European Funds
Knowledge Education Development

Wrocław University of Science and Technology

European Union
European Social Fund

ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej

Aplikacje webowe na platformę .NET

W04 – JavaScript, przegląd możliwości języka w odniesieniu do HTML/CSS

Syllabus

- JavaScript (JS) – język skryptowy
- Znacznik `<script>`
- Konsola JS
- Wybrane elementy JS:
  - Zmienne
  - Deklaracje `var`, `let`, `const`
  - Typy/wartości
  - Funkcje
  - Parametry funkcji
  - Obiekty (tablice mieszające/haszujące)
  - Klasy
  - Tablice indeksowane liczbami
- JS w dokumencie HTML:
  - Dostęp do elementów HTML
  - Model DOM
  - Manipulowanie elementami HTML
  - Manipulowanie stylami CSS
  - Reakcja na zdarzenia

JavaScript

- Podobieństwo – składniowo do Java-y:
  - Pętle, instrukcje warunkowe, operator podstawienia, operacje matematyczne, relacyjne, logiczne itp., średnik na końcu instrukcji
  - Operator kropki dla obiektów
  - Częściowo – metody/funkcje, zmienne
- Język **skryptowy**:
  - Błędy składniowe – skrypt się nie uruchomi
  - Błędy wykrywane w trakcie wykonania – następuje wtedy przerwanie danego skryptu
- Pierwotnie głównie do przeglądarek
  - Kiedyś mało wydajny
  - Obecnie kompilatory napisane od nowa (w związku z rozwojem aplikacji webowych, szczególnie SPA – ang. **Single Page Application**), zoptymalizowane.
  - Istnieją „wbudowane” zmienne/metody globalne związane z obsługą dokumentu HTML.
  - W elementach HTML można dołączać metody ze skryptów JavaScript.
- Cechy:
  - Brak określania typów zmiennych, parametrów, zwracanych wartości metod
    - W danym momencie zmienna ma określony typ, jednak zawsze można podstawić wyrażenie innego typu i wtedy typ zmiennej się zmienia.
  - Oparty na tablicach mieszających i indeksowanych liczbami
  - Specyficznie funkcyjny
  - Obiekty to w zasadzie tablica mieszająca, gdzie jako klucz jest nazwa pola.
  - Do obiektu można dodawać pola i je usuwać.
  - Klasy – wtórne pojęcie do obiektów, obecne w specyfikacji dopiero od niedawna (w nowszych wersjach tego języka).
    - I znaczeniowo dość odległe od rozumienia klas w językach obiektowych typu C++/Java/C# itp.

Znacznik `<script>`

- Znacznik `<script>` oznacza tekst będący skryptem w języku JavaScript (domyślnie)
- W HTML5 jest to język domyślny, w innych wersjach (HTML4/XHTML) należy dopisać atrybut typu MIME, czyli `type="text/javascript"`.
  - Teoretycznie mogą istnieć inne języki skryptowe (jak szukanie Yeti)
  - Można też używać `type="application/javascript"`
- Bardzo wiekowe przeglądarki nie miały obsługi skryptów, więc dla pewności umieszczano za tagiem otwierającym skrypt i przez tagiem zamykającym kod HTML tworzący z kodu JavaScript komentarz (przykład poniżej)
- Oczywiście kod JavaScript nie jest wyświetlany na stronie WWW

```
    <title>Old JS</title>
</head>
<body>
    <h1>Old JS</h1>
    <script type="text/javascript">
        <!--
        // tu kod JavaScript
        -->
    </script>
    <p>Lorem ipsum dolor sit amet conse
</body>
</html>
```

```
script {
  display: none;
}
```

Znacznik `<script>`

- Znacznik `<script>` można używać na dwa sposoby:
  - Pomiędzy znacznik otwierający o zamykający wstawić kod programu w Javascript
  - Dodać atrybut `src` ze wskazaniem pliku z kodem JavaScript

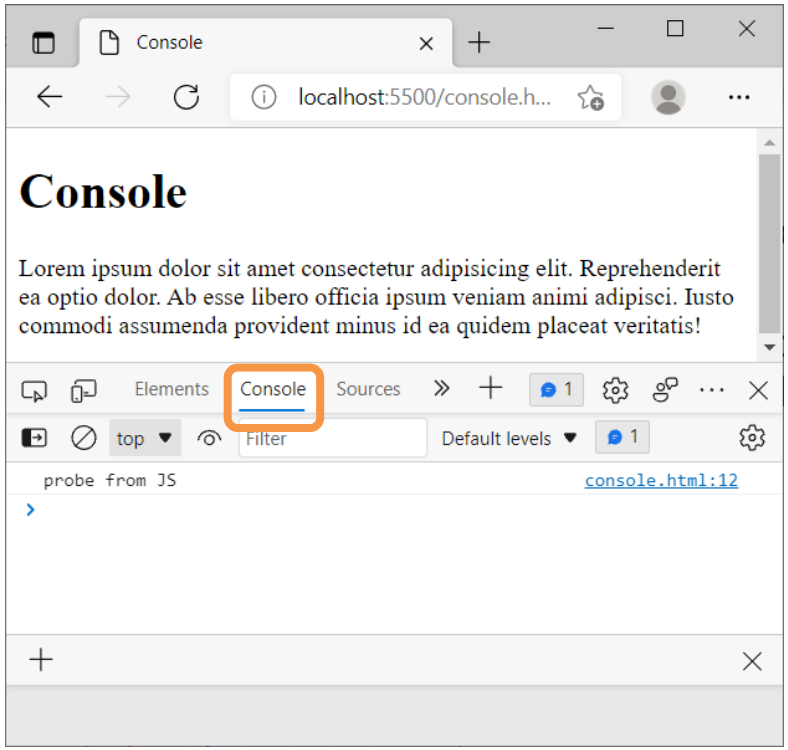
```
<script type="text/javascript"
src="lokalizacja skryptu"> </script>
```

- Miejsce wstawienia jest ważne, gdyż kod jest analizowany w kolejności występowania w trakcie przetwarzania strony WWW.
- Standardowo większość kodu jest albo w elemencie `<head>` albo na końcu strony.
  - Na początku można ładować pliki JS frameworków
  - Na końcu te fragmenty, które modyfikują coś w bieżącym dokumencie.

Konsola JS

- Aby obserwować działanie skryptu można wykorzystać konsolę, czyli obiekt `console` i jego metodę `log()`.
- Wynik wykonania tej metody można obserwować w narzędziu programistycznym przeglądarki.

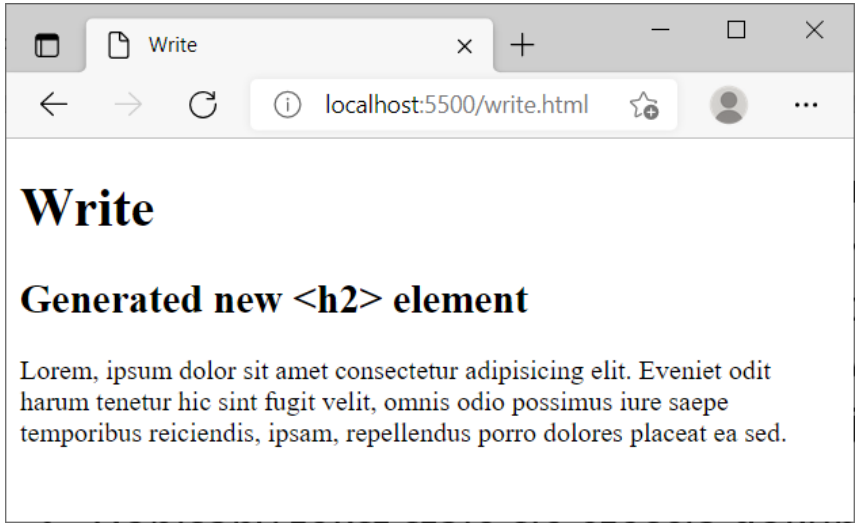
```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta http-equiv="X-UA-Compatible" conte
6       <meta name="viewport" content="width=dev
7       <title>Console</title>
8   </head>
9   <body>
10      <h1>Console</h1>
11      <script>
12          console.log("probe from JS");
13      </script>
14      <p>Lorem ipsum dolor sit amet consectetur
15  </body>
16  </html>
```



Wpisywanie tekstu do dokumentu

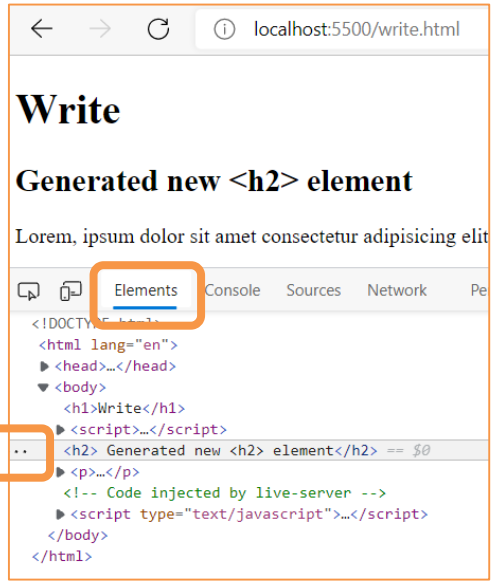
- Za pomocą obiektu `dokument` i jego metod `write()` i `writeln()` można wpisywać tekst do bieżącego dokumentu HTML w miejscu wykonywania skryptu.
  - Druga metoda dodaje na końcu kod przejścia do nowej linii (ale w tekście dokumentu, nie jest to `<br>`)
- Dopisany tekst staje się częścią dokumentu i podlega renderowaniu.
  - Czyli musi dokument po dopisaniu nowej zawartości musi być poprawny, aby poprawnie się wyświetlał.

