Płótno — wprowadzenie

Element `<canvas>` definiuje płótno, czyli obszar, na którym odbywa się rysowanie. Z perspektywy kodu HTML, jego deklaracja nie mogłaby być prostsza. Do znacznika otwierającego należy dołączyć trzy atrybuty: `id`, `width` i `height`.

```
<canvas id="drawingCanvas" width="500"
height="300"></canvas>
```

Atrybut `id` nadaje elementowi `<canvas>` unikalną nazwę, potrzebną do jego zidentyfikowania w kodzie JavaScript. Z kolei atrybuty `width` i `height` ustawiają — odpowiednio — szerokość i wysokość elementu, wyrażoną w pikselach.

Zwykle po wczytaniu strony element `<canvas>` widnieje na niej jako pusty, nieobramowany element (praktycznie niewidoczny na białym tle). Aby wyróżnić go, możesz nadać mu kolor tła lub ramę, używając w tym celu arkusza stylów.

```
canvas {
  border: 1px dashed black;
}
```

Do przetestowania możliwości jakie daje płótno użyjemy następującego szablonu:

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="utf-8">
  <title>Szablon płótna</title>

  <style>
canvas {
  border: 1px dashed black;
}
  </style>

  <script>
window.onload = function() {
  var canvas = document.getElementById("drawingCanvas");
  var context = canvas.getContext("2d");

  // Umieść kod odpowiedzialny za rysowanie w tym miejscu.
};
  </script>
</head>

<body>
  <canvas id="drawingCanvas" width="500" height="300"></canvas>
</body>
</html>
```

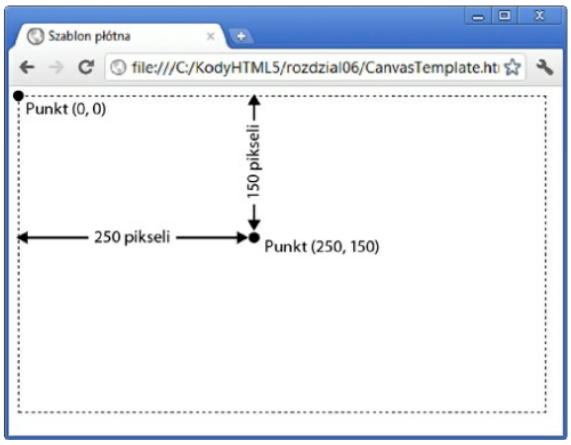
Zadeklarowane w elemencie `<style>` formatowanie otoczy komponent `<canvas>` obramowaniem. Sekcja `<script>` obsługuje zdarzenie `window.onload`, które zachodzi po tym, jak przeglądarka załaduje stronę. W dalszej części kod JavaScript pobiera element `<canvas>` i tworzy kontekst. Od tej chwili można go użyć do rysowania. Warto wykorzystać ten szablon jako punkt startowy Twoich własnych eksperymentów z tym elementem.

Ćwiczenie 32

1. Utwórz folder „32 Szablon canvas”.
2. W folderze zapisz plik z kodem HTML pokazanego wyżej szablonu.
3. Za pomocą dostępnych przeglądarek sprawdź, czy obramowanie jest widoczne.

Linie proste

Jesteś gotów, by przystąpić do pracy. Nim jednak zaczniesz rysować na płótnie, musisz zrozumieć, jak interpretować współrzędne wyznaczane w jego obrębie przez przeglądarkę.



Rysunek 1 Podobnie jak wszystkie inne elementy HTML, komponent wyznacza na swój punkt zerowy (współrzędne 0, 0) lewy górny róg.

Gdy poruszasz się wzdłuż poziomej krawędzi, wzrasta wartość na osi x; gdy schodzisz wzdłuż poziomego boku, zmienia się wartość współrzędnej y. Jeśli płótno ma rozmiar 500×300 pikseli, jego prawy dolny róg będzie miał współrzędne (500, 300).

Linia prosta jest najprostszą rzeczą, jaką możesz narysować na płótnie. Proces jej rysowania składa się z trzech etapów. Najpierw należy wykorzystać zawartą w obiekcie kontekstu metodę `moveTo()`, aby przenieść się do punktu, z którego zaczniesz malowanie. Następnym krokiem jest wywołanie metody `lineTo()`, która przeniesie kontekst na przekazane koordynaty. Na koniec wywołuje się metodę `stroke()`, odpowiedzialną za narysowanie linii między dwoma poprzednio wyznaczonymi punktami.

