JavaScript to obiektowy język programowania skryptowego, służący do tworzenia **dynamicznych** stron internetowych. Wzbogaca funkcjonalność stron **statycznych** (czyli samego HTMLa i CSSa) o pewne elementy interaktywne. Przykładowo pozwala zmienić styl elementów strony po kliknięciu na nie, obliczyć wartość na podstawie tego, co poda użytkownik itd. Skrypty wykonywane w JS realizują się po stronie użytkownika, na tzw. **front-endzie**. Wiąże się z tym fakt, że użytkownik nie musi przeładowywać strony, aby zobaczyć zmiany, jakie wprowadza skrypt. **Program** tworzony w JS będziemy nazywać **skryptem**. Skrypt może składać się z wielu **funkcji**, czyli takich kawałków kodu, które można wykonać wiele razy, z różnymi **parametrami** wejściowymi.

Skrypt JS możemy umieścić na stronie przy pomocy znacznika HTML na dwa sposoby (podobnie jak z CSSem):

- **Lokalnie**: <script> treść skryptu </script>
- **Zewnętrznie**: <script src="ścieżka do pliku .js"></script>

Skrypty JS operują na tzw. **obiektach**, czyli pewnych bytach, posiadających opisujące je **atrybuty**, które z kolei przyjmują pewne **wartości**. Przykładowo, każdy telewizor na świecie może zostać przedstawiony jako obiekt o nazwie „telewizor". Jego przykładowymi atrybutami będą: przekątna ekranu w calach, marka producenta, cena wyrażona w złotówkach, albo czy posiada funkcjonalność „smart". Wartościami tych atrybutów mogą być: „60", „Samsung", „2999.99", „tak".

Obiekt: telewizor	
Atrybuty	**Wartości**
Przekątna (w calach)	60
Marka	„Samsung"
Cena (w złotówkach)	2999.99
Czy jest „smart"?	Tak

Wartości tych obiektów są zapisane jako dane. Każda dana posiada **typ** – jakiego rodzaju jest to informacja. Wyróżniamy m.in. typy danych:

- **Całkowitoliczbowe** – integer – 1, 2, 3, 4, 5…
- **Zmiennoprzecinkowe** – float point – 1.23, 21.33333…
- **Tekstowe** – string – „jabłko"
- **Bool'owe** – bool – 1/0, tak/nie, prawda/fałsz

Ważne! JS sam określi jakiego typu jest dana, na podstawie tego, co wpiszemy. Zapis 12 oraz „12" nie będą równoznaczne! Wartości możemy ze sobą porównywać, służą do tego operatory:

- == - czy są równe
- >= - czy lewa wartość jest większa/równa prawej
- > - czy lewa wartość jest większa od prawej
- < - czy lewa wartość jest mniejsza od prawej
- <= - czy lewa wartość jest mniejsza/równa prawej
- != - czy lew wartość jest **różna** od prawej

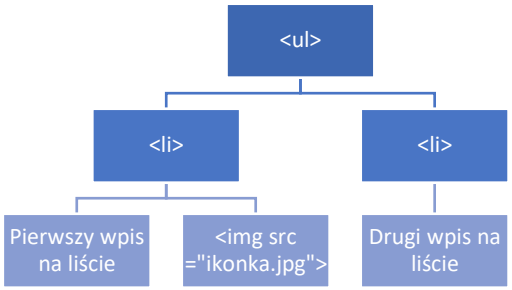
Porównania z jednym znakiem równości to **porównania miękkie**, czyli takie, które patrzą tylko na wartość danej, a nie na jej typ (stąd 2 == 2 to prawda, ale 2 == „2" to również prawda). Aby sprawdzić także typ danej w porównaniu, należy użyć zapisu z dwoma znakami równości – są to **porównania twarde** (czyli 2 == „2" to prawda, ale 2 === „2" już nie).

Każdy znacznik HTML (każdy div, header, main, img, p, itd.) na stronie tworzy nowy obiekt. Dwa podstawowe atrybuty, jakie może posiąść każdy z nich to **id** oraz **class** – często używane w CSS jako selektory (wskazania, który element ma podlegać jakim regułom – patrz poprzednia ściągawka). Możemy użyć tych atrybutów w JS, aby wskazać skryptowi, co ma zrobić z jakim elementem.

