

JavaScript

– DOM

- dzień 2

v 1.6

PLAN

- Eventy w DOM-ie
- Poruszanie się po drzewie DOM
- Tworzenie elementów
- Inputy i formularze

Eventy w DOM-ie

Eventy i funkcja callback

Eventy

- Są to wydarzenia odbywające się na naszej stronie WWW. Dzięki JavaScript jesteśmy w stanie przejąć kontrolę nad eventem i odpowiednio reagować.
- Eventy dzielimy wedle rodzaju interakcji np. użycie myszki czy klawiatury, edycja formularza lub okna przeglądarki itp.
- W obiekcie **event** są zawarte informacje dotyczące danej akcji.

Callback

- Jest to to specjalna funkcja, którą podajemy do wywołania. Nie jest uruchamiana od razu, lecz po wystąpieniu jakiegoś zdarzenia.
- Każdy event w JavaScript jest tworzony za pomocą funkcji callback.

Dodawanie eventów do elementów

Eventy dodajemy przez użycie metody `addEventListener(eventName, callback)` na obiekcie elementu.

Kod HTML

```
<button id="counter">Click me!</button>
```

Kod JavaScript

```
var button = document.querySelector("button");  
var clickCount = 0;  
button.addEventListener("click", function (event) {  
    clickCount += 1;  
    console.log("Click number", clickCount);  
});
```

Dodawanie eventów do elementów

Eventy dodajemy przez użycie metody `addEventListener(eventName, callback)` na obiekcie elementu.

Kod HTML

```
<button id="counter">Click me!</button>
```

Kod JavaScript

```
var clickCount = 0;  
  
function clickCounter (event) {  
    clickCount += 1;  
    console.log('Click number', clickCount);  
}
```

Dodawanie eventów do elementów

Kod HTML

```
<button id="counter">Click me!</button>
```

Kod JavaScript

```
var button = document.querySelector("button");
var clickCount = 0;
var randomWords = ['Some', 'Random', 'Words'];

function clickCounter (event) {
  clickCount += 1;
  console.log('Click number', clickCount);
}
```

Kod JavaScript – ciąg dalszy

```
function randomWord (event) {
  var myWord = randomWords[Math.floor(Math.random()
    * randomWords.length)];

  console.log(myWord);
}

button.addEventListener('click', clickCounter);
button.addEventListener('click', randomWord);
```

Usuwanie eventów z elementów

- Możemy też usunąć event z elementu. robimy to za pomocą metody:
removeEventListener(event, callback).
- Nie da się usunąć eventów, które zostały dodane za pomocą funkcji anonimowych!

Kod HTML

```
<button id="counter">Click me!</button>
```

Kod JavaScript

```
var button = document.querySelector('button');
var clickCount = 0;

function clickCounter (event) {
  console.log('Click number', clickCount);

  clickCount += 1;
  if(clickCount >= 10) {
    event.target.removeEventListener
      ('click', clickCounter);
  }
}

button.addEventListener('click', clickCounter);
```


Lista eventów

mouse:

mousedown, mouseup, click, dblclick,
mousemove, mouseover, mouseout

key:

keydown, keypress, keyup

touch:

touchstart, touchmove, touchend, touchcancel

control:

resize, scroll, focus, blur, change, submit

no arguments:

load, unload, DOMContentLoaded

Pełna lista eventów:

https://en.wikipedia.org/wiki/DOM_events

DOMContentLoaded

- DOMContentLoaded jest specjalnym eventem, uruchamiającym się w momencie załadowania całej strony.
- Nasz cały kod JavaScript operujący na DOM powinien znajdować się w tym evencie. Inaczej nie mamy gwarancji, że element którego szukamy, został już stworzony!

