ARRAYS PUSH

→ The push() method adds new item to the end of an array.

→ The push() method changes the length of the array.

→ The push() method returns the new length.

Syntax :-

parameters

```
array.push(item1, item2, ...., itemX)
```

The item(s) to add to the array.
Minimum one item is required.

Return value type :- → a number

The new length of the array

Javascript > Lecture_37 > JS 1_ArraysPush.js > ...

```
1
2   // push appends 1 or more values at the end of array
3   // push returns the new length of array
4
5   let arr = [20, 30, 80, 100, 40];
6   displayArray(arr);
7
8   arr.push(1000);
9   displayArray(arr);
10
11  arr.push(2000, 3000, 4000);
12  displayArray(arr);
13
14  let rv = arr.push(5000, 6000);
15  displayArray(arr);
16  console.log(rv);
17
18  function displayArray(arr){
19      console.log(arr + " = " + arr.length);
20  }
21
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_37 git:(main) ✗ node 1_ArraysPush.js
20,30,80,100,40 = 5
20,30,80,100,40,1000 = 6
20,30,80,100,40,1000,2000,3000,4000 = 9
20,30,80,100,40,1000,2000,3000,4000,5000,6000 = 11
11
→  Lecture_37 git:(main) ✗
```

ARRAYS POP

→ The pop() method removes (pops) the last element of an array

→ the pop() method changes the original array.

→ The pop() method returns the removed element.

Syntax

```
array.pop()
```

parameters → None

Return Value Type → A variable → The removed item

A string, a number, an array, or any other type allowed in an array.

JS 2_ArraysPop.js U

Javascript > Lecture_37 > JS 2_ArraysPop.js > displayArray

```
1
2   // pop removes 1 value from the end
3   // pop returns the removed value
4
5   let arr = [20, 30, 80, 100, 40];
6   displayArray(arr);
7
8   let rv = arr.pop();
9   displayArray(arr);
10  console.log(rv);
11
12  function displayArray(arr){
13      console.log(arr + " = " + arr.length);
14  }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_37 git:(main) ✗ node 2_ArraysPop.js
20,30,80,100,40 = 5
20,30,80,100 = 4
40
```

ARRAYS TRICK SET 1

JS 3_ArraysTricksSet_1.js U

Javascript > Lecture_37 > JS 3_ArraysTricksSet_1.js > ...

```
1
2   let arr = [20, 30, 80, 100, 40];
3   displayArray(arr);
4
5   arr[10] = 500; // no array out of index exception, increases the length
6   displayArray(arr);
7
8   arr["kuchbhi"] = 1000; // because every array can be used like an object also
9   displayArray(arr);
10  console.log(arr["kuchbhi"]);
11
12  function displayArray(arr){
13      console.log(arr + " = " + arr.length);
14  }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_37 git:(main) ✗ node 3_ArraysTricksSet_1.js
20,30,80,100,40 = 5
20,30,80,100,40,,,,,,500 = 11
20,30,80,100,40,,,,,,500 = 11
1000
```

ARRAY PUSH, POP, UNSHIFT, SHIFT

ADD	REMOVE	works where in an array?
PUSH	POP	works at the back of array
UNSHIFT	SHIFT	works at the front of array

ARRAYS UNSHIFT

→ The unshift() method adds new elements to the beginning of an array.

→ The unshift() method overwrites the original array.

Syntax :-

Parameters

```
array.unshift(item1, item2, ...., itemX)
```

The item(s) to add to the array.
Minimum one item is required.

Return value type :- → a number

→ The new length of the array

```
// unshift adds 1 or more values at the end of array
// unshift returns the new length of array

let arr = [20, 30, 80, 100, 40];
displayArray(arr);

arr.unshift(1000);
displayArray(arr);

arr.unshift(2000, 3000, 4000);
displayArray(arr);

let rv = arr.unshift(5000, 6000);
displayArray(arr);
console.log(rv);

function displayArray(arr){
    console.log(arr + " = " + arr.length);
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_37 git:(main) ✗ node 4_ArraysUnshift.js
20,30,80,100,40 = 5
1000,20,30,80,100,40 = 6
2000,3000,4000,1000,20,30,80,100,40 = 9
5000,6000,2000,3000,4000,1000,20,30,80,100,40 = 11
11
```

ARRAY SHIFT

The shift() method removes the first item of an array.

The shift() method changes the original array.

The shift() method returns the shifted element.

Syntax

```
array.shift()
```

parameters → None

Return Value Type → A variable → The removed item

A string, a number, an array, or any other type allowed in an array.

