

TESTOWANIE I DOKUMENTOWANIE APLIKACJI

Klaudia Wojciechowska | ZSK 2024

OMNIE

- **Wykształcenie:**
 - Inżynier Informatyki (Aplikacje Internetowe i Mobilne) - Collegium Da Vinci w Poznaniu
 - Technik Informatyki - Zespół Szkół Politechnicznych we Wrześni
- **Doświadczenie zawodowe:**
 - Wykładowczyni na uczelni Collegium Da Vinci - kompetencje miękkie, wejście na rynek IT, doradztwo zawodowe
 - Scrum Masterka, Testerka manualna, frontend developeka - NASK
 - Testerka manualna, frontend dev - e-LEA



KONTAKT Z PROWADZĄCĄ

- E-dziennik
- Email
 - klaudia.wojciechowska@zsk.poznan.pl



ZASADY OCENIANIA - WYKŁAD

- **Min 1 sprawdzian**
- **Min 2 kartkówki**
- **Praca na lekcji - według potrzeb**
- **Aktywność na lekcji - zbieranie plusików 5 plusików wpada piąteczka do dziennika**
- **Nieprzygotowanie 2x - poinformować na początku zajęć.**

ZASADY OCENIANIA - LABY

- **Min 1 sprawdzian**
- **Praca na lekcji - według potrzeb minimum 2 oceny**
- **Aktywność na lekcji - zbieranie plusików 5 plusików wpada piąteczka do dziennika**
- **Nieprzygotowanie 2x - poinformować na początku zajęć.**

UWAGI

- Zadania z lekcji będą sprawdzane wyrywkowo na kolejnych zajęciach. Nieobecność na lekcji nie zwalnia z wykonania zadania.
- Szczęśliwy numerek lub nieprzygotowanie zwalnia z wyrywkowego sprawdzania zadania, **ale NIE ze sprawdzianu, zapowiedzianej kartkówki i aktywności na zajęciach**
- W przypadku nieobecności na sprawdzianie i kartkówkach– **praca musi zostać uzupełniona**
- Sprawdzian musi być zaliczony na **ocenę pozytywną**
- Aby być klasyfikowanym ilość ocen pozytywnych musi być większa od ilości ocen negatywnych **oraz** należy zdobyć wymaganą minimalną liczbę ocen
- **Przystąpienie** do Próbnego Egzaminu Zawodowego **jest obowiązkowe**

POZNAJMY SIĘ

Opowiedz o sobie, jak masz na imię, co Cię interesuje

"So tell me more about yourself"

Me, trying to remember who I am:



DOKUMENTOWANIE

DOKUMENTACJA

CO TO JEST DOKUMENTACJA

Dokumentacja to zbiory różnych dokumentów i materiałów, które opisują różne aspekty projektu, takie jak:

- wymagania biznesowe,
- implementację oprogramowania, czy
- kod źródłowy.

Może przybierać formę tekstu, komentarzy w kodzie, diagramów lub elektronicznych rejestrów zadań w narzędziach do zarządzania projektami.

PO CO TWORZYMY DOKUMENTACJĘ?

Wyobraź sobie, że zakończyłeś ważny kamień milowy projektu – udało Ci się! Osiągnąłeś wszystkie cele i wysłałeś niesamowity produkt do produkcji.

Gratulacje! 

Ale jest jeden problem: nikt oprócz Ciebie nie wie, jak korzystać z tego produktu, a Ty zaraz wyjeżdżasz na sześciomiesięczne wakacje dookoła świata! 

Osiągnąłeś duży sukces, ale... co teraz???

KAMIENIE MILOWE -MILE STONES

- **Kamienie milowe** pomagają podzielić projekt na zarządzane etapy i są elementem sygnalizacyjnym, który ma informować o realizacji projektu, zgodnie lub niezgodnie z planem i obranym harmonogramem.
- Często jest to jeden z głównych mierników postępów, który nie tylko motywuje do dalszego działania, ale także koordynuje i wskazuje na ewentualne opóźnienia czy trudności.
- pewne istotne, jednorazowe zdarzenie określone jednoznacznie
- ważne zdarzenie w harmonogramie o czasie trwania 0 (zero)
- podsumowuje zestaw zadań lub fazę projektu



KAMIENIE MILOWE -MILE STONES

- [https://leadership-center.pl/blog/kamien-milowy/?
srltid=AfmBOopLx9Eb6QazwMtiNVjCcw03gwAuH1-KgisM8cUGoRwj1UrXI-T](https://leadership-center.pl/blog/kamien-milowy/?srltid=AfmBOopLx9Eb6QazwMtiNVjCcw03gwAuH1-KgisM8cUGoRwj1UrXI-T)



Wyobraź sobie, że zakończyłeś ważny kamień milowy projektu – udało Ci się! Osiągnąłeś wszystkie cele i wysłałeś niesamowity produkt do produkcji.

