Temat 12

Temat: Wybrane obiekty wbudowane JS – cz.1

Wbudowane obiekty JavaScript

JavaScript ma wbudowane konstruktory. Niektóre z nich:

```
let a = new Object();    // A new Object object
let b = new String();    // A new String object
let c = new Number();    // A new Number object
let d = new Boolean();   // A new Boolean object
```

W JavaScript ciągi znaków mogą być tworzone jako obiekty poprzez:

```
const name = new String ('John');
console.log(name); // "John"
```

Liczby mogą być tworzone jako obiekty poprzez:

```
const number = new Number (57);
console.log(number); // 57
```

i wartości logiczne:

```
const count = new Boolean(true);
console.log(count); // true
```

Uwaga : Zaleca się używanie prymitywnych typów danych i tworzenie ich w normalny sposób, takich jak `const name = 'John';`, `const number = 57;` i `const count = true;`
Nie należy deklarować łańcuchów, liczb i wartości logicznych jako obiektów, ponieważ spowalniają one program.

Obiekt window

Obiekt window reprezentuje okno przeglądarki i stoi na szczycie hierarchii obiektów. Jest to obiekt domyślny, tzn. do większości jego metod i właściwości można odwołać się bezpośrednio, pomijając jego nazwę.

Wybrane właściwości i metody obiektu window:

- **`window.alert("komunikat")`** – wyświetla okno dialogowe z komunikatem i przyciskiem „Ok."
- **`window.confirm("komunikat")`** – wyświetla okno decyzyjne z komunikatem i przyciskami „Ok." i „Anuluj"
- **`window.prompt("komunikat")`** – wyświetla okno z komunikatem oraz polem tekstowym do wprowadzania danych i przyciskami „Ok." i „Anuluj"
- **`window.open(url, name, options)`** – otwiera nowe okno ze stroną o podanym adresie URL. W miejscu **url** wstawiamy adres do strony która ma być umieszczona w nowo otwartym oknie. W miejscu **name** podajemy nazwę okna, którą będziemy wykorzystywać przy korzystaniu z atrybutu **target**. W miejscu **options** umieszczamy opcjonalnie ustawienia nowo powstałego okna. Ustawienia te mogą wyłączać nam menu standardowe, ustawiać wysokość i szerokość naszego okna itp. Właściwości są typu **boolean**. Oznacza to, że jeżeli nie podamy ich wartości (0 lub 1), to domyślnie przypisana zostanie im wartość **true** (1). W niektórych przeglądarkach trzeba zwracać uwagę, by przy wypisywaniu kolejnych właściwości nie używać spacji. Może to powodować błędy. Pamiętaj jednak, że w chwili obecnej większość przeglądarek domyślnie blokuje wszelkie próby otwarcia nowego okna, dlatego lepiej nie uzależniać działania naszej strony od nowych okien.
- **`window. close ()`** – zamyka aktualne okno. Okno główne przeglądarki pyta o potwierdzenie zamknięcia

Funkcje interwałowe - funkcje, które pozwalają odpalić kod z opóźnieniem czasowym, lub pozwalają odpalać taki kod cyklicznie co jakiś czas:

- **`setTimeout(fn, time).`**
  Służy do wywołania z opóźnieniem funkcji przekazanej w pierwszym parametrze. W drugim parametrze podajemy w milisekundach czas po jakim zostanie ta funkcja wywołana. (1000ms=1s). setTimeout wywołuje daną funkcję **tylko 1 raz**
- **`setInterval(fn, time),`** która działa bardzo podobnie do setTimeout. Kluczową różnicą jest to, że setInterval będzie odpalać funkcję co jakiś czas **cyklicznie .** Metoda `setInterval ()` będzie kontynuowała wywoływanie funkcji do momentu wywołania metody **`clearInterval ()`** lub zamknięcia okna.

