Instrukcja 5

1. Funkcje w JavaScript

Funkcją w języku programowania nazywa się zbiór (blok) instrukcji, które zwyczajowo mają wykonać jedno zadanie (np. obliczyć podatek VAT od wskazanej kwoty). W praktyce jedna funkcja może wykonać kilka zadań (np. zsumować wszystkie pozycje na fakturze, obliczyć VAT i zapisać fakturę do pliku pdf). Uniwersalna postać funkcji w JS:

```
function nazwaFunkcji(parameter1, parametr2,...,parametrN) {
//instrukcje
//tworzenie nowych zmiennych
//dzialania na parametrach wejsciowych,
//dalsze instrukcje...
//...
return wartosc; //opcjonana linia
}
```

Jak można wywnioskować, każda funkcja MUSI rozpocząć się od zastrzeżonego słowa function, po którym podaje się nazwę naszej funkcji. Nazwa musi być unikatowa (nie mogą istnieć dwie funkcje o tej samej nazwie w jednym skrypcie) oraz, podobnie jak nazwy zmiennych, nie może rozpoczynać się od liczby czy znaku specjalnego.
Zaraz po nazwie znajduje się zbiór ograniczony okrągłymi nawiasami (). Zbiór może być pusty bądź zawierać dowolną ilość parametrów wejściowych dla funkcji. Parametr wejściowy to po prostu zmienna, którą wprowadza się do ciała funkcji (parametry traktowane są przez funkcje jak zwykłe zmienne). Nazwy parametrów podlegają tym samym restrykcjom co nazwa funkcji – nie mogą istnieć 2 parametry (dla danej funkcji) posiadające identyczne nazwy. Przed nazwą parametrów nie należy pisać słowa kluczowego var – JavaScript automatycznie wie, że są one zmiennymi (natomiast wewnątrz funkcji przy deklaracji nowych zmiennych słowo to powinno wystąpić).
Ciało funkcji powinno składać się z co najmniej jednej instrukcji (może zawierać dowolną ilość). Istnieją pewne „umowne" ograniczenia – liczba linii (instrukcji) nie powinna przekraczać 100. Jeżeli liczba ta zostanie przekroczona to wedle konwencji inżynierskich tworzona funkcja powinna zostać podzielona na kilka mniejszych funkcji (podział powinien nastąpić wedle wykonywanych operacji – jeżeli funkcja np. wykonuje sumowanie oraz wydruk dokumentu to obie operacje powinny być przeniesione do dwóch różnych funkcji). Funkcje mogą wywoływać inne funkcje – w tym same siebie (tzw. funkcje rekurencyjne).
Jeżeli chcemy, by funkcja zwracała nam jakąś wartość to wystarczy użyć kluczowego słowa return, po którym podaje się nazwę zmiennej, której wartość ma zostać zwrócona. Zwracana wartość może być dowolnego typu. Jedynym ograniczeniem jest możliwość zwrócenia wartości tylko jednej zmiennej. Ograniczenie to można obejść poprzez zwracanie wartość w postaci obiektu bądź poprzez zmienne tablicowe (będą omówione na następnych zajęciach).
Przykłady użycia funkcji:

```
function sum(a, b) {
 return a + b;
}

var i = 5, j = 4;
var a = sum(i, j);

function someAction() {                    //funkcja może nie przyjmować żadnych parametrów
```

```
var a = 3;
a = sum(5, 4) / a;        //funkcja sum zwraca sumę 5+4 (9) i dzieli przez wartość a (3), po czym
                          //wynik przypisany jest do zmienne a
a *=  2;                  //skrócony zapis mnożenia: a = a * 2;
return a;
}
```

W dalszej części instrukcji zostaną zamieszczone przykłady funkcji.

