**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

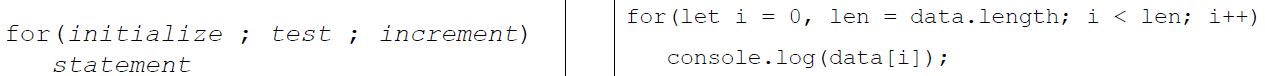
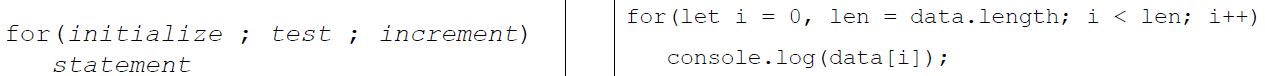
Description automatically generated Text

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

Chart, radar chart

Description automatically generatedText, letter

Description automatically generated

**JavaScript Objects**

* ECMAScript **objects as hash tables**: nothing more than a grouping of name‐value pairs where the value may be **data or a function**.
* **An object** is an **unordered collection of properties**
* An object is a **composite value**: it aggregates multiple values (primitive values or other objects) and allows you to store and retrieve those values by name.
* Property names are usually Strings or can also be Symbols.
* No object may have two properties with the same name.
* JavaScript objects are dynamic—properties can usually be added and deleted
* It is possible to create an instance of an “implicit” class without the need to actually create the class.

**JavaScript Object Examples**

Graphical user interface, application, Word

Description automatically generated

**Object Passing to functions by reference**

Objects are **mutable** and manipulated by reference rather than by value.

Text, letter

Description automatically generated

**Understanding Objects**

Text, letter

Description automatically generated

**Shorthand Object Methods**

* When function is defined as a property of an object, we call that function a method
* Prior to ES6

Text

Description automatically generated with medium confidence

* In ES6, the object literal syntax has been extended to allow a shortcut where the function keyword and the colon are omitted,

Graphical user interface, text

Description automatically generated

**Understanding Object Creation**

1. Simplest form with **object literals**, object literal is a comma‐separated list of {name: value} pairs.



1. with the **new** operator. Objects created using the new keyword and a constructor invocation use the value of the prototype property of the constructor function as their prototype.

Text

Description automatically generated

1. with the **Object.create()** function. Creates a new object, with specified prototypes.

Text, letter

Description automatically generated

**Understanding Object Creation**

Graphical user interface, application

Description automatically generated

**Object literals**

* The easiest way to create an object is to include an object literal in your JavaScript code.
* In its simplest form, an object literal is a comma‐separated list of colon‐separated **name: value** pairs, enclosed within curly braces {}.
* A **property name** is a JavaScript identifier or a string literal
* A **property value** is any JavaScript expression; the value of the expression (it may be a primitive value or an object value) becomes the value of the property

**Object literal Examples**

Graphical user interface, text, application

Description automatically generated

**Getting, Setting , Creating Object Properties**

* To obtain the value of a property, use the dot(.) or square bracket([]) operators

Text

Description automatically generatedText, letter

Description automatically generated

* with the[] array notation, the name of the property is expressed as a string.
* Strings are JavaScript data types, so they can be manipulated and created while a program is running.

Create class and constructor functions (ES6) Graphical user interface, text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

**Object Prototypes**

* **Prototypes** are the mechanism by which JavaScript objects inherit features from one another.
* **JavaScript** is often described as a prototype‐based language— to provide inheritance, objects can have a prototype object, which acts as a template object that it inherits methods and properties from.

**Prototype Chaining**

* ECMA‐262 describes **prototype chaining** as the primary method of **inheritance** in ECMAScript.
* The object created by **new** Object() or **object literal** inherit from **Object.prototype**
* Similarly, the object created by new Array()uses **Array.prototype** as its prototype, and the object created by new Date() uses **Date.prototype** as its prototype.
* **Date.prototype** inherits properties from **Object.prototype**, so a **Date** object created by **new Date()** inherits properties from both **Date.prototype** and **Object.prototype**.
* This linked series of prototype objects is known as a **prototype chain**.
* JavaScript objects have a set of “**own properties**” and they also inherit a set of properties from their prototype object.

