Jak podpiąć skrypt JS do pliku HTML?

Zacznijmy od fundamentów – aby skrypt mógł poprawnie wykonać powierzone mu zadania, należy go rzecz jasna poprawnie dołączyć (zainkludować) do dokumentu HTML. Istnieją na dwa podstawowe sposoby podpięcia osadzenia kodu JS w dokumencie HTML, a ponadto mamy do dyspozycji dwa atrybuty

`async` oraz `defer`, które kontrolują sposób wczytywania skryptu przez przeglądarkę. Przyjrzyjmy się temu bliżej.

Skrypt umieszczony w pliku HTML

Najprostsza metoda wywołania kodu JS, to umieszczenie tych linii bezpośrednio w pliku z rozszerzeniem

`*.html` lub `*.htm`, wewnątrz podwójnych tagów `<script>`. Dlaczego jest to znacznik podwójny? Oczywiście z tego powodu, iż przeglądarka musi wiedzieć gdzie zaczyna się skrypt i w którym momencie kończy, czyli kiedy następuje efektywny powrót do znaczników HTML.

```
<script>
    alert("Witaj Świecie!");
    // Kod JavaScript znajdujący się bezpośrednio w pliku HTML
</script>
```

Największą zaletą takiego sposobu jest szybkość wstawienia kodu, gdyż nie musimy stworzyć osobnego pliku

`*.js` ze skryptem. Natomiast w ten sposób “mieszamy” linie HTML i JS w jednym dokumencie, występuje więc brak dobrej separacji kodu, jak również trudność z wielokrotnym użyciem danego skryptu – w końcu kod w zewnętrznym pliku `*.js` można podpiąć do wielu różnych podstron.

Na egzaminie nie stosujmy tego rozwiązania (o ile zadanie w arkuszu nie będzie tego wymagać, a nie raczej nie będzie gdyż byłoby to nielogiczne) – pokażmy, iż rozumiemy potrzebę rozdzielenia kodów źródłowych JS od warstwy HTML i zawsze osadzajmy skrypty z zewnętrznego pliku:

Skrypt osadzony z zewnętrznego pliku

```
<script src="skrypt.js"></script>
```

Tym razem nie wstawiamy kodu wprost do dokumentu HTML, tylko zapewniamy lepsza organizację i separację źródeł. Ponadto współczesne przeglądarki internetowe dokonują często tzw. cache’owania (przechowywania w pamięci podręcznej) skryptów używanych pomiędzy podstronami, co staje się możliwe tylko w przypadku umieszczenia ich właśnie w zewnętrznych plikach.

Warto też wiedzieć, iż w edytorze *Visual Studio Code*, dzięki istniejącemu dodatkowi *Emmet* (podpowiadanie składni) łatwo wstawić podpięcie zewnętrznego skryptu wpisując w kodzie:

`script:src`. Spowoduje to wstawienie podwójnych tagów `<script>` wraz z ustawieniem kursora od razu w wartości atrybutu `src=""` co umożliwia szybkie podanie nazwy skryptu.

Przypomnijmy także, iż w HTML 5 nie trzeba już podawać w tagu

`<script>` dodatkowego atrybutu `type="text/javascript"` znanego z poprzednich wersji standardu. Przeglądarka i tak założy, iż ma do czynienia ze skryptem JS, bo ten język stał się domyślny dla nowych, podpinanych do dokumentu HTML skryptów.

Zazwyczaj rekomenduje się też podpinanie skryptów pod koniec sekcji

`<body>` (zamiast w `<head>`), bo wówczas przeglądarka zdąży wczytać i werenderować w zasadzie całą treść podstrony, a dopiero później zajmie się pobieraniem skryptów JS. A te mogą przecież posiadać wiele linii kodu, czytaj: swoje "ważyć". W praktyce webowej istnieją oczywiście wyjątki od tej reguły, np. skrypty analityki sieciowej lub reklamy od *Google*, które należy umieścić w sekcji `<head>`, lecz oczywiście z tym raczej nie będziemy mieć do czynienia na egzaminie INF.03 (gdzie dostępu internetu nie ma).

Inną metodą zmiany sposobu wczytywania skryptów jest także wykorzystanie dwóch atrybutów HTML:

`defer` oraz `async`. Wyjaśnijmy teraz ich zastosowanie.

Atrybut async – wykonywanie skryptu asynchronicznie

Domyślnie przeglądarka, kiedy napotka plik ze skryptem zatrzymuje dalsze wczytywanie strony, dopóki pliku nie załaduje i nie wykona tego skryptu. W praktyce oznacza to, że jeżeli skrypt jest "ciężki" albo ładowany z zewnętrznej domeny, to użytkownik może odczuwać opóźnienie w renderowaniu widocznej na ekranie zawartości strony (teksty, obrazki, tabele umieszczone pod skryptem).

