

Об'єкти - продовження

**Методи об'єктів - опис**

Об'єкти реальної дійсності характеризуються:

- деякими властивостями-характеристиками (деякими величинами (ім'я користувача, зріст, вік, ....))
- функціональними можливостями (діями, що може виконати сам об'єкт, або можна виконати над об'єктом). Функціональні можливості об'єкта описують за допомогою **методів** (властивостей – функцій).

Об'єкт реальної дійсності	Загальна форма опису
<div>----- Властивості ----- (параметри, характеристики)<ul style="list-style-type: none"><li>• властивість-характеристика 1</li><li>• властивість-характеристика 2</li><li>.....</li><li>• властивість-характеристика N</li></ul></div> <div>----- Функціональні можливості ----- (дії, які може виконувати сам об'єкт або можна виконувати з об'єктами)<div><ul style="list-style-type: none"><li>• функціональна можливість (дія) 1</li><li>• функціональна можливість (дія) 2</li><li>.....</li><li>• функціональна можливість (дія) M</li></ul></div></div>	<pre>let об'єкт = {   //----- Властивості-характеристики -----   <u>властивість1</u> : <u>значення1</u>,   <u>властивість2</u> : <u>значення2</u>,   ...   <u>властивістьN</u> : <u>значенняN</u>    //----- Властивості-методи -----   <u>властивість-метод1</u>: function () {     . . . . .   },   <u>властивість-метод2</u>: function () {     . . . . .   },   . . . . .   <u>властивість-методM</u>: function () {     . . . . .   } }</pre>

Методи об'єктів - опис. Методи – це властивості-функції

Об'єкт реальної дійсності – Користувач	Приклад опису user	Загальна форма опису
<div>----- Властивості -----<ul style="list-style-type: none"><li>ім'я</li><li>прізвище</li><li>вік</li></ul></div> <div>--- Функціональні можливості ---<div><div>• вивести привітання</div><div>• попрощатися</div></div></div>	<pre>let user = {    //----- Властивості -----   name: "Іван",    surname: "Петров",    age: 25,    //----- Методи -----   sayHi: function () {     alert('Привіт!');   },   sayBye: function () {     alert('До побачення!');   } }</pre>	<pre>let об'єкт = {    //---- Властивості-характеристики ----   <u>властивість1</u> : <u>значення1</u>,   <u>властивість2</u> : <u>значення2</u>,   ...   <u>властивістьN</u> : <u>значенняN</u>    //----- Властивості-методи -----   <u>властивість-метод1</u>: function () {     . . . . .   },   <u>властивість-метод2</u>: function () {     . . . . .   },   . . . . .   <u>властивість-методM</u>: function () {     . . . . .   } }</pre>

Методи об'єктів – звертання до методів.

Звертання до методів аналогічне, як і до звичайних властивостей

Об'єкт реальної дійсності – <i>Користувач</i>	Приклад опису <i>user</i>
	<code>let user = {</code>
----- Властивості ----- <ul style="list-style-type: none"><li>ім'я</li><li>прізвище</li><li>вік</li></ul>	<code>//----- Властивості -----   name: "Іван",    surname: "Петров",    age: 25,</code>
--- Функціональні можливості --- <ul style="list-style-type: none"><li>вивести привітання</li><li>попрощатися</li></ul>	<code>//----- Методи -----    sayHi: function () {     alert('Привіт!');   },    sayBye: function () {     alert('До побачення!');   }  }</code>

Звертання до методів

об'єкт . назва\_властивості (фактичні\_параметри)

Приклад.

`user . sayHi();`     // Привіт!

`user . sayBye();`     //До побачення!

об'єкт [ назва\_властивості ] (фактичні\_параметри)

Приклад.

`user [ 'sayHi' ] ( );`     // 'Привіт!'

`user [ 'sayBye' ] ( );`     //До побачення!

Методи об'єктів - звертання до інших властивостей та методів

- Для доступу до об'єкта з методу використовується ключове слово **this**.
- Значення **this** називається **контекстом виклику** і буде визначено в момент виклику функції.
- У методі значенням **this** є об'єкт, в контексті якого викликаний метод

Об'єкт реальної дійсності – Користувач	Приклад опису <i>user</i>	Загальна форма опису
<div>----- Властивості -----<ul style="list-style-type: none"><li>• ім'я</li><li>• прізвище</li><li>• вік</li></ul></div> <div>--- Функціональні можливості ---<ul style="list-style-type: none"><li>• вивести привітання</li><li>• попрощатися</li><li>• вивести привітання</li></ul></div>	<pre>let user = {    //----- Властивості -----   name: "Іван",   surname: "Петров",   age: 25,    //----- Методи -----    sayHi: function () {     alert(`Привіт від \${this.name}!`);   },   sayBye: function () {     alert(`До побачення!`);   },   greeting: function () {     this . sayHi ()     this . sayBye ()   } }</pre>	<pre>let об'єкт = {    //---- Властивості-характеристики ----   <u>властивість1</u> : <u>значення1</u>,   <u>властивість2</u> : <u>значення2</u>,   ...    //----- Властивості-методи -----    <u>властивість-метод1</u>: function () {     . . . . .     this. <u>властивість</u>     . . . . .     this. <u>метод (...)</u>     . . . . .   },   . . . . . }</pre>

Методи об'єктів - звертання до інших властивостей та методів

- Для доступу до об'єкта з методу використовується ключове слово **this**.
- Значення **this** називається **контекстом виклику** і буде визначено в момент виклику функції.
- У методі значенням **this** є об'єкт, в контексті якого викликаний метод

