pasja-informatyki.pl

Programowanie webowe

Mirosław Zelent

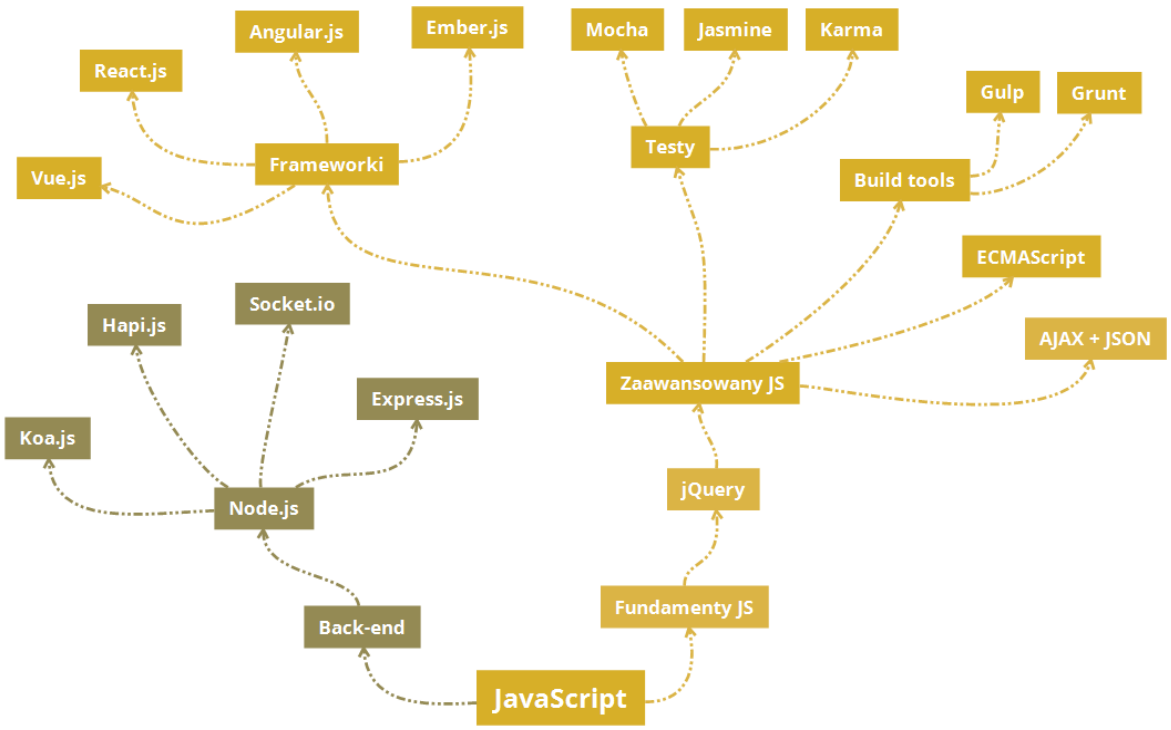
Odcinek #5

ten, w którym zrozumiemy mechanikę tworzenia skryptów z użyciem języka JavaScript

Spis treści

Język programowania JavaScript

JavaScript to pełnoprawny, skryptowy język programowania, w którym możemy zastosować cały repertuar klasycznych konstrukcji językowych (instrukcje warunkowe, pętle, zmienne, tablice, instrukcje wyboru, własne funkcje, obiekty, klasy, metody itd.). JavaScript (nie mylić z językiem Java, to co innego) to potężny front-endowy kombajn, za pomocą którego możemy tworzyć wyjątkowe od strony designu i interfejsu witryny.



Na samym początku swojej przygody z webdeveloperką, JS używamy najczęściej do ulepszenia interfejsu strony, wzbogacając ją o dodatkowe funkcjonalności, niedostępne w HTML czy CSS. Skrypty JS pozwolą nam tworzyć efektowne slidery, animowane galerie zdjęć, wyskakujące panele z nawigacją, interaktywne menu, zegary, animacje itd. Z czasem jednak przekonujemy się, że to był jedynie wstęp do prawdziwej potęgi tego języka - w pewnych zastosowaniach JS spełnia obecnie nawet rolę back-endową (technologia Node.js).

Egzamin zawodowy w technikum wymaga od nas przede wszystkim opanowania fundamentów zastosowania JS, co w realiach egzaminacyjnych oznacza najczęściej zdobycie umiejętności budowania działających od strony funkcjonalnej skryptów, które w jakiś sposób przetwarzają wprowadzone do niech dane (liczby bądź napisy). Poznajmy zatem świat zmiennych, zdarzeń, własnych funkcji, instrukcji warunkowych, wyboru, pętli – jakkolwiek może się on na początku

wydawać trudny i obcy, to przekonamy się, że bywa prawdziwie fascynujący! Zostawmy na chwilę języki opisowe (HTML i CSS) i zajmijmy się pełnoprawnym programowaniem w JS (oczywiście w kontekście omówienia fundamentów). Powodzenia!

Osadzenie skryptu JS w dokumencie HTML

Istnieją dwa podstawowe sposoby osadzenia („podpięcia") skryptu JS do kodu HTML tworzonej witryny internetowej:

1. Umieszczenie kodu źródłowego skryptu wprost pomiędzy znacznikami script:

```
<script>
    alert ("Witaj Świecie!");
    //kod JS znajduje się pomiędzy znacznikami script
</script>
```

   Wadą tego rozwiązania jest słabsze rozdzielenie warstwy zawartości strony www (tagów HTML) od jej front-endowej funkcjonalności (napisanej w JavaScript).

