

2.1 Переменные let

Переменные:

Вызываем `let x = 42`, где `x` – название переменной

2.2 Числа

Литеральная запись числа. – запись любого числа

`let digit = 25;`

`let fract = 25.03;`

`let exp = 2e10 (20000000000);`

`let expSmall = 2e-3 (0.002);`

`let expFract = 1.25e3 (1250);`

`let expFractSmall = 1.25e-3 (0.00125)`

`let fract1 = 25.;`

`let fract2 = -.25;`

Для записи в двоичной системе используем:

`let bin = 0b1001`

Для записи в восьмиричной системе используем:

`let bin = 0o10 (8)`

Для записи в шестнадцаричной системе используем:

`let x = 0x1b; (27)`

2.3 Тренажёр «Переменные»

```
//Исправьте объявление (создание) переменных так, что бы в консоль выводились значения этих переменных, а не ошибки
/*year = 1998;
weight let = 165;
let age = 17;

console.log(year);
console.log(weight);
```

```
console.log(age);
```

У нас есть переменные со значениями, необходимо исправить и вывести значения этих переменных

```
*/
```

```
let year = 1998;
```

```
let weight = 165;
```

```
let age = 17;
```

```
console.log(year);
```

```
console.log(weight);
```

```
console.log(age);
```

2.4 Математические операторы

Let $x = 10 + 2$; // 12

Let $y = 10 - 2$; // 8

Let $z = 10 * 2$; // 20

Let $w = 10 \% 3$; // 1 (остаток)

Для сложных операций

Let $complex = 10 + 5 * 2 - 8 / 2$;

Приоритеты также как и математике (остаток приоритет такой же как и умножение и деление)

Также важные прибавляющие единицу и вычитающие единицу – инкремент и декремент

Инкремент – прибавляющий единицу ($a++$)

Декремент – вычитающий единицу ($a--$), выглядит так:

Let $a = 10$;

$a++$ // (11)

$a--$ // (10)

префиксная запись. Если мы выведем переменную a , то сначала получим старое значение 10 а только потом 11.

То есть если `console.log(a++)`

То получим 10, а только потом 11.

Если хотим сразу значение, то пишем `console.log(++a)` - 11

Более короткая запись операций:

`let x = 5;`

`x = x+5//10` или же

`x += 5 //10`

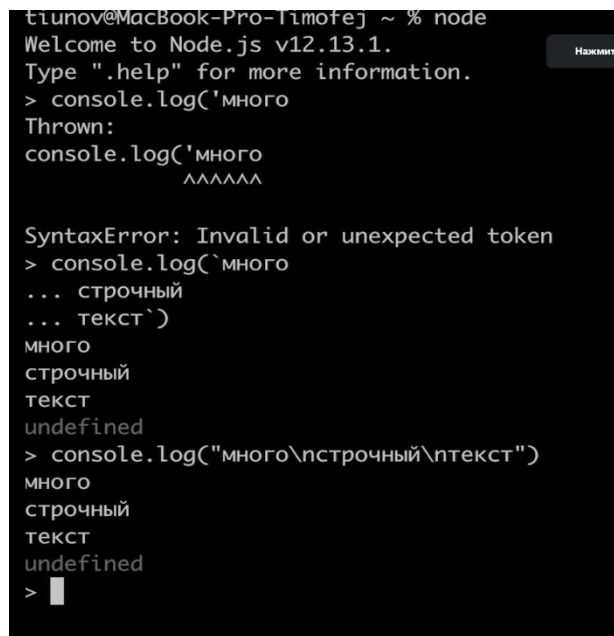
этот же прием работает на все знаки.

3.1 Типы данных

3.2 Строка

Перенос строки в обычных кавычках (обычные и двойные) используем `\n` «какое слово»\

В обратных кавычках переносы произвольные, как напишем, так и отобразится ``...``.



```
tiunov@MacBook-Pro-timofej ~ % node
Welcome to Node.js v12.13.1.
Type ".help" for more information.
> console.log('много
Thrown:
console.log('много
          ^^^^^^

SyntaxError: Invalid or unexpected token
> console.log(`много
... строчный
... текст`)
много
строчный
текст
undefined
> console.log("много\nстрочный\ntекст")
много
строчный
текст
undefined
> █
```

Строки также складываются.

