

Computer Science Principles

Web Programming

JavaScript Programming Essentials

CHAPTER 3: ARRAYS

DR. ERIC CHOU

IEEE SENIOR MEMBER



Overview

LECTURE 1



Array object

- An **array** is an ordered set of values that you refer to with a name and an index. For example, you could have an array called `emp` that contains employees' names indexed by their numerical employee number. So `emp[1]` would be employee number one, `emp[2]` employee number two, and so on.
- JavaScript does not have an explicit array data type. However, you can use the predefined `Array` object and its methods to work with arrays in your applications. The **Array** object has methods for manipulating arrays in various ways, such as joining, reversing, and sorting them. It has a property for determining the array length and other properties for use with regular expressions.



Why Array?

- Let's look at dinosaurs again. Say you want to use a program to keep track of the many kinds of dinosaurs you know about. You could create a variable for each dinosaur, like this:

```
var dinosaur1 = "T-Rex";  
var dinosaur2 = "Velociraptor";  
var dinosaur3 = "Stegosaurus";  
var dinosaur4 = "Triceratops";  
var dinosaur5 = "Brachiosaurus";  
var dinosaur6 = "Pteranodon";  
var dinosaur7 = "Apatosaurus";  
var dinosaur8 = "Diplodocus";  
var dinosaur9 = "Compsognathus";
```



Why Array?

- This list is pretty awkward to use, though, because you have nine different variables when you could have just one. Imagine if you were keeping track of 1000 dinosaurs!
- You'd need to create 1000 separate variables, which would be almost impossible to work with.



Declaration

LECTURE 2



Arrays

- Objects allow you to store **keyed** collections of values. That's fine.
- But quite often we find that we need an ordered collection, where we have a 1st, a 2nd, a 3rd element and so on. For example, we need that to store a list of something: users, goods, HTML elements etc.
- It is not convenient to use an object here, because it provides no methods to manage the order of elements. We can't insert a new property "between" the existing ones. Objects are just not meant for such use.
- There exists a special data structure named Array, to store **ordered collections**.



Declaration

- There are two syntaxes for creating an empty array:
 1. `var arr = new Array();` // use construction
 2. `var arr = [];` // use list notation.



Creating an Array

- To create an array, you just use square brackets, []. In fact, an **empty array** is simply a pair of square brackets, like this:
[];
- What is the difference between null and empty array?
- To create an array with values in it, enter the values, separated by commas, between the square brackets.



Creating an Array

- We can call the individual values in an array items or elements. In this example, our elements will be strings (the names of our favorite dinosaurs), so we'll write them with quote marks. We'll store the array in a variable called dinosaurs:

```
var dinosaurs = ["T-Rex", "Velociraptor",  
  "Stegosaurus", "Triceratops", "Brachiosaurus",  
  "Pteranodon", "Apatosaurus",  
  "Diplodocus", "Compsognathus"  
];
```



Array Literals

LECTURE 3



JavaScript : Array literals

Description

- In Javascript, an array literal is a list of expressions, each of which represents an array element, enclosed in a pair of square brackets ' [] '. When an array is created using an array literal, it is initialized with the specified values as its elements, and its length is set to the number of arguments specified. If no value is supplied it creates an empty array with zero length.



JavaScript : Array literals

Examples:

- Creating an empty array :

```
var fruits = [ ];
```

- Creating an array with four elements.

```
var fruits = ["Orange", "Apple", "Banana", "Mango"]
```



Comma in array literals

- There is no need to specify all elements in an array literal. If we put two commas in a row at any position in an array then an unspecified element will be created in that place.
- The following example creates the fruits array :
fruits = ["Orange", , "Mango"]
- This array has one empty element in the middle and two elements with values. (fruits[0] is "Orange", fruits[1] is set to undefined, and fruits[2] is "Mango").



Comma in array literals

- If you include a single comma at the end of the elements, the comma is ignored. In the following example, the length of the array is three. There are no fruits[2].

```
fruits = ["Orange", "Mango",]
```

- In the following example, the length of the array is four, and fruits[0] and fruits[2] are undefined.

```
fruits = [ , 'Apple', , 'Orange'];
```



Creation of Array

Demo Using Chrome Console:

1. Using constructor Array to create an array.
2. `ary[0];` // first element
3. `ary[3];` // undefined (no such element)

```
> var ary = Array("A", "B", "C");  
< undefined  
  
> ary  
< ▶ (3) ["A", "B", "C"]  
  
> ary[1];  
< "B"  
  
> ary[2];  
< "C"  
  
> ary[3];  
< undefined  
  
> ary[0];  
< "A"  
  
>
```




