Classes / Built-in Objects

IIKG1002/IDG1011 – Front-end web development

Johanna Johansen
johanna.johansen@ntnu.no

Classes

- Previous week: How objects are created with the **literal** and the **object constructor notations**
- **Prototype-based inheritance**
  - two or more objects inherit from the same prototype
  - two or more objects are created and initialized by the same **constructor function**
- Creating objects from **classes**
  - classes use the prototype-based inheritance

- With the ES6 the `class` keyword was introduced
- **Classes** – a set of objects that inherit properties from the same prototype object
  - a set of objects that inherit properties from the same prototype – **instances** of the same class

- Coding convention: classes have names that begin with **capital letter**
  - regular functions and methods have names that begin with lowercase letters
  - convention that keeps constructor functions/ classes distinct from regular functions
  - differentiate between function invocations and constructor invocations – constructors are invoked with the `new` keyword

Constructor function

```
function Hotel(name, rooms, booked) {
    this.name = name;
    this.rooms = rooms;
    this.booked = booked;

    this.checkAvailability = function() {
      return this.rooms - this.booked;
    };
}
```

Classes

```
class Hotel {
    constructor(name, rooms, booked) {
      this.name = name;
      this.rooms = rooms;
      this.booked = booked;
    }

    checkAvailability() {
      return this.rooms - this.booked;
    }
}
```

Syntax

- declared with the `class` keyword
- name of the class with first letter capitalized
- the class body in curly braces
  - the body includes method definitions where the function keyword is omitted
  - no commas are used to separate the methods from each other
  - the keyword `constructor` is used to define the constructor function for the class

The constructor function

- used to intialize objects – we provide some initial values
- The `this` keyword refers to the newly created object
  - the newly created object is the invocation context

The object creation expression

- similar to the object constructor notation
  - creates a new object and invokes the constructor function to initialize the properties of that object
- the constructor invocation can include an argument list in parentheses
  - the arguments are evaluated and passed to the function, similarly to functions and method invocations

```
new Hotel('Thon', 40, 25);
new Hotel('Radisson Blu', 120, 77);
```

- Other aspects of classes, more advanced, not for exam
  - static methods
  - getters, setters, and other method forms
  - public, private, and static fields
  - subclasses with extends and super

Syllabus – reading

- Flanagan – Chapter 9, pp. 221 – 231
  - optional (not for exam) – the rest of the chapter, pp. 231 – 248
- Alternatively – Mozilla Developer
  - "Classes" – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes
  - "Using classes" – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_Classes

Storing data

- We can store data in JavaScript using **variables**, **arrays**, **objects,** and **classes**
  - in **variables** we can store one piece of information
    - that value can be retrieved by using the name of the variable
  - with arrays and objects we can store multiple pieces of information
    - **arrays**
      - the order of the information is important, because the items in an array are assigned a number (an index)
      - to retrieve an item we use the index number; e.g. `hotel[1]`;
      - if the order of the information matters, use arrays
    - **objects**
      - we access items via a key (property/method name) and the dot / square bracket notation; e.g., `hotel.name;` or `hotel['name']`;
      - the key needs to be unique
    - we can combine arrays, with objects to create complex data structures; we can have arrays in objects and objects in arrays, i.e., one array as the value of an object

Built-in objects

The JavaScript standard library

- Useful classes and functions that are built in to JavaScript
  - made available to all JavaScript programs and implemented in web browsers
- we need to learn which these are, and what their main properties and methods do; what can we use them for
  - we access these using the dot notation, similarly with how we did for our self-created objects
- they are a toolkit with functionality
  - functionality that is commonly needed for many scripts
  - e.g., the width of the browser window or the content of the main heading in the page

Object model

- an object can be used to create a model of something from the real world using data
- an object model is a group of objects
  - these represent related things from the real world
  - together they form a model of something larger
- A property of an object can be another object
  - an object is nested inside another object; it is a **child object**

