JAVASCRIPT (JS) DATA STRUCTURES

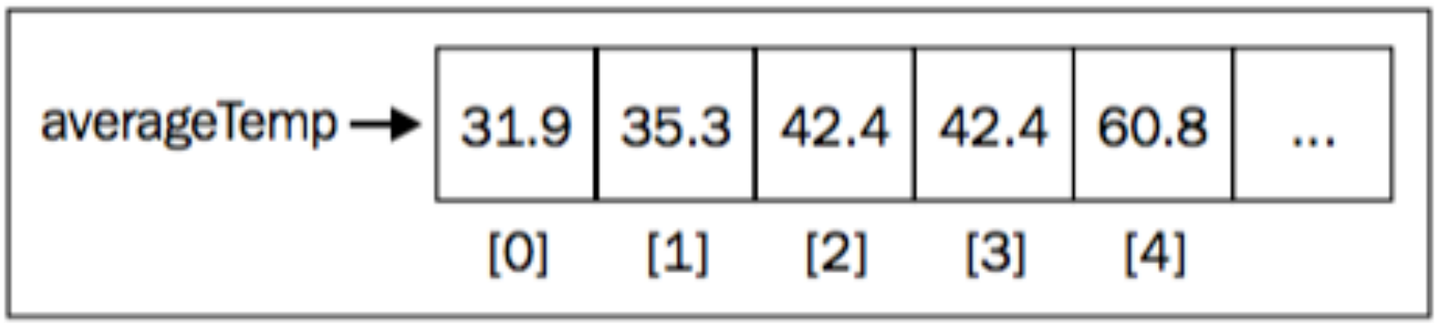
# ARRAYS

# ARRAYS

- is the simplest memory data structure
- all programming language have a built-in array type.
- An array store a sequence of values that are all of the same type
  - JS allow us to create arrays with values from different data types
  - The best practices tell us that we cannot do that (most languages not have this capability)

# JAVASCRIPT'S ARRAY STRUCTURE

```
averageTemp[0] = 31.9;
averageTemp[1] = 35.3;
averageTemp[2] = 42.4;
averageTemp[3] = 42.4;
averageTemp[4] = 60.8;
```



# CREATING AND INITIALIZING ARRAYS

```
var daysOfWeek = new Array(); //{1}
var daysOfWeek = new Array(7); //{2}
var daysOfWeek = new Array('Sunday', ‘Monday','Tuesday',
  'Wednesday','Thursday', 'Friday', 'Saturday'); //{3}
var daysOfWeek = [];//{4}
var daysOfWeek = ['Sunday', 'Monday', 'Tuesday',
  ‘Wednesday’,'Thursday', 'Friday', ‘Saturday’];//{5}
```

- **{1}**: we declare and instantiate a new array using the keyword **new**.
- **{2}**: using the keyword new, we create a new array specifying the length of array
- **{3}**: we create an array by passing de array elements directly to its constructor.
- **{4}**: Like **{1},** we create a new array, although, using the brackets (**[]**).
- **{5}**: we create a new array initialized with some elements, like **{3}**, but using the brackets (**[]**).

# LENGTH PROPERTY

- If we would like to know how many elements are in the array, we can use the `length` property.
- Considering the line {5} of last exemple, the following code will give an output of 7:

```
console.log(daysOfWeek.length);
```

# ACCESS ARRAY POSITIONS

- To access a particular position of the array, we use brackets, passing the numeric position we would like to know the value of or assign a new value to.
- For example, let's say we would like to output all elements from the `daysOfWeek` array. To do so, we need to loop the array and print the elements, as follows:

```
for (var i=0; i<daysOfWeek.length; i++){
        console.log(daysOfWeek[i]);
 }
```

# EXERCÍCIO

```
<script>
    var fibonacci = [];
    fibonacci[1] = 1;
    fibonacci[2] = 1;
    for(var i = 3; i < 20; i++){
        fibonacci[i] = fibonacci[i-1] + fibonacci[i-2];
    }
    for(var i = 1; i<fibonacci.length; i++){
        console.log(fibonacci[i]);
    }
</script>
```

