this & Object Prototypes
DOM

“People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on each other, like a wall of mini stones.”

– Donald Knuth

Agenda

- this
- Prototypes and inheritance
- DOM
- Exercises (exercise 7 as assignment)
- Assignment

this & Object Prototypes

this

What is **this**?

The JavaScript ***this*** keyword refers to the object it belongs to.

Global context

In the global execution context (outside of any function), ***this*** refers to the global object whether in strict mode or not.

```
console.log(this === window); // true

var age = 37;
console.log(window.age); // 37

this.theme = "Lectia 3";
console.log(window.theme)
console.log(this.theme)
console.log(theme)
```

Function context

Inside a function, the value of this depends on how the function is called.

Default Binding

Implicit Binding

Explicit Binding

new binding

Default Binding

When default binding is applied, the global object will be bind to the called function

```
function car() {
   console.log(this.color)
}
var color = 'red';
car(); // 'red'
```

Implicit Binding

The object that is standing before the dot is what this keyword will be bound to.

```
function carDetails() {
   return `${this.car} ${this.color}`;
}
var dacia = {
   car: 'dacia',
   color: 'red',
   getDetails: carDetails
}
var mercedes = {
   car: 'mercedes',
   color: 'green',
   getDetails: carDetails
}

dacia.getDetails();
mercedes.getDetails();
```

Explicit Binding

In this case, you can force a function call to use a particular object for this binding

```
function car() {
   return this.color;
}

var color = `red`;
var myCar = {
   color: `blue`
}

car.apply(myCar);
car.call(myCar);
var carDetails = car.bind(myCar)
carDetails();
```

“new” Binding

When a function is invoked with the new keyword, then the function is known as a constructor function and returns a new instance.

```
function car( color ) {
   this.color = color;
}

let firstCar = new car('yellow');
let seccondCar = new car('blue');

console.log(firstCar.color) // ??
console.log(seccondCar.color) // ??
```

Priority

The highest priority has new Binding. Then explicit binding and implicit binding. The lowest priority has default binding.

```
function sayHello() {
   this.text = 'hello'
   console.log(`${this.text} - ${this.who}`)
}

let who = 'global';
let text1 = {who: 'object 1', sayHello: sayHello }
let text2 = {who: 'object 2'}

sayHello();
text1.sayHello();
obj1.sayHello.call(obj2)

let fnText2 = sayHello.bind(obj2);
fnText2(); fnText2.call(text1);

new sayHello();
new text1.sayHello()
new fnText2();
```

Arrow Function

Normal functions abide by the 4 rules we just covered. But ES6 introduces a special kind of function that does not use these rules

```
const outerThis = this;

const func = () => {
   console.log(this === outerThis);
};

new func();
func.call(null);         //true
func.apply(undefined);   //true
func.bind({})();        //true
```

Class context

Within a class constructor, this is a regular object. All non-static methods within the class are added to the prototype of this:

```
class Example {
    constructor() {
        console.log(this)
    }
    first(){}
    second(){}
    static third(){}
}
new Example();
```

Derived classes

Unlike base class constructors, derived constructors have no initial this binding. Calling super() creates a this binding within the constructor

```
class Example {
    constructor() {
        console.log(this)
    }
    first(){}
    second(){}
    static third(){}
}
class Math extends Example {
    constructor() {
        super();
        console.log(this)
    }
}
new Math();
```

Prototypes and Inheritance

Function prototype

In JavaScript, every function and object has a property named prototype by default

```
function Person (name, age) {
   this.name = name,
   this.age = age
}

let person = new Person();
console.log(Person.prototype);
```

Prototype Inheritance

In JavaScript, a prototype can be used to add properties and methods to a constructor function.

```
// constructor function
function Person () {
   this.name = 'John',
   this.age = 23
}
// creating objects
let person1 = new Person();
let person2 = new Person();
// adding property to constructor function
Person.prototype.city = 'Oradea';
// prototype value of Person
console.log(Person.prototype);
// inheriting the property from prototype
console.log(person1.city);
console.log(person2.city);
```