- Three groups of built-in objects
  - The Browser Object Model
    - a group of objects that represent the current browser window or tab
    - contains objects that model things like browser history and the device's screen
  - The Document Object Model (one lecture dedicated to this topic)
    - a group of objects that represent the current page
    - creates a new object for each HTML element, as well as each individual section of text, within the page
  - The global JavaScript objects / classes
    - they do not form a single model
    - they are individual objects / classes that relate to different parts of the JavaScript language
    - e.g., classes for dealing with dates and time

Built-in Objects

– The Browser Object Model –

Window

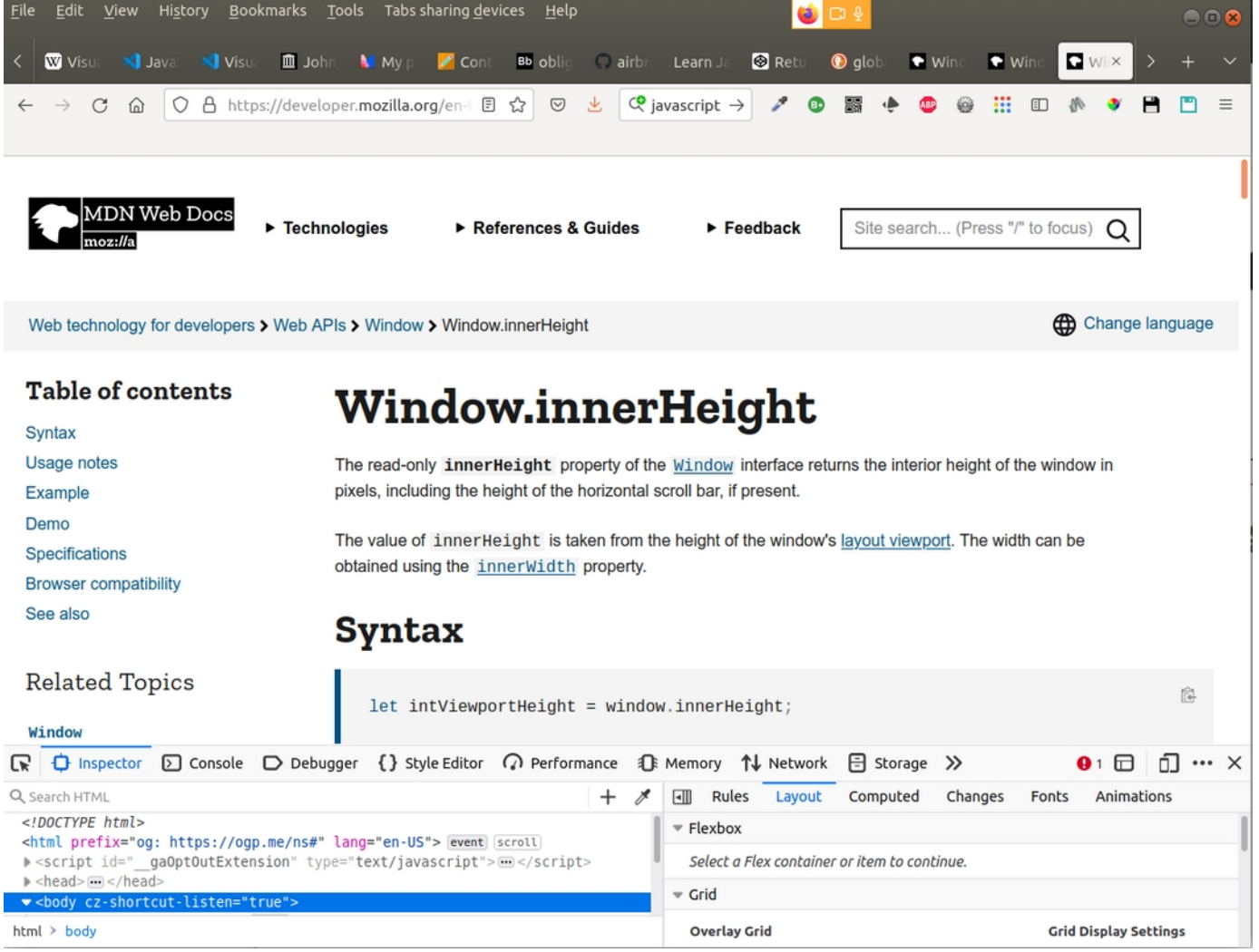
- The top most object is the *window* object → represents the current browser window/tab
  - overview of methods and props. for window object: https://developer.mozilla.org/en-US/docs/Web/API/Window
- Window interface – type of objects