Przykład 1.

```
<body>
    <h3>Obiekt window</h3>
    <p>Strona internetowa z przyciskiem po naciśnięciu którego
    otworzy się nowe okno ze stroną www.google.pl
    </p>
    <button onclick=klik()>Przejdź do stony google</button>
    <script>
        function klik() {
            window.open("http://www.google.pl");
            }
    </script>
```

Przykład 2.

```
<h3>Obiekt window</h3>
<input type="button" value="Pokaz Okno" id="buttonWindow">
<script>
    function myFunc() {
        console.log('Jakiś tekst');
    }
    setTimeout(myFunc, 3000); //odpali po 3s, sprawdź w
    konsoli

    function myWindow(url) {
    let newWindow = window.open(url, 'tytul-okna',
    'top=200,left=200,height=500,width=700');
    }
    document.getElementById('buttonWindow').addEventListener(
    'click', function() {
        myWindow('https://www.zsk.poznan.pl/');
        });
</script>
```

Przykład 3.

```
<h3>Obiekt window</h3>
<p>Kliknij przycisk, aby otworzyć nowe okno
i zamknąć je po trzech sekundach (3000 milisekund)</p>
<button onclick="openWin()">Otwórz nowe okno </button>
<script>
    function openWin() {
        let myWindow = window.open("", "Nowe okno",
        "width=200, height=100");
        myWindow.document.write("<p>To jest moje,
        nowiuteńkie</p>");
        setTimeout(function(){ myWindow.close() }, 3000);
    }
</script>
```

Przykład 4.

```
<h3>Funkcje interwałowe</h3>

<script>
    let a=1;
    let n=prompt("Do ilu policzyć?");
    function licz()
    // ustawiamy ograniczenie w wewnętrznej funkcji
    { if (a<=n){
        console.log(a);;
        a++;
        }
    }
    //bez powyższego ograniczenia setInterwal działałaby do
    zamknięcia okna
    window.setInterval(licz,1000);
</script>
```

Przykład 5.

```
<h3>Funkcje interwałowe</h3>
<button onclick="myStopFunction()">Zatrzymaj
odliczanie</button>
<p id="output"></p>
<script>
    let a=1;
    let output=document.getElementById('output');
    function licz(){
        output.innerHTML+=`${a} <br>`;
        a++;
    }
    let myVar =window.setInterval(licz,1000);
    //zatrzymanie działania funkcji setInterval:
    function myStopFunction() {
        clearInterval(myVar);
    }
</script>
```

Obiekt Date

Obiekt **`Date`** służy do przechowywania informacji o dacie i godzinie oraz wykonywaniu na nich określonych operacji. Obiekt Date musimy samodzielnie utworzyć za pomocą konstruktora **`Date().`** Konstruktor **`Date()`** bez parametrów tworzy obiekt zawierający informacje o bieżącej dacie i godzinie.

Wybrane właściwości i metody obiektu Date:

Odczyt bieżącego czasu i bieżącej daty:

- **`Date()`** –podaje bieżące: datę i czas (składnia: tylko metoda bez obiektu) - data z urządzenia danego użytkownika, a nie serwera. Może ona być inna u różnych użytkowników.

Odczyt daty:

- **`getDate()`** –zwraca liczbę z zakresu 1÷31 oznaczającą dzień miesiąca
- **`getDay()`** –zwraca liczbę z zakresu 0÷6 oznaczającą dzień tygodnia (0 –niedziela, 1 – poniedziałek, ...)
- **`getMonth()`** –zwraca liczbę z zakresu 0÷11 oznaczającą miesiąc (0 –styczeń, 1 –luty, ...)
- **`getYear()`** –zwraca dwu (między 1900 a 1999) lub czterocyfrowe oznaczenie roku (przed 1900 lub po 1999) –nie jest zalecana (zdeprecjonowana)
- **`getFullYear()`** –zwraca czterocyfrową liczbę reprezentującą rok

Odczyt czasu:

- **`getHours()`** –zwraca godzinę (liczba z zakresu 0÷23)
- **`getMinutes()`** –zwraca minuty (liczba z zakresu 0÷59)
- **`getSeconds()`** –zwraca sekundy (liczba z zakresu 0÷59)
- **`getMiliseconds()`** –zwraca milisekundy (liczba z zakresu 0÷999)
- **`getTime()`** –zwraca liczbę milisekund od 1 stycznia 1970
- **`getTimezoneOffset()`** –różnica w minutach pomiędzy czasem lokalnym a GMT

Ustawianie daty:

- **`setDate(dzień)`** –ustawienie dnia miesiąca (1÷31)
- **`setMonth(miesiąc)`** –ustawienie miesiąca (0÷11),
- **`setFullYear(rok)`** –ustawienie roku (liczba czterocyfrowa)
- **`setHours(godzina)`** –ustawienie godziny

