



IT-ACADEMY



JavaScript if else and else if

Условные операторы

- Условные операторы используются для выполнения разных действий в зависимости от разных условий.
- Условные операторы
- Очень часто, когда вы пишете код, вы хотите выполнять разные действия для разных решений.
- Для этого вы можете использовать в своем коде условные операторы.

В JavaScript есть следующие условные операторы:

- Используйте `if`, чтобы указать блок кода, который будет выполняться, если указанное условие истинно.
- Используйте `else`, чтобы указать блок кода, который нужно выполнить, если то же условие ложно.
- Используйте `else if`, чтобы указать новое условие для проверки, если первое условие ложно.
- Используйте переключатель, чтобы указать множество альтернативных блоков кода для выполнения.

Оператор switch описан в следующей главе.

- Оператор if
- Используйте оператор if, чтобы указать блок кода JavaScript, который будет выполняться, если условие истинно.

```
if (condition) {  
    // блок кода, который будет выполнен, если  
    условие истинно  
}
```

Обратите внимание, что если используется строчными буквами. Прописные буквы (If или IF) вызовут ошибку JavaScript.

```
if (hour < 18) {  
    greeting = "Good day";  
}
```

Оператор else

- Используйте оператор else, чтобы указать блок кода, который будет выполняться, если условие ложно.
- `if (condition) {`
 // block of code to be executed if the condition is true
`} else {`
 // block of code to be executed if the condition is false
`}`
- If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

Оператор else

- ```
if (hour < 18) {
 greeting = "Good day";
} else {
 greeting = "Good evening";
}
```

# Заявление else if

- Используйте оператор else if, чтобы указать новое условие, если первое условие ложно.
- `if (condition1) {`  
    *// block of code to be executed if condition1 is true*  
`} else if (condition2) {`  
    *// block of code to be executed if the condition1 is false*  
    *and condition2 is true*  
`} else {`  
    *// block of code to be executed if the condition1 is false*  
    *and condition2 is false*  
`}`

# Пример

- Если время меньше 10:00, создайте приветствие «Доброе утро», если нет, но время меньше 20:00, создайте приветствие «Добрый день», в противном случае - «Добрый вечер»:

```
if (time < 10) {
 greeting = "Good morning";
} else if (time < 20) {
 greeting = "Good day";
} else {
 greeting = "Good evening";
}
```

# JavaScript Заявление switch

- Заявление switch используется для выполнения переключения различных действий, основанных на различных условиях.
- Использовать заявление switch для выбора одного из многих блоков кода для выполнения.

```
switch(выражение) {
 case x:
 // code block
 break;
 case y:
 // code block
 break;
 default:
 // code block
}
```

Вот как это работает:

Выражение переключения вычисляется один раз. Значение выражения сравнивается со значениями каждого случая.

Если есть совпадение, выполняется связанный блок кода.

Если совпадений нет, выполняется блок кода по умолчанию.

# Пример

- Метод `getDay ()` возвращает день недели в виде числа от 0 до 6.
- (Воскресенье = 0, понедельник = 1, вторник = 2 ..)
- В этом примере номер дня недели используется для вычисления названия дня недели:

```
switch (new Date().getDay()) {
 case 0:
 day = "Sunday";
 break;
 case 1:
 day = "Monday";
 break;
 case 2:
 day = "Tuesday";
 break;
 case 3:
 day = "Wednesday";
 break;
 case 4:
 day = "Thursday";
 break;
 case 5:
 day = "Friday";
 break;
 case 6:
 day = "Saturday";
}
```

Примечание. Если вы опустите оператор `break`, следующий регистр будет выполнен, даже если оценка не соответствует регистру.

# Ключевое слово `break`

- Когда JavaScript достигает ключевого слова `break`, он выходит из блока `switch`.
- Это остановит выполнение внутри блока.
- Последний корпус в блоке переключателя ломать не нужно. Блок все равно там ломается (кончается).