2. Umieszczenie kodu źródłowego skryptu w zewnętrznym pliku z rozszerzeniem *.js, po czym zainkludowanie go w dokumencie HTML

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

   Zaletą tego rozwiązania jest dobre rozdzielenie znaczników HTML od kodu JS – cały skrypt „siedzi" przecież w zewnętrznym pliku.

Zmienne. Deklarowanie zmiennych

Zmienna to najprościej mówiąc pojemnik (szufladka, pudełko) na dane – możemy w nim przechować np. liczbę, napis (w programowaniu na napis mówimy często: łańcuch), wartość logiczną (true/false) albo dane jeszcze innego typu. Oczywiście wartość przechowywana w takiej zmiennej tak naprawdę „siedzi" w pamięci RAM komputera – i rzeczywiście czasami potrzebujemy w skrypcie jakąś wartość zapamiętać.

Zmienne deklarujemy z użyciem klauzuli var (ang. variable = zmienna), po czym podajemy nazwę tej nowej szufladki w pamięci:

```
var ile_jablek = 12;
```

Nazwę wymyślamy samodzielnie, lecz uwaga: nie powinna ona zaczynać się od cyfry (ze względu na istnienie szesnastkowego zapisu liczb), nie powinna zawierać spacji, a jeżeli chcemy, aby nazwa była złożona z kilku słów, to używamy znaku podkreślenia zamiast myślnika (tak jak w przykładzie: „ile_jablek", a nie „ile-jablek"). Oczywiście nazwa zmiennej nie może też być słowem kluczowym języka JS (zmienna o nazwie na przykład „var" jest w oczywisty sposób niedopuszczalna). Osobiście polecam także unikać polskich ogonków i docelowo przyzwyczajać się do nazewnictwa angielskiego (ma to największe znaczenie, kiedy współpracujemy w zespole programistów – nazwy angielskie są najbardziej czytelne dla osób różnych narodowości).

Istnieje także w JS inny (nowszy) sposób deklarowania zmiennych, z użyciem innej klauzuli, o nazwie let (ang. „niech [się stanie zmienna]"):

```
let liczba = 20;
```

Ten nowszy sposób rozwiązuje pewne problemy z zasięgami zmiennych tworzonych klasycznie (zmienne tworzone z użyciem let mają tzw. zasięg blokowy, czyli pozostają widoczne jedynie w zasięgu wyrażenia, w którym się znajdują, zaś zmienne tworzone z użyciem var są zawsze widoczne lokalnie w całej funkcji, a nie blokowo). Oczywiście w prostych konceptualnie skryptach egzaminacyjnych (a nawet w wielu zastosowaniach produkcyjnych), ta różnica nie ma większego znaczenia.

Stałe

Oprócz zmiennych, w których przechowywane wartości mogą się dynamicznie zmieniać w trakcie wykonania skryptu, możemy także w JS zadeklarować tzw. stałą – jest to taki pojemnik na dane, którego zawartość nie powinna ulegać zmianom.

Stałą można stworzyć z użyciem klauzuli const (łac. constans = stały, niezmienny):

```
const PI = 3.141592653;
PI = 14.3; // błąd – zmiana wartości stałej!
```

Przy okazji mówienia o stałych, warto powiedzieć o niepisanej tradycji – otóż wielu programistów lubi zapisywać ich nazwy (dla odróżnienia od zmiennych) WIELKIMI literami – nie jest to obowiązek, a raczej tradycja – można jej przestrzegać, ale nie trzeba.

Uchwycenie elementu HTML w JavaScript

Wybrany element HTML (który posiada ustawiony identyfikator) możemy „uchwycić" do edycji w JavaScript z użyciem metody document.getElementById(). Uchwyt pozwala uzyskać łatwy dostęp do wielu atrybutów obiektu, które możemy nie tylko odczytać, ale także odpowiednio zmodyfikować. W przykładzie poniżej uchwycono pole edycyjne o identyfikatorze „pole", po czym odczytano wartość atrybutu value (atrybut ten przechowuje napis aktualnie znajdujący się w polu tekstowym):

HTML

```
<input type="text" id="pole">
```

JavaScript

```
var napis = document.getElementById("pole").value;
```

Po wykonaniu powyższej linii w zmiennej napis znajdować się będzie tekst pobrany z wybranego pola edycyjnego. Co ważne – sam uchwyt to jeszcze nie wartość w polu! Zwróćmy uwagę, że dopiero użycie uchwytu do dostania się do atrybutu umożliwiło nam odczyt:

```
var napis = document.getElementById("pole").value;
            ---------------------------------- -----
                      uchwyt elementu          atrybut
```

Koniecznie nie zapomnij na egzaminie o tym, iż sam uchwyt elementu (bez określonego po kropce atrybutu) nie pozwoli dokonać odczytu tekstu w polu edycyjnym (czy jakiegokolwiek innego atrybutu – zapamiętaj: „uchwyt kropka atrybut", sam uchwyt to za mało).

Wyprowadzenie wartości zmiennej na ekran

Istnieje w JavaScript wiele metod wyprowadzenia wartości przechowywanej w zmiennej (czyli w pamięci RAM) na ekran komputera – poznajmy najbardziej elementarne sposoby!

