Семестр 2, четверть 2. **Урок 4.**

**Регулярные выражения**

|  |  |
| --- | --- |
| 4.1. Через конструктор (выражение):  найти индекс символа / буквы слова(с учетом регистра) |  |
| 4.2. Через конструктор (выражение, флаг **i**):  найти факт наличия символа /слова(с любым регистром) |  |
| 4.2.1 Через конструктор (выражение, флаги **‘ig’**):  найти факт вхождения искомого символа(глобальный поиск) |  |
| 4.3. **match** (найти)  Вариант 1: простой поиск (выдаст «пусто»)  Вариант 2 : добавляем два флага (i - убираем регистрозависимость, g – глобальное |  |
| 4.5. **replace** (найти и заменить)  (регулярка для замены одного / всех найденных элементов ) |  |
| 4.6. Сохранить совпадения и построить из них новую строку |  |
| 4.7. Классы (\d ... \W... ), квантификаторы ({m,n}, {,n}... ) , сокращения (+, ? ...) |  |
| 4.8. Практика |  |
| 4.9. Методы со стороны регулярных выражений  Метод **test**  Метод **exec** |  |
| 4.10. Написание поисковой строки и поискового запроса через new RegExp (selfmade) |  |

Урок№4\_\_Разбор кода урока (RegExp)

**4.1. Черезконструктор (выражение):**

**найти индекс символа / буквы слова(с учетом регистра)**

|  |  |  |
| --- | --- | --- |
| let regExp = **new**RegExp('выражение', 'флаги') | | 1 способ написания RegExp (конструктор) |
| let regExp = /expression/flags | | 2йспособ написания RegExp (ручной) |
| **Пример №1: (конструктор)**  letstr = 'Язык JavaScript популярен как язык Java';  let regExp = new RegExp('java');  console.log(str.**search**(regExp));  console.log(regExp) | Задано значение строки str = ‘…. ….’  Важен Регистр искомого слова!  ПОИСК ВХОЖДЕНИЙ  **// -1**  т.е. индекс = [-1]  нет такого значения  / java / | |
| **Пример №2: (конструктор)**  letstr = 'Язык JavaScript популярен как язык Java';  let regExp = new RegExp('**script**');  console.log(str.**search**(regExp));  console.log(regExp) | Строка letstr = …  Важен Регистр  letregExp = /**J**ava /  ПОИСК ВХОЖДЕНИЙ  **// 9**  т.е. индекс = [9]  индекс первого найденного вхождения  **/ Java /** | |

**4.2. Черезконструктор (выражение, флагi):**

**найти факт наличия символа /слова(с любым регистром)**

флаг **'i'** - регистронезависимость

|  |  |  |
| --- | --- | --- |
| let regExp = **new**RegExp('выражение', 'флаги') | | 1 способ написания RegExp (конструктор) |
| letstr = 'Язык JavaScript популярен как язык Java';  **let regExp = new RegExp ( 'JaVa' , 'i');**  console.log(str.**search**(regExp));  console.log(regExp) | значение строки  newRegExp (  'агрумент 1' , 'аргумент 2')  // **5**  **/JaVa/i**  Регулярка / флаг i | |

**4.2.1 Через конструктор (выражение, флаги‘ig’):**

**найти факт вхождения искомого символа(глобальный поиск)**

флаг **g**–global (глобальный поиск)

|  |  |
| --- | --- |
| letstr = 'Язык JavaScript популярен как язык Java';  **let regExp = new RegExp ( 'JaVa' , 'ig');**  console.log(str.**search**(regExp));  console.log(regExp) | // **5**  **/JaVa/gi** |

Словарик:

// Флаги (дают почти один и тот же результат)

// i - поиск без учета регистра

// g - множественный поиск

// m - многострочный поиск

**4.3. match (найти)**

**Вариант 1: простой поиск (выдаст «пусто»)**

|  |  |
| --- | --- |
| letstr = 'Язык JavaScript популярен как Java';  let regExp = new RegExp('java');  console.log (str.**match**(/java/ ) ); | Переменная **str**  Переменная **regExp**  // null  возвращает null если ничего не найдено |