# Ключевое слово по умолчанию default

- Ключевое слово по умолчанию определяет код для запуска, если нет совпадения по регистру:

```
switch (new Date().getDay()) {
 case 6:
 text = "Today is Saturday";
 break;
 case 0:
 text = "Today is Sunday";
 break;
 default:
 text = "Looking forward to the Weekend";
}
```

Случай по умолчанию не обязательно должен быть последним регистром в блоке переключения:

Если default - это не последний регистр в блоке switch, не забудьте завершить регистр по умолчанию перерывом. (break)

# Общие блоки кода

- Иногда вам может понадобиться, чтобы в разных случаях переключения использовался один и тот же код.
- В этом примере 4 и 5 используют один и тот же блок кода, а 0 и 6 совместно используют другой блок кода:

```
switch (new Date().getDay()) {
 case 4:
 case 5:
 text = "Soon it is Weekend";
 break;
 case 0:
 case 6:
 text = "It is Weekend";
 break;
 default:
 text = "Looking forward to the Weekend";
}
```

# Детали переключения

- Если несколько вариантов соответствуют значению case, выбирается первый вариант.
- Если подходящих случаев не найдено, программа переходит к метке по умолчанию.
- Если метка по умолчанию не найдена, программа переходит к оператору (операторам) после переключателя.

# Строгое сравнение

- В случаях переключения используется строгое сравнение (===).
- Для соответствия значения должны быть одного типа.
- Строгое сравнение может быть истинным, только если операнды одного типа.
- В этом примере совпадения по x не будет:

```
var x = "0";
switch (x) {
 case 0:
 text = "Off";
 break;
 case 1:
 text = "On";
 break;
 default:
 text = "No value found";
}
```

# Массивы

- Массивы JavaScript используются для хранения нескольких значений в одной переменной.

```
var cars = ["Saab", "Volvo", "BMW"];
```

Что такое массив?

Массив - это специальная переменная, которая может содержать более одного значения одновременно.

Если у вас есть список элементов (например, список названий автомобилей), хранение автомобилей в отдельных переменных может выглядеть следующим образом:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

Однако что, если вы хотите просмотреть машины и найти конкретную? А если бы у вас было не 3 машины, а 300?

Решение - массив!

Массив может содержать много значений под одним именем, и вы можете получить доступ к значениям, указав номер индекса.

# Создание массива

- Использование литерала массива - это самый простой способ создать массив JavaScript.
- Синтаксис: `var array_name = [item1, item2, ...];`

Пробелы и перенос строки не важны. Объявление может занимать несколько строк:

```
var cars = [
 "Saab",
 "Volvo",
 "BMW"
];
```

# Использование ключевого слова JavaScript

## НОВИНКА

В следующем примере также создается массив и присваиваются ему значения: `var cars = new Array("Saab", "Volvo", "BMW");`

Два приведенных выше примера делают то же самое. Нет необходимости использовать `new Array ()`.

Для простоты, удобочитаемости и скорости выполнения используйте первый (метод литерала массива).

### *Доступ к элементам массива*

Вы получаете доступ к элементу массива, ссылаясь на номер индекса. Этот оператор обращается к значению первого элемента в `cars`:

**Примечание.** Индексы массивов начинаются с 0. [0] - это первый элемент. [1] - второй элемент.

```
var name = cars[0];
```

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

# Изменение элемента массива

Этот оператор изменяет значение первого элемента в cars:

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

# Доступ к полному массиву

С помощью JavaScript полный массив можно получить, сославшись на имя массива:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

# Массивы - это объекты

- Массивы - это особый тип объектов. Оператор `typeof` в JavaScript возвращает «объект» для массивов.
- Но массивы JavaScript лучше всего описывать как массивы.
- Массивы используют числа для доступа к своим «элементам». В этом примере `person [0]` возвращает John:

```
var person = ["John", "Doe", 46];
```

Объекты используют имена для доступа к своим «членам». В этом примере `person.firstName` возвращает John:

```
var person = {firstName:"John",
lastName:"Doe", age:46};
```

# Элементы массива могут быть объектами

- Переменные JavaScript могут быть объектами. Массивы - это особые виды объектов.
- Благодаря этому в одном массиве могут быть переменные разных типов.
- Вы можете иметь объекты в массиве. Вы можете иметь функции в массиве. Вы можете иметь массивы в массиве:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

# Свойства и методы массива

- Настоящая сила массивов JavaScript - это встроенные свойства и методы массива:

```
var x = cars.length; // The length property returns the number of elements
var y = cars.sort(); // The sort() method sorts arrays
```

## Свойство длины

Свойство `length` массива возвращает длину массива (количество элементов массива).

