Wypisywanie: innerHTML, innerText, outerHTML, outerText

W JavaScript istnieje kilka sposobów na wypisywanie tekstów (np. wyników obliczeń lub tekstowych podsumowań działania logiki skryptu). Czasem też chcemy obudować takie wypisywane rezultaty otrzymane w JS kodem HTML (np. umieścić w akapicie, w liście wypunktowanej albo w tabeli). Poznamy więc teraz cztery popularne właściwości:

`innerText`, `textContent`, `innerHTML` oraz `outerHTML`. Są to oczywiście właściwości elementów blokowych, w których zazwyczaj umieszczamy wyniki działania skryptu – na przykład w przygotowanym uprzednio, a domyślnie pustym kontenerze typu: `<div class="wyniki"></div>` czy `<p class="wyniki"></p>`.

Wydawać się to może trywialne, bo w 90% przypadków ludzie na egzaminie korzystają z

`innerHTML`, lecz czy na pewno dobrze znamy wszystkie cztery właściwości i różnice pomiędzy nimi? A co, jeśli arkusz wymusi na nas użycie innej właściwości? Poznajmy zatem każdą z nich!

Dostęp do tekstu wewnątrz – innerText

Po pierwsze – gwoli formalności przypomnijmy, iż z użyciem JS możemy swobodnie manipulować (zarówno zmienić, jak i odczytać) pierwotnie istniejącą, zdefiniowaną w dokumencie HTML zawartością znaczników – może to być akapit, div, nagłówek itd.

Załóżmy więc, że istnieje na stronie np. nagłówek

`<h2>`:

```
<h2 class="tytul">Wylosuję dla Ciebie liczbę z przedziału od 1 do 100!</h2>
<button onclick="losujLiczbe()">Wylosuj dla mnie liczbę</button>
```

I w zadaniu egzaminacyjnym wymaga się od nas, żeby po kliknięciu przycisku ten nagłówek zmienił treść (zgodnie z obietnicą zawartą w jego treści) na faktyczną liczbę wylosowaną z zadanego przedziału 1..100. Aby to zrobić najprościej, możemy w JS użyć właściwości

`innerText` (*ang. inner = wewnętrzny, w środku*):

```
function losujLiczbe() {
  // Pobieramy nagłówek na podstawie klasy CSS (stąd kropka)
  var naglowek = document.querySelector(".tytul");
  // Losujemy liczbę z przedziału 1–100 (obiekt Math)
  var losowa = Math.floor(Math.random() * 100) + 1;
  // Podmieniamy tekst nagłówka na wylosowaną liczbę
  naglowek.innerText = "Twoja liczba to: " + losowa + ". Jeszcze raz?";
}
```

Jednak najważniejszym ograniczeniem właściwości

`innerText` jest fakt, iż nie pozwala na interpretację HTML, to znaczy nie możemy obudować wypisywanej treści znacznikami HTML. Rozważmy dodanie do wypisanej liczby jej podkreślenia:

```
function losujLiczbe() {
  // Pobieramy nagłówek na podstawie klasy CSS (stąd kropka)
  var naglowek = document.querySelector(".tytul");
  // Losujemy liczbę z przedziału 1–100 (obiekt Math)
  var losowa = Math.floor(Math.random() * 100) + 1;
  // Podmieniamy tekst nagłówka na wylosowaną liczbę
  naglowek.innerText = "Twoja liczba to: <u>" + losowa + "</u>. Jeszcze raz?";
}
```

Rezultat w przeglądarce (oczywiście dla akurat tej wylosowanej liczby) będzie taki:

Twoja liczba to: <u>45</u>. Jeszcze raz?

Jak widzimy przeglądarka nie interpretuje znacznika

`<u>` tylko pokazuje go jako tekst. I właśnie dlatego istnieje także właściwość `innerHTML` poprawnie obsługująca tagi, którą teraz się zajmijmy.

Tekst, lecz z działającymi tagami – innerHTML

Jeśli zatem chcemy z użyciem JS wygenerować nie tylko tekst, ale także obudować go znacznikami HTML, to zamiast

`innerText` użyjemy właściwości `innerHTML`. Dla tego samego zadania co powyżej, kod JS będzie następujący:

```
function losujLiczbe() {
  // Pobieramy nagłówek na podstawie klasy CSS (stąd kropka)
  var naglowek = document.querySelector(".tytul");
  // Losujemy liczbę z przedziału 1–100 (obiekt Math)
  var losowa = Math.floor(Math.random() * 100) + 1;
  // Podmieniamy tekst nagłówka na wylosowaną liczbę
  naglowek.innerHTML = "Twoja liczba to: <u>" + losowa + "</u>. Jeszcze raz?";
}
```

Teraz rezultat w przeglądarce będzie już zgodny z naszą intencją podkreślenia liczby:

Twoja liczba to: <u>45</u>. Jeszcze raz?

