Front-end

JavaScript

JS. Object

- Объекты

JS. Object

Копирование и сравнение объектов

Переменная, содержащая объект на самом деле содержит не сам объект, а только ссылку на него. При копировании объектов в отличие от значений примитивных типов происходит передача ссылки.

```
// присвоим переменной student1 объект, а точнее ссылку на него

const student1 = { name: 'Carl' };

// присвоим объект, содержащийся в student1 переменной student2

const student2 = student1;
```

JS. Object

Копирование и сравнение объектов

А что если нам необходимо скопировать не саму ссылку, а создать новый объект с такими же свойствами

```
const student3 = {};

for (const key in student1) {

 student3[key] = student1[key];

}
```

```
const person = {
    name: 'John',
    age: 20
};

const objCopy = {}
const objCopy2 = {...person}

for (const key in person) {
    objCopy[key] = person[key]
}

console.log(objCopy)
```

JS. Object

Копирование и сравнение объектов

Другой способ скопировать свойства – это воспользоваться методом Object.assign()

```
const student4 = Object.assign({}, student1)
```

JS. Object

Копирование и сравнение объектов

Object.assign() позволяет скопировать свойства из множества объектов. Объект, в который нужно скопировать указывается в качестве первого аргумента, а те из которых – после него

```
const target = { a: 1 };

const source1 = { b: 2 };

const source2 = { c: 3 };


Object.assign(target, source1, source2);

console.log(target); // {a: 1, b: 2, c: 3}
```

JS. Object

Проверка на наличие ключа в объекте

Оператор in возвращает true, если свойство содержится в указанном объекте или в его цепочке прототипов.

Синтаксис

prop in object

Пример: ‘name’ in user

JS. Object

Проверка на наличие ключа в объекте

Метод hasOwnProperty() возвращает логическое значение, которое указывает на то содержит ли объект указанное собственное (неунаследованное) свойство, или метод.

Метод возвращает true в том случае, если объект имеет неунаследованное свойство с указанным именем и false, если объект не имеет свойства с указанным именем или если это свойство он наследует от своего объекта прототипа.

В отличие от оператора in метод hasOwnProperty() не проверяет существование свойств в цепочке прототипов объекта.

JS. Object

Проверка на наличие ключа в объекте

```
const obj = {
myProperty: 1000
};


obj.hasOwnProperty( "myProperty" );    // true
```

JS. Object

Опциональная цепочка

Опциональная цепочка ?. — это безопасный способ доступа к свойствам вложенных объектов, даже если какое-либо из промежуточных свойств не существует.

Опциональная цепочка ?. останавливает вычисление и возвращает undefined, если значение перед ?. равно undefined или null.

user?.address?.street

JS. Object

Перебор объекта

Object.keys, values, entries

Для простых объектов доступны следующие методы:

Object.keys(obj) – возвращает массив ключей

Object.values(obj) – возвращает массив значений

Object.entries(obj) – возвращает массив пар [ключ, значение]

JS. Object

Перебор объекта

```
let user = {
  name: "John",
  age: 30
};
```

Object.keys(user) = ["name", "age"]

Object.values(user) = ["John", 30]

Object.entries(user) = [["name","John"], ["age",30]]

JS. Object

Перебор объекта

for...in позволяет пройти в цикле по перечисляемым свойствам объекта, в том числе по свойствам из прототипа.

```
for (переменная in объект) {
  // действия внутри цикла
}
const cat = {
  name: 'Борис',
  age: 8
}
for (const key in cat) {
  console.log(`${key} – ${cat[key]}`)
}// name – 'Борис', age – 8
```

JS. Object

Ключевое слово this

Как правило, методу объекта обычно требуется доступ к информации, хранящейся в объекте, для выполнения своей работы.

Для доступа к информации внутри объекта метод может использовать ключевое слово this.

Указывает на текущий контекст выполнения кода.

