## 1/41

**Создание объекта**

Объекты позволяют описать и сгруппировать характеристики объектов реального мира - пользователя, книги, продукта магазина, чего угодно. Объекты ещё называют словарями, то есть они содержат термины (свойства) и их определения (значения).

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

public: true,

rating: 8.38,

};

Для объявления используются фигурные скобки {} - литерал объекта. При создании объекту можно добавить свойства, каждое из которых описывается парами ключ:значение. Ключ ещё называют именем свойства и это всегда строка. Значением свойства могут быть любые типы: примитивы, массивы, объекты, були, функции и т. п. Свойства разделяются запятой.

Присвой переменной apartment объект описывающий квартиру со следующими характеристиками:

* imgUrl - фотография, значение "https://via.placeholder.com/640x480";
* descr - описание, значение "Spacious apartment in the city center";
* rating - рейтинг, значение 4;
* price - цена, значение 2153;
* tags - метаинформация, массив ["premium", "promoted", "top"].
* Объявлена переменная apartment
* Значение переменной apartment это объект
* В объекте есть свойство imgUrl
* Значение свойства imgUrl это строка "https://via.placeholder.com/640x480"
* В объекте есть свойство descr
* Значение свойства descr это строка "Spacious apartment in the city center"
* В объекте есть свойство rating
* Значение свойства rating это число 4
* В объекте есть свойство price
* Значение свойства price это число 2153
* В объекте есть свойство tags
* Значение свойства tags это массив ["premium", "promoted", "top"]

const apartment = {

    imgUrl: "https://via.placeholder.com/640x480",

    descr: "Spacious apartment in the city center",

    rating: 4,

    price: 2153,

    tags: ["premium", "promoted", "top"],

};

## 2/41

**Вложенные свойства**

Значением свойства может быть другой объект. Это используется для хранения вложенных и группированных данных.

Например, статистика пользователя социальной сети состоит из количества последователей, просмотров и лайков, и хранить эти данные удобнее всего в виде объекта. Тоже самое с местоположением, отдельно страна и город.

В будущем это можно будет использовать для поиска пользователей по стране, городу, минимальному или максимальному количеству последователей и т. д.

const user = {

## name: "Jacques Gluke",Создание объекта

Объекты позволяют описать и сгруппировать характеристики объектов реального мира - пользователя, книги, продукта магазина, чего угодно. Объекты ещё называют словарями, то есть они содержат термины (свойства) и их определения (значения).

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

public: true,

rating: 8.38,

};

Для объявления используются фигурные скобки {} - литерал объекта. При создании объекту можно добавить свойства, каждое из которых описывается парами ключ:значение. Ключ ещё называют именем свойства и это всегда строка. Значением свойства могут быть любые типы: примитивы, массивы, объекты, були, функции и т. п. Свойства разделяются запятой.

Присвой переменной apartment объект описывающий квартиру со следующими характеристиками:

* imgUrl - фотография, значение "https://via.placeholder.com/640x480";
* descr - описание, значение "Spacious apartment in the city center";
* rating - рейтинг, значение 4;
* price - цена, значение 2153;
* tags - метаинформация, массив ["premium", "promoted", "top"].
* Объявлена переменная apartment
* Значение переменной apartment это объект
* В объекте есть свойство imgUrl
* Значение свойства imgUrl это строка "https://via.placeholder.com/640x480"
* В объекте есть свойство descr
* Значение свойства descr это строка "Spacious apartment in the city center"
* В объекте есть свойство rating
* Значение свойства rating это число 4
* В объекте есть свойство price
* Значение свойства price это число 2153
* В объекте есть свойство tags
* Значение свойства tags это массив ["premium", "promoted", "top"]

tag: "jgluke",

location: {

country: "Jamaica",

city: "Ocho Rios",

},

stats: {

followers: 5603,

views: 4827,

likes: 1308,

},

};

Дополни объект квартиры свойством owner, значением которого будет объект с информацией о владельце. Добавь ему следующие свойства:

* name - имя владельца, значение "Henry";
* phone - телефон, значение "982-126-1588";
* email - почта, значение "henry.carter@aptmail.com".
* Объявлена переменная apartment
* Значение переменной apartment это объект
* У объекта переменной apartment присутствуют свойства imgUrl, descr, rating, price и tag со значениями
* В объекте apartment есть свойство owner
* Значение свойства owner это объект
* В объекте owner есть свойство name
* Значение свойства name это "Henry"
* В объекте owner есть свойство phone
* Значение свойства phone это "982-126-1588"
* В объекте owner есть свойствао email
* Значение свойства email это "henry.carter@aptmail.com"

**Редактор javascript:**

1const apartment = {

  imgUrl: "https://via.placeholder.com/640x480",

  descr: "Spacious apartment in the city center",

  rating: 4,

  price: 2153,

  tags: ["premium", "promoted", "top"],

  owner: {

    name: "Henry",

    phone: "982-126-1588",

    email: "henry.carter@aptmail.com",

  }

};

## 3/41

**Доступ к свойствам через точку**

Первый способ получить доступ к свойству объекта это синтаксис обьект.ключ\_свойства. Синтаксис «через точку» используется в большинстве случаев и подходит тогда, когда мы заранее знаем имя (ключ) свойства к которому хотим получить доступ.

* На место обращения будет возвращено значение свойства с таким именем.
* Если в объекте нет свойства с таким именем, на место обращения вернётся undefined.

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

public: true,

rating: 8.38,

};

const bookTitle = book.title;

console.log(bookTitle); // "The Last Kingdom"

const bookGenres = book.genres;

console.log(bookGenres); // ["historical prose", "adventure"]

const bookPrice = book.price;

console.log(bookPrice); // undefined

Дополни код присвоив объявленным переменным выражения обращения к соответствующим свойствам обьекта apartment.

* aptRating - рейтинг;
* aptDescr - описание;
* aptPrice - цена;
* aptTags - теги.
* Объявлена переменная apartment
* Значение переменной apartment это объект
* У объекта переменной apartment присутствуют свойства imgUrl, descr, rating, price и tag со значениями
* Объявлена переменная aptRating
* Значение переменной aptRating это число 4
* Объявлена переменная aptDescr
* Значение переменной aptDescr это строка "Spacious apartment in the city center"
* Объявлена переменная aptPrice
* Значение переменной aptPrice это число 2153
* Объявлена переменная aptTags
* Значение переменной aptTags это массив строк ["premium", "promoted", "top"]
* const apartment = {
* imgUrl: "https://via.placeholder.com/640x480",
* descr: "Spacious apartment in the city center",
* rating: 4,
* price: 2153,
* tags: ["premium", "promoted", "top"],
* };
* // Change code below this line
* const aptRating = apartment.rating;
* const aptDescr = apartment.descr;
* const aptPrice = apartment.price;
* const aptTags = apartment.tags;
* // Change code above this line

## 4/41

**Доступ к вложенным свойствам**

const user = {

name: "Jacques Gluke",

tag: "jgluke",

location: {

country: "Jamaica",

city: "Ocho Rios",

},

hobbies: ["swiming", "music", "sci-fi"],

};

Для доступа к вложенным свойствам используется цепочка обращений «через точку». Например, если необходимо получить значение страны пользователя, записываем user.location.country, где user.location это обращение (путь) к объекту в свойстве location, а user.locaton.country обращение к свойству country в этом объекте. То есть, «точка» указывает следующую вложенность.

const location = user.location;

console.log(location); // Объект location

const country = user.location.country;

console.log(country); // "Jamaica"

Если значение свойства это массив, то в нашем примере user.hobbies - обращение к этому массиву. Далее, можно получить доступ к его элементам через квадратные скобки и индекс или использовать свойства и методы.

const hobbies = user.hobbies;

console.log(hobbies); // ["swiming", "music", "sci-fi"]

const firstHobby = user.hobbies[0];

console.log(firstHobby); // "swiming"

const numberOfHobbies = user.hobbies.length;

console.log(numberOfHobbies); // 3

Дополни код присвоив объявленным переменным выражения обращения к соответствующим свойствам обьекта apartment.

* ownerName - имя владельца;
* ownerPhone - телефон владельца;
* ownerEmail - почта владельца;
* numberOfTags - количество элементов массива в свойстве tags;
* firstTag - первый элемент массива в свойстве tags;
* lastTag - последний элемент массива в свойстве tags.
* Объявлена переменная apartment с помощью const
* Значение переменной apartment это объект
* Объявлена переменная ownerName с помощью const
* Значение переменной ownerName это строка "Henry"
* Объявлена переменная ownerPhone с помощью const
* Значение переменной ownerPhone это "982-126-1588"
* Объявлена переменная ownerEmail с помощью const
* Значение переменной ownerEmail это "henry.carter@aptmail.com"
* Объявлена переменная numberOfTags с помощью const
* Значение переменной numberOfTags это 3
* Объявлена переменная firstTag с помощью const
* Значение переменной firstTag это "premium"
* Объявлена переменная lastTag с помощью const
* Значение переменной lastTag это "top"
* const apartment = {
* imgUrl: "https://via.placeholder.com/640x480",
* descr: "Spacious apartment in the city center",
* rating: 4,
* price: 2153,
* tags: ["premium", "promoted", "top"],
* owner: {
* name: "Henry",
* phone: "982-126-1588",
* email: "henry.carter@aptmail.com",
* },
* };
* // Change code below this line
* const ownerName = apartment.owner.name;
* const ownerPhone = apartment.owner.phone;
* const ownerEmail = apartment.owner.email;
* const numberOfTags = apartment.tags.length;
* const firstTag = apartment.tags[0];
* const lastTag = apartment.tags[2];
* // Change code above this line

## 5/41

**Доступ к свойствам через квадратные скобки**

Второй способ получить доступ к свойству объекта это синтаксис обьект["ключ\_свойства"]. Похоже на обращение к элементу массива с отличием в том, что в скобках указывается не индекс элемента, а имя свойства как строка.

