# Flex

parent:

- display: flex; - ustawienie flexa

- flex-direction: row/column/row-reverse/column-reverse; - w jaką stronę ma się fleksować poziomo czy pionowo

- flex-wrap: wrap/nowrap; - czy childy mają się zawijać do nowej linii czy nie

- justify-content: flex-start/fles-end/center/space-between/space-around; są to rodzaje horyzontalnych odstępów pomiędzy childami flex'a

- align-items: fles-start/fles-end/center/stretch/baseline; - są to rodzae odsępów wertykalnych między childami flex'a

- align-content: flex-start/flex-end/center/stretch/space-between/space-around; - jak się mają pozycionować wszystkie childy wewnątrz rodzica flex'a

child:

- flex-items: (number); - kontroluje koleność w jakiej się pojawiają

- flex-grow: (number); - definuje rozszerzanie się elementów;

# Grid

parent:

- display: grid; ustawienie grida

- .container {

grid-template-columns: [first] 40px [line2] 50px [line3] auto [col4-start] 50px [five] 40px [end];

grid-template-rows: [row1-start] 25% [row1-end] 100px [third-line] auto [last-line];

} - wartości względem których ustala się wielkości danychpól w gridzie

- grid-template-areas:

"header header header header"

"main main . sidebar"

"footer footer footer footer";

} - olreśla całą przestrzeń za pomocą nazwania ich

- grid-gap: (number); - określa przestrzeń jaka ma się znaajdować pomiędzy każdym boxem w gridzie

- justify-items: ...; - wypozcjonowanie wszsytkich horyzontalnie

- align-items: ...; - wypozcjonowanie wszsytkich wertykalnie

child:

- .item {

grid-column-start: <number> | <name> | auto;

grid-column-end: <number> | <name> | auto;

grid-row-start: <number> | <name> | auto;

grid-row-end: <number> | <name> | auto;

} - określa pozycję na siatce (czyli gridzie)

- grid-area: <name>; określa boxy w który ma się pojawić dany child

- justify-self: start/end/center/stretch; - jak ma się horyzontalnie wypozycjonować w danych polu grida

- align-self: start/end/center/stretch; - jak ma się wertykalnie wypozycjonować w danych polu grida

# Positions

- static - jest to defaultowe ustawienie property position - nie jest w żaden sposób specyficzne

- relative - element jest pozycionowany/ustawiany w stosunku do pierwotnej pozycji. Ustawianie left, czy top spowoduje, że zostanie on presunięty w stosunku do swojej normalnej pozycji.

- fixed - ten element jest "przyklejony na stałe" jest ona ustawiana względem okna przeglądarki

- absolute - jest ustawiany względem swojego najbliższego relatywnego elementu. Jeżeli żaden taki nie istnieje to odnosi się do okna przeglądarki. Nie jest dla niego generowane miejsce na stronie

- sticky - przełącza się pomiędzy relative, a fixed.

# Modyfikatory dostępu

> - Child Selector - dziecko

+ - Adjacent Sibling Selector - sąsiadujący

~ - General Sibling Selector – generalny

Attr Selectors:

a[target="\_blank"] - jeżeli element a będzie miał w sobie target="\_blank" - wtedy wykona się kod

a[atrybut ~= "flower"] - gdy pojawi się atrybut występujący gdzieś w class to wtedy się odpali

a[atrybut^="value"] - jeżeli zaczyna się na value

a[atrybut$="value"] - jeżeli kończy się na value

a[atrybut\*="value"] - jeżeli zawiera element value

# For ... in (w obiekcie ) vs for ... of (po tablicy)

# This

- odwołuje się do samego siebie za równo w funkcji jak i w obiecie, na podstawie tego możemy uzyskać specyficne operacje,

nie jeżeli funkcja jeszcze nie została przypisana do zmiennej (jako obiek oczywiście)

Czyli odwoływalnie się do samego siebie w obiekcie, czy innej konstrukcji

Różnica w this’ie jest zauważalna np czy funkcjach strzałkowych i zwykłych – przy użyciu arrow function nie nadpisujemy thisa

# Bind vs call vs apply

Jest to 2-stopniowe przypisanie funkcji do nowego obiektu - dopisze właściwości w "kopii", ale oryginał funkcji pozostawi niezmieniony.

# call - redefiniuje instancję obiektu na czas wywołania, każdy element funkcji wywołującej jest samodzielnym elementem

# \* cylinder.volume.call({pi: 3.14159}, 2, 4);

# apply - redefiniuje instancję obiektu na czas wywołania, początkowe elementy są wartościami obiektu,

# natomiast elementy funkcji można podać w tablicy

# \* cylinder.volume.apply({pi: 3.14159}, [2, 4]);

# Wniosek: call i apply są do siebie nawzajem stosunkowo podobne

# Operacje na arrayu

map - mapuje się po każdym elemencie z tablicy

filter - filtruje

reduce - można to uznać za połączenie mapa i filter

flatmap - flatmap spłaszcza tablicę tablic o jeden poziom, argument liczbowy definiuje ilość spłaszczeń - jeżeli damy liczbę większą

niż te spłaszczenia to zredukujemy tablicę tablic do pojedynczej tablicy

splice - wycina elementy i podmienia

slice - wycina od do i zwraca

some - zwraca true, jeżeli którykolwiek element tablicy spełnia warunek

every- zwraca true, jeżeli wszystkie elementy spełniają warunek

indexOf - wyszukuje index po płaskiej tablicy

findIndex - wyszukuje indes po tablicy obiektów (nie płaskiej)

