**Java Script**

Стиль коду:

1. Одинарні кавички
2. 2 пробіли відступ з нової строки
3. Пробіл після ключового слова
4. Використовувати строго рівно ===
5. Називати файли, змінні, функції максимально зрозуміло
6. Не дублювати код

**1. Перемінні(variables) і види данних(data types):**

Expression:

* let - змінна
* var - змінна вже не використовується
* const - незмінна

Types of data:

* underfined - не визначене значення
* NaN - not a number
* boolean - правда або брехня (true or false)
* string - строка (текстове значення)
* number - числове значення більше нід 53 ступінь
* BigInt - число більше за number
* symbol - символ
* null -пусте значення

Оператори: кожний оператор має свій індекс пріоритету **"precedenc"** і определяет порядок, в котором операторы выполняются. Операторы с более высоким приоритетом выполняются первыми.

1. Логічні:

* && (i) - якщо хоча б 1 з операнд = false то верне значення false
* || (or) - якщо хоча б 1 з операнд = true то верне значення true
* ! (not)

1. Арифметичні:

* % - остача від ділення
* -- забирає 1 від свого операнду // х=2; х--; х=0
* ++ додає до свого операнду
* + конвертує операнд в число // +"3" // 3
* \*\* підносить до ступеня // 2\*\*3 // 8

1. Порівняння:

* >=, =< // більше рівно або менше рівно
* == не строге рівно порівнює значення а не типи данних if ('42' ==42) -> true - хоча порівнюється число і строка
* === строго дорівнює - if ('42' ===42) = false
* !== не дорівнює

1. Строкові оператори:

* + "Ілля" + "Аня" // "Ілля Аня"

1. Оператор Type of:

* const text= 'Small'; typeOf text// String

1. Тернарний оператор:

* Структура: **умова ? правда : брехня**
* const age=16; let status=age>=18 ? 'adult' : 'minor'

7. Умовний оператор:

Якщо умова в дужках не задана то вона завжди буде true

Конструкція : **If (умова // condition true or false) { Правда роби цей блок коду } else{ цей }**

або використати тернарний оператор:

* const age=16;
* let status=age>=18 ? 'adult' : 'minor' // status=minor

**Функції** - будь яка функція це обєкт, котрий має в собі якусь логіку

console.dir(func) - інфо про функцію

**Function declaration:**

Через ключове слово **function** назва функції yearOfBirth параметри функції( ..., ....){логіка}

function sum ( a, b) { return a+b}

**Function expression:** функція присвоюється змінній

const fullName = function (firstName, SurName) { return 'firstName' + 'SurName'}

Різниця між **function declaration** i **function expression** в тому шо в випадку **function declaration** ми можемо викликати функцію перед написанням самої функції а **function expression** тільки ініціалізації.

**Анонімна функція:** функція без імені

**Стрілочна функція:**

const greet = (name)=> {console.log ('Привіт' , name)}

Якщо в функції всього лиш 1 аргумент і 1 строка логіки то можна без дужок:

const greet = name=> console.log('Привіт', name)

Якщо функція не приймає параментрів то треба писати фігурні скобки

const greet= ( ) => console.log ('Hello')

**Параметри за замовчуванням:** якшо не надати значення аргументу, то він має значення за замовчуванням

const sum=(a,b) =>a+b

console.log(sum(2,))=> undefined

const sum=(a,b=1) =>a+b

console.log(sum(2,))=> 3

**Функція котра приймає будь яку кількість параметрів:** за допомогою спред оператора '...'

function sumAll (...all) {

let result =0;

for(let num of all) {result+=num}

return result }

sumAll (1, 2, 3, 4, 5) => 15

**Замикання:** - з 1 функції повертаємо іншу функцію

function createMember (name) {

return function (lastName) {

console.log (name+ lastName) }

const logWithLastName = createMember ('Bob')

console.log (logWithLastName('SaaB') => Bob SaaB

console.log (logWithLastName ('Marley') => Bob Marley

Ідея в тому шо імя залишається постійно Bob а міняється тільки прізвище

**Цикли // Loops - Виконання коду n-кількість разів**

1. **For (let index початок; index<умова виходу; крок) { блок коду що робити}**

for (let i=0; i<=25; i+=1) {

console.log(i)} ----> (покаже числа від 1 до 25)

1. **Do {блок коду що робити} while (умова) - виконання коду хоча б 1 раз**

let i=0;

let result=" ";

do {i+=1; result+=i}

while(i<=5)

console.log(result)

1. **While(умова) { тіло цикла}**

**Дірективи:**

* **break --** перервати цикл
* **continue** - продовжити цикл
* **prompt** - виводить повідомлення для користувача з "ОК" або "Відміна"
* **Alert** - повідомлення для користувача

**Масиви // Arrays -** впорядкована колекція елементів (чисел, міст, людей). Кожний елемент масиву має свій індекс; Нумерація в масиві починається з 0; Плоский масив має 1 рівень вкладеності.

**Синтаксис:**

const cars= ['audi', 'volkswagen','renault', 'daewoo', 'hyundai']

**Створення масиву:**

* Літеральний спосіб ----> const arr = ['tom', 'kevin', 'clay'];
* Конструктор -----> const arr = newArray ('tom', 'kevin', 'clay')

**Доступ до елементів масиву:**

* **arr.length**  - показує довжину масиву;
* **arr.length-1** останній елемент масиву.
* **arr[arr.length]=you** - додасть елемент you в кінець масиву arr
* **arr[1]** - покаже елемент масиву з першим індексом
* **arr[5]=100** - присвоєння 5 елементу масиву нового значення

**Ітерація масиву:**

1. Цикл for стандартний як вище;
2. Цикл for (let el of array)

**Методи масиву:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Перебор всіх елементів** | **Додавання або усунення елементу** | **Перебір елементів** | **Пошук** |
| .forEach ( ) | .push ( )  .pop ( )  .unshift ( )  .shift ( )  .splice ( )  .slice ( )  .concat ( ) | .map ( )  .sort ( )  .reverse ( )  .join ( )  .reduce ( )  .flat ( )  .flatMap ( ) | .indexOf ( )  .lastIndexOf ( )  .includes ( )  .find ( )  .findIndex ( )  .filter ( )  .some ( )  .every ( ) |

|  |  |
| --- | --- |
| **Методи що змінюють масив:** | **.push ( )** - додає новий ел в кінець масиву  **.pop ( )** - видаляє останній ел масиву  **.unshift ( )** - додає ел на початок масиву  **.shift ( )** - видаляє ел з початку масива  **.splice (start, deleteCount, new )** - видаляє ел по індексу від і до і додає нові ел  **.reverse ( ) -** розвертає масив  **.sort ( ) -** сортує в залежності від функції |
| **Методи що не змінюють масив:** | **.forEach ( ) -** проходить по всіх ел масиву і повертає змінений масив  **.slice (start, end )** - робить копію цілого або масиву куска масиву  **.concat (arr) -** склеює 2 масиви  **.reduce (callback, accumulator)** - застосовує функцію callback, для всіх елементів зліва направо  **.join ( ) -** обєднує всі ел масиву в строку  **.flat (рівень) -** піднімає всі елементи вкладених підмасивів в залежності від рівня  **.flatMap ( )**  **.indexOf ( )** - шукає індекс по ел  **.lastIndexOf ( ) -** повертає останній індекс ел в масиві  **.includes ( ) -**  шукає ел в масиві повертає true or false  **.find (callback ) -** повертає перше значення яке задовільняє умови передані в callback  **.findIndex (callback )** - повертає індекс ел який задовільняє умови передані в callback  **.filter (callback ) -** створює новий масив з ел які задовільняють умови callback  **.some (callback ) -** перевіряє наявність ел в масиві який задовільняє умови  **.every (callback) -** перевіряє чи всі ел масиву задовільняють умови передані в callback  **.map (callback ) -** створює новий масив з результатом виклику callback для кожного ел |

Застосування методу **reduce** на масиві обєктів

const people = [

  { name: "Illia", budget: 5500 },

  { name: "Sasha", budget: 3500 },

  { name: "Ania", budget: 6300 },

];

const allBudget = people

  .filter((person) => person.budget > 4000)

  .reduce((acc, person) => {

    acc += person.budget;

    return acc;

  }, 0);

console.log(allBudget);

**Деструктеризація масиву:**

const [a,b]=arr

console.log (a,b) -----> 1,2

**Spread оператор:**

const [a,b,....num]=arr

console.log(a,b,num) -----> 1,2,3,4

Застосування методів **split, reverse**

const text = "Привіт як справи";

const reverseText = text.split("").reverse().join("");

console.log(reverseText);

**Обєкти**

**Обєкт** - це структура данних яка містить в собі інформацію в форматі імя і його значення (keys:value). Функція котра міститься в властивостях обєкту називається методом.

**Створення обєкту:**

const Kevin = {

age: '25',

heigth: '168cm',

weight: '82kg',

status: 'married',

};

**Додавання властивості в обєкт:**

Kevin.hobbies='football'

Kevin['hobbies']='football'

**Доставання властивості з обєкту:**

Kevin.age------>25

Kevin['age']-----> 25

**Ітерація обєкту:**

1. Цикл for (let el in object)
2. Object.keys (Kevin)-----> [age,heigth,weight,status]--> повертає масив ключів
3. Object.values (Kevin)-----> [25,168cm,82kg,married]---> повертає масив значень
4. Object.entries (Kevin)----->[['age', 25], ['heigth', 168cm], ['weigth',82kg], ['status',married]] ----> повертає масив масивів

**Як створити копію обєкту:**

1. Object.assign(Kevin,{})---->assign склеює 2 обєкти в 1
2. {...Kevin}-----> спред оператор створює обєкт зі властивостями Kevin
3. const boys = {Tom:'21',} let {Tom}=boys-----> деструктеризація

**Числа**

1.7e5=(1,7\*105)=170000

1,7e-5=(1,7\*10-5)=0,000017

Float - дробні числа

Int/Integer - ціле число

90082473457 - число

90082473457n - bigInt не працює з десятичними числами, вирішити цю проблему можна якшо число bigInt перевести в integer або десяткове в bigInt

* parseInt(10n)-4=6
* 10n-BigInt(4)=6n

**Приведення строки в число:**

* за допомогою унарного плюса +'17'=17
* за допомогою конструктора Number '17'=17

**Методи для роботи з числами:**

* .**isNan( )** - визначає чи передане число являється Nan
* **.isFinite ( )** - оприділяє чи число скінченне
* **.parseInt (12px )---->** 12 - приймає строку в якості аргумента і повертає ціле число
* **.parseFloat ( )** - приймає строку в якості аргументу і повертає десятичне число з плаваючою точкою
* **.isInteger ( )** - оприділяє чи передане значення є числом, а саме число цілим
* **12,5469.toFixed(2)=12,54** - заокруглює число до кількості знаків після коми

**Обєкт Math:**

* **Math.round** (17,4)--> 17 заокруглює число в сторну ближчого цілого
* **Math.ceil** (17,1)--> 18 заокругляє до вищого значення
* **Math.floor** (17,9) --> 17 заокругляє до меншого значення
* **Math.trunk** (17,5)--> 17 відкидає дробну частину
* **Math.abs** (-17 )---> 17 абсолютне значення числа
* **Math.min** (2,3,1,6 )--> 1 мінімальне з перелічених значень
* **Math.max** (2,4,1,8 )--> 8 максимальне з перелічених значень
* **Math.pow** (3,4 )-->81 піднімає основу 3 в 4 ступінь
* **Math.sqrt** (9)---> 3 додатній корінь числа

**Строки**

Строка - це масив символів.

