15 апреля 2023 г.

Строки

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков.

Внутренний формат для строк — всегда UTF-16, вне зависимости от кодировки страницы.

Кавычки

В JavaScript есть разные типы кавычек.

Строку можно создать с помощью одинарных, двойных либо обратных кавычек:

```
let single = 'single-quoted';
let double = "double-quoted";

let backticks = `backticks`;
```

Одинарные и двойные кавычки работают, по сути, одинаково, а если использовать обратные кавычки, то в такую строку мы сможем вставлять произвольные выражения, обернув их в `${…}`:

```
function sum(a, b) {
  return a + b;
}

alert(`1 + 2 = ${sum(1, 2)}.`); // 1 + 2 = 3.
```

Ещё одно преимущество обратных кавычек — они могут занимать более одной строки, вот так:

```
let guestList = `Guests:
 * John
 * Pete
 * Mary
`;

alert(guestList); // список гостей, состоящий из нескольких строк
```

Выглядит вполне естественно, не правда ли? Что тут такого? Но если попытаться использовать точно так же одинарные или двойные кавычки, то будет ошибка:

```
let guestList = "Guests: // Error: Unexpected token ILLEGAL
  * John";
```

Одинарные и двойные кавычки в языке с незапамятных времён: тогда потребность в многострочных строках не учитывалась. Что касается обратных кавычек, они появились существенно позже, и поэтому они гибче.

Обратные кавычки также позволяют задавать «шаблонную функцию» перед первой обратной кавычкой. Используемый синтаксис: `func`string`` . Автоматически вызываемая функция `func` получает строку и встроенные в неё выражения и может их обработать. Подробнее об этом можно прочитать в документации. Если перед строкой есть выражение, то шаблонная строка называется «теговым шаблоном». Это позволяет использовать свою шаблонизацию для строк, но на практике теговые шаблоны применяются редко.

Спецсимволы

Многострочные строки также можно создавать с помощью одинарных и двойных кавычек, используя так называемый «символ перевода строки», который записывается как `\n` :

```
let guestList = "Guests:\n * John\n * Pete\n * Mary";

alert(guestList); // список гостей, состоящий из нескольких строк
```

В частности, эти две строки эквивалентны, просто записаны по-разному:

```
// перевод строки добавлен с помощью символа перевода строки
let str1 = "Hello\nWorld";

// многострочная строка, созданная с использованием обратных кавычек
let str2 = `Hello
World`;

alert(str1 == str2); // true
```

Есть и другие, реже используемые спецсимволы. Вот список:

Символ	Описание
`\n`	Перевод строки
`\r`	В текстовых файлах Windows для перевода строки используется комбинация символов `\r\n` , а на других ОС это просто `\n` . Это так по историческим причинам, ПО под Windows обычно понимает и просто `\n` .
`\'` , `\"`	Кавычки
`\\`	Обратный слеш
`\t`	Знак табуляции
`\b` , `\f` , `\v`	Backspace, Form Feed и Vertical Tab — оставлены для обратной совместимости, сейчас не используются.

Как вы можете видеть, все спецсимволы начинаются с обратного слеша, `\` — так называемого «символа экранирования».

Он также используется, если необходимо вставить в строку кавычку.

К примеру:

```
1  alert( 'I\'m the Walrus!' ); // I'm the Walrus!
```

Здесь перед входящей в строку кавычкой необходимо добавить обратный слеш — `\'` — иначе она бы обозначала окончание строки.

Разумеется, требование экранировать относится только к таким же кавычкам, как те, в которые заключена строка. Так что мы можем применить и более элегантное решение, использовав для этой строки двойные или обратные кавычки:

```
1  alert( `I'm the Walrus!` ); // I'm the Walrus!
```

Заметим, что обратный слеш `\` служит лишь для корректного прочтения строки интерпретатором, но он не записывается в строку после её прочтения. Когда строка сохраняется в оперативную память, в неё не добавляется символ `\`. Вы можете явно видеть это в выводах `alert` в примерах выше.

Но что, если нам надо добавить в строку собственно сам обратный слеш `\`?

Это можно сделать, добавив перед ним... ещё один обратный слеш!

```
1  alert( `The backslash: \\` ); // The backslash: \
```

Длина строки

Свойство `length` содержит длину строки:

```
1  alert( `My\n`.length ); // 3
```

Обратите внимание, `\n` — это один спецсимвол, поэтому тут всё правильно: длина строки `3`.

