Javascript

Note 1: order of function defining and referencing

code

```
<!DOCTYPE html>
<html>
<body>



<h2>Demo External JavaScript</h2>

<p id="demo">A Paragraph.</p>



<p>This example links to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>

<script src="myScript.js"></script>
<script>
myFunction()  # this will run the function in myScript.js
</script>

<script>
function myFunction(){
document.getElementById("demo").innerHTML = "this shall not be done";
}
</script>
<script>
myFunction()
</script>

</body>
</html>
```

Explanation

The order of writing function and calling function has significant impact on running result. If external script is referenced after self-defined function, then calling the same-name function will run the script's function (running the "latest updated method").

Note 2: HTML creating bottons, + JS-modifying style, src, etc.

Code:

```
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>

<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').style=”width:50px”">Turn on the
light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the
light</button>

</body>
</html>
```

Explain:

#botton: the type is a button; onclick=(what to do when button is clicked)>the text to display on button</button>

Note 3: Display

Code:

```
<html>
<body>
<p id=”demo”> this is a test </p>

<script>
document.getElementById(“demo”).innerHTML = “test done”;
document.write(“are you sure?”);
window.alert(“never try this”);
console.log(“logging might be helpful”);
</script>
</body>
</html>
```

Explain:

Document.write() simply display text to the place where it is called;
If it is used when a whole HTML is loaded, everything will be deleted except content written

Window.alert() will show an alert window showing content inside
Console.log() will display the text during debugging.

Note 4: declare variables and simple operations

Keyword	Description
var	Declares a variable
let	Declares a block variable
const	Declares a block constant
if	Marks a block of statements to be executed on a condition
switch	Marks a block of statements to be executed in different cases
for	Marks a block of statements to be executed in a loop
function	Declares a function
return	Exits a function
try	Implements error handling to a block of statements

Code:

Code:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be executed by a computer.</p>

<p id="demo"></p>

<script>
var alpha
```

```
let x = 5, y, z;  // Statement 1
y = 7;       // Statement 2
z = x + y;    // Statement 3
alpha = 12
/*
this is a multi line comment
*/
document.getElementById("demo").innerHTML =
"The value of z is " + z + "." + alpha;
</script>

</body>
</html>
```

Explain:

The table is useful, and highlighted parts are how to declare variables and how operations are done; "//" is for comments

Note 5: Var, Let, Const, Object, "typeof" function

Code:

```
<html>
<body>

<h2>Redeclaring a Variable Using var</h2>

<p id="demo"></p>
<p id="demostringplusint"></p>

<script>
var  x = 10;
// Here x is 10

let y = 12

{
var x = 2;
// Here x is 2
let y = 1

//let y = 13
//uncommenting will result in syntax error since y is declared twice
}

// Here x is 2
```

```
document.getElementById("demo").innerHTML = x;

//below shows integer will be converted to string when adding with a string
document.getElementById("demostringplusint").innerHTML="5" + 2 + 3;
document.getElementById("demostringplusint").innerHTML=2 + 3 + “5”;

// below is a demo for “typeof” function; notice the difference of using brackets or not.
document.getElementById("demostringplusint").innerHTML=typeof (2 + 3 + "5");
document.getElementById("demostringplusint").innerHTML=typeof 2 + 3 + "5";


</script>
<button onclick="document.getElementById('demo').innerHTML=y"> click me for y </button>
<button onclick="document.getElementById('demo').innerHTML=x"> click me for x after
</button>

<p id="demopi"></p>

<script>
try {
  const PI = 3.141592653589793;
  PI = 3.14;
  // below’s catch block shows error of assigning value to a const
}
catch (err) {
  document.getElementById("demopi").innerHTML = err;
}
</script>

<button onclick="y=14"> modify y </button>

<p>value of global y can be modified outside; however if y=12 is commented, this line will
declare y automatically and give value 14 </p>