Синтаксис «квадратных скобок» используется значительно реже. Как правило в случаях когда имя свойства заранее неизвестно или оно хранится в переменной (как значение параметра функции, например).

* На место обращения будет возвращено значение свойства с таким именем.
* Если в объекте нет свойства с таким именем, на место обращения вернётся undefined.

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

public: true,

rating: 8.38,

};

const bookTitle = book["title"];

console.log(bookTitle); // "The Last Kingdom"

const bookGenres = book["genres"];

console.log(bookGenres); // ["historical prose", "adventure"]

const propKey = "author";

const bookAuthor = book[propKey];

console.log(bookAuthor); // "Bernard Cornwell"

Дополни код присвоив объявленным переменным выражения обращения к соответствующим свойствам обьекта apartment используя синтаксис «квадратных скобок».

* aptRating - рейтинг;
* aptDescr - описание;
* aptPrice - цена;
* aptTags - теги.
* Объявлена переменная apartment
* Значение переменной apartment это объект
* Объявлена переменная aptRating
* Значение переменной aptRating это 4
* Объявлена переменная aptDescr
* Значение переменной aptDescr это "Spacious apartment in the city center"
* Объявлена переменная aptPrice
* Значение переменной aptPrice это 2153
* Объявлена переменная aptTags
* Значение переменной aptTags это ["premium", "promoted", "top"]
* const apartment = {
* imgUrl: "https://via.placeholder.com/640x480",
* descr: "Spacious apartment in the city center",
* rating: 4,
* price: 2153,
* tags: ["premium", "promoted", "top"],
* };
* // Change code below this line
* const aptRating = apartment['rating'];
* const aptDescr = apartment['descr'];
* const aptPrice = apartment['price'];
* const aptTags = apartment['tags'];
* // Change code above this line

## 6/41

**Изменение значения свойства**

После того, как объект создан, значение его свойств можно изменить. Для этого необходимо обратиться к ним по имени, например, «через точку», и присвоить новое значение.

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

public: true,

rating: 8.38,

};

book.rating = 9;

book.public = false;

book.genres.push("drama");

console.log(book.rating); // 9

console.log(book.public); // false

console.log(book.genres); // ["historical prose", "adventure", "drama"]

Дополни код обновив значения свойств объекта apartment:

* цену в свойстве price на 5000;
* рейтинг квартиры в свойстве rating на 4.7;
* имя владельца во вложенном свойстве name на "Henry Sibola";
* массив тегов в свойстве tags добавив в конец строку "trusted".
* Объявлена переменная apartment
* Значение переменной apartment это объект
* Значение вложенного свойства price это число 5000
* Значение вложенного свойства rating это число 4.7
* Значение вложенного свойства name это строка "Henry Sibola"
* Значение вложенного свойства tags это массив ["premium", "promoted", "top", "trusted"]
* const apartment = {
* imgUrl: "https://via.placeholder.com/640x480",
* descr: "Spacious apartment in the city center",
* rating: 4,
* price: 2153,
* tags: ["premium", "promoted", "top"],
* owner: {
* name: "Henry",
* phone: "982-126-1588",
* email: "henry.carter@aptmail.com",
* },
* };
* // Change code below this line
* apartment.price = 5000;
* apartment.rating = 4.7;
* apartment.owner.name = "Henry Sibola";
* apartment.tags.push("trusted");

## 7/41

**Добавление свойств**

Операция добавления нового свойства после создания объекта ничем не отличается от изменения значения уже существующего свойства. Если при записи значения по имени, такого свойства в объекте нет, оно будет создано.

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

public: true,

rating: 8.38,

};

book.pageCount = 836;

book.originalLanguage = "en";

book.translations = ["ua", "ru"];

console.log(book.pageCount); // 836

console.log(book.originalLanguage); // "en"

console.log(book.translations); // ["ua", "ru"]

Добавь объекту apartment несколько новых свойств:

* area - площадь в квадратных метрах, число 60;
* rooms - количество комнат, число 3;
* location - местоположение квартиры, обьект со следующими вложенными свойствами;
  + country - страна, строка "Jamaica";
  + city - город, строка "Kingston".
* Объявлена переменная apartment
* Значение переменной apartment это объект
* Значение вложенного свойства area это число 60
* Значение вложенного свойства rooms это число 3
* Значение вложенного свойства location это объект
* Значение вложенного свойства location.country это строка "Jamaica"
* Значение вложенного свойства location.city это строка "Kingston"
* const apartment = {
* imgUrl: "https://via.placeholder.com/640x480",
* descr: "Spacious apartment in the city center",
* rating: 4.7,
* price: 5000,
* tags: ["premium", "promoted", "top", "trusted"],
* owner: {
* name: "Henry Sibola",
* phone: "982-126-1588",
* email: "henry.carter@aptmail.com",
* },
* };
* // Change code below this line
* apartment.area = 60;
* apartment.rooms = 3;
* apartment.location = {country: "Jamaica", city: "Kingston"};

## 8/41

**Короткие свойства**

Иногда, при создании объекта, значение свойства необходимо взять из переменной или параметра функции с таким же именем, как и само свойство.

Синтаксис в следующем примере слишком громоздкий, потому что приходится дублировать имя свойства и имя переменной, в которой хранится необходимое значение.

const name = "Генри Сибола";

const age = 25;

const user = {

name: name,

age: age,

};

console.log(user.name); // "Генри Сибола"

console.log(user.age); // 25

Синтаксис коротких свойств (shorthand properties) решает эту проблему, позволяя использовать имя переменной как имя свойства, а её значение как значение свойства.

const name = "Генри Сибола";

const age = 25;

const user = {

name,

age,

};

console.log(user.name); // "Генри Сибола"

console.log(user.age); // 25

То есть, при объявлении объекта достаточно указать только имя свойства, а значение будет взято из переменной с аналогичным именем.

Дополни код объявления объекта так, чтобы у него были свойства name, price, image и tags со значениями из переменных с аналогичными именами. Обязательно используй синтаксис коротких свойств.

* Объявлена переменная product
* Значение переменной product это объект
* Значение вложенного свойства name это строка "Repair Droid"
* Значение вложенного свойства price это число 2500
* Значение вложенного свойства image это строка "https://via.placeholder.com/640x480"
* Значение вложенного свойства tags это массив ["on sale", "trending", "best buy"]
* const name = "Repair Droid";
* const price = 2500;
* const image = "https://via.placeholder.com/640x480";
* const tags = ["on sale", "trending", "best buy"];
* const product = {
* // Change code below this line
* name,
* price,
* image,
* tags,
* // Change code above this line
* };

## 9/41

**Вычисляемые свойства**

Бывают ситуации, когда при объявлении обьекта необходимо добавить свойство с именем, которое мы заранее не знаем, потому что оно хранится как значение переменной или как результат выполнения функции.

Раньше для этого необходимо было сначала создать объект, а потом добавлять свойства через квадратные скобки, что не совсем удобно.

const propName = "name";

const user = {

age: 25

};

user[propName] = "Генри Сибола";

console.log(user.name); // "Генри Сибола"

Синтаксис вычисляемых свойств (computed properties) помогает избежать лишнего кода и в некоторых случаях упростить его. Значением вычисляемого свойства может быть любое валидное выражение.

const propName = "name";

const user = {

age: 25,

// Имя этого свойства будет взято из значения переменной propName

[propName]: "Генри Сибола"

};

console.log(user.name); // "Генри Сибола"

Дополни код объявления объекта credentials так, чтобы в результате у него были два свойства: email и password, имена которых хранятся в переменных emailInputName и passwordInputName.

Значением свойства email должна быть строка "henry.carter@aptmail.com", а значением свойства password - строка "jqueryismyjam".

* Объявлена переменная credentials
* Значение переменной credentials это объект
* В объекте credentials есть свойство email
* Значение вложенного свойства email это строка "henry.carter@aptmail.com"
* В объекте credentials есть свойство password
* Значение вложенного свойства password это строка "jqueryismyjam"
* const emailInputName = "email";
* const passwordInputName = "password";
* const credentials = {
* // Change code below this line
* [emailInputName]: 'henry.carter@aptmail.com',
* [passwordInputName]: 'jqueryismyjam',
* // Change code above this line
* };

## 10/41

**Цикл for...in**

В отличии от массива или строки, объект - это не итерируемая сущность, то есть его нельзя перебрать циклами for или for...of. Для перебора объектов используется специальный цикл for...in, который перебирает ключи объекта object.

for (key in object) {

// инструкции

}

Переменная key доступная только в теле цикла. На каждой итерации в неё будет записано значение ключа (имя) свойства. Для того чтобы получить значение свойства с таким ключом (именем), используется синтаксис квадратных скобок.

