Obiekty w JavaScript

Obiekt

W JavaScript prawie wszystko jest obiektem, a my w naszych skryptach nie raz i nie dwa odwołujemy się do obiektów. Dla przykładu `window`, `console`, `Math` to typowe obiekty, które posiadają swoje metody i właściwości.

```
const tab = [1,2,3];
tab.length //widzisz kropkę? Przez nią odwołujemy się do metod lub właściwości obiektu - w tym przypadku obiektu typu Array

const text = "Ala ma kota";
console.log(  text.charAt(0) ); //text - typ prosty został na chwilę zamieniony na obiekt (po czym od razu po wykonaniu akcji wrócił do typu prostego)

window.innerHeight //właściwość innerHeight obiektu window

Math.floor(21.3); //metoda obiektu Math

document.cookie //właściwość cookie obiektu document

button.innerHTML //właściwość innerHTML obiektu button
```

Obiekty można traktować jako pewne zamknięte pudełka. Weźmy na przykład psa. Taki obiekt ma brązową sierść, cztery łapy, brązowe oczy. To jego właściwości. Potrafi też biegać, jeść, szczekać. To jego czynności - w naszym języku zwane funkcjami czy metodami. A wszystko to upakowane w jeden obiekt - zwany psem. Dzięki temu jeżeli chcemy zrobić coś z psem - np. go umyć czy zabrać do parku, mamy dostęp do tych wszystkich rzeczy pod postacią pojedynczego psiego obiektu.

Przykład bliższy naszej Frontendowej działce to chociażby pierwszy lepszy element na stronie - np. pojedynczy button. Ma właściwości określające jego zawartość (np. tekst), szerokość, wysokość, pozycję na stronie. Ale też ma metody, czyli funkcje które może wykonać - np. focus czy blur, które go zaznaczą lub odznaczą gdy zostaną odpalane.

Tworzenie pojedynczego obiektu

Poza dostępem do gotowych obiektów, możemy tworzyć też swoje własne.

Obiekty możemy podzielić na pojedyncze instancje (pojedyncze egzemplarze), oraz grupy obiektów o podobnych właściwościach (np. tablice, linki, buttony, stringi itp).

Na początku zajmijmy się pojedynczymi instancjami.

Obiekty w Javascript możemy tworzyć na kilka sposobów. Jednym z najczęściej używanych jest użycie skróconego zapisu:

```
const dog = {
    name: "Szama",
    speed: 1000,
    showText: function() {
        return "Lubię walczyć ze złem";
    }
}
```

W dzisiejszych czasach metody dla obiektów możemy też definiować w skrócony sposób z pominięciem słowa `function`:

```
//zamiast
const dog = {
    name: "Szama",
    speed: 1000,
    showText: function() {
        return "Lubię walczyć ze złem";
    }
}

//możemy
const dog = {
    name: "Szama",
    speed: 1000,
    showText() {
        return "Lubię walczyć ze złem";
    }
}
```

Jeżeli pod dane właściwości podstawiamy wartości ze zmiennych, których nazwy są takie same jak danych kluczy, możemy taki zapis skrócić:

```
const name = "Szama";
const speed = 1000;

//zamiast
const dog = {
    name: name,
    speed: speed,
    showText() {
        return "Lubię walczyć ze złem";
    }
}

//możemy napisać
const dog = {
    name,
    speed,
    showText() {
        return "Lubię walczyć ze złem";
    }
}
```

Zastosowanie tej małej cechy może mocno uprościć nasz przyszły kod:

```
const tab = [];
const name = "Szama";
const speed = 1000;

//zamiast
const ob = {
    name: name,
    speed: speed
}
tab.push(ob);

//mogę
tab.push({
    name: name,
    speed: speed
});

//lub jeszcze lepiej
tab.push({name, speed});
```

Odwoływanie się do właściwości

Do właściwości i metod obiektu możemy się odwoływać na dwa sposoby:

```
const dog = {
    name: "Szama",
    speed: 1000,
    showText() {
        return "Lubię walczyć ze złem";
    }
}

//poprzez kropkę po której podajemy nazwę klucza
dog.name; //"Szama"
dog.speed; //1000
dog.showText(); //"Lubię walczyć ze złem"

//lub używając kwadratowych nawiasów
dog["name"]; //"Szama"
dog["speed"]; //1000
dog["showText"](); //"Lubię walczyć ze złem"
```

Najczęściej będziemy korzystać z notacji z kropką. Nie oznacza to jednak, że drugi sposób nie ma zastosowania.