Чаще всего является объектом.

this – объект перед точкой

JS. Object

Ключевое слово this

У стрелочных функций нет «this»

Стрелочные функции особенные: у них нет своего «собственного» this. Если мы ссылаемся на this внутри такой функции, то оно берётся из внешней «нормальной» функции.

JS. Object

Ключевое слово this

```
let group = {
  title: "Our Group",
  students: ["John", "Pete", "Alice"],
  showList() {
    this.students.forEach(
      student => alert(this.title + ': ' + student)
    );
  }
};
group.showList();
```

JS. Object

call apply bind – методы функций, работают с this

Функции - это особый тип объектов, поскольку они имеют множество встроенных свойств и методов, три из которых - call(), apply() и bind()

Эти методы помогают явно указать this

JS. Object

call

Синтаксис метода call:

func.call(context, arg1, arg2, ...)

При этом вызывается функция func, первый аргумент call становится её this, а остальные передаются «как есть».

JS. Object

apply

Если нам неизвестно, с каким количеством аргументов понадобится вызвать функцию, можно использовать более мощный метод: apply.

Вызов функции при помощи func.apply работает аналогично func.call, но принимает массив аргументов вместо списка.

```
func.call(context, arg1, arg2);

// идентичен вызову

func.apply(context, [arg1, arg2]);
```

Преимущество apply перед call отчётливо видно, когда мы формируем массив аргументов динамически.

JS. Object

Привязка контекста к функции

При передаче методов объекта в качестве колбэков, например для setTimeout, возникает проблема – потеря this.

Потеря «this» - как только метод передаётся отдельно от объекта – this теряется.

Вот как это может произойти в случае с setTimeout:

```
let user = {
 firstName: "Вася",
 sayHi() {
  alert(`Привет, ${this.firstName}!`);
 }
};
setTimeout(user.sayHi, 1000); // Привет, undefined!
```

JS. Object

Привязка контекста к функции

При запуске этого кода мы видим, что вызов this.firstName возвращает не «Вася», а undefined!

Это произошло потому, что setTimeout получил функцию sayHi отдельно от объекта user (именно здесь функция и потеряла контекст).

JS. Object

```
let user = {
  firstName: "Вася"
};

function func() {
  alert(this.firstName);
}

let funcUser = func.bind(user);
funcUser(); // Вася
```

JS. Object

Метод bind() создаёт новую функцию, которая при вызове устанавливает в качестве контекста выполнения this предоставленное значение. В метод также передаётся набор аргументов, которые будут установлены перед переданными в привязанную функцию аргументами при её вызове.

bind не вызовет функцию, а создаст новую функцию

JS. Object

Деструктуризация объектов

Деструктуризация помогает сделать сложную структуру более простой, разделить (деструктурировать) на более мелкие части, например, объект.

```
const obj = {
  firstName: 'John',
  lastName: 'Doe',
  age: 28
}


const { firstName, lastName, age } = obj;
```

JS. Object

Деструктуризация вложенных объектов

Принцип тот же, только вытаскиваем свойство не из родительского объекта, а из его вложенного объекта.

```
const obj = {
  firstName: 'John',
  lastName: 'Doe',
  bio: {
    birth: '20.08.1994',
    age: 28,
    hasWork: true,
    isMarried: false
  }
}

const { birth, age, hasWork, isMarried } = obj.bio;
```

JS. Object

Деструктуризация вложенных объектов

Также если вам необходимо получить не только вложенные свойства. Допустим из примера выше, вам нужны firstName и age:

```
const { firstName, bio: { age } } = obj;

console.log(firstName, age); // John 28
```

JS. Object

Значение по умолчанию (дефолтные) при деструктуризации

Если вы не уверены в наличии какого-либо свойства в объекте, вы можете установить значение по умолчанию:

```
const obj = {
  firstName: 'John',
  lastName: 'Doe',
  age: 28
}

const { isMarried = false } = obj;

console.log(isMarried); // false
```