```
context.moveTo(10,10);
context.lineTo(400,40);
context.stroke();
```

Spójrz na tę procedurę w ten sposób: najpierw unosisz flamaster nad miejscem, z którego chcesz zacząć rysować (metoda `moveTo`), po czym aż do pewnego punktu kreślisz linię (metoda `lineTo`), która jest wypełniana kolorem (metoda `stroke`).

W efekcie powstaje cienka (o grubości jednego piksela) czarna linia, która zaczyna się od punktu o współrzędnych `(10, 10)` i kończy się w punkcie `(400, 40)`.

Na szczęście, nawet z linii prostych da się wydobyć o wiele więcej. W dowolnym momencie przed wywołaniem funkcji `stroke()`, która sprawia, że linia pojawia się na ekranie, możesz zmodyfikować trzy własności kontekstu rysowania: `lineWidth`, `strokeStyle` i `lineCap`. Własności te wpływają na wszystko, co narysujesz, dopóki ich nie zmienisz.

Wyrażona w pikselach własność `lineWidth` ustawia grubość kreski. Grubą na 10 pikseli linię deklaruje się w ten sposób:

```
context.lineWidth = 10;
```

Z kolei własności `strokeStyle` używa się do zmiany koloru rysowanych linii. Definiuje się ją za pomocą nazwy koloru, szesnastkowego kodu barw HTML lub znanej z CSS funkcji `rgb()`, która pozwala utworzyć barwę z mieszaniny odpowiednich proporcji czerwieni, zieleni i błękitu. W większości programów graficznych używa się podobnej składni do deklarowania kolorów (przykłady składowych można odczytać tutaj: http://pl.wikipedia.org/wiki/Pomoc:Kolory). Niezależnie od tego, które rozwiązanie wybierzesz, całą wartość należy zawrzeć w cudzysłowie, tak jak na przykładzie:

```
// Wyrażona w kodzie szesnastkowym deklaracja zmienia kolor
na ceglaną czerwień.
context.strokeStyle = "#cd2828";
// Funkcja rgb() zmienia kolor na ceglaną czerwień.
context.strokeStyle = "rgb(205,40,40)";
```

Ostatnia własność — `lineCap` — służy do ustawienia wyglądu zakończeń linii prostej. Ustawienie domyślne kończy narysowany odcinek ostrym, prostokątnym koniuszkiem (deklaruje się go słowem kluczem `butt`). Możesz jednak nadać odcinkowi większe, zaokrąglone (słowo klucz `round`) lub kwadratowe (słowo klucz `square`) zwieńczenie (powstała końcówka będzie podobna do domyślnej, z tym że rozszerzona o połowę grubości linii).

Niżej przedstawiono pełny skrypt, który rysuje trzy poziome linie.

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="utf-8">
  <title>Linie</title>

  <style>
canvas {
  border: 1px dashed black;
```

```
}
  </style>

  <script>

window.onload = function() {
  var canvas = document.getElementById("drawingCanvas");
  var context = canvas.getContext("2d");
  // Umieść kod odpowiedzialny za rysowanie w tym miejscu.

  context.lineWidth = 20;
  context.strokeStyle = "rgb(205,40,40)";

  context.moveTo(10,50);
  context.lineTo(400,50);
  context.lineCap = "butt";
  context.stroke();

  context.beginPath();
  context.moveTo(10,120);
  context.lineTo(400,120);
  context.lineCap = "round";
  context.stroke();

  context.beginPath();
  context.moveTo(10,190);
  context.lineTo(400,190);
  context.lineCap = "square";
  context.stroke();
};

  </script>
</head>

<body>
  <canvas id="drawingCanvas" width="500" height="300"></canvas>
</body>
</html>
```

Ćwiczenie 33

1. Utwórz folder „33 Linie”.
2. Zapisz w folderze plik HTML, którego kod znajduje się powyżej.
3. Zmodyfikuj kod w taki sposób, aby linie były narysowane pionowo. Dodatkowo zmień kolory linii w taki sposób, aby każda była narysowana w innym kolorze.

