Front-end

JavaScript

JS. Lesson 2

- Операторы
- Приоритет операторов
- Преобразование типов (явное, неявное, isNaN, Boolean, Number, String)
- Шаблонные строки, бэктики
- Постфиксный и префиксный инкремент
- Условные операторы (If else, switch case)
- Тернарный оператор

JS. Operators

Оператор — это элемент языка, задающий полное описание действия, которое необходимо выполнить.

Оператор присваивания (=) присваивает значение к переменной.

Оператор сложения (+) слаживает числа

Оператор умножения (*) умножает числа

JS. Operators. Арифметические операторы

Арифметические операторы используются для выполнения арифметических действий с числами:

Оператор	Описание
+	Сложение
-	Вычитание
*	Умножение
**	Возведение в степень
/	Деление
%	Модуль (Остаток деления)
++	Инкремент (увеличение на 1)
--	Декремент (уменьшение на 1)

JS. Operators. Деление на 0

В программировании широко известна ошибка "деление на ноль". В низкоуровневых языках она приводит к краху программы и необходимости ее перезапуска. Там, где другие падают, JavaScript продолжает работать.

Деление на ноль действительно создает бесконечность. Бесконечность в JavaScript — самое настоящее число, с которым возможно проводить различные операции. В повседневных задачах смысла от этого мало, так как большинство операций с бесконечностью завершаются созданием бесконечности, например, при прибавлении любого числа к бесконечности мы все равно получим бесконечность.

Infinity + 4; => **Infinity**

Infinity - 4; => **Infinity**

Infinity * Infinity; => **Infinity**

JS. Operators. Деление на 0

Infinity переводится как «бесконечность».

Infinity или -Infinity (минус Infinity) в Javascript означает бесконечность.

В диапазон Infinity входят значения, выходящие за границы: 1.79769313486231570E+10308 - больше этого числа. Или меньше -1.79769313486231570E+10308 - для -Infinity.

Где встречается Infinity в JavaScript?

Infinity в JavaScript, например, можно получить при делении числа (кроме самого нуля) на ноль. Деление ноль на ноль в JavaScript дает NaN.

Или же Infinity можно увидеть, если результат вычислений не попадает в допустимый диапазон чисел.

JS. Operators. Операторы присваивания

Операторы присваивания присваивают значения к переменным JavaScript.

Оператор	Пример	Аналогично
=	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
**=	x **= y	x = x ** y

JS. Operators. Строковые операторы

Оператор **+**, также может быть использован для добавления (конкатенации) строк.

При использовании со строками оператор **+** называется оператором конкатенации.

```
let firstName = 'Fedor'
let surName = 'Ivanov'

let fullName = firstName + ' ' + surName

console.log(fullName) // 'Fedor Ivanov'
```

Оператор присваивания **+=**, также может использоваться для добавления (объединения) строк

JS. Operators. Добавление строк и чисел

Добавление двух чисел вернет сумму, а добавление числа и строки вернет строку

Если вы сложите число и строку, результатом будет строка!

```
let x = 2
let y = '2'

let result = x + y

console.log(result) // 22 -> string
```

JS. Operators. Операторы сравнения

Операторы сравнения используются в логических операторах для определения равенства или различия между переменными или значениями.

Оператор	Описание
==	равно
===	равное значение и равный тип
!=	не равно
!==	не равное значение или не равный тип
>	больше, чем
<	меньше, чем
>=	больше или равно
<=	меньше или равно
?	тернарный оператор

JS. Operators. Операторы сравнения

Оператор == сравнивает на равенство, а вот === — на идентичность.

Плюс оператора === состоит в том, что он не приводит два значения к одному типу.

=== - без приведения типов

== - с приведением типов

```
let num = 1234
let stringNum = '1234'

console.log(num == stringNum) // =>true
console.log(num === stringNum)  // => false
```

Двойной знак равенства (==) проверяет только равенство значений. Он выполняет приведение типов. Это означает, что перед проверкой значений он преобразует типы переменных, чтобы привести их в соответствие друг к другу.

Тройной знак равенства (===) не выполняет приведение типов. Он проверяет, имеют ли сравниваемые переменные одинаковое значение и тип.

JS. Operators. Операторы сравнения

!= (не равенство)

!== (строгое не равенство)

```
console.log('3' > '25') // => true

console.log('Hello' > 'hello') // => false
```

Строки сравниваются посимвольно. Т.е. сначала первые символы операндов. Если первый символ первого операнда большое второго, то значит первый операнд больше второго. Если первые символы равны, то сравнивается вторые символы операндов и т.д.

При этом какой символ больше другого определяется по их кодам в таблице Unicode.

При сравнении маленькие и большие буквы алфавита не равны, т.к. имеют разные коды в таблице Unicode. Поэтому при операциях сравнения строки желательно приводить к одному регистру.

JS. Operators. Операторы сравнения

Значения null и undefined равны == друг другу и не равны ничему другому.

JS. Operators. Логические операторы

Логические операторы используются для определения логики между переменными или значениями.