<h2>JavaScript const</h2>

<p>You can NOT reassign a constant array:</p>

<p id="demoassignerror"></p>
<p id="display"></p>

<script>
const cars = ["Saab", "Volvo", "BMW"];
// declares an array
const vehicle = {type:"Fiat", model:"500", color:"white"};
// declares an object; similar to how button is created in HTML -> {properties and values, }
// both const array’s elements and const object’s elements are changable
```

```
try {
  //const cars = ["Saab", "Volvo", "BMW"];
  // declaring const in here instead of outside will result in below's button not working,
  //due to const is block scoped.
  document.getElementById("display").innerHTML=cars[0] + cars[1] + cars[2];
  cars = ["Toyota", "Volvo", "Audi"];
}
catch (err) {
  document.getElementById("demoassignerror").innerHTML = err;
}
</script>

<button onclick="cars[2] = 'Toyota'"> modify const array </button>
//const array's inner elements are changable
<button onclick="document.getElementById('display').innerHTML=cars[0] + cars[1] + cars[2]">
display const array </button>
<p> Above line only works when cars array is declared globally (not in any block)</p>

</body>
</html>
```

Explain:

Var: for global/general block variables;
Let: for declaring local/block variables; variables cannot be re-declared in same block;
Const: for declaring block immutable variables (cannot be reassigned); however constant "arrays" elements could be modified, but not re-assigned -> see reassigning cars array's error

Blue contents shows how array and object could be declared

Adding integer to a string always converts the integer to a string -> result in a string also.

```
document.getElementById("demostringplusint").innerHTML="5" + 2 + 3;
document.getElementById("demostringplusint").innerHTML=2 + 3 + "5";

// below is a demo of "typeof" function
document.getElementById("demostringplusint").innerHTML=typeof 2 + 3 + "5";
// prints          number35 -> it only considered "2"
document.getElementById("demostringplusint").innerHTML=typeof (2 + 3 + "5");
 // prints          string   -> need to use bracket for computation results
```

Last line will get value "55" instead of "523", due to evaluation of operations is from left to right. (thus first evaluate first two integers and result in a sum)

Note 6: operations, including assigning, adding, multiplying, and functions

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Operator	Description
&&	logical and
||	logical or
!	logical not

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

!==: either the value OR the type is not the same; -> 1==true -> true; 1===true -> false

Code:

```
<html>
<body>

<h2>JavaScript Functions</h2>

<p>Outside myFunction() carName is undefined.</p>

<p id="demo1"></p>

<p id="demo2">can this part be changed?</p>

<script>
function testingFunction(alpha){
   var bravo = alpha;
   document.getElementById("demo1").innerHTML=alpha;
   return 57;
}

var charlie = testingFunction("this shall be a test?");
document.getElementById("demo2").innerHTML=bravo;
document.getElementById("demo2").innerHTML=typeof bravo; //prints “undefined”, since “var bravo” is inside function block, and it’s block-scoped.

</script>
```

```
<p id="demo3">demonstrating return value</p>

<script>
document.getElementById("demo3").innerHTML=charlie;
</script>

</body>
</html>
```

Explain:

Function inputs don't need type specification. However return value of function requires it.
Never forget brackets {} for defining functions, and use "function" as header.
Pay attention to function's "block-scoped variables which can't be used outside function"

Note 7: Objects (specifics including: object function, creating object using function)

Code:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p>If you access an object method without (), it will return the function definition:</p>

<p id="demo_obj_func"></p>
<p id="func_out"></p>
<p id="undefined_input"></p>

<script>
// Create an object:
// defining an object can use any key words, depending on the scope of this object
let person = {
  firstName: "John",
  lastName : "Doe",
  id     : 5566,
  fullName : function(as) {
    return this.firstName + " " + this.lastName + as;
  } // it is possible to define a method for an object, using "this" as key word for object's
properties, same as in Java.  Note that no ";" after defining function!
};


function createPerson(first, last, id_input){
```

```
        // declarance key word depends on the scope, by "treating this function as 'global scope'".
        var new_person = {
   firstName: first,
   lastName: last,
   id: id_input,
   full_description: function(){
   return "My name is " + this.firstName + " " + this.lastName + ", and my id is " + this.id + ". It's nice to work with all of you. ";
   }  //
  };
   return new_person; // simply return this object as output
};

new_p = createPerson("Jone", "David", 12);

// Display data from the object:
document.getElementById("demo_obj_func").innerHTML = new_p.full_description();
// above shows how to call an object's method.

document.getElementById("undefined_input").innerHTML = person.fullName();
// if a required argument is not given to a function, the function will treat it as "undefined".

document.getElementById("func_out").innerHTML = person.fullName(" never mind");

</script>

</body>
</html>
```