Z pomocą przychodzi nam tutaj atrybut

`async`, który pozwala przyspieszyć ładowanie i wykonywanie skryptów niejako w tle (ładnie powiemy: asynchronicznie), nie zatrzymując dalszego wczytywania strony HTML:

```
<script src="skrypt.js" async></script>
```

Kiedy dodamy atrybut

`async` (jak widać wystarczy sama obecność atrybutu, jego wartość nie jest potrzebna) do znacznika otwierającego `<script>` przeglądarka zachowa się wówczas inaczej niż domyślnie – to znaczy:

1. Po napotkaniu w dokumencie podpięcia skryptu rozpoczyna się jego pobieranie w tle, równolegle z dalszym parsowaniem HTML. A zatem przeglądarka nie czeka już na zakończenie ładowania pliku `skrypt.js`, tylko wczytuje i pokazuje na płótnie przeglądarki dalsze elementy podstrony,
2. Gdy tylko skrypt zostanie w całości pobrany, to zostanie wykonany – i to nawet jeśli HTML nie został jeszcze do końca przetworzony (czyli nie jest to oczekiwanie na załadowanie całej zawartości HTML).

Oczywiście takie szybsze ładowanie dokumentu jest świetne dla skryptów niezależnych od elementów występujących na stronie (np. analityka sieciowa). Gorzej, jeśli nasz skrypt ma za zadanie modyfikować obiekty zdefiniowane w DOM (czyli w strukturze zdefiniowanej w tym późniejszym HTML). Jeśli np. nasz skrypt zawiera uchwyt elementu DOM, np.

`document.getElementById()` czy `document.querySelector()`, to może się okazać, że te elementy jeszcze nie istnieją w DOM podczas asynchronicznego ładowania, bo zwyczajnie ten fragment HTML nie został jeszcze w pełni przetworzony! Używajmy więc tego atrybutu świadomie.

Polecam zobaczyć tutorial video na ten temat, znajdujący się na naszym kanale YouTube: Atrybuty async oraz defer.

Atrybut defer – wykonywanie skryptu po załadowaniu strony

Jeśli zaś dodamy do tagu

`<script>` drugi atrybut modyfikujący o nazwie `defer` (*ang. odroczyć, odłożyć w czasie*) to przeglądarka zachowa się jeszcze nieco inaczej:

1. Po napotkaniu w dokumencie podpięcia skryptu (analogicznie jak to ma miejsce dla

`async`) rozpocznie jego pobieranie w tle, czyli równolegle z dalszym parsowaniem HTML.

2. Jednak gdy skrypt zostanie w całości pobrany, to **nie zostanie od razu wykonany** – przeglądarka uruchomi skrypt dopiero wtedy, gdy **cały dokument HTML** zostanie wczytany i przetworzony.

Oczywiście gwoli formalności pokażmy także sposób użycia atrybutu – tak samo jak poprzednio, nie posiada on wartości:

```
<script src="skrypt.js" defer></script>
```

Dzięki `defer` zyskujemy gwarancję, iż np. uchwycony przez nas w kodzie JS element (obiekt) DOM na pewno istnieje na etapie "odpalenia" skryptu w przeglądarce – możemy więc bezpiecznie dokonywać na nim różnych modyfikacji.

Gdzie zatem najlepiej umieścić skrypt zewnętrzny?

Podsumujmy nieco wiedzę z tej lekcji, najlepiej w postaci czytelnej tabeli:

Które osadzenie?	Jak to działa w praktyce?	Kiedy zastosować?
W sekcji `<head>`, lecz dodatkowo istnieje atrybut `async` w tagu `<script>`	Skrypt pobiera się wraz z trwającym parsowaniem HTML i wykona się nawet przed dokończeniem renderowania DOM	Dla niezależnych skryptów (statystyki, reklamy *Google*)
Pod koniec sekcji `<body>`	Wczytywanie skryptu już po załadowaniu większości elementów HTML	Kiedy skrypt modyfikuje DOM, jest to najbardziej klasyczne podpięcie kodu JS do dokumentu HTML
W sekcji `<head>`, lecz dodatkowo istnieje atrybut `defer` w tagu `<script>`	Pobieranie skryptu w czasie trwania parsowania HTML, lecz dopiero wykonanie po załadowaniu całego dokumentu HTML	Dla skryptów operujących na elementach DOM, czyli np. kiedy wewnątrz kodu JS znajdują się uchwyty jakichś obiektów zdefiniowanych w HTML

Warto znać te podstawowe sposoby dołączania skryptów do HTML – to kluczowa wiedza w pracy z rzeczywistymi witrynami, lecz oczywiście także na egzaminie INF.03 nie stosujmy złej praktyki braku “wyjęcia” kodu JS do zewnętrznego pliku ze skryptem oraz użycia atrybutu

`async` w skrypcie manipulującym elementami DOM. Dokonujmy też domyślnie podpięcia plików skryptów pod koniec sekcji `<body>`, zamiast wewnątrz `<head>` – chodzi o pokazanie, iż rozumiemy najlepsze praktyki, pomimo iż skrypt na egzaminie zapewne nie będzie “ważyć” zbyt dużo.

Egzaminacyjnym (i nie tylko) klasykiem jest zatem podpięcie skryptu pod koniec sekcji `<body>` – jak to mówimy w języku angielskim: *usually you can’t go wrong with this one* – zazwyczaj nie można się pomylić używając tego sposobu.