JavaScript 9

Roi Yehoshua 2018

[DAN.IT] EDUCATION

What we learnt last time?

- JavaScript Arrays
- Array methods
- Rest and Spread operators
- Iterables

[Our targets for today]

- Set
- Map
- Date and Time

[DAN.IT EDUCATION]

[Set]

- → Set is a collection of values, where each value may occur only once
- → Its main methods are:
  - → **new Set**(iterable) – creates the set, optionally from an array of values (any iterable will do)
  - → **set.add**(value) – adds a value, returns the set itself
  - → **set.delete**(value) – removes the value
    - → returns true if value existed at the moment of the call, otherwise false
  - → **set.has**(value) – returns true if the value exists in the set, otherwise false
  - → **set.clear**() – removes everything from the set
  - → **set.size** – the elements count

[DAN.IT] EDUCATION

Set Example

→ For example, we'd like to store all the users who have visited our site

→ But repeated visits should not lead to duplicates (a visitor must be counted only once)

→ Set is just the right thing for that:

```
let set = new Set();
let john = { name: "John" };
let peter = { name: "Peter" };  let mary = { name: "Mary" };

// visits, some users come multiple times  set.add(john);
set.add(peter);  set.add(mary);  set.add(john);
set.add(mary);

// set keeps only unique values  alert(set.size); // 3

for (let user of set) {
    alert(user.name); // John (then Peter and Mary)
}
```

Exercise (17)

- → Let arr be an array
- → Create a function unique(arr) that should return an array with unique items of arr
- → Use set to make the function more efficient
- → For instance:

```
function unique(arr) {
                                    /* your code */
}
let values = ["John", "Harry", "Mary", "Harry", "Beth", "Harry", "Mary", "John"];

alert(unique(values)); // John, Harry, Mary, Beth
```

[DAN.IT] EDUCATION

Exercise (18)

→ Write a function subArrayZero(arr) that gets an array and returns whether it contains a contiguous subarray whose sum is equal to 0

→ Your function should go over the array elements only once

```
function subArrayZero(arr) {
    // your code
}

alert(subArrayZero([-5, 12, 4, -7, 2, 1, 8])); // true, 4 + (-7) + 2 + 1 = 0
alert(subArrayZero([3, -2, -6, 2, 1, -2])); // false
```

[DAN.IT] EDUCATION

[Map]

- → Map is a collection of keyed data items, just like an Object
- → The main difference is that Map allows keys of any type
  - → Objects can also be keys
- → The main methods are:
  - → **new Map()** – creates the map.
  - → **map.set**(key, value) – stores the value by the key and returns the map
  - → **map.get**(key) – returns the value by the key, undefined if key doesn't exist in map
  - → **map.has**(key) – returns true if the key exists, false otherwise
  - → **map.delete**(key) – removes the value by the key
  - → **map.clear**() – clears the map
  - → **map.size** – returns the current element count

[DAN.IT] EDUCATION

Map Examples

```
let map = new Map();
map.set('1', 'str1');  // a string key
map.set(1, 'num1');    // a numeric key
map.set(true, 'bool1'); // a boolean key

// Map keeps the key type (unlike Object), so these two are different:
alert(map.get(1));      // 'num1'
alert(map.get('1'));    // 'str1'
alert(map.size);        // 3
```

```
// Using objects as keys
let user = { name: "John" };

// for every user, let's store his visits count  let visitsCountMap = new
Map();

// john is the key for the map  visitsCountMap.set(user, 123);

alert(visitsCountMap.get(john)); // 123
```

[DAN.IT] EDUCATION

Map From Object

→ When a Map is created, we can pass an array (or another iterable) with key-value pairs, like this:

```
let map = new Map([  ['1',
    'str1'],
    [1, 'num1'],
    [true, 'bool1']
]);
```

→ There is a built-in method Object.entries(obj) that returns an array of key/value pairs for an object exactly in that format

→ So we can initialize a map from an object like this:

```
let map = new Map(Object.entries({  name:
    "John",
    age: 30
}));
```

Iteration over Maps

→ For looping over a map, there are 3 methods:

- → **map.keys()** – returns an iterable for keys
- → **map.values()** – returns an iterable for values
- → **map.entries()** – returns an iterable for entries [key, value]
  - →It is used by default in for..of

```
let recipeMap = new Map([
        ['cucumber', 10],
        ['tomatoes', 15],
        ['onion', 3]
]);

// iterate over keys (vegetables)
for (let vegetable of recipeMap.keys()) {
        alert(vegetable); // cucumber, tomatoes, onion
}

// iterate over values (amounts)
for (let amount of recipeMap.values()) {  alert(amount); // 10, 15,
        3
}

// iterate over [key, value] entries
for (let entry of recipeMap) { // the same as of
recipeMap.entries()
        alert(entry); // cucumber,10 (and so on)
}
```

[Exercise (19)]