# ADDING ELEMENTS IN THE LAST POSITION

```
var numbers = [0,1,2,3,4,5,6,7,8,9];
```

- using the index in brackets []

```
numbers[numbers.length] = 10;
```

- using push method

```
numbers.push(11);
numbers.push(12, 13);
```

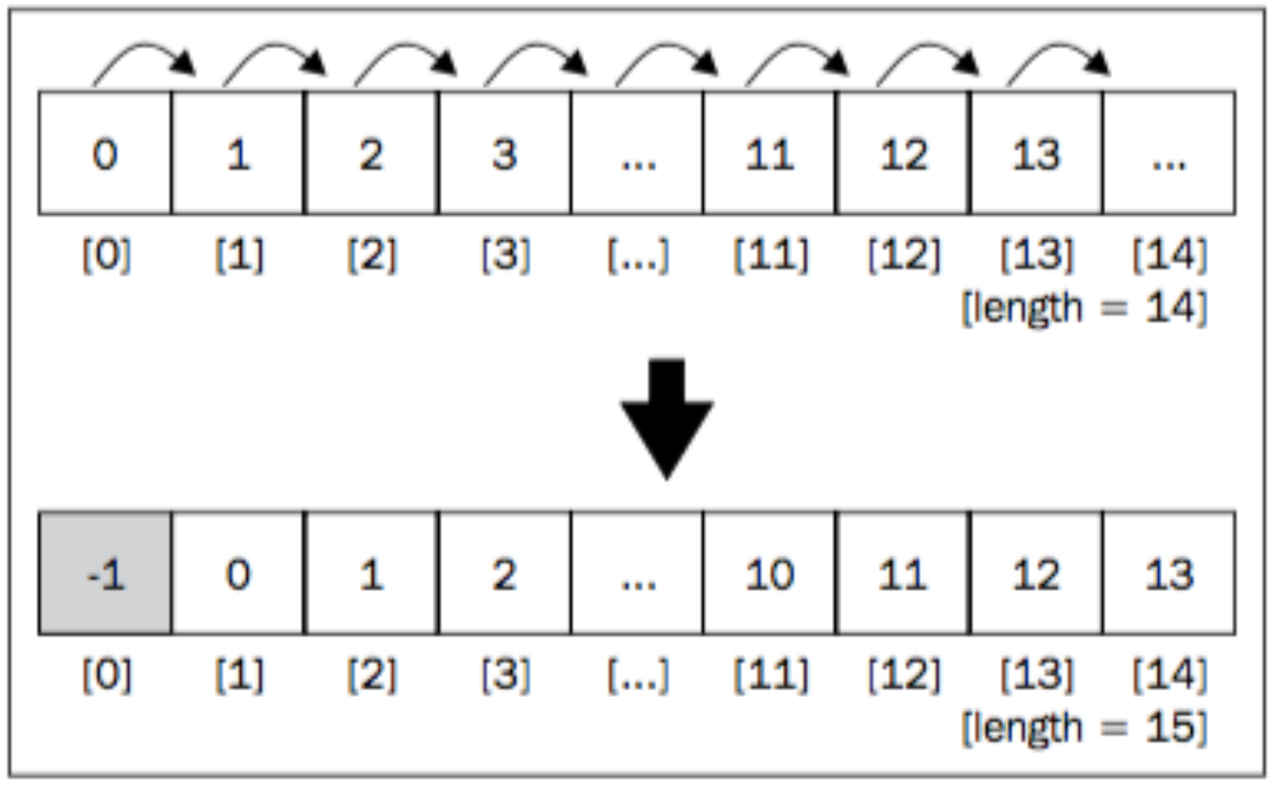
# ADDING ELEMENTS IN THE FIRST POSITION

```
var numbers = [0,1,2,3,4,5,6,7,8,9,10,11,12,13];
```

- using the index in brackets []

```
for (var i=numbers.length; i>=0; i--){
   numbers[i] = numbers[i-1];
}
numbers[0] = -1;
```