Ćwiczenie 34

1. Utwórz folder „34 Własny przykład zastosowania linii”.
2. W folderze należy wzorując się na poprzednim zadaniu narysować dowolną konstrukcję złożoną z różnokolorowych linii o różnej grubości.
3. Efekt należy sprawdzić w przeglądarce.

Ścieżki i figury

W poprzednim przykładzie nowe linie oddzieliłeś od narysowanych wcześniej, deklarując rysowanie oddzielnej ścieżki (ang. *path*) dla każdej z nich poprzez wywołanie metody `beginPath()`. Dzięki temu mogłeś każdej z nich nadać inną barwę (oraz grubość i zwieńczenie). Ścieżki są ważne z innego powodu — pozwalają na wypełnienie figur. Załóżmy, że korzystając z niżej przedstawionego kodu, utworzyłeś trójkąt z czerwonymi krawędziami:

```
context.moveTo(250,50);
context.lineTo(50,250);
context.lineTo(450,250);
context.lineTo(250,50);
context.lineWidth = 10;
context.strokeStyle = "red";
context.stroke();
```

Jeżeli zależy Ci na *wypełnieniu* pola trójkąta barwą, metoda `stroke()` Ci w tym nie pomoże. Aby to osiągnąć, w pierwszym rzędzie należy zamknąć ścieżkę, wywołując metodę `closePath()`, następnie wybrać kolor, korzystając z własności `fillStyle`, i na koniec wywołać metodę `fill()`, która pokryje wyznaczoną powierzchnię barwą.

```
context.closePath(); context.fillStyle = "blue";
context.fill();
```

W tym przykładzie warto dostroić kilka parametrów. Po pierwsze, jeżeli masz zamiar zamknąć ścieżkę, nie musisz rysować zamykającego odcinka, gdyż metoda `closePath()` wygeneruje linię pomiędzy ostatnim punktem końcowym a punktem startowym. Po drugie, lepiej wypełnić figurę *przed* narysowaniem jej krawędzi, bo kolor krawędzi może znaleźć się częściowo pod wypełnieniem.

Oto kompletny kod rysujący wypełniony trójkąt:

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="utf-8">
  <title>Trójkąt</title>

  <style>
canvas {
  border: 1px dashed black;
}
  </style>

  <script>

window.onload = function() {
  var canvas = document.getElementById("drawingCanvas");
  var context = canvas.getContext("2d");

  context.moveTo(250,50);
  context.lineTo(50,250);
  context.lineTo(450,250);
  context.closePath();
```

```
  // Pokrywa pole trójkąta kolorem.
  context.fillStyle = "blue";
  context.fill();

  // Rysuje krawędzie.
  context.lineWidth = 10;
  context.strokeStyle = "red";
  context.stroke();
};

  </script>
</head>

<body>
  <canvas id="drawingCanvas" width="500" height="300"></canvas>
</body>
</html>
```

UWAGA!

W trakcie rysowania łączących się ze sobą odcinków (np. boków trójkąta) warto użyć kolejnej własności kontekstu — `lineJoin`, aby zaokrąglić lub wyostrzyć krawędzie (deklarując słowa klucze — `round` i `level` — wartość domyślną ustawia hasło `mitre`).

W większości scenariuszy, jeśli chcesz skonstruować złożoną figurę, składasz ją z wielu odcinków w obrębie jednej ścieżki — linia po linii. Jedna figura jest na tyle ważna, że otrzymała własny zestaw metod. Jest to prostokąt. Metoda `fillRect()` pozwala na wypełnienie prostokątnego obszaru w jednym kroku. Twoja rola ogranicza się do podania współrzędnych lewego górnego wierzchołka oraz długości i wysokości.

Przykładowy prostokąt o wymiarach 100×200 pikseli, którego lewy górny róg leży w punkcie `(0, 10)`, powstaje w wyniku przetworzenia takiego kodu:

```
fillRect(0,10,100,200);
```

Metoda `fillRect()` do wypełnienia figury używa barwy ustawionej we własności `fillStyle()`, tak jak metoda `fill()`.

Na podobnej zasadzie działa funkcja `strokeRect()`, która już po jednym wywołaniu rysuje obramowanie prostokąta:

