**INT201 Client-side**

**JavaScript History**

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

**ECMAScript**

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

\* Started from ES6, version of the ECMAScript start naming the versions based on the year of published specification, for example, ES2015 (ES6),ES2016 (ES7), …

**JavaScript**

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

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

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

* Classes
* Modules
* Iterators
* Generators
* Promises
* Arrow functions

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

**JavaScript EcoSystem The different aspects of JavaScript**

Graphical user interface, application

Description automatically generatedDiagram

Description automatically generated

**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
* 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.

Table

Description automatically generated with medium confidence

**DOM**: The Document Object Model. Map out an entire page as a hierarchy of nodes

**BOM**: The Browser Object Model. Deals with the browser window and frames

Text

Description automatically generated

**JavaScript Development Environment**

Text, Word

Description automatically generated with medium confidence**In Web Brower Outside Web Browser** (based on Chrome V8 JavaScript Engine)

Text

Description automatically generatedText, application

Description automatically generated

**Demo JavaScript In and Outside Web Browser**

**JavaScript Language Features**

* 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.

A picture containing diagram

Description automatically generatedDiagram

Description automatically generated

A picture containing diagram

Description automatically generated**DOM**: The Document Object Model

A picture containing diagram

Description automatically generated

//01\_BasicJS/script3.js

**Asynchronous vs. Synchronous Programming**

* **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.
* Text

  Description automatically generated**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**

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.

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

[setTimeout()](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/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.

1. JavaScript Functions are **first-class citizens**

//01\_BasicJS/script2.js

* be assigned to variables (and treated as a value)
* Text

  Description automatically generatedbe passed as an argument of another function
* be returned as a value from another function

Text

Description automatically generated

Text, letter

Description automatically generated Text

Description automatically generated with low confidence Graphical user interface

Description automatically generated with low confidence

A picture containing graphical user interface

Description automatically generated A picture containing graphical user interface

Description automatically generated A picture containing table

Description automatically generated

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

Graphical user interface

Description automatically generated with medium confidence Graphical user interface

Description automatically generated with medium confidence Graphical user interface

Description automatically generated with medium confidence A picture containing text

Description automatically generated

**callback queue**

A picture containing graphical user interface

Description automatically generatedEvent 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

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

**Types, Values, and Variables**

1. **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**

Table

Description automatically generated with low confidence

1. **Types**
2. JavaScript types can be divided into **two** categories:
3. **1. primitive types**

* number -including integer and floating-point numbers between -2^53 to 2^53
* string
* boolean

**Primitive value**

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

1. Text

   Description automatically generated**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**

//02\_TypesValuesVariables/script2.jsA picture containing text, indoor

Description automatically generated

1. **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 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 typeofGraphical user interface, text, application

Description automatically generated

1. **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, …

1. **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

Graphical user interface, text, application

Description automatically generated

**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.

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

Text, letter

Description automatically generated

1. **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.
* Text, letter

  Description automatically generatedLet **can be updated but not re-declared.**

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

Text

Description automatically generatedif 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.

1. **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**.

1. A picture containing text, indoor

   Description automatically generated //01 \_BasicJS/ script 7 .js
2. **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
  + evaluating any included expressions,
  + converting the values of those expressions to strings and
  + combining those computed strings with the literal characters within the backticks

1. **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.
* Text, letter

  Description automatically generatedThe simplest way to perform an explicit type conversion is to use the Boolean(), Number(), and String() functions:

**How values convert from one type to another in JavaScript?**

Text

Description automatically generated with low confidence

**JavaScript implicit type conversions**

Graphical user interface

Description automatically generated with low confidenceThe 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

**Control Structures**

**JavaScript Operators**

//02\_TypesValuesVariables/script4.js

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

A picture containing text, screenshot, indoor

Description automatically generatedGraphical user interface, text

Description automatically generated

1. **Conversions and Equality**

* 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.

1. **Equality with type conversion**
2. 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

Text

Description automatically generated

1. **JavaScript String**

* 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

Text

Description automatically generatedText

Description automatically generated

1. **Comparing Primitives vs Objects**

* **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

Text

Description automatically generatedText, letter

Description automatically generated

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

//02\_TypesValuesVariables/script3.js

A picture containing chart

Description automatically generatedText, letter

Description automatically generated

**Conditionals - *if/else***

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

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

**Conditionals - *switch***

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

Text, letter

Description automatically generatedText, letter

Description automatically generated

**Loop - *while/do while***

Text

Description automatically generated Text

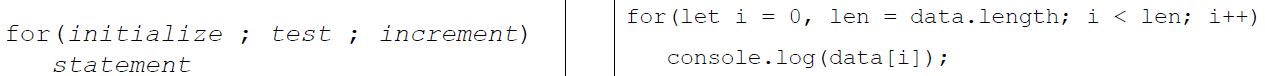
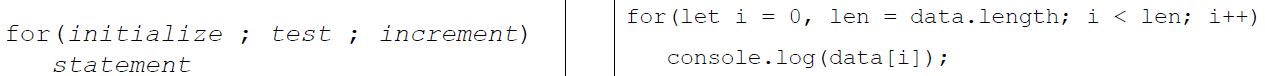
Description automatically generated Text

Description automatically generated Text

Description automatically generated

**Loop - *for***

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



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

A screenshot of a text message

Description automatically generated with low confidence Text, letter

Description automatically generated

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

Text

Description automatically generatedText

Description automatically generated