Text

Description automatically generated

prototypeObj.isPrototypeOf(object)

object ‐ the object whose prototype chain will be searched. Return a Boolean indicating whether the calling object lies in the prototype chain of the specified object.

Text

Description automatically generated

**How to Compare Objects in JavaScript**

1. Referential equality: ==, ===, Object.is()
2. Manual comparison of properties’ values.
3. Shallow Equality check the properties’ values for equality.

**Referential equality**

* Both are the same object means both object point to the same object instances.
* Three ways to compare objects:
  + The strict equality operator ===
  + The loose equality operator ==
  + Object.is() function

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

**Manual Comparison**

A manual comparison of properties’ values.

Text

Description automatically generated

**Shallow Equality**

Graphical user interface, text, application

Description automatically generated

**JSON – JavaScript Object Notation**

* JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax.
* It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).
* Even though it closely resembles JavaScript object literal syntax, it can be used independently from JavaScript, and many programming environments feature the ability to read (parse) and generate JSON.
* A picture containing timeline

  Description automatically generatedA JSON string can be stored in its own file, which is basically just a text file with an extension of .json, and a MIME type of application/json.

**JSON structure**

* **JSON is a string** whose format very much resembles JavaScript object literal format.
* JSON **requires double quotes** to be used around strings and property names. **Single quotes are not valid** other than surrounding the entire JSON string.
* You can include the same basic data types inside JSON as you can in a standard JavaScript object — strings, numbers, arrays, booleans, and other object literals.
* JSON is purely a string with a specified data format **— it contains only properties, no methods.**
* We can also convert arrays to/from JSON.

**JavaScript Arrays**

**Arrays**

* An array is an ordered collection of values . JavaScript arrays are object
* Each value is called an element , and each element has a numeric position in the array, known as its index ( zero based index
* JavaScript arrays are untyped : an array element may be of any type , and different elements of the same array may be of different types
* Array elements may even be objects or other arrays , which allows you to create complex data structures such as arrays of objects and arrays of arrays.
* JavaScript arrays are dynamic : they grow or shrink as needed , and there is no need to declare a fixed size for the array when you create it or to reallocate it when the size changes.
* Every JavaScript array has a length property.

**Creating Arrays** //05\_Arrays/script2.js

1. Array literals
2. The ... spread operator on an iterable object
3. The Array() constructor
4. The Array.of () and Array.from () factory methods

**1. Array literals**

* The simplest way to create an array is with an array literal, which is simply a comma separated list of array elements within square brackets

Text

Description automatically generated

**2. The ... spread operator on an iterable object**

* In ES6 and later, you can use the “spread operator,” ..., to include the elements of one array within an array literal:



* The three dots “spread” the array so that its elements become elements within the array literal that is being created.
* The spread operator is a convenient way to create a (shallow) copy of an array:

Text, letter

Description automatically generated

* Modifying the copy does not change the original

1. **3. The Array() Constructor**

* Call it with **no arguments**:

let a = new Array();

* Call it with a single numeric argument, which **specifies a length**:

let a = new Array(10);

* Explicitly **specify two or more array elements** or **a single non-numeric element for the array**:

let a = new Array(3, 2, 1, "testing");

1. **4. The Array.of() and Array.from() factory methods**

* The Array() constructor cannot be used to create an array with a single numeric element.
* InES6, theArray.of() function addresses this problem: it is a factory method that creates and returns a new array, using its argument values (regardless of how many of them there are) as the array elements:

Text

Description automatically generated

* Array.from is another array factory method introduced in ES 6.
* It expects an iterable or array like object as its first argument and returns a new array that contains the elements of that object.
* With an iterable argument, Array.from (iterable) works like the spread operator [... iterable ] does. It is also a simple way to make a copy of an array:

