JS: Arrays, loops

НАШИ ПРАВИЛА

Включенная камера

Вопросы по поднятой руке

Не перебиваем друг друга

Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору

Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ПОИГРАЕМ ;)

Какие математические операторы в знаете?

Чем отличается строгое и нестрогое равенство

Расскажите синтаксис условного оператора

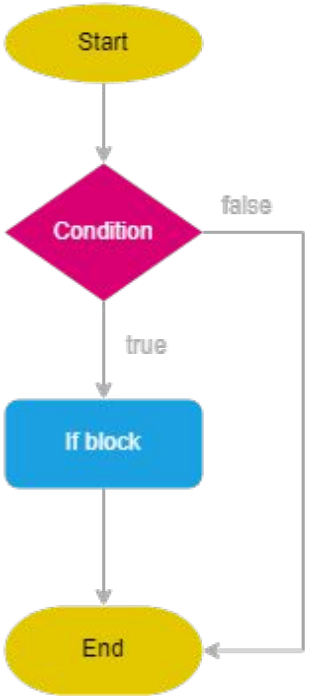
ЦЕЛЬ

Изучить особенности и синтаксис работы с массивами в js. Изучить методы push, pull, shift, unshift. Изучить циклы

ПЛАН ЗАНЯТИЯ

- Логические операторы
- Шаблонные строки
- Массивы: доступ к элементам, длинна, добавление, удаление
- Циклы

Условные операторы



Условный оператор „?“ (тернарный)

Так называемый **«условный»** оператор «вопросительный знак» позволяет нам сделать это более коротким и простым способом.

Оператор представлен знаком вопроса **?**. Его также называют «тернарный», так как этот оператор, единственный в своём роде, имеет три аргумента.

```
let result = условие ? значение1 : значение2;
```

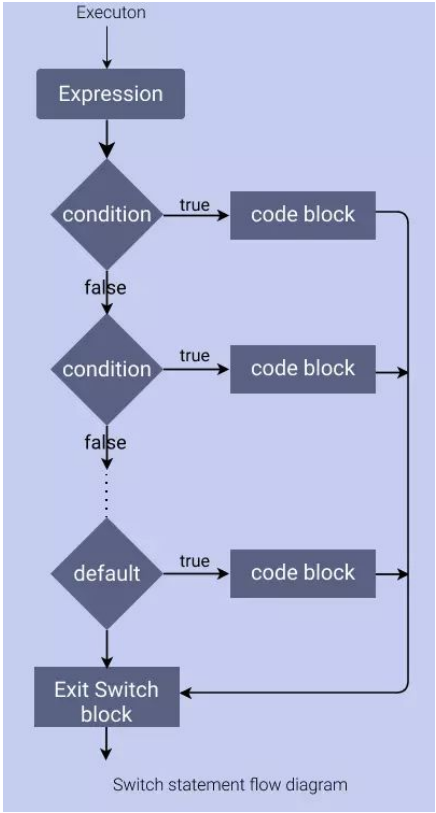


```
let accessAllowed = (age > 18) ? true : false;
```

Конструкция "switch"

Конструкция **switch** заменяет собой сразу несколько if.

Она представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами.



Конструкция switch имеет один или более блок case и необязательный блок default.

Выглядит она так:

```
switch(x) {
  case 'value1':  // if (x === 'value1')
    ...
    [break]

  case 'value2':  // if (x === 'value2')
    ...
    [break]

  default:
    ...
    [break]
}
```

Переменная **x** проверяется на строгое равенство первому значению **value1**, затем второму **value2** и так далее.
Если соответствие установлено – **switch** начинает выполняться от соответствующей директивы case и далее, до ближайшего **break** (или до конца **switch**).
Если ни один case не совпал – выполняется (если есть) вариант **default**.

Пример использования switch (сработавший код выделен):

```
let a = 2 + 2;

switch (a) {
  case 3:
    alert( 'Маловато' );
    break;
  case 4:
    alert( 'В точку!' );
    break;
  case 5:
    alert( 'Перебор' );
    break;
  default:
    alert( "Нет таких значений" );
}
```

Логические операторы

|| (ИЛИ)

Оператор «ИЛИ» выглядит как двойной символ вертикальной черты:

```
result = a || b;
```

Существует всего четыре возможные логические комбинации:

```
alert( true || true );   // true
alert( false || true );  // true
alert( true || false );  // true
alert( false || false ); // false
```

|| (ИЛИ)

Обычно оператор || используется в if для проверки истинности любого из заданных условий.

```
let hour = 9;

if (hour < 10 || hour > 18) {
  alert( 'Офис закрыт.' );
}
```

Можно передать и больше условий:

```
let hour = 12;
let isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {
  alert( 'Офис закрыт.' ); // это выходной
}
```