> **`length` — это свойство**
>
> Бывает так, что люди с практикой в других языках случайно пытаются вызвать его, добавляя круглые скобки: они пишут `str.length()` вместо `str.length`. Это не работает.
>
> Так как `str.length` — это числовое свойство, а не функция, добавлять скобки не нужно.

Доступ к символам

Получить символ, который занимает позицию `pos`, можно с помощью квадратных скобок: `[pos]`. Также можно использовать метод str.at(pos). Первый символ занимает нулевую позицию:

```
1  let str = `Hello`;
2
3  // получаем первый символ
4  alert( str[0] ); // H
```

```
alert( str.at(0) ); // H

// получаем последний символ
alert( str[str.length - 1] ); // o
alert( str.at(-1) ); // o
```

Как вы можете видеть, преимущество метода `.at(pos)` заключается в том, что он допускает отрицательную позицию. Если `pos` – отрицательное число, то отсчет ведется от конца строки.

Таким образом, `.at(-1)` означает последний символ, а `.at(-2)` – тот, что перед ним, и т.д.

Квадратные скобки всегда возвращают `undefined` для отрицательных индексов. Например:

```
let str = `Hello`;

alert( str[-2] ); // undefined
alert( str.at(-2) ); // l
```

Также можно перебрать строку посимвольно, используя `for..of` :

```
for (let char of "Hello") {
  alert(char); // H,e,l,l,o (char — сначала "H", потом "e", потом "l" и т.д.)
}
```

Строки неизменяемы

Содержимое строки в JavaScript нельзя изменить. Нельзя взять символ посередине и заменить его. Как только строка создана — она такая навсегда.

Давайте попробуем так сделать, и убедимся, что это не работает:

```
let str = 'Hi';

str[0] = 'h'; // ошибка
alert( str[0] ); // не работает
```

Можно создать новую строку и записать её в ту же самую переменную вместо старой.

Например:

```
let str = 'Hi';

str = 'h' + str[1]; // заменяем строку

alert( str ); // hi
```

В последующих разделах мы увидим больше примеров.

Изменение регистра

Методы toLowerCase() и toUpperCase() меняют регистр символов:

```
alert( 'Interface'.toUpperCase() ); // INTERFACE
alert( 'Interface'.toLowerCase() ); // interface
```

Если мы захотим перевести в нижний регистр какой-то конкретный символ:

```
alert( 'Interface'[0].toLowerCase() ); // 'i'
```

Поиск подстроки

Существует несколько способов поиска подстроки.

str.indexOf

Первый метод — str.indexOf(substr, pos).

Он ищет подстроку `substr` в строке `str`, начиная с позиции `pos`, и возвращает позицию, на которой располагается совпадение, либо `-1` при отсутствии совпадений.

Например:

```
let str = 'Widget with id';

alert( str.indexOf('Widget') ); // 0, потому что подстрока 'Widget' найдена в
alert( str.indexOf('widget') ); // -1, совпадений нет, поиск чувствителен к ре

alert( str.indexOf("id") ); // 1, подстрока "id" найдена на позиции 1 (..idget
```

Необязательный второй аргумент позволяет начать поиск с определённой позиции.

Например, первое вхождение `"id"` — на позиции `1`. Для того, чтобы найти следующее, начнём поиск с позиции `2`:

```
let str = 'Widget with id';

alert( str.indexOf('id', 2) ) // 12
```

Чтобы найти все вхождения подстроки, нужно запустить `indexOf` в цикле. Каждый раз, получив очередную позицию, начинаем новый поиск со следующей:

```
let str = 'Ослик Иа-Иа посмотрел на виадук';

let target = 'Иа'; // цель поиска

let pos = 0;
```

```
6  while (true) {
7    let foundPos = str.indexOf(target, pos);
8    if (foundPos == -1) break;
9
10   alert( `Найдено тут: ${foundPos}` );
11   pos = foundPos + 1; // продолжаем со следующей позиции
12 }
```

Тот же алгоритм можно записать и короче:

```
1  let str = "Ослик Иа-Иа посмотрел на виадук";
2  let target = "Иа";
3
4  let pos = -1;
5  while ((pos = str.indexOf(target, pos + 1)) != -1) {
6    alert( pos );
7  }
```

str.lastIndexOf(substr, position)

Также есть похожий метод str.lastIndexOf(substr, position), который ищет с конца строки к её началу.