Heterogenous Array

LECTURE 4

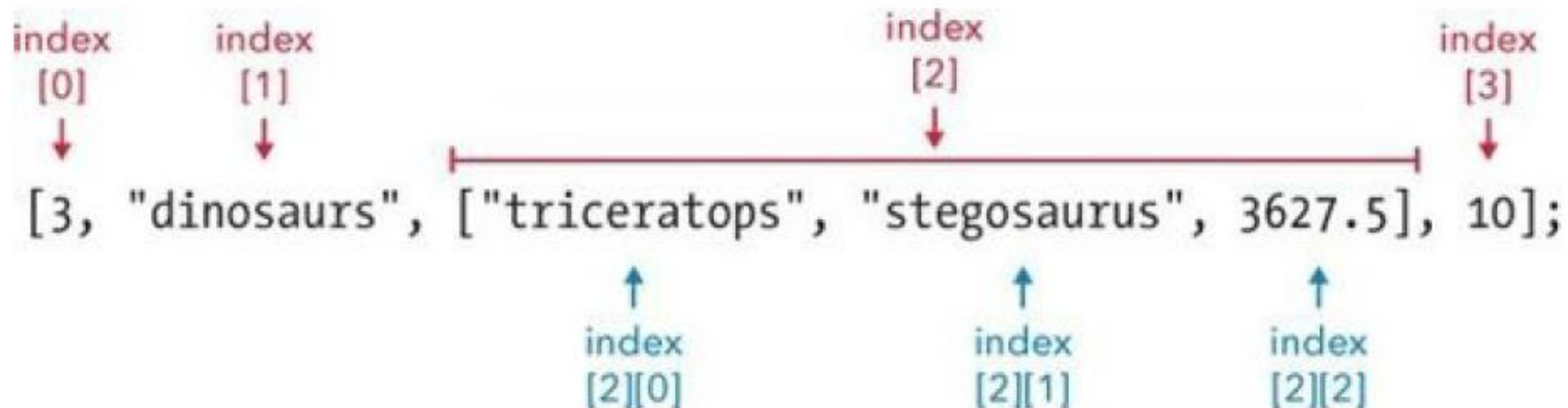


Mixing Data Types in an Array

- Array elements don't all have to be the same type. For example, the next array contains a number (3), a string ("dinosaurs"), an array (["triceratops", "stegosaurus", 3627.5]), and another number (10):

var **dinosaursAndNumbers** =

[3, "dinosaurs", ["triceratops", "stegosaurus", 3627.5], 10];





Creating an Array Using new Operator

1. Use new operator to create an array.
2. toString()
3. sort();
4. reverse();

```
> var alist = new Array(1, 2, 3, 4, 5);  
< undefined  
  
> alist  
< ► (5) [1, 2, 3, 4, 5]  
  
> alist.toString()  
< "1,2,3,4,5"  
  
> alist.sort();  
< ► (5) [1, 2, 3, 4, 5]  
  
> alist.reverse();  
< ► (5) [5, 4, 3, 2, 1]  
  
>
```



Setting or Changing Elements in an Array

Demo Using Chrome Console:

1. JavaScript Array can be homogenous/heterogenous
2. **ary.length** is a instance variable for the array length.

```
> ary.length
```

```
< 3
```

```
> ary = [1, 3, 5];
```

```
< ▶ (3) [1, 3, 5]
```

```
> ary[1];
```

```
< 3
```

```
> ary[2] = "C";
```

```
< "C"
```

```
> ary
```

```
< ▶ (3) [1, 3, "C"]
```

```
>
```



JavaScript Array is an Object

LECTURE 5



JavaScript Array is of Object Type

- The **Array** built-in object can be used to construct objects with special properties and that inherit various methods

```
var ary1 = new Array();
```

Array Constructor
No arguments

ary1
length (0)
toString() sort() shift() ...

Properties

Inherited
methods



JavaScript Arrays

- The **Array** built-in object can be used to construct objects with special properties and that inherit various methods

```
var ary2 = new Array(4, true, "OK");
```

ary2	
Elements of array {	length (3)
	"0" (4)
	"1" (true)
	"2" ("OK")
toString()	
...	

Accessing array elements:

✓ ary2[1]
✓ ary2["1"]
✗ ary2.1

Must follow identifier
syntax rules



JavaScript Arrays

- The `Array` constructor is indirectly called if an **array initializer** is used

```
var ary2 = new Array(4, true, "OK");
```



```
var ary3 = [4, true, "OK"];
```

- Array initializers can be used to create **multidimensional arrays**

```
var ttt = [ [ "X", "0", "0" ],  
            [ "0", "X", "0" ],  
            [ "0", "X", "X" ] ];
```

`ttt[1][2]`



JavaScript Arrays

```
var ary2 = new Array(4, true, "OK");
```

```
ary2[3] = -12.6;
```

Creates a new element dynamically,
increases value of length

ary2	
length	(4)
"0"	(4)
"1"	(true)
"2"	("OK")
"3"	(-12.6)
toString()	
...	



JavaScript Arrays

- Changing the number of elements:

```
var ary2 = new Array(4, true, "OK");  
ary2[3] = -12.6;
```

`ary2.length = 2;` Decreasing length can delete elements

ary2
length (2)
"0" (4)
"1" (true)
toString() ...



JavaScript Arrays

- Value of `length` is not necessarily the same as the actual number of elements

```
var ary4 = new Array(200);
```

Calling constructor with single argument sets `length`, does not create elements

ary4
length (200)
toString() sort() shift() ...


TABLE 4.7 Methods Inherited by Array Objects. Unless Otherwise Specified, Methods Return a Reference to the Array on Which They are Called.