- Jeżeli wykonujesz operacje na DOM, upewnij się, że cały dokument został uprzednio załadowany!

```
document.addEventListener("DOMContentLoaded", function () {  
    console.log("DOM fully loaded and parsed");  
});
```

this w eventach

- W każdym evencie mamy możliwość odwołania się do zmiennej **this**.
- Jest to specjalna zmienna reprezentująca element, na którym został wywołany event.
- Jest ona szczególnie przydatna, jeżeli taki sam event nastawiamy na wiele elementów.
- W przykładzie w jednym miejscu zakładamy event na wszystkie guziki.
- Event ten zmieni kolor tylko tego guzika, w który klikamy, nie wpływa na inne.

Kod HTML

```
<button class="btn">Click me!</button>
<button class="btn">Click me!</button>
<button class="btn">Click me!</button>
```

Kod JavaScript

```
var buttons = document.querySelectorAll(".btn");

for(var i=0; i < buttons.length; i++) {
    buttons[i].addEventListener("click", function(event) {
        this.style.backgroundColor = "red";
    });
}
```

Obiekt event

Event jest opisywany przez specjalny obiekt. Dzięki niemu możemy dowiedzieć się wielu przydatnych rzeczy na temat zdarzenia. Oto jego przykładowe właściwości:

- **event.currentTarget** – zwraca element, na którym wywołany został event,
- **event.target** – zwraca element, który spowodował wywołanie eventu, w przypadku propagacji jest to element potomka
- **event.timeStamp** – zwraca czas, w którym został wywołany event,
- **event.type** – zwraca typ eventu (jak string).

Obiekt Event ma jeszcze kilka przydatnych metod:

- **event.preventDefault()** – anuluj oryginalną akcję,
- **event.stopPropagation()** – anuluj wszystkie eventy tego samego typu z elementów nadrzędnych,
- **event.stopImmediatePropagation()** – anuluj wszystkie eventy tego samego typu przypięte do tego elementu oraz wszystkich elementów nadrzędnych.

Propagacja eventów

Event bubbling

W DOM mamy do czynienia z tak zwaną propagacją eventów. Polega ona na przekazywaniu eventu w górę drzewa DOM. Nazywa się to **event bubbling**.

Event capturing

Stare przeglądarki czasami implementowały **event capturing**, czyli przekazywanie eventów w dół drzewa. Jest to jednak metoda przestarzała.

Kod HTML

```
<div id="foo">  
  <button id="bar">Click me!</button>  
</div>
```

Kod JavaScript

```
document.querySelector('#foo').addEventListener  
    ('click', function () {  
    console.log('Event zarejestrowany, element #foo');  
});  
  
document.querySelector('#bar').addEventListener  
    ('click', function () {  
    console.log('Event zarejestrowany, element #bar');  
});
```

Document Object Model (DOM)