**Вариант 2 (более информативный):**

**добавляем два флага (i - убираем регистрозависимость, g – глобальное)**

|  |  |
| --- | --- |
| letstr = 'Язык JavaScript популярен как Java';  let regExp = new RegExp('java');  console.log(str.**match**(/java/i));  console.log (str.**match**(/java/ig ) ); | Переменная **str**  Переменная **regExp**    возвращает массив-объект с подробнойиформацией    Найдены все слова целиком  (массив с количеством найденных вхождений) |

**4.5. replace(найти и заменить)**

**(регулярка для замены одного / всех найденных элементов )**

|  |  |
| --- | --- |
| console.log('+7 (000)000-00-00'.**replace**(**'-', ':'**));  console.log('+7 **(000)**000-00-00'.replace('(000)', '(495)'));  console.log('+7 (000)000-00-00'.replace **( /-/g , ':' ) );** | +7 (000)000**:**00-00  Заменит только первый найденный элемент  +7 **(495)**000-00-00  +7 (000)000**:**00**:**00  Заменит все найденные элементы |

Timecode: 00:34+

**4.6. Сохранить совпадения и построить из них новую строку**

Все совпадения будут записаны во внутренние переменные (каждое совпадение – запишется в переменную $1, $2, $3 ….). Далее эти совпадения можно составлять в новую строку.

Задача: 'Snow, John' переделать в строку JohnSnow

Применение: переформатирование дат …(месяц-день-год -> день месяц год )

|  |  |  |
| --- | --- | --- |
| letname = 'Snow, John';  console.log**(name.replace (/([a-z]+), ([a-z]+) /i, '$2 $1') )**  **([a-z]+)**  диапазон искомых букв в ПЕРВОМ слове (от aдо z)  **+ (квантификатор)** любое кол-во может входить в выборку  В итоге – создастся переменная $1  **([a-z]+)**  диапазон искомых букв во ВТОРОМ слове (от aдо z)  **+**  любое кол-во может входить в выборку  В итоге – создастся переменная $2  /i - убираем регистрозависимость  **, '$2 $1'**  После поиска – поменять создавшиеся переменные местами | Переменная для переделки  ИТОГ:  JohnSnow | |
| console.log(name.replace(/([a-z]+), ([a-z]+)/i, 'Было: $&\nСтало: $2 $1')); | |  |

**4.7. Классы, квантификаторы, сокращения**

**Классы**

любое из чисел

**\d - [0123456789] или [0-9]**

Не одно из чисел (^ - отрицание):

**\D - [^0123456789] или [^0-9]**

Найти любую латинскую букву и любую цифру и нижнее подчеркивание

**\w - [a-zA-Z0-9\_]**

Найти любую букву кириллицы (нет класса для этого правила):

**[а-яА-ЯёЁ]**

Все что НЕ буквы и НЕ нижнее подчеркивание:

**\W**

Пробел, табуляция, символ новой строки

**\s - space, tab, \n**

НЕ \s (… )

**\S**

Граница слова (искомый элемент может или не может граничить с другими)

**\b**

НЕ граница слова

**\B**

**Квантификаторы**

{m} - строго m раз (из скольких букв может состоять слово, максимум)

{m,n} - от m до n раз

{m,} - от m до бесконечности

{,n} - от 0 до n раз (либо нет, либо – до бесконечности)

**Сокращения для Квантификаторов**

+ - это аналог :{1,}

\* - это аналог: {0,} от нуля до бесконечности (типа: \*.exe)

? - аналог: {0,1} искать всё! (найдется- хорошо, не найдется - пофиг)

**4.8. Практика**

**Задача№1.**на replace (найти +замена):

из безобразия +7-(000) 000:00-00 получить

Вариант№1 (по ходу мой):

удалить **replace** «что не цифра» **\D**+ везде**/g**+ убрать пробелы**, ''**

|  |  |
| --- | --- |
| console.log('+7-(000) 000:00-00'.replace(/\D/g, '')); | // 70000000000 |

Вариант№2 (более сложнее):

найти все цифры везде **match(/\d+/g)**и объединитьбез пробела**join('')**

|  |  |
| --- | --- |
| console.log('+7-(000) 000:00-00'.match(/\d+/g).join('')); | // 70000000000 |

**Задача №2.** на match

Отловить все написания слова 'color .. colour'. Исключая ошибочные написания.

