**CONST**

const abc = 12;

abc = 10; //error

для значимых типов данных не позволяет изменять значение переменной;

для ссылочных (массив, объект) позволяет изменить значение, но не сам массив/объект;

const arr = [2, 5, 43];

arr[1] = 10; [2, 10, 43];

arr = [9]; // error

так как в arr лежит не массив, а ссылка на него, то как раз-таки ссылку изменять нельзя;

**Система типов javascript**

1. Number - дробные и целые числа. Специальные числовые значения Infinity (при делении на 0) и NaN
2. String
3. Boolean
4. Null – отсутствие значения
5. Undefined - если переменная не инициализирована. В явном виде обычно не присваивают.
6. Object –объекты, массивы, коллекции (map), даты и функция (typeof==function)
7. В ES5+ cymbol

Оператор typeof ( x)

**DOM - Document Object Model**

Это представление документа в виде дерева объектов. Каждый HTML тег образует узел дерева с типом «элемент» Вложенные в него теги становятся дочерними элементами. А для текста создаются узлы с типом «текст». Комментарии тоже отображаются в дереве объектов. DOM нужен для манипулирования страницей – читать информацию из html, создавать и изменять элементы.

**Контекст выполнения** – окружение, в котором выполняется код.

Глобальный контекст выполнения. Код, находящийся вне функции, принадлежит глобальному контексту. У глоб.конт. есть глобальный объект, в браузере это window и ключевое слово this указывает на window.

Контекст выполнения функции. Каждый раз, когда вызывается функция, для нее создается контекст. Каждая функция имеет свой контекст.

**Замыкание** – это внутренняя функция, которая запоминает окружение, в котором она была создана.

Это механизм, который запоминает функцию и лексическое окружение, в котором она была создана.

**Лексическое окружение**

Все аргументы, переменные и функции внутри функции – это свойства внутреннего объекта LexicalEnvironment.

При вызове функции создается объект LexicalEnvironment и заполняется. Аргументы функции попадают в этот объект сразу же, а переменные изначально == undefined. По ходу выполнения функции идет присвоение переменным значений.

function sayHi(name) {

// LexicalEnvironment = { name: 'Вася', phrase: undefined }

var phrase = "Привет, " + name;

// LexicalEnvironment = { name: 'Вася', phrase: 'Привет, Вася'}

alert( phrase );

}

sayHi('Вася');

**Контекст – this** – ссылка на объект, из которого вызывается метод. Определяется во время вызова.

Внешнюю функцию нужно присвоить переменной и вызывать эту переменную.

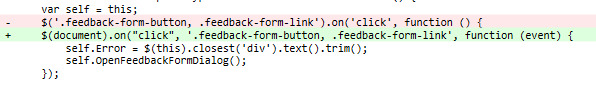
function createCounter() {  
   var numberOfCalls = 0;  
   return function() {  
      return ++numberOfCalls;  
   }  
}  
var fn = createCounter();  
fn(); //1  
fn(); //2  
fn(); //3

Замыкания инкапсулируют (защищают) данные. Замыкание нужно, если внутренняя функция вешает событие, поскольку событие возникает после того, как отработала функция.

**Делегирование события**

directly events vs delegated events

Навешивание обработчика события для динамически создающихся элементов.



прототипы

Event loop

async/defer атрибуты

классы

Promises