Кросс-платформенная разработка

Лекция 2

Темы

- Типы данных
- Классы

Arrays

```
let arr = new Array();
let arr = [];

let fruits = ["Apple", "Orange", "Plum"];


alert( fruits[0] ); // Apple
alert( fruits[1] ); // Orange
alert( fruits[2] ); // Plum
```

Arrays 2

```
fruits[2] = 'Pear'; // now ["Apple", "Orange", "Pear"]
fruits[3] = 'Lemon'; // now ["Apple", "Orange", "Pear", "Lemon"]
```

Push/Pop

```
let fruits = ["Apple", "Orange", "Pear"];
alert( fruits.pop() ); // remove "Pear" and alert it
alert( fruits ); // Apple, Orange


let fruits = ["Apple", "Orange"];
fruits.push("Pear");
alert( fruits ); // Apple, Orange, Pear
```

Shift/Unshift

```
let fruits = ["Apple", "Orange", "Pear"];
alert( fruits.shift() ); // remove Apple and alert it
alert( fruits ); // Orange, Pear


let fruits = ["Orange", "Pear"];
fruits.unshift('Apple');
alert( fruits ); // Apple, Orange, Pear
```

Multiple Additions

```
let fruits = ["Apple"];

fruits.push("Orange", "Peach");
fruits.unshift("Pineapple", "Lemon");

// ["Pineapple", "Lemon", "Apple", "Orange", "Peach"]
alert( fruits );
```

By reference

```
let fruits = ["Banana"]

let arr = fruits;

alert( arr === fruits ); // true

arr.push("Pear"); // modify the array by reference

alert( fruits ); // Banana, Pear - 2 items now
```

Bad Practices

```
let fruits = [];

fruits[99999] = 5;

fruits.age = 25;
```

How to process elements?

```
let arr = ["Apple", "Orange", "Pear"];

for (let i = 0; i < arr.length; i++) {
    alert( arr[i] );
}
```

```
let fruits = ["Apple", "Orange", "Plum"];

// iterates over array elements
for (let fruit of fruits) {
    alert( fruit );
}
```

Multidimensional arrays

```
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

alert( matrix[1][1] ); // 5, the central element
```

splice

```
let arr = ["I", "study", "JavaScript"];
arr.splice(1, 1);
alert( arr );


let arr = ["I", "study", "JavaScript", "right", "now"];
// remove 3 first elements and replace them with another
arr.splice(0, 3, "Let's", "dance");
alert( arr ) // now ["Let's", "dance", "right", "now"]
```

splice 2

```
let arr = ["I", "study", "JavaScript", "right", "now"];

// remove 2 first elements
let removed = arr.splice(0, 2);

alert( removed );
```

Negative indexes

```
let arr = [1, 2, 5];

// from index -1 (one step from the end)
// delete 0 elements,
// then insert 3 and 4
arr.splice(-1, 0, 3, 4);

alert( arr ); // 1,2,3,4,5
```

What else?

```
let arr = ["t", "e", "s", "t"];
alert( arr.slice(1, 3) );
alert( arr.slice(-2) );

let arr = [1, 2];
alert( arr.concat([3, 4]) ); // 1,2,3,4
alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6
alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

forEach

```
arr.forEach(function(item, index, array) {
  // ... do something with item
});

["Bilbo", "Gandalf", "Nazgul"].forEach(alert);

["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
  alert(`${item} is at index ${index} in ${array}`);
});
```

Searching in array

```
// looks for item starting from index from,
// and returns the index where it was found, otherwise -1.
arr.indexOf(item, from)
// same, but looks for from right to left.
arr.lastIndexOf(item, from)
// looks for item starting from index from, returns true if found.
arr.includes(item, from)
```

Searching in array examples

```
let arr = [1, 0, false];

alert( arr.indexOf(0) ); // 1
alert( arr.indexOf(false) ); // 2
alert( arr.indexOf(null) ); // -1

alert( arr.includes(1) ); // true
```

Differencies

```
const arr = [NaN];
alert( arr.indexOf(NaN) );  ===
alert( arr.includes(NaN) );
```

Arrays of objects