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

rating: 8.38,

};

for (const key in book) {

// Ключ

console.log(key);

// Значение свойства с таким ключом

console.log(book[key]);

}

Перебери объект apartment используя цикл for...in и запиши в массив keys все его ключи, а в массив values все значения его свойств.

* Объявлена переменная apartment
* Значение переменной apartment это объект
* Объявлена переменная keys
* Значение переменной keys это массив ["descr", "rating", "price"]
* Объявлена переменная values
* Значение переменной values это массив ["Spacious apartment in the city center", 4, 2153]
* const apartment = {
* descr: "Spacious apartment in the city center",
* rating: 4,
* price: 2153,
* };
* const keys = [];
* const values = [];
* // Change code below this line
* for (const key in apartment){
* keys.push(key);
* values.push(apartment[key]);
* };

## 11/41

**Метод hasOwnProperty()**

Разберём концепцию собственных и несобственных свойств объекта и научимся правильно использовать цикл for...in.

const animal = {

legs: 4,

};

const dog = Object.create(animal);

dog.name = "Mango";

console.log(dog); // {name: 'Mango'}

console.log(dog.name); // 'Mango'

console.log(dog.legs); // 4

Метод Object.create(animal) создаёт и возвращает новый объект, связывая его с объектом animal. Поэтому можно получить значение свойства legs обратившись к нему как dog.legs, хотя его нет в объекте dog - это несобственное свойство из объекта animal.

Оператор in, который используется в цикле for...in, не делает различия между собственными и несобственными свойствами объекта. Эта особенность мешает, так как мы всегда хотим перебрать только собственные свойства. Для того чтобы узнать есть в объекте собственное свойство или нет, используется метод hasOwnProperty(key), который возвращает true или false.

// ❌ Возвращает true для всех свойств

console.log("name" in dog); // true

console.log("legs" in dog); // true

// ✅ Возвращает true только для собственных свойств

console.log(dog.hasOwnProperty("name")); // true

console.log(dog.hasOwnProperty("legs")); // false

Поэтому при переборе циклом for...in необходимо на каждой итерации добавить проверку на собственное свойство. Даже если сейчас мы уверены в том что у объекта нет несобственных свойств, это оградит от возможных ошибок в будущем.

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

rating: 8.38,

};

for (const key in book) {

// Если это собственное свойство - выполняем тело if

if (book.hasOwnProperty(key)) {

console.log(key);

console.log(book[key]);

}

// Если это не собственное свойство - ничего не делаем

}

Выполни рефакторинг решения предыдущего задания добавив в цикл for...in проверку на собственное свойство.

* Объявлена переменная advert.
* Значение переменной advert это объект.
* Объявлена переменная apartment.
* Значение переменной apartment это объект.
* Объявлена переменная keys.
* Значение переменной keys это массив ["descr", "rating", "price"].
* Объявлена переменная values.
* Значение переменной values это массив ["Spacious apartment in the city center", 4, 2153].
* const keys = [];
* const values = [];
* const advert = {
* service: "apt",
* };
* const apartment = Object.create(advert);
* apartment.descr = "Spacious apartment in the city center";
* apartment.rating = 4;
* apartment.price = 2153;
* for (const key in apartment) {
* // Change code below this line
* if (apartment.hasOwnProperty(key)) {
* keys.push(key);
* values.push(apartment[key]);
* }
* // Change code above this line
* }

## 12/41

**Задача. Подсчёт свойств**

Напиши функцию countProps(object), которая считает и возвращает количество собственных свойств объекта в параметре object. Используй переменную propCount для хранения количества свойств объекта.

* Объявлена функция countProps(object)
* Вызов countProps({}) возвращает 0
* Вызов countProps({ name: "Mango", age: 2 }) возвращает 2
* Вызов countProps({ mail: "poly@mail.com", isOnline: true, score: 500 }) возвращает 3
* Функция подсчитывает только собственные свойства объекта
* function countProps(object) {
* let propCount = 0;
* // Change code below this line
* for (const key in object) {
* if (object.hasOwnProperty(key)){
* propCount += 1;
* }
* }
* // Change code above this line
* return propCount;
* }

## 13/41

**Метод Object.keys()**

У встроенного класса Object есть несколько полезных методов для работы с объектами. Первый из них это Object.keys(obj), который принимает объект и возвращает массив ключей его собственных свойств. Если в объекте нет свойств, метод вернёт пустой массив.

const book = {

title: "The Last Kingdom",

genres: ["historical prose", "adventure"],

rating: 8.38,

};

const keys = Object.keys(book);

console.log(keys); // ['title', 'author', 'genres', 'rating']

Скомбинировав результат Object.keys() и цикл for...of можно удобно перебрать собственные свойства объекта, не прибегая к использованию архаического цикла for...in с проверками принадлежности свойств.

const book = {

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

rating: 8.38,

};

const keys = Object.keys(book);

for (const key of keys) {

// Ключ

console.log(key);

// Значение свойства

console.log(book[key]);

}

Мы перебираем массив ключей объекта и на каждой итерации получаем значение свойства с таким ключом.

Перебери объект apartment используя метод Object.keys() и цикл for...of. Запиши в переменную keys массив ключей собственных свойств объекта apartment, и добавь в массив values все значения его свойств.

* Объявлена переменная apartment.
* Значение переменной apartment это объект.
* Объявлена переменная keys.
* Значение переменной keys это массив ["descr", "rating", "price"].
* Значение переменной keys получено с помощью метода Object.keys().
* Объявлена переменная values.
* Значение переменной values это массив ["Spacious apartment in the city center", 4, 2153].
  + Значение переменной values получено с помощью цикла for...of.
* const apartment = {
* descr: "Spacious apartment in the city center",
* rating: 4,
* price: 2153,
* };
* const values = [];
* // Change code below this line
* const keys = Object.keys(apartment);
* for (const key of keys){
* values.push(apartment[key])
* }

## 14/41

**Задача. Подсчёт свойств 2.0**

Выполни рефакторинг функции countProps(object) используя метод Object.keys() и, возможно, но необязательно, цикл for...of.

* Объявлена функция countProps(object)
* Вызов countProps({}) возвращает 0
* Вызов countProps({ name: "Mango", age: 2 }) возвращает 2
* Вызов countProps({ mail: "poly@mail.com", isOnline: true, score: 500 }) возвращает 3
* Функция подсчитывает только собственные свойства объекта
* Функция использует метод Object.keys() и, возможно, цикл for...of
* function countProps(object) {
* // Change code below this line
* let propCount = 0;
* for (const key of Object.keys(object)) {
* propCount += 1;
* }
* return propCount;
* // Change code above this line
* }

## 15/41

**Метод Object.values()**

Если метод Object.keys(obj) возвращает массив ключей собственных свойств объекта, то метод Object.values(obj) возвращает массив значений его собственных свойств. Если в объекте нет свойств, метод Object.values(obj) вернёт пустой массив.

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

rating: 8.38,

};

const keys = Object.keys(book);

console.log(keys); // ["title", "author", "rating"]

const values = Object.values(book);

console.log(values); // ["The Last Kingdom", "Bernard Cornwell", 8.38]

Массив значений свойств также можно перебрать циклом for...of, например для получения общей суммы числовых значений.

Запиши в переменную keys массив ключей собственных свойств объекта apartment, а в переменную values массив всех значений его свойств. Используй методы Object.keys() и Object.values().

* Объявлена переменная apartment
* Значение переменной apartment это объект
* Объявлена переменная keys
* Значение переменной keys это массив ["descr", "rating", "price"]
* Объявлена переменная values
* Значение переменной values это массив ["Spacious apartment in the city center", 4, 2153]
* Для получения массива ключей объекта apartment используется Object.keys()
* Для получения массива значений объекта apartment используется Object.values()
* const apartment = {
* descr: "Spacious apartment in the city center",
* rating: 4,
* price: 2153,
* };
* // Change code below this line
* const keys = Object.keys(apartment);
* const values = Object.values(apartment);

## 16/41

**Задача. Расходы на зарплату**

Напиши функцию countTotalSalary(salaries) которая принимает объект зарплат, где имя свойства это имя сотрудника, а значение свойства это зарплата. Функция должна рассчитать общую сумму зарплат сотрудников и вернуть её. Используй переменную totalSalary для хранения общей суммы зарплаты.