Gratulacje! 

Ale jest jeden problem: nikt oprócz Ciebie nie wie, jak korzystać z tego produktu, a Ty zaraz wyjeżdżasz na sześciomiesięczne wakacje dookoła świata! 

Osiągnąłeś duży sukces, ale... co teraz???

Nie ma problemu – masz to! Zaczynasz wszystko zapisywać... z wyjątkiem tego, że nie pamiętasz, jak to się stało, że rozwiązałeś ten jeden problem cztery miesiące temu lub pokonałeś poważną przeszkodę w zeszłym roku.

**Większość tego, co piszesz, zakłada, że czytelnik wie
tyle samo, co Ty.**

**I szczerze mówiąc, nawet nie pamiętasz, jak niektóre
części są ze sobą połączone i jak powinny ze sobą
współpracować.**

Co teraz??!!

- **Dokumentacja techniczna** często jest pisana po zakończeniu projektu i tuż przed terminem. Może więc zawierać luki i założenia, które utrudniają jej użyteczność.
- **Zapominamy o kluczowych krokach** w naszych instrukcjach i przyjmujemy założenia dotyczące doświadczenia i wiedzy naszych czytelników.
- Wszystko to sprawia, że dokumentacja jest trudna w czytaniu i co gorsza, zniechęcająca czytelników.