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length; // the length of fruits is 4
```

Свойство `length` всегда на единицу больше, чем самый высокий индекс массива.

# Доступ к элементам массива

- Доступ к первому элементу массива
- `fruits = ["Banana", "Orange", "Apple", "Mango"];`  
`var first = fruits[0];`
- Доступ к последнему элементу массива
- `fruits = ["Banana", "Orange", "Apple", "Mango"];`  
`var last = fruits[fruits.length - 1];`

# Зацикливание элементов массива

- Самый безопасный способ перебрать массив - использовать цикл for:

```
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;
```

```
text = "";
for (i = 0; i < fLen; i++) {
 text += "" + fruits[i] + "";
}
text += "";
```

## Вы также можете использовать функцию `Array.forEach ()`:

```
var fruits, text;
fruits =
["Banana", "Orange", "Apple", "Mango"];
```

```
text = "";
fruits.forEach(myFunction);
text += "";
```

```
function myFunction(value) {
 text += "" + value + "";
}
```

# Добавление элементов массива

- Самый простой способ добавить новый элемент в массив - использовать метод `push ()`:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon"); // adds a new element (Lemon) to fruits
```

Новый элемент также можно добавить в массив с помощью свойства `length`:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon"; // adds a new element (Lemon) to fruits
```

## **ВНИМАНИЕ!**

Добавление элементов с высокими индексами может создать неопределенные «дыры» в массиве:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[6] = "Lemon"; // adds a new element (Lemon) to fruits
```

# Ассоциативные массивы

- Многие языки программирования поддерживают массивы с именованными индексами.
- Массивы с именованными индексами называются ассоциативными массивами (или хэшами).
- JavaScript не поддерживает массивы с именованными индексами.
- В JavaScript массивы всегда используют нумерованные индексы.

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length; // person.length will return 3
var y = person[0]; // person[0] will return "John"
```

## ВНИМАНИЕ !!

Если вы используете именованные индексы, JavaScript переопределит массив в стандартный объект.

После этого некоторые методы и свойства массива будут давать неверные результаты.

# Разница между массивами и объектами

```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
var x = person.length; // person.length
will return 0
var y = person[0]; // person[0] will
return undefined
```

**Массивы** - это особый вид объектов с пронумерованными индексами.

***Когда использовать массивы. Когда использовать объекты.***

JavaScript не поддерживает ассоциативные массивы.

Вы должны использовать объекты, если хотите, чтобы имена элементов были строками (текстом).

Вы должны использовать массивы, если хотите, чтобы имена элементов были числами.

- В JavaScript массивы используют нумерованные индексы.
- В JavaScript объекты используют именованные индексы.

# Избегайте нового массива ()

- Нет необходимости использовать встроенный конструктор массива JavaScript `new Array ()`.
- Вместо этого используйте `[]`.
- Эти два разных оператора создают новый пустой массив с именем `points`:  
`var points = new Array(); // Bad`  
`var points = []; // Good`

Эти два разных оператора создают новый массив, содержащий 6 чисел:

```
var points = new Array(40, 100, 1, 5, 25, 10); // Bad
var points = [40, 100, 1, 5, 25, 10]; // Good
```

Ключевое слово `new` только усложняет код. Это также может привести к неожиданным результатам:

```
var points = new Array (40, 100); // Создает массив из двух элементов (40 и 100)
```

```
var points = new Array (40); // Создает массив из 40 неопределенных элементов !!!!!
```

# Как распознать массив

- Распространенный вопрос: как узнать, является ли переменная массивом?
- Проблема в том, что оператор JavaScript `typeof` возвращает «объект»:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
typeof fruits; // returns object
```

Оператор `typeof` возвращает объект, потому что массив JavaScript является объектом.

<p>The new ECMASCRIPT 5 method `isArray` returns true when used on an array:</p>

<p id="demo"></p>

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = Array.isArray(fruits);
</script>
```

## ВЫ МОЖЕТЕ СОЗДАТЬ СВОЮ СОБСТВЕННУЮ ФУНКЦИЮ isArray ():

- Приведенная выше функция всегда возвращает истину, если аргумент является массивом.
- А точнее: он возвращает истину, если прототип объекта содержит слово «Массив».

```
<p>This "home made" isArray() function returns true
when used on an array:</p>
```

```
<p id="demo"></p>
```

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML =
isArray(fruits);

function isArray(myArray) {
 return myArray.constructor.toString().indexOf("Array")
> -1;
}
</script>
```

Оператор instanceof возвращает true, если объект создается данным конструктором:

- **JavaScript Arrays**
- The instanceof operator returns true when used on an array:
- true

```
<h2>JavaScript Arrays</h2>
```

```
<p>The instanceof operator returns true when used on an array:</p>
```

```
<p id="demo"></p>
```

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits
instanceof Array;
</script>
```

# Циклы

- Циклы могут выполнять блок кода несколько раз.
- Циклы JavaScript
- Циклы удобны, если вы хотите запускать один и тот же код снова и снова, каждый раз с другим значением.
- Часто это бывает при работе с массивами:



**Спасибо**