**Template String** - це динамічна строка в яку можна додавати аргументи.

**Створення динамічної строки:**

* ` ` - косі кавички
* ${ result} - передача аргументів
* **Приклад**: console.log ( ` Hello ${user}, have a nice ${currentDay}` )

/n - перенесення тексту на нову строку

**Методи для роботи з строками:**

* **.toUpperCase ( )** - піднімає всі символи до верхнього регістру
* **.toLowerCase ( )** - понижує всі символи до нижнього регістру
* 'hello'**.concat** ('hi' )='hello hi' - обєднує строки
* .**split (' ')** - ділить строку по вказаному символу
* **.repeat (n )** - повторює строку n разів
* **. charAt (2)** - повертає символ з строки по вказаному індексові
* **.includes ( )** - перевіряє чи строка містить підстроку і повертає буліан
* **.indexOf ( )** - повертає індекс першого шуканого символу
* **.lastIndexOf ( )** - повертає індекс останнього шуканого сумволу
* **.endsWith ( )** - перевіряє чи строка закінчується на вказаний аргумент, true or false
* **.startsWith ( )** - перевіряє чи строка починається на вказаний аргумент
* **.search ( )** - шукає в строці вказаний символ і повертає індекс
* **.slice (2,-5 )** - вирізає з тексту символи по індексу від 2 до 5 включно
* **.substring (5,2 ) -**вирізає символи з тексту по індексу (stard,end)
* **.substr (0,4 )** - вирізає символи від 0 до 4
* **.replace ( )** - шукає символ або вираз в строці і замінює його
* **.replaceAll ( )** - змінює всі знайдені значення
* **.padEnd ( ) -** додає в кінці символи до вибраної довжини
* **.padStart ( )** - додає відступи на початку поки строка не досягне бажаної довжини
* **.trim ( )** - обрізає пробіли
* **.trimEnd ( )** - обрізає пробіли в кінці
* **.trimStart (** ) - обрізає пробіли на початку
* **.charCodeAt ('H' )=72** - повертає числове значення юнікоду
* **String.fromCharCode** (72 )='H' - перетворює числове значення юнікоду на символ

**Оператор SWITCH**

**Switch** - порівнює вирази з випадками описаними всередині нього а потім виконує певні дії.

**Синтаксис:**

**switch (expression) {**

**case value 1 :**

(тут виконується код якшо результат виразу дорівнює value1)

**break;**

**case value 2 :**

(інструкції до value 2)

**break;**

**ПРИКЛАД функції з використанням SwitchCase:**

const calc = (expression) => {

  const arr = expression.split(" ");

  let result;

  switch (arr[1]) {

    case "+":

      result = Number(arr[0] + arr[2]);

      break;

    case "-":

      result = arr[0] - arr[2];

      break;

  }

  return expression + "=" + result;

};

**Unit tests**

Структура:

it("Повідомлення", () => {

  результат=функція(пробні значеня);

expect(результат).toEqual(результат відпрацьованої функції з пробними знач.)

});

Краще коли файл JS з функцією і тест JEST створені в різних файлах.

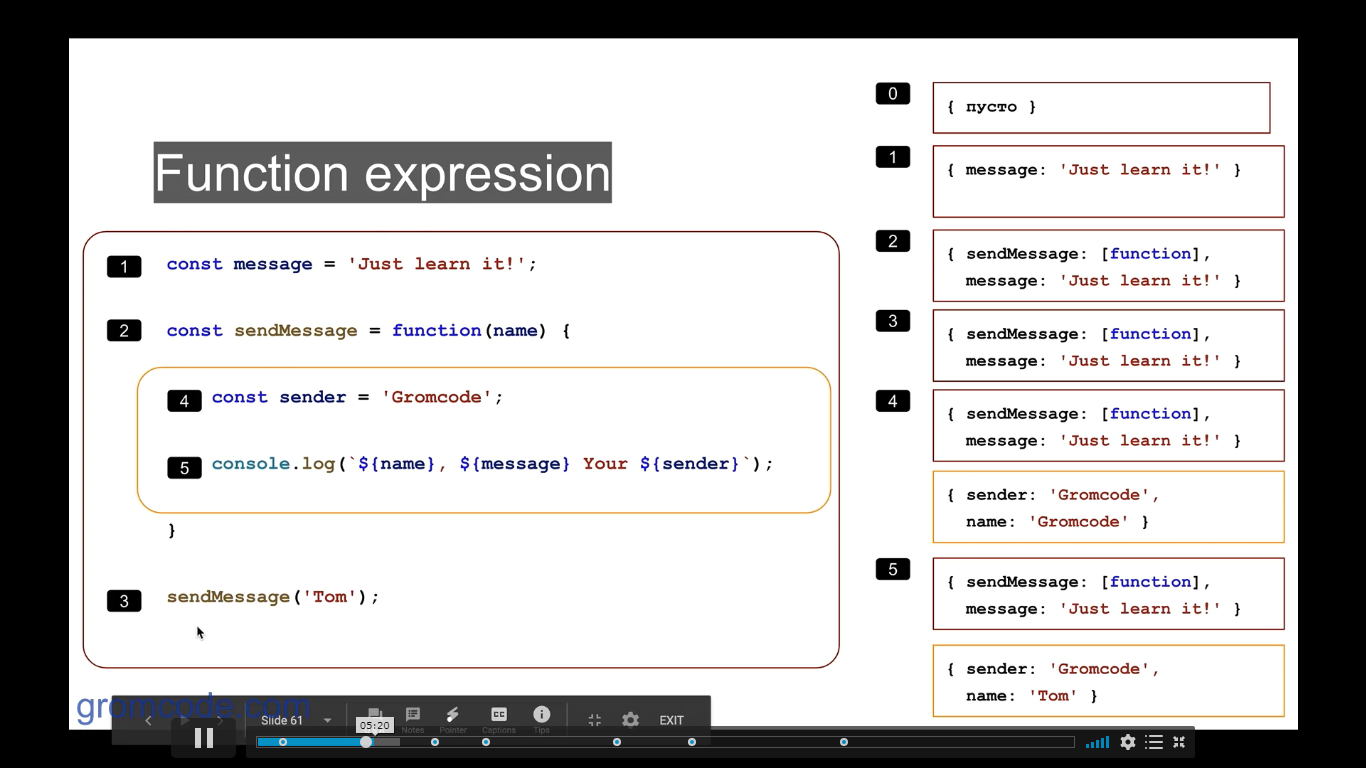
Щоб надіслати функцію з одного файлу в інший використовується EXPORT i IMPORT

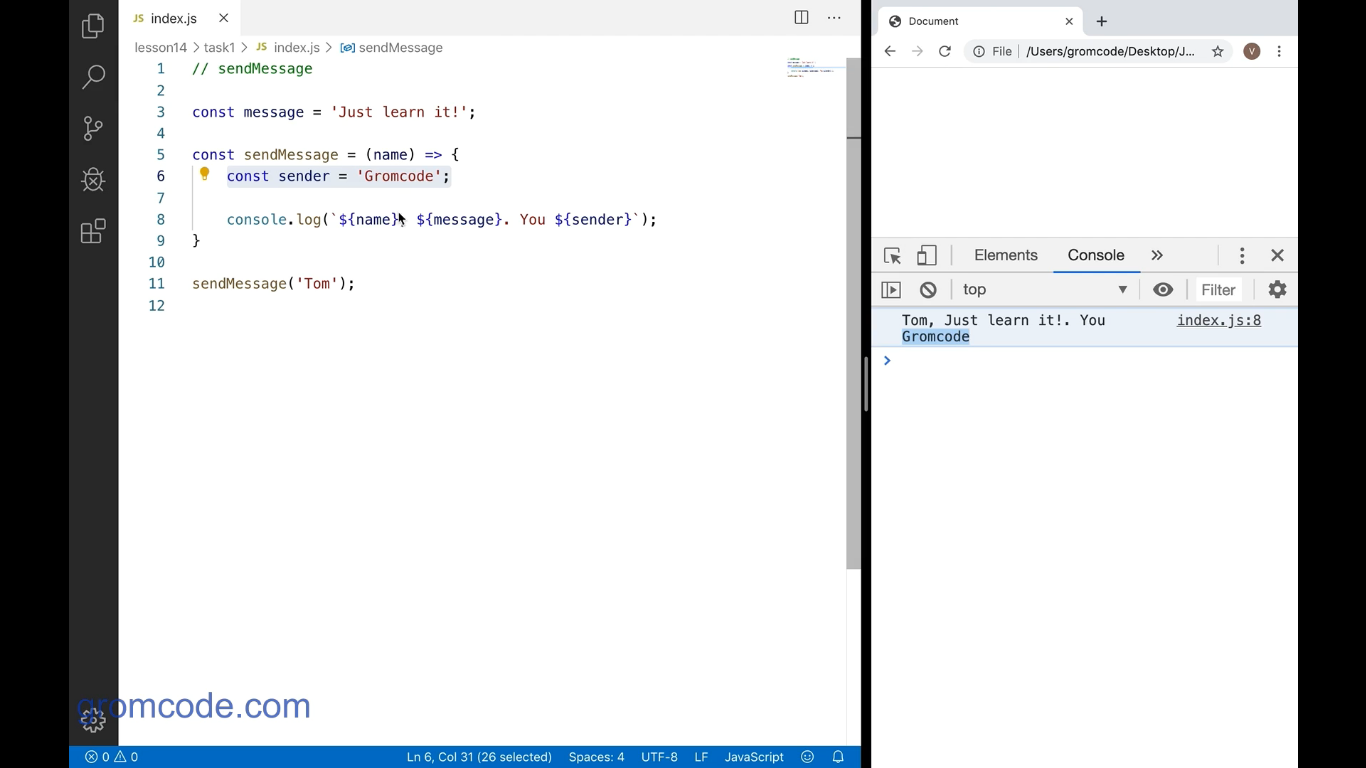
|  |  |  |
| --- | --- | --- |
|  | Export | Import |
| Іменний | // calculator.js  export const getSum=(a,b)=>a+b | // main.js  import {getSum} from './calculator.js  getSum (4,7) // 11 |
| Дефолтний | //calculator.js  export default (a,b) => a+b; | //main.js  import getSum from './calculator.js';  getSum (4,7); |

Щоб експорт і імпорт працювали потрібно в терміналі прописати **npm instal -D @babel/core @babel/preset-env** і в папці де знаходиться **package.json** створити файл **.babelrc** і в ньому прописати { "presets": ["@babel/preset-env"]}

Щоб запустити перевірку в терміналі вводимо jest і шлях до папки в котрій лежить файл test ./java\_scriptbeginner/full