Он используется тогда, когда нужно получить самое последнее вхождение: перед концом строки или начинающееся до (включительно) определённой позиции.

При проверке `indexOf` в условии `if` есть небольшое неудобство. Такое условие не будет работать:

```
1  let str = "Widget with id";
2
3  if (str.indexOf("Widget")) {
4      alert("Совпадение есть"); // не работает
5  }
```

Мы ищем подстроку `"Widget"`, и она здесь есть, прямо на позиции `0`. Но `alert` не показывается, т. к. `str.indexOf("Widget")` возвращает `0`, и `if` решает, что тест не пройден.

Поэтому надо делать проверку на `-1`:

```
1  let str = "Widget with id";
2
3  if (str.indexOf("Widget") != -1) {
4      alert("Совпадение есть"); // теперь работает
5  }
```

Трюк с побитовым НЕ

Существует старый трюк с использованием побитового оператора НЕ — `~`. Он преобразует число в 32-разрядное целое со знаком (signed 32-bit integer). Дробная часть, в случае, если она присутствует, отбрасывается. Затем все биты числа инвертируются.

На практике это означает простую вещь: для 32-разрядных целых чисел значение `~n` равно `-(n+1)`.

В частности:

```
alert( ~2 ); // -3, то же, что -(2+1)
alert( ~1 ); // -2, то же, что -(1+1)
alert( ~0 ); // -1, то же, что -(0+1)
alert( ~-1 ); // 0, то же, что -(-1+1)
```

Таким образом, `~n` равняется 0 только при `n == -1` (для любого `n`, входящего в 32-разрядные целые числа со знаком).

Соответственно, прохождение проверки `if ( ~str.indexOf("…") )` означает, что результат `indexOf` отличен от `-1`, совпадение есть.

Это иногда применяют, чтобы сделать проверку `indexOf` компактнее:

```
let str = "Widget";

if (~str.indexOf("Widget")) {
  alert( 'Совпадение есть' ); // работает
}
```

Обычно использовать возможности языка каким-либо неочевидным образом не рекомендуется, но этот трюк широко используется в старом коде, поэтому его важно понимать.

Просто запомните: `if (~str.indexOf(…))` означает «если найдено».

Впрочем, если быть точнее, из-за того, что большие числа обрезаются до 32 битов оператором `~`, существуют другие числа, для которых результат тоже будет `0`, самое маленькое из которых — `~4294967295=0`. Поэтому такая проверка будет правильно работать только для строк меньшей длины.

На данный момент такой трюк можно встретить только в старом коде, потому что в новом он просто не нужен: есть метод `.includes` (см. ниже).

includes, startsWith, endsWith

Более современный метод str.includes(substr, pos) возвращает `true`, если в строке `str` есть подстрока `substr`, либо `false`, если нет.

Это — правильный выбор, если нам необходимо проверить, есть ли совпадение, но позиция не нужна:

```
alert( "Widget with id".includes("Widget") ); // true

alert( "Hello".includes("Bye") ); // false
```

Необязательный второй аргумент `str.includes` позволяет начать поиск с определённой позиции:

```
alert( "Midget".includes("id") ); // true
alert( "Midget".includes("id", 3) ); // false, поиск начат с позиции 3
```

Методы str.startsWith и str.endsWith проверяют, соответственно, начинается ли и заканчивается ли строка определённой строкой:

```
alert( "Widget".startsWith("Wid") ); // true, "Wid" — начало "Widget"
alert( "Widget".endsWith("get") ); // true, "get" — окончание "Widget"
```

Получение подстроки

В JavaScript есть 3 метода для получения подстроки: `substring`, `substr` и `slice`.

`str.slice(start [, end])`

Возвращает часть строки от `start` до (не включая) `end`.

Например:

```
let str = "stringify";
// 'strin', символы от 0 до 5 (не включая 5)
alert( str.slice(0, 5) );
// 's', от 0 до 1, не включая 1, т. е. только один символ на позиции 0
alert( str.slice(0, 1) );
```

Если аргумент `end` отсутствует, `slice` возвращает символы до конца строки:

```
let str = "stringify";
alert( str.slice(2) ); // ringify, с позиции 2 и до конца
```

Также для `start/end` можно задавать отрицательные значения. Это означает, что позиция определена как заданное количество символов *с конца строки*:

```
let str = "stringify";

// начинаем с позиции 4 справа, а заканчиваем на позиции 1 справа
alert( str.slice(-4, -1) ); // gif
```

`str.substring(start [, end])`

Возвращает часть строки *между* `start` и `end` (не включая) `end`.

