Najlepszy podręcznik do HTML5!

Rusz głową!

HTML5

Przewodnik po tworzeniu aplikacji internetowych za pomocą JavaScriptu

Poznaj sekrety Mistrza HTML5

Załaduj HTML5 i JavaScript prosto do swojego mózgu

Dowiedz się, dlaczego wszystko, co Twoi znajomi wiedzą o wideo, to zwykłe plotki

Wystrzegaj się typowych pułapek w przeglądarkach

Naucz się unikać krępujących wpadek ze wsparciem w przeglądarkach

O'REILLY®

Eric Freeman, Elisabeth Robson

Helion

Tytuł oryginału: Head First HTML5 Programming: Building Web Apps with JavaScript

Tłumaczenie: Aleksander Lamża

ISBN: 978-83-246-4339-4

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/htm5rg.zip

Spis treści (skrócony)

	Wprowadzenie	21
1.	Poznajemy HTML5. *Witaj w Webowicach*	33
2.	Wstęp do JavaScriptu i struktury DOM. *Odrobina kodu*	65
3.	Zdarzenia, ich obsługa i takie tam. *Odrobina interakcji*	113
4.	Funkcje i obiekty w JavaScripcie. *Z JavaScriptem na poważnie*	139
5.	Twoja strona wie, gdzie jesteś. *Geolokalizacja*	189
6.	Rozmawiamy z siecią. *Aplikacje otwarte na innych*	235
7.	Odkryj w sobie artystę. *Element canvas*	301
8.	Telewizja po liftingu. *Element video... gościnnie występuje canvas*	367
9.	Lokalne składowanie danych. *Mechanizm Web Storage*	429
10.	Zaprzęgamy JavaScript do pracy. *Wątki robocze*	487
	Dodatek. Dziesięć najciekawszych tematów (o których nie wspomnieliśmy)	543
	Skorowidz	561

Spis treści (z prawdziwego zdarzenia)

Wprowadzenie

Twój mózg koncentruje się na programowaniu w HTML5. *Starasz* się czegoś *nauczyć*, ale Twój *mózg* robi Ci wątpliwą przysługę i nie przykłada się do *utrwalania* zdobytej wiedzy. Pewnie sobie myśli: „Lepiej zostawię więcej miejsca na ważniejsze informacje, jak na przykład to, jakich dzikich zwierząt powinienem unikać albo czy jazda nago na snowboardzie będzie dobrym pomysłem". Jak w takim razie oszukać mózg, by zaczął myśleć, że od znajomości HTML5 i JavaScriptu zależy Twoje życie?

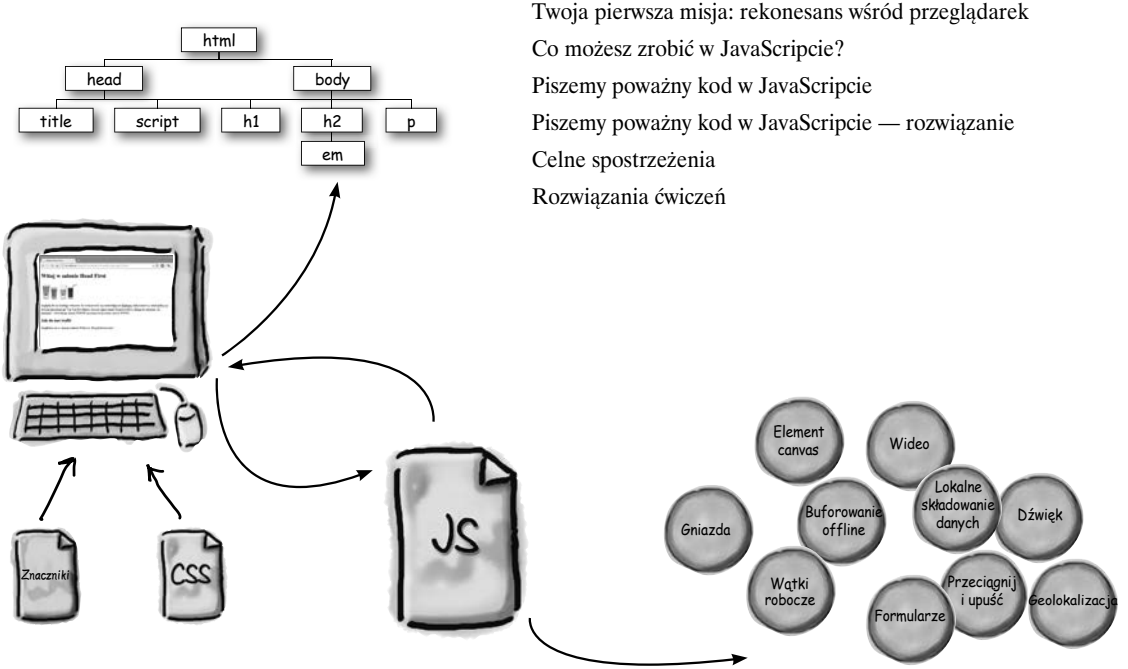
Dla kogo jest ta książka?	22
Wiemy, co sobie myślisz	23
Wiemy też, co sobie myśli Twój mózg	23
Metapoznanie — myślenie o myśleniu	25
Redaktorzy techniczni	30
Podziękowania	31

Poznajemy HTML5

1 Witaj w Webowicach

HTML pokonał długą i krętą drogę. Jasne, że HTML na początku był zwykłym językiem znacznikowym, ale ostatnio przybyło mu trochę mięśni. Stał się teraz językiem dostosowanym do tworzenia prawdziwych aplikacji internetowych z możliwością lokalnego składowania danych, rysowania 2D, pracy offline, obsługi gniazd i wątków, a na tym nie koniec. HTML ma w swojej historii wiele nie najlepszych, a nawet dramatycznych momentów (do wszystkiego z czasem dojdziemy). W tym rozdziale wybierzemy się na krótką przejażdżkę po Webowicach, by poznać wszystko to, co składa się na HTML5. No dalej, wskakuj! Kierunek Webowice — od zera do HTML5 w 3,8 strony (na pełnym gazie).

JUŻ DZIŚ przesiądź się na HTML5!	34
Wprowadzenie do *MASZYNKI DO HTML5*. Zaktualizuj swój HTML już dziś!	36
Jesteś bliżej znaczników HTML5, niż myślisz!	39
Wywiad tygodnia: Najnowszy HTML wyznaje swoje sekrety	43
Czy mógłby się w końcu pokazać PRAWDZIWY HTML5?	44
Jak tak naprawdę działa HTML5?	46
Kto się czym zajmuje?	48
Twoja pierwsza misja: rekonesans wśród przeglądarek	49
Co możesz zrobić w JavaScripcie?	54
Piszemy poważny kod w JavaScripcie	57
Piszemy poważny kod w JavaScripcie — rozwiązanie	58
Celne spostrzeżenia	63
Rozwiązania ćwiczeń	64



2 Wstęp do JavaScriptu i struktury DOM

Odrobina kodu

JavaScript zabierze Cię w całkiem nowe miejsca. Wiesz już wszystko o znacznikach HTML (związanych ze *strukturą*), stylach CSS (związanych z *prezentacją*), więc tym, czego Ci brakuje, jest język JavaScript (związany z *zachowaniem*). Jeśli Twoja wiedza ogranicza się do struktury i prezentacji, możesz — rzecz jasna — tworzyć doskonale wyglądające strony, ale pozostaną one *tylko stronami*. Dodanie za pośrednictwem JavaScriptu zachowań niesamowicie zwiększa możliwości interakcji, a nawet lepiej — możesz tworzyć prawdziwe aplikacje internetowe. Przygotuj się na dołożenie do Twojego pudełka na narzędzia internetowe najbardziej interesującego i uniwersalnego sprzętu: JavaScriptu i programowania!

Mechanizm działania JavaScriptu	66
Co możesz zrobić w JavaScripcie?	67
Deklarowanie zmiennych	68
Jak nazywać zmienne	70
Sztuka wyrażania (się)	73
Powtarzaj to w kółko...	76
Podejmowanie decyzji w JavaScripcie	79
Podejmowanie wielu decyzji i blok wyłapujący	80
Jak i gdzie dodać JavaScript do strony	83
Jak JavaScript współpracuje ze stroną	84
Jak upiec swój własny DOM	85
Pierwsza degustacja modelu DOM	86
HTML5 jest z Marsa, JavaScript jest z Wenus	88
Nie możesz mieszać w strukturze DOM, zanim cała strona nie zostanie załadowana	94
Do czego jeszcze może się przydać DOM?	96
Pomówmy jeszcze chwilę o JavaScripcie, czyli jak przechowywać wiele wartości	97
Sloganomat	101
Celne spostrzeżenia	105
Rozwiązania ćwiczeń	106

3 Zdarzenia, ich obsługa i takie tam...

Odrobina interakcji

Wciąż jeszcze nie zatroszczyłeś się o kontakt z użytkownikiem. Poznałeś podstawy JavaScriptu, ale czy umiesz nawiązać bezpośredni kontakt z użytkownikiem? Gdy strony zaczynają odpowiadać na działania użytkownika, przestają być zwykłymi dokumentami, a stają się żywymi, czującymi i reagującymi aplikacjami. W tym rozdziale dowiesz się, jak obsługiwać jedną z form wprowadzania danych przez użytkownika i powiązać stary dobry element `<form>` z kodem. Być może brzmi to trochę groźnie, ale, uwierz, otwiera niesamowite możliwości. Lepiej zapnij pasy, bo przed Tobą ekstremalna jazda — w mgnieniu oka od zera do interaktywnej aplikacji.

Przygotuj się na potańcówkę w Webowicach 114
Zaczynamy... 115
Ale nic się nie dzieje po kliknięciu przycisku „Dodaj piosenkę” 116
Obsługa zdarzeń 117
Trzeba to dobrze zaplanować... 118
Dostajemy się do przycisku „Dodaj piosenkę” 118
Wskazujemy przyciskowi funkcję obsługi zdarzenia 119
Przyjrzyjmy się temu bliżej... 120
Pobieranie tytułu piosenki 122
Jak umieścić piosenkę na stronie 125
Jak utworzyć nowy element 127
Dodawanie elementu do struktury DOM 128
Łączymy to wszystko ze sobą... 129
...i bierzemy na jazdę próbną 129
Podsumowanie — co udało się nam zrobić? 130
Jak dodać przygotowany kod 133
Integrowanie z przygotowanym kodem 134
Celne spostrzeżenia 136
Rozwiązania ćwiczeń 137

4

Funkcje i obiekty w JavaScripcie

Z JavaScriptem na poważnie

Czy możesz się już nazwać prawdziwym twórcą skryptów? Najprawdopodobniej tak, w końcu wiesz już całkiem sporo o JavaScripcie. Ale kto chciałby poprzestać na tworzeniu skryptów, skoro może zostać programistą? Żarty się skończyły — przygotuj się na spotkanie z **funkcjami** i **obiektami**. Stanowią one klucz do wydajniejszego, lepiej zorganizowanego i łatwiejszego w utrzymaniu kodu. Są też powszechnie stosowane w dostępnych z poziomu JavaScriptu interfejsach API HTML5, więc im lepiej je poznasz, tym szybciej zaczniesz korzystać z nowych możliwości dostępnych w HTML5. A teraz skup się, bo ten rozdział wymaga Twojej wzmożonej uwagi...

Poszerzanie słownictwa	140
Jak utworzyć własną funkcję	141
Jak działa funkcja	142
Anatomia funkcji	147
Zmienne lokalne i globalne	149
Poznaj zasięg lokalnych i globalnych zmiennych	150
Och! A czy wspomnieliśmy, że funkcje są również wartościami?	154
Czy ktoś tu powiedział „obiekt”?!	157
Jak utworzyć obiekt w JavaScripcie	158
Przykładowe operacje na obiektach	159
Pomówmy o przekazywaniu obiektów do funkcji	162
Obiekty mogą mieć też zachowania...	168
Tymczasem w kinie Webowice...	169
Dodajemy słowo kluczowe „this”	171
Jak utworzyć konstruktor	173
Jak naprawdę działa this	175
Pierwsza jazda próbna konstruktora	179
Czym tak naprawdę jest obiekt window?	181
Bierzemy window.onload pod lupę	182
Ponowne spojrzenie na obiekt document	183
Bierzemy document.getElementById pod lupę	183
Jeszcze jeden obiekt do przeanalizowania — obiekt reprezentujący element	184
Celne spostrzeżenia	186

Twoja strona wie, gdzie jesteś

5 Geolokalizacja

Gdziekolwiek pójdziesz, tam będziesz. Czasem wiedza o tym, gdzie się znajdujesz, robi wielką różnicę (zwłaszcza w przypadku aplikacji internetowych). W tym rozdziale pokażemy Ci, jak tworzyć strony internetowe, które znają lokalizację użytkownika — czasem z dokładnością do metrów, a czasem przybliżoną, określającą jedynie dzielnicę (ale zawsze będziesz wiedział, w którym mieście!). Niestety czasem nie będzie to jednak możliwe albo ze względów technicznych, albo z uwagi na to, że użytkownik nie życzy sobie takiej inwigilacji. W każdym razie w tym rozdziale opiszemy javascriptowy interfejs Geolocation API. Przygotuj najlepszy sprzęt szpiegowski, który masz na stanie (może być nawet zwykły pecet) i do dzieła.

Lokalizacja, lokalizacja i jeszcze raz lokalizacja	190
Szerokość i długość...	191
W jaki sposób Geolocation API określa Twoje położenie	192
W porządku, ale gdzie właściwie jesteś?	196
Jak to wszystko współdziała	200
Odkrywanie lokalizacji naszej tajnej siedziby	203
Piszemy kod wyznaczający odległość	205
Jak umieścić mapę na stronie	207
Wbijanie pinezek w mapę...	210
Inne wspaniałości oferowane przez API map Google'a	212
Pomówmy o dokładności	215
„Gdziekolwiek pójdziesz, tam będziesz"	216
Zaczynamy pracę nad aplikacją	217
Modyfikujemy istniejący kod...	218
Czas się ruszyć!	220
Masz kilka opcji...	222
Czas na timeout i maximumAge	223
Nie próbujcie robić tego w domu! (Badanie granic wytrzymałości geolokalizacji)	226
Kończymy pracę nad aplikacją	228
Integrowanie aplikacji z nową funkcją	229
Celne spostrzeżenia	231
Rozwiązania ćwiczeń	232

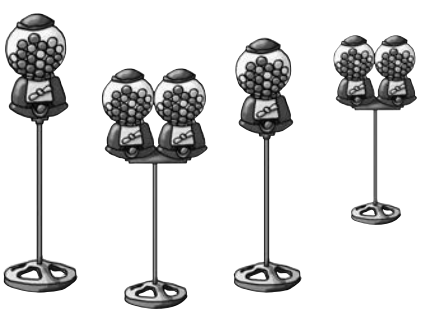
Rozmawiamy z siecią

6 Aplikacje otwarte na innych

Już za długo siedzisz tylko na swojej stronie. Czas wyjść do świata, pogadać z usługami sieciowymi, pozbierać z nich dane i w ten sposób tworzyć jeszcze lepsze aplikacje ze znacznie większymi możliwościami. To bardzo istotny obszar nowoczesnych aplikacji tworzonych w HTML5, ale żeby cokolwiek zrobić, musisz *wiedzieć, jak korzystać* z usług sieciowych. W tym rozdziale właśnie o tym będziemy mówić i pokażemy, w jaki sposób dane pochodzące z prawdziwej usługi sieciowej połączyć ze stroną. Kiedy już się tego dowiesz, będziesz mógł korzystać z dowolnych dostępnych usług. Nauczysz się nawet specjalnego nowego dialektu stosowanego podczas „rozmów" z usługami sieciowymi. A zatem do dzieła — poznaj kolejne API, tym razem służące do komunikacji.

Uważaj na przerwane połączenia!

Firma Megagumy potrzebuje aplikacji internetowej 236
Trochę więcej szczegółów na temat przedsięwzięcia 238
Jak zgłaszać żądania do usług sieciowych 241
Jak zgłosić żądanie z poziomu JavaScriptu 242
Wystarczy tego XML-a, poznaj JSON-a 248
Piszemy funkcję zwrotną onload 251
Wyświetlanie danych o sprzedaży gum 252
Jak postawić własny serwer WWW 253
Poprawiamy kod, by korzystał z JSON-a 258
Przesiadamy się na prawdziwy serwer 259
Jednym słowem, zawiecha! 261
Wciąż wisisz, pamiętasz? To błąd... 264
Zasady bezpieczeństwa w przeglądarce 266
Jakie mamy możliwości? 269
Poznaj JSONP 274
O co chodzi z tym „P" w JSONP? 275
Zmodyfikujmy w końcu naszą aplikację 278
Krok 1. Zajmujemy się elementem skryptu... 286
Krok 2. Czas na odmierzanie czasu 287
Krok 3. Piszemy na nowo obsługę JSONP 289
Prawie zapomnieliśmy — miejcie się na baczności przed straszliwą pamięcią podręczną przeglądarki 294
Jak usunąć powtarzające się dane 295
Dodanie do adresu parametru lastreporttime 297
Celne spostrzeżenia 299

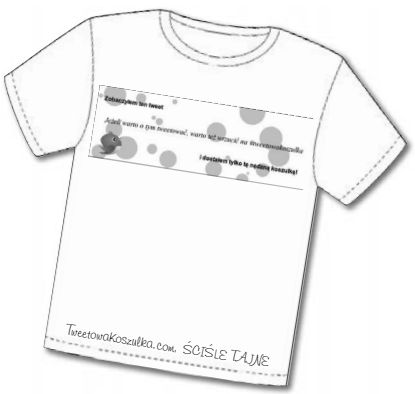


7 Odkryj w sobie artystę

Element canvas

HTML przestał już być tylko językiem znacznikowym. Dzięki nowemu elementowi canvas wprowadzonemu w HTML5 możesz własnymi rękami tworzyć, zmieniać i niszczyć *piksele*. Element canvas, czyli wirtualne płótno, pozwoli Ci odkryć w sobie artystę — koniec z HTML-em, który opisuje jedynie semantykę i nie ma związku z prezentacją. Na płótnie możesz malować i rysować, więc *wszystko* tu dotyczy prezentacji. Zobaczymy, jak umieszczać ten element na stronie, jak rysować na nim tekst i grafikę (oczywiście za pomocą JavaScriptu), a nawet jak poradzić sobie z przeglądarkami, które nie obsługują elementu canvas. I jeszcze jedno — to nie jest tak, że opiszemy ten element i o nim zapomnimy — będziemy z niego korzystać jeszcze w kolejnych rozdziałach.

Nowy projekt HTML5 już się nie może doczekać, aż się nim zajmiesz!



Nasz nowy projekt: TweetowaKoszulka	302
Przyglądamy się „makiecie”	303
Jak umieścić element canvas na stronie	306
Jak zobaczyć element canvas	308
Rysowanie na płótnie	310
Jak wyjść z twarzą z opresji	315
TweetowaKoszulka — obraz całości	317
Najpierw umieśćmy HTML na swoim miejscu	320
Teraz możemy dodać formularz	321
Czas na obliczenia — w roli głównej JavaScript	322
Piszemy funkcję drawSquare	324
Dodajemy wywołanie metody fillBackgroundColor	327
Tymczasem na TweetowaKoszulka.com...	329
Rysujemy z geekami...	331
Rozkładamy metodę arc na części pierwsze	334
Smaczki metody arc	336
Ja mówię stopnie, a Ty radiany	337
Wracamy do kodu rysującego kółka	338
Piszemy funkcję drawCircle...	339
Pobieranie tweetów	343
Zbliżenie na tekst na płótnie	348
Ożywiamy funkcję drawText	350
Kończymy funkcję drawText	351
Celne spostrzeżenia	358
Rozwiązania ćwiczeń	360

8 Telewizja po liftingu

Element video... gościnnie występuje canvas

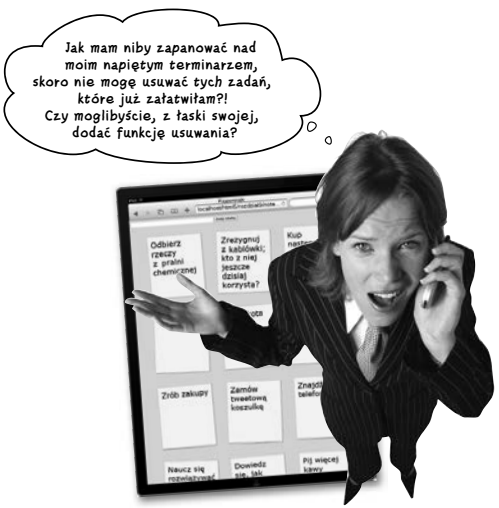
Nie potrzebujemy już żadnych dodatkowych wtyczek. W końcu wideo stało się pełnoprawnym członkiem rodziny HTML. Wystarczy wstawić znacznik `<video>` na stronę i już! W momencie masz dostęp do wideo, a co ważne, działa to na większości urządzeń. Jednak element video to *znacznie więcej* niż tylko *zwykły znacznik* — to javascriptowe API, które umożliwia sterowanie odtwarzaniem, tworzenie własnych interfejsów odtwarzacza i integrowanie materiału wideo z pozostałymi elementami strony w całkowicie nowy sposób. A skoro mowa o *integracji*... pamiętasz, co mówiliśmy o powiązaniu elementów video i canvas? Zobaczysz na własne oczy, jak za pomocą tych dwóch elementów *przetwarzać wideo* w czasie rzeczywistym. W tym rozdziale pracę rozpoczniemy od umieszczenia na stronie i uruchomienia elementu video, a później weźmiemy na warsztat javascriptowe API i podkręcimy tempo. Zdziwisz się, ile można zrobić za pomocą kilku znaczników, odrobiny kodu JavaScript oraz elementów video i canvas.

Dostrój odbiornik na TV Webowice

Poznaj TV Webowice	368
Podłącz odbiornik i sprawdź, czy działa...	369
Jak działa element video	371
Dokładna inspekcja atrybutów...	372
Co musisz wiedzieć o formatach wideo?	374
Trudna sztuka żonglowania formatami...	376
Czy dobrze zrozumiałem, że jest jakieś API?	381
Planujemy program dla TV Webowice	382
Jak napisać funkcję obsługi zdarzenia ended	385
Jak działa metoda canPlayType	387
Odpakowujemy testową budkę	393
Przeglądamy pozostałą część fabrycznego kodu	394
Funkcje setEffect i setVideo	396
Implementujemy własne kontrolki wideo	402
Przełączanie testowych plików wideo	405
Czas na efekty specjalne	407
Jak przetwarzać wideo	410
Jak przetwarzać wideo z użyciem bufora	411
Używamy elementu canvas jako bufora	413
Teraz zajmiemy się efektami	417
Jak korzystać ze zdarzeń błędów	424
Celne spostrzeżenia	426
Rozwiązania ćwiczeń	427

9 Lokalne składowanie danych
Mechanizm Web Storage

Masz już dosyć upychania danych klienta w tych malutkich ~~szefach~~ ciasteczkach? Może to było zabawne w latach 90. zeszłego wieku, ale teraz aplikacje internetowe mają znacznie większe wymagania i potrzeby. Co byś pomyślał, gdybyśmy powiedzieli, że możemy zapisać pięć megabajtów w przeglądarce każdego użytkownika? Pewnie popatrzyłbyś na nas z politowaniem i pomyślał, że coś z nami nie tak. Zdziwisz się, ale jest to jak najbardziej możliwe — interfejsowi API Web Storage dostępnemu w HTML5. W tym rozdziale pokażemy Ci wszystko, co musisz wiedzieć, by zapisywać dowolne obiekty lokalnie, na urządzeniu użytkownika, i korzystać z tych danych w aplikacji.



Jak działało składowanie danych w przeglądarce (1995 – 2010)	430
Jak działa mechanizm Web Storage w HTML5	433
Przypominajki...	434
Czy przypadkiem localStorage i tablice nie są bliźniakami?	440
Tworzymy interfejs	445
Teraz dodamy trochę JavaScriptu	446
Kończymy pracę nad interfejsem użytkownika	447
Musimy się zatrzymać na mały przegląd	450
Zrób to sam: konserwacja aplikacji	451
Mamy odpowiednią technologię...	455
Przerabiamy aplikację, by korzystała z tablicy	456
Modyfikujemy funkcję createSticky, by używała tablicy	457
Usuwanie notatek	462
Funkcja deleteSticky	465
Jak wskazać notatkę do usunięcia	466
Jak z obiektu zdarzenia wyciągnąć usuwaną notatkę	467
Notatkę usuwamy także ze struktury DOM	468
Ulepszamy interfejs użytkownika — dodajemy możliwość wyboru koloru	469
JSON.stringify nie tylko do tablic	470
Używamy nowego obiektu stickyObj	471
Nie próbujcie robić tego w domu! (Rozsadzamy 5-megabajtową składnicę danych)	474
Poznałeś już localStorage. Jak i kiedy go zastosować?	478
Celne spostrzeżenia	480
Rozwiązania ćwiczeń	481

Zaprzęgamy JavaScript do pracy

10 Wątki robocze

Powolny skrypt — czy na pewno chcesz go uruchamiać? Jeśli spędziłeś już wystarczająco dużo czasu z JavaScriptem i przeglądałeś wiele stron, z pewnością trafiłeś na „powolne skrypty". Jak to możliwe, że skrypt może się wykonywać *za wolno*, skoro w naszych komputerach siedzą potężne, wielordzeniowe procesory? Głównym powodem jest to, że JavaScript potrafi wykonywać tylko jedną operację w tym samym czasie. Jednak dzięki HTML5 i technologii Web Workers, czyli wątkom roboczym, *wszystko się zmienia*. Masz już możliwość uruchomienia wielu *własnych* wątków roboczych, które mogą wykonać więcej zadań. Bez względu na to, czy starasz się stworzyć aplikację, która jest bardziej responsywna, czy tylko wycisnąć z procesora maksimum możliwości, wątki robocze są do Twoich usług. Załóż swój magiczny kapelusz mistrza JavaScriptu i do dzieła!