# Operacje na obiektach

isArray - sprawdza, czy element jest tablicą

hasOwnProperty() - sprawdzenie czy posiada property o danej nazwie

for ... in - przechodzenie po polach z obiektu

# DOM - document object model

# Do czego służy strict mode

Aby uruchomić w tanym pliku strict mode musimy napisać - 'use strict';

Jest to dodany w ECMAScript5 dodatkowy tryb interpretowania kodu, który jeszcze bardziej rygrystycznie podchodzi do jego oceny.

Kod strict mode i zwwykłęgo mogą współ istnieć.

# Event loop

call stack - stos wywołoń (funkcji najczęściej) - bufor typu LiFo - wszystie wywołańa pokolejni są

zbierane i od odstatniego dorzuconego zostają one wywołane.

Maksymalna pojemność stosu to 16 000 wywołań.

W czasie jednej iteracji kończy się tylko jedno zadanie, a cały stos kończ swoje działanie, gdyzdejmiemy z nie wszystkie elementy.

quene - kolejka - FiFo wywołania wiadomości są wywolywane w kolejności takiej jakiej się pojawiły na tej kolejce

Heap - sterta - przechowywane są tam dynamicznie tworzone struktury danych w czasie trwania samego programu.

Za zwalnianie pamięci z nieużywanych elementów odpowiada garbage collector.

setTimeout() - podana jest minimalna wartość w milisekundach, po której wykona się kod

- może wywołaś się później niż zakładane z powodu innych elementów w stosie lub kolejce

mikrotaski - obszar który obsługuje kod asynchroniczny - z tego obszaru może podczas jednej pętli zejść więcej niż jeden element

znajdują się tam między innymi setTimeout(), Promise oraz ręcznie napisane funkcje asynchroniczne.

# Wbudowane typy generyczne: partial, readonly, required

# Lifecycle hooks

ngOnChanges - wywoływany za każdym razem kiedy zmienia się wartość pola oznaczonego dekoratorem @Input()

ngOnInit - wywoływany tylko raz podczas tworzenia komponentu, wywoływany po konstuktorze

ngDoCheck - reaguje na zmianę każdego innego pola niż @Input - lecz nie zaleca się, ponieważ aplikacja możne poważnie zacząć zwalniać

ngAfterContentInit - po inicjalizacji <ng-content> - tylko raz

ngAfterContentChecked - po każdej zmianie w <ng-content> - wiele razy

ngAfterViewInit - po zainicjalizowaniu widoku komponentu - tylko raz

ngAfterViewChecked - po każdej zmianie w widoku - wiele razy

ngOnDestroy - przy "zniszczeniu" komponentu - tylko raz

# Kilka komend z cli

npm install -g @angular/cli

ng help

ng new my-first-project

ng serve --o

ng build my-app -c production

# Elementy składowe modułu

- declaration - tutaj znajdują się deklaracje wszystkich komponetów w ramach tego modułu.

- import - tutaj wpisujemy komponenty, które będziemy wykorzystaywać w ramach tego modułu.

- export - komponeny, które eksportujemy w ramach tego modułu, aby mogły być używane w innych modułach.

- providers – umieszczamy tam np interceptory (przechwytuje i obsługuje zapytania i odpowiedzi http)

- bootstrap - używa się go w appModule – mówi o komponencie głównym (który tworzy cały widok)

# Dependency injection (całość)

DI - to w zasadzie mapa (provide: klucz, useValue: wartość)

const inject = Injector.create([

{provide: klucz, useValue: wartość}

]);

inject.get("klucz");

Rejestracja serwisu, który można wstrzykiwać:

@Injectable({ providedIn: 'root', })

export class MyService { ... }

wstrzykuje się zależność w konstruktorze komponentu np.:

component(private myService: MyService){}

Rejestracja serwisu, który można wstrzykiwać:

@Injectable({ providedIn: 'root', })

export class MyService { ... }

component(private myService: MyService){}

"Drzewo wstrzyknięć" - poniżej jest pokazana hieriarchia wstrzyknięć. Każdy z nich ma rodzica (prototyp), poza wstrzykiwaczem null'owym.

null

|

platform

|

root

Dalej są normalne komponenty.

# Stub

* jest to funkcja która zastępuje implementację oryginalnej funkcji i zwraca tylko oczekiwany wynik z pominięciem pozostałych
* właściwości tej funkcji. Dzięki temu podczas tesu nie jest wykonywana niepotrzebna komunikacja dzięki czemu oszczędza się
* za równo pamięć jak i czas.

# AAA : Arrange-act-assert

# TDD : Test-Driven-Development