- **`setMinutes(minuty)`** –ustawienie minut
- **`setSeconds(sekundy)`** –ustawienie sekund
- **`setMiliseconds(milisekundy)`** –ustawienie milisekund
- **`setTime(liczba_milisekund)`** –ustawienie daty i czasu względem 1 stycznia 1970 o północy poprzez dodanie lub odjęcie (wartość ujemna) liczby_milisekund
- **`toString()`** –zamienia datę na łańcuch znaków, który zwracany jest jako wynik np."Sun Jan 07 2007 15:14:42 GMT+0100"
- **`toLocaleString()`** –zamienia datę wg czasu lokalnego i zamienia go na łańcuch znaków "7 styczeń 2007 15:14:42"

Przykład 6.

```
<p>Jaki dziś dzień?</p>
<button onclick="toDay()">Wyświetl dzień</button>
<h3 id="czas2"></h3>
<p id="czas"></p>
<script>
    function toDay() {
        let dzienSlownie='';
        let data=new Date();
        let data2 = data.toLocaleString();
        let dzienLiczbowo=data.getDay();
        switch(dzienLiczbowo){
            case 0: dzienSlownie='niedziela';break;
            case 1: dzienSlownie='poniedziałek';break;
            case 2: dzienSlownie='wtorek';break;
            case 3: dzienSlownie='środa';break;
            case 4: dzienSlownie='czwartek';break;
            case 5: dzienSlownie='piątek';break;
            case 6: dzienSlownie='sobota';break;
        }
    document.getElementById("czas").innerHTML = `Dzisiaj jest ${dzienSlownie}, ${data2}`;
    }
    // inny sposób, z wykorzystaniem tablic
    let currentDate = new Date();
    let days = ["Niedziela", "Poniedziałek", "Wtorek", "Środa", "Czwartek", "Piątek", "Sobota"];
    document.getElementById('czas2').innerHTML = `Dzisiaj jest  ${days[currentDate.getDay()]},
    ${currentDate.toLocaleString()}`;
</script>
```

Przykład 7.

```
<p>Skrypt  uruchamia i zatrzymuje zegar:</p>
<p id="czas"></p>
<button onclick="myStopFunction()">Wstrzymaj czas</button>
<script>
    let myVar = setInterval(myTimer, 1000);
    function myTimer() {
        let d = new Date();
        let t = d.toLocaleTimeString();
        document.getElementById("czas").innerHTML = t;
    }
    function myStopFunction() {
        clearInterval(myVar);
    }
</script>
```

Przykład 8.

```
<h3>Wyświetlenie godziny i/lub daty</h3>
<div id="czas"></div>
<div id="czas2"></div>
<script>
    //Przykładowe wyświetlenie godziny
    const currentDate = new Date();
    console.log(currentDate.getHours());
    //lub krótszy zapis
    console.log((new Date()).getHours());
    //Wypisywanie sformatowanej daty na ekran
    //Do metody getMonth() dodajemy 1, ponieważ JS numeruje miesiące od 0
    const element = document.getElementById('czas');
    element.innerHTML =
    `Aktualna godzina: ${currentDate.getHours()}:${currentDate.getMinutes()}:${
    currentDate.getSeconds()}<br>
    Dnia: ${currentDate.getDate()}.${(currentDate.getMonth()+1)}.${currentDate.getFullYear()}`;
```

```
    //z użyciem funkcji leadingZero()-
    function leadingZero(i) {
        return (i < 10)? '0'+i : i;//operator warunkowy
    }
    const currentDate1 = new Date();
    const element1 = document.getElementById('czas2');
    element1.innerHTML = "Aktualna godzina z użyciem funkcji leadingZero: ";
    element1.innerHTML += leadingZero(currentDate1.getHours()) + ':' + leadingZero
    (currentDate1.getMinutes()) + ':' + leadingZero(currentDate1.getSeconds())
    + '<br>'
    element1.innerHTML += "Dnia: " + leadingZero(currentDate1.getDate()) + "." +
    leadingZero(currentDate1.getMonth()+1) + "." + currentDate1.getFullYear();
</script>
```

Funkcja **`leadingZero()`** wymaga podania w parametrze liczby i gdy ta jest mniejsza od 10 wówczas dopisuje do niej 0.

Obiekt Math

Obiekt ten za pomocą swoich funkcjonalności ułatwia przeprowadzanie matematycznych operacji.