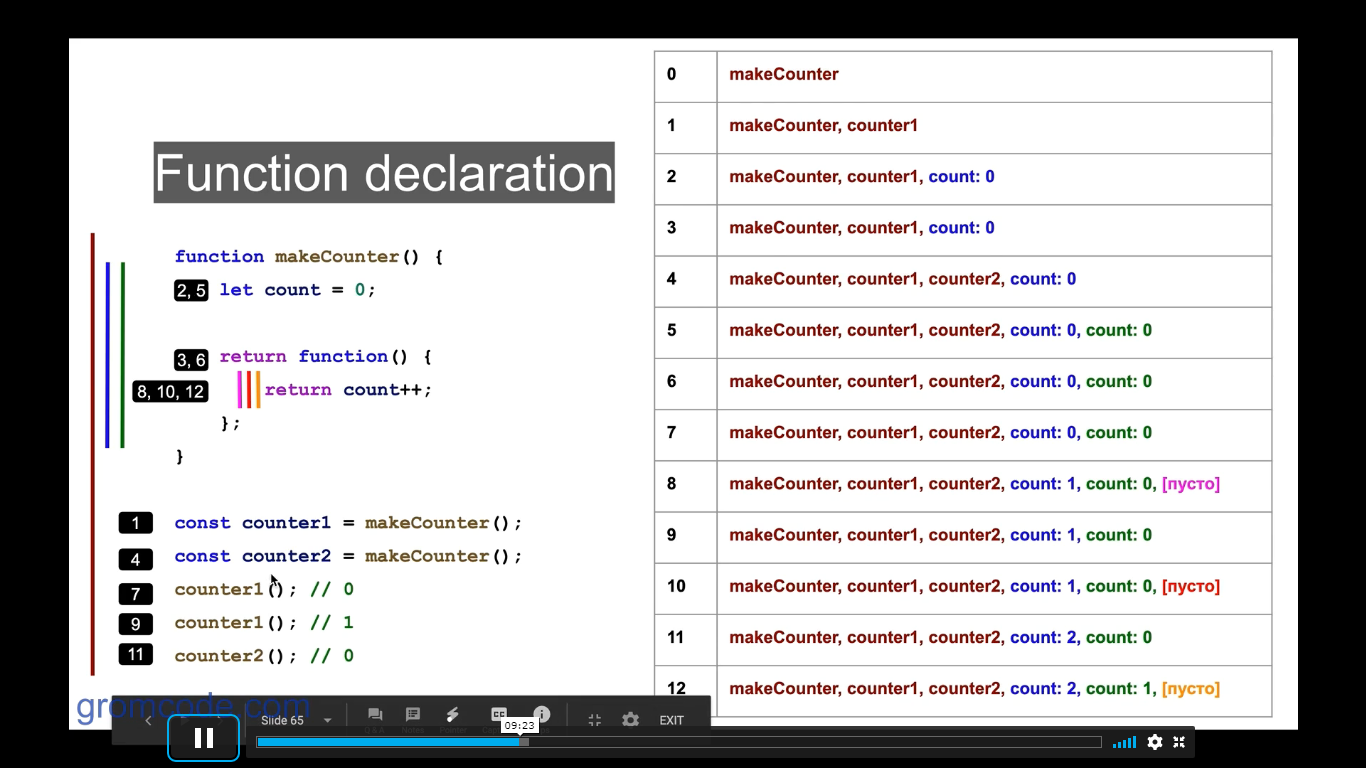
**Lexical environment**





Лексичне оточення створюється фігурними скобками, коли всередині функції не обявлені перемінні то функція шукає значення поза функцією в глобальному лексичному оточенні, але внутрішні перемінні недоступні для глобального оточення тільки локально

**Closure (замикання)**

****

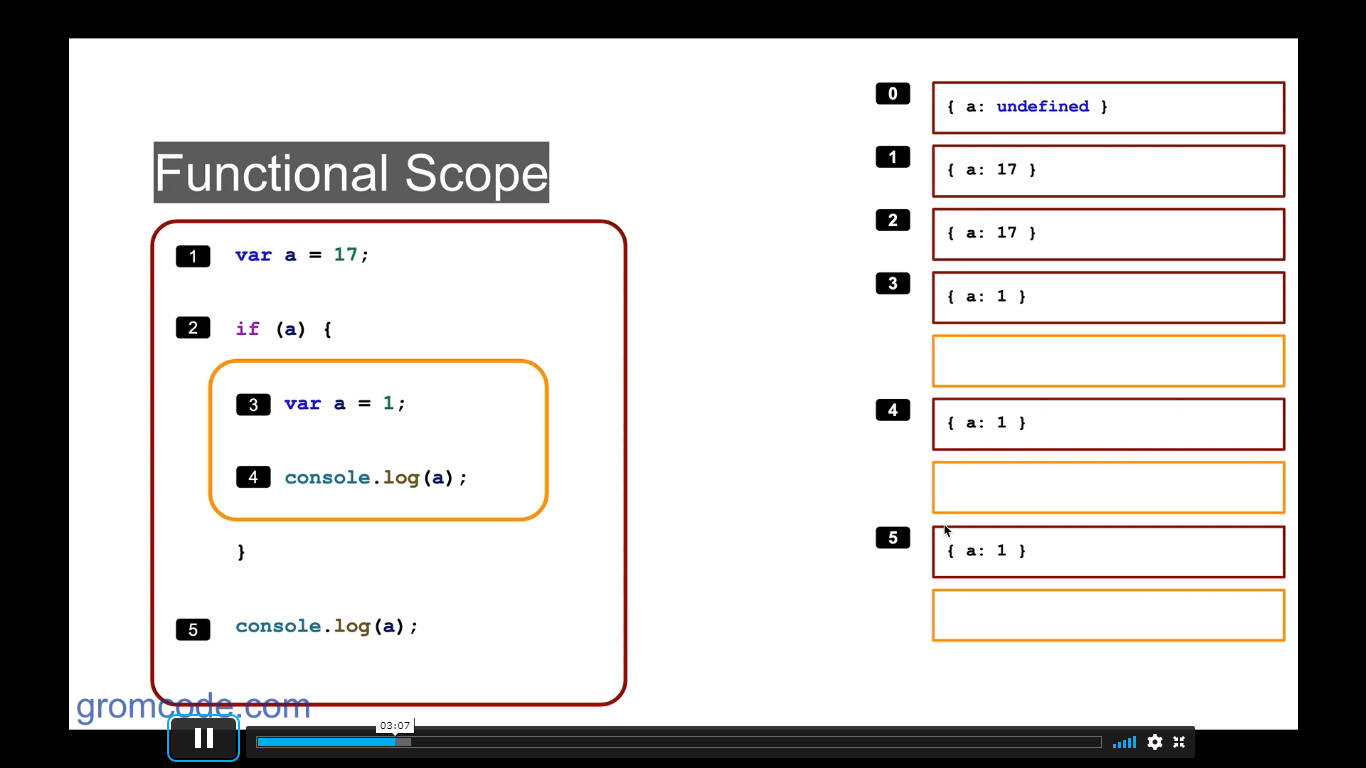
Замикання - це функція і всі зовнішні змінні, котрі їй доступні

**Hoisting (спливання)**

var - вспливає, це часто приводить до непередбачуваного результату.

var - не має блочної області видимості, це незручно і може призвести до непередбачуваного результату.

Якщо за допомогою var обявити змінну всередині функції то її буде видно ззовні



**Вспливання** - дозволяє користуватись змінною до того як вона обявлена в коді.

**НЕ працює** для const, let, class - і це добре

**Контекст виконання (context this)**

В обєкті зазвичай зберігається інформація про шось, наприклад людину. Внутрішні дані обєкту називаються властивостями, якшо їхнім значеням присвоїти функцію тоді це називатиметься методом обєкту. Якшо в методі обєкту використовуються дані з цього самого обєту то ссилка на ці дані передається через ключове слово **THIS.**

НАприклад:

const person = {

  name: "John",

  weight: "83kg",

};

person.greeting = function () {

  console.log(`Hi i am ${this.name}`);

};

person.greeting();

Або короткий запис:

const person = {

  name: "John",

  weight: "83kg",

  greeting() {

    console.log(`Hi i am ${this.name}`);

  },

};

person.greeting();

При присвоєнні перемінній метода обєкту втрачається контекст:

const sayHi=person.greeting()

console.log(sayHi)//undefined --> втрата контексту

Але контекст можна закріпити за допомогою метода **.bind (назва обэкту)**

person.greeting(); //--> 'Hi i am John'

const sayHi = person.greeting().bind(person) або .bind({name:'John,})

console.log(sayHi); //--> 'Hi i am John'

Виклик функції методом **.call({ })---->** sayHi.call ({name:'Tom'})----> Hi i am Tom

const person = {

  name: "John",

  weight: "83kg",

  message: "Hi",

  greeting(age, message) {

    console.log(`${message} ${this.name} ${age}`);

  },

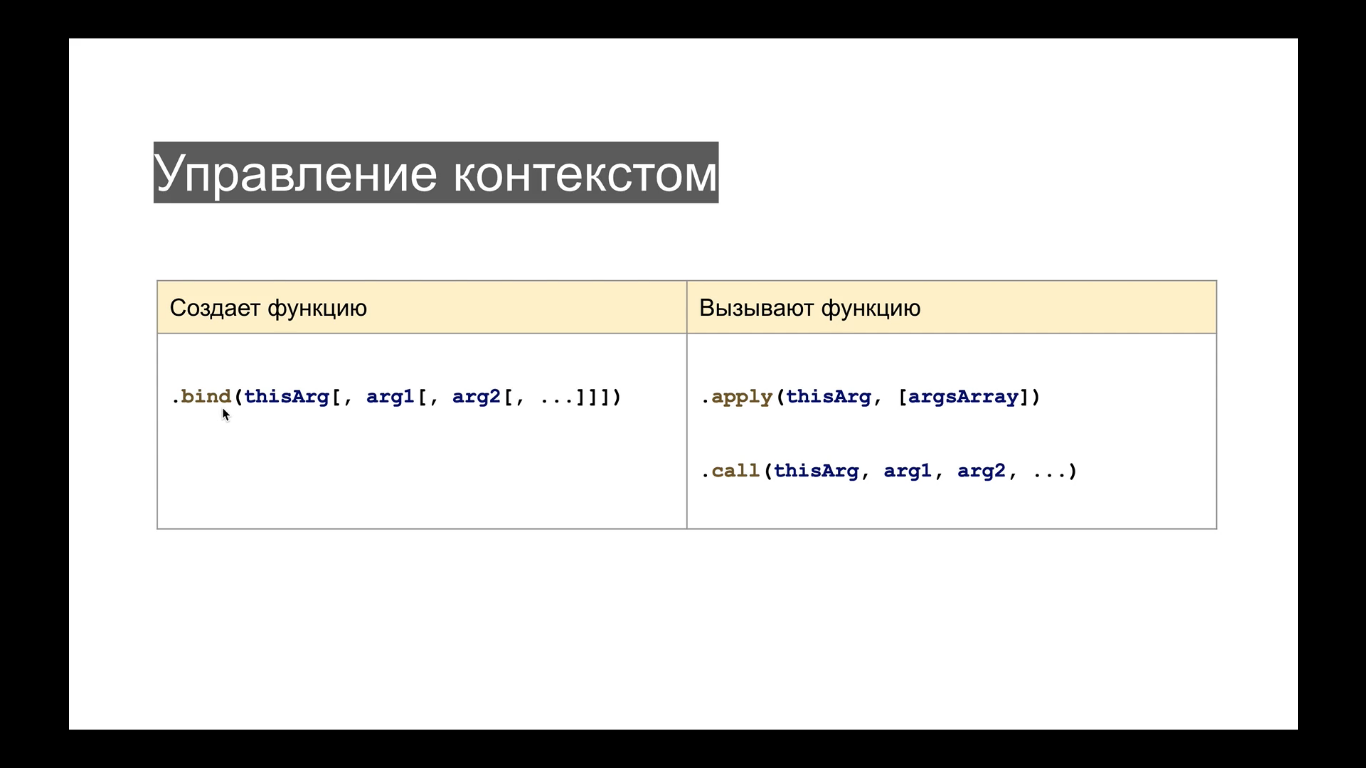
};