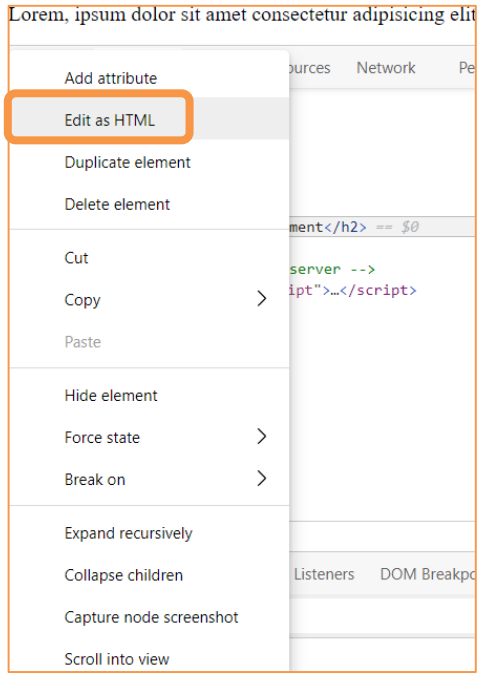
```
    <title>Write</title>
</head>
<body>
    <h1>Write</h1>
    <script>
        document.writeln("<h2> Generated new");
        document.writeln("&lt;h2&gt; element</h2>");
    </script>
    <p>
        Lorem, ipsum dolor sit amet consectetur adipisicing
    </p>
</body>
</html>
```



Znajdowanie zmodyfikowanego tekstu dokumentu

- Oglądając „źródło strony” zobaczy się tylko tekst PRZED wykonaniem wszelkich skryptów.
- Dopiero po znalezieniu elementu `<h2>` za pomocą narzędzia programistycznego przeglądarki i użycia dla niego opcji „Edit as HTML” widać przejście do nowej linii wygenerowane przez `writeln()`.

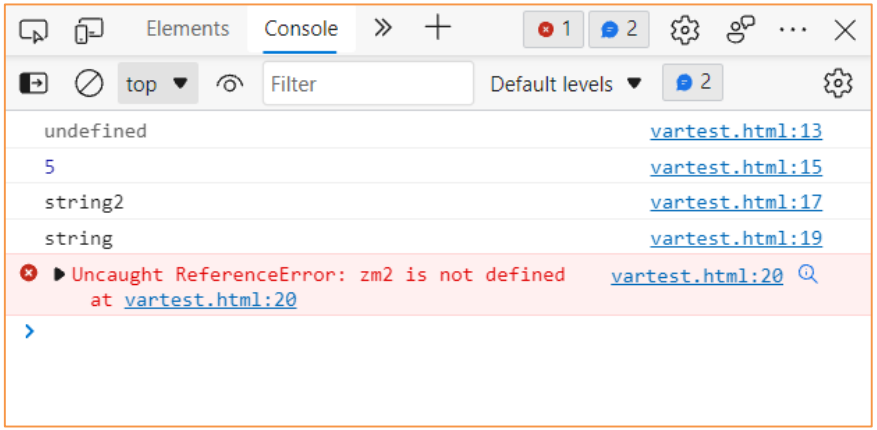




Zmienne

- Identyfikator zmiennej (ale też funkcji, klasy itp.)
  - Ciąg liter, cyfr, znaków dolara ‚$' i podkreślników ‚_'
  - Nie może zaczynać się od cyfry
  - Wielkość liter ma znaczenie (ang. case-sensitive)
  - Słowa zarezerwowane **nie mogą** być identyfikatorem
- Deklaracja zmiennych (poprzedzenie zmiennej słowem kluczowym):
  - **`var`** (od zawsze)
  - **`let`** (od ES6 2015)
  - **`const`** (od ES6 2015)
- Zmienna zadeklarowana za pomocą **`var`** bez zainicjowania ma wartość **`undefined`** (słowo kluczowe)
- Za pomocą **`var`** można wiele razy deklarować zmienną o tym samym identyfikatorze! Nie traci ona poprzedniej wartości, jeśli nic nowego nie przypiszemy.
- Chociaż typ jest ustalany na podstawie przypisania do zmiennej, to może ulec zmianie przy kolejnym przypisaniu.
- Użycie zmiennej nie zadeklarowanej powoduje przerwanie skryptu/bloku kodu z wyjątkiem.

```
7       <title>var Test</title>
8   </head>
9   <body>
10      <h1><code>var</code> Test</h1>
11      <script>
12          var zm1;        // type/value undefinded
13          console.log(zm1);
14          zm1=3;          // type number
15          console.log(zm1+2);
16          zm1="string";   // type string
17          console.log(zm1+2);
18          var zm1;
19          console.log(zm1);
20          console.log(zm2); // not exists
21          console.log("after");
22      </script>
23      <p>
24          Lorem ipsum dolor sit amet consectetur adipisic
25      </p>
26  </body>
27  </html>
```



Problemy z deklaracją **`var`**

- Mają zasięg **globalny** – zmienne globalne!!!
  - Redeklaracja w bloku kodu (np. pętli `for`) operuje na tej samej zmiennej
- Deklaracja zmiennej z **`var`** w ramach funkcji tworzy nową, **lokalną zmienną**.

```
7       <title>var Global</title>
8   </head>
9   <body>
10      <h1><code>var</code> Global</h1>
11      <script>
12          function set(){
13              var zm1=20;
14          }
15          var zm1=5;
16          {
17              var zm1=10;
18          }
19          console.log(zm1);
20          set();
21          console.log(zm1);
22      </script>
23      <p>
24          Lorem ipsum dolor sit amet conse
25      </p>
26  </body>
27  </html>
```

```
10                                  varGlobal.html:19
10                                  varGlobal.html:21
>
```

Widoczność zmiennej zadeklarowanej z var

- (ang. hoisting) Wszystkie zmienne zadeklarowane z **`var`** mają widoczność od początku elementu `<script>`, w którym wystąpią (lub od początku funkcji), mimo że deklaracja będzie w późniejszym fragmencie kodu (ang. hoisting).
- Wszystkie kody JS stanowią **jeden wspólny skrypt**, ale **kompilowany i uruchamiany blokami** `<script>`
  - Zmienne zadeklarowane w jednym elemencie `<script>` są używalne w późniejszych elementach `<script>` (ale nie wcześniejszych)
  - Wyjątki przerywają tylko kod w bieżący elemencie `<script>`, ale **wszystkie** deklaracje **`var`** (bez inicjalizacji) będą zrealizowane.

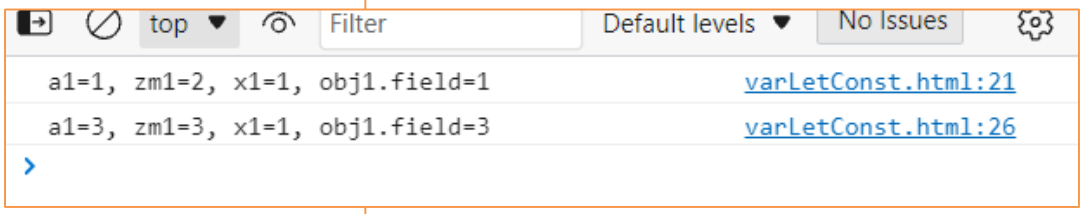
```
10      <h1><code>var</code> From start</h1>
11      <script>
12          function set(){
13              lok=10; // use before declaration
14              zm1=20;
15              var lok; // local declaration
16              return lok+zm1;
17          }
18          zm1=5;      // use before declaration
19          var zm1;
20          var x=set();
21          console.log("zm1="+zm1);
22          // console.log("zm2="+zm2); // generates exception
23          console.log("x="+x);
24          console.log("lok="+lok);
25          var zm3=333; // only declaration happends
26      </script>
27      <p>
28          Lorem ipsum dolor sit amet consectetur adipisicing e
29      </p>
30      <script>
31          console.log("next fragment of script");
32          var zm2=100;
33          console.log("zm1="+zm1);
34          console.log("zm3="+zm3);
35          console.log("x="+x);
36      </script>
```

```
zm1=20                                          varFromStart.html:21
x=30                                            varFromStart.html:23
▶ Uncaught ReferenceError: lok is not defined   varFromStart.html:24
    at varFromStart.html:24
next fragment of script                         varFromStart.html:31
zm1=20                                          varFromStart.html:33
zm3=undefined                                   varFromStart.html:34
x=30                                            varFromStart.html:35
>
```

Deklaracja zmiennej za pomocą `let` i `const`

- Zmienna zadeklarowana w bloku jako `let`/`const` nie może być ponownie zadeklarowana w tym bloku.
- W wewnętrznym bloku deklaracja taka oznacza nową zmienną, której zakres kończy się z końcem bloku.
- Zmienne te są widoczne dopiero od momentu deklaracji.
- Zmienną zadeklarowaną przez `let` można zmieniać, zmienne zadeklarowane jako `const` zmieniać nie wolno.
  - Ale dla `const` można nadal **zmieniać zawartość** (obiekt,tablica) takiej zmiennej, czyli jest stałą referencją (do tablicy, obiektu)

```
10      <h1><code>var/let/const</code></h1>
11      <script>
12          let a1=1;
13          var zm1=1;
14          const x1=1;
15          const obj1={field:1};
16          {
17              let a1=2;
18              var zm1=2;
19              const x1=2;
20          }
21          console.log("a1="+a1+", zm1="+zm1+", x1="+x1+", obj1.field="+obj1.field);
22          a1=3;
23          zm1=3;
24          //x1=3; Uncaught TypeError: Assignment to constant variable.
25          obj1.field=3;
26          console.log("a1="+a1+", zm1="+zm1+", x1="+x1+", obj1.field="+obj1.field);
27      </script>
```



Typy/wartości

- Tylko typy:
  - `undefined` – zmienna nie ma zdefiniowanej wartości ani typu.
  - `number` (liczby)
  - `string` (ciągi znaków)
  - `object` – obiekty (też tablice), istnieje wartość `null`.
  - `function` – typ funkcyjny
  - logiczny (`true`/`false`)
- Klasy (ES6 2015) – do szybszego tworzenia obiektów, bardziej adekwatna byłaby nazwa „prototyper obiektów” lub wzorce obiektów.
  - Operator `new`

```
11    <script>
12        let $num = 5.67;
13        let str$ ="A small sentence.";
14        let obj={ name:"Einstein", salary:2000};
15        // existing variables
16        console.log($num+", "+str$+", "+obj+", "+ obj.name+", "+obj.salary);
17        console.log("types: "+(typeof $num)+", "+typeof(str$)+", "+typeof(obj));
18        // undefined
19        console.log("types? :"+(typeof x)+", "+typeof(obj.noExist)+", "+typeof(obj[2]));
20        console.log("values? :"+ x+", "+obj.noExist+", "+obj[2]);
21        // impossible
22        console.log(obj.noExist.nextNotExist);
23        var x=5;
24    </script>
```

```
5.67, A small sentence., [object Object], Einstein, 2000        types.html:16
types: number, string, object                                   types.html:17
types? :undefined, undefined, undefined                         types.html:19
values? :undefined, undefined, undefined                        types.html:20
▶ Uncaught TypeError: Cannot read properties of undefined (reading 'nextNotExist')   types.html:22
    at types.html:22
```

