JavaScript

Jeszcze raz, od początku

Definicja

- JavaScript to **skryptowy język programowania**, służący do pisania skryptów, które mają wzbogacić funkcjonalność stron internetowych (zmiana strony ze statycznej, w dynamiczną).
- JS opiera się na manipulacji **obiektami**, którymi mogą być wszystkie elementy na stronie (każdy nagłówek, akapit, obrazek, blok itd.)
- Każdy **obiekt** posiada pewne **atrybuty**, czyli cechy, które go opisują (np. wysokość, szerokość, kolor, ID, itd.)
- Te **atrybuty** przyjmują określone **wartości**, które mówią konkretnie o danej cesze (np. **wysokość** = 200 px, **kolor tła** = niebieski)

Obiekt: header
Atrybut: background-image
Wartość: „baner.png”

Obiekt: nav
Atrybut: display
Wartość: inline-block

Strona główna | Historia turnieju | Uczestnicy | Galeria

Nasz turniej z roku na rok cieszy się coraz większym powodzeniem. Uczestników przybywa rokrocznie, a pula nagród rośnie jak ciasto na pierogi i apetyt naszych łakomych zawodników.

Ilość uczestników zawodów w kolejnych latach:

2006. - 123,
2007. - 244,
2008. - 592,
2009. - 712,
2010. - 1022 - w tym roku przebiliśmy tysiąc!
2011. - 1432,
2012. - 2452,
2013. - 2625 - tegoroczny czempion zdublował wynik swojego konkurenta
2014. - 4020,
2015. - 4222,
2016. - 5632,
2017. - 7620 - oficjalnym sponsorem turnieju zostaje wójt Pcimia Dolnego
2018. - 10023 - ponad dziesięć tysięcy uczestników
2019. - 14236 - transmisja turnieju w serwisie twitch.tv
2020. - 15235 - edycja zdalna, spowodowana pandemią
2021. - 20301 - 2. edycja zdalna
2022. - 40213 - wielki powrót do Pcimia i nowy rekord frekwencji

Obiekt: main
Atrybut: float
Wartość: left

Formularz zgłoszeniowy na turniej

Imię:
Nazwisko:
Telefon:
E-mail:
Wyślij za

Obiekt: main
Atrybut: float
Wartość: right

Obiekt: footer
Atrybut: clear
Wartość: both

Stronę przygotował: Adam Kwiatkowski

Podstawowe polecenia JS

- let – tworzy nową zmienną („szufladkę”, w której możemy umieścić jakąś jedną wartość, a potem ją zmieniać)
- const – tworzy nową stałą („szufladkę”, w której możemy umieścić jakąś jedną wartość, która nie będzie się zmieniać)
- if, else – tworzy warunek, zapytanie sprawdzające prawdziwość jakiegoś zdania i wykonujące odpowiedni kawałek kodu, w zależności, czy owo zapytanie jest prawdziwe, lub fałszywe
- for – pętla, pozwala na wykonanie danego kawałka kodu kilka razy

Algorytmika

- Programy (nie tylko w JS) wykonują kolejne polecenia linijka po linijce.
- Algorytm to zbiór takich poleceń, wykonujących konkretne zadania krok po kroku, zgodnie z pewnym planem.
- Wynik algorytmu zawsze powinien być taki sam, jeżeli za każdym razem używamy tych samych danych wejściowych (startowych).
- Funkcja to taki kawałek programu, który może być wywołany wiele razy, ale z różnymi parametrami wejściowymi