Window – properties

- `window.innerHeight` / `window.innerWidth`
  - height/width of the window in px,
    - excluding browser chrome
    - including the browser's scroll if present
  - the **layout viewport** / visual viewport

Viewport

- the part of the document that is currently visible
- everything that is currently visible in the browser window, without the browser's chrome, i.e., without menu and address bar
- the size of the viewport depends on the size of the screen or whether the browser is in full screen mode
- is part of the document that we are viewing
  - a document can be very long; you may need to scroll in order to bring another part of the document to be visible in the viewport
- Read more at: https://developer.mozilla.org/en-US/docs/Web/CSS/Viewport_concepts

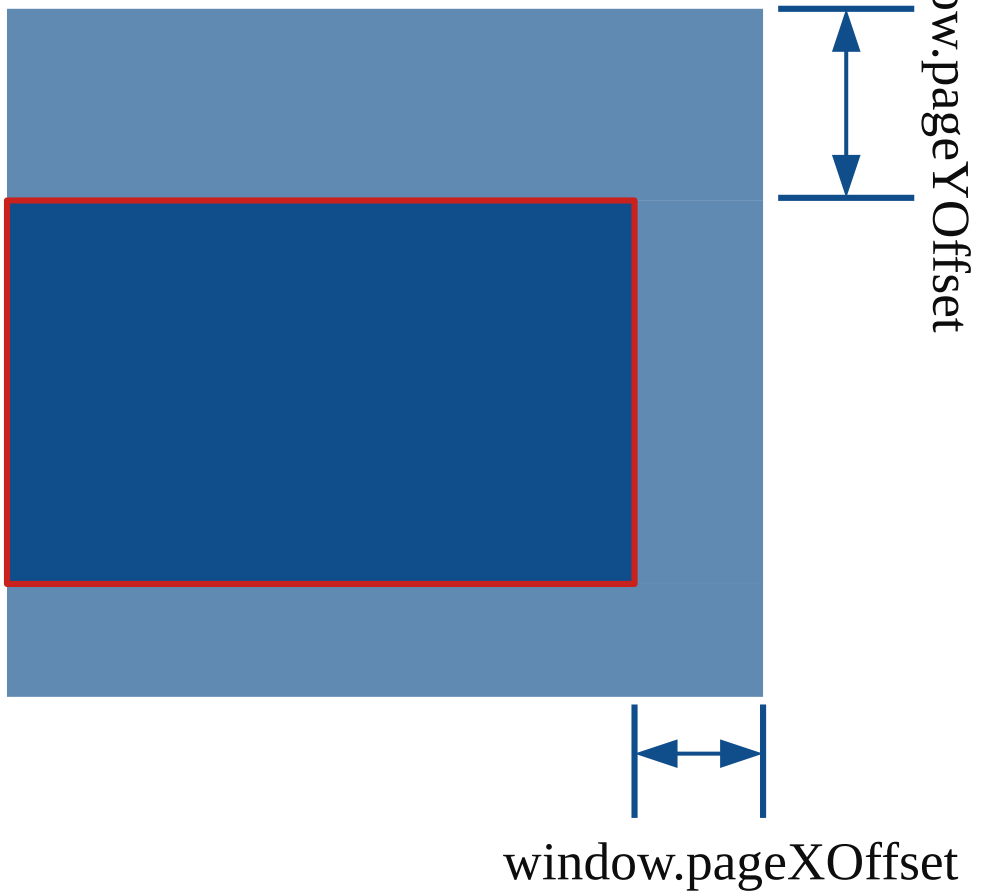


window.innerHeight

window.innerWidth

Window – properties

- `window.pageXOffset` / `window.pageYOffset`
  - The *pageXOffset* property is an alias for the `scrollX` / `scrollY` properties
  - distance the document has been scrolled horizontally/vertically in px