2. Wstęp do funkcji warunkowych – operatory logiczne.

Operatory to w pewnym sensie spójniki wyrażeń. Wyrażenia to z kolei podstawowe operacje jakie dany skrypt ma wykonać. Wyrażeniem jest np. przypisanie wartości do zmiennej (operator przyrównania =), dodania dwóch zmiennych (operator +) itp. Operatory te noszą nazwę arytmetycznych (ze względu na zastosowanie).
W językach programowania mamy także inny typ operatorów – operatory porównujące oraz logiczne. Operatory porównania to:

> - operator większy niż
< - operator mniejszy niż
== - operator sprawdzający równoważność (równość)
>= - operator większe bądź równe
<= - operator mniejsze bądź równe
!= - operator nie równe
=== - operator sprawdzający równoważność (prócz sprawdzenia wartości sprawdza także typ zmiennej)
!== - operator nie równe (prócz sprawdzenia wartości sprawdza także typ zmiennej)

Dwa ostatnie operatory przydają się w wypadku gdy chcemy porównać zmienne z konkretnie tym samym typem danych. Rozważmy taki przykład:

```
var a = „5”;
var b = 5;
```

Przy porównaniu:

a == b

wynik będzie prawdziwy (bo wartość liczbowa 5 jest równa wartości liczbowej tekstu „5”)

Jednak przy porównaniu

a === b

wynik będzie już fałszem (bo wartość liczbowa 5 nie jest równa wartości tekstowej „5”);

Operatory logiczne:

(wartosc_A) || (wartosc_B) - operator logicznej sumy (wartosc_A lub wartosc_B)
(wartosc_A) && (wartosc_B) – operator logicznego iloczynu (wartosc_A ORAZ wartosc_B)
!(wartosc_A) – operator logicznej negacji (jeżeli nie wartosc_A; stosowana do wartości logicznych)

Jak działają przedstawione operatory? Używane są przede wszystkim w instrukcjach decyzyjnych języków maszynowych. Dzięki nim program „myśli” czyli reaguje odpowiednio na konkretne zachowanie użytkownika. W związku z powyższym wynik działania skryptu dla poszczególnych użytkowników będzie się zmieniał. Operatory te stosuje się także dla pętli powtórzeniowych – program może powtarzać określoną część kodu do czasu, gdy pewien warunek nie zostanie spełniony.
Typowymi przykładami funkcji decyzyjnych mogą być gry komputerowe – to od decyzji gracza zależy czy np. odblokowany zostanie kolejny etap (element decyzyjny warunkowy), ile razy zostanie powtórzony wyścig samochodowy (element pętli) czy też kiedy nastąpi pokonanie przeciwnika (element pętli + element decyzyjny w postaci trafień w określone punkty).

3. Funkcje warunkowe.

a) if...else

Jedna z podstawowych funkcji decyzyjnych w językach programistycznych. Jej składnia wygląda następująco:

```
if (warunek) {
//kod jaki ma zostac wykonany
//w przypadku gdy warunek zostanie spelniony
}
```

Funkcja sprawdzi podany warunek i jeżeli zostanie on spełniony wykona kod zawarty pomiędzy klamrami. Sprawdzanie warunku odbywa się na zasadzie potwierdzenia go bądź odrzucenia (prawda lub fałsz) Przykładowe warunki:

i > 5 – wcześniej zadeklarowana zmienna ma mieć większą wartość od 5
i == a – wartość zadeklarowanej zmiennej i ma być identyczna z wartością zmiennej a
i >= 5 – wcześniej zadeklarowana zmienna ma mieć wartość równą bądź większą od 5
i != 5 = wcześniej zadeklarowana zmienna ma mieć wartość inną (różną) od 5

Warunki można ze sobą łączyć:

(i > 5) && (i < 10) – wcześniej zadeklarowana zmienna ma mieć wartość większą od 5 lecz nie większą od 10 (to znaczy, że akceptowalne wartości to 6,7,8 oraz 9)

(i == 5) || (i == 15) – wcześniej zadeklarowana zmienna powinna mieć wartość 5 LUB 15 (tylko dla tych dwóch wartości zostanie spełniony warunek)

(i >5) && (i != 10) – wcześniej zadeklarowana zmienna powinna posiadać wartość większą od 5 i być różna od 10.