Algorytmika

- Przykładowo, algorytm który ma sprawdzić, która z dwóch liczb jest większa, będzie składać się z następujących kroków:
  - weź dwie liczby i przypisz je do zmiennych
  - sprawdź, czy pierwsza jest większa od drugiej?
    - jeśli tak, powiedz, że pierwsza jest większa i skończ program
    - jeśli nie, idź dalej
  - sprawdź, czy druga jest większa od pierwszej?
    - jeśli tak, powiedz, że druga jest większa i skończ program
    - jeśli nie, powiedz, że są równe i skończ program

```
1   let pierwszaLiczba = 2
2   let drugaLiczba = 5
3
4   if (pierwszaLiczba > drugaLiczba) {
5       console.log(pierwszaLiczba)
6   }
7   else if (drugaLiczba > pierwszaLiczba) {
8       console.log(drugaLiczba)
9   }
10  else {
11      console.log("Liczby sa rowne")
12  }
```

Utworzono dwie zmienne o nazwie „pierwszaLiczba” i „drugaLiczba” Przypisano tym zmiennym wartości. Od teraz, jeżeli w skrypcie gdzieś pojawi się słowo „pierwszaLiczba”, to będzie się z nim wiązała wartość 2. Analogicznie z „drugaLiczba”

Zadano pytanie, czy pierwsza liczba jest większa od drugiej?

Jeżeli odpowiedź na to pytanie to tak, wykonaj tę linijkę kodu. Polecenie console.log() sprawia, że coś wyświetla się w konsoli przeglądarki (F12). Tutaj, polecenie to wyświetli wartość zmiennej

Jeżeli odpowiedź na poprzednie pytanie brzmi „nie”, to idziemy dalej, zadajemy kolejne pytanie: czy druga liczba jest większa od pierwszej?

Jeżeli odpowiedź na to drugie pytanie brzmi tak, wykonaj tę linijkę kodu.

Jeżeli odpowiedź zarówno na pierwsze i drugie pytanie to „nie”, wykonaj tę ostatnią linijkę.

Warunki logiczne

X > Y	X większe od Y
X < Y	X mniejsze od Y
X >= Y	X większe lub równe od Y
X <= Y	X mniejsze lub równe od Y
X == Y	X takie samo jak Y (porównanie miękkie)
X === Y	X **dokładnie** takie samo jak Y (por. twarde)
X != Y	X inne/różne od Y

(X > Y) || (X > Z)	X większe od Y **LUB** X większe od Z
(X < Y) && (X < Z)	X mniejsze od Y **ORAZ** X mniejsze od Z

Wskazywanie obiektów skryptowi

- Aby wskazać skryptowi, który obiekt na stronie ma zmieniać, można zastosować kilka **metod**:
  - **document.getElementById** – wyszukanie **jednego** elementu, który posiada atrybut ID o wybranej wartości
  - **document.querySelector** – wyszukanie **jednego** elementu, który odpowiada na złożone zapytanie (np. elementu, który posiada kilka klas)
  - **document.getElementsByClassName** – wyszukanie **wszystkich** elementów, które posiadają atrybut klasy o wybranej wartości
  - **document.getElementsByTagName** – wyszukanie **wszystkich** elementów, które są tworzone przed wybrany znacznik HTML (np. wszystkich <li>, wszystkich , wszystkich <a> itd.)
  - **document.querySelectorAll** – wyszukanie **wszystkich** elementów, które odpowiadają na złożone zapytanie (np. elementów, które posiadają kilka klas)

```
let pierwszyNaLiscie = document.getElementById("list1")

let zielonyDiv = document.querySelector("div#zielony")

let wszystkiePojemniki = document.getElementsByClassName("pojemnik")

let wszystkieHeadery = document.getElementsByTagName("header")

let wybraneZListy = document.querySelectorAll("li.a1.b2")
```

UWAGA!
Metody, które zwracają wszystkie elementy pasujące do zapytania, zwracają tzw. **listę węzłów (nodeList).** Jeżeli skorzystamy z metody np. getElementsByClassName, ale chcemy odwołać się do konkretnego elementu z tego zbioru, musimy skorzystać z **indeksu** tego elementu na liście.