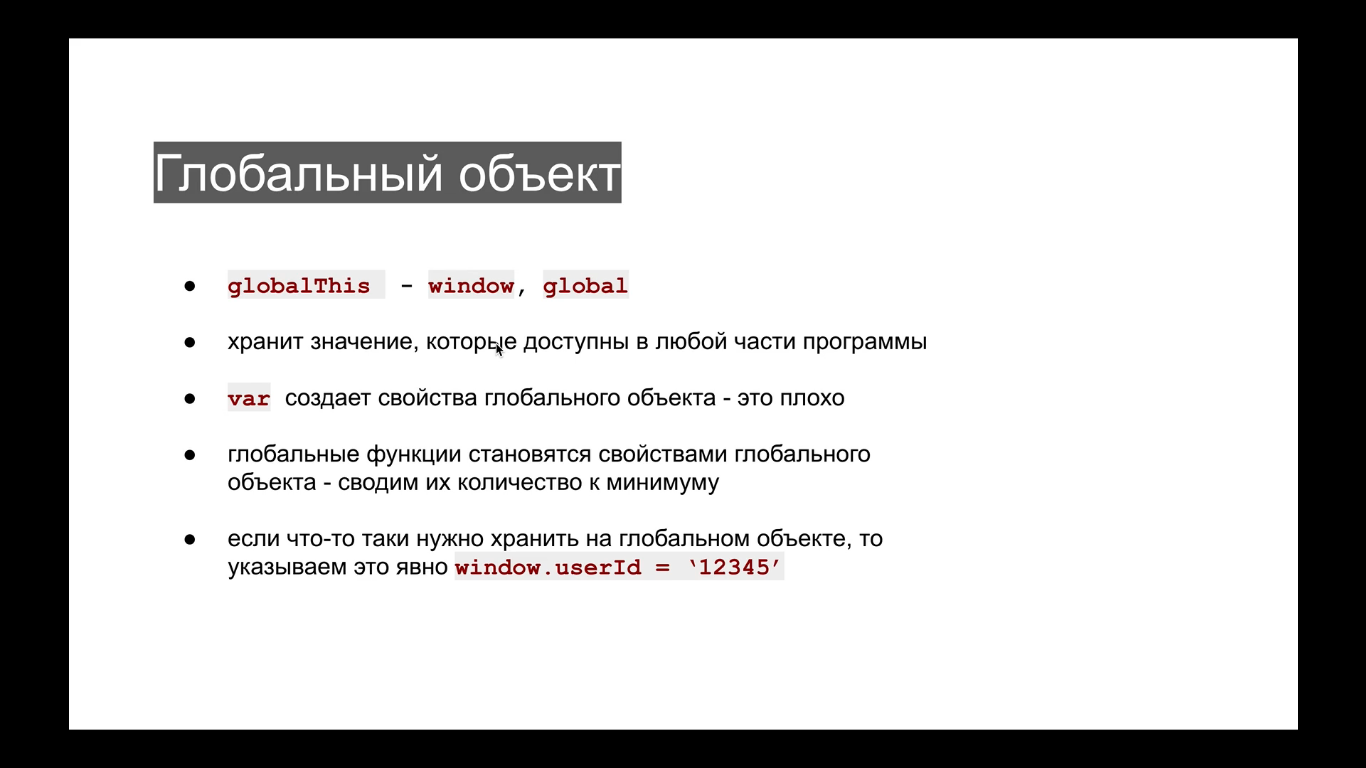
const sayHi = person.greeting(17, "Hi");

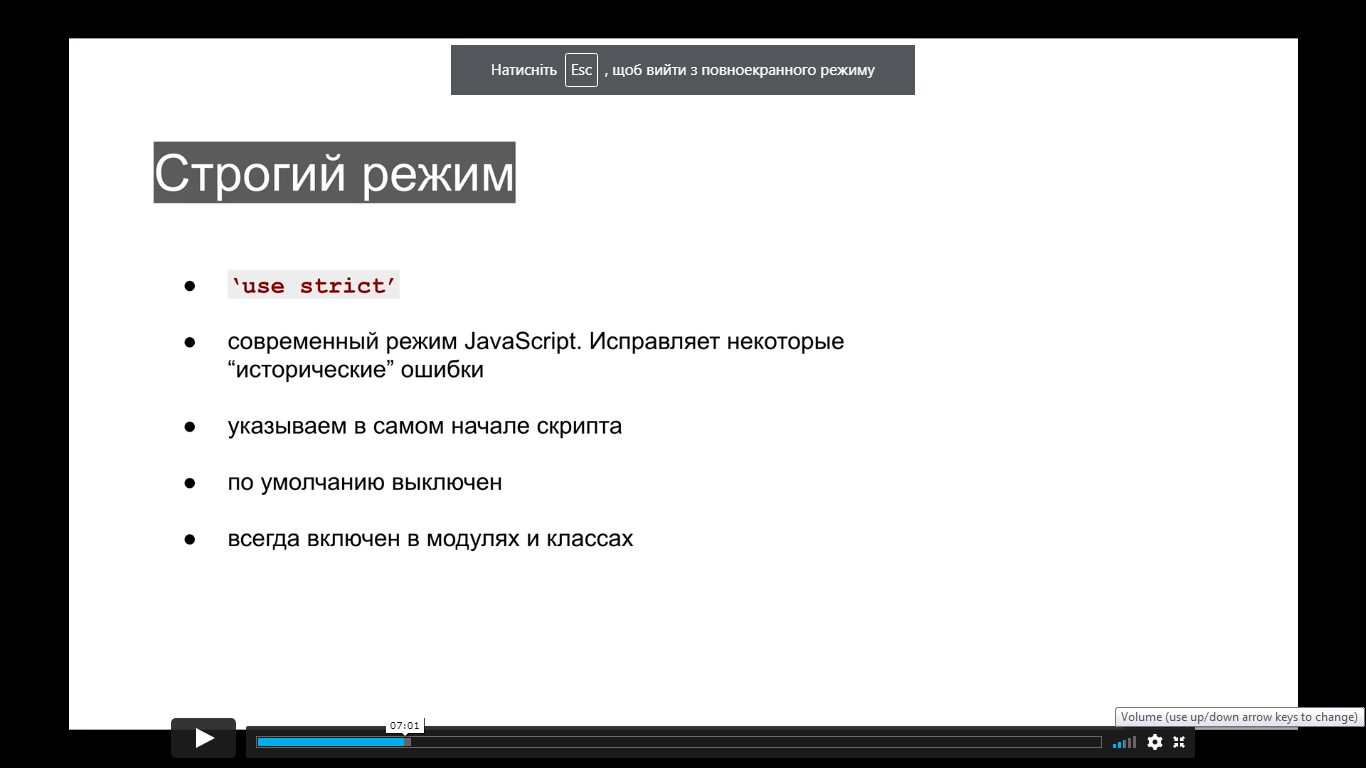
sayHi.apply({name:'Tom',}[17,hi])---> змінює контекст

**Глобальні методи JS:**

* prompt( )---> зворотній звязок з користувачем
* setTimeout( ( )=> { }, time);----> запускає функцію через вказаний час