Każdy język, w tym JavaScript, pozwala na łączenie wielu warunków w ramach pojedynczej funkcji decyzyjnej. Należy jednak mieć na uwadze iż duża ilość warunków do sprawdzenia może spowodować znaczące opóźnienie działania skryptu (każdy musi zostać sprawdzony osobno). Warunek if() można przyrównać do działania funkcji natychmiastowej - jeżeli warunek (w tym wypadku parametr) zostanie spełniony to od razu wykona się kod mieszczący się w klamrach. W przeciwieństwie do regularnej funkcji warunku nie można wielokrotnie wywołać – jego działanie jest natychmiastowe. Aby wielokrotnie wywołać dany warunek należy umieścić go w funkcji. Czasami może się zdarzyć, że jeżeli dana część kodu objęta warunkiem nie wykona się (warunek nie zostanie spełniony) to chcielibyśmy, aby wykonał się kod alternatywny; przykładowo jeżeli

logowanie użytkownika nie przebiegnie prawidłowo (hasło będzie nieprawidłowe) to ma wyświetlić się informacja o problemie, który wystąpił (złe hasło) bądź jeżeli hasło będzie poprawne to ma zostać wyświetlony panel użytkownika. Do obsługi takiego zadania konstrukcja warunku powinna wyglądać następująco:

```
if (warunek) {
//kod jeżeli warunek został spelniony
}
else {
//kod jeżeli warunek nie został spelniony
}
```

Jak widać, wcześniej poznana klauzula if () jest na swoim miejscu. Dodana została dodatkowa klauzula else – kod zawarty w niej wykona się jeżeli warunek z sekcji if nie zostanie spełniony.

Może się jednak zdarzyć przypadek, w którym chcielibyśmy sprawdzić kilka warunków i dopasować wykonanie kodu dla konkretnych wartości testowanej zmiennej/zmiennych. W takim wypadku konstrukcja warunku będzie wyglądać tak:

```
if (warunek1) {
//wykona się kod jeżeli warunek1 zostanie spełniony
}
else if (warunek2) {
//wykona się kod jeżeli warunek2 zostanie spełniony
}
else if(warunekN) {
//wykona się kod jeżeli warunekN zostanie spełniony
}
else {
//jezeli żaden warunek nie zostanie spelniony to wykona się ten kod
}
```

W tym wypadku możemy tworzyć dowolną ilość kolejnych warunków else if(). Dzięki tej konstrukcji staje się możliwe np. reagowanie na konkretne przedziały czasowe (chcemy brać pod uwagę godziny: 9, 12, 15, 18, 21 i 0) i dla każdej z nich ustawić inne zdarzenia – o godzinie 9 przypomnieć o wypiciu kawy, około 12 iż zostaliśmy gdzieś umówieni, o 15 że jest to godzina obiadowa itd. itp. Opcjonalnie (nie jest to konieczne) można dodać fragment else – jeżeli żaden z warunków nie zostanie spełniony to wykona się fragment kodu zawarty w tej klauzuli (identycznie jak podczas opisu if() {} else {}).

b) switch

Funkcja switch to prostsza (dla programisty) wersja sekwencyjnej konstrukcji if() .. else if() … else. W większości języków klauzula switch działa tylko z wartościami liczbowymi (JavaScript działa także ze zmiennymi tekstowymi). Ogólna postać warunku switch wygląda następująco:

```
switch (nazwaZmiennej) {
case wartosc#0: //instrukcje;
                  //instrukcje;
                 //instrukcje;
                 break;
case wartosc#1: //instrukcje;
```

```
                //instrukcje;
                //instrukcje;
                break;
case wartosc#N: //instrukcje;
                //instrukcje;
                //nstrukcje;
                break:
default:        //instrukcje;
                //instrukcje;
                //instrukcje;
}
```

W przeciwieństwie do if() po słowie switch, zamiast podawać odpowiedni warunek, umieszcza się zmienną, która ma być brana pod uwagę podczas porównywania wartości. Porównywanie wartości zmiennej odbywa się w ciele instrukcji switch – poszczególne wartości naszej zmiennej, dla której ma wykonać się dany blok kodu, podawane są po słowie case. Oznacza to, że klauzula switch będzie przydatna wtedy, gdy chcemy porównywać tylko konkretne wartości oraz gdy zmienna ma być równa danej wartości (nie ma możliwości stworzenia warunków z odpowiednimi operatorami). Ponadto operacja switch jest bardziej złożona dla maszyny niż sekwencyjna konstrukcja if... else if... else.

W bloku switch zostało użyte słowo break. Mówi ono parserowi, że w momencie jego wystąpienia ma nie wykonywać reszty instrukcji znajdujących się w bloku warunkowym switch. Gdyby nie zostało ono użyte, a przykładowo pierwszy warunek na liście zostałby spełniony to wykonały by się także instrukcje INNYCH warunków (wszystkich kolejno wymienionych – wraz z podblokiem default).

