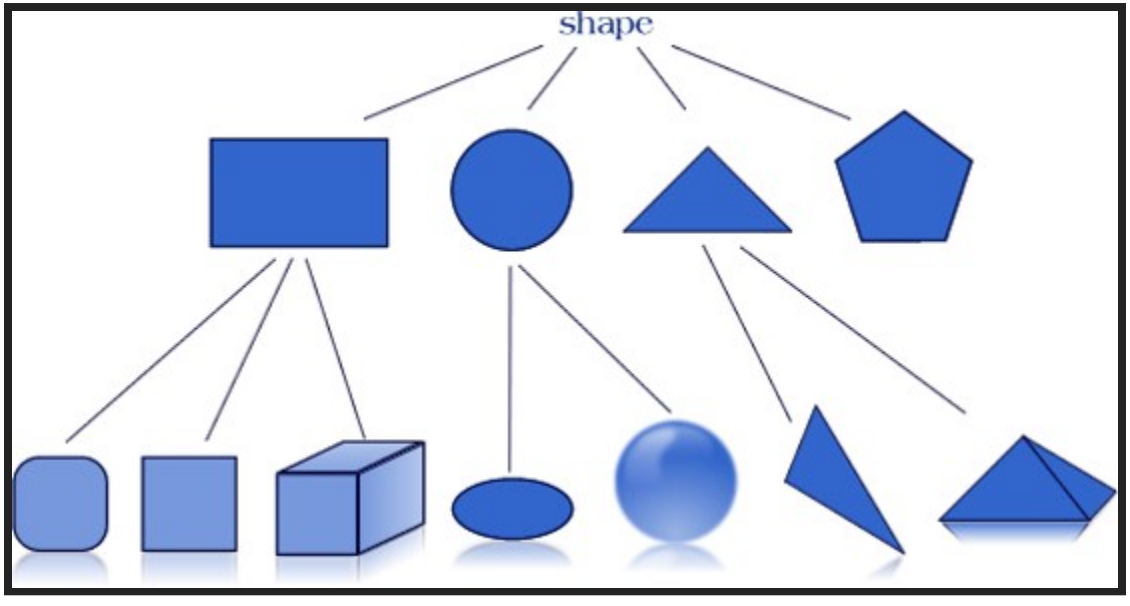
JAVASCRIPT

OOP. PROTOTYPES. INHERITANCE. CLOSURES

СЪДЪРЖАНИЕ

ООП



КАКВО Е ООП?

ООП е парадигма в компютърното програмиране

Програмата се моделира като набор от обекти

ОБЕКТИ

- Всеки обект има цел
- Обектите могат да имат свойства (property)
- Обектите могат да извършват действия (method)

ООП В JAVASCRIPT

ООП В JAVASCRIPT

- В JS няма класове и конструктори
- Обектите се създават и наследяват от обекти
- Прототипно ориентиран език

КЛАСИЧЕСКО ООП

```
function Animal(name) {
    this.name = name
}

Animal.prototype._doWalk = function() {
    alert("running")
}

Animal.prototype.walk = function() {
    this._doWalk()
}
```

КАКВО Е КЛАСИЧЕСКО ООП?

ООП, наподобяващо ООП от другите езици

КЛАСИЧЕСКО ООП

- Използват се функции за създаването на обекти
- Обектите се създават чрез извикване на функция с **new**
- Функцията се явява като конструктор на обекти

КЛАСИЧЕСКО ООП - ПРИМЕР

```
function Person() { }

var firstPerson = new Person(); // first instance of Person
```

КЛАСИЧЕСКО ООП

- Всяка инстанция е независима
- Конструктор функциите могат да приемат различен брой аргументи

КЛАСИЧЕСКО ООП - ПРИМЕР

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

var firstPerson = new Person("Adam", 33); // first instance of Person
var secondPerson = new Person("Eva", 30); // second instance of Perso
```


Demo

THIS B JAVASCRIPT

THIS

- Специален обект в JavaScript
- Стойността му се определя от начина, по който е извикана функцията

THIS В GLOBAL КОНТЕКСТ

```
console.log(this);
```

- Глобалният обект (в browser-a - window)

THIS В КОНТЕКСТА НА ФУНКЦИЯ

```
function test() {
        console.log(this);
}
test();
```

- Глобалният обект (в browser-a - window)

THIS В КОНТЕКСТА НА МЕТОД

```
var obj = {
        method: function () {
                console.log(this);
        }
}
obj.method();
```

- Обектът, от който е извикан методът

THIS В КОНТЕКСТА НА КОНСТРУКТОР

```
function Person(name) {
        this.name = name;
}
var firstPerson = new Person("Adam");
console.log(firstPerson);
```

- Обектът, който се създава

МЕТОДИ ЗА ПРОМЯНА НА THIS (CALL И APPLY)