* Объявлена функция countTotalSalary(salaries)
* Вызов countTotalSalary({}) возвращает 0
* Вызов countTotalSalary({ mango: 100, poly: 150, alfred: 80 }) возвращает 330
* Вызов countTotalSalary({ kiwi: 200, poly: 50, ajax: 150 }) возвращает 400
* Функция учитывает только собственные свойства объекта
* function countTotalSalary(salaries) {
* let totalSalary = 0;
* // Change code below this line
* for (const value of Object.values(salaries)){
* totalSalary += value
* }
* // Change code above this line
* return totalSalary;
* }

## 17/41

**Массив объектов**

В стандартный набор повседневных задач разработчика входит манипуляция массивом однотипных объектов. Это значит что все объекты в массиве гарантированно будут иметь одинаковый набор свойств, но с разными значениями.

const books = [

{

title: "The Last Kingdom",

author: "Bernard Cornwell",

rating: 8.38,

},

{

title: "Beside Still Waters",

author: "Robert Sheckley",

rating: 8.51,

},

{

title: "The Dream of a Ridiculous Man",

author: "Fyodor Dostoevsky",

rating: 7.75,

},

];

Для перебора такого массива используется стандартный цикл for...of. Значения свойств каждого объекта можно получить используя синтаксис «через точку», так как в каждом объекте набор свойств и их имена будут одинаковые, отличаются только значения.

for (const book of books) {

// Объект книги

console.log(book);

// Название

console.log(book.title);

// Автор

console.log(book.author);

// Рейтинг

console.log(book.rating);

}

Перебери массив объектов colors используя цикл for...of. Добавь в массив hexColors значения свойств hex, а в массив rgbColors значения свойств rgb из всех объектов массива colors.

* Объявлена переменная colors
* Значение переменной colors это массив
* Объявлена переменная hexColors
* Значение переменной hexColors это массив ["#f44336", "#2196f3", "#4caf50", "#ffeb3b"]
* Объявлена переменная rgbColors
* Значение переменной rgbColors это массив ["244,67,54", "33,150,243", "76,175,80", "255,235,59"]
* const colors = [
* { hex: "#f44336", rgb: "244,67,54" },
* { hex: "#2196f3", rgb: "33,150,243" },
* { hex: "#4caf50", rgb: "76,175,80" },
* { hex: "#ffeb3b", rgb: "255,235,59" },
* ];
* const hexColors = [];
* const rgbColors = [];
* // Change code below this line
* for (const color of colors){
* hexColors.push(color.hex);
* rgbColors.push(color.rgb);
* }

## 18/41

**Задача. Поиск объекта по значению свойства**

Напиши функцию getProductPrice(productName) которая принимает один параметр productName - название продукта. Функция ищет объект продукта с таким именем (свойство name) в массиве products и возвращает его цену (свойство price). Если продукт с таким названием не найден, функция должна возвращать null.

* Объявлена функция getProductPrice(productName).
* Вызов getProductPrice("Radar") возвращает 1300.
* Вызов getProductPrice("Grip") возвращает 1200.
* Вызов getProductPrice("Scanner") возвращает 2700.
* Вызов getProductPrice("Droid") возвращает 400.
* Вызов getProductPrice("Engine") возвращает null.
* const products = [
* { name: "Radar", price: 1300, quantity: 4 },
* { name: "Scanner", price: 2700, quantity: 3 },
* { name: "Droid", price: 400, quantity: 7 },
* { name: "Grip", price: 1200, quantity: 9 },
* ];
* function getProductPrice(productName) {
* // Change code below this line
* for (const product of products) {
* if (product.name === productName) {
* return product.price;
* }
* }
* return null;
* // Change code above this line
* }

## 19/41

**Задача. Коллекция значений свойства**

Напиши функцию getAllPropValues(propName) которая принимает один параметр propName - имя (ключ) свойства. Функция должна вернуть массив всех значений свойства с таким именем из каждого объекта в массиве products. Если в объектах нет свойства с таким именем, функция должна вернуть пустой массив.

* Объявлена функция getAllPropValues(propName)
* Вызов getAllPropValues("name") возвращает ["Radar", "Scanner", "Droid", "Grip"]
* Вызов getAllPropValues("quantity") возвращает [4, 3, 7, 9]
* Вызов getAllPropValues("price") возвращает [1300, 2700, 400, 1200]
* Вызов getAllPropValues("category") возвращает []
* const products = [
* { name: "Radar", price: 1300, quantity: 4 },
* { name: "Scanner", price: 2700, quantity: 3 },
* { name: "Droid", price: 400, quantity: 7 },
* { name: "Grip", price: 1200, quantity: 9 },
* ];
* function getAllPropValues(propName) {
* // Change code below this line
* const result = [];
* for (const product of products) {
* if (product.hasOwnProperty(propName)) {
* result.push(product[propName]);
* }
* }
* return result;
* // Change code above this line
* }

## 20/41

**Задача. Общая стоимость товара**

Напиши функцию calculateTotalPrice(productName) которая принимает один параметр productName - название товара. Функция должна вернуть общую стоимость (цена \* количество) товара с таким именем из массива products.

* Объявлена функция calculateTotalPrice(productName)
* Вызов calculateTotalPrice("Blaster") возвращает 0
* Вызов calculateTotalPrice("Radar") возвращает 5200
* Вызов calculateTotalPrice("Droid") возвращает 2800
* Вызов calculateTotalPrice("Grip") возвращает 10800
* Вызов calculateTotalPrice("Scanner") возвращает 8100
* const products = [
* { name: "Radar", price: 1300, quantity: 4 },
* { name: "Scanner", price: 2700, quantity: 3 },
* { name: "Droid", price: 400, quantity: 7 },
* { name: "Grip", price: 1200, quantity: 9 },
* ];
* function calculateTotalPrice(productName) {
* // Change code below this line
* let totalPrice = 0;
* for (const product of products) {
* if (product.name === productName) {
* totalPrice += product.price \* product.quantity;
* }
* }
* return totalPrice;
* // Change code above this line
* }

## 21/41

**Деструктуризация объектов**

Сложные данные всегда представлены объектом. Множественные обращения к свойствам объекта визуально загрязняют код.

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

public: true,

rating: 8.38,

};

const accessType = book.public ? "публичном" : "закрытом";

const message = `Книга ${book.title} автора ${book.author} с рейтингом ${book.rating} находится в ${accessType} доступе.`;

Деструктуризация позволяет «распаковать» значения свойств объекта в локальные переменные. Это делает код в месте их использования менее «шумным».

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

genres: ["historical prose", "adventure"],

public: true,

rating: 8.38,

};

// Деструктуризируем

const { title, author, public, rating, coverImage } = book;

console.log(coverImage); // undefined

const accessType = public ? "публичном" : "закрытом";

const message = `Книга ${title} автора ${author} с рейтингом ${rating} находится в ${accessType} доступе.`;

Деструктуризация всегда находится в левой части операции присвоения. Переменным внутри фигурных скобок присваиваются значения свойств объекта. Если имя переменной и имя свойства совпадают, то происходит присваивание, в противном случае ей будет присвоено undefined. Порядок объявления переменных в фигурных скобках не важен.

Пришел трёхдневный прогноз максимальных температур и мы считаем среднюю температуру за три дня (meanTemperature). Замени объявления переменных yesterday, today и tomorrow одной операцией деструктуризации свойств объекта highTemperatures.

* Объявлена переменная highTemperatures
* Значение переменной highTemperatures это объект
* Объявлена переменная yesterday с помощью деструктуризации
* Значение переменной yesterday это число 28
* Объявлена переменная today с помощью деструктуризации
* Значение переменной today это число 26
* Объявлена переменная tomorrow с помощью деструктуризации
* Значение переменной tomorrow это число 33
* Объявлена переменная meanTemperature
* Значение переменной meanTemperature это число 29
* Используется синтаксис деструктуризации объекта highTemperatures
* const highTemperatures = {
* yesterday: 28,
* today: 26,
* tomorrow: 33,
* };
* // Change code below this line
* const {yesterday, today, tomorrow } = highTemperatures;
* // Change code above this line
* const meanTemperature = (yesterday + today + tomorrow) / 3;

## 22/41

**Значения по умолчанию**

Для того чтобы избежать присвоения undefined при деструктуризации несуществующих свойств, можно задать переменным значения по умолчанию, которые будут присвоены только в случае когда в объекте нет свойства с таким именем.

const book = {

title: "The Last Kingdom",

author: "Bernard Cornwell",

};

// Добавим картинку обложки если её нет в объекте книги

const {

title,

coverImage = "https://via.placeholder.com/640/480",

author,

} = book;

console.log(title); // "The Last Kingdom"

console.log(author); // "Bernard Cornwell"

console.log(coverImage); // "https://via.placeholder.com/640/480"

В прогнозе максимальных температур также может быть необязательное свойство icon - иконка погоды. Замени объявления переменных yesterday, today, tomorrow и icon одной операцией деструктуризации свойств объекта highTemperatures. Задай значение по умолчанию для icon - строку "https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg".