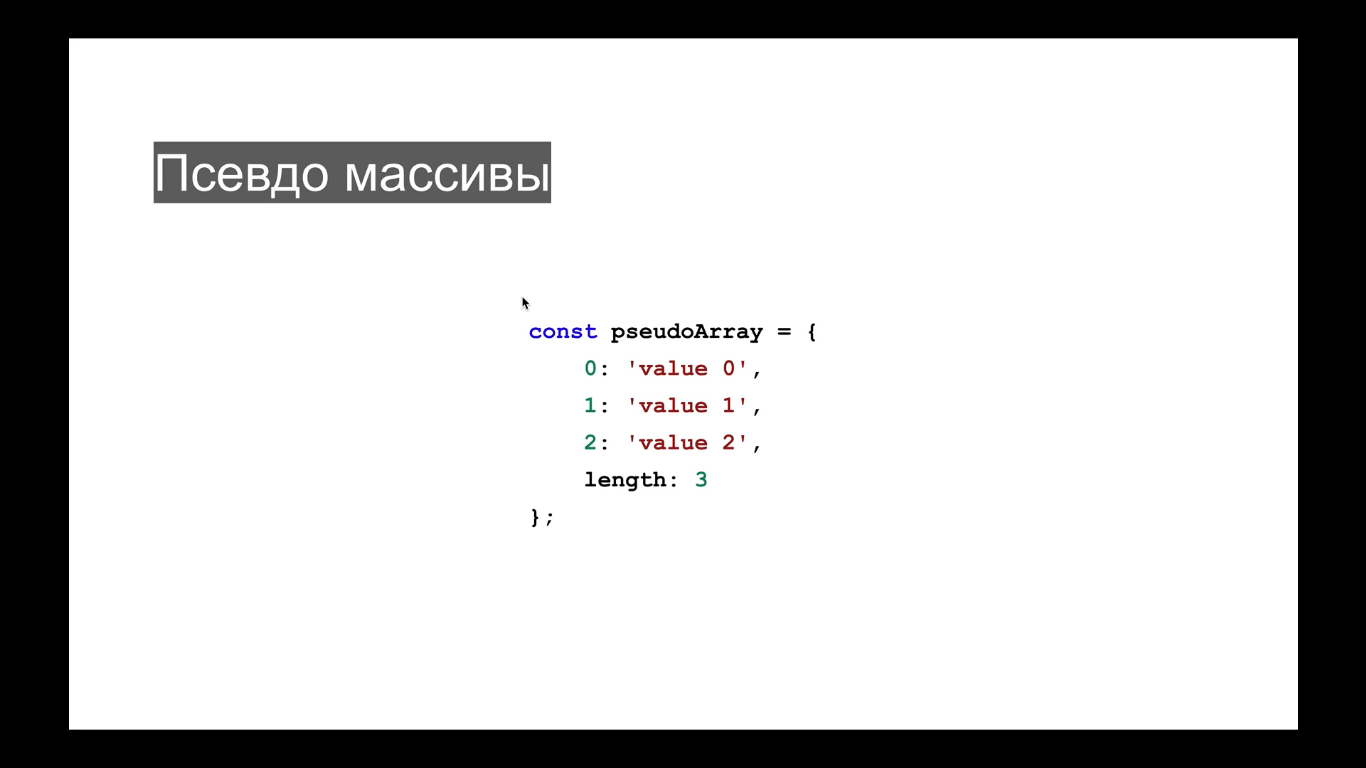




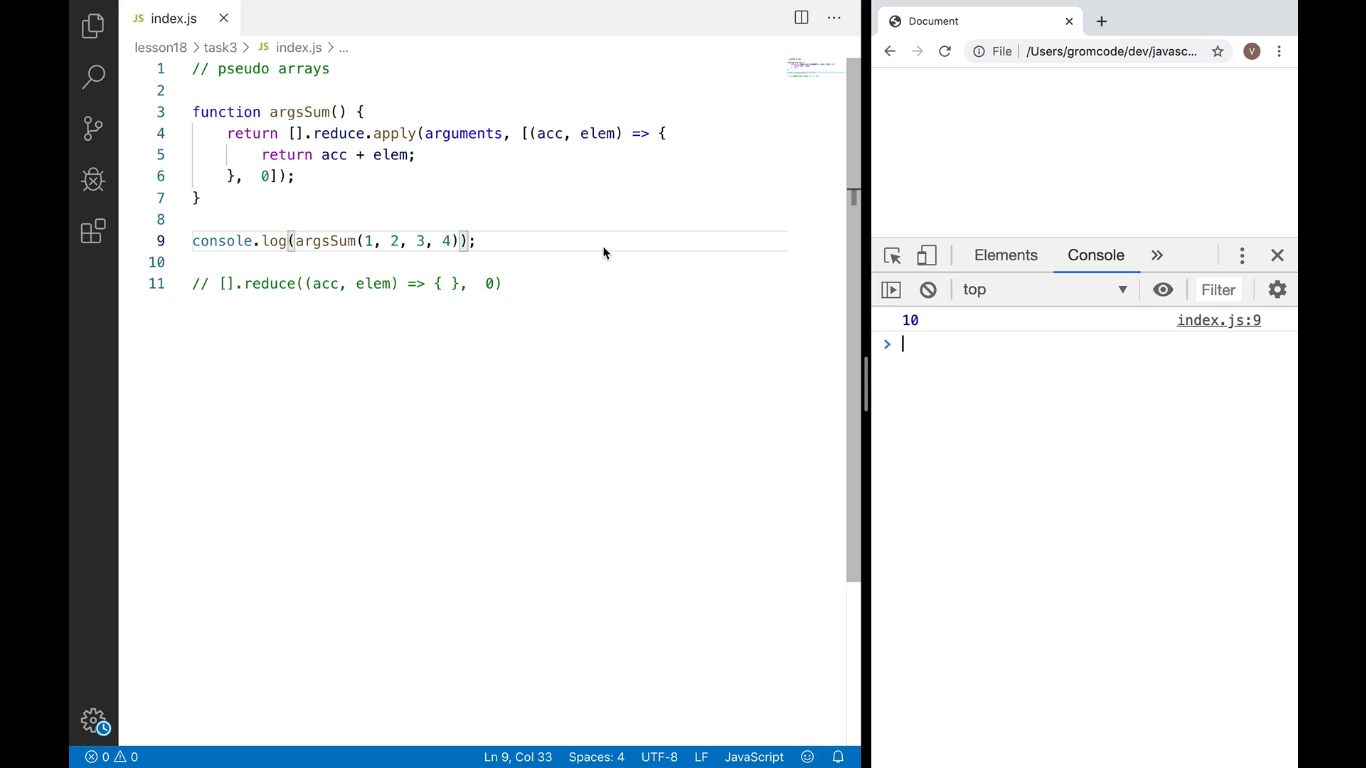


В методах масивів крім колбек функції може бути переданий обєкт 2 аргументом на який буде ссилатись THIS.

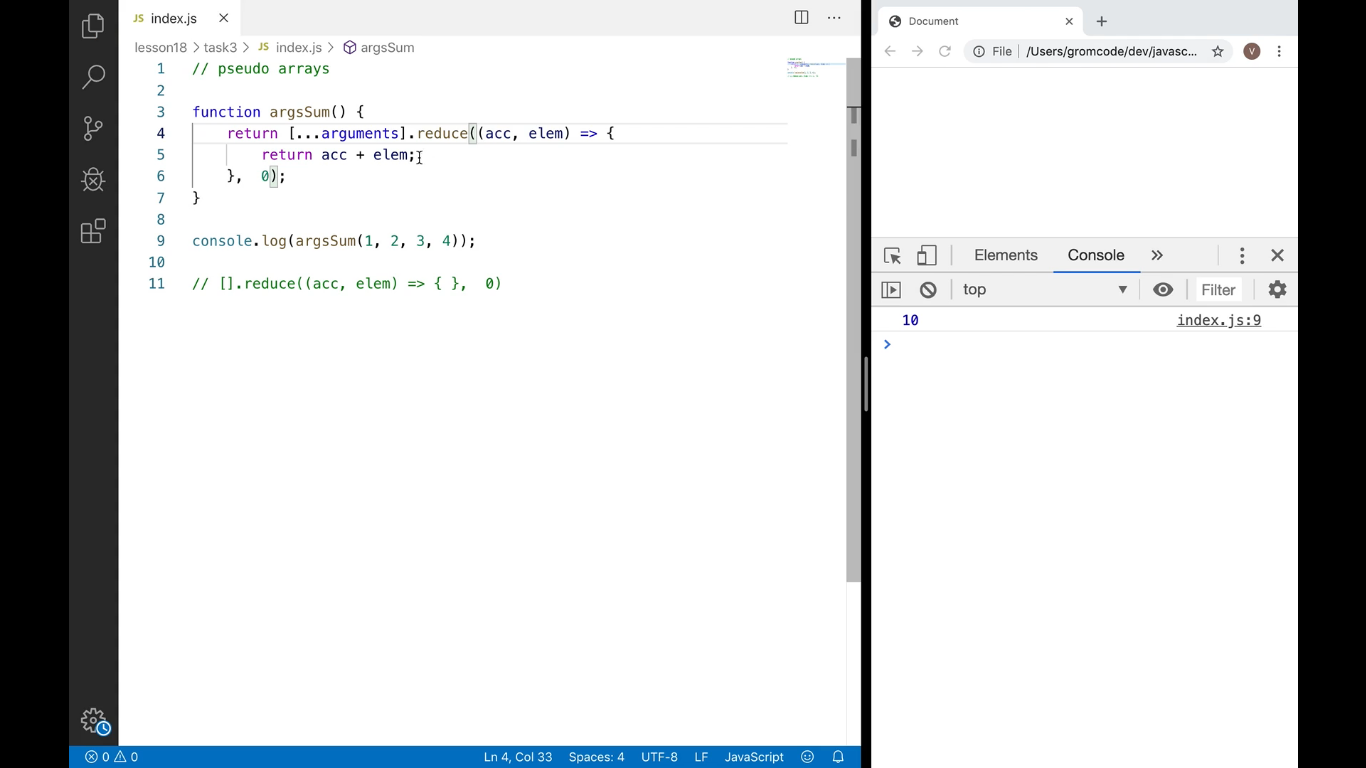
В масивах нема методу Math але ми можемо його застосувати до масиву через .apply наприклад: Math.max.apply(null, [1,2,3,5,6]) null тому що ми не передаємо контекст, замість масиву ми можемо передати перемінну в якій зберігається масив, або за допомогою спред оператора Math.max(.....array) котрий розбере масив на аргументи.

