STUDY OF JAVASCRIPT PROGRAMMING LEARNING ASSISTANT SYSTEM (JSPLAS)

JavaScript Basic Concepts

Funabiki Laboraty

Department of Electrical and Communication Engineering

Okayama University

JavaScript Programming Learning Assistant System (JSPLAS)

Correspondence Between Each Topic and Related VTPs

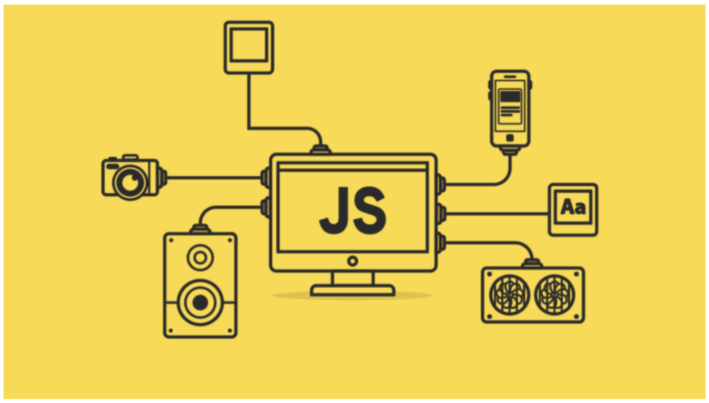
- Correspondence Between Each Topic and Related VTPs

※ Click the link to go reference pages directly.

JavaScript Concepts	**Related VTPs Version**	**Related Problem Numbers**
Basic Syntax	VTP_P1	1 to 32
JavaScript Data Types	VTP_P1	1
JavaScript Operators	VTP_P1	2,3
JavaScript Objects	VTP_P1	5
JavaScript Functions	VTP_P1	4
JavaScript Strings	VTP_P1	6,7,8,9,10,11,12,13,14,15
JavaScript Conditional Code (If/Else/Switch)	VTP_P1	25,26,27,28
JavaScript Loops	VTP_P1	29,30,31,32
JavaScript Arrays	VTP_P1	16 to 32

JavaScript Concepts	**Related VTPs Version**	**Related Problem Numbers**
JavaScript Data Types	VTP_P2	1,2,10
JavaScript Operators	VTP_P1	3,4,5
JavaScript operator precedence and associativity chart	VTP_P2	6
JavaScript RegExp Object Methods	VTP_P2	7
JavaScript Functions	VTP_P2	9
JavaScript Global Functions	VTP_P2	12,13,14,15,16,17
JavaScript Strings	VTP_P2	8
JavaScript Conditional Code (If/Else/Switch)	VTP_P2	25,26,27,28
JavaScript Loops	VTP_P2	11
JavaScript Arrays	VTP_P2	18
JavaScript Dates	VTP_P2	19
The difference of Var, Let and Const	VTP_P2	20,21,22,23,24
Arrow function expressions	VTP_P2	25

- What is JavaScript



JavaScript often abbreviated as JS, is a high-level, interpreted programming language that conforms to the ECMAScript specification. It is a programming language that is characterized as dynamic, weakly typed, prototype-based, and multi-paradigm. It is also a programming language used primarily by Web browsers to create a dynamic and interactive experience for the user. Most of the functions and applications that make the Internet indispensable to modern life are coded in some form of JavaScript. It is a tool for developers to add interactivity to websites.

JavaScript is a simple and easy-to-learn programming language as compared to other languages such as C++, Ruby, and Python. It is a high-level, interpreted language that can easily be embedded with languages like HTML. It was developed by Netscape Communications Corporation, Mozilla Foundation, and ECMA International. Brendan Erich is known as the creator or designer of the JavaScript programming language

- Why is JavaScript Important?

Since JavaScript is released, it has become the most popular programming language for web development today. JavaScript is currently used by 94.5% of all websites and, despite originally being designed as a client-side language, JavaScript has now made its way to the server-side of websites, mobile devices, and desktop. Developers can also create web pages which work well across various browsers, platforms, and devices by combining JavaScript, HTML5, and CSS3.

One of the good things about JavaScript is that it is supported by multiple web browsers like Google Chrome, Internet Explorer, Firefox, Safari, and Opera etc. Hence the users can access the web applications on any web browser of their choice easily. They can just enable the JavaScript language if it is disabled and can enjoy all the functionality of the site. JavaScript Frameworks and libraries make it easy for web developers to build large, JavaScript-based web applications. JavaScript also has lots of libraries which can be used as per the requirements.

JavaScript code can be written easily without using any program or tool. Even a notepad, it can be used for creating JavaScript Programming. So, most of the coding editors can be used for JavaScript for debugging. JavaScript is an interpreted programming language still it simplifies development of complex web applications by allowing the developers to simplify the application's composition. The developers can use JavaScript libraries to create DOM boundaries.

The most popular Search Engine, Google has started the Accelerated Mobile Pages (AMP) project to create websites which deliver a richer user experience across various mobile devices by using JavaScript for optimizing websites for mobile devices for AMP.

JavaScript is very useful for not only building a highly interactive web application, but also for helping to enhance the speed, performance, functionality, usability, and features of the application without any hassles. That's why, most of the developers and programmers are using JavaScript to make web applications deliver best and optimal user experience across various devices, browsers, and operating systems. Developers must learn about various JavaScript libraries, frameworks, and tools and combine multiple libraries and frameworks to use the JavaScript as per the projects' requirements.

・ Special Features of the JavaScript

JavaScript is one of the most popular languages which includes numerous features when it comes to web development. JavaScript has so many special features as very useful and popular web programming language, but the following will be the most popular features.

- Light Weight Scripting Language
- Dynamic Typing
- Object-Oriented Programming Support
- Functional Style
- Platform Independent
- Prototype-based
- Interpreted Language
- Async Processing
- Client-Side Validation
- More control in the browser

Correspondence Between Each Topic and Related VTPs

・ <u>What are the differences and similarities between Java and JavaScript?</u>

Java is an object-oriented programming language and have a virtual machine platform that allows to be creating compiled programs that run on nearly every platform. Java promised, “Write Once, Run Anywhere”.