Nazwy kluczy dla obiektów w większości przypadków przypominać będą nazwy zwykłych zmiennych. Jeżeli tak jest, możemy odwoływać się do nich notacją z kropką.

Nie zawsze jednak użycie zapisu z kropką będzie możliwe. Klucze obiektów przyjmują postać dowolnego tekstu lub Symbolu. Oznacza to, że mogą zdarzyć się sytuacje, gdzie klucz będzie miał wartość, do której nie będziemy się mogli dostać za pomocą notacji z kropką:

```
const calendar = {
    "2022-11-11" : "Narodowe Święto Niepodległości"
}

calendar.2022-11-11 //oczywisty błąd bo wykonujemy jakieś równanie
```

W takich przypadkach zmuszeni jesteśmy do użycia notacji z kwadratowymi nawiasami:

```
calendar["2022-11-11"] //"Narodowe Święto Niepodległości"
```

Z podobną sytuacją już się zresztą spotykaliśmy. W przypadku odwoływania się do kluczy tablic (które też są obiektami) używaliśmy notacji z kwadratowymi nawiasami:

```
const tab = ["kot", "pies", "chomik ninja"];

tab.0 //błąd
tab[0] //"kot"
tab["0"] //"kot"

tab.2 //błąd
tab[2] //"chomik ninja"
tab["2"] //"chomik ninja"

tab.2.length //błędne odwołanie
tab["2"].length //12
tab["2"]["length"] //12

tab.length //3 - bo length to właściwość o nazwie "length"
tab["length"] //3
```

Poza powyższymi dwoma sposobami, do wyciągania danych z obiektów możemy użyć tak zwanego przypisania destrukturyzacji. Zajmiemy się nim w w kolejnym rozdziale.

```
const obj = {
    name: "Marcin",
    surname: "Kowalski",
    age: 10
}

//zamiast
const name = obj.name;
const surname = obj["surname"];
const age = obj.age;

//mogę
const {name, surname, age} = obj;
```

```
const tab = ["Ala", "Ola", "Ela"];

//zamiast
const name1 = tab[0];
const name2 = tab[1];

//mogę
const [name1, name2] = tab;
```

Powyżej tworzone obiekty to proste byty.

Obiekty zazwyczaj są bardziej rozbudowane, bo każda właściwość może wskazywać na kolejne obiekty, a te z kolei na kolejne:

```
const person = {
    name: "Marcin",

    pet: {
        name: "Szama",
        color: "brown",
        speed: 1000,

        collar: {
            color: "red",
            length: "25cm"
        },

        favoriteFood: ["mięso", "mięso", "mięso"]
    }
}

person.name //"Marcin"
person.pet.name //"Szama"
person.pet.collar.color //"red"
person.pet.favoriteFood[1] //"mięso"
```

W ramach testu wpisz w konsoli debuggera nazwy popularnych obiektów: `window`, `console`, `Date`, po każdym naciśnij enter i zbadaj każdy z nich rozwijając jego zawartość. Spróbuj odwołać się do wybranych kluczy.

Dodawanie nowych właściwości

Do stworzonego wcześniej obiektu możemy dodawać metody i właściwości nie tylko w jego ciele podczas tworzenia, ale także poza nim:

```
const dog = {
    name: "Szama",
    speed: 1000,
    showText() {
        return "Lubię walczyć ze złem";
    }
}

dog.type = "pies";
dog.legs = 4;
dog.eat = function() {
    return "Jem dobre rzeczy";
}

dog.eat();
dog.showText();
```

Do takiego dodawania funkcjonalności możemy też oczywiście skorzystać z notacji z kwadratowymi nawiasami:

```
dog["type"] = "pies";
dog["legs"] = 4;
dog["eat"] = function() {
    return "Jem dobre rzeczy";
}
```

Podobnej notacji możemy też użyć wewnątrz obiektu, co przydaje się szczególnie gdy chcemy stworzyć nowy klucz o wartości jakiejś zmiennej:

```
const keyName = "show";

const ob = {
    name: "Karol",
    surname: "Nowak",
    [keyName]: "Lubię jeść jabłka"
}

console.log(ob.show);
```