Значение строк в переменных пишем, и просто складываем:

```
Let str1 = ('привет')
```

```
Let str1 = ('привет')
```

```
Let concat = Str1 + str2.
```

Такой процесс называется **конкатинация**.

Также можно складывать и выводить вот так:

```
Let concat = `${str1}
                ${str2}`;
```

Или же

```
Let concat = (`${str1}, ${str2} ! Добро пожаловать`);
```

```
1 console.log('"'Кавычка в кавычках: \'"');
2 console.log('"'Кавычка в кавычках: \'"');
3 console.log(`'Кавычка в кавычках: \'"`);
4 console.log('Символ табуляции: \t');|
5 console.log('Обратный слэш: \\');
6 console.log('\x31'); // цифра 1 в UTF-8 в hex
7 console.log('\u0031'); // цифра 1 в UTF-16 в hex
8 console.log('\u{1F354}'); // символ эмоджи гамбургер, код в UTF-32 в hex
9
```

Какая то хуета. Выводим кавычки и прочую хуйню.

3.4 Boolean

```
JS boolean.js X
boolean > JS boolean.js > ...
1 let Yep = true;
2 let nope = false;
3
4 let yes = 5 > 3;
5 let no = 5 < 3;
6
7 // инвертирование
8 let alsoNo = !yes;
9
10 let x = 10;
11 let y = 20;
12
13 if (x > y) {
14 |   console.log('x больше y');
15 | }
16
17 if (x < y) {
18 |   console.log('x меньше y');
19 | }
20 |
```

3.6 Типы и сравнение

```
JS comparison.js
1 true === true; // true
2 true !== true; // false
3 true === false; // false
4 true !== false; // true
5
6 "строка" === 'строка'; // true
7 'строка' === `строка`; // true
8 "строка" === `строка`; // true
9
10 "строка" === 'строка' === `строка`; // false, важен порядок вычислений
11
12 'строка1' !== 'строка2'; // true
13
14 // === и !== всегда вернёт false при сравнении значений разных типов
15 false !== 0;
16 true !== 1;
17 0 !== '';
18 3 !== '3'; // вот это поворот!
19 false !== '';
20 true !== 'true';
21
22 // сравнение строк происходит посимвольно по кодам символа, условно "по алфавиту"
23 'z' > 'a'; // 122 > 97
24 'az' > 'axzzz'; // a === a, z > a, дальше не проверяем
25 'z' > 'Z'; // 122 > 90
26 '10' < '5'; // вот так сюрприз
27 '10' > '05'; // а вот теперь всё на своих местах
28
```

```
JS comparison.js
22 // сравнение строк происходит посимвольно по кодам символа, условно "по алфавиту"
23 'z' > 'a'; // 122 > 97
24 'az' > 'axzzz'; // a === a, z > a, дальше не проверяем
25 'z' > 'Z'; // 122 > 90
26 '10' < '5'; // вот так сюрприз
27 '10' > '05'; // а вот теперь всё на своих местах
28
29
30 // DANGER ZONE!!! Ниже запрещённая чёрная магия!
31
32 // сравнение строк и чисел
33 '10' > 5; // true
34 10 > '5'; // true
35 10 > 'x'; // false, 'x' не число (NaN)
36 10 < 'x'; // false, 'x' не число (NaN)
37
38 // сравнения с boolean всегда сводятся к сравнению чисел
39 // true - 1, false - 0
40 1 > false; // true, 1 > 0
41 0 < true; // true, 0 < 1
42 '10' > true; // true, 10 > 1
43 '1' > true; // false, 1 > 1
44 '1' > false; // true, 1 > 0
45 'x' > true; // false, NaN > 1
46
```

3.7 Условные операторы

```

JS conditionals.js
1  if (condition) {
2      // код, выполняющийся при соблюдении условия
3  }
4
5
6  if (condition) {
7      // код, выполняющийся при соблюдении условия
8  } else {
9      // код, выполняющийся в противном случае
10 }
11
12
13 if (condition1) {
14     // при первом условии
15 } else if (condition2) {
16     // при втором условии
17 } else if (condition3) {
18     // при третьем условии
19 } else {
20     // если ни одно условие не соблюдено
21 }
22

```

Наконец то дошли до условий.