Kod HTML

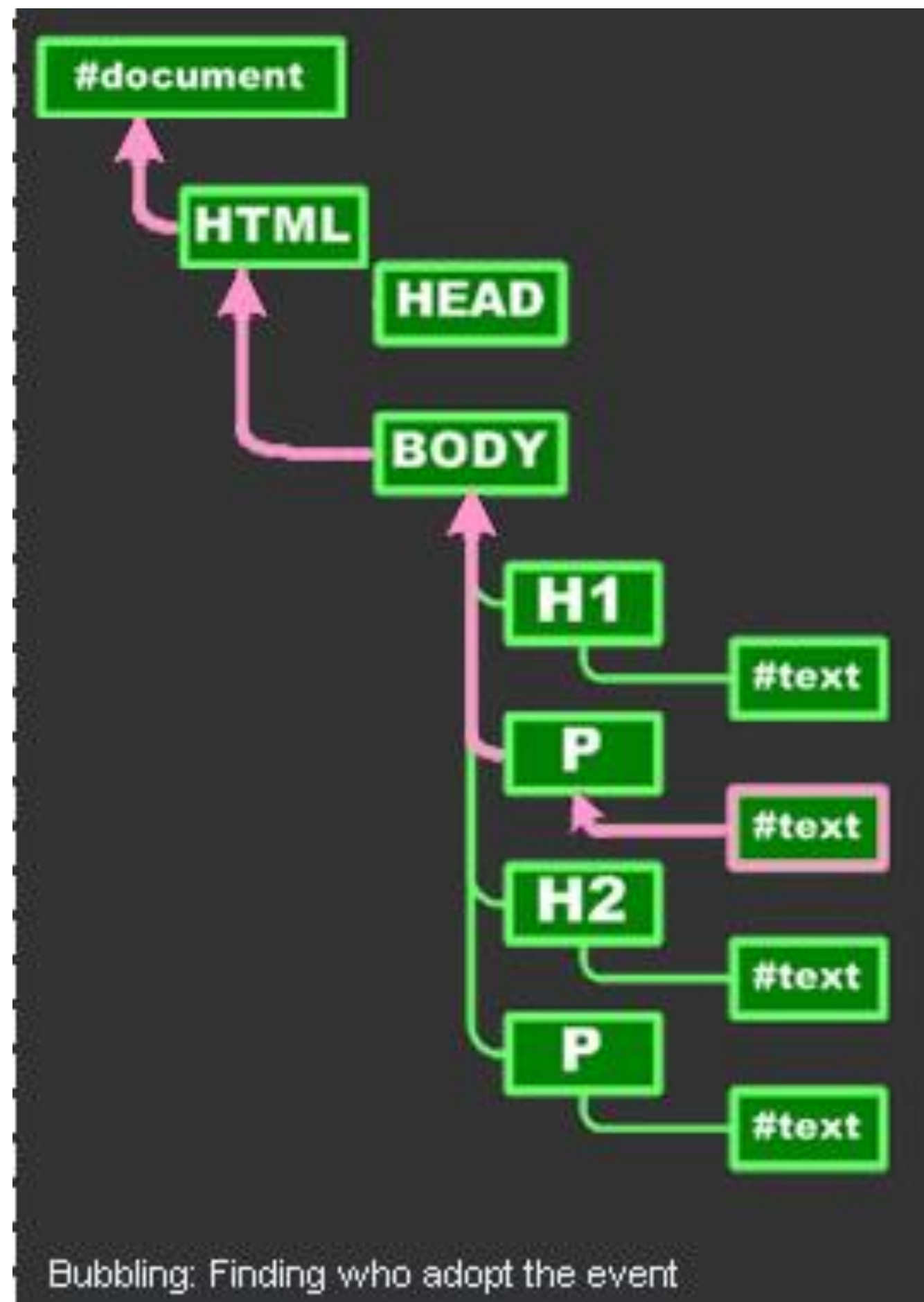
```
<div id="foo">  
  <button id="bar">Click me!</button>  
</div>
```

