1. В html код JS должен быть межжду тегами: <script> </script>. Он может размещаться в <head> или в <body>. Скрипт размещенный в <head> будет выполнен до отображения <body>.
2. В JS можно использовать html теги для форматирования текста.
3. Скрипты могут быть размещены во внешний файлах. Можно размещать в <head> или в <body>

<head>

<script src=”demo.js”></script>

</head>

1. //однострочный комментарий

/\* многострочный комментарий\*/

# ПЕРЕМЕННЫЕ

1. Переменные описываются с помощью let (или var – устаревшее)
2. Если нужно написать апостроф в строке используем обратный слэш \

let sayHello = ‘Hello! I\’m a boy’;

1. В булевом типе два значения: true или false
2. Деление по модулю возвращает остаток от деления:

25 % 5 = 0

26 % 6 = 2

1. Инкремент ++ увеличивает на 1

Декремент -- уменьшает на 1

|  |  |  |  |
| --- | --- | --- | --- |
| var++ | Постинкремент | var a=0, b=10;  var a=b++; | a=10  b=11 |
| ++var | Преинкремент | var a=0, b=10;  var a=++b; | a=11  b=11 |
| var-- | Постдекремент | var a=0, b=10;  var a=b--; | a=10  b=9 |
| --var | Предекремент | var a=0, b=10;  var a=--b; | a=9  b=9 |

1. Математические опериции имеют сокращенный вид:

x -= y += 9

|  |  |
| --- | --- |
| x = y | x = y |
| x += y | x = x + y |
| x -= y | x = x - y |
| x \*= y | x = x \* y |
| x /= y | x = x / y |
| x %= y | x = x % y |

7) Операторы сравнения возвращают true или false

|  |  |  |
| --- | --- | --- |
| == | равно | 5 == 10 ложь |
| === | идентично (равно и одинакового типа) | 5 === 10 ложь |
| != | не равно | 5 != 10 истина |
| !== | не идентично | 10 !== 10 ложь |
| > | больше чем | 5 > 10 ложь |
| >= | больше или равно |  |
| < |  |  |
| <= |  |  |

1. Логические (булевые) операторы:

AND (И) && возвращает true если обе операнды истинны

OR (ИЛИ) || возвращает true если одна из операнд истинна

NOT(НЕ) ! возвращает true если операнд ложный; возвращает false если

операнд истинный

(4>2)&&(10<15) – true

1. Условный (тернарный) оператор

присваевает значение при выполнении условия

variable = (condition) ? value1 : value2

let N = (age > 18) ? ‘Too young’ : ‘Too old’;

1. Конкатерация: построение строк, путем соединения других строк

let n = ‘Nadd’;

let m = ‘hedd’;

document.write(n + m);

# УСЛОВИЯ И ЦИКЛЫ

## if else

1. if (condition) {

statements

}

1. let n=7, m=8;

if n>m {

alert (‘it is ok’)

}

else{

alert (‘it is wrong’)

}

1. let c=1;

if (c == 1) {

alert (‘1’);

} else if (c == 2){

alert (‘2’);

} else {

alert (‘wrong’);

}

## switch

let day = 2;

switch (day) {

case 1:

document.write (“monday”);

break;

case 2:

document.write (“tuesday”);

break;

default:

document.write (“another day”);

}

## Цикл for:

for (i=1; i<=5; i++) {

do document.write (i + “<br/>”);

}

где i=1 – устанавливает переменную до запуска цикла (let i=1)

i<=5 – определяет условие запуска цикла

i++ -- увеличивает значение счетчика после каждого выполнения кода в цикле

Можно так:

for (i=1, text=” “; i<=5; i++){

text = I;

do document.write (i + “<br/>”);

}

## while

let i=0;

while (I <=10) {

do document.write (i + “<br/>”);

i++;

}

## do while

let I = 20;

do {

do document.write (i + “<br/>”);

i++;

}

while (i <= 25);

## break

выпрыгивает из цикла

for (i = 0; i <= 10; i++){

if (I == 5){

break;

}

do document.write (i + “<br/>”);

}

## continue

останавливает одну итерацию и продолжает со следующей

for (i = 0; i <= 10; i++){

if (I == 5){

continue;

}

do document.write (i + “<br/>”);

}

# ФУНКЦИИ

//чтобы выполнить:

function MyFunction (){

alert ('alert box');

}

//чтобы вызвать:

MyFunction ();

MyFunction ();

в скобках параметры функций. Их может быть несколько:

function sayHello (name){

alert ('Hi,' + name);

}

sayHello ('David');

sayHello ('Sarah');

sayHello ('Masha');

## return

возвращение значения функции

function MyFunc (a, b) {

return a\*b;

}

let x = MyFunc(5,6);

alert – окно предупреждения («Добро пожаловать»)

prompt – введите ваше имя

confirm – окно подтверждения («Вы действительно хотите покинуть страницу?»)

# ОБЬЕКТЫ

Объекты могут хранить много значений. Литеральный синтаксис:

let user = {

name: 'John',

age: 30

};

где person – имя обьекта

name – ключ, имя, идентификатор

в {} – свойства объекта

'John' – значение свойства

То есть, первое свойство с именем "name" и значением "John".