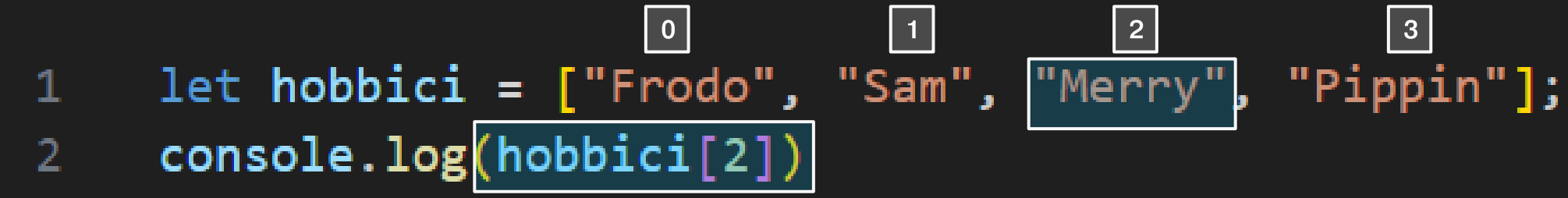
PRZYKŁAD:
Chcąc odwołać się tylko do trzeciego elementu o klasie „pojemnik”, mogę zastosować metodę document.getElementsByClassName(„pojemnik”)[2]

```
<body>
    <header>
        <h1>Tresc naglowka</h1>
    </header>
    <main>
        <div class="pojemnik">Pierwszy pojemnik</div>
        <div class="pojemnik">Drugi pojemnik</div>
        <div class="pojemnik">Trzeci pojemnik</div>
    </main>
    <aside>
        <ul>
            <li id="list1" class="a1, b1">jablko</li>
            <li id="list2" class="a1, b2">banan</li>
            <li id="list3" class="a1, b2">gruszka</li>
            <li id="list4" class="a2, b2">sliwka</li>
        </ul>
    </aside>
    <div class="zielony">Zielone pudelko</div>
    <div class="czerwony">Czerwone pudelko</div>
</body>
</html>
```

Czemu [2]? Po pierwsze, nawiasy kwadratowe wskazują na to, że mamy do czynienia z jakąś listą lub tablicą, czyli „szufladką”, w której jest **więcej niż jedna wartość**. Po drugie, listy i tablice zaczynają numerowanie **od zera**. To znaczy, że pierwszy element listy ma indeks 0, drugi ma indeks 1, itd.

Tablica

- Tablica, to taki rodzaj obiektu (zmiennej) w JS, który może przechowywać **wiele** wartości.
- Każdy element w tablicy posiada swoje ponumerowane „miejsce” na liście, tzw. **indeks**.
- Tablice zaczynają indeksowanie (numerowanie) od zera.
- Tablice (i inne listy elementów) są oznaczane nawiasami kwadratowymi [ ].
- Chcąc odwołać się do konkretnego elementu z tablicy/listy, w nawiasach należy podać indeks tego elementu.


0
1
2
3
1 let hobbici = ["Frodo", "Sam", "Merry", "Pippin"];
2 console.log(hobbici[2])

Wpływanie na styl obiektu

- Większość obiektów na stronie posiada atrybut **style**, który odpowiada za jego wygląd.
- Ów atrybut posiada pewne pod-atrybuty, którymi są konkretne „polecenia” (deklaracje) CSS.
- Jeśli chcemy zmienić np. kolor tła danego elementu, w CSS powołamy się na atrybut „background-color” – tzw. snake-case.
- Aby go zmienić w JS, należy zastosować zapis „backgroundColor” – tzw. camelCase.

```js
let obiektDoZmiany = document.getElementById("blok")
obiektDoZmiany.style.width = "200px"
obiektDoZmiany.style.height = "40px"
obiektDoZmiany.style.marginTop = "20px"
obiektDoZmiany.style.paddingTop = "40px"
obiektDoZmiany.style.backgroundColor = "#002137"
obiektDoZmiany.style.color = "lightblue"
obiektDoZmiany.style.textAlign = "center"

let innyObiektDoZmiany = document.getElementById("innyBlok")
innyObiektDoZmiany.style.width = "100px"
innyObiektDoZmiany.style.height = "100px"
innyObiektDoZmiany.style.backgroundColor = "red"
innyObiektDoZmiany.style.color = "orange"
innyObiektDoZmiany.style.textAlign = "center"
```