&& (И)

Оператор И пишется как два амперсанда **&&** и возвращает true, если оба аргумента истинны, а иначе – false:

```
1  alert( true && true );   // true
2  alert( false && true );  // false
3  alert( true && false );  // false
4  alert( false && false ); // false
```

&& (И)

Пример с if:

```
1  let hour = 12;
2  let minute = 30;
3
4  if (hour == 12 && minute == 30) {
5    alert( 'Время 12:30' );
6  }
```

! (НЕ)

Оператор **НЕ** представлен восклицательным знаком **!**

Оператор принимает один аргумент и выполняет следующие действия:

1. Сначала приводит аргумент к логическому типу true/false.
2. Затем возвращает противоположное значение.

```
1  alert( !true ); // false
2  alert( !0 ); // true
```

Шаблонные строки (Template Strings)

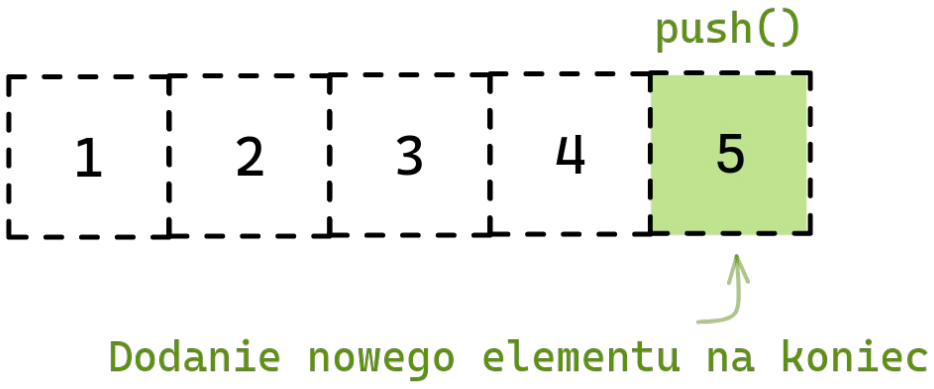
Шаблонные строки предоставляют удобный способ вставки переменных и выражений в строки.

В шаблонных строках можно использовать выражения, заключенные в ${}

```
let myName = "Alice";
let greeting = `Hello, ${myName}!`;

console.log(greeting); // "Hello, Alice!"
```

Массивы



Довольно часто мы понимаем, что нам необходима упорядоченная коллекция данных, в которой присутствуют 1-й, 2-й, 3-й элементы и т.д.

Например, она понадобится нам для хранения списка чего-либо: пользователей, товаров, элементов HTML и т.д.

Для хранения упорядоченных коллекций существует особая структура данных, которая называется массив, Array.

Объявление

Существует два варианта синтаксиса для создания пустого массива:

```
1  let arr = new Array();
2  let arr = [];
```

Практически всегда используется второй вариант синтаксиса. В скобках мы можем указать начальные значения элементов:

```
1  let fruits = ["Яблоко", "Апельсин", "Слива"];
```

Объявление

Элементы массива нумеруются, начиная с нуля.

Мы можем получить элемент, указав его номер в квадратных скобках:

```
1  let fruits = ["Яблоко", "Апельсин", "Слива"];
2
3  alert( fruits[0] ); // Яблоко
4  alert( fruits[1] ); // Апельсин
5  alert( fruits[2] ); // Слива
```

Мы можем заменить элемент:

```
1  fruits[2] = 'Груша'; // теперь ["Яблоко", "Апельсин", "Груша"]
```

…Или добавить новый к существующему массиву:

```
1  fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]
```

Длина массива

Общее число элементов массива содержится в его свойстве length:

```
1  let fruits = ["Яблоко", "Апельсин", "Слива"];
2
3  alert( fruits.length ); // 3
```

Содержание массива

В массиве могут храниться элементы любого типа.

```
let mixedArray = [42, "Hello", true, [1, 2, 3]];
```

Методы pop/push, shift/unshift

Очередь – один из самых распространённых вариантов применения массива. В области компьютерных наук так называется упорядоченная коллекция элементов, поддерживающая два вида операций:

- push добавляет элемент в конец.
- shift удаляет элемент в начале, сдвигая очередь, так что второй элемент становится первым.

На практике необходимость в этом возникает очень часто. Например, очередь сообщений, которые надо показать на экране.

Методы pop/push, shift/unshift

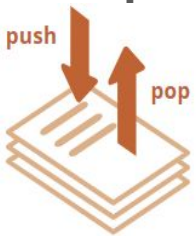
Существует и другой вариант применения для массивов – структура данных, называемая стек.