Об'єкт реальної дійсності – Користувач	Приклад опису <i>user</i>	Загальна форма опису
<div>----- Властивості -----<ul style="list-style-type: none"><li>• ім'я</li><li>• прізвище</li><li>• вік</li></ul></div> <div>--- Функціональні можливості ---<ul style="list-style-type: none"><li>• вивести привітання</li><li>• попрощатися</li><li>• вивести привітання</li></ul></div>	<pre>let user = {    //----- Властивості -----   name: "Іван",   surname: "Петров",   age: 25,    //----- Методи -----    sayHi: function () {     alert(`Привіт від \${this.name}!`);   },   sayBye: function () {     alert(`До побачення!`);   },   greeting: function () {     this.sayHi();     this.sayBye();   } }</pre>	<pre>let об'єкт = {    //---- Властивості-характеристики ----   <u>властивість1</u> : <u>значення1</u>,   <u>властивість2</u> : <u>значення2</u>,   ...    //----- Властивості-методи -----    <u>властивість-метод1</u>: function () {     ...     this. <u>властивість</u>     ...     this. <u>метод (...)</u>     ...   },   ... }</pre>

# Методи об'єктів

Об'єкт реальної дійсності – <i>Учень</i>	Приклад опису <i>user</i>	Загальна форма опису
		<code>let об'єкт = {</code>
<div>----- Властивості -----</div> <ul style="list-style-type: none"><li>• <i>Ім'я</i></li><li>• <i>Клас</i></li><li>• <i>Оцінки з 3-х предметів</i></li></ul>		<div>//---- Властивості-характеристики ----</div> <div>-</div> <div><i>властивість1</i> : <i>значення1</i>,</div> <div><i>властивість2</i> : <i>значення2</i>,</div> <div>...</div>
<div>--- Функціональні можливості ---</div> <ul style="list-style-type: none"><li>• <i>Знаходження середнього балу</i></li></ul>		<div>//----- Властивості-методи -----</div> <div><i>властивість-метод1</i>: function () {</div> <div>    . . . . .</div> <div>    this. <i>властивість</i></div> <div>    . . . . .</div> <div>    this. <i>метод (...)</i></div> <div>    . . . . .</div> <div>},</div> <div>    . . . . .</div>
		<code>}</code>

Методи об'єктів

Об'єкт реальної дійсності – <i>Учень</i>	Приклад опису <i>user</i>	Загальна форма опису
	<code>let pupil = {</code>	<code>let об'єкт = {</code>
<div>----- Властивості -----<ul style="list-style-type: none"><li>• <i>Ім'я</i></li><li>• <i>Клас</i></li><li>• <i>Оцінки з 3-х предметів</i></li></ul></div>	<div>//----- Властивості -----     <i>Name</i>      : "Ivan",     <i>class_</i>   : "8B",     <i>marks</i>    : [10, 11, 12],</div>	<div>//---- Властивості-характеристики ---- -     <u><i>властивість1</i></u> : <i>значення1</i>,     <u><i>властивість2</i></u> : <i>значення2</i>,     ...  //----- Властивості-методи -----      <u><i>властивість-метод1</i></u>: function () {         . . . . .         this. <u><i>властивість</i></u>         . . . . .          this. <u><i>метод (...)</i></u>         . . . . .     },     . . . . .</div>
<div>--- Функціональні можливості ---      • <i>Знаходження середнього балу</i></div>		
	<code>}</code>	<code>}</code>



## Методи об'єктів

Об'єкт реальної дійсності – <i>Учень</i>	Приклад опису <i>user</i>	Загальна форма опису
<p>----- Властивості -----</p> <ul style="list-style-type: none"> <li>• <i>Ім'я</i></li> <li>• <i>Клас</i></li> <li>• <i>Оцінки з 3-х предметів</i></li> </ul>	<pre>let pupil = {     //----- Властивості -----     Name    : "Ivan",     class_   : "8B",     marks    : [10, 11, 12],</pre>	<pre>let об'єкт = {     //---- Властивості-характеристики ----     -     <u>властивість1</u> : <u>значення1</u>,     <u>властивість2</u> : <u>значення2</u>,     ...</pre>
<p>--- Функціональні можливості ---</p> <ul style="list-style-type: none"> <li>• <i>Знаходження середнього балу</i></li> </ul>	<pre>    //----- Методи -----      getAverage: function () {         let s = 0;         for (let i = 0; i &lt; this.marks.length; i++)         {             s += this.marks[i];         }         return s / this.marks.length;     } }</pre>	<pre>    //----- Властивості-методи -----      <u>властивість-метод1</u>: function () {         . . . . .         this. <u>властивість</u>         . . . . .          this. <u>метод (...)</u>         . . . . .     },     . . . . .</pre>
	}	}

Методи об'єктів. Формальні параметри

При описі методів об'єктів як формальні параметри треба передвати тільки те, чого у об'єкті немає