JavaScript is a lightweight programming language (“scripting language”) and used to make web pages interactive. It can insert dynamic text into HTML. JavaScript is also known as browser’s language. JavaScript (JS) is not similar or related to Java. Both the languages have a C like a syntax and are widely used in client-side Web applications, but there are few similarities only.

Though the name of Java and JavaScript is very similar, the usage of both is obviously different. Only some similarities will be between Java and JavaScript. Both Java and JavaScript have many libraries and frameworks. These frameworks and libraries help developers to reduce coding time significantly. Likewise, the developers also have the option to reuse the code-shared by communities to avoid writing any additional code. The libraries and frameworks further contribute to making web technologies popular and current.

For example, Both JavaScript and Java:
use {and} as code block delimiters
use; to end statements
have a Math library e.g. Math. Pow
have if, do...while (pretty much same syntax)
have return statements

・ <u>Differences between Java and JavaScript</u>

1. <u>OOPS</u>
   Java: Java is an object-oriented programming language. It uses objects to perform actions based on relations between objects.
   JavaScript: JavaScript is an object-oriented scripting language. It uses the objects to perform actions similar in Java.

2. <u>Platform</u>
   Java: Java applications and programs run in Java Virtual Machine (JVM) which required installing JDK and JRE on a system.
   JavaScript: JavaScript applications run on a web browser and no need of any initial setup.

3. Mobile Application

   Java: Old Mobile applications are mostly written in Java and Smartphone platforms like Symbian and Android also support Java.

   JavaScript: Using JavaScript we can develop mobile applications but there are few limitations as we need to use third-party tools like phone gap to convert it to native code which mobile OS platform can execute.

4. Syntax

   Java: The syntax of Java is similar to C/C++ programming language. It uses classes and objects.

   JavaScript: The syntax of JavaScript is similar to C language, but it uses the naming conventions similar to Java.

5. Compilation

   Java: Java programs are compiled and interpreted as it is a scripting language.

   JavaScript: JavaScript is only interpreted as it is a scripting language or a plain text code.

6. Learning Curve

   Java: Java has various online forums, documentation and community support. You can learn this language to build various applications.

   JavaScript: JavaScript also has extensive documentation and online resources. You can learn JavaScript to build web applications and websites.

7. Scope

   Java: Java uses block-based scoping. In this, the variable goes out of scope once the control comes out of a block.

   JavaScript: JavaScript uses function-based scoping as the variable can be accessed in the function.

8. Support

   Java: Java is supported by almost all the operating systems.

   JavaScript: JavaScript supported by almost all the web browsers that come with different operating systems.

- Basic Syntax

1. A simple variable declaration

```
var greet = 'Hello world' ;
var n = 10 ;
```

Any data types can be declared by using with var. It can be automatically recognized as assigned values.

2. Comments usage

```
/* How to assign variables
   Data types can be automatically changed */


var greet = 'Hello world' ;              //String variable assignment
var n = 10 ;                             //Number variable assignment
```

/ *…. */

When we need to describe comments with multiple line, this syntax can be used.

//…..

When we need to describe comments with single line or short message, this syntax can be used.

3. Output

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

```
var greet = 'Hello world' ;
console.log(greet);
```

Remarks: We will use only console.log() in basic grammar part I and part II.

4. JavaScript Statements

```
var a, b, c;          //Declare 3 variables
a = 5;                // Assign the value 5 to a
```

```
b = 6;                  // Assign the value 6 to b
c = a + b;              // Assign the sum of a and b to c
```

JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments. JavaScript programs (and JavaScript statements) are often called JavaScript code.

5. Semicolons

```
var greet = 'Hello world' ;                 //String variable assignment
```

Semicolons separate JavaScript statements. Add a semicolon at the end of each executable statement:

- JavaScript Data Types

Data types basically specify what kind of data can be stored and manipulated within a program.

There are six basic data types in JavaScript which can be divided into three main categories: primitive (or primary), composite (or reference), and special data types.

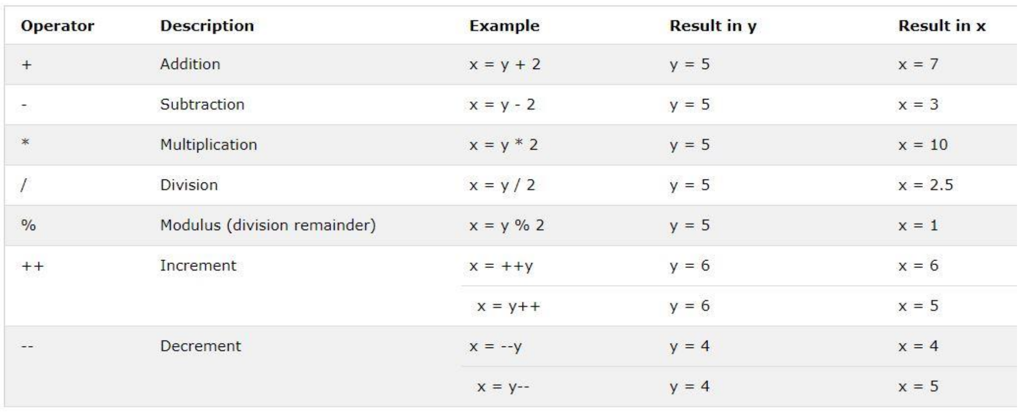
- String, Number, and Boolean are primitive data types.
- Object, Array, and Function (which are all types of objects) are composite data types.
- Undefined and Null are special data types.

