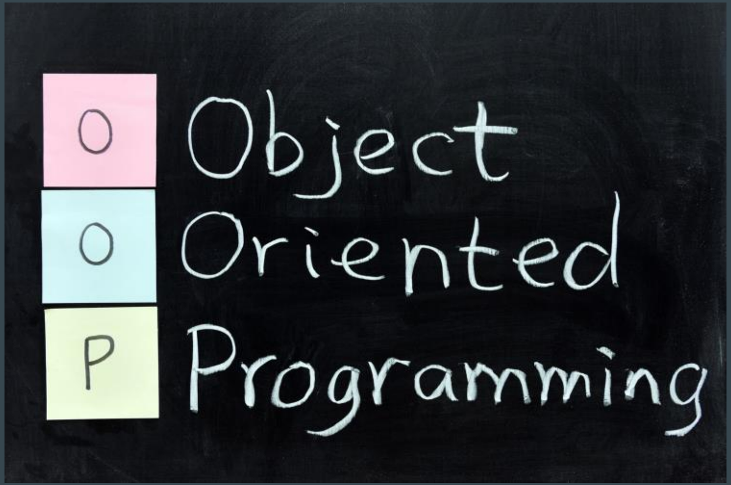
M04 - Object-Oriented Programming

•••

Objects

Index



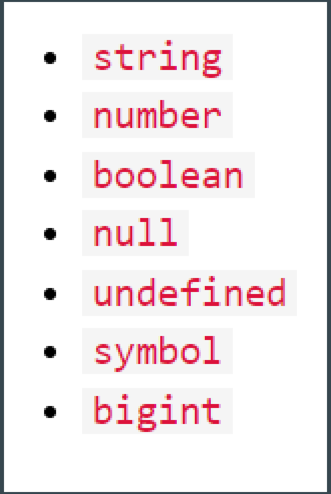
Objects

1. Creation

Objects

1. Creation

- So far we've only seen primitive data types
  - strings ("John Doe")
  - numbers (3.14)
  - booleans (true, false)
  - null and undefined
  - A primitive value is a value that has no associated properties or methods

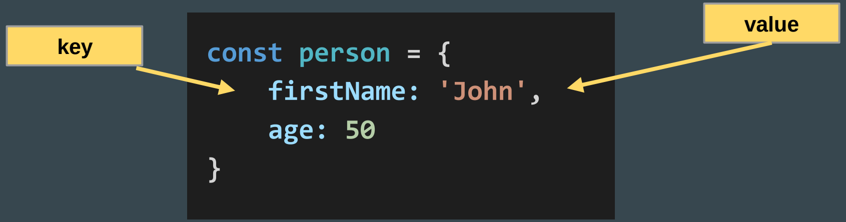


- `string`
- `number`
- `boolean`
- `null`
- `undefined`
- `symbol`
- `bigint`

Objects

1. Creation

- An object is
  - a complex data type
  - represents an instance of an entity to model
  - contains a set of key-value pairs

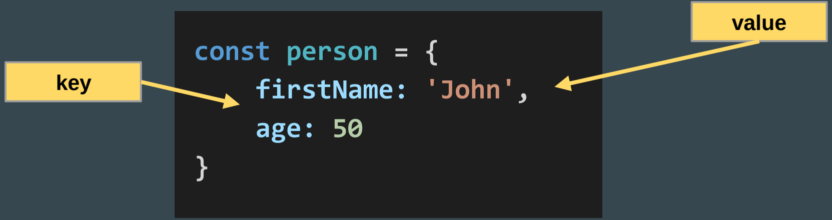


```
const person = {
    firstName: 'John',
    age: 50
}
```

Objects

1. Creation

- The content of an object is made up of properties (separated by commas)
- The properties consist of a key:value pair
  - keys must be strings or symbols
  - values can be of any type (including functions, arrays, or other objects)



```
const person = {
    firstName: 'John',
    age: 50
}
```

- As an example, the key firstName has the value “John”
- Objects can be empty

```
const person = {}
```

Objects

1. Creation

- Object Comparison
  - In JavaScript, objects are a reference
    - Two distinct objects are never the same, even with the same properties
    - They point to a completely different memory address
    - Objects that share a common reference are true in the comparison

```
const num = 2
const str = '2'

console.log(num == str)     // true
console.log(num === str)    // false
```

```
const obj1 = {name: 'John'}
const obj2 = {name: 'John'}

console.log(obj1 == obj2)     // false
console.log(obj1 === obj2)    // false
```

```
const obj1 = {name: 'John'}
const obj2 = obj1

console.log(obj1 == obj2)     // true
console.log(obj1 === obj2)    // true
```

Objects

1. Creation

- There are several ways to create objects:
    a. create a single object using an object literal
    b. create a single object using the new keyword
    c. define an object constructor, and then create objects of the constructor type
    d. using classes (studied later)

Objects

1. Creation

- Create an object literal
    - list of key:value pairs inside {}
    - simple and readable
    - possibility of creating the object in a single declaration

```
const person = {firstName: 'John', lastName: 'Doe', age: 50, eyeColor: 'blue'}
// OR
const person = {
    firstName: 'John',
    lastName: 'Doe',
    age: 50,
    eyeColor: 'blue'
}
```

Objects

1. Creation

- Create an object literal
  - based on variables

```
const firstName = 'John'
const age = 50

const person = { firstName, age }
```

  - or the reverse (destructuring)
    - breaking the structure of an object
    - you can extract data from arrays or objects in distinct variables

```
const emp = { name: 'Peter', age: 22 }
const { name, age } = emp

console.log(name)  // Peter
console.log(age)   // 22
```

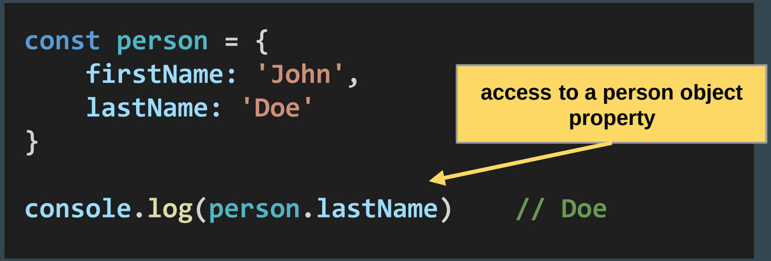
Objects

2. Properties

Objects

2. Properties

- A JavaScript object is a collection of disordered properties
- Properties can usually be added, changed and removed
- Syntax: *object.property*



```javascript
const person = {
    firstName: 'John',
    lastName: 'Doe'
}

console.log(person.lastName)     // Doe
```

Objects

2. Properties

- Alternative syntaxes:
  - object.property (usual)
  - object[“property”]
  - object[expression]

```
const person = {
    firstName: 'John',
    lastName: 'Doe'
}

console.log(person.lastName)

console.log(person['lastName'])

const x = 'lastName'
console.log(person[x])
```

Objects

2. Properties iteration:

- The for...in statement iterates through the properties of an object
- The number of iterations in the cycle is equal to the number of properties
- There are other techniques, but this is the fastest!

```
const person = {
    firstName: 'John',
    lastName: 'Doe',
    age: 50
}

let text = ''
for (let prop in person) {
    text += `name: ${prop} value: ${person[prop]} \n`
}

console.log(text)

/*
    name: firstname value: John
    name: lastName value: Doe
    name: age value: 50
*/
```

iterating over all the properties of the person object

Objects

2. Properties

- Adding properties:
    - You can add new properties to an existing object, just give it a value

```
const person = {
    firstName: 'John',
    lastName: 'Doe',
    age: 50
}

person.city = 'Porto'

console.log(person.city)   // Porto
```

It may seem that this line would cause an error, but there is no problem. This is because **const** contains a **reference** to the **person** object. The line makes changes within the object, but does not change the reference.

Objects

2. Properties

- Removing properties:
  - The delete keyword deletes a property from an object

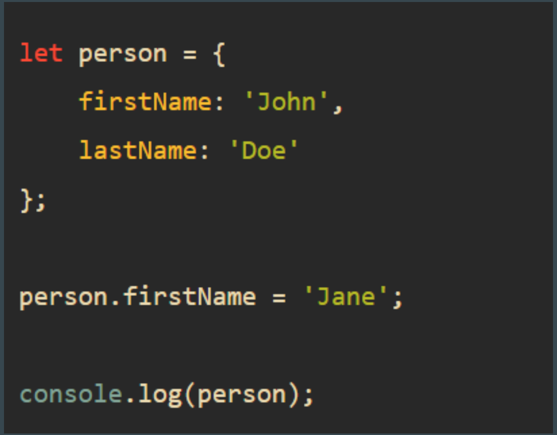
```
const person = { firstName: 'John', age: 50 }

delete person.age
console.log(person.age)   // undefined
```

  - After removal, the property cannot be used before being added again

Objects

2. Properties

- Modifying the value of a property:
  - Use the operator =



```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};

person.firstName = 'Jane';

console.log(person);
```

Objects

2. Properties

- Checking if a property exists:
  - To check if a property exists in an object, you use the in operator

```
let employee = {
    firstName: 'Peter',
    lastName: 'Doe',
    employeeId: 1
};

console.log('ssn' in employee);                    // false
console.log('employeeId' in employee);             // true
```

Objects

2. Properties

- Summary:

  - An object is a collection of key-value pairs.

  - Use the dot notation (.) or array-like notation ([]) to access a property of an object.

  - The delete operator removes a property from an object.

  - The in operator check if a property exists in an object.

Objects

3. Methods

Objects

- Reserved word this
  - the this keyword refers to an object
  - In an object method, this refers to the current object.

```
const person = {
    firstName: 'John',
    lastName: 'Doe',
    fullName: function () {
        return `${this.firstName} ${this.lastName}`
    }
}

// Method invocation
console.log(person.fullName())   // John Doe
```

ect.

Shortened syntax

```
fullName() {
    return `${this.firstName} ${this.lastName}`
}
```

Objects

3. Methods

- Reserved word this

```
const person = {
    firstName: 'John',
    lastName: 'Doe',
    fullName: function () {
        return `${this.firstName} ${this.lastName}`
    }
}

// Method invocation
console.log(person.fullName())   // John Doe
```

To access the object itself, a method can use the keyword **this**

- this is not a variable. It is a keyword. You cannot change the value of this
- JavaScript methods are actions that can be performed on objects.
- A JavaScript method is a property containing a function definition
- **Methods are functions stored as object properties**

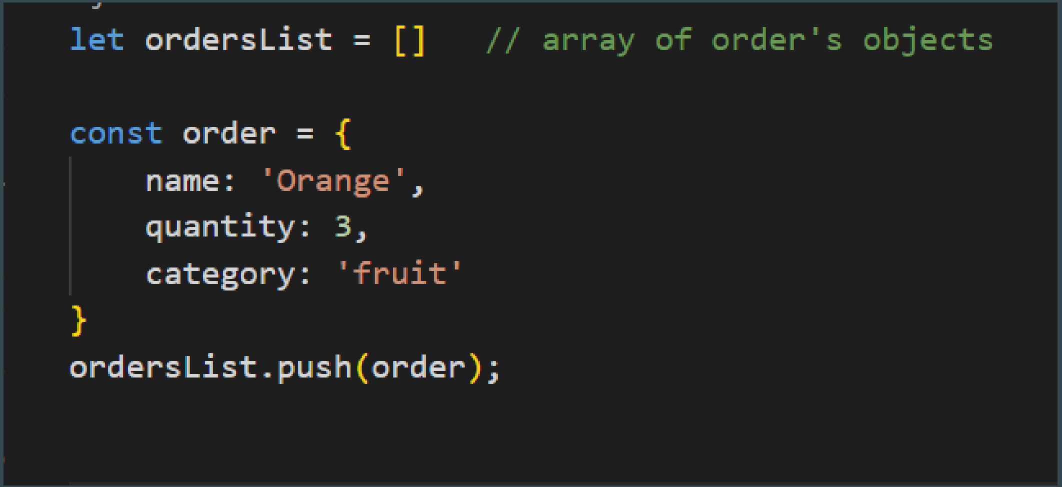
Objects

4. Array os objects

Objects

4 Arrays of objects

- Adding an object to an array

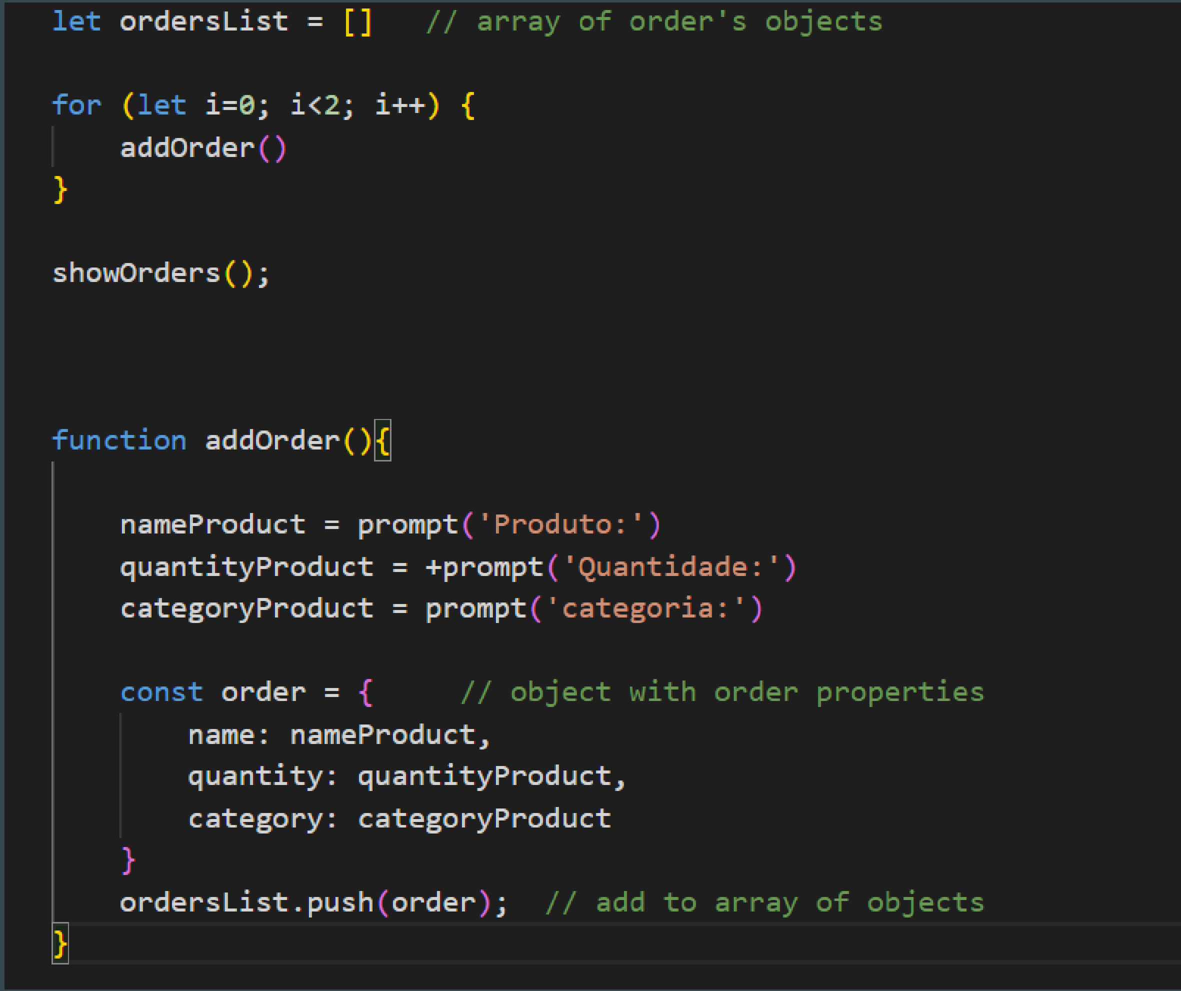


```
let ordersList = []   // array of order's objects

const order = {
    name: 'Orange',
    quantity: 3,
    category: 'fruit'
}
ordersList.push(order);
```

Objects

4 Arrays of objects

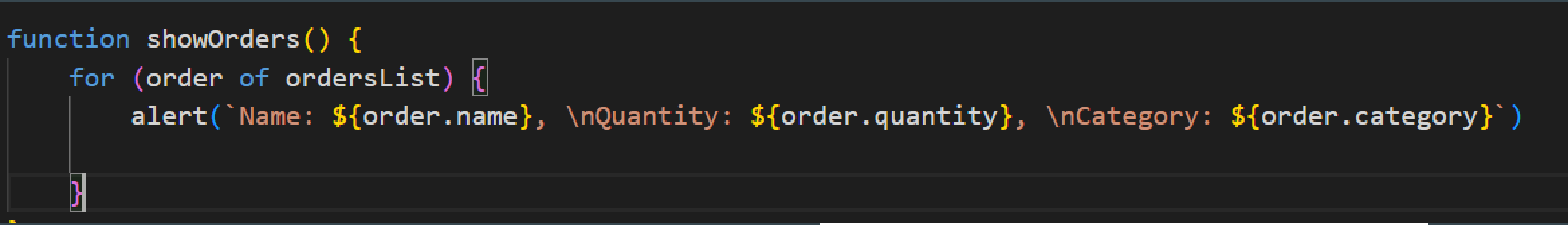
- Adding objects to an array through inputs



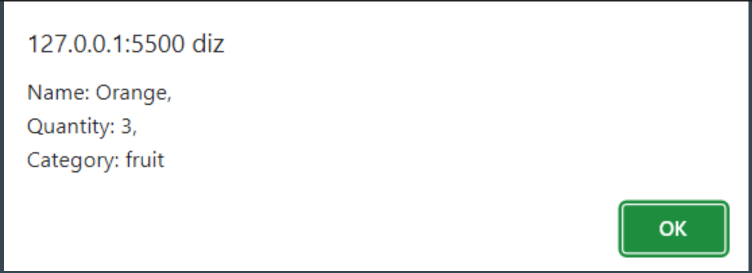
```
let ordersList = []   // array of order's objects

for (let i=0; i<2; i++) {
    addOrder()
}

showOrders();



function addOrder(){

    nameProduct = prompt('Produto:')
    quantityProduct = +prompt('Quantidade:')
    categoryProduct = prompt('categoria:')

    const order = {      // object with order properties
        name: nameProduct,
        quantity: quantityProduct,
        category: categoryProduct
    }
    ordersList.push(order);  // add to array of objects
}
```

Objects

4 Iterate array



```
function showOrders() {
    for (order of ordersList) {
        alert(`Name: ${order.name}, \nQuantity: ${order.quantity}, \nCategory: ${order.category}`)

    }
```

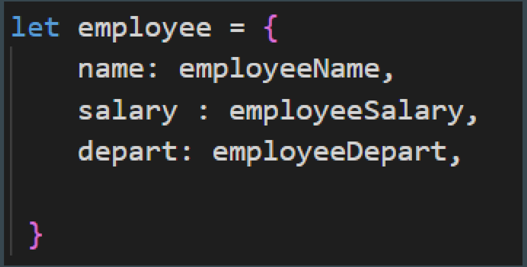


```
127.0.0.1:5500 diz
Name: Orange,
Quantity: 3,
Category: fruit
```

Objects

Exercices

- Create a function **addEmployee()**:
  - that allows you to read data from a company's employees: **name**, **salary** and **department** (from prompt function), adding to an employee object, such as:



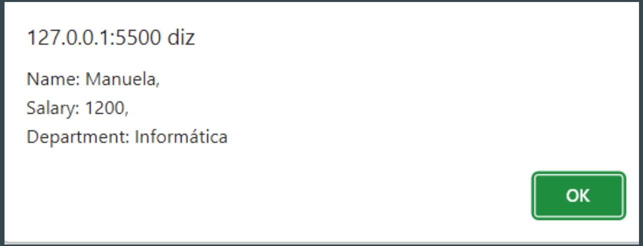
```
let employee = {
    name: employeeName,
    salary : employeeSalary,
    depart: employeeDepart,

 }
```

  - push de new object to an array of employees (**employeeList**)
- create a cycle to add 3 employees

Objects

Exercices

- Create a function **showEmployees** () to iterate the list objects, and printing them, one by one, in an alert box



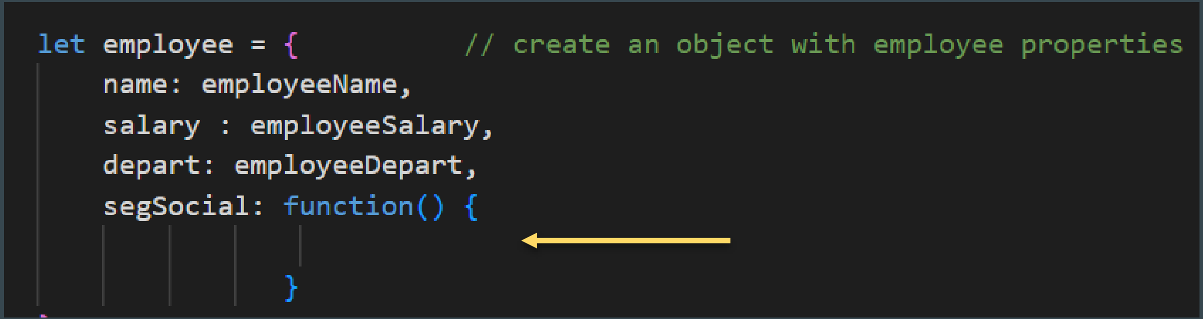
127.0.0.1:5500 diz

```
Name: Manuela,
Salary: 1200,
Department: Informática
```

- Create a function **totalSalaries** that returns the total salaries of employees

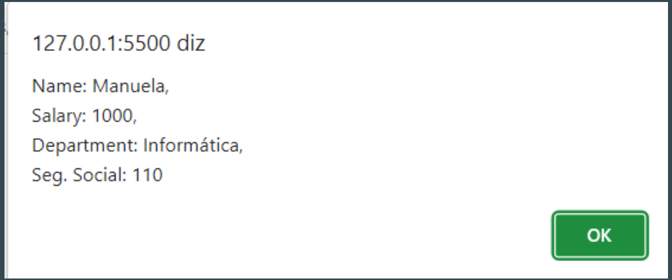
Objects

Exercices



```
let employee = {           // create an object with employee properties
    name: employeeName,
    salary : employeeSalary,
    depart: employeeDepart,
    segSocial: function() {

            }
```

- Edit the employee object, adding a new property: **segSocial** .
  - This property must be a method **(function stored as object property)** that calculates the value of social security: **salary * 0.11**
- Change the function **showEmployees()** in order to also show the segSocial property, when prints employee's data



```
127.0.0.1:5500 diz

Name: Manuela,
Salary: 1000,
Department: Informática,
Seg. Social: 110
```

Objects

4 Arrays of objects

- Using array methods with objects: we can use the previously covered methods on arrays
  - Add / remove methods: push, pop, shift, unshift, ...
  - Search elements: some, includes, find, filter, ...
  - Transform arrays: map, reduce, sort, reverse, ...

In the next class ...

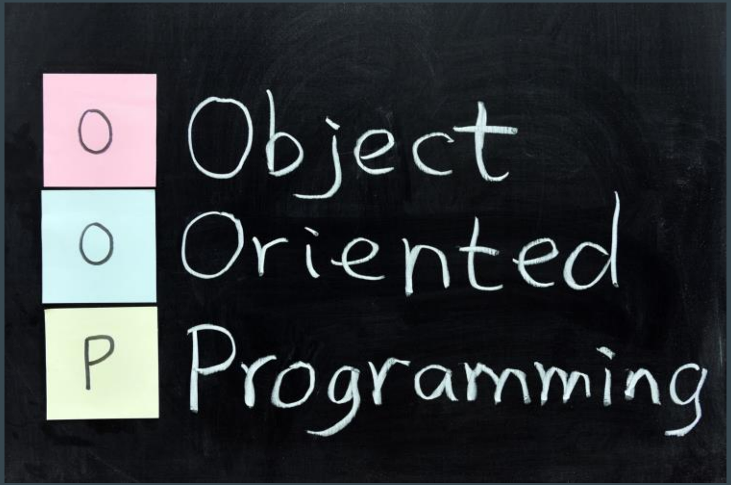
Object Oriented Programming

•••

M03 - Arrays

M04 - Arrays

1. Arrays
    a. Iteration



M04 - Arrays

1. Arrays > Iteration

M04 - Objects

1. Arrays > Iteration

- We often need an ordered collection, where we have a 1st, 2nd, 3rd element and so on. For example, we need this to store a list of something: users, articles, HTML elements, etc.

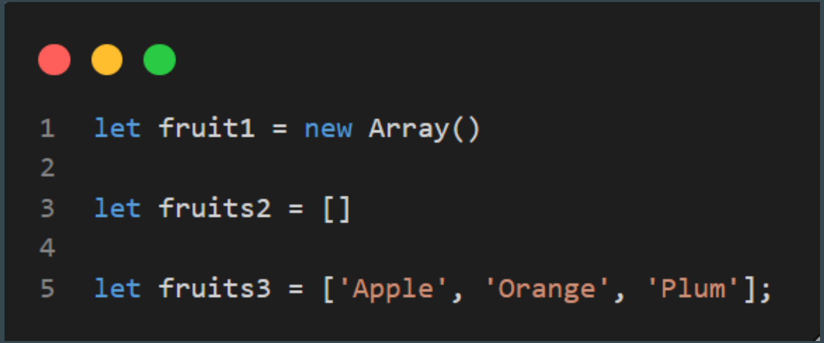
- It is not convenient to use objects as they do not provide methods to manage the order of the elements

- There is a special data structure called Array, for storing ordered collections

M04 - Objects

1. Arrays > Iteration (Arrays)

- Declaration

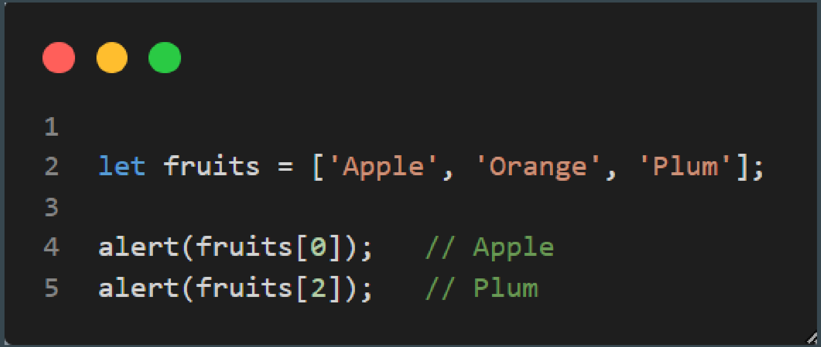


```
let fruit1 = new Array()

let fruits2 = []

let fruits3 = ['Apple', 'Orange', 'Plum'];
```

M04 - Objects

1. Arrays > Iteration (Arrays)

- Access
  - The elements of the array are numbered (indexes), starting with the index zero.
  - We can obtain an element by placing its index between braces:

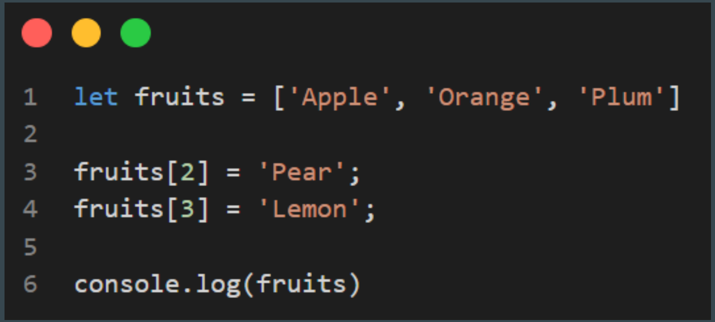


```

let fruits = ['Apple', 'Orange', 'Plum'];

alert(fruits[0]);    // Apple
alert(fruits[2]);    // Plum
```

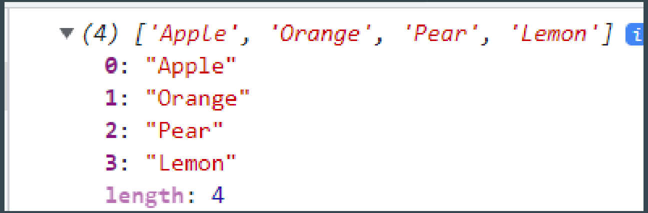
M04 - Objects

1. Arrays > Iteration (Arrays)

- Transformations
  - We can **change an element** by giving its index and assigning a new value
  - We can **add a new element** by giving a new index and assigning a new value



```
let fruits = ['Apple', 'Orange', 'Plum']

fruits[2] = 'Pear';
fruits[3] = 'Lemon';

console.log(fruits)
```

