**Основы Java script**

JavaScript является нетипизированным языком

Типизированный/нетипизированный определяется тем ,если язык автоматически смешивает в выражениях различные типы и выполняет автоматические неявные преобразования

*Статическая* / *динамическая* типизация. Статическая определяется тем, что конечные типы переменных и функций устанавливаются на этапе компиляции. Т.е. уже компилятор на 100% уверен, какой тип где находится. В динамической типизации все типы выясняются уже во время выполнения программы.

*Явная* / *неявная* типизация. Явно-типизированные языки отличаются тем, что тип новых переменных / функций / их аргументов нужно задавать явно. Соответственно языки с неявной типизацией перекладывают эту задачу на компилятор / интерпретатор.

Тип переменной может изменяться в процессе совершения операций с данными (**динамическое приведение типов**) - var message = 10 + " дней до отпуска";

Идентификатор = переменная

Можна использовать js в виде - внутренние , в гиперссылке , обработчик событий.

**Обработчик событий** в html form:

<script>

var colorArray = ["#5A9C6E", "#A8BF5A", "#FAC46E", "#FAD5BB", "#F2FEFF"]; // создаем массив с цветами фона

var i = 0;

function changeColor(){

document.body.style.background = colorArray[i];

i++;

if( i > colorArray.length - 1){

i = 0;

}

}

</script>

<button onclick="changeColor();">Change background</button>

**Типы данных и переменные :**

Экспоненциальная запись чисел

**Литералы** в JavaScript представляют собой особый класс типа данных, фиксированные значения одного из трех типов данных — строкового, числового или логического

alert("Hellow"); // "Hellow" - это литерал

var myVariable = 15; // 15 - это литерал

**Примитивный тип данных** является экземпляром определенного типа данных, таких как строковый, числовой, логический, null и undefined.

#### Глобальные и локальные переменные

Переменные по области видимости делятся на **глобальные** и **локальные**.

Со своей областью видимости.

Переменные, объявленные внутри тела функции, называются **локальными**, их можно использовать только в этой функции.

Переменные, объявленные внутри элемента <script>, или внутри функции, но без использования ключевого слова var, называются **глобальными**.

**Функция - анонимная, именованная**

Функции начинают с глагола

Замыкание функции - функция вместе со внешними переменными которые ей доступны

когда проверяется значение переменной сначала внутри , а потом уже во внешних

**Глобальные** можно изменить и в теле разных функций, снаружи

**Локальные** переменные функции доступные только локально , кроме

f retVar(){let num = 50;return num;} let anotherNum = retVar();

Функция не будет работать(alert , console.log) если ее не вызвать

можно передавать по ссылке в **качестве аргумента другой функции**, например

function sum(x,y) {

return x+y;

}

function factor(f,i) { // factor-функция множитель, умножает

return i\*f(2,1); //9 результат функции f

}

factor(sum,3); // в качестве функции f используем функцию sum

**Выражения Java script**

**Выражения в JavaScript представляют собой комбинацию операндов и операторов.**

**Для начала приведу немного теории.**

На самом деле, в JavaScript логические операторы || и && работают особым образом.

Оператор || возвращает *первый из операндов*, значение которого может быть приведено к логическому true. Если же оба операнда приводятся к логическому false, то оператор || вернет *последнее значение*.

console.log('foo' || false); // 'foo'

console.log(null || 'bar'); // 'bar'

console.log(false || 1); // 1

console.log(false || null); // null

Это позволяет использовать всеми любимый широко известный хак со значением переменной по умолчанию:

function f(arg) {

var a = arg || 0;

// ...

}

Оператор && возвращает *первый из операндов*, значение которого приводится к логическому false. Если же оба операнда приводятся к true, то оператор && вернет *последний операнд*.

console.log(true && []); // []

console.log(null && 'foo'); // null

console.log(0 && 'foo'); // 0

console.log('foo' && 'bar'); // 'bar'

alert( [1,2,3] == [1,2,3] );

...it does three things:

1. Place an array ([1,2,3]) onto the heap
2. Place another array ([1,2,3]) onto the heap (notice it will have a different memory location)
3. Compare the two references. They point to different objects in different locations in memory, thus it is considered not equal.

var a = [], b = [];

console.log(!4);

console.log(typeof(a));

console.log(Object == true);

 console.log((a == b) + ` ` + (a == !b) + ` ` + (!a == b)+ ` ` + (!a == !b));

**false , true , true , true**

**1.Арифметические операторы**

**2. Операторы присваивания(комбинированный оператор позволяет сохранить первоначальный и последующее значение переменной)**

x += 10; // равнозначно x = x + 10;

x -= 10; // равнозначно x = x - 10;

x \*= 10; // равнозначно x = x \* 10;

x /= 10; // равнозначно x = x / 10;

x %= 10; // равнозначно x = x % 10;

### 3. Операторы инкремента и декремента

Операции **инкремента и декремента** являются унарными и производят увеличение и уменьшение значения операнда на единицу.