Вариант№1

Ищем совпадения:**match**+ везде:/**g+** словить буквуU:**u?**

|  |  |
| --- | --- |
| console.log('color colour'.match(/colo**u?**r/g)); |  |

Вариант№2

|  |  |
| --- | --- |
| console.log('color colourcolotrcoloutr'.match(/colo[ut]?r/g)); |  |

**'color colour colotr coloutr'** – набор слов среди которых ищем

**match (…**/**g )–** ищем совпадения глобально (везде)

**/colo[ut]?r** - варианты, в т.ч. может быть набор [ u, t]: **[ut]?**

**Задача №3.** на границы слова **\b** или **\B**

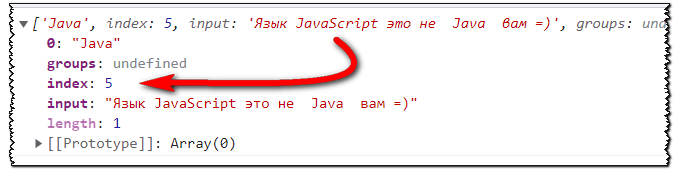
Найти отдельное слово Java (не словосочетание типа JavaScript)

Вариант№1

**\b**(слева – не граница слова) а **\B**(справа – граница слова), т.е.

JavaScript – подпадает под правило /\bjava\B

|  |  |
| --- | --- |
| console.log('ЯзыкJavaScriptэтонеJavaвам =)'.match(/\bjava\B/i)); |  |



**Консоль: нашло JavaScript, проигнорировало отдельное слово Java**

Вариант№2

**\b**(слева – НЕ граница слова) и **\b**(справа – НЕ граница слова), т.е.

Java (index: 24) – подпадает под правило

|  |  |
| --- | --- |
| console.log('ЯзыкJavaScriptэтонеJavaвам =)'.match(/\bjava\b/i)); |  |



**Консоль: нашло отдельное Java, но проигнорировало слово JavaScript**

**Задача №4.1**Задать поиск с начала строки

|  |  |
| --- | --- |
| console.log('JavaScript этонеJava'.match(**/^java/i**)); |  |

Найдет Java (index: 0) в слове JavaScript

НЕ найдет Java(index: 0), если перед этим словом есть другие символы

**Задача №4.2**Задать поиск в конце строки

|  |  |
| --- | --- |
| console.log('JavaScript этонеJava'.match(**/java$/i**)); |  |

Найдет Java (index: 24) в слове Java

НЕ найдет Java(index: 24), если после этого слова есть другие символы

Для чего нужны такие поиски?

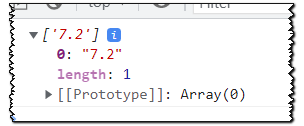
Ответ: для полной валидации строки (формы логин-пароль-почта …) Не понял пояснений.

**Задача №5**Найти все главы в списке глав (Цифра Точка Цифра)

Схема поиска:

Цифра **\d**Точка(с экраном!)**\.**Цифра\d искать везде **/g**

|  |  |
| --- | --- |
| console.log('Chapter 7.2 Chapter 7h2'.match(/\d\.\d/g)); |  |



**Нашлось только 7.2, а 7h2 – пропустило**

**4.9. Методы со стороны регулярных выражений**

Метод test

Где применять: при валидации (есть или нет эта строка, и принять решение)

|  |  |
| --- | --- |
| let str = 'Язык JavaScript популярен как Java';  let regExp = new RegExp( 'java' );    console.log(/java/i.test(str)); | //true  (слово java в этой строке реально есть) |

