Frontend development

Wykład 5

Podstawy JavaScript

Mateusz Pawełkiewicz

1. Wprowadzenie do języka JavaScript

1.1 Czym jest JavaScript

JavaScript to wysokopoziomowy, interpretowany język programowania, który jest jednym z fundamentów współczesnego internetu. Razem z HTML i CSS tworzy trójcę technologii frontendowych, umożliwiając tworzenie interaktywnych i dynamicznych stron oraz aplikacji webowych.

- **Interaktywność**: JavaScript pozwala na reagowanie na działania użytkownika, takie jak kliknięcia, ruchy myszą czy wprowadzanie danych.
- **Dynamika**: Umożliwia dynamiczną manipulację elementami strony bez konieczności jej przeładowywania.
- **Wszechstronność**: Stosowany nie tylko w przeglądarkach, ale również po stronie serwera (Node.js), w aplikacjach mobilnych i desktopowych.

1.2 Krótka historia JavaScript

- **1995**: Brendan Eich tworzy JavaScript w ciągu 10 dni dla przeglądarki Netscape Navigator.
- **1996**: Microsoft wprowadza własną implementację o nazwie JScript.
- **1997**: ECMA International standaryzuje język jako ECMAScript.
- **2009**: Powstaje Node.js, umożliwiając uruchamianie JavaScript po stronie serwera.
- **2015**: Wydanie ECMAScript 6 (ES6), wprowadzające znaczące ulepszenia i nowe funkcje.

1.3 Zastosowania JavaScript

- **Front-end Development**: Tworzenie interaktywnych interfejsów użytkownika.
- **Back-end Development**: Dzięki Node.js, JavaScript jest używany do tworzenia serwerów i API.
- **Aplikacje mobilne**: Frameworki takie jak React Native pozwalają na tworzenie aplikacji mobilnych.
- **Aplikacje desktopowe**: Electron umożliwia tworzenie aplikacji desktopowych za pomocą JavaScript.

---

2. Zmienne, typy danych i operatory

2.1 Deklaracja zmiennych

W JavaScript zmienne można deklarować za pomocą słów kluczowych var, let lub const.

2.1.1 var

- **Zakres**: Funkcyjny lub globalny.
- **Hoisting**: Zmienna deklarowana za pomocą var jest przenoszona na początek zakresu.

```
var x = 10;
```

2.1.2 let

- **Zakres**: Blokowy (wewnątrz { }).
- **Hoisting**: Zmienna jest hoistowana, ale nie jest inicjalizowana (dostępna dopiero po deklaracji).

```
let y = 20;
```

2.1.3 const

- **Zakres**: Blokowy.
- **Stała**: Wartość nie może być ponownie przypisana.
- **Hoisting**: Podobnie jak let.

```
const z = 30;
```

2.2 Typy danych

JavaScript jest językiem dynamicznie typowanym, co oznacza, że typy danych są określane w czasie wykonywania.

2.2.1 Typy prymitywne

- **String**: Ciąg znaków.

  ```
  let tekst = "Hello, World!";
  ```

- **Number**: Liczby całkowite i zmiennoprzecinkowe.

  ```
  let liczba = 42;
  ```

- **Boolean**: Wartości logiczne true lub false.

  ```
  let czyPrawda = true;
  ```

- **Undefined**: Wartość zmiennej niezainicjalizowanej.

  ```
  let niezdefiniowana;
  ```

- **Null**: Wartość oznaczająca brak wartości.

  ```
  let pustaWartosc = null;
  ```

- **Symbol** (ES6): Unikalna i niezmienna wartość.

  ```
  let symbol = Symbol("opis");
  ```

- **BigInt** (ES2020): Liczby całkowite o dowolnej precyzji.

  ```
  let duzaLiczba = 123456789012345678900n;
  ```

2.2.2 Typy złożone (Obiekty)

- **Object**: Kolekcja par klucz-wartość.

```
let osoba = {
  imie: "Jan",
  nazwisko: "Kowalski",
  wiek: 30
};
```

- **Array**: Lista uporządkowanych elementów.

```
let liczby = [1, 2, 3, 4, 5];
```

- **Function**: Blok kodu, który można wywołać.

```
function przywitaj() {
  console.log("Cześć!");
}
```

