[Lesson 8]

Roi Yehoshua 2018

[DAN.IT EDUCATION]

What we learnt last time?

- Basics of Object Oriented Programming
- JavaScript objects
- Object methods
- Object cloning
- ***this*** keyword

[Our targets for today]

- Working with strings
- Date and Time

[DAN.IT EDUCATION]

Strings

→ In JavaScript, the textual data is stored as strings

→ There is no separate type for a single character

→ The internal format for strings is always UTF-16, it is not tied to the page encoding

→ Strings can be enclosed within either single quotes, double quotes or backticks:

```
let  single  =  'single-quoted';
let  double  =  "double-quoted";
let backticks = `backticks`;
```

→ Single and double quotes are essentially the same

→ Backticks, however, allow us to embed any expression into the string, including function calls:

```
function sum(a, b) {
    return a + b;
}

alert(`1 + 2 = ${sum(1, 2)}.`); // 1 + 2 = 3.
```

[DAN.IT] EDUCATION

Strings

→ Another advantage of using backticks is that they allow a string to span multiple lines:

```
let guestList = `Guests:
        * John
        * Peter
        * Mary
`;
alert(guestList); // a list of guests, multiple lines
```

Special Characters

→ You can create multiline strings with single quotes by using a so-called “newline character”, written as **\n**, which denotes a line break:

```
let guestList = "Guests:\n * John\n * Peter\n * Mary";
alert(guestList); // a multiline list of guests
```

→ There are other, less common “special” characters as well

→ All special characters start with a backslash character \, also called an “escape character”

Character	Description
\b	Backspace
\r	Carriage return
\t	Tab
\uNNNN	A unicode symbol with the hex code NNNN, for instance \u00A9 – is a unicode for the copyright symbol ©. It must be exactly 4 hex digits.
\u{NNNNNNNN}	Some rare characters are encoded with two unicode symbols, taking up to 4 bytes

[DAN.IT] EDUCATION

Special Characters

→ Example with unicode:

```
alert("\u00A9"); // ©
alert("\u{20331}"); // 佱 , a rare chinese hieroglyph (long unicode)
alert("\u{1F60D}"); // 😍 , a smiling face symbol (another long unicode)
```

→ But what if we need to show an actual backslash \ within the string?

→ That's possible, but we need to double it like \\:

```
alert(`The backslash: \\`); // The backslash: \
```

[DAN.IT EDUCATION]

String Length

→ The length property has the string length:

```
alert('My\n'.length); // 3
```

→ Note that \n is a single “special” character, so the length is indeed 3

→ Please note that str.length is a numeric property, not a function

→ There is no need to add brackets after it

Accessing Characters

→ To get a character at position pos, use square brackets [pos] or call str.charAt(pos)

→ charAt() exists mostly for historical reasons

→ The first character starts from the zero position:

```
let str = 'Hello';

// the first character
alert(str[0]); // H
alert(str.charAt(0)); // H

// the last character
alert(str[str.length - 1]); // o
```

→ We can also iterate over characters using for..of:

```
for (let char of 'Hello') {
    alert(char); // H,e,l,l,o
}
```

[DAN.IT EDUCATION]

String are Immutable

→ Strings can't be changed in JavaScript. It is impossible to change a character.

→ Let's try it to show that it doesn't work:

```
let str = 'Hi';

str[0] = 'h';    // doesn't work
alert(str[0]);   // H
```

→ The usual workaround is to create a whole new string and assign it to str instead of the old one:

```
str = 'h' +  str[1];   // replace the string
alert(str);            // hi
```

[DAN.IT] EDUCATION

Changing the Case

→ Methods **toLowerCase()** and **toUpperCase()** change the case:

```
alert('Interface'.toUpperCase());  // INTERFACE
alert('Interface'.toLowerCase());  // interface
```

→ Or, if we want a single character lowercased:

```
alert('Interface'[0].toLowerCase()); // 'i'
```

Searching for substrings

→ There are multiple ways to look for a substring within a string

→ **str.indexOf**(substr, pos) looks for the substr in str, starting from the given position pos, and returns the position where the match was found or -1 if nothing can be found

```
let str = 'Widget with id';

alert(str.indexOf('Widget')); // 0, because 'Widget' is found at the beginning
alert(str.indexOf('widget')); // -1, not found, the search is case-sensitive
alert(str.indexOf("id")); // 1, "id" is found at the position 1 (..idget with id)
alert(str.indexOf("id", 2)) // starting the search from position 2
```