1. Metoda document.write() – największą wadą tej prostej instrukcji jest **zniszczenie dotychczasowej zawartości witryny** – cały dokument zostanie nadpisany i wypełniony nową, podaną w nawiasie zawartością (na ekranie pozostanie tylko wartość zmiennej).

```
var liczba = 12;
document.write(liczba);
```

2. Użycie okna popup typu alert() – wartość przechowywaną w zmiennej możemy wyprowadzić w malutkim, wyskakującym w przeglądarce okienku dialogowym – w prostych szkoleniowych skryptach to może wydawać się dobrym pomysłem, ale w realiach internetowych używanie alertów jest zazwyczaj uznawane za działanie inwazyjne (zniechęci internautę do odwiedzin, a sama przeglądarka posiada mechanizmy ograniczające nadużycia).

```
var liczba = 12;
alert(liczba);
```

3. Przygotowanie bloku (np. diva, wraz z nadaniem mu identyfikatora) i wyprowadzenie wartości zmiennej do jego wewnętrznego HTML – wygodna metoda pozwalająca pokazać wartość zmiennej w uprzednio przygotowanym miejscu witryny

HTML – przygotowany pojemnik z nadanym identyfikatorem:

```
<div id="wynik"></div>
```

JavaScript – zmiana wartości atrybutu innerHTML uchwyconego diva:

```
var liczba = 12;
document.getElementById("wynik").innerHTML = liczba;
```

4. Wyprowadzenie wartości zmiennej do logu konsoli w przeglądarce – log dostępny jest w zakładce *Console* (aby wywołać Narzędzia developerskie naciskamy w przeglądarce Google Chrome klawisz F12, albo używamy w witrynie prawego przycisku myszy i wybieramy z menu kontekstowego opcję *Zbadaj*).

```
var liczba = 12;
console.log(liczba);
```

Podstawowe operatory w JavaScript

Operatory to symbole reprezentujące jakieś działanie, na przykład symbol **+** może oznaczać dodawanie liczb albo konkatenację (sklejanie) napisów. Poznajmy podstawowe operatory, których znajomość jest koniecznie potrzebna na egzaminie zawodowym!

Operator	Działanie operatora
+	dodawanie liczb oraz klejenie łańcuchów (konkatenacja)
-	odejmowanie liczb
*	mnożenie liczb
/	dzielenie liczb
++	inkrementacja (zwiększenie liczby o dokładnie 1)
--	dekrementacja (zmniejszenie liczby o dokładnie 1)
%	reszta z dzielenia (tzw. modulo), na przykład: 27 % 6 jest równe 3, gdyż cztery szóstki mieszczą się w 27 (co daje 24 i reszty zostaje 3)
=	przypisanie wartości do zmiennej
==	porównanie wartości
<	mniejsze od
<=	mniejsze lub równe od
>	większe od
>=	większe lub równe od
!	zaprzeczenie (negacja)
&&	logiczne „i” (and)
||	logiczne „lub” (or)
+=	skrócony zapis, np. a += b odpowiada: a = a + b
-=	skrócony zapis, np. a -= b odpowiada: a = a - b
*=	skrócony zapis, np. a *= b odpowiada: a = a * b
/=	skrócony zapis, np. a /= b odpowiada: a = a / b
%=	skrócony zapis, np. a %= b odpowiada: a = a % b

Okna dialogowe: alert, prompt, confirm

Okna dialogowe (ang. popup = „wyskakujące okno") namiętnie wykorzystywane są w wielu „szkolnych" skryptach, dlatego koniecznie poznamy teraz ich rodzaje. Oczywiście w codziennej praktyce surfowania w internecie, witryna nie powinna używać zbyt wielu okien tego typu, by nie irytować niepotrzebnie użytkownika.

- alert – informacja dla użytkownika, wyposażona jedynie w przycisk „OK", służy do wyprowadzania danych (instrukcja wyjścia)

```
alert("Wyskakujące okienko!");
```

- confirm (ang. „potwierdzenie") – to okno dialogowe wyposażone jest w dwa przyciski: („OK" oraz „Anuluj"), co w kombinacji z instrukcją warunkową if może pozwolić nam poprosić użytkownika o podjęcie decyzji

```
if (confirm("Podejmij decyzję!")) {
  alert("Wybrano opcję: OK");
} else {
  alert("Wybrano opcję: Anuluj");
}
```

- prompt – okno, które może posłużyć do wprowadzania danych (instrukcja wejścia) – oczywiście w praktyce lepiej jest użyć pól edycyjnych, ale istnieje możliwość pobrania wartości także w oknie dialogowym. Drugi argument funkcji prompt() (w przykładzie poniżej słowo „Adam") to tzw. placeholder, czyli wartość domyślna (od razu będzie ona zaznaczona, tak aby rozpoczęcie pisania usunęło ją z okna dialogowego)

```
var imie = prompt("Podaj imię", "Adam");
document.write("Twoje imię: "+imie);
```

Reasumując: okna dialogowe to relikty przeszłości (zaczerpnięte z interfejsu graficznego systemu operacyjnego) – w praktyce warto raczej używać kontrolek HTML formularza, gdyż wyskakujące modalnie okna mogą niepotrzebnie irytować użytkownika witryny.