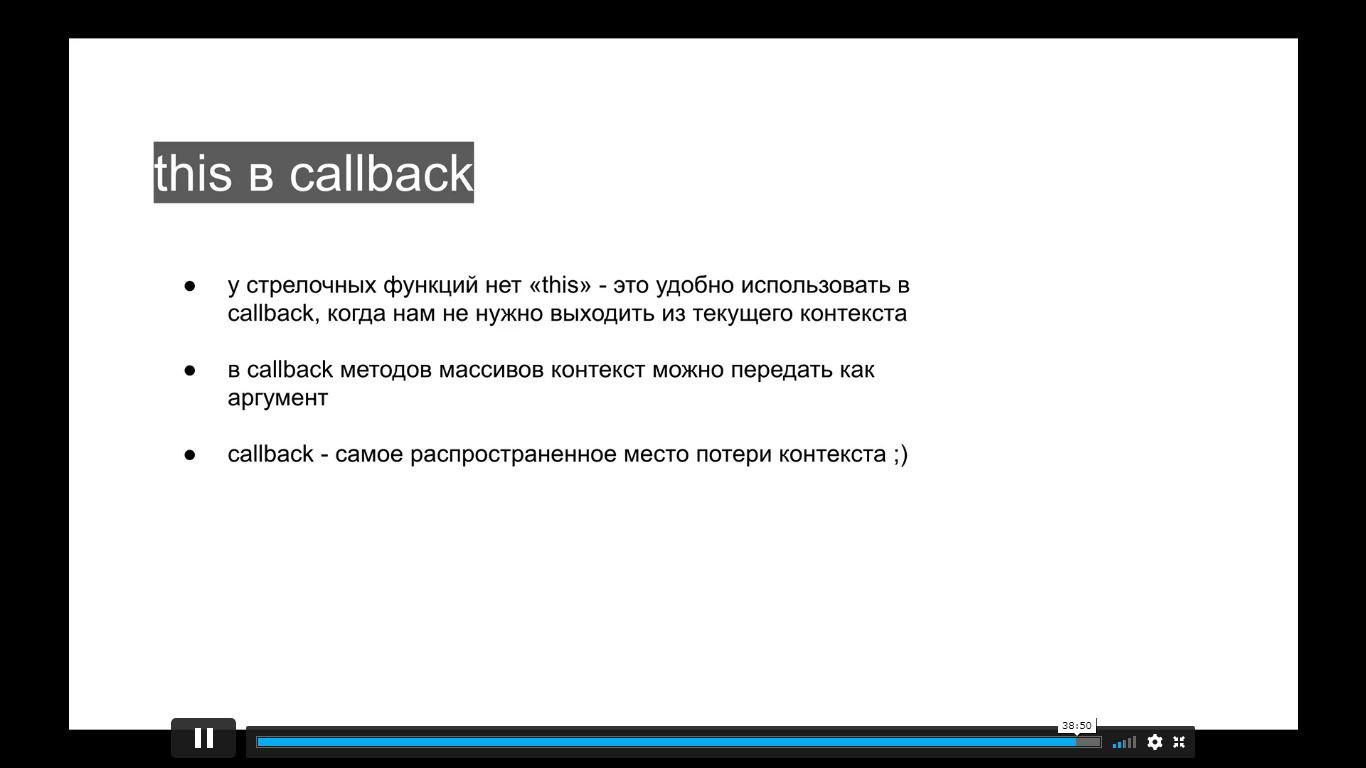


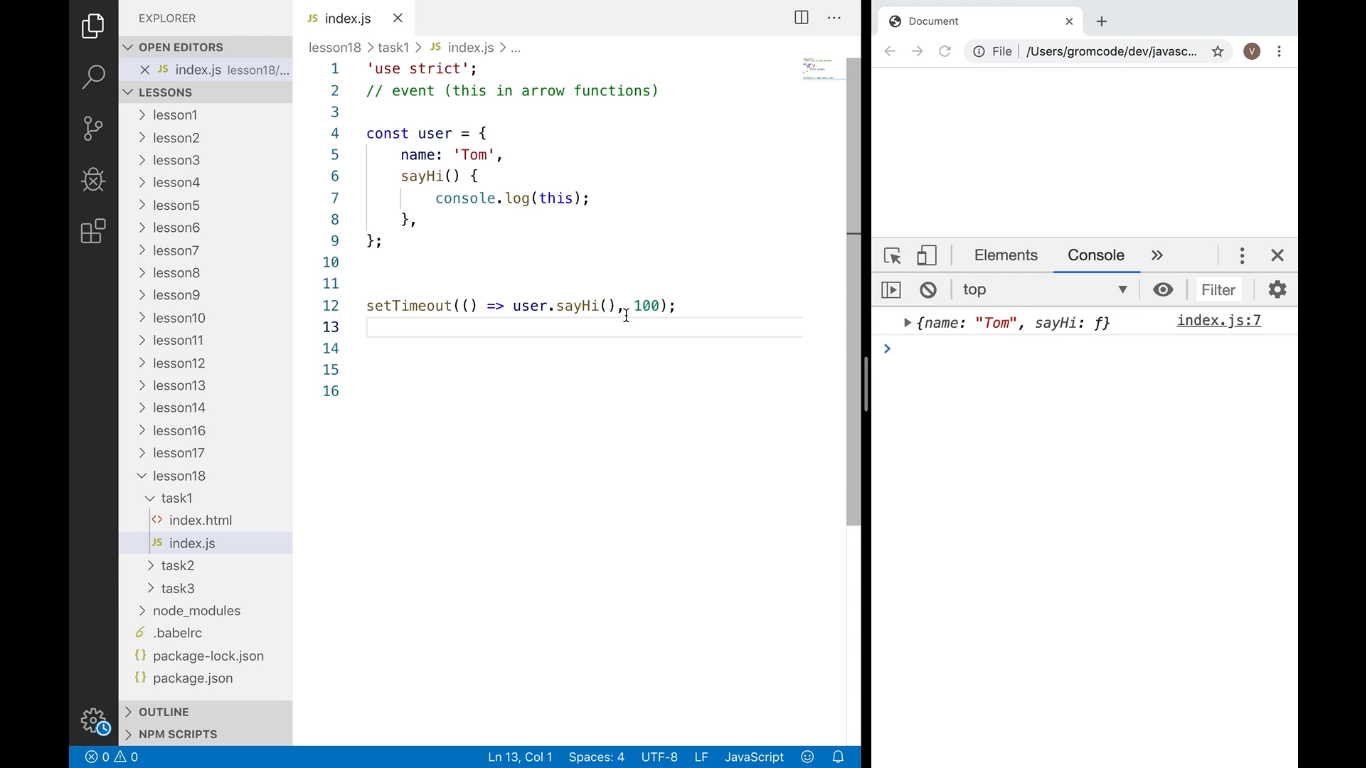
Псевдо масив - це обєкт, в якого свойства це порядковий номер від 0 і далі, та є метод length.



В данному випадку ми в функцію передаємо звичайні числа як аргументи, але хочемо до них використати метод reduce котрий є тільки в масивів, тому ми повертаємо масив з методом reduce і викликаємо його за допомогою apply.

 Або за допомогою спред оператора створити копію аргументів в вигляді масиву і користуватись як звичайним масивом





**Урок 19 Наслідування**

В обєкті нижче методів прописати **\_\_proto\_\_:User** ---- і вказати обєкт методи якого треба наслідувати.

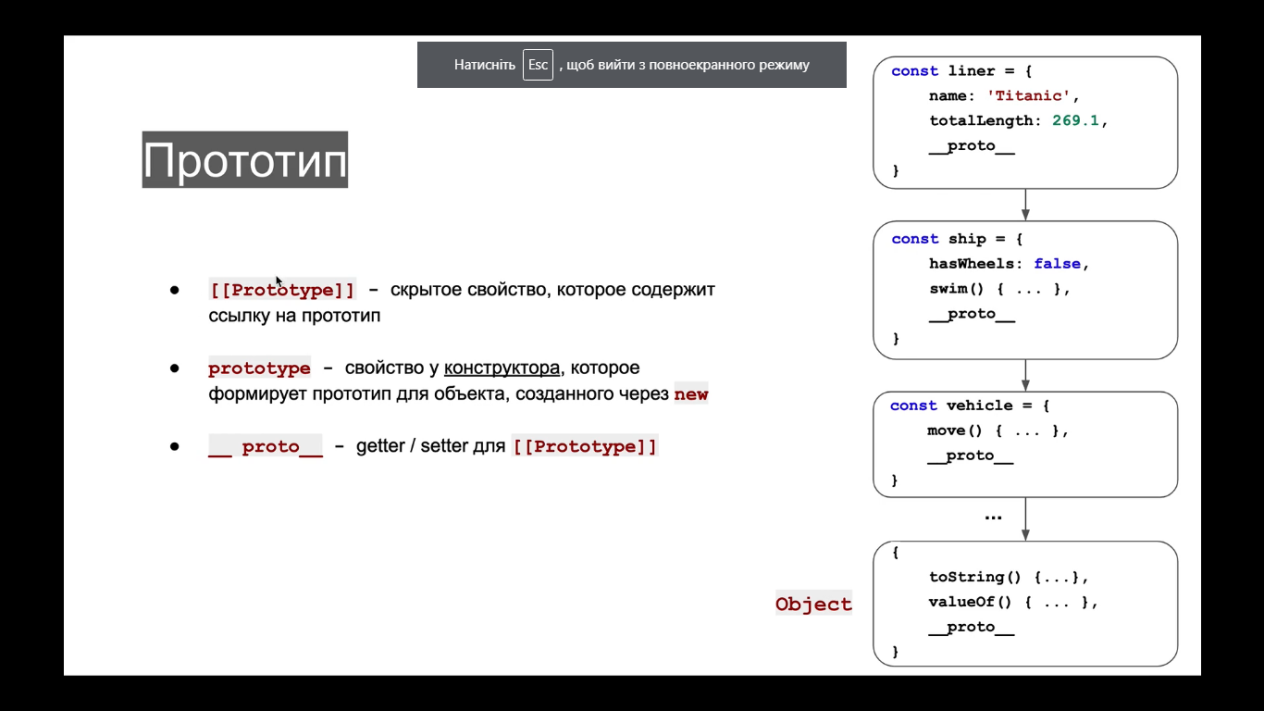
**Object.setPrototypeof(ship,vehicle)** --- методи ship для vehicle

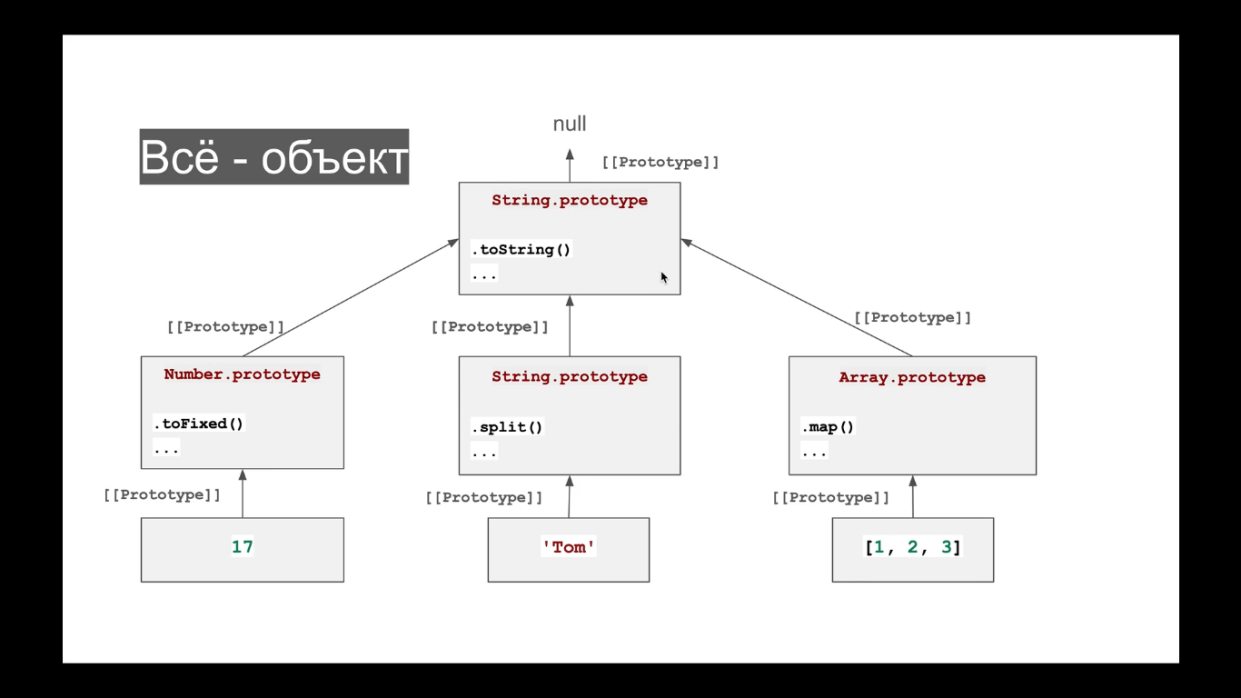
Розміщення в прототипі потрібної функції щоб мати завжди до неї доступ:

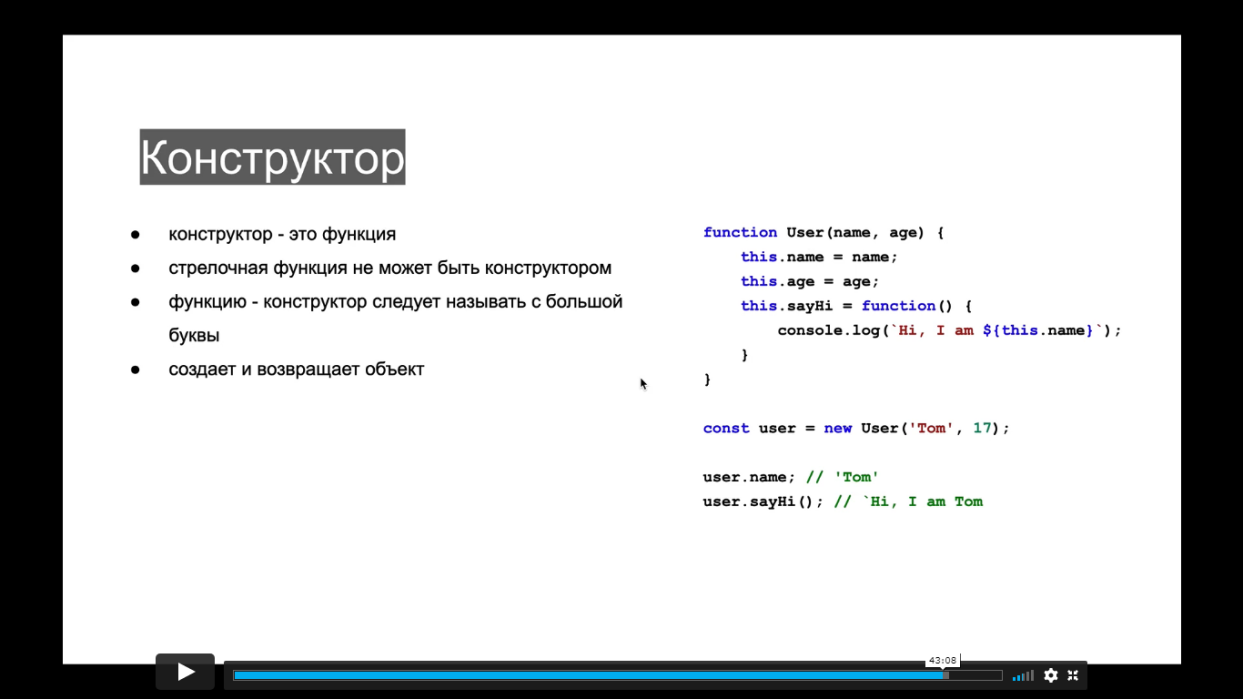
**Object.prototype.sayHi=function(){}**

Метод **getter** дозволяє тільки читати дані

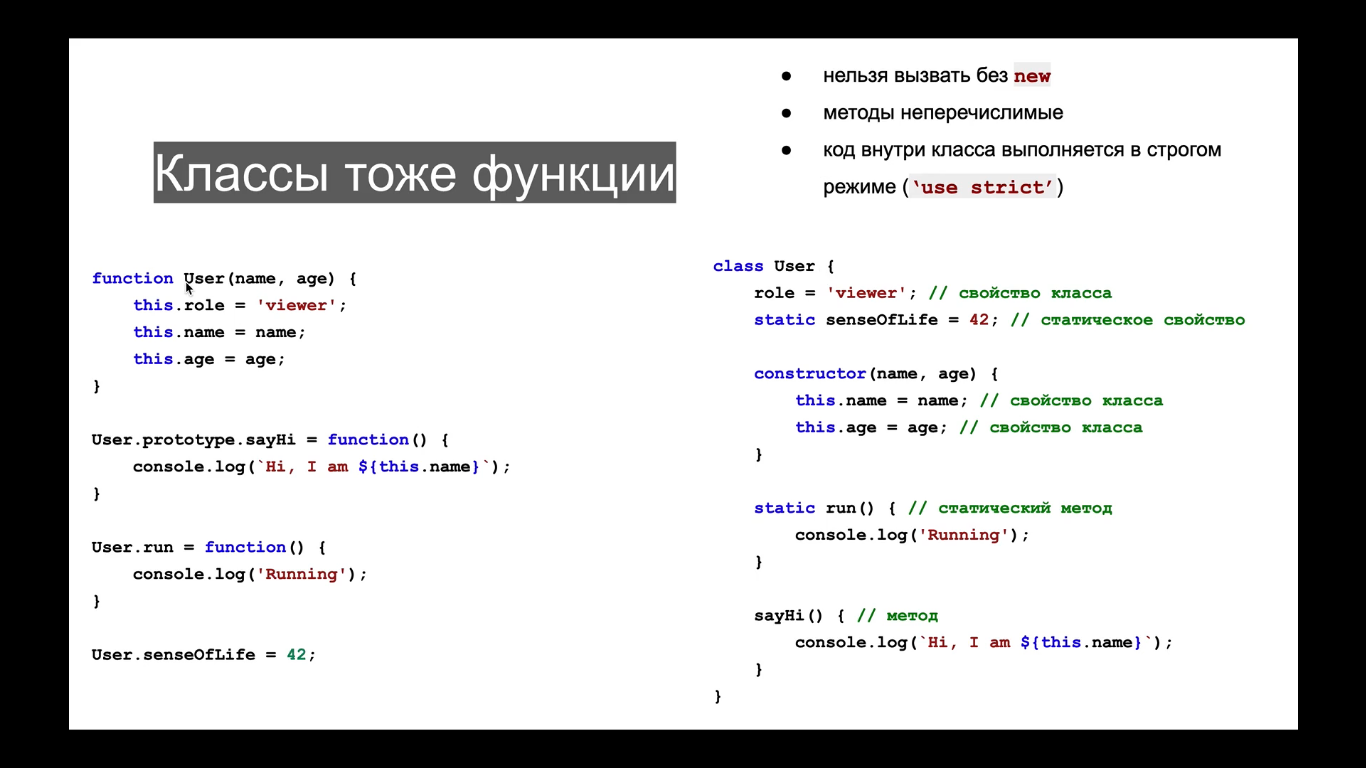
Метод **setter** встановлювати дані



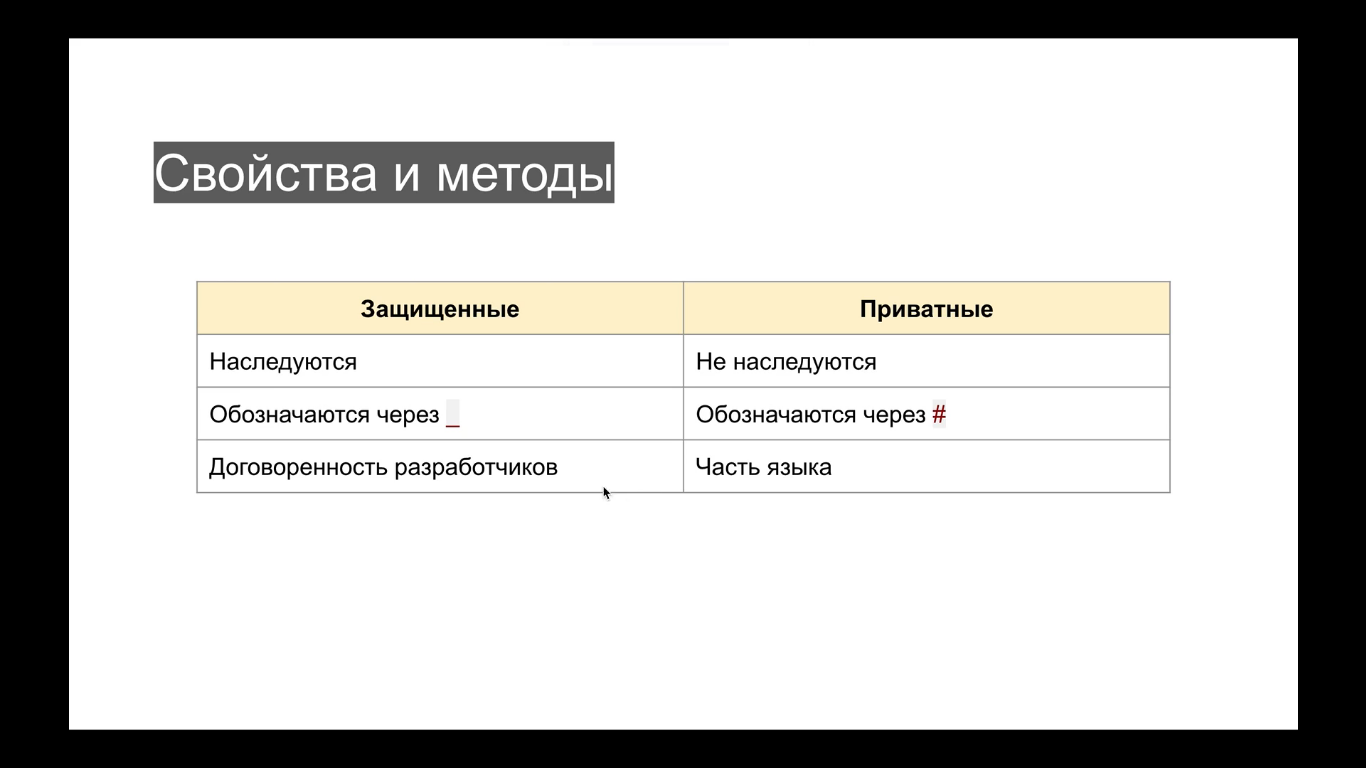




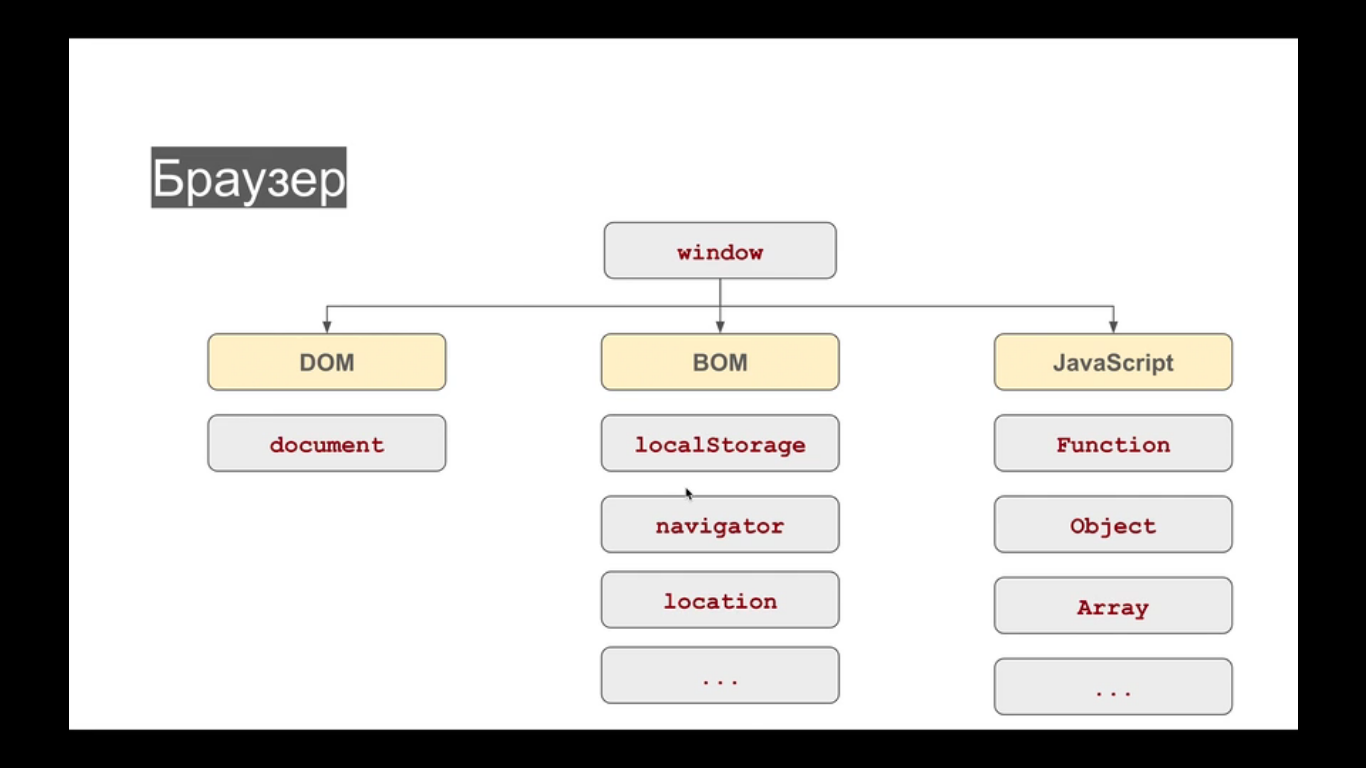
**Урок 20 (Classes)**

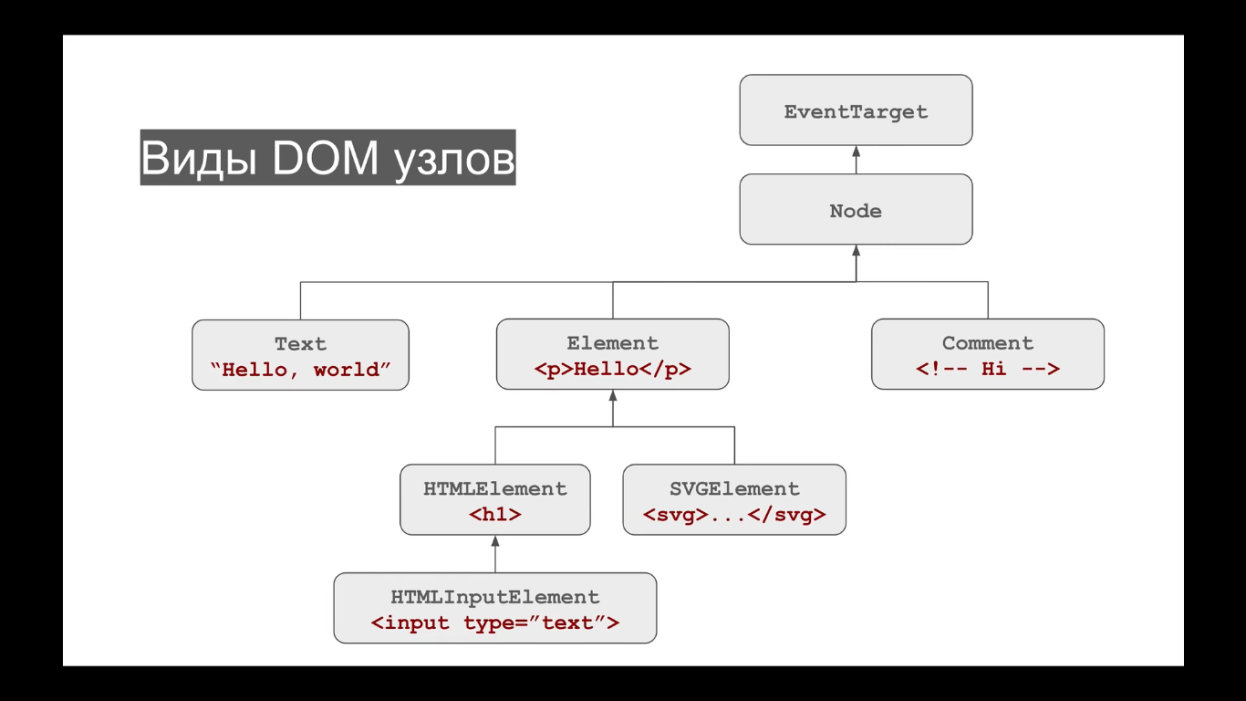


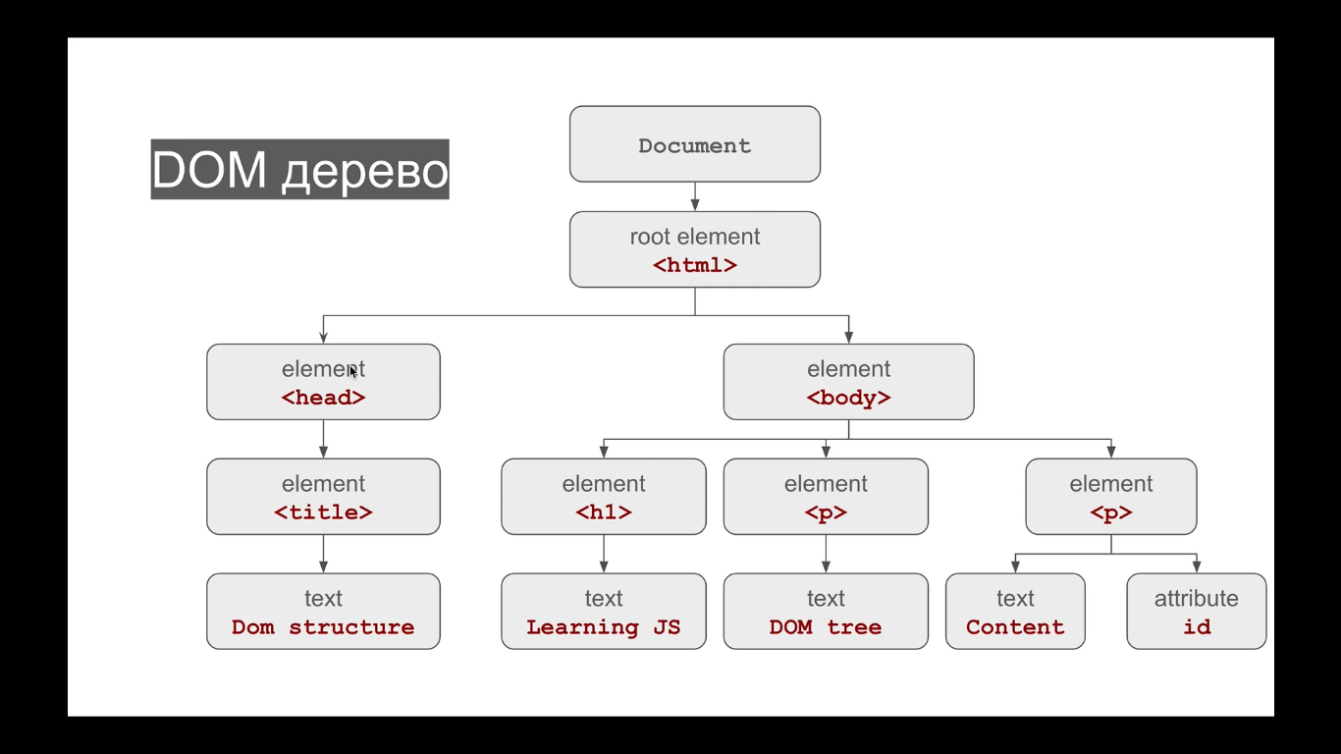
* **class User ( ) { }.....methods( ) { }** // створення обєкту за допомогою класу
* **class Ship extends Vehicle ( ) { }** // клас ship прийняв методи з класу Vehicle
* constructor ( name, speed) {this.name=name; **super**(name, false) // наслідування з батьківського класу
