

FRONTEND

Лекція 21. Об'єкти

Об'єкти

Об'єкт — це набір властивостей, і кожна властивість складається з імені та значення, асоційованого з цим ім'ям. Всі інші типи даних у **JavaScript** можуть зберігати лише одне значення, тому вони називаються – примітивними. **Об'єкти** використовуються для зберігання колекцій різних значень (або більш складних структур даних).

Оголошення об'єктів — для створення об'єктів потрібно вказати фігурні дужки **{ }** та записати його властивості у форматі **ключ: значення**. Якщо властивостей декілька, вони відокремлюються друг від друга комою.

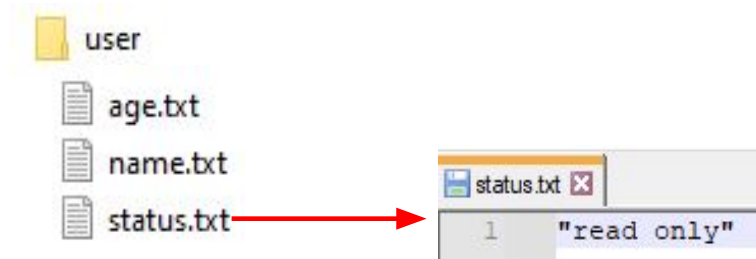


Об'єкти, приклад

Приклад запису об'єкту

```
let user = {  
  name: 'Mike',  
  age: 18,  
  status: 'read only',  
}
```

Об'єкти можна уявити у вигляді структури папок. Папки це – властивості, файли у папках це – ключі, а інформація у файлах це – значення ключа



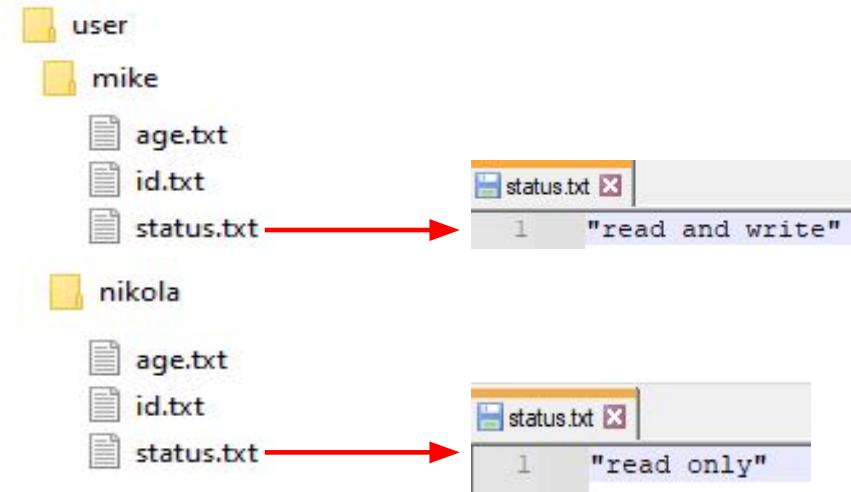
Об'єкти, приклад

Значення також можуть
бути об'єктами

```
let users = {  
  mike: {  
    id: 1,  
    age: 18,  
    status: 'read and write',  
  },  
  nikola: {  
    id: 2,  
    age: 25,  
    status: 'read only',  
  }  
}
```

Фігурні дужки
створюють ще один

Такий запис можна умовно порівняти
з структурою папок таким чином:



Об'єкти — доступ до значень

Для того щоб отримати значення з об'єкту ми вказуємо назву змінної яка зберігає посилання на об'єкт та через крапку `.` пишемо

```
const user = {  
  name: 'Mike',  
  age: 18,  
  status: 'read only',  
}  
console.log(user.name); // Mike
```

Якщо в об'єкті в якості значень знаходяться інші об'єкти доступ до таких значення вказується через додаткову крапку `.`

```
const users = {  
  mike: {  
    id: 1,  
    age: 18,  
    status: 'read and write',  
  },  
  nikola: {  
    id: 2,  
    age: 25,  
    status: 'read only',  
  }  
}  
console.log(users.mike.age); // 18
```



Об'єкти — перезапис та створення значень

Для того що б перезаписати значення об'єкту, потрібно привласнити йому

```
const user = {  
  name: 'Mike',  
  age: 18,  
  status: 'read only',  
}  
user.status = 'read and write';  
console.log(user.status); // read and write
```