→ Create a function countWords(sentence) that gets a sentence and prints to the console the number of occurrences of each word in the sentence

→ For instance:

```
function countWords(sentence) {
                        // your code
}

let sentence = "John the second is the son of John the
first,  while the second son of John the second is William
the  second.";
countWords(sentence);
```

```
John 3
the 6
second 4
is 2
son 2
of 2
first 1
while 1
William 1
```

[DAN.IT] EDUCATION

Exercise (20)

- → Anagrams are words that have the same number of same letters, but in different order
- → For instance:
  - → nap - pan
  - → ear - are - era
  - → cheaters - hectares – teachers
- → Write a function aclean(arr) that returns an array cleaned from anagrams
- → For instance:

```
let arr = ["nap", "teachers", "cheaters", "PAN", "ear", "era",
"hectares"];

alert(aclean(arr)); // "nap,teachers,ear" or "PAN,cheaters,era"
```

  - →From every anagram group should remain only one word, no matter which one

[DAN.IT EDUCATION]

Date and Time

- → Let's meet a new built-in object: **Date**
- → It stores the date, time and provides methods for date/time management
- → For instance, we can use it to measure time, or just to print out the current date
- → To create a new Date object call new Date() with one of the following arguments:
  - → **new Date()** - creates a Date object for the current date and time
  - → **new Date(milliseconds)** - creates a Date object with the time equal to number of milliseconds passed after the Jan 1st of 1970 UTC+0 (this is called a timestamp)
  - → **new Date(datestring)** - reads the date from a string
  - → **new Date(year, month, date, hours, minutes, seconds, ms)** - creates the date with the given components in the local time zone
    - → The year must have 4 digits: 2013 is okay, 98 is not
    - → The month count starts with 0 (Jan), up to 11 (Dec)
    - → The date parameter is actually the day of month, if absent then 1 is assumed
    - → If hours/minutes/seconds/ms is absent, they are assumed to be equal 0

[DAN.IT] EDUCATION

Date Creation Example

```
let now = new Date();
alert(now); // shows current date/time

// 0 means 01.01.1970 UTC+0  let Jan01_1970 = new
Date(0);  alert(Jan01_1970);

let date = new Date("2018-05-25");  alert(date); //
Fri May 25 2018 ...

let date2 = new Date(2011, 0, 1, 2, 3, 4, 567);
alert(date2); // 1.01.2011, 02:03:04.567

new Date(2011, 0, 1); // 1 Jan 2011, 00:00:00
```

Access Date Components

→ There are many methods to access the year, month and so on from the Date object:

- → **getFullYear()** - get the year (4 digits)
- → **getMonth()** - get the month, **from 0 to 11**
- → **getDate()** - get the day of month, from 1 to 31 (the method name may look strange)
- → **getHours(), getMinutes(), getSeconds(), getMilliseconds()** - get the corresponding time components
- → **getDay()** - get the day of week, from 0 (Sunday) to 6 (Saturday)

→ All the methods above return the components relative to the local time zone

→ There are also their UTC-counterparts, that return day, month, year and so on for the time zone UTC+0: getUTCFullYear(), getUTCMonth(), getUTCDay()

Access Date Components

```
let currDay = now.getDate();
let currMonth = now.getMonth() + 1;  let currYear = now.getFullYear();
alert(`${currDay}/${currMonth}/${currYear}`); // 25/5/2018

// the hour in your current time zone  alert(now.getHours());

// the hour in UTC+0 time zone (London time without daylight savings)
alert(now.getUTCHours());
```

Measuring Time Difference

→ Dates can be subtracted, the result is their difference in ms

→ However, if we only want to measure the difference, we don't need the Date object

→ There's a special method **Date.now()** that returns the current timestamp

→ It is semantically equivalent to new Date().getTime(), but it doesn't create an intermediate Date object, so it's faster

→ For instance:

```
let start = Date.now(); // milliseconds count from 1 Jan 1970

// do the job
for (let i = 0; i < 100000; i++) {  let doSomething = i * i * i;
}

let end = Date.now(); // done
alert(`The loop took ${end - start} ms`); // subtract numbers, not dates
```

[Exercise (22)]

→ Create a function getSecondsToTomorrow() that returns the number of seconds till tomorrow

→ For instance, if now is 23:00, then:

```
getSecondsToTomorrow() == 3600
```

→ Note that the function should work at any day

[DAN.IT] EDUCATION

Control questions

1. What is Set?
2. What is the difference between Array.push and Set.add?
3. What is Map?
4. What can be used as Map key?
5. How is time and date stored in JavaScript?
6. How can we find how much time have passed between two dates?

Materials

Core materials:
https://learn.javascript.ru/set-map

https://learn.javascript.ru/datetime

Additional materials:

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Set
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Map
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Date

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

https://habr.com/company/ruvds/blog/350536/

Video materials:
https://youtu.be/SbPtW-hiWZI

[DAN.IT] EDUCATION