Właściwości:

Math.E	- Zwraca stałą Eulera, która wynosi ok. 2.71	2.718281828459045
Math.LN2	- Zwraca logarytm dwóch, tj. ok. 0.69	0.6931471805599453
Math.LN10	- Zwraca logarytm z dziesięciu, tj. ok. 2.30	2.302585092994046
Math.LOG2E	- Zwraca logarytm o podstawie 2 z liczby E, czyli ok. 1.44	1.4426950408889634
Math.LOG10E	- Zwraca logarytm o podstawie 10 z E, czyli ok. 0.43	0.4342944819032518
Math.PI	- Zwraca wartość liczby Pi, czyli ok. 3.14	3.141592653589793

Math.SQRT1_2	- Zwraca pierwiastek kwadratowy z 0.5, czyli ok. 0.70	0.7071067811865476
Math.SQRT2	- Zwraca pierwiastek kwadratowy z 2, czyli ok. 1.41	1.4142135623730951

Metody

Math.abs(liczba)	- Zwraca wartość bezwzględną liczby
Math.ceil(liczba)	- Zwraca najmniejszą liczbę całkowitą, większą lub równą podanej liczbie
Math.exp(liczba)	- Zwraca wartość E podniesionej do potęgi wyrażonej podanym argumentem
Math.floor(liczba)	- Zwraca największą liczbę całkowitą mniejszą lub równą podanej liczbie
Math.log(liczba)	- Zwraca logarytm naturalny liczby
Math.max(liczba1, liczba2)	- Zwraca większą z dwóch liczb
Math.min(liczba1, liczba2)	- Zwraca mniejszą z dwóch liczb
Math.pow(liczba1, liczba2)	- Zwraca wartość liczby1 podniesionej do potęgi liczby2
Math.random()	- Zwraca wartość pseudolosową z przedziału [0 ; 1)
Math.round(liczba)	- Zwraca zaokrąglenie danej liczby do najbliższej liczby całkowitej
Math.sqrt(liczba)	- Zwraca pierwiastek kwadratowy liczby
Math.sin(liczba)	- Zwraca sinus liczby (podanej w radianach)
Math.tan(liczba)	- Zwraca tangens liczby (podanej w radianach)
Math.cos(liczba)	- Zwraca cosinus liczby (podanej w radianach)

Przykład 9.

```
<h3>Obiekt Math</h3>
<script>
    const value1 = 56.5;
    const value2 = 74.3;
    console.log(Math.min(value1, value2));
    console.log(Math.max(value1, value2));
    console.log(Math.max(1,3,6,2));
    console.log(Math.cos(0));
    console.log(Math.abs(-1));
    console.log(Math.round(value1));
    console.log(Math.round(-10.21));
    console.log(Math.floor(value1));
    console.log(Math.floor(-10.21));
    console.log(Math.ceil(value1));
    console.log(Math.ceil(-10.21));
</script>
```

Przykład 10.

```
<h3>Obiekt Math - liczby losowe</h3>
<p>efekt w konsoli</p>
<script>
    let number1=Math.random()//Liczba reprezentująca liczbę od 0 do 1, ale nie obejmującą 1
    console.log(number1);
    let number2=Math.floor((Math.random() * 10) + 1); //Zwraca liczbę losową od 1 do 10
    console.log(number2);
    let number3=Math.floor((Math.random() * 100) + 1); //Zwraca liczbę losową od 1 do 100
    console.log(number3);
    //liczba losowa z przedziału np. 3-7:
    let min = 3;
    let max = 7;
    let number4 =  Math.floor(Math.random() * (max-min+1) + min);
    console.log(number4);
    //losowanie 10 liczb z zakresu 1 do 10 i zapisanie ich w tablicy:
    let table1=[];
    for (let i=0;i<10;i++){
    table1[i]=Math.floor((Math.random()*10)+1);
    }
    console.log(table1);
</script>
```

Obiekt String

Obiekt String służy do przechowywania i manipulowania napisami.

Aby utworzyć obiekt typu String wystarczy zadeklarować zmienną tekstową przy pomocy literału znakowego:

```
let nazwa_zmiennej = "łańcuch_znaków"
```

lub użyć konstruktora by uzyskać obiekt:

```
let nazwa = new String("łańcuch_znaków")
```

Przykład 11.

```
<h3>Obiekt - String</h3>
<p>efekt w konsoli</p>
<script>
    // za pomocą literału
    let text1="tekst1";
    console.log(text1);
    // z uzyciem konstruktora:
    let text2=new String ("tekst2");
    console.log(text2);
</script>
```