Instrukcja warunkowa if

Instrukcja warunkowa to po prostu rozgałęzienie w działaniu programu (ang. if = „jeżeli"). W zależności od tego, czy warunek zawarty w instrukcji jest prawdziwy lub fałszywy, wykonane zostają inne instrukcje. Klauzula else jest opcjonalna, to znaczy nie musi koniecznie wystąpić – zależy to od nas i rozpatrywanego problemu. Składnia w tzw. pseudokodzie:

```
if (warunek_logiczny)
{
   //instrukcje jeśli warunek_logiczny prawdziwy
}
else
{
   //instrukcje jeśli warunek_logiczny fałszywy
}
```

Rozważmy przykład bankomatu. Aby móc dokonać transakcji, użytkownik musi podać poprawny numer PIN. Sprawdzenia poprawności możemy dokonać instrukcją warunkową – załóżmy, że poprawny numer PIN to 1945:

```
if (pin == 1945)
{
   alert("Poprawny nr PIN");
}
else
{
   alert("Niepoprawny nr PIN!");
}
```

Zwróć uwagę na operator porównania „==", będący podwójnym znakiem równości (w odróżnieniu od operatora przypisania wartości, który jest pojedynczy: „=". Warunki mogą być złożone, zaś łącznikami są operatory: &&, ||. Używamy także negacji (zaprzeczenia), którą zapisujemy operatorem wykrzyknika: !

WARUNEK 1	WARUNEK 2	OR (||)	AND (&&)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

WARUNEK	NOT (!)
0	1
1	0

Zobaczmy teraz, jak działają oba spójniki logiczne:

- **Spójnik logiczny OR ("lub") – operator: | |**

wystarczy spełnienie co najmniej jednego warunku składowego, aby warunek złożony okazał się prawdziwy. Przykład: sprawdzenie, czy przynajmniej jedno z pól edycyjnych jest puste (nie zostało uzupełnione tekstem):

```
var a = document.getElementById("a").value;
var b = document.getElementById("b").value;

if (a == "" || b == "")
{
  alert("Proszę uzupełnić obie liczby!");
}
```

- **Spójnik logiczny AND ("i") – operator: &&**

konieczne jest spełnienie co wszystkich warunków składowych, aby warunek złożony okazał się prawdziwy. Przykład: mechanizm logowania (zarówno login jak i hasło muszą się zgadzać):

```
if (login == "admin" && haslo == "admin123")
{
   alert("Poprawne logowanie");
}
```

- **Negacja NOT („nie”) – operator: !**

zgodnie ze swoją nazwą, zawsze zmienia wartość logiczną na przeciwną (prawda zmienia się na fałsz, zaś fałsz na prawdę). Przykład: zanegowanie wykrzyknikiem warunku (liczba >= 0) sprawia, że spełni się on w sytuacji odwrotnej (zaprzeczonej), czyli kiedy liczba będzie ujemna:

```
var liczba = document.getElementById("pole").value;

if (!(liczba >= 0))
{
  alert("To jest na pewno liczba ujemna!");
} else {
  alert("To jest na pewno liczba większa lub równa zero!");
}
```

Uwaga - instrukcje warunkowe można zagnieżdżać, zatem po klauzuli else może pojawić się kolejna instrukcja warunkowa if (tak jak to miało miejsce w drugim zadaniu na filmie).

Pętle. Rodzaje pętli

Pętle (ang. loop) to konstrukcje językowe służące do zdefiniowania szeregu instrukcji, które będą powtarzane wielokrotnie. Poznajmy trzy najbardziej podstawowe rodzaje pętli: „for", „while" oraz „do .. while".

- **Pętla for** - w przypadku tej pętli z góry powinniśmy wiedzieć, ile razy instrukcje mają się wykonać. Aby zdefiniować w JS pętlę for wykonującą się np. 10 razy zapiszemy:

```
for (i = 1; i <= 10; i++)
{
   //instrukcje, które będą się powtarzać
}
```

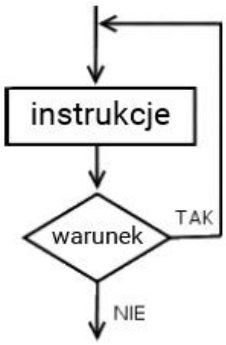
Zmienna i to liczba nazywana zmienną sterująca pętlą (albo *iteratorem pętli*). W powyższym przykładzie w pierwszym wywołaniu pętli przypisujemy jej wartość startową i=1. W każdym przebiegu pętli inkrementujemy jej wartość (zapis i++), czyli zwiększamy o dokładnie jeden. Warunek (i<=10) określa koniec iterowania (powtarzania) instrukcji. Aby najlepiej to zrozumieć, zapamiętajmy prostą regułę: pętla wykonuje się, dopóki warunek w środku zwraca wartość true (jest spełniony). Jeśli choć raz ten warunek się nie spełni, pętla zostaje zerwana (zakończona). Pętla for może także zliczać w dół:

```
for (i = 10; i >= 0; i--)
{
   //instrukcje, które będą się powtarzać
}
```