Первый вариант конструкции условия

```

If (какое ни будь условие, например a>b) {
    Console.log ('a больше b')
    else{
        console.log('b больше a')
    }
}

```

Второй вариант конструкции условия

```

If (какое ни будь условие, например a>b) – первое условие
Else if (второе условие){
    console.log('b больше a')
}
Else if (третье условие){
    console.log('b больше a')
}
else{если никакое условие не подошло}
    console.log('b больше a')
}

```

```

conditionals > JS switch.js > ...
1  let fruit = 'Яблоко';
2
3  switch (fruit) {
4      case 'Яблоко':
5          console.log('Перед нами яблоко');
6          break;
7      case 'Банан':
8          console.log('Перед нами банан');
9          break;
10     case 'Арбуз':
11     case 'Вишня':
12     case 'Клубника':
13         console.log('Это ягода, а не фрукт');
14         break;
15     default:
16         console.log('Не знаю такого фрукта');
17         break;
18 }
19

```

Switch – это чисто сравнения.

Есть переменная, мы ее заводим в switch и с помощью case сравниваем с и выводим какой либо результат

Если условий не много, то используем if else, если условий много, то юзаем switch. Задача:

Задача

Решение

Привет, Оля!

"Исправьте код так, что бы программа запускалась без ошибок и в консоли отображалось сообщение:
Привет, Оля!"

JavaScript

```

1 // Синтаксис переключения в строгий режим всего скрипта
2 "use strict";
3
4 let user = 'Оля';
5
6 if (user === 'Оля') {
7     console.log('Привет, ${user}!');
8 }else{
9     console.log("Здравствуйте!");
10 }

```

ТЕРМИНАЛ

Все юнит-тесты пройдены успешно, поздравляем!

Время выполнения: 0.218325 сек

Общая статистика

Всего тестов: 1. Пройдено: 1. Не пройдено: 0.

Подробную информацию по каждому тесту смотрите ниже.

Тест 1

Заряжено Skillbox & IT Resume

Скрыть терминал

Выполнить

Проверить

Свойства дискриминанта

- если дискриминант меньше нуля $D < 0$ — корней нет
- если дискриминант равен нулю $D = 0$ — есть один корень
- если дискриминант больше нуля $D > 0$ — есть два различных корня

```
//Решим квадратное уравнение с помощью условий

let a = 5;
let b = 10;
let c = 3;

let d = b * b - (4 * a * c);

if (d < 0){
  console.log('Дискриминант < 0, корней нет');
}
else if (d === 0){
  let x1 = -b / 2 * a;
  console.log('Дискриминант = 0, один корень уравнения x1 = ', x1);
}
else if (d > 0){
  let dRoot = Math.sqrt(d);
  let x1 = (-b + dRoot) / (2 * a);
  let x2 = (-b - dRoot) / (2 * a);
  console.log(`Дискриминант равен ${dRoot}, \nесть 2 корня уравнения: \nx1
= ${x1} \nx2 = ${x2}`);
}
```

3.11 Тернарный оператор

Тернарный оператор — это упрощенная версия записи условия if и else.

```
Let x = 0;
If (math.random() > 0.5) {
  X = 10;
}Else {
  X= 20;
}
```


Тоже самое можно записать в одну строку:

`X = Math.random() > 0.5 ? 10 : 20;`

`X = Math.random() > 0.5` – это условие

`?` – условный элемент

`10` – значение, которое присвоится переменной при `try`

`20` – значение переменной, которое присвоится при `else`

Используем только в том случае, когда нам нужно вернуть значение этой переменной!

```
///Тернарный оператор
//Замените конструкция if на тернарный оператор

let randomNumber = 30
console.log(`Произвольное число: ${randomNumber}`)
/*
if (randomNumber > 20) {
  console.log("Число > 20")
} else {
  console.log("Число <= 20")
}*/

let x = randomNumber > 20 ? 'Число > 20' : 'Число <= 20';
console.log(x);
```

3.13 Основы булевой алгебры

Строится на операторах **И** и **ИЛИ**.