Об'єкт реальної дійсності – <i>Учень</i>	Приклад опису <i>user</i>	Загальна форма опису
<div>----- Властивості -----<ul style="list-style-type: none"><li>• <i>Ім'я</i></li><li>• <i>Клас</i></li><li>• <i>Оцінки з 3-х предметів</i></li></ul></div> <div>--- Функціональні можливості ---<ul style="list-style-type: none"><li>• <i>Знаходження середнього балу (всі дані є в об'єкті, нічого додатково передвати у функцію не потрібно)</i></li><li>• <i>Підрахувати скільки раз зустрічається вказана оцінка (потрібно додатково вказати оцінку <code>searchScore</code>, кількість якої підраховуємо)</i></li></ul></div>	<pre>let pupil = {    //----- Властивості -----   Name   : "Ivan",   class_ : "8B",   marks  : [10, 11, 12],    //----- Методи -----    getAverage: function () {     let s = 0;     for (let i = 0; i &lt; this.marks.length; i++)     {       s += this.marks[i];     }     return s / this.marks.length;   }    getScoreNumber: function (searchScore) {     return this.marks.reduce(       (prevCount, score) =&gt;         score===searchScore ? prevCount+1         : prevCount,       0     )   },  }</pre>	<pre>let об'єкт = {    //---- Властивості-характеристики ----   -   <u>властивість1</u> : <u>значення1</u>,   <u>властивість2</u> : <u>значення2</u>,   ...    //----- Властивості-методи -----    <u>властивість-метод1</u>: function () {     . . . . .     this. <u>властивість</u>     . . . . .     this. <u>метод (...)</u>     . . . . .   },   . . . . . }</pre> <div>Приклад виклику. Підрахувати скільки є оцінок 10 pupil.getScoreNumber(10)</div>

Приклад. Створити об'єкт товар. Для товару зберігаються назва товару, ціна, кількість та вартість зберігання в день. Передбачити можливість знаходження загальної вартості наявної кількості одиниць, визначення вартості зберігання для заданої кількості днів, зменшення ціни на вказану кількість відсотків та збільшення ціни на вказану кількість відсотків

Приклад. Створити об'єкт товар.

----- Властивості - характеристики -----

- назва товару
- ціна
- кількість
- вартість зберігання в день

----- Методи (функціональні можливості) -----

- визначення загальної вартості наявної кількості одиниць
- визначити вартість зберігання для заданої кількості днів
- зменшення ціни на вказану кількість відсотків
- збільшення ціни на вказану кількість відсотків

Спеціальні методи об'єктів – перетворення у рядок «toString»

**toString** - метод, що використовується при перетворенні об'єкта до рядка (програміст сам вирішує у залежності від задачі що повинна повертати дана функція). Якщо такого метода немає, то виводиться інформація про тип об'єкта

Об'єкт реальної дійсності – <i>Учень</i>	Приклад опису <i>user</i>	Загальна форма опису
<div>----- Властивості -----<ul style="list-style-type: none"><li>• <i>Ім'я</i></li><li>• <i>Клас</i></li><li>• <i>Оцінки з 3-х предметів</i></li></ul></div>	<pre>let pupil = {    //----- Властивості -----   Name    : "Ivan",   class_  : "8B",   marks   : [10, 11, 12],    //----- Методи -----    getAverage: function () {     let s = 0;     for (let i = 0; i &lt; this.marks.length; i++)     {       s += this.marks[i];     }     return s / this.marks.length;   }    toString: function () {     return `\${this.name}, \${this.marks},            оцінки: \${this.marks}`   },  }</pre>	<pre>let об'єкт = {    //---- Властивості-характеристики ----   -   <u>властивість1</u> : <u>значення1</u>,   <u>властивість2</u> : <u>значення2</u>,   ...    //----- Властивості-методи -----    <u>властивість-метод1</u>: function () {     . . . . .     this. <u>властивість</u>     . . . . .     this. <u>метод (...)</u>     . . . . .   },   . . . . .    toString: function () {     . . . . .     return <u>рядкове_представння_об'єкта</u>   }  }</pre>

Спеціальні методи об'єктів – перетворення у рядок «toString»

**toString** - метод, що використовується при перетворенні об'єкта до рядка (програміст сам вирішує у залежності від задачі що повинна повертати дана функція)

Об'єкт реальної дійсності – <i>Учень</i>	Приклад опису <i>user</i>
----- Властивості ----- <ul style="list-style-type: none"><li>• <i>Ім'я</i></li><li>• <i>Клас</i></li><li>• <i>Оцінки з 3-х предметів</i></li></ul>	<pre>let pupil = {    //----- Властивості -----   Name    : "Ivan",   class_  : "8B",   marks   : [10, 11, 12],    //----- Методи -----    getAverage: function () {     let s = 0;     for (let i = 0; i &lt; this.marks.length; i++)     {       s += this.marks[i];     }     return s / this.marks.length;   }    toString: function () {     return `\${this.name}, \${this.marks},            оцінки: \${this.marks}`   },  }</pre>
---- Функціональні можливості ---- <ul style="list-style-type: none"><li>• <i>Знаходження середнього балу</i></li></ul> <div>• <i>Перетворення у string</i></div>	

неявне перетворення у string

alert(pupil)

//аналог  
//alert(pupil.toString())

неявне перетворення у string у виразі

let s1 = 'Data: ' + pupil

//аналог  
// 'Data: ' + pupil.toString()  
document.write(s1)

явний виклик метода toString

let s2 = pupil.toString()  
document.write(s2)

Спеціальні методи об'єктів – перетворення у число

**valueOf** - метод, що використовується при перетворенні об'єкта до числа (програміст сам вирішує у залежності від задачі що повинна повертати дана функція) . Якщо такого метода немає, то використовується **toString** з подальшим перетворенням у число.

