Когда мы передаём объект или массив, в функцию, то ей может понадобиться не объект/массив целиком, а элементы по отдельности.

Деструктурирующее присваивание – это специальный синтаксис, который позволяет «распаковать» массивы или объекты в несколько переменных.

Деструктуризация также прекрасно работает со сложными функциями, которые имеют много параметров, значений по умолчанию и так далее.

**[Деструктуризация массива](https://learn.javascript.ru/destructuring-assignment" \l "destrukturizatsiya-massiva)**

Вот пример деструктуризации массива на переменные:

// у нас есть массив с именем и фамилией

let arr = ["Ilya", "Kantor"];

// деструктурирующее присваивание

// записывает firstName = arr[0]

// и surname = arr[1]

let [firstName, surname] = arr;

alert(firstName); // Ilya

alert(surname); // Kantor

Отлично смотрится в сочетании со split или другими методами, возвращающими массив:

let [firstName, surname] = "Ilya Kantor".split(' ');

alert(firstName); // Ilya

alert(surname); // Kantor

Как вы можете видеть, синтаксис прост. Однако есть несколько странных моментов. Давайте посмотрим больше примеров, чтобы лучше понять это.

**«Деструктурирующее присваивание» не уничтожает массив.** Массив никак не изменяется, его значения просто копируются в переменные. Деструктуризация, это просто короткий вариант записи:

// let [firstName, surname] = arr;

let firstName = arr[0];

let surname = arr[1];

**Можно пропустить элементы при помощи запятой**

Нежелательные элементы массива также могут быть отброшены с помощью дополнительной запятой:

// второй элемент не нужен

let [firstName, , title] = ["Julius", "Caesar", "Consul", "of the Roman Republic"];

alert( title ); // Consul

В примере второй элемент массива пропускается, а третий присваивается переменной title, оставшиеся элементы массива также пропускаются (так как для них нет переменных).

**Работает с любым перебираемым объектом**

Можно использовать любой перебираемый объект, не только массивы:

let [a, b, c] = "abc";

let [one, two, three] = new Set([1, 2, 3]);

**Присваивайте чему угодно**

Можно использовать что угодно «присваивающее» с левой стороны.

Например, можно присвоить свойству объекта:

let user = {};

[user.name, user.surname] = "Ilya Kantor".split(' ');

alert(user.name); // Ilya

alert(user.surname); // Kantor

**Цикл с Object.entries(obj).**

Мы можем использовать его с деструктуризацией для цикличного перебора ключей и значений объекта:

let user = {

name: "John",

age: 30

};

// цикл по ключам и значениям

for (let [key, value] of Object.entries(user)) {

alert(`${key}:${value}`); // name:John, затем age:30

}

…то же самое для map:

let user = new Map();

user.set("name", "John");

user.set("age", "30");

// Map перебирает как пары [ключ, значение], что очень удобно для деструктурирования

for (let [key, value] of user) {

alert(`${key}:${value}`); // name:John, затем age:30

}

**Трюк обмена переменных**

let guest = "Jane";

let admin = "Pete";

// Давайте поменяем местами значения: сделаем guest = "Pete", а admin = "Jane"

[guest, admin] = [admin, guest];

alert(`${guest} ${admin}`); // Pete Jane (успешно заменено!)

Здесь создается временный массив из двух переменных и немедленно деструктурируется в нужном порядке.

Таким образом, мы можем поменять местами даже более двух переменных.

**Остаточные параметры «…»**

Если массив длиннее, чем список слева, «лишние» элементы опускаются. Чтобы не просто получить первые значения, но и собрать все остальные, можно добавить ещё один параметр, который получает остальные значения, используя оператор «остаточные параметры» – троеточие ("..."):

let [name1, name2, ...rest] = ["Julius", "Caesar", "Consul", "of the Roman Republic"];

// rest это массив элементов, начиная с 3-го

alert(rest[0]); // Consul

alert(rest[1]); // of the Roman Republic

alert(rest.length); // 2

Переменная rest является массивом из оставшихся элементов. Переменная для массива должна стоять на последнем месте в деструктурирующем присваивании.