Funkcje

- Funkcje można deklarować na dwa poniższe sposoby
  - Pierwszy sposób to tylko skrót notacyjny.
  - Wytwarza się zmienną o nazwie funkcji, która jest typu `function`.
  - Jak każda zmienna, można zmienić jej wartość i/lub typ.
- Parametry funkcji nie mają podanych typów
  - Mogą mieć wartości domyślne
- Typ wyniku może być zależny od wykonania funkcji
- **Nieokreślone** w wywołaniu funkcji argumenty mają wartość `undefined`.
- **Nadmiarowe** argumenty są ignorowane.
  - Nie do końca (dalsze slajdy)
- Od ES6 można stosować zapis funkcji wyrażeniem lambda np.

```
var addNew=(x,y) => x+y;
```

lub

```
var addNew=(x,y) => { return x+y;};
```

```
12      function add(x,y){
13          let result=x+y;
14          return result;
15      }
16      console.log(add(2,3));
17      var addNew=function(x,y){ // var/let/const
18          return x+y;
19      }
20      console.log(addNew(2,3));
21      add=function(name){ return "name="+name;};
22      console.log(add("Jane"));
23      console.log(add(2,3));
24      console.log(add());
```

```
5                   functions.html:16
5                   functions.html:20
name=Jane           functions.html:22
name=2              functions.html:23
name=undefined      functions.html:24
```

Parametry funkcji

- Może być **tylko jedna** funkcja o danej **nazwie**.
  - Brak przeciążania funkcji
- Można stworzyć długą listę parametrów i sprawdzić, które są zdefiniowane
- Można użyć **tablicy parametrów**, gdyż dla każdego wywołania funkcji tworzy się tablica indeksowana od 0 z argumentami wywołania funkcji pod lokalną zmienną `arguments`, która jest podobna do tablicy indeksowanej liczbami.
- Argumenty to kopie liczb, string-ów lub (referencji obiektu-sprawdzić).

```
10      <h1>Arguments</h1>
11      <script>
12          function showArgs(x,y){
13              for(let i=0;i<arguments.length;i++)
14                  console.log(`[${i}]=`+arguments[i]);
15          }
16          function check(){
17              showArgs("Jan",6);
18          }
19          console.log("check()");
20          check();
21          console.log("showArgs()");
22          showArgs(3,"Jane");
23          console.log("showArgs() with 5 args");
24          showArgs(0,"Jane",2,3,4);
25          console.log("showArgs() with 1 arg");
26          showArgs("one");
```

```
check()                          arguments.html:19
[0]=Jan                          arguments.html:14
[1]=6                            arguments.html:14
showArgs()                       arguments.html:21
[0]=3                            arguments.html:14
[1]=Jane                         arguments.html:14
showArgs() with 5 args           arguments.html:23
[0]=0                            arguments.html:14
[1]=Jane                         arguments.html:14
[2]=2                            arguments.html:14
[3]=3                            arguments.html:14
[4]=4                            arguments.html:14
showArgs() with 1 arg            arguments.html:25
[0]=one                          arguments.html:14
```

Obiekty

- Obiekt to słownik par klucz-wartość. Klucz zwany jest polem/atrybutem.
  - Słownik zrealizowany jako tablica haszująca.
- **Klucz** to poprawny **identyfikator** (ale …), a **wartość** może być wyrażeniem **dowolnego typu** (również innym obiektem, tablicą czy funkcją).
- Klucz z wartością łączony jest dwukropkiem ‚:'
- Pary rozdziela się przecinkiem i wstawia w nawias klamrowy.
- Dostęp do atrybutów w zmiennej obiektowej uzyskuje się poprzez operator kropki.

- W trakcie korzystania z obiektu można mu **dodać** kolejny atrybut lub **usunąć**!

```
12      let person={ name:"Einstein", salary:2000};
13      console.log(person);
14      person.salary+=100;
15      console.log(person);
16      person.year=1900;
17      console.log(person);
18      person.name=undefined;
19      console.log(person);
20      delete person.salary;
21      console.log(person);
22      person["forname"]="Albert";
23      console.log(person);
24      person.name=() => "Name";
25      console.log(person);
26      console.log(person.name());
```

```
▶ {name: 'Einstein', salary: 2000}                    objects.html:13
▶ {name: 'Einstein', salary: 2100}                    objects.html:15
▶ {name: 'Einstein', salary: 2100, year: 1900}        objects.html:17
▶ {name: undefined, salary: 2100, year: 1900}         objects.html:19
▶ {name: undefined, year: 1900}                       objects.html:21
▶ {name: undefined, year: 1900, forname: 'Albert'}    objects.html:23
▶ {year: 1900, forname: 'Albert', name: ƒ}            objects.html:25
Name                                                  objects.html:26
```

Zasada podstawienia Liskov:
Funkcje które używają wskaźników lub referencji do klas bazowych, muszą być w stanie używać również obiektów klas dziedziczących po klasach bazowych, bez dokładnej znajomości tych obiektów

Nie działa

Klasy (od ES6)

- W JS klasa to generator obiektu
- Przetwarzany obiekt w konstruktorze lub funkcjach jest dostępny pod zmienną `this`.
- Konstrukcja obiektu wykonuje się przez operator **`new`**. Uruchamia się wtedy metoda `constructor()`, która może być **tylko jedna**.
- Po stworzeniu nadal wszystko można powymieniać w obiekcie!
  - Nie można mówić, że obiekt `obj` jest typu klasy `Class`, nawet jak powstał z jej konstruktora.
  - Ale informacja o prototypie (klasie) jest dostępna

```
12      class Person{
13          constructor(name,salary){
14              this.name=name;
15              this.salary=salary;
16          }
17          getFormated(separator){
18             let result=this.name+separator+this.salary;
19             //let result=""+this.name+separator+this.salary;
20             return result;
21          }
22      }
23      let person=new Person("Einstein",2000);
24      console.log(person);
25      console.log(person.getFormated(" -> "));
26      let number=new Person(1,2);
27      console.log(number.getFormated(3)); // operacja plus dla liczb
```

```
▶ Person {name: 'Einstein', salary: 2000}   class.html:24
Einstein -> 2000                            class.html:25
6                                           class.html:27
```

odkomentowanie

```
▶ Person {name: 'Einstein', salary: 2000}   class.html:24
Einstein -> 2000                            class.html:25
132                                         class.html:27
```

Wywołanie funkcji w obiekcie stworzonym z klasy

- Ogólnie: najpierw poszukiwana jest funkcja o danej nazwie w obiekcie i zostanie wykonana.
- Jeśli takiej nie ma, poszukuje się w prototypie klasy z której obiekt powstał
- Ewentualnie w klasie nadrzędnej. Itd.
- Jak nie znajdzie się jej nigdzie – wyjątek wykonania

- Poniżej: obiekty mają dużo informacji we właściwości `prototype`.

Tablice indeksowane liczbami

- Tablice to specjalne obiekty z wieloma funkcjami i polem `length`, indeksowane liczbami całkowitymi nieujemnymi
  - Zamęt: teoretycznie można dodawać też indeksy ujemne, niecałkowite i nie liczbowe – bo to nadal jest poprawne dla słownika. Teoretycznie indeksem może być inny obiekt, funkcja czy tablica. **Pole** `length` **pamięta największy dodatni indeks całkowity**.
- Dostęp do elementów za pomocą nawiasów kwadratowych z indeksem.
- Można tworzyć przez zapis klamrowy lub konstruktor dla klasy `Array`
  - Z jednym argumentem liczbą całkowitą – parametr określa długość tablicy (z elementami równymi `undefined`)
  - Z zerem lub wieloma argumentami: argumenty stają się kolejnymi elementami tablicy.

Przykład tablicy indeksowanej liczbami

```
12      var arr=[5,10,15];
13      console.log("len="+arr.length+", "+arr[1]); //3, 10
14      arr[3]=20;
15      console.log("len="+arr.length+", "+arr[3]); //3, 20
16      console.log(arr[5]); //undefined
17      arr[10]=100;
18      console.log("len="+arr.length+", "+arr[10]); //11, undefined
19      arr[-1]=-10;
20      console.log("len="+arr.length+", "+arr[-1]); //11, -10
21      const arr2=[]; // constant reference
22      arr2[2]="two";
23      console.log("len="+arr2.length+", "+arr2[2]); //3, two
24      const arr3=new Array();  //empty array
25      const arr4=new Array("one","two","three");  // 3-elements array
26      const arr5=new Array(1,"two",arr4);  // 3-elements array
27      const arr6=new Array("one");  // 1-elements array
28      const arr7=new Array(11);  // 11-elements array
29      console.log(arr3,arr4,arr5,arr6,arr7);
30      console.log();
```

```
len=3, 10                                                                    arrays.html:13
len=4, 20                                                                    arrays.html:15
undefined                                                                    arrays.html:16
len=11, 100                                                                  arrays.html:18
len=11, -10                                                                  arrays.html:20
len=3, two                                                                   arrays.html:23
▼ []                ▼ (3) ['one', 'two', 'three']   ▼ (3) [1, 'two', Array(3)]   ▼ ['one']   ▼ (11) [empty × 11]   arrays.html:29
    length: 0           0: "one"                        0: 1                        0: "one"        length: 11
  ▶ [[Prototype]]: Array(0)   1: "two"                  1: "two"                    length: 1     ▶ [[Prototype]]: Array(0)
                        2: "three"                    ▶ 2: (3) ['one', 'two', 'three']   ▶ [[Prototype]]: Array(0)
                        length: 3                       length: 3
                      ▶ [[Prototype]]: Array(0)       ▶ [[Prototype]]: Array(0)
```