Об'єкт реальної дійсності – <i>Учень</i>	Приклад опису <i>user</i>	Загальна форма опису
<div>----- Властивості -----</div> <ul style="list-style-type: none"><li>• <i>Ім'я</i></li><li>• <i>Клас</i></li><li>• <i>Оцінки з 3-х предметів</i></li></ul>	<pre>let pupil = {    //----- Властивості -----   Name    : "Ivan",   class_  : "8B",   marks   : [10, 11, 12],    //----- Методи -----    getAverage: function () {     let s = 0;     for (let i = 0; i &lt; this.marks.length; i++)     {       s += this.marks[i];     }     return s / this.marks.length;   }    valueOf: function () {     return this.getAverage ()   },  }</pre>	<pre>let об'єкт = {    //---- Властивості-характеристики ----   -   <u>властивість1</u> : <u>значення1</u>,   <u>властивість2</u> : <u>значення2</u>,   ...    //----- Властивості-методи -----    <u>властивість-метод1</u>: function () {     . . . . .     this. <u>властивість</u>     . . . . .     this. <u>метод (...)</u>     . . . . .   },   . . . . .    valueOf: function () {     . . . . .     return <u>числове_представння_об'єкта</u>   }  }</pre>
<div>--- Функціональні можливості ---</div> <ul style="list-style-type: none"><li>• <i>Знаходження середнього балу</i></li></ul> <div>• <i>Перетворення у число</i></div>		

Спеціальні методи об'єктів – перетворення у число

**valueOf** - метод, що використовується при перетворенні об'єкта до числа (програміст сам вирішує у залежності від задачі що повинна повертати дана функція). Якщо такого метода немає, то використовується **toString** з подальшим перетворенням у число.

Об'єкт реальної дійсності – <i>Учень</i>	Приклад опису <i>user</i>
<div>----- Властивості -----<ul style="list-style-type: none"><li>• <i>Ім'я</i></li><li>• <i>Клас</i></li><li>• <i>Оцінки з 3-х предметів</i></li></ul></div>	<pre>let pupil = {    //----- Властивості -----   Name    : "Ivan",   class_  : "8B",   marks   : [10, 11, 12],    //----- Методи -----    getAverage: function () {     let s = 0;     for (let i = 0; i &lt; this.marks.length; i++)     {       s += this.marks[i];     }     return s / this.marks.length;   }    valueOf: function () {     return this.getAverage ()   },  }</pre>
<div>---- Функціональні можливості ----<ul style="list-style-type: none"><li>• <i>Знаходження середнього балу</i></li></ul></div> <div>• <i>Перетворення у число</i></div>	

неявне перетворення у виразі

alert(pupil \* 3)

//аналог  
//alert(pupil.valueOf() \* 3)

явний виклик метода valueOf

let num = pupil.valueOf()  
  
document.write(num)



Приклад. Описати об'єкт «Інвойс»

----- Властивості-характеристики (дані) -----

номер інвойса

переліз виконаних робіт

сума грошей

----- Методи -----

перетворення у рядок (вивести номер інвойса, кількість виконаних робіт, сума грошей

перетворення у число (повертається сума грошей)

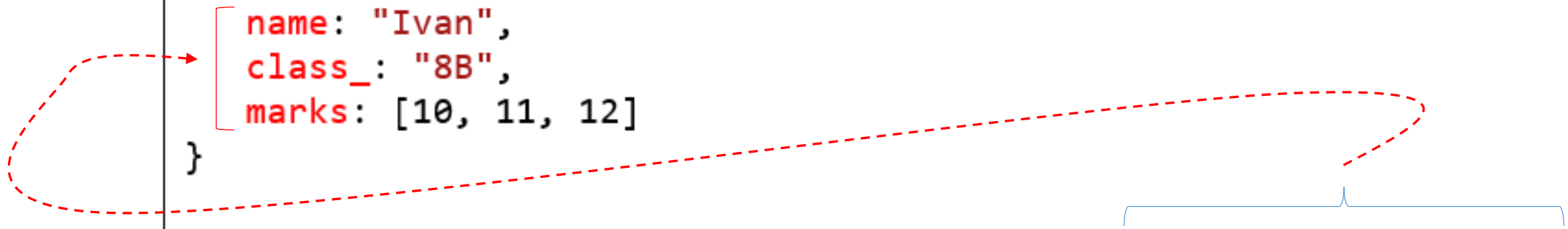
Використовуючи такий опис створити масив інвойсів. Вивести інформацію про інвойси у формі нумерованого списку та знайти загальну вартість використовуючи метод перетворення об'єкта у число

