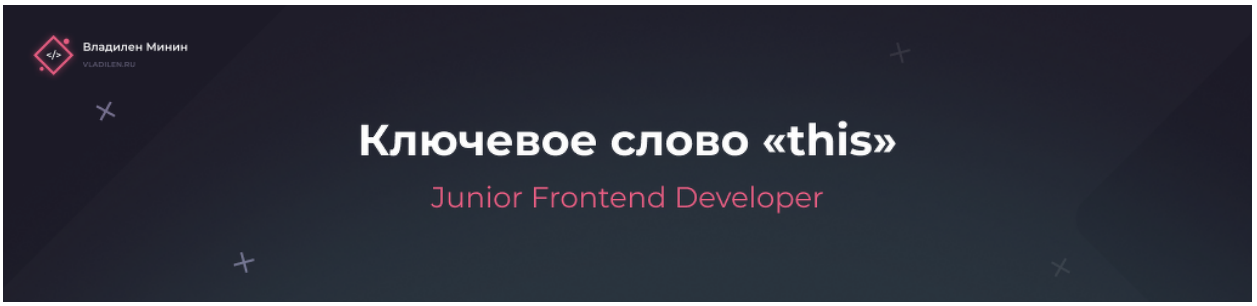

Цель урока:

Вы узнаете, что такое ключевое слово «this», зачем оно используется и как с ним работать.

Содержание урока:

1. Что такое this?
2. Потеря контекста
3. Привязка контекста при помощи bind, call и apply

Дополнительные материалы:

1. Методы объекта и ключевое слово this: https://learn.javascript.ru/ob…
2. JavaScript. Что такое контекст this. Как работает call, bind, apply: https://youtu.be/UGapN-hrekw
3. Как создать свой bind (4 способа + call, apply): https://youtu.be/fJqYa3BuwaU

Задание #1

Представьте, что вас как веб-разработчика попросили разработать веб-платформу, которая будет отслеживать прогресс студентов в обучении веб-разработке.

Для **начала вам необходимо** создать объект **student**, в котором будут 3 свойства:

1. stack. Массив из строк, где каждая строка - это технология. Изначально, массив stack должен быть равен [‘HTML’].
2. level. Тип данных number. level отвечает за текущий уровень студента и изначально равен 1.
3. improveLevel. Функция, которая сначала увеличивает значение свойства level у студента на единицу. Если level равен 2-м, то вам необходимо добавить в конец массива stack значение ‘CSS’, если 3-м - добавляете в конец stack значение ‘JavaScript’, 4-м - ‘React’, 5-ти - ‘NodeJS’. Если значение level стало больше 5-ти, то вам необходимо вывести в модальном окне через alert сообщение “Студент выучил все технологии!”. Также функция improveLevel должна возвращать в самом конце обновленный объект student.

В итоге значение свойства **stack** после выполнения кода ниже должно быть равно ["HTML", "CSS", "JavaScript", "React", "NodeJS"].

```
student
   .improveLevel()
```

```
    .improveLevel()
    .improveLevel()
    .improveLevel()
    .improveLevel()
```

Примечание: цепочка из подряд идущих функций improveLevel работает так, потому что improveLevel возвращает объект student, а у объекта student есть метод improveLevel.

Посмотрите на пример:

```
const student = {
    improveLevel() {
        return this;
    }
};

student
    .improveLevel()
    .improveLevel()
    .improveLevel();
```

Задание #2

У вас есть 2 объекта **dog** и **bird**, в которых содержится их описание.

```
const dog = {
    name: 'Чарли',
    type: 'Собака',
    makeSound() {
        return 'Гав-Гав';
    }
}

const bird = {
    name: 'Петя',
    type: 'Воробей',
    makeSound() {
        return 'Чик-чирик';
    }
}
```

Вам необходимо создать функцию **makeDomestic**, которая будет работать с ключевым словом **this** (пример использования ниже). Она должна выводить в консоль всю информацию о животном с помощью сообщения: “**type** по имени **name** говорит **sound**”, где **name** - имя животного, **type** - его тип и **sound** - результат вызова функции makeSound.

Также функция **makeDomestic** должна возвращать новый объект животного, в котором будут все прошлые параметры (name, type, makeSound), а также будет добавлен новый параметр **isDomestic**. Он является типом данных **boolean** и показывает, домашнее ли

животное или нет. Значение для свойства **isDomestic** (true либо false) вам необходимо передать как параметр в функцию **makeDomestic**.

Кроме этого для обращения к свойствам животного функция **makeDomestic** должна использовать только **this**. Т.е. вам разрешено обращаться к свойствам только через **this.name** либо **this.makeSound**, но никак не **bird.name** либо **bird.makeSound**.

Вам необходимо решить эту задачу 3-мя способами: через **bind, call и apply**. Вызовите функцию **makeDomestic** 3 раза используя данные методы. При использовании **bind**, функция **makeDomestic** должна работать с объектом dog, при **apply** и **call** - с объектом **bird**. Не забывайте передавать параметр **isDomestic** в функцию при ее вызове.