Wszystkie znaczniki i elementy znajdujące się na stronie internetowej można przypisać do zmiennych w skrypcie i nimi manipulować. Jest to możliwe dzięki modelowi **DOM (Document Object Model)**, czyli pomysłowi na przedstawienie każdego elementu strony jako obiekt. Wszystkie obiekty na stronie tworzą powiązane ze sobą **węzły nadrzędne i podrzędne** (obiekty-rodzice i obiekty-dzieci). Jeżeli w ramach jednego węzła nadrzędnego jest kilka węzłów podrzędnych, to te węzły stojące obok siebie są nazywane **równorzędnymi** (obiekt-rodzeństwo). Przykład:

```
<ul>
        <li>Pierwszy wpis na liście <img src=”ikonka.jpg”></li>
        <li>Drugi wpis na liście</li>
</ul>
```

Elementem nadrzędnym jest <ul>. Jego elementami podrzędnymi są <li>. Oba <li> są dla siebie równorzędne.
Elementami podrzędnymi dla pierwszego <li> jest jego tekst wewnętrzny oraz obrazek „ikonka.jpg”.
Elementem nadrzędnym dla tekstu wewnętrznego pierwszego <li> oraz obrazka „ikonka.jpg” jest właśnie to <li>.
Tekst wewnętrzny pierwszego <li> oraz obrazek „ikonka.jpg” są dla siebie równorzędne .
Teksty wewnętrzne pierwszego i drugiego <li> **nie** są dla siebie równorzędne, ponieważ znajdują się w osobnych elementach.



Programowanie w JS (jak i w każdym innym języku programowania) wymaga zrozumienia **podejścia algorytmicznego (algorytmiki)**, czyli sposobu jak przedstawić problem tak, aby był zrozumiały m.in. dla komputera. Musimy zapisać kolejne czynności, jakie algorytm ma wykonać – często przywoływanym przykładem algorytmu jest „pieczenie ciasta”. Potrzebna jest nam lista kroków, jakie trzeba po kolei zrobić, aby zawsze zrobić takie samo (tak samo dobre) ciasto.

Algorytmy operują na takich pojęciach jak:

- **Zdarzenia początkowe i końcowe** – pewne wydarzenia, które mają **rozpocząć**, bądź **zakończyć** działanie algorytmu (np. naciśnięcie przycisku może być zdarzeniem startującym, a wyliczenie jakiejś wartości może prowadzić do końca). Na diagramach są oznaczone kształtem **owalu**
- **Stała** – taka wartość w pamięci, która **nie chcemy, aby uległa zmianie** w trakcie działania programu (szuflada, w której zawsze będę miał tę samą skarpetę i nie mogę jej zamienić nigdy na żadną inną)
- **Zmienna** – taka wartość w pamięci, która **będzie się zmieniała** (szuflada, do której mogę wkładać różne skarpety, ale na raz może być w tej jednej szufladzie tylko jedna sztuka)
- **Tablica** – zmienna, która może przyjmować **wiele** wartości (szuflada z wieloma skarpetami (a nawet szuflada z kilkoma mniejszymi szufladami w środku! – **tablica wielowymiarowa**)). Tworzenie nowych stałych i zmiennych nazywamy **deklarowaniem zmiennych**. Każda wartość wpisana do tablicy zajmuje jakieś ponumerowane miejsce, nazywane **indeksem**. Indeksowanie (numerowanie) tych miejsc rozpoczyna się od wartości 0. To znaczy, że jeżeli tablica ma w sobie 3 elementy (jej **długość** jest równa 3), to zajmą one miejsca o indeksach 0, 1 i 2.
- **Warunki** – podjęcie **decyzji**, którą część kodu ma wykonać skrypt, w zależności od jakiejś wartości jakiejś zmiennej (np. jeżeli reszta z dzielenia liczby X na 2 jest równa 1, wyświetl napis „nieparzysta”). Na diagramach są oznaczone kształtem **rombu**
- **Pętle** – takie fragmenty kodu, **które mają wykonać się zadaną liczbę razy** pod rząd (np. zamiast pisać 100 razy linijkę kodu wyświetlającą liczbę 1, 2, 3, 4,… 100, mogę napisać pętlę, która z każdym swoim wykonaniem będzie wyświetlała liczbę o jeden większą od poprzedniej)
- Kroki – kolejny „etap” programu do wykonania, kolejna linijka. Na diagramie mają kształt **prostokąta.**
- Interakcja z użytkownikiem – miejsce w algorytmie, gdzie albo prosimy użytkownika o podanie jakiejś wartości, albo to algorytm ma jakąś wartość zwrócić. Na diagramie są oznaczane **równoległobokami**