**Спеціальні методи об'єктів – блокування об'єктів (константні об'єкти)**

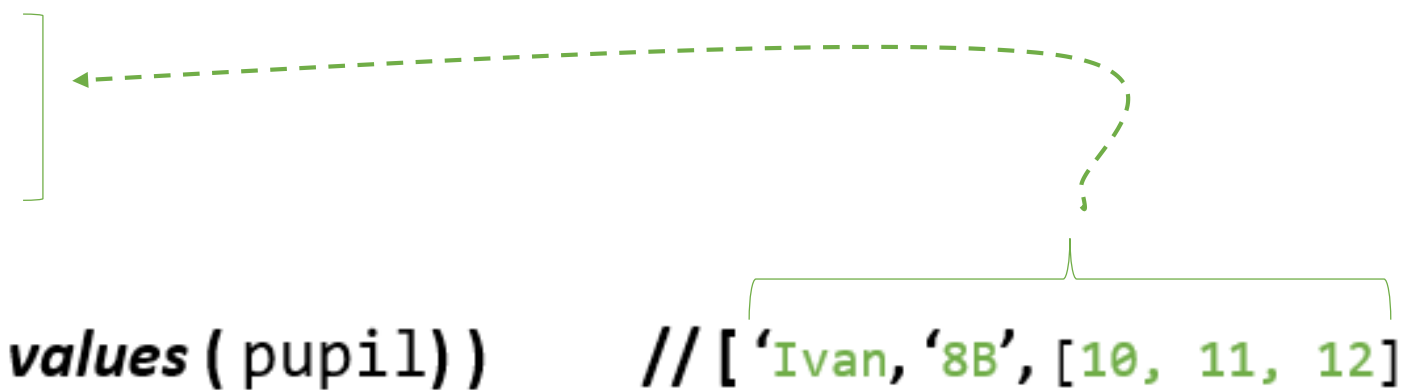
Для того, щоб заборонити зміну значень об'єкта необхідно використати ***Object.freeze***

Загальна форма	<b><i>Object . freeze</i></b> ( <span style="border: 1px dashed black; padding: 2px;">об'єкт_що_потрібно_зафіксувати</span> )
Приклад	<pre>let pupil = {   name: "Ivan",   class_: "8B",   marks: [10, 11, 12] }  <b><i>Object.freeze</i></b> ( pupil )  pupil.name = 'Petro' // pupil.name = 'Ivan' (не змінено) pupil.height = 150   // height - не додано delete pupil.name    // name - не видалено  console.log(pupil)</pre>

## Спеціальні методи об'єктів – список назв властивостей об'єкта

Загальна форма	<b><i>Object . keys</i></b> ( <i>об'єкт_для_якого_треба_знайти_список_його_властивостей</i> )
Приклад	<pre>let pupil = {   name: "Ivan",   class_: "8B",   marks: [10, 11, 12] }  console.log( <b><i>Object . keys</i></b> ( pupil ) )    // [ 'name', 'class_', 'marks' ]</pre> 

## Спеціальні методи об'єктів - список значень властивостей об'єкта

Загальна форма	<b><i>Object . values</i></b> ( <span style="border: 1px dashed black; padding: 2px;">об'єкт_для_якого_треба_знайти_список_значень_властивостей</span> )
Приклад	<pre>let pupil = {   name: "Ivan",   class_: "8B",   marks: [10, 11, 12] }  console.log( <b><i>Object . values</i></b> ( pupil ) )</pre>  <pre>// [ 'Ivan', '8B', [10, 11, 12] ]</pre>

## При створенні багатьох об'єктів

- легко помилитись у назвах властивостей
- потрібно весь час повторювати описи методів

```
let book2 = {  
  title: 'JavaScript',  
  year: 2022,  
  pagesNumber: 1500,  
  price: 500,  
  toString: function () {  
    return `${this.title} - ${this.year}`  
  },  
}
```

```
let book3 = {  
  title: 'Python',  
  year: 2021,  
  pagesNumber: 1800,  
  price: 800,  
  toString: function () {  
    return `${this.title} - ${this.year}`  
  },  
}
```

## При створенні багатьох об'єктів

- легко помилитись у назвах властивостей
- потрібно весь час повторювати описи методів

```
let book2 = {  
  title: 'JavaScript',  
  year: 2022,  
  pagesNumber: 1500,  
  price: 500,  
  toString: function () {  
    return `${this.title} - ${this.year}`  
  },  
}
```

```
let book3 = {  
  title: 'Python',  
  year: 2021,  
  pagesNumber: 1800,  
  price: 800,  
  toString: function () {  
    return `${this.title} - ${this.year}`  
  },  
}
```

## Функції - фабрики

### При створенні багатьох об'єктів

- легко помилитись у назвах властивостей
- потрібно весь час повторювати описи методів

**Можна створити функцію-фабрику, що буде створювати об'єкти з використанням початкових значень властивостей**

```
//--- Функція-фабрика, що створює об'єкт ґрунтуючись на початкових значеннях полів ---  
function getBookObject(initTitle, initYear, initPagesNumber, initPrice) {  
  return {  
    title: initTitle,  
    year: initYear,  
    pagesNumber: initPagesNumber,  
    price: initPrice,  
    toString: function () {  
      return `${this.title} - ${this.year}`  
    },  
  },  
}
```

параметри – початкові значення

формуємо об'єкт, використовуючи передані початкові значення і повертаємо його

для створення об'єкта викликаємо функцію-фабрику і передаємо потрібні початкові значення полів

```
//--- Створення об'єктів з використанням функцій-фабрик ---  
let book2 = getBookObject('JavaScript', 2022, 1500, 500);  
let book3 = getBookObject('Python', 2021, 1800, 800)
```

## Функції - фабрики

Без використання функцій-фабрик

```
let book1 = {
  title: 'JavaScript',
  year: 2022,
  pagesNumber: 1500,
  price: 500,
  toString: function () {
    return `${this.title} - ${this.year}`
  },
}

let book2 = {
  title: 'Python',
  year: 2020,
  pagesNumber: 1800,
  price: 700,
  toString: function () {
    return `${this.title} - ${this.year}`
  },
}

let book3 = {
  title: 'C#',
  year: 2023,
  pagesNumber: 2500,
  price: 600,
  toString: function () {
    return `${this.title} - ${this.year}`
  },
}
```