Prototype Changing

If a prototype value is changed, then all the new objects will have the changed property value. All the previously created objects will have the previous value.

```
// constructor function
function Person() {
    this.name = 'John'
}
// add a property
Person.prototype.age = 20;
// creating an object
let person1 = new Person();
console.log(person1.age); // 20
// changing the property value of
prototype
Person.prototype = { age: 50 }
// creating new object
let person3 = new Person();
console.log(person3.age); // 50
console.log(person1.age); // 20
```

Prototype Chaining

If an object tries to access the same property that is in the constructor function and the prototype object, the object takes the property from the constructor function.

```
function Person() {
    this.name = 'John'
}

// adding property
Person.prototype.name = 'Peter';
Person.prototype.age = 23

let person1 = new Person();

console.log(person1.name); // John
console.log(person1.age); // 23
```

Object Links

```
const animal = {
   age: 1
}

const rabbit = Object.create(animal)

rabbit.age++ // ups implicit shadowing!
rabbit.name = 'Oreo'
```

New Object Methods

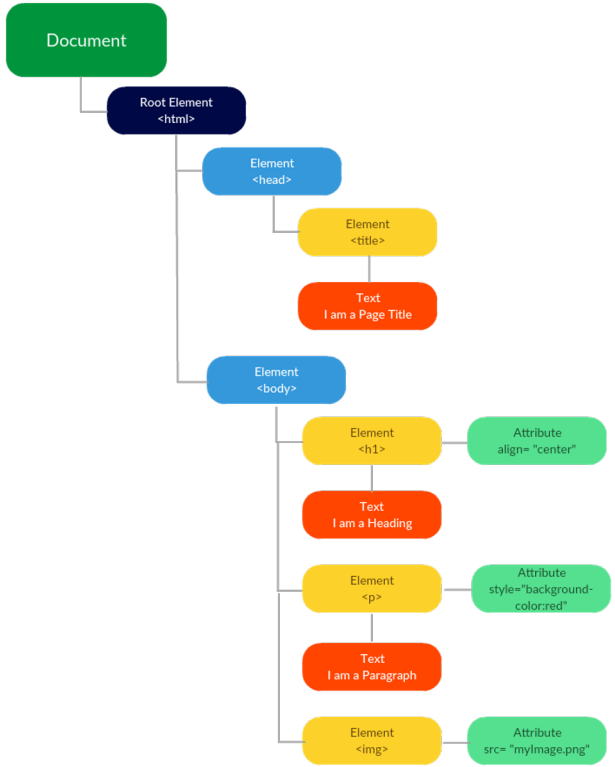
```
// Adding or changing an object property
Object.defineProperty(object, property, descriptor)
// Adding or changing many object properties
Object.defineProperties(object, descriptors)
// Accessing Properties
Object.getOwnPropertyDescriptor(object, property)
// Returns all properties as an array
Object.getOwnPropertyNames(object)
// Returns enumerable properties as an array
Object.keys(object)
// Accessing the prototype
Object.getPrototypeOf(object)
```

```
// Prevents adding properties to an object
Object.preventExtensions(object)
// Returns true if properties can be added to an
object
Object.isExtensible(object)
// Prevents changes of object properties (not values)
Object.seal(object)
// Returns true if object is sealed
Object.isSealed(object)
// Prevents any changes to an object
Object.freeze(object)
// Returns true if object is frozen
Object.isFrozen(object)
```

Javascript HTML DOM

What is the HTML DOM?

- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
- When a web page is loaded, the browser creates a Document Object Model of the page.



Finding HTML Elements

- **.querySelector(***CSS selectors***)** - returns the first element that matches one or more CSS selectors.
- **.querySelectorAll(***CSS selectors***)** - returns all elements that match the specified CSS selector.
- **.getElementById(***id***)** - object representing the element whose id property matches the specified string.

```
<script>
   // Get the first <p> element in the document:
   var firstParagraph = document.querySelector('p');

   // Get the first <p> element in the document with class='example':
   var exampleParagraph = document.querySelector('p.example');

   // Get the paragraph with the id demo
   var demoParagraph = document.querySelector('#demo')

   // Get all <p> elements in the document
   var allParagraphs = document.querySelectorAll('p');

   // Get all <p> elements in the document with class='example'
   var exampleParagraps = document.querySelectorAll('p.example');

   // Get the element with the specified ID:
   var demoParagraph = document.getElementById('demo');
</script>
```