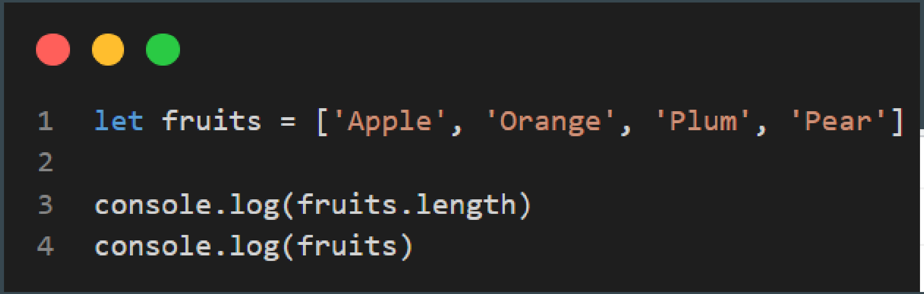


```
▼ (4) ['Apple', 'Orange', 'Pear', 'Lemon']
    0: "Apple"
    1: "Orange"
    2: "Pear"
    3: "Lemon"
    length: 4
```

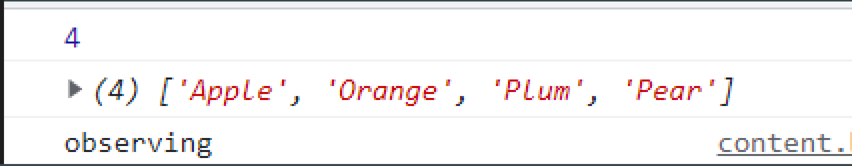
M04 - Objects

1. Arrays > Iteration (Arrays)

- Counting
    - The total number of elements in an array is given by the length property
    - We can show the entire array



```javascript
let fruits = ['Apple', 'Orange', 'Plum', 'Pear']

console.log(fruits.length)
console.log(fruits)
```

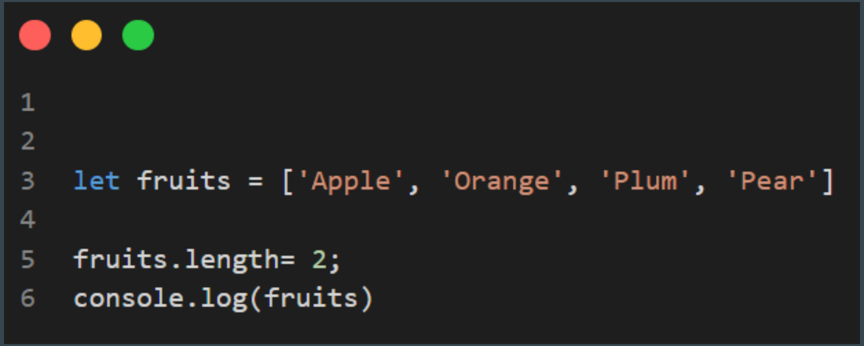


```
4
▶ (4) ['Apple', 'Orange', 'Plum', 'Pear']
observing                                          content.
```

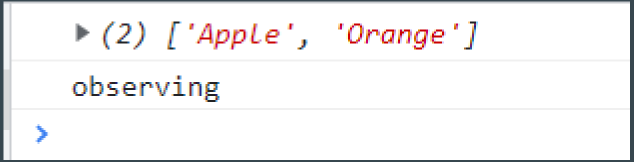
M04 - Objects

1. Arrays > Iteration (Arrays)

- Counting
  - The length property is not read-only, in fact it is also used to truncate or clean the array



```
let fruits = ['Apple', 'Orange', 'Plum', 'Pear']

fruits.length= 2;
console.log(fruits)
```



```
▶ (2) ['Apple', 'Orange']
observing
>
```

M04 - Objects

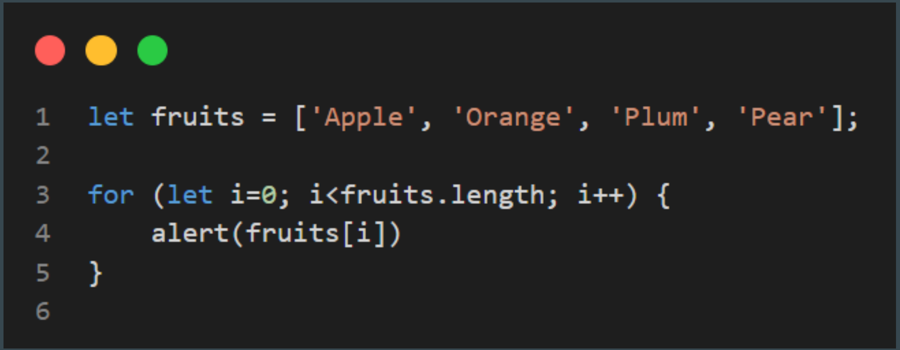
1. Arrays > Iteration (Arrays)

- Reference
    - Remember, there are only 7 basic types in JavaScript
    - An array is a special type of object and therefore behaves like an object
    - Arrays extend objects by providing special methods for working with ordered collections of data and also with the length property

M04 - Objects

1. Arrays > Iteration (Arrays)

- Iteration
  - One of the oldest ways to iterate over elements of an array is with the for cycle that traverses the array using its indexes:

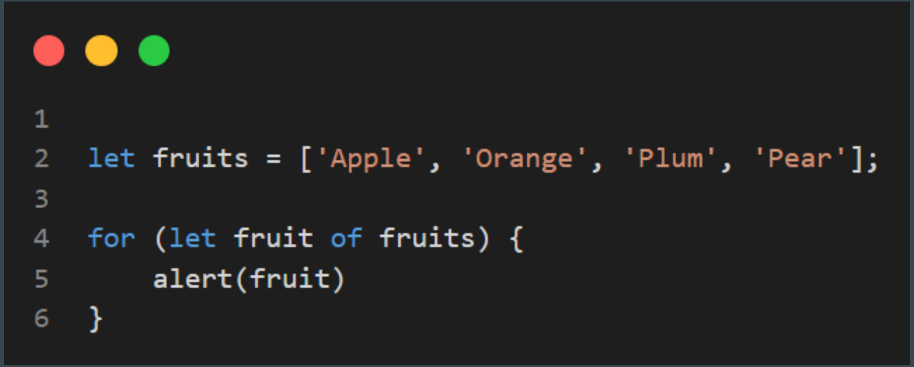


```
let fruits = ['Apple', 'Orange', 'Plum', 'Pear'];

for (let i=0; i<fruits.length; i++) {
    alert(fruits[i])
}

```

M04 - Objects

1. Arrays > Iteration (Arrays)

- Iteration (for...of)
  - Another popular way is to use the for...of cycle



```

let fruits = ['Apple', 'Orange', 'Plum', 'Pear'];

for (let fruit of fruits) {
    alert(fruit)
}
```

  - For..of does not give access to the position (index) of the current element, only its value, but in most cases it is enough. And it's shorter.

M04 - Objects

1. Arrays > Iteration (Arrays)

- Methods on arrays:

Add / Remove elements

Search for elements

Iterate over elements

Transform the array

M04 - Objects

1. Arrays > Iteration (Arrays)

- Methods on arrays:

Add / Remove elements

Search for elements

Iterate over elements

Transform the array

M04 - Objects

1. Arrays > Iteration (Arrays)

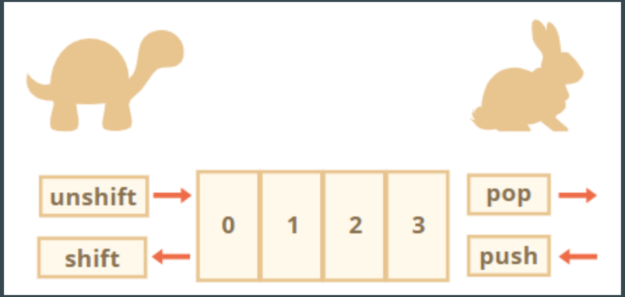
- Add / Remove elements:
  - push(...elems) - add elements to the end
  - pop() - extracts an element from the end
  - shift() - extracts an element from the beginning
  - unshift(...elems) - add elements to the beginning

  - splice(pos, delCount, ...elems) - in the pos index, deletes delCount elements and inserts elems
  - slice(init, end) - creates new array and copies elements from position init to end
  - concat(...elems) - returns new array: copies current members and adds elems to it

M04 - Objects

1. Arrays > Iteration (Arrays)

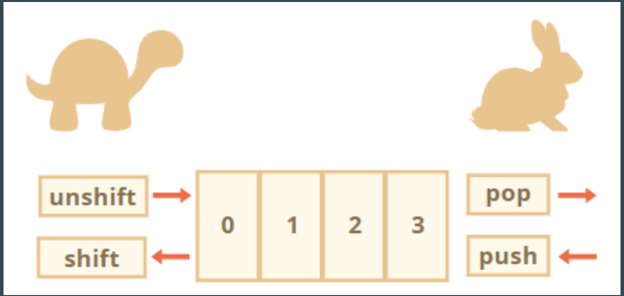
- Add / Remove elements:
    - push(...elems) - add elements to the end
    - pop() - extracts an element from the end
    - shift() - extracts an element from the beginning
    - unshift(...elems) - add elements to the beginning

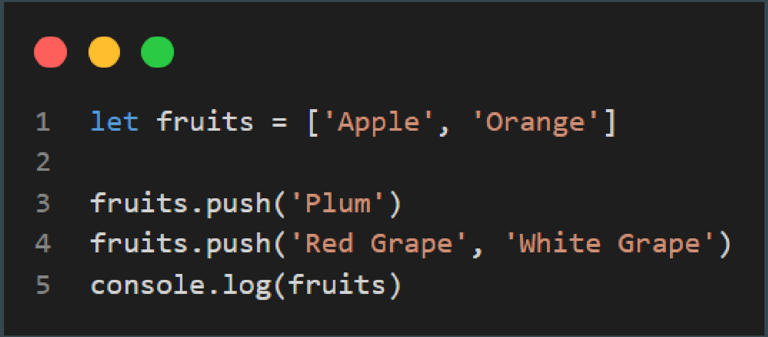


M04 - Objects

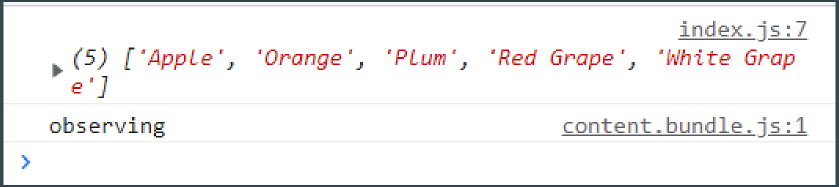
1. Arrays > Iteration (Arrays)

- Add / Remove elements:
  - push(...elems)
    - add elements to the end
    - returns the current length (length) of the array





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

fruits.push('Plum')
fruits.push('Red Grape', 'White Grape')
console.log(fruits)
```

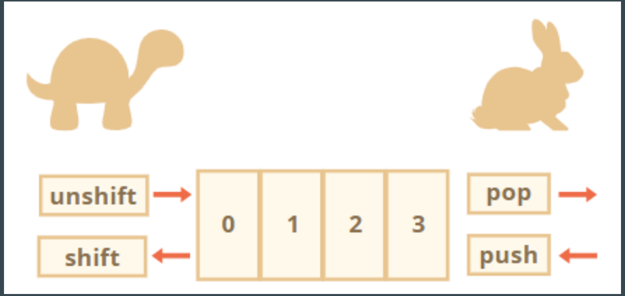


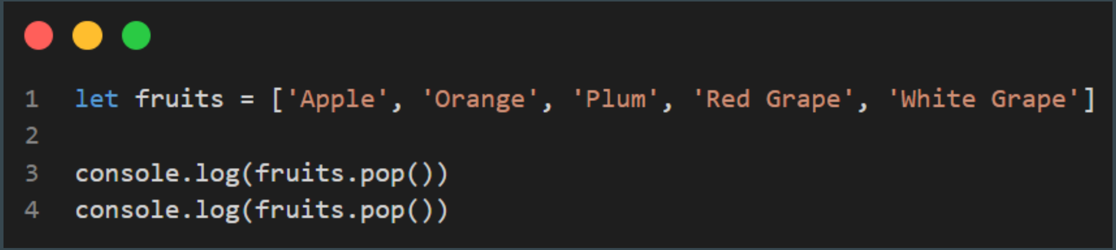
```
                                                        index.js:7
▸ (5) ['Apple', 'Orange', 'Plum', 'Red Grape', 'White Grap
  e']
observing                                       content.bundle.js:1
>
```

M04 - Objects

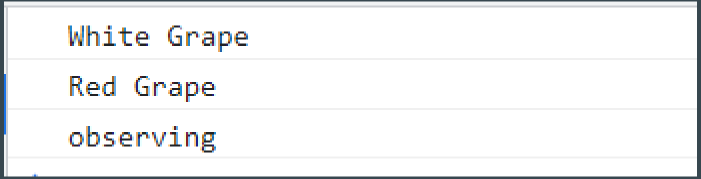
1. Arrays > Iteration (Arrays)

- Add / Remove elements:
  - pop()
    - extracts an element from the end of the array
    - returns the removed element





```
let fruits = ['Apple', 'Orange', 'Plum', 'Red Grape', 'White Grape']

console.log(fruits.pop())
console.log(fruits.pop())
```

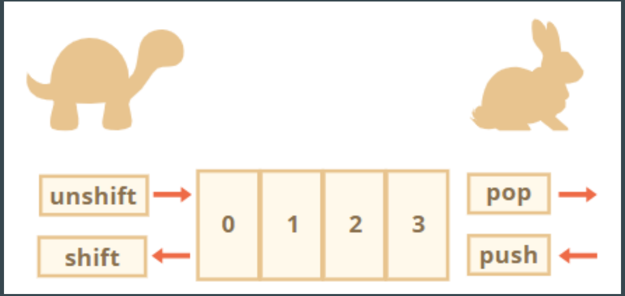


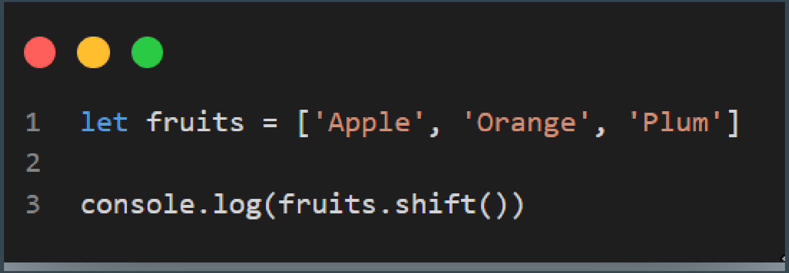
```
White Grape
Red Grape
observing
```

M04 - Objects

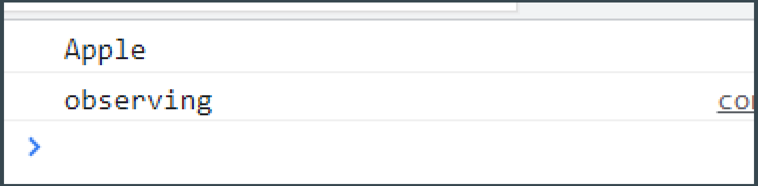
1. Arrays > Iteration (Arrays)

- Add / Remove elements:
  - shift()
    - extracts an element from the beginning of the array
    - returns the removed element





