**Szkolenie 9**

**Funkcje Rozszerzenie**

**Wstęp**

Szkolenie to będzie uzupełnieniem Szkolenia nr 7, w którym poznaliśmy ideę wykorzystania funkcji. To szkolenie chciałbym poświęcić rozszerzeniu wiadomości właśnie z ich zakresu. Nie będzie to duża ilość wiadomości, jednak postanowiłem pozostawić ją na oddzielny materiał - aby nie przeciążać Cię ilością nowych informacji w pojedynczym szkoleniu. Tak więc, zanim przejdziesz do zapoznawania się z niżej przedstawionymi informacjami, upewnij się, że w pełni opanowałeś dotychczas przedstawione informacje.

**Wyrażenia funkcyjne**

W poprzednich szkoleniach poznałeś jeden ze sposobów definiowania funkcji, np.

| function showMessage(){  console.log("Hi User!"); } |
| --- |

Powinieneś wiedzieć, że istnieje również inny sposób na tworzenie funkcji. Może się to odbywać przez, tzw. wyrażenia funkcyjne. W takim podejściu wszystko sprowadza się do tego, aby odwołanie do funkcji znalazło się pod zmienną. Wówczas, aby wywołać funkcję, po prostu wykorzystamy nazwę danej zmiennej. Spójrz poniżej:

| let message = function showMessage(){  console.log("Hi User!"); }; |
| --- |

Użycie tej zmiennej będzie wyglądało następująco:

message();

Co spowoduje wywołanie funkcji showMessage i wyświetlenie komunikatu *Hi User!*

Zauważ jednak, że od tego momentu, gdy będzie próbowali wywołać funkcję po jej właściwej nazwie (nie po nazwie zmiennej, w której jest umieszczona), czyli:

showMessage()

Interpreter wskaże nam na błąd:

| Uncaught ReferenceError: showMessage is not defined |
| --- |

Dzieje się tak, ponieważ funkcja “przypisana” do zmiennej jest dostępna tylko przez odwołanie do niej za pomocą nazwy takiej zmiennej. Funkcja “showMessage” została jedynie zdefiniowana “w locie” i nie jest widoczna globalnie w programie.

Czyli możesz zapytać: Po co w takim razie nazywać w jakikolwiek sposób tworzoną funkcję? Bardzo słuszna uwaga! Otóż równie dobrze nasz kod moglibyśmy przekształcić do takiej postaci:

| let message = function() {  console.log("Hi User!"); }; |
| --- |

Zauważ, że w powyższym przykładzie zrezygnowałem z zapisania nazwy funkcji, użyłem jedynie słowa kluczowego function wraz z pustą listą parametrów wyrażającą się przez (). To, co tak naprawdę utworzyliśmy, to funkcja bez nazwy (przypisana do zmiennej), czyli inaczej **funkcja anonimowa**.

Bez problemu możemy stworzyć funkcję takiego rodzaju, która przyjmować będzie dowolne argumenty:

| let sayHello = function(name){  console.log(`Hi, ${name}`); }; sayHello("Kacper"); // Hi, Kacper |
| --- |

**Możliwości płynące z przypisania funkcji do zmiennej**

Poznałeś już sposób na przypisanie funkcji do zmiennej. Dzięki takiej możliwości, kod staje się o wiele bardziej elastyczny i uniwersalny. Okazuje się bowiem, że skoro funkcja może być przypisana do zmiennej to również może być przekazana jako argument innej funkcji. Może to okazać się bardzo praktyczne, gdy będziemy chcieli uzależniać wywołanie pewnych jednej funkcji od drugiej. Taki proces przekazywania (i później wykorzystania) jednej funkcji do drugiej nazywamy **callback-iem**. Spójrz na przykładowy program:

| function manager(hello, goodbye){  let choice = prompt("Podaj nazwę funkcji, którą chcesz wywołać: ")  if(choice == "hello"){  alert("Wywołuję funkcję hello!");  hello();  }  else if(choice == "goodbye"){  alert("Wywołuję funkcję goodbye!");  goodbye();  }  else{  alert("Nieznany wybór!");  } } manager(  function(){  alert("Hello, User!");  },  function(){  alert("Goodbye, User!");  }  ); |
| --- |

Kluczową częścią powyższego przykładu jest wywołanie funkcji manager. Zauważ, że pod parametry hello oraz goodbye, wysyłamy dwie funkcje anonimowe. To właśnie one są wywoływane, gdy użytkownik poda odpowiednią nazwę hello lub goodbye.