Method	Description
<code>toString()</code>	Return a String value representing this array as a comma-separated list.
<code>sort(Object)</code>	Modify this array by sorting it, treating the Object argument as a function that specifies sort order (see text).
<code>splice(Number, 0, any type)</code>	Modify this array by adding the third argument as an element at the index given by the first argument, shifting elements up one index to make room for the new element.
<code>splice(Number, Number)</code>	Modify this array by removing a number of elements specified by the second argument (a positive integer), starting with the index specified by the first element, and shifting elements down to take the place of those elements removed. Returns an array of the elements removed.
<code>push(any type)</code>	Modify this array by appending an element having the given argument value. Returns <code>length</code> value for modified array.
<code>pop()</code>	Modify this array by removing its last element (the element at index <code>length-1</code>). Returns the value of the element removed.
<code>shift()</code>	Modify this array by removing its first element (the element at index 0) and shifting all remaining elements down one index. Returns the value of the element removed.



toString(): comma-separated format

HTML online Editor

tutorialspoint  **SIMPLY EASY LEARNING** **HTML Online Editor**

Execute

index.htm

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

<html>

<head>

<title>JavaScript Array toString Method</title>

</head>

<body>

<script type="text/javascript">

var arr = new Array("orange", "mango", "banana",

"sugar");

var str = arr.toString();

document.write("Returned string is : " + str);

</script>

</body>

</html>


Result

Returned string is : orange,mango,banana,sugar



splice():

HTML online Editor

tutorialspoint  SIMPLY EASY LEARNING HTML Online Editor

▶ Execute

index.htm

```
1 <html>
2   <head>
3     <title>JavaScript Array splice Method</title>
4   </head>
5
6   <body>
7
8     <script type="text/javascript">
9       var arr = ["orange", "mango", "banana", "sugar",
10        "tea"];
11
12       var removed = arr.splice(2, 0, "water");
13       document.write("After adding 1: " + arr );
14       document.write("<br />removed is: " + removed);
15
16       removed = arr.splice(3, 1);
17       document.write("<br />After adding 1: " + arr );
18       document.write("<br />removed is: " + removed);
19     </script>
20
21   </body>
22 </html>
23
```

☒ Result

>>

After adding 1: orange,mango,water,banana,sugar,tea
removed is:
After adding 1: orange,mango,water,sugar,tea
removed is: banana

```
// ArrayMethods.js
var numArray = [1,3,8,4,9,7,6,2,5];

// Sort in ascending order
numArray.sort(
    function compare (first, second) {
        return first - second;
    }
);
// numArray.toString(): 1,2,3,4,5,6,7,8,9

numArray.splice(2, 0, 2.5);
// numArray.toString(): 1,2,2.5,3,4,5,6,7,8,9

// output of following: 5,6,7
window.alert(numArray.splice(5,3).toString());
// numArray.toString(): 1,2,2.5,3,4,8,9
window.alert(numArray.toString());
```

← Add element with value 2.5 at index 2, shift existing elements

```
// ArrayMethods.js
var numArray = [1,3,8,4,9,7,6,2,5];

// Sort in ascending order
numArray.sort(
    function compare (first, second) {
        return first - second;
    }
);
// numArray.toString(): 1,2,3,4,5,6,7,8,9

numArray.splice(2, 0, 2.5);
// numArray.toString(): 1,2,2.5,3,4,5,6,7,8,9

// output of following: 5,6,7
window.alert(numArray.splice(5,3).toString());
// numArray.toString(): 1,2,2.5,3,4,8,9
window.alert(numArray.toString());
```

Remove 3 elements starting at index 5



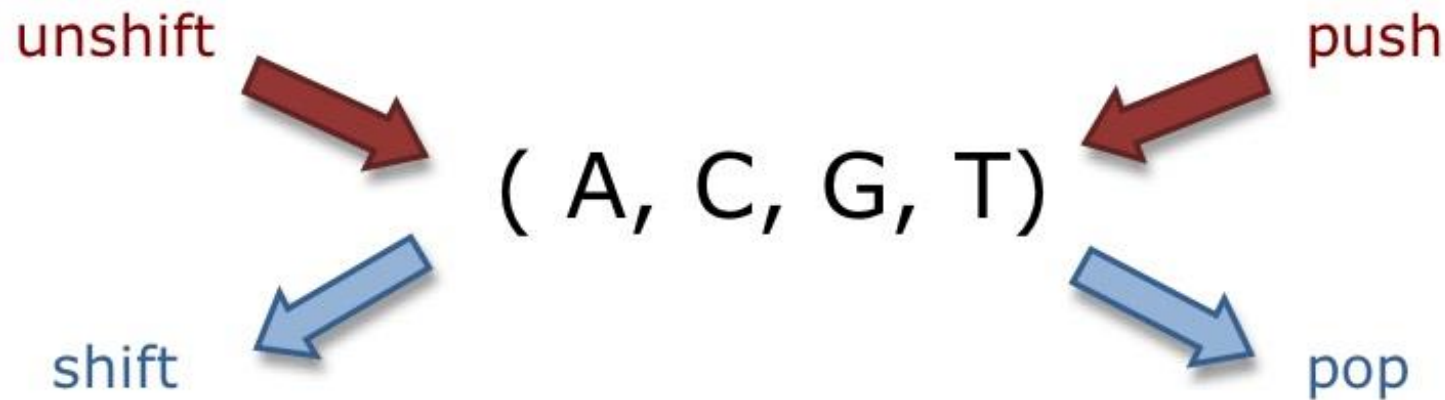


JavaScript Array is a List

LECTURE 6



JavaScript Array has List Operations



- Use `unshift`. It's like `push`, except it adds elements to the beginning of the array instead of the end.
 - `unshift/push` - add an element to the beginning/end of an array
 - `shift/pop` - remove and return the first/last element of an array



push() as add()