* Объявлена переменная highTemperatures
* Значение переменной highTemperatures это объект
* Объявлена переменная highTemperatures
* Значение переменной highTemperatures это объект
* Объявлена переменная yesterday с помощью деструктуризации
* Значение переменной yesterday это число 28
* Объявлена переменная today с помощью деструктуризации
* Значение переменной today это число 26
* Объявлена переменная tomorrow с помощью деструктуризации
* Значение переменной tomorrow это число 33
* Объявлена переменная icon с помощью деструктуризации
* Значение переменной icon это строка "https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg".
* Используется деструктуризация объекта
* const highTemperatures = {
* yesterday: 28,
* today: 26,
* tomorrow: 33,
* };
* // Change code below this line
* const {
* yesterday,
* today,
* tomorrow,
* icon = "https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg",
* } = highTemperatures;
* // Change code above this line
* const meanTemperature = (yesterday + today + tomorrow) / 3;

## 23/41

**Изменение имени переменной**

При деструктуризации можно изменить имя переменной в которую распаковывается значение свойства. Сначала пишем имя свойства из которого хотим получить значение, после чего ставим двоеточие и пишем имя переменной в которую необходимо поместить значение этого свойства.

const firstBook = {

title: "Последнее королевство",

coverImage:

"https://images-na.ssl-images-amazon.com/images/I/51b5YG6Y1rL.jpg",

};

const {

title: firstTitle,

coverImage: firstCoverImage = "https://via.placeholder.com/640/480",

} = firstBook;

console.log(firstTitle); // Последнее королевство

console.log(firstCoverImage); // https://images-na.ssl-images-amazon.com/images/I/51b5YG6Y1rL.jpg

const secondBook = {

title: "Сон смешного человека",

};

const {

title: secondTitle,

coverImage: secondCoverImage = "https://via.placeholder.com/640/480",

} = secondBook;

console.log(secondTitle); // Сон смешного человека

console.log(secondCoverImage); // https://via.placeholder.com/640/480

Такая запись читается как «Создать переменную firstTitle, в которую поместить значение свойства title из объекта firstBook» и т. д.

Замени объявления переменных highYesterday, highToday, highTomorrow и highIcon одной операцией деструктуризации свойств объекта highTemperatures. Задай значение по умолчанию для highIcon - строку "https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg".

* Объявлена переменная highTemperatures
* Значение переменной highTemperatures это объект
* Объявлена переменная highYesterday
* Значение переменной highYesterday это число 28
* Объявлена переменная highToday
* Значение переменной highToday это число 26
* Объявлена переменная highTomorrow
* Значение переменной highTomorrow это число 33
* Объявлена переменная highIcon
* Значение переменной highIcon это строка "https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg"
* Используется деструктуризация объекта
* const highTemperatures = {
* yesterday: 28,
* today: 26,
* tomorrow: 33,
* };
* // Change code below this line
* const {
* yesterday: highYesterday,
* today: highToday,
* tomorrow: highTomorrow,
* icon: highIcon = "https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg",
* } = highTemperatures;
* // Change code above this line
* const meanTemperature = (highYesterday + highToday + highTomorrow) / 3;

## 24/41

**Деструктуризация в циклах**

Выполни рефакторинг цикла for...of так, чтобы в нём использовалась деструктуризация объекта..

* Объявлена переменная colors
* Значение переменной colors это массив
* Объявлена переменная hexColors
* Значение переменной hexColors это массив ["#f44336", "#2196f3", "#4caf50", "#ffeb3b"]
* Объявлена переменная rgbColors
* Значение переменной rgbColors это массив ["244,67,54", "33,150,243", "76,175,80", "255,235,59"]
* Для перебора массива используется цикл for...of
* В цикле for...of используется деструктуризация объекта
* const colors = [
* { hex: "#f44336", rgb: "244,67,54" },
* { hex: "#2196f3", rgb: "33,150,243" },
* { hex: "#4caf50", rgb: "76,175,80" },
* { hex: "#ffeb3b", rgb: "255,235,59" },
* ];
* const hexColors = [];
* const rgbColors = [];
* // Change code below this line
* for (const { hex, rgb } of colors) {
* hexColors.push(hex);
* rgbColors.push(rgb);
* }

## 25/41

**Глубокая деструктуризация**

Мы получили прогноз погоды на два дня, с минимальными и максимальными температурами, а также необязательными иконками. Замени объявления всех переменных одной операцией деструктуризации свойств объекта forecast. Задай значение по умолчанию для иконок, переменных todayIcon и tomorrowIcon - строку "https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg".

* Объявлена переменная forecast
* Значение переменной forecast это объект
* Объявлена переменная highToday с помощью деструктуризации
* Значение переменной highToday это число 32
* Объявлена переменная lowToday с помощью деструктуризации
* Значение переменной lowToday это число 28
* Объявлена переменная todayIcon с помощью деструктуризации
* Значение переменной todayIcon это строка "https://www.flaticon.com/svg/static/icons/svg/861/861059.svg"
* Объявлена переменная highTomorrow с помощью деструктуризации
* Значение переменной highTomorrow это число 31
* Объявлена переменная lowTomorrow с помощью деструктуризации
* Значение переменной lowTomorrow это число 27
* Объявлена переменная tomorrowIcon с помощью деструктуризации
* Значение переменной tomorrowIcon это строка "https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg"
* Используется синтаксис деструктуризации объекта highTemperatures
* const forecast = {
* today: {
* low: 28,
* high: 32,
* icon: "https://www.flaticon.com/svg/static/icons/svg/861/861059.svg",
* },
* tomorrow: {
* low: 27,
* high: 31,
* },
* };
* // Change code below this line
* const {
* today: {
* high: highToday,
* low: lowToday,
* icon: todayIcon = 'https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg',
* },
* tomorrow: {
* low: lowTomorrow,
* high: highTomorrow,
* icon: tomorrowIcon = 'https://www.flaticon.com/svg/static/icons/svg/2204/2204346.svg',
* },
* } = forecast;

## 26/41

**Паттерн «Объект настроек»**

Функция calculateMeanTemperature(forecast) принимает один параметр forecast - объект температур на два дня следующего формата.

{

today: { low: 10, high: 20 },

tomorrow: { low: 20, high: 30 }

}

Замени объявления переменных todayLow, todayHigh, tomorrowLow и tomorrowHigh одной операцией деструктуризации свойств объекта forecast.

* Объявлена функция calculateMeanTemperature(forecast)
* В теле функции используется деструктуризация объекта
* В теле функции объявлена переменная todayHigh с помощью деструктуризации
* В теле функции объявлена переменная todayLow с помощью деструктуризации
* В теле функции объявлена переменная tomorrowLow с помощью деструктуризации
* В теле функции объявлена переменная tomorrowHigh с помощью деструктуризации
* Вызов calculateMeanTemperature({ today: {low: 28, high: 32}, tomorrow: {low: 25, high: 29} }) возвращает 28.5
* Вызов calculateMeanTemperature({ today: {low: 37, high: 40}, tomorrow: {low: 33, high: 38} }) возвращает 37
* // Change code below this line
* function calculateMeanTemperature(forecast) {
* const {
* today: {
* high: todayHigh,
* low: todayLow,
* },
* tomorrow: {
* low: tomorrowLow,
* high: tomorrowHigh,
* },
* } = forecast;
* // Change code above this line
* return (todayLow + todayHigh + tomorrowLow + tomorrowHigh) / 4;
* }

## 27/41

**Операция spread при передаче аргументов**

Синтаксис ... (spread) позволяет распылить коллекцию элементов (массив, строку или объект) в место, где ожидается набор отдельных значений. Конечно есть некоторые ограничения, например нельзя распылить массив в объект и наоборот.

Можно привести аналогию с ящиком яблок. Поставив ящик на пол не вынимая из него яблоки, получим аналог массива значений. Если высыпать яблоки из ящика на пол, произойдёт распыление - набор отдельных значений.

Отличие всего одно - в JavaScript распыление не изменяет оригинальную коллекцию, то есть делается копия каждого элемента. После распыления останется и ящик полный яблок, и копия каждого яблока на полу.

Например, метод Math.max(аргументы) ищет и возвращает самый большой из аргументов (чисел), то есть ожидает не массив значений, а произвольное количество аргументов.

const temps = [14, -4, 25, 8, 11];

// В консоли будет массив

console.log(temps);

// ❌ Так не сработает, потому что передаём целый массив

console.log(Math.max(temps)); // NaN

// В консоли будет набор отдельных чисел

console.log(...temps);

// ✅ Распылим коллекцию элементов как отдельные аргументы

console.log(Math.max(...temps)); // 25

То есть запись Math.max(...[14, -4, 25, 8, 11]), после интерпретации превращается в Math.max(14, -4, 25, 8, 11) - синтаксис ... возвращает распакованный массив, то есть распыляет его элементы как отдельные аргументы.

В переменной scores хранится массив результатов тестирования. Используя распыление и методы Math.max() и Math.min() дополни код так, чтобы в переменной bestScore был самый высокий балл, а в worstScore самый низкий.