Оператор	Описание
&&	логический AND (И)
||	логический OR (ИЛИ)
!	логический NOT (НЕ)

JS. Operators. Логические операторы

Оператор «НЕ»

```
console.log(!false)// => true
console.log(!true)// => false
```

Оператор «НЕ» обозначается в JavaScript с помощью !. Он является префиксным унарным оператором, который всегда возвращает значение логического типа, т.е. true или false.

Ложными значениями в JavaScript являются те, которые при приведении к логическому типу дают false. Конвертировать любое значение в логическое можно с помощью функции Boolean() или двойного оператора «НЕ» (!!):

```
let res1 = Boolean(10)
let res2 = !!10 // аналог Boolean

console.log(res1) // => true
console.log(res2) // => true
```

JS. Operators. Логические операторы

Оператор «И»

&& - означает «И» (в математической логике это называют конъюнкцией). Всё выражение считается истинным только в том случае, когда истинен каждый операнд — каждое из составных выражений. Иными словами, && означает «и то, и другое».

```
console.log(2 > 1 && 4 < 5) // => true
```

```
console.log(2 < 1 || 4 < 5) // => true
```

JS. Operators. Логические операторы

Оператор «ИЛИ»

|| — «ИЛИ» (дизъюнкция). Он означает «или то, или другое, или оба». Операторы можно комбинировать в любом количестве и любой последовательности, но когда одновременно встречаются && и ||, то приоритет лучше задавать скобками.

```
console.log(2 > 1 && 4 < 5) // => true
```

```
console.log(2 < 1 || 4 < 5) // => true
```

JS. Operators. Приоритет

Приоритет операторов определяет порядок, в котором операторы выполняются. Операторы с более высоким приоритетом выполняются первыми.

Каждый оператор имеет соответствующий номер приоритета. Тот, у кого это число больше, – выполнится раньше. Если приоритет одинаковый, то порядок выполнения – слева направо.

JS. Преобразование типов (Type Conversion)

Переменные JavaScript можно преобразовать в новую переменную и другой тип данных:

- с помощью функции JavaScript
- автоматически с помощью самого JavaScript

Приведение (или преобразование) типов — это процесс конвертации значения из одного типа в другой.

В JavaScript типы можно преобразовывать явно и неявно.

Когда мы вызываем функцию, чтобы получить конкретный тип — это явное преобразование

Так как JavaScript — это язык со слабой типизацией, значения могут быть конвертированы между различными типами автоматически. Это называют неявным приведением типов.

JS. Преобразование типов

Явно:

- Number(value) - преобразуется в числа => number
- String(value) - преобразуется в строки => string
- Boolean(value) - преобразуется в логический тип => true, false
- +, *, / - приведение к числу => number

```
let a = 2
let b = '4'

console.log(a * b) // 8 type - number
console.log(a + +b) // 6 type - number
console.log(b / a) // 2 type - number
```

JS. Преобразование типов

Неявно:

- alert(value) => string
- неявное преобразование вызывает использование обычного оператора сложения, +, с двумя операндами, если один из них является строкой
- !, ||, && (не, или, и)
- нестрогое сравнение значений между собой с помощью ==, JavaScript приводит типы самостоятельно
- If (value)

JS. Преобразование типов

Ложноподобное (falsy) значение — значение, которое становится false в булевом контексте.

JavaScript использует преобразование типов, чтобы привести значение к булевому типу, там, где это требуется (например, в условных конструкциях и циклах)

Значения, которые всегда ложные Boolean():

- false
- 0
- '' - пустая строка
- undefined неопределенность, эквивалент "ничто"
- null эквивалент "ничто"
- NaN специальное числовое значение, означающее "не число"

JS. Преобразование типов

NaN (not a number) - специальное значение "не число", которое обычно говорит о том, что была выполнена бессмысленная операция.

Результатом практически любой операции, в которой участвует NaN, будет NaN

NaN + 1 => **NaN**

isNaN(value) – проверка на NaN

```
let z = '2value'

console.log(+z) // -> NaN
console.log(isNaN(+z)) // -> true
```

```
console.log(isNaN(NaN)) // => true

console.log(typeof(NaN)) // => number

console.log('Hello' * 2) // => NaN
console.log('Hello' / 2) // => NaN
console.log('Hello' - 2) // => NaN
console.log(2 - 'Hello') // => NaN
```

JS. Template literals (Template strings)

Шаблонные строки — строки, внутри которых можно использовать выражения: например, значения переменных.

Такие строки оборачиваются в обратные апострофы `` `...` ``.
Используемое в строке выражение — в фигурные скобки со знаком доллара `${...}`

Позволяют писать многострочные строки, использовать внутри себя цитаты (другие кавычки) без необходимости экранирования.

JS. Template literals (Template strings)

Интерполяция

При объединении строк можно использовать конкатенацию:

```
const firstName = 'David';

const greeting = 'Hello';

console.log(greeting + ', ' + firstName + '!'); => Hello, David!
```

Это довольно простой случай, но даже здесь нужно приложить усилия, чтобы увидеть, какая в итоге получится строка. Нужно следить за несколькими кавычками и пробелами, и без вглядывания не понять, где что начинается и кончается.