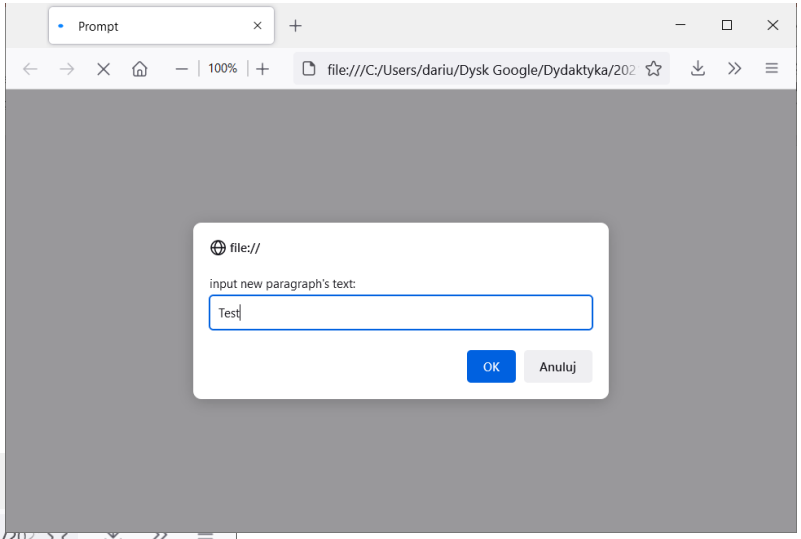
JavaScript a strona HTML

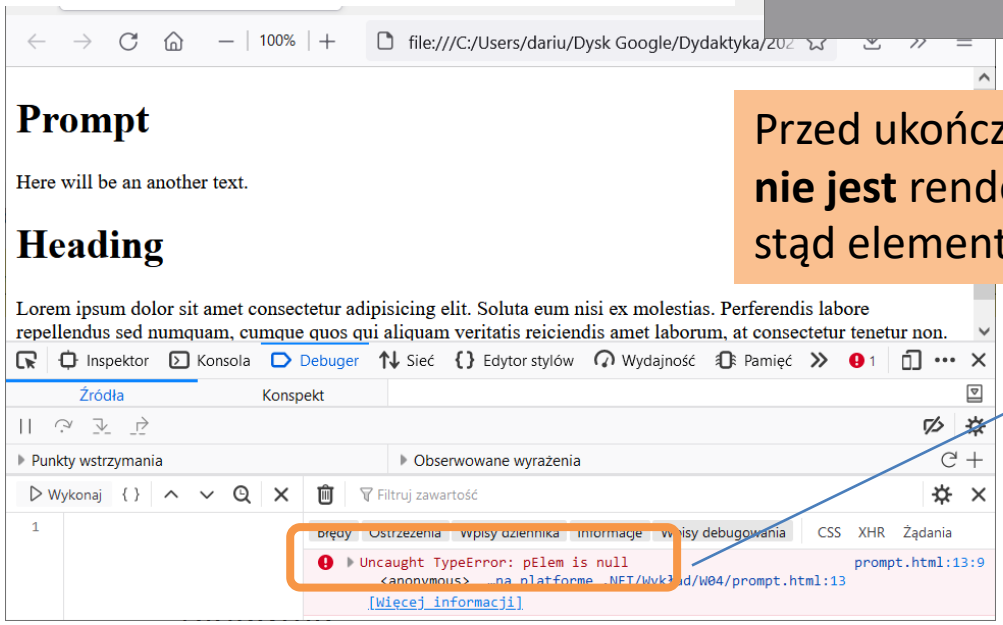
- Pewne obiekty są zawsze obecne dla kodu w języku JavaScript na stronie WWW i posiadają bardzo dużo właściwości i funkcji:
  - `window` – obiekt z danymi o oknie (zakładce) z dokumentem HTML
  - `document` (kopia `window.document`) – do zarządzania elementami dokumentu HTML

- Prosta komunikacja z użytkownikiem:
  - `window.prompt(tekstZapytania,domyślnaWartość)`
    - Otwiera okno z prośbą o tekst, zwraca wpisany string.
  - `pElem=document.getElementById(idElementu)`
    - Referencja na obiekt elementu o podanym identyfikatorze lub `undefined`
  - pole `pElem.innerHTML` – po przypisaniu nowej wartość rendering tekstu traktowane jak fragment HTML
  - Pole `pElem.innerText` – po przypisaniu dokładnie taki sam tekst wewnętrzny elementu (z ewentualną zmianą na znaki specjalne)

Przykład - prompt

```
7       <title>Prompt</title>
8   </head>
9   <body>
10      <script>
11          var text1=window.prompt("input new paragraph's text:","example");
12          var pElem=document.getElementById("p1");
13          pElem.innerHTML=text1;  // tag rendering
14          var text1=window.prompt("input new heading text:","Main");
15          var pElem=document.getElementById("myH1");
16          pElem.innerText=text1; // no rendering
17          //console.log();
18      </script>
19      <h1>Prompt</h1>
20      <p id="p1">
21          Here will be an another text.
22      </p>
23      <h1 id="myH1"> Heading </h1>
24      <p>
25          Lorem ipsum dolor sit amet consectetur adipisicing elit. Soluta eum
26      </p>
27  </body>
```



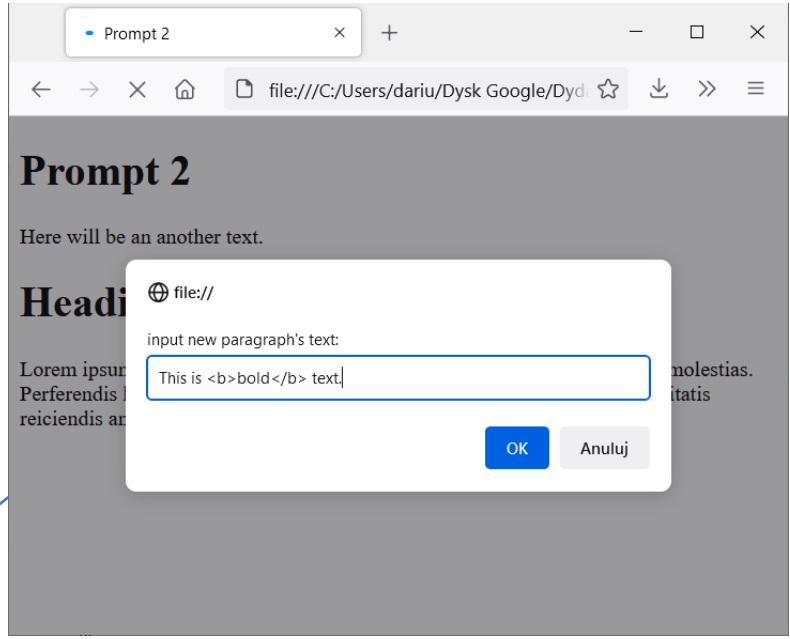


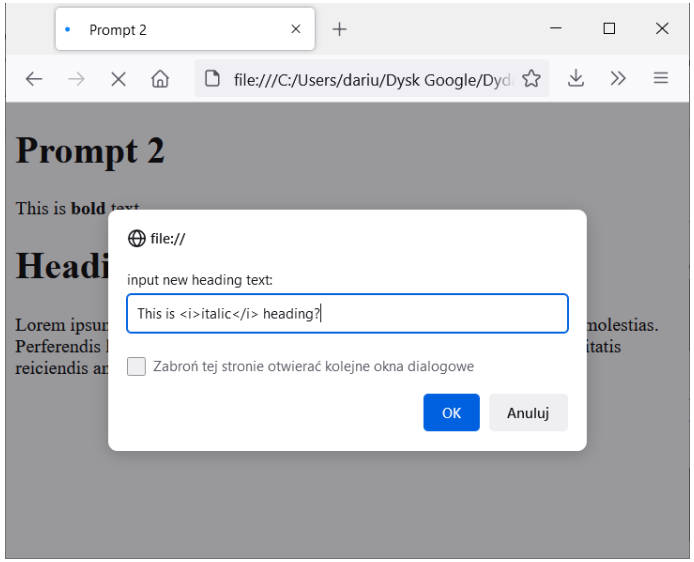
Przed ukończeniem skryptu reszta strona **nie jest** renderowana,
stąd elementu `#p1` jeszcze nie ma!

Można przenieść skrypt na koniec, ale nie zawsze jest to najlepsze wyjście

Przykład – Prompt 2

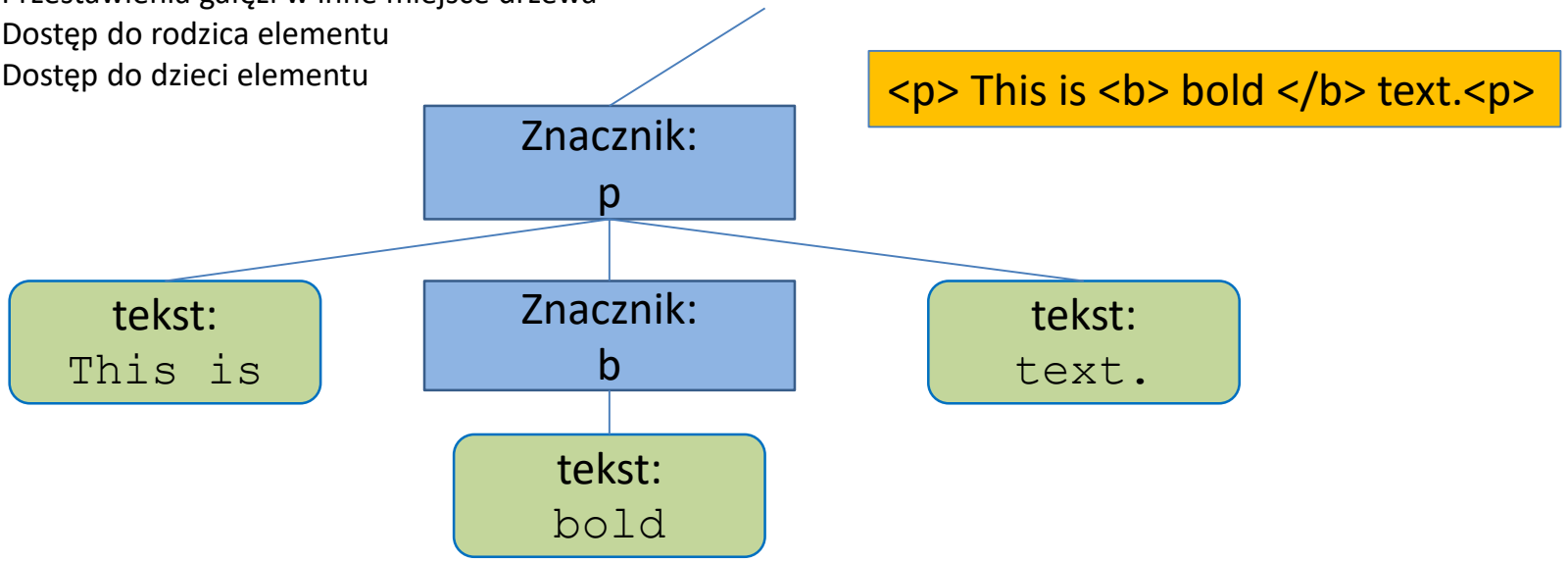
```
7       <title>Prompt 2</title>
8   </head>
9   <body>
10      <h1>Prompt 2</h1>
11      <p id="p1">
12          Here will be an another text.
13      </p>
14      <h1 id="myH1"> Heading </h1>
15      <p>
16          Lorem ipsum dolor sit amet consectetur adipisicing elit. Soluta eum
17      </p>
18      <script>
19          var text1=window.prompt("input new paragraph's text:","example");
20          var pElem=document.getElementById("p1");
21          pElem.innerHTML=text1;  // tag rendering
22          var text1=window.prompt("input new heading text:","Main");
23          var pElem=document.getElementById("myH1");
24          pElem.innerText=text1; // no rendering
25      </script>
26  </body>
```