- To add an element to the end of an array, you can use the push method. Add .push to the array name, followed by the element you want to add inside parentheses, like this:

```
> var animals = [];  
> animals.push("Cat");  
1  
> animals.push("Dog");  
2  
> animals.push("Llama");  
3  
> animals;  
["Cat", "Dog", "Llama"]  
> animals.length;  
3
```



unshift as add(0, data)

- The act of running a method in computer-speak is known as **calling** the method. When you call the push method, two things happen.
- First, the element in parentheses is added to the array.
- Second, the new length of the array is returned. That's why you see those numbers printed out every time you call push.



unshift as add(0, data)

To add an element to the beginning of an array, you can use `.unshift(element)`, like this:

```
> animals;  
["Cat", "Dog", "Llama"]  
> animals[0]; // ❶  
"Cat"  
> animals.unshift("Monkey");  
4  
> animals;  
["Monkey", "Cat", "Dog", "Llama"]  
> animals.unshift("Polar Bear");  
5  
> animals;  
["Polar Bear", "Monkey", "Cat", "Dog", "Llama"]  
> animals[0];  
"Polar Bear"  
> animals [2];      // ❷  
"Cat"
```



pop() as remove()

- To remove the last element from an array, you can pop it off by **adding.pop()** to the end of the array name.
- The **pop** method can be particularly handy because it does two things: it removes the last element, *and* it returns that last element as a value.
- For example, let's start with our animals array, ["Polar Bear", "Monkey", "Cat", "Dog", "Llama"]. Then we'll create a new variable called **lastAnimal** and save the last animal into it by calling **animals.pop()**.



pop() as remove()

```
> animals;  
["Polar Bear", "Monkey", "Cat", "Dog", "Llama"]  
> var lastAnimal = animals.pop(); lastAnimal; // ❶  
"Llama"  
> animals;  
["Polar Bear", "Monkey", "Cat", "Dog"]  
> animals.pop(); // ❷  
"Dog"  
> animals;  
["Polar Bear", "Monkey", "Cat"]  
> animals.unshift(lastAnimal); // ❸  
4  
> animals;  
["Llama", "Polar Bear", "Monkey", "Cat"]
```



shift() as remove(0)

To remove and return the first element of an array, use .shift():

```
> animals;
```

```
["Llama", "Polar Bear", "Monkey", "Cat"]
```

```
> var firstAnimal = animals.shift();
```

```
> firstAnimal;
```

```
"Llama"
```

```
> animals;
```

```
["Polar Bear", "Monkey", "Cat"]
```




JavaScript Array for all Data Collections

Stack: push() and pop()

Queue: unshift() and pop(), or shift() and push()

Deque: push()/unshift() and pop()/shift()