2.3 Operatory

2.3.1 Operatory arytmetyczne

- **Dodawanie**: +
- **Odejmowanie**: -
- **Mnożenie**: *
- **Dzielenie**: /
- **Modulo**: % (reszta z dzielenia)
- **Potęgowanie**: **

Przykład:

```
let suma = 5 + 3; // 8
let iloraz = 10 / 2; // 5
```

2.3.2 Operatory przypisania

- **Proste przypisanie**: =
- **Przypisanie z operatorem**: +=, -=, *=, /=, %=, **=

Przykład:

```
let liczba = 10;
liczba += 5; // liczba = 15
```

2.3.3 Operatory porównania

- **Równość wartości**: ==
- **Ścisła równość (wartość i typ)**: ===
- **Nierówność wartości**: !=

- **Ścisła nierówność**: !==
- **Większe niż**: >
- **Mniejsze niż**: <
- **Większe lub równe**: >=
- **Mniejsze lub równe**: <=

Przykład:

```
let a = 5;
let b = "5";

console.log(a == b);  // true
console.log(a === b); // false
```

2.3.4 Operatory logiczne

- **AND (i)**: &&
- **OR (lub)**: ||
- **NOT (nie)**: !

Przykład:

```
let x = true;
let y = false;

console.log(x && y); // false
console.log(x || y); // true
console.log(!x);     // false
```

2.3.5 Operatory inkrementacji i dekrementacji

- **Inkrementacja**: ++
- **Dekrementacja**: --

Przykład:

```
let licznik = 0;
licznik++; // licznik = 1
```

---

3. Struktury kontrolne

3.1 Instrukcje warunkowe

3.1.1 if...else

Pozwala na wykonanie kodu w zależności od spełnienia warunku.

```
let wiek = 18;

if (wiek >= 18) {
```

```
  console.log("Jesteś pełnoletni.");
} else {
  console.log("Jesteś niepełnoletni.");
}
```

3.1.2 else if

Pozwala na sprawdzenie wielu warunków.

```
let temperatura = 25;

if (temperatura > 30) {
  console.log("Gorąco.");
} else if (temperatura > 20) {
  console.log("Ciepło.");
} else {
  console.log("Chłodno.");
}
```

3.1.3 Operator warunkowy (ternary operator)

Skrócona forma instrukcji if...else.

```
let wynik = (warunek) ? wartość_jeśli_prawda : wartość_jeśli_fałsz;

let wiek = 18;
let status = (wiek >= 18) ? "dorosły" : "nieletni";
```

3.1.4 switch

Używany do porównania jednej wartości z wieloma przypadkami.

```
let dzien = 3;
let nazwaDnia;

switch (dzien) {
  case 1:
    nazwaDnia = "Poniedziałek";
    break;
  case 2:
    nazwaDnia = "Wtorek";
    break;
  case 3:
    nazwaDnia = "Środa";
    break;
  default:
    nazwaDnia = "Nieznany dzień";
}

console.log(nazwaDnia); // Środa
```

3.2 Pętle

3.2.1 for

Wykonuje blok kodu określoną liczbę razy.

```
for (let i = 0; i < 5; i++) {
  console.log("Iteracja nr " + i);
}
```

3.2.2 while

Wykonuje blok kodu, dopóki warunek jest prawdziwy.

```
let i = 0;
while (i < 5) {
  console.log("Iteracja nr " + i);
  i++;
}
```

3.2.3 do...while

Wykonuje blok kodu co najmniej raz, a następnie kontynuuje, dopóki warunek jest prawdziwy.

```
let i = 0;
do {
  console.log("Iteracja nr " + i);
  i++;
} while (i < 5);
```

3.2.4 Pętle po kolekcjach

- **for...in**: Iteruje po kluczach obiektu.

  ```
  let osoba = { imie: "Jan", wiek: 30 };

  for (let klucz in osoba) {
    console.log(klucz + ": " + osoba[klucz]);
  }
  ```

- **for...of**: Iteruje po wartościach iterowalnych (tablice, stringi).

  ```
  let liczby = [1, 2, 3, 4, 5];

  for (let liczba of liczby) {
    console.log(liczba);
  }
  ```

4. Funkcje i zakresy zmiennych

4.1 Funkcje

Funkcje to bloki kodu wykonujące określone zadania, które można wywołać w dowolnym miejscu programu.