Tym razem tagi HTML są już interpretowane przez przeglądarkę, a nie pokazywane jako tekst. Koniecznie też zwróćmy uwagę na zapis właściwości w JS – jest to

`innerHTML`, a nie często zapisywany błędnie: `innerHtml` – wydawać się to może trywialne, ale to popularna pomyłka!

W tym miejscu wiele osób zapyta o to, po co w ogóle istnieje właściwość

`innerText`, skoro `innerHTML` również umożliwia wypisywanie tekstów, a dodatkowo pozwala (jeśli chcemy) używać znaczników? Oczywiście w grę wchodzi tutaj przede wszystkim kontekst bezpieczeństwa (możliwość używania tagów niesie ze sobą ryzyko), lecz pełną odpowiedź na to pytanie najlepiej przedstawić w postaci tabelarycznej:

Opis potencjalnego działania	innerText	innerHTML
Przekazuje tekst do elementu HTML	Tak	Tak
Pozwala również na tagi HTML w tekście	Nie	Tak
Usuwanie potencjalnych niebezpiecznych tagów, czyli tzw. sanityzacja kodu)	Tak, bo nie obsługujemy tagów, pokazując znaczniki jako zwykły tekst	Nie, bo pozwoli “wykonać” wiele znaczników w przeglądarce

Opis potencjalnego działania	innerText	innerHTML
Szybkość przetworzenia (bo uwzględniamy styl, widoczność)	Nieco wolniejsze	Nieco szybsze

Wiele osób najczęściej używa na egzaminie właśnie właściwości

`innerHTML` do wypisywania tekstów i to nawet jeśli ten tekst nie zawiera znaczników. Raczej nie jest to błąd, a sytuacja typu *overkill* (ang. przesada), lecz my rozważmy użycie `innerText` wszędzie tam, gdzie znaczniki nie są potrzebne. W końcu nie zawsze potrzebujemy wszechstronności – czasem zależy nam najbardziej na bezpieczeństwie i prostocie zapisu.

Alternatywny dostęp do tekstu – textContent

Właściwość

`textContent` jest podobna do `innerText`, natomiast różnica polegała na tym, iż umożliwiała nam ona odczytanie lub ustawienie tekstu w elemencie nie przejmując się widocznością tego kontenera, zdefiniowaną w CSS. Czyli podczas używania `textContent` przeglądarka nie analizowała, czy coś jest aktualnie widoczne na stronie (ustawiona właściwość CSS `display:none` ukrywa przecież obiekt), co jak najbardziej było uwzględniane przy użyciu `innerText`.

A zatem próba odczytania tekstu z ukrytego akapitu poprzez

`innerText` zakończyłaby się niepowodzeniem – w poniższym skrypcie otrzymalibyśmy pustą wartość w drugim alercie:

```
<style> .napis { display:none; } </style>
<p class="napis">Tajemnicza, ukryta na stronie treść akapitu.</p>
<button onclick="pokazZawartosc()">Odczytaj niewidoczną treść</button>
<script>
function pokazZawartosc()
{
    var akapit = document.querySelector(".napis");

    alert(akapit.textContent);
    alert(akapit.innerText);
}
</script>
```

Dlaczego jednak cały czas mówimy o tym w czasie przeszłym? Ponieważ współczesne przeglądarki (np. *Google Chrome, Firefox*) zaktualizowały sposób działania

`innerText` ujednolicając ich zachowanie – oba alerty zwrócą zawartość akapitu. Ta drobna różnica w działaniu obu sposobów jest więc dziś nieaktualna, lecz oczywiście właściwość `textContent` wciąż pozostaje jak najbardziej działającą alternatywą dla `innerText` i możemy jej bezproblemowo także używać.

Właściwość outerHTML – dostęp także do elementu

Kolejna właściwość, czyli

`outerHTML` (*ang. outer = zewnętrzny*) pozwala na dostęp (odczyt, zapis) do całego elementu, a nie tylko jego zawartości! Rozważmy więc w poniższym przykładzie zmianę także samego znacznika, a nie samej jego wewnętrznej zawartości:

```
<p id="tekst">To jest paragraf.</p>
<button onclick="zmienOuterHTML()">Zmień także rodzaj znacznika</button>
```

```
function zmienOuterHTML() {
    var tekst = document.getElementById("tekst");
    tekst.outerHTML = "<h2>Czary mary - teraz to nagłówek!</h2>";
}
```

I tak oto akapit

`<p>` został podmieniony w przeglądarce w JS na nagłówek `<h2>`. Zależnie od arkusza z zadaniem, czasami może się to okazać przydatne na egzaminie INF.03 oraz oczywiście także w praktyce tworzenia realnych stron internetowych, zależnie od naszych intencji.

