนแล้ว

string pool

ของ sign คือเปลี่ยนชี้ใหม่

Template Literals

```
let name = `Umaporn`;

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

console.log("Hello" + msg) Hellojs

(`Hello, 'section' ${msg}`) Hello, 'section'js

(`Hello world`) Hello World

expression เขียนได้เป็นนิพจน์

หมายถึงตัวแปร

- 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
  - evaluating any included expressions, ประมวลผลได้เลยบรรทัดก่อน (operation) ทุกอันที่ประมวลผลได้จะปรับเป็น string
  - converting the values of those expressions to strings and
  - combining those computed strings with the literal characters within the backticks

- ${x+y} operation (arithmetic/logical) and operands (x,y)
- ${let value = x+y} correct to let value = ${x+y}
- Simple expression -> complicated exp

console.log(`Hello ${msg}`) Hello js

+1}`) Hello js1

f1000}`) Hello 1000

{true && false}`) Hello false

{msg.charAt(1)}`) Hello s

- (Boolean exp) ? (true) expression : (false) expression

if else

✗ x>y ? let value = x+y : let value = x-y

✓ Let value = x>y ? : x+y : x-y

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

คำนวณโดยเรามองไม่เห็น "2" - 1 → 1 → Number("2") - 1

↳ เราไม่เห็นคำสั่ง แต่มันทำให้

emplicit ≈ manual คือชัดเจนเลยว่ามันเป็นอะไร x+y

implicit ≈ auto คือยังไม่ชัดเจน $x+y^3$

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_TypesValuesVariables/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`

Control Structures

JavaScript Operators

//02_TypesValuesVariables/script4.js

Operator precedence and associativity specify the order in which operations are performed in a complex expression.

ค.สำคัญ มาก

- Increment and Decrement
  ++ --
- Invert Boolean value
  !
- Type of operand
  typeof
- Arithmetic operators
  * / % + -
- Relational operators
  < <= > >=
- Equality operators
  == != (non strict equality)
  === !== (strict equality)
- Logical operators
  && ||
- Conditional operators
  ?:
- Assignment operators
  += -= *= /= %=

น้อย

Conversions and Equality

//02_TypesValuesVariables/script5.js

```
null == undefined // true: These two values are treated as equal.
"0" == 0          // true: String converts to a number before comparing.
0 == false        // true: Boolean converts to number before comparing.
"0" == false      // true: Both operands convert to 0 before comparing.


//if change from  == to strict equality ===, the results are all FALSE!
```

เท่ากัน

- JavaScript has two operators that test whether two values are equal.
- The "strict equality operator,"===, does not consider its operands to be equal if they are not of the same type.
- But because JavaScript is so flexible with type conversions, it also defines the==operator with a flexible definition of equality.

Equality with type conversion

The equality operator == is like the strict equality operator, but it is less strict. If the values of the two operands are not the same type, it attempts some type conversions and tries the comparison again:

- If the **two values have the same type**, test them for strict equality as described previously. **If they are strictly equal, they are equal. If they are not strictly equal, they are not equal.**

- If the **two values do not have the same type, the == operator may still consider them equal**. It uses the following rules and type conversions to check for equality:
  - If **one value is null and the other is undefined**, they are **equal.**
  - If **one value is a number and the other is a string, convert the string to a number** and try the comparison again, using the converted value.
  - **If either value is true, convert it to 1** and try the comparison again. **If either value is false, convert it to 0** and try the comparison again.
  - Any other combinations of values are not equal.

//02_TypesValuesVariables/script4.js

```
//Arithmetic operators
console.log(5 + 2); // => 7: addition
console.log(5 - 2); // => 3: subtraction
console.log(5 * 2); // => 10: multiplication
console.log(5 / 2); // => 2.5: division

// JavaScript defines some shorthand arithmetic operators
let count = 0; // Define a variable
count++; // Increment the variable
count--; // Decrement the variable
count += 3; // Add 3: same as count = count + 3;
count *= 2; // Multiply by 2: same as count = count * 2;
console.log(`count = ${count}`); // => 6: variable names are expressions

