Деструктурирующее присваивание

Деструктурирующее присваивание – это специальный синтаксис, который позволяет нам «распаковать» массивы или объекты в кучу переменных, так как иногда они более удобны.

let [firstName, surname] = arr;
Обратить внимание, что **let** - обязательно.

Полный синтаксис для объекта:

let {prop : varName = default, ...rest} = object

Свойства, которые не были упомянуты, копируются в объект rest.

Полный синтаксис для массива:

let [item1 = default, item2, ...rest] = array

Первый элемент отправляется в item1; второй отправляется в item2, все остальные элементы попадают в массив rest.

Можно извлекать данные из вложенных объектов и массивов, для этого левая сторона должна иметь ту же структуру, что и правая.

Подробнее с примерами:

Деструктуризация массива или любого перебираемого объекта

Примеры:

```
1)
 // у нас есть массив с именем и фамилией
let arr = ["Ilya", "Kantor"]
// деструктурирующее присваивание записывает firstName=arr[0], surname=arr[1]
let [firstName, surname] = arr;
alert(firstName); // Ilya
alert(surname);  // Kantor

2)
 let [firstName, surname] = "Ilya Kantor".split(' ');

3)
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
}
```

Ненужные элементы массива также могут быть отброшены через запятую:

```
// второй элемент не нужен
let [firstName, , title] = ["Julius", "Caesar", "Consul", "of the Roman Republi
```

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
```

Остаточные параметры «...»

Если мы хотим не просто получить первые значения, но и собрать все остальные, то мы можем добавить ещё один параметр, который получает остальные значения, используя оператор «остаточные параметры» – троеточие ("..."):

```
let [name1, name2, ...rest] = ["Julius", "Caesar", "Consul", "of the Roman Republic"];
alert(name1); // Julius
alert(name2); // Caesar
```

```
// Обратите внимание, что `rest` является массивом.
alert(rest[0]); // Consul
alert(rest[1]); // of the Roman Republic
alert(rest.length); // 2
```

Деструктуризация объекта

Деструктурирующее присваивание также работает с объектами. Синтаксис:

let {var1, var2} = {var1:…, var2:…}

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

alert(title);  // Menu
alert(width);  // 100
alert(height); // 200
```

Порядок свойств не имеет значения

Если мы хотим присвоить свойство объекта переменной с другим названием, например, свойство options.width присвоить переменной w, то мы можем использовать 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
```

Для потенциально отсутствующих свойств мы можем установить **значения по умолчанию**, используя "=", как здесь:

```
let options = {
  title: "Menu"
};
let {width = 100, height = 200, title} = options;
alert(title);  // Menu
alert(width);  // 100
alert(height); // 200

let options = {
  title: "Menu"
};
```

Значениями по умолчанию могут быть любые выражения или даже функции. Они выполнятся, если значения отсутствуют.

```
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
```

Остаток объекта «...»

Что если в объекте больше свойств, чем у нас переменных? Можно использовать троеточие, как и для массивов.

```
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
```

Вложенная деструктуризация

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

```
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

Есть ситуации, когда функция имеет много параметров, большинство из которых не обязательны. Это особенно верно для пользовательских интерфейсов. Представьте себе функцию, которая создаёт меню. Она может иметь ширину, высоту, заголовок, список элементов и так далее. В реальной жизни проблема заключается в том, как запомнить порядок всех аргументов. Обычно IDE пытаются помочь нам, 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. 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);
```

Обратить внимание, что такое деструктурирование подразумевает, что в showMenu() будет обязательно передан аргумент. Если нам нужны все значения по умолчанию, то нам следует передать **пустой объект**:

```
showMenu({}); // ок, все значения - по умолчанию
showMenu(); // так была бы ошибка
```

Мы можем исправить это, сделав {} значением по умолчанию для всего объекта параметров:

```
function showMenu({ title = "Menu", width = 100, height = 200 } = {}) {
  alert( `${title} ${width} ${height}` );
}
showMenu(); // Menu 100 200
```