# ADDING ELEMENTS IN THE FIRST POSITION



# ADDING ELEMENTS IN THE FIRST POSITION

- using unshift method

```
numbers.unshift(-2);
numbers.unshift(-4, -3);
```

```
var numbers = [-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12];
```

- using pop method

```
numbers.pop();
```

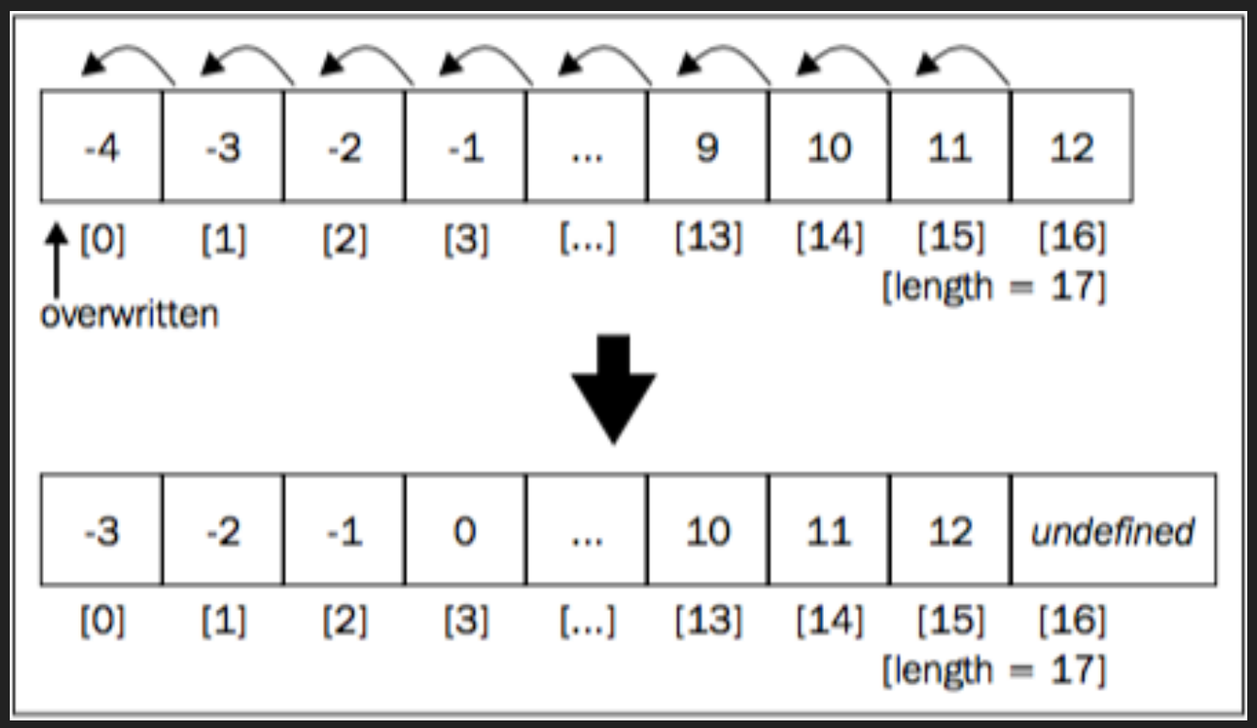
The push and pop methods allow an array to emulate a basic **stack** data structure, which is the subject of the next chapter.

# FOR

```
var numbers = [-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12];
```

- using the index in brackets []

```
for (var i=0; i<numbers.length; i++){
   numbers[i] = numbers[i+1];
}
```



```
var numbers = [-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12];
```

- using the index in brackets []

```
numbers.shift();
```

The `shift` and `unshift` methods allow an array to emulate a basic **queue** data structure, which is the subject of *Chapter 4, Queues*.

# REMOVE ELEMENTS FROM ANY ARRAY POSITION

```
var numbers = [-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12];
```

- `splice` method: to remove , simply specifying the **position/index** we would like to delete from and **how many** elements we would like to remove.