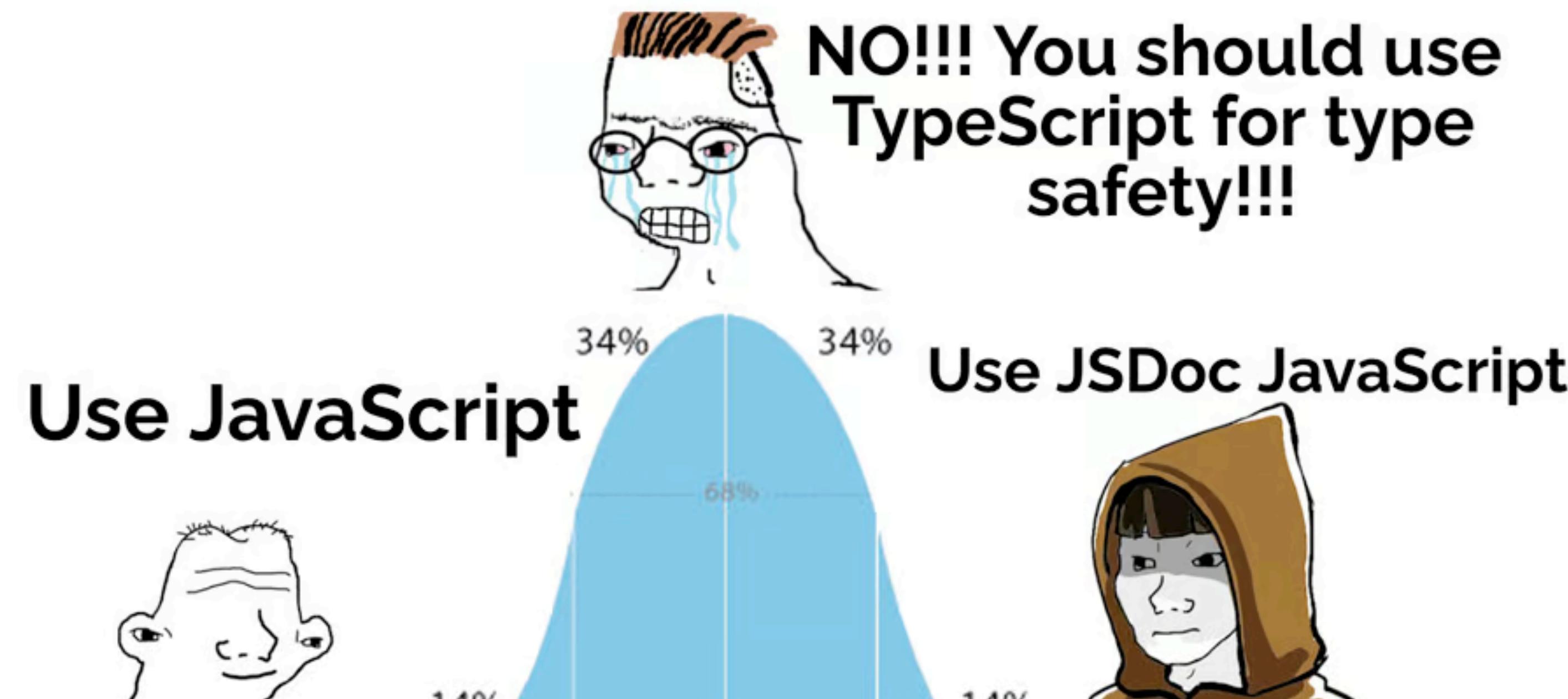
PO CO TWORZYĆ DOKUMENTACJĘ

- Po pierwsze – **za pół roku nie będziesz pamiętać, co chciałeś osiągnąć w danej linijce!**
- Jeśli chcesz, żeby ktoś korzystał z Twojej pracy – **to musi wiedzieć jak.**
- Mając do dyspozycji dokumentację, możesz odwoływać się do prac wykonanych w przeszłości i **czerpać z nich wnioski, zamiast robić wszystko od nowa** z takim samym wynikiem.
- **Proces zatrudniania i onboardingu** jest o wiele łatwiejszy
- **Dokumentacja zwiększa zakres wiedzy zbiorowej każdej osoby, z którą współpracujesz.**
Gdy dzielenie się informacjami stanie się w Twoim zespole normą, zwiększy się przejrzystość pracy, a kultura będzie w większym stopniu oparta na współpracy i podejściu strategicznym.

JSDOC

JSDOCS

JSDocs to sposób na udokumentowanie i dodawanie informacji do kodu, który napisaliśmy. Informacje te pojawiają się w okienku narzędziowym i ułatwiają programistom korzystanie z naszego kodu.



Me : mom can we have



?

Mom : no, we have



at home



at home :

```
1  /**
2   * JSDoc
3   * @param {number} a
4   * @param {number} b
5   */
6  function sum(a, b){
7    return a + b;
8 }
```

JAK TO DZIAŁA?

- Dodajemy JSDocs do funkcji, dodając komentarz blokowy z tagiem otwierającym (`/**`) i tag zamykający (`**/`). Wszelkie dodatkowe linie między tymi znacznikami rozpoczynającymi się i zamykającymi zawierają gwiazdkę, `*..`
- Uwaga: Dodatek JSDocs musi być bezpośrednio powyżej bloku kodu, który dokumentujesz.
- W poniższym przykładzie mamy funkcję, która dodaje dwie liczby i zwraca wynik. Ponadto dodaliśmy prosty opis tego, co będzie działać.

```
/**  
 * Adds two values together and returns the result.  
 */  
  
function addNumbers(value1, value2) {  
    return value1 + value2;  
}
```

JAK TO DZIAŁA?

```
1  /**
2  * Adds two values together and returns the result.
3  */
4
5  function addNumbers(value1, value2) { Show usages
6      return value1 + value2;
7  };
8
9  addNumbers( value1: 1, value2: 2);
```

function addNumbers(
 value1,
 value2): any

Adds two values together and returns the
result.

js test.js



⋮



PARAMETRY

- **@param** Określa parametry użyte w funkcji wraz z typem parametru i opisem
- Dodatkowo można określić typ danych parametrów za pomocą {}

```
/**  
 * Adds two numbers together and returns the result.  
 * @param {number} value1 The first value  
 * @param {number} value2 The second value  
 */  
function addNumbers(value1, value2) {  
    return value1 + value2;  
}
```

PARAMETRY

```
1  /**
2  * Adds two numbers together and returns the result.
3  * @param {number} value1 The first value
4  * @param {number} value2 The second value
5  */
6  function addNumbers(value1 :number , value2 :number ) { Show usages
7      return value1 + value2;
8  }
9  ⚡
10 addNumbers()
```

```
function addNumbers(
  value1: number,
  value2: number): any
```