Kod JavaScript

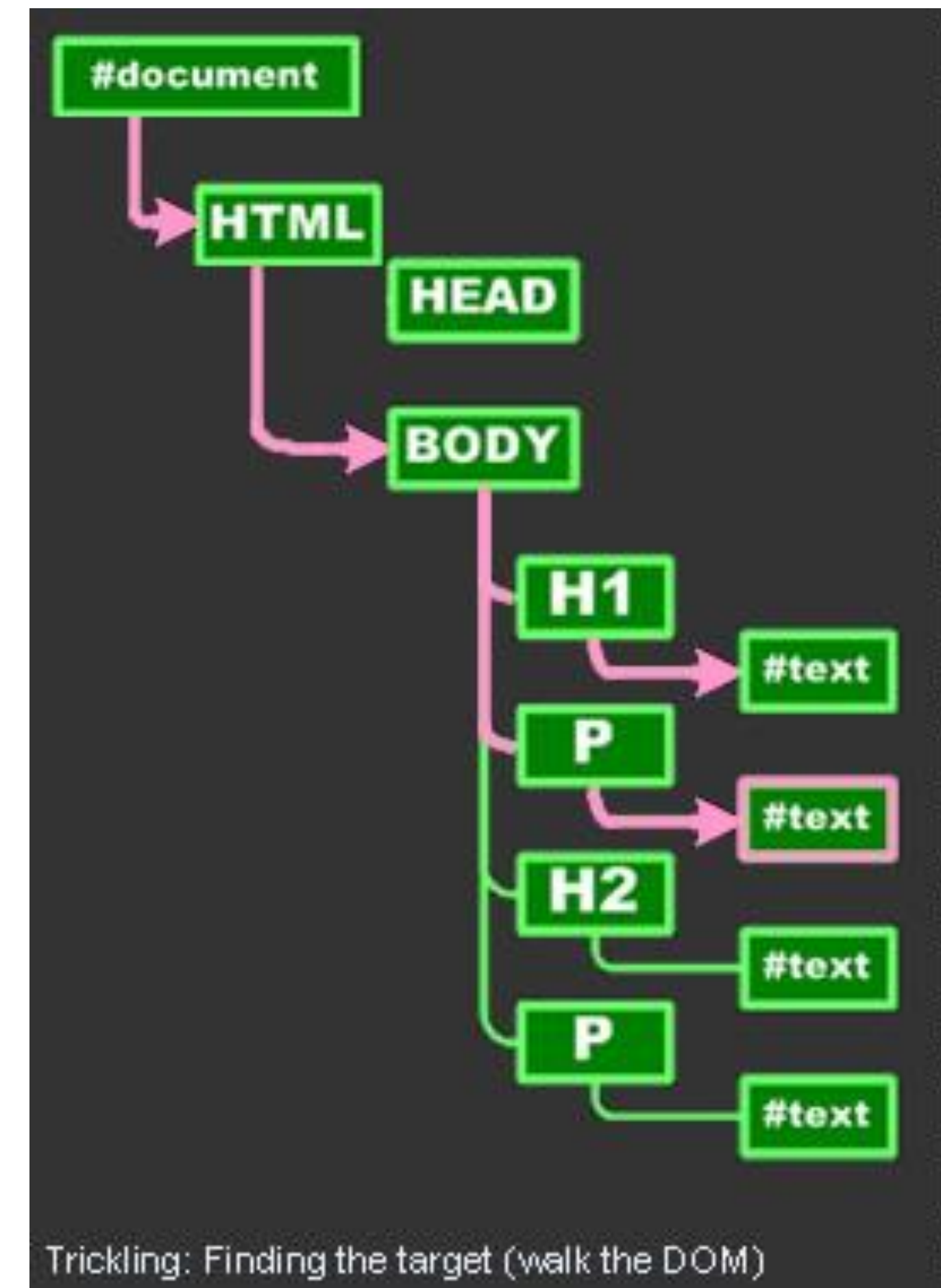
```
document.querySelector('#foo').addEventListener  
('click', function (e) {  
  console.log('Target:', e.target.id);  
  console.log('CurrentTarget:', e.currentTarget.id);  
});  
  
// "Target:" "bar"  
// "CurrentTarget" "foo"
```


Bubbling vs capturing

Event bubbling



Event capturing



MouseEvent

Jest to specjalny typ eventu tworzony podczas zdarzeń związanych z myszką. Rozszerza on podstawowy event o następujące atrybuty:

- **event.button** – zwraca przycisk myszki, który został naciśnięty,
- **event.clientX** – zwraca koordynat X (horyzontalny) myszki, relatywnie do górnego, lewego rogu strony,
- event.clientY** – zwraca koordynat Y (wertykalny) myszki relatywnie do górnego, lewego rogu strony,

- **event.screenX** – zwraca koordynat X (horyzontalny) myszki, relatywnie do górnego, lewego rogu okna,
- **event.screenY** – zwraca koordynat Y (wertykalny) myszki, relatywnie do górnego, lewego rogu okna.