Это — почти то же, что и `slice`, но можно задавать `start` больше `end`.
Если `start` больше `end`, то метод `substring` сработает так, как если бы аргументы были поменяны местами.

Например:

```
let str = "stringify";

// для substring эти два примера — одинаковы
alert( str.substring(2, 6) ); // "ring"
alert( str.substring(6, 2) ); // "ring"
```

```
6
7  // …но не для slice:
8  alert( str.slice(2, 6) ); // "ring" (то же самое)
9  alert( str.slice(6, 2) ); // "" (пустая строка)
```

Отрицательные значения `substring`, в отличие от `slice`, не поддерживает, они интерпретируются как `0`.

`str.substr(start [, length])`

Возвращает часть строки от `start` длины `length`.

В противоположность предыдущим методам, этот позволяет указать длину вместо конечной позиции:

```
1  let str = "stringify";
2  // ring, получаем 4 символа, начиная с позиции 2
3  alert( str.substr(2, 4) );
```

Значение первого аргумента может быть отрицательным, тогда позиция определяется с конца:

```
1  let str = "stringify";
2  // gi, получаем 2 символа, начиная с позиции 4 с конца строки
3  alert( str.substr(-4, 2) );
```

Этот метод находится в Annex B спецификации языка. Это означает, что его должны поддерживать только браузерные движки JavaScript, и использовать его не рекомендуется. Но на практике он поддерживается везде.

Давайте подытожим, как работают эти методы, чтобы не запутаться:

метод	выбирает…	отрицательные значения
`slice(start, end)`	от `start` до `end` (не включая `end`)	можно передавать отрицательные значения
`substring(start, end)`	между `start` и `end`	отрицательные значения равнозначны `0`
`substr(start, length)`	`length` символов, начиная от `start`	значение `start` может быть отрицательным

> **Какой метод выбрать?**
>
> Все эти методы эффективно выполняют задачу. Формально у метода `substr` есть небольшой недостаток: он описан не в собственно спецификации JavaScript, а в приложении к ней — Annex B. Это приложение описывает возможности языка для использования в браузерах, существующие в основном по историческим причинам. Таким образом, в другом окружении, отличном от браузера, он может не поддерживаться. Однако на практике он работает везде.
>
> Из двух других вариантов, `slice` более гибок, он поддерживает отрицательные аргументы, и его короче писать. Так что, в принципе, можно запомнить только его.

Сравнение строк

Как мы знаем из главы Операторы сравнения, строки сравниваются посимвольно в алфавитном порядке.

Тем не менее, есть некоторые нюансы.

1. Строчные буквы больше заглавных:

```
alert( 'a' > 'Z' ); // true
```

2. Буквы, имеющие диакритические знаки, идут «не по порядку»:

```
alert( 'Österreich' > 'Zealand' ); // true
```

Это может привести к своеобразным результатам при сортировке названий стран: нормально было бы ожидать, что `Zealand` будет после `Österreich` в списке.

Чтобы разобраться, что происходит, давайте ознакомимся с внутренним представлением строк в JavaScript.

Строки кодируются в UTF-16. Таким образом, у любого символа есть соответствующий код. Есть специальные методы, позволяющие получить символ по его коду и наоборот.

str.codePointAt(pos)

Возвращает код для символа, находящегося на позиции `pos` :

```
// одна и та же буква в нижнем и верхнем регистре
// будет иметь разные коды
alert( "z".codePointAt(0) ); // 122
alert( "Z".codePointAt(0) ); // 90
```

String.fromCodePoint(code)

Создаёт символ по его коду `code`

```
alert( String.fromCodePoint(90) ); // Z
```

Давайте сделаем строку, содержащую символы с кодами от `65` до `220` — это латиница и ещё некоторые распространённые символы:

```
let str = '';

for (let i = 65; i <= 220; i++) {
  str += String.fromCodePoint(i);
}
alert( str );
// ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
// ¡¢£¤¥¦§¨©ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜ
```

Как видите, сначала идут заглавные буквы, затем несколько спецсимволов, затем строчные и `Ö` ближе к концу вывода.

