Язык JavaScript (ООП)

Создание объектов

```
var storage = {
        store: [],
        set: function (key, value) {
                this.store[key] = value;
        },
        get: function(key) {
                if(!key) return;
                return this.store[key];
        }
}
storage.set('name', 'Peter');
storage.get('name');
```

Как задать еще один объект хранилище ?

Добавление методов и свойств к уже созданному объекту

```
storage.clear = function() {
        this.store = [];
};
storage.type = 'digit';
storage.maxSize = 100;
```

Функция как шаблон для новых объектов

```
function Storage() {
        this.store = [];
        this.set = function (key, value) {
                this.store[key] = value;
        };
        this.get = function (key) {
                return this.store[key];
        };
}
```

С помощью ключевого слова new и функции шаблона, мы можем создать объект с одними и теми же свойствами столько раз, сколько нам необходимо:

```
var store1 = new Storage();
store1.set('name', 'Peter');
var store2 = new Storage();
```

Добавление свойств к шаблону

Так как функция является по сути объектом то мы смело можем добавить к ней свойства и методы

```
Storage.type = 'digit';
Storage.maxSize = 100;
Storege.setMaxSize = function (value) {
                this.maxSize = value;
        };
```

В данном примере эти свойства и методы не будут частью шаблона если мы выполним `new Storage();`

Создание объектов в конструкторах

```
function Storage(max) {
        this.store = [];
        this.set = function (key, value) {
                this.store[key] = value;
        };
        this.get = function (key) {
                return this.store[key];
        };

        return { maxSize: max };
}

var hub = new Storage(20); // создаем объект
hub.set('name', 'Peter'); // undefined!
hub.maxSize // 20
```

Приватные переменные, доступные только внутри нашего объекта

```
function Storage(max) {
        var store = [];
        this.set = function (key, value) {
                store[key] = value;
        };
        this.get = function (key) {
                return store[key];
        };
}

var db = new Storage();
db.store; // undefined
db.set('name', 'Peter');
```

Преобразование объектов: toString и valueOf

```
function User(firstName, lastName, age) {

        this.toString = function() {
                return firstName + ' ' + lastName;
        };
        this.valueOf = function(){
                return age;
        };
}

var user = new User("Иван", "Иванов", 18);
alert(user);
alert("Пользователь совершенно летний: " +
        (user >= 18?"Да":"Нет"));
```

Ссылочный тип

```
function User() {

        this.firstName = "Иван";
        this.lastName = "Иванов";

        this.fullName = function() {
                return this.firstName + ' ' + this.lastName;
        };
}

var user = new User();
var fullName = user.fullName;
alert(fullName()); //Что будет выведено??
```

Явное указание this

```
function User(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
};

var fullName = function() {
        return this.firstName + ' ' + this.lastName;
};

var user = new User("Иван", "Иванович");
alert(fullName.call(user));
alert(fullName.apply(user));
```

Привязка контекста

```
function User() {

        this.firstName = "Иван";
        this.lastName = "Иванов";

        this.fullName = function() {
                return this.firstName + ' ' + this.lastName;
        }.bind(this);
}

var user = new User();
var fullName = user.fullName;
alert(fullName());
```

Наследование в функциональном стиле

```
function Shape(centerX, centerY){
        this.centerX = centerX;
        this.centerY = centerY;
        this.toString = function(){
                return "Координаты центра " + this.centerX +
                        ":" + this.centerY;
        }
}

function Circle(centerX, centerY, radius){
        Shape.call(this, centerX, centerY);
        this.radius = radius;
        this.toString = function(){
                return "Координаты центра " + this.centerX +
                        ":" + this.centerY + " Радиус " + this.radius;
        }
}
```

Полиморфный конструктор

```
function Shape(centerX, centerY, radius){
        if (arguments.length == 3) {
                this.centerX = centerX;
                this.centerY = centerY;
                this.radius = radius;
                this.toString = function(){
                        return "Круг: Координаты центра " + this.centerX
+ ":" + this.centerY + " Радиус " + this.radius;
                }
        } else if (arguments.length == 2) {
                this.centerX = centerX;
                this.centerY = centerY;
                this.toString = function(){
                        return "Точка: Координаты " + this.centerX + ":" +
this.centerY;
                }
        } else {
                this.toString = function(){ return "Абстрактная фигура"; }
        }
}
```

Фабричные методы

```
function Shape(){
        this.toString = function(){
                return "Абстрактная фигура";
        }
}

Shape.point = function(data){
        var shape = new Shape();
        shape.centerX = data.centerX;
        shape.centerY = data.centerY;
        shape.toString = function(){
                return "Точка: Координаты " + this.centerX + ":" +
this.centerY;
        }
        return shape;
}
```

Задача

Создать шаблон объектов «Целочисленный итератор».

Аргументы шаблона – это левая и правая граница целочисленного диапазона.

Экземпляр объекта «Целочисленный итератор» имеет одну функцию, вызов которой возвращает новое число равное предыдущему + 1 в направлении от левой границы до правой.

Задача

Создать шаблон объектов «Целочисленный итератор второго поколения».

Наследует функционал «Целочисленный итератора».

В шаблон добавляется новая функция, которая расширяет функционал итерирования, тем что проверяет достижения итератором правого края, и в случае этого достижения меняет местами правый и левый край.

Задача

Создать шаблон объектов «Пользователь»: Свойство name (имя пользователя) и метод преобразования к строке выводит имя пользователя.
Шаблон обладает полиморфным конструктором:
- Если нет аргументов, то «Создается анонимный пользователь» в качества имя пользователя устанавливается «Аноним».
- Если в качестве первого аргумента строка, то её записать в свойство name, т.е. «Создать пользователя из данных».