4. Funkcje powtarzające.

W przeciwieństwie do funkcji warunkowych, funkcje powtarzające wykonują wskazany im fragment kodu do czasu aż nie zostanie spełniony warunek.

a) for

Pętla for wykonuje zawarty w niej kod określoną ilość razy. Jej postać:

```
for (zmiennaIndex; warunek; <zmiana wartosci zmiennej index>) {
//kod w petli
}
```

Jak widać pętla ta składa się z trzech osobnych poleceń. Pierwsze z nich, opisane jako 'zmiennaIndex' może wyglądać:

```
var i = 0;
i;
i = 15;
var k = a + b;
```

Reasumując za 'zmiennaIndex' można wstawić bezpośrednio zmienną/wartość lub dowolną instrukcję/operację, której wynikiem będzie jakaś wartość (wartość tą trzeba KONIECZNIE podstawić pod zmienną – można ją utworzyć na potrzebę pętli for bądź wykorzystać dowolną zmienną deklarowaną wcześniej).

Pod 'warunek' wstawia się dowolny warunek jaki ma zostać spełniony względem wartości 'zmiennaIndex'. Warunki pętli for tworzy się identycznie jak w przypadku pętli warunkowej if(). Ostatnie pole <zmiana wartosci zmiennej index> informuje nasz skrypt jak ma być zmieniona 'zmiennaIndex' przy każdym cyklu wykonania bloku skojarzonego z projektowaną pętlą. Przykład instrukcji:

i++ - zmienna zostanie za każdym wykonaniem pętli zwiększona o 1
i-- - zmienna zostanie za każdym wykonaniem pętli zmniejszona o 1
i+5 – zmienna zostanie za każdym wykonaniem pętli zwiększona o wartość 5
i/2 – zmienna zostanie za każdym wykonaniem pętli podzielona przez 2

Najistotniejszą cechą pętli for jest możliwość jej wykonania BEZ PODAWANIA wybranych bądź wszystkich wspomnianych wyżej instrukcji. Pętla for bez tych instrukcji będzie wykonywać się w nieskończoność!

```
for(;;) {
//instrukcje w tej pętli będą wykonywać się w nieskocznosc!
}

for (var i = 0; i < 5; ) {
//instrukcje w tej petli bez zmiany wartosci i w bloku instrukcji petli for beda wykonywac się w nieskonczonosc!!
i += 1;
}

for (var i = 0; i < 5; i++) {
//ta petla wykona się 5 razy (0,1,2,3,4) po czym zakoczny swoje dzialanie i wykona się dalsza część skryptu
}
//dalsze instrukcje skryptu

for (var i = 5; i > 0; i--) {
//tutaj z kolei petla także wykona się 5 razy, z tym ze liczac wstecz (5,4,3,2,1) po czym zakoczy swoje dzialanie
}
//gdy petla się zakoncz wkonaja się dalsze instrukcje skryptu
```

b) while

Kolejna z pętli programowych. Jej działanie tym różni się od pętli for, że posiada ona tylko jeden parametr – warunek, który do czasu spełnienia powodował będzie wykonywanie się pętli kodu. Trzeba pamiętać, że źle postawiony warunek może doprowadzić do wykonywania pętli w nieskończoność! (bądź blok instrukcji w pętli nie wykona się wcale). Przykłady pętli while:

```
while (i < 5) {
//instrukcje zapetlone; programista sam musi zadbac o to by i w pewnym momencie było większe od 5 (inaczej program zapetli się w nieskocznonosc)
}

var i = 6;
while (i == 5) {
//instrukcje w tej petli nie wykonaja się ani razu – warunek nie został spelniony na wejsciu petli (i
```

```
nie było rowne 5)
}

while (1 == 1) {
//petla będzie wykonywac się w nieskonczonosc!
}

while(true) {
//ta również będzie dzialac w nieskocznosc
}

while (false) {
//ta nie wykona się ani razu
}
```

c) do...while

Ostatnia z dostępnych w JavaScript pętli. Jedyną różnicą pomiędzy nią a pętlą while jest fakt, że kod w jej bloku wykona się CO NAJMNIEJ jeden raz (nawet w wypadku gdyby warunek nie był spełniony). Jej postać:

```
var i = 6;
do {
//instrukcje które maja  wykonac się w petli; należy pamietac iz wykonaja się co najmniej raz nawet gdy warunek nie zostanie spelniony!
} while (i <5);
```