Window – properties / child objects

- `window.location`
  - gets/sets the location, or current URL, of the window object
- `window.document`
  - returns a reference to the document contained in the window
  - `window.document.title` → sets or gets the title of the current document

Window – properties / child objects

- `window.history`
  - returns a reference to the history object
  - `window.history.length` → returns an integer representing the number of elements in the session history, including the currently loaded page
- `window.screen`
  - returns a reference to the screen object associated with the window
  - `window.screen.height` / `window.screen.width` → read-only properties that return the height/width of the device's screen in pixels

Window – methods

- `window.alert()`
  - creates a dialog box with messages for the users
- `window.open()`
  - opens new browser window with URL specified as parameter
- `window.print()`
  - tells the browser that the user wants to print the contents of the current page
    - acts the same as when the user clicked the print option from the browser's user interface

Exercise

- Write to the Console (Browser Developer Tools) of one chosen page

  e.g., https://developer.mozilla.org/en-US/docs/Web/API/Window

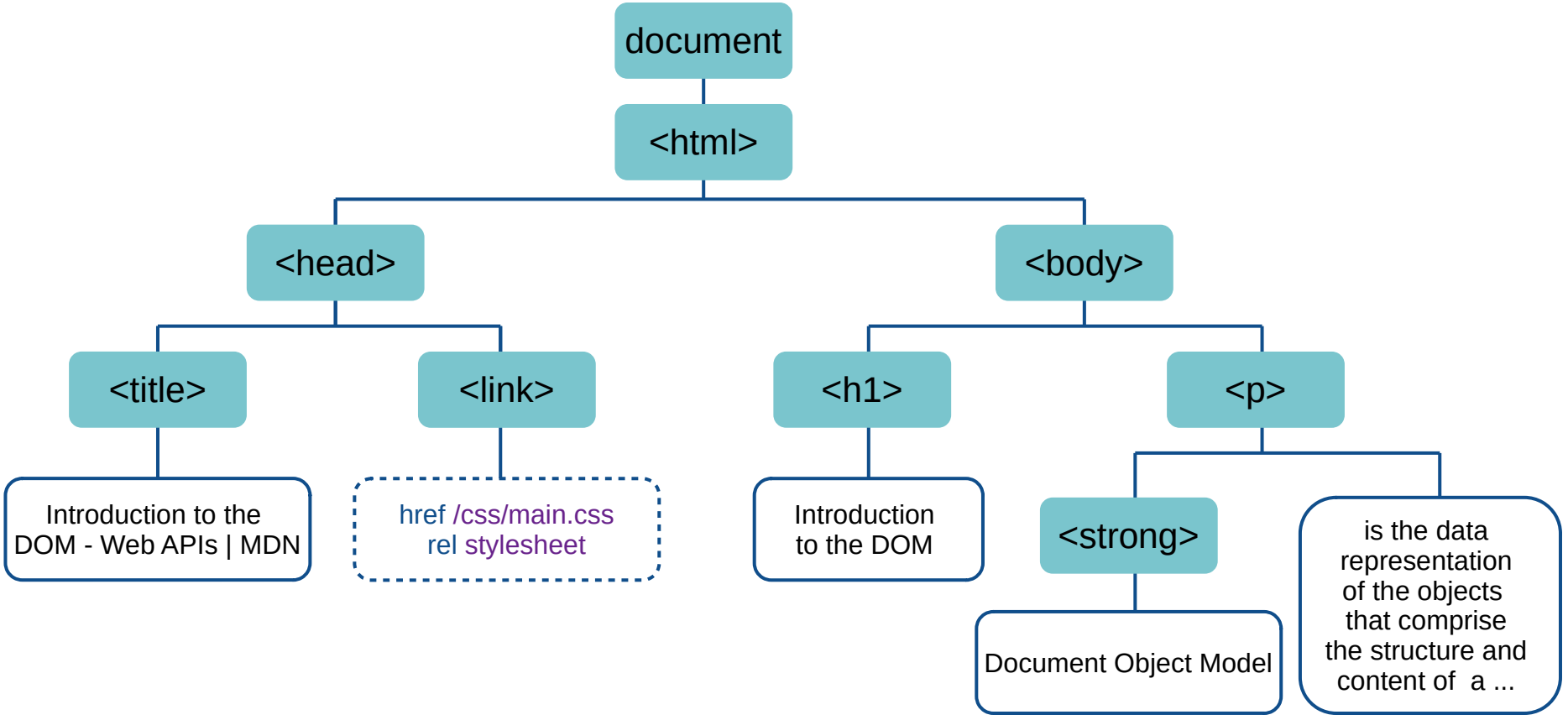
  the properties presented on page 124 of the Duckett book, and “Run” the code to see the output.