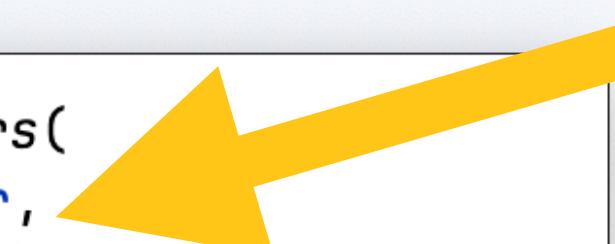
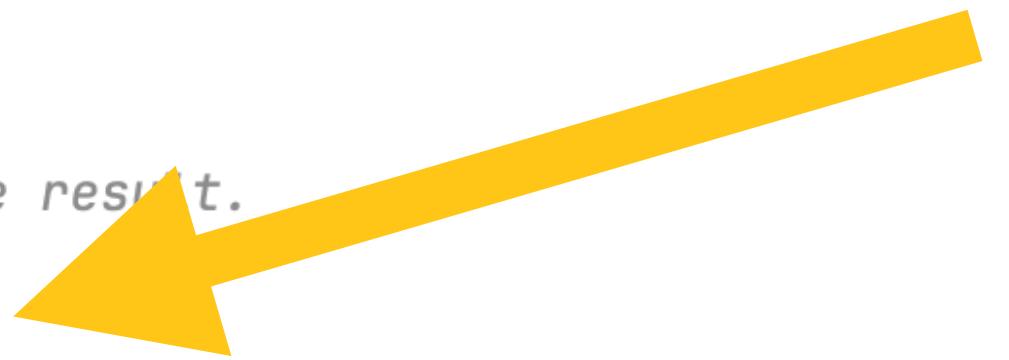
Adds two numbers together and returns the result.

Params: value1 – The first value
value2 – The second value

js test.js



:



ZWRACANIE

- **@returns tag dokumentuje wartość zwracaną przez funkcję.**

```
/**  
 * Adds two numbers together and returns the result.  
 * @param {number} value1 - The first value  
 * @param {number} value2 - The second value  
 * @returns {number} The values that have been added together  
 */  
function addNumbers(value1, value2) {  
    return value1 + value2;  
}
```

ZWRACANIE

```
1  /**
2   * Adds two numbers together and returns the result.
3   * @param {number} value1 - The first value
4   * @param {number} value2 - The second value
5   * @returns {number} The values that have been added together
6  */
7  function addNumbers(value1 :number , value2 :number ) :number
8      return value1 + value2;
9
10
11 addNumbers()
```

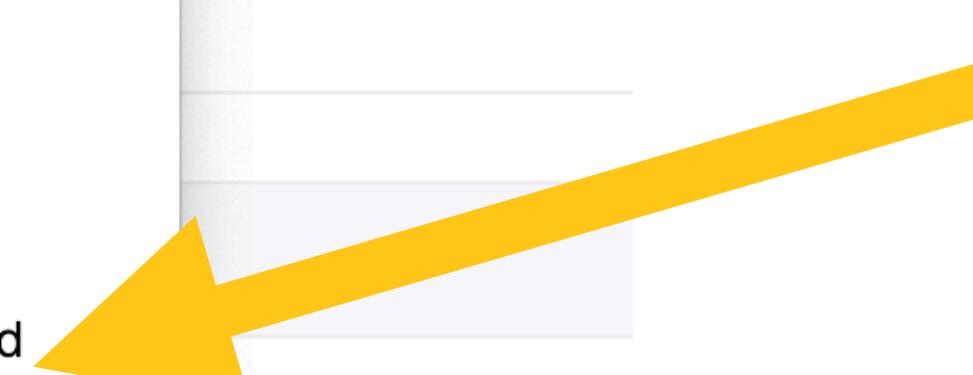
```
function addNumbers(
  value1: number,
  value2: number): number
```

Adds two numbers together and returns the result.

Params: value1 – The first value
value2 – The second value

Returns: The values that have been added together

js test.js



PRZYKŁAD

- **@example Pozwala dodać przykład kodu z podświetlaniem składni, aby pokazać, jak używać kodu, który napisałesz.**
- **Podanie przykładu kodu przyspiesza przepływ pracy, informując innych programistów, jak używać kodu.**

```
/**  
 * Adds two numbers together and returns the result.  
 * @param {number} value1 - The first value  
 * @param {number} value2 - The second value  
 * @returns {string} The values that have been added together  
 * @example  
 * const a = 10;  
 * const b = 20;  
 *  
 * const result = addNumbers(a, b);  
 * console.log(result);  
 * // Logs: 30  
 */  
  
function addNumbers(value1, value2) {  
    return value1 + value2;  
}
```

PRZYKŁAD

```
function addNumbers(  
  value1: number,  
  value2: number): string
```

Adds two numbers together and returns the result.