Для створення нового ключа для вже існуючого об'єкту потрібно вказати назву змінної яка зберігає посилання на об'єкт та через крапку . написати новий ключ та привласнити йому

```
const user = {  
  name: 'Mike',  
  age: 18,  
  status: 'read only',  
}  
user.id = 12345;  
console.log(user.id); // 12345
```

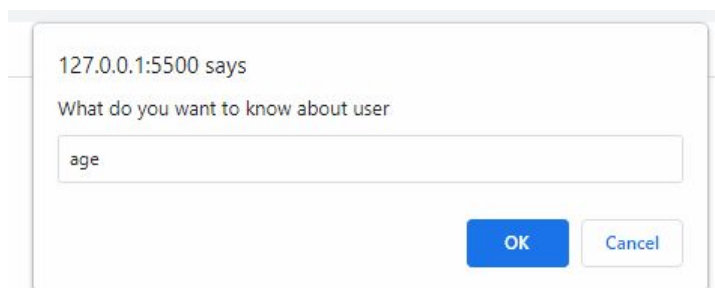


Об'єкти — квадратні дужки []

Якщо властивість об'єкту зберігається у змінній ви можете підставити цю змінну у об'єкт використовуючи квадратні

[]

```
let user = {  
  name: 'Mike',  
  age: 18,  
  status: 'read only',  
}  
let key = prompt('What do you want to know about user', 'age');  
alert(user[key]); // 18
```



127.0.0.1:5500 says

What do you want to know about user

OK Cancel

Об'єкти — квадратні дужки []

Якщо ключ об'єкту складається з декількох слів розкормлених пробілами, запис такого ключа здійснюється у лапках ' ', а доступ до таких ключів здійснюється через квадратні дужки з лапками

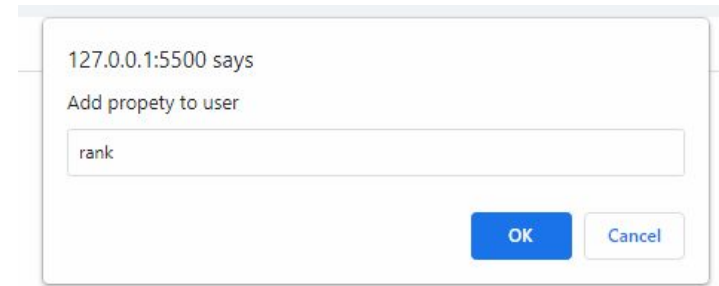
' '

```
let user = {  
  name: 'Mike',  
  age: 18,  
  status: 'read only',  
  'login name': 'MK_18',  
}  
console.log(user['login name']) //MK_18
```


Об'єкти — квадратні дужки []

Якщо ключ об'єкту об'єкту зберігається у змінній ви можете підставити цю змінну використовуючи квадратні дужки []

```
let additionalProperty = prompt('Add property to user', 'rank');
let user = {
  name: 'Mike',
  age: 18,
  status: 'read only',
  'login name': 'MK_18',
  [additionalProperty]: '1',
}
alert(user.rank);//1
```



Об'єкти — копіювання та посилання

Коли ми створюємо об'єкт та присвоюємо його значення змінній, то в змінній зберігається не сам об'єкт, а посилання на нього. Якщо ми привласнимо значення цієї змінної в іншу змінну то ми копіюємо посилання на

```
const pet = {  
  type: 'cat',  
  color: 'ginger',  
  size: 'large'  
}  
const anotherPet = pet;  
anotherPet.type = 'dog';  
console.log(pet.type); // dog  
console.log(anotherPet.type); // dog
```

Змінні `pet` та `anotherPet` містять посилання на один і той самий об'єкт, тому зміна у ключа `type` у `anotherPet` призведе до зміни об'єкту на який посилаються обидві ці змінні.

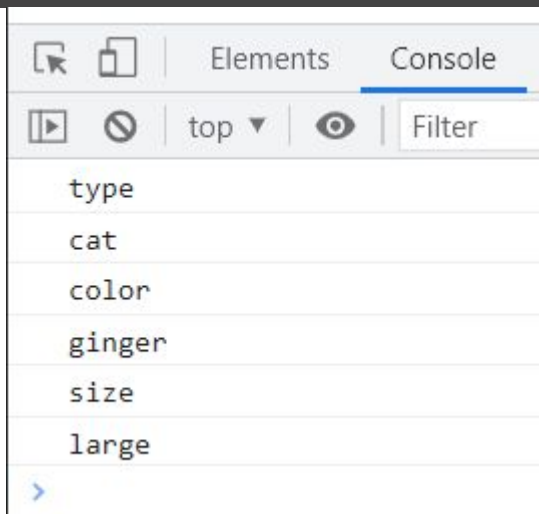
```
console.log(anotherPet === pet); // true
```