Primitive data types can hold only one value at a time, whereas composite data types can hold collections of values and more complex entities.

```
var length = 16;                                 // Number
var lastName = "Johnson";                        // String
var x = {firstName:"John", lastName:"Doe"};      // Object
var length = x;                                  // Undefined
var inProgress = true;                           // Boolean
var obj = null;                                  // Null
```

- JavaScript Operators

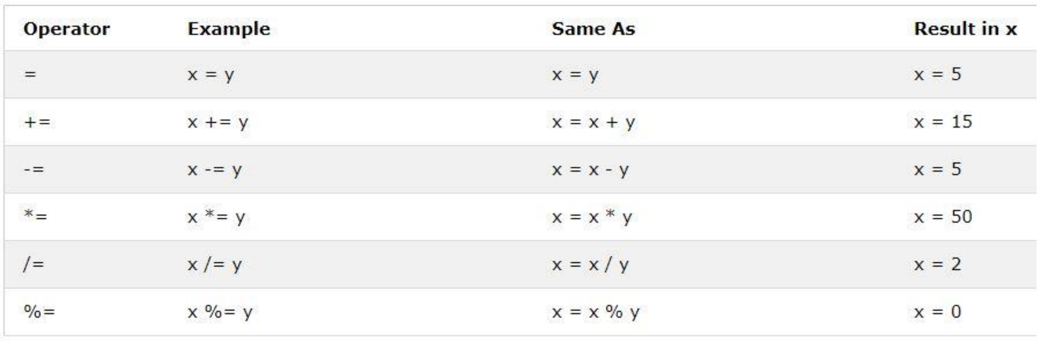
1. Arithmetic Operators



Operator	Description	Example	Result in y	Result in x
+	Addition	x = y + 2	y = 5	x = 7
-	Subtraction	x = y - 2	y = 5	x = 3
*	Multiplication	x = y * 2	y = 5	x = 10
/	Division	x = y / 2	y = 5	x = 2.5
%	Modulus (division remainder)	x = y % 2	y = 5	x = 1
++	Increment	x = ++y	y = 6	x = 6
		x = y++	y = 6	x = 5
--	Decrement	x = --y	y = 4	x = 4
		x = y--	y = 4	x = 5

Arithmetic operators are used to perform arithmetic between variables and/or values.

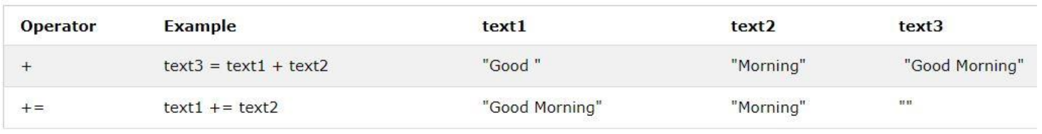
2. Assignment Operators



Operator	Example	Same As	Result in x
=	x = y	x = y	x = 5
+=	x += y	x = x + y	x = 15
-=	x -= y	x = x - y	x = 5
*=	x *= y	x = x * y	x = 50
/=	x /= y	x = x / y	x = 2
%=	x %= y	x = x % y	x = 0

Assignment operators are used to assign values to JavaScript variables.

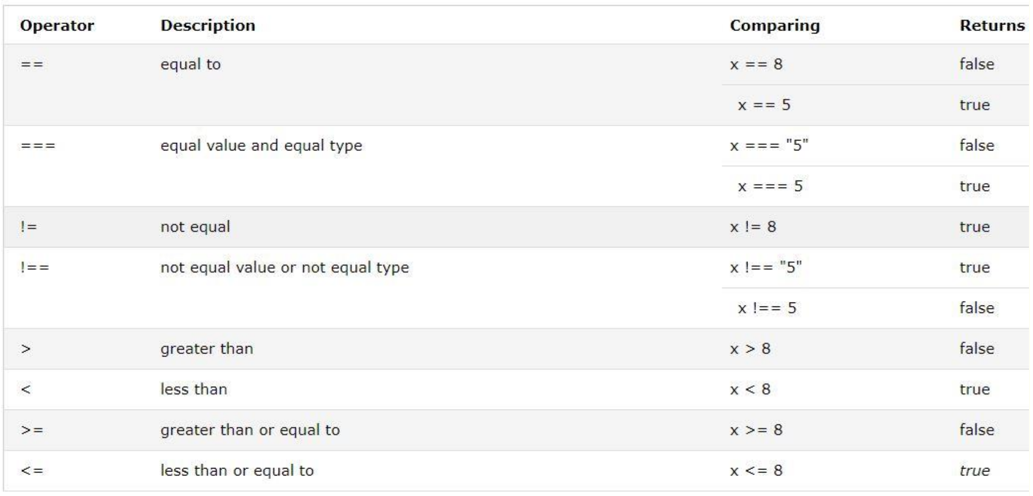
3. String Operators



Operator	Example	text1	text2	text3
+	text3 = text1 + text2	"Good "	"Morning"	"Good Morning"
+=	text1 += text2	"Good Morning"	"Morning"	""

The + operator, and the += operator can also be used to concatenate (add) strings.

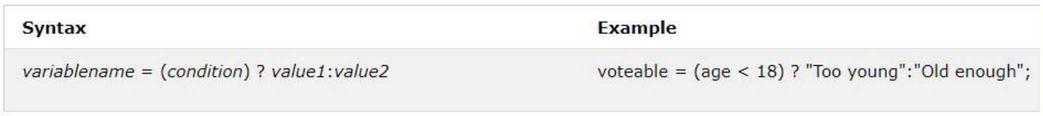
4. Comparison Operators



Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
===	equal value and equal type	x === "5"	false
		x === 5	true
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== "5"	true
		x !== 5	false
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	*true*

Comparison operators are used in logical statements to determine equality or difference between variables or values.

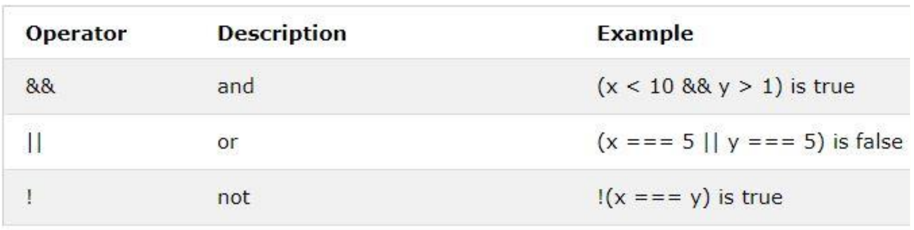
5. Conditional (Ternary) Operator



Syntax	Example
variablename = (*condition*) ? *value1*:*value2*	voteable = (age < 18) ? "Too young":"Old enough";

The conditional operator assigns a value to a variable based on a condition.