KeyboardEvent

Jest to specjalny typ eventu tworzony podczas zdarzeń związanych z klawiaturą. Rozszerza on podstawowy event o następujące atrybuty:

- **event.altKey** – zwraca **true**, jeżeli **alt** był naciśnięty,
- **event.ctrlKey** – zwraca **true**, jeżeli **ctrl** był naciśnięty,

- **event.charCode** – zwraca znak opisujący klawisz, który wywołał event jako kod klawisza będący liczbą
- **event.key** – zwraca wartość klawisza, który wywołał event, np. "d", "8",")"
- **event.keyCode** – zwraca wartość Unicode klawisza, który wywołał event,
- **event.shiftKey** – zwraca **true**, jeżeli shift był naciśnięty.

Pełna lista typów eventów:

http://www.w3schools.com/jsref/dom_obj_event.asp

Zadania

Wykonaj zadania
z działu
2.DOM
z folderu
4.Eventy

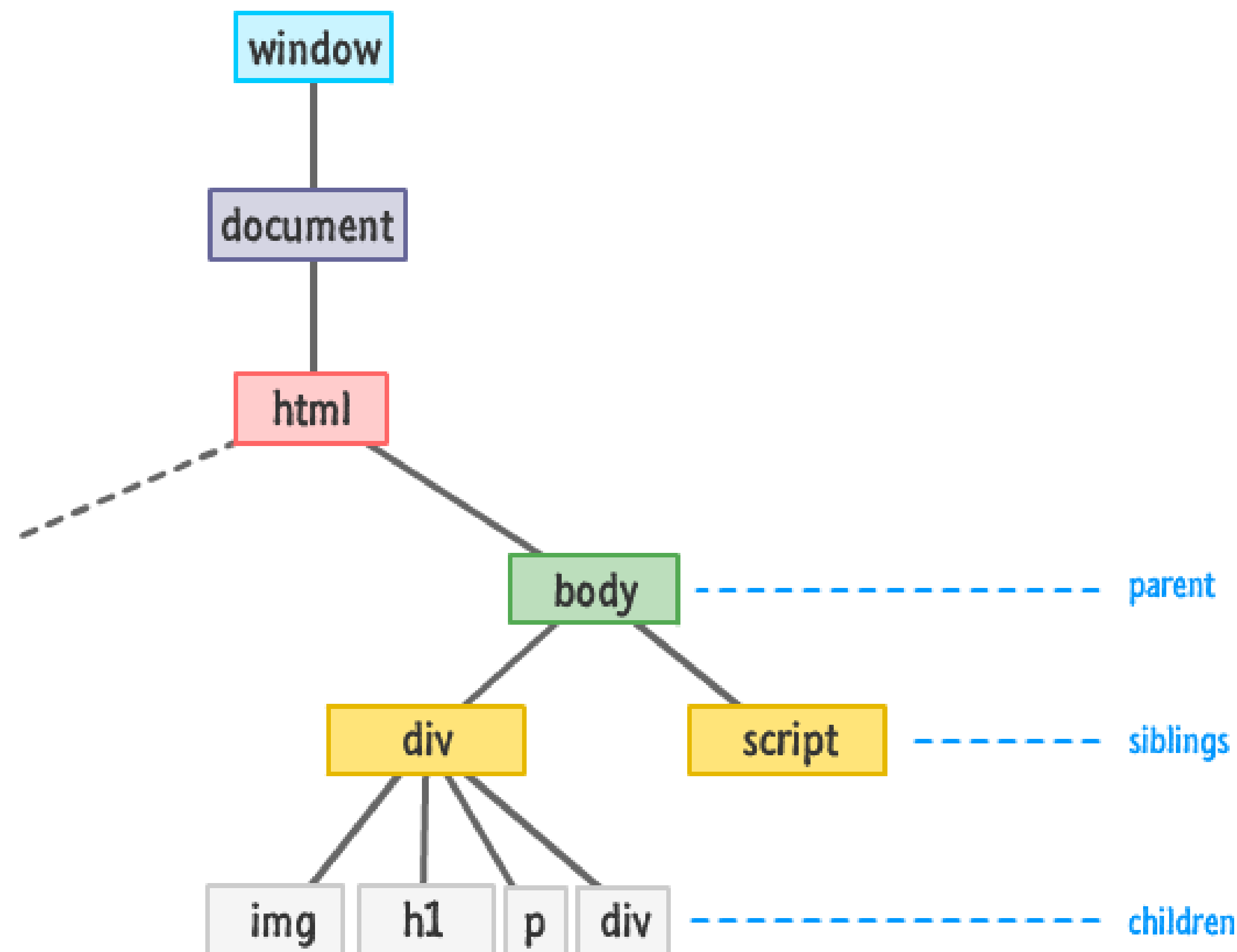
Poruszanie się po drzewie DOM

Poruszanie się po drzewie DOM

Dzięki odpowiednim metodom elementów możemy się swobodnie poruszać po całym dokumencie.

W drzewie rozróżniamy trzy ważne nazwy:

- rodzic (parent),
- brat (sibling),
- dziecko (child).



Poruszanie się w górę

- Poruszanie się w górę drzewa jest najłatwiejsze – istnieje tylko jedna ścieżka, którą możemy pójść, a wyznacza ją rodzic (parent) elementu.
- Żeby uzyskać element rodzica, należy użyć atrybutu:
parentElement

```
<div id="foo">
```

```
  <span id="bar">Bar</span>
```

```
</div>
```

```
var barElement = document.querySelector('#bar');
```

```
var barParent = barElement.parentElement;
```

Poruszanie się na boki

Mamy dwie możliwości poruszania się na boki w drzewie DOM:

- **el.nextElementSibling** – zwraca następny element mający tego samego rodzica,
- **el.previousElementSibling** – zwraca poprzedni element mający tego samego rodzica.

```
<div id="foo">  
  <span id="bar">Bar</span>  
  <span id="baz">Baz</span>  
  <span id="buz">Buz</span>  
</div>
```

```
var bazElement = document.querySelector('#baz');  
var bar = bazElement.previousElementSibling;  
var buz = bazElement.nextElementSibling;
```


Poruszanie się w dół drzewa

Jeśli poruszamy się w dół drzewa, to mamy do wyboru wszystkie dzieci danego elementu.