Te okropne powolne skrypty	488
Jak JavaScript spędza czas	488
Kiedy jednowątkowość się NIE sprawdza	489
Nowy wątek przybywa z pomocą	490
Jak działają wątki robocze	492
Twój pierwszy wątek roboczy	497
Tworzymy kod zarządzający wątkami — manager.js	498
Odbieranie komunikatu z wątku	499
Teraz napiszemy kod wątku	500
Wirtualna eksploracja nieznanych lądów	508
Jak wyznaczyć zbiór Mandelbrota	510
Jak używać wielu wątków	511
Budujemy aplikację Eksplorator fraktali	517
Kod gotowy do użycia	518
Tworzenie wątków i zlecanie im zadań	522
Piszemy kod	523
Uruchamiamy wątki	524
Implementujemy wątek	525
Wracamy do kodu — jak przetworzyć wyniki otrzymane z wątku	528
Dopasowujemy płótno do okna przeglądarki	531
Upierdliwy szef programista	532
Laboratorium	534
Celne spostrzeżenia	538
Rozwiązania ćwiczeń	539

Dodatek

Ścinki

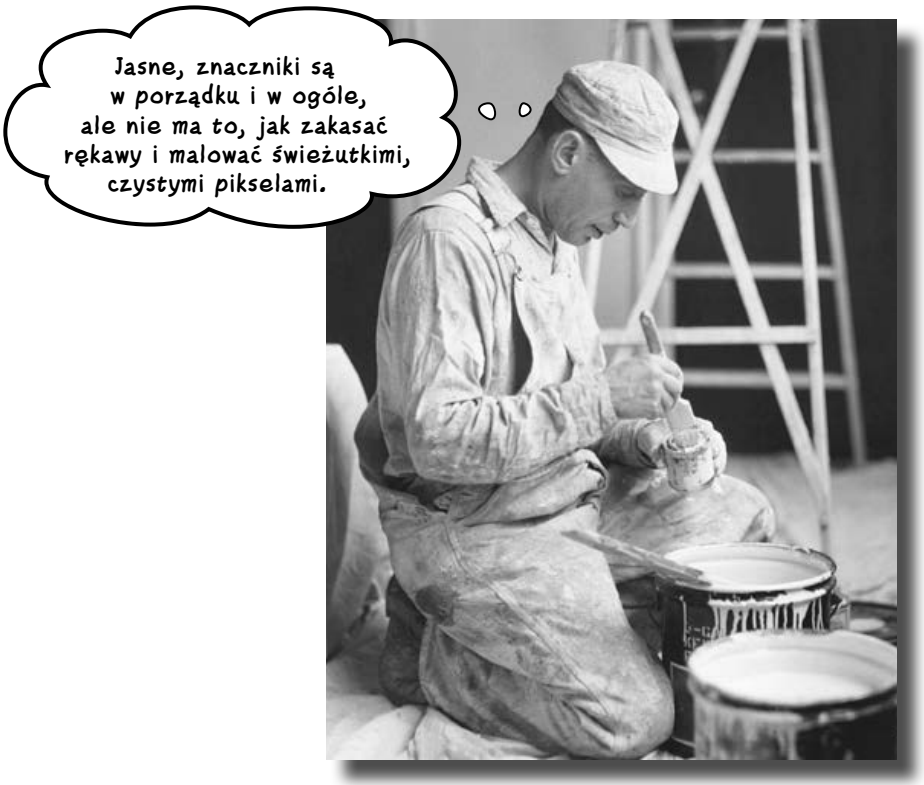
Udało się nam omówić naprawdę sporo zagadnień, a to już niestety ostatnie strony tej książki. Będziemy za Tobą tęsknić, ale zanim pozwolimy Ci odejść, musimy koniecznie wspomnieć jeszcze o kilku sprawach. Raczej nie uda nam się omówić w tym króciutkim rozdziale wszystkiego, co chciałbyś wiedzieć. Tak naprawdę to pierwotnie *opisaliśmy* wszystko, co musisz wiedzieć o HTML5 (poza materiałem z pozostałych rozdziałów), ale musieliśmy zmniejszyć stopień pisma do 0,00004. Wszystko byłoby świetnie, gdyby ktokolwiek potrafił to odczytać. W związku z tym wyrzuciliśmy większość materiału i zostawiliśmy w tym dodatku tylko dziesięć najciekawszych tematów.

1. Modernizr 544
2. Dźwięk 545
3. jQuery 546
4. Umarł XHTML, niech żyje XHTML 548
5. SVG 549
6. Aplikacje internetowe działające offline 550
7. Gniazda 551
8. Jeszcze raz o elemencie canvas 552
9. API selektorów 554
10. To nie wszystko! 555
Przewodnik HTML5 po nowych konstrukcjach 557
Przewodnik po elementach semantycznych w HTML5 558
Przewodnik po właściwościach CSS3 560

Skorowidz 561

7. Odkryj w sobie artystę

Element canvas



HTML przestał już być tylko językiem znacznikowym. Dzięki nowemu elementowi canvas wprowadzonemu w HTML5 możesz własnymi rękami tworzyć, zmieniać i niszczyć *piksele*. Element canvas, czyli wirtualne płótno, pozwoli Ci odkryć w sobie artystę — koniec z HTML-em, który opisuje jedynie semantykę i nie ma związku z prezentacją. Na płótnie możesz malować i rysować, więc *wszystko* tu dotyczy prezentacji. Zobaczymy, jak umieszczać ten element na stronie, jak rysować na nim tekst i grafikę (oczywiście za pomocą JavaScriptu), a nawet jak poradzić sobie z przeglądarkami, które nie obsługują elementu canvas. I jeszcze jedno — to nie jest tak, że opiszemy ten element i o nim zapomnimy — będziemy z niego korzystać jeszcze w kolejnych rozdziałach.

W porządku, „niszczyć" to może za mocno powiedziane.

Słyszeliśmy, że tak naprawdę elementy <canvas> i <video> łączy coś więcej niż tylko obecność na stronach internetowych... O szczegółach porozmawiamy później.

Nasz nowy projekt: TweetowaKoszulka

Naszym mottem jest: „Jeżeli warto o tym tweetować, warto też wydrukować na koszulce".

Połowa sukcesu w próbach stania się dziennikarzem zależy od słów umieszczanych w druku. Dlaczego by nie nosić ich na własnej (lub cudzej) piersi? Przyjmijmy takie założenie i trzymajmy się go.

Teraz pozostało nam rozwiązanie tylko jednego problemu — potrzebujemy wygodnej aplikacji internetowej, która umożliwi klientom zaprojektowanie własnej koszulki z jednym ze swoich tweetów.

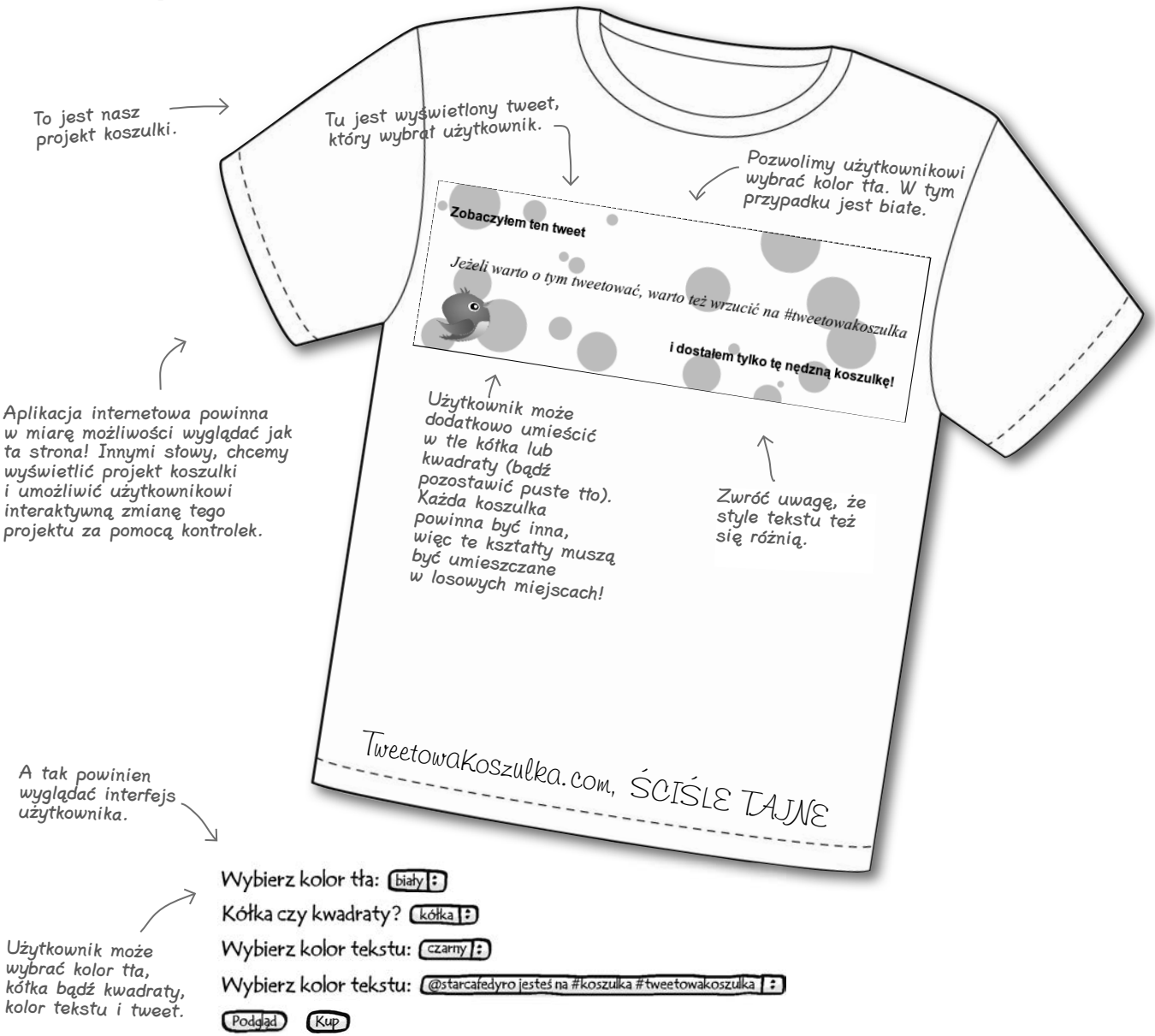
Pewnie sobie myślisz: „Hm... to nie jest taki zły pomysł". W takim razie dołącz do nas, pod koniec rozdziału będziemy mieli gotową aplikację. A... jeszcze jedno — jeżeli postanowisz uruchomić taki serwis i robić na nim grubą kasę, nie będziemy zgłaszać żadnych roszczeń, ale mamy jedną prośbę — prześlij nam koszulkę!

Założycielka serwisu TweetowaKoszulka.com

Przyglądamy się „makiecie"

Po wyczerpujących i wieloetapowych pracach projektowych oraz zakrojonych na szeroką skalę badaniach fokusowych otrzymaliśmy makietę (czyli wstępny projekt graficzny), którą zamieściliśmy poniżej:

No dobra, przecież to tylko szkic, który zrobiliśmy na serwetce w Star Café...



WYSIL SZARE KOMÓRKI

Jeszcze raz przyjrzyj się wymaganiom przedstawionym na poprzedniej stronie. Jak myślisz, uda się to zrealizować za pomocą HTML5? Pamiętaj, że jednym z wymogów jest to, by aplikacja działała na możliwie wielu różnych urządzeniach.

Przejrzyj poniższe możliwości i zaznacz tę, która jest Twoim zdaniem najlepsza:

☐ Najlepiej użyć Flasha, bo działa w większości przeglądarek.

☐ Trzeba się przyjrzeć, czy w HTML5 są dostępne jakieś nowe technologie, które mogłyby się tu przydać (podpowiedź: być może mógłby to być element canvas).

☐ Trzeba napisać osobną aplikację dla każdego urządzenia. Dzięki temu na pewno uda się w każdym przypadku osiągnąć zamierzony cel.

☐ Obraz można wygenerować po stronie serwera i dostarczyć go do przeglądarki użytkownika.

Nie istnieją głupie pytania

P: Ale serio, czemu nie użyjemy Flasha albo osobnej aplikacji?

O: Flash to świetna technologia i oczywiście moglibyśmy z niej skorzystać. Jednak w ostatnim czasie wszyscy zwracają się raczej w stronę HTML5, a poza tym (przynajmniej na razie) Flash nie działa prawidłowo na wszystkich urządzeniach, nawet tych bardziej popularnych.

Osobna, specjalizowana aplikacja sprawdza się, kiedy oczekujesz pełnego dostosowania funkcjonalności do urządzenia. Weź jednak pod uwagę, że opracowanie specjalizowanych aplikacji dla wielu różnych urządzeń jest bardzo drogie.

Decydując się na HTML5, nie będziemy mieli problemów z działaniem aplikacji ani na urządzeniach mobilnych, ani na komputerach stacjonarnych. Poza tym można w ten sposób tworzyć aplikacje, korzystając w zasadzie z jednej technologii.

P: Podoba mi się pomysł z tworzeniem obrazu na serwerze. W ten sposób mógłbym napisać jeden kod, który działałby dla wielu urządzeń. Znam trochę PHP, więc bez problemu powinienem sobie z tym poradzić.

O: To kolejna droga, którą moglibyśmy pójść. Ma jednak sporą wadę — gdyby się okazało, że zyskaliśmy miliony użytkowników, musielibyśmy się zmierzyć z problemem skalowalności i dostosować serwery do takiego obciążenia. W proponowanym przez nas rozwiązaniu każdy użytkownik tworzy podgląd nadruku na koszulce na własnym komputerze, więc nie obciąża serwera. Poza tym, jeżeli aplikacja znajdzie się w całości po stronie klienta, czyli przeglądarki, będzie w większym stopniu interaktywna, a jej obsługa wygodniejsza.

Jak to? Dobrze, że spytałeś...

Krótka wizyta w zespole TweetowejKoszulki

Łukasz, Iza i Przemek

Znasz już wymagania i widziałeś szkic projektu, nadszedł więc czas na przemyślenia, by można było zacząć tworzyć aplikację. Przysłuchaj się rozmowie zespołu i zobacz, dokąd to zmierza...

Przemek: Myślałem, że to będzie proste, aż zobaczyłem te kółka w tle.

Łukasz: O co ci chodzi? Przecież to tylko obrazek...

Iza: Wcale nie! Założycielka serwisu chce, żeby te kółka były rozmieszczone losowo, dzięki czemu kółka na mojej koszulce będą inne niż na twojej. Zresztą to samo dotyczy kwadratów.

Łukasz: W porządku, przecież już kiedyś generowaliśmy obrazki na serwerze.

Przemek: Tak, ale to nie było idealne rozwiązanie. Pamiętacie, ile musieliśmy płacić za ten serwer?

Łukasz: Ech, faktycznie... Nic nie mówiłem.

Przemek: To bez znaczenia, bo i tak chcemy, żeby podgląd był widoczny od razu, bez tych długich, denerwujących przestojów na pobieranie obrazka z serwera. Musimy założyć, że wszystko robimy po stronie klienta, o ile to możliwe...

Iza: Chłopaki, myślę, że spokojnie damy radę. Czytałam ostatnio o wprowadzonym w HTML5 elemencie canvas.

Łukasz: Element canvas? Pamiętaj, że ja jestem tylko od interfejsu użytkownika, więc mnie oświeć.

Iza: Musiałeś słyszeć o elemencie canvas! Łukasz, przecież to jest nowy element wprowadzony w HTML5 tworzący płaszczyznę, po której można rysować, wstawiać na niej tekst i bitmapy.

Łukasz: To mi wygląda na nową odsłonę elementu `<img>`. Nieraz umieszczaliśmy go na stronie, ustalaliśmy szerokość i wysokość, a przeglądarka robiła resztę.

Iza: W sumie niezłe porównanie. W przypadku elementu canvas też ustalamy szerokość i wysokość, ale możemy po nim rysować z poziomu kodu JavaScript.

Przemek: No dobra, a jak to się ma do pozostałych znaczników? Czy za pomocą JavaScriptu możemy powiedzieć elementowi canvas: „Umieść ten element `<h1>` w tym miejscu”?

Iza: Nie, po umieszczeniu elementu canvas na stronie opuszczamy świat znaczników. Z poziomu JavaScriptu możemy umieszczać punkty, linie, ścieżki, obrazki i tekst. To jest naprawdę niskopoziomowe API.

Przemek: No cóż, jeżeli tylko pozwoli nam to wymalować tło tymi losowymi kółkami, jestem za. Dobra, my tu gadu-gadu, a robota czeka!

Jak umieścić element canvas na stronie

Łukasz miał rację, w pewnym sensie element canvas przypomina <img>. Dodaje się go tak:

Element canvas jest zwykłym elementem HTML, który rozpoczyna się znacznikiem otwierającym <canvas>.

Atrybut width określa szerokość elementu umieszczonego na stronie wyrażoną w pikselach.

Analogicznie atrybut height określa wysokość elementu, w tym przypadku równą 200 pikseli.

```
<canvas id="lookwhatIdrew" width="600" height="200"></canvas>
```

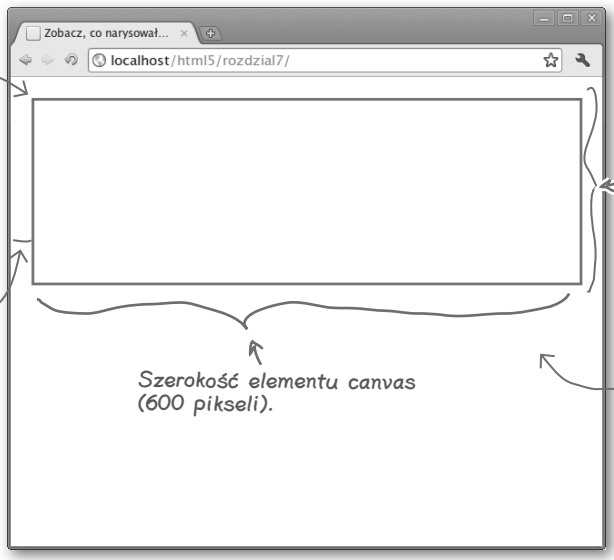
Dodaliśmy identyfikator (id), tak by można było wskazać ten konkretny element. Już za chwilę zobaczysz, jak z niego skorzystamy...

Atrybut width jest ustawiony na 600 pikseli.

A to jest znacznik zamykający.

Przeglądarka przydziela miejsce na element canvas, biorąc pod uwagę podaną szerokość i wysokość.

W tym przypadku szerokość jest równa 600, a wysokość 200 pikseli.



To jest lewy górny wierzchołek elementu canvas. Będziemy korzystać z jego współrzędnych jako punktu odniesienia (już niedługo to zobaczysz).

Wysokość elementu canvas (w naszym przypadku równa 200 pikseli).

Naokoło elementu canvas jest trochę wolnego miejsca — to domyślne marginesy elementu body.

Szerokość elementu canvas (600 pikseli).

Na stronie mogą się oczywiście znaleźć inne elementy HTML-a. Element canvas zachowuje się tak samo jak pozostałe elementy (np. img itp.).

Jazda próbna z nowym elementem canvas

Pora sprawdzić, jak to wygląda w przeglądarce. Utwórz nowy plik, wpisz do niego poniższy kod i załaduj do przeglądarki:

```
<!doctype html>
<html lang="pl">
<head>
   <title>Zobacz, co narysowałem!</title>
   <meta charset="utf-8">
</head>
<body>

<canvas id="lookwhatIdrew" width="600" height="200"></canvas>

</body>
</html>
```

Wpisz ten kod i zobacz, jak działa.

Na następnej stronie znajdziesz ciąg dalszy...

Jak zobaczyć element canvas

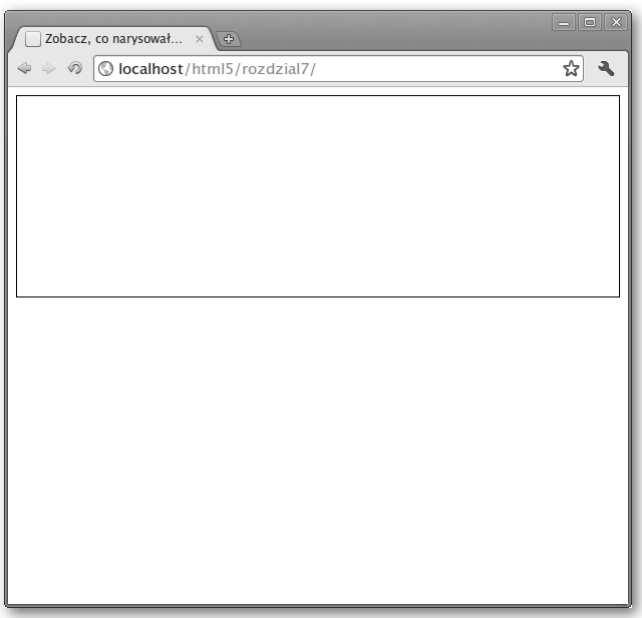
Elementu canvas nie zobaczymy, dopóki czegoś na nim nie narysujemy. Jest on po prostu przestrzenią w oknie przeglądarki, na której można rysować. Już niedługo zajmiemy się rysowaniem, ale na razie chcemy się upewnić, że płótno naprawdę znajduje się na stronie.

Jest na to pewien sposób. Jeżeli dla znacznika `<canvas>` zdefiniujemy styl wyświetlający ramki, element pojawi się na stronie. Dodajmy prostą definicję stylu odpowiadającą 1-pikselowej ramce.

```
<!doctype html>
<html lang="pl">
<head>
   <title>Zobacz, co narysowałem!</title>
   <meta charset="utf-8">
   <style>
      canvas {
         border: 1px solid black;
      }
   </style>
</head>
<body>
<canvas id="lookwhatIdrew" width="600" height="200"></canvas>
</body>
</html>
```

Elementowi canvas przypisaliśmy styl definiujący 1-pikselową, czarną ramkę. Dzięki temu element stanie się widoczny.

Dużo lepiej! Teraz widzimy, gdzie jest płótno. Wypadałoby zrobić z nim coś ciekawego...



Nie istnieją głupie pytania

P: Czy na stronie mogę mieć tylko jeden element canvas?

O: Nie, możesz mieć ich tyle, ile chcesz (a właściwie tyle, ile jest w stanie obsłużyć przeglądarka). Jeżeli nadasz każdemu z nich unikalny identyfikator, będziesz mógł na nich niezależnie rysować. Już za chwilę pokażemy, jak korzystać z identyfikatorów.

P: Czy element canvas jest przezroczysty?

O: Tak, element canvas jest przezroczysty. Możesz na nim rysować, wypełniając go w ten sposób kolorowymi pikselami. W dalszej części rozdziału dowiesz się, jak się to robi.

P: Skoro jest przezroczysty, to mogę go umieścić nad innym elementem strony i w ten sposób niejako na nim rysować, prawda?

O: Tak jest! To jest jedna z fajniejszych cech elementu canvas. Dzięki niemu możesz dodawać grafikę w dowolnym miejscu strony.

P: Czy wysokość i szerokość elementu mogę ustalić za pomocą stylów, czy muszę to robić poprzez atrybuty `width` i `height` znacznika `<canvas>`?

O: Możesz, ale działa to trochę inaczej, niż mógłbyś się spodziewać. Element canvas ma domyślną szerokość równą 300, a wysokość 150 pikseli. Jeżeli nie ustalisz wymiarów za pomocą atrybutów `width` i `height` znacznika `<canvas>`, na stronę zostanie wstawiony element o domyślnych wymiarach. Jeżeli później określisz rozmiar z poziomu CSS na, powiedzmy, 600px na 200px, element canvas (300px na 150px) zostanie *przeskalowany* do nowego rozmiaru i to samo stanie się z całą jego zawartością. Dokładnie tak samo zachowuje się zwykły obrazek, gdy ustalimy mu nowe wartości szerokości i wysokości. Jeżeli powiększysz obraz, najprawdopodobniej dojdzie do pikselizacji.

To samo dzieje się z płótnem. Jeżeli wcześniej miało 300 pikseli szerokości, a powiększymy je do 600, liczba pikseli pozostanie bez zmian, ale każdy piksel stanie się dwa razy szerszy, więc obraz będzie „chropowaty". Jeśli jednak ustawisz nowe wymiary elementu canvas za pomocą atrybutów `width` i `height`, wszystko będzie rysowane normalnie. W związku z tym polecamy określanie wymiarów za pomocą atrybutów znacznika, a nie z poziomu CSS, chyba że chcesz przeskalować element canvas.