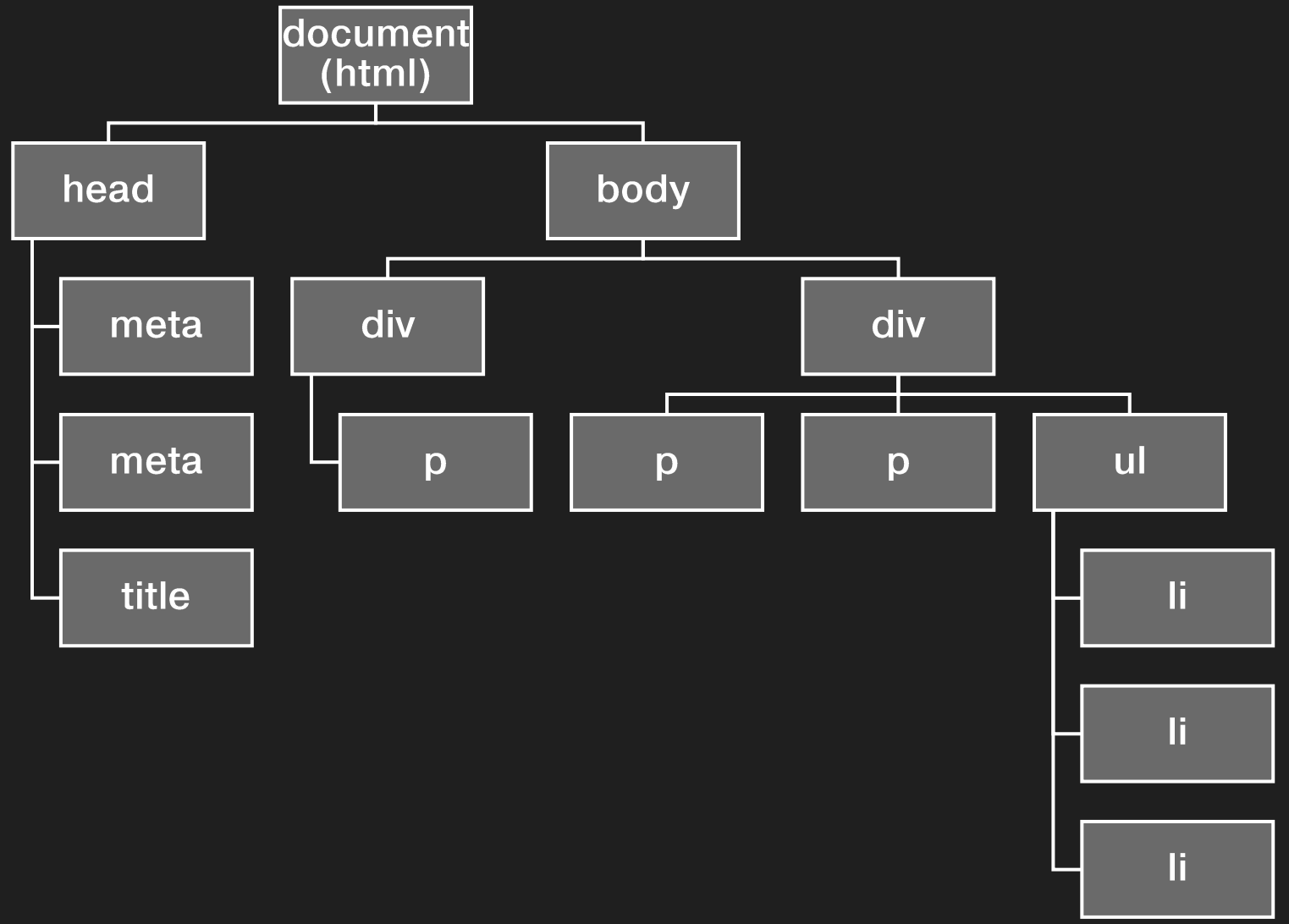
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="innyBlok">
        <p>Tu jest cos napisane</p>
    </div>
    <div id="blok">
        <p>W tym bloku jest jakas tresc</p>
    </div>
</body>
<script src="script.js"></script>
</html>
```

Tak podłączamy skrypt do pliku HTML

Relacja dziecko-rodzic

- JS wykorzystuje Obiektowy Model Dokumentu **(DOM – Document Object Model)** aby odnajdować żądane elementy na stronie.
- Każdy obiekt umieszczony wewnątrz innego obiektu wchodzi z nim w relację rodzic-dziecko **(parent-child).**
- Można to wykorzystać, aby dodawać, bądź usuwać kolejne elementy wewnątrz jakiegoś „pojemnika”.
- Relację tę często przedstawia się jako schemat drzewa. Każdy element takiego drzewa nazywamy węzłem **(node)**.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="innyBlok">
        <p>Tu jest cos napisane</p>
    </div>
    <div id="blok">
        <p>W tym bloku jest jakas tresc</p>
        <p>Tu jest jakas lista</p>
        <ul>
            <li>Element 1</li>
            <li>Element 2</li>
            <li>Element 3s</li>
        </ul>
    </div>
</body>
<script src="script.js"></script>
</html>
```