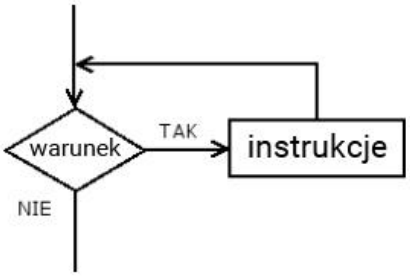
Wówczas dokonujemy w każdym kroku zmniejszenia wartości iteratora o jeden – dekrementujemy jego wartość. Zmienił się też sprawdzany warunek, tak aby był prawdziwy tak długo, jak wartość iteratora jest większa lub równa jeden. Oczywiście możemy także dodawać bądź odejmować w każdym wywołaniu pętli dowolną wartość całkowitą, zamiast inkrementować lub dekrementować (czyli zmienić wartość o dokładnie jeden):

```
for (i = 0; i <= 10; i=i+2)
{
   alert(i);
}
```

- **Pętla while, do..while** - tym razem pętle sterowane są warunkiem, a zatem instrukcje powtarzane są wielokrotnie dopóki warunek w nawiasie jest spełniony (prawdziwy). W przeciwieństwie do pętli for nie musimy od razu definiować po ilu iteracjach pętla zakończy działanie. Różnica między pętlą while i do..while polega na tym, iż w przypadku pętli while warunek sprawdzany jest na początku, zaś w do..while na końcu bloku instrukcji. Stąd linie kodu zawarte w pętli do..while wykonają się zawsze przynajmniej jeden raz:



to jest pętla do..while, gdyż najpierw jest instrukcja, zaś potem sprawdzamy warunek



to jest pętla while, bo najpierw sprawdzamy jest warunek i tylko jeśli jest spełniony wykona się instrukcja

Oto składnia pętli while ukazana na przykładzie odgadywania liczby ("strzelamy" w skrypcie, jaka to wartość może znajdować się w zmiennej „liczba”):

```
while (strzal != liczba)
{
  //instrukcje realizujące dalsze odgadywanie
  //liczby, gdyż nie trafiliśmy, liczby są różne!
}
```

Dopóki (while) liczba, którą wpisaliśmy do zmiennej "strzal" (np. pobierając ja z pola edycyjnego) jest różna od liczby w zmienne "liczba" to powtarzamy instrukcje odpowiedzialne za odgadywanie. Taka sama pętla do..while ma następującą składnię:

```
do
{
  //instrukcje realizujące dalsze odgadywanie
  //liczby, gdyż nie trafiliśmy, liczby są różne!

} while (strzal != liczba);
```

Na koniec zapamiętajmy jeszcze następującą definicję - występuje na wielu egzaminach oraz w literaturze:

- **Instrukcja iteracyjna** - jest to każda pętla (while, do..while, for) czyli instrukcja powtarzania pewnego zestawu poleceń. Nazwa wzięła się od łacińskiego słowa iteratio, itero ("powtarzanie" w wolnym tłumaczeniu). Iteracja oznacza także pojedyncze wykonanie pętli.

Własne funkcje

Funkcja to wydzielony, autonomiczny fragment kodu, który nie wykona się automatycznie sam, tylko zostanie zawołany do pracy w konkretnym miejscu kodu, który nazywamy "wywołaniem" funkcji. Wywołania dokonujemy po nazwie funkcji, koniecznie używając także nawiasów okrągłych (aby było jasne, że chodzi o nazwę funkcji, a nie zmiennej).

W języku JavaScript własne funkcje możemy także podpinać do obsługi zdarzeń (ang. event) zachodzących w przeglądarce. Przykładem może być zdarzenie click (ang. "przy kliknięciu, podczas kliknięcia"). Kiedy zajdzie zdarzenie click elementu HTML, który posiada ustawiony atrybut onclick, to wywołana zostanie funkcja o takiej nazwie, jaką zapisano w cudzysłowie atrybutu (tutaj znajduje się to logiczne połączenie pomiędzy funkcją i obsługą zdarzenia):

HTML

```
<input type="button" value="DODAWANIE" onclick="suma()">
```

JavaScript

```
function suma()
{
  //instrukcje realizujące dodawanie
}
```

Ponieważ wywołania funkcji przypisujemy do zajścia zdarzeń w przeglądarce, to kod JS wewnątrz funkcji staje się dzięki temu mniej wrażliwy na miejsce podpięcia skryptu (bo dokładnie określiliśmy moment wykonania fragmentu kodu). Kod staje się także bardziej poukładany, gdyż tworząc funkcje realizację zadań dzielimy na poszczególne autonomiczne bloki kodu.

Istnieje także alternatywny, nowszy sposób podpięcia funkcji do obsługi zdarzenia, z użyciem metody addEventListener() – pozwala ona dodać ("przypiąć") do elementu dowolną ilość "nasłuchiwaczy" danego zdarzenia (w przeciwieństwie do sposobu z atrybutem HTML, w którym jesteśmy ograniczeni do jednej funkcji obsługującej zdarzenie). Dzięki tej nowej metodzie podpinania obsługi zdarzeń lepiej też oddzielamy kod JavaScript od HTML - następuje lepsza separacja pomiędzy mechaniką a wyglądem strony (gdyż nie trzeba już używać atrybutu onclick). Przykładowy sposób wywołania metody:

HTML

```
<input type="button" value="DODAWANIE" id="przycisk">
```

JavaScript

```
var przycisk = document.getElementById('przycisk');
przycisk.addEventListener("click", function() { suma(); });
```

Dużo więcej na temat właśnie takiej obsługi zdarzeń przez własne funkcje opowiedziano w piątym odcinku kursu JavaScript.