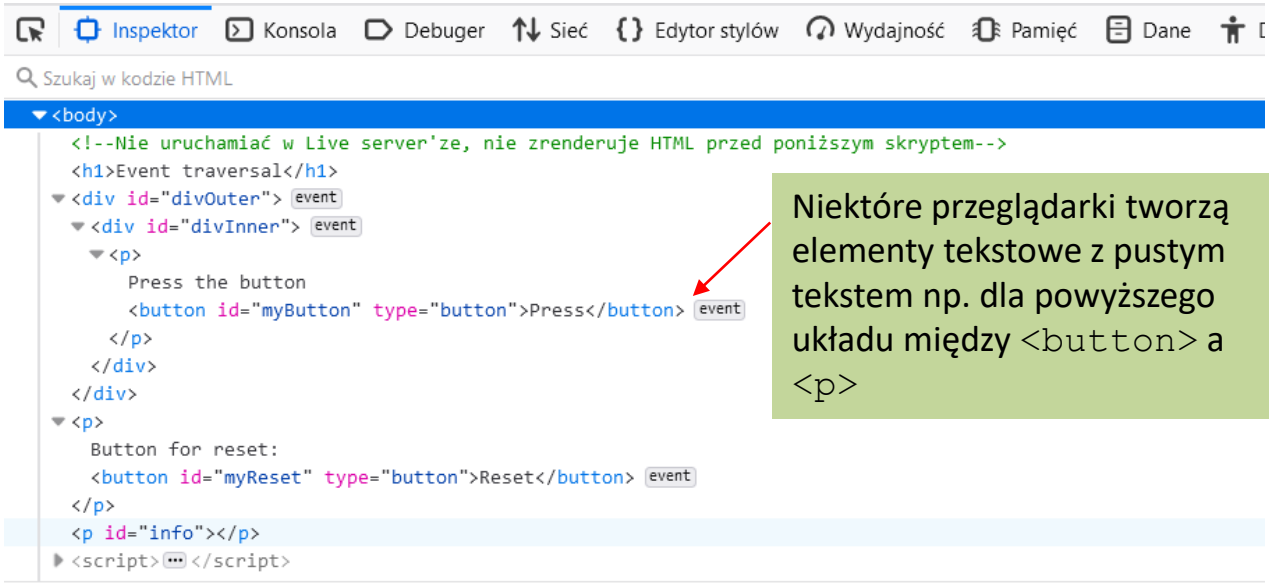


DOM, atrybuty elementu

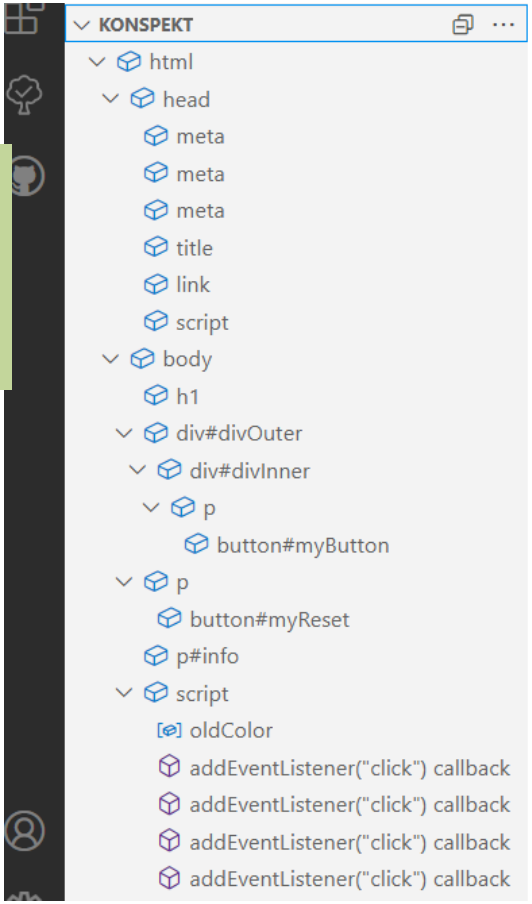
- Obiektowy Model Dokumentu (ang. Document Object Model – DOM) pozwala na dostęp do wszystkich elementów na stronie.
- Elementy HTML (i nie tylko, np. po prostu tekst) stanowią drzewo
- Poprzez wykorzystanie JavaScript można dynamicznie tworzyć, modyfikować i usuwać elementy ze strony.
- Tworzenie nowego znacznika
  - `document.createElement(nazwaElementu)`
- Tworzenie nowego elementu tekstowego
  - `document.createTextNode(tekst)`
- Istnieją metody do manipulowania elementami w ramach modelu DOM:
  - Usuwania z modelu DOM
  - Dodania do rodzica jako dziecko
    - Na początku listy dzieci, na końcu, za wybranym dzieckiem itp.
  - Przestawienia gałęzi w inne miejsce drzewa
  - Dostęp do rodzica elementu
  - Dostęp do dzieci elementu

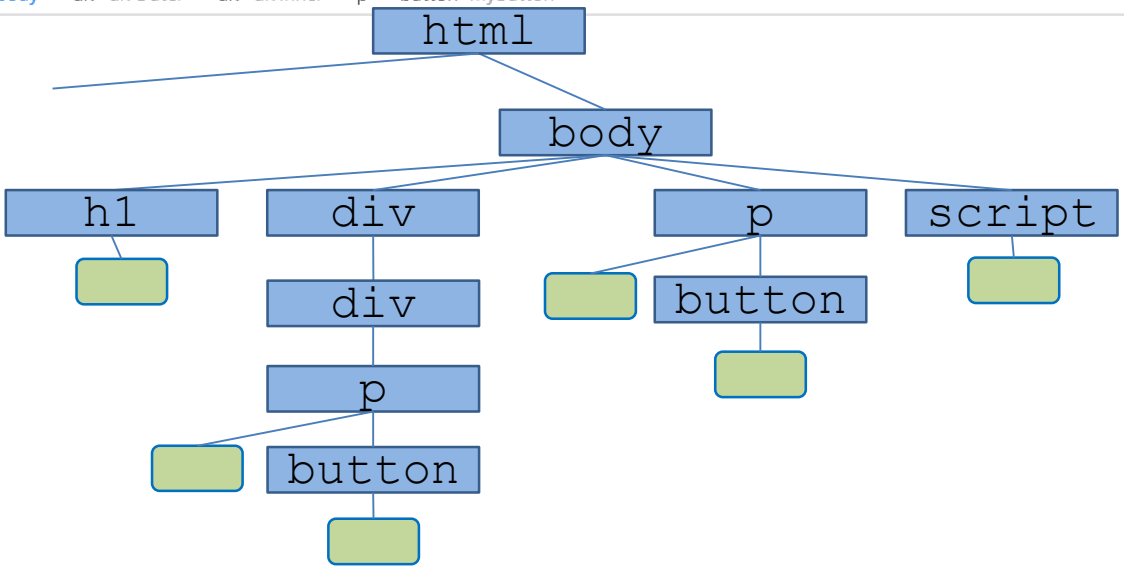


Model DOM w narzędziu programistycznym i VS Code



Niektóre przeglądarki tworzą elementy tekstowe z pustym tekstem np. dla powyższego układu między `<button>` a `<p>`