Built-in Objects

– The Document Object Model –

Document Object Model

- creates a model of the current web page
- the top most object is the *document* object
  - represents the web page loaded into the current browser window
- its child objects represent other items on the page
- e.g., `document.getElementById(id)`;
  - method of the document object that gets an HTML element by the value of its *id* attribute


document
<html>
<head>
<body>
<title>
<link>
<h1>
<p>
Introduction to the DOM - Web APIs | MDN
href /css/main.css
rel stylesheet
Introduction to the DOM

is the data representation of the objects that comprise the structure and content of a ...
Document Object Model

DOM – properties

- `document.title` → title of the current document
- `document.lastModified` → date on which document was last modified
- `document.URL` → returns a string containing URL of the current document

DOM – methods

- `document.write(markup)`
  - writes text to document
  - can include markup
  - after the pages has loaded, it removes all existing nodes from the current document
  - write

    ```
    document.write("<h1>Out with the old, in with the new!</h1>");
    ```

    to the console

DOM – methods

- `document.querySelectorAll(selectors)`
  - returns a list of **all** elements that match a CSS selector, which is specified as parameter

    e.g., `document.querySelectorAll('p')` → gives a list with all the paragraph elements in a document

    `document.querySelectorAll('div.note, div.alert')` → returns a list of all elements within the document with a *class* of either *note* or *alert*
- `querySelector(selectors)` → returns the **first** element within the document that matches the specified selector, or group of selectors
- if no matches are found, *null* is returned for both cases

DOM – methods

- `document.createElement(tagName)`
  - creates a new HTML element
  - e.g., `document.createElement('div');` → creates a new div element
- `document.createTextNode(data)`
  - *data* → a string containing data to be put in a text node
  - e.g., `document.createTextNode('I have created a text node!');`

Built-in Objects

– Global Objects –

Data types

- Previously learned about three data types: string, number, and boolean
  - there are three more: objects, null, undefined
- grouped as:
  - simple/primitive data types
    - string, number, boolean, undefined, null
  - complex data type
    - objects
      - arrays and functions are considered types of objects

Data types

- undefined – important for debugging
  - the value of variables that have not been initialized
  - the value you get when you query the value of an object property or array element that does not exist
  - is the return value of functions that do not explicitly return a value
  - the value of function parameters for which no argument is passed
- null
  - indicates the absence of value
  - it indicates “no value” for numbers and strings, as well as objects

Data types

- Differences between *null* and *undefined*
  - both *null* and *undefined* indicate the absence of value and can often be used interchangeable
  - both are *falsy* values; they behave like *false* when a boolean value is required
  - neither one have properties or methods
    - using . or [] to access a property or method of these values causes a *TypeError*
  - *undefined* → unexpected or error-like absence of value
  - *null* → normal or expected absence of value

Data types

- The web browser and the current document can be modeled using objects
  - objects can have methods and properties
- Also simple values such as *string*, *number*, and *boolean* can have methods and properties
  - JavaScript treats any variable as an object in its own right
  - If a variable, or the property of an object contains a **string / number**, we can use the properties and methods of the **String / Number** object on it
  - **Boolean** object: rarely used

Data types

- **Arrays** are objects
  - a set of key/value pairs as any other object
  - the name of the key is an index number
  - they have methods and properties
    - property `length` → tells how many items are in an array
    - methods for adding items or removing items from an array

Data types

- Functions are objects
  - they have an additional feature; they are callable
    - you can tell the interpreter when you want to execute the statements that it contains