* Объявлена переменная scores
* Значение переменной scores это массив [89, 64, 42, 17, 93, 51, 26]
* Объявлена переменная bestScore
* Значение переменной bestScore это число 93
* Объявлена переменная worstScore
* Значение переменной worstScore это число 17
* Для передачи аргументов методу Math.max() используется синтаксис ... на массиве scores
* Для передачи аргументов методу Math.min() используется синтаксис ... на массиве scores
* const scores = [89, 64, 42, 17, 93, 51, 26];
* // Change code below this line
* const bestScore = Math.max(...scores);
* const worstScore = Math.min(...scores);

## 28/41

**Операция spread при создании нового массива**

Операция spread позволяет создать копию массива или «склеить» произвольное количество массивов в один новый. Раньше для этого использовали методы slice() и concat(), но операция распыления позволяет сделать тоже самое в более краткой форме.

const temps = [14, -4, 25, 8, 11];

// Это точная, но независимая копия массива temps

const copyOfTemps = [...temps];

console.log(copyOfTemps); // [14, -4, 25, 8, 11]

В примере выше у нас есть ящик яблок temps и мы хотим сделать его точную копию. Берём пустой ящик и пересыпаем в него яблоки из исходного ящика temps - распыляем его в другую коллекцию. При этом ящик temps не изменится, в нём все ещё будут яблоки, а в новом ящике - их точные копии.

В следующем примере мы ссыпаем яблоки из двух ящиков в один новый. Оригинальные ящики (массивы) не изменятся, а в новом будут копии всех их яблок (элементов). Порядок распыления важен - он влияет на порядок элементов в новой коллекции.

const lastWeekTemps = [14, 25, 11];

const currentWeekTemps = [23, 17, 18];

const allTemps = [...lastWeekTemps, ...currentWeekTemps];

console.log(allTemps); // [14, 25, 11, 23, 17, 18]

В переменных firstGroupScores, secondGroupScores и thirdGroupScores хранятся результаты тестирования отдельных групп. Используя распыление дополни код так, чтобы:

* В переменной allScores хранился массив всех результатов от первой до третьей группы.
* В переменной bestScore был самый высокий общий балл.
* В переменной worstScore был самый низкий общий балл.
* Объявлена переменная firstGroupScores
* Значение переменной firstGroupScores это массив [64, 42, 93]
* Объявлена переменная secondGroupScores
* Значение переменной secondGroupScores это массив [89, 14, 51, 26]
* Объявлена переменная thirdGroupScores
* Значение переменной thirdGroupScores это массив [29, 47, 18, 97, 81]
* Объявлена переменная allScores.
* Значение переменной allScores это массив [64, 42, 93, 89, 14, 51, 26, 29, 47, 18, 97, 81]
* Объявлена переменная bestScore
* Значение переменной bestScore это число 97
* Объявлена переменная worstScore
* Значение переменной worstScore это число 14
* При присвоении значения переменной allScores использовался синтаксис ... для заполнения массива
* Для передачи аргументов методу Math.max() используется синтаксис ... на массиве allScores
* Для передачи аргументов методу Math.min() используется синтаксис ... на массиве allScores
* const firstGroupScores = [64, 42, 93];
* const secondGroupScores = [89, 14, 51, 26];
* const thirdGroupScores = [29, 47, 18, 97, 81];
* // Change code below this line
* const allScores = [...firstGroupScores, ...secondGroupScores, ...thirdGroupScores];
* const bestScore = Math.max(...allScores);
* const worstScore = Math.min(...allScores);

## 29/41

**Операция spread при создании нового объекта**

Операция spread позволяет распылить свойства произвольного количества объектов в один новый.

const first = { propA: 5, propB: 10 };

const second = { propC: 15 };

const third = { ...first, ...second };

console.log(third); // { propA: 5, propB: 10, propC: 15 }

Порядок распыления имеет значение. Имена свойств объекта уникальные, поэтому свойства распыляемого объекта могут перезаписать значение уже существующего свойства, если их имена совпадают.

const first = { propA: 5, propB: 10, propC: 50 };

const second = { propC: 15, propD: 20 };

const third = { ...first, ...second };

console.log(third); // { propA: 5, propB: 10, propC: 15, propD: 20 }

const fourth = { ...second, ...first };

console.log(fourth); // { propA: 5, propB: 10, propC: 50, propD: 20 }

Если бы яблоки в ящике имели наклейки с метками, то в одном ящике не может быть двух яблок с одинаковыми метками. Поэтому при пересыпании второго ящика, все яблоки, метки которых совпадут с теми что уже есть в новом, заменят те что уже есть.

Во время распыления можно добавлять свойства в произвольное место. Главное помнить про уникальность имени свойства и о том, что его значение может быть перезаписано.

const first = { propA: 5, propB: 10, propC: 50 };

const second = { propC: 15 };

const third = { propB: 20, ...first, ...second };

console.log(third); // { propA: 5, propB: 10, propC: 15 }

const fourth = { ...first, ...second, propB: 20 };

console.log(fourth); // { propA: 5, propB: 20, propC: 15 }

const fifth = { ...first, propB: 20, ...second };

console.log(fifth); // { propA: 5, propB: 20, propC: 15 }

В конструкторе можно создавать новые тесты, для которых есть настройки по умолчанию которые хранятся в переменной defaultSettings. Во время создания теста, все или часть настроек можно переопределить, они хранятся в переменной overrideSettings.

Для того чтобы получить финальные настройки теста, необходимо взять настройки по умолчанию и поверх них применить переопределённые настройки. Дополни код так, чтобы в переменной finalSettings получился объект финальных настроек теста.

* Объявлена переменная defaultSettings
* Значение переменной defaultSettings это объект
* Объявлена переменная overrideSettings
* Значение переменной overrideSettings это объект
* Объявлена переменная finalSettings
* Значение переменной finalSettings это объект
* Значение свойства finalSettings.theme равно "light"
* Значение свойства finalSettings.public равно "false"
* Значение свойства finalSettings.withPassword равно "true"
* Значение свойства finalSettings.minNumberOfQuestions равно 10
* Значение свойства finalSettings.timePerQuestion равно 30
* При присваивании значения переменной finalSettings используется синтаксис ...
* const defaultSettings = {
* theme: "light",
* public: true,
* withPassword: false,
* minNumberOfQuestions: 10,
* timePerQuestion: 60,
* };
* const overrideSettings = {
* public: false,
* withPassword: true,
* timePerQuestion: 30,
* };
* // Change code below this line
* const finalSettings = {...defaultSettings, ...overrideSettings};

## 30/41

**Задача. Карточки задач**

Напиши функцию makeTask(data) которая принимает один параметр data - объект со следующими свойствами.

* text - текст задачи.
* category - категория задачи.
* priority - приоритет задачи.

Функция должна составить и вернуть новый объект задачи, не изменяя напрямую параметр data. В новом объекте должно быть свойство completed, значение которого хранится в одноимённой локальной переменной.

В параметре data гарантированно будет только свойство text, а остальные два, category и priority, могут отсутствовать. Тогда, в новом объекте задачи, в свойствах category и priority должны быть значения по умолчанию, хранящиеся в одноимённых локальных переменных.

* Объявлена функция makeTask(data)
* Вызов makeTask({}) возвращает { category: "General", priority: "Normal", completed: false }
* Вызов makeTask({ category: "Homemade", priority: "Low", text: "Take out the trash" }) возвращает { category: "Homemade", priority: "Low", text: "Take out the trash", completed: false }
* Вызов makeTask({ category: "Finance", text: "Take interest" }) возвращает { category: "Finance", priority: "Normal", text: "Take interest", completed: false }
* Вызов makeTask({ priority: "Low", text: "Choose shampoo" }) возвращает { category: "General", priority: "Low", text: "Choose shampoo", completed: false }
* Вызов makeTask({ text: "Buy bread" }) возвращает { category: "General", priority: "Normal", text: "Buy bread", completed: false }
* function makeTask(data) {
* const completed = false;
* const category = "General";
* const priority = "Normal";
* // Change code below this line
* // Change code above this line
* }

**31/41**

**Операция rest для сбора всех аргументов функции**

Операция ... (rest) позволяет собрать группу независимых элементов в новую коллекцию. Синтаксически это близнец операции распыления, но отличить их просто - распыление это когда ... находится в **правой** части операции присваивания, а сбор это когда ... находится в её **левой** части.

Вернёмся к аналогии с яблоками. Если на полу лежат яблоки и у нас есть пустой ящик, то операция rest позволит «собрать» яблоки в ящик. При этом оригинальные яблоки останутся на полу, а в ящике будет копия каждого яблока.

Одна из областей применения операции rest это создание функций которые могут принимать произвольное количество аргументов.

// Как объявить параметры функции так,

// чтобы можно было передать любое кол-во аргументов?

function multiply() {

// ...

}

multiply(1, 2);

multiply(1, 2, 3);

multiply(1, 2, 3, 4);

Если убрать весь «синтаксический шум» и посмотреть на аргументы и параметры функции, то аргументы находятся в правой части операции присваивания, а параметры в левой, потому что значения аргументов присваиваются объявленным параметрам. Значит можно «собрать» все аргументы функции в один параметр используя операцию rest.

function multiply(...args) {

console.log(args); // массив всех аргументов

}

multiply(1, 2);

multiply(1, 2, 3);

multiply(1, 2, 3, 4);

Имя параметра может быть произвольным. Чаще всего его называют args, restArgs или otherArgs, сокращённое от arguments.

Используя операцию rest дополни код функции add() так, чтобы она принимала любое количество аргументов, считала и возвращала их сумму.