При чем оператор И – это `&&`

А оператор ИЛИ – это `||`

```
JS or-and-not.js > ...
1  let day = 'Воскресенье'
2
3  // Суббота или Воскресенье
4  if (day === 'Суббота' || day === 'Воскресенье') {
5    console.log(day + ' – это выходной');
6  } else {
7    console.log(day + ' – это точно не выходной');
8  }
9
10
11 let password = '*****';
12
13 // Длина больше 3 и меньше 26
14 if (password.length > 3 && password.length < 26) {
15   console.log('Допустимый пароль');
16 } else {
17   console.log('Такой пароль не подходит');
18 }
19
20
21 let x = 8;
22
23 // приоритет – сначала &&, потом ||
24 if (x === 13 || x > 0 && x < 11) {
25   console.log('x – число 13 или число от 1 до 10');
26 }
27
28
29 let f = 102; // 123
30
31 // управление приоритетом с помощью скобок
32 if ((f % 3 === 0 || f % 2 === 0) && x > 100) {
33   console.log('f делится на 3 или на 2 и f больше 100');
34 }
35
```

При чем, операции сравнения выполняются в приоритете так также как и в математике, то есть, в булевой алгебре операция И выполняется в приоритете операции ИЛИ.

Также можно искусственно расставлять приоритет с помощью ()

3.15 Практическая работа

Задача 1

Цель задания

Научиться искать подстроку внутри строки и попрактиковаться со сложными условиями.

Что нужно сделать

В переменную `password` запишите строку с любым произвольным паролем. Проверьте надёжность пароля с помощью условного оператора `if`. Пароль является надёжным, когда в нём есть хотя бы четыре символа, один из которых — это дефис или нижнее подчёркивание. Выведите в консоль сообщения «Пароль надёжный» или «Пароль недостаточно надёжный».

Советы и рекомендации

Для проверки наличия в строке другой строки можно воспользоваться конструкцией `password.includes('x')`, где `'x'` — строка для поиска. Поэкспериментируйте с этой командой, посмотрите, что она будет выводить в консоль, попробуйте подставить разные параметры. Это поможет понять принцип её работы.

Проверка результата

Для проверки запустите код с разными вариантами надёжных и ненадёжных паролей.

Примеры корректных паролей:

- 1234-
- 4321_
- qaz-xsw
- _zxd

Примеры некорректных паролей:

- _-a
- qaz
- _-3
- 123456789

Что оценивается

Код корректно выводит сообщение в зависимости от значения переменной `password`.

Задача 2

Цель задания

Узнать, как преобразовывать строку в верхний/нижний регистр и извлекать произвольные куски из строки.

Что нужно сделать

В переменных `userName`, `userSurname` даны имя и фамилия пользователя. При этом в строках беспорядок с большими и маленькими буквами, и нужно оформить строки единообразно. Для этого первые буквы имени и фамилии приведите к верхнему регистру (большие буквы), а оставшиеся — к нижнему (маленькие буквы). Запишите результат в новые переменные и выведите их значения с помощью `console.log`. С помощью тернарных операторов и `console.log` выведите сообщение «Имя было преобразовано» или «Имя осталось без изменений» для имени и фамилии в зависимости от того, были ли исходные строки равны преобразованным.

Проверка результата

Для любых имени и фамилии в любом регистре должны выводиться такие же имя и фамилия, но первая буква у них большая, а все остальные — маленькие.

Что оценивается

Код корректно выводит все сообщения в зависимости от значения переменных `userName` и `userSurname`.

Советы и рекомендации

Для получения куска строки можно воспользоваться конструкцией `str.substring(indexA, indexB)`, где `str` — название переменной с исходной строкой, `indexA` — целое число от нуля до длины строки, указывающее номер символа **начала** куска строки, `indexB` — целое число от нуля до длины строки, указывающее номер символа **конца** куска строки. Если `indexB` не указывать, то вы получите кусок от `indexA` до конца строки. Например, так можно получить первый символ строки: `let first = str.substring(0, 1)`, а так — остаток строки: `let last = str.substring(1)`. Конструкцию можно присвоить переменной, с которой потом можно работать как с обычной строкой. Для преобразования букв строки к верхнему регистру воспользуйтесь конструкцией `str.toUpperCase()`, к нижнему — `str.toLowerCase()`. Результат аналогично можно присвоить переменной.

Задание 3

Цель задания

Научиться проверять числа на чётность.