Теперь очевидно, почему `a > Z` .

Символы сравниваются по их кодам. Больший код — больший символ. Код `a` (97) больше кода `Z` (90).

- Все строчные буквы идут после заглавных, так как их коды больше.
- Некоторые буквы, такие как `Ö`, вообще находятся вне основного алфавита. У этой буквы код больше, чем у любой буквы от `a` до `z`.

Правильное сравнение

«Правильный» алгоритм сравнения строк сложнее, чем может показаться, так как разные языки используют разные алфавиты.

Поэтому браузеру нужно знать, какой язык использовать для сравнения.

К счастью, все современные браузеры (для IE10− нужна дополнительная библиотека Intl.JS) поддерживают стандарт ECMA 402, обеспечивающий правильное сравнение строк на разных языках с учётом их правил.

Для этого есть соответствующий метод.

Вызов str.localeCompare(str2) возвращает число, которое показывает, какая строка больше в соответствии с правилами языка:

- Отрицательное число, если `str` меньше `str2`.
- Положительное число, если `str` больше `str2`.
- `0`, если строки равны.

Например:

```
1  alert( 'Österreich'.localeCompare('Zealand') ); // -1
```

У этого метода есть два дополнительных аргумента, которые указаны в документации. Первый позволяет указать язык (по умолчанию берётся из окружения) — от него зависит порядок букв. Второй — определить дополнительные правила, такие как чувствительность к регистру, а также следует ли учитывать различия между `"a"` и `"á"`.

Итого

- Есть три типа кавычек. Строки, использующие обратные кавычки, могут занимать более одной строки в коде и включать выражения `${…}`.
- Строки в JavaScript кодируются в UTF-16.
- Есть специальные символы, такие как разрыв строки `\n`.
- Для получения символа используйте `[]` или метод `at`.
- Для получения подстроки используйте `slice` или `substring`.
- Для того, чтобы перевести строку в нижний или верхний регистр, используйте `toLowerCase/toUpperCase`.
- Для поиска подстроки используйте `indexOf` или `includes/startsWith/endsWith`, когда надо только проверить, есть ли вхождение.
- Чтобы сравнить строки с учётом правил языка, используйте `localeCompare`.

Строки также имеют ещё кое-какие полезные методы:

- `str.trim()` — убирает пробелы в начале и конце строки.
- `str.repeat(n)` — повторяет строку `n` раз.
- …и другие, которые вы можете найти в справочнике.

Для строк предусмотрены методы для поиска и замены с использованием регулярных выражений. Но это отдельная большая тема, поэтому ей посвящена отдельная глава учебника Регулярные выражения.

Также, на данный момент важно знать, что строки основаны на кодировке Юникод, и поэтому иногда могут возникать проблемы со сравнениями. Подробнее о Юникоде в главе Юникод, внутреннее устройство строк.

Задачи

Сделать первый символ заглавным

важность: 5

Напишите функцию `ucFirst(str)`, возвращающую строку `str` с заглавным первым символом. Например:

```
ucFirst("вася") == "Вася";
```

Открыть песочницу с тестами для задачи.

решение

Мы не можем просто заменить первый символ, так как строки в JavaScript неизменяемы.

Но можно пересоздать строку на основе существующей, с заглавным первым символом:

```
let newStr = str[0].toUpperCase() + str.slice(1);
```

Однако есть небольшая проблемка. Если строка пуста, `str[0]` вернёт `undefined`, а у `undefined` нет метода `toUpperCase()`, поэтому мы получим ошибку.

Выхода два:

1. Использовать `str.charAt(0)`, поскольку этот метод всегда возвращает строку (для пустой строки — пустую).
2. Добавить проверку на пустую строку.

Вот второй вариант:

```
function ucFirst(str) {
  if (!str) return str;

  return str[0].toUpperCase() + str.slice(1);
}

alert( ucFirst("вася") ); // Вася
```

Открыть решение с тестами в песочнице.

Проверка на спам

важность: 5

Напишите функцию `checkSpam(str)`, возвращающую `true`, если `str` содержит `'viagra'` или `'XXX'`, а иначе `false`.

Функция должна быть нечувствительна к регистру:

```
checkSpam('buy ViAgRA now') == true
checkSpam('free xxxxx') == true
checkSpam("innocent rabbit") == false
```

Открыть песочницу с тестами для задачи.