- the following code will remove three elements starting from 5 index. This means numbers[5], numbers[6] e numbers[7] will be removed from the numbers array (numbers 2, 3, and 4)

```
numbers.splice(5,3);
```

```
var numbers = [-3,-2,-1,0,1,5,6,7,8,9,10,11,12];
```

```
var numbers = [-3,-2,-1,0,1,5,6,7,8,9,10,11,12];
```

- `splice` method: to add more than one number, simply specifying the **position/index** we would like to delete from and **how many** elements we would like to remove (0 in this case) and the **values we would link to insert** into the array.

- the following code will add three elements starting from 5 index. This means that will be insert to the numbers array the numbers 2, 3, and 4.

```
numbers.splice(5,0,2,3,4);
```

```
var numbers = [-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12];
```


JAVASCRIPT (JS) DATA STRUCTURES

# ARRAY METHODS

# ARRAYS IN JAVASCRIPT ARE MODIFIED OBJECTS, MEANING THAT EVERY ARRAY THAT WE CREATE HAS A FEW METHODS AVAILABLE TO BE USED

Loiane Groner

# JS ARRAY METHODS

| Method | Description |
|---|---|
| `concat` | Joins multiple arrays and returns a copy of the joined arrays |
| `every` | Calls a function for every element of the array until `false` is returned |
| `filter` | Creates an array with each element that evaluates to `true` in the function provided |
| `forEach` | Executes a specific function on each element of the array |
| `join` | Joins all the array elements into a string |
| `indexOf` | Searches the array for specific elements and returns its position |
| `lastIndexOf` | Returns the last item in the array that matches the search criteria and returns its position |
| `map` | Creates a new array with the result of calling the specified function on each element of the array |
| `reverse` | Reverses the array so the last items become the first and vice versa |
| `slice` | Returns a new array from the specified index |
| `some` | Passes each element through the supplied function until `true` is returned |
| `sort` | Sorts the array alphabetically or by the supplied function |
| `toString` | Returns the array as a string |
| `valueOf` | Like the method `toString`, this returns the array as a string |

# EVERY METHOD

```
var isEven = function (x) {
   // returns true if x is a multiple of 2.
   console.log(x);
   return (x % 2 == 0) ? true : false;
};
var numbers = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
numbers.every(isEven);
```

- The every method will iterate each element of the array until the return of the function is `false`.
- In this case, our first element of the `numbers` array is the number 1. 1 is not a multiple of 2 (it is an odd number), so the `isEven` function will return `false` and this will be the only time the function will be executed.

# SOME METHOD

```
var isEven = function (x) {
   // returns true if x is a multiple of 2.
   console.log(x);
   return (x % 2 == 0) ? true : false;
};
var numbers = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
numbers.some(isEven);
```

- The same behavior as the `every` method, however, the `some` method will iterate each element of the array until the return of the function is `true`.
- In our case, the first even number of our `numbers` array is 2 (the second element). The first element that will be iterated is number 1; it will return `false`. Then, the second element that will be iterated is number 2, and it will return `true`—and the iteration will stop.

# FOREACH

- If we need the array to be completely iterated no matter what, we can use the forEach function.
- It has the same result as using a for loop with the function's code inside it:

```
var numbers.forEach(function(x){
   console.log((x % 2 == 0));
});
```

# MAP

- Iterate and return a new array with a result.

```
var isEven = function (x) {
   return (x % 2 == 0) ? true : false;
};
var numbers = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
var myMap = numbers.map(isEven);
```

- The myMap array will have the following values: `[false, true, false, true, false, true, false, true, false, true, false, true, false, true, false]`

# FILTER

- Iterate and It returns a new array with the elements that the function returned true:

```
var isEven = function (x) {
   return (x % 2 == 0) ? true : false;
};
var numbers = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
var evenNumbers = numbers.filter(isEven);
```

- The myMap array will have the following values: `[2, 4, 6, 8, 10, 12, 14]`

# REDUCE

- The reduce method also receives a function with the following parameters: `previousValue`, `currentValue`, `index`(optional), and `Array`(optional).
- We can use this function to return a value that will be added to an accumulator

```
var numbers = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
var total = numbers.reduce(function (previous, current){
   return previous + current;
});
```

- total is going to be 120.