JS 5_ArraysShift.js U

Javascript > Lecture_37 > JS 5_ArraysShift.js > rv

```
1
2   // shift removes 1 value from the front
3   // shift returns the removed value
4
5   let arr = [20, 30, 80, 100, 40];
6   displayArray(arr);
7
8   let rv = arr.shift();
9   displayArray(arr);
10  console.log(rv);
11
12  function displayArray(arr){
13      console.log(arr + " = " + arr.length);
14  }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_37 git:(main) ✗ node 5_ArraysShift.js
20,30,80,100,40 = 5
30,80,100,40 = 4
20
```

Ques :- Rearrange array so that all even elements are at front and all odd elements are at back (push, pop, shift, unshift).

```js
// rearrange array so that all even elements are at the fr

// approach 1

let arr = [23, 89, 66, 45, 78, 13, 44, 34];
let temp = [];
while(arr.length > 0){
    let val = arr.shift();
    if(val % 2 == 0){
        temp.unshift(val);
    }else{
        temp.push(val);
    }
}

arr = temp;
console.log(arr);
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

→  Lecture_37 git:(main) ✗ node 6_Ques_1.js
[
  34, 44, 78, 66,
  23, 89, 45, 13
]
```

Javascript > Lecture_37 > JS 6_Ques_1.js > ...

```
19
20    // approach 2
21
22    let arr = [20, 30, 12, 17, 9, 18, 43, 64, 11, 47];
23    let odd = [];
24    let even = [];
25
26    let i = 0;
27
28    while(arr.length > 0){
29        let val = arr.shift();
30        if(val % 2 == 0){
31            even.push(val)
32        }else{
33            odd.push(val);
34        }
35    }
36
37    arr = even.concat(odd);
38
39    displayArray(arr);
40    displayArray(odd);
41    displayArray(even);
42
43
44    function displayArray(arr){
45        console.log(arr + " = " + arr.length);
46    }
47
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_37 git:(main) ✗ node 6_Ques_1.js
20,30,12,18,64,17,9,43,11,47 = 10
17,9,43,11,47 = 5
20,30,12,18,64 = 5
→  Lecture_37 git:(main) ✗
```

ARRAY SLICE

The slice() method returns selected elements in an array, as a new array.

The slice() method selects from a given start, up to a (not inclusive) given end.

The slice() method does not change the original array.

Syntax

parameters

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

start is inclusive
end is exclusive

start: Optional. Start position. Default is 0. Negative number select from the end of the array.

end: Optional. End position. Default is last element. Negative numbers select from the end of the array.

Return Value Type :- → A new array containing the selected element.

eg:-

[10, 20, 30, 40] → a
 0   1   2   3

(works like substring in Java)

```
na = a.slice(0, 2)   → [10, 20]
na = a.slice(1, 3)   → [20, 30]
na = a.slice(1)      → [20, 30, 40]
na = a.slice()       → [10, 20, 30, 40]
```

→ for cloning an array.

Negative index

```
[10, 20, 30, 40, 50]
 0   1   2   3   4
-5  -4  -3  -2  -1
```