З використанням функцій-фабрик

```
//--- Функція-фабрика, що створює об'єкт ґрунтуючись
//..... на початкових значеннях полів ---
function getBookObject(initTitle, initYear,
  .....|.....|.....|.....|.....|initPagesNumber, initPrice) {
  .....return {
  .....  title: initTitle,
  .....  year: initYear,
  .....  pagesNumber: initPagesNumber,
  .....  price: initPrice,
  .....  toString: function () {
  .....    return `${this.title} - ${this.year}`
  .....  },
  .....}
}

//--- Створення об'єктів з використанням функцій-фабрик ---
let book1 = getBookObject('JavaScript', 2022, 1500, 500)
let book3 = getBookObject('Python', 2020, 1800, 700)
let book2 = getBookObject('C#', 2023, 2500, 600).....
```



## Конструктори об'єктів

- це функція (схожа на функцію-фабрику), яку використовуємо для створення об'єктів (як параметри передаємо значення, що використовуються при створенні об'єкта (початкові значення полів))
- викликаємо (створюємо об'єкт) з використанням оператора **new** (об'єкт = **new** конструктор (... параметри ...))
- в середині функції-конструктора об'єкт, який створюємо представлений за допомогою **this** (тобто властивості і методи додаємо до this). Повертати this (return this) як у методі-фабриці не потрібно (це робиться автоматично)

Загальна форма конструктора	Об'єкт реальної дійсності	Приклад опису конструктора
<pre>function ім'я_функції-конструктора (...параметри...){      //----- Опис полів -----     this.властивість1 = значення1;     this.властивість2 = значення2;      . . .      //----- Опис методів -----     this.функція-метод_1 = function (форм.парам.)     {         . . .     }     this.функція-метод_2 = function (форм.парам.)     {         . . .     }     . . . }</pre>	<p>Приклад. Описати об'єкт реальної дійсності «Діапазон чисел»</p> <p>--Властивості-характеристики--</p> <ul style="list-style-type: none"><li>• Мінімальне значення</li><li>• Максимальне значення</li></ul> <p>----- Властивості-методи -----</p> <ul style="list-style-type: none"><li>• Визначення чи належить деяке число діапазону</li><li>• Отримання випадкового числа з діапазону</li><li>• Рядкове представлення об'єкта (toString)</li></ul>	<pre>//Описати об'єкт реальної дійсності «Діапазон чисел»  function Range (defaultMin, defaultMax) {      //----- Опис полів     this.min = defaultMin     this.max = defaultMax      //----- Методів     this.isInRange = function (value) {         return (value &gt;= this.min &amp;&amp;                 value &lt;= this.max)     }      this.getRandomValueFromRange = function () {         return this.min + Math.floor(Math.random() *             (this.max -this.min + 1))     }      this.toString = function () {         return ` [\${this.min} ; \${this.max} ]"     } }  //----- Створення об'єкта ----- let r1 = new Range(1,12)</pre>

Описувати методи в конструкторі не дуже добре (кожен об'єкт буде мати свої копії функцій)

Конструктори об'єктів. Опис спільних для всіх об'єктів властивостей

Функція конструктор

- має властивість **prototype** (прототип функції) (функція є об'єктом *Function*)
- **усі властивості, які додані до *prototype* будуть спільними для всіх об'єктів, що створені з використанням цього конструктора**

Загальна форма конструктора	Приклад опису конструктора
<pre>function <b>функція-конструктор</b> (...параметри...){   //----- Опис полів (індивідуальні дані об'єктів) -----   <b>this.властивість1</b> = <b>значення1</b>;   <b>this.властивість2</b> = <b>значення2</b>;   . . . }</pre> <div>Поля додаємо у конструктор</div> <pre>//----- Опис методів (спільні дані для усіх об'єктів) ----- <b>конструктор.prototype.функція-метод_1</b> = function (форм.парам.) {   . . . } <b>конструктор.prototype.функція-метод_2</b> = function (форм.парам.) {   . . . } . . . . .</pre> <div>Методи додаємо у <i>prototype</i></div> <pre>//---- створення об'єкта ---- об'єкт = <b>new</b> конструктор (... параметри ...)</pre>	<pre>function <b>Range</b> (defaultMin, defaultMax) {   //--- Опис полів (індивідуальних даних об'єктів)---   -   <b>this.min</b> = defaultMin   <b>this.max</b> = defaultMax }  //----- Методи (спільні для усіх об'єктів) <b>Range.prototype.isInRange</b> = function (value) {   return (value &gt;= this.min &amp;&amp;           value &lt;= this.max) }  <b>Range.prototype.getRandomValueFromRange</b> = function() {   return this.min + Math.floor(Math.random() *                                 (this.max -this.min + 1)) }  <b>Range.prototype.toString</b> = function () {   return ` [\${this.min} ; \${this.max} ]` }  //----- Створення об'єкта ----- let r1 = <b>new</b> Range(1,12) let num = r1. <b>getRandomValueFromRange()</b></pre>

## ***Конструктори об'єктів. Опис спільних для всіх об'єктів властивостей (prototype)***

Приклад. Гра «Рулетка»

----- Властивості -----

- кількість полів рулетки
- мінімальне значення балів
- максимальне значення балів
- список згенерованих значень

----- Методи -----

- генерування полів рулетки
- виведення списку згенерованих значень
- приведення до рядка
- крутити рулетку (отримання випадкового балу)
- метод гри (користувач крутить рулетку поки не відмовиться)