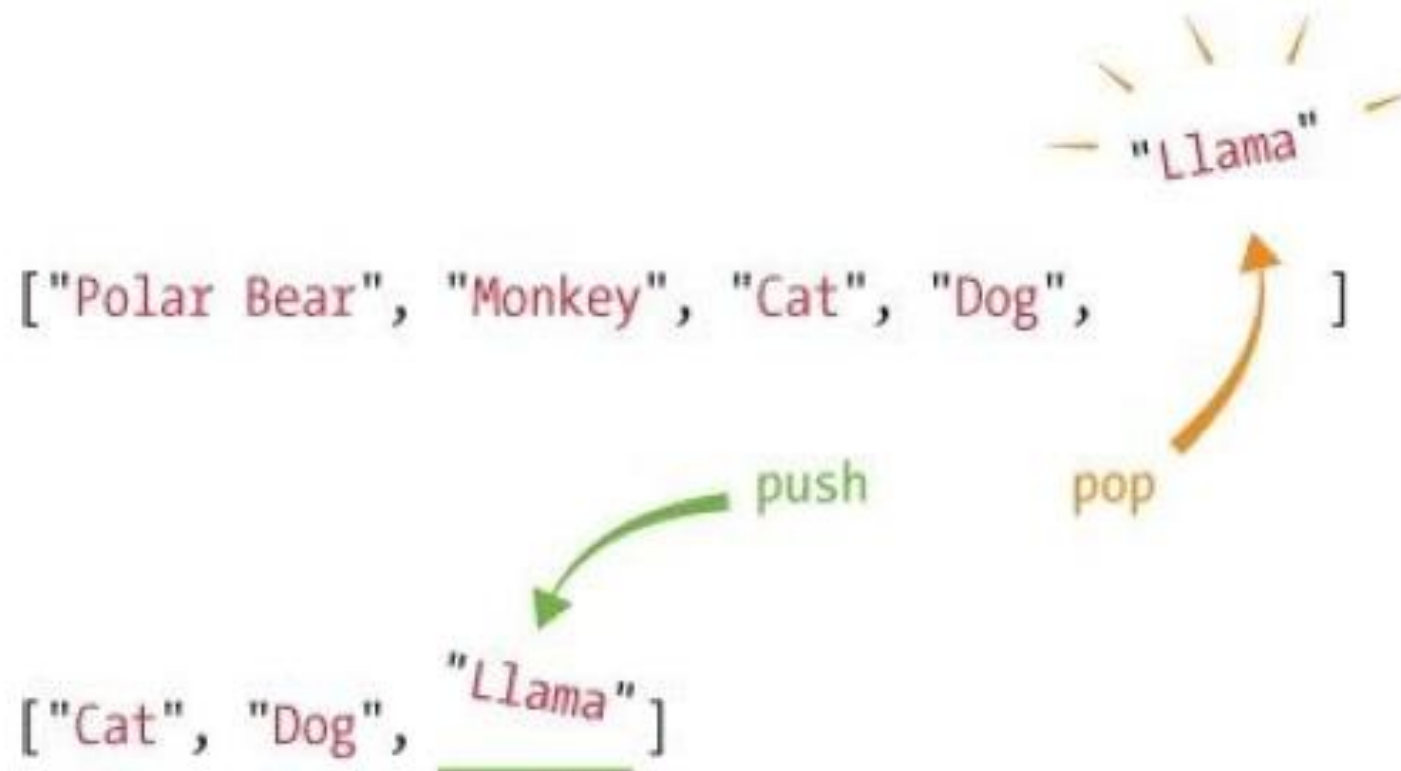


Stack Operations

Pushing and popping are a useful pair because sometimes you care about only the end of an array. You can push a new item onto the array and then pop it off when you're ready to use it. We'll look at some ways to use pushing and popping later in this chapter.

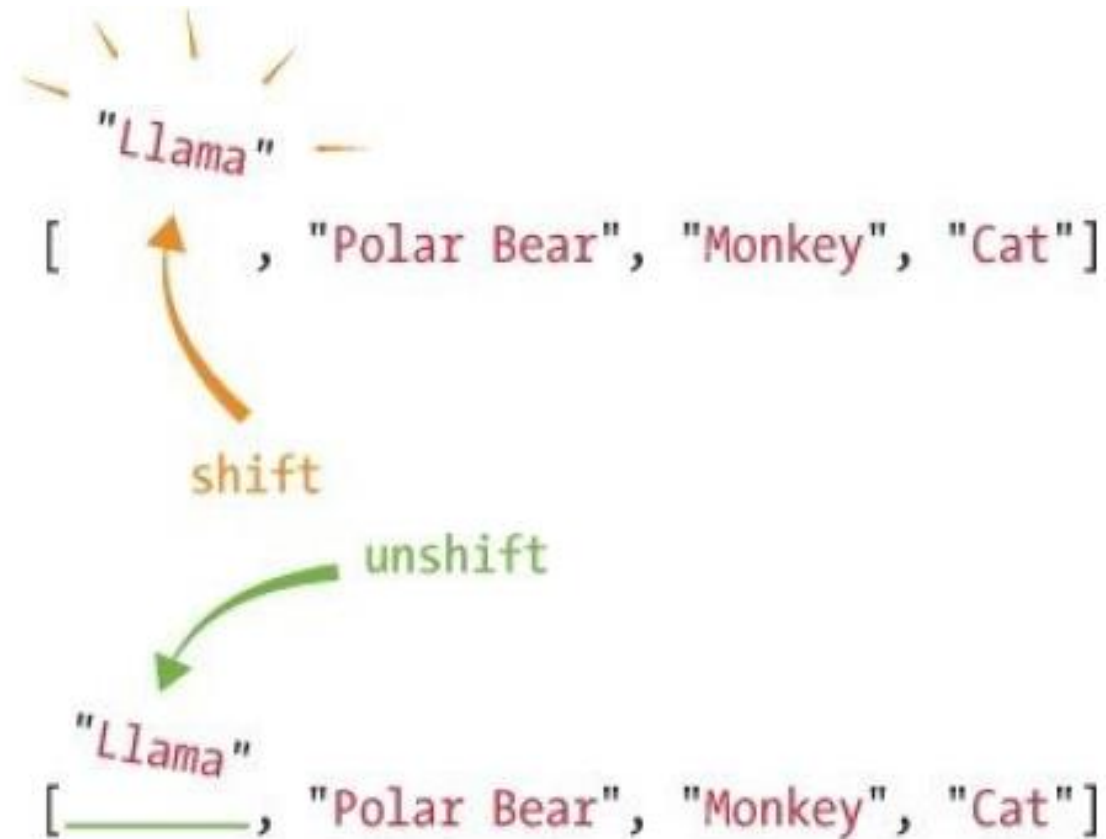


Stack Operations





Stack Operations





Adding Arrays

LECTURE 7



Adding Arrays

- `ary1.concat(ary2)`: adding array `ary2` to `ary1`.

```
item 1 = [ "a", "b", "c" ]
```

```
item 2 = [ "d", "e", "f" ]
```

```
Concat = [ "a", "b", "c", "d", "e", "f" ]
```



Joining Multiple Arrays

- You can use `concat` to join more than two arrays together. Just put the extra arrays inside the parentheses, separated by commas:

```
var furryAnimals = ["Alpaca", "Ring-tailed Lemur", "Yeti"];  
var scalyAnimals = ["Boa Constrictor", "Godzilla"];  
var featheredAnimals = ["Macaw", "Dodo"];  
var allAnimals = furryAnimals.concat(scalyAnimals, featheredAnimals);
```

```
> allAnimals;
```

```
["Alpaca", "Ring-tailed Lemur", "Yeti", "Boa Constrictor", "Godzilla", "Macaw", "Dodo"]
```

- Here the values from `featheredAnimals` get added to the very end of the new array, since they are listed last in the parentheses after the `concat` method.