Что нужно сделать

В переменной `number` записано число. Необходимо с помощью `console.log` вывести сообщение, указывающее на чётность или нечётность числа.

Проверка результата

Для проверки подставляйте различные значения в переменную `number` и оценивайте корректность результата.

Пример:

- `number = 2, console.log("Число чётное");`
- `number = 5, console.log("Число нечётное");`
- `number = 8, console.log("Число чётное");`

Что оценивается

Алгоритм выводит правильный ответ на экран.

Советы и рекомендации

Для проверки числа на чётность можно использовать оператор «Остаток от деления» — `%`. Например, если произвести операцию `5 % 2`, вы получаете 1. Это означает, что число 5 — нечётное. Если использовать другое число: `4 % 2`, в ответе получается 0. Это означает, что число чётное. Попробуйте совместить эту команду с условным оператором.

Сейчас может быть непонятно, как применять эти задачи в реальной практике. Но после изучения базовых команд — инструментов языка программирования мы перейдём к более реальным заданиям и работе с графическими элементами на странице в браузере.

```
console.log(' _____')
console.log('Задача №1 - Пароль')
let password = 'sdc2sa_dfd';

if ((password.length >= 4) && ((password.includes('_')) || (password.includes('-')))){
    console.log(password + ' - Пароль надежный')
}
else{
    console.log(password + ' - Пароль ненадежный');
}

console.log(' _____')
console.log('Задача №2 - Имя пользователя')

let userName = 'даРья';
let userSurname = 'сМанухова';

let fersuserName = userName.substring(0, 1);
let lastuserName = userName.substring(1);

let ferstuserSurname = userSurname.substring(0, 1);
let lastuserSurname = userSurname.substring(1);
```

```

console.log(fersuserName.toUpperCase() + lastuserName.toLowerCase(),
ferstuserSurname.toUpperCase() + lastuserSurname.toLowerCase());

console.log('_____')
console.log('Задача №3 - числа на четность')

let number = 5;

let ziro = (number === 0) ? ('- Число 0 на 2 не делится') : ('Остаток от
деления ' + number % 2);

if ((ziro === 0) || (number === 0)){
    console.log(number + ' - Число четное')
}
else{
    console.log(number + ' - Число не четное')
}

```

4.1 Примитивные типы и массивы

Сложные типы данных – это данные включающие все в себя несколько подтипов типов данных

4.2 Что такое массив

Массив – сложный тип данных, который позволяет хранить множество разных данных в определенном порядке

Let fruit = ['яблоко', 'груша', 'апельсин']; - массив со строками.

Элементы расположены в определенном порядке и каждого есть свой индекс, начиная с лева на право с 0. То есть ['яблоко – (0)', 'груша – (1)', 'апельсин – (2)']

Через консоль можно обращаться к элементам массива:

Let fruit = ['яблоко', 'груша', 'апельсин'];

Fruit[0] – это будет яблоко

Если мы не знаем количество элементов массива, создаем пустой и добавляем туда по готовности элементы с помощью **push – в конец**, **unshift** – в начало:

Let aphavit = [];

Aphavit.push('r')

Aphavit. Unshift('Д') и тд

```

js arrays.js > ...
1  let alphabet = [];
2
3  // добавить в конец массива
4  alphabet.push('Г');
5  alphabet.push('Д');
6  alphabet.push('Е', 'Ж', 'З');
7
8  // добавить в начало
9  alphabet.unshift('В');
10 alphabet.unshift('А', 'Б');
11
12 console.log(alphabet);

```

Также можем узнать длину массива с помощью команды `length`.

```
Let fruit = ['яблоко', 'груша', 'апельсин'];
```

`Fruit.length` – выведет 3 элемента массива.

С помощью `length` можем вывести последний элемент массива:

```
Fruit[Fruit.length - 1] – выведет апельсин.
```

Также с помощью индекса можем изменять элементы массива, если допустили ошибку с помощью `greeting[индекс массива]`:

```
Let fruit = ['яблоко', 'груша', 'апельсин'];
```

```
Fruit[2] = 'арбуз'
```

`Console.log(fruit[2]);` - выведет исправленный массив с арбузом.