WYSIL SZARE KOMÓRKI

Zauważyłeś pewnie, że element canvas nie ma żadnej zawartości.
Jak myślisz, co by się stało po załadowaniu strony, gdybyś umieścił tu jakiś tekst?

Rysowanie na płótnie

Do tej pory udało nam się umieścić na stronie pusty element canvas. Nie sądzisz, że najwyższy czas napisać trochę kodu JavaScript? Naszym celem będzie wyświetlenie na płótnie pięknego czarnego prostokąta. Najpierw zastanówmy się, gdzie go umieścić i jak duży powinien być. Co powiesz na współrzędne x = 10 i y = 10, a szerokość i wysokość równą 100 pikseli? Zajmijmy się tym.

Przyjrzyj się, jak jest zbudowany kod, który to robi:

Zacznijmy od standardowego dokumentu HTML5.

```
<!doctype html>
<html lang="pl">
<head>
   <title>Zobacz, co narysowałem!</title>
   <meta charset="utf-8" />
   <style>
      canvas { border: 1px solid black; }
   </style>
   <script>
      window.onload = function() {
         var canvas = document.getElementById("tshirtCanvas");

         var context = canvas.getContext("2d");

         context.fillRect(10, 10, 100, 100);
      };
   </script>
</head>
<body>
   <canvas width="600" height="200" id="tshirtCanvas"></canvas>
</body>
</html>
```

Na razie zachowamy styl nadający czarne obramowanie.

Kod umieszczamy w funkcji onload, dzięki czemu rysowanie rozpocznie się po załadowaniu całej strony.

Aby rysować na płótnie, musimy mieć referencję do niego. Korzystamy tu ze znanej metody getElementById, która pobiera ją ze struktury DOM.

Hm... to ciekawe, wygląda na to, że do rysowania będzie nam potrzebny kontekst „2d" elementu canvas...

Czarny prostokąt rysujemy na płótnie za pośrednictwem pobranego kontekstu.

Te liczby oznaczają współrzędne x i y elementu canvas, w których zostanie umieszczony prostokąt.

Podajemy też szerokość i wysokość (wyrażone w pikselach).

Ciekawe jest też to, że metoda rysująca wypełniony prostokąt nie pobiera koloru... Już za chwilę to wyjaśnimy.

A! I nie możemy zapomnieć o naszym elemencie canvas. Ustalamy jego szerokość na 600 pikseli, wysokość na 200 pikseli i identyfikator na „tshirtCanvas".

Krótka jazda próbna elementu canvas...

Wpisz kod z poprzedniej strony (lub pobierz go pod adresem *ftp://ftp.helion.pl/przyklady/htm5rg.zip*) i otwórz w przeglądarce. Zakładając, że korzystasz z nowoczesnej przeglądarki, zobaczysz coś takiego:

Kod z bliska

To był świetny test, ale musimy się nieco bardziej zagłębić w przedstawiony kod:

1. W dokumencie HTML za pomocą znacznika `<canvas>` zdefiniowaliśmy element canvas i nadaliśmy my identyfikator. Pierwsze, co trzeba zrobić, zanim coś się narysuje w tym elemencie, to pobranie go ze struktury DOM. Jak zwykle robimy to za pomocą metody `getElementById`:

```
var canvas = document.getElementById("tshirtCanvas");
```

2. Referencję do elementu canvas przypisaliśmy do zmiennej `canvas`. Zanim coś narysujemy, musimy grzecznie poprosić ten element, by udostępnił nam tzw. kontekst. W tym przypadku chodzi o kontekst 2D (czyli dwuwymiarowy). Zwrócony kontekst przypisujemy do zmiennej `context`:

```
var context = canvas.getContext("2d");
```

Zanim zaczniemy rysować, musimy poprosić o kontekst.

3. Teraz, kiedy mamy już kontekst, możemy go użyć do rysowania na płótnie. W tym przypadku stosujemy metodę `fillRect`, która umieszcza wypełniony prostokąt o wymiarach 100 na 100 pikseli we współrzędnych x = 10 i y = 10.

Zwróć uwagę, że metodę fillRect wywołujemy na kontekście, a nie na płótnie.

```
context.fillRect(10, 10, 100, 100);
```

Najlepiej sam sprawdź, jak to działa. Powinieneś zobaczyć czarny kwadrat. Spróbuj zmienić wartości dla współrzędnych x i y, a także szerokość i wysokość. Zobacz, co się stanie.

WYSIL SZARE KOMÓRKI

Co należałoby zrobić, żeby w przeglądarkach, które obsługują element canvas, wszystko działało jak należy, a w starszych przeglądarkach, które tego nie potrafią, wyświetlił się na przykład taki komunikat: „Hej, ty tam! Tak, do ciebie mówię! Zaktualizuj swoją przeglądarkę!"?

Nie istnieją głupie pytania

P: Skąd płótno wiedziało, że chcemy narysować czarny prostokąt?

O: Czarny jest po prostu domyślnym kolorem wypełnienia. Możesz oczywiście zmienić kolor — służy do tego właściwość `fillStyle`, którą niedługo omówimy.

P: A co, gdybym chciał narysować prostokąt bez wypełnienia?

O: W takim przypadku powinieneś zastosować metodę `strokeRect`, a nie `fillRect`. O metodzie `strokeRect` jeszcze wspomnimy.

P: Co to jest ten kontekst „2d”? I dlaczego nie mogę rysować bezpośrednio na płótnie?

O: Element canvas, czyli inaczej płótno, jest obszarem graficznym wyświetlanym na stronie internetowej. Kontekst jest obiektem powiązanym z elementem canvas, który dostarcza zbiór właściwości i metod używanych do rysowania na płótnie. Możesz nawet zachować bieżący stan płótna i przywrócić go później, co się czasami przydaje. W tym rozdziale poznasz jeszcze wiele właściwości i metod kontekstu.

Płótno zostało tak zaprojektowane, by mogło korzystać z więcej niż jednego interfejsu (2d, 3d i innych, o których nawet jeszcze nie pomyśleliśmy). Dzięki zastosowaniu kontekstu możemy pracować z różnymi interfejsami na tym samym elemencie — płótnie. Nie można rysować bezpośrednio na płótnie, ponieważ trzeba określić używany interfejs, a tego dokonuje się właśnie poprzez wybór kontekstu.

P: Czy to oznacza, że jest dostępny kontekst „3d”?

O: Jeszcze nie. W wyścigu bierze udział kilka standardów, ale jak na razie żaden z nich nie objął prowadzenia. Sytuacja zmienia się dosyć dynamicznie, więc cały czas trzymaj rękę na pulsie. W wolnej chwili spójrz na WebGL i biblioteki, które z tego korzystają: SpiderGL, SceneJS i three.js.

Kodowanie na serio

Czy zastanawiasz się, jak z poziomu kodu sprawdzić, czy przeglądarka obsługuje element canvas?

Oczywiście jest to możliwe. Przedstawimy Ci rozwiązanie tego problemu jak najszybciej, zwłaszcza że przed momentem założyliśmy, iż Twoja przeglądarka to potrafi. Pamiętaj, że w kodzie produkcyjnym zawsze powinieneś sprawdzać, czy działa wszystko, z czego zamierzasz skorzystać.

Wystarczy, że sprawdzisz, czy w obiekcie `canvas` (pobranym za pomocą metody `getElementById`) znajduje się metoda `getContext`:

Najpierw pobieramy referencję do elementu canvas umieszczonego na stronie.

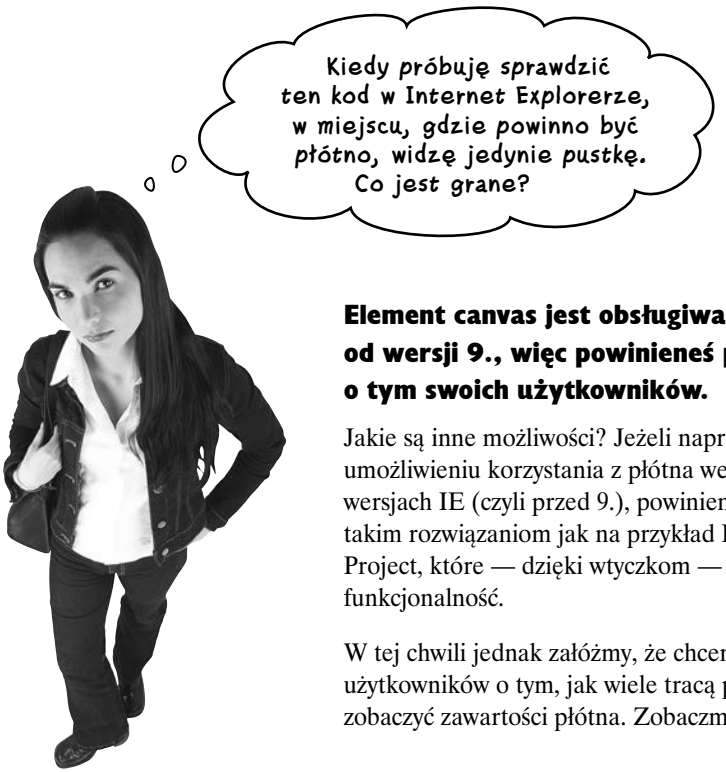
```
var canvas =
    document.getElementById("tshirtCanvas");
if (canvas.getContext) {
    // płótno jest obsługiwane
} else {
    // przykro nam, ale płótno nie jest obsługiwane
}
```

Później sprawdzamy, czy znajduje się w nim metoda getContext. Zwróć uwagę, że jej nie wywołujemy, a tylko sprawdzamy, czy posiada wartość.

Jeżeli chcesz sprawdzić, czy przeglądarka obsługuje płótno, a w dokumencie nie masz umieszczonego elementu canvas, możesz go utworzyć w locie za pomocą jednej z technik, które już dobrze znasz, na przykład:

```
var canvas =
    document.createElement("canvas");
```

Koniecznie zajrzyj do dodatku, w którym przedstawiliśmy bibliotekę umożliwiającą sprawdzenie całej funkcjonalności HTML5 w spójny sposób.



Element canvas jest obsługiwany w IE dopiero od wersji 9., więc powinieneś poinformować o tym swoich użytkowników.

Jakie są inne możliwości? Jeżeli naprawdę zależy Ci na umożliwieniu korzystania z płótna we wcześniejszych wersjach IE (czyli przed 9.), powinieneś przyjrzeć się bliżej takim rozwiązaniom jak na przykład Explorer Canvas Project, które — dzięki wtyczkom — dostarczają taką funkcjonalność.

W tej chwili jednak załóżmy, że chcemy tylko powiadomić użytkowników o tym, jak wiele tracą przez to, że nie mogą zobaczyć zawartości płótna. Zobaczmy, jak tego dokonać...

Możesz też zasugerować, by zaktualizowali IE do najnowszej wersji!

Jak wyjść z twarzą z opresji

No cóż, jest więcej niż prawdopodobne, że gdzieś tam, w innym miejscu i czasie, znajdzie się użytkownik, który będzie chciał odwiedzić Twoją stronę, a jego przeglądarka nie będzie obsługiwała elementu canvas. Czy nie należałoby wyświetlić mu komunikatu sugerującego zaktualizowanie przeglądarki? Możesz to zrobić w ten sposób:

To jest zwykły element canvas.

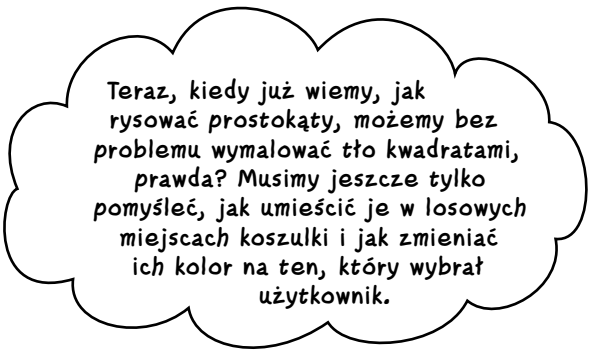
```
<canvas id="awesomecontent">
   Hej, ty tam! Tak, do ciebie mówię! Zaktualizuj swoją przeglądarkę!
</canvas>
```

Tu umieść komunikat, który ma się pojawić w przeglądarkach, które nie obsługują elementu canvas.

Jak to działa? Przeglądarka domyślnie zachowuje się tak: za każdym razem, gdy trafi na element, którego nie rozpoznaje, wyświetla po prostu tekst zawarty w tym elemencie. A zatem przeglądarki nieobsługujące płótna, gdy trafią na znacznik `<canvas>`, wyświetlą tekst: „Hej, ty tam! Tak, do ciebie mówię! Zaktualizuj swoją przeglądarkę!”. Z kolei przeglądarki, które obsługują element canvas, kompletnie ignorują zawartość umieszczoną między znacznikiem otwierającym i zamykającym, więc tekst nie zostanie wyświetlony.

To, że jest to takie proste, jest zasługą chłopaków (i dziewczyn) odpowiedzialnych za standardy HTML5. Wielkie dzięki!

Inną możliwością wykrycia, czy przeglądarka obsługuje element canvas, jest — co już wiesz — zastosowanie JavaScriptu. To rozwiązanie jest nawet bardziej elastyczne, ponieważ umożliwia skorzystanie z alternatywnych rozwiązań, w przypadku gdy przeglądarka nie obsługuje płótna — można przekierować użytkownika na inną stronę lub wyświetlić zwykły obrazek.



Łukasz: Jasne, ale będzie nam też potrzebny interfejs użytkownika, w którym można ustalić wszystkie parametry. Mamy już co prawda wstępny szkic, ale teraz trzeba go zaimplementować.

Iza: Masz rację, Łukasz. Nie ruszymy z miejsca bez interfejsu użytkownika.

Przemek: A nie jest to zwykły HTML?

Łukasz: Myślę, że tak, ale jak to ma właściwie działać, jeżeli założymy, że wszystko ma się odbywać po stronie klienta? Na przykład podstawowa sprawa — gdzie mają zostać przesłane dane z formularza? Ja chyba nie do końca rozumiem, jak to ma wszystko ze sobą współpracować.

Przemek: Łukasz, przecież kliknięcie przycisku może po prostu wywołać javascriptową funkcję, w której na płótnie wyświetlimy podgląd koszulki.

Łukasz: No dobra, to brzmi rozsądnie, ale jak w takim razie odczytamy wartości, które użytkownik wybrał w polach formularza, skoro wszystko odbywa się po stronie klienta?

Iza: Z pomocą przyjdzie nam struktura DOM, a właściwie metoda `document.getElementById`, dzięki której odczytamy te wartości. Przecież już to robiliśmy!

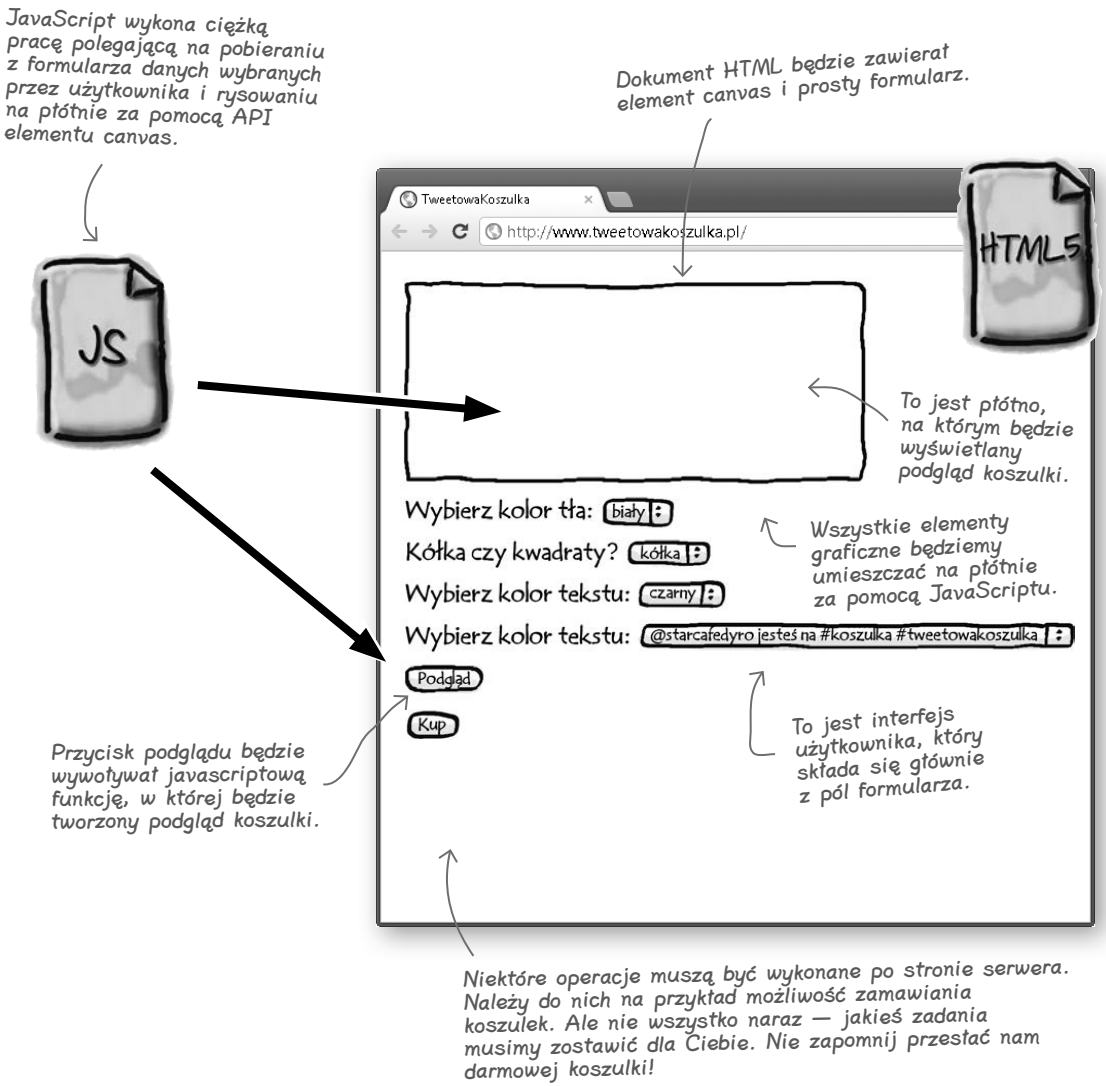
Łukasz: Wygląda na to, że się zgubiłem...

Przemek: W porządku, przejdziemy przez to razem. Zaczniemy od przyjrzenia się całości problemu.

TweetowaKoszulka — obraz całości

Zanim rzucimy się w wir pracy, zróbmy krok w tył i przyjrzyjmy się całości. Zamierzamy stworzyć aplikację internetową z *elementem canvas* w roli głównej i kilkoma polami formularza pełniącymi rolę interfejsu użytkownika. Wszystko to będzie spięte kodem JavaScript i interfejsem *API elementu canvas*.

Wygląda to tak:



BĄDŹ przeglądarką

Poniżej zamieściliśmy kod HTML formularza będącego interfejsem użytkownika. Twoim zadaniem jest odegranie roli przeglądarki i wyrenderowanie tego interfejsu. Kiedy skończysz, porównaj swój wynik z tym, co przedstawiliśmy na poprzedniej stronie. Wszystko się zgadza?

```
<form>
<p>
   <label for="backgroundColor">Wybierz kolor tła:</label>
   <select id="backgroundColor">
      <option value="white" selected="selected">biały</option>
      <option value="black">czarny</option>
   </select>
</p>
<p>
   <label for="shape">Kółka czy kwadraty?</label>
   <select id="shape">
      <option value="none" selected="selected">brak</option>
      <option value="circles">kółka</option>
      <option value="squares">kwadraty</option>
   </select>
</p>
<p>
   <label for="foregroundColor">Wybierz kolor tekstu:</label>
   <select id="foregroundColor">
      <option value="black" selected="selected">czarny</option>
      <option value="white">biały</option>
   </select>
</p>
<p>
   <label for="tweets">Wybierz tweet:</label>
   <select id="tweets">
   </select>
</p>
<p>
   <input type="button" id="previewButton" value="Podgląd">
</p>
</form>
```

Wyrenderuj interfejs użytkownika. Narysuj stronę, której kod widzisz po lewej.

BĄDŹ przeglądarką — ciąg dalszy

Załóż, że wartości parametrów dla koszulki pobierasz z elementów interfejsu.

Skoro masz już interfejs, wykonaj ten kod JavaScript i zapisz wartości dla każdego elementu. Odpowiedzi możesz sprawdzić z naszym rozwiązaniem, które znajdziesz pod koniec rozdziału.

```
var selectObj = document.getElementById("backgroundColor");
var index = selectObj.selectedIndex;
var bgColor = selectObj[index].value;   ..................................................

var selectObj = document.getElementById("shape");
var index = selectObj.selectedIndex;
var shape = selectObj[index].value;     ..................................................

var selectObj = document.getElementById("foregroundColor");
var index = selectObj.selectedIndex;
var fgColor = selectObj[index].value;   ..................................................
```

Najpierw umieśćmy HTML na swoim miejscu

Koniec gadania! Zabierzmy się w końcu do pracy. Zanim zajmiemy się ciekawszymi sprawami, potrzebujemy prostej strony HTML. Zaktualizuj plik *index.html*, tak by wyglądał w ten sposób:

```
<!doctype html>
<html lang="pl">
<head>
   <title>TweetowaKoszulka</title>
   <meta charset="utf-8" />
   <style>
      canvas {border: 1px solid black;}
   </style>
   <script src="tweetshirt.js"></script>
</head>
<body>
   <h1>TweetowaKoszulka</h1>
   <canvas width="600" height="200" id="tshirtCanvas">
      <p>Aby korzystać z aplikacji TweetowaKoszulka, musisz zaktualizować przeglądarkę!</p>
   </canvas>
   <form>


   </form>
</body>
</html>
```

O tak, kolejny fajny plik zgodny z HTML5!

Zwróć uwagę, że zmieniliśmy tytuł na „TweetowaKoszulka".

Cały kod JavaScript umieścimy w osobnym pliku, aby łatwiej nam się pracowało.

A oto płótno!

Dodaliśmy też krótki komunikat dla użytkowników starszych przeglądarek.

To jest formularz, w którym umieścimy wszystkie kontrolki naszej aplikacji. Zajmiemy się tym już na następnej stronie...

WYSIL SZARE KOMÓRKI

Co jeszcze musisz wiedzieć, by zamienić ramkę płótna dodaną za pomocą CSS na ramkę rysowaną z poziomu JavaScriptu? Którą z metod uważasz za lepszą (CSS kontra JavaScript) i dlaczego?

Teraz możemy dodać formularz

Dobra, teraz zajmiemy się interfejsem użytkownika, tak byśmy w końcu mogli napisać trochę kodu tworzącego podgląd koszulki. Co prawda już to wcześniej widziałeś, ale tym razem dodaliśmy kilka komentarzy, żeby wszystko stało się jasne, więc poza wpisaniem kodu koniecznie je przeczytaj:

Cały ten kod powinien się znaleźć między znacznikami <form>, które wstawiliśmy na poprzedniej stronie.

```
<form>
<p>
   <label for="backgroundColor">Wybierz kolor tła:</label>
   <select id="backgroundColor">
      <option value="white" selected="selected">biały</option>
      <option value="black">czarny</option>
   </select>
</p>
<p>
   <label for="shape">Kółka czy kwadraty?</label>
   <select id="shape">
      <option value="none" selected="selected">brak</option>
      <option value="circles">kółka</option>
      <option value="squares">kwadraty</option>
   </select>
</p>
<p>
   <label for="foregroundColor">Wybierz kolor tekstu:</label>
   <select id="foregroundColor">
      <option value="black" selected="selected">czarny</option>
      <option value="white">biały</option>
   </select>
</p>
<p>
   <label for="tweets">Wybierz tweet:</label>
   <select id="tweets">
   </select>
</p>
<p>
   <input type="button" id="previewButton" value="Podgląd">
</p>
</form>
```

Tutaj użytkownik wybiera kolor tła nadruku na koszulkę. Do wyboru ma biały i czarny, ale oczywiście możesz dodać inne kolory.

Kolejna kontrolka <select> służy do wyboru kształtów nakładanych na tło (kółek lub kwadratów). Użytkownik może też wybrać opcję „brak" (w takim przypadku tło nie powinno zawierać dodatkowych elementów).

Ta kontrolka służy do wyboru koloru tekstu (dostępne opcje to czarny i biały).

Tutaj znajdą się wszystkie tweety. Dlaczego teraz nic tu nie ma? Wypełnimy to później (podpowiedź: musimy je pobrać z Twittera; w końcu to jest aplikacja internetowa, prawda?!).

Na końcu umieszczamy przycisk generujący podgląd nadruku na koszulkę.

Jeżeli nie pierwszy raz masz kontakt z formularzami, możesz się zastanawiać, dlaczego pominęliśmy atrybut action (przecież bez niego nic się nie stanie po kliknięciu przycisku). Już za chwilę się tym zajmiemy...

Czas na obliczenia — w roli głównej JavaScript

Znaczniki są fajne, ale to JavaScript odpowiada za spięcie wszystkich elementów aplikacji internetowej. Umieścimy więc trochę kodu w pliku *tweetshirt.js*. Zaczniemy od małego kroku i zajmiemy się rysowaniem kwadratów w losowych miejscach nadruku na koszulkę. Ale żebyśmy mogli wykonać ten krok, musimy uruchomić przycisk podglądu, tak by jego kliknięcie powodowało wywołanie funkcji.