Parametry funkcji

Do własnej funkcji możemy wysłać jakieś wartości jako tzw. parametry – umieszczamy je w nawiasach okrągłych i dowolnie nazywamy (klauzula var nie jest tym razem wymagana), a jeśli parametrów jest więcej niż jeden, to oddzielamy je przecinkiem:

```
var a = 3;
var b = 4;

alert(suma(a,b)); //wywołanie funkcji suma w alercie

function suma(x, y)
{
  return x + y;
}
```

Co ważne – te same wartości 3 i 4 (umieszczone w zmiennych: „a”, „b”) funkcja na własne potrzeby nazywa: „x”, „y”, a zatem jak widać przechowuje te liczby w innych szufladkach.

Oznacza to, iż funkcja suma() tak naprawdę działa na kopiach zmiennych „a”, „b”. Dzięki temu oryginalne wartości nie ulegną przypadkowej podmianie wartości! Dlatego też domyślnie, parametr funkcji jest jedynie kopią przekazanej do funkcji, oryginalnej wartości (zwiększamy bezpieczeństwo danych, oszczędzamy także wiele czasu przy szukaniu ewentualnych przypadkowych podmian, możemy łatwiej przenosić funkcje do innych źródeł, gdyż jest całkowicie niezależna od nazw zmiennych spoza jej zasięgu).

Parsowanie liczb

Kiedy przetwarzamy liczbę w JavaScript, którą wczytaliśmy z pole edycyjnego (nawet pola typu number), to najlepiej dodatkowo - jak to mówimy - przeparsować jej wartość. Dajemy wówczas znać komputerowi, że dana zmienna na pewno jest numeryczna i unikamy kłopotliwych pomyłek. Rozważmy następujący skrypt:

HTML

```
<input type="number" id="a">
<input type="number" id="b">

<input type="button" value="DODAWANIE" onclick="suma()">
```

JavaScript

```
function suma()
{
   var a = document.getElementById("a").value;
   var b = document.getElementById("b").value;

   alert(a + b);
}
```

Rezultatem wykonania dla np. a=5 i b=7 będzie wartość 57, a nie 12. Otóż komputer potraktował wczytane wartości jak łańcuchy (napisy) i skleił znak „5” ze znakiem „7” (operator + jest przeciążony, to znaczy może służyć także do konkatenacji). Dopiero po przeparsowaniu wartości JavaScript będzie wiedzieć, że nas interesuje dodanie do siebie dwóch liczb, a nie sklejenie ze sobą łańcuchów. Nie zapomnij o parsowaniu na swoim egzaminie zawodowym! Poprawione dodawanie zrealizowane jest następująco:

```
function suma()
{
   var a = document.getElementById("a").value;
   var b = document.getElementById("b").value;

   a = parseFloat(a);
   b = parseFloat(b);

   alert(a + b);

 }
```

Oczywiście teraz rezultatem dla np. a=5 i b=7 jest już 12. Oprócz funkcji parseFloat() istnieje także funkcja parseInt():

- parseFloat() - parsuj do liczby zmiennoprzecinkowej (ang. floating point = liczba zmiennoprzecinkowa, np. 3.14)
- parseInt() - parsuj do liczby całkowitej (ang. integer = liczba całkowita)

Instrukcja switch

Jest nazywana instrukcją wyboru, jako że pozwala wybrać zachowanie skryptu w kilku scenariuszach. Zapis „switch (zmienna)" możemy przeczytać jako: „przełącz w zależności od wartości zmiennej". Poniżej składnia przykładowego przełącznika:

```
switch (nr_miesiaca)
{
   case 1:
      alert("styczeń");
   break;

   case 2:
      alert("luty");
   break;

   case 3:
      alert("marzec");
   break;

   // ... analogicznie kontynuujemy dla 12 miesięcy

  default:

   alert("Podano nr miesiąca inny niż z przedziału 1-12");
}
```

Jak widzimy powyżej, ten switch realizuje zamianę numeru miesiąca na jego nazwę. Każdy scenariusz rozpoczyna się od słowa kluczowego „case” (ang. „w przypadku”) wraz z podaną wartością zmiennej „nr_miesiaca” (i dwukropkiem), dla której ten scenariusz zostanie wykonany. Każdy scenariusz kończy się zaś klauzulą „break”, która znaczy: zerwij w tym miejscu instrukcję switch. Brak tego słowa kluczowego np. w case: 2 sprawiłby, że kolejny scenariusz (trzeci) również wykonał by się dla wartości równej 2 (możemy więc tego „breaka” nie umieszczać, jeśli dla jednego scenariusza ma się wykonać kilka operacji normalnie przewidzianych dla innych scenariuszy.

Zwróćmy też uwagę na specjalny scenariusz oznaczony klauzulą „default” – jest to domyślne zachowanie przełącznika, jeśli zmienna sterująca (u nas: „nr_miesiaca”) przyjmie wartość spoza case’ów (przypadków) znajdujących się pomiędzy klamrami. Oczywiście ta sekcja jest opcjonalna, ale powala zareagować w przypadku zaistnienia scenariusza spoza listy.