Она поддерживает два вида операций:

- push добавляет элемент в конец.
- pop удаляет последний элемент.

Таким образом, новые элементы всегда добавляются или удаляются из «конца».

Примером стека обычно служит колода карт: новые карты кладутся наверх и берутся тоже сверху:



Методы, работающие с концом массива:

pop

Удаляет последний элемент из массива и возвращает его:

```
1  let fruits = ["Яблоко", "Апельсин", "Груша"];
2
3  alert( fruits.pop() ); // удаляем "Груша" и выводим его
4
5  alert( fruits ); // Яблоко, Апельсин
```

Методы, работающие с концом массива:

push

Добавляет элемент в конец массива:

```
let fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша
```

Методы, работающие с началом массива:

shift

Удаляет из массива первый элемент и возвращает его:

```
1  let fruits = ["Яблоко", "Апельсин", "Груша"];
2
3  alert( fruits.shift() ); // удаляем Яблоко и выводим его
4
5  alert( fruits ); // Апельсин, Груша
```

Методы, работающие с началом массива:

unshift

Добавляет элемент в начало массива:

```
1  let fruits = ["Апельсин", "Груша"];
2
3  fruits.unshift('Яблоко');
4
5  alert( fruits ); // Яблоко, Апельсин, Груша
```

Методы push и unshift могут добавлять сразу несколько элементов:

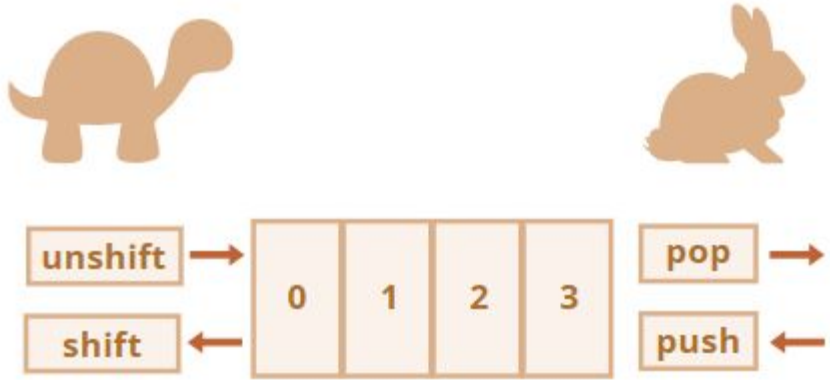
```
let fruits = ["Яблоко"];

fruits.push("Апельсин", "Груша");
fruits.unshift("Ананас", "Лимон");

// ["Ананас", "Лимон", "Яблоко", "Апельсин", "Груша"]
alert( fruits );
```

Эффективность

Методы `push/pop` выполняются быстро, а методы `shift/unshift` – медленно.



Циклы

Цикл “for”

Более сложный, но при этом самый распространённый цикл — цикл for.

```
1  for (начало; условие; шаг) {
2    // ... тело цикла ...
3  }
```

Цикл “for”

Давайте разберёмся, что означает каждая часть, на примере. Цикл ниже выполняет alert(i) для i от 0 до (но не включая) 3:

```
1 for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2
2   alert(i);
3 }
```

Прерывание цикла: «break»

Обычно цикл завершается при вычислении условия в false.

Но мы можем выйти из цикла в любой момент с помощью специальной директивы break.

Например, следующий код подсчитывает сумму вводимых чисел до тех пор, пока посетитель их вводит, а затем – выдаёт:

```
let sum = 0;

while (true) {

  let value = +prompt("Введите число", '');

  if (!value) break; // (*)

  sum += value;

}
alert( 'Сумма: ' + sum );
```

Переход к следующей итерации: continue

Директива continue – «облегчённая версия» break. При её выполнении цикл не прерывается, а переходит к следующей итерации (если условие все ещё равно true).

Её используют, если понятно, что на текущем повторе цикла делать больше нечего.

Например, цикл ниже использует continue, чтобы выводить только нечётные значения:

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

  // если true, пропустить оставшуюся часть тела цикла
  if (i % 2 == 0) continue;

  alert(i); // 1, затем 3, 5, 7, 9
}
```

Перебор элементов массива - for

Одним из самых старых способов перебора элементов массива является цикл for по цифровым индексам:

```
1  let arr = ["Яблоко", "Апельсин", "Груша"];
2
3  for (let i = 0; i < arr.length; i++) {
4    alert( arr[i] );
5  }
```

Перебор элементов массива - for...of

Оператор for...of выполняет цикл обхода

```
1 let iterable = [10, 20, 30]
2
3 for (let value of iterable) {
4   value += 1
5   console.log(value)
6 }
7 // 11
8 // 21
9 // 31
```

Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH

AIT
Hi-tech team