Usuwanie właściwości i metod

Aby usunąć właściwość lub metodę obiektu, skorzystamy ze słowa **delete:**

```
const car = {
    brand: "Mercedes",
    color: "czerwony",
    showText() {
        console.log(`${this.brand} koloru ${this.color}`);
    }
}

console.log(car.color); //czerwony
delete car.color;
console.log(car.color); //undefined
```

W JavaScript nie musimy usuwać całych obiektów. Jeżeli na dany obiekt nie będzie wskazywała żadna zmienna (czyli dany obiekt nie będzie w żaden sposób dostępny), jego automatycznym usunięciem zajmie się tak zwany Garbage Collection. Jeżeli chciałbyś usunąć cały obiekt, nie musisz tego robić. Wystarczy, że zmienną wskazującą na dany obiekt ustawisz na null, co zerwie referencje. W większości przypadków jednak nie musisz się tym przejmować, bo Javascript dba o to za ciebie.

Pętla po właściwościach obiektu

Jeżeli chcemy zrobić pętlę po kluczach obiektu, najczęściej stosowaną będzie pętla **for in**. Jej zastosowanie ma następującą postać:

```
const car = {
    brand: "Mercedes",
    color: "czerwony",
    showText() { ... }
}

for (const key in car) {
    console.log(key); //brand, color, showText
}
```

Robiąc iteracje pod zmienną key podstawiane są kolejne klucze. Aby pobrać ich wartość zastosujemy notację z kwadratowymi nawiasami:

```
for (const key in car) {
    console.log("Klucz: ", key);
    console.log("Wartość: ", car[key]);
}
```

Pętlę `for in` można traktować jako taki klasyk istniejący w naszym języku od zawsze. Gdy przyjrzymy się jej działaniu, okaże się, że nie jest ona idealna, ponieważ może niechcący zahaczyć też o klucze prototypu danego obiektu. Wszysko zależy od danej sytuacji i tego jak dany obiekt został stworzony. Możesz to sprawdzić na tej stronie.

Żeby mieć pewność, że iterujemy tylko po kluczach danego obiektu możemy wykonać odpowiedni test za pomocą funkcji **hasOwnProperty()**:

```
const car = {
    brand: "Mercedes",
    color: "czerwony",
    showText() {
        ...
    }
}

for (const key in car) {
    if (car.hasOwnProperty(key)) {
        console.log(key);
    }
}
```

W dzisiejszych czasach możemy też zastosować dodatkowe rodzaje pętli:

```
for (let key of Object.keys(car)) {
    ...
}

for (let val of Object.values(car)) {
    ...
}

for (let [key, val] of Object.entries(car)) {
    ...
}
```

Pętle takie biorą pod uwagę tylko klucze danego obiektu.

Więcej na ten temat dowiesz się w rozdziale o iteratorach.

this

Tworząc nasze obiekty bardzo często będziemy chcieli w kodzie jego metody odwoływać się do właściwości lub metod danego obiektu. Do takiego odwoływania się używamy słowa kluczowego **this**, które wskazuje na obiekt, do którego należy dana metoda:

```
const car = {
    name: "Mercedes",
    color: "czerwony",

    drive() {
        console.log(this); //car
        console.log(`${this.name} sobie jedzie`);
    },

    showText() {
        console.log(`${this.name} koloru ${this.color}`);
    }
}

car.drive(); //"Mercedes sobie jedzie"
car.showText(); //"Mercedes koloru czerwony"
```

W powyższym kodzie obie metody `drive()` i `showText()` należą do obiektu `car`, stąd słowo *this* w ich wnętrzu wskazuje na ten obiekt.

To, że są to metody danego obiektu możemy oczywiście rozpoznać po tym, że należą do jego składni (jak powyżej), ale też po tym jak je wywołujemy np. `car.drive()`. Jeżeli obiekt nie ma danej metody, nie możemy dla niego jej odpalić (a przynajmniej w taki sposób).
I tak `window.drive()` czy `console.drive()` rzuci nam błąd. Wiem, że dla wielu z was wydaje się to oczywiste. W kolejnych rozdziałach zobaczysz jednak, że `this` może nas czasami zaskoczyć. Na chwilę obecną zapamiętaj tylko, że `this` wskazuje na obiekt, do którego dana metoda należy.