String object

- is a **global object** because it works anywhere in our script
- the properties and methods of the *String* object can be used on any string type of value → this is why is also known as a **wrapper object**
- used to work with text stored in variables or objects

String object

```
let stringDefinition = 'A string represents a text in JS';
```

- each character in a string, including white space, is given a number, called an index number
  - like the items in array
  - the numbering starts at 0 (zero)

```
A| |s|t|r|i|n|g| |r|e|p|r|e|s|e|n|t|s| |a| |t|e|x|t| |i|n| |J|S|
0|1|2|3|4|5|6|7|8|9|...
```

- this allows us to get fragments of strings or individual characters based on their index number, or the index number of a certain character or group of characters

String object

let stringDefinition = 'A string represents a text in JS';

- property
  - `length` → returns the number of characters in a string

    ```
    stringDefinition.length; // 32
    ```
- methods
  - `toUpperCase()` → changes strings to uppercase characters

    ```
    stringDefinition.toUpperCase(); //  'A STRING REPRESENTS A TEXT IN JS'
    ```
  - `toLowerCase()` → lowercase letters

    ```
    stringDefinition.toLowerCase(); // 'a string represents a text in js'
    ```

String object – methods

- `chartAt()` → takes an index umber as parameter and returns the character found at that position

```
stringDefinition.chartAt(0); // 'A'
```

- `indexOf()` → returns the number of the **first time** a character or a set of characters is found within the string

```
stringDefinition.indexOf('JS'); // 30
```

String object – methods

- `lastIndexOf()` → returns index number of the **last time** a character or set of characters is found within the string

```
const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was it really lazy?';

const searchTerm = 'dog';

console.log(`The index of the first "${searchTerm}" from the end is ${paragraph.lastIndexOf(searchTerm)}`);

// expected output: "The index of the first "dog" from the end is 52"
```

```
let stringDefinition = 'A string represents a text in JS';
```

String object – methods

- `substring()`
  - returns the characters found between two index numbers
    - the character for the first index is included
    - the character for the last index is excluded

```
stringDefinition.substring(20, 32); // 'a text in JS'
```

- `split()`
  - splits the string every time the specified character is found and
  - stores each individual part in an array

```
stringDefinition.split(' '); // ['A', 'string', 'represents',
'a', 'text', 'in', 'JS' ]
```

String object – methods

- `trim()` → removes white space from the start and end of the string

```
stringDefinition.trim(); // 'A string represents a text in JS'
```

- `replace()`
  - like search and replace in word processors
  - takes one value that is the character/s to be found
  - the second value is the replacement

```
stringDefinition.replace('represents', 'is'); // 'A string is a text in JS'
```

Exercise

- Create a variable containing a short string value; choose the string that you like.
- Apply to it the properties and methods learned on page 128 of the Duckett book
- Do this in codepen.io/pen
- Display the results to the interface by using `el.innerHTML`

Number object

- whenever we have a value that is a number we can use the methods and objects of the *Number* object on it
- example of use: for financial calculations, one has sometimes to round up the numbers to a specific number of decimal places
- Terms
  - **integer** → a whole number, e.g., 10, 1, 345, …
  - **real number** (a floating point number) → a number that contains a fractional part (decimals), e.g, 10.55, 1.6789, 0.4, …
    - the 55 is the fractional part of the 10.55
    - **floating point** → the decimal point

Number object – methods

- `isNaN()`
  - check if the value is not-a-number
  - it returns *true* if its argument is *NaN*, or if that argument is a non-numeric value that cannot be converted to a number

```
isNaN(37);    // false
isNaN('37');  // false: '37' is converted to the number 37
```

Number object – methods

- `toFixed(digits)`
  - rounds to specified number of decimal places
  - the number of digits to appear after the decimal point
    - this may be a value between 0 and 20, inclusive
    - if this argument is omitted, it is treated as 0
  - returns a string

```
let originalNumber = 10.23456;
originalNumber.toFixed(3);  // '10.234'
```

Number object – methods

- `toPrecision(precision)`
  - rounds to total number of places
  - precision should be a number between 1 and 100 (inclusive)
  - returns a string

```
let numObj = 5.123456

numObj.toPrecision()    // logs '5.123456'

numObj.toPrecision(5)   // logs '5.1235'; rounds 4 up to 5,
because the number after 4 is => 5

numObj.toPrecision(2)   // logs '5.1'

numObj.toPrecision(1)   // logs '5'
```