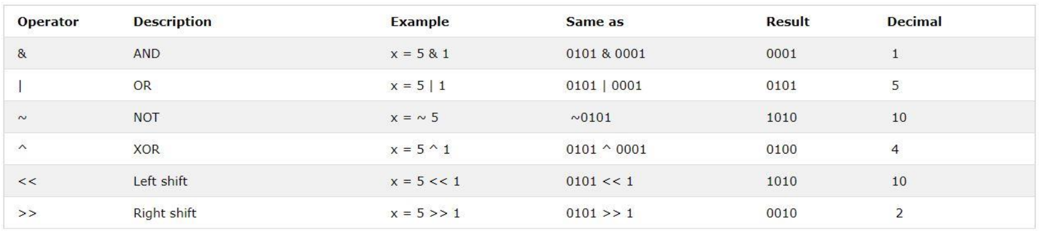
6. Logical Operators



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

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

7. Bitwise Operators



Operator	Description	Example	Same as	Result	Decimal
&	AND	x = 5 & 1	0101 & 0001	0001	1
|	OR	x = 5 | 1	0101 | 0001	0101	5
~	NOT	x = ~ 5	~0101	1010	10
^	XOR	x = 5 ^ 1	0101 ^ 0001	0100	4
<<	Left shift	x = 5 << 1	0101 << 1	1010	10
>>	Right shift	x = 5 >> 1	0101 >> 1	0010	2

Bit operators work on 32 bits numbers. Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

8. The typeof Operator

```
typeof "John”;                  //Returns string
typeof 3.14;                    // Returns number
typeof NaN;                     // Returns number
typeof false;                   // Returns boolean
typeof [1, 2, 3, 4];            // Returns object
typeof {name:'John', age:34};   // Returns object
typeof new Date();              // Returns object
typeof function () {};          // Returns function
typeof myCar;                   // Returns undefined (if myCar is not declared)
typeof null;                    // Returns object
```

The **typeof** operator returns the type of a variable, object, function or expression:

Note: The data type of NaN is number
The data type of an array is object
The data type of a date is object
The data type of null is object
The data type of an undefined variable is undefined

9. The delete Operator

```
var person = {firstName:”John”, lastName:”Smith”, age:50, sex:”male”};
delete person.age;    // or delete person[“age”];
```

The delete operator deletes both the value of the property and the property itself. After deletion, the property cannot be used before it is added back again. The delete operator is designed to be used on object properties. It has no effect on variables or functions. [Note: The delete operator should not be used on predefined JavaScript object properties. It can crash your application.]

10. The In Operator

```
// Arrays
var cars = ["Toyoya", "Mazda", "Suzuki”];
"Saab" in cars          // Returns false (specify the index number instead of value)
0 in cars               // Returns true
```

Correspondence Between Each Topic and Related VTPs

```
1 in cars                // Returns true
4 in cars                // Returns false (does not exist)
"length" in cars         // Returns true (length is an Array property)

// Objects
var person = {firstName:"John", lastName:"Doe", age:50};
"firstName" in person          // Returns true
"age" in person                // Returns true

// Predefined objects
"PI" in Math             // Returns true
"NaN" in Number          // Returns true
"length" in String       // Returns true
```

The in operator returns true if the specified property is in the specified object, otherwise false:

11. The instanceof Operator

```
// Arrays
var cars = ["Toyoya", "Mazda", "Suzuki”];
cars instanceof Array;         //Returns true
cars instanceof Object;        //Returns true
cars instanceof String;        //Returns false
cars instanceof Number;        //Returns false
```

The instanceof operator returns true if the specified object is an instance of the specified object:

- JavaScript operator precedence and associativity chart

Operator	Precedence	Associativity
• . (dot property reference) • [] (bracket property reference) • new • ()	1	Left to right
• ++ • --	2	**Right to left**
• + (unary +, NOT the addition sign) • - (unary negate -, NOT subtract sign) • ~ (bitwise not) • ! (not) • delete • typeof • void	3	**Right to left**
• * • / • % • + (addition) • - (subtraction)	4	Left to right
• << • >> • >>>	5	Left to right
• < • <= • > • >= • instanceof • in	6	Left to right
• &	7	Left to right
• ^	8	Left to right
• |	9	Left to right
• &&	10	Left to right

Correspondence Between Each Topic and Related VTPs

• ||	11	Left to right
• ?:	12	**Right to left**
• =	13	**Right to left**
• *= • /= • %= • += • -= • <<== • >>== • >>>= • &= • ^= • |=	14	**Right to left**
• , (comma)	15	Left to right

- JavaScript Objects

  Object is a non-primitive data type in JavaScript. It is like any other variable, the only difference is that an object holds multiple values in terms of properties and methods. Properties can hold values of primitive data types and methods are functions.
  Object properties and methods can be accessed using dot notation or [ ] bracket.
  An object is passed by reference from one function to another.
  An object can include another object as a property.

  Creating Objects in JavaScript

  There are 2 ways to create objects.

  1. Object literal
  2. Object constructor

  JavaScript Object by object literal

```
var person = {
    firstName: "John",
    lastName: "Smith",
    age: 25,
    getFullName: function () {
        return this.firstName + ' ' + this.lastName
        }
};
```

  JavaScript Object by Object constructor

```
var person = new Object();

person.firstName = "John";
person["lastName"] = "Smith";
person.age = 25;
person.getFullName = function () {
        return this.firstName + ' ' + this.lastName;
};
```

・ JavaScript RegExp Object Methods

(1) test() Method

The test() method tests for a match in a string.

This method returns true if it finds a match, otherwise it returns false.

- JavaScript Functions

  JavaScript provides functions similar to most of the scripting and programming languages. In JavaScript, a function allows you to define a block of code, give it a name and then execute it as many times as you want.

  A JavaScript function can be defined using function keyword.

  JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

  Advantage of JavaScript function

  There are mainly two advantages of JavaScript functions.

