JavaScript

LEVEL UP: DESTRUCTURING

Destructuring assignment: arrays

Деструктурирующее присваивание – способ извлечения данных из массива или объекта специальным синтаксисом.

Для извлечения данных из массива используются квадратные скобки **[]**:

```
const [ firstItem, secondItem ] = [ item1, item2];
firstItem === item1, secondItem === item2;

const [ name1, name2 ] = [ 'John', 'Mike', 'Abraham', 'Piter' ];
console.log(name1, name2); //  John Mike
```

Destructuring assignment: arrays

Данный код:

```
const users =  [ 'John', 'Mike', 'Abraham', 'Piter' ];
const [ name1, name2 ] = users;
console.log(name1, name2); //  John Mike
```

→

эквивалентен следующему:

```
const users =  [ 'John', 'Mike', 'Abraham', 'Piter' ];
const name1 = users[0];
const name2 = users[1];
```

Destructuring assignment: arrays

Для того, чтобы получить элементы из середины массива, нужно в деструктурюрующем присваивании пропустить ненужные элементы:

```
const [ , name2, name3 ] = [ 'John', 'Mike', 'Abraham', 'Piter' ];
console.log(name2, name3);  // Mike Abraham
```

Здесь мы оставили пустой первую переменную, указав лишь запятую.

Destructuring assignment: arrays

Если воспользуемся rest оператором, можем получить остаток массива в переменную:

const [name1, name2, **...other]** = ['John', 'Mike', 'Abraham', 'Piter'];
console.log(name1, name2); // *John Mike*
console.log(**other**); //*["Abraham", "Piter"]*

rest оператор в деструктурирующем приравнивании может быть только в конце выражения (перед закрывающей квадратной скобкой).

Destructuring assignment: arrays

Если в деструктурирующем присваивании больше переменных, чем элементов в массиве, то последним переменным, которым “не хватило” элементов из массива, будет присвоено undefined:

const [name1, name2, **name3**] = ['John', 'Mike'];
console.log(name1, name2, **name3**); // *John Mike* ***undefined***

Однако этого можно избежать, используя значения по умолчанию:

const [name1, name2, **name3="Unknown"]** = ['John', 'Mike'];
console.log(name1, name2, **name3**); // *John Mike* ***Unknown***

Destructuring assignment: objects

Работа с деструктурирующим присваиванием на объектах производится аналогично работе с массивами.

Для извлечения данных из объекта путем деструктуризации используются фигурные скобки **{}** и имена переменных, соответствующие полям объекат:

const figure = { *width*: 10, *height*: 12, *type*: 'square' };
const { *width*, *height*, *type* } = figure;

console.log(*width*, *height*, *type*); // *10 12 "square"*

Destructuring assignment: objects

Данный код:

```
const figure = { width: 10, height: 12, type: 'square' };
const { width, height, type } = figure;
```

→

эквивалентен следующему:

```
const figure = { width: 10, height: 12, type: 'square' };
const width = figure.width,
      height = figure.height,
      type = figure.type;
```

Destructuring assignment: objects

При необходимости использовать другие имена переменных в деструктуризации используется двоеточие:

const figure = { *width*: 10, *height*: 12, *type*: 'square' };
const { *width*: w, *height*: h, *type*: t } = figure;

console.log(w, h, t); // 10 12 "square"

В традиционном синтаксисе это выглядело бы так:
const figure = { *width*: 10, *height*: 12, *type*: 'square' };
const *w* = figure.width, *h* = figure.height, *t* = figure.type;

Destructuring assignment: objects

Destructuring также позволяет использовать свойства по умолчанию:

```
const info = { type: 'html' };
const { type, size = 0} = info;

console.log(type, size) // 'html' 0
```

В данном примере *size* принял значение по умолчанию, равное нулю.