Об'єкти — цикл for in

цикл for in — використовується для перебору всіх властивостей об'єкта.

```
const pet = {  
  type: 'cat',  
  color: 'ginger',  
  size: 'large'  
}  
for (const key in pet){  
  console.log(key); // type, color, size  
  console.log(pet[key]); // cat, ginger, large  
}
```



Створюється змінна **key** яка на кожній ітерації циклу повертає значення ключа

pet

Об'єкти — копіювання

Ми можемо скористатися циклом **for in** для копіювання

```
let pet = {  
  type: 'cat',  
  color: 'ginger',  
  size: 'large'  
}  
let anotherPet = {};  
for(let key in pet){  
  anotherPet[key] = pet[key];  
}  
anotherPet.type = 'dog';  
console.log(anotherPet.type); //dog  
console.log(pet.type); //cat  
console.log(pet === anotherPet); //false
```

Створимо новий пустий об'єкт

Створимо цикл який на кожній ітерації буде повертати назви ключів об'єкта **pet**, та запишемо ці значення у об'єкт **anotherPet**

Отримаємо два різні об'єкти з однаковими значеннями

Об'єкти — методи

Методи об'єкта — це функції які записані у його значення ключа. За допомогою методів ми можемо робити різні корисні дії з об'єктом.

```
let salaries = {  
  fronted: 12000,  
  backend: 10000,  
  designer: 8000,  
  dayPay: function(){  
    alert('We must pay salary on Tuesday!');  
  },  
}  
salaries.dayPay(); // We must pay salary on Tuesday!
```

В якості значення ключа **dayPay** написана функція.
Це називається **методом** об'єкту

Ми можемо викликати цю функцію таким чином



Об'єкти — методи, скорочений запис

Ми можемо скоротити запис, та записати метод таким

```
let salaries = {  
  fronted: 12000,  
  backend: 10000,  
  designer: 8000,  
  dayPay() {  
    alert('We must pay salary on Tuesday!');  
  },  
}  
salaries.dayPay(); // We must pay salary on Tuesday!
```



Об'єкти — this

Ключове слово **this** — це об'єкт перед точкою **.** дозволяє методу об'єкта отримати інформацію яка зберігається в іншому ключі цього об'єкта.

```
const salaries = {
  fronted: 12000,
  backend: 10000,
  designer: 8000,
  dayPay() {
    alert('We must pay salary on Tuesday!');
  },
  total() {
    let sum = this.fronted + this.backend + this.designer;
    console.log(sum);
  }
}
salaries.total();// 30000
```



Об'єкти — this

Ми можемо переписати цю функцію звернувшись безпосередньо до об'єкта, але такий код буде ненадійним

```
total() {  
    let sum = salaries.fronted + salaries.backend + salaries.designer;  
    console.log(sum);  
}  
let expensese = salaries;  
salaries = {};  
expensese.total() //NaN
```

Якщо змінній буде присвоєне нове значення або об'єкт, усі скопійовані посилання в методі будуть звертатися до **salaries.fronted** і якщо ця зміна буде в подальшому перезаписана у всіх інших змінних які

Якщо ми використаємо **this** такого не станеться, оскільки **this** буде повертати

```
total() {  
    let sum = this.fronted + this.backend + this.designer;  
    console.log(sum);  
}  
let expensese = salaries;  
salaries = {};  
expensese.total() //30000
```


Об'єкти — конструктори, оператор new

Функції конструктори — призначені для створення однотипних об'єктів. Функції конструкторі завжди починаються з великої літери (технічно ви можете написати функцію і з маленької літери, але є така домовленість що допомагає в процесі підтримки коду зрозуміти що це саме **функція конструктор** яка буде створювати об'єкт).

Оператор new — виклик функції конструктора повинен здійснюватися з цим оператором.



Об'єкти — конструктори, оператор new, приклад

Створимо функцію конструктор **Pet** яка буде створювати різні види домашніх тварин

Ключ об'єкту

Передаємо у функцію конструктор аргументи які будуть записані у значення ключів об'єкту.