Join Arrays into String

- `ary.join("Separators")`: adding all arrays into one string.

```
function myFunction() {  
    var fruits = ["Banana", "Orange", "Apple", "Mango"];  
    var x = document.getElementById("demo");  
    x.innerHTML = fruits.join();  
}
```

Banana,Orange,Apple,Mango



Turning an Array into a String

- You can use `.join()` to join all the elements in an array together into one big string.

```
var boringAnimals = ["Monkey", "Cat", "Fish", "Lizard"];
```

```
> boringAnimals.join();
```

```
"Monkey,Cat,Fish,Lizard"
```



Turning an Array into a String with Separator

- You can use `.join(separator)` to do the same thing, but with your own chosen separator between each value.
- The separator is whatever string you put inside the parentheses. For example, we can use three different separators: a hyphen with spaces on either side, an asterisk, and the word *sees* with spaces on either side.
- Notice that you need quote marks around the separator, because the separator is a string.

```
var boringAnimals = ["Monkey", "Cat", "Fish", "Lizard"];  
> boringAnimals.join(" - ");  
"Monkey - Cat - Fish - Lizard"  
> boringAnimals.join("*")  
"Monkey*Cat*Fish*Lizard"  
> boringAnimals.join(" sees ")  
"Monkey sees Cat sees Fish sees Lizard"
```



Turning an Array into a String

- This is useful if you have an array that you want to turn into a string. Say you have lots of middle names and you've got them stored in an array, along with your first and last name. You might be asked to give your full name as a string. Using `join`, with a single space as the separator, will join all your names together into a single string:

```
var myNames = ["Nicholas", "Andrew", "Maxwell", "Morgan"];  
myNames.join(" ");  
"Nicholas Andrew Maxwell Morgan"
```

- If you didn't have `join`, you'd have to do something like this, which would be really annoying to type out:

```
myNames[0] + " " + myNames[1] + " " + myNames[2] + " " + myNames[3];  
"Nicholas Andrew Maxwell Morgan"
```



Searching Arrays

LECTURE 8



indexOf()

Finding the Index of an Element in an Array

- To find the index of an element in an array, use `.indexOf(element)`. Here we define the array `colors` and then ask for the index positions of "blue" and "green" with `colors.indexOf("blue")` and `colors.indexOf("green")`.
- Because the index of "blue" in the array is 2, `colors.indexOf("blue")` returns 2. The index of "green" in the array is 1, so `colors.indexOf("green")` returns 1.

```
var colors = ["red", "green", "blue"];
```

```
> colors.indexOf("blue");
```

```
2
```

```
> colors.indexOf("green");
```

```
1
```



indexOf()

Finding the Index of an Element in an Array

- indexOf is like the reverse of using square brackets to get a value at a particular index; colors[2] is "blue", so colors.indexOf("blue") is 2:\

```
> colors[2];
```

```
"blue"
```

```
> colors.indexOf("blue");
```

```
2
```



indexOf()

Finding the Index of an Element in an Array

- Even though "blue" appears third in the array, its index position is 2 because we always start counting from 0. And the same goes for "green", of course, at index 1.

- If the element whose position you ask for is not in the array, JavaScript returns -1.

```
> colors.indexOf("purple");
```

-1

- This is JavaScript's way of saying "That doesn't exist here," while still returning a number.

- If the element appears more than once in the array, the indexOf method will return the first index of that element in the array.

```
var insects = ["Bee", "Ant", "Bee", "Bee", "Ant"];
```

```
> insects.indexOf("Bee");
```

0

Goal	Array	String
Copy	.slice()	.slice()
Iterate through	.forEach()	
Extract section	.slice()	.slice(), .substr(), .substring()
Remove/insert value	.splice()	
Modify given a rule	.map()	.replace()
Confirm that it has certain contents	.includes()	.includes()
Get keys	.keys()	
Extract given a rule	.filter()	.match()
Append	.push(), .concat()	.concat(), + operator
Prepend	.unshift()	
Remove from end	.pop()	
Remove from beginning	.shift()	
Convert to type	.split(), Array.from()	String(), .join()
Confirm type	Array.isArray()	typeof
Reduce to one value	.reduce(), .reduceRight()	
Find index of value	.find, .findIndex()	.indexOf(), .search()
Find last index of value	.lastIndexOf()	.lastIndexOf()
Repeat value	.fill()	.repeat()
Reverse	.reverse()	
Check if any value matches test	.some()	.includes()
Check if all values match test	.every()	
Sort	.sort()	
Get length	.length()	.length()



Project 1: Backtracking

LECTURE 9



Building the Array with Push

- Here we create an empty array named landmarks and then use push to store all the landmarks you pass on the way to your friend's house.

```
var landmarks = [];  
landmarks.push("My house");  
landmarks.push("Front path");  
landmarks.push("Flickering streetlamp");  
landmarks.push("Leaky fire hydrant");  
landmarks.push("Fire station");  
landmarks.push("Cat rescue center");  
landmarks.push("My old school");  
landmarks.push("My friend's house");
```



Going in Reverse with pop

- Once you arrive at your friend's house, you can inspect your array of landmarks.
- Sure enough, the first item is "My house", followed by "Front path", and so on through the end of the array, with the final item "My friend's house".
- When it's time to go home, all you need to do is pop off the items one by one, and you'll know where to go next.



