**Zmienne i stałe**

Zmienne to coś w rodzaju "pudełek", w których pod nazwami możemy przechowywać pewne rzeczy. Znasz jakąś zmienną? Pewnie że znasz. Na przykład twoje imię jest zmienną. Zamiast mówić "ten wysoki gość, który jest najlepszy na świecie", użyję zmiennej i powiem po prostu "Marcin". Proste - prawda?  
  
Zanim przejdziemy do szczegółów językowych JS mini teoria...

**Po co stosować zmienne?**

Po co w ogóle stosować zmienne? Rozpatrzmy prosty przykład dodawania liczb:

<script>

console.log(9 + 5 + 1);

console.log(9 + 5 + 2);

console.log(9 \* 5 + 5 - 2);

console.log(9 + (5 \* 10) + 3);

console.log(9 + (5 \* 10) + 5 + 9 + (5 \* 10) + 5);

...

...

<script>

Przypuśćmy, że teraz chcielibyśmy zmienić cyfrę 5 tak, aby równanie miało postać 9 + 7 + 1 itp.  
Dla tak małego skryptu zmiana tej cyfry nie jest wielkim problemem. Co by się stało gdybyśmy takich równań w naszym przykładzie mieli na przykład tysiąc lub nawet sto tysięcy?  
Zmiana każdej linii była by dość uciążliwa. I tu przychodzą nam z pomocą nasze "pudełka":

<script>

var x = 5;

console.log(9 + x + 1);

console.log(9 + x + 2);

console.log(9 \* x + x - 2);

console.log(9 + (x \* 10) + x + 9 + (x \* 10) + x);

</script>

Pod zmienne/stałe możemy podstawiać nie tylko liczby, ale teksty, funkcje, obiekty, tablice itp. Dowiesz się o tym w następnych rozdziałach. Ogólnie więc **zmienna/stała to takie pudełko**, które zawiera lub wskazuje na "coś".

Powyższe przykłady są proste. Może coś bardziej skomplikowanego? Podstawmy pod zmienną jakiś obiekt ze strony:

<script>

var btn = document.querySelector('.btn');

console.log('Pobrany guzik: ', btn);

btn.classList.add('btn-important');

btn.setAttribute('Kliknij mnie');

console.log(btn.innerText);

btn.innerText = "Kliknij mnie!"

</script>

Zdajesz sobie sprawę, jak by wyglądał ten kilku liniowy skrypt bez użycia zmiennej? Ano mniej więcej tak:

<script>

console.log('Pobrany guzik: ', document.querySelector('.btn'));

document.querySelector('.btn').classList.add('btn-important');

document.querySelector('.btn').setAttribute('Kliknij mnie');

console.log(document.querySelector('.btn').innerText);

document.querySelector('.btn').innerText = "Kliknij mnie!"

</script>

W powyższym przykładzie zmienna pełni jeszcze jedną bardzo ważną rolę. Dzięki zmiennej bardzo zyskujemy na [wydajności](http://kursjs.pl/kurs/dom/dom.php#elementy-pod-zmienne) naszego skryptu. Ale to wiedza, którą poznasz później.

**Nazewnictwo zmiennych**

Nazwy zmiennych i stałych które deklarujemy nie mogą być byle jakie. Istnieją pewne zasady których musimy się trzymać. I tak:

* nazwa zmiennej **nie może zaczynać się od cyfry**,
* nazwa zmiennej **nie może zawierać spacji** (można zamiast spacji używać podkreślenia),
* nazwą zmiennej **nie może być** [słowo kluczowe](http://www.w3schools.com/js/js_reserved.asp) zarezerwowane przez JavaScript.

Po prostu nie zaczynaj nazw od cyfr i pisz nazwy zmiennych po angielsku. Bardzo częstą praktyką jest stosowanie zapisu camelCase, czyli np. "veryImportantThing".  
  
Jest jednak ważniejsza sprawa, którą zapamiętaj.  
**Nazywaj swoje zmienne tak, by dało się zrozumieć do czego się odnoszą.** Zmienna o nazwie **elementsCount** jest bardziej czytelne, niż **aaaaa**.

Dla ciekawskich - nazwy zmiennych mogą być bardzo różne. [Bardzo, bardzo różne](https://mathiasbynens.be/notes/javascript-identifiers). Nie jest to oczywiście zalecane i trzeba to traktować jako ciekawostkę.

