

Associative Arrays

A Key-Value Pair Structure



SoftUni Team

Technical Trainers



SoftUni



Software University

<https://softuni.bg>

1. Associative Arrays

- Definition
- Attributes
- Iteration

2. Map

- Methods
- Sorting

3. Set



Have a Question?

sli.do

#fund-js




Associative Arrays

Storing Key-Value Pairs

What is an Associative Array ?

- Arrays indexed by **string keys**
- Hold a set of pairs **[key => value]**
 - The key is a **string**
 - The **value** can be of **any** type



Key	Value
John Smith	+1-555-8976
Lisa Smith	+1-555-1234
Sam Doe	+1-555-5030

- An associative array in JavaScript is just an **object**
- We can declare it **dynamically**

```
let assocArr = {  
  'one': 1,  
  'two': 2,  
  'three': 3,  
  [key]: 6  
};
```

Quotes are used if the key contains **special characters**

```
assocArr['four'] = 4;
```

```
assocArr.five = 5;
```

```
let key = 'six';  
assocArr[key] = 6;
```

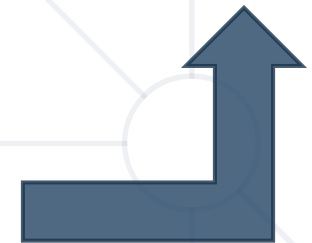
Valid ways to access **values** through **keys**

Using for – in

- We can use **for-in** loop to iterate through the keys

```
let assocArr = {};  
assocArr['one'] = 1;  
assocArr['two'] = 2;  
assocArr['three'] = 3;  
  
for(let key in assocArr) {  
    console.log(key + " = " + assocArr[key]);  
}
```

```
// one = 1  
// two = 2  
// three = 3
```



Problem: Phone Book

- Write a function that reads **names** and **numbers**
- Store them in an associative array and print them
- If the same name occurs, save the **latest** number

```
['Tim 0834212554',  
 'Peter 0877547887',  
 'Bill 0896543112',  
 'Tim 0876566344']
```



```
Tim -> 0876566344  
Peter -> 0877547887  
Bill -> 0896543112
```


Solution: Phone Book

```
function solve(input) {  
  let phonebook = {};  
  for (let line of input) {  
    let tokens = line.split(' ');  
    let name = tokens[0];  
    let number = tokens[1];  
    phonebook[name] = number;  
  }  
  for (let key in phonebook) {  
    console.log(`${key} -> ${phonebook[key]}`);  
  }  
}
```

Manipulating Associative Arrays

- Check if a key is **present**:

```
let assocArr = { /* entries */ };  
if (assocArr.hasOwnProperty('John Smith')) { /* Key found */ }
```

- **Remove** entries:

```
delete assocArr['John Smith'];
```

- Iterate **destructured** entries:

```
for (let [key, value] of Object.entries(assocArr)) {  
  console.log(`${key} -> ${value}`);  
}
```

Problem: Meetings

- Write a function that reads **weekdays** and **names**
- Print a **success** message for every successful appointment
- If the same weekday occurs a second time, print a **conflict message**
- In end, print a list of all meetings
- See **example** input and output on **next slide**

Example: Meetings

- Parsing input and success/conflict messages

```
['Monday Peter',  
'Wednesday Bill',  
'Monday Tim',  
'Friday Tim']
```



```
Scheduled for Monday  
Scheduled for Wednesday  
Conflict on Monday!  
Scheduled for Friday
```

- Final list output

```
Monday -> Peter  
Wednesday -> Bill  
Friday -> Tim
```

Solution: Meetings

```
function solve(input) {  
  let meetings = {};  
  for (let line of input) {  
    let [weekday, name] = line.split(' ');  
  
    if (meetings.hasOwnProperty(weekday)) {  
      console.log(`Conflict on ${weekday}!`);  
    } else {  
      meetings[weekday] = name;  
      console.log(`Scheduled for ${weekday}`);  
    }  
  }  
  
  // TODO: Print result  
}
```