a.slice(-3, -1) → [30, 40]

```
// Like substring
// start is inclusive, end is exclusive

let arr = [10, 20, 30, 40, 50];
let na1 = arr.slice(1, 4); // from 1 to 3 (4 not included)
displayArray(na1);

let na2 = arr.slice(1); // from 1 to end
displayArray(na2);

function displayArray(arr){
    console.log(arr + " = " + arr.length);
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_37 git:(main) ✗ node 7_ArraysSlice.js
20,30,40 = 3
20,30,40,50 = 4
→  Lecture_37 git:(main) ✗
```

```
// Like substring
// start is inclusive, end is exclusive

let arr = [10, 20, 30, 40, 50];
let na1 = arr.slice(1, 4); // from 1 to 3 (4 not included)
displayArray(na1);

let na2 = arr.slice(1); // from 1 to end
displayArray(na2);

let na3 = arr.slice();
displayArray(na3); // the entire array (can be used for cloning)

let na4 = arr.slice(-3, -1); // from -3 to -2
displayArray(na4);

let na5 = arr.slice(-3); // from -3 to end
displayArray(na5);

let na6 = arr.slice(1, -2); // from -3 to end
displayArray(na6);

function displayArray(arr){
    console.log(arr + " = " + arr.length);
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

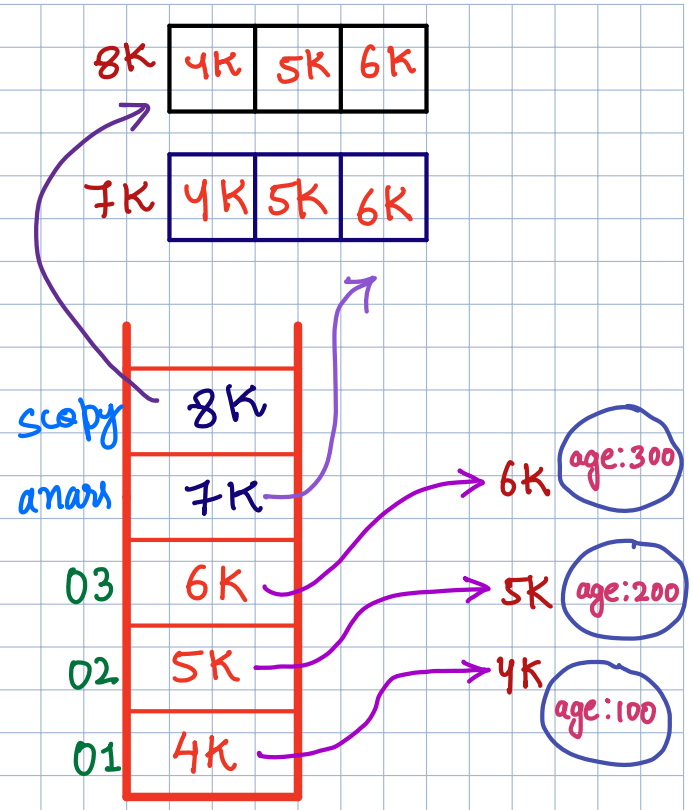
```
→  Lecture_37 git:(main) ✗ node 7_ArraysSlice.js
20,30,40 = 3
20,30,40,50 = 4
10,20,30,40,50 = 5
30,40 = 2
30,40,50 = 3
20,30 = 2
```

SHALLOW COPIES

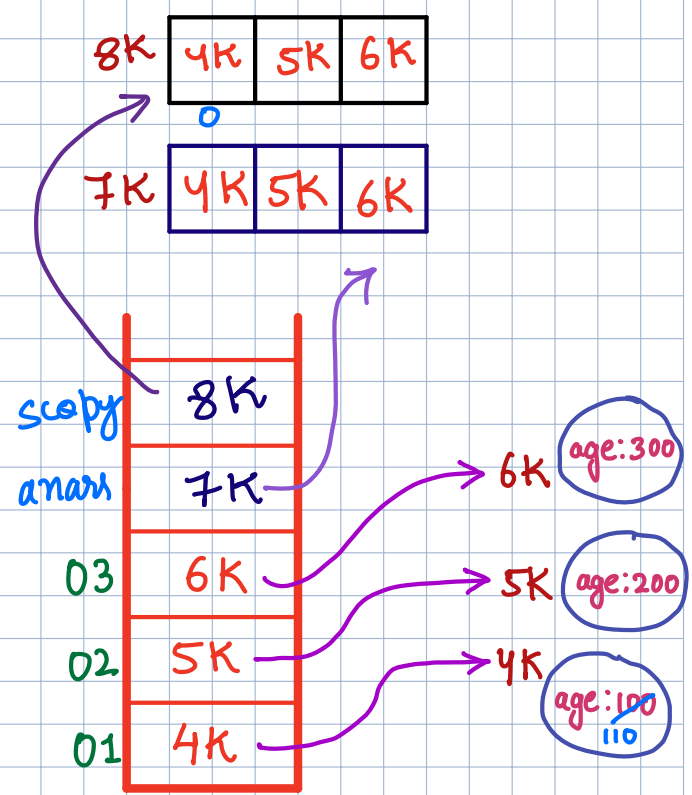
```
28  // shallow copies
29  let o1 = {
30      age: 100
31  };
32  let o2 = {
33      age: 200
34  }
35  let o3 = {
36      age: 300
37  }
38
39  let anarr = [o1, o2, o3];
40  displayObjectArray(anarr);
41
42  let scopy = anarr.slice();
43  displayObjectArray(scopy);
44
45  scopy[0].age =110;
46  displayObjectArray(anarr);
47  displayObjectArray(scopy);
48
49  function displayObjectArray(arr){
50      let str = "";
51
52      for(let i = 0; i < arr.length; i++){
53          str += arr[i].age + ", ";
54      }
55
56      console.log(str + ".");
57  }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→ Lecture_37 git:(main) ✗ node 7_ArraysSlice.js
100, 200, 300, .
100, 200, 300, .
110, 200, 300, .
110, 200, 300, .
```



scopy[0].age = 110



Shallow Copy: When a reference variable is copied into a new reference variable using the assignment operator, a shallow copy of the referenced object is created. In simple words, a reference variable mainly stores the address of the object it refers to. When a new reference variable is assigned the value of the old referenced variable, the address stored in the old reference variable is copied into the new one. The means both the old and new reference variable point to the same object in memory. As a result if the state of the object changes through any of the reference variables it is refected for both.