```
let fruits = ['Apple', 'Orange', 'Plum']

console.log(fruits.shift())
```

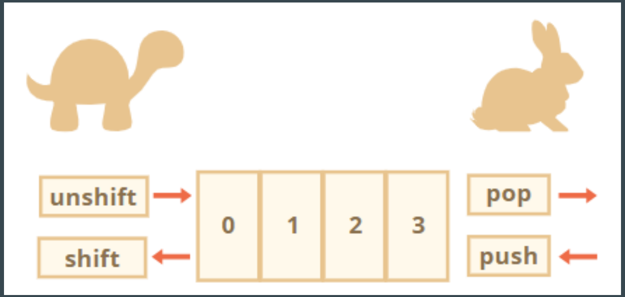


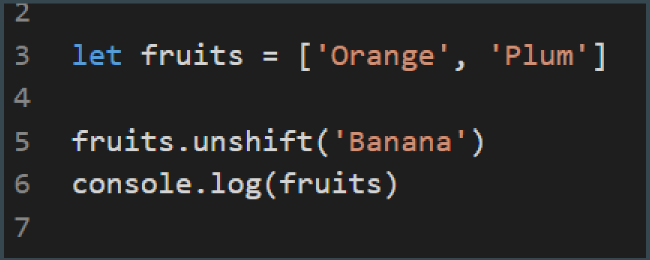
```
Apple
observing                                                     co
>
```

M04 - Objects

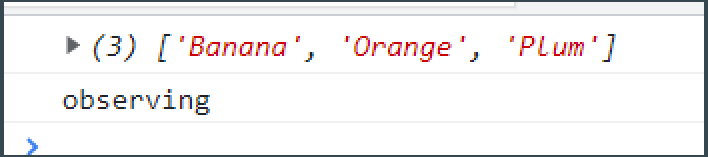
1. Arrays > Iteration (Arrays)

- Add / Remove elements:
    - unshift(...elems)
        - add elements to the beginning





```
let fruits = ['Orange', 'Plum']

fruits.unshift('Banana')
console.log(fruits)
```

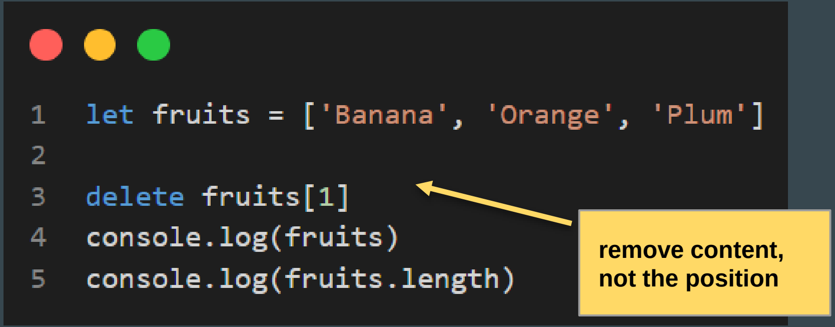


```
▶ (3) ['Banana', 'Orange', 'Plum']
observing
```

M04 - Objects

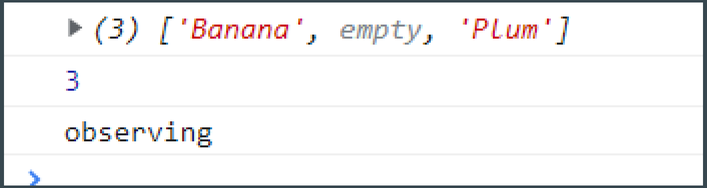
1. Arrays > Iteration (Arrays)

- Add / Remove elements:
  - How to delete an element from an array?
  - Arrays are objects, so you can use delete:



```javascript
let fruits = ['Banana', 'Orange', 'Plum']

delete fruits[1]
console.log(fruits)
console.log(fruits.length)
```

remove content, not the position

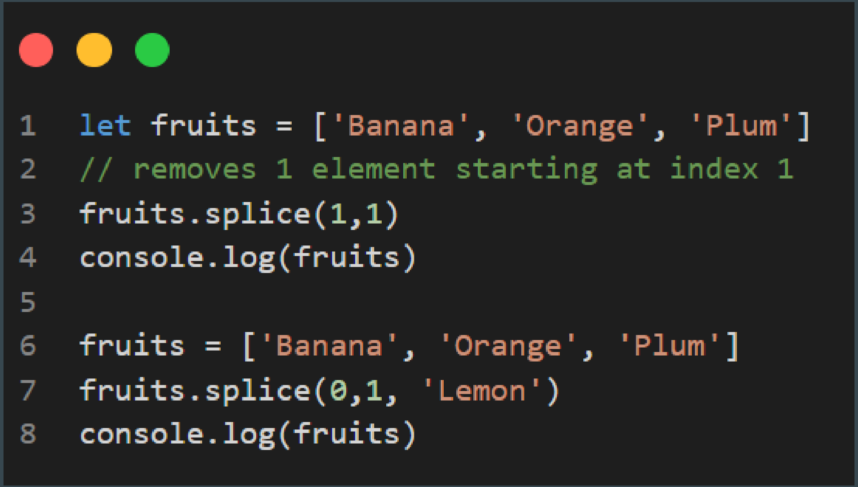


```
▶ (3) ['Banana', empty, 'Plum']
3
observing
>
```

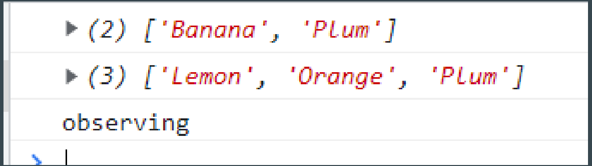
M04 - Objects

1. Arrays > Iteration (Arrays)

- Add / Remove elements:
  - Solution to avoid the previous problem of generating "holes", use the splice method: splice(pos, delCount, ...elems) - in the pos index, deletes delCount elements and inserts elems



```
let fruits = ['Banana', 'Orange', 'Plum']
// removes 1 element starting at index 1
fruits.splice(1,1)
console.log(fruits)

fruits = ['Banana', 'Orange', 'Plum']
fruits.splice(0,1, 'Lemon')
console.log(fruits)
```

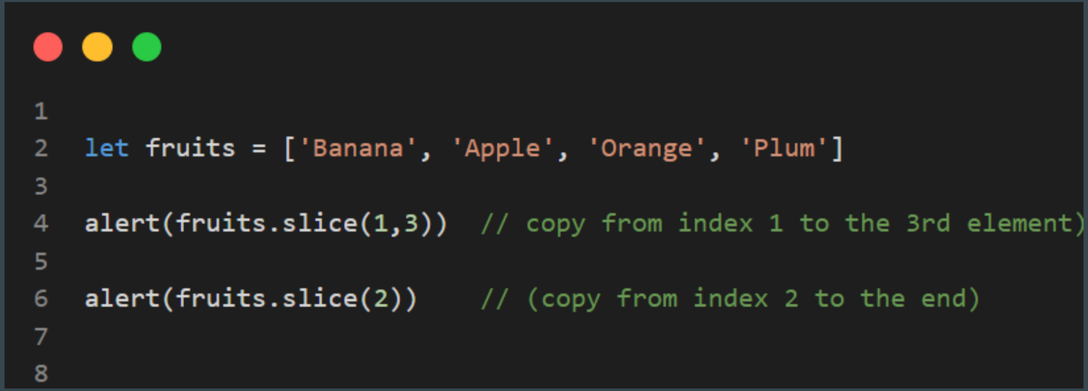


```
▸ (2) ['Banana', 'Plum']
▸ (3) ['Lemon', 'Orange', 'Plum']
observing
```

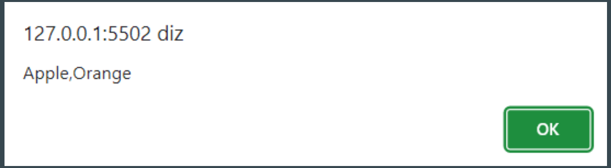
M04 - Objects

1. Arrays > Iteration (Arrays)

- Add / Remove elements:
  - slice(init, end) - create a new array and copy elements from position init to end (not inclusive)

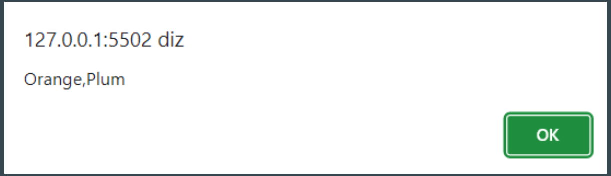


```

let fruits = ['Banana', 'Apple', 'Orange', 'Plum']

alert(fruits.slice(1,3))  // copy from index 1 to the 3rd element)

alert(fruits.slice(2))    // (copy from index 2 to the end)


```



127.0.0.1:5502 diz

Apple,Orange



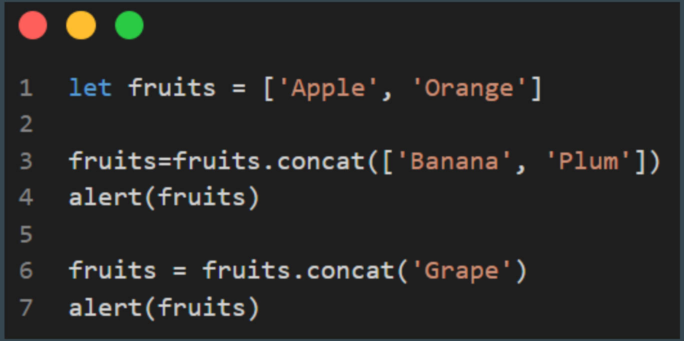
127.0.0.1:5502 diz

Orange,Plum

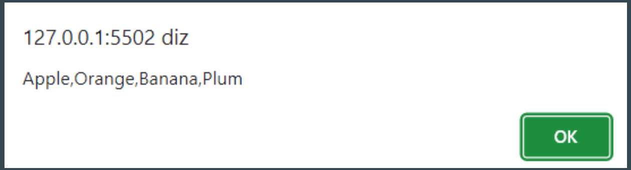
M04 - Objects

1. Arrays > Iteration (Arrays)

- Add / Remove elements:
  - concat(...elems)
    - copies all members of the current one and adds elems to it. If any of the elements is an array, its elements will be used
    - returns a new array

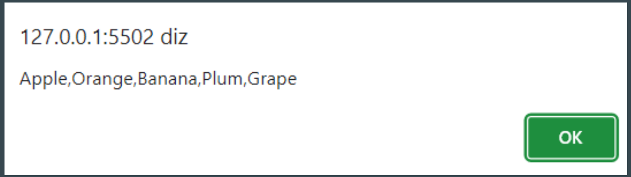


```javascript
let fruits = ['Apple', 'Orange']

fruits=fruits.concat(['Banana', 'Plum'])
alert(fruits)

fruits = fruits.concat('Grape')
alert(fruits)
```



127.0.0.1:5502 diz

Apple,Orange,Banana,Plum



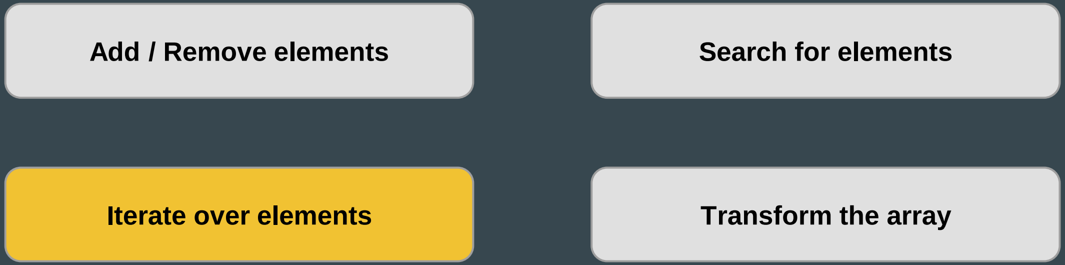
127.0.0.1:5502 diz

Apple,Orange,Banana,Plum,Grape

M04 - Objects

1. Objects > Iteration (Arrays)

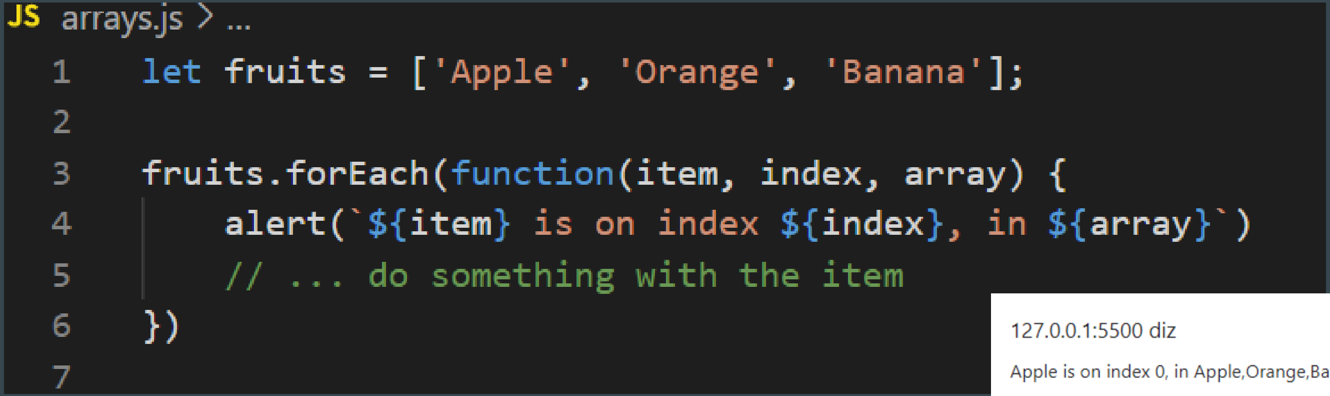
- Methods on arrays:



M04 - Objects

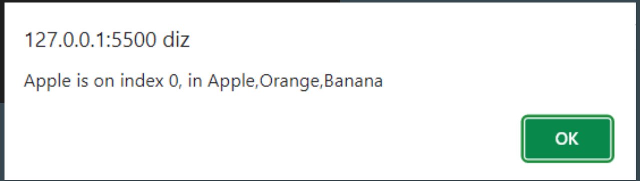
1. Arrays > Iteration (Arrays)

- Iterate over elements:
  - The forEach method allows you to execute a function for each element of the array
  - Syntax:



arrays.js > ...