Префиксная форма возвращает измененное , а постфиксная старое

**i++ i— = постфиксный инкремент и постфиксный декремент**

**--i ++i =префиксный декремент и префиксный инкремент**

var x = y = m = n = 5, z, s, k, l;

z = ++x \* 2; /\* в результате вычислений вернет значение z = 12, x = 6, т.е. значение x сначала увеличивается на 1, а после выполняется операция умножения \*/

s = y++ \* 2; /\* в результате вычислений вернет значение s = 10, y = 6, т.е. сначала выполняется операция умножения, а после в переменной y сохраняется увеличенное на 1 значение \*/

k = --m \* 2; // вернет значение k = 8, m = 4

l = n-- \* 2; // вернет значение l = 10, n = 4

**4.Операторы сравнения**

5 === "5"; // вернет false

false === false; // вернет true

1 !== true; // вернет true

1 != true; // вернет false, так как true преобразуется в 1

|  |  |  |
| --- | --- | --- |
| === Идентичность | Строгая проверка по типам данных | 9 |
| = | присваивание |  |
| == Равенство | Проверка на равенство по значению | |

5.Логические операторы

|  |  |
| --- | --- |
| ! Логическое НЕ | Изменяет значение оператора на обратное - с true на false и наоборот. |

(2 < 3) && (3===3); // вернет true, так как выражения в обеих скобках дают true

(x < 10 && x > 0); // вернет true, если значение x принадлежит промежутку от 0 до 10

!false; // вернет true

### 6. Побитовые операторы

Побитовые операции могут использоваться, например, при шифровании данных, для работы с флагами, разграничения прав доступа.

### 7. Строковые операторы

|  |  |  |
| --- | --- | --- |
| + Конкатенация | Оператор работает слева направо, выполняя объединение строк. Если первый операнд является строкой, последующие операнды будут преобразованы в строки и далее выполнится их объединение. | 12 |
| += Конкатенация с присваиванием | Выполняется объединение двух строк и результат присваивается переменной. | 12 |

### 8. Специальные операторы

|  |  |  |  |
| --- | --- | --- | --- |
| . Обращение к свойству | | Осуществляет доступ к свойству объекта. | |
| instanceof Проверка типа объекта | Оператор проверяет, является ли объект экземпляром определенного класса. Левый операнд должен быть объектом, правый - должен содержать имя класса объектов. Результат будет true, если объект, указанный слева, представляет собой экземпляр класса, указанного справа, в противном случае - false. | | 10 |
| in Проверка наличия свойства | В качестве левого операнда должна быть строка, а правым - массив или объект. Если левое значение является свойством объекта, вернется результат true. | | 10 |
| new Создание объекта | Оператор создает новый объект с неопределенными свойствами, затем вызывает функцию-конструктор для его инициализации (передачи параметров). Также может применяться для создания массива. | | 1 |
| delete Удаление | Оператор позволяет удалять свойство из объекта или элемент из массива. Возвращает true, если удаление прошло успешно, в противном случае false. При удалении элемента массива его длина не меняется. | | 14 |
| void Определение выражения без возвращаемого значения | Унарный оператор, отбрасывает значение операнда и возвращает underfined. | | 14 |
| ?: Операция условного выражения | Тернарный оператор, позволяет организовать простое ветвление. В выражении участвуют три операнда, первый должен быть логическим значением или преобразовываться в него, а второй и третий - любыми значениями. Если первый операнд равен true, то условное выражение примет значение второго операнда; если false - то третьего. | |  |

typeof {a:1}; // вернет "object"

var d = new Date(); // создаем новый объект с помощью конструктора Date()

d instanceof Date; // вернет true

var mycar = {make: "Honda", model: "Accord", year: 2005};

"make" in mycar; // вернет true

var obj = new Object(); // создает пустой объект

var food = ["milk", "bread", "meat", "olive oil", "cheese"];

delete food[3]; // удаляет четвертый элемент из массива food

x > 10 ? x \* 2 : x / 2; // возвращает значение x \* 2, если x > 10, в противном

### 7.  Массивы

Есть два способа как задать массив, пустой массив или со значениями

**Пустой**

var a = new Array(); // в качестве аргумента можно задать длину массива

// а можно так

var a = []

**Со значениями**

var a = new Array("a", 1, true)

// а можно так

var a = ["a", 1, true];

Чтобы добавить новый элемент (**цифру 7**)  в массив

a.push(7);

// или так

a[a.length]=7;

### 8. Объекты

**В JavaScript в отличие от С++ нет понятие классов(к**лассы JavaScript, представленные в ECMAScript 2015)**, объекты тут создаются не как экземпляры некого класса, а с нуля, с последующим указанием свойств и методов или с помощью функции-конструктора**

**Все свойства объектов — public**

var myObject= {

heroname : "Ember",

herosurname: "Spirit",

print:function(str){

if(str=="heroname")

console.log(this.heroname);

if(str=="herosurname")

console.log(this.herosurname);

}

};

console.log(myObject.heroname);

myObject.print("herosurname");

**Есть второй вариант** как создать объект, но в этом случае **свойства и методы добавляются отдельно**

## Using the JavaScript Keyword new

var myOb = new Object();

myOb.name = "Vladik"; // добавляем свойство name

myOb.family = "Ivanov"; // добавляем свойство family

myOb.print=function(str) { //добавляем метод print(str)

if (str=="name")

alert(this.name);

if (str=="family")

alert(this.family);

};

## let дает вам привилегию объявлять переменные, ограниченные по объему блоком({}),в отличие от var(global);

## The **this** Keyword

**In JavaScript**, the thing called this is the object that "owns" the code.

The value of this, when used **in an object**, is the object itself.

**In a constructor function** this does not have a value. It is a substitute for the new object. The value of this will become the new object when a new object is created.

Note that this is not a variable. It is a keyword. You cannot change the value of this.

// Constructor function for Person objects

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

}

// Create 2 Person objects

var myFather = new Person("John", "Doe", 50, "blue");

var myMother = new Person("Sally", "Rally", 48, "green");