```
strokeRect(0,10,100,200);
```

Metoda ta wykorzystuje informacje pozyskane przez własności `lineWidth` i `strokeStyle`, aby określić grubość i barwę krawędzi, tak samo jak metoda `stroke()`.

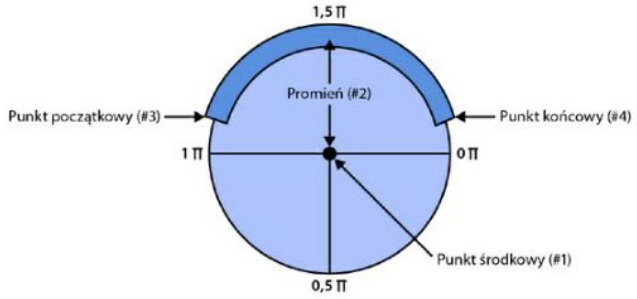
Ćwiczenie 35

1. Utwórz folder „35 Ścieżki i figury”.
2. Umieść w nim kod zamieszczony powyżej i sprawdź jego działanie w przeglądarce.
3. Zmodyfikuj kod odwracając trójkąt o 180 stopni, zamieniając kolor wypełnienia i obramowania oraz umieszczając go na żółtym prostokącie.

Krzywe

Ciekawszy efekt uda nam się osiągnąć za pomocą metod, odpowiedzialnych za rysowanie krzywych; są to metody `arc()`, `arcTo()`, `bezierCurveTo()` i `quadraticCurveTo()`. Każda z tych funkcji rysuje krzywe w inny sposób.

Metoda `arc()` jest najprostsza. Rysuje łuk okręgu. Zanim go narysujesz, wyobraź sobie okrąg, a następnie zdecyduj, który jego fragment pasuje do Twoich celów. Rysunek niżej pomoże Ci wybrać parametry, które przekażesz do metody `arc()`.



Gdy już je ustalisz, możesz wywołać funkcję:

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

// Tworzy zmienne, które przechowają parametry łuku.
var centerX = 150;
var centerY = 300;
var radius = 100;
var startingAngle = 1.25 * Math.PI;
var endingAngle = 1.75 * Math.PI;

// Wykorzystuje podane wyżej parametry do narysowania łuku.
context.arc(centerX, centerY, radius, startingAngle,
endingAngle);
context.stroke();
```

Inną opcją jest wywołanie metody `closePath()` przed użyciem funkcji `stroke()`, co doda linię prostą między dwoma końcami łuku. W rezultacie powstanie domknięty półokrąg.

Rysunek 2 Narysowanie łuku wydaje się proste, lecz wymaga podania kilku parametrów

Najpierw musisz podać dane wystarczające do utworzenia okręgu. Wymaga to ustalenia współrzędnych jego środka (#1) oraz promienia (#2), który określa wielkość okręgu. Następnie należy zdefiniować długość łuku, co wiąże się z wyznaczeniem jego punktu początkowego (#3) oraz punktu końcowego (#4). Punkty te definiowane są za pomocą kątów, wyrażonych w radianach, które są wielokrotnością stałej Π (1Π to kąt połowy okręgu, 2Π to kąt całego okręgu, jak ukazano na rysunku)

Nota bene, okrąg jest łukiem, który rozciąga się po kącie 360 stopni. Poniższy kod wygeneruje w przeglądarce właśnie pełny okrąg:

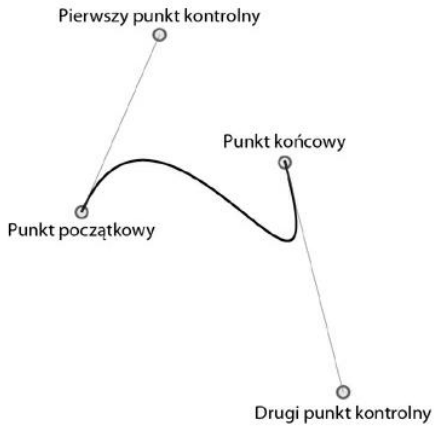
```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");


var centerX = 150;
var centerY  = 300;
var radius  = 100;
var startingAngle = 0;
var endingAngle  = 2 *  Math.PI;


context.arc(centerX, centerY, radius, startingAngle,
endingAngle);
context.stroke();
```

Ćwiczenie 36

1. Utwórz folder „36 Krzywe – okrąg”.
2. W folderze zredaguj plik HTML rysujący pełny okrąg.
3. Efekt sprawdź w przeglądarce.