Explain:

Defining an object with method is possible; for any object's property, use "this." key word. To call an object's method, must first instantiate an object and call as this object's method (instance.method).
Javascript also have "Classes", will be introduced later. (including privacy of variables and functions)

Note 8: Strings

Some methods of String requires usage of regular expression to achieve desired effect.

Methods:

Str.length -> returns number of characters in a string

Str.slice(start-index, Optional[end-index]) -> returns a string (input string not changed!!!)

if index is negative, will count in reversed (as in python); if no end-index, will slice remaining of string

Str.substring(start-index, Optional[end-index]) -> returns a string (input string not changed!!!)
cannot accept negative index

Str.substr(start-index, Optional[length-to-extract]) -> returns a string (input not changed!!!)
same as Str.slice(start-index, start-index + length-to-extract)
(allows negative index)

str.replace("string-to-replace", "replace-as"); -> returns a string (input not changed!!!)
will only replace the first apparence of "string-to-replace"; to replace all, use following regular expression: [/string-to-replace/g]; no quotation marks; "g" can be combined with "i" for case-insensitive replace(/gi).

Str.toUpperCase() -> returns a new string;
Str.toLowerCase() -> returns a new string

Str1.concat("str to separate", str2) -> returns a new string as: str1 + "str to separate" + str2

Str.trim() -> returns a new string with front and back white space removed
(same as python_str.strip())

Str.padStart(stop-index, "string-to-repeatedly-fill-out-spaces-until-'str'-is-met") -> return new string
Str.padEnd(stop-index, "string-to-repeatedly-fill-out-space-after-'str'") -> return new string

Str.charAt(index) same as below:
str[index] -> return the char at given index of str;

str.indexOf("string", Optional[start-index]) -> return the FIRST apparence of the string in "str"
or -1 if "string" DNE. Search parts of "str" AFTER start-index
Str.lastIndexOf("string", Optional[start-index]) -> return the LAST apparence of string in "str"
or -1. Search parts of "str" BEFORE "NEGATIVE start-index"

Str.match(Union["string", /string/reg]) -> return the matched parts of 'str' as an ARRAY.
Str.includes("string", Optional[start-index]) -> returns a boolean
showing whether "string" is in "str", search after start-index of str
str.startsWith("string", Optional[start-index of "string"'s **first** char]) -> boolean
str.endsWith("string", Optional[end-index of "string"'s **last** char]) -> boolean

str.split("string-as-a-breakpoint") -> an array containing components of "str"
excluding "breakpoint"

Code: (includes string template, similar to python_str.format())

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Template Literals</h2>

<p>Template literals allows variables in strings:</p>

<p id="demo"></p>

<p>Template literals allows multi-line input, as below.</p>

<script>
let text =
`The quick
brown fox
jumps over
the lazy dog`;

</script>

<p>Template literals are not supported in Internet Explorer.</p>

<script>
let header = "Templates Literals";
let tags = ["template literals", "javascript", "es6"];

let html = `<h2>${header}</h2><ul>`;
// inside the brace javascript code could be written, without brackets (or syntax error)
let html2 = "<h2>${header}</h2><ul>";
// if replace html as html2, the template won't work. -> requires using `` for template strings.