поки користувач хоче крутити

визначаємо рандомне значення рулетки

додаємо до загальної суми

повідомляємо користувача про результат та загальну кількість балів

## Значення *this* у функціях

звичайні (не стрілкові) функції контекст беруть з зовнішнього оточення під час виклику

1)Такий опис *var myVar* додасть її як властивість *window (this)*

```
function testFunc() {  
  let myVar = 0  
  console.log(this) //window  
  document.write(this.myVar) //window.myVar = 22  
}  
  
var myVar = 22 //window.myVar = 22  
  
testFunc()
```

3) При використанні «use strict» this = undefined

```
'use strict'  
function testFunc() {  
  let myVar = 0  
  console.log(this) //undefined  
  document.write(this.myVar) //undefined.myVar - error  
}  
  
var myVar = 22  
  
testFunc()
```

2)Такий опис *let myVar* не додасть її як властивість *window (this)*

```
function testFunc() {  
  let myVar = 0  
  console.log(this) //window  
  document.write(this.myVar) //undefined  
}  
  
let myVar = 22  
  
testFunc()
```

## Значення *this* у функціях

стрілкові функції контекст беруть з зовнішнього оточення під час опису (і фіксують його)

*this* береться з зовнішнього оточення  
(у даному випадку це *window*)

```
var myVar = 77
let func2 = () => {
  console.log(this) //window
  document.write(this.myVar) //window.myVar = 77
}
func2()
```

Під час виклику *this* береться з зовнішнього оточення

```
let obj1 = {
  myVar: 77,
  method1: function () {
    let func = function () {
      console.log(this) //this = window
      document.write(this.myVar) //obj1.myVar = undefined
    }
    return func
  },
}

let f = obj1.method1()
f()
```

Під час опису *this* береться з зовнішнього оточення і фіксує його

```
let obj1 = {
  myVar: 77,
  method1: function () {
    let func = () => {
      console.log(this) //this = obj1
      document.write(this.myVar) //obj1.myVar = 77
    }
    return func
  },
}

let f = obj1.method1()
f()
```

## Значення *this* у функціях

стрілкові функції контекст беруть з зовнішнього оточення під час опису (і фіксують його)

```
let obj1 = {
  names: ['Ivan', 'Petro', 'Olga', 'Hanna'],
  positions: ['Driver', 'Manager', 'Programmer', 'Director'],
  showNames: function () {
    this.names.forEach(function (name, index) {
      document.write(
        `${name} ${this.positions[index]} <br>`
      )
    })
  },
}
obj1.showNames()
```

Під час виклику звичайної функції  
this=window

**Результат роботи програми**

Ivan undefined  
Petro undefined  
Olga undefined  
Hanna undefined

```
let obj1 = {
  names: ['Ivan', 'Petro', 'Olga', 'Hanna'],
  positions: ['Driver', 'Manager', 'Programmer', 'Director'],
  showNames: function () {
    this.names.forEach((name, index) => {
      document.write(
        `${name} ${this.positions[index]} <br>`
      )
    })
  },
}
obj1.showNames()
```

Під час виклику стрілкової функції  
this=obj1

**Результат роботи програми**

Ivan Driver  
Petro Manager  
Olga Programmer  
Hanna Director

## Позичання функцій-методів

у об'єкт можна додати новий метод, беручи його з іншого метода (не дуже добре, бо додаємо нові властивості у об'єкт)

*об'єкт\_2 . назва\_метода = об'єкт\_1 . назва\_власного\_метода*

```
let obj1 = {  
  prop1: 11,  
  prop2: 22,  
  showProp1: function () {  
    document.write(this.prop1)  
  },  
}
```

```
let obj2 = {  
  prop1: 21,  
  prop3: 23,  
}
```

```
obj2.myShow = obj1.showProp1
```

```
obj2.myShow() // 21
```

Зараз **obj2** має метод **myShow**

```
obj2 = {  
  prop1: 21,  
  prop3: 23,  
  myShow: function () {  
    document.write(this.prop1)  
  },  
}
```

## Виклик функції з наперед заданим this. *call*

Під час виклику функції можна задати контекст **this**

Загальна форма	<code>функція . <u>call</u> ( новий_контекст , аргумент_1, аргумент_2, ... )</code>
Приклад	<pre>let obj1 = {   prop1: 11,   prop2: 22,   showProp1: function () {     document.write(this.prop1)   },   getSum: function (val1, val2) {     return this.prop1 + val1 + val2   }, }  let obj2 = {   prop1: 21,   prop3: 23, }  obj1. showProp1 . <u>call</u>(obj2) // 21  let s = obj1. getSum . <u>call</u>(obj2, 10, 7) // 38 document.write(s)</pre>



## Виклик функції з наперед заданим this. *apply*

Під час виклику функції можна задати контекст **this**

Загальна форма	<code>функція . <u>apply</u> ( новий_контекст , список_аргументів )</code>
Приклад	<pre>let obj1 = {   prop1: 11,   prop2: 22,   showProp1: function () {     document.write(this.prop1)   },   getSum: function (val1, val2) {     return this.prop1 + val1 + val2   }, }  let obj2 = {   prop1: 21,   prop3: 23, }  obj1. showProp1 . <u>apply</u> (obj2) // 21  let s = obj1. getSum . <u>apply</u> (obj2, [ 10, 7 ]) // 38 document.write(s)</pre>

Приклад. Дано два об'єкта. Обидва містять масив цілих чисел. При цьому у одному з них є функція знаходження суми, а в іншому – функція для знаходження добутку тих, які знаходяться між заданими мінімальним і максимальних значенням. Використати обидва методи стосовно обидвох об'єктів.