Pop Operations

```
> landmarks.pop();  
"My friend's house"  
> landmarks.pop();  
"My old school"  
> landmarks.pop();  
"Cat rescue center"  
> landmarks.pop();  
"Fire station"  
> landmarks.pop();  
"Leaky fire hydrant"  
> landmarks.pop();  
"Flickering streetlamp"  
> landmarks.pop();  
"Front path"  
> landmarks.pop();  
"My house"
```



Back to Home

Demo Program: Push_Pop.html

Go Brackets!!!

Push operations:

My house
Front path
Flickering streetlamp
Leaky fire hydrant
Fire station
Cat rescue center
My old school
My friend's house

Pop operations:

My friend's house
My old school
Cat rescue center
Fire station
Leaky fire hydrant
Flickering streetlamp
Front path
My house

```
1 ▼ <html>
2 ▼ <body>
3 ▼ <script>
4   var landmarks = [];
5   landmarks.push("My house");
6   landmarks.push("Front path");
7   landmarks.push("Flickering streetlamp");
8   landmarks.push("Leaky fire hydrant");
9   landmarks.push("Fire station");
10  landmarks.push("Cat rescue center");
11  landmarks.push("My old school");
12  landmarks.push("My friend's house");
13  // Push operations
14  document.write("<h3>Push operations: </h3><p>");
15 ▼ for (var i=0; i<landmarks.length; i++){
16     document.write(landmarks[i]+"<br>");
17 }
18 document.write("</p>");
19 // pop operations
20 document.write("<h3>Pop operations: </h3><p>");
21 var len = landmarks.length;
22 ▼ for (var i=0; i<len; i++){
23     var popString = landmarks.pop();
24     document.write(popString+"<br>");
25 }
26 document.write("</p>");
27 </script>
28 </body>
29 </html>
```



Project 2: Decision Maker

LECTURE 10



Using Math.random()

- We can produce random numbers using a special method called Math.random(), which returns a random number between 0 and 1 each time it's called. Here's an example:

```
> Math.random();  
0.8945409457664937  
> Math.random();  
0.3697543195448816  
> Math.random();  
0.48314980138093233
```




Using Math.random()

Count parameter

- If you want a bigger number, just multiply the result of calling Math.random(). For example, if you wanted numbers between 0 and 10, you would multiply Math.random() by 10:

```
> Math.random() * 10;
```

```
7.648027329705656
```

```
> Math.random() * 10;
```

```
9.7565904534421861
```

```
> Math.random() * 10;
```

```
0.21483442978933454
```



Rounding Down with Math.floor()

Quantization

- We can't use these numbers as array indexes, though, because indexes have to be whole numbers with nothing after the decimal point. To fix that, we need another method called Math.floor().
- This takes a number and rounds it down to the whole number below it (basically getting rid of everything after the decimal point).

```
> Math.floor(3.7463463);
```

3

```
> Math.floor(9.9999);
```

9

```
> Math.floor(0.793423451963426);
```

0



Rounding Down with Math.floor()

Similar to Java (int) (Math.random()* count)

- We can combine these two techniques to create a random index. All we need to do is multiply Math.random() by the length of the array and then call Math.floor() on that value. For example, if the length of the array were 4, we would do this:

```
> Math.floor(Math.random() * 4);
```

2 // could be 0, 1, 2, or 3

- Every time you call the code above, it returns a random number from 0 to 3 (including 0 and 3). Because Math.random() always returns a value less than 1, Math.random() * 4 will never return 4 or anything higher than 4. Now, if we use that random number as an index, we can select a random element from an array:

```
var randomWords = ["Explosion", "Cave", "Princess", "Pen"];
```

```
var randomIndex = Math.floor(Math.random() * 4);
```

```
> randomWords[randomIndex];
```

"Cave"



Rounding Down with Math.floor()

- Here we use `Math.floor(Math.random() * 4);` to pick a random number from 0 to 3. Once that random number is saved to the variable `randomIndex`, we use it as an index to ask for a string from the array `randomWords`.
- In fact, we could shorten this by doing away with the `randomIndex` variable altogether and just say:

```
> randomWords[Math.floor(Math.random() * 4)];  
"Princess"
```



The Complete Decision Maker

Practice with HTML online editor

```
var phrases = [  
  "That sounds good",  
  "Yes, you should definitely do that",  
  "I'm not sure that's a great idea",  
  "Maybe not today?",  
  "Computer says no."  
];  
// Should I have another milkshake?  
> phrases[Math.floor(Math.random() * 5)];  
"I'm not sure that's a great idea"  
// Should I do my homework?  
> phrases[Math.floor(Math.random() * 5)];  
"Maybe not today?"
```



Decision Maker

Demo Program: [decisionmaker.html](#)

Go Brackets!!!