```
function add(number3){
        console.log(this.number1 + this.number2 + number3);
}
```

```
var o = { number1: 1, number2: 2};
add.call(o, 3);
add.apply(o, [3]);
```

МЕТОДИ ЗА ПРОМЯНА НА THIS (BIND)

```
var o = { number1: 1, number2: 2};
var add = function add(){
        console.log(this.number1 + this.number2);
}.bind(o);
add();
```

PROTOTYPE

КАКВО Е PROTOTYPE В JAVASCRIPT?

- prototype е обект с properties и methods
- Всеки обект има скрито property, което държи неговия прототип (__proto__)
- Всички инстанции от даден тип споделят прототипа на типа
- Всички обекти наследяват Object

ДОБАВЯНЕ НА PROPERTY В ПРОТОТИПА

```
function Person(name) {
        this.name = name;
}

Person.prototype.introduce = function () {
  console.log("Hello, I am " + this.name);
};

var firstPerson = new Person("Adam");
firstPerson.introduce();
```


Demo

СЪЗДАВАНЕ НА PROPERTIES И METHODS

PROPERTY НА ИНСТАНЦИЯТА

```
function Person (name) {
        this.name = name;
}

var firstPerson = new Person("Adam");
var secondPerson = new Person("Eva");
console.log(firstPerson.name);
console.log(secondPerson.name);
```

PROPERTY НА КЛАСА (Т.НАР СТАТИЧНИ)

```
Math.PI
```

METHOD НА ИНСТАНЦИЯТА

```
function Person (name) {
        this.name = name;
        this.introduce = function () {
                console.log("My name is " + this.name);
        }
}

var firstPerson = new Person("Adam");
firstPerson.introduce();
```

ПО-ДОБРИЯТ НАЧИН ЗА ДЕФИНИРАНЕ НА METHOD

```
function Person (name) {
        this.name = name;
}

Person.prototype.introduce = function () {
        console.log("My name is " + this.name);
};

var firstPerson = new Person("Adam");
firstPerson.introduce();
```

НАСЛЕДЯВАНЕ В КЛАСИЧЕСКОТО ООП И PROTOTYPE CHAIN

НАСЛЕДЯВАНЕ В КЛАСИЧЕСКОТО ООП

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

Person.prototype.introduce = function () {
        console.log("My name is " + this.name + " and I am " + this.a
};
```

НАСЛЕДЯВАНЕ В КЛАСИЧЕСКОТО ООП

```
function Student(name, age, course) {
        Person.apply(this, arguments);
        this.course = course;
}

Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

Student.prototype.getCourse = function () {
        console.log(this.course);
}
```

PROTOTYPE CHAIN

- Обектите в JavaScript могат да имат един prototype
- Прототипът също има prototype, който също има prototype...
- Това се нарича prototype chain

КАК РАБОТИ PROTOTYPE CHAIN?

Когато изивкваме property на обект

1. Търси се в обекта за това property
2. Ако го няма в обекта, се търси в неговия prototype, ако го няма там, в неговия prototype...
3. Ако се стигне до края на prototype chain, се връща undefined

PROTOTYPE CHAIN - DEMO

```
function Person(name) {
        this.name = name;
}

Person.prototype.age = 25;

var firstPerson = new Person("Adam");

console.log(firstPerson.name);
console.log(firstPerson.age);
console.log(firstPerson.valueOf());
console.log(firstPerson.notExistingProperty);
```

ИЗВИКВАНЕ НА РОДИТЕЛСКИ МЕТОД

```
function Person() {}

Person.prototype.introduce = function () {
        console.log("Introduce from Person introduce method");
};

function Student() {};
Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;
```

```
Student.prototype.introduce = function () {
        Person.prototype.introduce.call(this);
        console.log("Introduce from Student introduce method");
}
```

ПРОТОТИПНО НАСЛЕДЯВАНЕ

JavaScript е динамичен език и всяка задача може да бъде реализирана по няколко начина

ПРОТОТИПНО НАСЛЕДЯВАНЕ - DEMO

```
var person = {
        introduce: function () {
                console.log("My name is " + this.name);
        }
}

var firstPerson = Object.create(person);
firstPerson.name = "Adam";

firstPerson.introduce();
```