```javascript
let fruits = ['Apple', 'Orange', 'Banana'];

fruits.forEach(function(item, index, array) {
    alert(`${item} is on index ${index}, in ${array}`)
    // ... do something with the item
})

```



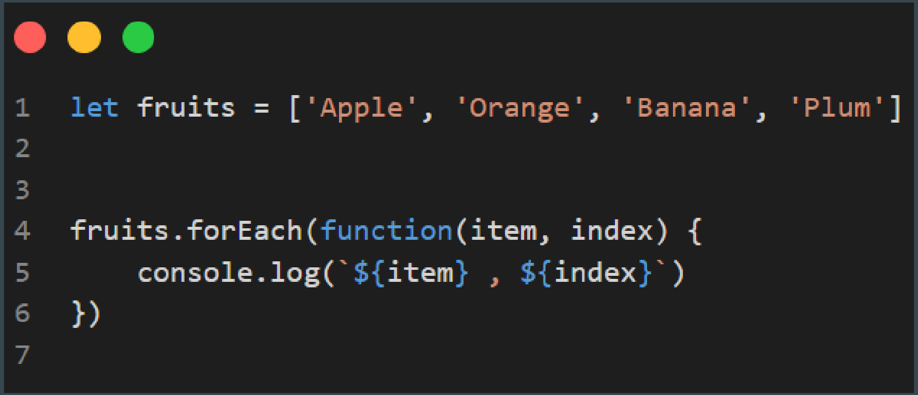
127.0.0.1:5500 diz

Apple is on index 0, in Apple,Orange,Banana

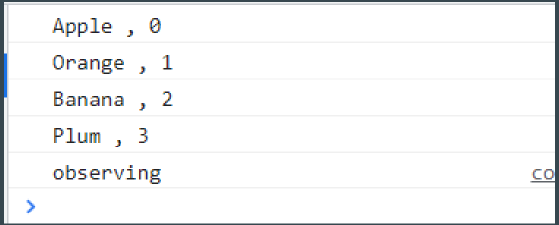
M04 - Objects

1. Arrays > Iteration (Arrays)

- Iterate over elements:
  - The forEach method allows you to execute a function for each element of the array
  - Syntax:



```javascript
let fruits = ['Apple', 'Orange', 'Banana', 'Plum']


fruits.forEach(function(item, index) {
    console.log(`${item} , ${index}`)
})

```



```
Apple , 0
Orange , 1
Banana , 2
Plum , 3
observing                                  co
>
```

M04 - Objects

1. Arrays > Iteration (Arrays)

- Methods on arrays:

Add / Remove elements
Search for elements
Iterate over elements
Transform the array

M04 - Objects

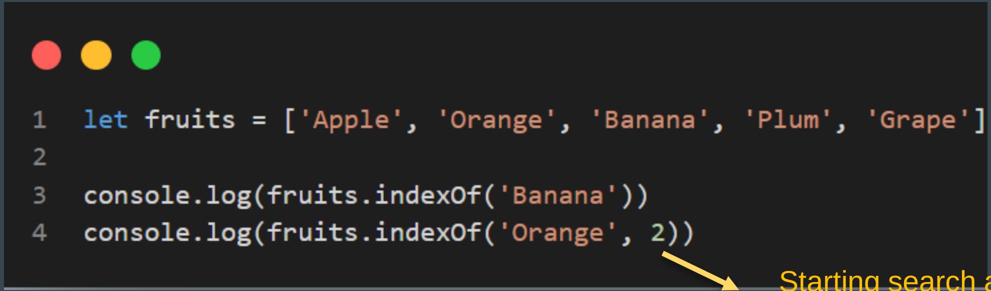
1. Arrays > Iteration (Arrays)

- Search for elements in an array:
    - indexOf/lastIndexOf(elem, pos) - searches for elem starting at the pos position, and returns the index or -1 if not found
    - includes(value) - returns true if the array has a value, otherwise false
    - some(fn) - tests if at least one element of the array passes the test implemented
    - by the function provided
    - every(fn) - tests whether all elements of the array pass the test implemented
    - by the function provided
    - find/filter(func) - filters the elements through the function, returns the first/all values that make it return true
    - findIndex(func) - it's like find, but returns the index instead of a value

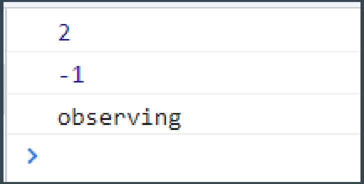
M04 - Objects

1. Arrays > Iteration (Arrays)

- Search for elements in an array:
  - indexOf(elem, pos) - searches for the elem starting in the pos position,
  - and returns the index or -1 if not found
  - lastIndexOf(elem, pos) - the same, but looking from the right to left.



```
let fruits = ['Apple', 'Orange', 'Banana', 'Plum', 'Grape']

console.log(fruits.indexOf('Banana'))
console.log(fruits.indexOf('Orange', 2))
```



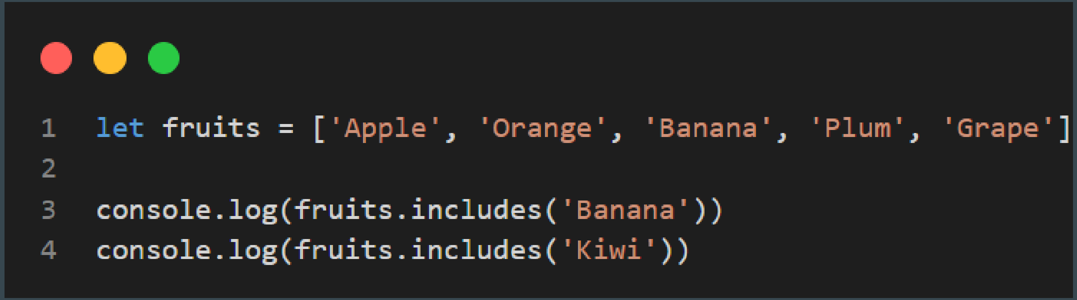
```
2
-1
observing
>
```

Starting search at position 2

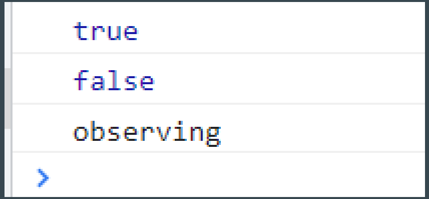
M04 - Objects

1. Arrays > Iteration (Arrays)

- Search for elements in an array:
  - includes(value) - returns true if the array has value, otherwise, false.



```javascript
let fruits = ['Apple', 'Orange', 'Banana', 'Plum', 'Grape']

console.log(fruits.includes('Banana'))
console.log(fruits.includes('Kiwi'))
```

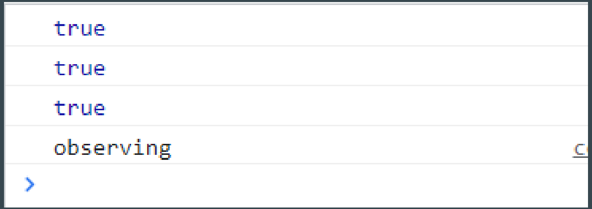


```
true
false
observing
>
```

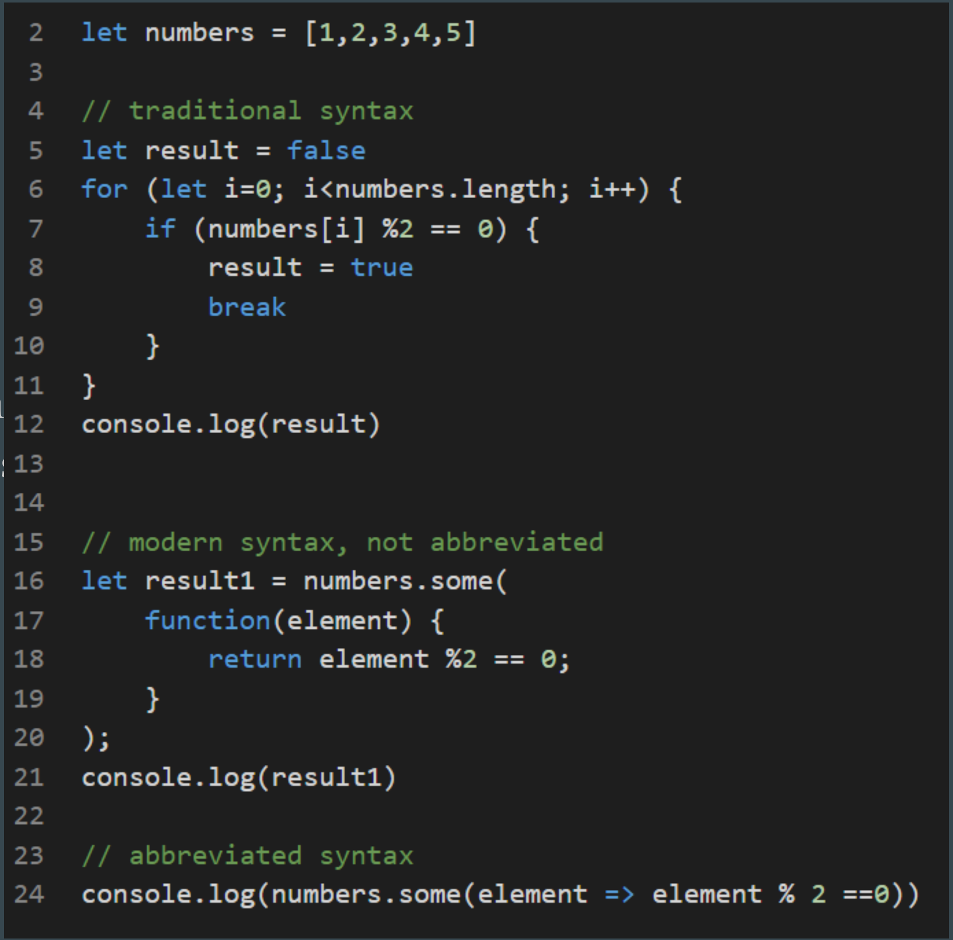
M04 - Objects

1. Arrays > Iteration (Arrays)

- Search for elements in an array:
  - some(func)
    - tests if at least one element of th
      array passes the implemented te
      by the function provided
    - returns a boolean value



```
true
true
true
observing
>
```

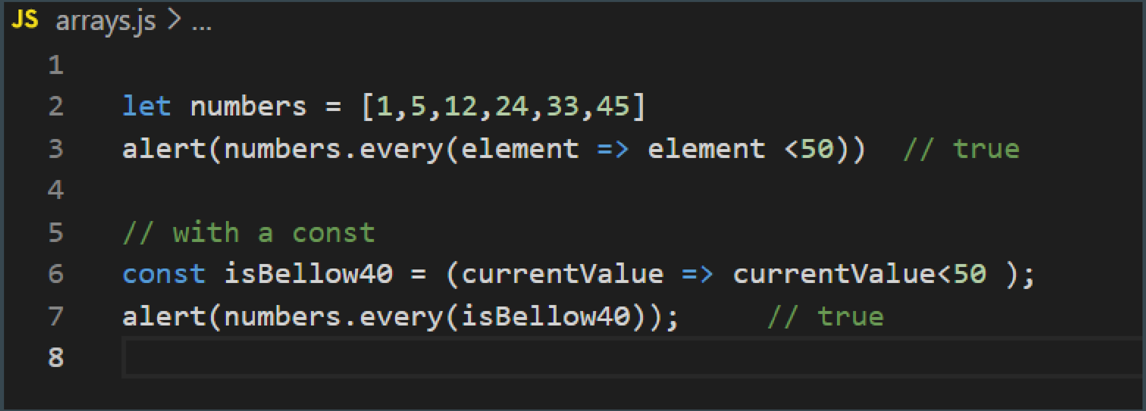


```
let numbers = [1,2,3,4,5]

// traditional syntax
let result = false
for (let i=0; i<numbers.length; i++) {
    if (numbers[i] %2 == 0) {
        result = true
        break
    }
}
console.log(result)


// modern syntax, not abbreviated
let result1 = numbers.some(
    function(element) {
        return element %2 == 0;
    }
);
console.log(result1)

// abbreviated syntax
console.log(numbers.some(element => element % 2 ==0))
```

M04 - Objects

1. Arrays > Iteration (Arrays)

- Search for elements in an array:
  - every(func)
    - tests whether all elements of the array pass the function provided
    - returns a boolean value

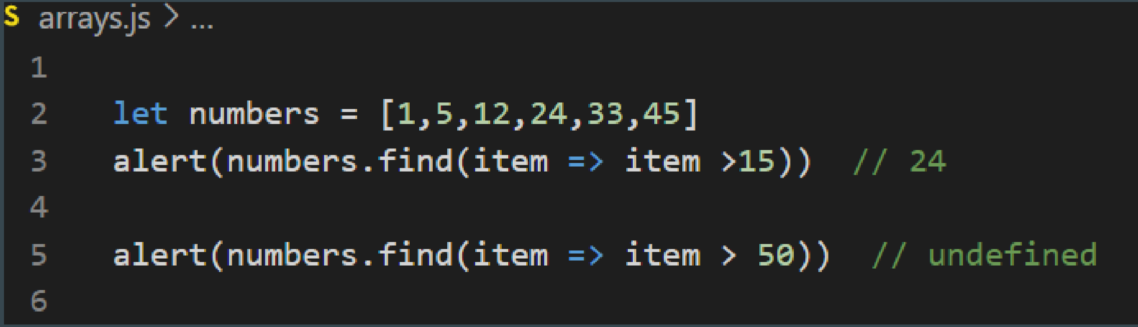


JS arrays.js > ...

```javascript

let numbers = [1,5,12,24,33,45]
alert(numbers.every(element => element <50))  // true

// with a const
const isBellow40 = (currentValue => currentValue<50 );
alert(numbers.every(isBellow40));     // true

```

M04 - Objects

1. Arrays > Iteration (Arrays)


arrays.js > ...

```
let numbers = [1,5,12,24,33,45]
alert(numbers.find(item => item >15))  // 24

alert(numbers.find(item => item > 50))  // undefined
```

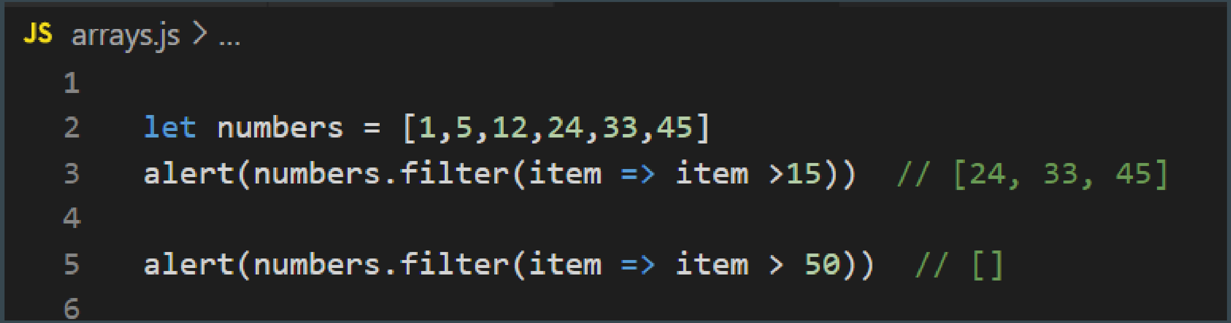
- Search for elements in an array:
    - find(func)
        - filters the elements through a function, returns the first element that make it return true
        - If not found, returns undefined.

    - The findIndex(fn) method is similar by returning the index or -1 if there are no occurrences

M04 - Objects

1. Arrays > Iteration (Arrays)

- Search for elements in an array:
  - filter(func)
    - filters the elements with a function
    - returns an array with all elements that make the function return true
    - If not found, returns [ ]



JS arrays.js > ...