Destructuring assignment: objects

При использовании деструктурирующего присваивания в правой части выражения можно использовать любые валидные операции, предоставляющие объект:

```
function getUser() {
  return { name: 'John', age: 15 };
}

const { name, age } = getUser();
console.log(name, age); // John 15
```

Destructuring assignment: objects

Destructuring позволяет присваивать и вложенные объекты, для этого используется вложенные фигурные скобки:

```
const element = {
  name: 'div',
  attributes: { className: 'box', title: 'info' }
};

const { name, attributes: { className, title } } = element;
console.log(name, className, title);
// “div” “box” “info”
```

Destructuring assignment: objects

Destructuring позволяет комбинировать присваивание объектов, внутри которых вложены массивы:

```
const element = {
  name: 'div',
  children: [ { name: 'span' }, { name: 'em' } ]
};

const { name: parentEl, children: [ child1 ] } = element;
console.log(parentEl, child1); →
// “div” Object {name: "span"}
```

Destructuring assignment: objects

Можно деструктурировать сущности любой сложности:

```
const element = {
  name: 'div',
  children: [ { name: 'span' }, { name: 'em' } ]
};

const { children: [ { name } ] } = element;
console.log(name); // "span"
```

Destructuring assignment: objects & functions

Destructuring можно использовать внутри функции при работе с аргументами:

```
const user = {  name: 'Louis', surname: 'Watkins', gender: 'Male', age: 50 };

function getBaseInfo( { name, age } ) {
  return `name - ${name}; age - ${age}`;
}

getBaseInfo(user); // "name - Louis; age - 50"
```

Destructuring assignment: objects & functions

Здесь также возможны значения по умолчанию:

```
function getBaseInfo( { name = 'John', age = 18} ) {
  return `name - ${name}; age - ${age}`;
}
getBaseInfo({}); // "name - John; age - 18"
```

Однако вызов без аргумента приведёт к ошибке:

```
getBaseInfo(); →
Uncaught TypeError: Cannot destructure property  `name` of 'undefined' or 'null'
```

Destructuring assignment: objects & functions

Для того, чтобы ошибка не возникала, нужно указать функции, что если ничего не передано, то использовать по умолчанию пустой объект:

```
function getBaseInfo( { name = 'John', age = 18} = {} ) {
  return `name - ${name}; age - ${age}`;
}
getBaseInfo(); // "name - John; age - 18"
```

Значения по умолчанию внутри функции рассматриваются в следующей презентации.

Destructuring assignment: tasks

1. Используя rest оператор и деструктуризацию, создать функцию, которая принимает любое количество аргументов и возвращает объект, содержащий первый аргумент и массив из остатка:

   func(‘a’, ‘b’, ‘c’, ‘d’) →
   {
   first: ‘a’,
   other: [‘b’, ‘c’, ‘d’]
   }

Destructuring assignment: tasks

2. Организовать функцию *getInfo*, которая принимает объект вида
   { name: ..., info: { employees: ..., partners: [...] } }
   и выводит в консоль имя (если имени нет, показывать ‘Unknown’) и первые две компании из массива employees:

```
const organisation = {
  name: 'Google',
  info: { employees: 1536, partners: ['Microsoft', 'Facebook', 'Xing']   }
};
getInfo(organisation); →
```

Name: Google
Partners: Microsoft Facebook

Destructuring assignment: tasks

3. Организовать функцию *createElement*, которая принимает объект, а возвращает строку вида "<TAG_NAME>SOME_TEXT</TAG_NAME>".

   Передаваемый в функцию объект имеет вид { type: ..., text: ...}, где значение type соответствует TAG_NAME, а text - SOME_TEXT.
   Значения по умолчанию: TAG_NAME - "p", SOME_TEXT - "Hello".
   При построении строки использовать template string.

   createElement(); → *" Hello "*
   createElement({ type: 'div', text: 'Test me' }) → *" Test me "*
   createElement({ text: 'Lorem ipsum' }) → *" Lorem ipsum "*