W poprzednim punkcie wspomniane zostało o instrukcji (słowie) break. Ma ona zastosowanie także w przypadku pętli (zarówno for jak i while/do...while). Przykład:

```
var a = 0;
for (var i = 0; i < 10; i++) {
        a += i * i;
        if (a > 50)
                break;
}
alert (a);
```

Powyższy kod wyświetli wartość a równą 55 (gdyby pętla wykonała się pełne 10 razy wynik byłby równy 285). Dlaczego? Ponieważ w ciele pętli znajduje się instrukcja warunkowa, która co krok (iterację; kolejne wykonanie pętli) sprawdza jaka jest aktualna wartość zmiennej a – jeżeli przekroczy ona założoną wartość (w tym wypadku 50) zostanie wykonana instrukcja break. To z kolei mówi parserowi JavaScript iż pętlę należy w tej chwili przerwać. Pętle można zagnieżdżać - instrukcja break zadziała tylko na tą pętlę, w której się bezpośrednio znajduje (inaczej mówiąc – „najbliższą sobie”).

Prócz słowa break istnieje jeszcze rozkaz continue. Stosuje się go w wypadku gdy chcemy, aby np. wskazana iteracja(iteracje) została pominięta, a kolejne wykonywały się dalej. Najlepiej zilustruje to przykład:

```
var a = 0;
for (var i = 0; i < 10; i++) {
        if (i == 5)
```

```
                continue;
        a += i * i;
}
alert (a);
```

Odpowiedź będzie 260 (zamiast 285). Pominięte zostało działanie dla wartości zmiennej i = 5.

5. Obsługa błędów.

Domyślnie interpreter stara się wykonać wszystkie napisane przez programistę instrukcje. Problem polega jednak na tym, że czasami pisany skrypt może odwoływać się do np. niezdefiniowanej zmiennej i/lub funkcji (przykładowo jest wykonywany na dwóch różnych przeglądarkach z różnymi interpreterami JS, z czego jeden nie rozpoznaje którejś funkcji). Interpreter JavaScript domyślnie, po napotkaniu tego typu problemu, przerywa działanie skryptu.
Opisanej sytuacji można zaradzić. Wystarczy instrukcje, które mogą być niekompatybilne (bądź mogą powodować wadliwe działanie skryptu), umieścić w bloku obsługi błędów try{} catch{}. W podbloku try{} należy wpisać instrukcje, które mają się wykonać, jednak mogą spowodować wadliwe działanie skryptu. Natomiast w podbloku catch{} umieszcza się najczęściej instrukcje, które mają wyświetlić opis błędu (bądź wykonać alternatywny kod). Jak widać blok ten podobny jest do warunku if{} else{} z tą jednak różnicą, że przy try nie podaje się warunku. Natomiast po słowie catch w nawiasie podaje się nazwę zmiennej, która reprezentować będzie obiekt z dostępem do szczegółowych informacji na temat błędu. Przykład użycia:

```
try {
aler('Prawdopodobnie interpreter JS nie ma tej funkcji...');
}
catch (e) {
alert('Wystąpił błąd ' + e.message + '!');
}
```

lub:

```
try {
aler('Prawdopodobnie interpreter JS nie ma tej funkcji...');
}
catch (e) {
alert('Wystąpił błąd ' + e + '!');
}
```

Innym sposobem obsługi błędów jest własne stworzenie informacji w przypadku, gdy program nie zadziała po naszej myśli. Służy do tego funkcja throw. Jej działanie najlepiej będzie zrozumieć poprzez poniższy przykład:

```
<script>
function test() {
        try {
                var zmienna = document.getElementById("pole").value;
                if (zmienna == "") throw 'Zmienna jest pusta';
                if (isNaN(zmienna)) throw 'Zmienna nie jest liczbą!';
                if (zmienna < 0) throw 'Zmienna jest ujemna';
                alert("Poprawna liczba!");
        }
```

```
	catch (e) {
		alert('Pojawił się następujący problem: ' + e);
	}
}
</script>

<input id='pole' type='text'/>
<p onclick='test();'>Sprawdź dane</p>
```