Pozostałe trzy metody rysowania łuków

Czas na metody `arcTo()`, `bezierCurveTo()` i `quadraticCurveTo()`, które wymagają odrobinę więcej namysłu i wyobraźni przestrzennej. Opierają się na idei **punktów kontrolnych**, czyli punktów ulokowanych poza łukiem, które jednak wpływają na sposób jego rysowania. Najsłynniejszym przykładem metody rysowania łuków w ten właśnie sposób jest krzywa Béziera, którą wykorzystuje się w praktycznie każdym komputerowym programie graficznym. Jej popularność wynika z faktu, iż tworzy krzywą, która wygina się płynnie niezależnie od wielkości rysunku.



Rysunek 3 Krzywa Béziera ma dwa punkty kontrolne

Początek krzywej związany jest z pierwszym punktem kontrolnym, a jej koniec — z drugim. Pomiędzy nimi linia prosta ulega przekrzywieniu. Stopień zakrzywienia zależy od odległości punktów kontrolnych od punktu końcowego lub początkowego — im dalej te pierwsze są oddalone od drugich, tym linia jest bardziej „wyciągnięta". W działaniu przypomina to odwróconą grawitację

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");
// Definiuje punkt początkowy krzywej
context.moveTo(62, 242);
// Tworzy zmienne, w których zapisywane są współrzędne dwóch punktów kontrolnych i punktu końcowego.
var control1_x = 187;
var control1_y = 32;
var control2_x = 429;
var control2_y = 480;
var endPointX = 365;
var endPointY = 133;
// Rysuje krzywą.
```

```
context.bezierCurveTo(control1_x, control1_y, control2_x,
control2_y, endPointX, endPointY);

context.stroke();
```

Powstały rysunek jest skomplikowaną, nieforemną figurą, składającą się z zespolonych ze sobą łuków i krzywych. Na koniec możesz wywołać metodę `closePath()`, by później wypełnić lub dodać obrys tego tworu. Najlepiej nauczysz się, ćwicząc.

Ćwiczenie 37

1. Utwórz folder „37 Krzywe Beziera”.
2. Posługując się przykładowym kodem oraz witryną *http://tinyurl.com/html5bezier* utwórz kod HTML rysujący figurę wykorzystującą krzywą Beziera.
3. Efekt sprawdź w przeglądarce.

Transformaty

Transformatą nazywamy technikę rysowania, która umożliwia przesunięcie w ramach skali współrzędnych płótna. Wyobraź sobie, że chcesz narysować ten sam kwadrat w trzech miejscach. Możesz wywołać metodę `fillRect()` trzy razy, z trzema różnymi zestawami parametrów.

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

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

// Rysuje kwadrat o rozmiarach 30 x 30 pikseli w trzech
miejscach.

context.rect(0, 0, 30, 30);

context.rect(50, 50, 30, 30);

context.rect(100, 100, 30, 30);

context.stroke();
```

Alternatywą jest wywołanie funkcji `fillRect()` trzy razy z *tymi samymi* parametrami, po czym przesunięcie skali dla każdego z nich o pewien wektor, w trzy różne punkty:

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

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


// Rysuje kwadrat w punkcie (0, 0).

context.rect(0, 0, 30, 30);


// Przesuwa punkt zerowy płótna o 50 pikseli w dół i w
prawo.

context.translate(50, 50);

context.rect(0, 0, 30, 30);
```

```
// Przesuwa punkt zerowy płótna odrobinę dalej. Transformaty kumulują się ze sobą.
// W rezultacie punkt (0, 0) według starej miary jest punktem (100, 100).
context.translate(50, 50);
context.rect(0, 0, 30, 30);

context.stroke();
```

Efekt przetworzenia obu wersji kodu jest ten sam: w trzech różnych miejscach generowane są trzy kwadraty.

Ćwiczenie 38

1. Utwórz folder „38 Transformaty”.
2. Wykorzystując powyższy przykład napisz kod HTML , który rysuje 3 przesunięte prostokąty wypełnione różnymi kolorami.
3. Efekt sprawdź w przeglądarce.