  1. Code reusability: We can call a function several times so it save coding.
  2. Less coding: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

  JavaScript Function Syntax

  JavaScript Functions can have 0 or more arguments.

```
//defining a function
function functionName([arg1, arg2, ...argN])
{
        //code to be executed
}
//calling a function
<functionName >([arg1, arg2, ...argN]);
```

  Function Parameters

  A function can have one or more parameters, which will be supplied by the calling code and can be used inside a function. JavaScript is a dynamic type scripting language, so a function parameter can have value of any data type.

```
function ShowMessage(firstName, lastName) {
        console.log("Hello " + firstName + " " + lastName);
}
ShowMessage("Steve", "Jobs");
ShowMessage("Bill", "Gates");
ShowMessage(100, 200);
```

Argument Objects

All the functions in JavaScript can use arguments object by default. An arguments object includes value of each parameter.

The arguments object is an array like object. You can access its values using index similar to array. However, it does not support array methods.

```
function ShowMessage(firstName, lastName) {
        console.log("Hello " + arguments[0] + " " + arguments[1]);
}
ShowMessage("Steve", "Jobs");
ShowMessage("Bill", "Gates");
ShowMessage(100, 200);
```

Return Values

A function can return zero or one value using return keyword.

```
function Sum(val1, val2)
{
        return val1 + val2;
};
var result = Sum(10,20);            // returns 30

function Multiply(val1, val2)
{
        console.log( val1 * val2);
};
result = Multiply(10,20);            // undefined
```

Function Expressions

JavaScript allows us to assign a function to a variable and then use that variable as a function. It is called function expression.

```
var add = function sum(val1, val2) {
        return val1 + val2;
};
var result1 = add(10,20);
var result2 = sum(10,20);            // not valid
```

Correspondence Between Each Topic and Related VTPs

Anonymous Function

JavaScript allows us to define a function without any name. This unnamed function is called anonymous function. Anonymous function must be assigned to a variable.

```
var showMessage = function ()
{
        alert("Hello World!");
};
showMessage();

var sayHello = function (firstName) {
        alert("Hello " + firstName);
};
showMessage();
sayHello("Bill");
```

Nested Function

In JavaScript, a function can have one or more inner functions. These nested functions are in the scope of outer function. Inner function can access variables and parameters of outer function. However, outer function cannot access variables defined inside inner functions.

```
function ShowMessage(firstName)
{
        function SayHello()
        {
                alert("Hello " + firstName);
        }
        return SayHello();
}
ShowMessage("Steve");
```

- JavaScript Global Functions

The JavaScript global properties and functions can be used with all the built-in JavaScript objects.

JavaScript Global Properties

(1) Infinity

A numeric value that represents positive/negative infinity

(2) NaN

"Not-a-Number" value

(3) undefined

Indicates that a variable has not been assigned a value

JavaScript Global Functions

(1) decodeURI()

Decodes a URI

(2) decodeURIComponent()

Decodes a URI component

(3) encodeURI()

Encodes a URI

(4) encodeURIComponent()

Encodes a URI component

(5) escape()

Deprecated in version 1.5.

Use encodeURI() or encodeURIComponent() instead

(6) eval()

Evaluates a string and executes it as if it was script code

(7) isFinite()

Determines whether a value is a finite, legal number

(8) isNaN()

Determines whether a value is an illegal number

(9) Number()

Converts an object's value to a number

(10) parseFloat()

Parses a string and returns a floating point number

(11) parseInt()

Parses a string and returns an integer

Correspondence Between Each Topic and Related VTPs

(12) String()

Converts an object's value to a string

(13) unescape()

Deprecated in version 1.5.

Use decodeURI() or decodeURIComponent() instead

- JavaScript Strings

  A JavaScript string stores a series of characters like "Hello World".

  A string can be any text inside double or single quotes:

```
var greet = “Hello”;
var Greet = ‘Hello World’;
※ The javaScript is case-sensitive.
```

  String indexes are zero-based: The first character is in position 0, the second in 1, and so on.

String Properties

1. constructor

   Returns the string's constructor function
2. length

   Returns the length of a string
3. prototype

   Allows you to add properties and methods to an object

String Methods

1. charAt()

   Returns the character at the specified index (position)
2. charCodeAt()

   Returns the Unicode of the character at the specified index
3. concat()

   Joins two or more strings, and returns a new joined strings
4. endsWith()

   Checks whether a string ends with specified string/characters
5. fromCharCode()

   Converts Unicode values to characters
6. includes()

   Checks whether a string contains the specified string/characters
7. indexOf()

   Returns the position of the first found occurrence of a specified value in a string
8. lastIndexOf()

   Returns the position of the last found occurrence of a specified value in a string

Correspondence Between Each Topic and Related VTPs

9. localeCompare()

   Compares two strings in the current locale

10. match()

    Searches a string for a match against a regular expression, and returns the matches

11. repeat()

    Returns a new string with a specified number of copies of an existing string

12. replace()

    Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced

13. search()

    Searches a string for a specified value, or regular expression, and returns the position of the match

14. slice()

    Extracts a part of a string and returns a new string

15. split()

    Splits a string into an array of substrings

16. startsWith()

    Checks whether a string begins with specified characters

17. substr()

    Extracts the characters from a string, beginning at a specified start position, and through the specified number of character

18. substring()

    Extracts the characters from a string, between two specified indices

19. toLocaleLowerCase()

    Converts a string to lowercase letters, according to the host's locale

20. toLocaleUpperCase()

    Converts a string to uppercase letters, according to the host's locale

21. toLowerCase()

    Converts a string to lowercase letters

22. toString()

    Returns the value of a String object

23. toUpperCase()

    Converts a string to uppercase letters

24. trim()

    Removes whitespace from both ends of a string

Correspondence Between Each Topic and Related VTPs

25. valueOf() Returns the primitive value of a String object

- JavaScript Conditional Code (If/Else/Switch)

The if/else statement executes a block of code if a specified condition is true. If the condition is false, another block of code can be executed.

The if/else statement is a part of JavaScript's "Conditional" Statements, which are used to perform different actions based on different conditions.