Utwórz plik tweetshirt.js i wpisz w nim ten kod.

Najpierw pobieramy element previewButton.

```
window.onload = function() {
    var button = document.getElementById("previewButton");
    button.onclick = previewHandler;
};
```

Do przycisku dodajemy funkcję zwrotną, dzięki czemu po kliknięciu (lub puknięciu w przypadku urządzeń mobilnych) jest wywoływana funkcja previewHandler.

A zatem po kliknięciu przycisku podglądu zostanie wywołana funkcja `previewHandler`, w której możemy zaktualizować zawartość płótna, tak by odpowiadała parametrom ustawionym przez użytkownika. W takim razie zaczynamy pisać funkcję `previewHandler`:

```
function previewHandler() {
    var canvas = document.getElementById("tshirtCanvas");
    var context = canvas.getContext("2d");

    var selectObj = document.getElementById("shape");
    var index = selectObj.selectedIndex;
    var shape = selectObj[index].value;

    if (shape == "squares") {
        for (var squares = 0; squares < 20; squares++) {
            drawSquare(canvas, context);
        }
    }
}
```

Najpierw pobieramy element canvas i prosimy go o kontekst 2d.

Teraz musimy sprawdzić, jaki kształt wybrał użytkownik. Zaczynamy od pobrania elementu o identyfikatorze „shape".

Następnie sprawdzamy, która pozycja została wybrana (kwadraty czy kółka). W tym celu odczytujemy indeks wybranej pozycji i przypisujemy go do zmiennej shape.

Jeżeli wartością zmiennej shape jest „squares", musimy narysować kilka kwadratów. Co powiesz na 20 sztuk?

Za rysowanie pojedynczego kwadratu odpowiada funkcja drawSquare, którą musimy sami napisać. Zwróć uwagę, że przekazujemy jej zarówno element canvas, jak i kontekst. Już za chwilę zobaczysz, dlaczego tak postąpiliśmy.

Nie istnieją głupie pytania

P: Jak działa właściwość `selectedIndex`?

O: Kontrolki `<select>` formularzy mają właściwość `selectedIndex`, która zwraca numer opcji wybranej z rozwijanego menu. Każda lista opcji jest zamieniana na tablicę, której elementy to kolejne pozycje listy. Powiedzmy, że mamy listę trzech opcji: „pizza", „pączki" i „herbatniki". Jeżeli zaznaczysz „pączki", właściwość `selectedIndex` przyjmie wartość 1 (pamiętaj, że tablice w JavaScripcie są indeksowane od 0).

Bardzo często sam indeks Ci nie wystarczy i chciałbyś poznać wartość opcji o danym indeksie (w naszym przypadku „pączki"). Aby poznać wartość opcji, najpierw trzeba odczytać z tablicy element o tym indeksie, co spowoduje zwrócenie *obiektu* opcji. Do *wartości* tego obiektu można się dostać za pośrednictwem właściwości `value`, która zwraca łańcuch tekstowy przypisany do atrybutu `value` tej opcji.

Magnesiki z pseudokodem

Użyj swoich tajemnych mocy programistycznych i poskładaj pseudokod zamieszczony poniżej, który ma ilustrować działanie funkcji `drawSquare`. Do tej funkcji są przekazywane element canvas i kontekst, a jej zadaniem jest narysowanie kwadratu o losowej wielkości w losowym miejscu płótna. Zanim przejdziesz dalej, porównaj swoje rozwiązanie z naszym, które znajdziesz na końcu tego rozdziału.

Ten magnesik umieściliśmy za Ciebie.

```
function drawSquare (          ,  context  ){

                                  Magnesiki umieść w tym miejscu !

}
```

We wstępnym projekcie ustaliliśmy, że kwadraty będą jasnoniebieskie, stąd to „lightblue".

canvas

narysuj kwadrat o szerokości w we współrzędnych x i y

wyznacz losową wartość współrzędnej y, tak by kwadrat znalazł się wewnątrz płótna

ustaw właściwość fillStyle na "lightblue"

wyznacz losową szerokość kwadratu

wyznacz losową wartość współrzędnej x, tak by kwadrat znalazł się wewnątrz płótna

Piszemy funkcję drawSquare

Jeżeli już wykonałeś całą tę ciężką pracę i uporządkowałeś pseudokod, czas przekuć wiedzę w czyn i napisać działającą funkcję `drawSquare`:

Funkcja ma dwa parametry: canvas (czyli płótno) oraz context (czyli kontekst rysowania).

Do wygenerowania losowej wartości szerokości oraz współrzędnych x i y używamy metody Math.random(). Za chwilę powiemy o niej trochę więcej...

```
function drawSquare(canvas, context) {
  var w = Math.floor(Math.random() * 40);

  var x = Math.floor(Math.random() * canvas.width);

  var y = Math.floor(Math.random() * canvas.height);

  context.fillStyle = "lightblue";
  context.fillRect(x, y, w, w);
}
```

Potrzebujemy losowych wartości dla szerokości kwadratu oraz jego współrzędnych x i y.

Maksymalna długość boku kwadratu jest równa 40, by nie był za duży.

Współrzędne x i y są obliczane w oparciu o szerokość i wysokość płótna. Generujemy losową wartość z przedziału od 0 do, odpowiednio, szerokości lub wysokości.

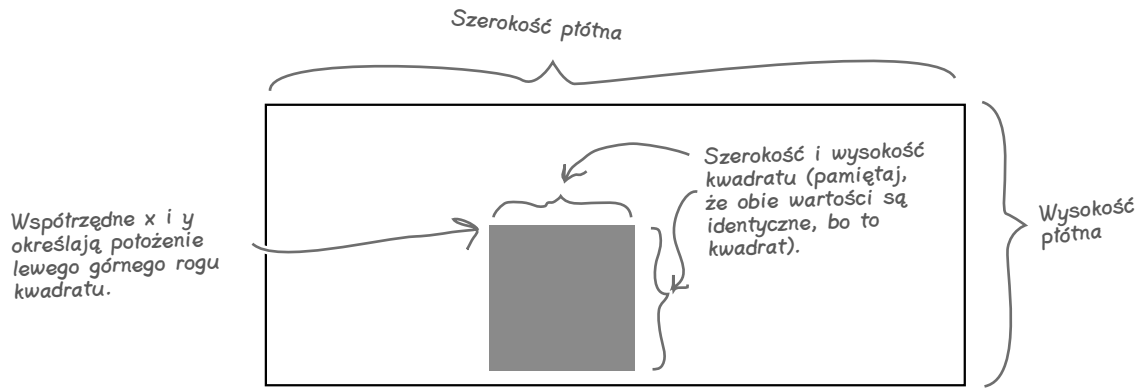
Za pomocą właściwości fillStyle ustalamy, by kwadraty miały jasnoniebieski kolor. O tej właściwości też powiemy za chwilę...

W końcu rysujemy kwadrat za pomocą metody fillRect.

Jeżeli chcesz sobie przypomnieć co nieco na ten temat, zajrzyj do książki „Head First HTML with CSS & XHTML. Edycja polska".

W jaki sposób dobraliśmy liczby, przez które mnożymy wartości zwracane przez metodę `Math.random`, by otrzymać szerokość oraz współrzędne x i y? W przypadku szerokości ustaliliśmy wartość 40, ponieważ naszym zdaniem jest odpowiednia dla tej wielkości płótna. Ponieważ jest to kwadrat, stosujemy tę samą wartość dla szerokości i wysokości. Do obliczenia współrzędnych używamy wymiarów płótna (jego szerokości i wysokości), tak by kwadrat zmieścił się w jego granicach.

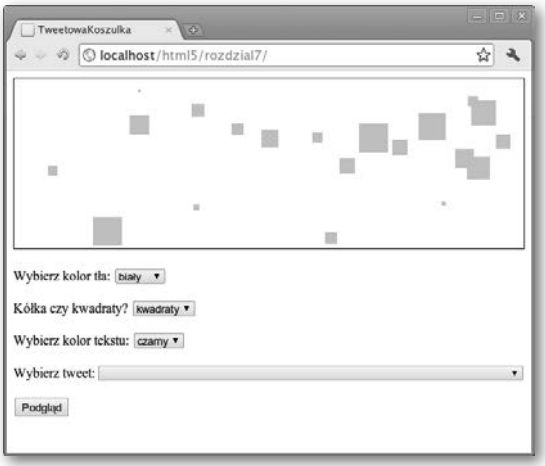
W swoim kodzie możesz wpisać inną wartość!



Czas na jazdę próbną!

No dobra, koniec pisania, teraz czas na testowanie! Otwórz w przeglądarce dokument *index.html* i kliknij przycisk podglądu, aby zobaczyć niebieskie kwadraty.

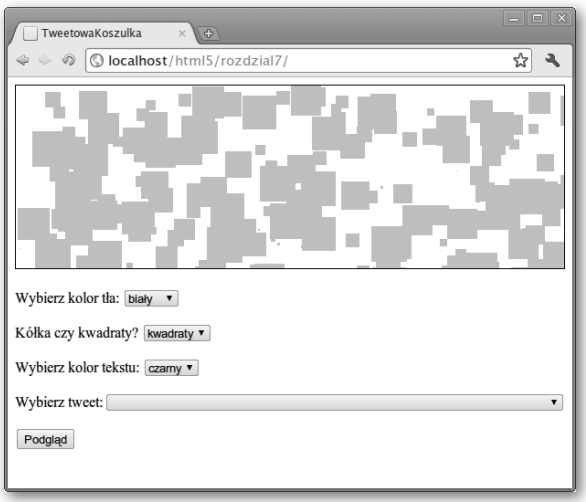
My zobaczyliśmy coś takiego:



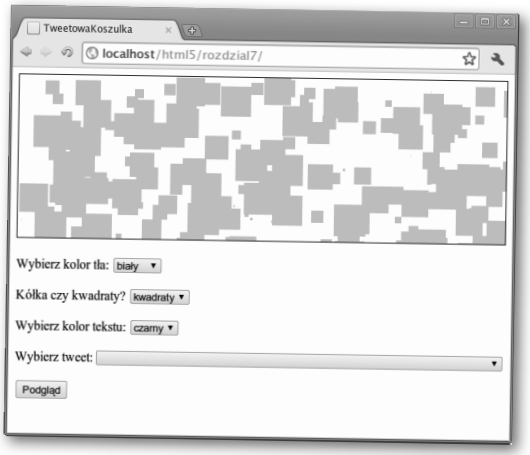
Świetnie, właśnie o to nam chodziło!

Chwila, nie tak szybko... Jeżeli wciśniesz przycisk podglądu kilka razy, zobaczysz MNÓSTWO kwadratów. Nie powinno tak być!

On niestety ma rację, mamy mały problem. Wciśnij przycisk podglądu kilka razy, a zobaczysz coś takiego.



Dlaczego na podglądzie widzimy stare i nowe kwadraty?



Tak właściwie to efekt jest całkiem ciekawy... ale z całą pewnością nie o to nam chodziło. Chcieliśmy, by nowe kwadraty *zastępowały* stare za każdym razem, gdy klikamy przycisk podglądu (podobnie będzie z tweetami — nowe muszą zastępować stare).

Kluczową sprawą, o której musisz pamiętać, jest to, że my malujemy piksele na płótnie. Kiedy klikniesz przycisk podglądu, pobierany jest element canvas i rysowane są na nim kwadraty. To, co znajduje się na płótnie, jest zamalowywane nowymi rysunkami!

Nie martw się. Wiesz już wystarczająco dużo, by rozwiązać ten problem. Oto, co zamierzamy zrobić:

1. Odczytujemy wybrany kolor tła z obiektu „backgroundColor".
2. Za każdym razem, zanim zaczniemy rysować kwadraty, wypełniamy tło, używając właściwości fillStyle i metody fillRect.

Zaostrz ołówek

Aby po każdym kliknięciu przycisku podglądu widzieć tylko nowe kwadraty, musimy za każdym razem zaczynać od wypełnienia płótna kolorem tła wybranym przez użytkownika za pomocą pola „backgroundColor". Zaczniemy od zaimplementowania funkcji wypełniającej płótno wybranym kolorem. W puste pola w poniższym kodzie wstaw brakujące fragmenty. Zanim przejdziesz dalej, porównaj swoje odpowiedzi z naszymi, które znajdziesz pod koniec tego rozdziału.

```
function fillBackgroundColor(canvas, context) {
    var selectObj = document.getElementById("_____________");
    var index = selectObj.selectedIndex;
    var bgColor = selectObj.options[index].value;
    context.fillStyle = _____________;
    context.fillRect(0, 0, _____________, _____________);
}
```

Podpowiedź: to, co odczytasz z obiektu wybranej opcji, będzie łańcuchem tekstowym opisującym kolor, którego możesz użyć tak samo jak wartości „lightblue" dla kwadratów.

Podpowiedź: chcemy wypełnić tym kolorem CAŁE płótno!

Dodajemy wywołanie metody fillBackgroundColor

Mamy nadzieję, że masz już kompletną funkcję `fillBackgroundColor`. Teraz wystarczy ją wywołać z funkcji `previewHandler`. Dodamy ją w takim miejscu, by była wywoływana na samym początku, dzięki czemu przed narysowaniem czegokolwiek będziemy dysponować pięknym, czyściutkim tłem.

```
function previewHandler() {
    var canvas = document.getElementById("tshirtCanvas");
    var context = canvas.getContext("2d");
    fillBackgroundColor(canvas, context);

    var selectObj = document.getElementById("shape");
    var index = selectObj.selectedIndex;
    var shape = selectObj[index].value;

    if (shape == "squares") {
        for (var squares = 0; squares < 20; squares++) {
            drawSquare(canvas, context);
        }
    }
}
```

Wywołanie funkcji fillBackgroundColor dodajemy przed kodem rysującym kwadraty, dzięki czemu nowy rysunek powstanie na pustym tle.

Kolejna krótka jazda próbna — tym razem sprawdzamy, czy działa funkcja fillBackgroundColor

Dopisz kod do pliku *tweetshirt.js* i przeładuj stronę w przeglądarce, a następnie wybierz kwadraty i kliknij przycisk podglądu. Kliknij jeszcze raz. Teraz za każdym razem, gdy klikasz przycisk, powinieneś zobaczyć tylko nowe kwadraty.

Teraz za każdym razem, gdy generujemy podgląd, widzimy tylko nowe kwadraty.

WYSIL SZARE KOMÓRKI

Policz kwadraty w kilku różnych podglądach. Czy nie jest tak, że zawsze jest ich mniej niż 20? Być może.

Dlaczego tak się dzieje? Co zrobić, by to naprawić? (Przecież nie chcesz oszukać swoich klientów na liczbie kwadratów, prawda?)

Zbliżenie na JavaScript

Przypatrzmy się dokładniej właściwości `fillStyle`, ponieważ dopiero niedawno ją poznałeś. Jest to właściwość obiektu kontekstu, która przechowuje kolor rysunków umieszczanych na płótnie.

Tak samo jak w przypadku metody fillRect, właściwość fillStyle należy do kontekstu.

Jednak w przeciwieństwie do fillRect, fillStyle jest właściwością, a nie metodą, więc jej nie wywołujemy, ale ją ustawiamy.

A tym, co ustawiamy za jej pomocą, jest kolor. Możesz zastosować ten sam format zapisu koloru jak w przypadku stylów. Dopuszczalne jest więc użycie nazw kolorów (jak „lightblue"), wartości w postaci #ccccff lub rgb(0, 173, 239). Sprawdź sam!

```
context.fillStyle = "lightblue";
```

Zwróć uwagę, że — inaczej niż w przypadku stylów — wartości musisz ujmować w cudzysłowy (chyba że korzystasz ze zmiennych).

Nie istnieją głupie pytania

P: Spodziewałem się, że kolory tła i kwadratów przekażemy metodzie `fillRect`. Nie do końca rozumiem, o co chodzi z tym `fillStyle`. W jaki sposób ta właściwość wpływa na działanie metody `fillRect`?

O: Świetne pytanie! To faktycznie może wyglądać dosyć dziwnie. Pamiętaj, że kontekst jest obiektem kontrolującym dostęp do płótna. Ustawiając właściwość `fillStyle`, informujesz płótno, że cokolwiek zostanie na nim narysowane, ma być w takim właśnie kolorze. A zatem cokolwiek będziesz wypełniał kolorem (np. prostokąt narysowany metodą `fillRect`) po ustawieniu właściwości `fillStyle`, będzie zawsze w tym kolorze aż do ponownego przypisania do właściwości `fillStyle` innego koloru.

P: Dlaczego kolor muszę podawać w cudzysłowie, skoro definiując style, nie muszę tego robić? Przecież nie używam cudzysłowów, gdy określam na przykład wartość `background-color`.

O: No cóż, język CSS jest zupełnie inny niż JavaScript. I tak się składa, że w tej sytuacji w CSS nie stosuje się cudzysłowów. Jednak gdybyś w JavaScripcie nie umieścił nazwy koloru w cudzysłowie, zostałaby potraktowana jako nazwa zmiennej, więc ustawiłbyś kolor na podstawie wartości tej zmiennej, a nie samego koloru.

Przypuśćmy, że masz zmienną `fgColor = "black"`. Jeżeli napiszesz `context.fillStyle = fgColor`, to zadziała, ponieważ wartością zmiennej `fgColor` jest `"black"`.

Jednak kod `context.fillStyle = black` nie zadziała, ponieważ `black` nie jest zmienną (chyba że takiej zmiennej używasz, a to z kolei może być mylące). O swojej pomyłce możesz się dowiedzieć dzięki komunikatowi o błędzie JavaScriptu, który informuje o nieznalezieniu zmiennej `black`. (Nie przejmuj się, każdy czasem popełnia błędy. Ważne, żeby się na nich uczyć!)

P: Dobra, poddaję się! Nie mam pojęcia, dlaczego pojawia się mniej niż 20 kwadratów. O co tu chodzi?

O: Współrzędne x i y oraz szerokość kwadratu to wartości losowe, a więc niektóre kwadraty mogą być lepiej widoczne od innych. Jeżeli zostaną wylosowane na przykład współrzędne 599; 199, będziesz w stanie zobaczyć tylko jeden piksel kwadratu (bo pozostała jego część znajdzie się poza płótnem). Niektóre kwadraty mogą mieć bok o długości 1 piksela, a niektóre nawet 0, ponieważ metoda `Math.random` może zwrócić 0. Może być też tak, że dwa kwadraty znajdą się w tym samym miejscu, więc zleją się w jeden.

W przypadku tej aplikacji nie jest to wielką wadą — w końcu na tym polega „losowość" takiego tła. Być może w innej sytuacji tego typu działanie byłoby niedopuszczalne, więc należałoby je wyeliminować.

Tymczasem na TweetowaKoszulka.com...

Kuba: Racja. W dodatku jestem pod wrażeniem, jak mało kodu trzeba było wpisać. Pomyśl tylko, że gdybyśmy postąpili tak, jak robiliśmy to wcześniej, i zdecydowali się na rozwiązanie tego po stronie serwera, bylibyśmy jeszcze daleko w lesie.

Łukasz: Poza tym wygląda na to, że jesteśmy już całkiem blisko załatwienia sprawy kółek — w końcu niewiele się różnią od kwadratów.

Kuba: Zgadzam się. Słuchaj, a gdzie jest Iza? Ona już na pewno poznała całe API z kółkami włącznie. Domyślam się, że wystarczy wywołać metodę `fillCircle`.

Łukasz: Brzmi nieźle! W sumie to po co nam Iza? Sami sobie z tym poradzimy!

Kilka godzin później...

Łukasz: Nie rozumiem, o co chodzi... Dwa razy wszystko sprawdziłem, a to dalej nie chce działać! Bez względu na to, co robię, wywołanie metody `fillCircle` nie przynosi żadnych efektów.

Iza: Dobra, to pokaż mi tę twoją metodę `fillCircle`.

Łukasz: Hm... jak to „moją”? Ja jej nie napisałem — korzystam z metody API elementu canvas...

Iza: Przecież w API płótna nie ma metody `fillCircle`!

Łukasz: A... nie ma? No wiesz, *założyłem*, że jest, skoro korzystaliśmy z `fillRect`...

Iza: Założyć to sobie możesz nogę na nogę! Odpal przeglądarkę i sprawdź, co siedzi w tym API:

http://dev.w3.org/html5/2dcontext/.

No dobrze, tak w skrócie: rysowanie kółek wymaga trochę więcej wysiłku niż wywołanie jednej metody. Musisz najpierw dowiedzieć się co nieco o ścieżkach i łukach.

Kuba (wchodzi): Cześć, Iza! Czy Łukasz już ci mówił, jak sobie elegancko poradziliśmy z kółkiem?

Łukasz: Tak jakby... Kuba, *ienaj aczynajzaj ematutaj, nowuzaj topawaj!*

Polecamy serwis tłumaczący na piglatin.bavetta.com.

Rysujemy z geekami...

Zanim zajmiemy się kółkami, musimy pomówić o ścieżkach i łukach. Na pierwszy ogień pójdą ścieżki — za ich pomocą narysujemy kilka trójkątów. Jak się pewnie domyślasz, nie ma metody `fillTriangle`, która rysowałaby na płótnie trójkąt. W związku z tym, aby powstał trójkąt, musimy najpierw utworzyć *ścieżkę* w kształcie trójkąta, a później *odrysować* ją na płótnie.

Co to oznacza? Wyobraź sobie, że chcesz coś bardzo precyzyjnie narysować. Mógłbyś wziąć ołówek i delikatnie naszkicować nim kształt (nazwijmy go ścieżką). Ten szkic jest tak delikatny, że tylko Ty go widzisz. Później, kiedy jesteś już zadowolony ze swojego dzieła, bierzesz do ręki pióro (o określonej grubości i wybranym kolorze) i rysujesz nim na szkicu, tak żeby rysunek (który naszkicowałeś ołówkiem) stał się widoczny.

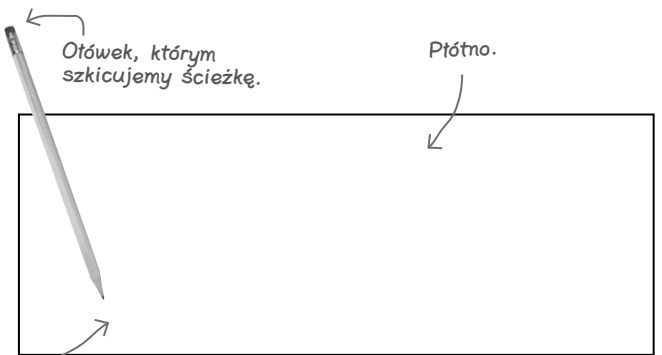
Właśnie na tej zasadzie rysuje się na płótnie dowolne kształty. Spróbujmy więc narysować trójkąt:

Za pomocą metody beginPath informujemy płótno, że rozpoczynamy nową ścieżkę.

```
context.beginPath();
context.moveTo(100, 150);
```

Metoda moveTo służy do przesunięcia „ołówka" w określone miejsce płótna. Wyobraź to sobie jako przytknięcie ołówka w danym miejscu.

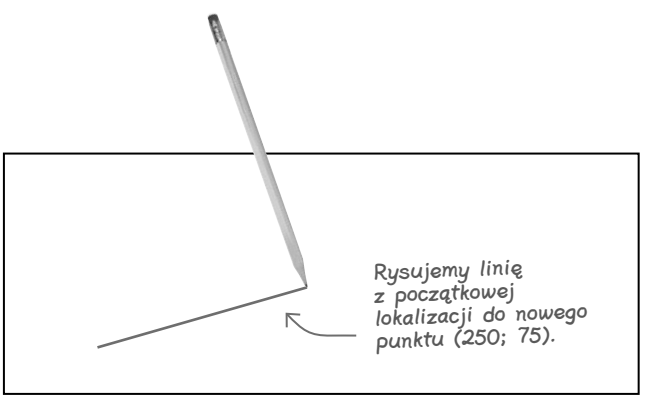
W tym przypadku przytykamy ołówek we współrzędnych x = 100 i y = 150. To jest pierwszy punkt ścieżki.



Za pomocą metody lineTo tworzymy ścieżkę od bieżącej lokalizacji do innego punktu na płótnie.

```
context.lineTo(250, 75);
```

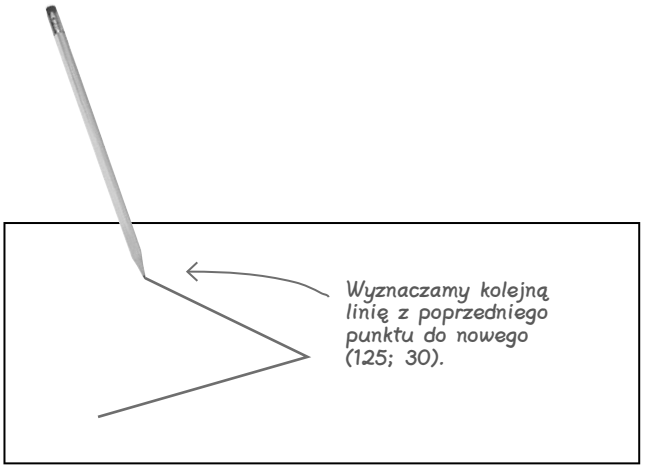
Ołówek był początkowo we współrzędnych 100; 150, a teraz przenosimy go do punktu o współrzędnych x = 250, y = 75, pozostawiając ścieżkę.