Możemy użyć następujących atrybutów:

- **el.children** – zwraca tablicę wszystkich dzieci,
- **el.firstElementChild**
– zwraca pierwsze dziecko,
- **el.lastElementChild**
– zwraca ostatnie dziecko.

Metoda sprawdzającą, czy element ma dzieci:

- **el.hasChildNodes()** – zwraca **true** albo **false**.

Kod HTML

```
<div id="foo">  
  <span id="bar">Bar</span>  
  <span id="baz">Baz</span>  
  <span id="buz">Buz</span>  
</div>
```

Kod JavaScript

```
var fooElement = document.querySelector('#foo');  
var allChildren = fooElement.children;  
var bar = fooElement.firstElementChild; // lub children[0]  
var buz = fooElement.lastElementChild;  
// lub allChildren[allChildren.length - 1]
```

Zadania

Wykonaj zadania
z działu
2.DOM
z folderu

5.Poruszanie_po_DOM

Tworzenie elementów

Tworzenie elementów

- Potrafimy już wyszukiwać gotowe elementy istniejące na stronie.
- WJavaScriptcie możemy też tworzyć nowe elementy, które na bieżąco dołączamy do strony. Dzięki temu zwiększamy jej interaktywność.

- Elementy możemy tworzyć przez użycie metody **createElement()** na obiekcie **document**.
- Do metody tej przekazujemy napis oznaczający, jakiego typu element chcemy stworzyć.
- Nowo utworzony element najlepiej zapamiętać do zmiennej, żeby potem nim manipulować.

```
var newDiv = document.createElement("div");
```

Klonowanie elementów

- Jeżeli mamy już element, na którym chcemy się wzorować (np. ma ustawione odpowiednie klasy, atrybuty), to łatwo możemy go sklonować dzięki metodzie **cloneNode(deep)**.
- Wartość **deep** oznacza, czy klonowanie ma być głębokie czy nie.
- Głębokie klonowanie kopiuje i zwraca węzeł wraz z całym poddrzewem.

```
var toClone = document.querySelector('#foo');  
var newDiv = toClone.cloneNode(true);
```

Element stworzony a element dodany do DOM

- Stworzenie elementu nie oznacza, że jest dodany do DOM. Możemy go przypisać do zmiennej, pracować na nim, ale nie będzie on dostępny dla użytkownika naszej strony.
- Element stanie się widoczny na stronie dopiero w chwili, w której zostanie on do niej podpięty.