→ There is also a similar mфethod **str.lastIndexOf**(pos) that searches from the end of a string to its beginning

```
alert(str.lastIndexOf("id")); // 12
```

[DAN.IT] EDUCATION

Searching for substrings

→ If we're interested in all occurrences, we can run indexOf in a loop

→ Every new call is made with the position after the previous match

```
let str = 'As sly as a fox, as strong as an ox';
let target = 'as'; // let's look for it
let pos = 0;
while (true) {
    let foundPos = str.indexOf(target, pos);
    if (foundPos == -1) break;

    alert(`Found at ${foundPos}`);
    pos = foundPos + 1; // continue the search from the next position
}
```

→ The same algorithm can be layed out shorter:

```
let pos = -1;
while ((pos = str.indexOf(target, pos + 1)) != -1) {
    alert(`Found at ${pos}`);
}
```

[DAN.IT] EDUCATION

Searching for substrings

→ **str.includes**(substr, pos) returns whether str contains substr within

→ It's useful if we need to test for the match, but don't need its position

→ The optional second argument of str.includes is the position to start searching from

```
alert("Midget".includes("id")); // true
alert("Midget".includes("id", 3)); // false, from position 3 there is no "id"
```

→ The methods **str.startsWith**() and **str.endsWith**() do exactly what they say:

```
alert("Widget".startsWith("Wid")); // true, "Widget" starts with "Wid"
alert("Widget".endsWith("get"));   // true, "Widget" ends with "get"
```

[DAN.IT] EDUCATION

Getting a substring

→ There are 3 methods in JavaScript to get a substring:

Method	Selects…	Negatives
slice(start, end)	from start to end (not including end)	allows negatives
substring(start, end)	between start and end allows start to be greater than end	negative values mean 0
substr(start, length)	from start get length characters	allows negative start

→ Negative values for start/end mean that the position is counted from the string end

→ Examples for slice():

```
let str = "stringify";

alert(str.slice(0, 5)); // 'strin', the substring from 0 to 5 (not including 5)
alert(str.slice(0, 1)); // 's', from 0 to 1, but not including 1, so only character at 0
alert(str.slice(2)); // ringify, from the 2nd position till the end

alert(str.slice(-4, -1)); // gif, start at the 4th position from the right, end at the 1st from the right
```

[DAN.IT] EDUCATION

Getting a substring

→ Examples for substring():

```
let str = "stringify";

// these are same for substring
alert(str.substring(2, 6)); // "ring"
alert(str.substring(6, 2)); // "ring"

// ...but not for slice:
alert(str.slice(2, 6)); // "ring" (the same)
alert(str.slice(6, 2)); // "" (an empty string)
```

→ Examples for substr():

```
let str = "stringify";
alert(str.substr(2, 4)); // ring, from the 2nd position get 4 characters
alert(str.substr(-4, 2)); // gi, from the 4th position get 2 characters
```

→ Although all three methods can do the same job, slice() is more commonly used

[DAN.IT] EDUCATION

Exercise (1)

→ Write a function checkSpam(str) that returns true if str contains ‘viagra’ or ‘XXX’, otherwise false

→ The function must be case-insensitive:

```
alert(checkSpam('buy ViAgRA now')); // true
alert(checkSpam('free xxxxx')); // true
alert(checkSpam("innocent rabbit")); // false
```

Exercise (2)

→ Create a function truncate(str, maxlength) that checks the length of the str and, if it exceeds maxlength – replaces the end of str with the ellipsis character "…", to make its length equal to maxlength

→ The result of the function should be the truncated (if needed) string

→ For instance:

```
alert(truncate("What I'd like to tell on this topic is:", 20)); // "What I'd like to te…"
alert(truncate("Hi everyone!", 20)); // "Hi everyone!"
```

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

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()

[DAN.IT EDUCATION]

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
```

[DAN.IT] EDUCATION

Exercise (3)

→ 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

Control questions

1. How can we add special character on page?
2. How can we find a substring inside a string?
3. How is time and date stored in JavaScript?
4. How can we find how much time have passed between two dates?