Mamy już pierwszy bok trójkąta, więc pora na drugi. W tym celu posłużymy się znowu metodą lineTo:

```
context.lineTo(125, 30);
```

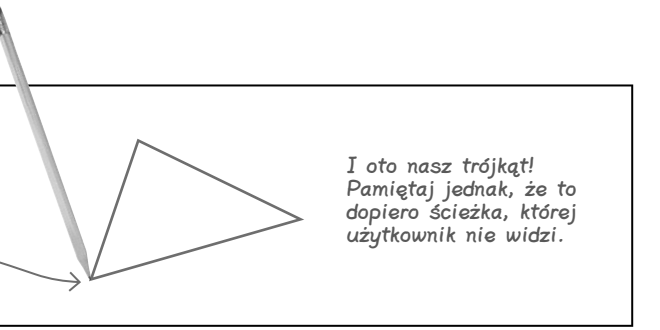
Tutaj tworzymy szkic z bieżącej pozycji otówka (250; 75) do nowego punktu o współrzędnych x = 125 i y = 30. W ten sposób narysowaliśmy drugi bok.



Już prawie skończyliśmy! Pozostała do narysowania tylko jedna linia, która zamknie ścieżkę. W tym celu posłużymy się metodą closePath().

```
context.closePath();
```

Metoda closePath łączy początkowy punkt ścieżki (100; 150) z końcowym (125; 30).



Masz już ścieżkę! Co dalej?

Ćwiczenie

Dzięki ścieżce będziesz mógł narysować linie i wypełnić powstały kształt kolorem! Utwórz prosty dokument HTML z elementem canvas i wpisz cały kod, który do tej pory napisaliśmy. Potem koniecznie go przetestuj!

```
context.beginPath();
context.moveTo(100, 150);
context.lineTo(250, 75);
context.lineTo(125, 30);
context.closePath();
context.lineWidth = 5;

context.stroke();

context.fillStyle = "red";

context.fill();
```

To jest kod, który do tej pory napisaliśmy.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

A tu fragment nowego kodu. Pomyśl chwilę i zapisz, za co Twoim zdaniem on odpowiada. Otwórz stronę w przeglądarce. Miałeś rację? Odpowiedź znajdziesz na końcu tego rozdziału.

Aby utworzyć kółko, musimy najpierw utworzyć ścieżkę.

Pokażemy Ci, jak naszkicować ścieżkę w kształcie koła. Kiedy będziesz już to wiedział, możesz rysować takie kółka, jakie tylko chcesz.

Przejdźmy do konkretów. Wiesz już, jak rozpocząć ścieżkę, prawda? Robimy to tak jak poprzednio:

```
context.beginPath();
```

Teraz skorzystamy z metody kontekstu, o której jeszcze nie wspominaliśmy, mianowicie `arc`:

```
context.arc(150, 150, 50, 0, 2 * Math.PI, true);
```

Czy pamiętasz wzór na obwód koła? Chwila... jak to było? $2\pi r$? Czas na małą powtórkę z matematyki...

I co to robi? No cóż, na szczegółowy opis poświęciliśmy kilka kolejnych stron, ale — jak się pewnie domyślasz — w ten sposób rysujemy ścieżkę w kształcie koła.

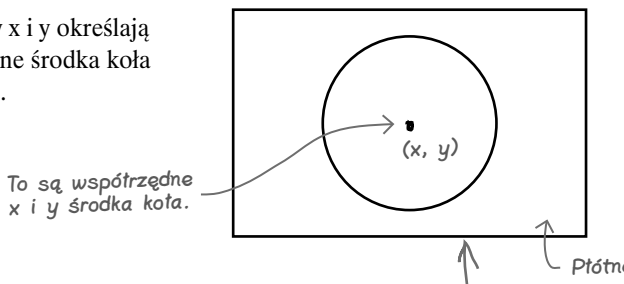
Rozkładamy metodę arc na części pierwsze

Przyjrzyjmy się metodzie `arc` (która tak naprawdę służy do rysowania łuków, a nie tylko kół) i sprawdźmy, do czego służą jej parametry.

```
context.arc(x, y, promień, kątPoczątkowy, kątKońcowy, kierunek)
```

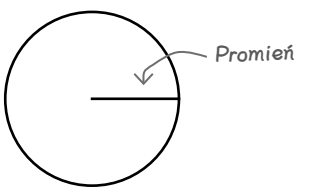
Podczas korzystania z metody `arc` najistotniejsze jest określenie, jak ma wyglądać ścieżka rysowana po kole. Przypatrzmy się, jaki wpływ na końcowy efekt mają poszczególne parametry:

x,y Parametry x i y określają współrzędne środka koła na płótnie.

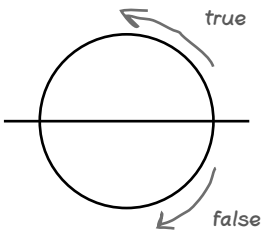



```
context.arc(x, y, promień,
```

promień Ten parametr określa połowę szerokości koła.



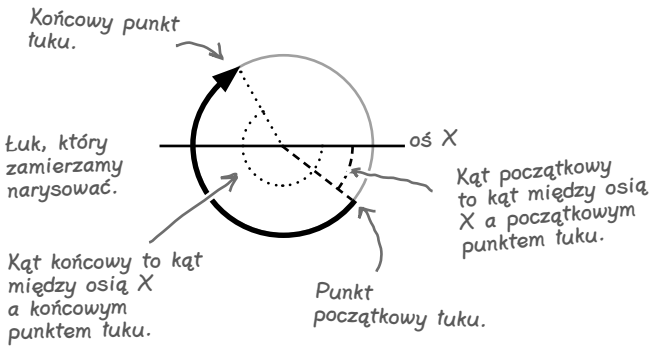
kierunek Ten parametr określa, czy łuk jest tworzony w kierunku zgodnym z ruchem wskazówek zegara, czy przeciwnym. Jeżeli parametr ma wartość `true`, obowiązuje kierunek przeciwny do ruchu wskazówek zegara, a jeżeli `false` — zgodny.



Jeżeli parametr „kierunek" ma wartość true, łuk jest rysowany przeciwnie do ruchu wskazówek zegara, a jeżeli false — zgodnie z nim.

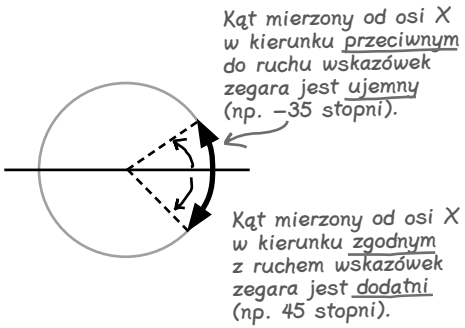
kątPoczątkowy, kątKońcowy, kierunek)

kątPoczątkowy, kątKońcowy Początkowy i końcowy kąt określają miejsca na okręgu, w których łuk się rozpoczyna i kończy.



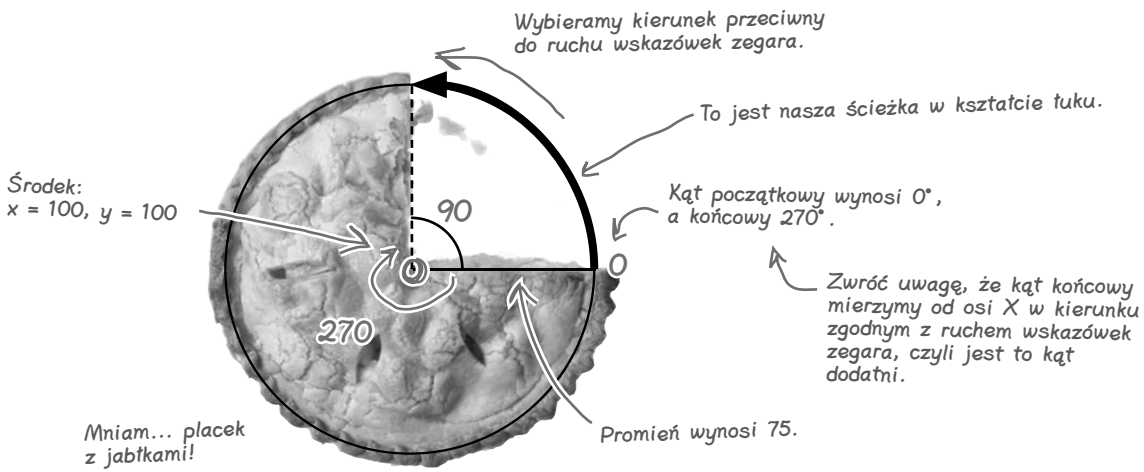
Bardzo ważny drobiazg!

Koniecznie to przeczytaj. Kąty mogą być mierzone w kierunku ujemnym (przeciwnie do ruchu wskazówek zegara) lub dodatnim (zgodnie z ruchem wskazówek zegara). To nie ten kierunek określa parametr `kierunek` metody `arc`! (Powiemy jeszcze o tym na następnej stronie).



Smaczki metody arc

Teraz potrzebujemy dobrego przykładu. Powiedzmy, że chcesz wyznaczyć łuk wzdłuż koła o szerokości 150 pikseli (czyli o promieniu 75) oraz środku w punkcie o współrzędnych x = 100 i y = 100. A łuk, który ma powstać, ma być tylko jedną czwartą koła:



Utwórzmy teraz wywołanie metody `arc`, która wyznaczy taki łuk:

1. Zaczynamy od współrzędnych środka koła: x = 100, y = 100.

```
context.arc(100, 100, __, __, _____________________, ____ );
```

2. Następnie określamy promień koła równy 75.

```
context.arc(100, 100, 75, __, _____________________, ____ );
```

3. A co z początkowym i końcowym kątem? No tak, kąt początkowy wynosi 0, ponieważ początkowy punkt leży na osi X. Kąt końcowy wyznaczamy między osią X a punktem końcowym łuku. Ponieważ łuk ma kąt 90 stopni, końcowy kąt będzie wynosił 270 stopni (90+270 = 360). (Gdybyśmy określali kąty ujemne, czyli w kierunku przeciwnym do ruchu wskazówek zegara, kąt końcowy byłby równy –90 stopni).

```
context.arc(100, 100, 75, 0, degreesToRadians(270), ____ );
```

Już za chwilę do tego wrócimy. Ta funkcja zamienia stopnie (bo ich używamy) na radiany (które są wyraźnie preferowane przez obiekt kontekstu).

4. Na samym końcu podajemy kierunek rysowania łuku: wybraliśmy przeciwny do ruchu wskazówek zegara, więc przekazujemy wartość `true`.

```
context.arc(100, 100, 75, 0, degreesToRadians(270), true);
```

Ja mówię stopnie, a Ty radiany

Wbrew pozorom bardzo często zdarza nam się mówić o kątach: „niezłe 360", „zwrot o 180 stopni" albo... no dobrze, już wiesz, o czym mowa. Jest jednak pewien problem: my myślimy w *stopniach*, a kontekst płótna woli *radiany*.

Radiany to inna jednostka miary kąta. Jeden radian to 180/3,14159265... (lub, inaczej mówiąc, 180 dzielone przez π).

Zacznijmy od tego, że:

```
360 stopni = 2π radianów
```

Jeżeli czujesz się na siłach, możesz od teraz sam przeliczać stopnie na radiany. Jeżeli z jakiegoś powodu nie chcesz tego liczyć w pamięci, możesz skorzystać z poręcznej funkcji, która zrobi to za Ciebie:

```
function degreesToRadians(degrees) {
    return (degrees * Math.PI)/180;
}
```

Używaj tej funkcji zawsze wtedy, gdy w myślach używasz stopni, a potrzebujesz miary kąta w radianach (na przykład w metodzie arc).

Być może pamiętasz, że podobne wyrażenie pojawiło się w rozdziale dotyczącym geolokalizacji.

Aby przeliczyć stopnie na radiany, musisz je przemnożyć przez π i podzielić przez 180.

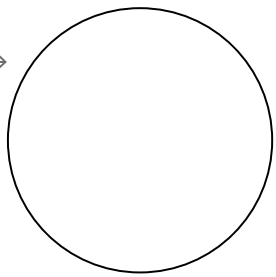
We fragmencie kodu, w którym użyliśmy metody arc do narysowania koła, końcowy kąt określiliśmy jako 2*Math.PI. Oczywiście można go podać w tej postaci, ale równie dobrze można by napisać degreesToRadians(360).

BĄDŹ przeglądarką

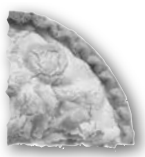
Twoim zadaniem jest wcielenie się w rolę przeglądarki. Musisz sparsować HTML i stworzyć z niego model DOM. Dalej, śmiało parsuj i rysuj na dole tej strony. Już za Ciebie zaczęliśmy.

```
context.arc(100, 100, 75, degreesToRadians(270), 0, true);
```

Zaznacz na tym kółku wartości wszystkich argumentów i narysuj ścieżkę, która powstanie po wywołaniu tej metody.



Podpowiedź: co zostanie z placka po zjedzeniu tego kawałka?



Wracamy do kodu rysującego kółka

Teraz, kiedy już wiesz, jak rysować kółka, czas na powrót do kodu naszej aplikacji. Dodamy nową funkcję — `drawCircle`. Chcemy narysować 20 losowo rozmieszczonych kółek, tak samo jak to robiliśmy z kwadratami. Najpierw musimy jednak sprawdzić, czy użytkownik zaznaczył odpowiednie pole. W tym celu zmodyfikujemy funkcję `previewHandler`.

Przejdź do pliku *tweetshirt.js* i dodaj nowy blok kodu:

```
function previewHandler() {
   var canvas = document.getElementById("tshirtCanvas");
   var context = canvas.getContext("2d");
   fillBackgroundColor(canvas, context);

   var selectObj = document.getElementById("shape");
   var index = selectObj.selectedIndex;
   var shape = selectObj[index].value;

   if (shape == "squares") {
      for (var squares = 0; squares < 20; squares++) {
         drawSquare(canvas, context);
      }
   } else if (shape == "circles") {
      for (var circles = 0; circles < 20; circles++) {
         drawCircle(canvas, context);
      }
   }
}
```

Ten kod wygląda prawie tak samo jak kod rysujący kwadraty. Jeżeli użytkownik wybrał opcję „kółka", a nie „kwadraty", rysujemy 20 kółek, korzystając z funkcji drawCircle (którą zaraz napiszemy).

Do funkcji drawCircle przekazujemy obiekty canvas i context, czyli tak samo jak w przypadku funkcji drawSquare.

WYSIL SZARE KOMÓRKI

Jakie wartości początkowego i końcowego kąta zastosujesz do narysowania pełnego koła?

Jaki wybierzesz kierunek rysowania: zgodny z ruchem wskazówek zegara czy przeciwny? Czy to ma znaczenie?

O: Początkowy kąt powinien wynosić 0°, a końcowy — 360°. Nie ma znaczenia, jaki kierunek rysowania wybierzesz, ponieważ ma powstać pełne koło.

Piszemy funkcję drawCircle...

Teraz zajmiemy się funkcją `drawCircle`. Pamiętaj, że rysowane kółka mają być losowo rozmieszczone na płótnie i mają mieć losowe wymiary. Za wywołanie tej funkcji 20 razy odpowiada kod, który przedstawiliśmy przed chwilą.

```
function drawCircle(canvas, context) {
   var radius = Math.floor(Math.random() * 40);
   var x = Math.floor(Math.random() * canvas.width);
   var y = Math.floor(Math.random() * canvas.height);

   context.beginPath();
   context.arc(x, y, radius, 0, degreesToRadians(360), true);

   context.fillStyle = "lightblue";
   context.fill();
}
```

Tak samo jak w przypadku kwadratów, ustalamy maksymalny rozmiar (w tym przypadku promień) równy 40, tak by kółka nie były za duże.

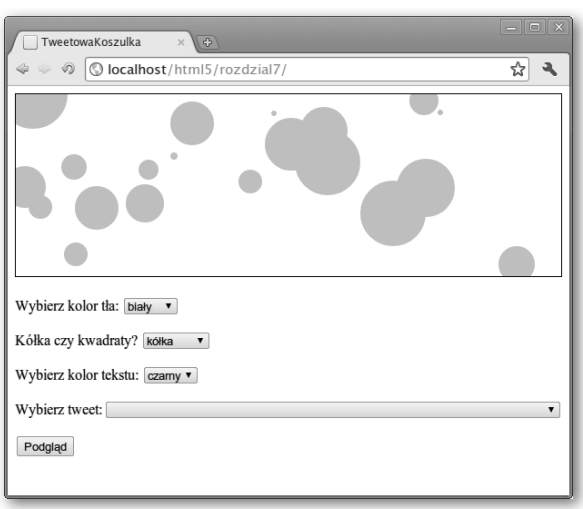
I znów podobnie — współrzędne x i y środka koła są wyznaczane w oparciu o szerokość i wysokość płótna. Wartości mieszczą się w zakresie od 0 do, odpowiednio, szerokości i wysokości.

Końcowy kąt wynosi 360°, tak aby uzyskać pełne koło. Rysujemy w kierunku przeciwnym do ruchu wskazówek zegara, ale w przypadku pełnego koła nie ma to żadnego znaczenia.

Właściwości fillStyle również przypisujemy wartość „lightblue" i wypełniamy ścieżkę metodą context.fill().

...i pędzimy na jazdę próbną!

Wpisz kod (nie zapomnij o funkcji `degreesToRadians`), zapisz plik i otwórz w przeglądarce. My zobaczyliśmy coś takiego (ponieważ kółka są rozmieszczane losowo, Ty z dużym prawdopodobieństwem zobaczysz coś innego):



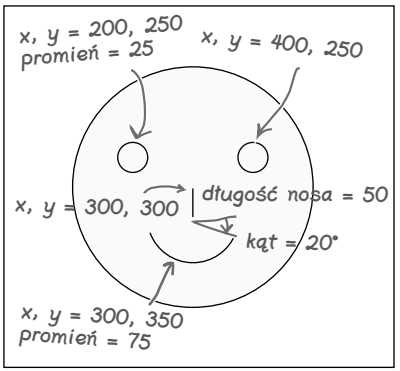
Przerwa

Krótka przerwa na ciasteczka

No, no! Ostatnie strony były niczego sobie. Nie wiemy jak Ty, ale my chętnie wzięlibyśmy coś na ząbek. Może ciasteczka? Zróbmy sobie przerwę! Nie bylibyśmy sobą, gdybyśmy nie zaproponowali Ci czegoś ciekawego, byś urozmaicił sobie czas podczas jedzenia (rzuć okiem na prawo).

Siądź więc wygodnie, odpręż się, skubnij czasem i pozwól swojemu mózgowi (oraz żołądkowi) skonsumować coś innego niż zwykle. Po przerwie wracaj — musimy skończyć kod naszej aplikacji!

Z prawej strony widzisz uśmiechniętą twarz (prawie taką samą jak na naszych ulubionych wesołych czekoladowych ciasteczkach). Kod, który umieściliśmy poniżej, jest prawie skończony. Twoim zadaniem jest uzupełnienie brakujących fragmentów. Po tym, czego się dowiedziałeś do tej pory, nie powinno to stanowić dla Ciebie żadnego problemu. Kiedy skończysz, możesz sprawdzić, czy rozwiązaliśmy to zadanie tak samo jak Ty (odpowiedzi szukaj na końcu tego rozdziału).



To chcemy uzyskać. Być może podczas pracy nabierzesz ochoty na upieczenie prawdziwych wesołych czekoladowych ciasteczek...

```
function drawSmileyFace() {
  var canvas = document.getElementById("smiley");
  var context = canvas.getContext("2d");

  context.beginPath();
  context.arc(300, 300, 200, 0, degreesToRadians(360), true);
  context.fillStyle = "#ffffcc";
  context.fill();
  context.stroke();
```

Owal twarzy. To kółko narysowaliśmy za Ciebie (i w dodatku wypełniliśmy je na żółto).

```
  context.beginPath();
  context.arc(___, ___, 25, ___, ________, true);
  context.stroke();
```

To jest lewe oko.

```
  context.beginPath();
  context.arc(400, ___, ___, ___, _______, _____);
  context.stroke();
```

To jest prawe oko.

```
  context.beginPath();
  context._______(___, ___);
  context._______(___, ___);
  context._______();
```

To jest nos.

```
  context.beginPath();
  context.___(300, 350, ___, degreesToRadians(___), degreesToRadians(___), _____);
  context.stroke();
}
```

A to usta. Uważaj, bo to jest podchwytliwe!

Witamy po przerwie...

Już wróciłeś, wypoczęty i odświeżony, więc pora zabrać się za wykańczanie aplikacji. Jeżeli się nad tym zastanowić, to zostało nam tylko wyświetlanie na płótnie wybranych tweetów i innych tekstów.

No właśnie, abyśmy mogli umieścić na płótnie jakieś tweety, musimy je najpierw pobrać — a w tym zadaniu pomoże nam JSONP. Jeżeli pamiętasz jeszcze to, o czym mówiliśmy w rozdziale 6., z pewnością wiesz, jak można to zrobić. Jeśli chcesz, możesz jeszcze zerknąć do tego rozdziału i odświeżyć sobie pamięć. Oto, co zamierzamy zrobić:

1. Na końcu pliku *tweetshirt.js* dodamy element `<script>`, który będzie odpowiadał za wywołania API Twittera zgodnego z JSONP. Zamierzamy w ten sposób pobierać najnowsze statusy określonego użytkownika.

2. Zaimplementujemy funkcję zwrotną pobierającą tweety odsyłane przez Twittera. Nazwę tej funkcji dodamy do adresu URL wykorzystywanego przez element `<script>` z pierwszego punktu.

To jest kod HTML naszej aplikacji.

Tu znajduje się zawartość elementu head i formularza (nie zamieszczamy pełnego kodu, by ocalić kilka drzew).

Wywołanie JSONP. Służy ono do pobrania z Twittera danych JSON o wskazanym adresie; dane zostają przekazane funkcji zwrotnej (którą już za chwilę utworzymy).

To jest wywołanie API Twittera. Prosimy o oś czasu użytkownika, która dostarczy najnowsze statusy.

Tu wpisz dowolną nazwę użytkownika (możesz podać swoją).

```
<html>

...

<body>

   <form>

   ...

   </form>

   <script src="http://twitter.com/statuses/user_timeline/wickedsmartly.json?
callback=updateTweets">

   </script>

</body>

</html>
```

A to jest nazwa funkcji zwrotnej, do której zostaną przekazane dane JSON.

Wpisz to w jednym wierszu (tutaj się nie zmieściło).

Dużo się tu dzieje. Jeżeli coś nie jest dla Ciebie jasne, wróć na chwilę do poprzedniego rozdziału i przypomnij sobie tematy związane z JSONP.

Pobieranie tweetów

Większość pracy związanej z pobieraniem tweetów z Twittera już wykonaliśmy. Teraz musimy tyko dodać je do elementu `<select>` formularza umieszczonego na stronie. Powtórzmy jeszcze raz: kiedy zostaje wywołana funkcja zwrotna (w naszym przypadku jest to `updateTweets`), Twitter przekazuje jej odpowiedź zawierającą tweety zapisane w formacie JSON.

Odpowiedź z Twittera jest tablicą tweetów. Każdy tweet zawiera całą masę danych. My skorzystamy tylko z tekstu tweeta.

Na końcu pliku *tweetshirt.js* dodaj funkcję `updateTweets`. Oto jej kod:

To jest nasza funkcja...

...do której zostaje przekazana odpowiedź w postaci tablicy zawierającej tweety z osi czasu użytkownika.

Pobieramy referencję do elementu select znajdującego się w formularzu.

```
function updateTweets(tweets) {
    var tweetsSelection = document.getElementById("tweets");

    for (var i = 0; i < tweets.length; i++) {
        tweet = tweets[i];
        var option = document.createElement("option");
        option.text = tweet.text;
        option.value = tweet.text.replace("\"", "'");

        tweetsSelection.options.add(option);
    }

    tweetsSelection.selectedIndex = 0;
}
```

Dla każdego tweeta w tablicy...

...pobieramy tweet z tablicy...

...tworzymy nowy element option...

...do jego właściwości text przypisujemy tekst tweeta...

...a do jego właściwości value przypisujemy ten sam tekst, ale trochę przetworzony. Zamieniamy podwójne cudzysłowy na pojedyncze, czyli apostrofy (aby uniknąć problemów z formatowaniem w HTML-u).

Następnie dodajemy utworzony element option do pola wyboru formularza.

Gdy już przeprowadzimy te operacje dla wszystkich tweetów, element <select> zawiera pozycję option dla każdego tweeta.

Na koniec ustalamy, by pierwszy tweet był zaznaczony — do właściwości selectedIndex elementu <select> przypisujemy wartość 0 (odpowiadającą pierwszemu elementowi <option>).

Jazda próbna z tweetami

Wybierzmy się na krótką jazdę próbną. Sprawdź, czy zaktualizowałeś cały kod w plikach *tweetshirt.js* i *index.html*. Upewnij się też, że w adresie URL podanym w elemencie `<script>` wpisałeś nazwę użytkownika, który ostatnio coś tweetnął (tak żebyś zobaczył jakiekolwiek pozycje!). Przeładuj stronę i kliknij rozwijaną listę tweetów. My zobaczyliśmy to:

To jest rozwijana lista zawierająca PRAWDZIWE tweety. Super!

