Temat 10

Temat: Obiekty1

Poznaliśmy już podstawowe typy w języku JavaScript: String (łańcuchy znaków), Number(liczby), Boolean (wartości logiczne), Null (typ pusty), Undefined (typ nieokreślony). Poniższa tabela stanowi krótkie podsumowanie tego, co robią.

Podstawowe typy JavaScript:

Typ	**Co robi**
String	Podstawowa struktura do pracy z tekstem
Number	Pozwala pracować z liczbami
Boolean	Powstaje, gdy używamy true i false
Null	Reprezentuje cyfrowy odpowiednik niczego
Undefined	Choć trochę podobny do null, jest zwracany, gdy wartość powinna istnieć, ale nie istnieje... tak jak w sytuacji, w której deklarujemy zmienną, ale niczego do niej nie przypisujemy
Object	**Działa jak opakowanie dla innych typów, w tym innych obiektów**

Można te typy w prosty sposób pogrupować. Podobnie jak w przypadku prostych i złożonych składników pizzy, typy również mogą być proste lub złożone. W terminologii języka JavaScript typy proste i złożone są formalnie znane odpowiednio jako **prymitywy** i **obiekty**. Mówiąc inaczej, typy w języku JavaScript są znane jako **typy prymitywne** (lub po prostu **prymitywy**) oraz **typy obiektowe** (lub po prostu **obiekty**).

Typami **prymitywnymi** są String, Number, Boolean, Null i Undefined. Żadna wartość, która kryje się pod tym „parasolem", nie może być dalej podzielona.

Wszystko, co tworzysz jako typ **Object** lub czego używasz jako tego typu, potencjalnie składa się z innych typów pierwotnych lub innych obiektów. Typy Object mogą być puste.

Czym są obiekty?

Koncepcja obiektów w języku programowania JavaScript ma odpowiedniki w świecie rzeczywistym. W rzeczywistym świecie jesteśmy otoczeni obiektami. Komputer jest obiektem. Książka na półce jest obiektem. Ziemniak jest obiektem. Twój budzik jest obiektem. Rzeczywisty świat jest więc wypełniony obiektami, takimi jak psy czy samochody. Większość z nich składa się z różnych części. Każdy pies ma ogon, głowę i cztery łapy; a samochody mają drzwi, koła, światła, klaksony i tak dalej. Obiekty mogą także wykonywać pewne czynności — samochód może transportować pasażerów,

a pies — szczekać. Co więcej, poszczególne części obiektów mogą coś robić; na przykład ogon psa może machać, a klakson może trąbić.

Także świat języka JavaScript jest wypełniony obiektami, takimi jak okno przeglądarki, dokument, łańcuchy znaków, liczby czy daty. Podobnie jak obiekty w rzeczywistym świecie, także obiekty języka JavaScript składają się z wielu różnych części. W terminologii programistycznej te części obiektów to ***właściwości*****.** Z kolei akcje, które obiekty mogą wykonywać, są nazywane ***metodami*****;** to funkcje (podobne do wbudowanej funkcji alert()) charakterystyczne dla konkretnego obiektu.

Predefiniowane obiekty JS

W języku JavaScript mamy oprócz poznanych wcześniej typów wbudowanych garść predefiniowanych obiektów, których możemy użyć od ręki. Te obiekty pozwalają nam pracować ze wszystkim, począwszy od kolekcji danych, przez daty, aż po tekst i liczby. Tabela poniższa przedstawia niektóre z tych obiektów wraz z krótką notatką na temat tego, co one robią.

Niektóre predefiniowane obiekty JavaScript

Typ	**Co robi**
Array	Pomaga przechowywać i pobierać kolekcję danych oraz manipulować nią
Boolean	Działa jak obiekt opakowujący dla prymitywu boolean;
Date	Pozwala łatwiej reprezentować daty i pracować z nimi
Function	Pozwala m.in. wywołać jakiś kod
Math	zapewnia lepszą pracę z liczbami
Number	Działa jak obiekt opakowujący dla prymitywu number
RegExp	Zapewnia funkcjonalność dla dopasowywania wzorców w tekście
String	Działa jak obiekt opakowujący dla prymitywu string

Obiektów wbudowanych używamy w nieco inny sposób niż prymitywów. Każdy obiekt posiada również własne metody. Opisem każdego obiektu i wyjaśnieniem, jak należy go używać będziemy się zajmować w kolejnych lekcjach. W tej lekcji zajmiemy się tworzeniem własnych obiektów.

Tworzenie obiektów za pomocą literałów

Najprostszym sposobem utworzenia obiektu jest użycie literału obiektowego, zgodnego z następującym schematem:

```
{
  "nazwa_właściwości_1":wartość_właściwości_1,
  "nazwa_właściwości_2":wartość_właściwości_2,
  /* tutaj definicje dalszych właściwości */
  "nazwa_właściwości_N":wartość_właściwości_N
}
```

Nazwa powinna być ujmowana w znaki cudzysłowu prostego, choć można je pominąć, jeżeli nie zawiera spacji ani innych znaków niestandardowych. **Właściwościami obiektu mogą być również inne obiekty** definiowane w nawiasie klamrowym:

```
{
  nazwa_obiektu_składowego:{
    właściwości obiektu składowego
  }
}
```

Mogą to być również tablice definiowane za pomocą nawiasu kwadratowego, ogólnie:

```
{
  tablica:[
   wartość1,
   wartość2,
   /* kolejne wartości */
   wartośćN
  ]
}
```

Obiekt może być przypisany zmiennej:

```
const zmienna = {/* definicja obiektu */}
```