Dodawanie elementów do DOM

W celu poprawnego dodania elementu do DOM możemy użyć następujących metod:

- **el.appendChild(nowyElement)** – dodaj element jako ostatnie dziecko danego węzła,
- **el.insertBefore(nowyElement, dziecko)** – dodaj element przed jednym z podanych dzieci,
- **el.replaceChild(nowyElement, dziecko)** – zamień podane dziecko.

**Te metody stosujemy
na elementach należących już do DOM.**

```
<div id="foo"></div>
```

```
var fooElement =  
document.querySelector('#foo');
```

```
var newBar = document.createElement("div");  
fooElement.appendChild(newBar);
```

```
var newBuz = document.createElement("h1");  
fooElement.insertBefore(newBuz, newBar);
```

```
var newBuz = document.createElement("p");  
fooElement.replaceChild(newBuz, newBar);
```

Usuwanie elementów z DOM

W celu usunięcia elementu już istniejącego na stronie musimy użyć metody na jego rodzicu:

removeChild(element)

```
var toDelete = document.querySelector('#bar');  
toDelete.parentNode.removeChild(toDelete);
```


Zadania

Wykonaj zadania
z działu
2.DOM
z folderu
6.Tworzenie_nowych_elementow

Inputy i formularze

Elementy typu form

Formularze mają kilka własnych atrybutów, możemy je przypisać tylko do nich. Są to:

- **form.action** – adres URL, do którego prowadzi formularz,
- **form.method** – metoda, którą wysyłany jest formularz,
- **form.elements** – kolekcja elementów należących do tego formularza (w kolejności wpisanej w kodzie HTML).

Formularze mają też specjalne eventy:

- **submit** – jest wywoływany przed wysłaniem formularza. Wysyłanie możemy zablokować przez **preventDefault()** albo zwrócenie **false** z tego eventu,
- **reset** – wywoływane po zresetowaniu formularza (rzadko używane).

Input.value

Elementy typu **input** mają kilka atrybutów specjalnych. Jeden z nich jest następujący:

- **input.value** – zwraca wartość, na jaką nastawiony jest **input**. Może służyć też do nastawienia wartości.

Kod HTML

```
<input id="name">
```

Kod JavaScript

```
var input = document.querySelector('#name');
```

input.value

Zwróci treść
wpisaną przez użytkownika

ImięUżytkownika

```
input.value = "Adam"
```

Nastawi wartość
Inputa na napis **Adam**

Adam

Elementy typu input

- **input.type** – inputy trzymają swój typ. Można go też łatwo zmienić na inny.
- **input.disabled** – zwraca wartość booleanowską, która oznacza, czy element jest włączony czy nie. Możemy ją zmieniać.
- **input.checked** (tylko checkboxy) – zwraca wartość booleanowską, która oznacza, czy element jest zaznaczony czy nie.
- **input.selected** (tylko elementy select) – zwraca wartość booleanowską, która oznacza, czy element jest wybrany czy nie.

Elementy typu **input** mają kilka specjalnych eventów, są to:

- **blur** – wywoływany, gdy użytkownik opuści pole,
- **focus** – wywoływany, gdy użytkownik zaznaczy pole,
- **change** – wywoływany, gdy zmieni się wartość pola,
- **keydown, keyup, keypress** – eventy związane z pisanie na klawiaturze.

Zadania

Wykonaj zadania
z działu
2.DOM
z folderu
7.Inputy_i_formularze