Dodanie elementu do strony

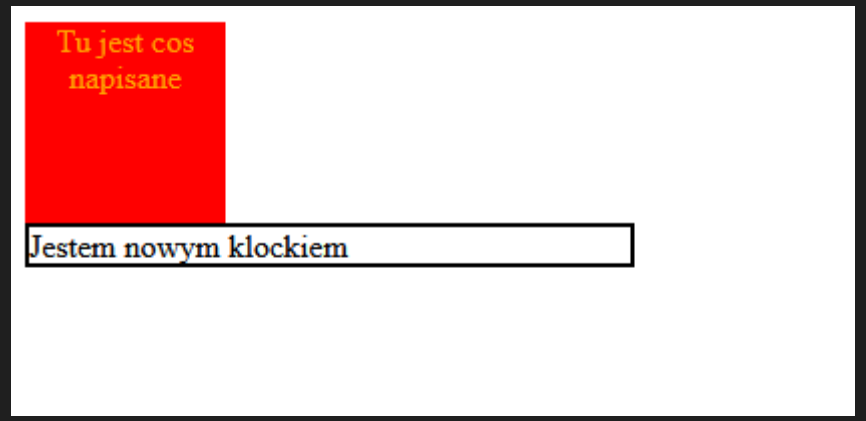
- Chcąc dodać jakiś element na stronę przy pomocy JS musimy:
  - Utworzyć nowy obiekt – **createElement()**
  - Nadać mu pożądane cechy
  - Umieścić go na stronie jako element podrzędny jakiegoś elementu nadrzędnego – **appendChild()**

```js
let innyObiektDoZmiany = document.getElementById("innyBlok")
innyObiektDoZmiany.style.width = "100px"
innyObiektDoZmiany.style.height = "100px"
innyObiektDoZmiany.style.backgroundColor = "red"
innyObiektDoZmiany.style.color = "orange"
innyObiektDoZmiany.style.textAlign = "center"

let nowyObiekt = document.createElement("div")
nowyObiekt.style.border = "2px solid black"
nowyObiekt.style.width = "300px"
nowyObiekt.innerText = "Jestem nowym klockiem"

document.getElementById("pojemnikNaDivy").appendChild(nowyObiekt)
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=
    <title>Document</title>
</head>
<body>
    <div id="pojemnikNaDivy">
        <div id="innyBlok">
            <p>Tu jest cos napisane</p>
        </div>
    </div>
</body>
<script src="script.js"></script>
</html>
```