```javascript

let numbers = [1,5,12,24,33,45]
alert(numbers.filter(item => item >15))  // [24, 33, 45]

alert(numbers.filter(item => item > 50))  // []

```

M04 - Objects

1. Arrays > Iteration (Arrays)

- Methods on arrays:

Add / Remove elements

Search for elements

Iterate over elements

Transform the array

M04 - Objects

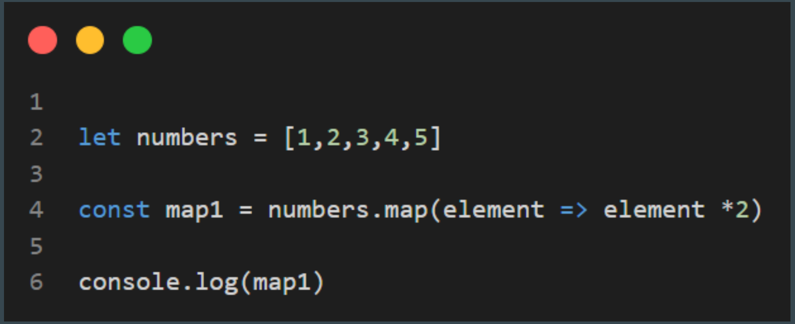
1. Arrays > Iteration (Arrays)

- Transform an array:
  - map(func) - creates a new array from the results of the func call for each element of the array
  - sort(func) - orders an array (*in place*). Uses func to control ordering.
  - reverse() - inverts the array *in place*
  - split(sep)/join(sep) - converts a string to an array and vice versa based on sep.
  - reduce(func, init) - calculates a single value on the matrix calling func for each element and passing an intermediate result between calls.
  - fill(value, start, end) - fills the array with repeated values from the beginning to the end of the index.

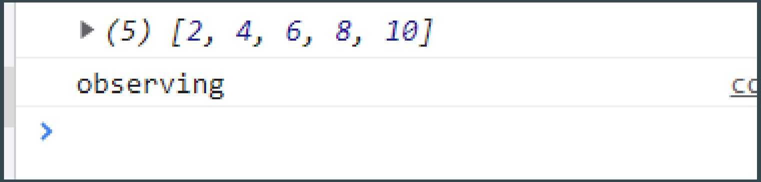
M04 - Objects

1. Arrays > Iteration (Arrays)

- Transform an array:
  - map(func)
    - Creates a new array filled with the results of the calling of a function provided on all elements of the given array



```
let numbers = [1,2,3,4,5]

const map1 = numbers.map(element => element *2)

console.log(map1)
```

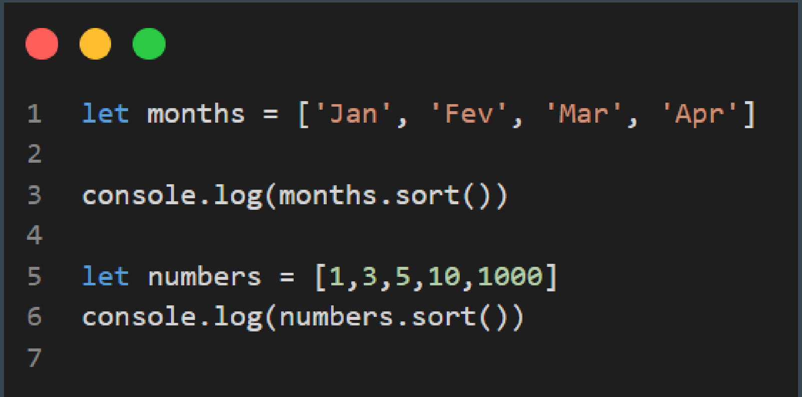


```
▶ (5) [2, 4, 6, 8, 10]
observing                                                    co
>
```

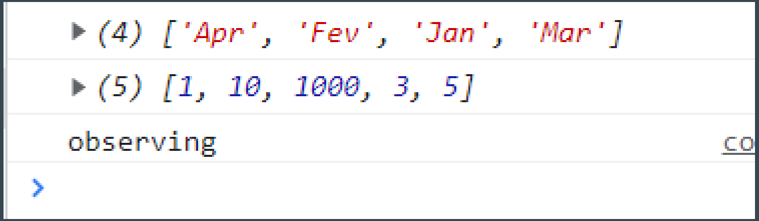
M04 - Objects

1. Arrays > Iteration (Arrays)

- Transform an array:
  - sort(func)
    - sorts an array *in place*, changing its order of elements.



```
let months = ['Jan', 'Fev', 'Mar', 'Apr']

console.log(months.sort())

let numbers = [1,3,5,10,1000]
console.log(numbers.sort())

```



```
▶ (4) ['Apr', 'Fev', 'Jan', 'Mar']
▶ (5) [1, 10, 1000, 3, 5]
observing                                                  co
>
```

M04 - Objects

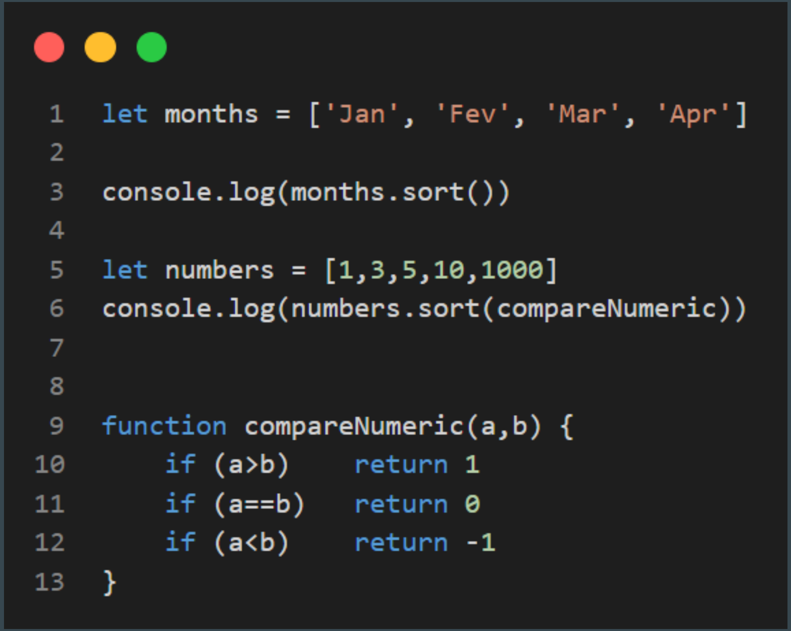
1. Arrays > Iteration (Arrays)

- Transform an array:
  - sort(func)
    - as you must have noticed the numerical ordering is incorrect! Why?
    - elements, by default, are ordered as strings
    - for strings, the lexicographic order is applied and, in fact, "30"> "100000".
    - to control sorting, you must provide a function as an sort argument

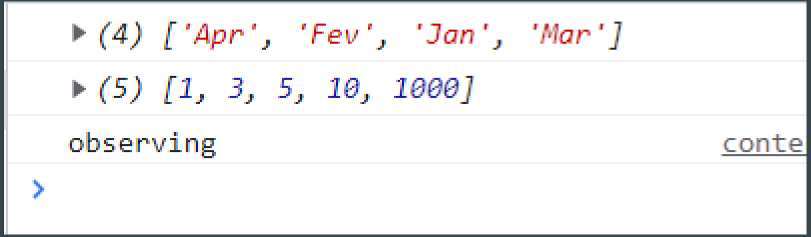
M04 - Objects

1. Arrays > Iteration (Arrays)

- Transform an array:
  - sort(func)
    - Example:



```javascript
let months = ['Jan', 'Fev', 'Mar', 'Apr']

console.log(months.sort())

let numbers = [1,3,5,10,1000]
console.log(numbers.sort(compareNumeric))


function compareNumeric(a,b) {
    if (a>b)    return 1
    if (a==b)   return 0
    if (a<b)    return -1
}
```

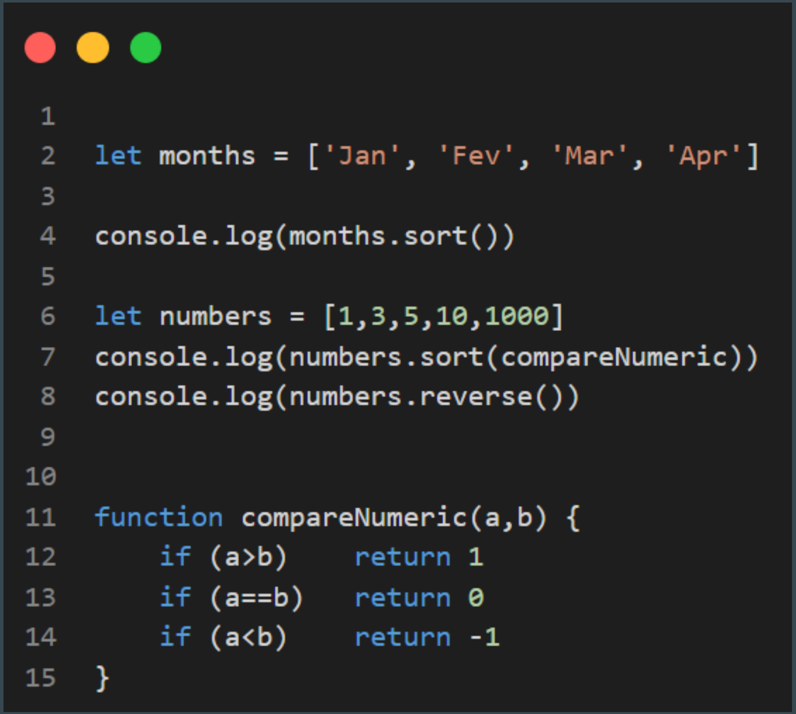


```
▶ (4) ['Apr', 'Fev', 'Jan', 'Mar']
▶ (5) [1, 3, 5, 10, 1000]
observing                                          conte
>
```

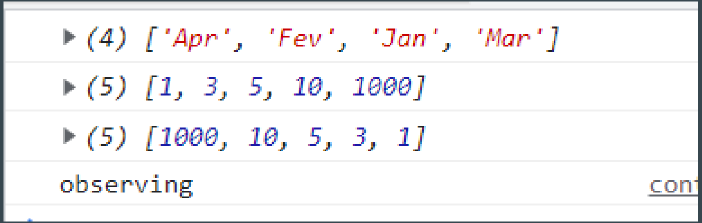
M04 - Objects

1. Arrays > Iteration (Arrays)

- Transform an array:
  - reverse(func)
    - Inverts an array *in place*, changing the order of the elements.



```

let months = ['Jan', 'Fev', 'Mar', 'Apr']

console.log(months.sort())

let numbers = [1,3,5,10,1000]
console.log(numbers.sort(compareNumeric))
console.log(numbers.reverse())


function compareNumeric(a,b) {
    if (a>b)    return 1
    if (a==b)   return 0
    if (a<b)    return -1
}
```

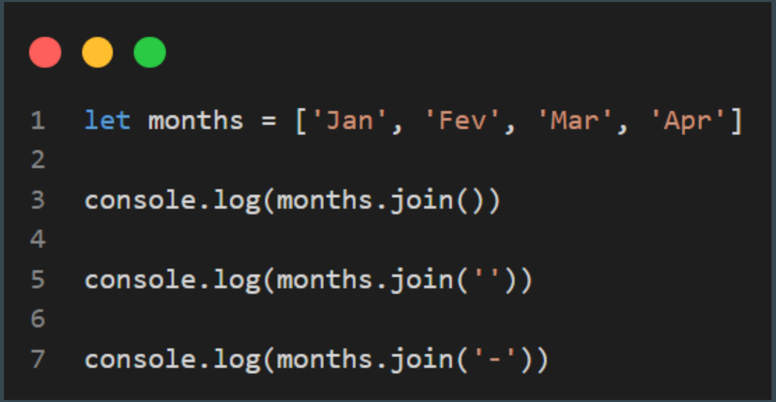


```
▶ (4) ['Apr', 'Fev', 'Jan', 'Mar']
▶ (5) [1, 3, 5, 10, 1000]
▶ (5) [1000, 10, 5, 3, 1]
observing                                          con
```

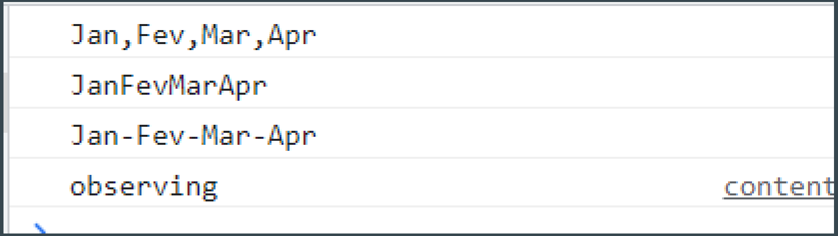
M04 - Objects

1. Arrays > Iteration (Arrays)

- Transform an array:
  - join(sep)
    - Creates and returns a new string concatenating all elements of an array, separated by commas or a specified separator sequence (sep)