JS. Template literals (Template strings)

Интерполяция

Есть другой, более удобный и изящный способ решения той же задачи — интерполяция. Вот, как это выглядит:

const firstName = 'David';

const greeting = 'Hello';

Интерполяция не работает с одинарными и двойными кавычками

console.log(`${greeting}, ${firstName}!`); => Hello, David!

Мы просто создали одну строку и «вставили» в неё в нужные места константы с помощью знака доллара и фигурных скобок ${ }. Получился как будто бланк, куда внесены нужные значения. И нам не нужно больше заботиться об отдельных строках для знаков препинания и пробелов — все эти символы просто записаны в этой строке-шаблоне.

В одной строке можно делать сколько угодно подобных блоков.

Интерполяция работает только со строками в бэктиках. Это символ `.

Почти во всех языках интерполяция предпочтительнее конкатенации для объединения строк. Строка при этом получается склеенная, и внутри неё хорошо просматриваются пробелы и другие символы. Во-первых, интерполяция позволяет не путать строки с числами (из-за знака +), а во-вторых, так гораздо проще (после некоторой практики) понимать строку целиком.

JS. Постфиксный и префиксный инкремент

Из языка Си в JavaScript перекочевали две операции:

- **инкремент ++**
- **декремент --**

которые очень часто встречаются вместе с циклами. Эти унарные операции увеличивают и уменьшают на единицу число, записанное в переменную

Кроме постфиксной формы, у них есть и префиксная

При использовании префиксной нотации сначала происходит изменение переменной, а потом возврат.

При использовании постфиксной нотации — наоборот: можно считать, что сначала происходит возврат, а потом изменение переменной.

```
let i = 0
i++ // 0
i++ // 1

i-- // 2
i-- // 1


let i = 0;
++i // 1
++i // 2

--i // 1
--i // 0
```

JS. Условные операторы. If else

Условные операторы используются для выполнения разных действий в зависимости от разных условий.

Очень часто, когда вы пишете код, вы хотите выполнять разные действия для разных решений.

Вы можете использовать условные операторы в своем коде, чтобы сделать это.

В JavaScript есть следующие условные выражения:

- используйте if, чтобы указать блок кода, который нужно выполнить, если указанное условие true (истинно)
- используйте else, чтобы указать блок кода, который будет выполнен, если то же условие false (ложно)
- используйте else if, чтобы указать новое условие для проверки, если первое условие false (ложно)
- используйте switch, чтобы указать много альтернативных блоков кода, которые должны быть выполнены

JS. Условные операторы. If else

Оператор if

Используйте if заявление, чтобы указать блок кода JavaScript, который будет выполняться, если условие истинно.

Синтаксис:

```
if (condition) {
  блок кода, который должен быть выполнен, если условие истинно
}
```

Обратите внимание, что if это строчные буквы.

Заглавные буквы (If или IF) приведут к ошибке JavaScript.

JS. Условные операторы. If else

Оператор else

Используйте else оператор, чтобы указать блок кода, который будет выполняться, если условие ложно.

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

JS. Условные операторы. If else

Оператор else if

Используйте else if оператор, чтобы указать новое условие, если первое условие ложно.

Синтаксис

```
if (condition 1) {
  блок кода, который должен быть выполнен, если условие 1 истинно
} else if (condition 2) {
  блок кода, который должен быть выполнен, если условие 1 ложно, а условие2 истинно
} else {
  блок кода выполниться, если условие1 ложно, и условие 2 тоже ложно
}
```

JS. Условные операторы. Switch case

Оператор **switch** используется для выполнения различных действий, основанных на различных условиях.

Используйте оператор **switch**, чтобы выбрать один из множества блоков кода для выполнения.

JS. Условные операторы. Switch case

Вот как это работает:

- выражение switch (переключателя) вычисляется один раз
- значение выражения сравнивается со значениями каждого случая
- если есть совпадение, соответствующий блок кода выполняется
- если совпадений нет, выполняется блок кода по умолчанию

```
switch (expression) {
    case x:
        // блок кода
        break
    case y:
        // блок кода
        break
    default:
    // блок кода
}
```

JS. Условные операторы. Switch case

Ключевое слово break

Когда JavaScript достигает ключевого слова break, он выходит из блока switch.

Это остановит выполнение внутри блока.

Нет необходимости обрывать последний кейс в блоке switch. Блок всё равно обрывается (заканчивается).

Примечание: Если вы пропустите оператор break, следующий кейс будет выполнен, даже если оценка не соответствует кейсу.

JS. Условный (тернарный) оператор

Тернарный оператор — это короткая запись условного оператора if...else. Он есть во многих языках программирования. Часто применятся при написании кода, для упрощения в том случае, если вам не нужно делать несколько операций при выполнении условия

Синтаксис тернарного оператора в JS такой:

```
'ваше_условие' ? 'в_случае_true' : 'в_случае_false'
```