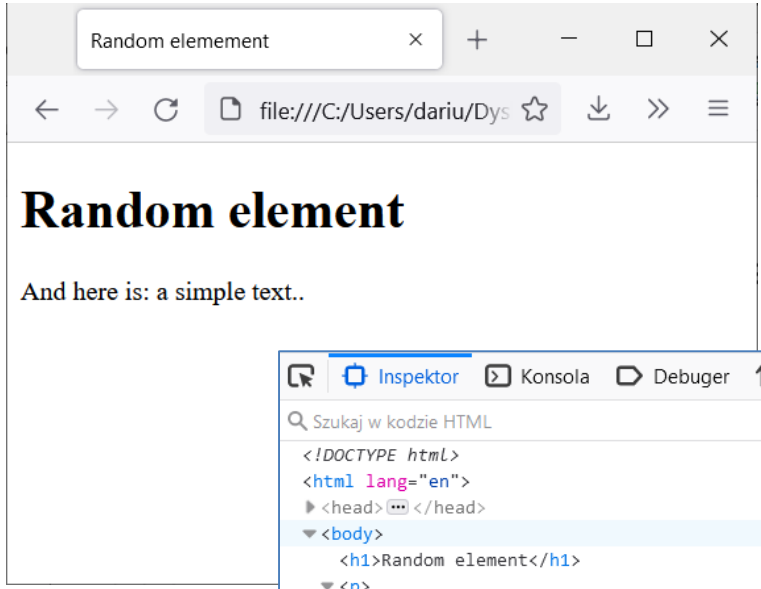


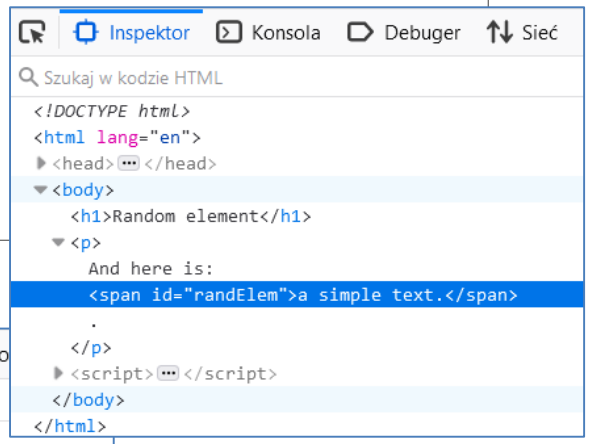
Atrybuty elementów HTML

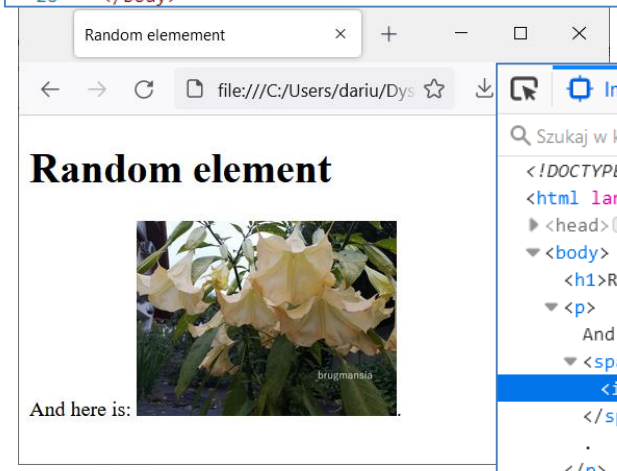
- Można również manipulować atrybutami elementu:
  - `getAtributte(nazwaAtrybutu)`
  - `setAtributte(nazwaAtrybutu, wartośćAtrybutu)`
- Atrybuty dostępne są również poprzez operator kropki i nazwę atrybutu
  - Jeśli w nazwie atrybutu są minusy (notacja kebab-case) zamieniana jest na notację wielbłądzią (camelCase)
    - Np. atrybut `background-color` zamieniony zostanie na pole `backgroundColor`.
  - Pewne atrybuty, które są słowami zarezerwowanymi mają inną nazwę np. `className`
    - Nie ma `elem.class`, ale można też użyć `elem.setAttribute("class", nazwyKlas);`
    - Lub `elem["class"]=nazwyKlas;`
    - Jest też właściwość `classList` z metodami `add()`,`constains()`,`remove()` itp.
- Przykład: losowo wybrany element w elemencie `<span id="randElem">`.
  - Albo obrazek, albo zwykły tekst.
  - Dla obrazka różne sposoby ustawiania atrybutów.
  - Wykorzystanie obiektu `Math` i jego statycznych funkcji:
    - `random()` – zwraca wartość losową w przedziale <0;1)
    - `floor(x)` – zwracający najwyższą liczbę całkowitą nie większą niż `x`.

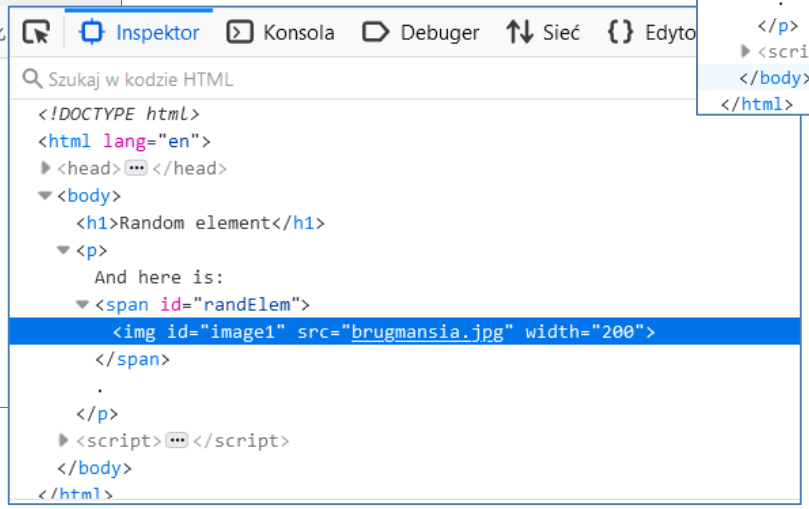
Przykład – losowy obrazek z dwóch

```
7       <title>Random elemement</title>
8   </head>
9   <body>
10      <h1>Random element</h1>
11      <p>
12          And here is: <span id="randElem"></span>.
13      </p>
14      <script>
15          if(Math.floor((Math.random()*2))==0){
16              var elem=document.createElement("img");
17              elem.src="brugmansia.jpg";
18              elem.setAttribute("width","200");
19              elem["id"]="image1";
20          }
21          else{
22              var elem=document.createTextNode(" a simple text.");
23          }
24          var parent=document.getElementById("randElem");
25          parent.appendChild(elem);
26          // document.getElementById("randElem").appendChild(elem);
27      </script>
28  </body>
```









Wybrane operacje na drzewie DOM

- Dodawanie elementu na końcu listy dzieci:
  - `elemRodzic.appendChild(elemDziecko)`
- Dodawanie elementu na początku listy dzieci:
  - `elemRodzic.prepend(elemDziecko)`
- Usunięcie wybranego dziecka z elementu rodzica:
  - `elemRodzic.removeChild(elemDziecko)`
- Dostęp do pierwszego lub ostatniego dziecka:
  - `elemRodzic.firstChild`
  - `elemRodzic.lastChild`
- Podmiana dziecka u rodzica (zwraca zastąpiony element):
  - `elemRodzic.replaceChild(noweDziecko,stareDziecko)`
- Dostęp do rodzica:
  - `elemDziecko.parentNode`
- Dostęp do wszystkich dzieci:
  - `elemRodzic.childNodes`
    - Pętla dla wszystkich dzieci: `for(var child of node.childNodes)`

Przykład - operacje na elementach DOM

```
7       <link rel="stylesheet" type="text/css" href="domOperations.css">
8       <title>DOM operations</title>
9   </head>
10  <body>
11      <h1>DOM operations</h1>
12      <div id="p1">
13          <article id="art3"> Article3</article>
14          <article id="art1"> Article1</article>
15      </div>
16
17      <h1 id="myH1"> Next part </h1>
18      <p>
19          <img id="img1" src="bergenia.jpg" height="200px">
20          Lorem ipsum dolor sit amet consectetur adipisicing elit. Solut
21      </p>
22      <script>
23          var p1=document.getElementById("p1");
24          var art1=document.getElementById("art1");
25          p1.removeChild(art1);
26          p1.prepend(art1);
27          var art2=document.createElement("article");
28          art2.id="art2";
29          art2.innerText="Article2";
30          p1.insertBefore(art2,document.getElementById("art3"));
31          p1.parentNode.style.backgroundColor="yellow";
32          p1.appendChild(document.getElementById("img1"));
33      </script>
34  </body>
```

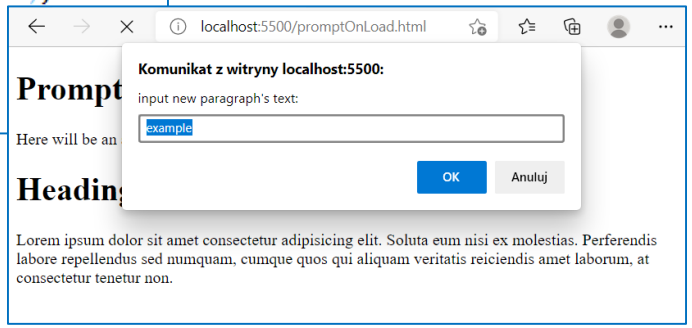
domOperations.css > ...

```
1   article {
2       background-color: aqua;
3       border: brown 2px solid;
4       margin:  3px 3px 3px 20px;
5   }
6
7   #art2 {
8       font-size: 200%;
9   }
```

Zdarzenia na stronie WWW

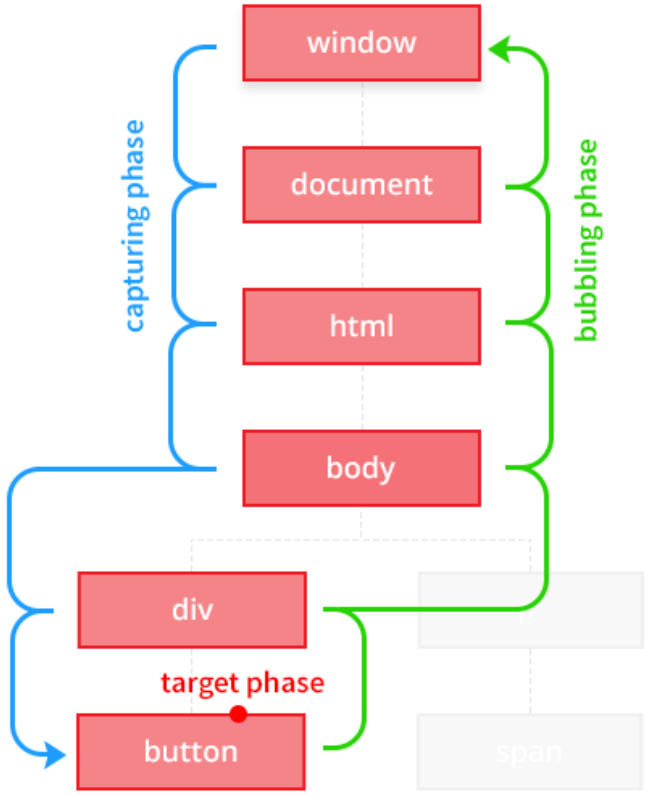
- Aby najpierw wczytała się strona, a potem wykonał się skrypt należy zrobić z niego funkcję, która uruchomi się w momencie zdarzenia `load` dla obiektu `window` (oznaczającego, że strona została wczytana i zrenderowana).
- Polega to np. na rejestracji funkcji w słuchaczu zdarzenia.
  - Może być wiele funkcji dodanych do słuchacza zdarzenia.
  - Można też usuwać funkcję ze słuchacza zdarzenia.
- Przykładowe możliwe zdarzenia: „`load`”, „`click`”,”`focus`” itp.
- Metoda `window.addEventListener(nazwaZdarzenia,nazwaFunkcji,podczasCapture)` dodaje do słuchacza okna (`window`) dla zdarzenia `nazwaZdarzenia` funkcję `nazwaFunkcji`. Parametr `podczasCapture` będzie omówiony dalej.
  - metoda `addEventListener()` istnieje dla wszystkich elementów dokumentu HTML (i nie tylko).

```
18      <script>
19          window.addEventListener("load",
20              function(){
21                  var text1=window.prompt("input new paragraph's text:","example");
22                  var pElem=document.getElementById("p1");
23                  pElem.innerHTML=text1;  // tag rendering
24                  var text1=window.prompt("input new heading text:","Main");
25                  var pElem=document.getElementById("myH1");
26                  pElem.innerText=text1; // no rendering
27              },false);
28      </script>
```