**Deklarowanie zmiennych i stałych**

Wiesz już, że skrypty umieszczamy w znacznikach scripts. Od tej pory dla poprawy czytelności będę pomijał te znaczniki skupiając się na samym kodzie.

Przez lata aby zadeklarować zmienne w Javascript korzystało się (a i korzysta dalej!) tak jak w powyższych przykładach ze słowa kluczowego **var**:

var myVar = "Przykładowy tekst"; //zmienna o nazwie myVar

Słowo to pozwala deklarować zmienne.  
No ale przecież tytuł tego działu to "zmienne i stałe".  
Tak - w każdym porządnym języku mamy zmienne i stałe. Zmienne są zmienne - czyli po zadeklarowaniu możemy zmieniać ich wartość. Stałe to stałe - po deklaracji nie można zmieniać ich wartości. W JS stałe nie istniały...  
  
Niektórzy stosowali konwencję pisania nazw stałych dużymi literami np.

var COUNT = 200;

var URL = 'http://kursjs.pl';

co miało ułatwiać zorientowanie się, by potem nie zmieniać takiej zmiennej, ale tak czy siak - stałe nie istniały.  
  
Poza powyższym var ma jeszcze kilka dziwnych zachowań (o których poniżej).

Wraz z rozwojem JS doszły nam nowsze zapisy takie jak **let** (zmienna) i **const** (stała), które część problemów, które miał var naprawiają.  
  
Chociaż w wielu skryptach będziesz natrafiał na stary zapis za pomocą var:

//------------------------

//starszy zapis

//------------------------

var myVar = "Przykładowy tekst"; //zmienna o nazwie myVar

myVar = "Inny tekst";

var JAKASSTALA = 200;

JAKASSTALA = 300; //zmieniłem czyli to żadna stała

w dzisiejszych czasach zalecam stosować nowe zapisy:

//------------------------

//nowszy zapis

//------------------------

let myVar = "Przykładowy tekst"; //zmienna o nazwie myVar

myVar = "Inny tekst";

const myImportantVar = "Stała o niezmiennej wartości";

myImportantVar = "Inny tekst"; //błąd - nie można zmieniać stałej

const btn = document.querySelector('.btn'); //pobieram element ze strony i podstawiam pod stałą

console.log( btn.innerText ); //za pomocą zmiennej wypisuję tekst w buttonie

**Problemy z var**

Początek nauki, a już zaczynają się udziwnienia. Niestety musisz się do tego przyzwyczaić. Javascript w wielu miejscach ma takie smaczki.

Skoro istniało słowo var, to po co nowa wersja JS wprowadzała dodatkowe sposoby deklaracji? Wynikało to z pewnych rzeczy, które miały miejsce przy **var**

Pierwsza z nich to tak zwany **hoisting zmiennych**.  
O co chodzi? Jak nie wiadomo o co chodzi, to chodzi o kasę.  
  
Tutaj na szczęście wiadomo o co chodzi.  
Jeżeli zmienna była zadeklarowana za pomocą słowa var gdzieś w środku skryptu, js próbując ułatwić nam życie automatycznie (za naszymi plecami) przenosił jej deklarację na początek kontekstu (powiedzmy dla ułatwienia że początek skryptu). Równocześnie to wspaniałe ułatwienie sprawiało, że logika kodu mogła stać się zachwiana:

console.log(myVar); //używam zmiennej, której jeszcze nie ma. Nie ma błędu!

var myVar = 20;

Powyższy skrypt nie rzuci nam błędem, mimo, że w 1 linijce kodu odwołuję się do zmiennej, której jeszcze nie ma (pamiętaj, że skrypt czytany jest od góry do dołu).  
Działa tu właśnie wspomniany ułatwiający życie **hoisting**. Deklaracja zmiennej (bez jej wartości!) wynoszona jest automatem na początek skryptu (a w zasadzie na początek danego zakresu - np wewnątrz funkcyjnego), w wyniku czego nasz skrypt ma teoretycznie postać:

var myVar; //js przeniósł tutaj deklarację zmiennej ale bez jej wartości!

console.log(myVar); //wypisze undefined, ale błędu nie ma

var myVar = 20;

Powyższe działanie nie jest najlepsze dla logiki kodu, szczególnie w bardziej złożonych programach np.

if (a > 20) {

var name = "Marcin";

console.log(name); //wypisze Marcin

} else {

console.log(name); //zmienna name istnieje i ma wartość undefined

}

Zawsze powinniśmy na początku deklarować zmienne, a dopiero potem ich używać, dlatego niektórzy stosowali konwencję deklaracji każdej zmiennej na początku kontekstu, mimo, że początkowo nie przypisywali im wartości:

var text, age;

age = 20;

var text = "Ala ma " + age + " lat";

Dzięki czemu kod był bardziej zrozumiały i nie zaskakiwał nas niewidoczny dla nas hoisting.