//conditional operator
let result = count > 5 ? 'count > 5' : ' count<=5';
console.log(`result = ${result}"`);

//== and != non-strict equality
//If the two operands are different types, interpreter attempts to convert them to suitable type.
console.log(`15 == '15' ${15 == '15'}`); //true

//=== and !=== strict equality without type conversion
console.log(`15 === '15' ${15 === '15'}`); //false

//logical operators
// && (and), || (or), ! (not)

console.log(`5 < '10' && '1' > 5 is  ${5 < '10' && '1' > 5}`); //false

console.log(`5 < '10' || '1' > 5 is  ${5 < '10' || '1' > 5}`); //true
console.log(`!(0) is ${!0}`); //true
```

คำตอบคือ ?, ใช้ implicit function ?

- 1 == true true, Number() convert true -> 1
- * "1" == [] false, Number([]) == Number("1") อาจมาเป็น obj.
- "1" == 1 true, Number("1")
- * 1 == null false, Number(null) ยังไม่แน่ใจ
- * "1" == undefined false, Number(undefined) = NaN
- typeof (null) 'object'
- null == 0 false
- typeof ([]) 'object'
- "0" == [] false
- 0 == [] true

JavaScript String

เปรียบเทียบ string 'B' < 'a' ดูตาม atly code 'A' - 'Z' = 65-90
65 < 97 'a' - 'z' = 97 - 122

- Strings can be compared with the standard === equality and !== inequality operators
- two strings are equal if and only if they consist of exactly the samesequence of 16-bit values.
- Strings can also be compared with the <, <=, >, and >= operators. String comparison is done simply by comparing the 16-bit values.
- To determine the length of a string—the number of 16-bit values it contains—use the length property of the string: str.length

//02_TypesValuesVariables/script3.js

```
let str1 = 'Hello';
let str2 = 'hello';
console.log(`str1 === str2 is ${str1 === str2}`);
console.log(`str1 < str2 is ${str1 < str2}`);
console.log(`str1 > str2 is ${str1 > str2}`);
console.log(`str1.length = ${str1.length}`);
console.log(
  `str1.toLowerCase === str2.toLowerCase is ${
    str1.toLowerCase === str2.toLowerCase
  }`
);
console.log(`str1.charAt(str1.length-1) = ${str1.charAt(str1.length - 1)}`);
```

```
//output
str1 === str2 is false
str1 < str2 is true
str1 > str2 is false
str1.length = 5
str1.toLowerCase === str2.toLowerCase is true
str1.charAt(str1.length-1) = o
```

Comparing Primitives vs Objects

value address

- **Primitives are also compared by value**: two values are the same only if they have the same value.
- Objects are not compared by value: two distinct objects are not equal even if they have the same properties and values.
- **Objects** are sometimes called **reference types** to distinguish them from JavaScript's primitive types
- we say that objects are compared by reference: two object values are the same if and only if they refer to the same underlying object.

//02_TypesValuesVariables/script2.js

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

let myObj2 = {
  id: 1,
  task: 'grading exam'
};

newObj = myObj;
console.log(`newObj === myObj is ${newObj === myObj}`);
console.log(`myObj1 === myObj2 is ${myObj === myObj2}`);
```

myObj. myObj2 newObj. id = 1 task = xxx

```
//output
newObj === myObj is true
myObj1 === myObj2 is false
```

And two distinct arrays are not equal even if they have the same elements in the same order:

```
let a = [];
let b = a;
b[0] = 1;
let c = [1];
console.log(`a === b is ${a === b}`);
console.log(`b == c is ${b == c}`);
```