Jak widać, jeżeli wartość zmiennej nas nie satysfakcjonuje możemy spowodować wyświetlenie odpowiedniej informacji. Można oczywiście uzyskać identyczny efekt za pomocą takiego kodu:

```
<script>
function test() {
	var zmienna = document.getElementById("pole").value;
	if (zmienna == "")
		alert('Pojawił się następujący problem: Zmienna jest pusta');
	else if (isNaN(zmienna))
		alert('Pojawił się następujący problem: Zmienna nie jest liczbą!');
	else if (zmienna < 0)
		alert('Pojawił się następujący problem: Zmienna jest ujemna');
	else
		alert("Poprawna liczba!");
}
</script>
<input id='pole' type='text'/>
<p onclick='test();'>Sprawdź dane</p>
```

Działanie będzie identyczne, jednak rozwiązanie jest o wiele mniej „rozwijalne” - za każdym razem trzeba pisać pełny opis błędu. Można oczywiście zapisywać opis błędu do zmiennych i później go wyświetlać (proszę spróbować przebudować kod samodzielnie). Wybierając własną obsługę błędów trzeba jednak pamiętać, że inni programiści, którzy będą rozwijać przez nas pisany kod mogą z początku mieć problemy aby zrozumieć nasze rozwiązanie.

6. Łączenie funkcji JavaScript z HTML.

Funkcje JavaScript wywołuje się poprzez zdarzenia znaczników HTML. Większość zdarzeń można przypisać do wszystkich znaczników dokumentu, jednak niektóre znaczniki posiadają swoje własne zdarzenia (przykładowo element <body>).
Lista najpopularniejszych zdarzeń znacznika <body>:
- onload – skrypt wykona się gdy strona zostanie załadowana
- onunload – skrypt wykona się w chwili gdy okno (zakładka) ze stroną będzie zamykana
- onresize – skrypt będzie aktywowany w chwili gdy okno przeglądarki będzie zmieniać swój rozmiar

Lista najpopularniejszych zdarzeń dla pól formularzy:
- onblur – skrypt wykona się gdy element formularza straci zogniskowanie (kursor wprowadzania tekstu zostanie przeniesiony w inne pole)
- onfocus – skrypt wykona się gdy element formularza zyska zogniskowanie (kursor wprowadzania tekstu znajdzie się we wskazanym polu)
- onchange – skrypt wykona się gdy użytkownik będzie zmieniał zawartość elementu formularza
- onsubmit – skrypt wykona się gdy formularz będzie przesyłany

Lista najpopularniejszych zdarzeń dla myszy:
- onclick – skrypt wykona się gdy element zostanie kliknięty
- ondblclick – skrypt wykona się gdy element zostanie podwójnie kliknięty
- onmousedown – skrypt wykona się gdy przycisk myszy będzie wciśnięty nad danym elementem
- onmouseup – skrypt wykona się gdy przycisk myszy zostanie odciśnięty nad danym elementem
- onmousemove – skrypt wykona się gdy kursor myszy będzie poruszał się nad elementem
- onmouseover – skrypt wykona się gdy kursor myszy będzie znajdował się nad danym elementem
- onmouseout – skrypt wykona się gdy kursor myszy opuści pole danego elementu

Lista zdarzeń dla klawiatury:
- onkeypress – skrypt wykona się gdy zostanie wciśnięty klawisz
- onkeydown – skrypt wykona się gdy zostanie przytrzymany klawisz
- onkeyup – skrypt wykona się gdy zostanie odciśnięty klawisz

Pełna lista wszystkich zdarzeń (wraz z nowymi, wprowadzonymi w standardzie HTML5) można znaleźć pod adresem http://www.w3schools.com/tags/ref_eventattributes.asp.

7. Włączanie zewnętrznych skryptów JavaScript do istniejącego kodu HTML

Podobnie jak w przypadku CSS, skrypty JavaScript można pisać w osobnych plikach, które następnie da się podłączyć do istniejącego kodu strony WWW. W celu włączenia kodu wystarczy użyć konstrukcji

```
<script src="skrypt.js"></script>
```

Wszystkie funkcje mieszczące się w pliku skrypt.js staną się dostępne w dokumencie, do którego został on dołączony.