Kiedy obiekt jest gotowy i istnieje w kodzie skryptu, można się **odwoływać do zawartych w nim danych (właściwości).** Używa się w tym celu **operatora . (znak kropki)** lub składni z nawiasem kwadratowym. Wartość danej właściwości można więc odczytać za pomocą jednej z poniższych konstrukcji:

```
const zmienna = nazwa_obiektu.nazwa_właściwości;

const zmienna = nazwa_obiektu["nazwa_właściwości"];
```

Przykład 1. Przygotuj stronę html wykorzystującą poniższy kod.

```
<body>
    <p id="output"></p>
    <script>
        const out = document.getElementById("output");
        //definicja obiektu
        const pies = {
          imie: 'Irys',
          rasa: 'kundelek',
          waga: 10  // zwróć uwagę na brak przecinka po ostatniej właściwości
        }
        console.log(pies); // wyświetlenie obiektu w konsoli
        document.write(pies); // wyświetla tylko informacje o stworzeniu obiektu
        document.write("<br>imię psa: "+pies.imie); // odwołanie do właściwości z kropką
        document.write("<br>rasa psa: "+pies.rasa);
        document.write("<br>waga psa: "+pies["waga"]); // odwołanie do właściwości z []
        // odwołanie do właściwości z kropką w nowszych rozwiązaniach:
        out.innerHTML = `To odwołanie do właściwości obiektu w akapicie: ${pies.imie}`;
    </script>
</body>
```

Obiekt i tablica jako właściwość obiektu

Umieśćmy w obiekcie dane dotyczące dwóch dowolnych książek: tytuł, autor, wydawnictwo. Informacje o wydawnictwach powinny zawierać nazwy oraz adresy. W przypadku pierwszej książki dane o wydawnictwie zapiszemy jako obiekt, a w przypadku drugiej — jako tablicę.

Przykład 2. Przygotuj stronę html wykorzystującą poniższy kod.

```
<body>
    <p id="output"></p>
    <script>
        const out = document.getElementById("output");
        //definicja obiektu przy pomocy literału
        const book1 = {
          "tytul" : "Pan Lodowego Ogrodu",
          "autor" : "Jarosław Grzędowicz",
          // właściwość jako obiekt:
          "wydawnictwo" : {
            "nazwa" : "Fabryka Słów ",
            "adres" : "ul. Irysowa 25 A, 20-834 Lublin"
          }
        };
        const book2 = {
          "tytul" : "PHP i MySQL. Dla każdego",
          "autor" : "Marcin Lis",
          // właściwość jako tablica:
          "wydawnictwo" : ["Helion S.A.", "Kościuszki 1c, 44-100 Gliwice"]
        };
        // dane pierwszej książki
        document.write("Pierwsza książka:<br>");
        document.write("Tytuł = " + book1.tytul + "<br>");
        document.write("Autor = " + book1.autor + "<br>");
        document.write("Nazwa wydawnictwa = " + book1.wydawnictwo.nazwa + "<br>");
        document.write("Adres wydawnictwa = " + book1.wydawnictwo.adres + "<br>");
        console.log(book1);
        // dane drugiej książki
        out.innerHTML = `Druga książka:<br>
                         Tytuł = ${book2.tytul}<br>
                         Autor = ${book2.autor}<br>
                         Nazwa wydawnictwa = ${book2.wydawnictwo[0]}<br>
                         Adres wydawnictwa = ${book2.wydawnictwo[1]}`;
        console.log(book2);
    </script>
```

Konstruktor typu obiektowego

Obiekt może zostać utworzony za pomocą słowa kluczowego **new**, po którym musi wystąpić wywołanie funkcji konstruującej obiekt, czyli konstruktora (ang.*constructor*). Schemat takiego wywołania jest następujący:

```
const zmienna = new Konstruktor();
```

W najprostszym przypadku będzie to wyglądało tak:

```
const obiekt = new Object();
```

Powstanie w ten sposób nowy pusty obiekt, który zostanie przypisany zmiennej obiekt. Jest to więc odpowiednik konstrukcji:

```
const obiekt = {};
```

UWAGA: w kolejnych lekcjach będziemy raczej wykorzystywali literał, zgodnie z tendencją wielu programistów.

Przykład 3. Przygotuj stronę html wykorzystującą poniższy kod.

```
<script>
    //definicja obiektu przy pomocy literału
    const myObj = {
            Car:"honda",
            Year:2017,
            Model:"civic",
            cost:30000};

    console.log(myObj);
    console.log(myObj["Car"]);

    //definicja obiektu przy pomocy konstruktora
    const myObj1 = new Object({
        Car: "Toyota",
        Year: 2012,
        Model: "Corolla",
        cost:60000
        })
    console.log(myObj1);
    console.log(myObj1["Car"]);

    //definicja obiektu przy pomocy konstruktora
    const myObj2 = new Object(); // pusty obiekt
    // przypisanie właściwości do pustego obiektu
        myObj2.Car = "Ford";
        myObj2.Year = 2010;
        myObj2.Model = "Mustang";
        myObj2.cost = 50000;

    console.log(myObj2);
    console.log(myObj2["Car"]);
</script>
```

Modyfikowanie zawartości obiektu

Przykład 4. Przygotuj stronę html wykorzystującą poniższy kod.

```
<script>
    //definicja obiektu przy pomocy literału
    const myObj = {
            Car:"honda",
            Year:2017,
            Model:"civic",
            cost:30000};

    //zmiana właściwości w obiekcie
    myObj.Year=2000;
    console.log(myObj["Year"]);

    //dodanie właściwości do obiektu
    myObj.Color="green";
    console.log(myObj);
</script>
```