|  |
| --- |
| **BOM (Browser Object Model)** - рендерира обекти като прозорец, документ, екран, история, местоположение, навигатор и т.н.  Прозорецът е глобалният обект. |
| **DOM** – съдържание, презентация и поведение (HTML, CSS и JavaScript). Мостът между програмите на JavaScript и съдържанието, което се изобразява в браузъра, е API на обектния модел на документ (DOM). Това е дървесна структура на елементи. Най-често манипулираме document.body, тъй като това е единствената част от документа, която представя визуални компоненти. DOM определя HTML елементите като обекти, които имат: свойства, методи, събития. Елементите са вложени един в друг и създават йерархия. DOM методите са действията, които извършваме върху HTML елементи (добавяне, изтриване, скриване). DOM Properties са стойностите на елементите, които можем да зададем или променим. DIV и paragraphs имат свойство .innerText, а полетата (inputs) за въвеждане имат свойство value (стойност). Тези две свойства не могат да се прилагат заедно и едновременно. |
| **Какво е DOM API?** API е съкращението от **Application Programming Interface**, което е софтуерен посредник, който позволява на две приложения да говорят помежду си. Всеки път, когато използвате приложение като Facebook, изпратите незабавно съобщение или проверите времето на телефона си, използвате API. DOM API се състои от интерфейс, изграден за програмна комуникация с DOM в браузъра. DOM е начинът, по който се моделира съдържанието на дадена страница в браузъра, а DOM API е начинът, по който браузърът осигурява програмен достъп до това съдържание.  => Представете си, че седите на маса в ресторант с меню за избор, от което можете да поръчате. Кухнята е частта от „системата“, която ще подготви вашата поръчка. Това, което липсва, е критичната връзка за предаване на вашата поръчка до кухнята и доставка на храната ви обратно на вашата маса. Тук идва сервитьорът или API. Сервитьорът е пратеникът - или API - който взема вашата заявка или поръчка и казва на кухнята - системата - какво да прави. Тогава сервитьорът ви връща отговора обратно.  HTML елементите са JS обекти и могат да се обработват като такива:  1. Изберете обект, за да получите неговата референция;  2. Променете свойствата му.  HTML атрибут! = DOM свойство, но почти няма разлика между тях, тъй като тяхното използване води до един и същ резултат.  DOM API може да прави:  • промяна на съдържанието и структурата на HTML елементи,  • промяна на стил и свойства,  • чете потребителски вход и реагира на събития - задейства се, когато има някакво взаимодействие с потребителя,   * • създаване и премахване на елементи |
| **NodeList** е колекция от елементи на документа. NodeList и **HTML колекция** са почти едно и също нещо. И **HTMLCollection е** обект, и NodeList е обект, и двете са подобни на масив, но не са масиви. И двете имат свойство за дължина (length), определящо броя на елементите в списъка (колекцията).  Спецификите на Judge ни задължават да използваме **Array.from ()**, за да преобразуваме колекцията в масив.  **NodeList** може да съдържа всеки тип елемент, включително бели пространства и текст. Има итериращи методи. Това е или жива, или статична колекция. Може да се индексира. Може да се преобразува в масив.  **HTMLCollection** съдържа само елементи на HTML. Той има итериращи методи, но има и метод namedItem. Това е жива колекция. Може да се индексира. Може да се преобразува в масив. |
| **Промяна на HTML съдържание**  **- innerHTML - за достъп до суров HTML. Внимание: Промяната на innerHTML или textContent ще премахне всички дъщерни елементи.**  **- textContent - свойството на интерфейса на Node представлява текстовото съдържание на елемента и неговите деца.**  **Забележка: textContent и HTMLElement.innerText лесно се бъркат, но двете свойства са различни по важни начини. Element.innerHTML връща HTML, както показва името му. Понякога хората използват innerHTML за извличане или писане на текст вътре в елемент, но textContent има по-добра производителност, тъй като стойността му не се анализира като HTML. Освен това използването на textContent може да предотврати XSS хакерски атаки.**  **- value - стойностите на inout елементи са низови (string) свойства върху тях**  **- style - съдържанието може да бъде скрито чрез манипулиране на дисплея -> element.style.display = “none” / block, inline или празен низ ‘’ ’’;** |
| **Хващане на HTML елементи** - тези методи връщат препратка към елемента, който може да бъде манипулиран от JS  **По ID** - getElementById() - Атрибутът ID трябва да е уникален на страницата, не е добра практика id да се повтаря  **По име на клас** - getElementsByClassName() – използва се за стилизиране и за по-лесно хващане на елементи. Връща жива HTMLCollection – тя също си променя състоянието, след като променим елементи на страницата!  **По име на таг** - getElementsByTagName()- указва типа на елемента (div, p, a ...).  Връща жива HTMLCollection.  **Чрез CSS селектор:**  - querySelector() - избира първия съвпадащ елемент  - querySelectorAll() - избира всички съвпадащи елементи и връща статичен NodeList (не като жива HTMLCollection)  Примери: #id, .className, input[name = [‘’login ”] |
| **Родители и деца елементи**  Всеки DOM елемент има родител, който може да бъде достъпен чрез свойството .parentElement или .parentNode. Децата също могат да бъдат достъпни със свойството .children. |