4.1.1 Deklaracja funkcji

- **Funkcja nazwane**:

```
function dodaj(a, b) {
  return a + b;
}

let wynik = dodaj(5, 3); // 8
```

- **Funkcje anonimowe (wyrażenia funkcyjne)**:

```
let odejmij = function(a, b) {
  return a - b;
};

let wynik = odejmij(10, 4); // 6
```

- **Funkcje strzałkowe (ES6)**:

```
let pomnoz = (a, b) => {
  return a * b;
};

// Skrócona forma dla funkcji jedno-liniowych
let podziel = (a, b) => a / b;

let wynik = pomnoz(2, 3); // 6
```

4.1.2 Parametry domyślne (ES6)

Pozwala na ustawienie domyślnych wartości dla parametrów funkcji.

```
function powitanie(imie = "Gość") {
  console.log("Witaj, " + imie + "!");
}

powitanie(); // Witaj, Gość!
```

4.1.3 Operator rest i spread (ES6)

- **Operator rest (…)**: Pozwala na przekazanie dowolnej liczby argumentów jako tablicę.

```
function suma(...liczby) {
  return liczby.reduce((a, b) => a + b, 0);
```

```
}

let wynik = suma(1, 2, 3, 4); // 10
```

- **Operator spread (…)**: Rozwija iterowalny obiekt (np. tablicę) w miejscu, w którym oczekiwane są zero lub więcej argumentów.

```
let liczby = [1, 2, 3];
let liczbyRozwiniete = [...liczby, 4, 5]; // [1, 2, 3, 4, 5]
```

4.2 Zakresy zmiennych

Zakres zmiennej określa, gdzie zmienna jest dostępna w kodzie.

4.2.1 Zakres globalny

Zmienna jest dostępna w całym skrypcie.

```
var globalna = "Jestem globalna";

function pokaz() {
  console.log(globalna);
}

pokaz(); // Jestem globalna
```

4.2.2 Zakres funkcji

Zmienna jest dostępna tylko wewnątrz funkcji.

```
function test() {
  var lokalna = "Jestem lokalna";
  console.log(lokalna);
}

test(); // Jestem lokalna
console.log(lokalna); // Błąd: lokalna is not defined
```

4.2.3 Zakres blokowy (ES6)

Zmienna zadeklarowana za pomocą `let` lub `const` jest dostępna tylko wewnątrz bloku `{ }`.

```
if (true) {
  let blokowa = "Jestem w bloku";
  console.log(blokowa); // Jestem w bloku
}

console.log(blokowa); // Błąd: blokowa is not defined
```

4.2.4 Hoisting

Mechanizm przenoszenia deklaracji zmiennych i funkcji na początek zakresu.

- **Zmienne zadeklarowane za pomocą `var`** są hoistowane, ale ich wartość jest `undefined` przed przypisaniem.

  ```
  console.log(x); // undefined
  var x = 5;
  ```

- **Funkcje deklarowane za pomocą `function`** są hoistowane z całą definicją.

  ```
  pokaz();

  function pokaz() {
    console.log("Funkcja hoistowana");
  }
  ```

---

5. Manipulacja Document Object Model (DOM)

5.1 Czym jest DOM

DOM (Document Object Model) to interfejs programistyczny, który reprezentuje dokument HTML jako strukturę drzewiastą, gdzie każdy węzeł jest obiektem reprezentującym część dokumentu (elementy, atrybuty, tekst).

5.2 Wybieranie elementów

5.2.1 `getElementById`

Wybiera element o określonym `id`.

```
let element = document.getElementById("naglowek");
```

5.2.2 `getElementsByClassName`

Wybiera wszystkie elementy o określonej klasie. Zwraca kolekcję (live HTMLCollection).

```
let elementy = document.getElementsByClassName("klasa");
```

5.2.3 `getElementsByTagName`

Wybiera wszystkie elementy o określonym tagu.

```
let paragrafy = document.getElementsByTagName("p");
```

5.2.4 `querySelector`

Wybiera pierwszy element pasujący do selektora CSS.

```
let pierwszyLink = document.querySelector("a");
```

5.2.5 querySelectorAll

Wybiera wszystkie elementy pasujące do selektora CSS. Zwraca statyczną kolekcję (NodeList).

```
let linki = document.querySelectorAll("a");
```