*или примерный вариант (опытным путем):*

|  |  |
| --- | --- |
| let str = 'Язык JavaScript популярен как Java';  let regExp = new RegExp( 'Java' );    console.log( regExp.test(str) ) | Пример с соблюдением регистра ( нет ‘i’)  //true  (слово Java есть) |

Метод exec

Статистика по КАЖДОМУ найденному вхождению!

|  |  |  |  |
| --- | --- | --- | --- |
| letstr = 'Язык JavaScript популярен как Java';  let regExp = new RegExp('java') | |  | |
| let result;  let regExp = /java/ig; | Место складирования для каждой итерации**result**  Переменная **regExp** | | |
| console.log(`НачальныйиндексlastIndex ${regExp.lastIndex}`);  while(result = regExp.exec(str)) {  console.log(result);  console.log(`ТекущийlastIndex ${regExp.lastIndex}`);  }  console.log(`КонечныйlastIndex ${regExp.lastIndex}`  ); | | | Выдает статистику по …. |

Урок№4\_\_Разбор кода ПОИСКОВОЙ СТРОКИ

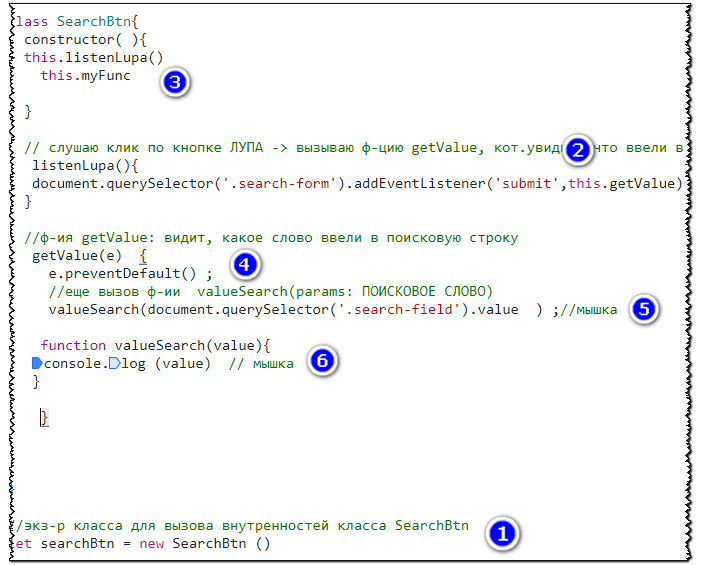
разбор д/з урока №3 (которое .я решал около с 2.02.по 16.02.22)

Разбор создания строки поиска

timecode: 1час 41мин 10+

* 3 минуты: с 01:41:10 до 01:44:00 = работа алгоритма скрытия не попавших под фильтр поиска товаров
* 1 минута с 01:48:30 = вижу, что есть кнопка, которая слушает клик по значку Лупа.

**4.10. Разбираюсь с прокидками функций внутри класса**



**п.1..2..3 – проверка до клика на Лупу**

**п. 4…5..6 – работа кода после ввода слова в строку поиска (ф-ция поиска еще не сделана)**

**4.11. Проба как в консоли виден new RegExp (value, 'i')**

function valueSearch(value){

console.log (value) // мыш

const **regexp** = new RegExp (value, '**i**'); // (мыш / ‘любойРегистр’)

}



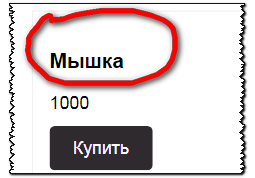
**Консоль: что будет под const regexp**

**Например, regexp.source // ‘мыш’**

**4.12. Как можно указать на тег с именем товара, с которым буду сверять введенное слово в ПОИСКОВОЙ СТРОКЕ**

**// найти все теги в карточке товара, где живет слово изначально**

let descDiv = document.querySelectorAll('.desc') ;