Sorting Associative Arrays


- Objects **cannot be sorted**; they must be converted first
 - Convert to **array** for **sorting**, **filtering** and **mapping**:

```
let phonebook = { 'Tim': '0876566344',  
                  'Bill': '0896543112' };  
  
let entries = Object.entries(phonebook);  
console.log(entries); // Array of arrays with two elements each  
// [ ['Tim', '0876566344'],  
//   ['Bill', '0896543112'] ]  
  
let firstEntry = entries[0];  
console.log(firstEntry[0]); // Entry key -> 'Tim'  
console.log(firstEntry[1]); // Entry value -> '0876566344'
```

The entry is turned into an array of **[key, value]**

- The **entries** array can be **sorted**, using a **Compare function**
 - To **sort by key**, use the **first element** of each entry

```
entries.sort((a, b) => {  
  keyA = a[0];  
  keyB = b[0];  
  // Perform comparison and return negative, 0 or positive  
});
```



- You can also **destructure** the entries

```
entries.sort(([keyA, valueA], [keyB, valueB]) => {  
  // Perform comparison and return negative, 0 or positive  
});
```

Problem: Sort Addressbook

- Write a function that reads **names** and **addresses**
- Values will be separated by ":"
- If same name occurs, save the **latest** address
- Print list, **sorted** alphabetically by **name**

```
['Tim:Doe Crossing',  
'Bill:Nelson Place',  
'Peter:Carlyle Ave',  
'Bill:Ornery Rd']
```



```
Bill -> Ornery Rd  
Peter -> Carlyle Ave  
Tim -> Doe Crossing
```


Solution: Sort Addressbook

```
function solve(input) {  
  let addressbook = {};  
  for (let line of input) {  
    let [name, address] = line.split(':');  
    addressbook[name] = address;  
  }  
  let sorted = Object.entries(addressbook);  
  sorted.sort((a, b) => a[0].localeCompare(b[0]));  
  // TODO: Print result  
}
```

- The **destructuring assignment** syntax makes it possible to unpack values from arrays, or properties from objects, into distinct variables.
- On the left-hand side of the assignment to define what values to unpack from the sourced variable.

```
const x = [1, 2, 3, 4, 5];  
const [y, z] = x;  
console.log(y); // 1  
console.log(z); // 2
```

```
obj = { a: 1, b: 2 };  
const { a, b } = obj;  
// is equivalent to:  
// const a = obj.a;  
// const b = obj.b;
```

- To **sort by value**, use the **second element** of each entry

```
entries.sort((a, b) => {  
  valueA = a[1];  
  valueB = b[1];  
  // Perform comparison and return negative, 0 or positive  
});
```

- You can also **destructure** the entries

```
entries.sort(([keyA, valueA], [keyB, valueB]) => {  
  // Perform comparison and return negative, 0 or positive  
});
```




Set()

Sets

Storing Unique Elements

What is a Set?

- Store **unique values** of any type, whether **primitive** values or **object** references
- Set objects are **collections** of values



```
let set = new Set([1, 2, 2, 4, 5]);  
// Set(4) { 1, 2, 4, 5 }  
set.add(7); // Add value  
console.log(set.has(1));  
// Expected output: true
```

- Can **iterate** through the elements of a set in **insertion** order



Live Exercises

- We can use both **Objects** and **Maps** to store **key-value** pairs
- In practice, **Objects** are used more often
- **Maps** have advantages in some cases:
 - You may use **any data type** as **key**
 - They are **iterable**
 - They have a **size property**



Questions?



SoftUni Diamond Partners

SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето утре



POKERSTARS



CAREERS



AMBITIONED

DXC
TECHNOLOGY



**SOFTWARE
GROUP**

Bosch.IO

INDEAVR
Serving the high achievers

 **DRAFT
KINGS**

 **PHAR
VISION**



SmartIT

createX

**SUPER
HOSTING
.BG**



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg
 - Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University Forums
 - forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

