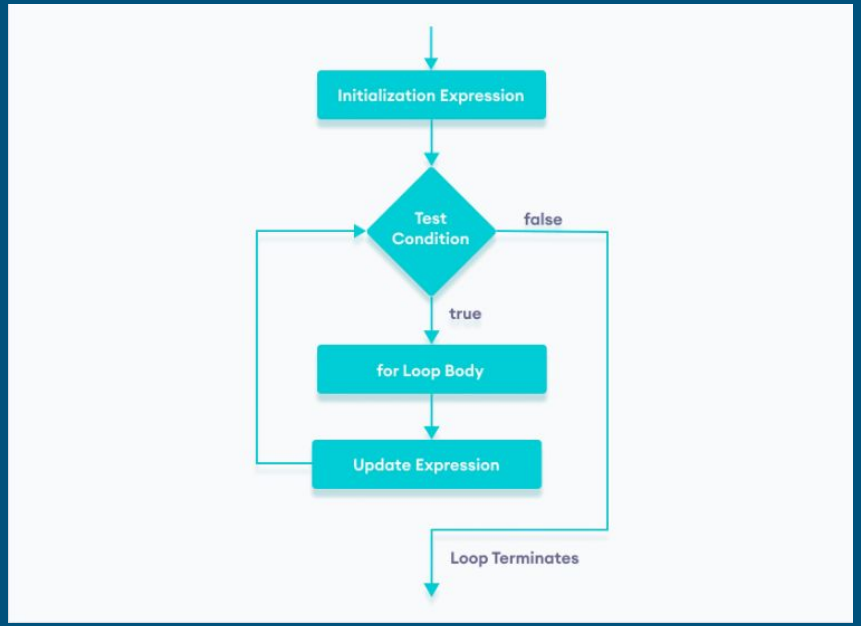
Loops, Arrays, Objects

Mod-2, Class-4

Loops : What is a loop? Different Loops



Different Loops

1. For
2. While
3. Do While
4. For of
5. For in

While

```
while (condition) {
  statement
}
```

```
let e = 0;
while (e < 4) {
 e++;
}
```

Before the statement is executed, `condition` is tested. If it evaluates to `true`, then the statement is executed. As long as `condition` is `true`, the statement continues to execute. When `condition` becomes `false`, the statement stops executing.

For Loop

```
for (initialExpression; conditionExpression; incrementExpression) {
 statement
}
```

```
for (let i = 1; i <= 5; i++) {
 console.log("I can count to " + i)
}
```

For In and For of Loops will be discussed later……to be cntd…

Do While

```
do {
  statement
} while (condition);
```

```
let booksRead = 10;
do {
 console.log(`I read ${booksRead} books this year`);
 booksRead++;
} while (booksRead < 14);
```

The first thing that happens in this loop is the statement is executed. Once that happens, `condition` is checked. If `condition` evaluates to `true`, the statement executes again. The statement keeps executing until `condition` evaluates to `false`. The major difference between the `do...while` loop and the `while` loop is that the statement will always be executed **at least once.**

Arrays

```
let array = [1, 12, 2.5, null, 'John', true, 100]
```

	int	int	float	Null	string	bool	number
Elements: →	1	12	2.5	null	'John'	true	100
Index : → (position)	0	1	2	3	4	5	6

Javascript Array

An array can hold many values under a single name, and you can access the values by referring to an index number.

Length and accessing elements

The `length` property of an array returns the length of an array (the number of array elements).

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
```

Accessing the First Array Element

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0];
```

Methods in Array

JavaScript Array Methods

pop()	shift()	find()
push()	unshift()	forEach()
toString()	reverse()	map()
join()	concat()	reduce()
splice()	slice()	every()
sort()	filter()	some()

Splice vs Slice

`slice`

The term ‘slice’ literally means to cut something into pieces. It is used to cut out elements from an array. It does not affect the original array.

Syntax

```
array.slice(start, end)
```

- `start` denotes the index at which the array starts slicing.
- `end` denotes the index at which the array stops slicing.

`splice`

‘Splice’, according to the dictionary, means to join things together. It is used to remove elements from an array or replace them.

Syntax

```
array.splice(start, deleteCount, newElem1, newElem2, ..., newElemN;
```

- `start` denotes the index from which the method will start its operation on the array.
- `deleteCount` denotes the number of values to be deleted from the `start`. If the value is `0`, nothing will be deleted.
- `newElem1` to `newElemN` denote the values that would be added after the `start`.

Task 1

1. Given two arrays A1 = [‘d’,’e’,’f’,1,2,3,’A’,’B’,’C’], A2=[‘D’,’E’,’F’,4,5,6,’a’,’b’,’c’]
   Use slice and splice or any other array functions
   make the result as A1=[1,2,3,4,5,6] A2=[‘A’,’B’,’C’,’D’,’E’,’F’] A3=[‘a’,’b’,’c’,’d’,’e’,’f’]
2. Print the following stars pattern using for loop

```
N = 5
* * * * * * * * *
 * * * * * * *
  * * * * *
   * * *
    *
```

For loop for an array

Start i at 0

While i < length of array

Use loop counter i as index of array

```
for (int i = 0; i < arrayname.length; i++)
{
    System.out.println( arrayname[i] );
}
```

For of

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

```
const array = [5, 10, 15];
for (const value of array) {
 console.log(value);
}
```

The above loop would console log `5`, `10`, `15`.