Koledzy, świetna sprawa! Mamy już kwadraty i kółka, a Kuba przed chwilą rozwiązał problem z Twitterem. Co dalej?

Tablet Łukasza

Kuba: Już prawie skończyliśmy. Wystarczy tylko wyświetlić kilka tekstów. Mamy dwa komunikaty: „Zobaczyłem ten tweet” i „i dostałem tylko tę nędzną koszulkę!” oraz tweet, który użytkownik wybrał z listy. Teraz musimy się dowiedzieć, jak je wyświetlić i jak zastosować do nich różne style.

Łukasz: Zakładam, że możemy umieścić tekst na płótnie, a potem zastosować do niego style z arkusza CSS. Co myślicie?

Przemek: Nie sądzę, żeby to miało zadziałać. Na płótnie się rysuje, więc nie wydaje mi się, żeby dało się na nim umieścić tekst, który można by obstylować. Chyba będziemy musieli „narysować” ten tekst...

Kuba: No cóż, tym razem odrobiłem lekcje i sprawdziłem API związane z tekstem.

Przemek: Świetnie, bo ja jeszcze nie miałem okazji. I jak to wygląda?

Kuba: Pamiętacie metodę `arc`? Do rysowania tekstu musimy użyć właśnie jej.

Łukasz: Chyba żartujesz! Przecież to mi zajmie cały weekend!

Kuba: Ha, ha, dałeś się nabrać! A tak naprawdę to jest metoda `fillText`, która rysuje na płótnie przekazany tekst w podanych współrzędnych x i y.

Przemek: Bułka z masłem. A jak zastosować różne style? Z tego, co pamiętam, na makiecie tekst tweeta był zapisany kursywą, a pozostałe napisy pogrubionym fontem.

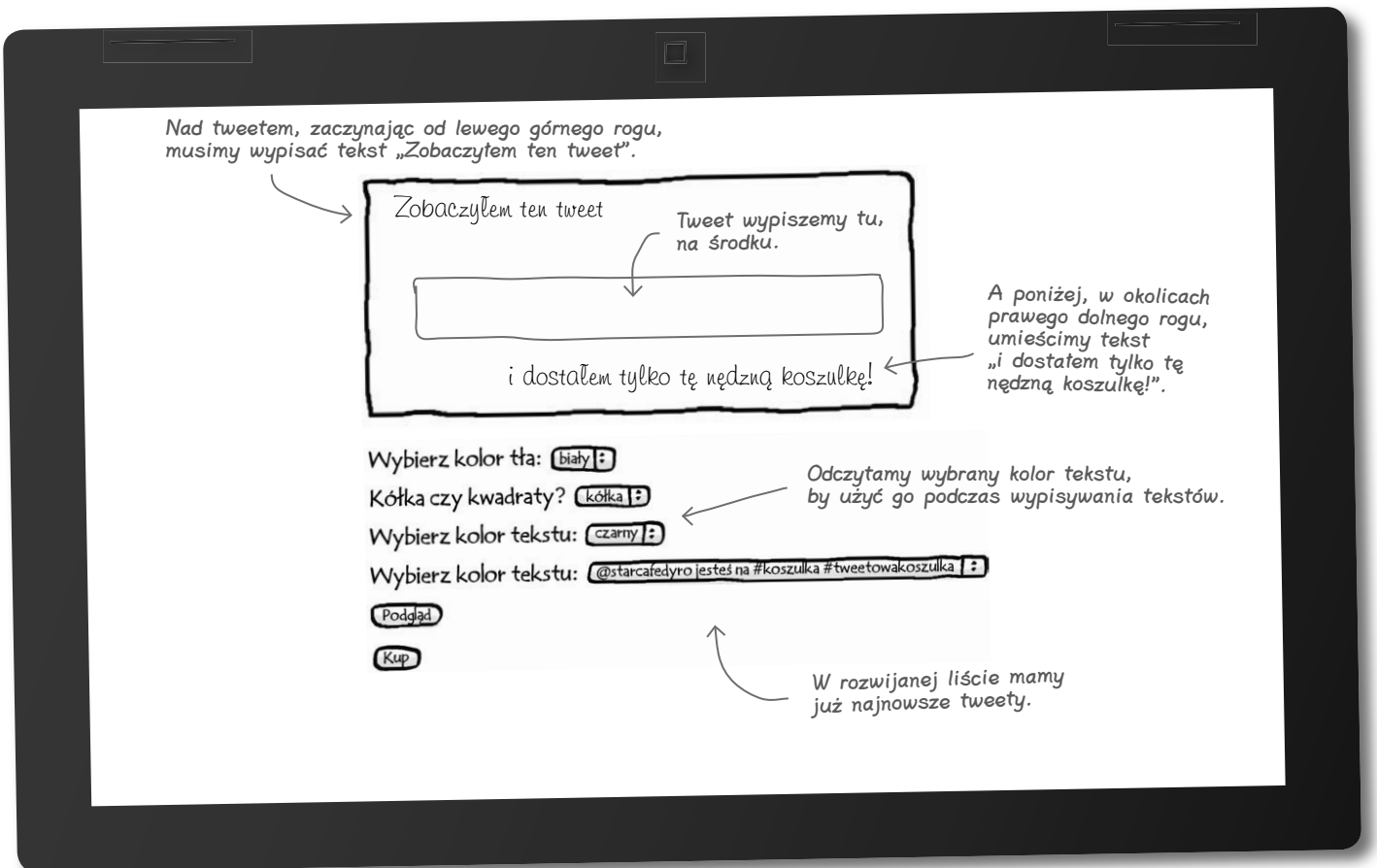
Kuba: Musimy się temu bliżej przyjrzeć, bo jest kilka różnych metod ustalania wyrównania, fontów i stylów wypełnienia, ale prawdę mówiąc, nie za bardzo wiem, jak z nich skorzystać.

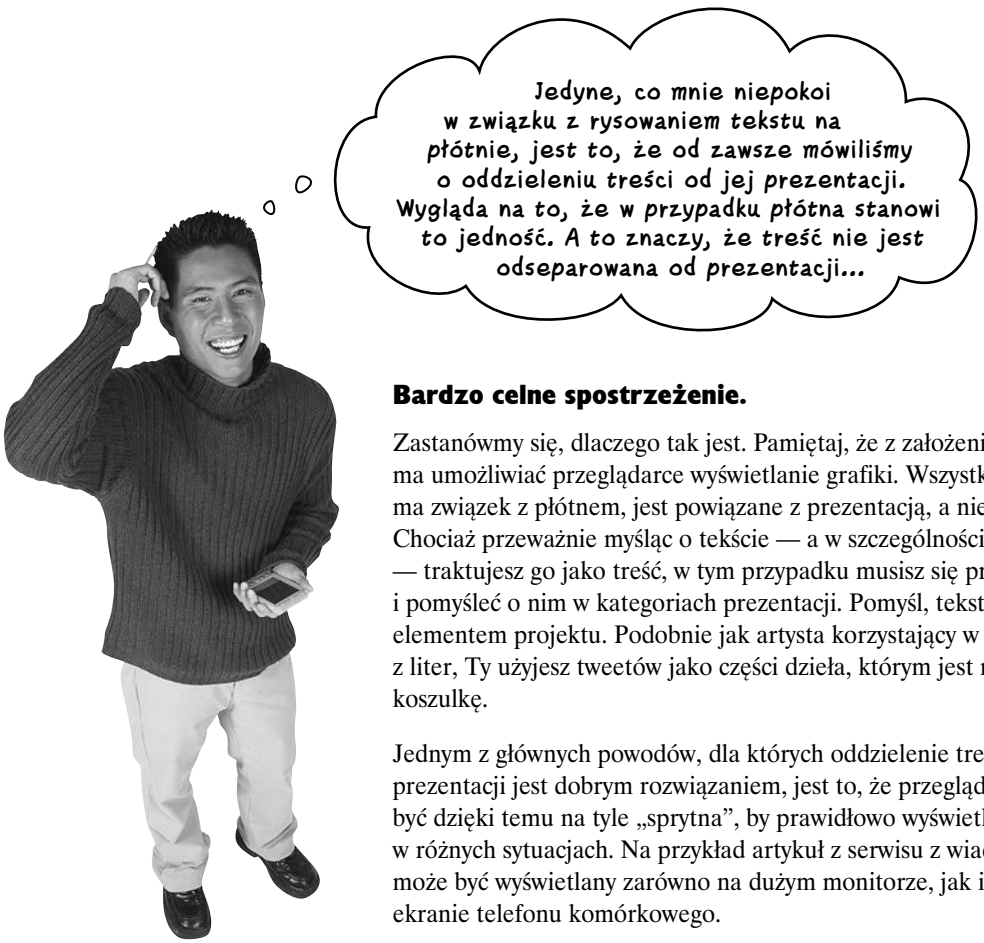
Łukasz: Tak sobie myślę, że może mógłbym pomóc... Ale bez CSS-a?

Kuba: Przykro mi, ale tak jak powiedział Przemek, to jest API służące do rysowania na płótnie, więc nie stosuje się tu ani HTML-a, ani CSS-a.

Przemek: No dobra, rzucimy okiem na to API i spróbujemy wyświetlić na płótnie tekst „Zobaczyłem ten tweet". Chodź, Łukasz, pomożesz nam, a to na pewno nie jest takie straszne, na jakie wygląda. Poza tym może nam się przydać twoja wiedza o fontach, stylach i takich tam...

Łukasz: W porządku, ale robię to tylko dla ciebie!





Bardzo celne spostrzeżenie.

Zastanówmy się, dlaczego tak jest. Pamiętaj, że z założenia płótno ma umożliwiać przeglądarce wyświetlanie grafiki. Wszystko, co ma związek z płótnem, jest powiązane z prezentacją, a nie treścią. Chociaż przeważnie myśląc o tekście — a w szczególności o tweetach — traktujesz go jako treść, w tym przypadku musisz się przestawić i pomyśleć o nim w kategoriach prezentacji. Pomyśl, tekst jest tu elementem projektu. Podobnie jak artysta korzystający w swoim dziele z liter, Ty użyjesz tweetów jako części dzieła, którym jest nadruk na koszulkę.

Jednym z głównych powodów, dla których oddzielenie treści od jej prezentacji jest dobrym rozwiązaniem, jest to, że przeglądarka może być dzięki temu na tyle „sprytna", by prawidłowo wyświetlać treści w różnych sytuacjach. Na przykład artykuł z serwisu z wiadomościami może być wyświetlany zarówno na dużym monitorze, jak i małym ekranie telefonu komórkowego.

W przypadku naszego projektu nadruku chcemy, by zawartość płótna przypominała raczej obrazek, który ma wyglądać tak samo, niezależnie od sposobu wyświetlania.

Do dzieła! Umieśćmy w końcu ten tekst na płótnie!

Magnesiki z kodem

Przyszedł czas na to, byś trochę poeksperymentował z tekstem na płótnie. Poniżej możesz zobaczyć zalążek kodu funkcji `drawText`, której będziemy używać do wyświetlania tekstu na podglądzie nadruku na koszulkę. Spróbuj go uzupełnić, tak by wypisywał na płótnie napisy „Zobaczyłem ten tweet" i „i dostałem tylko tę nędzną koszulkę!". Treścią tweeta zajmiemy się później. Zanim przejdziesz dalej, sprawdź, czy Twoje rozwiązanie jest zbliżone do naszego (odpowiedź znajdziesz pod koniec tego rozdziału).

```
function drawText(canvas, context) {

    var selectObj = document.getElementById("__________________");

    var index = selectObj.selectedIndex;

    var fgColor = selectObj[index].value;

    context._____________ = fgColor;

    context._________ = "bold 1em sans-serif";

    context._____________ = "left";

    context._________(_________________, 20, 40);


    // Pobierz wybrany tweet z listy rozwijanej

    // Wypisz tekst na płótnie


    context.font = "____________________";

    context._____________ = "_________";

    context._____________("i dostałem tylko tę nędzną koszulkę!",

                          ____________________, ____________________);

}
```

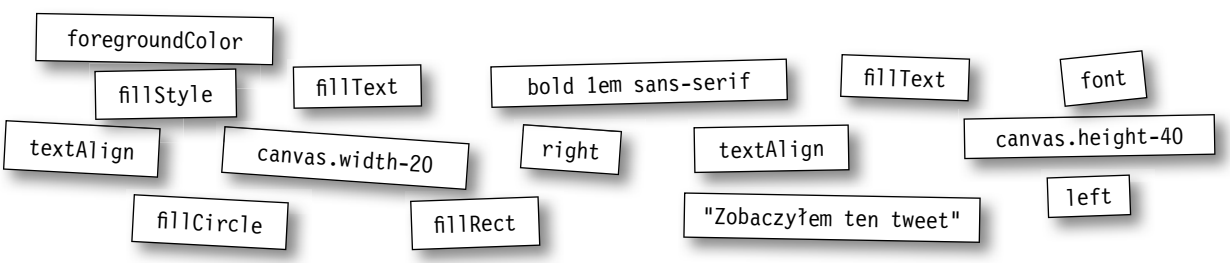
Podpowiedź: to są współrzędne x i y tekstu „Zobaczyłem ten tweet".

Na razie w miejscu, gdzie ma się znaleźć kod wyświetlający tweet, umieściliśmy komentarz.

Podpowiedź: co prawda treść tweeta chcemy wyświetlić pochylonym bezszeryfowym fontem, ale ten tekst ma być pogrubiony.

Podpowiedź: ten tekst chcemy umieścić w prawym dolnym rogu.

Chcemy, by tekst znalazł się 20 pikseli od prawej i 40 pikseli od dolnej krawędzi płótna, tak aby zrównoważyć pierwszą część zdania umieszczoną na górze z lewej strony.



Zbliżenie na tekst na płótnie

Spróbowałeś już wypisać jakiś tekst na płótnie, więc teraz należałoby się dokładniej przyjrzeć metodom i właściwościom związanym z tekstem, a dostępnym w API płótna. Podczas ostatniego ćwiczenia z pewnością zauważyłeś, że jest to dosyć niskopoziomowe API — musisz wskazać tekst, który ma zostać wypisany, jego położenie oraz font.

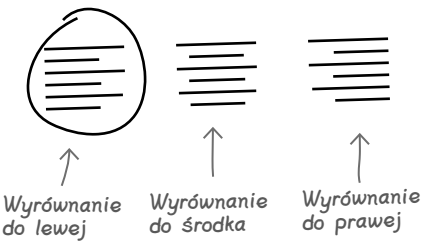
W tej sekcji **Zbliżenie** zajmiemy się szczegółowo wyrównywaniem tekstu, fontami, właściwościami linii bazowej oraz metodami wypełniania i obrysowywania znaków. Kiedy przewrócisz kartkę, będziesz już ekspertem w temacie tekstu na płótnie!

Wyrównanie tekstu

Właściwość `textAlign` określa miejsce, w którym jest zakotwiczony tekst. Domyślną wartością jest `start`.

```
context.textAlign = "left";
```

Właściwość może przyjąć jedną z wartości: `start`, `end`, `left`, `right` i `center`. Wartości `start` i `end` znaczą to samo co `left` i `right` w językach, w których pisze się od lewej strony do prawej, natomiast w językach o przeciwnym kierunku pisania (np. hebrajskim) znaczenia są zamienione.



Wypełnienie i obrys

Podobnie jak w przypadku prostokątów, tekst również możemy wypełnić lub obrysować. Dwóm metodom — `fillText` i `strokeText` — przekazujemy wyświetlany tekst, współrzędne x i y oraz opcjonalny parametr określający maksymalną szerokość tekstu. Jeżeli wypisywany tekst okaże się dłuższy niż podana wartość tego parametru, tekst jest skalowany.

```
context.fillText("Pies", 100, 100, 200);
context.strokeText("Kot", 100, 150, 200);
```

Jeżeli tekst okaże się szerszy niż 200 pikseli, jest automatycznie skalowany do podanej szerokości.

Font

Aby ustawić właściwości fontu, można skorzystać z tego samego formatu, jaki jest stosowany w arkuszach CSS, co jest bardzo wygodne. Jeżeli chcesz za jednym razem określić wszystkie właściwości, zachowaj taką ich kolejność: styl fontu, waga, rozmiar i rodzina.

```
context.font = "2em Lucida Grande";
context.fillText("Herbata", 100, 100);
context.font = "italic bold 1.5em Times, serif";
context.fillText("Kawa", 100, 150);
```

Herbata

Kawa

W specyfikacji można znaleźć zalecenie, by stosować wyłącznie czcionki wektorowe (czcionki bitmapowe mogą nie wyglądać najlepiej na płótnie).

Linia bazowa

Właściwość `textBaseline` ustawia w foncie punkty wyrównania, określając linię, na której są umieszczane znaki. Aby zobrazować wpływ tej właściwości na sposób umieszczenia znaków, wyświetl linię w tych samych współrzędnych co tekst.

```
context.beginPath();
context.moveTo(100, 100);
context.lineTo(250, 100);
context.stroke();
context.textBaseline = "middle";
context.fillText("Alfabet", 100, 100);
```

Alphabet ← alphabetic

Alphabet ← bottom

Alphabet ← middle

Alphabet ← top

Dozwolone wartości to: `top`, `hanging`, `middle`, `alphabetic`, `ideographic` i `bottom`. Domyślną wartością jest `alphabetic`. Poeksperymentuj z różnymi wartościami, aby się dowiedzieć, co masz do dyspozycji (więcej informacji możesz jak zawsze znaleźć w dokumentacji).

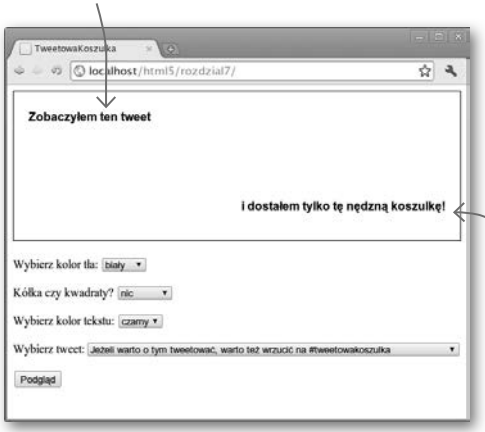
Ożywiamy funkcję drawText

Wiesz już znacznie więcej na temat API związanego z tekstem, więc możemy wrócić do kodu, który skompletowałeś w ćwiczeniu „Magnesiki z kodem”. Poniżej zamieściliśmy kompletny kod:

```
function drawText(canvas, context) {
    var selectObj = document.getElementById("foregroundColor");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;

    context.fillStyle = fgColor;
    context.font = "bold 1em sans-serif";
    context.textAlign = "left";
    context.fillText("Zobaczyłem ten tweet", 20, 40);

                                          Już za chwilę wstawimy tu
                                          kod wypisujący treść tweeta.

    context.font = "bold 1em sans-serif";
    context.textAlign = "right";
    context.fillText("i dostałem tylko tę nędzną koszulkę!",
             canvas.width-20, canvas.height-40);
}
```

Po wpisaniu powyższego kodu uaktualnij funkcję `updateHandler`, tak by wywoływała funkcję `drawText`, a następnie przetestuj aplikację w przeglądarce. Powinieneś zobaczyć coś takiego:

Oto nasz tekst. Znajduje się tam, gdzie powinien, i jest zapisany bezszeryfowym, pogrubionym fontem.



A tu mamy tekst wyrównany do prawej krawędzi.

Zaostrz ołówek

Pomyślmy, co trzeba jeszcze zrobić w funkcji `drawText`. Musisz odczytać wybrany tweet, ustawić szeryfowy, pochylony font, który jest trochę większy (`1.2em`) niż domyślny, wybrać wyrównanie do lewej strony i ustalić współrzędne x = 30 i y = 100. I tyle! Od tego momentu będziesz mógł podziwiać w pełni działającą aplikację.

Wpisz kod opisany powyżej i, zanim nie skończysz, pod żadnym pozorem nie przechodź na kolejną stronę (naprawdę!).

Kończymy funkcję drawText

Poniżej nasze rozwiązanie. Czy Twoje jest podobne? Jeżeli jeszcze tego nie zrobiłeś, wpisz teraz ten kod (albo własną wersję, jeżeli wolisz), a następnie przeładuj stronę *index.html*. Wyniki naszych testów możesz zobaczyć na kolejnej stronie.

```
function drawText(canvas, context) {
   var selectObj = document.getElementById("foregroundColor");
   var index = selectObj.selectedIndex;
   var fgColor = selectObj[index].value;

   context.fillStyle = fgColor;
   context.font = "bold 1em sans-serif";
   context.textAlign = "left";
   context.fillText("Zobaczyłem ten tweet", 20, 40);

   selectObj = document.getElementById("tweets");
   index = selectObj.selectedIndex;
   var tweet = selectObj[index].value;
   context.font = "italic 1.2em serif";
   context.fillText(tweet, 30, 100);

   context.font = "bold 1em sans-serif";
   context.textAlign = "right";
   context.fillText("i dostałem tylko tę nędzną koszulkę!",
            canvas.width-20, canvas.height-40);
}
```

Nie musimy ustawiać wyrównania treści tweeta do lewej, ponieważ właśnie takie zostało ustawione wcześniej.

Odczytujemy tweet wybrany z listy rozwijanej.

Ustawiamy szeryfowy, pochylony i trochę większy font...

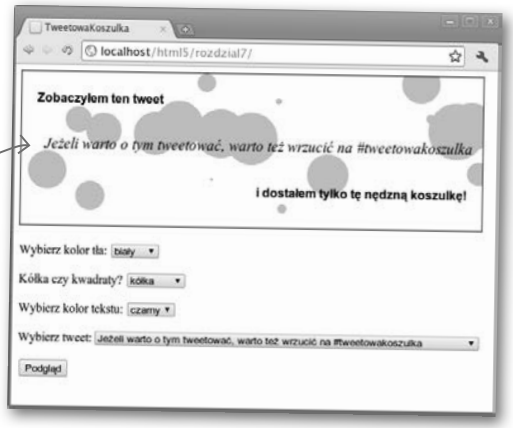
...i wypisujemy tekst we współrzędnych 30; 100.

Pospiesz się, Łukasz, wciśnij ten przycisk podglądu! Chcę w końcu zobaczyć naszą aplikację!

Szybka jazda próbna, a potem wydajemy ~~przejęcie~~ oficjalną wersję

Mamy nadzieję, że widzisz to samo co my! Piękne, prawda? Przetestuj sumiennie wszystkie funkcje aplikacji: modyfikuj kolory i kształty, a jeżeli chcesz, możesz też zmienić konto, z którego są pobierane tweety.

Czy nie sądzisz, że jesteśmy gotowi, by wypuścić naszą aplikację na rynek? Do dzieła!



Na podglądzie nadruku mamy już tweet. Nieźle!

1. Przede wszystkim potrzebujemy obrazka. Plik *twitterBird.png* umieszczamy w tym samym folderze co kod. Aby umieścić go na płótnie, musimy utworzyć javascriptowy obiekt obrazka. Robimy to tak:

```
var twitterBird = new Image();
twitterBird.src = "twitterBird.png";
```

Tworzymy nowy obiekt obrazka.

Ustawiamy jego właściwość src na plik z obrazkiem przedstawiającym ptaszka.

2. Kolejny etap jest dosyć oczywisty: musimy narysować obrazek na płótnie za pomocą pewnej metody kontekstu. Jakiej? Na pewno się domyśliłeś — `drawImage`.

```
context.drawImage(twitterBird, 20, 120, 70, 70);
```

Korzystamy z metody drawImage.

To jest nasz obiekt obrazka.

Określamy współrzędne x i y, szerokość oraz wysokość.

3. Jest jeszcze jedna sprawa związana z obrazkami, o której musisz wiedzieć: nie zawsze ładują się natychmiast. Z tego wniosek, że przed narysowaniem obrazka musisz sprawdzić, czy został pobrany. W jaki sposób możemy odczekać z wykonaniem pewnych operacji do czasu załadowania się jakiegoś zasobu? Wystarczy skorzystać z funkcji zwrotnej:

W tym miejscu mówimy: kiedy obrazek się załaduje, wykonaj tę funkcję.

```
twitterBird.onload = function() {
   context.drawImage(twitterBird, 20, 120, 70, 70);
};
```

Rysujemy obrazek na płótnie, korzystając z metody drawImage kontekstu.

Ćwiczenie

Spróbuj poskładać wszystkie elementy, o których mówiła Iza, i zaimplementować funkcję `drawBird`. Funkcji tej powinniśmy przekazywać obiekty `canvas` i `context`, a jej zadaniem jest narysowanie na płótnie ptaszka. Załóż, że obrazek znajduje się w pliku *twitterBird.png*, umieszczamy go we współrzędnych x = 20 i y = 120, a jego szerokość i wysokość wynoszą 70. Napisaliśmy już za Ciebie krótki fragment: deklarację funkcji i pierwszy wiersz kodu. Rozwiązanie tego ćwiczenia znajdziesz na końcu rozdziału.

```
function drawBird(canvas, context) {
    var twitterBird = new Image();

Tu wpisz kod.



}
```

Upewnij się, że w funkcji previewHandler dodałeś wywołanie funkcji drawBird.

Kolejna jazda próbna

Dokładnie sprawdź wpisany kod i znowu go przetestuj. Super! Teraz aplikacja jest naprawdę dopieszczona w najdrobniejszym szczególe.

Spróbuj kilku możliwości, zobacz, jak koszulka wygląda z kółkami, a jak z kwadratami. Jak pewnie zauważyłeś, użyliśmy pliku PNG z przezroczystym tłem, więc nic nie zasłania narysowanych kształtów.