Drugi problem to zasięg zmiennych zadeklarowanych za pomocą słowa var.

function testVar() {

var a = 30;

if (true) {

var a = 50;

console.log(a); /50

}

console.log(a); //50

}

testVar(); //50 50

W językach opartych o język C zmienne mają zasięg blokowy, czyli ich zasięg określają najbliższe klamry. W powyższym przykładzie więc zmienna **a** zadeklarowana wewnątrz instrukcji **if** powinna mieć zasięg wewnętrzny, więc wynik powinien być 30, 50. Wychodzi nam jednak wynik 50, 50.  
Wynika to z tego, że zmienne zadeklarowane za pomocą słowa var mają zasięg nie blokowy a funkcyjny. Oznacza to, że ich zasięg definiowany jest przez klamry najbliższej funkcji. W powyższym przykładzie na początku funkcji deklarujemy zmienną **a**, a w instrukcji **if** nie deklarujemy nowej zmiennej, a zmieniamy tą samą zmienną, bo jej zasięg określa funkcja testVar. Może to wydawać się zawiłe, i w sumie trochę takie jest. Więcej na ten temat poznasz w [dziale o funkcjach](http://kursjs.pl/kurs/super-podstawy/funkcje.php#zasieg-zmiennych).

W nowej wersji JS powyższe równanie działa podobnie jak w językach wysokiego poziomu:

function testVar() {

const a = 30;

if (true) {

const a = 50;

console.log(a); //50

}

console.log(a); //30

}

testVar(); //50 30

**Let i const**

**Przy nowych zapisach hoising już nie działa:**

console.log(myVar); //błąd - bo zmienna myVar nie istnieje

let myVar = 20;

To miejsce przed deklaracją zmiennej zwie się **temporary dead zone** (tymczasowa strefa śmierci), bo nie możemy odwoływać się do zmiennej, której jeszcze nie zadeklarowaliśmy (i dobrze!)

Poprawna wersja powinna wyglądać tak:

let myVar = 20;

console.log(myVar); //20 - wszystko cacy

Drugim problemem z var jest zasięg zmiennych tak deklarowanych.  
W każdym porządnym języku zakres zmiennych definiuje blok w jakim znajduje się deklaracja zmiennych. Blok taki definiowany jest przez klamry:

{

let myVar = 20;

}

console.log(myVar); //błąd - jestem poza blokiem, tutaj zmienna nie ma zakresu

W przypadku var, zakres zmiennej definiuje ciało funkcji, czyli znowu pojawia się rzecz, która działa inaczej niż w normalnych językach programowania:

{

var myVar = 20;

}

console.log(myVar); //20 - wszystko działa ok

Powyższym tematem dokładniej zajmiemy się przy omawianiu funkcji.

Chciałem ci tylko tutaj pokazać, że poprzedni sposób deklaracji za pomocą słowa var powodował pewne udziwnienia. Dało się z tym żyć (w końcu przez tyle lat jakoś te skrypty się pisało), szczególnie jak się do tego przyzwyczaiłeś. Ale ogólnie dziwy się działy - szczególnie gdy byłeś początkujący i nie miałem pewnych nawyków w układaniu kodu (zmienne na początek itp)  
  
Stąd właśnie zostały wprowadzone słowa kluczowe let i const.   
  
**Podsumowując ten długi i nudny wywód - do deklaracji zmiennych używaj słowa let, a dla stałych używaj const**

**Const**

No dobrze - poznaliśmy dwa nowe zapisy. Let i const. Let tworzy zmienne, którym potem możemy zmieniać wartość, a const stałe - które powinny być niezmienne.

let text = "ala";

text = "ala ma kota"; // wszystko ok, bo let to zmienna

let i = 0;

i = 10; //wszystko ok, bo let

const count = 20;

count = 30; //błąd - stałej się nie zmienia

const name = "Ala";

name = "Monika"; //błąd - znowu stała

Hej! Konwencja z pisaniem stałych dużymi literami wcale nie była taka zła. Mimo tego, że mamy teraz let i const, niektórzy programiści JS dalej jej używają... Chociaż - zauważysz, że w naszych skryptach większość zmiennych będzie deklarowana przez const, więc większość naszych zmiennych pisana była by z dużych liter. Wybór jak zwykle należy do ciebie...