JS operuje na pewnych gotowych funkcjach i poleceniach do wykonania; są one nazywane **metodami.** Metody są wywoływane przez **zdarzenia,** np. naciśnięcie przycisku, spełnienie warunku, itd. Podstawowe polecenia i słowa kluczowe związane z JS:

- console.log() – wyświetla treść w konsoli przeglądarki
  console.log(„Hello World!”);

- let – tworzy nową zmienną

  let x = 2;

  Wartości zmiennych można do siebie dodawać, odejmować itd. np. let z = y + x;

  Możemy także zaktualizować wartość zmiennej, wykorzystując jej obecną wartość i jeszcze jakąś inną, np. let x = x + y – weź wartość zmiennej x, dodaj do niej y, wynik dodawania przypisz ponownie do x (innymi słowy „powiększ x o y”. (Alternatywny zapis: x += y). Tak samo z każdym innym działaniem.
- var – również tworzy nową zmienną, ale jest to deklaracja starsza i bardziej podatna na „nasze” błędy (np. nadpisanie wartości)

  var y = 3;
- const – tworzy nową stałą

  const tekst = „jabłko”
- if() – warunek „jeżeli”, powinien być sparowany z zapisem else, czyli „w przeciwnym przypadku”. Istnieje także polecenie else if, czyli kolejny warunek do sprawdzenia. Warunki można łączyć ze sobą zapisem && (oraz – wszystkie warunki muszą być spełnione) albo || (lub – minimum jeden warunek musi być spełniony)

```
if (A>B) {
   console.log(„Większa liczba to:” + A);
}
else if (A == B) {
   console.log(A + “oraz” + B + “są równe”);
}
else {
   console.log(„Większa liczba to:” + B);
}
```

- alert() – wyświetla użytkownikowi komunikat z treścią

  alert(„Dzień dobry”)
- prompt() – wyświetla użytkownikowi komunikat z treścią, a także pole na wpisanie jakiejś odpowiedzi. Ta odpowiedź jest przekazywana jako wartość typu tekstowego. Pobrana wartość powinna zostać przypisana do zmiennej

  prompt(„Podaj liczbę A”);
- parseInt() – zamienia typ danej wartości na całkowitoliczbowy

```
let x = prompt(“Podaj liczbę X:”)
let xJakoLiczba = parseInt(x);
```

  albo szybciej

  let x = parseInt(prompt(„Podaj liczbę X:”)
- parseFloat() – zamienia typ danej wartości na zmiennoprzecinkowy (ułamki w JS zapisuje się ze znakiem kropki, a nie przecinka (np. 1.5, a nie 1,5)

  składnia taka sama jak wyżej
- for() – pętla składająca się z trzech elementów: zmiennej sterującej (pełni rolę „licznika”, na którym wykonaniu pętli jesteśmy); warunku działania (dopóki ten warunek jest spełniony, należy wykonać kod w pętli); zmiany zmiennej sterującej (po każdym wykonaniu kodu w pętli powinniśmy zmienić wartość zmiennej sterującej, aby odliczyć, ile jeszcze zostało nam wykonań)

```
for (i = 0; i < 10; i++) {
  console.log(i);
}
```

  Ta pętla wykona się 10 razy (dla i = 0, i = 1, i = 2,…, i = 9, ale dla i = 10 już nie). Na początku ustala się startową wartość zmiennej sterującej. Z każdym wykonaniem pętli najpierw sprawdza się warunek, potem wykonuje kod, potem zmienia zmienną sterującą i znów sprawdza warunek

- document.getElementById() – wyszukuje w dokumencie HTML obiekt o zadanym ID

  document.getElementById(„pudelko”);
- document.getElementsByClassName() – wyszukuje w dokumencie HTML obiekty posiadające zadaną klasę

  document.getElementsByClassName(„gorne”);