//Доступ к свойствам объекта:

let x = person.age;

let y = person['age'];

Или:

// получаем свойства объекта:

alert( user.name ); // John

alert( user.age ); // 30

Чтобы добавить еще одно свойство:

user.isAdmin = true;

Чтобы удалить лишнее свойство:

delete user.age;

Имя свойства может состоять из нескольких слов, но тогда оно должно быть заключено в кавычки:

let user = {};

// присваивание значения свойству

user["likes birds"] = true;

// получение значения свойства

alert(user["likes birds"]); // true

// удаление свойства

delete user["likes birds"];

Можно так: (запись через точку этого не позволяет (user.key))

let user = {

name: "John",

age: 30

};

let key = prompt("Что вы хотите узнать о пользователе?", "name");

// доступ к свойству через переменную

alert( user[key] ); // John (если ввели "name")

let fruit = prompt("Какой фрукт купить?", "apple");

let bag = {

[fruit]: 5, // имя свойства будет взято из переменной fruit

};

alert( bag.apple ); // 5, если fruit="apple"

Смысл вычисляемого свойства прост: запись [fruit] означает, что имя свойства необходимо взять из переменной fruit.

Конкатерация:

let fruit = 'apple';

let bag = {

[fruit + 'Computers']: 5 // bag.appleComputers = 5

};

function makeUser(name, age) {

return {

name: name,

age: age

// ...другие свойства

};

}

let user = makeUser("John", 30);

alert(user.name); // John

В примере выше название свойств name и age совпадают с названиями переменных, которые мы подставляем в качестве значений этих свойств.

Можно записать так:

function makeUser(name, age) {

return {

name, // то же самое, что и name: name

age // то же самое, что и age: age

// ...

};

}

## Проверка существования свойства:

1 способ:

let user = {};

alert( user.noSuchProperty === undefined ); // true означает "свойства нет"

2 способ:

let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age существует

alert( "blabla" in user ); // false, user.blabla не существует

3 способ:

Для перебора всех свойств объекта используется цикл for..in.

for (key in object) {

// тело цикла выполняется для каждого свойства объекта

}

let user = {

name: "John",

age: 30,

isAdmin: true

};

for (let key in user) {

// ключи

alert( key ); // name, age, isAdmin

// значения ключей

alert( user[key] ); // John, 30, true

}

При этом свойства с целочисленными ключами сортируются по возрастанию. остальные располагаются в порядке создания.

let codes = {

"49": "Германия",

"41": "Швейцария",

"44": "Великобритания",

// ..,

"1": "США"

};

for (let code in codes) {

alert(code); // 1, 41, 44, 49

}

Не целочисленные располагаются в порядке создания.

let codes = {

"+49": "Германия",

"+41": "Швейцария",

"+44": "Великобритания",

// ..,

"+1": "США"

};

for (let code in codes) {

alert( +code ); // 49, 41, 44, 1

}

## Копирование по ссылке:

Примитивные типы: строки, числа, логические значения – присваиваются и копируются «по значению».

let message = "Hello!";

let phrase = message;

Обьекты хранятся и копируются «по ссылке».

**Переменная хранит не сам объект, а его «адрес в памяти», другими словами «ссылку» на него.**

**Когда переменная объекта копируется – копируется ссылка, сам же объект не дублируется.**

let user = { name: 'John' };

let admin = user;

admin.name = 'Pete'; // изменено по ссылке из переменной "admin"

alert(user.name); // 'Pete', изменения видны по ссылке из переменной "user"

Операторы равенства == и строгого равенства === для объектов работают одинаково.

**Два объекта равны только в том случае, если это один и тот же объект.**

let a = {};

let b = a; // копирование по ссылке

alert( a == b ); // true, обе переменные ссылаются на один и тот же объект

alert( a === b ); // true

В примере ниже два разных объекта не равны, хотя и оба пусты:

let a = {};

let b = {}; // два независимых объекта

alert( a == b ); // false

## Клонирование и объединение объектов

При обычном копировании, создается новая ссылка на тот же обьект. Но если нужно создать клон обьекта:

let user = {

name: "John",

age: 30

};

let clone = {}; // новый пустой объект

// скопируем все свойства user в него

for (let key in user) {

clone[key] = user[key];

}

// теперь в переменной clone находится абсолютно независимый клон объекта.

clone.name = "Pete"; // изменим в нём данные

alert( user.name ); // в оригинальном объекте значение свойства `name` осталось прежним – John.

2 способ:

Object.assign(dest, [src1, src2, src3...])

* Аргументы dest, и src1, ..., srcN (может быть столько, сколько нужно) являются объектами.
* Метод копирует свойства всех объектов src1, ..., srcN в объект dest. То есть, свойства всех перечисленных объектов, начиная со второго, копируются в первый объект. После копирования метод возвращает объект dest.

let user = { name: "John" };

let perm1 = { canV: true };

let perm2 = { canE: true };

// копируем все свойства из permissions1 и permissions2 в user

Object.assign(user, perm1, perm2);

// now user = { name: "John", canV: true, canE: true }

Или так:

let user = {

name: "John",

age: 30

};

let clone = Object.assign({}, user);

Все свойства объекта user будут скопированы в пустой объект, и ссылка на этот объект будет в переменной clone. На самом деле, такое клонирование работает так же, как и через цикл, но короче.