Pętla for

- Pętla for służy do wykonania danego fragmentu kodu wielokrotnie.
- Składa się z czterech elementów:

```
for (licznik; warunek działania; zmiana licznika) {
        kod do wykonania
}
```

- Kolejność wykonywania operacji w pętli to:
  - Zainicjowanie licznika i przypisanie mu wartości startowej – to dzieje się tylko raz
  - Sprawdzenie prawdziwości warunku działania:
    - Jeśli warunek jest spełniony:
      - Wykonaj kod
      - Zmień wartość licznika
      - Wróć do sprawdzenia warunku
    - Jeśli warunek jest niespełniony:
      - Opuść pętlę

```
for (licznik; warunek działania; zmiana licznika) {
        kod do wykonania
}
```

```
1  for (i = 0; i < 10; i++) {
2      console.log(i)
3  }
4
5  for (j = 5; j >= 0; j--) {
6      console.log(j)
7  }
```

Zapis i++ oznacza **zwiększenie** wartości zmiennej i o jeden **(inkrementacja)**

Zapis j– oznacza **zmniejszenie** wartości zmiennej j o jeden **(dekrementacja)**

Trzeba pamiętać, że ilość wykonań pętli zależy od wartości licznika.

Dla najprostszej pętli (takiej, jak ta u góry), będzie to 10, ponieważ pętla wykona się dla i = 0, i = 1, i = 2,..., i = 8, i =9, ale dla i = 10 już nie (ponieważ 10 nie jest mniejsze od 10 – jest równe)

```
for (licznik; warunek działania; zmiana licznika) {
        kod do wykonania
}
```

```
1   let licznik = 200
2   let koncowa = -300
3   let zmiana = 50
4
5   for (x; x > koncowa; x -= zmiana) {
6       console.log(x)
7   }
```

Aby zmienić wartość jakiejś zmiennej, możemy zastosować zapis:
x = nowa wartość.

Jeśli chcemy, aby nowa wartość **tej samej** zmiennej była jej np. pomniejszeniem, stosujemy zapis:
x = x – 50 czyli „niech **nową** wartością zmiennej x będzie jej stara (**obecna**) wartość, minus 50.

Innym zapisem tego działania jest:
x -= 50. Podobnie można zrobić z innymi operacjami matematycznymi (np. x += 20, x*=10, itd.)

```
for (licznik; warunek działania; zmiana licznika) {
        kod do wykonania
}
```

0 1 2

```
let tanks = ["M1A1 Abrams", "M4A1 Sherman", "M24 Chaffee"]

for (i = 0; i < tanks.length; i++) {
    console.log(tanks[i])
}
```

Pętle są bardzo przydatne przy „przewijaniu się” przez elementy listy/tablicy.

Można np. zbudować pętle, której licznik będziemy używać jako indeks elementu w tablicy.

Niech pętla rozpocznie działanie od i = 0.

Pętla ma działać tak długo, aż i osiągnie wartość „tanks.length” – słowo „length” oznacza długość; w przypadku tablic, ich długość to ilość elementów, które się w nich znajdują.

Tablica „tanks” posiada 3 elementy (o indeksach 0, 1, 2). Zatem jej długość (tanks.length) = 3

Wykonaj kod w pętli: wyświetl w konsoli i-ty element tablicy „tanks” (wartość i zależy od tego, na którym wykonaniu pętli jesteśmy)

Zwiększ i o jeden

Sprawdź, czy i jest dalej mniejsze od tanks.length. Jeśli tak, znów wykonaj kod, jeśli nie, skończ pętlę

Funkcja

- Funkcje w JS tworzymy poleceniem function xyz().
- W miejsce „xyz” należy wpisać nazwę naszej funkcji, którą będziemy się posługiwać potem w „głównym” programie.
- Nawiasy w tym słowie kluczowym są miejscem na tzw. parametr funkcji, czyli jakąś wartość, którą chcemy do funkcji przekazać z zewnątrz.
- Funkcje są niezwykle przydatne, jeżeli jakiś kod chcemy wykonać kilka razy.