**Funkcja strzałkowa, czyli jeszcze krótszy zapis funkcji?**

Jakby tego było mało, funkcja strzałkowa umożliwia nam je jeszcze bardziej skrócenie kodu! Wówczas, aby stworzyć funkcję, wykorzystamy jedynie nawiasy, strzałkę (=>) oraz ewentualne klamry. Spójrz, jak łatwo od teraz możemy modyfikować definicje funkcji:

**Bezparametrowe wyrażenie funkcyjne:**

| ley saySth = function(){ // funkcja bezparametrowa  console.log("Say sth!"); }; saySth(); |
| --- |

**Odpowiednik w postaci funkcji strzałkowej:**

| let saySth = () => {  console.log("Say sth!"); }; saySth(); |
| --- |

**Wyrażenie funkcyjne z jednym parametrem:**

| let sayHello = function(name){ // funkcja z jednym parametrem  console.log(`Hi, ${name}`); }; sayHello('Kacper'); |
| --- |

**Odpowiednik w postaci funkcji strzałkowej z jednym parametrem:**

| let sayHello = (name) => {  console.log(`Hi, ${name}`); }; sayHello('Kacper'); |
| --- |

Jak widzisz z powyższych przykładów, główną różnicą między przedstawionymi sposobami zapisu funkcji, jest zamiana słowa function na zapis przypominający strzałkę.

**Zasady skracania zapisu:**

W zasadzie powyższe informacje w większości wyczerpują omawiane zagadnienie, jednak przedstawię Ci pewne zasady, które warto zapamiętać, gdy tworzymy kod z wykorzystaniem funkcji strzałkowych:

- Jeżeli funkcja wymaga tylko jednego parametru, możemy (ale nie musimy) dodatkowo pominąć nawiasy:

| let sayHello = (name) => { // zapis z nawiasem  console.log(`Hi, ${name}`); };  let sayHello = name => { // zapis bez nawiasu  console.log(`Hi, ${name}`); }; |
| --- |

Pamiętaj jednak, że jeżeli nawiasów jest więcej, lub nie ma żadnego, nawiasy muszą zostać!

| let saySth = () => { // tutaj nawiasu nie możemy usunąć...  console.log("Say sth!"); };  array = ['apple', 'orange', 'banana'] array.forEach((item, index, array) => { // ...tutaj również  alert(item); }); |
| --- |

- Jeżeli funkcja ma ciało w postaci jednej linii, możemy pominąć klamry (jednak nie jest to najlepsza praktyka, tak jak wspominałem już w szkoleniu dotyczącym porównań i instrukcji warunkowych).

| let saySth = () => console.log("Say sth!"); |
| --- |

**Funkcje czasowe**

Będąc już przy temacie funkcji, chciałbym zaprezentować Ci, jak w łatwy sposób przeprowadzać operacje związane z czasem. Do realizacji tego założenia, wykorzystamy obiekt Date().

| const now = new Date(); console.log(now); // Fri Aug 27 2021 16:24:52 GMT+0200 (Central European Summer Time) |
| --- |

Powyżej prosty przykład tego, co potrafi biblioteka czasu (pobraliśmy aktualny czas i datę urządzenia, na którym wykonujemy kod). Na rzecz zmiennej now możemy wykonywać szereg różnych funkcjonalności, które pozwalą nam wyłuskiwać wybrane informacje o aktualnym czasie. Niektóre z poniższych funkcji przedstawiam poniżej:

**Przykładowe funkcje do wykorzystania z obiektem Date:**

| **getDate()** | - zwraca dzień miesiąca |
| --- | --- |
| **getDay()** | - zwraca dzień tygodnia (0 dla niedzieli, 1 dla poniedziałku, 2 dla wtorku itd) |
| **getYear()** | - zwraca liczbę reprezentującą rok (dla lat 1900 - 1999 jest to wartość skrócona, np. 99, a dla późniejszych jest to liczba 4-cyfrowa np. 2002) |
| **getFullYear()** | - zwraca pełną liczbę reprezentującą rok (np. 2021 lub 1989) |
| **getHours()** | - zwraca aktualną godzinę (wartość z przedziału 0 - 23) |
| **getMinutes()** | - zwraca minuty (wartość z przedziału 0 - 59) |
| **getMonth()** | - zwraca aktualny miesiąc (0 - styczeń, 1 - luty) |
| **getSeconds()** | - zwraca aktualną liczbę sekund (wartość z przedziału 0 - 59) |

**Kod:**

| const now = new Date(); const xmas = new Date('December 25, 1990 23:20:10'); console.log("Cała data: ", now); console.log("Data świąt: ", xmas)  console.log("Dzień świąt: ", xmas.getDay()); // 2 console.log("Rok świąt: ", xmas.getYear()); // 90 console.log("Pełny rok teraz: ", now.getFullYear()); // 2021 console.log("Godzina świąt: ", xmas.getHours()); // 23 console.log("Miesiąc teraz: ", now.getMonth()); // 9 console.log("Sekundy teraz:", now.getSeconds()); // 42 console.log("Minuty świąt: ", xmas.getMinutes()); // 20 |
| --- |

**UWAGA:**

1. Dość podchwytliwą metodą jest .getDay(). Zauważ, że zwraca on dzień tygodnia, a nie dzień miesiąca. W celu realizacji drugiego założenia służy .getDate().

2. Numer miesiąc jest naliczany od wartości 0. Tak więc styczeń będzie miał swój odpowiednik liczbowy jako 0, luty jako 1, marzec jako 2 itd.

**Formatowanie daty**

Od teraz, bazując na powyższej wiedzy, możemy bez problemu skorzystać z poznanej funkcjonalności, aby wyświetlać sformatowaną postać daty.

| const formatted = `Obecny czas to: ${now.getHours()} h : ${now.getMinutes()} m : ${now.getSeconds()} s Dnia: ${now.getDate()} -- ${now.getMonth()+1} -- ${now.getFullYear()}` |
| --- |

**Ustawianie daty i czasu**

Do tej pory poznałeś, w jaki sposób pobierać aktualną datę oraz nieśmiało przedstawiłem Ci również sposób na ustawienie innej wartości daty (new Date('December 25, 1990 23:20:10')). Poniżej zaprezentuję Ci inne sposoby na zainicjalizowanie obiektu daty z określonym czasem:

**Przez podanie ciągu: rok, miesiąc, dzień…**

Wówczas stworzenie obiektu będącego datą, w której piszę to szkolenie, będzie miało następującą postać:

| const time = new Date(2021, 7, 31, 16, 40, 18); |
| --- |

Czyli 31 sierpnia 2021 rok, godzina 16:40:18 (sierpień jest reprezentowany w postaci wartości 7, ponieważ liczymy od 0).

Przy czym, inicjalizując datę, nie musimy podawać wszystkich wartości. Możemy pominąć podawanie dowolnych z nich, np. poniższy zapis...

| const time = new Date(2021, 7, 31); |
| --- |

… utworzy obiekt daty o roku 2021, miesiącu sierpień i dniu 31, a wartości za godzinę, minutę i sekundy zostaną domyślnie ustawione na 0.

**Przez wykorzystanie metod set()**

Drugim sposobem na dostosowanie postaci daty do swoich potrzeb, może być wykorzystanie takich metod jak setYear, setMonth, setDate itd. Dzięki temu datę możemy dostosowywać w dowolnym momencie pracy z kodem (niekoniecznie podczas tworzenia i inicjalizacji obiektu).

| const time = new Date(); time.setYear(2021); time.setMonth(7);  time.setDate(31); time.setHours(16); time.setMinutes(50); time.setSeconds(12); |
| --- |