Функція конструктор побудує нові об'єкти.

```
function Pet(type, name, color){  
  this.type = type;  
  this.name = name;  
  this.color = color;  
}  
let dog = new Pet('dog', 'Cooper', 'black');  
let cat = new Pet('cat', 'Ashley', 'white');  
let parrot = new Pet('parrot', 'Polly', 'green');  
console.log(dog.name); //Cooper  
console.log(cat.type); //cat  
console.log(parrot.color); //green
```

Об'єкти — оператор new, принцип дії

Виклик функції разом з оператором **new** змушує функцію зробити три наступні дії:

1) Створити новий об'єкт та привласнити його **this** (не явно)

2) Виконати код записаний у тілі функції

3) **this**

```
function Pet(type, name, color){  
  // this = {};  
  this.type = type;  
  this.name = name;  
  this.color = color;  
  // return this = {};  
}
```

Виконується не явно якщо перед функцією вказано оператор new

Перетворення об'єктів у примітиви

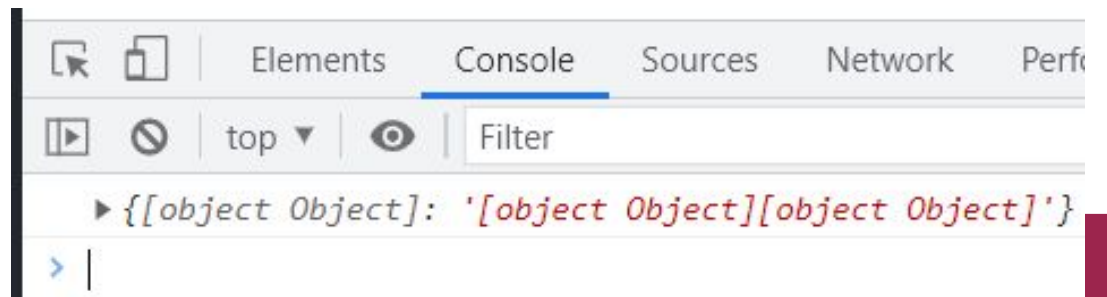
Створимо функцію конструктор яка буде

```
function User(name, rating){  
  this.name = name;  
  this.rating = rating;  
}  
let user_1 = new User('Mike', 25);  
let user_2 = new User('Nikola', 36);
```

Припустимо що **user_1** створює групу **group** та додає до неї **user_2** яка буде складатися з цих двох користувачів.

```
let group = {};  
group[user_1] = user_1 + user_2;  
console.log(group);
```

Отримаємо результат:

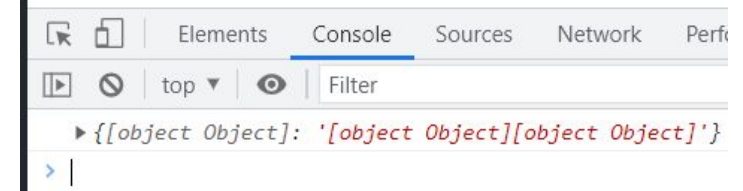


Перетворення об'єктів у примітиви, пояснення коду

- 1) Створюємо об'єкт `group`
- 2) пробуємо привласнити об'єкту `group` в якості ключа об'єкт `user_1`.
- 3) В якості значень встановлюємо суму з двох об'єктів `user_1` та `user_2`

```
let group = {};  
group[user_1] = user_1 + user_2;  
console.log(group);
```

В об'єктах може зберігатися багато різних типів даних, коли движок `JavaScript` не розуміє які самі (примітивні) дані необхідно повернути з об'єкту він повертає значення `[object Object]`



Але ми можемо налаштувати повернення цих значень за допомогою спеціального вбудованого методу `[Symbol.toPrimitive]`

Перетворення об'єктів у примітиви — [Symbol.toPrimitive]

[Symbol.toPrimitive] — це спеціальний вбудований метод, який використовується для приведення об'єкту до примітиву (примітивного значення). Він приймає один параметр **hint** який може повертати три значення: **string**, **number** та **default**.

string — означає що примітивне значення об'єкту буде повертатися типом даних **string** (рядок).

number — означає що примітивне значення об'єкту буде повертатися типом даних **number** (число).

default — означає що тип примітивного значення об'єкту до кінця не визначено, за замовчуванням повертає тип даних **number** але можна налаштувати інше значення.