In JavaScript we have the following conditional statements:

1. Use if to specify a block of code to be executed, if a specified condition is true
2. Use else to specify a block of code to be executed, if the same condition is false
3. Use else if to specify a new condition to test, if the first condition is false
4. Use switch to select one of many blocks of code to be executed

1. If …

   The if statement specifies a block of code to be executed if a condition is true:

   #Syntax

```
  if (condition)
  {
          // block of code to be executed if the condition is true
  }
```

2. Else….

   The else statement specifies a block of code to be executed if the condition is false:

   #Syntax

```
  if (condition)
  {
         // block of code to be executed if the condition is true
  } else {
          // block of code to be executed if the condition is false
  }
```

3. Else if …

   The else if statement specifies a new condition if the first condition is false:

   #Syntax

```
if (condition1)
{
```

```
            // block of code to be executed if condition1 is true
  } else if (condition2) {
            // block of code to be executed if the condition1 is false and condition2
is true
} else {
            // block of code to be executed if the condition1 is false and condition2
is false
}
```

4. Switch

   Rather than using a series of if/else if/else blocks, sometimes it can be useful to use a switch statement instead. [Definition: Switch statements look at the value of a variable or expression and run different blocks of code depending on the value.]

   #Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

# Example for Switch Statement

```
var day;
switch (new Date().getDay()) {
case 0:
    day = "Sunday";
   break;
case 1:
     day = "Monday";
   break;
case 2:
        day = "Tuesday";
      break;
    case 3:
        day = "Wednesday";
      break;
    case 4:
        day = "Thursday";
      break;
     case 5:
        day = "Friday";
       break;
     case 6:
       day = "Saturday";
    }
    console.log(day);
```

- JavaScript Loops

Loops are used in JavaScript to perform repeated tasks based on a condition. Conditions typically return true or false when analyzed. A loop will continue running until the defined condition returns false.

The three most common types of loops are:

1. for
2. while
3. do while

1. for loop Syntax

```
for ([initialization]); [condition]; [final-expression])
{
      // statement
}
```

For in loop Syntax

Syntax

```
for (variable in object)
{
        // statement
 }
```

for...of loop Syntax

```
for (variable of object)
{
        // statement
}
```

2. while loop Syntax

```
while (condition)
{
        /* A statement that is executed as long as the condition evaluates to true. */
        //statement(s);
}
```

3. do while loop Syntax

```
do {
        /* A statement that is executed as long as the condition evaluates to true. */
        //statement(s);
} while (*condition*);
```

- JavaScript Arrays

  The Array object is used to store multiple values in a single variable:

```
var cars = ["Toyota", "Mazda", "Suzuki"];
```

  Array indexes are zero-based: The first element in the array is 0, the second is 1, and so on.

Array Properties

1. constructor
   Returns the function that created the Array object's prototype
2. length
   Sets or returns the number of elements in an array
3. prototype
   Allows you to add properties and methods to an Array object

Array Methods

1. concat()
   Joins two or more arrays, and returns a copy of the joined arrays
2. copyWithin()
   Copies array elements within the array, to and from specified positions
3. entries()
   Returns a key/value pair Array Iteration Object
4. every()
   Checks if every element in an array pass a test
5. fill()
   Fill the elements in an array with a static value
6. filter()
   Creates a new array with every element in an array that pass a test
7. find()
   Returns the value of the first element in an array that pass a test
8. findIndex()
   Returns the index of the first element in an array that pass a test
9. forEach()
   Calls a function for each array element
10. from()
    Creates an array from an object
11. includes()
    Check if an array contains the specified element

Correspondence Between Each Topic and Related VTPs

12. indexOf()

    Search the array for an element and returns its position
13. isArray()

    Checks whether an object is an array
14. join()

    Joins all elements of an array into a string
15. keys()

    Returns a Array Iteration Object, containing the keys of the original array
16. lastIndexOf()

    Search the array for an element, starting at the end, and returns its position
17. map()

    Creates a new array with the result of calling a function for each array element
18. pop()

    Removes the last element of an array, and returns that element
19. push()

    Adds new elements to the end of an array, and returns the new length
20. reduce()

    Reduce the values of an array to a single value (going left-to-right)
21. reduceRight()

    Reduce the values of an array to a single value (going right-to-left)
22. reverse()

    Reverses the order of the elements in an array
23. shift()

    Removes the first element of an array, and returns that element
24. slice()

    Selects a part of an array, and returns the new array
25. some()

    Checks if any of the elements in an array pass a test
26. sort()

    Sorts the elements of an array
27. splice()

    Adds/Removes elements from an array
28. toString()

    Converts an array to a string, and returns the result
29. unshift()

Adds new elements to the beginning of an array, and returns the new length

30. valueOf()

Returns the primitive value of an array

A simple array

```
var myArray = [ 'hello', 'world' ];
```

Accessing array items by index

```
var myArray = [ 'hello', 'world', 'foo', 'bar' ];
console.log(myArray[3]);     // logs 'bar'
```

Testing the size of an array

```
var myArray = [ 'hello', 'world' ];
console.log(myArray.length);     // logs 2
```

Changing the value of an array item

```
var myArray = [ 'hello', 'world' ];
myArray[1] = 'changed';
```

While it’s possible to change the value of an array item as shown in “Changing the value of an array item”, it’s generally not advised.

Adding elements to an array

```
var myArray = [ 'hello', 'world' ];
myArray.push('new');
```

Working with arrays

```
var myArray = [ 'h', 'e', 'l', 'l', 'o' ];
```

Correspondence Between Each Topic and Related VTPs

```
var myString = myArray.join('');     // 'hello'
var mySplit = myString.split('');   // [ 'h', 'e', 'l', 'l', 'o' ]
```

- JavaScript Dates

  Date Object

  The Date object is used to work with dates and times.

  Date objects are created with new Date().