```javascript
let months = ['Jan', 'Fev', 'Mar', 'Apr']

console.log(months.join())

console.log(months.join(''))

console.log(months.join('-'))
```

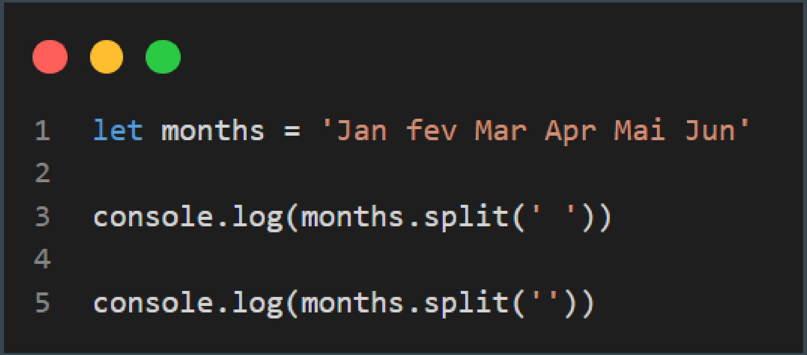


```
Jan,Fev,Mar,Apr
JanFevMarApr
Jan-Fev-Mar-Apr
observing                                    content
```

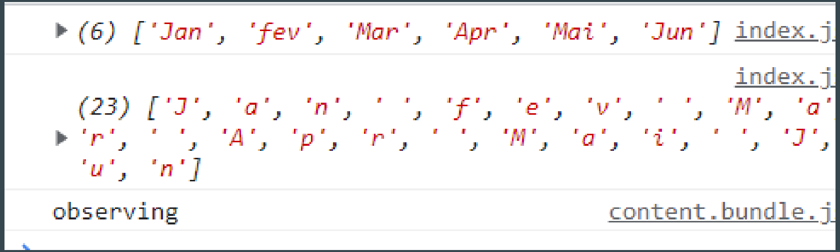
M04 - Objects

1. Arrays > Iteration (Arrays)

- Transform an array:
  - split(sep)
    - The inverse of join is the split method applied to a string



```
let months = 'Jan fev Mar Apr Mai Jun'

console.log(months.split(' '))

console.log(months.split(''))
```



```
▶ (6) ['Jan', 'fev', 'Mar', 'Apr', 'Mai', 'Jun'] index.j
                                                  index.j
  (23) ['J', 'a', 'n', ' ', 'f', 'e', 'v', ' ', 'M', 'a'
▶ 'r', ' ', 'A', 'p', 'r', ' ', 'M', 'a', 'i', ' ', 'J',
  'u', 'n']
observing                                content.bundle.j
```

M04 - Objects

1. Arrays > Iteration (Arrays)

- Transform an array:
    - reduce(func)
        - when we need to iterate over an array - we use forEach, for or for...of
        - when we need to iterate and return the data for each element - we use map
        - the reduce method allows you to calculate a single value based on an array
        - syntax:

```
let value = arr.reduce(function (accumulator, item, index, array) {
    // ...
}, [initial]);
```

M04 - Objects

1. Arrays > Iteration (Arrays)

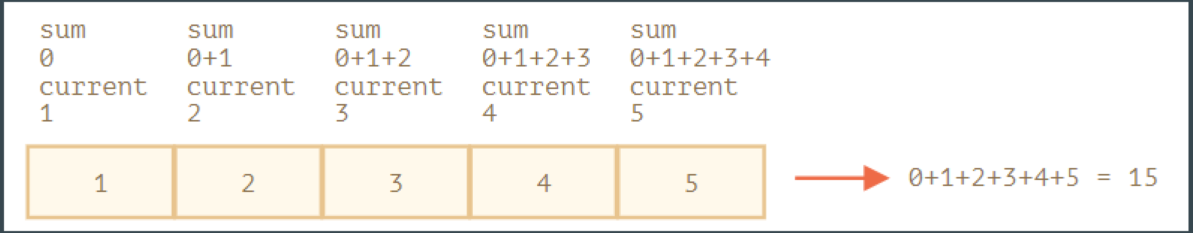
```
let value = arr.reduce(function (accumulator, item, index, array) {
    // ...
}, [initial]);
```

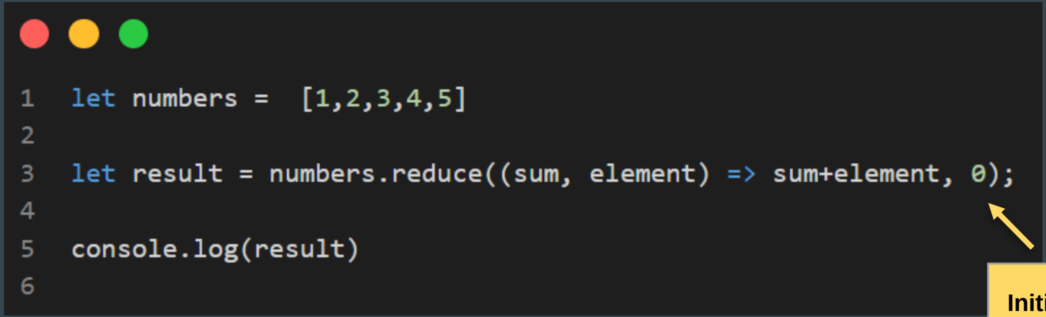
- Transform an array:
  - reduce(func)
    - the function is applied to all elements of the array, one after the other, and "continues" the result on the next call.
    - arguments:
      - accumulator - is the result of the previous function call, equal to initial the first time (if initial is provided)
      - item - is the current element of the array
      - index - the position
      - array - the array

M04 - Objects

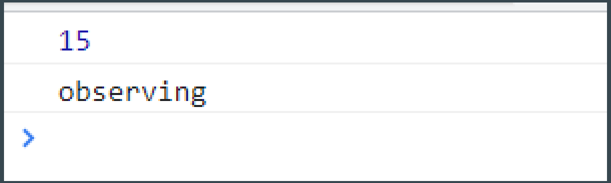
1. Arrays > Iteration (Arrays)

- Transform an array:
    - reduce(func)
        - example:





```
let numbers =  [1,2,3,4,5]

let result = numbers.reduce((sum, element) => sum+element, 0);

console.log(result)

```



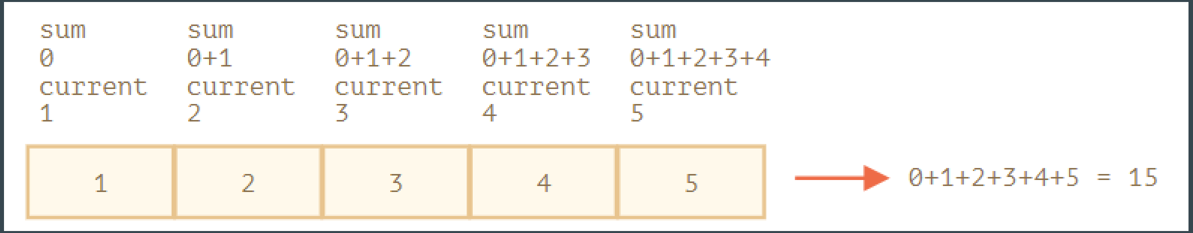
```
15
observing
>
```

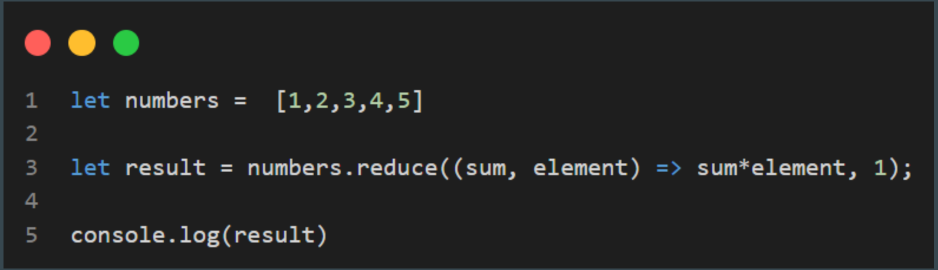
Initial value of accumulator (sum)

M04 - Objects

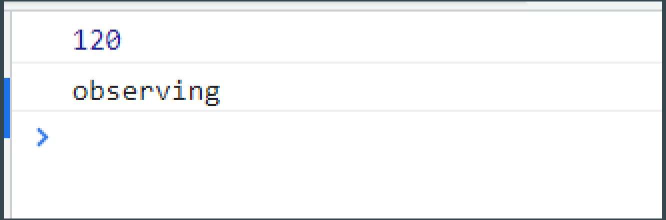
1. Arrays > Iteration (Arrays)

- Transform an array:
  - reduce(func)
    - example:





```
let numbers =  [1,2,3,4,5]

let result = numbers.reduce((sum, element) => sum*element, 1);

console.log(result)
```



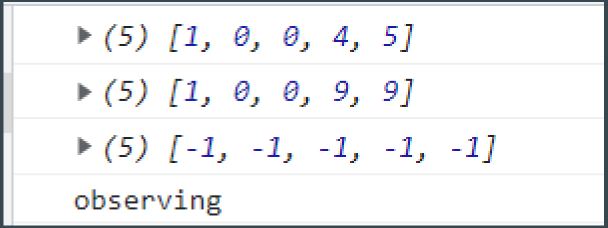
```
120
observing
>
```

Initial value of accumulator (sum)

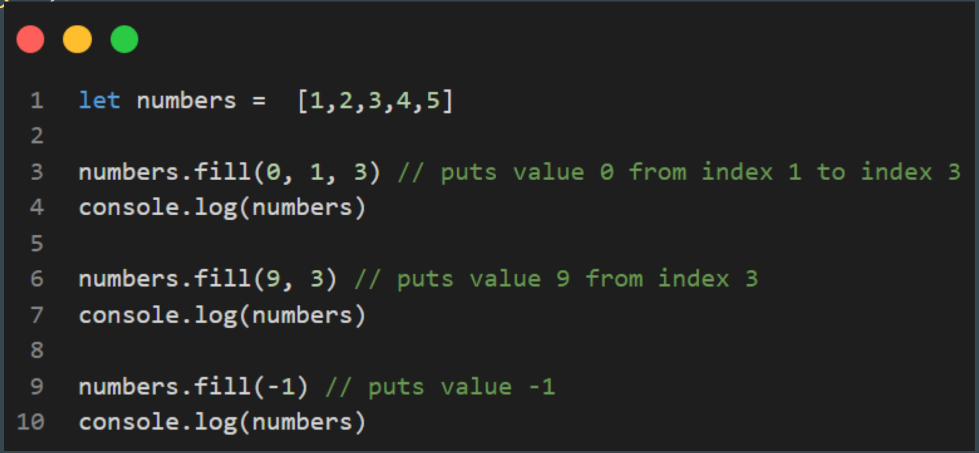
M04 - Objects

1. Arrays > Iteration (Arrays)

- Transform an array:
  - fill(value[, start[, end]])
    - changes all elements of an array to a static value, from an initial index (default 0) to a final index (default array.length)
    - returns the modified array



```
▸ (5) [1, 0, 0, 4, 5]
▸ (5) [1, 0, 0, 9, 9]
▸ (5) [-1, -1, -1, -1, -1]
observing
```

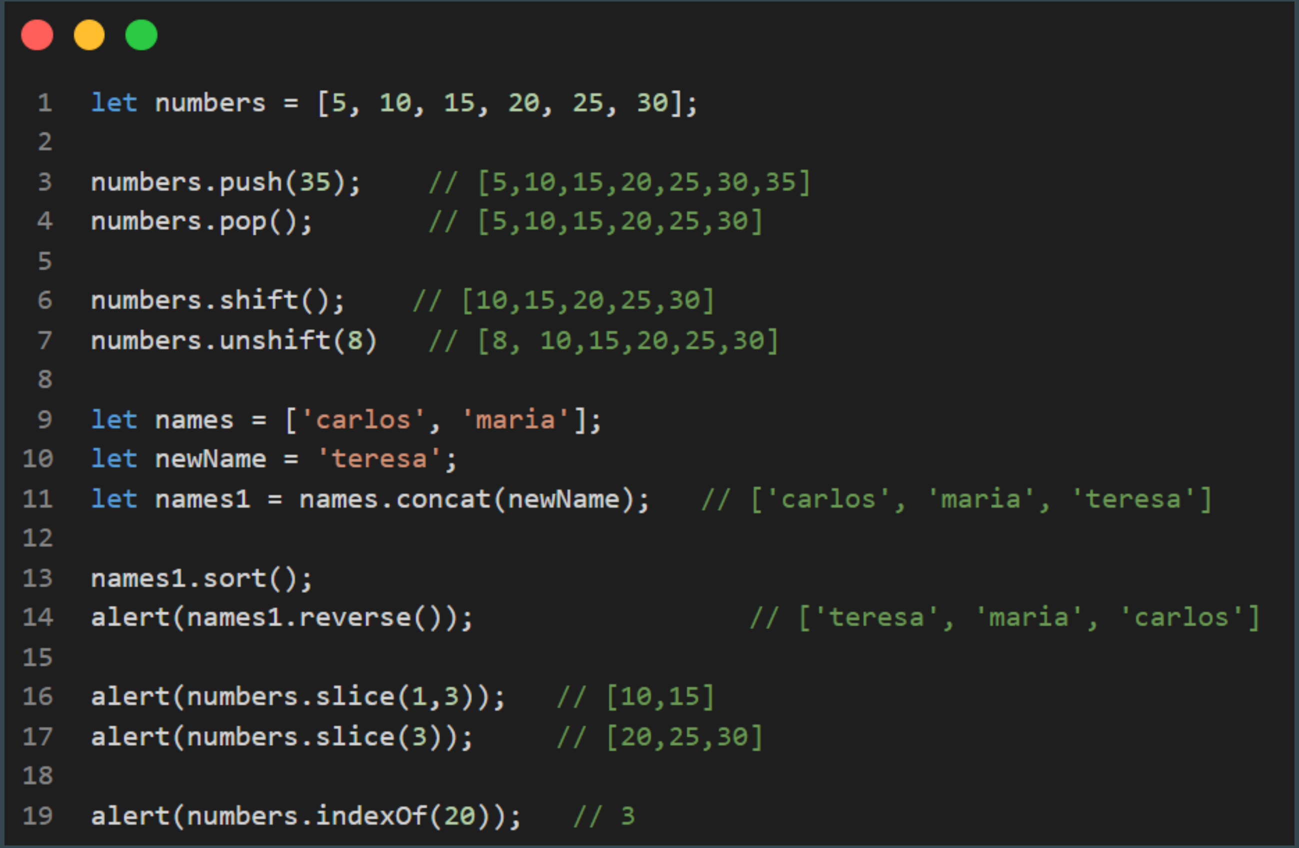