### [Значения по умолчанию](https://learn.javascript.ru/destructuring-assignment" \l "znacheniya-po-umolchaniyu)

Если в массиве меньше значений, чем в присваивании, то ошибки не будет. Отсутствующие значения считаются неопределёнными:

let [firstName, surname] = [];

alert(firstName); // undefined

alert(surname); // undefined

Можно указать значения по умолчанию для отсутствующих элементов.

// значения по умолчанию

let [name = "Guest", surname = "Anonymous"] = ["Julius"];

alert(name); // Julius (из массива)

alert(surname); // Anonymous (значение по умолчанию)

Значения по умолчанию могут быть гораздо более сложными выражениями или даже функциями. Они выполняются, только если значения отсутствуют.

Например, здесь используется prompt для указания двух значений по умолчанию, но будет запущен только для пропущенного значения (surname).

// prompt запустится только для surname

let [name = prompt('name?'), surname = prompt('surname?')] = ["Julius"];

alert(name); // Julius (из массива)

alert(surname); // результат prompt

## [Деструктуризация объекта](https://learn.javascript.ru/destructuring-assignment" \l "destrukturizatsiya-obekta)

Синтаксис:

let {var1, var2} = {var1:…, var2:…}

У нас есть существующий объект с правой стороны, который мы хотим разделить на переменные. Левая сторона содержит «шаблон» для соответствующих свойств. В простом случае это список названий переменных в {...}.

let options = {

title: "Menu",

width: 100,

height: 200

};

let {title, width, height} = options;

alert(title); // Menu

alert(width); // 100

alert(height); // 200

Свойства options.title, options.width и options.height присваиваются соответствующим переменным.

Порядок не имеет значения. Вот так – тоже работает:

// изменён порядок в let {...}

let {height, width, title} = { title: "Menu", height: 200, width: 100 }

Шаблон с левой стороны может быть более сложным и определять соответствие между свойствами и переменными.

Если мы хотим присвоить свойство объекта переменной с другим названием, например, свойство options.width присвоить переменной w, то используется двоеточие.

let options = {

title: "Menu",

width: 100,

height: 200

};

// { sourceProperty: targetVariable }

let {width: w, height: h, title} = options;

// width -> w

// height -> h

// title -> title

alert(title); // Menu

alert(w); // 100

alert(h); // 200

Двоеточие показывает «что : куда идёт». В примере выше свойство width сохраняется в переменную w, свойство height сохраняется в h, а title присваивается одноимённой переменной.

Для потенциально отсутствующих свойств можно установить значения по умолчанию, используя "=":

let options = {

title: "Menu"

};

let {width = 100, height = 200, title} = options;

alert(title); // Menu

alert(width); // 100

alert(height); // 200

Как и в случае с массивами, значениями по умолчанию могут быть любые выражения или даже функции. Они выполнятся, если значения отсутствуют.

Например prompt запросит width, но не title:

let options = {

title: "Menu"

};

let {width = prompt("width?"), title = prompt("title?")} = options;

alert(title); // Menu

alert(width); // (результат prompt)

Мы также можем совмещать : и =:

let options = {

title: "Menu"

};

let {width: w = 100, height: h = 200, title} = options;

alert(title); // Menu

alert(w); // 100

alert(h); // 200

Из большого объекта с множеством свойств, можно взять только то, что нужно:

let options = {

title: "Menu",

width: 100,

height: 200

};

// взять только title, игнорировать остальное

let { title } = options;

alert(title); // Menu

### [Остаток объекта «…»](https://learn.javascript.ru/destructuring-assignment" \l "ostatok-obekta)

Чтобы не просто получить первые значения, но и собрать все остальные, можно добавить ещё один параметр, который получает остальные значения, используя оператор «остаточные параметры» – троеточие ("...").

В некоторых старых браузерах (IE) это не поддерживается, нужен Babel для полифила.

let options = {

title: "Menu",

height: 200,

width: 100

};

// title = свойство с именем title

// rest = объект с остальными свойствами

let {title, ...rest} = options;

// сейчас title="Menu", rest={height: 200, width: 100}

alert(rest.height); // 200

alert(rest.width); // 100

**Обратите внимание на let**

В примерах выше переменные были объявлены в присваивании: let {…} = {…}. Можно использовать существующие переменные и не указывать let, но тут есть подвох.