for (const x of tags) {
  html += `<li>${x}</li>`;
  // an example of for loop;
  // addition of template string is supported
}

html += `</ul>`;
document.getElementById("demo").innerHTML = html;
</script>
```

</body>
</html>

Explain:

Use (\ + " or ' or \) to represent (", ', \) in string.
Breaking a string into multiple lines: use (\) in string and then press "Enter"
String objects (new String()) will always evaluates to false for (==) or (===).
\n, \t -> new line, tabular space
Template literals can be useful to create HTML documents as a function and for-loop, as shown in coding part.

Note 9: Numbers/Numeric

Methods:

Numeric.toString(Optional[int]): -> return a new string contaning digits of numeric
"int" indicates the ~nary of numeric; int=2 means convert to binary…

Numeric.toExponential(Optional[int]) -> return a new string written in scientific digits
"int" is the number of digits appear after fraction dot. (will round when necessary)
Unspecified "int" results in no rounding and no extra-zeros added

Numeric.toFixed(Optional[int]) -> return a string written with "int" many digits
After fractional dot.
If "int" is unspecified, will return integer;
This method rounds result when necessary

Numeric_object.valueOf() -> return a primitive type of numeric_object, with type "numeric"
Converts an object to primitive type.

"N"umber(Jsvariable) -> return a numeric converted from jsvariable with various types
Various types means boolean, string or numeric.
If a string contains non-numeric symbols (except space), will return NAN.

parseInt(Jsvariable) -> return an integer converted from jsvariable
will convert the first number appeared in string, then stop searching.
if there are non-numeric symbols (except space) appear before a digit, will return NAN.
No rounding of decimal digits

parseFloat(Jsvariable) -> returna a number (int/float) converted
converts the first number appears in string
if there are non-numeric characters before first number, return NAN

Constants: Number.MAX_VALUE, Number.MIN_VALUE, Number.POSITIVE_INFINITY, Number.NEGATIVE_INFINITY

+Infinity occurs when value is beyond MAX_VALUE or 1/0 is calculated.
Similar for -Infinity

Code:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>
<p> below demonstrates some properties of numeric</p>
<p id="demo"></p>

<script>
function scientific(digit){
	return digit*1e-12;
}

function int_accuracy(){
	return 999999999999999 + 9e15;
}

function string_plus_int(){
	document.getElementById("demo").innerHTML="answer for '10' + 20 + 30 = " + ("10" + 20 + 30) + "<br>" + "answer for '10' + (20 + 30) = " + ("10" + (20 + 30)) + "<br>" + "answer for 10 + 20 + '30' is: " + (10 + 20 + "30") + "<br> And answer for 10 + (20 + '30') is: " + (10 + (20 + "30")) + "<br> note the difference of adding a bracket or not";
// relizing the <br>, an element of HTML is a string of innerHTML element.
}

function demo_infinity(){

document.getElementById("demo").innerHTML = (1/0) + "<br> the type if infinity is: " + typeof Infinity;
}

</script>

<button onclick="document.getElementById('demo').innerHTML = scientific(120)"> scientific digit</button>
<button onclick="document.getElementById('demo').innerHTML = int_accuracy()"> integer accuracy</button>
<button onclick="string_plus_int()"> integer accuracy</button>
<button onclick="demo_infinity()"> demonstrate infinity</button>
</body>
</html>
```

Explain:

Scientific expression: a number + e + power of 10 (no spaces between all symbols)
Precision: integers are accurate up to 15 digits;
Float operations performs better if converting to integer and then divide.
Addition of number and strings: see code (using brackets to avoid wrong type-conversion)
String to numeric: if a string contains only digits, it may be used for arithmetic operations;
(except "+" which performs string concatenation)
NaN and isNAN() -> illegal number /identifier of illegal number returning a boolean
typeof NAN -> returns numeric

"Infinity" -> infinity; typeof Infinity -> outputs numeric

Note 10: Array

Methods:

Array.toString(), array.join("separator") -> both returns a string of array elements
concatenated into a string by: ","(toString) or "separator"(join)

array.pop() -> return the last element of array, and remove it from array
array.push(element) -> return length of new array, and push "element" into **end** of "array".