Example:

```
const a = 10;  
const b = 20;  
  
const result = addNumbers(a, b);  
console.log(result);  
// Logs: 30
```



Params: value1 – The first value

value2 – The second value

Returns: The values that have been added together

js test.js

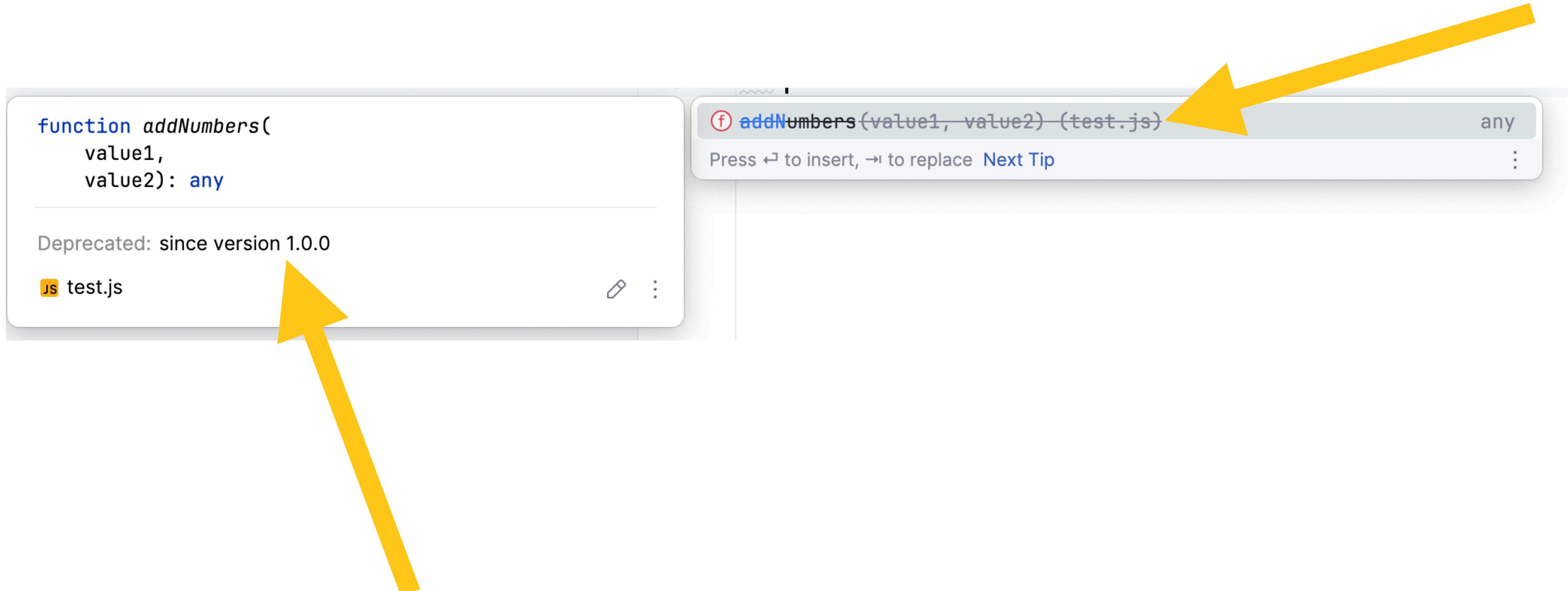


NIEAKTUALNE / ZDEPRECJONOWANE

- `@deprecated` tag pokazuje, że kod stał się przestarzały.
- Przestarzała funkcja będzie miała efekt przekreślenia w nazwie funkcji podczas uzupełniania kodu. Możesz podać więcej informacji, np. jakiej funkcji zastąpczej użyć.

```
/**  
 * @deprecated since version 1.0.0  
 */  
function addNumbers(value1, value2) {  
    return value1 + value2;  
}
```