Przykład:

```
//plik skrypt.js
function test() {
        alert('Funkcja wywołana z zewnętrznego pliku!');
}

//plik strona.html
<!DOCTYPE html>
<html>
<head>
<script src='skrypt.js'></script>
</head>
<body>
<p onclick='test();'>Kliknij mnie!</p>
</body>
</html>
```

8. Przykłady podsumowujące.

a) prosty sumator dwóch liczb

```
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="utf-8"/>
<script>//<![CDATA[
function sum() {
var a = new Number(document.getElementById("value_a").value); //bez new Number() zmienne
var b = new Number(document.getElementById("value_b").value); //bylyby znakami!!
if (!isNaN(a) && !isNaN(b))
  document.getElementById("result").innerHTML = a + b; //ta linia wypełnia znacznik result
                                                        //odpowiednim tekstem
else
  alert ("Jeden z parametrów (lub oba) nie jest liczbą!");
}
//]]></script>
</head>
<body >
Składnik a:
<input type="text" id="value_a"/><br/>
Składnik b:
<input type="text" id="value_b"/><br/>
<button onclick="sum();">Dodaj</button>
<p>Wynik działania: <span id="result"></span></p>
</body>
</html>
```

b) operacje na znakach i liczbach (działania na kilku funkcjach)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<script>//<![CDATA[
function examine() {
  var a = document.getElementById("exampleString").value;
  var b = splitString(a);
  document.getElementById("result").innerHTML = b;
  if (b != "") {
    if (!isNaN(a))
      showAlert(a);
    else
      showAlert();
  }
}

function splitString(str) {
  if (str === undefined) {
    showAlert("Ciąg znaków nie został zdefiniowany!");
    return "";
  }
  if (str == "") {
    showAlert("Ciąg znaków jest pusty!");
    return "";
```

```
  }
  var b = str.split(' ');
  var c = "";
  for (var i = 0; i < b.length; i++)
      c += b[i] + "<br/>";
  c += "Łączna liczba słów: " + b.length;
  return c;
}

function showAlert(a) {
  if (a === undefined)
    alert ("Jeden z parametrów (lub oba) nie jest liczbą!");
  else if (isNaN(a))
    alert("Wystąpił problem: " + a);
  else
    alert ("Podano liczbę: " + a);
}
//]]></script>
</head>
<body >
Wpisz dowolny ciąg znakowy:
<input type="text" id="exampleString"/><br/>
<button onclick="examine();">Dodaj</button>
<p id="result"></p>
</body>
</html>
```

c) działanie warunku switch

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<script>//<![CDATA[
function switchTest() {
  var num = document.getElementById("exampleString").value;
  switch(num) {
    case '1': document.getElementById("result").innerHTML = "Podano liczbę 1";
              break;
    case '5': var i = 5 * 4;
                document.getElementById("result").innerHTML = "5 * 4 = " + i;
              break;
    case '10':  document.getElementById("result").innerHTML = "Liczba dziesiętna...";
              break;
    default:  document.getElementById("result").innerHTML = "Nie trafiłeś!";
  }
}
//]]></script>
</head>
<body >
Wpisz dowolny ciąg znakowy:
<input type="text" id="exampleString"/><br/>
```

```
<button onclick="switchTest();">Dodaj</button>
<p id="result"></p>
</body>
</html>
```

d) while

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<script>//<![CDATA[
function switchTest() {
  var i = new Number(document.getElementById("exampleString").value) + 1;
  var index = 1;
  var retValue = 1;
  while (index < i) {
    retValue *= index;
    index++;
  }
  alert("Silnia z liczby " + (i - 1) + " wynosi: " + retValue);
}
//]]></script>
</head>
<body >
Wpisz liczbę:
<input type="text" id="exampleString"/><br/>
<button onclick="switchTest();">Oblicz silnię</button>
</body>
</html>
```

e) do... while

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<script>//<![CDATA[
function switchTest() {
  var i = 1;
  var index = 2;
  var retValue = 16;
  do {
    retValue *= index;
    index++;
  } while (retValue < 10);
  alert("Wynik: " + retValue);
}
//]]></script>
</head>
<body >
<button onclick="switchTest();">Oblicz silnię</button>
```

</body>
</html>

ZADANIA
1. Proszę sprawdzić działanie podanych przykładów. Proszę przetestować je ze zmienionymi warunkami/danymi.
2. Proszę spróbować napisać funkcję rekurencyjną.