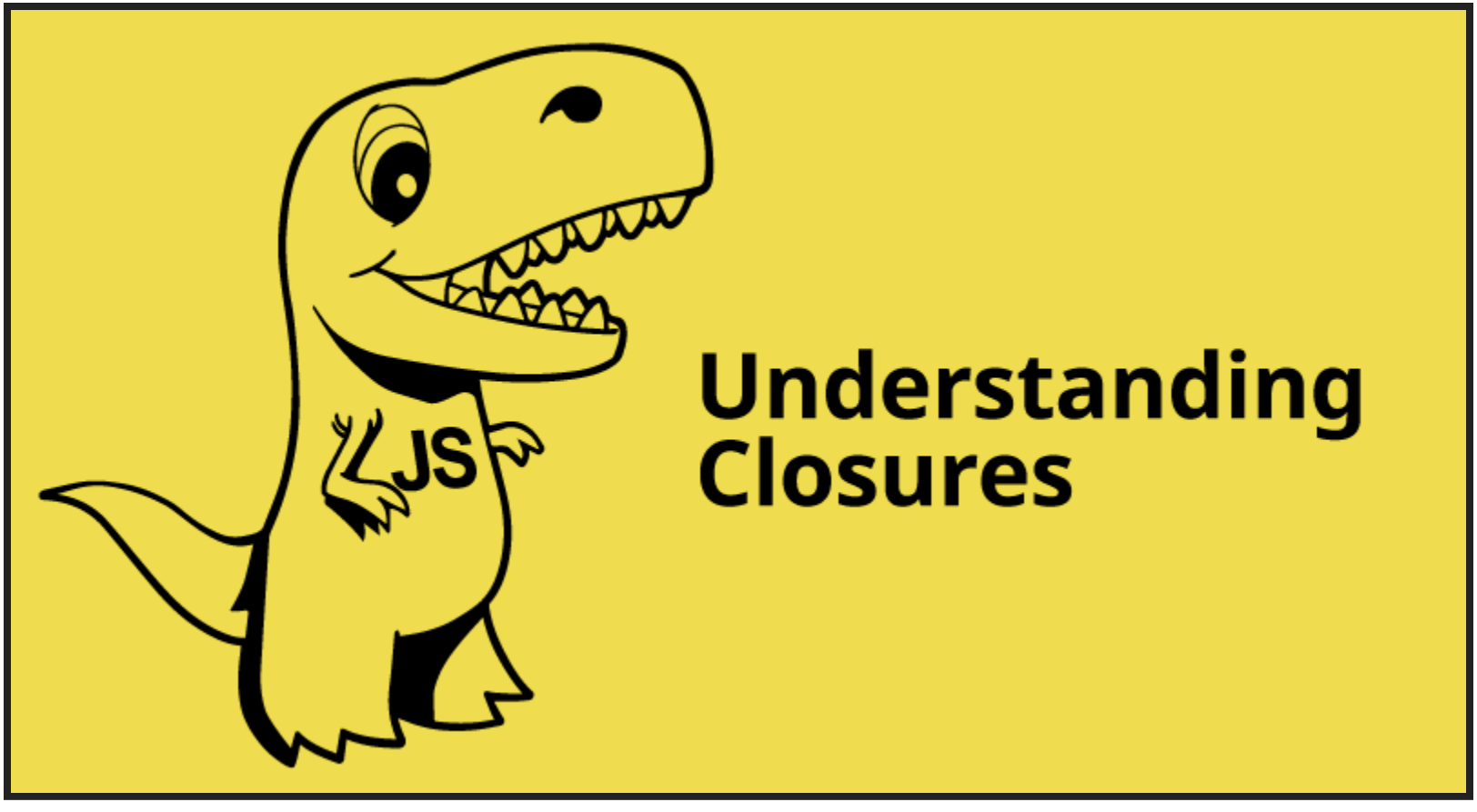
ПРОТОТИПНО НАСЛЕДЯВАНЕ С INIT - DEMO

```
var person = {
        init: function (name) {
                this.name = name;

                return this;
        },
        introduce: function () {
                console.log("My name is " + this.name);
        }
}

var firstPerson = Object.create(person).init("Adam");
firstPerson.introduce();
```

CLOSURES



КРАЧКА НАЗАД - КАКВО Е SCOPE?

- Място, в което променливите са декларирани и могат да бъдат използвани
- В JavaScript има два вида scope - global и function

ПОНЯТИЯ, СВЪРЗАНИ СЪС SCOPE

- Hoisting
- Scope chain

КАКВО Е CLOSURE?

- Специален вид структура
- Запазва функция и контекста, в който е дефинирана функцията

```
function outer() {
        var x = 5;

        return function inner(y) {
                console.log(x + y);
        }
}

var newFunc = outer();
newFunc(4);
```

ЗА КАКВО МОГАТ ДА БЪДАТ ИЗПОЛЗВАНИ CLOSURES?

- Криене на информация (private променливи и методи)
- И (всичко)/др.

ПОЛЕЗНИ ВРЪЗКИ:

The Two Pillars of JavaScript Object-oriented Programming - Eloquent JavaScript

VAR THANK = "YOU!";

```
var tutors = [{
    name: "Адриан Бобев",
    email: "adrian.bobev@sap.com"
}];
```