What is an Object

```
let person = {
  name:  'John',
  age:  20
};
```

Keys — Values

Here, an object `object_name` is defined. Each member of an object is a **key: value** pair separated by commas and enclosed in curly braces `{}`.

Object Interaction

```
const person = {
    name: 'John',
    age: 20,
};

// accessing property
console.log(person.name); // John
```

```
const person = {
    name: 'John',
    age: 20,
};

// accessing property
console.log(person["name"]); // John
```

Array of objects example,

Task 3

Loop through the following Array

Print three arrays which has

A1=[<name1>,<name2>.....]

A2=[<cmpny1>,<cmpny2>.....]

A3=[<des1>,<des2>]

```
const list = [
  {
    name: 'Michael Scott',
    company: 'Dunder Mufflin',
    designation: 'Regional Manager',
    show: 'The Office'
  },
  {
    name: 'Barney Stinson',
    company: 'Golaith National Bank',
    designation: 'Please',
    show: 'How I met your mother'
  },
  {
    name: 'Jake Peralta',
    company: 'NYPD',
    designation: 'Detective',
    show: 'Brooklyn 99'
  },
]
```

Map() Filter() Reduce()

```
[🐮, 🥔, 🐔, 🌽].map(cook) ⇒ [🍔, 🍟, 🍗, 🍟]
[🍔, 🍟, 🍗, 🍟].filter(isVegetarian) ⇒ [🍟, 🍟]
[🍔, 🍟, 🍗, 🍟].reduce(eat) ⇒ 💩
```

Map()

```
var new_array = arr.map(function callback(element, index, array) {
    // Return value for new_array
}[, thisArg])
```

In the callback, only the array `element` is required. Usually some action is performed on the value and then a new value is returned.

Example

In the following example, each number in an array is doubled.

```
const numbers = [1, 2, 3, 4];
const doubled = numbers.map(item => item * 2);
console.log(doubled); // [2, 4, 6, 8]
```

Filter

```
var new_array = arr.filter(function callback(element, index, array) {
    // Return true or false
}[, thisArg])
```

The syntax for `filter` is similar to `map`, except the callback function should return `true` to keep the element, or `false` otherwise. In the callback, only the `element` is required.

Examples

In the following example, odd numbers are "filtered" out, leaving only even numbers.

```
const numbers = [1, 2, 3, 4];
const evens = numbers.filter(item => item % 2 === 0);
console.log(evens); // [2, 4]
```

Reduce

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce(function (result, item) {
  return result + item;
}, 0);
console.log(sum); // 10
```

```
arr.reduce(callback[, initialValue])
```

The `callback` argument is a function that will be called once for every item in the array. This function takes four arguments, but often only the first two are used.

- *accumulator* - the returned value of the previous iteration
- *currentValue* - the current item in the array
- *index* - the index of the current item
- *array* - the original array on which reduce was called
- The `initialValue` argument is optional. If provided, it will be used as the initial accumulator value in the first call to the callback function.

Task 3 use only reduce

Input

```
var pets = ['dog', 'chicken', 'cat', 'dog', 'chicken', 'chicken', 'rabbit'];
```

Output

```
/*
Output:
 {
    dog: 2,
    chicken: 3,
    cat: 1,
    rabbit: 1
 }
 */
```

setTimeOut

```
setTimeout(function(){
    console.log("Hello World");
}, 2000);

console.log("setTimeout() example...");
```

The `setTimeout()` method allows you to execute a piece of code after a certain amount of time has passed. You can think of the method as a way to set a timer to run JavaScript code at a certain time.

setInterval

`setInterval()` is a function in JavaScript that is used to repeatedly execute a given function at a specified interval. It takes two arguments: the first argument is the function to be executed, and the second argument is the time interval (in milliseconds) between each execution.

Here is an example to illustrate how `setInterval()` works:

javascript Copy code

```javascript
setInterval(function() {
  // code to be executed repeatedly
  console.log("This will be printed every 2 seconds");
}, 2000);
```

In the example above, the provided anonymous function will be executed every 2 seconds (2000 milliseconds) and will print the message "This will be printed every 2 seconds" to the console.

It's important to note that `setInterval()` will continue to execute the function indefinitely until it is stopped using the `clearInterval()` function.

Task 4

1. Given an array [1,3,5,2,8,12]
   Sort the array
   A.using inbuilt functions acending, descending
   B. Without using inbuilt functions
   C. loop thorough the array and print values with a delay in 2s

For each and searching in an array

```
1  let arr = [1, 0, false];
2
3  alert( arr.indexOf(0) ); // 1
4  alert( arr.indexOf(false) ); // 2
5  alert( arr.indexOf(null) ); // -1
6
7  alert( arr.includes(1) ); // true
```

```
arr.forEach(function(item, index, array) {
  // ... do something with item
});
```

or instance, this shows each element of the array:

```
// for each element call alert
["Bilbo", "Gandalf", "Nazgul"].forEach(alert);
```

nd this code is more elaborate about their positions in the target array:

```
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
  alert(`${item} is at index ${index} in ${array}`);
});
```

Task 5

```
1  let john = { name: "John", age: 25 };
2  let pete = { name: "Pete", age: 30 };
3  let mary = { name: "Mary", age: 28 };
4
5  let arr = [ pete, john, mary ];
6
7  sortByAge(arr);
8
9  // now: [john, mary, pete]
10 alert(arr[0].name); // John
11 alert(arr[1].name); // Mary
12 alert(arr[2].name); // Pete
```

Thank You