Примеры использования:

```
function makeDomestic(isDomestic) {
    // Ваша реализация
 }

/*
Сообщение в консоли: "Собака по имени Чарли говорит Гав-Гав"

domesticDog = {
  name: 'Чарли',
  type: 'Собака',
  isDomestic: true,
  makeSound() {
    return 'Гав-Гав'
  }
}
*/
```

Задание #3

Дан объект **footballer**, в котором есть некоторая информация о футболисте и действия, которые он может выполнять.

```
const footballer = {
    fullName: 'Cristiano Ronaldo',
    attack: () => {
        console.log(`${this.fullName} сейчас с мячом и начинает атаку!`);
    },
    scoreGoal(sound) {
        console.log(`${this.fullName} забил гол! Вот это да!`);
        this.celebrate(sound);
    },
    celebrate(sound) {
        console.log(sound);
    },
    goToSubstitution: function(newPlayer) {
```

```
        console.log(`${this.fullName} уходит на замену.
        На поле выходит ${newPlayer}`);
    }
};

const attack = footballer.attack;
const score = footballer.scoreGoal;
const substitute = footballer.goToSubstitution;
attack();
score('Сиииии');
substitute('Paulo Dibala');
```

К сожалению, данный код работает неверно. В некоторых случаях он выводит undefined, а в других - вообще ошибку.

Вам необходимо исправить данный код используя **bind, call и apply**.

- Функцию **attack** необходимо исправить при помощи **bind** + нужно вспомнить особенность стрелочных функций (нет своего **this**)
- Функцию **score** - при помощи **call**
- Функцию **substitute** - при помощи **apply**

Задание #4 (*дополнительное*)

К вам пришел один владелец небольшого стартапа. Он предлагает вам сделать игру, которая поразит игровую индустрию. Вы конечно же соглашаетесь, так как предложение звучит заманчиво.

Суть игры состоит в том, что есть 2 замка (атакующий и защищающийся), которые враждуют между собой. Пользователь играет за атакующий замок. Царю атакующего замка необходимо каждый раз просчитывать возможности захвата враждующего замка. Когда возможность будет достаточно большой, то необходимо не задумываясь атаковать.

В коде у вас изначально есть 2 объекта **attacker** и defender, которые являются аналогиями замков и у которых есть свойства, отвечающие за количество боевых единиц каждого типа. Например, **archer** - это лучники, **footSoldiers** - пехотинцы, и т.д.

```
const attacker = {
    archer: 30,
    footSoldier: 55,
    cavalry: 10,
    artillery: 3,
}

const defender = {
    archer: 33,
    footSoldier: 50,
    cavalry: 40,
    artillery: 10,
}
```

В объекте attacker вам необходимо создать 3 функции:

1. **checkChancesToWin** Проверяет шансы атакующего замка захватить защищающийся замок. Данная функция принимает в себя 1 параметр defenderObject, который является объектом защищающегося замка - defender. Она должна сверять количество всех боевых единиц у обоих замков. Например, если у атакующего замка значение свойства archer больше, чем у защищающегося, то к шансам захвата необходимо прибавить 1 (изначально значение шансов должно быть равно нулю). Функция должна возвращать массив, в котором 1-й элемент - это шансы атакующего замка на захват, а 2-й - это значение максимального шанса на захват (количество ключей в объекте defenderObject). То есть, если у атакующего замка значения свойств archer и cavalry больше, чем у защищающегося, а другие значения свойств - меньше, то шансы на захват должны быть [2, 4] (образно шансы равны 2 из 4 либо 50%)
2. **improveArmy** Прибавляет к каждому числовому значению объекта attacker по 5 единиц (обновляет свойства archer, footSoldier, cavalry, artillery). Данная функция не принимает никаких параметров
3. **attack** Принимает в себя 1 параметр - это объект защищающегося замка defender. Сначала эта функция проверяет, если шансы на захват (вызываем функцию **checkChancesToWin**) меньше, чем 70% от максимальных шансов, то необходимо усилиться (вызвать функцию **improveArmy**) и вывести сообщение через alert “Наши шансы равны ${**ourArmyChances**}/${**maximumChances**}. Необходимо укрепление!” (**ourArmyChances** - шансы атакующего замка на захват, **maximumChances** - максимальный шанс на захват). Иначе же требуется вывести сообщение в модальном окне “Мы усилились! Мы несомненно победим! Наши шансы высоки!”

Пример результата работы функции **attack:**

```
attacker.attack(defender); // Наши шансы равны 1/4. Необходимо
укрепление!
attacker.attack(defender); // Наши шансы равны 2/4. Необходимо
укрепление!
attacker.attack(defender); // Мы усилились! Мы несомненно победим!
Наши шансы высоки!
```