Вот так не будет работать:

let title, width, height;

// ошибка будет в этой строке

{title, width, height} = {title: "Menu", width: 200, height: 100};

JavaScript обрабатывает {...} в основном потоке кода (не внутри другого выражения) как блок кода. Такие блоки кода могут быть использованы для группировки операторов, например:

{

// блок кода

let message = "Hello";

// ...

alert( message );

}

Чтобы показать JavaScript, что это не блок кода, мы можем заключить выражение в скобки (...):

let title, width, height;

// сейчас всё работает

({title, width, height} = {title: "Menu", width: 200, height: 100});

alert( title ); // Menu

## [Вложенная деструктуризация](https://learn.javascript.ru/destructuring-assignment" \l "vlozhennaya-destrukturizatsiya)

Если объект или массив содержит другие вложенные объекты или массивы, то можно использовать более сложные шаблоны с левой стороны, чтобы извлечь более глубокие свойства.

Например:

let options = {

size: {

width: 100,

height: 200

},

items: ["Cake", "Donut"],

extra: true

};

// деструктуризация разбита на несколько строк для ясности

let {

size: { // положим size сюда

width,

height

},

items: [item1, item2], // добавим элементы к items

title = "Menu" // отсутствует в объекте (используется значение по умолчанию)

} = options;

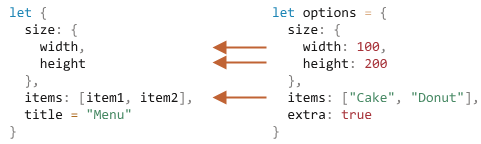
alert(title); // Menu

alert(width); // 100

alert(height); // 200

alert(item1); // Cake

alert(item2); // Donut



## [Умные параметры функций](https://learn.javascript.ru/destructuring-assignment" \l "umnye-parametry-funktsiy) – не закончена

Есть ситуации, когда функция имеет много параметров, большинство из которых не обязательны. Это особенно верно для пользовательских интерфейсов. Например, функция, которая создаёт меню. Она может иметь ширину, высоту, заголовок, список элементов и так далее.

Вот так – плохой способ писать подобные функции:

function showMenu(title = "Untitled", width = 200, height = 100, items = []) {

// ...

}

Проблема заключается в том, как запомнить порядок всех аргументов. Обычно IDE помогают подсказками, особенно если код хорошо документирован, но всё же… Другая проблема заключается в том, как вызвать функцию, когда большинство параметров передавать не надо, и значения по умолчанию вполне подходят.

Разве что вот так?

// undefined там, где подходят значения по умолчанию

showMenu("My Menu", undefined, undefined, ["Item1", "Item2"])

Это выглядит ужасно. И становится нечитаемым, когда мы имеем дело с большим количеством параметров.

На помощь приходит деструктуризация!

Мы можем передать параметры как объект, и функция немедленно деструктурирует его в переменные:

// мы передаём объект в функцию

let options = {

title: "My menu",

items: ["Item1", "Item2"]

};

// ...и она немедленно извлекает свойства в переменные

function showMenu({title = "Untitled", width = 200, height = 100, items = []}) {

// title, items – взято из options,

// width, height – используются значения по умолчанию

alert( `${title} ${width} ${height}` ); // My Menu 200 100

alert( items ); // Item1, Item2

}

showMenu(options);

Можно использовать более сложное деструктурирование с вложенными объектами и двоеточием:

let options = {

title: "My menu",

items: ["Item1", "Item2"]

};

function showMenu({

title = "Untitled",

width: w = 100, // width присваиваем в w

height: h = 200, // height присваиваем в h

items: [item1, item2] // первый элемент items присваивается в item1, второй в item2

}) {

alert( `${title} ${w} ${h}` ); // My Menu 100 200

alert( item1 ); // Item1

alert( item2 ); // Item2

}

showMenu(options);