Завантаження документа **HTML** та зовнішніх ресурсів. Завантаження скриптів.

Події завантаження документа та зовнішніх ресурсів.

***DOMContentLoaded*** *генерується на* ***document****, коли* ***DOM*** *готовий (коли HTML розмітка прочитано).* Ми можемо застосувати **JavaScript** до елементів на цьому етапі

Приклад:

**document.addEventListener(‘DOMContentLoaded’,func);**

*Зображення* ***<img>****, зовнішні стилі, скрипти та інші ресурси надають події* ***load*** *та* ***error*** *для відстеження завантаження.*

Приклад:

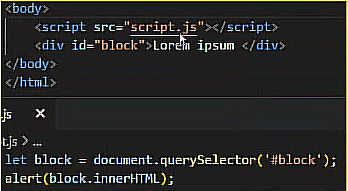
**img.addEventListener(‘load’,func);**

**script.addEventListener(‘load’,func);**

*Якщо відвідувач збирається піти зі сторінки або закрити вікно, обробник* ***beforeunload*** *попросить додаткове підтвердження.*

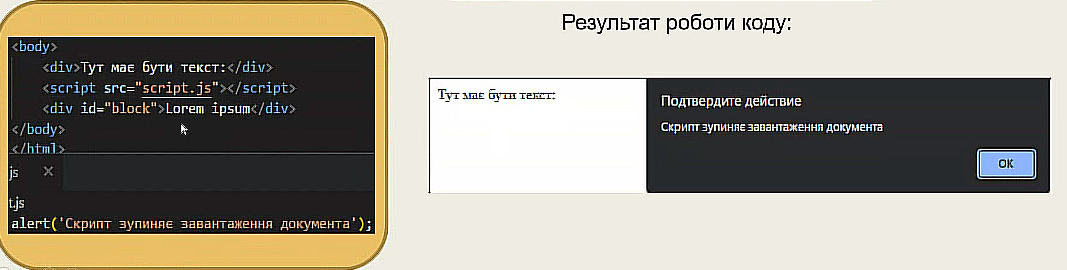
Якщо ми скасовуємо цю подію, то браузер запитає відвідувача, чи впевнений він.

Завантаження скриптів.

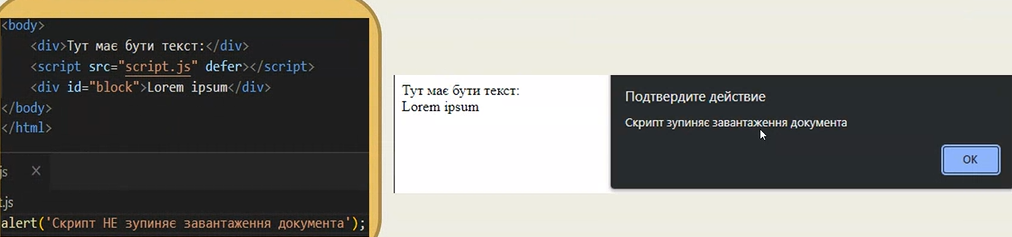
Проблеми цього коду:

*Скрипт не бачать* ***DOM****-елементи нижче за себе*, тому до них не можна звернутися.

Скрипт “блокує” сторінку. Користувачі не бачать вміст, доки скрипт не виконається.

*Атрибут* ***defer*** *повідомляє браузеру, що він повинен продовжувати обробляти сторінку і завантажувати скрипт у фоновому режимі, а потів запустити цей скрипт, коли дерево* ***DOM*** *буде повністю побудовано.*

* Скрипти з **defer** ніколи не блокують сторінку.
* Скрипти з **defer** завжди виконуються, коли дерево **DOM** готове.

Пам’ятка:

*Атрибут* ***defer*** *буде проігнорований, якщо тег* ***<script>*** *не має* ***src****.*

*Атрибут* ***async*** *означає, що скрипт є абсолютно незалежним:*

* *Сторінка не чекає асинхронних скриптів, вміст обробляється та відображається.*
* *Асинхронні скрипти не чекають один на одного.*
* *Найменший скрипт* ***shortScript.js*** *йде другим, але швидше за все завантажиться раніше* ***google-analytics.js****, тому і запуститься першим.*

Тому скрипти виконуються в порядку завантаження.