Fazy zdarzenia

- Zdarzenie przebiega w 3 fazach:
  - faza **capture** (przechwytywanie)- kiedy zdarzenie podąża w dół drzewa (od window) do danego elementu
  - faza **target** - kiedy zdarzenie dotrze do elementu, który wywołał to zdarzenie
  - faza **bubbling** (bąbelkowanie) - kiedy zdarzenie pnie się w górę drzewa aż dotrze do window
- To samo zdarzenie można obsługiwać w wielu miejscach drzewa DOM na w/w ścieżkach.
- Podczas rejestracji metody obsługi, w trzecim parametrze, podaje się czy obsługa ma być podczas fazy capture (`true`) czy podczas fazy bubbling (`false`, wartość domyślna).
  - Dla fazy target nie ma to znaczenia.



https://kursjs.pl/kurs/events/events.php

Funkcja obsługi zdarzenia

- Funkcja obsługi zdarzenia otrzymuje jako parametr zdarzenie `e`, które zawiera między innymi:
  - `e.target` – element fazy target
  - `e.currentTarget` – element, który obecnie obsługuje zdarzenie
  - `e.key` – wciśnięty klawisz
  - `e.code` – kod wciśniętego klawisza (tekstowy)
  - `e.ctrlKey` – czy wciśnięty klawisz Ctrl
  - `e.shiftKey` - czy wciśnięty klawisz Shift
  - `e.altKey` - czy wciśnięty klawisz Alt
  - `e.clientX` i `e.clientY` – współrzędne myszki względem obszaru klienta użytkownika (viewportu przeglądarki)
  - `e.screenX`, `e.screenY` - współrzędne myszki względem całego ekranu urządzenia
  - `e.type` – typ zdarzenia bez prefiksu `on-`

- **Zewnętrzne skrypty** JS wstawić można poprzez znacznik `<script>` z atrybutem `src` wskazującym miejsce pliku.

- Testowy program:
  - Zależność: `#divOuter > #divInnder > p > #myButton`
  - Kliknięcie przycisku `#myButton` powoduje, że podczas fazy capture wykonana się obsługa w `<div id="divInner">`,
    - Zmiana koloru tła tego elementu
  - Następnie zdarzenie dotrze elementu `<button id="myButton">`
    - Wypisze alert „`before`”, zmieni kolor przycisku, wypisze alert „`after`”
    - W elemencie `<p id="info">` wypisze dane nt. zdarzenia
  - Aby w fazie bubbling dotrzeć do elementu `<div id="divOuter">`
    - Zmieni kolor tła tego elementu
  - Wciśnięcie przycisku „`Reset`” cofa wszystkie zmiany.
    - Przycisk musi być poza elementem `#divOuter`, inaczej uruchomi się obsługa zdarzenia „`click`” dla tego elementu.

Właściwość `style`

- Dostęp do stylu elementu uzyskuje się przez atrybut style.
- Wiele styli jest dostępnych przez zamianę notacji kebab-case na notację wielbłądzią (camelCase) i użycie notacji kropki (analogicznie jak dla atrybutów HTML).
  - Np. `elem.style.backgroundColor`
  - Jest to równoważne stylowaniu inline.
- Element musi być zrenderowany, aby miał przydzielony właściwy styl z reguł CSS. Np. dopiero co stworzony element przez `document.createElement(nazwaElementu)` ma style domyślne.

Przykład

```html
    <link rel="stylesheet" type="text/css" href="eventTraversal.css">
    <script src="eventTraversal.js"></script>
</head>
<body>
    <!-- Nie uruchamiać w Live server'ze, nie zrenderuje HTML przed poniższym skryptem -->
    <h1>Event traversal</h1>
    <div id="divOuter">
        <div id="divInner">
            <p>
            Press the button <button type="button" id="myButton">Press</button>
            </p>
        </div>
    </div>
    <p>
        Button for reset:<button type="button" id="myReset">Reset</button>
    </p>
    <p id="info"></p>
    <script>
        var oldColor=document.getElementById("myButton").style.backgroundColor;
        document.getElementById("myButton").addEventListener("click",
            function(e){
                window.alert("before"); // step 2
                e.target.style.backgroundColor="red"; //step 3
                document.getElementById("info").innerHTML=eventInfo(e);
                window.alert("after");  // step 4
            },false);  // target phase
        document.getElementById("divOuter").addEventListener("click",
            function(e){
                //window.alert("jestem");
                e.currentTarget.style.backgroundColor="green";
            },false);  // bubbling phase // step 5
        document.getElementById("divInner").addEventListener("click",
            (e)=>{
                e.currentTarget.style.backgroundColor="aqua";
            },true);  // capture phase (step 1)
        document.getElementById("myReset").addEventListener("click",
            ()=>{
                document.getElementById("myButton").style.backgroundColor=oldColor;
                document.getElementById("divOuter").style.backgroundColor="yellow";
                document.getElementById("divInner").style.backgroundColor="pink";
                document.getElementById("info").innerHTML="";
            },false);  // target phase
    </script>
```

eventTraversal.css

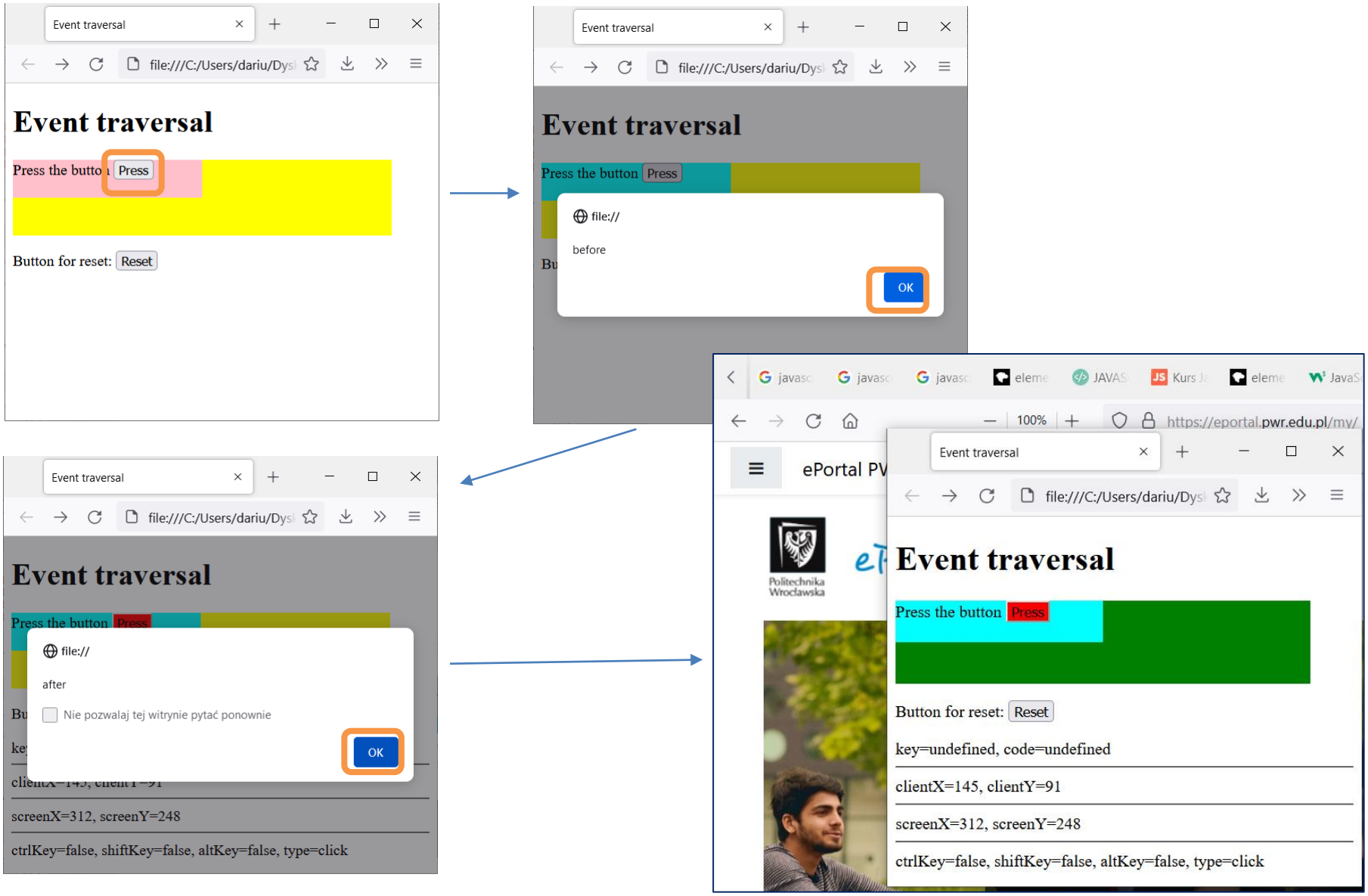
```css
#divOuter{
    width: 400px;
    height: 80px;
    background-color: yellow;
}

#divInner{
    width: 200px;
    height: 40px;
    background-color: pink;
}
```