# A LITTLE BIT ABOUT...

- map(), filter() and reduce() - http://desenvolvimentoparaweb.com/javascript/map-filter-reduce-javascript/

# SORTING AN ARRAY

- JavaScript also has a sorting method and a couple of searching methods available.
- we can apply the `reverse` method, where the last item will be the first and vice versa;

```
numbers.reverse();
```

- Then, we can apply the `sort` method;

```
numbers.sort();
```

# SORTING RULES

- We can also write our own comparison function

```
numbers.sort(function(a,b){
        return a-b;
});
```

- This code will return a negative number if b is bigger than a, a positive number if a is bigger than b, and zero if they are equal

# SORTING RULES

- The previous code can be represented by the following code as well

```
function compare(a, b) {
  if (a < b) {
    return -1;
  }
  if (a > b) {
    return 1;
  }
  // a must be equal to b
  return 0;
}
numbers.sort(compare);
```

# SORTING RULES

- We can sort an array with any type of object in it

```
var friends = [
        {name: 'John', age: 30},
        {name: 'Ana', age: 20},
        {name: 'Chris', age: 25}
];
function comparePerson(a, b){
   if (a.age < b.age){
      return -1
   }
   if (a.age > b.age){
      return 1
   }
   return 0;
}
console.log(friends.sort(comparePerson));
```

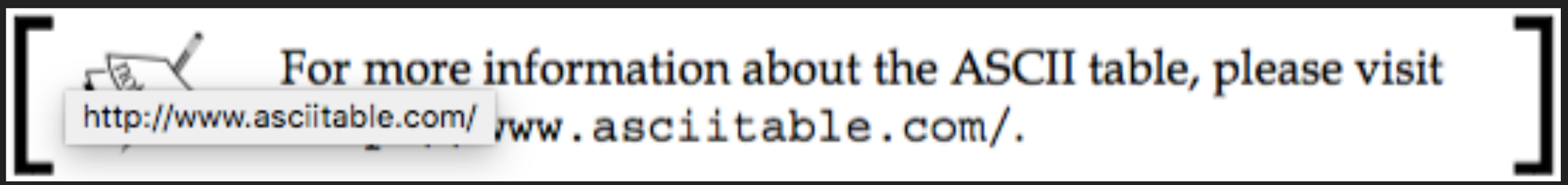
# SORTING STRINGS

- JavaScript compares each character according to its ASCII value. For example:

```
Var names =['Ana', 'ana', 'john', ‘John'];
console.log(names.sort());

=> ["Ana", "John", "ana", “john”]
```

- A, J, a, and j have the decimal ASCII values of A: 65, J: 74, a: 97, and j: 106.



# SORTING STRINGS

- if we pass a compareFunction that contains the code to ignore the case of the letter, we will have the output ["Ana", "ana", "John", "john"], as follows:

```
names.sort(function(a, b){
    if (a.toLowerCase() < b.toLowerCase()){
        return -1
    }
    if (a.toLowerCase() > b.toLowerCase()){
        return 1
    }
    return 0;
});
```

# OUTPUTTING THE ARRAY INTO A STRING

- to output all the elements of the array into a single string, we can use the toString method as follows:

```
console.log(numbers.toString());
```

- to separate the elements by a different separator, such as -, we can use the join method to do just that:

```
var numbersString = numbers.join('-');
console.log(numbersString);
```

# UM POUCO MAIS SOBRE ARRAYS...

There are some great resources that you can use to boost your knowledge about arrays and their methods:

- The first one is the arrays page from w3schools at `http://www.w3schools.com/js/js_arrays.asp`
- The second one is the array methods page from w3schools at `http://www.w3schools.com/js/js_array_methods.asp`

- Mozilla also has a great page about arrays and their methods with great examples at `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array` (`http://goo.gl/vu1diT`)
- There are also great libraries that are very useful when working with arrays in JavaScript projects:
  - The Underscore library: `http://underscorejs.org/`
  - The Lo-Dash library: `http://lodash.com/`