Array.shift() -> return first element of array, and remove it. (similar to dequeue())
Array.unshift(element) -> return new array's length.
append a new element at the FRONT of array

array[index] -> return the element at position "index" of "array";
array[index] = element -> modify the element at "index" of "array" as "element".

Array.splice(index, int, element1, element2, ….) -> returns an array with deleted elements from "array"
This is a function which can perform both insertion and deletion on array.
index: the position in array where addition/deletion should happen
int: representing how many elements after "index" should be deleted. If no deletion happens, set to 0.
remaining arguments indicate elements which should be inserted from "index".
In practise, deletion happens before insertion.

Arr1.concat(arr2, …) -> return a new array being the concatenation of arr1 and arr2, (arr3…)

Array.slice(union[int, [int, int]]) -> return the sliced new array (original array remains)
Int: returns array[int:]
[int1, int2]: returns array[int1:int2], array[int2] excluded

Array.sort(Optional[compare-function]) -> return an array with elements in “Array” sorted.

Array.reverse() -> return an array with elements in reversed order (relative to “Array”’s order)

“Math.max/min.apply(null, Array)” -> return max or min in “Array”, applies to numeric only.
If non-numeric: result in NAN.
(“M” in “Math” is **capitalized**)

Below are iterating methods for array

Array.forEach(function) -> return None
Performs an action on each element of “Array”
Gives “function” three arguments: value for each iteration, index of value and “Array”.
Requires “function” (written for programmer) to accept these three arguments in order.
e.g: function myFunc(value, index, array)

Array.map(function) -> return a new array
“function” specifies how the new array should be created based on “Array”’s elements.
“function” returns an element.
Function(value, index, Array)

Array.filter(function) -> return a new array with elements satisfying “function”’s constraints.
“function” specifies the criterion for choosing an element.
“function” returns a boolean; if “True” the value iterating will be placed into new array.
“function(value, index, Array)”

Array.reduce(function) -> return a value produced from “Array” and “function” (e.g, sum)
“function” performs a cumulative operation on each array elements.
“function(previously_returned, value, index, Array)”
Previously_returned is a value generated on one previous iteration of “function”.

Array.reduceRight(function) -> return a value
Same effect as “Array.reduce()”, but in OPPOSITE direction. (from back to front of Array)

Array.every(function) -> return a boolean indicating whether
“all Array elements” satisfies “function” criterion
“function” is applied to each single element of Array, and returns a BOOLEAN
If any iteration of “function” returns “false”, “every” also returns “false”

Array.some(function) -> return a boolean indicating whether
“there is an Array elements” satisfies “function” criterion
“function” is applied to each single element of Array, and returns a BOOLEAN
If any iteration of “function” returns “true”, “some” also returns “true”

Below is for array indexing of elements

Array.indexOf(search_element, Optional[start_index]) -> return the index of “search_
element” after “start_index/0” or -1.

Array.lastIndexOf(search_element, Optional[start_index]) -> return the index of “element”
The search happens in “Array[start_index:]”, returns the last apparence of “element”

Array.includes(element) -> returns a boolean indicating whether “element” is in “Array”.

Array.find(function) -> return the VALUE of first element satisfying “function”.
“function” is applied to each element in “Array”, returns a boolean.
If “function” returns “true” on an item, “find” stops iterating and return the element.
“function(value, index, array)”

Array.findIndex(function) -> return the index of first item satisfying “function”
Besides return value, everything is the same as Array.find()

Array.from(variable) -> return a new array where elements are from “splitting” “variable”.

Code:

Explain:

Array creation requires variable declaration, and filling elements inside a square bracket []
Array indexing and modification could be done by: array[index] = element
However array[-1] is undefined; should use “array[array.length - 1]” to access last element
“new Array(int)” -> create an array with length “int” !!!
Sorting numerics requires using a comparison function since array.sort() treats elements as string during sorting. (demonstration of comparison-function on sorting)