[Symbol.toPrimitive], приклад

```
function User(name, rating) {
  this.name = name;
  this.rating = rating;
  this[Symbol.toPrimitive] = function (hint){
    console.log(hint); // 2*default, string, 2*number, number, number, string
    switch(hint){
      case 'string':
        return this.name
      case 'number':
        return this.rating
      case 'default':
        return this.name + '_' + this.rating + ' '
    }
  }
}

let user_1 = new User('Mike', 25);
let user_2 = new User('Nikola', 36);
let group = {};
group[user_1] = user_1 + user_2;
console.log(group); // {Mike: 'Mike_25 Nikola_36 '}
console.log(user_2 - user_1); // 11
console.log(+user_2 + +user_1); // 61
console.log(String(user_2)); // Nikola
```

[Symbol.toPrimitive], пояснення коду

Пояснення для рядка: `group[user_1] = user_1 + user_2;`

Шаг 1: виконується права частина від знаку `=`. Об'єкт `user_1` має додатися до об'єкту `user_2`. В цих об'єктах є вбудований метод `[Symbol.toPrimitive]` оскільки знак `+` може означати як конкатенацію (об'єднання рядків) так і додавання чисел, то параметр `hint` приймає значення `default`.

Шаг 2: конструкція `switch` перебирає можливі значення `hint` та повертає умову зазначену у рядку `case 'default':`

`return this.name + '_' + this.rating + ''`

Тобто отримаємо такий вираз:

`'Mike_25 ' + 'Nikola_36 '`

що дорівнює:

`'Mike_25 Nikola_36 '`

```
function User(name, rating) {
  this.name = name;
  this.rating = rating;
  this[Symbol.toPrimitive] = function (hint){
    switch(hint){
      case 'string':
        return this.name
      case 'number':
        return this.rating
      case 'default':
        return this.name + '_' + this.rating + ''
    }
  }
}

let user_1 = new User('Mike', 25);
let user_2 = new User('Nikola', 36);
let group = {};
group[user_1] = user_1 + user_2;
console.log(group); // {Mike: 'Mike_25 Nikola_36 '}
```


[Symbol.toPrimitive], пояснення коду

Шаг 3: виконується запис об'єкту `user_1` в якості ключа `group[user_1]`. При такому записі `hint` повертає значення `string` а отже виконується код:

`return this.name`
що дорівнює:
`Mike`

Отже отримаємо ключ `Mike` та об'єкт
`{Mike: 'Mike_25 Nikola_36 '}`

```
function User(name, rating) {  
  this.name = name;  
  this.rating = rating;  
  this[Symbol.toPrimitive] = function (hint){  
    switch(hint){  
      case 'string':  
        return this.name  
      case 'number':  
        return this.rating  
      case 'default':  
        return this.name + '_' + this.rating + ' '  
    }  
  }  
}  
  
let user_1 = new User('Mike', 25);  
let user_2 = new User('Nikola', 36);  
let group = {};  
group[user_1] = user_1 + user_2;  
console.log(group); // {Mike: 'Mike_25 Nikola_36 '}
```

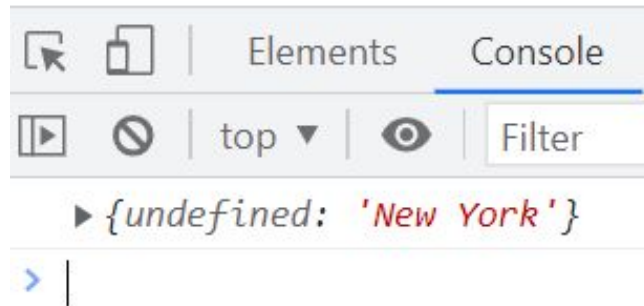


Функції стрілки та this

УВАГА!

Функції стрілки не мають this!

Створимо об'єкт який в методі `[Symbol.toPrimitive]` використовує функцію



Оскільки функції стрілки не мають **this**, результатом виразу `city[user]` буде **undefined**

```
let user = {
  nik: "Mike_25",
  age: 25,
  [Symbol.toPrimitive]: (hint) => {
    switch(hint){
      case 'string':
        return this.nik
      case 'number':
        return this.age
    }
  }
}
let city = {};
city[user] = "New York";
console.log(city); // undefined
```

Функції стрілки та this

Все почне працювати якщо ми оголосимо звичайну функцію

```
let user = {  
  nik: "Mike_25",  
  age: 25,  
  [Symbol.toPrimitive]: function(hint) {  
    switch(hint){  
      case 'string':  
        return this.nik  
      case 'number':  
        return this.age  
    }  
  }  
}  
  
let city = {};  
city[user] = "New York";  
console.log(city); // undefined
```