* Объявлена функция add
* Функция add использует параметр args
* Для сбора аргументов в переменную args, в подписи функции используется синтаксис ... (оперетор rest)
* Вызов add(15, 27) возвращает 42
* Вызов add(12, 4, 11, 48) возвращает 75
* Вызов add(32, 6, 13, 19, 8) возвращает 78
* Вызов add(74, 11, 62, 46, 12, 36) возвращает 241
* // Change code below this line
* function add(...args) {
* let sum = 0;
* for (const item of args) {
* sum += item;
* }
* return sum;
* // Change code above this line
* }

**32/41**

**Операция rest для сбора части аргументов функции**

Операция ... (rest) также позволяет собрать в массив только ту часть аргументов, которая необходима, объявив параметры до «сбора».

function multiply(firstNumber, secondNumber, ...otherArgs) {

console.log(firstNumber); // Значение первого аргумента

console.log(secondNumber); // Значение второго аргумента

console.log(otherArgs); // Массив остальных аргументов

}

multiply(1, 2);

multiply(1, 2, 3);

multiply(1, 2, 3, 4);

Все аргументы, для которых будут объявлены параметры, передадут им свои значения, остальные аргументы будут помещены в массив. Операция rest собирает все оставшиеся аргументы и поэтому должна идти последней в подписи функции, иначе будет ошибка.

Функция addOverNum() считает сумму всех аргументов. Измени параметры и тело функции addOverNum() так, чтобы она считала сумму только тех аргументов, которые больше чем заданное число. Это число должно быть первым параметром функции.

* Объявлена функция addOverNum()
* Вызов addOverNum(50, 15, 27) возвращает 0
* Вызов addOverNum(10, 12, 4, 11, 48, 10, 8) возвращает 71
* Вызов addOverNum(15, 32, 6, 13, 19, 8) возвращает 51
* Вызов addOverNum(20, 74, 11, 62, 46, 12, 36) возвращает 218
* // Change code below this line
* function addOverNum(firstarg, ...args) {
* let total = 0;
* for (const arg of args) {
* if (firstarg < arg){
* total += arg;
* }
* }
* return total;
* // Change code above this line
* }

**33/41**

**Задача. Массив совпадений**

Функция findMatches() принимает произвольное количество аргументов. Первым аргументом всегда будет массив чисел, а остальные аргументы будут просто числами.

Дополни код функции так, чтобы она возвращала новый массив matches, в котором будут только те аргументы, начиная со второго, которые есть в массиве первого аргумента.

Например, findMatches([1, 2, 3, 4, 5], 1, 8, 2, 7) должна вернуть массив [1, 2], потому что только они есть в массиве первого аргумента.

* Объявлена функция findMatches()
* Вызов findMatches([1, 2, 3, 4, 5], 1, 8, 2, 7) возвращает [1, 2]
* Вызов findMatches([4, 89, 17, 36, 2], 8, 17, 89, 27, 2) возвращает [17, 89, 2]
* Вызов findMatches([10, 24, 41, 6, 9, 19], 24, 11, 9, 23, 41) возвращает [24, 9, 41]
* Вызов findMatches([63, 11, 8, 29], 4, 7, 16) возвращает []
* // Change code below this line
* function findMatches(first, ...rest) {
* const matches = []; // Don't change this line
* for (i = 0; i < first.length; i += 1){
* if (rest.includes(first[i])) {
* matches.push(first[i]);
* }
* }
* // Change code above this line
* return matches;
* }

**34/41**

## Методы объекта

До сих пор мы рассматривали объекты только как хранилища взаимосвязанных данных, например информация о книге и т. п. Объекты-хранилища обычно находятся в массиве таких же объектов, который представляет коллекцию однотипных элементов.

Объекты могут хранить не только данные, но и функции для работы с этими данными - методы. Если значение свойства это функция, такое свойство называется методом объекта.

// ✅ Логиески и синтаксически сгруппированные сущности

const bookShelf = {

books: ["Последнее королевство", "Страж снов"],

// Это метод объекта

getBooks() {

console.log("Этот метод будет возвращать все книги - свойство books");

},

// Это метод объекта

addBook(bookName) {

console.log("Этот метод будет добавлять новую книгу в свойство books");

},

};

// Вызовы методов

bookShelf.getBooks();

bookShelf.addBook("Новая книга");

Такие объекты можно назвать «моделями». Они связывают данные и методы для работы с этими данными. Например, можно было объявить переменную books и две функции getBooks() и addBook(bookName), но тогда это были бы три независимые сущности без явной синтаксической, и со слабой логической связями.

// ❌ Слабосвязанные, независмые сущности

const books = [];

function getBooks() {}

function addBook() {}

Добавь объекту bookShelf ещё два метода, которые пока что будут возвращать просто строки по аналогии с getBooks() и addBook(bookName).

Метод removeBook(bookName) будет удалять книгу по имени. Возвращает строку "Deleting book <имя книги>", где <имя книги> это значение параметра bookName.

Метод updateBook(oldName, newName) будет обновлять название книги на новое. Возвращает строку "Updating book <старое имя> to <новое имя>", где <старое имя> и <новое имя>это значения параметров oldName и newName соотвественно.

* Объявлена переменная bookShelf
* Значение переменной bookShelf это объект
* Значение свойства bookShelf.getBooks это метод объекта
* Вызов метода bookShelf.getBooks() возвращает строку "Returning all books"
* Значение свойства bookShelf.addBook это метод объекта
* Вызов метода bookShelf.addBook("Haze") возвращает строку "Adding book Haze"
* Значение свойства bookShelf.removeBook это метод объекта
* Вызов метода bookShelf.removeBook("Red sunset") возвращает строку "Deleting book Red sunset"
* Значение свойства bookShelf.updateBook это метод объекта
* Вызов метода bookShelf.updateBook("Sands of dune", "Dune") возвращает строку "Updating book Sands of dune to Dune"
* const bookShelf = {
* // Change code below this line
* books: ["The last kingdom", "The guardian of dreams"],
* getBooks() {
* return "Returning all books";
* },
* addBook(bookName) {
* return `Adding book ${bookName}`;
* },
* removeBook(bookName) {
* return `Deleting book ${bookName}`;
* },
* updateBook(oldName, newName) {
* return `Updating book ${oldName} to ${newName}`;
* }
* // Change code above this line
* };

**35/41**

## Доступ к свойствам объекта в его методах

Методы используются для работы со свойствами объекта, их изменения. Для доступа к объекту в методе используется не имя переменной, например bookShelf, а ключевое слово this - контекст. Значением this будет объект перед «точкой», то есть объект который вызвал этот метод, в нашем случае это ссылка на объект bookShelf.

const bookShelf = {

books: ["Последнее королевство"],

getBooks() {

console.log(this);

},

};

// Перед точкой стоит объект bookShelf,

// поэтому при вызове метода, this будет хранить ссылку на него.

bookShelf.getBooks(); // {books: ["Последнее королевство"], getBooks: f}

Для того, чтобы получить доступ к свойствам объекта в методах, мы обращаемся к нему через this и дальше как обычно - «через точку» к свойствам.

const bookShelf = {

books: ["Последнее королевство"],

getBooks() {

return this.books;

},

addBook(bookName) {

this.books.push(bookName);

},

removeBook(bookName) {

const bookIndex = this.books.indexOf(bookName);

this.books.splice(bookIndex, 1);

},

};

console.log(bookShelf.getBooks()); // []

bookShelf.addBook("Мгла");

bookShelf.addBook("Страж снов");

console.log(bookShelf.getBooks()); // ["Последнее королевство", "Мгла", "Страж снов"]

bookShelf.removeBook("Мгла");

console.log(bookShelf.getBooks()); // ["Последнее королевство", "Страж снов"]

Будет логично задуматься - почему бы не использовать при обращении к свойствам имя объекта, ведь мы явно не собираемся его менять. Дело в том, что имя объекта штука не надежная, методы одного объекта можно копировать в другой (с другим именем), а в будущем узнаем, что часто, при создании объекта мы заранее вовсе не знаем имени. Использование this гарантирует, что метод работает именно с тем объектом, который его вызвал.

Дополни метод updateBook(oldName, newName) так, чтобы он изменял название книги с oldName на newName в свойстве books. Используй indexOf() для того, чтобы найти нужный элемент массива, и splice() для того чтобы заменить этот элемент

* Объявлена переменная bookShelf
* Значение переменной bookShelf это объект
* Значение свойства bookShelf.updateBook это метод объекта
* После вызова метода bookShelf.updateBook("Haze", "Dungeon chronicles"), значение свойства books это массив ["The last kingdom", "Dungeon chronicles", "The guardian of dreams"]
* После вызова метода bookShelf.updateBook("The last kingdom", "Dune"), значение свойства books это массив ["Dune", "Haze", "The guardian of dreams"]
* const bookShelf = {
* books: ["The last kingdom", "Haze", "The guardian of dreams"],
* updateBook(oldName, newName) {
* // Change code below this line
* this.books.indexOf(oldName);
* this.books.splice(this.books.indexOf(oldName), 1, newName);
* // Change code above this line
* }
* };

**36/41**

**Задача. Лавка зелий «У старой жабы»**

К нам обратилась владелица лавки зелий «У старой жабы» и заказала программу для ведения инвентаря - добавления, удаления, поиска и обновления зелий. Добавь объекту atTheOldToad свойство potions, значением которого сделай пустой массив.