4.3 Цикл for

Цикл `for` позволяет выполнять определённые действия n количество раз

```
JS for.js > ...
1  let fibo = [1, 1];
2
3  for (let i = 1; i < 49; ++i) {
4    fibo.push(fibo[i] + fibo[i - 1]);
5  }
6
7  // 1) let i = 1;
8  // 2) i < 49 - 1 < 49
9  // 3) [1, 1, 2]
10 // 4) ++i; i = 2
11 // 5) i < 49 - 2 < 49
```

JS for-parts.js X

JS for-parts.js > ...

```
1  // цикл, который никогда не завершится и повесит вашу программу
2  for (;;) console.log('Я буду сниться тебе в кошмарах');
3
4  // цикл, который может завершиться на любой итерации с вероятностью 10%
5  for (; Math.random() >= 0.1;) console.log('И ещё разок');
6
7  // выносим инициализатор за пределы цикла, а завершающую инструкцию – в тело
8  let i = 10;
9  for (; i > 0;) { console.log('i = ' + i--); }
10
11 // вечный цикл, считающий до бесконечности (на самом деле нет)
12 for (let i = 0;; ++i) { console.log(i); }
13
```

4.4 Циклы for of, for in

For of

Короче

```
JS for-of.js  X
JS for-of.js > ...
1  let fruits = ['Яблоко', 'Банан', 'Апельсин', 'Ананас', 'Дыня'];
2
3  console.log('Сегодня я съел:');
4
5  for (let fruit of fruits) {
6      console.log(fruit);
7  }
8  |
```

В строке №5 переменной fruit поочередно присваивается элемент массива. При каждом выполнении цикла, разный фрукт в переменной fruit.

Цикл for in

```
JS for-in.js > ...
1  let rating = ['Катя', 'Вася', 'Маша', 'Петя', 'Лена'];
2
3  console.log('Рейтинг студентов:');
4
5  for (let i in rating) {
6      console.log(`${parseInt(i) + 1} место: ${rating[i]}`);
7  }
8
9  // 01 11 21|
```

Короче, в переменную i заносится строка (она же одновременно и индекс)

В данном примере сопоставляется имя студента и номер его посадочного места.

В строке №6 мы выводим значение первого элемента массива преобразованного в помощью parseInt в число (0 +1) получаем посадочное место, далее выводим само значение строки и получаем:

1 место: Катя

4.5 Циклы while и do-while

```
JS while.js X
JS while.js > ...
1  let teaVolume = 200;
2
3  console.log('Вы налили себе ' + teaVolume + 'мл чая');
4
5  while (teaVolume > 0) {
6      teaVolume -= 10;
7      console.log('В чашке осталось ' + teaVolume + 'мл чая');
8  }
9  console.log('Вы выпили весь чай');
10 |
```

Пока не выполнится условие, цикл будет повторяться, уменьшая переменную на 10.

```
//цикл while

let varior = 500;

console.log('Вариативный объект', + varior + 'шт.')

while (varior > 0){
    varior = varior - 50;
    console.log('первая итерация', + varior + ' шт');
}

console.log('цикл завершен');
```

цикл do while

То же самое, что и while, за исключением, что условие проверяется после выполнения цикла а не в начале.

```
JS do-while.js X
JS do-while.js
1  // для простоты примера представим, что в пистолете 5 патронов
2  // вероятность выстрела - 20%
3  do {
4      console.log('Нажимаем на курок');
5  } while (Math.random() > 0.2);
6
7  console.log('Похоже, вам не повезло');
8
```


Если выстрел произошел, вам не повезло. Тот же самый принцип, что и while, только не забываем про do в начале.

Используем только в том случае, когда мы уверены, что тело цикла должно выполниться хотя бы 1 раз.

4.6 Операторы continue и break

continue

```
// колода
let cards = ['2', 'Король', 'Туз', '5', '6', 'Король', 'Дама'];
// карты в руке
let hand = [];

for (let card of cards) {
  // выполняем только для нечётного индекса
  if (card !== 'Король' && card !== 'Туз') continue;
  hand.push(card);
  console.log('Карта ' + card + ' добавлена в руку');
}

console.log('Карты в руке', hand);
```

Операция continue – нужна для продолжения выполнения цикла в том случае, который описан выше., как например с колодой карт. Если нам из списка массива не выпадает то, что нам нужно, мы снова проходимся по нему.