Interpolacje podczas wypisywania

W JavaScript bardzo często chcemy wypisać wartość jakiejś zmiennej razem z tekstem. Na przykład: użytkownik podał imię, a my chcemy napisać:

`Witaj, Janek!` albo `Podano liczbę: 42`. Właśnie w takich sytuacjach do gry wchodzi tzw. interpolacja łańcuchów znaków – czyli sposób łączenia tekstu z wartością zmiennej – najlepiej gdyby był czytelny, nowoczesny i wygodny.

Klasyczny sposób w JS to oczywiście używany także w tej lekcji operator konkatenacji

`+`:

```
let imie = "Kasia";
console.log("Witaj, " + imie + "!");
```

Zapis działający i w zupełności wystarczający do egzaminu. Natomiast stanie się on mniej nieczytelny, gdy chcemy wstawić do łańcuch kilka zmiennych lub dłuższy tekst. Cudzysłowy, plusy i spacje mieszają się – dla wielu osób łatwo się w tym miejscu pomylić. I dlatego warto poznać nowocześniejszy sposób – interpolację z użyciem zapisu

`${zmienna}`:

```
let imie = "Kasia";
console.log(`Witaj, ${imie}!`);
```

Są to tzw. “szablony tekstowe” (*ang. template strings*), jak widać zapisane w tzw. *backticks* (czyli apostrofach typu

`, które wstawiamy na klawiaturze za pomocą klawisza tyldy – pod Escape). Dzięki takiemu zapisowi zdania wyglądają podobnie jak w języku naturalnym – nie ma tylu spójników – jest to bardzo intuicyjne i czytelne.

Gdy chcemy wstawić kilka wartości – np. imię, nazwisko i wiek – to taki zapis z interpolacją stanie się już nie tylko czytelniejszy, ale także krótszy:

```
let imie = "Kasia";
let nazwisko = "Nowak";
let wiek = 18;
// template strings
console.log(`Użytkownik: ${imie} ${nazwisko}, wiek: ${wiek} lat`);
// klasyczna konkatenacja - zapis dłuższy oraz łatwiej tu o literówkę!
```

```
console.log("Użytkownik: " + imie + " " + nazwisko + ", wiek: " + wiek + " lat");
```

Porównanie czytelności i długości widać gołym okiem. Gdzie możemy używać zapisu ${zmienna} w *template strings*? W codziennych egzaminacyjnych sytuacjach – na przykład:

- `console.log(...)` – informacja w logach konsoli w *Devtools*
- `alert(...)` – tekst w wyskakującym oknie typu alert
- `innerHTML = ...` – wyświetlanie tekstu w div czy akapicie na stronie
- przekazywanie wartości np. adresów URL do zmiennych

Czego potrzebujemy, żeby to nowoczesne wypisywanie działało? Musimy pamiętać o dwóch rzeczach:

- Zamiast zwykłych cudzysłowów `"..."` lub apostrofów prostych `'...'` używamy *backticków* – czyli: `` `...` ``.
- W środku zapisujemy zmienną w formacie: `${nazwaZmiennej}`

Podsumowanie

Na koniec jeszcze jedna uwaga, konieczna w tym miejscu – zwróćmy uwagę, iż akapity, nagłówki czy divy i pojemniki semantyczne (article, section, header, footer, aside) faktycznie posiadają w JS cztery omówione przez nas w tej lekcji właściwości, natomiast co innego pola formularzy (inputy, textarea, przyciski) – aby dostać się do zawartości wewnątrz nich wykorzystujemy właściwość

`value`, czyli wartość w środku pola. Pomieszanie tych właściwości to stosunkowo częsta pomyłka.

Co na pewno warto zapamiętać?

- `innerText` – pozwala na dostęp do tekstu, podobnie jak `textContent` – dawniej co prawda istniała opisana przez nas różnica przy ustawieniu właściwości `display:none` w CSS dla elementu (obiektu) zdefiniowanego w HTML, jednak we współczesnych przeglądarkach działanie obu właściwości zostało ujednolicone
- `innerHTML` – pozwala dodawać znaczniki HTML podczas wypisywania, co może być niebezpieczne
- `outerHTML` – dając dostęp także do samego znacznika, pozwala nawet podmienić cały element (obiekt) na inny
- Interpolacja łańcuchów, czyli zastosowanie "szablonów tekstowych" (*ang. template strings*), zapisanych w tzw. *backticks* `` ` `` poprawia czytelność wypisywania, zwłaszcza dla wielu zmiennych, które normalnie trzeba by konkatenować operatorem `+`:

```
// template strings
console.log(`Użytkownik: ${imie} ${nazwisko}, wiek: ${wiek} lat`);
// klasyczna konkatenacja - zapis dłuższy oraz łatwiej tu o literówkę!
console.log("Użytkownik: " + imie + " " + nazwisko + ", wiek: " + wiek + " lat");
```