NIEAKTUALNE



WYJĄTKI

- **@throws** pozwala udokumentować błąd, który może zgłosić funkcja. Możesz uwzględnić
- Tag **@throws** więcej niż raz w jednym komentarzu JSDoc

```
/**  
 * A function that throws an error  
 *  
 * @param {Boolean} input If `true` trows a simple error  
 * @throws {Error} If `input` is `true`  
 * @throws {TypeError} If the `input` is `false`  
 * @returns {void}  
 */  
  
const throwErrorOnInput = input => {  
    if (input) {  
        throw new Error(`This is an Error`)  
    }  
    throw new TypeError(`This is a TypeError`)  
}
```

WYJĄTKI

16 **throwE**

```
function throwErrorOnInput(  
    input: Boolean): void
```

A function that throws an error

Params: input – If true trows a simple error

Throws: Error – If input is true
 TypeError – If the input is false

src/add.js

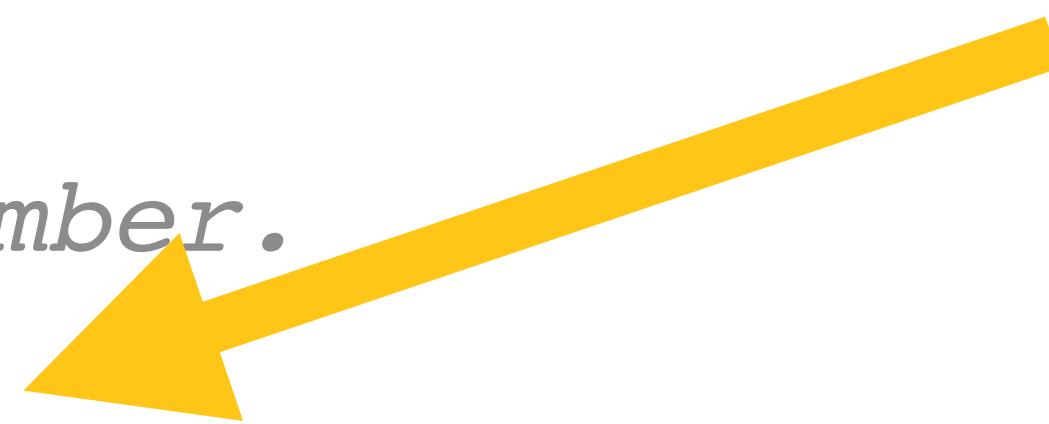
(f) throwErrorOnInput(input: Boolean) (add.js) void

Press ↲ to insert, → to replace Next Tip

Niestandardowe

- Znacznik `@typedef` jest przydatny do dokumentowania typów niestandardowych, szczególnie jeśli chcesz się do nich odwoływać wielokrotnie.

```
/**  
 * A number, or a string containing a number.  
 * @typedef {(number|string)} zupa  
 */  
  
/**  
 * Set the magic number.  
 * @param {zupa} x - The magic number.  
 */  
const setMagicNumber = (x) =>{  
}
```



NIESTANDARDOWE

A screenshot of a code editor showing a tooltip for a function named `setMagicNumber`. The tooltip contains the function signature `(f) setMagicNumber (x: zupa) (test.js)`, a description "Press ⇨ to insert, →! to replace Next Tip", and a small icon.

The code editor shows the following code:

```
function setMagicNumber(  
  x: zupa)  
Set the magic number.  
Params: x – The magic number.  
JS test.js
```

Two yellow arrows point from the word "zupa" in the tooltip to the word "zupa" in the code editor's code block.