Instrukcja wyboru switch pozwala zastąpić w kodzie kilka instrukcji warunkowych „if”, poprawiając dzięki temu przejrzystość kodu źródłowego. Oczywiście jeśli ktoś to samo zadanie zrealizuje na instrukcjach if, to nie popełnia błędu – jest to po prostu kolejna konstrukcja językowa do wykorzystania.

Obiekt Math

To w JavaScript specjalny, predefiniowany obiekt dostępny w globalnym zasięgu, który zawiera wiele metod i stałych związanych z matematyką. Poznajmy koniecznie przed egzaminem najbardziej podstawowe jego zastosowania!

- Math.PI – stała matematyczna, jak wiemy jest to stosunek obwodu koła do długości jego średnicy.

```
alert(Math.PI); // wynikiem będzie 3.141592653589793
```

- Math.E – stała Eulera (podstawa logarytmu naturalnego).

```
alert(Math.E); // wynikiem będzie 2.718281828459045
```

- Math.sqrt(x) – metoda obliczająca pierwiastek kwadratowy z liczby x.

```
var x = 2;
alert(Math.sqrt(x)); // wynikiem będzie 1.4142135623
```

- Math.sin(x), Math.cos(x), Math.tan(x), Math.asin(x), Math.acos(x), Math.atan(x) – metody obliczania funkcji trygonometrycznych (nazwy mówią same za siebie, zaś jednostką miary kąta x są radiany)

```
var x = 0.523599; // miara w radianach kąta 30 stopni
alert(Math.sin(x)); // sinus 30 stopni to 0.5
```

- Math.abs(x) – metoda zwracająca wartość bezwzględną liczby x

```
var x = -5;
alert(Math.abs(x)); // wynikiem będzie 5
```

- Math.ceil(x), Math.floor(x), Math.round(x) – zaokrąglenie liczby x – kolejno: zawsze w górę, zawsze w dół, zależnie od ostatniej cyfry (0-4 w dół, 5-9 w górę)

```
var x = 4.8;

alert(Math.ceil(x)); //wynikiem będzie 5
alert(Math.floor(x)); //wynikiem będzie 4
alert(Math.round(x)); //wynikiem będzie 5
```

- Math.random() – zwraca liczbę pseudolosową z zakresu od 0 do 1

```
// liczba pseudolosowa z przedziału 0 - 1
alert(Math.random());

// liczba pseudolosowa z przedziału 1 - 10
alert(Math.floor(Math.random() * 10) + 1);
```

- Math.pow(x, y) - zwraca liczbę x podniesioną do potęgi y

```
alert(Math.pow(2,3)); // wynikiem będzie 8
```

Tablice

W momencie, kiedy tworzymy w skrypcie zmienną rezerwujemy w pamięci RAM jeden pojemnik na dane. Jeżeli natomiast potrzebujemy zarezerwować więcej miejsc w pamięci (na przykład trzy), to możemy zastosować tablicę. Wtedy zamiast tworzyć trzy zmienne i pisać:

```
var liczba1 = 5;
var liczba2 = 8;
var liczba3 = 13;
```

możemy za jednym zamachem zadeklarować tablicę mającą trzy szufladki:

```
var liczby = [5, 8, 13]; // tablica z trzema liczbami
```

Powstają wówczas w pamięci trzy "szufladki" na liczby – ponumerowane od 0 do 3. Ponumerowane od zera, ponieważ stosowana jest notacja amerykańska, nie europejska:

Wartość przechowywana w pamięci RAM	5	8	13
Indeks szufladki	0	1	2

Ten numer szufladki nazywamy **indeksem tablicy** (zapamiętaj to pojęcie). A zatem jeżeli chcemy wyświetlić liczbę przechowaną w drugiej z kolei komórce, to posłużymy się instrukcją:

```
var liczby = [5, 8, 13]; // tablica z trzema liczbami
alert(liczby[1]); // w tej szufladce znajduje się wartość 8
```

Oczywiście tablice nie muszą przechowywać jedynie liczb, mogą też mieścić w sobie łańcuchy, wartości logiczne, znaki, itd.

Dużo więcej na temat tablic opowiedziano także w piątym odcinku kursu JavaScript.

Przetwarzanie łańcuchów w JavaScript

Łańcuchami nazywamy w programowaniu napisy (gdyż podobnie jak łańcuchy składają się z ogniw, tak napisy złożone są ze znaków). W praktyce łańcuch możemy zatem zwizualizować sobie jako tablicę (patrz poprzedni rozdział), która indeksowana jest standardowo od zera.

```
var napis = "Anna";
```

Znak (w praktyce jego kod numeryczny)	A	n	n	a
Indeks szufladki	0	1	2	3

Poznajmy podstawowe (w kontekście egzaminu) atrybuty łańcuchów oraz metody ich przetwarzania.