  There are four ways of instantiating a date:

```
var d = new Date();
var d = new Date(milliseconds);
var d = new Date(dateString);
var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

  Date Object Properties

(1) Constructor

   Returns the function that created the Date object's prototype

(2) prototype

   Allows you to add properties and methods to an object

  Date Object Methods

(1) getDate()

   Returns the day of the month (from 1-31)

(2) getDay()

   Returns the day of the week (from 0-6)

(3) getFullYear()

   Returns the year

(4) getHours()

   Returns the hour (from 0-23)

(5) getMilliseconds()

   Returns the milliseconds (from 0-999)

(6) getMinutes()

   Returns the minutes (from 0-59)

(7) getMonth()

Correspondence Between Each Topic and Related VTPs

Returns the month (from 0-11)

(8) getSeconds()

Returns the seconds (from 0-59)

(9) getTime()

Returns the number of milliseconds since midnight Jan 1 1970,
and a specified date

(10) getTimezoneOffset()

Returns the time difference between UTC time and local time, in minutes

(11) getUTCDate()

Returns the day of the month, according to universal time (from 1-31)

(12) getUTCDay()

Returns the day of the week, according to universal time (from 0-6)

(13) getUTCFullYear()

Returns the year, according to universal time

(14) getUTCHours()

Returns the hour, according to universal time (from 0-23)

(15) getUTCMilliseconds()

Returns the milliseconds, according to universal time (from 0-999)

(16) getUTCMinutes()

Returns the minutes, according to universal time (from 0-59)

(17) getUTCMonth()

Returns the month, according to universal time (from 0-11)

(18) getUTCSeconds()

Returns the seconds, according to universal time (from 0-59)

(19) getYear()

Deprecated. Use the getFullYear() method instead

(20) now()

Returns the number of milliseconds since midnight Jan 1, 1970

(21) parse()

Parses a date string and returns the number of milliseconds
since January 1, 1970

(22) setDate()

Sets the day of the month of a date object

(23) setFullYear()

Sets the year of a date object

(24) setHours()

Sets the hour of a date object

(25) setMilliseconds()

Sets the milliseconds of a date object

(26) setMinutes()

Set the minutes of a date object

(27) setMonth()

Sets the month of a date object

(28) setSeconds()

Sets the seconds of a date object

(29) setTime()

Sets a date to a specified number of milliseconds after/before January 1, 1970

(30) setUTCDate()

Sets the day of the month of a date object, according to universal time

(31) setUTCFullYear()

Sets the year of a date object, according to universal time

(32) setUTCHours()

Sets the hour of a date object, according to universal time

(33) setUTCMilliseconds()

Sets the milliseconds of a date object, according to universal time

(34) setUTCMinutes()

Set the minutes of a date object, according to universal time

(35) setUTCMonth()

Sets the month of a date object, according to universal time

(36) setUTCSeconds()

Set the seconds of a date object, according to universal time

(37) setYear()

Deprecated. Use the setFullYear() method instead

(38) toDateString()

Converts the date portion of a Date object into a readable string

(39) toGMTString()

Deprecated. Use the toUTCString() method instead

(40) toISOString()

Returns the date as a string, using the ISO standard

(41) toJSON()

Returns the date as a string, formatted as a JSON date

Correspondence Between Each Topic and Related VTPs

(42) toLocaleDateString()

Returns the date portion of a Date object as a string, using locale conventions

(43) toLocaleTimeString()

Returns the time portion of a Date object as a string, using locale conventions

(44) toLocaleString()

Converts a Date object to a string, using locale conventions

(45) toString()

Converts a Date object to a string

(46) toTimeString()

Converts the time portion of a Date object to a string

(47) toUTCString()

Converts a Date object to a string, according to universal time

(48) UTC()

Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time

(49) valueOf()

Returns the primitive value of a Date object

- The difference of Var, Let and Const

  In Javascript variables can be defined using the keywords var, let or const.

```
var a = 10;
let b = 20;
const PI = 3.14;
```

var

The scope of a variable defined with the keyword “var” is limited to the “function” within which it is defined. If it is defined outside any function, the scope of the variable is global. var is “*function scoped*”.

Let

The scope of a variable defined with the keyword “let” or “const” is limited to the “block” defined by curly braces i.e. {}. “let” and “const” are “*block scoped*”.

Const

The scope of a variable defined with the keyword “const” is limited to the block defined by curly braces. However, if a variable is defined with keyword const, it cannot be reassigned. “const” cannot be re-assigned to a new value. However, it can be mutated.

Block scoped VS Function scoped

Block Scope

In JavaScript, a code block can be defined by using curly braces i.e {}.

Consider the following code that has 2 code blocks each delimited by {}.

```
{
  var a = 10;
  console.log(a);
} //block 1
{
  a++;
  console.log(a);
} //block 2
```

/* Since we are using "var a=10", scope of "a" is limited to the function within which it is defined. In this case it is within the global function scope */

In the above example, since we are using the keyword var to define the variable a, the scope of a is limited to the function within which it is defined. Since a is not defined within any function, the scope of the variable a is global, which means that a is recognized within block 2.

In effect if a variable is defined with keyword var, JavaScript does not recognize the {} as the scope delimiter. Instead the variable must be enclosed within a “function” to limit it’s scope to that function. Let us re-write the code above using the keyword let. The let keyword was introduced as part of ES6 syntax, as an alternative to var to define variables in JavaScript.

```
{
        let a = 10;
        console.log(a);
} //block 1
{
         a++;
         console.log(a);
} //block 2
/* Since we are using "let a=10", scope of "a" is limited to block 1 and "a" is not
recognized in block 2 */
```

Note that now when you run the code above you will get an error, variable a not recognized in block2. This is because we have defined the variable a using the keyword let, which limits the scope of variable a to the code block within which it was defined.

<u>Function Scope</u>

In JavaScript you limit the scope of a variable by defining it within a function. This is known as function scope.

The keyword var is function scoped i.e. it does not recognize curly brackets i.e. {}, as delimiters. Instead it recognizes the function body as the delimiter.

If we want to define a variable using var and prevent it from being defined in the global namespace you can re-write it by enclosing the code blocks within functions.