GENEROWANIE DOKUMENTACJI

- Instalacja globalna:

```
npm install -g jsdoc
```

- Instalacja lokalna:

```
npm install --save-dev jsdoc
```

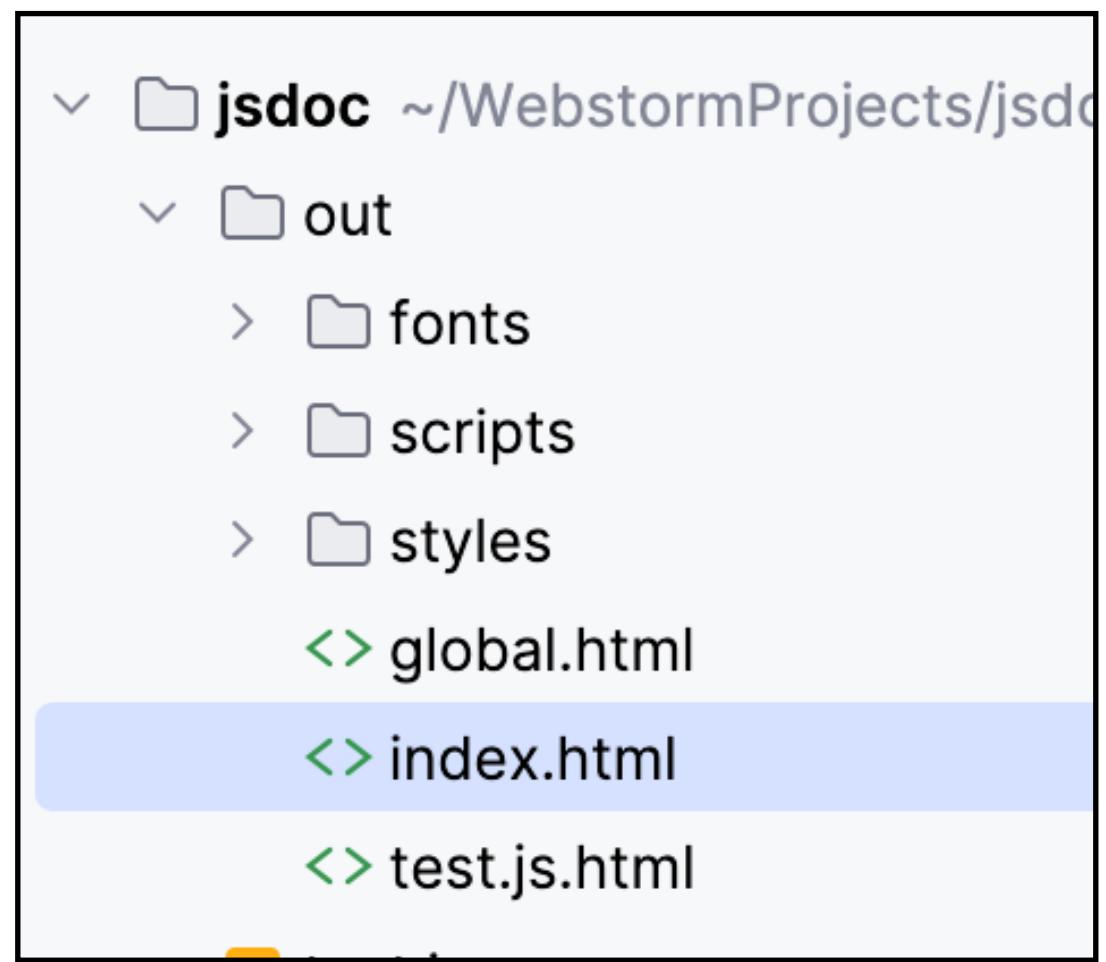
- Po zainstalowaniu JSDocs możesz uruchomić następujące polecenie, aby wygenerować dokumentację dla pliku o nazwie `index.js`.

```
jsdoc index.js
```

- Uruchamianie JSDocs na pojedynczym pliku nie jest zbyt przydatne. Poniższy przykład uruchomi rekursively generowanie JSDocs dla wszystkich plików w pliku `src` informator:

```
jsdoc src -r
```

DOKUMENTACJA JSDOC



Global

Methods

`setMagicNumber(x)`

Set the magic number.

Parameters:

Name	Type	Description
x	<code>zupa</code>	The magic number.

Source: [test.js, line 10](#)

Type Definitions

`zupa`

A number, or a string containing a number.

Type:

- `number | string`

[Home](#)

Global

`setMagicNumber`

DOKUMENTOWANIE KODU NA EGZAMINIE INF.04

INF.04-01-24.01-SG

Część III. Dokumentacja aplikacji

Wykonaj dokumentację do aplikacji utworzonych na egzaminie. W kodzie źródłowym aplikacji konsolowej utwórz komentarz do dowolnej funkcji, według wzoru z listingu 1. Komentarz powinien znaleźć się nad funkcją lub pod jej nagłówkiem. W miejscu nawiasów <> należy podać odpowiednie opisy. W miejscu autor należy podać numer zdającego.