С помощью **for of** мы пробегаемся по нашему массиву и записываем в переменную card каждый элемент массива после каждой итерации и далее с помощью if сравниваем его с условием.

break

```
let cards = ['2', 'Туз', 'Король', 'Дама', 'шестерка', 'семерка'];

let found = false;

for (let card of cards){
  console.log(`вытащили карту ${card}`);
  if(card === 'Король'){
    found = true;
    break;
  }
}

console.log(found ? 'Мы нашли короля' : 'Мы ничего не нашли');
```

Не понимаю, почему у меня в переменную card выводит все элементы массива а не поочередно каждый

```
User@DESKTOP-1JOL8GF MINGW64 /c/Git_work/primer/js (main)
$ node main4_6_1.js
вытащили карту 2, Туз, Король, Дама, шестерка, семерка
вытащили карту 2, Туз, Король, Дама, шестерка, семерка
вытащили карту 2, Туз, Король, Дама, шестерка, семерка
вытащили карту 2, Туз, Король, Дама, шестерка, семерка
вытащили карту 2, Туз, Король, Дама, шестерка, семерка
вытащили карту 2, Туз, Король, Дама, шестерка, семерка
Мы нихуя не нашли

User@DESKTOP-1JOL8GF MINGW64 /c/Git_work/primer/js (main)
$ node main4_6_1.js
вытащили карту 2, Туз, Король, Дама, шестерка, семерка
вытащили карту 2, Туз, Король, Дама, шестерка, семерка
вытащили карту 2, Туз, Король, Дама, шестерка, семерка
Мы наши короля

User@DESKTOP-1JOL8GF MINGW64 /c/Git_work/primer/js (main)
$ |
```

```
arrays > JS while-true-break.js > ...
1   let i;
2
3   console.log('Цикл for');
4
5   // цикл for
6   for (i = 0; i < 11; i += 2) {
7     console.log(i);
8   }
9
10  console.log('Цикл while');
11
12  // аналогичный цикл while с условием для завершения в теле цикла
13  i = 0;
14  while (true) {
15    if (i > 10) break;
16    console.log(i);
17    i += 2;
18  }
19
```

4.7 Какой цикл выбрать

```
arrays > JS playground.js > ...
1  // Простое повторение действий по счётчику
2  for (let i = 0; i < 10; ++i) {
3  |   console.log(i);
4  | }
5
6  // Заполнение массива на основе счётчика
7  let a = [];
8  for (let i = 0; i < 10; ++i) {
9  |   a.push(i * i);
10 | }
11
12 // Заполнение массива на основе значений другого массива
13 let b = [];
14 for (let pow2 of a) {
15 |   b.push(pow2 / 2);
16 | }
17
18 // Заполнение пустого массива на основе других данных (длина массива неизвестна)
19 let lines = [];
20 let next;
21 while (next = file.nextLine()) {
22 |   lines.push(next);
23 | }
24
25 // Обработка значений массива
26 for (let line of lines) {
27 |   console.log('Длина строки:', line.length);
28 | }
29
30 // Обработка индексов массива
31 for (let number in lines) {
32 |   console.log(`Длина строки №${number}: ${lines[number].length}`);
33 | }
```

```
// Обработка значений или индексов массива в обратном порядке
let aReversed = [];
for (let i = a.length - 1; i >= 0; --i) {
|   aReversed.push(a[i]);
| }
```

arrays > JS playground.js > ...

```
35 // Обработка значений или индексов массива в обратном порядке
36 let aReversed = [];
37 for (let i = a.length - 1; i >= 0; --i) {
38     aReversed.push(a[i]);
39 }
40
41 // Сложная логика выхода из цикла
42 let currentAttempt = 0;
43 while (currentAttempt++ < 1000) {
44     if (crayfishWhistles()) break;
45 }
46
47 // Обработка нескольких массивов одинаковой длины
48 for (let i in a) {
49     console.log(a[i] + aReversed[i]);
50 }
51 for (let i = 0; i < a.length; ++i) {
52     console.log(a[i] + aReversed[i]);
53 }
54
55 // Цикл со счётчиком и сложной логикой изменения значения счётчика
56 for (let x = 0; x < 100; x += Math.round(Math.random() * 5)) {
57     console.log(x);
58 }
59 |
```