Zmienne lokalne i globalne

- Każda zmienna w skrypcie JS ma swój zasięg.
- Zmienna może być lokalna lub globalna
- Zmienne lokalne są widziane tylko w obrębie danego bloku kodu, ograniczonego przez nawiasy klamrowe { }.
- Zmienne globalne są widziane w obrębie całego programu.

```
1   let zmiennaGlobalna = "Czesc"
2   console.log(zmiennaGlobalna)
3
4   function funkcjaLokalna() {
5       let zmiennaLokalna = "Witam"
6       console.log(zmiennaGlobalna)
7       console.log(zmiennaLokalna)
8   }
9
10  console.log(zmiennaLokalna)
```

Tę zmienną widać „wszędzie”, zarówno wewnątrz funkcji o nazwie funkcjaLokalna(), jak i poza nią

funkcjaLokalna() ogranicza pewnie kawałek kodu klamrami – wszystko, co dzieje się w środku tej funkcji, tam zostanie

funkcjaLokalna() nie będzie miała problemu z wyświetleniem w konsoli wartości obydwu zmiennych, bo:

- zmiennaGlobalna jest globalna – widoczna nawet „ponad” funkcją
- zmiennaLokalna jest lokalna – utworzona wewnątrz funkcji

Reszta programu nie zobaczy zmiennej zmiennaLokalna, tutaj pojawi się błąd (widać go nawet w edytorze, litera „z” jest podkreślona znakami ...)

Wyzwalacz funkcji

- Skrypty w JS mogą wykonywać się automatycznie, lub po ich wywołaniu.
- Jeżeli cały skrypt ma w sobie zarówno „luźno umieszczony” kod, jak i funkcje, to w momencie, gdy przeglądarka dotrze do miejsca w kodzie HTML strony, gdzie umieszczony jest skrypt, ów „luźny” kod zostanie automatycznie wykonany.
- Kod, który jest umieszczony w funkcjach, będzie oczekiwał na wywołanie. Wywołanie może nastąpić poprzez:
  - Wyzwalacz przypisany do jakiegoś obiektu HTML
  - Odwołanie do żądanej funkcji w innej funkcji

```
let klocekDoZmiany = document.getElementById("klocek")
klocekDoZmiany.style.backgroundColor = "red"
klocekDoZmiany.style.width = "200px"
klocekDoZmiany.style.height = "200px"

function zmianaKoloru() {
    klocekDoZmiany.style.backgroundColor = "blue"
    zmianaKoloruTekstu()
}

function zmianaKoloruTekstu() {
    klocekDoZmiany.style.color = "pink"
}
```

Ten kod nie jest w funkcji, więc wykona się automatycznie, gdy przeglądarka zacznie wczytywać plik ze skryptem.

Ten kod będzie czekał na wywołanie. Wyzwalaczem dla funkcji zmianaKoloru() jest tzw. wydarzenie „onclick”, powiązane z blokiem . Kliknięcie gdziekolwiek w ten uruchomi funkcję.

Ten kod również czeka na wywołanie. Ta funkcja zostanie uruchomiona, gdy inna funkcja się do niej odwoła – w bloku kodu powyżej jest linijka, która zawiera w sobie tylko nazwę drugiej funkcji – to właśnie wywołanie poprzez odwołanie.