Nie zaleca się tej drugiej metody. Spowodowane jest to tym, że obiekty są typem referencyjnym, dlatego przy operacjach możemy dostawać dziwne wyniki:

Przykład 12.

```
<h3>Obiekt - String</h3>
<p>efekt w konsoli</p>
<script>
    //deklaracja za pomocą literału
    let text1 = "abc";
    let text2 = "abc";
    console.log(text1 == text2); //true

    //deklaracja za pomocą konstruktora
    let text3 = new String("abc");
    let text4 = new String("abc");
    //false, ponieważ porównujemy 2 obiekty, a nie wartości
    console.log(text3 == text4);
    //false, ponieważ porównujemy 2 obiekty, a nie wartości
    console.log(text3 === text4);
    console.log(typeof text3); //object
</script>
```

Literały znakowe nie są obiektami String

Można jednak wywołać dowolną metodę obiektu String na wartości literału znakowego - JavaScript automatycznie skonwertuje literał znakowy do tymczasowego obiektu String, wywoła metodę, a następnie pozbędzie się tymczasowego obiektu String. Dzięki tej własności JS możesz wykorzystywać właściwości i metody obiektu String do literałów.

Wybrane właściwości i metody obiektu String:

- **length** – właściwość określająca długość napisu

Wyszukiwanie:

- **charAt(*pozycja*)** –zwraca znak występujący na określonej pozycji (pierwsza pozycja to 0)
- **charCodeAt(*pozycja*)** –zwraca kod Unicode znaku występującego na określonej pozycji
- **indexOf(*tekst*, *pozycja_początkowa*)** –zwraca pozycję, od której zaczyna się *tekst,* poszukiwanie rozpoczyna się od *pozycji_początkowej*(wielkość liter jest istotna, gdy *tekst* nie występuje zwracana jest wartość -1, drugi argument opcjonalny)
- **match()** – wyszukiwanie z użyciem wyrażeń regularnych
- **search()** – wyszukiwanie z użyciem wyrażeń regularnych lub tekstu

Zamiana tekstu:

- **replace(stary_tekst, nowy_tekst)** –zamiana w danym ciągu podciągu stary_tekst na nowy_tekst (wpisanie modyfikatora i sprawia, że wielkość liter nie ma znaczenia, a flaga g oznacza, że ma być zastąpione każde wystąpienie starego tekstu, a nie tylko pierwsze)

Wyodrębnianie podciągu znaków:

- **slice(pozycja_początkowa,pozycja_końcowa)** –wycinanie fragmentu tekstu od pozycja_początkowa do pozycja_końcowa-1 (wartość ujemna oznacza pozycję od końca, musi być spełniony warunek pozycja_początkowa < pozycja_końcowa, drugi argument opcjonalny –brak oznacza, że pozycja_końcowa= koniec łańcucha)
- **subsrtring(pozycja_początkowa,pozycja_końcowa**) – wycina tekst od pozycji początkowej do końcowej, ale bez ostatniego znaku (pozycja_końcowa-1) – zastosowanie argumentu ujemnego oznacza to samo co wpisanie zera
- **substr(pozycja_początkowa, długość)-**wycina z łańcucha liczbę znaków określoną przez długość począwszy od pozycja_początkowa (ujemna wartość pierwszego argumentu oznacza pozycję od końca, drugi argument opcjonalny –brak= koniec łańcucha)
- **split(separator, liczba)**–podzielenie łańcucha znaków na tablicę ciągów, pierwszy argument jest znakiem, który rozdziela ciągi w łańcuchu znaków (np. "." przy podziale na zdania, " " przy podziale na wyrazy, "" przy podziale na litery, itd.) , drugi opcjonalny argument służy do określenia liczby ciągów

Przykład 13.

```
<h3>Obiekt - String</h3>
<p>efekt w konsoli</p>
<script>
    let text = "Uczę się obiektów wbudowanych JS";
    console.log(text);
    console.log("długość tekstu: "+text.length);
    console.log("pierwszy znak tekstu: "+text.charAt(0));
    console.log("ostatni znak tekstu: "+text.charAt(text.length-1));
    console.log("kod pierwszego znaku: "+text.charCodeAt(0));
    console.log("pozycja słowa 'obiektów' w tekście: "+text.indexOf("obiektów"));
    console.log("text ze zmienionym słowem 'Uczę' na 'Nauczyłem': "+text.replace("Uczę",
    "Nauczyłem"));
    console.log("Text ze zmienionymi literami o na 0: "+text.replace(/o/g,"0"));
    console.log("4 początkowe litery: "+text.slice(0,4));
    console.log("słowa zapisane w tablicy: "+text.split(" "));
    console.log(text.split(" "));
</script>
```

Pozostałe:

- **toLowerCase()** –małe litery
- **toUpperCase()** –wielkie litery
- **link(*url*)** –tekst będzie wyświetlony jako hiperłącze do dokumentu o adresie *url*

Przykład 14.