Wyszło nam świetnie! Jesteś na najlepszej drodze do tworzenia naprawdę fajnych aplikacji. Ale pamiętaj, o czym mówiliśmy — liczymy, że zaimplementujesz jeszcze funkcjonalność związaną ze sklepem internetowym.

Nie istnieją głupie pytania

P: Wcześniej nie mieliśmy do czynienia z obiektem obrazu. Teraz użyliśmy go w celu umieszczenia obrazu na płótnie. O co tu chodzi? Dlaczego nie utworzyliśmy go po prostu metodą `document.createElement("img")`?

O: Dobre pytanie. Obie metody, o których wspomniałeś, tworzą obiekt obrazu. Zastosowanie konstruktora `Image` jest lepszym sposobem na tworzenie obrazów z poziomu JavaScriptu, ponieważ daje nam większą kontrolę nad procesem ładowania (możemy na przykład użyć funkcji zwrotnej wywoływanej w chwili pobrania obrazu).

Naszym zadaniem było utworzenie obrazu i umieszczenie go na płótnie dopiero wtedy, gdy zostanie w pełni załadowany, więc zastosowanie obiektu `Image` jest najlepszym rozwiązaniem.

P: Płótno jest fajne... ale praca z nim jest trudniejsza niż z HTML-em. Narysowanie czegoś bardziej skomplikowanego niż tylko proste kształty może być dosyć trudne.

O: Co do tego nie ma wątpliwości. Żeby oprogramować wyświetlanie w elemencie canvas, musisz napisać kod, który rozmieści poszczególne elementy w określonych miejscach płótna. Inaczej jest w przypadku, gdy za to zadanie odpowiada przeglądarka, która dba o szczegóły, takie jak choćby rozmieszczenie elementów na stronie.

Jednak element canvas daje Ci bardzo duże możliwości rysowania wszystkich typów grafiki (obecnie tylko 2D). Weź pod uwagę, że to dopiero początek rozwoju tego elementu, więc można się spodziewać, że w przyszłości rysowanie na płótnie będzie znacznie prostsze.

P: Zauważyłem, że w przypadku długich tweetów tekst jest przycinany na krawędzi płótna. Jak sobie z tym poradzić?

O: Jedną z metod jest sprawdzenie długości tekstu w znakach i, jeżeli jest większa niż pewna ustalona wartość, podzielenie tego tekstu na wiele wierszy i umieszczenie na płótnie każdego z osobna. Do kodu, który można pobrać ze strony wydawnictwa, dołączyliśmy funkcję `splitIntoLines`.

P: Zwróciłem też uwagę, że w niektórych tweetach pojawiają się encje HTML, takie jak `&quot;` czy `&amp;`. O co tu chodzi?

O: API Twittera, z którego korzystamy do pobrania danych JSON, zamienia niektóre znaki umieszczone w tweetach na encje HTML. To bardzo dobrze, ponieważ znaki specjalne, a nawet cudzysłowy mogłyby sporo namieszać i uniemożliwić prawidłowe odczytanie danych z JSON-a. Gdybyśmy wyświetlali tweety bezpośrednio w HTML-u, wszystkie encje zostałyby zamienione z powrotem na znaki, więc widziałbyś dokładnie to samo co wpisujący je użytkownik. Jednak, jak zauważyłeś, na płótnie nie wyglądają za dobrze. Niestety jak na razie w API elementu canvas nie ma funkcji, która dokonywałaby takiej konwersji, więc musisz się tym zająć sam.

P: Czy płótno pozwala na dodawanie ozdobników, takich jak na przykład cienie rzucane przez tekst albo kształty?

O: Tak! Masz możliwość ozdabiania grafiki na wiele sposobów, a jednym z nich są właśnie cienie. Jak się pewnie domyślasz, cienie tworzy się poprzez ustawienie pewnych właściwości kontekstu. Na przykład do ustawienia stopnia rozmycia cienia służy właściwość `context.shadowBlur`. Położenie cienia określa się właściwościami `context.shadowOffsetX` i `context.shadowOffsetY`, a kolor właściwością `context.shadowColor`.

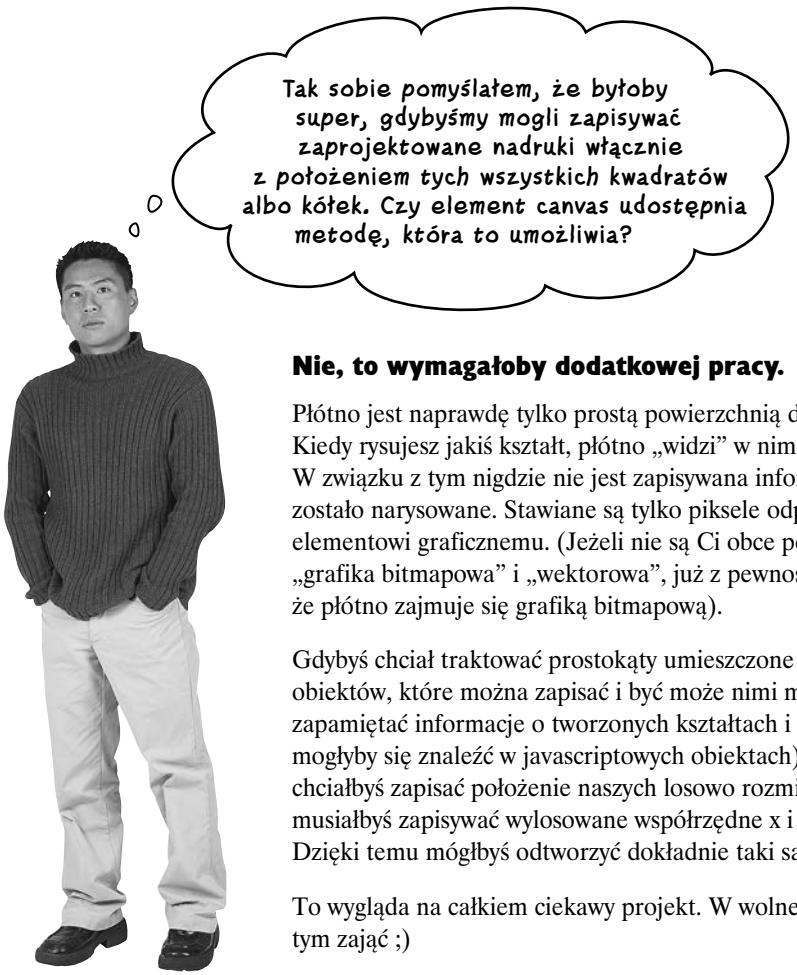
Innymi ciekawymi możliwościami oferowanymi przez element canvas są między innymi gradienty, obroty i zaokrąglone wierzchołki prostokątów.

P: Co jeszcze ciekawego mogę zrobić z płótnem?

O: Mnóstwo! W kolejnych rozdziałach pokażemy jeszcze kilka ciekawych możliwości, ale jeżeli chcesz poznać wszystkie, zajrzyj do dokumentacji API elementu canvas: *http://dev.w3.org/html5/2dcontext/*.

P: Czy element canvas działa prawidłowo na urządzeniach mobilnych? Czy może powinienem napisać osobną wersję kodu dla użytkowników tych urządzeń?

O: Jeżeli urządzenie mobilne jest wyposażone w nowoczesną przeglądarkę (a tak jest w przypadku urządzeń Android, iPhone i iPad), wszystko powinno działać bez problemów (rozmiar strony może być inny, ale funkcjonalność powinna być taka sama). Dzięki temu, że na płótnie operujesz bezpośrednio pikselami, jego zawartość wszędzie będzie wyglądała jednakowo (to znaczy wszędzie, gdzie jest obsługiwany element canvas). Na szczęście w nowoczesnych urządzeniach mobilnych, takich jak iPhone czy iPad, i tych opartych na systemie Android są zainstalowane przeglądarki posiadające większość funkcjonalności swoich stacjonarnych odpowiedników.



Nie, to wymagałoby dodatkowej pracy.

Płótno jest naprawdę tylko prostą powierzchnią do rysowania. Kiedy rysujesz jakiś kształt, płótno „widzi" w nim tylko piksele. W związku z tym nigdzie nie jest zapisywana informacja o tym, co zostało narysowane. Stawiane są tylko piksele odpowiadające danemu elementowi graficznemu. (Jeżeli nie są Ci obce pojęcia takie jak „grafika bitmapowa" i „wektorowa", już z pewnością zauważyłeś, że płótno zajmuje się grafiką bitmapową).

Gdybyś chciał traktować prostokąty umieszczone na płótnie jako zbiór obiektów, które można zapisać i być może nimi manipulować, musiałbyś zapamiętać informacje o tworzonych kształtach i ścieżkach (takie dane mogłyby się znaleźć w javascriptowych obiektach). Jeżeli na przykład chciałbyś zapisać położenie naszych losowo rozmieszczonych kółek, musiałbyś zapisywać wylosowane współrzędne x i y, promień i kolor. Dzięki temu mógłbyś odtworzyć dokładnie taki sam układ kółek.

To wygląda na całkiem ciekawy projekt. W wolnej chwili możesz się tym zająć ;)

Gratuluję, doskonale wam się to udało! W dodatku aplikacja działa na moim iPadzie, więc sprawdzi się u użytkowników, którzy są cały czas w biegu. Jestem naprawdę podekscytowana. Wydajemy przyjęcie, dołączcie do nas!

Założycielka serwisu TweetowaKoszulka cieszy się, że aplikacja działa tak samo na iPadzie i iPhonie. Cieszymy się jej szczęściem.

CELNE SPOSTRZEŻENIA

- Element canvas (czyli płótno) umieszcza się na stronie, by utworzyć miejsce, na którym można rysować.
- Płótno nie ma domyślnego stylu ani zawartości (więc po umieszczeniu go na stronie nic nie zobaczysz, chyba że coś narysujesz albo dodasz obramowanie za pomocą stylów).
- Na stronie można umieścić więcej niż jedno płótno. Oczywiście w takiej sytuacji każdemu z nich trzeba nadać inny identyfikator, by móc odwoływać się do nich z poziomu JavaScriptu.
- Szerokość i wysokość elementu canvas określaj za pomocą atrybutów `width` i `height`.
- Rysowanie na płótnie jest możliwe tylko i wyłącznie z poziomu JavaScriptu.
- Aby rysować na płótnie, musisz najpierw utworzyć kontekst. Obecnie dostępny jest tylko dwuwymiarowy kontekst, ale w przyszłości z pewnością pojawią się kolejne.
- Kontekst jest niezbędny, ponieważ udostępnia specyficzny interfejs (np. 2D albo, w przyszłości, 3D). Dostępne będą różne interfejsy umożliwiające rysowanie na płótnie.
- W celu odwołania się do elementu canvas korzystasz z właściwości i metod kontekstu.
- Za rysowanie prostokątów wypełnionych kolorem odpowiada metoda `context.fillRect`.
- Za rysowanie obrysu prostokąta odpowiada metoda `strokeRect`, a nie `fillRect`.
- Aby zmienić kolor wypełnienia i kreski (domyślnie czarny), należy skorzystać z właściwości `fillStyle` i `strokeStyle`.
- Kolory można określać w ten sam sposób co w arkuszach CSS (np. `"black"`, `"#000000"`, `"rgb(0, 0, 0)"`). Pamiętaj, że wartości przypisywane właściwości `fillStyle` muszą być umieszczone w cudzysłowach.
- Nie ma metody `fillCircle`. Aby umieścić na płótnie koło, musisz narysować łuk.
- W celu narysowania dowolnych kształtów lub łuków musisz najpierw utworzyć ścieżkę.
- Ścieżka jest niewidoczną linią lub kształtem definiującym linię lub kształt na płótnie. Ścieżki nie widać do czasu jej wypełnienia lub nadania obrysu.
- Aby narysować trójkąt, musisz utworzyć ścieżkę, rozpoczynając ją metodą `beginPath`, a później — za pomocą metod `moveTo` i `lineTo` — narysować tę ścieżkę. Metoda `closePath` zamyka ścieżkę, łącząc dwa wolne punkty.
- Aby narysować koło, utwórz łuk, w którym kąt początkowy wynosi 0, a końcowy 360 stopni.
- Podczas pracy z płótnem kąty trzeba wyrażać w radianach, więc wszelkie wartości kątów podane w stopniach musisz przeliczyć na radiany.
- 360 stopni = 2π radianów.
- Do umieszczania na płótnie tekstu służy metoda `fillText`.
- Rysowanie tekstu na płótnie wymaga podania położenia, stylu i innych właściwości ustawianych za pośrednictwem kontekstu.
- Po ustawieniu jakiejś właściwości kontekstu jest ona stosowana podczas rysowania aż do jej ponownej zmiany. Na przykład zmiana właściwości `fillStyle` ma wpływ na kolor wszystkich rysowanych kształtów i wypisywanego tekstu.
- Do umieszczania na płótnie obrazów służy metoda `drawImage`.
- Aby dodać obraz, najpierw trzeba utworzyć obiekt `Image` i odczekać na jego załadowanie.
- Rysowanie na płótnie przypomina pracę w programie graficznym operującym na grafice rastrowej (bitmapowej).

WEBOWICKI ŚLEDCZY

Z ostatniej chwili: <canvas> i <video> pokazują się razem!

Od nas dowiecie się pierwsi

Z ekskluzywnego wywiadu wiemy, że <canvas> i <video> łączy więcej niż tylko siedzenie na tej samej stronie. Tak, to prawda... Ujmę to w ten sposób: oni mieszali swoje zawartości!

Maksymilian Golonko
DZIENNIKARZ ŚLEDCZY

<Video> powiedział: „Tak, to prawda, weszliśmy w dosyć ścisłe relacje. Słuchaj, tak naprawdę jestem prostym kolesiem. Wiem, jak wyświetlać wideo, i robię to bardzo dobrze. Ale to w zasadzie wszystko, co umiem. Dzięki temu, co dał mi <canvas>, wszystko się zmieniło! Mogę ubrać się w dowolne kontrolki, mogę filtrować moje wideo, mogę nawet wyświetlać kilka klipów w tym samym czasie".

Poprosiliśmy <canvas> o komentarz. Może za <video> kryje się kobieta? <Canvas> odpowiedział: „No cóż, <video> dobrze wykonuje swoje zadania, no wiecie, dekodowanie wideo za pomocą kodeków, utrzymywanie stałej prędkości odtwarzania i tak dalej. To naprawdę ciężka praca. Ja nigdy bym się tego nie podjął. Ale dzięki mnie ma szansę uciec od normalności, tak, tego nudnego, monotonnego wyświetlania wideo. Całe mnóstwo nowych, wspaniałych możliwości łączenia wideo z różnymi rozwiązaniami internetowymi może w końcu stać się jego udziałem".

No cóż, kto by pomyślał... Biegnijcie do najbliższego sklepu — założę się, że można już dostać ciekawe rzeczy zrodzone ze związku <canvas> z <video>!

Dalszego ciągu historii tego dobrze zapowiadającego się związku możecie się spodziewać w kolejnym rozdziale, w którym wystawimy go na widok publiczny.

Sąsiadka, Janina Mientka, była zszokowana, kiedy poznała prawdę o tych dwóch elementach.

BĄDŹ przeglądarką Rozwiązanie

Skoro masz już interfejs, wykonaj ten kod JavaScript i zapisz wartości dla każdego elementu.

Załóż, że wartości parametrów dla koszulki pobierasz z elementów interfejsu.

```
var selectObj = document.getElementById("backgroundColor");
var index = selectObj.selectedIndex;
var bgColor = selectObj[index].value;    ..... white .....

var selectObj = document.getElementById("shape");
var index = selectObj.selectedIndex;
var shape = selectObj[index].value;      ..... circles .....

var selectObj = document.getElementById("foregroundColor");
var index = selectObj.selectedIndex;
var fgColor = selectObj[index].value;    ..... black .....
```

Zwróć uwagę, że w każdym przypadku pobieramy element select zawierający daną opcję, odczytujemy indeks wybranej pozycji za pomocą właściwości selectedIndex, a następnie pobieramy wartość.

Pamiętaj, że wartość opcji zapisana we właściwości value może się różnić od tekstu wyświetlanego na stronie. Właśnie tak jest w naszym przypadku: w kontrolkach są wyświetlane polskie odpowiedniki angielskich nazw.

Po wybraniu takich opcji uzyskaliśmy wartości wpisane w powyższym kodzie.

Jeżeli musisz sobie przypomnieć wartości, jakie przypisaliśmy poszczególnym opcjom, zajrzyj jeszcze raz do pliku HTML.

Wybierz kolor tła: biały

Kółka czy kwadraty? kółka

Wybierz kolor tekstu: czarny

Wybierz tweet:

Podgląd

Magnesiki z pseudokodem. Rozwiązanie

Użyj swoich tajemnych mocy programistycznych i poskładaj zamieszczony poniżej pseudokod, który ma ilustrować działanie funkcji `drawSquare`. Do tej funkcji są przekazywane element canvas i kontekst, a jej zadaniem jest narysowanie kwadratu o losowej wielkości w losowym miejscu płótna. Zanim przejdziesz dalej, porównaj swoje rozwiązanie z naszym, które znajdziesz na końcu tego rozdziału.

```
function drawSquare ( canvas context ){
```

Ten magnesik umieściliśmy za Ciebie.

```
    wyznacz losową szerokość kwadratu

    wyznacz losową wartość współrzędnej x,
    tak by kwadrat znalazł się wewnątrz płótna

    wyznacz losową wartość współrzędnej y,
    tak by kwadrat znalazł się wewnątrz płótna

    ustaw właściwość fillStyle na "lightblue"

    narysuj kwadrat o szerokości
    w we współrzędnych x i y

}
```

Magnesiki umieść w tym miejscu!

Zaostrz ołówek
Rozwiązanie

Aby po każdym kliknięciu przycisku podglądu widzieć tylko nowe kwadraty, musimy za każdym razem zaczynać od wypełnienia płótna kolorem tła wybranym przez użytkownika za pomocą pola „backgroundColor". Zaczniemy od zaimplementowania funkcji wypełniającej płótno wybranym kolorem. W puste pola w poniższym kodzie wstaw brakujące fragmenty. Oto nasze rozwiązanie.

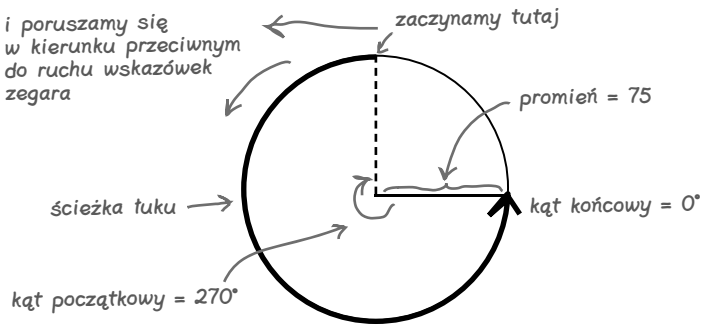
```
function fillBackgroundColor(canvas, context) {
    var selectObj = document.getElementById(" backgroundColor ");
    var index = selectObj.selectedIndex;
    var bgColor = selectObj.options[index].value;
    context.fillStyle = bgColor ;
    context.fillRect(0, 0, canvas.width , canvas.height );
}
```

Wyczyszczenie tła polega na narysowaniu prostokąta o takich samych wymiarach jak płótno, wypełnionego wybranym kolorem.

BĄDŹ przeglądarką. Rozwiązanie

Zinterpretuj poniższe wywołanie metody arc i zaznacz na rysunku wszystkie wartości, włącznie ze ścieżką, która powstaje w wyniku tego wywołania.

```
context.arc(100, 100, 75, degreesToRadians(270), 0, true);
```



Masz już ścieżkę! Co dalej?

Rozwiązanie ćwiczenia

Dzięki ścieżce będziesz mógł narysować linie i wypełnić powstały kształt kolorem! Utwórz prosty dokument HTML z elementem canvas i wpisz cały kod, który do tej pory napisaliśmy. Potem koniecznie go przetestuj!

```
context.beginPath();
context.moveTo(100, 150);
context.lineTo(250, 75);
context.lineTo(125, 30);
context.closePath();
```

To jest kod, który do tej pory napisaliśmy.

```
context.lineWidth = 5;
context.stroke();
context.fillStyle = "red";
context.fill();
```

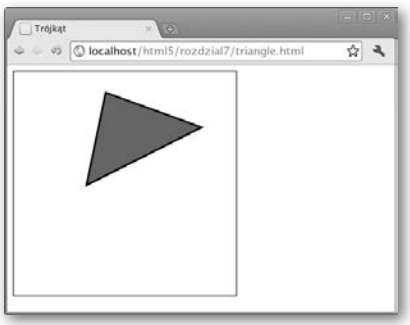
Ustawiamy szerokość linii nakładanej na ścieżkę.

Obrysowujemy ścieżkę wybraną linią.

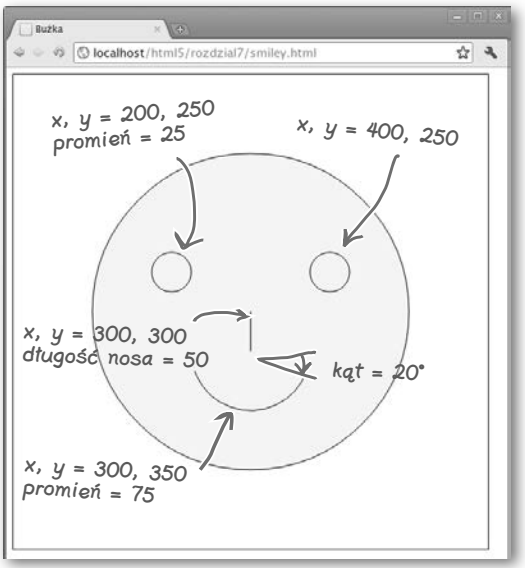
Ustawiamy czerwony kolor wypełnienia trójkąta.

Wypełniamy trójkąt wybranym kolorem.

Po załadowaniu strony zobaczysz coś takiego (do strony dodaliśmy element canvas o wymiarach 300×300).



Przerwa. Rozwiązanie



Czas na przećwiczenie dopiero co zdobytych umiejętności w posługiwaniu się łukiem i ścieżką. Wypełnij luki w kodzie, by na płótnie powstała uśmiechnięta buźka. Daliśmy Ci kilka podpowiedzi co do lokalizacji i wielkości poszczególnych elementów: oczu, nosa i ust.

Oto nasze rozwiązanie:

```
function drawSmileyFace() {
  var canvas = document.getElementById("smiley");
  var context = canvas.getContext("2d");

  context.beginPath();
  context.arc(300, 300, 200, 0, degreesToRadians(360), true);
  context.fillStyle = "#ffffcc";
  context.fill();
  context.stroke();

  context.beginPath();
  context.arc(200, 250, 25, 0, degreesToRadians(360), true);
  context.stroke();

  context.beginPath();
  context.arc(400, 250, 25, 0, degreesToRadians(360), true);
  context.stroke();

  context.beginPath();
  context.moveTo(300, 300);
  context.lineTo(300, 350);
  context.stroke();

  context.beginPath();
  context.arc(300, 350, 75, degreesToRadians(20), degreesToRadians(160), false);
  context.stroke();
}
```

Owal twarzy. To kółko narysowaliśmy za Ciebie (i w dodatku wypełniliśmy je na żółto).

To jest lewe oko. Współrzędne środka koła to x = 200 i y = 250, jego promień jest równy 25, kąt początkowy wynosi 0, a końcowy Math.PI * 2 radiany (360 stopni). Wywołaliśmy metodę stroke, więc koło jest obrysowane, ale nie wypełnione.

To jest prawe oko. Wszystko jest tak jak w lewym oku, z wyjątkiem współrzędnej x = 400. Ostatni argument ustawiliśmy na true, więc wybraliśmy kierunek przeciwny do ruchu wskazówek zegara (co nie ma znaczenia w przypadku pełnych kół).

To jest nos. Wywołaliśmy metodę moveTo(300, 300), aby przesunąć ołówek do punktu o współrzędnych x = 300 i y = 300, gdzie rozpoczynamy rysowanie linii. Następnie wywołujemy metodę lineTo(300, 350), rysując linię o długości 50. Na końcu wywołujemy metodę stroke, aby nadać obrys.

Aby uśmiech był bardziej realistyczny, zaczynamy i kończymy rysowanie łuku 20 stopni pod osią X. Z tego wniosek, że kąt początkowy wynosi 20°, a końcowy 160°.

Wybraliśmy kierunek zgodny z ruchem wskazówek zegara (wartość false), ponieważ zależy nam na uśmiechu. Pamiętaj, że punkt początkowy znajduje się z prawej strony środka łuku.