Najpierw przeglądarka wczytuje zawartość dokumentu HTML, linijka po linijce.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="klocek" onclick="zmianaKoloru()">
        <p>Ten blok div ma zmienic kolor po kliknieciu na niego</p>
    </div>
</body>
<script src="script.js"></script>
</html>
```

Przeglądarka trafia na odniesienie do skryptu – w tym miejscu zatrzymuje na chwilę wczytywanie dokumentu HTML i skupia się na pliku JS.
Dlatego warto umieszczać odniesienia do skryptu na dole dokumentu – cała strona „wizualnie” zdąży się już wczytać, użytkownik „ma co oglądać”, a skrypt ma czas, żeby się w pełni załadować do przeglądarki.

Istnieją także inne „wyzwalacze” wydarzeń, np.:
- onmouseover – po najechaniu myszą
- onkeydown – po naciśnięciu klawisza
- onscroll – po przewinięciu kółkiem myszy
- i wiele, wiele innych…

Parametr funkcji

- Parametr funkcji pozwala funkcji na pobranie jakiejś wartości „z zewnątrz” – inaczej, parametr funkcji określa jakąś wartość startową, z którym funkcja może rozpocząć działanie.
- Ta sama funkcja może zostać wywołana wiele razy, ale z inną wartością parametru.

Tworząc funkcję musimy nazwać nasz parametr – będzie on działał jak zmienna, która otrzymuje wartość spoza tej funkcji. W miejscu słowa „liczba” będzie umieszczana wartość, pochodząca z nawiasów przy wywołaniu funkcji.

```
function potega(liczba) {
    wynik = liczba * liczba
    console.log(wynik)
}
```

Wszystkie trzy przyciski mają wywołać tę samą funkcję o nazwie „potega()” (5, 10, 20…). Liczba (X) umieszczona w nawiasach to właśnie parametr funkcji – jej „wartość startowa”.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <button onclick="potega(5)">Policz 2-ga potege liczby 5.</button>
    <button onclick="potega(10)">Policz 2-ga potege liczby 10</button>
    <button onclick="potega(20)">Policz 2-ga potege liczby 20</button>
</body>
<script src="script.js"></script>
</html>
```

Klikając pierwszy przycisk, funkcja „potega()” zostanie wywołana z parametrem 5 – jej wynikiem będzie 25.

Drugi przycisk wywołuje funkcję „potega()” z parametrem 10 – wynik = 100.

Trzeci przycisk, parametr = 20, wynik = 400.

Zawsze wywoływana jest ta sama funkcja – jedyne, co ulega zmianie, to wartość parametru, który sprawia, że uzyskujemy inny wynik obliczeń.

Co do zapamiętania?

- JavaScript – obiekty, atrybuty, wartości
- let – zmienne, const – stałe, if else – warunki, for – pętle
- Tablice, indeksy
- Program, a funkcja, wywoływanie funkcji, wyzwalacze, parametry
- Wskazywanie obiektów – np. document.getElementByld
- Zmiana stylu obiektu
- Model DOM, relacja parent-child, dodanie nowego obiektu

Zadanie

- Utwórz stronę kurortu (na wyspach, w górach, dowolnie), w którym chcesz spędzić wakacje.
- Zastosuj standardowy układ nagłówka, zawartości głównej, zawartości pobocznej i stopki.
- W tle strony umieść jakiś ładny krajobraz związany z tym miejscem (np. zachód słońca na plaży, ośnieżone szczyty).
- W nagłówku umieść logo kurortu i jego nazwę, jakieś motto, itd.
- W bloku głównym opisz krótko gdzie znajduje się ten kurort, jakie są główne atrakcje, itd. Wstaw dużo zdjęć, np. w tabeli, żeby były ładnie poukładane
- W bloku pobocznym umieść kilka opinii od zadowolonych (lub niezadowolonych) klientów – wstaw zdjęcie (profilowe) i opisową ocenę. Opinie mają być jedna pod drugą.
- W stopce podpisz się (umieść też znak ©).
- W nagłówku, obok nazwy kurortu, umieść przycisk, do którego podepniesz skrypt.
- Ten skrypt ma za zadanie zamienić obrazek w tle strony na inny (o podobnej tematyce).
- * dodatkowo możesz sprawić, żeby przycisk działał „w dwie strony” – po ponownym kliknięciu niech przywróci poprzedni obrazek.