Zmiana wielkości liter

```
<h3>Obiekt - String</h3>
<p>efekt w konsoli</p>
<script>
    let text = "uczę się Obiektów Wbudowanych JS";
    console.log(text);
    console.log("duże litery; "+text.toUpperCase());
    console.log("małe litery: "+text.toLowerCase());
    console.log("duża pierwsza litera tekstu: "+text.charAt(0).toUpperCase()
    + text.slice(1));
</script>
```

Metoda **`indexOf`** służy do podawania pozycji szukanego fragmentu w tekście (ale także w tablicy, bo metoda ta dostępna jest dla stringów i tablic). Jeżeli zwróconą wartością jest -1, to szukanego tekstu nie ma. Podobne działanie ma metoda **`lastIndexOf`**, podaje ona jednak numer **ostatniego** wystąpienia podtekstu

Przykład 15.

```
<h3>Obiekt - String</h3>
<p>efekt w konsoli</p>
<script>
    let text =  "Ala ma psa";
    //sprawdzamy czy ciąg "psa" istnieje
    if (text.indexOf("psa") > -1) {
        console.log("Ala ma psa" );
    } else {
    console.log("Ala ma kota" );
    }
    // sprawdzamy czy ciąg "kota" istnieje
    if (text.indexOf("kota") > -1) {
        console.log("Miau, miau" );
    } else {
    console.log("No nie, kota nie ma" );
    }
</script>
```

Przykład 16.

```
<h3>Obiekt - String</h3>
<p>efekt w konsoli</p>
<script>
    let text1="Ala ma kota i tak już jest".lastIndexOf("a");
    console.log(text1)
    //15 - bo ostatnia litera a występuje na pozycji 15

    let url = "http://nazwastrony.pl/przykladowaNazwaPliku.php";
    //korzystając z metod  tniemy url na części
    console.log( url.slice(url.lastIndexOf(".")+1) ); //php
    console.log( url.slice(url.lastIndexOf("/")+1, url.lastIndexOf(".")) );
     //przykladowaNazwaPliku
</script>
```

Pobieranie kawałka tekstu za pomocą metody substr() i metody substring()

Metoda **`substr(start,  dlugosc)`** służy do pobierania kawałka tekstu. Pierwszym jej parametrem jest początek pobieranego kawałka tekstu, a drugi opcjonalny wskazuje długość pobieranego tekstu. Jeżeli drugi parametr nie zostanie podany, wówczas pobierany kawałek będzie pobierany do końca tekstu.

Metoda **`substring(start,  stop)`** ma bardzo podobne działanie co powyższa. Różnicą jest drugi parametr, który zamiast długości wyznacza miejsce końca pobieranego kawałka. Jeżeli drugi parametr nie zostanie podany, wtedy kawałek będzie pobierany do końca tekstu. Jeżeli zostaną podane oba parametry, ale drugi będzie mniejszy od pierwszego, wtedy automatycznie zostaną one zamienione miejscami.

Przykład 17.

```
<h3>Obiekt - String</h3>
<p>efekt w konsoli</p>
<script>
    let text = "Ala ma kota";

    console.log(text.substr(0)); //Ala ma kota
    console.log(text.substr(0, 3)); //Ala
    console.log(text.substr(7, 4)); //kota
    //tekst od 4 litery do końca - "ma kota":
    console.log(text.substr(4, text.length - 4));

    console.log(text.substring(0, 3)); //Ala
    console.log(text.substring(3)); //ma kota
    console.log("Ala ma kota".substring(6, 4)); //ma
</script>
```

Przykład 18.

```
<h3>Obiekt - String</h3>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
    let txt = "Dziś pracuje mi się źle";
    // wyświetlenie zmienionego tekstu na stronie
    document.getElementById("demo1").innerHTML = txt.replace("źle",
    "super!!!").toUpperCase();
    // ilość znaków tekście
    document.getElementById("demo2").innerHTML = `użyliśmy w tekście
    ${txt.length} znaki`;
</script>
```