* Объявлена переменная atTheOldToad
* Значение переменной atTheOldToad это объект
* Значение свойства atTheOldToad.potions это массив []
* const atTheOldToad = {
* // Change code below this line
* potions: [],
* // Change code above this line
* };

**37/41**

**Задача. Получаем все зелья**

Добавь объекту atTheOldToad метод getPotions(), который просто возвращает значение свойства potions.

* Объявлена переменная atTheOldToad
* Значение переменной atTheOldToad это объект
* Значение свойства atTheOldToad.potions это массив ["Speed potion", "Dragon breath", "Stone skin"]
* Значение свойства atTheOldToad.getPotions это метод объекта
* Вызов метода atTheOldToad.getPotions() возвращает ["Speed potion", "Dragon breath", "Stone skin"]
* const atTheOldToad = {
* potions: ["Speed potion", "Dragon breath", "Stone skin"],
* // Change code below this line
* getPotions(){
* return this.potions
* },
* // Change code above this line
* };

**38/41**

**Задача: добавляем новое зелье**

Дополни метод addPotion(potionName) так, чтобы он добавлял зелье potionName в конец массива зелий в свойстве potions.

* Объявлена переменная atTheOldToad
* Значение переменной atTheOldToad это объект
* Значение свойства atTheOldToad.potions это массив ["Speed potion", "Dragon breath", "Stone skin"]
* Значение свойства atTheOldToad.addPotion это метод объекта
* После первого вызова метода atTheOldToad.addPotion("Invisibility"), в свойстве potions будет массив ["Speed potion", "Dragon breath", "Stone skin", "Invisibility"]
* После второго вызова метода atTheOldToad.addPotion("Power potion"), в свойстве potions будет массив ["Speed potion", "Dragon breath", "Stone skin", "Invisibility", "Power potion"]
* const atTheOldToad = {
* potions: ["Speed potion", "Dragon breath", "Stone skin"],
* addPotion(potionName) {
* // Change code below this line
* this.potions.push(potionName)
* // Change code above this line
* },
* };

**39/41**

**Задача. Удаляем зелье**

Дополни метод removePotion(potionName) так, чтобы он удалял зелье potionName из массива зелий в свойстве potions.

* Объявлена переменная atTheOldToad
* Значение переменной atTheOldToad это объект
* Значение свойства atTheOldToad.potions это массив ["Speed potion", "Dragon breath", "Stone skin"]
* Значение свойства atTheOldToad.removePotion это метод объекта
* После первого вызова метода atTheOldToad.removePotion("Dragon breath"), в свойстве potions будет массив ["Speed potion", Stone skin"]
* После второго вызова метода atTheOldToad.removePotion("Speed potion"), в свойстве potions будет массив ["Stone skin"]
* const atTheOldToad = {
* potions: ["Speed potion", "Dragon breath", "Stone skin"],
* removePotion(potionName) {
* // Change code below this line
* this.potions.indexOf(potionName);
* this.potions.splice(this.potions.indexOf(potionName)
* , 1);
* // Change code above this line
* },
* };

**40/41**

**Задача: обновляем зелье**

Дополни метод updatePotionName(oldName, newName) так, чтобы он обновлял название зелья с oldName на newName, в массиве зелий в свойстве potions.

* Объявлена переменная atTheOldToad
* Значение переменной atTheOldToad это объект
* Значение свойства atTheOldToad.potions это массив ["Speed potion", "Dragon breath", "Stone skin"]
* Значение свойства atTheOldToad.updatePotionName это метод объекта
* После первого вызова метода atTheOldToad.updatePotionName("Dragon breath", "Polymorth"), в свойстве potions будет массив ["Speed potion", "Polymorth", "Stone skin"]
* После второго вызова метода atTheOldToad.updatePotionName("Stone skin", "Invisibility"), в свойстве potions будет массив ["Speed potion", "Polymorth", "Invisibility"]
* const atTheOldToad = {
* potions: ["Speed potion", "Dragon breath", "Stone skin"],
* updatePotionName(oldName, newName) {
* // Change code below this line
* this.potions.indexOf(oldName);
* this.potions.splice(this.potions.indexOf(oldName), 1, newName);
* // Change code above this line
* },
* };

**41/41**

**Задача: расширяем инвентарь**

Заказчица хочет чтобы каждое зелье было представлено не только именем, но и ценой, а в будущем может быть и другими характеристиками. Поэтому теперь в свойстве potions будет храниться массив объектов со следующими свойствами.

{

name: "Dragon breath",

price: 700

}

Выполни рефакторинг методов объекта atTheOldToad так, чтобы они работали не с массивом строк, а с массивом объектов.

* getPotions() - метод для получения всех зелий. Возвращает значение свойства potions.
* addPotion(newPotion) - добавляет зелье newPotion (уже объект) в массив в свойстве potions, но только если такого зелья еще нет в инвентаре. В противном случае возвращается строка.
* removePotion(potionName) - удаляет объект зелья с именем potionName из массива в свойстве potions.
* updatePotionName(oldName, newName) - обновляет свойство name объекта-зелья с названием oldName на newName в массиве potions.
* Объявлена переменная atTheOldToad
* Значение переменной atTheOldToad это объект
* Значение свойства atTheOldToad.getPotions это метод объекта
* Вызов метода atTheOldToad.getPotions() для исходного объекта возвращает [ { name: "Speed potion", price: 460 }, { name: "Dragon breath", price: 780 }, { name: "Stone skin", price: 520 } ]
* Значение свойства atTheOldToad.addPotion это метод объекта.
* Для исходного объекта после вызова метода atTheOldToad.addPotion({ name: "Invisibility", price: 620 }), в массиве potions последним элементом будет этот объект
* Для исходного объекта после вызова метода atTheOldToad.addPotion({ name: "Power potion", price: 270 }), в массиве potions последним элементом будет этот объект
* Если добавляемое зелье уже есть в массиве potions, метод addPotion возвращает строку с текстом из исходного кода
* Если добавляемое зелье уже есть в массиве potions, метод addPotion не добавляет в него передаваемый обьект
* Для исходного объекта после вызова atTheOldToad.addPotion({ name: "Dragon breath", price: 700 }), массив potions не изменяется
* Для исходного объекта после вызова atTheOldToad.addPotion({ name: "Stone skin", price: 240 }), массив potions не изменяется
* Для исходного объекта вызов atTheOldToad.addPotion({ name: "Dragon breath", price: 700 }), возвращает строку "Error! Potion Dragon breath is already in your inventory!"
* Для исходного объекта вызов atTheOldToad.addPotion({ name: "Stone skin", price: 240 }), возвращает строку "Error! Potion Stone skin is already in your inventory!"
* Значение свойства atTheOldToad.removePotion это метод объекта
* Для исходного объекта после вызова метода atTheOldToad.removePotion("Dragon breath"), в свойстве potions будет массив [ { name: "Speed potion", price: 460 }, { name: "Stone skin", price: 520 } ]
* Для исходного объекта после вызова метода atTheOldToad.removePotion("Speed potion"), в свойстве potions будет массив [ { name: "Dragon breath", price: 780 }, { name: "Stone skin", price: 520 }]
* Значение свойства atTheOldToad.updatePotionName это метод объекта
* Для исходного объекта после вызова метода atTheOldToad.updatePotionName("Dragon breath", "Polymorth"), в свойстве potions будет массив [{ name: "Speed potion", price: 460 }, { name: "Polymorth", price: 780 }, { name: "Stone skin", price: 520 } ]
* Для исходного объекта после вызова метода atTheOldToad.updatePotionName("Stone skin", "Invulnerability potion"), в свойстве potions будет массив [{ name: "Speed potion", price: 460 }, { name: "Dragon breath", price: 780 }, { name: "Invulnerability potion", price: 520 } ]
* const atTheOldToad = {
* potions: [
* { name: 'Speed potion', price: 460 },
* { name: 'Dragon breath', price: 780 },
* { name: 'Stone skin', price: 520 },
* ],
* // Change code below this line
* getPotions() {
* return this.potions;
* },
* addPotion(newPotion) {
* for (const potion of this.potions) {
* if (newPotion.name === potion.name) {
* return `Error! Potion ${potion.name} is already in your inventory!`;
* }
* }
* this.potions.push(newPotion);
* },
* removePotion(potionName) {
* let potionIndex = -1;
* for (i = 0; i < this.potions.length; i += 1) {
* if (potionName === this.potions[i].name) {
* potionIndex = i;
* break;
* }
* }
* const potionIndex1 = this.potions.indexOf(potionName);
* if (potionIndex === -1) {
* return `Potion ${potionName} is not in inventory!`;
* }
* this.potions.splice(potionIndex, 1);
* },
* updatePotionName(oldName, newName) {
* const { potions } = this;
* for (const potion of potions) {
* if (potion.name === oldName) {
* potion.name = newName;
* return;
* }
* }
* return `Potion ${oldName} is not in inventory!`;
* }
* // Change code above this line
* };