* move ( ) { this.move( ) } // нескінченний цикл, бо функція саме себе викликає, щоб уникнути то можна написати: move ( ) { super.move( ) } // метод move ссилається на батьківський клас
* **static** compare ( ) { } // не статичні методи не видні в класах, статичні методи не видні в екземплярах класу
* \_balance - нижня риска позначка для розробників про приватні властивості і методи
* # -- значок котрий робить свойства і методи приватними і до них немає доступу

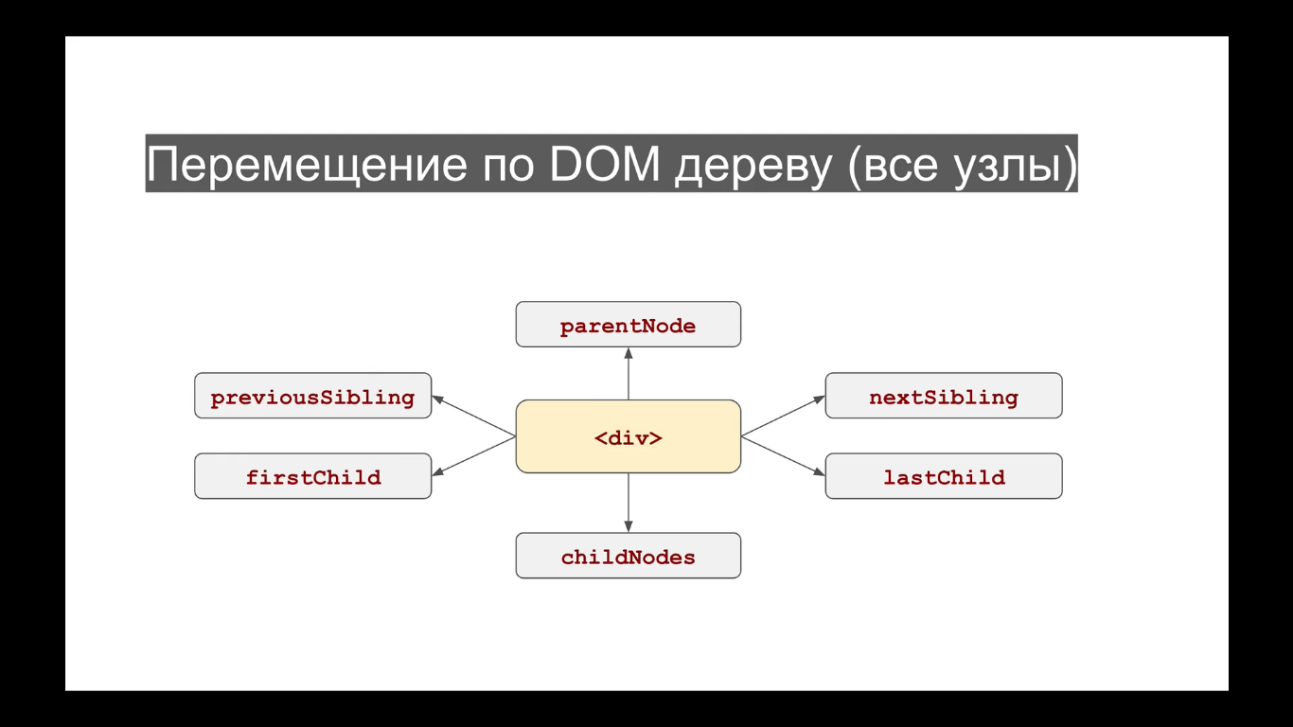


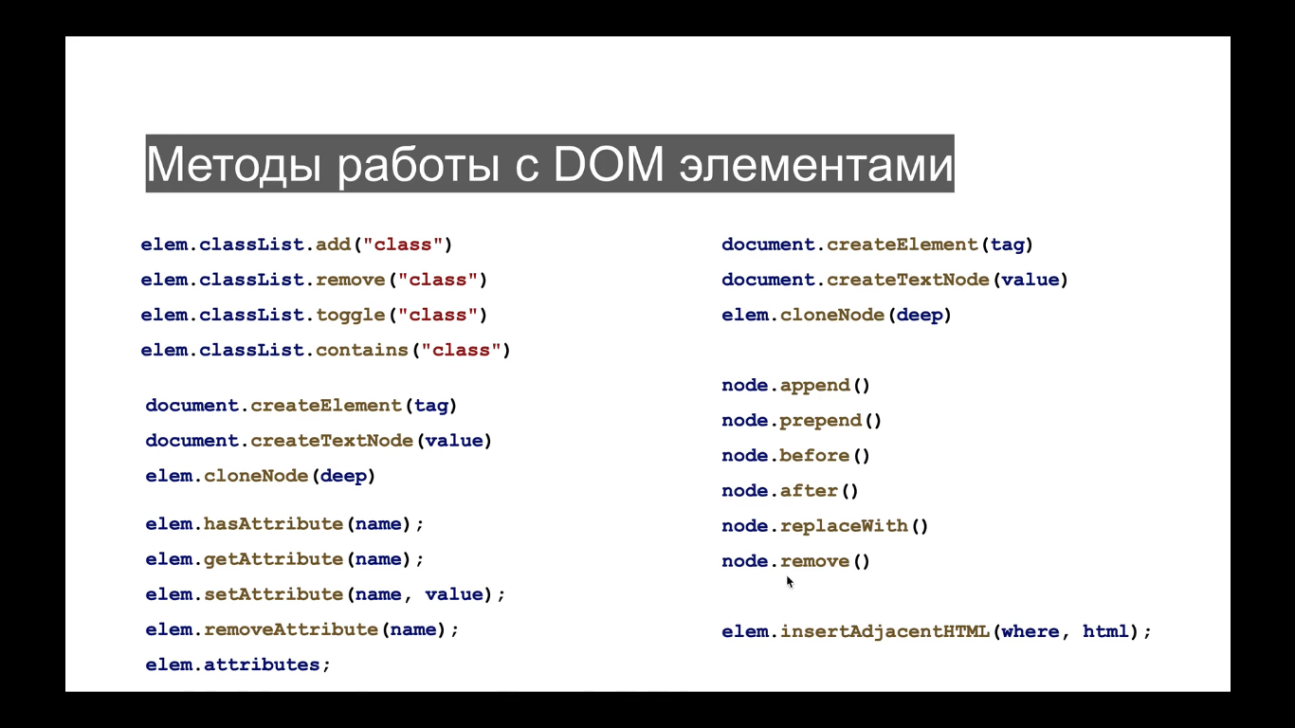
**Урок 21 Object Model**

****

****

****

****

****