решение

Для поиска без учёта регистра символов переведём всю строку в нижний регистр, а потом проверим, есть ли в ней искомые подстроки:

```
function checkSpam(str) {
  let lowerStr = str.toLowerCase();

  return lowerStr.includes('viagra') || lowerStr.includes('xxx');
}

alert( checkSpam('buy ViAgRA now') );
alert( checkSpam('free xxxxx') );
alert( checkSpam("innocent rabbit") );
```

Открыть решение с тестами в песочнице.

Усечение строки

важность: 5

Создайте функцию `truncate(str, maxlength)`, которая проверяет длину строки `str` и, если она превосходит `maxlength`, заменяет конец `str` на `"…"`, так, чтобы её длина стала равна `maxlength`.

Результатом функции должна быть та же строка, если усечение не требуется, либо, если необходимо, усечённая строка.

Например:

```
truncate("Вот, что мне хотелось бы сказать на эту тему:", 20) = "Вот, что мне

truncate("Всем привет!", 20) = "Всем привет!"
```

Открыть песочницу с тестами для задачи.

решение

Строка, которую мы возвращаем, должна быть не длиннее `maxlength`, поэтому, если мы обрезаем строку, то мы должны убрать на один символ больше, чем `maxlength` — чтобы хватило места на многоточие.

Имейте в виду, что в качестве многоточия здесь используется `…` — ровно один специальный Юникодный символ. Это не то же самое, что `...` — три точки.

```
function truncate(str, maxlength) {
  return (str.length > maxlength) ?
    str.slice(0, maxlength - 1) + '…' : str;
}
```

Открыть решение с тестами в песочнице.

Выделить число

важность: 4

Есть стоимость в виде строки `"$120"`. То есть сначала идёт знак валюты, а затем – число.

Создайте функцию `extractCurrencyValue(str)`, которая будет из такой строки выделять числовое значение и возвращать его.

Например:

```
alert( extractCurrencyValue('$120') === 120 ); // true
```

Открыть песочницу с тестами для задачи.

решение

```
function extractCurrencyValue(str) {
  return +str.slice(1);
}
```

Открыть решение с тестами в песочнице.

Предыдущий урок

Следующий урок

Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите на GitHub.
- Для одной строки кода используйте тег `<code>` , для нескольких строк кода — тег `<pre>` , если больше 10 строк — ссылку на песочницу (plnkr, JSBin, codepen...)
- Если что-то непонятно в статье — пишите, что именно и с какого места.

adriian

9 часов назад edited

task 2 - проверка на спам

```
function checkSpam(str) {
  return str.toLowerCase().includes('viagra') ||
    str.toLowerCase().includes('xxx') ;
}

checkSpam('buy ViAgRA now') == true
checkSpam('free xxxxx') == true
checkSpam("innocent rabbit") == false
```

task 3 - выделить число

```
let str = "$120";

function extractCurrencyValue(str) {
        let parse = parseInt(str.slice(1));
        return parse;

}

console.log( extractCurrencyValue('$120') === 120 );
```


adriian

10 часов назад

task 1 - Сделать первый символ заглавным

```
function ucFirst(str) {
        str = str.at(0).toUpperCase() + str.slice(1);
        console.log(str)
        return str;

}
```

```
console.log(ucFirst("вася") == "Вася");
```


5 дней назад

Я хочу рассказать вам о своем пути в IT-сфере. Давным-давно я мечтал стать разработчиком, но не знал, с чего начать. И тогда я обнаружил этот крутой сайт, который помог мне начать этот путь. 🤓

Я был так *вдохновлен этим обучением*, что создал **телеграм-канал "Джун на фронте"**, где записываю успехи. Если вы тоже мечтаете о карьере в разработке или дизайне, то присоединяйтесь к нам. 🚀

0 0 Ответить • Поделиться ›

Джун на фронте

5 дней назад

👋 Привет, друзья!

Хочу поделиться с вами своим путем в IT-сферу. Ровно год назад я мечтал стать разработчиком, но не знал, как начать. И тогда я узнал об этом сайте. 🤓

И вот, я так вдохновился этим обучением, что создал **телеграм-канал "Джун на фронте"**. Где я записываю свои успехи и опыт.