**// вытащить слово через TextContent**

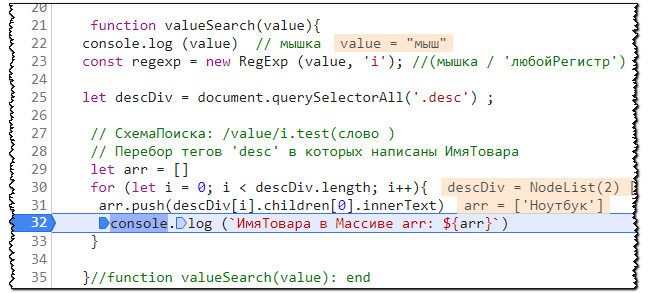
descDiv [1].children[0].textContent

'Мышка'

descDiv [0].children[0].textContent

'Ноутбук'

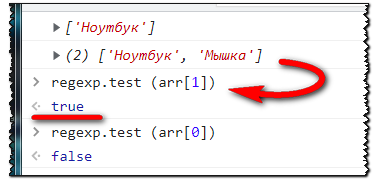
**4.13. Вариант сравнения наличия искомого слова (попытка через массивы)**



**Код (фрагмент с массивом arr )**



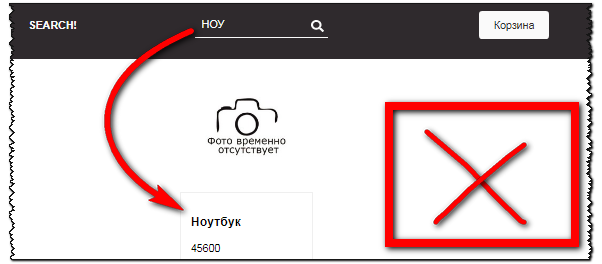
**Ищу по слову «мышка» (важно: с маленькой буквы!) Будет искать и по неполному слову («мыш», «мышк»)**



**Коносоль: regexp.test (arr[1]) дало true/ Т.е. искомое слово с любой буквы / не полное слово - видит**

**4.14. Код ПОИСКА срабатывает, прячет лишние карточкиТовара, но выдает all.js в закладке sources**

|  |  |
| --- | --- |
| **// Скрипт для строки поиска**  class SearchBtn{  constructor( ){  this.listenLupa()  this.myFunc  }  **// слушаю клик по кнопке ЛУПА -> вызываю ф-цию getValue,**  **//кот.увидит, что ввели в поисковую строку**  listenLupa( ){  document.querySelector('.search-form') .addEventListener ('submit',this.getValue)  }  **//ф-ия getValue: видит, какое слово ввели в поисковую строку**  getValue(e) {  e.preventDefault() ;  **//еще вызов ф-ии valueSearch(params: ПОИСКОВОЕ СЛОВО)**  valueSearch(document.querySelector('.search-field').value ) ;//мышка  function valueSearch(value){  console.log (value) **// мышка**  const regexp = new RegExp (value, 'i'); **//(мышка / 'любойРегистр')**    **//div 'desc', где найдем имя Товара (для сравнения с Поиском)**  let descDiv = document.querySelectorAll('.desc') ;  **// СхемаПоиска: /value/i.test(слово )**  let arr = [ ]  **// Перебор тегов 'desc' в которых написаны ИмяТовара**  for (let i = 0; i < descDiv.length; i++){  **//тег с именем Товара помещу в [ ] при помощи arr.push(...)**  arr.push(descDiv[i].children[0].innerText)  console.log (`ИмяТовара в Массиве arr: ${arr}`)  **//если нет искомого слова в данноv div'e, то этот div скрыть**  if (regexp.test(arr[i]) !== true){  console.log (`блок ${arr[i]} скрыть, в нем нет искомого слова`)  **//скрыть div, в кот.нет искомого слова**  descDiv[i].parentElement.style.display = 'none'  } }  }**//function valueSearch(value): end**  } **//getValue(e):end**  }  **//экз-р класса для вызова внутренностей класса SearchBtn**  let searchBtn = new SearchBtn () |  |



Итог поиска по слову «ноу»: показало карточку «Ноутбук»