//02_TypesValuesVariables/script3.js

```
//output
a === b is true
b == c is false
```

obj. เดียวกัน อยู่ที่เดียวกัน

a b 0 1 c 1

คนละ array

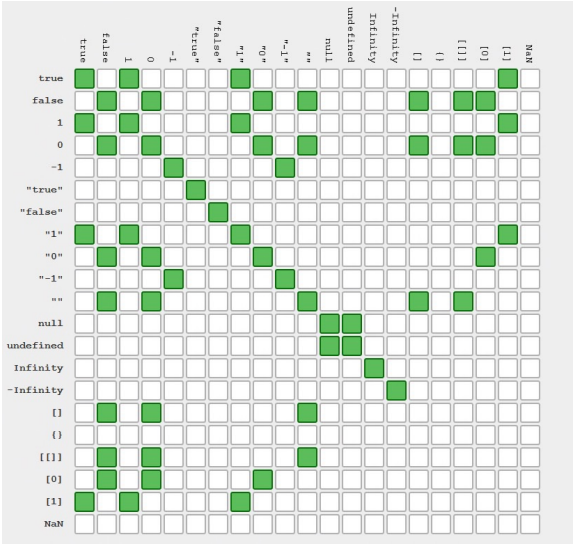
Conditionals - *if/else*

use a statement block { } to combine multiple statements into one

```
if (expression)
  statement
```

```
if (expression)
    statement1
else
    statement2
```

```
if (expression1) {
    // Execute code block #1
}
else if (expression2) {
    // Execute code block #2
}
else if (expression3) {
    // Execute code block #3
}
else {
    // If all else fails, execute block #4
}
```



Conditionals - *switch*

The matching case is determined using the === identity operator, not the==equality operator, so the expressions must match without any type conversion.

```
switch(n) {
   case 1: // Start here if n === 1
   // Execute code block #1.
   break; // Stop here
   case 2: // Start here if n === 2
   // Execute code block #2.
   break; // Stop here
   case 3:
   // Start here if n === 3 // Execute code block #3.
   break; // Stop here
   default:
   // If all else fails... // Execute code block #4.
   break; // Stop here
}
```

```
switch(expression){
    statements
}
```

Loop - *while/do while*

```
while (expression)
    statement
```

```
let count = 0;
while(count < 10) {
    console.log(count);
    count++;
}
```

```
do
    statement
while (expression);
```

```
let count = 0;
do {
  console.log(count);
  count++;
} while (count < 10);
```

Loop - *for*

The **for** statement simplifies loops that follow a common pattern.

```
for(initialize ; test ; increment)
    statement
```

```
for(let i = 0, len = data.length; i < len; i++)
    console.log(data[i]);
```

The **for/of** loopworks with *iterable* objects, arrays, strings, sets, and maps are iterable:

```
for(variable of iterableObject)
    statement
```

```
let data = [1, 2, 3, 4, 5, 6, 7, 8, 9]
let sum = 0;
for(let element of data) {
    sum += element;
});
console.log(`sum = ${sum}`); //sum=45
```

The **for/in** statement loops through the property names of a specified object

```
for (variable in object)
    statement
```

```
for(let property in object) {
    console.log(property); //print property name
    console.log (object[property]); //print value of each property
}
```

<u>แบบฝึกหัด 1</u> เขียนโปรแกรมเพื่อแสดงราศีที่ตรงกับปีที่กำหนดไว้ โดยมีทั้งหมด 12 ราศี ซึ่งแทนโดยสัตว์ประเภทต่าง ๆ ตัวอย่างเช่น ปี 1900 % 12 จะมีค่า 4 ซึ่งจะแทนด้วยราศีหนู

0: monkey
1: rooster
2: dog
3: pig
4: rat
5: ox
6: tiger
7: rabbit
8: dragon
9: snake
10: horse
11: sheep

https://www.hisgo.com/us/destination-japan/blog/japanese_horoscope.html

<u>แบบฝึกหัดที่ 4 ให้เขียนโปรแกรมเพื่อทำเมนูให้เลือกกับการจัดการ Text String ให้ทดสอบโดยใช้ String อย่างน้อย 3 กรณีที่แตกต่างกัน</u>

- ให้เขียน Function เพื่อแสดงเมนูให้เลือกในการจัดการ String
  - 1: Reverse String
  - 2: Replace Vowels with '*'
  - 3: Count Vowels in String
- ตัวอย่างเช่น "Hello World"
  - กด 1 ได้ "dlroW olleH"
  - กด 2 ได้ "H*ll* W*rld"
  - กด 3 ได้ 3