The Math Object

- JavaScript programs work with numbers using arithmetic operators
  - '+' for addition, '-' for subtraction, '*' for multiplication, '/' for division, '%' for modulo (reminder after division)
- used for more complex mathematical operations → a set of mathematical functions and constants defined as properties of the Math object
- Math is a **global object** → use the Math object followed by the property or method we want to access

The Math Object – methods

- `Math.round(x)`
  - x is a number
  - rounds (up or down) number to the nearest integer (whole number)
    - if the fractional portion of the argument is equal or greater than 0.5, the argument is rounded to the integer with the next higher absolute value
    - if it is less than 0.5, the argument is rounded to the integer with the lower absolute value

```
Math.round( 20.49);  //  20
Math.round( 20.5 );  //  21
```

The Math Object – methods

- `Math.floor(x)`
  - x is a number
  - rounds number **down** to the nearest integer

```
Math.floor(5.95); // expected output: 5
Math.floor(5.05); // expected output: 5
Math.floor(5); // expected output: 5
Math.floor(-5.05); // expected output: -6
```

The Math Object – methods

- `Math.ceil(x)`
  - x is a number
  - rounds number **up** to the nearest integer

```
Math.ceil(.95);    // 1
Math.ceil(4);      // 4
Math.ceil(7.004);  // 8
Math.ceil(-0.95);  // -0
Math.ceil(-4);     // -4
Math.ceil(-7.004); // -7
```

The Math Object – methods

- `Math.random()`
  - generates a number between 0 (inclusive) and 1 (not inclusive)
- to output integers and not real numbers, use `Math.floor()`
  - with *Math.round()* the first and the last number will be chosen half of the times the numbers in between

```
function getRandomInt(max) {
  return Math.floor(Math.random() * max);
}

getRandomInt(3); // expected output: 0, 1 or 2

getRandomInt(1); // expected output: 0

Math.random(); // expected output: a number from 0 to <1
```

Exercise

- Implement the exercise on the page 135 of the Duckett book in your text editor

  https://javascriptbook.com/code/c03/ (*135. Math Object to Create Random Numbers*)

  - **TO DO:** replace the value of the variable `randomNum` to be the value that is returned by the function below

```
// Getting a random integer between two values, inclusive
function getRandomIntInclusive(min, max) {
  min = Math.ceil(min);
  max = Math.floor(max);
  return Math.floor(Math.random() * (max - min + 1) + min); //The maximum is inclusive and the minimum is inclusive
}
```

The Date Object

- represents a single moment in time in a platform-independent format
- **Date()** constructor – creates a *Date* instance or return a string representing the current time
  Remember: *the Hotel() object constructor function that we defined ourselves*

```
// prints out Wed Feb 20 2023 11:00:55 GMT+0100 (Central European
Standard Time) → represents the moment the date was printed out

let today = new Date();
```

The Date Object

- By default the new instance holds today's date and the current time
  - the date and time when the JavaScript interpreter encounters that line of code
  - the current date/time is determined by the computer's clock
    - users in different time zones will get different dates/time
    - if the computer of the user holds the wrong date and time, the Date object will reflect this wrong date
- We can also specify another date and time, by passing it as argument for the new object we create

The Date Object

- The **syntax** and **order** for specifying date and time
  - Year → four digits; e.g.,1996
  - Month → a number between 0 and 11 (January is 0)
  - Day → 1-31
  - Hour → 0-23
  - Minutes → 0-59
  - Seconds → 0-59
  - Milliseconds → 0-999
- A quirk of the Date object
  - the first month of the year is number 0, but the first day of the month is number 1

The Date Object

- The date object does not store the names of days or months because they vary between languages
  - “January” is “januar” in Norwegian
  - we have a number to represent the month, i.e., 0 for January, 1 for February …, and a number to represent the day of the week, i.e., 0 for Sunday, 1 for Monday

Tid og dato

Dato

I allmennspråket og i løpende tekst generelt skal datoer skrives med rekkefølgen dag-måned-år:

den 5. juni 2014
5. juni 2014
5.6.2014
05.06.2014
5.6.14
05.06.14

Dersom årstall er utelatt, skal det være punktum etter dag og måned: 5.6.

