# Associative Arrays

## A Key-Value Pair Structure


**SoftUni Team**

**Technical Trainers**

Software University


SoftUni

Software University

https://softuni.bg

# Table of Contents



# Have a Question?

Software University

## 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 |

# Declaration

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

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

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

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

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

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

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 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 **conflict**
- At 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]

# Sorting By Key

- 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
}
```

# Sorting By Value

- 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
});
```

# Nested Data Structures

- The **values** of associative arrays can be **objects**, or **arrays**
- Once we have a **reference** to the value, we can **manipulate** it like any other object

```
let contacts = { 'Tim': { phone: '0876566344',
                          address: 'Doe Crossing' },
                 'Bill': { phone: '0896543112',
                           address: 'Nelson Place' } };

let billsContact = contacts['Bill']; // Get reference
console.log(billsContact.phone); // '0896543112'
```

# Sorting Nested Data Structures

- We can **sort** them by the **property values** of each entry
  - E.g. sort a contact book **alphabetically**, by person's address:

```
let entries = Object.entries(contacts);
entries.sort(([keyA, refA], [keyB, refB]) => {
  let addrA = refA.address;
  let addrB = refB.address;
  return addrA.localeCompare(addrB);
});
```


# Maps

## Storing Key-Value Pairs

# What is a Map?

- A **Map** collection stores its elements in **insertion order**
- A for-of loop returns an array of **[key, value]** for each iteration
- Pure **JavaScript objects** are like **Maps** in that both let you:
  - Assign **values** to **keys**
  - Detect whether something is stored in a key
  - Delete keys

# Adding/Accessing Elements

- `.set`(key, value) – adds a new key-value pair

```
let map = new Map();
map.set(1, "one"); // key - 1, value - one
map.set(2, "two"); // key - 2, value - two
```

- `.get`(key) – returns the value of the given key

```
map.get(2); // two
map.get(1); // one
```

- `.size` – **property**, holding the number of stored entries

# Contains / Delete

- `.has(key)` - checks if the map has the given key

```
map.has(2); // true
map.has(4); // false
```

- `.delete(key)` - removes a key-value pair

```
map.delete(1); // Removes 1 from the map
```

- `.clear()` - removes all key-value pairs

# Iterators

- `.entries()` - returns Iterator - array of **[key, value]**
- `.keys()` - returns Iterator with all the **keys**
- `.values()` - returns Iterator with all the **values**

```
let entries = Array.from(map.entries());
// [ [2, 'two'], [3, 'three'] ]
let keys = Array.from(map.keys()); // [2, 3]
let values = Array.from(map.values()); // ['two', 'three']
```

**These methods return an Iterator, transform it into an Array**

# Iterating a Map

- To print a map simply use one of the **iterators** inside a **for-of**

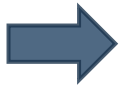
```
let iterable = phonebookMap.keys();
for(let key of iterable) {
  console.log(`${key} => ${phonebookMap.get(key)}`);
}
```

```
for(let [key, value] of phonebookMap) {
  console.log(`${key} => ${value}`);
}
```

# Problem: Storage

- Write a function that **stores products** and their **quantity**
- If the same product appears **more than once**, **add** the new quantity to the old one

```
['tomatoes 10',
 'coffee 5',
 'olives 100',
 'coffee 40']
```



```
tomatoes -> 10
coffee -> 45
olives -> 100
```

# Solution: Storage

```
let map = new Map();
for(let string of input) {
  let tokens = string.split(' ');
  let product = tokens[0];
  let quantity = Number(tokens[1]);
  if(!map.has(product)) {
    map.set(product, quantity);
  } else {
    let currQuantity = map.get(product);
    let newQuantity = currQuantity += quantity;
    map.set(product, newQuantity);
  }
}
// TODO: Print Map
```

# Map Sorting

- To **sort** a Map, first transform it into an **array**
- Then use the **`sort()`** method

Sort **ascending** by value

```
let map = new Map();
map.set("one", 1);
map.set("eight", 8);
map.set("two", 2);
let sorted = Array.from(map.entries())
                  .sort((a, b) => a[1] - b[1]);
for (let kvp of sorted) {
    console.log(`${kvp[0]} -> ${kvp[1]}`);
}
```

# Problem: School Grades

- Write a function to **store students** with all their **grades**
- If a student appears **more than once** add the **new grades**
- Print the students sorted **ascending** by **average grade**

```
['Lilly 4 6 6 5',
 'Tim 5 6',
 'Tammy 2 4 3',
 'Tim 6 6']
```

```
Tammy: 2, 4, 3
Lilly: 4, 6, 6, 5
Tim: 5, 6, 6, 6
```

# Solution: School Grades

```
function solve(input) {
  let map = new Map();
  for (let string of input) {
    let tokens = string.split(' ');
    let name = tokens.shift();
    let grades = tokens.map(Number);

    if (!map.has(name)) { map.set(name, []); }

    let existing = map.get(name);
    for (let grade of grades) { existing.push(grade); }
  }

  let sorted = Array.from(map).sort(compareAverage); // See next slide

  // TODO: Print result
}
```

# Solution: School Grades – Compare Function



```
function compareAverage(a, b) {
  // Calculate first average
  let avgA = 0;
  a[1].forEach(x => avgA += x);
  avgA /= a[1].length;

  // Calculate second average
  let avgB = 0;
  b[1].forEach(x => avgB += x);
  avgB /= b[1].length;

  // Comparison
  return avgA - avgB;
}
```


# Sets

Storing Unique Elements

# What is a Set?

Software University

- 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

# Summary

- 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

Software University | SoftUni Svetlina | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids

# SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

SmartIT

XS SOFTWARE

zühlke
empowering ideas

INFRAGISTICS®

Coca-Cola HBC Bulgaria

DRAFT KINGS

Postbank
Решения за твоето утре

SOFTWARE GROUP

# Educational Partners

Software University



# Trainings @ Software University (SoftUni)

- 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

# License

- 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