```
let result = arr.find(function(item, index, array) {
    // if true is returned, item is returned and iteration is stopped
    // for falsy scenario returns undefined
});
```

find / filter

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
];

let user = users.find(item => item.id == 1);

alert(user.name); // John
```

Map

```
let result = arr.map(function(item, index, array) {
  // returns the new value instead of item
});
```

Homework

- sort
- split
- join
- reduce

- Map
- Set
- Iterable

Destructuring assignment

```
let [firstName, surname] = "John Doe".split(' ');


let options = {
   title: "Menu",
   width: 100,
   height: 200
};

let {title, width, height} = options;
```

Prototypal inheritance

```
let animal = {
  eats: true
};
let rabbit = {
  jumps: true
};

rabbit.__proto__ = animal;
```

Own Properties

```
for(let prop in rabbit) {
    let isOwn =
rabbit.hasOwnProperty(prop);

    if (isOwn) {
        alert(`Our: ${prop}`);
    } else {
        alert(`Inherited: ${prop}`);
    }
}
```

F.prototype

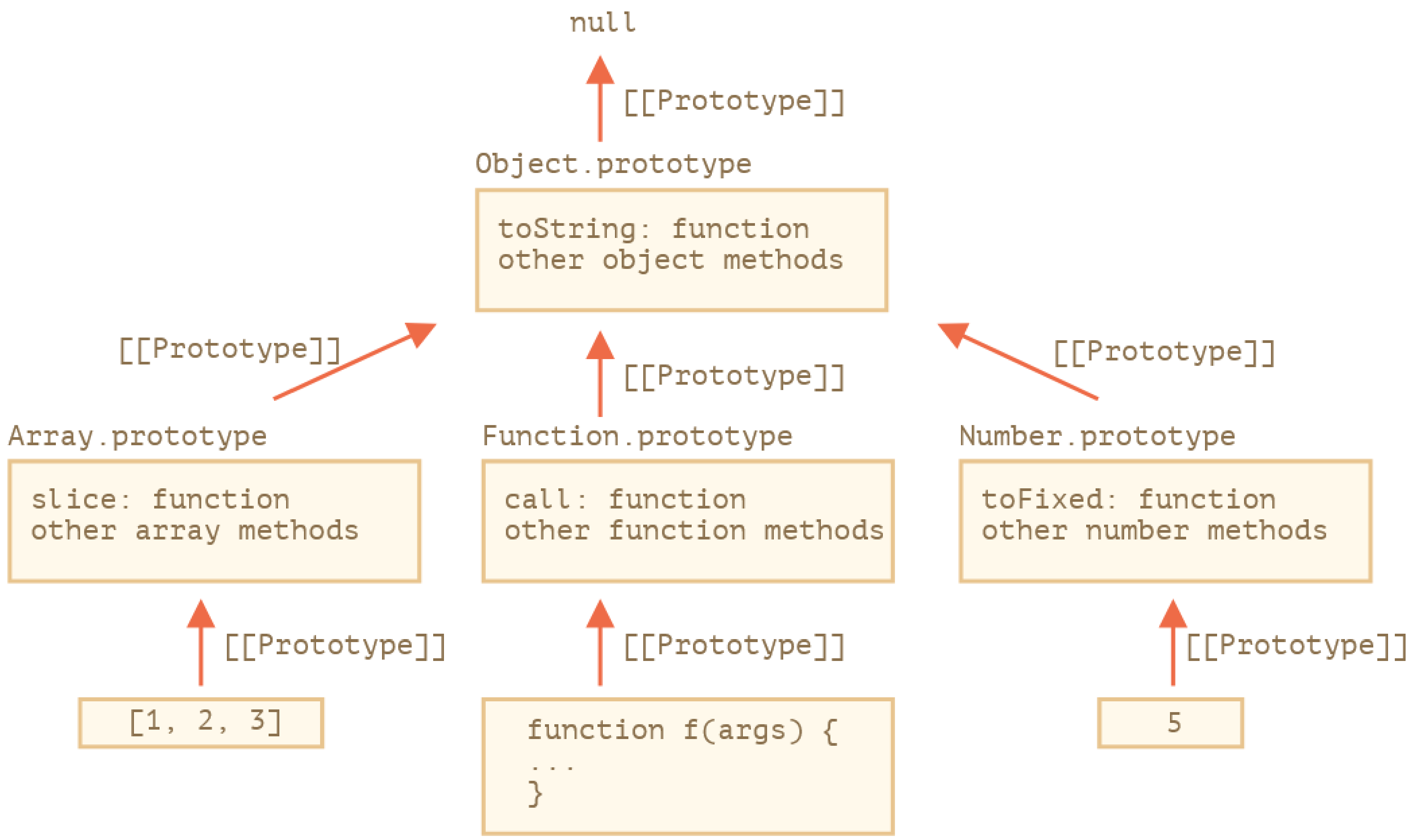
```
let animal = {
  eats: true
};
function Rabbit(name) {
  this.name = name;
}
Rabbit.prototype = animal;
let rabbit = new Rabbit("White Rabbit");
alert( rabbit.eats ); // true
```