```
let numbers =  [1,2,3,4,5]

numbers.fill(0, 1, 3) // puts value 0 from index 1 to index 3
console.log(numbers)

numbers.fill(9, 3) // puts value 9 from index 3
console.log(numbers)

numbers.fill(-1) // puts value -1
console.log(numbers)
```

M04 - Objects

1. Arrays

- Examples:

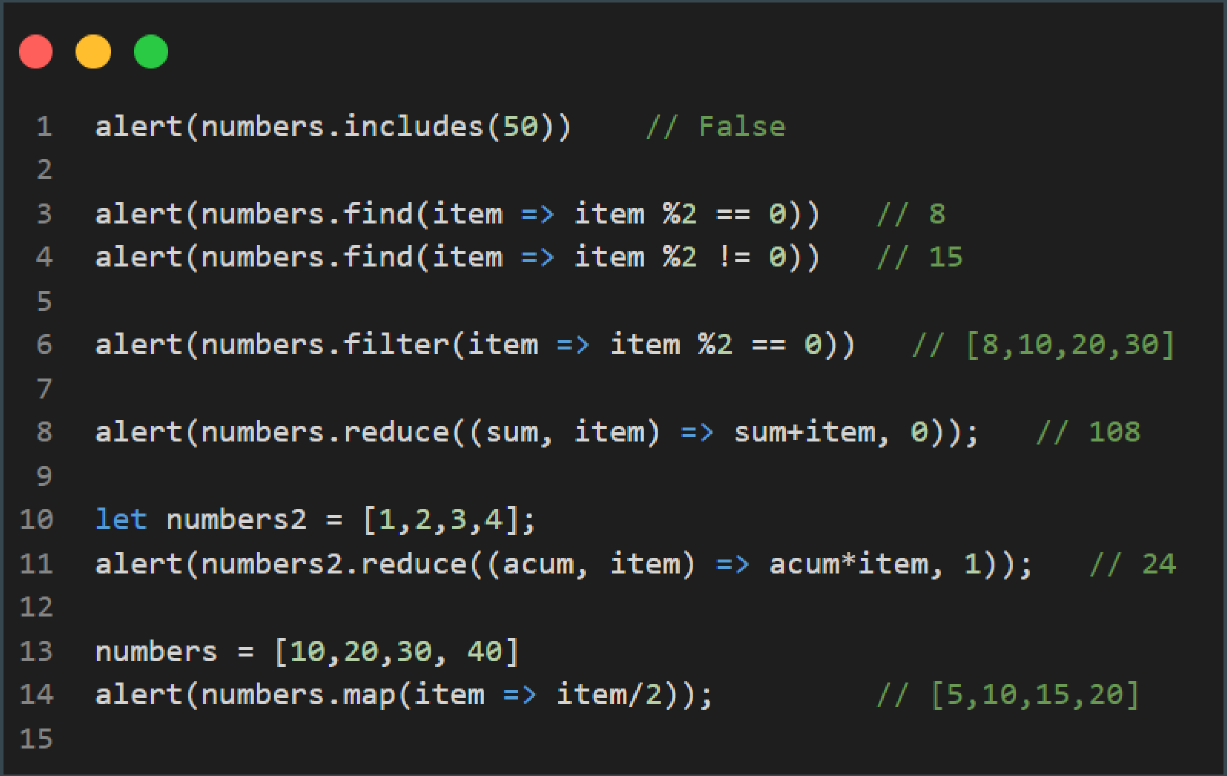


```javascript
let numbers = [5, 10, 15, 20, 25, 30];

numbers.push(35);     // [5,10,15,20,25,30,35]
numbers.pop();        // [5,10,15,20,25,30]

numbers.shift();     // [10,15,20,25,30]
numbers.unshift(8)   // [8, 10,15,20,25,30]

let names = ['carlos', 'maria'];
let newName = 'teresa';
let names1 = names.concat(newName);   // ['carlos', 'maria', 'teresa']

names1.sort();
alert(names1.reverse());                 // ['teresa', 'maria', 'carlos']

alert(numbers.slice(1,3));   // [10,15]
alert(numbers.slice(3));     // [20,25,30]

alert(numbers.indexOf(20));   // 3
```

M04 - Objects

1. Arrays

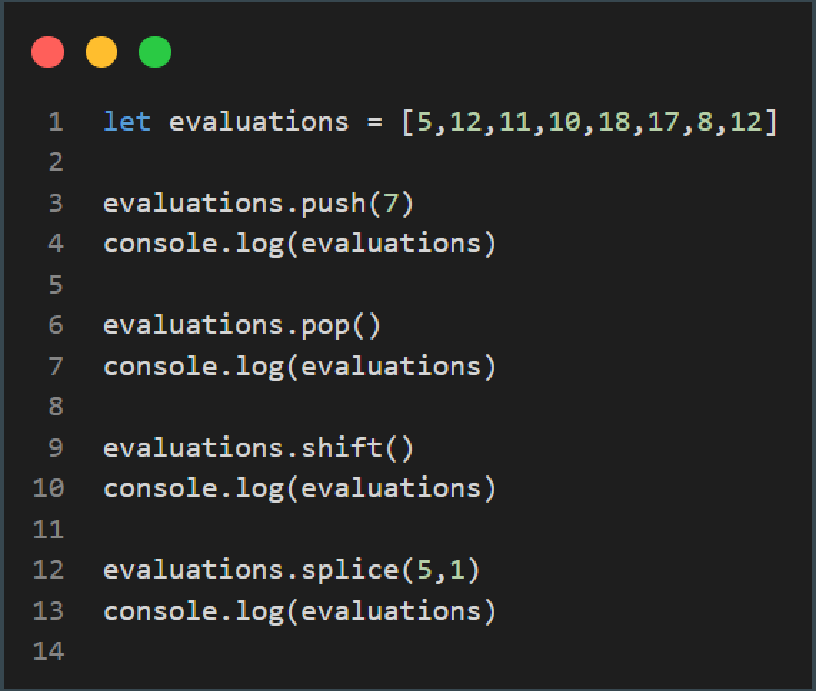
- Examples:



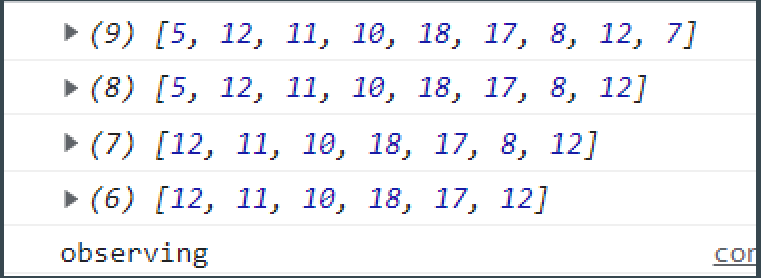
```
alert(numbers.includes(50))    // False

alert(numbers.find(item => item %2 == 0))   // 8
alert(numbers.find(item => item %2 != 0))   // 15

alert(numbers.filter(item => item %2 == 0))   // [8,10,20,30]

alert(numbers.reduce((sum, item) => sum+item, 0));   // 108

let numbers2 = [1,2,3,4];
alert(numbers2.reduce((acum, item) => acum*item, 1));   // 24

numbers = [10,20,30, 40]
alert(numbers.map(item => item/2));         // [5,10,15,20]

```

M04 - Objects

1. Arrays



```
let evaluations = [5,12,11,10,18,17,8,12]

evaluations.push(7)
console.log(evaluations)

evaluations.pop()
console.log(evaluations)

evaluations.shift()
console.log(evaluations)

evaluations.splice(5,1)
console.log(evaluations)

```

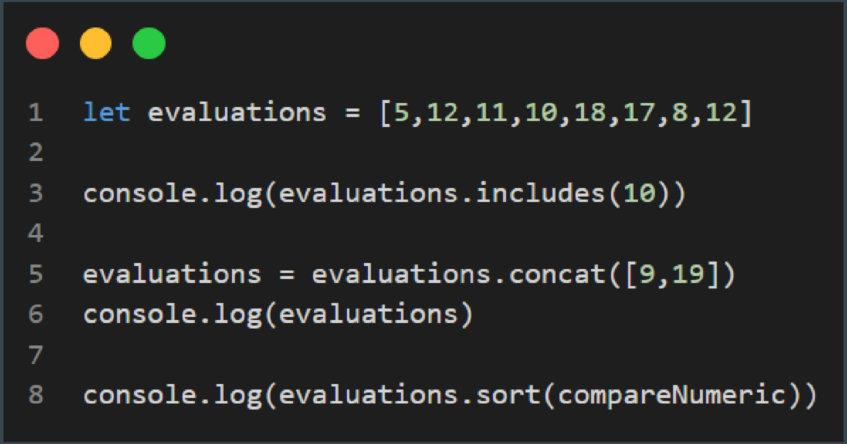


```
▶ (9) [5, 12, 11, 10, 18, 17, 8, 12, 7]
▶ (8) [5, 12, 11, 10, 18, 17, 8, 12]
▶ (7) [12, 11, 10, 18, 17, 8, 12]
▶ (6) [12, 11, 10, 18, 17, 12]
observing                                                con
```

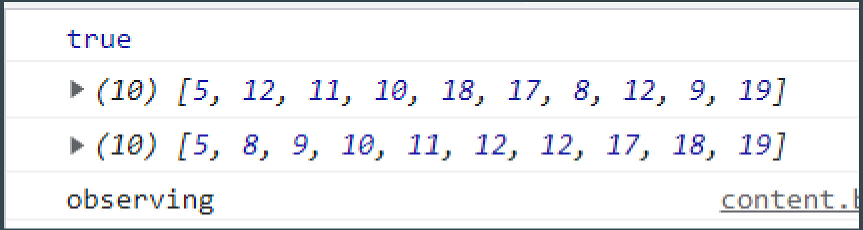
M04 - Objects

1. Arrays

- Examples:



```javascript
let evaluations = [5,12,11,10,18,17,8,12]

console.log(evaluations.includes(10))

evaluations = evaluations.concat([9,19])
console.log(evaluations)

console.log(evaluations.sort(compareNumeric))
```

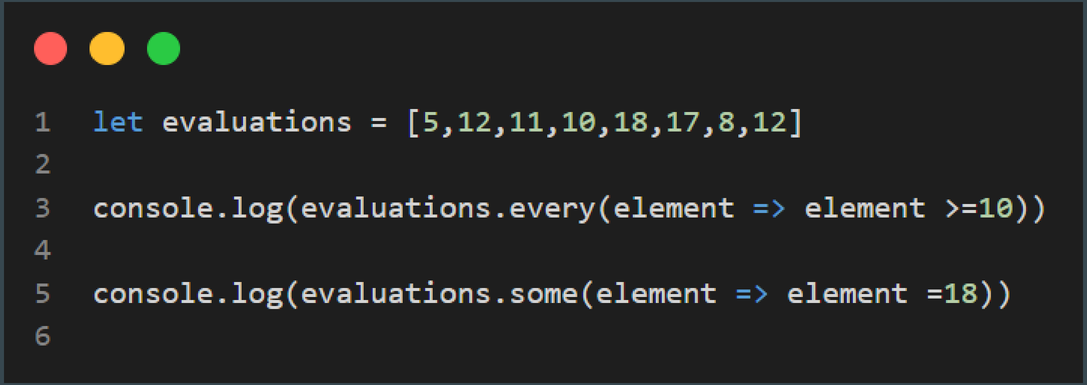


```
true
▶ (10) [5, 12, 11, 10, 18, 17, 8, 12, 9, 19]
▶ (10) [5, 8, 9, 10, 11, 12, 12, 17, 18, 19]
observing                                          content.
```

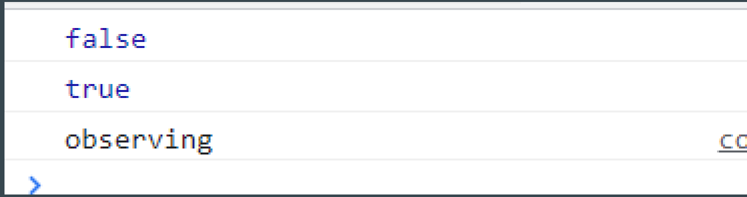
M04 - Objects

1. Arrays

- Examples:



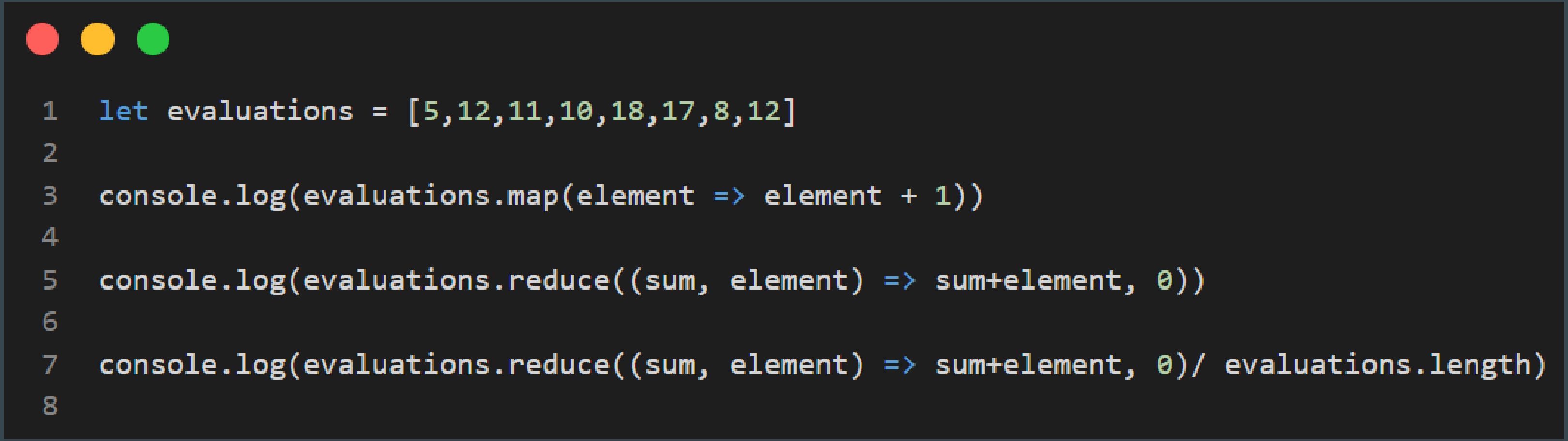
```
let evaluations = [5,12,11,10,18,17,8,12]

console.log(evaluations.every(element => element >=10))

console.log(evaluations.some(element => element =18))

```



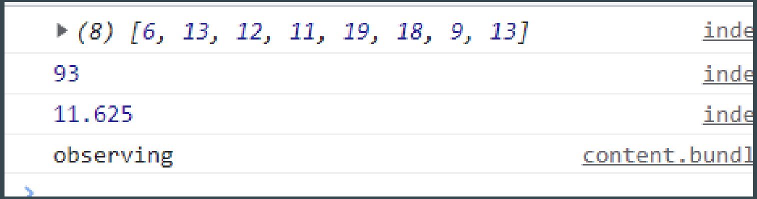
```
false
true
observing
```

M04 - Objects

1. Arrays



```javascript
let evaluations = [5,12,11,10,18,17,8,12]

console.log(evaluations.map(element => element + 1))

console.log(evaluations.reduce((sum, element) => sum+element, 0))

console.log(evaluations.reduce((sum, element) => sum+element, 0)/ evaluations.length)
```



```
▶ (8) [6, 13, 12, 11, 19, 18, 9, 13]          inde
93                                             inde
11.625                                         inde
observing                             content.bundl
```