Det skal ikke være skråstrek i datoangivelser. Ikke slik: 5/6 2014

Den tradisjonelle rekkefølgen dag-måned-år (f.eks. 17.05.2014) er i samsvar med vanlig talemål og hvordan rekkefølgen er når man skriver månedsnavnet fullt ut: 17. mai 2014.

I teknisk bruk kan datoer ha formatet år-måned-dag:

20140517
2014-05-17

Rekkefølgen *år-måned-dag* er norsk og internasjonal standard og mye brukt ved utveksling av data, i kronologiske lister m.m.
Engelskspråklige land har ulik dateringspraksis. I Storbritannia setter de dagen først og så måneden, i USA er det omvendt.

Tall, tid og dato.
Regler for skrivemåten av tidsuttrykk og talluttrykk.

https://www.sprakradet.no/sprakhjelp/Skriveregler/Dato/

The Date Object

- We can **set** the date and/or time using one of the following formats
  - YYYY, MM, DD, HH, MM, SS

```
let myDate = newDate(1996, 11, 25, 15, 45, 55); // Represents 3:45pm and 55 seconds on the December 25, 1996
```

  - MMM DD, YYYY HH:MM:SS

```
let myDate = newDate('Dec 25, 1996, 15:45:55');
```

    When you pass a string to the Date() constructor, it will attempt to parse that string as a date and time specification

  - YYYY, MM, DD (omitting the time portion)

```
let myDate = newDate(1996, 11, 25);
```

    When omitting the time fields , the Date() constructor defaults them all to 0; the time is set to midnight

The Date Object

- Once we have created a Date object, we can use its methods to both retrieve and set the date and time it represents
- To see all methods for the Date object
  - write `new Date();` to the console
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date#
- We have methods that allows us to query (get) and modify (set) the year, month, day-of-month, hour, minute, second, and milliseconds
- Each of these methods have two forms
  - one that gets/sets using local time
  - one that gets/sets using UTC time (Universal Coordinated Time)

The Date Object – methods

- `getDate() / setDate()` → returns / sets the **day** of the month (1-31)

```
let dateToday = new Date();

dateToday.getDate(); // returns 24, the day of the lecture

dateToday.setDate(27);

dateToday.getDate(); // returns now 27
```

The Date Object – methods

- `getDay()` → returns the day of the week (0-6, 0 for Sunday)

```
let Xmas95 = new Date('December 25, 1995 23:15:30');
Xmas95.getDay(); // prints 1, as it is a Monday
```

  NOTE: no corresponding *setDay()* method for setting the day-of-week; it is read-only

- `getFullYear() / setFullYear()` → returns/sets the year with four digits

```
let Xmas95 = new Date('December 25, 1995 23:15:30');
Xmas95.getFullYear();  // prints 1995
Xmas95.setFullYear(1997);
Xmas95.getFullYear();  // prints 1997
```

The Date Object – methods

- Methods that output/set the date and time in Coordinated Universal Time (UTC)
  - `Date.UTC();` → takes the same arguments as the *Date()* constructor
  - the global standard time defined by the World Time Standard
  - also known as the Greenwich Mean Time, as UTC lies along the meridian that includes London—and nearby Greenwich—in the United Kingdom
  - e.g., `getUTCFullYear() / setUTCFullYear(), getUTCHours() / setUTCHours()`
  - `getTimezoneOffset()` → the difference, in minutes, between date, as evaluated in the UTC time zone, and as evaluated in the local time zone
    - for Norway the output is “-60”; one hour difference

The Date Object

- Methods for returning date/time as human readable strings
  - `toDateString()`

    ```
    new Date().toDateString();  // prints "Wed Feb 23 2022"
    ```

  - `toTimeString()`

    ```
    new Date().toTimeString();  // "21:11:26 GMT+0100 (Central European Standard Time)"
    ```