A diagram illustrating a simple addition problem: $2 + 4 =$ followed by a box containing the number 0. Below the equation are two buttons labeled "Check" and "Reset". Arrows indicate a flow: one arrow points from the left towards the "Check" button, another from the "Check" button towards the equation, and a third from the "Reset" button towards the equation. A fourth arrow points from the bottom right towards the "Reset" button.

```
60 <body onload="reset()">  
61   <div>  
62     <label id="equation">  
63       </label>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
64     <input id="ans" onchange="getAnswer()" placeholder="0">  
65   </div>  
66   <br>  
67   <button onclick="checkAnswer()">&nbsp;Check&nbsp;</button><button  
    onclick="reset()">&nbsp;Reset&nbsp;</button>  
68   <br>  
69   <p id="messagePanel"></p>  
70 </body>
```



Label (Randomized)

```
29 ▼ function reset(){  
30     //alert("I am there");  
31     a = Math.floor(Math.random()*10);  
32     b = Math.floor(Math.random()*10);  
33     var outString = "   " + a.toString() + " + " + b.toString() + " = ";  
34     //alert(outString);  
35     document.getElementById("equation").innerHTML = outString;  
36     answer=0;  
37     document.getElementById("ans").value| = "0";  
38     var messagePanel = document.getElementById("messagePanel");  
39     messagePanel.innerHTML = "";  
40 }
```




Input Form onchange="getAnswer()"

```
42 ▼    function getAnswer(){  
43        var v = document.getElementById("ans").value;  
44        //alert(v);  
45        answer = parseInt(v);  
46        //alert(answer+1);  
47    }
```



Check Answer

```
48 ▼    function checkAnswer(){
49        //alert("I am here");
50        var equal = (answer == (a+b));
51        var messagePanel = document.getElementById("messagePanel");
52        if (equal)
53            messagePanel.innerHTML = "&nbsp;&nbsp;&nbsp;Correct !";
54 ▼    else{
55        messagePanel.innerHTML = "&nbsp;&nbsp;&nbsp;Wrong !";
56    }
57 }
```



Project 3: Random Insult Generator

LECTURE 11



Creating a Random Insult Generator

- We can extend the decision maker example to create a program that generates a random insult every time you run it!

```
var randomBodyParts = ["Face", "Nose", "Hair"];
var randomAdjectives = ["Smelly", "Boring", "Stupid"];
var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];
// Pick a random body part from the randomBodyParts array:
❶ var randomBodyPart = randomBodyParts[Math.floor(Math.random() * 3)];
// Pick a random adjective from the randomAdjectives array:
❷ var randomAdjective = randomAdjectives[Math.floor(Math.random() * 3)];
// Pick a random word from the randomWords array:
❸ var randomWord = randomWords[Math.floor(Math.random() * 5)];
// Join all the random strings into a sentence:
var randomInsult = "Your " + randomBodyPart + " is like a " +
randomAdjective + " " + randomWord + "!!!";
randomInsult;
"Your Nose is like a Stupid Marmot!!!"
```



Creating a Random Insult Generator

- Here we have three arrays, and in lines ❶, ❷, and ❸, we use three indexes to pull a random word from each array. Then, we combine them all in the variable `randomInsult` to create a complete insult.
- At ❶ and ❷ we're multiplying by 3 because `randomAdjectives` and `randomBodyParts` both contain three elements. Likewise, we're multiplying by 5 at ❸ because `randomWords` is five elements long.
- Notice that we add a string with a single space between `randomAdjective` and `randomWord`. Try running this code a few times — you should get a different random insult each time!



Random Insult Generator

Demo Program: [insult.html](#)

Go Brackets!!!

```

1 <html>
2 <body>
3 <script>
4 function insult(){
5   var randomBodyParts = ["Face", "Nose", "Hair"];
6   var randomAdjectives = ["Smelly", "Boring", "Stupid"];
7   var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];
8   // Pick a random body part from the randomBodyParts array:
9   var randomBodyPart = randomBodyParts[Math.floor(Math.random() * 3)];
10  // Pick a random adjective from the randomAdjectives array:
11  var randomAdjective = randomAdjectives[Math.floor(Math.random() * 3)];
12  // Pick a random word from the randomWords array:
13  var randomWord = randomWords[Math.floor(Math.random() * 5)];
14  // Join all the random strings into a sentence:
15  var randomInsult = "Your " + randomBodyPart + " is like a " +
16                    randomAdjective + " " + randomWord + "!!!";
17
18  // output strings
19  var insultment = "<h3>Random Insult Generator: </h3>"+"<p>" + randomInsult + "</p>";
20
21  var division = document.getElementById("AA");
22    division.innerHTML = insultment;
23  }
24 </script>
25 <button onclick="insult()">&nbsp; Click to Show Insultment &nbsp;</button>
26 <div id="AA"></div>
27 </body>
28 </html>

```

Click to Show Insultment

Random Insult Generator:

Your Nose is like a Boring Fly!!!

Click to Show Insultment

Random Insult Generator:

Your Nose is like a Smelly Rat!!!

Click to Show Insultment

Random Insult Generator:

Your Nose is like a Boring Stick!!!