A picture containing table

Description automatically generated

**Reading and Writing Array Elements**

* You access an element of an array using the [ ] operator.
* An arbitrary expression that has a non negative integer value should be inside the brackets.
* You can use this syntax to both read and write the value of an element of an array. Thus, the following are all legal JavaScript statements:

Text

Description automatically generated

**Adding and Deleting Array Elements**

Text

Description automatically generated

**Note that** using delete on an array element does **not alter the length property** and **does not shift elements with higher indexes down to fill in the gap** that is left by the deleted property.

**Iterating Arrays**

* As of ES 6, the easiest way to loop through each of the elements of an array (or any iterable object) is with the for/of loop

Graphical user interface, text, application

Description automatically generated

1. **destructuring assignment**

* The **destructuring assignment** syntax is a JavaScript expression that makes it possible to **unpack values from arrays, or properties from objects, into distinct variables.**

A picture containing text

Description automatically generated

1. **Iterating Arrays (with index of each array element)**

* If you want to use **a for/of loop** for an array and **need to know the index** of each array element, use the entries() method of the array, along with destructuring assignment.

Graphical user interface, text, application

Description automatically generated

The **entries()** method returns a new **Array Iterator** object that contains the key/value pairs for each index in the array

**Array Methods**

**Array Iterator Methods:** Iterator methods loop over the elements of an array

* **forEach()** iterates through an array, invoking a function you specify for each element
* **map()** passes each element of the array on which it is invoked to the function you specify and returns an array containing the values returned by your function.
* **filter()** returns an array containing a subset of the elements of the array on which it is invoked.
* **find()** returns the matching element, If no matching element is found, find() returns undefined
* **findIndex()** returns the index of the matching element. If no matching element is found, If no matching element is found, find() returns -1
* **every() and some()** they apply a predicate function you specify to the elements of the array, then return true or false.
* **reduce()** combine the elements of an array, using the function you specify, to produce a single value.

**Stack and queue methods** add and remove array elements to and from the beginning and the end of an array.

* **push()** appends one or more new elements to the end of an array and returns the new length of the array.

**pop()** deletes the last element of an array, decrements the array length, and returns the value that it removed.

* **unshift()** adds an element or elements to the beginning of the array, shifts the existing array elements up to higher indexes to make room, and returns the new length of the array.
* **shift()** removes and returns the first element of the array, shifting all subsequent elements down one place to occupy the newly vacant space at the start of the array.

**Subarray methods** are for extracting, deleting, inserting, filling, and copying contiguous regions of a larger array.

* **slice()** returns a slice, or subarray, of the specified array. Its two arguments specify the start and end of the slice to be returned.
* **splice()** a general-purpose method for inserting or removing elements from an array.
* **fill()** sets the elements of an array, or a slice of an array, to a specified value. It mutates the array it is called on, and also returns the modified array:

**Searching and sorting methods** are for locating elements within an array and for sorting the elements of an array.

* **indexOf()** search an array for an element with a specified value and return the index of the first such element found, or -1 if none is found.
* **includes()** takes a single argument and returns true if the array contains that value or false otherwise. It does not tell you the index of the value, only whether it exists.
* **sort()** sorts the elements of an array in place and returns the sorted array.
* **reverse()** reverses the order of the elements of an array and returns the reversed array.

**Array to String Conversions**

* **join()** converts all the elements of an array to strings and concatenates them, returning the resulting string.

**Function Expressions**

**Function expression**

A picture containing text

Description automatically generated

**Named function expression**

Graphical user interface, text, application

Description automatically generatedText

Description automatically generated with low confidence

Text

Description automatically generated**Arrow Function Expressions** - is a compact alternativeto a traditional function expression but is limitedand can't be used in all situations.

A picture containing calendar

Description automatically generated**Comparing traditional functions to arrow functions**

Text

Description automatically generated Text

Description automatically generated