Ale teraz uwaga! Przy stałych chodzi o ustawianie nowych rzeczy pod daną nazwę, a nie zmianie składowych tych rzeczy:

const apiUrl = 'https://jsonplaceholder.typicode.com';

apiUrl = "http://kursjs.pl" //błąd - podstawiliśmy nową rzecz!

const tab = [1,2,3];

tab[3] = 4; //nie ma błędu, bo tylko zmieniłem składową obiektu (tablica to obiekt)

const tab = [1,2,3];

tab = [1,2,3,4]; //błąd - podstawiłem zupełnie nową tablicę

const tab = [1,2,3];

tab = [1,2,3]; //błąd - podstawiłem zupełnie nową tablicę, nie ważne że podobnie wygląda

const currentUser = {name: "Piotr", age: 18}

currentUser.age = "Marcin"; //nie ma błędu - zmieniłem tylko składową

currentUser = {name: "Piotr", age: 20} //błąd - bo podstawiłem totalnie nowy obiekt

Dlaczego tak się dzieje? Pamiętasz w rozdziale o typach jak działają przypisania wartości do zmiennych typu prostego a jak działają dla obiektów?  
Gdy przypisywałem wartość do typu prostego, robił się oddzielny byt danej wartości. Dwie oddzielne zmienne mimo, że miały tą samą wartość były oddzielnymi bytami.  
Przy obiektach jest inaczej - tutaj działa tak zwana referencja. Zmienne wskazują tylko na dany obiekt, który sobie siedzi gdzieś tam w pamięci. Jeżeli w powyższym skrypcie dodajemy tylko coś do obiektu, nie zmieniamy tego gdzie wskazuje dana nazwa stałej. Ale jeżeli przypisujemy jej całkowicie nowy obiekt - jej wskaźnik się zmienia i mamy w rezultacie błąd.  
  
**Podsumowując. Dla stałych nie możemy na nowo przypisywać całkowicie nowych wartości. Ale jeżeli stała wskazuje na obiekt, spokojnie możemy w nim mieszać...**

Czemu o tym piszę? Bo bardzo często w tym kursie będziesz widział takie zapisy jak poniżej:

const tab = [1,2,3,4];

tab.push(5); //dodaje do tablicy element

I pojawiło by się zmieszanie - co to za stała, skoro możemy ją zmieniać. No ale to jak z nami. Jutro pójdę do fryzjera. Zmienię uczesanie. Ale to dalej ja. Więc stała "Marcin" nie wyrzuci błędu :)

Jeżeli przyzwyczaiłeś się do stosowania var, to wiedz, że to nic złego. Wiele lat wszyscy pisali z wykorzystaniem tego słowa i świat się nie zawalił.  
W tym kursie trochę umyślnie chcę ci wpajać nowe techniki.  
Nie zawsze mi się to będzie udawać, bo czasami by poznać nowsze rozwiązanie, trzeba też znać starsze. Z let i const jest na szczęście całkiem łatwo. A i nowe podejście często wymusza bardziej poprawne podejście do kodu.  
Spróbuj - najwyżej zostaniesz przy var. Sporo osób tak robi i nic im to nie szkodzi.

**Trening czyni mistrza**

Poniżej zamieszczam kilka zadań, które w ramach ćwiczenia możesz wykonać:

1. Za pomocą **var** zadeklaruj 2 numeryczne zmienne **myVar1** i **myVar2**.  
   Wypisz w konsoli wynik ich dodawania, odejmowania, mnożenie i dzielenia. Wypisz też ich typ.

Za pomocą **var** zadeklaruj dwie stałe CVAR1 i CVAR2. W kolejnych liniach kodu zmień im wartości. Czy możesz to zrobić? Czemu tak się dzieje?

Wypisz w konsoli zmienną text. W kolejnej linii zdefiniuj tą zmienną za pomocą linijki:

var text = "przykładowy tekst";

Co zobaczyłeś w konsoli? Czemu tak się dzieje?

Za pomocą **const** zadeklaruj dwie stałe CONST1 i CONST2. W kolejnych liniach kodu zmień im wartości. Czy możesz to zrobić? Czemu tak się dzieje?

Wypisz w konsoli zmienną text2. W kolejnej linii zdefiniuj tą zmienną za pomocą linijki:

let text2 = "przykładowy tekst";

Co zobaczyłeś w konsoli? Czemu tak się dzieje?