UWAGA: Dokumentację należy umieścić w komentarzu (wieloliniowym lub kilku jednoliniowych). Znajdujący się w listingu 1 wzór dokumentacji jest bez znaków początku i końca komentarza, gdyż te są różne dla różnych języków programowania

Listing 1. Wzór dokumentacji funkcji (liczba gwiazdek dowolna)

```
*****  
nazwa funkcji:      <nazwa>  
opis funkcji:       <krótki opis co robi funkcja>  
parametry:          <nazwa parametru - opis >  
zwracany typ i opis: <nazwa typu i opis co jest zwracane>  
autor:              <numer zdającego>  
*****
```

INF_04_2024_01_01_SD

Część III. Dokumentacja aplikacji konsolowej

Wykonaj dokumentację do aplikacji utworzonych na egzaminie. W kodzie źródłowym aplikacji konsolowej utwórz komentarz do klasy, według wzoru z listingu 1. Komentarz powinien znaleźć się nad klasą lub pod jej nagłówkiem. W miejscu nawiasów <> należy podać odpowiednie opisy. W miejscu autor należy podać numer zdającego.

UWAGA: Dokumentację należy umieścić w komentarzu (wieloliniowym lub kilku jednoliniowych). Znajdujący się w listingu 1 wzór dokumentacji jest bez znaków początku i końca komentarza, gdyż te są różne dla różnych języków programowania

Listing 1. Wzór dokumentacji klasy (liczba gwiazdek dowolna)

```
*****  
klasa: <nazwa klasy>  
opis: <co klasa reprezentuje i wykonuje?>  
metody: <nazwa metody1 - co zwraca>  
        <nazwa metody2 - co zwraca>  
autor: <numer zdającego>  
*****
```

ZADANIA

ZADANIA

ZADANIE 1

- 1. Stwórz folder jsdoc_twojeNazwisko_twojelmie**
- 2. Zainstaluj według komend jsdoc**
- 3. Utwórz w nim folder src**
- 4. Utwórz pliki: divide.js, cricleArea.j, isAdult.js, hello.js**

ZADANIA

ZADANIE 2 Dzielenie

W pliku divide.js napisz funkcję divide, która wykonuje dzielenie dwóch liczb. Upewnij się, że funkcja obsługuje przypadek dzielenia przez zero.

Za pomocą jsdoc napisz:

- krótki opis co robi funkcja,
- parametry (typ i opis),
- zwracany typ i opis,
- przykład zastosowania funkcji
- rzucany wyjątek (błąd, gdy mianownik jest równy zero)
- autor (proszę poszukać w dokumentacji jsdoc) imię nazwisko klasa

ZADANIA

Zadanie 3

W pliku `cricleArea.js` napisz funkcję `calculateArea`, która przyjmuje jeden parametr `radius` (liczba) i zwraca pole koła obliczone na podstawie podanego promienia. Promień musi być liczbą dodatnią większą od zera.

Użyj JSDoc do dodania odpowiedniej dokumentacji dla tej funkcji.

Za pomocą jsdoc napisz:

- krótki opis co robi funkcja,
- parametry (typ i opis),
- zwracany typ i opis,
- rzucany wyjątek (błąd, gdy promień jest liczbą ujemną lub równą zero)
- autor (proszę poszukać w dokumentacji jsdoc) imię nazwisko klasa

ZADANIA

Zadanie 4

W pliku `isAdult` napisz funkcję `isAdult`, która sprawdza, czy osoba jest pełnoletnia. Funkcja przyjmuje jeden parametr `age` (liczba) i zwraca wartość typu boolean (true lub false), w zależności od tego, czy podany wiek jest równy lub większy niż 18 lat (wtedy true). Wiek musi być liczbą dodatnią większą od zera.

Użyj JSDoc do dodania odpowiedniej dokumentacji dla tej funkcji.

Za pomocą jsdoc napisz:

- krótki opis co robi funkcja,
- parametry (typ i opis),
- zwracany typ i opis,
- rzucany wyjątek (błąd, gdy wiek jest mniejszy bądź równy zero)
- autor (proszę poszukać w dokumentacji jsdoc) imię nazwisko klasa

ZADANIA

Zadanie 5

W pliku hello.js przekopij funkcję i udokumentuj ją za pomocą JSDoc.

```
const sayHello = (somebody) => {
  alert(`Hello ${somebody}`);
};
```

<https://developer.mozilla.org/en-US/docs/Web/API/Window/alert>

Za pomocą jsdoc napisz:

- krótki opis co robi funkcja,
- parametry (typ i opis),
- zwracany typ i opis (polecam wejść w linka powyżej),
- autor (proszę poszukać w dokumentacji jsdoc) imię nazwisko klasa

ZADANIA

Zadanie 6

Wygeneruj Dokumentację dla całego foldera src.

Folder jsdoc_twojeNazwisko_twojelmie spakuj w zip i prześlij