Re-use constructor

```
function Rabbit(name) {
  this.name = name;
  alert(name);
}

let rabbit = new Rabbit("White Rabbit");

let rabbit2 = new rabbit.constructor("Black Rabbit");
```


null
[[Prototype]]
Object.prototype
toString: function
other object methods
[[Prototype]]
[[Prototype]]
[[Prototype]]
Array.prototype
slice: function
other array methods
Function.prototype
call: function
other function methods
Number.prototype
toFixed: function
other number methods
[[Prototype]]
[[Prototype]]
[[Prototype]]
[1, 2, 3]
function f(args) {
...
}
5

Classes

```
class MyClass {
  // class methods
  constructor() { ... }
  method1() { ... }
  method2() { ... }
  method3() { ... }
...
}
```

Getters/Setters

```
class User {
  constructor(name) {
    this.name = name;
  }
  get name() {
    return this._name;
  }
  set name(value) {
    if (value.length < 4) {
      alert("Name is too short.");
      return;
    }
    this._name = value;
  }
}
```

```
let user = new User("John");
alert(user.name);
user = new User("");
```

Properties

```
class User {
  name = "John";

  sayHi() {
    alert(`Hello, ${this.name}!`);
  }
}

new User().sayHi();
```

Losing this

```
class Button {
  constructor(value) {
    this.value = value;
  }
  click() {
    alert(this.value);
  }
}

let button = new Button("hello");
setTimeout(button.click, 1000); // undefined
```

Not Losing this

```
class Button {
    constructor(value) {
        this.value = value;
    }
    click = () => {
        alert(this.value);
    }
}

let button = new Button("hello");

setTimeout(button.click, 1000);
```

Super Class

```
class Animal {

  constructor(name) {
    this.speed = 0;
    this.name = name;
  }
}
class Rabbit extends Animal {

  constructor(name, earLength) {
    super(name);
    this.earLength = earLength;
  }
}
```

```
let rabbit = new Rabbit("White Rabbit", 10);
alert(rabbit.name);
alert(rabbit.earLength);
```

Overriding Fields

```
class Animal {
  name = 'animal'

  constructor() {
    alert(this.name);
  }
}

class Rabbit extends Animal {
  name = 'rabbit';
}

new Animal();
const rabbit = new Rabbit();
```

Static

```
class User {
    static staticMethod() {
        alert(this === User);
    }
}

User.staticMethod();
```

Static 2

```
class Article {
  constructor(title, date) {
    this.title = title;
    this.date = date;
  }

  static compare(articleA, articleB) {
    return articleA.date - articleB.date;
  }
}
```

```
let articles = [
  new Article("HTML", new Date(2019, 1, 1)),
  new Article("CSS", new Date(2019, 0, 1)),
  new Article("JavaScript", new Date(2019, 11, 1))
];

articles.sort(Article.compare);

alert( articles[0].title );
```

References

- https://javascript.info/
- https://developer.mozilla.org/