- document.getElementsByTagName() – wyszukuje w dokumencie HTML obiekty o zadanym rodzaju (np. wszystkie elementy div, wszystkie elementy td…)
  document.getElementsByTagName(„li”);
- document.createElement() – tworzy nowy element dokumentu HTML o zadanym rodzaju (np. nowy div)
  document.createElement(„img”);
- document.querySelector() – wyszukuje pierwszy w kolejności element, spełniający warunek zapytania
  document.querySelector(„p.zielone”) – znajdź pierwszy element p z klasą zielone
  document.querySelector(„td#pierwszy”) – znajdź pierwszy element td z id pierwszy
- document.querySelectorAll() – wyszukuje wszystkie elementy spełniające warunek zapytania i zapisuje je w tablicy
  document.querySelectorAll(„li.czerwone”) – znajdź wszystkie elementy li z klasą czerwone
- obiektRodzic.appendChild(obiektDziecko) – znajduje pewien element-rodzic, do wnętrza którego ma zostać podłączony element-dziecko, czyli element mu podrzędny
  duzePudelko.appendChild(nowyObrazek);
- let tablica1 = [2, 3, 4, 5] – utworzy nową zmienną tablicową o nazwie tablica1 i umieści w niej wartości 2, 3, 4, 5
- for (i=0; i<tablica1.length; i++) {
  console.log(tablica1[i]); - pętla for, która wykona się tyle razy, ile jest wartości w zmiennej tablicowej tablica1 (tablica1.length przyjmuje wartość równą długości (liczbie elementów) tablicy). Pętla ma wyświetlić w konsoli przeglądarki i-ty w kolejności element tejże tablicy, gdzie i na początku wynosi 0, a potem rośnie o jeden (zapis i++ to **inkrementacja**, czyli zwiększenie o jeden. Zapis i—to **dekrementacja**, czyli zmniejszenie o jeden).
  }

- let box = document.getElementById(„pojemnik”) – utworzy nową zmienną o nazwie box i przypisze do niej cały element o ID = „pojemnik”, razem z jego zawartością
- box.id = „pudelko” – ustali ID obiektu przypisanego do zmiennej box na „pudelko”
- box.classList.add(„duze”) – doda do listy klas obiektu box klasę o nazwie „duze”
- box.classList.remove(„male”) – usunie z listy klas obiektu box klasę o nazwie „male”
- box.classList.contains(„okragle”) – zadaje pytanie, czy obiekt box posiada na liście klas klasę „okragle”? Można to zastosować w poleceniu if().
- box.style.width = „20px” – ustali szerokość obiektu box na 20 pikseli;
- box.style.height = „30px” – ustali wysokość obiektu box na 30 pikseli;
- box.style.backgroundColor = „blue” – ustali kolor tła obiektu box na niebieski;
  Atrybut .style pozwala na zastosowanie wszystkich reguł CSS w JS. Trzeba tylko pamiętać o zastosowaniu stylu camelCase (czyliKażdeNoweSłowoZWielkiejLitery) w przeciwieństwie do używanego w CSS stylu snake-case (czyli-każde-nowe-słowo-po-myślniku).
- box.innerText = „Siemka” – ustali tekst wewnętrzny obiektu box (to, co jest napisane wewnątrz tego bloku) na napis „Siemka” (tak, jakbyśmy dodali ten tekst w HTMLu, ale bez żadnego znacznika typu <p>)
- box.innerHTML = „<h2>Jestem nagłówkiem</h2>” – ustali wewnętrzną zawartość hipertekstową bloku box na nagłówek drugiego stopnia z napisem „Jestem nagłówkiem” (tak, jakbyśmy dodali ten tekst w HTMLu, z użyciem odpowiedniego znacznika)
- let nowyObrazek = document.createElement(„img”) – utworzy nowy obiekt HTML typu obrazkowego, przypisze go do zmiennej nowyObrazek, ale jeszcze nigdzie go nie umieści na stronie
- nowyObrazek.src = „img/zdjecie1.jpg” – ustali źródło dla obiektu obrazkowego nowyObrazek na plik znajdujący się w folderze img o nazwie zdjecie1.jpg
- box.appendChild(nowyObrazek) – umieści element obrazkowy nowyObrazek wewnątrz obiektu box. Obiekt nowyObrazek stanie się dzieckiem obiektu box, a obiekt box stanie się rodzicem obiektu nowyObrazek
- let obrazki = [] – utworzy pustą zmienną tablicową o nazwie obrazki;
- obrazki.push(„img/zdj2.jpg”, „img/zdj3.jpg”) – doda (dopchnie na pierwsze z brzegu wolne miejsce) do zmiennej tablicowej obrazki dwie wartości tekstowe: img/zdj2.jpg oraz img/zdj3.jpg
- let kolejnyObrazek = document.createElement(„img”) – utworzy kolejny obiekt obrazkowy i przypisze go do zmiennej kolejnyObrazek
- kolejnyObrazek.src = obrazki[0] – jako źródło obiektu obrazkowego kolejnyObrazek ustali wartość pobraną z tablicy obrazki, która znajduje się na miejscu o indeksie równym 0 (czyli pierwszą w kolejności)