Changing HTML Elements

- element.**innerHTML** = *new html content*
- element.style.***property*** = *new style*
- element.**setAttribute**(*attribute*, *value*)

```
<script>
   // Change the HTML content
   document.getElementById('demo').innerHTML = 'Paragraph changed!';

   // Change the color
   document.querySelector('#demo').style.color = 'red';

   // Change image src
   document.querySelector('img').setAttribute('src', '/new_path');
</script>
```

HTML DOM classList Property

- .**add**(*class1*, *class2*, ...)
- .**contains**(*class*)
- .**remove**(*class1*, *class2*, ...)
- .**toggle**(*class*)

```
<script>
   // Add a class to a <div> element:
   document.querySelector('div').classList.add('myClass');

   // Remove a class from a <div> element:
   document.querySelector('div').classList.remove('myClass');

   // if visible is set remove it, otherwise add it
   document.querySelector('div').classList.toggle('visible');
</script>
```

Adding and Deleting Elements

- document.**createElement**(*element*)
- *element*.**appendChild**(*newElement*)
- *element*.**remove**()

```
<script>
    // Create element
    var myContent = document.createElement('div');
    myContent.innerHTML = 'This is my content';

    // Append element
    document.querySelector('body').appendChild(myContent);

    // Remove element
    document.querySelector('#demo').remove();
</script>
```

Events

- element.**addEventListener(**evt, listener, [options]**);**
- element.**removeEventListener(**evt, listener, [options]**);**

```
<script>
    // Create element
    var button1 = document.querySelector('#button1');
    var button2 = document.querySelector('#button2');

    var onButton1Click = () => {
        alert('You have clicked Button 1');
    };

    var onButton2Click = () => {
        alert('You have clicked Button 2');
        button1.removeEventListener('click', onButton1Click)
    }

    button1.addEventListener( 'click', onButton1Click)
    button2.addEventListener( 'click', onButton2Click)
</script>
```

The Location Object

The location object contains information about the current URL.

```
<script>
    // Create element
    var button1 = document.querySelector('#button1');
    console.log(document.location)
    var onClick = () => {
        document.location.href = 'http://www.mozilla.org'
    }

    button1.addEventListener('click', onClick);

</script>
```

Exercises

check reference inside folder

git clone git@github.com:alexghi/fasttrack-web-course.git

Exercises - 1

1. Here is a sample html file with a submit button. Now modify the style of the paragraph text through javascript code.

Exercises - 2

2. Write a JavaScript function to get the values of First and Last name of the following form.

Exercises – 3 (assignment)

3. Write a JavaScript program to set the background color of a paragraph.

Exercises – 4 (assignment)

4. Here is a sample html file with a submit button. Write a JavaScript function to get the value of the href, hreflang, rel, target, and type attributes of the specified link

Exercises – 5 (assignment)

5. Write a JavaScript function to add rows to a table.

Exercises – 6 (assignment)

6. Write a JavaScript function that accept row, column, (to identify a particular cell) and a string to update the content of that cell.

Exercises – 7 - mandatory

5. Write a JavaScript function to add rows to a table.

Assignment

Assignment – 1

Using the knowledge you gained so far:

- Create a branch from like `alex_c8` (yourName_c8).
- In this branch you will commit your modifications
- Given the array you have in the script file, feel free to modify it's shape (**but use the pictures provided**) and:
  - Populate the DOM with all the images
  - Create a carousel from the images you have (so two buttons – one for next, one for previous)
- On one image click show some extra information about the picture (e.g. name, description)
- Also on image click show a download button which on click downloads that image

Assignment – 1

Using the knowledge you gained so far:

- Display a counter in the middle of the image down below with current image out of total (e.g. 7/11 – when you are viewing image no 7)
- When on last image on next click start from no1, when on first image on previous click go to last image (so infinite behaviour)
- Commit your changes in this branch (do as many commits as you want)
- In the end push your branch in my remote repo (if you stumble into permission problems we'll solve them)

References

References

- Array https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
- Object https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object
- DOM https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
- DOM https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (must read)
- this https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this

References

- Object Prototype https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain