

Алгоритмы обработки данных на JavaScript

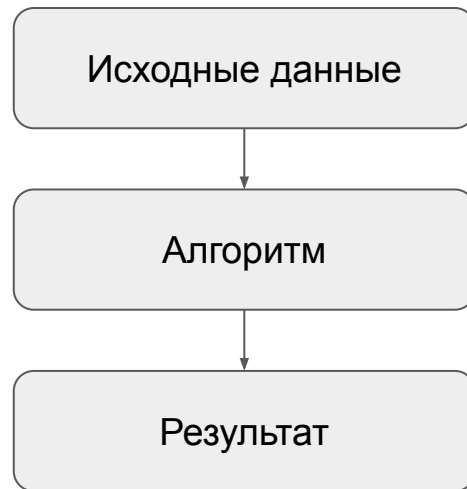
ОСНОВЫ языка JavaScript

Алгоритм

Алгоритм - описание последовательности шагов в решении задачи, приводящих от исходных данных к требуемому результату

- дискретность;
- определенность;
- результативность;
- понятность;
- массовость.

Язык описания алгоритма - язык программирования



Исторический экскурс



- 1995 - Брэндан Эйх в рамках работы над браузером Netscape Navigator создаёт встроенный скриптовый язык Mocha
- Из-за популярности языка Java был переименован в JavaScript

ECMAScript

Язык возник на основе нескольких технологий, самыми известными из которых являются языки JavaScript (Mozilla) и JScript (Microsoft).

Предназначен для стандартизации скриптовых языков программирования. Он же задаёт их вектор дальнейшего развития.

Программа на JavaScript

Программа на JavaScript состоит из:

- выражение - фрагмент кода, результатом работы которого является некая величина (фраза языка).
- инструкции - синтаксические конструкции и команды, которые выполняют действия (предложение языка). Отделяются друг от друга точкой с запятой.


Программа = список инструкций.

Система типов

В JavaScript представлена следующая система типов:

- Примитивные
 - string - последовательность символов, e.g. 'text', "text"
 - number - числа, e.g. 123
 - boolean - булево значение (true или false)
 - null - отсутствие значения
 - undefined - присваивается неинициализированной переменной
 - symbol - используется для создания уникальных идентификаторов
- Объектные
 - object

Числа



```
1 // Десятичная запись
2 1000
3 // Экспоненциальная запись
4 1e3 == 1 * 10 ** 3
5 // Двоичная запись
6 0b11 == 1 * 2 ** 1 + 1 * 2 ** 0
7 // Шестнадцатеричная запись
8 0xFF == 15 * 16 ** 1 + 15 * 16 ** 0
9 // Вещественное число
10 2.5
```

В JavaScript числа, объявленные непосредственно в коде, называются числовыми литералами.

Представляют числа в диапазоне $[-(2^{53}-1); 2^{53}-1]$

Специальные числовые значения



```
1 1 / 0 === Infinity
2 Math.log(0) === -Infinity // Math.log - логарифм
3 -1 / 0 === -Infinity
4 2 ** 1024 === Infinity
5
6 Infinity === Infinity + 1
7 Infinity + Infinity === Infinity
8 2 * Infinity === Infinity
```

Значение `Infinity` больше любого другого числа, включая саму положительную бесконечность. Это значение ведёт себя как математическая бесконечность.

Можно получить:

- при делении на ноль
- если результат вычислений не попадает в допустимый диапазон чисел
- в результате выполнения некоторых математических функций, например, логарифм, тангенс, e.t.c.

Специальные числовые значения



```
1  Math.sqrt(-1) === NaN
2  parseInt('два') === NaN
3
4  NaN === NaN // false!
```

Значение NaN используется для обозначения математической ошибки, которая возникает в том случае, если математическая операция не может быть совершена, или когда функция, пытающаяся считать число из строки, терпит неудачу по причине того, что в строке не число.

Для проверки на NaN используется функция [Number.isNaN](#)

Для общей проверки на специальные числовые значения следует использовать функцию [Number.isFinite](#)

Арифметические операторы

Оператор	Имя	Пример
+	Сложение	$2.1 + 3.2 = 5.3$
-	Вычитание	$3 - 2 = 1$
/	Деление	$10 / 2 = 5$
*	Умножение	$3 * 2 = 6$
%	Взятие остатка	$17 \% 5 = 2$
**	Показатель степени	$9 ** 2 = 81$

Приоритет операторов

В JS порядок вычисления операторов определяется [приоритетом операторов](#)

Приоритет умножения выше оператора сложения, вычитания. Правила, в целом, совпадают с теми, что изучались в школе

Для увеличения приоритета оператора используются скобки

> 2 + 3 * 5

< 17

> (2+3)*5

< 25

Операторы сравнения

Оператор	Имя	Назначение	Пример
===	Строгое равенство	Проверяет левый и правый операнд на идентичность с учётом типов	2 === 2
!==	Строгое неравенство	Проверяет левый и правый операнд на неидентичность с учётом типов	'abc' !== 'a'; 23 !== '23'
>	Больше	Проверяет, больше ли левое значение правого	2 > 1
<	Меньше	Проверяет, меньше ли левое значение правого	3 < 5
>=	Больше или равно	Проверяет, больше ли левое значение правого (или равно ему)	4 >= 2
<=	Меньше или равно	Проверяет, меньше ли левое значение правого (или равно ему)	3 <= 5

Логические операторы

Название логической операции	Логическая связка
Логическое НЕ (инверсия)	не; неверно, что
Логическое И (конъюнкция)	и; а; но; хотя
Логическое ИЛИ (дизъюнкция)	или

Конъюнкция

Логическая операция, ставящая в соответствие каждому двум высказываниям новое высказывание, являющееся истинным тогда и только тогда, когда оба исходных высказывания истинны

Пример: На улице светит солнце и безветренная погода

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

Дизъюнкция

Логическая операция, которая каждому двум высказываниям ставит в соответствие новое высказывание, являющееся ложным тогда и только тогда, когда оба исходных высказывания ложны.

Пример: На улице светит солнце или идёт дождь

A	B	$A \parallel B$
true	true	true
true	false	true
false	true	true
false	false	false

Инверсия

Логическая операция, которая каждому высказыванию ставит в соответствие новое высказывание, значение которого противоположно исходному.

Пример: неверно, что сегодня понедельник

A	!A
true	false
false	true

Строки

Строка - последовательность из нуля или более Unicode символов, заключенная в кавычки



```
1  const userName = 'John Doe';  
2  const greeting = "Hello"  
3  const greetingMessage = `${greeting} ${userName}`;
```

Строки

```
> const firstName = 'Артём';  
   const lastName = 'Тарасов';  
   'I\'m ' + firstName + ' ' + lastName
```

```
< "I'm Артём Тарасов"
```

```
> firstName.length
```

```
< 5
```

```
> firstName[2]
```

```
< 'Т'
```

- Конкатенация: оператор “+” при применении к строкам объединяет их
- Определение длины строки осуществляется при помощи свойства `length`
- Символ обратного слэша используется для управляющих последовательностей. Они используются, например, для перевода строк либо для экранирования кавычек
- Для получения символа по индексу используются квадратные скобки

typeof

x	typeof x
string	'string'
number	'number'
boolean	'boolean'
undefined	'undefined'
null	'object'
symbol	'symbol'
function	'function'
Все остальные объекты	'object'

Оператор typeof позволяет определить тип передаваемого аргумента

typeof null is 'object'

Данная ошибка не может быть исправлена, так как исправление сломает множество написанных программ.

Подробнее: <https://2ality.com/2013/10/typeof-null.html>

Переменная

Переменная – это «именованное хранилище» для данных.

Для создания переменных используется ключевое слово *let*

```
let message;
```

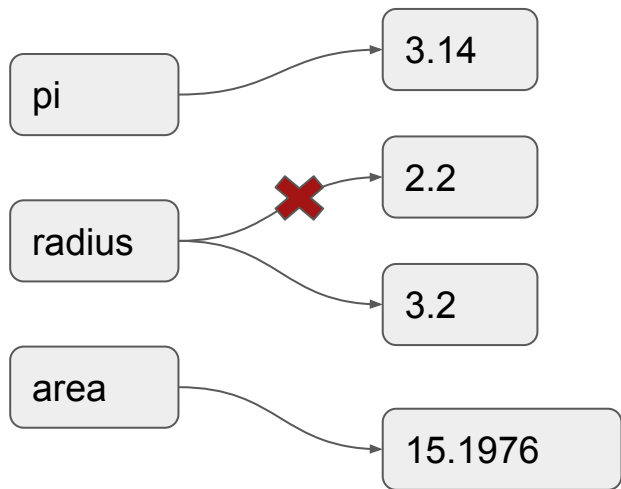
Данные помещаются в переменную при помощи оператора присваивания “ = ”

```
message = 'Hello world';
```

Либо в одну строку:

```
let message = 'Hello world';
```

Переменная



```
1  const pi = 3.14;  
2  let radius = 2.2;  
3  let area = pi * radius ** 2;  
4  radius += 1;
```

Правило именования переменных

Имя переменной = идентификатор

Имя переменной может содержать:

- строчные и прописные буквы
- цифры
- символы _ и \$
- имя переменной не может начинаться с цифры
- в качестве идентификатора не могут использоваться зарезервированные слова

Константы

Для создания констант используется ключевое слово *const*

Значение константы не может быть изменено по ходу выполнения программы



```
1  const year = 2022;  
2  year += 1  // Uncaught TypeError: Assignment to constant variable.
```


Стили написания составных слов в идентификаторах



```
1  // Camel case
2  const userName = 'John Doe';
3  // Snake case
4  const user_name = 'John Doe';
5  // Upper case
6  const COLOR_RED = 0xFF0;
```

Преобразование в число



```
1  const value = '123';  
2  
3  Number(value);  
4  +value;  
5  parseFloat(value);  
6  parseInt(value);
```

- Функция [Number](#) не может преобразовывать строки, содержащие не числовые символы
- Функции [parseInt](#) и [parseFloat](#) преобразуют строки до первого символа, отличного от 0-9, минуса, плюса, разделительной точки (для `parseFloat`) или показателя степени

Неявное преобразования в числа происходит в результате математических операций

```
> '12' * '3'
```

```
< 36
```

parseInt/parseFloat

```
> parseInt('19px')
```

```
< 19
```

```
> parseInt('19 рублей')
```

```
< 19
```

```
> parseFloat('3.14.15')
```

```
< 3.14
```

parseInt и parseFloat следует использовать если строка, например, содержит единицу измерения

Преобразование различных типов данных в число

x	Number(x)
undefined	NaN
null	0
true	1
false	0
number	x
string	Пустая строка становится нулём. Из любой другой строки выделяется число, если это невозможно, то NaN

Преобразование в булево

Преобразование `x` в булево осуществляется при помощи `!!x` либо при помощи функции `Boolean(x)`.

Все значения в JS можно разделить на истинноподобные (Truthy) и ложноподобные (Falsy).

Ложноподобные:

- `false`
- `0`
- `""`
- `null`
- `undefined`

Истинноподобные:

- `true`
- Любое число кроме 0 и NaN
- Любая строка кроме пустой
- Любой объект

Преобразование в строки

Строковое преобразование осуществляется при помощи функции `String(x)`. Неявное же строковое преобразование осуществляется когда требуется представление объекта в виде строки (например, при распечатке в консоль).

Преобразование в строки осуществляется по довольно очевидным правилам: `true` \rightarrow `'true'`, `123` \rightarrow `'123'`, `false` \rightarrow `'false'` и т.д.

Оператор равенства

Equality in JavaScript

	true	false	1	0	-1	"true"	"false"	"1"	"0"	"-1"	""	null	undefined	Infinity	-Infinity	[]	{}	[[]]	[0]	[1]	NaN
true	True	False	True	False	False	True	False	True	False	False	False	False	False	False	False	False	False	False	True	True	False
false	False	True	False	True	True	False	True	False	True	True	True	False	False	False	False	False	False	False	False	False	False
1	True	False	True	False	False	True	False	True	False	False	False	False	False	False	False	False	False	False	True	True	False
0	False	True	False	True	True	False	True	False	True	True	True	False	False	False	False	False	False	False	True	True	False
-1	False	True	False	False	True	True	False	False	True	True	True	False	False	False	False	False	False	False	False	False	False
"true"	True	False	False	False	False	True	False	True	False	False	False	False	False	False	False	False	False	False	True	True	False
"false"	False	True	False	True	True	False	True	False	True	True	True	False	False	False	False	False	False	False	False	False	False
"1"	True	False	True	False	False	True	False	True	False	False	False	False	False	False	False	False	False	False	True	True	False
"0"	False	True	False	True	True	False	True	False	True	True	True	False	False	False	False	False	False	False	True	True	False
"-1"	False	True	False	False	True	True	False	False	True	True	True	False	False	False	False	False	False	False	False	False	False
""	False	True	False	False	True	True	False	False	True	True	True	False	False	False	False	False	False	False	False	False	False
null	False	False	False	False	False	False	False	False	False	False	False	True	True	False	False	False	False	False	False	False	False
undefined	False	False	False	False	False	False	False	False	False	False	False	True	True	True	False	False	False	False	False	False	False
Infinity	False	False	False	False	False	False	False	False	False	False	False	False	False	True	True	False	False	False	False	False	False
-Infinity	False	False	False	False	False	False	False	False	False	False	False	False	False	True	True	True	False	False	False	False	False
[]	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	True	False	False	False	False
{}	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	True	False	False	False
[[]]	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	True	False	False
[0]	False	False	False	True	True	False	False	False	True	True	True	False	False	False	False	False	False	False	True	True	False
[1]	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False	False	True	False
NaN	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True

Not equal

Loose equality
Often gives "false"
positives like "1" is
true; [] is "0"

Strict equality
Mostly evaluates as
one would expect.

В JavaScript также представлен оператор равенства “==” и неравенства “!=”, который перед сравнением производит приведение типов.

Его использование не рекомендуется, так как результат его использования бывает сложно предугадать

[Сравнение результатов === и ==](#)