5.3 Modyfikowanie elementów

5.3.1 Modyfikacja treści

- **textContent**: Ustawia lub pobiera tekst wewnątrz elementu.

  ```
  element.textContent = "Nowy tekst";
  ```

- **innerHTML**: Ustawia lub pobiera kod HTML wewnątrz elementu.

  ```
  element.innerHTML = "<strong>Nowy tekst</strong>";
  ```

5.3.2 Modyfikacja atrybutów

- **setAttribute**: Ustawia wartość atrybutu.

  ```
  element.setAttribute("class", "nowa-klasa");
  ```

- **getAttribute**: Pobiera wartość atrybutu.

  ```
  let klasa = element.getAttribute("class");
  ```

5.3.3 Modyfikacja stylów

- **Bezpośrednia modyfikacja stylu**:

  ```
  element.style.color = "red";
  element.style.fontSize = "20px";
  ```

- **Klasy CSS**:
  - **Dodawanie klasy**:

    ```
    element.classList.add("nowa-klasa");
    ```

  - **Usuwanie klasy**:

    ```
    element.classList.remove("stara-klasa");
    ```

  - **Przełączanie klasy**:

    ```
    element.classList.toggle("aktywny");
    ```

5.4 Tworzenie i usuwanie elementów

5.4.1 Tworzenie nowego elementu

```
let nowyElement = document.createElement("div");
nowyElement.textContent = "Jestem nowy!";
```

5.4.2 Dodawanie elementu do DOM

- **appendChild**: Dodaje element jako ostatnie dziecko.

  ```
  rodzic.appendChild(nowyElement);
  ```

- **insertBefore**: Wstawia element przed wskazanym elementem.

  ```
  rodzic.insertBefore(nowyElement, istniejącyElement);
  ```

5.4.3 Usuwanie elementu

- **removeChild**:

  ```
  rodzic.removeChild(elementDoUsuniecia);
  ```

- **remove** (nowoczesne przeglądarki):

  ```
  elementDoUsuniecia.remove();
  ```

---

6. Obsługa zdarzeń w przeglądarce

6.1 Dodawanie nasłuchiwaczy zdarzeń

6.1.1 Metoda addEventListener

```
element.addEventListener("typZdarzenia", funkcja);
```

Przykład:

```
let przycisk = document.getElementById("przycisk");

przycisk.addEventListener("click", function() {
  alert("Kliknięto przycisk!");
});
```

6.1.2 Usuwanie nasłuchiwaczy zdarzeń

```
element.removeEventListener("typZdarzenia", funkcja);
```

6.2 Typy zdarzeń

- **Zdarzenia myszy**: click, dblclick, mousedown, mouseup, mouseover, mouseout, mousemove
- **Zdarzenia klawiatury**: keydown, keyup, keypress
- **Zdarzenia formularzy**: submit, focus, blur, change, input
- **Zdarzenia okna**: load, resize, scroll, unload

6.3 Obiekt zdarzenia

Obiekt zdarzenia dostarcza informacji o zdarzeniu i jest automatycznie przekazywany do funkcji obsługującej zdarzenie.

```
element.addEventListener("click", function(event) {
  console.log(event.type); // "click"
  console.log(event.target); // Element, na którym wywołano zdarzenie
});
```

6.4 Propagacja zdarzeń

Zdarzenia w DOM propagują się przez fazy:

- **Faza przechwytywania**: Zdarzenie idzie od korzenia dokumentu do elementu docelowego.
- **Faza celu**: Zdarzenie dotarło do elementu docelowego.
- **Faza bąbelkowania**: Zdarzenie propaguje się z powrotem do korzenia.

6.4.1 Zatrzymywanie propagacji

- **event.stopPropagation()**: Zatrzymuje propagację zdarzenia.

  ```
  element.addEventListener("click", function(event) {
    event.stopPropagation();
  });
  ```

6.4.2 Zapobieganie domyślnym działaniom

- **event.preventDefault()**: Zapobiega domyślnej akcji przeglądarki.

  Przykład: Zapobieganie wysłaniu formularza.

  ```
  formularz.addEventListener("submit", function(event) {
    event.preventDefault();
    // Własna logika wysyłki formularza
  });
  ```