Если вы тоже мечтаете о карьере в разработке или дизайне, то присоединяйтесь. 🚀

0 0 Ответить • Поделиться ›

JuranTouran

18 дней назад

function truncate(str, maxlength) {

if (str.length > maxlength) {

return str.slice(0, maxlength - 1) + "..." ;
} else {
return str}
}

0 0 Ответить • Поделиться ›

JuranTouran

19 дней назад

function checkSpam(str) {
let strtoLowerCase = str.toLowerCase();
if (strtoLowerCase.indexOf('viagra') > -1) {
return true;
} else if (strtoLowerCase.indexOf('xxx') > -1) {
return true;
} else {

```
return false;
}
}
```

0 0 Ответить • Поделиться ›

Kate One
месяц назад

Всем привет! Мое решение на 3-ю задачу:

```
function truncate(str, maxlength) {
str = str.slice(0, maxlength) + '...';
return str;
};
```

0 0 Ответить • Поделиться ›

DemiMurych
месяц назад

Все что описано выше, имеет отношение к обьекту
созданному при помощи конструктора String,
и не имеет никакого отношения к типу String.

Это имеет принципиальное значение потому,
что все описанные методы переопределяемы, как следствие
вызвав str.substr(1) совсем не обязательно Вы увидите
дейтствительно часть строки с 1 символа, но увидите то, что
определено методом в прототипе обьекта String.

И так далее и тому подобное.

0 0 Ответить • Поделиться ›

Никита Чернышёв → DemiMurych
9 дней назад

```
alert( dolboyeb("DemiMurych") ); // true
```

0 0 Ответить • Поделиться ›

KingNut Academy
месяц назад

Разбор задач, тестов и всего что поможет при прохождении собеседования на должность frontend developer (html, css, js, ts, react). Заходи в ТГ @interview_masters

0 0 Ответить • Поделиться ›

Джун на фронте
месяц назад

Пссс чувачки, JS надо?!

Я смогу его дать в **тг блоге Джун на фронте**! Там вас ждет познавательный контент который я поглощаю в течении дня, мой прогресс в обучении и вакансии из мира разработки.

С уважением, Юрий...

0 0 Ответить • Поделиться ›

V **Vlofer**
месяц назад

function truncate(str, maxlength) {
if (str.length > maxlength) {
str = str.slice(0, maxlength - 1) + '…';
}
return str;
}
console.log(truncate("maximka", 3));
Почему лучше не сделать так? Через конструктор if все понитнее, это задача 3

0 0 Ответить • Поделиться ›

F **Furax**
2 месяца назад

```
const spamWords = ['viagra', 'xxx'];
function checkSpam(str) {
  return spamWords.some(w => str.toLowerCase().includes(w))
}
```

2 0 Ответить • Поделиться ›

C **Станислав Лапшин**
2 месяца назад

Вот интересно у написателя этих толмутов было бы поинтересоваться: А почему я только в решении к задаче узнаю о том что .slice() может превращать строку в число? Где это в примерах? Где упор на гибкость языка и методов?
Вы главу потратили на никому неинтересный отступ в скобочках и наименования, но про действительно важные вещи вы не написали.

0 0 Ответить • Поделиться ›

P **pechenka** → Станислав Лапшин
2 месяца назад

Что ты только что высрал? .slice() не превращает строку в число

2 0 Ответить • Поделиться ›

J **JabaDushnila**
3 месяца назад edited

песочек слишком специфичен. Ругается постоянно на рабочие варианты. В частнсти, в последнем задании даже typeof верно передается , число выводится, а песочек утверждает, что нет. Ругается даже на решение из учебника

0 0 Ответить • Поделиться ›

0 0 Ответить • Поделиться ›

александр ковалев
3 месяца назад

function extractCurrencyValue(str) {

return isNaN(parseInt(str)) ? parseInt(str.slice(1)) : parseInt(str);

}
числовое значение если знак валюты в начале либо в конце

1 0 Ответить • Поделиться ›

Apple_Gold
3 месяца назад

Во втором задании этот вариант мокка не пропускает.

0 0 Ответить • Поделиться ›

Владимир → Apple_Gold
3 месяца назад

Выражения str.includes('xxx') и str.includes('viagra') уже сами по себе возвращают true или false. И в результате может получиться такое: false || true ? true : false.

0 0 Ответить • Поделиться ›

Александр Балков → Apple_Gold
3 месяца назад

Ты забыл добавить return.

1 0 Ответить • Поделиться ›

Загрузить ещё комментарии