Приклад. Дано функцію без формальних параметрів. Знайти суму усіх аргументів, які буде передано у функцію, використавши `reduce` стосовно `arguments`

**Фіксування контекста. `bind`**

*Для функції можна зафіксувати контекст, і тоді передача функції будь-куди не призведе до втрати контекста*

Загальна форма	<code>функція . <b><u>bind</u></b> ( новий_контекст )</code>
Приклад	<pre>let obj1 = {   prop1: 11,   prop2: 22,   showProp1: function () {     document.write(this.prop1)   },   getSum: function (val1, val2) {     return this.prop1 + val1 + val2   }, }  let func = obj1. showProp1 . <b><u>bind</u></b> (obj1) // прив'язуємо до obj1 func = func() // 11</pre>

Дано об'єкт, що містить масив чисел. Кожну секунду виводити випадкове число з цього масиву

**Фіксування контекста і частковим заданням значення параметрів функцій. `bind`**

Для функції можна зафіксувати контекст, і тоді передача функції будь-куди не призведе до втрати контекста. Також можна наперед визначити деякі параметри функцій

Загальна форма	<code>функція . <u>bind</u> ( новий_контекст , список_аргументів )</code>
Приклад	<pre>let obj1 = {   prop1: 11,   prop2: 22,   showProp1: function () {     document.write(this.prop1)   },   getSum: function (val1, val2) {     return this.prop1 + val1 + val2   }, }  let func = obj1. <u>getSum</u>. <u>bind</u> (obj1, 100) // прив'язуємо до obj1, val1=100  func = func(51) // getSum(100, 51) func = func(200) // getSum(100, 200)</pre>

## Класи

У нових версіях додано можливість описувати класи, що спрощують опис і створення об'єктів

Загальна форма <i>конструктора-функції</i>	Приклад опису класу
<pre>function <u>функція-конструктор</u> (...параметри...){     //----- Опис полів (індивідуальні дані об'єктів) -----     this.властивість1 = значення1;     this.властивість2 = значення2;      . . . }</pre>	<pre>class <u>Назва класу</u> {      constructor(форм.параметри){         //--- Опис полів (індивідуальні дані об'єктів)---         this.властивість1 = значення1;         this.властивість2 = значення2;          . . .     }  }</pre>
<pre>//----- Опис методів (спільні дані для усіх об'єктів) ----- <u>конструктор</u>.<u>prototype</u>.функція-метод_1 = function (форм.парам.) {     . . . }  <u>конструктор</u>.<u>prototype</u>.функція-метод_2 = function (форм.парам.) {     . . . }  . . . . .</pre>	<pre>//-- Опис методів (спільні дані для усіх об'єктів) -- функція-метод_1 (форм.парам.) {     . . . } функція-метод_2 (форм.парам.) {     . . . }  . . . }</pre>
<pre>//---- створення об'єкта ---- об'єкт = new <u>функція-конструктор</u> (... параметри ...)</pre>	<pre>//---- створення об'єкта ---- об'єкт = new <u>Назва класу</u> (... параметри ...)</pre>

Приклад. Описати клас «Діапазон чисел» (властивості : мін./макс. значення, методи: `isInRange` - визначення належності числа діапазону, `getRandomValueFromRange` - отримання випадкового числа з діапазону, `toString` - отримання рядкового представлення об'єкта )

Опис з використанням <i>класу</i>	Опис з використанням <i>функції-конструктора</i>
<pre>class <b>Range</b> {   constructor(defaultMin, defaultMax) {     //----- Опис полів     this.min = defaultMin     this.max = defaultMax   }    //----- Опис методів   <b>isInRange</b>(value) {     return value &gt;= this.min &amp;&amp; value &lt;= this.max   }    <b>getRandomValueFromRange</b>() {     return (       this.min + Math.floor(Math.random() *         (this.max - this.min + 1))     )   }    <b>toString</b>() {     return ` [\${this.min} ; \${this.max} ]`   } }</pre>	<pre>function <b>Range</b>(defaultMin, defaultMax) {   //----- Опис полів   this.min = defaultMin   this.max = defaultMax }  //----- Опис методів <b>Range</b>.prototype.<b>isInRange</b> = function (value) {   return value &gt;= this.min &amp;&amp; value &lt;= this.max }  <b>Range</b>.prototype.<b>getRandomValueFromRange</b> = function () {   return this.min + Math.floor(Math.random() *     (this.max - this.min + 1)) }  <b>Range</b>.prototype.<b>toString</b> = function () {   return ` [\${this.min} ; \${this.max} ]` }</pre>
<pre>//----- Створення об'єкта ---- let r1 = new <b>Range</b>(1, 12) document.write(r1.getRandomValueFromRange())</pre>	<pre>//----- Створення об'єкта ---- let r1 = new <b>Range</b>(1, 12) document.write(r1.getRandomValueFromRange())</pre>

Приклад. Розробити клас «Передбачувач». Дозволяє кожні вказані кількість секунд отримувати передбачення

Властивості :

- масив можливоих передбачень,
- інтервал між передбаченнями

Методи:

- вибір випадкового передбачення
- метод *run*, що ініціює запуск таймера і генерування передбачень