- **length** – atrybut pozwalający odczytać długość łańcucha:

```
var napis = "Anna";
alert(napis.length); // długość tego łańcucha będzie równa 4
```

- **charAt(x)** – kiedy chcemy uzyskać dostęp do pojedynczego znaku łańcucha (o indeksie równym x), to nie posługujmy się standardowym zapisem z użyciem nawiasów kwadratowych (nie jest on poprawnie zaimplementowany we wszystkich przeglądarkach) – bezpieczniej jest użyć metody charAt(x).

```
var napis = "Anna";
alert(napis.charAt(3)); // indeks równy 3 to litera "a"
```

- **charCodeAt(x)** metoda zwracająca kod ASCII znaku na pozycji x. Kod ASCII to numer znaku w tzw. tablicy ASCII

```
var napis = "Anna";
alert(napis.charCodeAt(3)); // Kod ASCII litery "a" to 97
```

- **search(fraza)** - metoda, która wyszukuje pozycję (indeks) w łańcuchu podanej frazy. W przypadku wielokrotnego występowania frazy można użyć metod: **indexOf(fraza)** oraz **lastIndexOf(fraza)**, które zwrócą kolejno: indeks pierwszego oraz indeks ostatniego wystąpienia podanej frazy w łańcuchu. Uwaga: jeśli frazy nie uda się odnaleźć w tekście, to metody zwrócą wartość -1 (jako umowny sygnał, że tak się stało).

```
var napis = "Anna ma kota";
var pozycja = napis.search("kot");

alert(pozycja); // pozycja frazy "kot" to indeks 8
```

- **slice(start, stop)** – metoda, która wyjmuje z napisu jego fragment i wkłada do nowego łańcucha. Wyjmujemy od indeksu start do indeks stop (już bez tego znaku). Słówko „slice" oznacza dosłownie „kroić" – łatwo zapamiętać :)

```
var napis1 = "Anna ma kota";
var napis2 = napis1.slice(8, 11);

alert(napis2); // wyjęto napis2: "kot"
```

- **substr(start, ile_znakow)** oraz **substring(start, ile)** – metody, które również wyjmują z napisu jego fragment i wkładają do nowego łańcucha. Pierwszy argument to indeks określający początek wyjmowania, zaś metody sobą różnią się drugim argumentem. W metodzie substr() podajemy ile znaków wyjmujemy, zaś w przypadku substring() drugi argument to indeks kończący wyjmowanie – ostatnim wyjętym znakiem będzie ten, który znajdował się w szufladce o indeksie o 1 mniejszym od podanego – podobnie jak to miało miejsce w metodzie slice().

```
var napis1 = "Anna ma kota";

var napis2 = napis1.substring(8, 11);
var napis3 = napis1.substr(8, 3);

alert(napis2); // wyjęto napis2: "kot"
alert(napis3); // wyjęto napis3: "kot"
```

- **replace(fraza, inna_fraza)** - metoda, która podmienia w łańcuchu podaną frazę na inną

```
var napis1 = "Anna ma kota";
var napis2 = napis1.replace("kota", "psa");

alert(napis2); // napis2 to: "Anna ma psa"
```

- **rtoLowerCase(x) oraz toUpperCase(x)** - zamiana liter łańcucha x na kolejno: małe oraz WIELKIE litery

```
var napis = "Anna ma kota";

alert(napis.toLowerCase()); // napis to: "anna ma kota"
alert(napis.toUpperCase()); // napis to: "ANNA MA KOTA"
```

Zadania do samodzielnego wykonania

Poniżej znajdziesz listę 5 zadań do samodzielnego zrealizowania – ich przepracowanie pozwoli utrwalić mechanikę realizowania skryptów:

1. Napisz program, który obliczy pole trójkąta na podstawie podanych długości trzech boków a, b, c (ale tylko pod warunkiem, że z tych boków można stworzyć trójkąt – jeśli trójkąta nie można utworzyć, to program zamiast dokonać obliczeń wypisze tekst: *Z podanych boków nie sposób utworzyć trójkąta!*
2. Napisz program o nazwie *Wyrocznia delficka*, który w interfejsie zawierać będzie napis: *„To ja, przemożna Pytia – zadaj mi w myślach pytanie, a ja odpowiem czy to o czym pomyślałeś się spełni”* oraz przycisk *„Sprawdź”*. Po kliknięciu na przycisk skrypt pokaże na ekranie napis: „Praw*dopodobieństwo, że tak będzie wynosi: 41%”* - przy czym wartość procentowa ma zostać wygenerowana pseudolosowo przy każdym kliknięciu na przycisk z przedziału 1-100 (41 to tylko przykładowe wykonanie). Dodatkowo, jeśli prawdopodobieństwo wynosi od 0-33% to napis ma być czerwony, jeśli 34-66% niebieski, a dla 67-100% zielony.
3. Napisz skrypt, który wczytuje od użytkownika dwa słowa, po czym zamienia po kliknięciu w przycisk pierwsze litery w obu wyrazach wypisując taką wersję słów pod formularzem. Na przykład: *ford mustang* po kliknięciu przekształci się w: *mord fustang*
4. Napisz skrypt, który po wczytaniu trzech liczb znajdzie najmniejszą i największą z podanych wartości.

5. Napisz skrypt, który sprawdzi, czy podany napis ma przynajmniej 6 znaków, a dodatkowo sprawdzi także, czy łańcuch zakończony jest wyrazem „kot”.
   Przykład 1: „warkot” –Wyraz ma co najmniej 6 znaków i kończy się napisem kot
   Przykład 2: „łaskotki” – Wyraz ma co najmniej 6 znaków, ale nie kończy się napisem kot
   Przykład 3: „kot” – Wyraz nie ma co najmniej 6 znaków, ale kończy się napisem kot

Test wiedzy z odcinka

Sprawdź się! :) Zapraszamy do testu wiedzy udostępnionego tutaj:
http://pasja-informatyki.pl/javascript-potrzebny-do-egzaminu-test-wiedzy/