```
function block1() {
        var a=10;
        console.log(a);
} //function scope of block 1
function block2(){
        a++;
        console.log(a);
} //function scope of block 2
```

/* Since we have enclosed block1 and block2, within separate functions, the scope of "var a=10", is limited to block 1 and "a" is not recognized in block 2 */

The above code is in effect the same as if we were using let a=10 instead of var a=10. The scope of the variable a is limited to the function within which it is defined, and a is no longer in the global namespace.

Why would you chose “let” over “var”?

While programming in Javascript it is a good practice not to define variables as global variables. This is because it is possible to inadvertently modify the global variable from anywhere within the Javascript code. To prevent this one needs to ensure that the scope of the variables are limited to the code block within which they need to be executed.

In the past before keyword let was introduced as part of ES6, to circumvent the issue of variable scoping using var, programmers used the IIFE pattern to prevent the pollution of the global name space. However since the introduction of let, the IIFE pattern is no longer required, and the scope of the variable defined using let is limited to the code block within which it is defined.

Correspondence Between Each Topic and Related VTPs

Const

If a variable is defined using the const keyword, its scope is limited to the block scope. In addition the variable cannot be reassigned to a different value.

```
{
            const PI=3.14;
            console.log(PI);
} //block 1
{
            console.log(PI);
} //block 2
```

/* Since we are using "const PI=3.14", scope of "PI" is limited to block 1 and "PI" is not recognized in block 2 */

Note that it is important to understand that const does NOT mean that the value is fixed and immutable. This is a common misunderstanding amongst many JavaScript developers, and they incorrectly mentioned that a value defined by the const keyword is immutable (i.e. it cannot be changed).

In the following example we can show that the value of the variable defined within the const keyword is mutable, i.e. it can be changed.

```
{
            const a = [1,2,3];
            const b = {name: "hello"};
            a.push(4,5);        //mutating the value of constant "a"
            b.name="World"; //mutating the value of constant "b"

            console.log(a); //this will show [1,2,3,4,5]
            console.log(b); //this will show {name: "World"}
 }
```

/* This code will run without any errors, and shows that we CAN mutate the values that are defined by "const" */

However, note that these variables defined by const cannot be re-assigned.

```
{
          const name = "Mike";
          const PI = 3.14;
          const a = [1,2,3];
          const b = {name: "hello"};

          name="Joe";
          /*this will throw an error, since we are attempting to re-assign
"name" to a different value. */

          PI = PI + 1;
          /*this will throw an error, since we are attempting to re-assign PI to a
different value.*/

          a = [1,2,3,4,5];
          /*this will throw an error, since we are attempting to re-assign "a" to a
different value.*/

          b = {name: "hello"};
          /*this will throw an error, since we are attempting to re-assign "b" to a
different value. */
}
```

- Arrow function expressions
  An arrow function expression is a compact alternative to a traditional function expression but is limited and can't be used in all situations.

  Differences & Limitations:

  - Does not have its own bindings to this or super and should not be used as methods.
  - Does not have arguments, or new.target keywords.
  - Not suitable for call, apply and bind methods, which generally rely on establishing a scope.
  - Can’t be used as constructors.
  - Can’t use yield, within its body.

```
const materials = [
  'Hydrogen',
  'Helium',
  'Lithium',
  'Beryllium'
];

console.log(materials.map(material => material.length));

// expected output: Array [8, 6, 7, 9]
```

Comparing traditional functions to arrow functions

```
// Traditional Function
function (a) {
          return a + 100;
}

// Arrow Function Break Down
// 1. Remove the word "function" and place arrow between the argument and opening body bracket

(a) => {
```

Correspondence Between Each Topic and Related VTPs

```
		return a + 100;
}
// 2. Remove the body brackets and word "return" -- the return is implied.

(a) => a + 100;

// 3. Remove the argument parentheses

a => a + 100;
```

For example, if you have multiple arguments or no arguments, you'll need to re-introduce parentheses around the arguments:

```
// Traditional Function
function (a, b) {
		return a + b + 100;
}

// Arrow Function
(a, b) => a + b + 100;

// Traditional Function (no arguments)
let a = 4;
let b = 2;
function () {
		return a + b + 100;
}

// Arrow Function (no arguments)
let a = 4;
let b = 2;
() => a + b + 100;
```

Likewise, if the body requires additional lines of processing, you'll need to re-introduce brackets PLUS the "return" (arrow functions do not magically guess what or when you want to "return"):

```
// Traditional Function
function (a, b){
   let chuck = 42;
   return a + b + chuck;
}

// Arrow Function
(a, b) => {
   let chuck = 42;
   return a + b + chuck;
}
```

And finally, for named functions we treat arrow expressions like variables

```
// Traditional Function
function bob (a) {
            return a + 100;
}

// Arrow Function
let bob = a => a + 100;
```

Basic Syntax

One param. With simple expression return is not needed:

```
param => expression
```

Multiple params require parentheses. With simple expression return is not needed:

```
(param1, paramN) => expression
```

Multiline statements require body brackets and return:

```
param => {
  let a = 1;
  return a + param;
}
```

Multiple params require parentheses. Multiline statements require body brackets and return:

```
(param1, paramN) => {
    let a = 1;
    return a + param1 + paramN;
}
```

Advanced Syntax

To return an object literal expression requires parentheses around expression:

```
params => ({foo: "a"}) // returning the object {foo: "a"}
```

Rest parameters are supported:

```
(a, b, ...r) => expression
```

Default parameters are supported:

```
(a=400, b=20, c) => expression
```

Destructuring within params supported:

```
([a, b] = [10, 20]) => a + b;              // result is 30
({ a, b } = { a: 10, b: 20 }) => a + b;    // result is 30
```

- JavaScript Reversed Words

  JavaScript has a number of “reserved words,” or words that have special meaning in the language. You should avoid using these words in your code except when using them with their intended meaning.

abstract	boolean	break	byte
case	catch	char	class
const	continue	debugger	default
delete	do	double	else
enum	export	extends	final
finally	float	for	function
goto	if	implements	import
in	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	super	switch
synchronized	this	throw	throws
transient	try	typeof	var
void	volatile	while	with
long	native	new	package