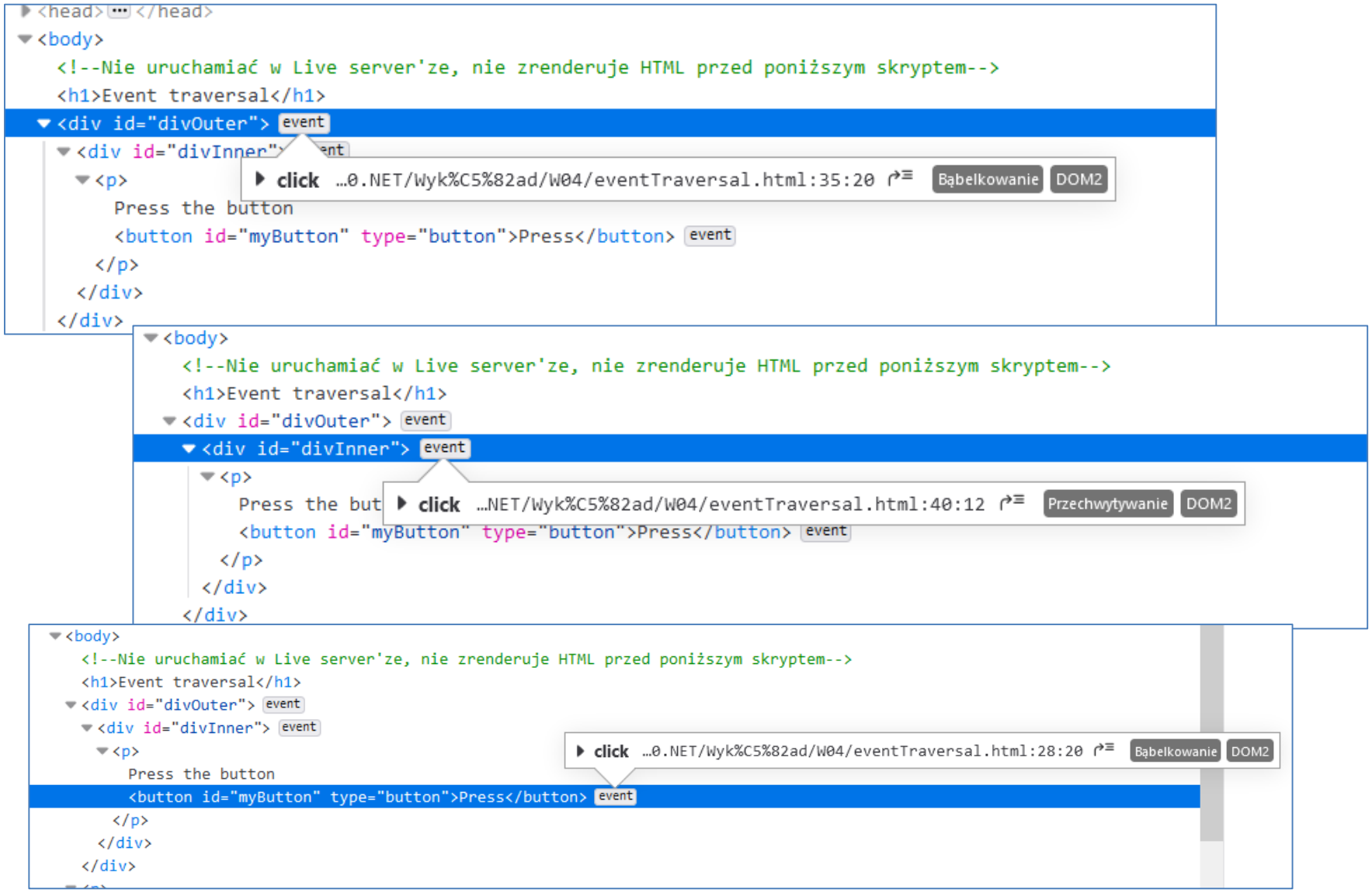
eventTraversal.js

```js
function eventInfo(e){
    return  "key="+e.key+
            ", code="+e.code+" <hr> "+
            "clientX="+e.clientX+
            ", clientY="+e.clientY+" <hr> "+
            "screenX="+e.screenX+
            ", screenY="+e.screenY+" <hr> "+
            "ctrlKey="+e.ctrlKey+
            ", shiftKey="+e.shiftKey+
            ", altKey="+e.altKey+
            ", type="+e.type;
}
```

Działanie


Event traversal
Event traversal
Press the button Press
Button for reset: Reset
file://
before
OK
file://
after
Nie pozwalaj tej witrynie pytać ponownie
OK
screenX=312, screenY=248
ctrlKey=false, shiftKey=false, altKey=false, type=click
ePortal PW
Politechnika Wrocławska
Press the button Press
Button for reset: Reset
key=undefined, code=undefined
clientX=145, clientY=91
screenX=312, screenY=248
ctrlKey=false, shiftKey=false, altKey=false, type=click

Zdarzenia – narzędzie programistyczne



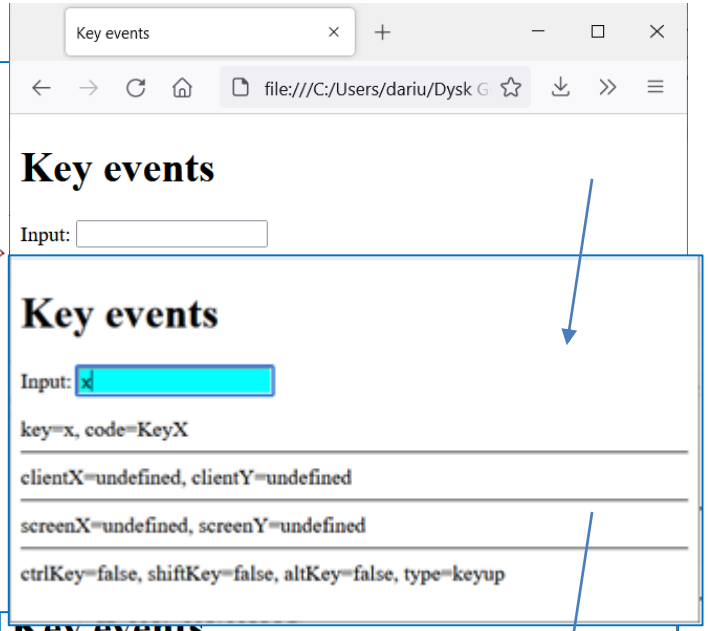
Inne wybrane zdarzenia HTML DOM

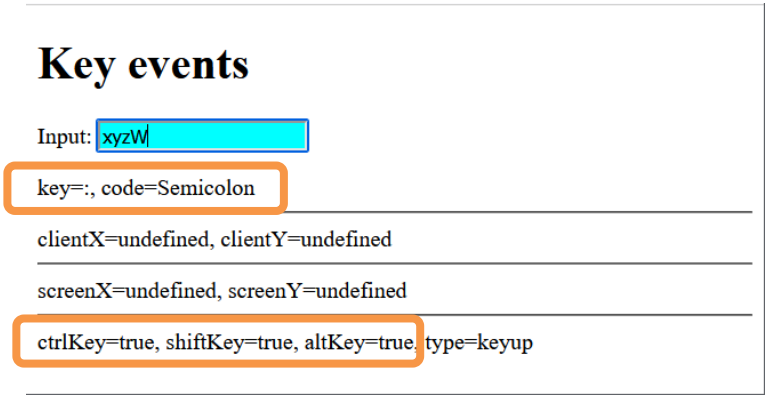
- Zdarzenia:
  - Ogólne: `change`, `invalid`, …
  - W przeglądarce: `resize`, `scroll`, …
  - Klawiatura: `keypress`, `keydown`, `keyup`, `focusin`,`focus`, `focusout`, `blur`, `copy`, `paste`…
  - Myszka: `mouseover`, `mousedown`, `mouseenter`, `mouseleave`,…
  - Inne: `beforeprint`, `submit`, `clear`,…

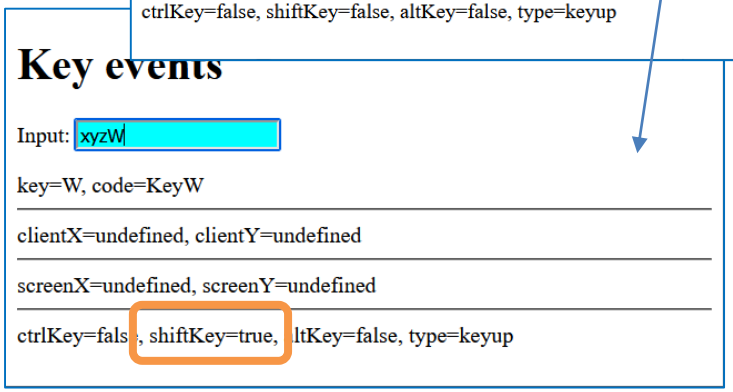
- Przykład:
  - Pole tekstowe, które pokazuje informacje o zdarzeniu w elemencie `p#myInfo`.
    - Wykorzystać funkcję `eventInfo(e)` z pliku `eventTraversal.js`.
  - Pole ma zmieniać kolor na czerwony, gdy liczba znaków w polu jest podzielna przez 3. Jeśli nie to zmiana koloru na „cyan”.
  - Przetestować wciskanie klawiszy Alt, Ctrl, Shift w połączenie z klawiszami znaków drukowalnych

Przykład

```
7       <script src="eventTraversal.js"></script>
8       <title>Key events</title>
9   </head>
10  <body>
11
12      <h1>Key events</h1>
13      <p id="p1">
14          <label>Input: <input id="myInput" type="text" name="name" onkeyup="count3(event)"></label>
15      </p>
16      <p id="info"></p>
17      <script>
18          function count3(e){
19              var count=e.target.value.length;
20              document.getElementById("info").innerHTML=eventInfo(e);
21              var styler=document.getElementById("myInput").style;
22              if(count%3==0)
23                  styler.backgroundColor="red";
24              else
25                  styler.backgroundColor="cyan";
26          }
27      </script>
28  </body>
```







Dołączanie obsługi zdarzeń w kodzie HTML

- Inna możliwość (starsza) to przypisanie funkcji JS do atrybutu elementu. Nazwa atrybutu to przedrostek `on`, po którym następuje nazwa zdarzenia
- Wstawia się atrybut z przedrostkiem `on` i nazwą zdarzenia (np. `onload`, `onclick`, `onfocus` itp.), a jako wartość podaje się kod w języku JavaScript (najczęściej - wywołanie metody).
  - Np. `<body onload="start()">`
- Powoduje to wyrejestrowanie wszystkich wcześniej zarejestrowanych funkcji.
- Można tak dodać tylko jedną metodę.

- Uwaga ogólna: Warto sprawdzać, czy wybrane rozwiązanie (funkcja w JavaScript) nie jest zbyt wolne i skorzystać z innych funkcji bibliotecznych.
  - Języki skryptowe często mają narzut czasowy związany z ciągłą kompilacją kodu, więc oprócz efektywnego algorytmu należy wybrać efektywny sposób jego implementacji.

Inne elementy JS

- Inne możliwości pobierania elementu DOM:
  - `document.getElementsByClassName(nazwaKlasy)`,
  - `document.getElementsByTagName(nazwaTagu)`,
  - …
- Mechanizm wyjątków – blok `try/catch`
- Operator `===`
  - różnica działania w stosunku do `==`
- Automatyczne rzutowanie na inne typy oraz `true/false`, `null` itp. w operatorze `==`
  - (`10 == "10"`)
- Obiekty `Iterable`
  - Posiadają metodę `next()`
- Kolekcje `Map`, `Set`, `Array`.
- Pętle:
  - `for (key in object) { ... }`
  - `for (variable of iterable) {...}` // (ES6 2015)
- Akcesory (ang. accessors) (ES5 2009):
  - gettery/settery (podobnie działają właściwości w języku C#)
- itp.

- jQuery – biblioteka JavaScript ułatwiająca operowanie na modelu DOM
  - Używa selektorów CSS i wprowadza obiekt $ np.:

```
$("button.continue").html("Next Step...")
```

- …