```
59   console.log("String Case -> primitive");
60
61   var animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];
62
63   var t = animals.slice(2,4);
64   console.log(t);
65
66   t[0] = 'aaa';
67   console.log(t);
68   console.log(animals);
69
70   console.log("Object form of array Case");
71
72   var animals = [{name: 'ant'}, {name: 'bison'}, {name: 'camel'}, {name: 'duck'}, {name: 'elephant'}];
73
74   var t = animals.slice(2,4);
75   console.log(t);
76
77   t[0].name = 'aaa';
78   console.log(t);
79   console.log(animals);
```

Strings are primitive types in Javascript, so you will get a new array with new strings inside.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                    zsh - Le

→  Lecture_37 git:(main) ✗ node 7_ArraysSlice.js
String Case -> primitive
[ 'camel', 'duck' ]
[ 'aaa', 'duck' ]
[ 'ant', 'bison', 'camel', 'duck', 'elephant' ]
Object form of array Case
[ { name: 'camel' }, { name: 'duck' } ]
[ { name: 'aaa' }, { name: 'duck' } ]
[
  { name: 'ant' },
  { name: 'bison' },
  { name: 'aaa' },
  { name: 'duck' },
  { name: 'elephant' }
]
```

Slice does not alter the original array. It returns a shallow copy of elements from the original array.

Elements of the original array are copied into the returned array as follows:

For object references (and not the actual object), slice copies object references into the new array. Both the original and new array refer to the same object. If a referenced object changes, the changes are visible to both the new and original arrays.

For strings, numbers and booleans (not String, Number and Boolean objects), slice copies the values into the new array. Changes to the string, number or boolean in one array do not affect the other array. If a new element is added to either array, the other array is not affected.

Ques :- Print all subarrays using slice.

[10, 20, 30, 40]

[10] [20] [30] [40]

[10, 20] [20, 30] [30, 40]

[10, 20, 30] [20, 30, 40]

[10, 20, 30, 40]

```
1   // Print all subarrays using slice
2
3   let arr = [10, 20, 30, 40];
4
5   for(let i = 0; i < arr.length; i++){
6       for(let j = i + 1; j <= arr.length; j++){
7           let sa = arr.slice(i, j);
8           displayArray(sa);
9       }
10  }
11
12  function displayArray(arr){
13      console.log(arr + " = " + arr.length);
14  }
15
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_37 git:(main) ✗ node 8_Ques2.js
10 = 1
10,20 = 2
10,20,30 = 3
10,20,30,40 = 4
20 = 1
20,30 = 2
20,30,40 = 3
30 = 1
30,40 = 2
40 = 1
→  Lecture_37 git:(main) ✗
```

ARRAY SPLICE

The splice() method add and/or removes array elements.

The splice() method overwrites the original array.

Syntax

parameter

array.splice(start index, deleteCount, item1, ..., item X)

Required
The position to add/remove items. Negative value defines the position from the end of the array

Optional
Number of items to be removed.

Optional
New elements(s) to be added.

Return Value Type :- → An array containing the removed items.

```
2
3   let arr = [10, 20, 30, 40, 50];
4
5   let na = arr.splice(2, 2, 300, 400, 500);
6
7   displayArray(arr);
8   displayArray(na);
9
10  function displayArray(arr){
11      console.log(arr + " = " + arr.length);
12  }
```

[10, 20, 30, 40, 50]
0 1 2 3 4

→ [30,40]

→ [10,20,300,400,500,50]

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

→  Lecture_37 git:(main) ✗ node 9_ArraysSplice.js
10,20,300,400,500,50 = 6
30,40 = 2
→  Lecture_37 git:(main) ✗
```

arr.slice(i,1)
↓
used for removal

arr.splice(i,0,val)
↓
used for insertion

Ques :- Remove all prime numbers from array.

```
1
2   // Remove all prime numbers from array
3
4   let arr = [10, 47, 15, 17, 92, 97, 93, 31, 46, 54];
5
6   for(let i = arr.length - 1; i >= 0; i--){
7       let val = arr[i];
8       let isPrime = IsPrime(val);
9       if(isPrime == true){
10          arr.splice(i, 1);
11      }
12  }
13
14  displayArray(arr);
15
16
17  function displayArray(arr){
18      console.log(arr + " = " + arr.length);
19  }
20
21  function IsPrime(val){
22      for(let div = 2; div * div <= val; div++){
23          if(val % div == 0){
24              return false;
25          }
26      }
27      return true;
28  }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_37 git:(main) ✗ node 10_Ques3.js
10,15,92,93,46,54 = 6
→  Lecture_37 git:(main) ✗
```