Magnesiki z kodem. Rozwiązanie

Przyszedł czas na to, byś trochę poeksperymentował z tekstem na płótnie. Poniżej możesz zobaczyć zalążek kodu funkcji `drawText`, której będziemy używać do wyświetlania tekstu na podglądzie nadruku na koszulkę. Spróbuj go uzupełnić, tak by wypisywał na płótnie napisy „Zobaczyłem ten tweet" i „i dostałem tylko tę nędzną koszulkę!". Treścią tweeta zajmiemy się później. Zanim przejdziesz dalej, sprawdź, czy Twoje rozwiązanie jest zbliżone do naszego (odpowiedź znajdziesz pod koniec tego rozdziału).

```
function drawText(canvas, context) {

    var selectObj = document.getElementById("foregroundColor");

    var index = selectObj.selectedIndex;

    var fgColor = selectObj[index].value;

    context.fillStyle = fgColor;

    context.font = "bold 1em sans-serif";

    context.textAlign = "left";

    context.fillText("Zobaczyłem ten tweet", 20, 40);

    // Pobierz wybrany tweet z listy rozwijanej

    // Wypisz tekst na płótnie


    context.font = "bold 1em sans-serif";

    context.textAlign = "right";

    context.fillText("i dostałem tylko tę nędzną koszulkę!",
                     canvas.width-20, canvas.height-40);
}
```

Podpowiedź: to są współrzędne x i y tekstu „Zobaczyłem ten tweet".

Na razie w miejscu, gdzie ma się znaleźć kod wyświetlający tweet, umieściliśmy komentarz.

Podpowiedź: co prawda treść tweeta chcemy wyświetlić pochylonym bezszeryfowym fontem, ale ten tekst ma być pogrubiony.

Podpowiedź: ten tekst chcemy umieścić w prawym dolnym rogu.

Chcemy, by tekst znalazł się 20 pikseli od prawej i 40 pikseli od dolnej krawędzi płótna, tak aby zrównoważyć pierwszą część zdania umieszczoną na górze z lewej strony.

Te magnesiki zostały.

`left`

`fillCircle`

`fillRect`

Rozwiązanie ćwiczenia

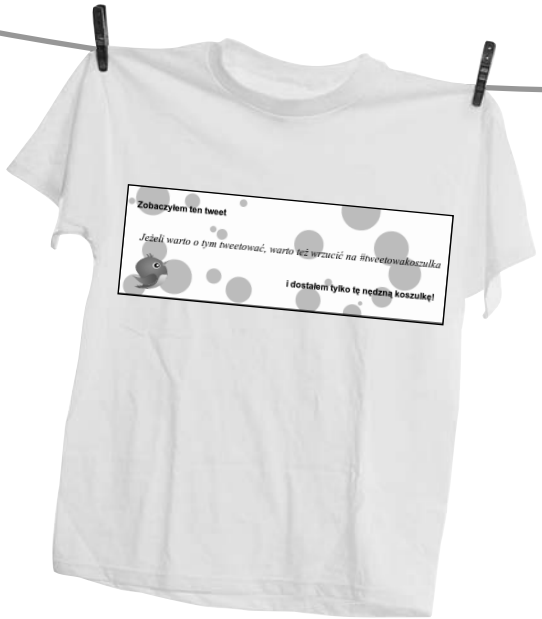
Spróbuj poskładać wszystkie elementy, o których mówiła Iza, i zaimplementować funkcję `drawBird`. Funkcji tej powinniśmy przekazywać obiekty `canvas` i `context`, a jej zadaniem jest narysowanie na płótnie ptaszka. Załóż, że obrazek znajduje się w pliku *twitterBird.png*, umieszczamy go we współrzędnych x = 20 i y = 120, a jego szerokość i wysokość wynoszą 70. Za Ciebie napisaliśmy już krótki fragment: deklarację funkcji i pierwszy wiersz kodu. Oto nasze rozwiązanie.

Tu wpisz kod.

```
function drawBird(canvas, context) {
  var twitterBird = new Image();
  twitterBird.src = "twitterBird.png";
  twitterBird.onload = function() {
    context.drawImage(twitterBird, 20, 120, 70, 70);
  };
}
```

Upewnij się, że w funkcji previewHandler dodałeś wywołanie funkcji drawBird.

Na koniec niespodzianka



A zatem przygotowałeś perfekcyjną aplikację. I co teraz? No cóż, jeżeli naprawdę chciałbyś koszulkę z zaprojektowanym nadrukiem, możesz to zrobić! Jak? Poniżej znajdziesz ekstradodatek, który możesz dopisać do swojego kodu. Dzięki niemu przygotowany przez siebie projekt będziesz mógł zamienić na obrazek gotowy do przesłania do firmy, która zajmuje się wykonywaniem nadruków na koszulkach (bez problemu znajdziesz taką firmę w internecie).

W jaki sposób można to osiągnąć? To proste! Możemy skorzystać z metody `toDataURL` obiektu `canvas`. Spójrz na to:

Definiujemy nową funkcję, makeImage, która dostarczy tę funkcjonalność.

```
function makeImage() {
    var canvas = document.getElementById("tshirtCanvas");
    canvas.onclick = function () {
        window.location = canvas.toDataURL("image/png");
    };
}
```

Pobieramy obiekt płótna...

...i dodajemy funkcję obsługi zdarzenia, która — po kliknięciu elementu canvas — tworzy obraz.

Ustawiamy lokalizację okna przeglądarki, tak by wskazywała wygenerowany obraz. Dzięki temu w oknie przeglądarki zostanie wyświetlony tylko ten obraz.

Prosimy płótno o utworzenie pliku PNG na podstawie zawartości elementu canvas.

Zapamiętaj, że PNG jest jedynym formatem, który musi być obsługiwany przez przeglądarki. Z tego względu polecamy korzystanie właśnie z niego.

Teraz wystarczy, byś dodał wywołanie funkcji `makeImage` w funkcji `onload` obiektu `window`. Dzięki temu po kliknięciu płótna zostanie utworzony obraz zawierający projekt nadruku. Sprawdź, czy to działa, i koniecznie daj nam znać, kiedy zrobisz sobie koszulkę!

```
window.onload = function() {
    var button = document.getElementById("previewButton");
    button.onclick = previewHandler;
    makeImage();
}
```

Wywołaj funkcję makeImage, aby do obiektu płótna dodać funkcję obsługi zdarzenia kliknięcia. I to wszystko!

Niektóre przeglądarki mogą nie zezwolić na pobranie obrazu z płótna, jeżeli kod jest uruchamiany z adresu file://.

Jeżeli chcesz, by to działało we wszystkich przeglądarkach, uruchom stronę z adresu localhost albo z zewnętrznego serwera.

Skorowidz

A

abbr, element, 559
addEventListener, metoda, 385
anonimowe, funkcje, 154
API, 47, 63
aplikacja internetowa
  a strona internetowa, 60
  działająca offline, 64, 550
appendChild, 128, 136
Application Programming Interface, *Patrz* API
arc, metoda, 333, 334, 335, 336
  kąty, 335
argument funkcji, 146
article, element, 558
aside, element, 559
audio, znacznik, 545

B

beginPath, metoda, 331
bezpieczeństwo, zasady, 266, 267
blok wyłapujący, 80

C

cache manifest, *Patrz* plik manifestu
camel case, *Patrz* notacja wielbłądzia
canPlayType, metoda, 381, 386, 387, 389, 426
canvas, 64, 301, 305, 355, 358, 552
  a wideo, 406
  arc, metoda, 333, 334, 335, 336
  beginPath, metoda, 331
  cienie, 355
  closePath, metoda, 332
  definiowanie ramki, 308
  drawImage, metoda, 358
  fillRect, metoda, 310, 312, 328
  fillStyle, właściwość, 313, 328, 358
  fillText, metoda, 345
  font, właściwość, 349
  getContext, metoda, 310, 312
  kąty, 335
  kolory, 358
  koło, 358
  kontekst 2d, 313
  kontekst 3d, 313
  liczba elementów na stronie, 309
  lineTo, metoda, 331
  łuki, rysowanie, 334, 335, 336
  moveTo, metoda, 331
  obsługa przez przeglądarki, 313, 314, 315
  restore, metoda, 552
  rotate, metoda, 552, 553
  rysowanie, 310, 311, 312, 313
  save, metoda, 552
  strokeRect, metoda, 313
  strokeText, metoda, 348
  ścieżki, rysowanie, 331, 332
  textAlign, właściwość, 348
  textBaseline, właściwość, 349
  translate, metoda, 552, 553
  trójkąt, rysowanie, 358
  umieszczanie na stronie, 306, 307
  urządzenia mobilne, 355
  wysokość i szerokość, 309
Chrome
  canvas, 389
  manipulowanie składnicą danych, 450
  wątki robocze, 496
  wideo, 389
ciasteczka, 430, 431, 442, 443
cienie, canvas, 355
clearInterval, metoda, 293
closePath, metoda, 332
CSS3, 64, 560
czas, odmierzanie, 287

D

dane, składowanie, 430
  ciasteczka, 430, 431, 442, 443
  localStorage, 433, 436, 437, 438, 439, 440, 441, 442, 443
Date, obiekt, 458, 480
  getTime, metoda, 480
delete, słowo kluczowe, 161
doctype, definicja, 35, 36, 41
Document Object Model, *Patrz* DOM
document, obiekt, 183, 186
document.createElement, 127, 136
document.getElementById, 89, 90, 105, 183, 186
document.querySelector, 554
document.write, 60
DOM, 46, 63, 66, 84, 85, 96
  dodawanie elementu, 96, 127, 128
  omówienie, 86
  pobieranie elementu, 89, 90, 96
  ustawianie atrybutów, 96
  usuwanie elementu, 96
  wstawianie elementu <script>, 290, 291
drawImage, metoda, 358
dynamiczne typowanie, 69
dystans, obliczanie, 204
dźwięk, 64
  kodowanie, 545
  sprawdzenie możliwości odtwarzania, 545

E

enumeracja, 159

F

fillRect, metoda, 310, 312, 328
fillStyle, właściwość, 328, 358
fillText, metoda, 345

Flash, 52, 304
footer, element, 558
for, pętla, 77, 105
formularze, 64
  selectedIndex, właściwość, 323
  tworzenie, 115, 321
  value, atrybut, 123
function, słowo kluczowe, 186
funkcje, 145
  a metody, 177
  anonimowe, 154
  argument, 146
  jako wartości, 154, 155
  miejsce deklaracji, 153
  nazewnictwo, 147
  parametr, 146, 147
  przekazywanie obiektów, 162, 163
  przypisanie do zmiennej, 154
  składnia, 147
  tworzenie, 141, 143, 186

G

Geolocation API, 190, 214, 231
  adres IP, 192
  altitude, właściwość, 221
  altitudeAccuracy, właściwość, 221
  clearWatch, metoda, 214
  definiowanie opcji, 225
  diagnozowanie błędów, 201, 202
  dokładność, 215, 222
  dystans, obliczanie, 204
  enableHighAccuracy, właściwość, 231
  getCurrentPosition, metoda, 198, 199, 214, 222, 231
  GPS, 192, 231
  heading, właściwość, 221
  mapy Google'a, 206, 207, 208, 210, 211, 212
  maximumAge, opcja, 223
  navigator.geolocation, 198, 231
  opcje, 234
  pierwszy program, 196, 197, 198, 199, 200
  prywatność, 190
  speed, właściwość, 221
  telefony komórkowe, 193
  timeout, opcja, 223
  watchPosition, metoda, 214, 216, 221, 222, 231
  WiFi, 193
  współrzędne geograficzne, 191
geolokalizacja, 64, 190, 213
getContext, metoda, 310, 312
getCurrentPosition, metoda, 198, 199, 214, 222, 231
getElementsByTagName, metoda, 292
getFormatExtension, metoda, 388
getItem, metoda, 437, 480
getTime, metoda, 458, 480
globalne, zmienne, 149, 150, 151, 152, 153
gniazda, 551
Google, mapy, 206, 212
  pinezki, 210, 211
  umieszczanie na stronie, 207
  wyświetlanie, 208
GPS, 192, 231

H

H.264, format, 370, 375
header, element, 558
hosting, 254
HTML, 33
  doctype, definicja, 35
  zamiana na HTML5, 34, 36, 37
HTML5, 34, 38, 43, 44, 45, 61, 63
  doctype, definicja, 36, 41
  elementy semantyczne, 558, 559
  link, znacznik, 37, 41
  ładowanie strony, 46, 47
  meta, znacznik, 36, 41, 63
  rodzina technologii, 46, 64
  script, znacznik, 37, 41
  wsparcie przeglądarek, 50, 51
  zatwierdzenie standardu, 52

I

importScripts, funkcja, 507, 537, 538
IndexedDB, 555
innerHtml, właściwość, 184
instrukcje, tworzenie, 67

J

JavaScript, 53, 54, 55, 56, 57, 58, 63, 491
  addEventListener, metoda, 385
  appendChild, 128, 136
  blok wyłapujący, 80
  clearInterval, metoda, 293
  czas wykonywania, 488, 489, 491
  Date, obiekt, 458, 480
  delete, słowo kluczowe, 161
  document, obiekt, 183, 186
  document.createElement, 127, 136
  document.getElementById, 89, 90, 105, 183, 186
  document.write, 60
  dodawanie do strony, 83
  dynamiczne typowanie, 69
  enumeracja, 159
  for, 77, 105
  funkcje, 145
    a metody, 177
    anonimowe, 154
    argument, 146
    jako wartości, 154, 155
    miejsce deklaracji, 153
    parametr, 146, 147
    przekazywanie obiektów, 162, 163
    przypisanie do zmiennej, 154
    składnia, 147
    tworzenie, 141, 143, 186
  getElementsByTagName, metoda, 292
  innerHtml, właściwość, 184
  instrukcje, tworzenie, 67
  jednowątkowość, 488, 489, 538
  lastreporttime, parametr, 297
  Math, biblioteka, 103, 105
  mechanizm działania, 66

metody, 168
a funkcje, 177
null, wartość, 105
obiekty, 157, 186
konstruktor, 172, 173, 174, 177
metody, 168
operacje, 159, 160, 161
przekazywanie do funkcji, 162, 163
właściwości, 158
obsługa zdarzeń, 117, 136
odmierzanie czasu, 287
onclick, 119
pętle, 76, 105
podejmowanie decyzji, 67, 79, 80, 105
replaceChild, metoda, 292, 299
setInterval, metoda, 287, 293, 299
setTimeout, metoda, 415, 416
słowa zarezerwowane, 71
tablice, 97, 98, 103, 105
this, słowo kluczowe, 171, 175, 177
typy, 69, 105
undefined, wartość, 105
uruchamianie po załadowaniu strony, 94
wartości logiczne, 105
wątki robocze, 487
while, 67, 76, 77, 105
window, obiekt, 181, 182, 184, 186
window.onload, 105, 182, 184, 185
współpraca ze stroną, 84
wyrażenia, 73
XMLHttpRequest, obiekt, 242, 243, 247, 261, 262, 263, 282, 283, 299
zmienne, 69, 149, 150, 151, 152, 153, 186
deklarowanie, 68, 105
nazywanie, 70, 71, 72
przypisanie funkcji, 154, 155
zmiana wartości, 69
JavaScript Object Notation, *Patrz* JSON
jQuery, 546, 547
JSON, 247, 248, 249
Json with Padding, *Patrz* JSONP
JSON.parse, 248, 261
JSON.stringify, 248
JSONP, 269, 270, 274, 275, 282, 283, 299
bezpieczeństwo, 281

K

kąty, 335
zamiana stopni na radiany, 337
key, metoda, 440, 441, 480
klucze
generowanie, 458
nazewnictwo, 461
kod odpowiedzi „200”, 261
kodek, 376
koło, rysowanie, 358
konstruktor, 172, 173, 174, 177
kontener, 376
kropka, operator, 160

L

lastreporttime, parametr, 297
lineTo, metoda, 331
link, znacznik, 37, 41
Linux, serwer WWW, 253
load, metoda, 381
localStorage, 136, 433, 434, 435, 437, 438, 440, 442, 443, 480
getItem, metoda, 437, 480
jako tablica asocjacyjna, 440
key, metoda, 440, 441, 480
kolejność elementów, 441
length, właściwość, 440, 441, 480
nazewnictwo elementów, 461
przechowywanie obiektu, 461
przechowywanie tablicy, 461, 482
removeItem, metoda, 462, 463
setItem, metoda, 437, 438, 480
zapis liczb, 439
zastosowanie, 478, 479
lokalne składowanie danych, 64, 433, 436, 438
lokalne, zmienne, 149, 150, 151, 152, 153

Ł

łuki, rysowanie, 334, 335, 336

M

Mac, serwer WWW, 253
Mandelbrot, Benoit, 509
manifestu, plik, 550
mapy Google’a, 206, 212
pinezki, 210, 211
umieszczanie na stronie, 207
wyświetlanie, 208
mark, element, 559
Math, biblioteka, 103, 105
Math.floor, 103, 105
Math.random, 103, 105
meta, znacznik, 36, 41, 63
metody, 168
a funkcje, 177
Modernizr, 544
moveTo, metoda, 331
MP4, 375

N

nav, element, 559
navigator.geolocation, obiekt, 198, 231
notacja wielbłądzia, 72
null, wartość, 105

O

obiekty, 157, 158, 186
konstruktor, 172, 173, 174, 177
metody, 168
operacje, 159, 160, 161
przechowywanie w localStorage, 461
przekazywanie do funkcji, 162, 163
właściwości, 158
obsługa błędów, wątki robocze, 536
obsługa zdarzeń, 117, 136
kliknięcie przycisku, 117, 118, 119, 136
odległość, obliczanie, 204
Ogg, 375
onclick, 119

P

palindrom, 81
pamięć podręczna, 294
parametr funkcji, 146
pause, metoda, 381
pętle
  for, 77, 105
  while, 67, 76, 77, 105
play, metoda, 381
plik manifestu, 550
postMessage, metoda, 498, 538
progress, 559
przeciągnij i upuść, technika, 555
przeglądarki
  dostępność odtwarzania dźwięku, 545
  manipulowanie składnicą danych, 450
  obsługa elementu canvas, 313, 314, 315
  obsługa elementu wideo, 370, 375
  obsługa Web Workers, 496
  pamięć podręczna, 294
  pobieranie danych z serwera, 241
  składowanie danych, 430, 442
  wielkość składnicy danych, 474, 484
  wsparcie HTML5, 50, 51
  zasady bezpieczeństwa, 266, 267
przycisk, obsługa kliknięć, 116, 117, 118, 119, 136

Q

QUOTA_EXCEEDED_ERR, wyjątek, 474, 483, 484

R

radiany, 337
removeItem, metoda, 462, 463
replaceChild, metoda, 292, 299

S

Safari, wideo, 389
Scalable Vector Graphics, *Patrz* SVG
script, znacznik, 37, 41
section, element, 558
selectedIndex, właściwość, 323
semantyczne, elementy, 558, 559
  abbr, 559
  article, 558
  aside, 559
  footer, 558
  header, 558
  mark, 559
  nav, 559
  progress, 559
  section, 558
  time, 559
serwer WWW
  hosting, 254
  Linux, 253
  Mac, 253
  Windows, 253
sessionStorage, 476, 480
setInterval, metoda, 287, 293, 299
setItem, metoda, 437, 438, 480
setTimeout, metoda, 415, 416
składnica danych
  manipulowanie za pomocą przeglądarek, 450
  wielkość, 474
słowa zarezerwowane, 71
stopnie, zamiana na radiany, 337
strokeRect, metoda, 313
strokeText, metoda, 348
strona internetowa
  a aplikacja, 60
  JavaScript, dodawanie, 83
  ładowanie, 46, 47
  współpraca z JavaScriptem, 84
strumieniowanie, 421, 422
SVG, 549

Ś

ścieżki, rysowanie, 331, 332, 333

T

tablice, 97, 105
  dodawanie elementu, 98
  pobieranie elementu, 98
  przechowywanie w localStorage, 461
  rozmiar, 98
  usuwanie elementu, 103
terminate, metoda, 538
this, słowo kluczowe, 171, 175, 177
time, element, 559
timer
  tworzenie, 287
  zatrzymywanie, 293
trójkąt, rysowanie, 358
tweety, pobieranie, 342, 343
typy, 69, 105

U

undefined, wartość, 105
urządzenia mobilne, canvas, 355
usługi sieciowe
  funkcja zwrotna, 276
  zgłaszanie żądania, 241, 242, 243
UTF-8, 41

V

value, atrybut, 123
video, *Patrz* wideo

W

wartości logiczne, 105
watchPosition, metoda, 214, 216, 221, 222, 231
wątki robocze, 54, 64, 487, 490, 505, 538
  importScripts, funkcja, 507, 537, 538
  liczba wątków, 515
  obsługa błędów, 536
  odbieranie komunikatu, 499
  onerror, właściwość, 538
  onmessage, 499
  postMessage, metoda, 498, 538
  sposób działania, 492, 493, 494
  terminate, metoda, 538
  tworzenie, 497
  wątki podrzędne, 537
  zakańczanie, 536, 538
  zalety, 514
Web Socket, 551
Web SQL, 555
Web Workers, 490, 491, 538

importScripts, funkcja, 507, 537, 538
liczba wątków, 515
obsługa przez przeglądarki, 496
onerror, właściwość, 538
onmessage, 499
postMessage, metoda, 538
terminate, metoda, 538
zakańczanie, 538
WebM, format, 370, 375
WebStorage, 136, 433, 438, 480
wsparcie, 438
while, pętla, 67, 76, 77, 105
wideo, 64, 367, 371, 406
a canvas, 406
abort waiting, zdarzenie, 381
API, 381
autoplay, 371, 372, 426
błędy, 423, 424, 426
canPlayType, metoda, 381, 386, 387, 389, 426
controls, 372, 426
currentTime, właściwość, 381
duration, właściwość, 381
efekty specjalne, dodawanie, 407, 408, 409, 410, 411, 412, 413, 414, 415, 417
ended, 381, 385, 426
error, 381
Flash, 380
formaty, 370, 374, 375, 376, 377, 378, 426
getFormatExtension, metoda, 388
głośność, 378
kodek, 376
kontener, 376
kontrolki, 373
kontrolki, własne, 402, 403
load, metoda, 381
loadeddata, zdarzenie, 381
loadedmetadata, zdarzenie, 381
loop, 372, 381
muted, właściwość, 381
obsługa przez przeglądarki, 370, 375
pause, metoda, 381
paused, właściwość, 381
pełny ekran, 378
play, metoda, 381
poster, 372, 426
preload, 372
progress, zdarzenie, 381
przetwarzanie, 410, 411
readyState, właściwość, 381
seeking, właściwość, 381
source, znacznik, 426
src, 372, 426
strumieniowanie, 421, 422
timeupdate, zdarzenie, 381
umieszczanie kilku formatów, 376, 377
videoHeight, właściwość, 381
videoWidth, właściwość, 381
volume, właściwość, 381
volumechange, zdarzenie, 381
width i height, 372
window, obiekt, 181, 182, 184, 186
window.onload, 105, 182, 184, 185
Windows, serwer WWW, 253
współrzędne geograficzne, 191
WWW serwer
hosting, 254
Linux, 253
Mac, 253
Windows, 253
wyrażenia
liczbowe, 73
logiczne, 73, 79
tekstowe, 73

X

XHTML, 41, 43, 63, 548
XMLHttpRequest, obiekt, 242, 243, 247, 261, 262, 282, 283, 299
onload, właściwość, 261, 263
responseText, właściwość, 244

Z

zbiór Mandelbrota, 509, 510
zdarzenia
błędy wideo, 423, 424, 426
ended, 385, 426
kliknięcie przycisku, 117, 118, 119, 120, 136
obsługa, 117, 136
zmienne, 68, 69
deklarowanie, 68, 105
globalne, 149, 150, 151, 152, 153, 186
lokalne, 149, 150, 151, 152, 153, 186
nazywanie, 70, 71, 72
przypisanie funkcji, 154, 155
zmiana wartości, 69

Najlepszy podręcznik do HTML5!

HTML5. Rusz głową!

Chcesz tworzyć strony internetowe, które są dynamiczne, interaktywne, bogate w treści i utrzymują łączność z innymi serwisami. Chwila, na pewno chodzi Ci o strony internetowe? A może lepiej użyć HTML5 do tworzenia pełnokrwistych aplikacji internetowych? A jeżeli już, to czemu nie skorzystać z najnowszych technologii, które sprawdzą się zarówno w przeglądarkach desktopowych, jak i urządzeniach mobilnych? Poza tym na pewno interesują Cię nowe możliwości oferowane przez HTML5, takie jak geolokalizacja, wideo, grafika 2D, składowanie danych, wątki robocze i wiele innych, prawda?

Kolejna książka z serii *Rusz głową!* to najlepszy sposób na opanowanie nowości HTML5. Niezwykle atrakcyjna forma graficzna oraz nowoczesna metodologia nauki sprawią, że już za kilkanaście dni będziesz specjalistą w zakresie HTML5. Dowiedz się, jak wykorzystać usługi geolokalizacyjne, jak rysować na płótnie (canvas) oraz składować dane na komputerze użytkownika. Sprawdź również, co możesz zyskać dzięki WebSockets oraz jak wycisnąć siódme poty z języka JavaScript. Nauka HTML5 jeszcze nigdy nie była tak łatwa!

Poznaj HTML5 i korzystaj z:

- usług geolokalizacyjnych
- elementów multimedialnych
- bazy danych w przeglądarce
- zaawansowanych funkcji graficznych

Najnowsze standardy, najlepsze praktyki — Twoja przepustka do świata aplikacji internetowych!

helion.pl
helion sa
Helion
Nr katalogowy: 8731
Księgarnia internetowa: http://helion.pl
Zamówienia telefoniczne: 0 801 339900
0 601 339900
Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
e-mail: helion@helion.pl
http://helion.pl
ISBN: 978-83-246-4339-4
Cena 87,90 zł
informatyka w najlepszym wydaniu