Tekst w canvas

Aby wypisać tekst, możemy skorzystać z dwóch funkcji:

- ✓ `fillText("tekst", x, y)` wypisuje wypełniony tekst w pozycji x, y
- ✓ `strokeText("tekst", x, y)` wypisuje obrysowany tekst w pozycji x, y

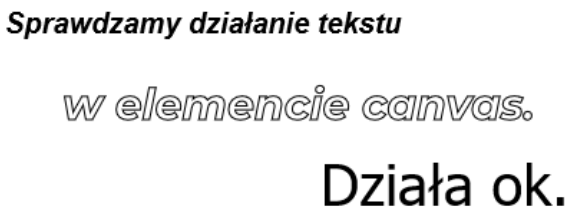
W przypadku operacji na tekście możemy skorzystać z właściwości `font`, która pozwala na określenie wyglądu czcionki w taki sam sposób jaki stosujemy w CSS:

- ✓ styl (font-style) – normal, italic, oblique
- ✓ grubość (font-weight) – normal, bold, bolder, lighter lub wartości od 100 do 900 (co 100)
- ✓ font-size – określa rozmiar czcionki/wysokość interlinii
- ✓ font-family – określa krój czcionki

Możemy zastosować wymienione wyżej właściwości w następującej kolejności: `[font style][font weight][font size][font family]`.

Zobaczmy na przykładzie, jak wygląda praca z tekstem w canvas:

```
<canvas width="400" height="400" id="canvas"></canvas>

<script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");

    ctx.font = "italic bold 20px Arial";
    ctx.fillText("Sprawdzamy działanie tekstu", 10, 60);

    ctx.font = "italic bold 30px Montserrat";
    ctx.strokeText("w elemencie canvas.", 50, 120);

    ctx.font = "normal 40px Tahoma";
    ctx.fillText("Działa ok.", 230, 180);
</script>
```



Możemy również wykorzystać kolorowanie tekstu. Działa ono analogicznie jak w przypadku figur geometrycznych – możemy korzystać ze `strokeStyle`, jak i `fillStyle`:

```
<canvas width="400" height="400" id="canvas"></canvas>

<script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");

    ctx.fillStyle = "red"
    ctx.font = "italic bold 20px Arial";
    ctx.fillText("Sprawdzamy działanie tekstu", 10, 60);

    ctx.strokeStyle = "blue"
    ctx.font = "italic bold 30px Montserrat";
    ctx.strokeText("w elemencie canvas.", 50, 120);

    ctx.fillStyle = "teal"
    ctx.font = "normal 40px Tahoma";
    ctx.fillText("Działa ok.", 230, 180);
</script>
```

Spróbujmy pokolorować tekst z wykorzystaniem gradientu liniowego – najpierw zajmiemy się obramowaniem:

```
<canvas width="600" height="400" id="canvas"></canvas>

<script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");

    const gradient = ctx.createLinearGradient(50, 120, 500, 200)
    gradient.addColorStop(0, "yellow")
    gradient.addColorStop(0.25, "orange")
    gradient.addColorStop(0.5, "red")
    gradient.addColorStop(0.75, "purple")
    gradient.addColorStop(1, "blue")

    ctx.strokeStyle = gradient
    ctx.font = "bold 27px Montserrat";
    ctx.strokeText("Obramowanie tekstu", 50, 120);
    ctx.strokeText("z wykorzystaniem gradientu", 50, 160);
</script>
```

Obramowanie tekstu
z wykorzystaniem gradientu

Teraz spróbujmy pokolorować wypełnienie:

```
<canvas width="600" height="400" id="canvas"></canvas>

<script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");

    const gradient = ctx.createLinearGradient(50, 120, 500, 200)
    gradient.addColorStop(0, "yellow")
    gradient.addColorStop(0.25, "orange")
    gradient.addColorStop(0.5, "red")
    gradient.addColorStop(0.75, "purple")
    gradient.addColorStop(1, "blue")

    ctx.fillStyle = gradient
    ctx.font = "bold 27px Montserrat";
    ctx.fillText("Wypełnienie tekstu", 50, 120);
    ctx.fillText("z wykorzystaniem gradientu", 50, 160);
</script>
```

Wypełnienie tekstu
z wykorzystaniem gradientu

Jak widać, gradienty również mogą być wykorzystywane do wypełniania bądź obrysowywania tekstów.

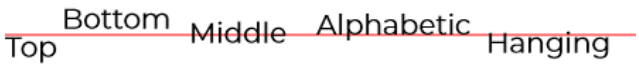
Poza możliwością konfigurowania wyglądu oraz koloru czcionki, możemy stosować także wyrównanie tekstu:

- ✓ `textBaseline` określa pionowe wyrównanie tekstu względem danego punktu, możliwe wartości to: top, hanging, middle, alphabetic, ideographic, bottom; domyślną jest alphabetic
- ✓ `textAlign` pozwala na wyrównanie tekstu w poziomie, możliwe wartości to: start, end, left, right, center; domyślną wartością jest start

Ilustracja poniżej pokazuje, jak działają poszczególne wyrównania tekstu w pionie:

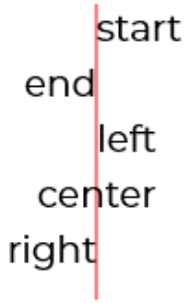
Na przykładzie poniżej zobaczymy w praktyce, jak działa wyrównanie w pionie – narysujemy dodatkową linię pomocniczą, względem której będziemy obserwować zmiany:

```
<canvas width="600" height="400" id="canvas"></canvas>

<script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");

    ctx.strokeStyle = "red"
    ctx.moveTo(20, 100)
    ctx.lineTo(500, 100)
    ctx.stroke()

    ctx.font = "20px Montserrat"

    ctx.textBaseline = "top"
    ctx.fillText("Top", 20, 100)
    ctx.textBaseline = "bottom"
    ctx.fillText("Bottom", 60, 100)
    ctx.textBaseline = "middle"
    ctx.fillText("Middle", 150, 100)
    ctx.textBaseline = "alphabetic"
    ctx.fillText("Alphabetic", 240, 100)
    ctx.textBaseline = "hanging"
    ctx.fillText("Hanging", 360, 100);
</script>
```



Teraz przyjrzyjmy się wyrównaniu w poziomie:

```
<canvas width="600" height="400" id="canvas"></canvas>

<script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");

    ctx.strokeStyle = "red"
    ctx.moveTo(200, 20)
    ctx.lineTo(200, 180)
    ctx.stroke()

    ctx.font = "20px Montserrat"

    ctx.textAlign = "start";
    ctx.fillText("start", 200, 40)
    ctx.textAlign = "end"
    ctx.fillText("end", 200, 70)
    ctx.textAlign = "left"
    ctx.fillText("left", 200, 100)
    ctx.textAlign = "center"
    ctx.fillText("center", 200, 130)
    ctx.textAlign = "right"
    ctx.fillText("right", 200, 160)
</script>
```



`Start` oraz `left` oznacza, że tekst rozpoczyna się w danym punkcie, `end` i `right` – tekst kończy się w danym punkcie, `center` – tekst jest wyśrodkowany względem danego punktu. Mówimy oczywiście o punkcie będącym „początkiem” danego tekstu.

Jeżeli byśmy chcieli policzyć szerokość ile zajmuje wypisany tekst, możemy skorzystać z funkcji `measureText()`, która zwraca dane o tekście pisanym z danym formatowaniem:

```html
<script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");

    const str = "Test wymiaru"

    ctx.font = "bold 40px Comic Sans MS";
    ctx.fillText(str, 50, 100);

    const textMetrics  = ctx.measureText(str);

    ctx.beginPath();
    ctx.moveTo(50, 120);
    ctx.lineTo(50 + textMetrics.width, 120);
    ctx.stroke();
</script>
```

Definiujemy zmienną tekstową przechowującą treść naszego napisu. Następnie po zdefiniowaniu właściwości tekstu oraz utworzeniu go za pomocą `fillText`, tworzymy zmienną przechowującą informacje na temat naszego tekstu.

```
TextMetrics
    actualBoundingBoxAscent: 29.6875
    actualBoundingBoxDescent: 11.30859375
    actualBoundingBoxLeft: -0.1953125
    actualBoundingBoxRight: 258.6145833333333
    width: 259.8833312988281
```

Obok przedstawiłam informacje wypisane w konsoli za pomocą `console.log(textMetrics)`. Możemy z nich odczytać właściwość `width` – czyli szerokość tekstu.

Rozpoczynamy rysowanie prostej w punkcie (50, 120). Kontynuujemy jej rysowanie do punktu oddalonego od punktu początkowego i długość tekstu. W ten sposób powstaje podkreślenie tekstu.

ĆWICZENIE 1 – tekst

Utwórz plik o nazwie imie_nazwisko_cw1.html. Utwórz prostą stronę zawierającą element canvas o dowolnych wymiarach, a następnie rozmieść w nim cztery różne teksty. Do ich formatowania wykorzystaj pokazane powyżej wiadomości: kolor obrysu, kolor wypełnienia, styl, grubość, rozmiar, krój czcionki oraz różne sposoby wyrównania. Gotowy plik zamieść na moodle.

Zapisywanie i odczytywanie stanu

W celu zapisu i odczytu stanu canvasu możemy wykorzystać podane niżej metody:

- ✓ `save()` służy do zapisania stanu canvasu - w tym właśnie ustawień odnośnie rysowania
- ✓ `restore()` służy do odczytania zapisanego wcześniej stanu

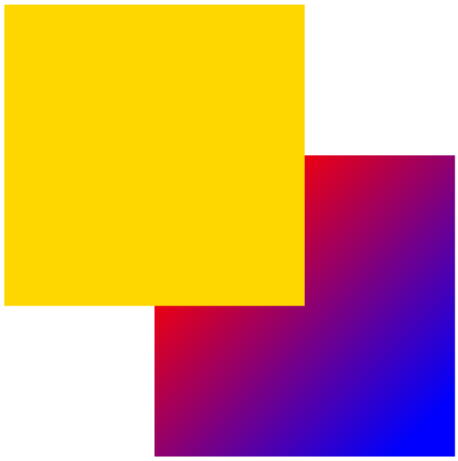
Metody te działają na zasadzie stosu. `Save()` odkłada coś na stos, a `restore()` pobiera ostatni zapisany stan.

W przypadku canvas zapisywane są następujące rzeczy:

- ✓ transformacje
- ✓ obszar przycinania za pomocą clip()
- ✓ ustawienia właściwości: globalAlpha, globalCompositeOperation, strokeStyle, textAlign, textBaseline, lineCap, lineJoin, lineWidth, miterLimit, fillStyle, font, shadowBlur, shadowColor, shadowOffsetX, i shadowOffsetY

Jak to wygląda w praktyce? Przykładowo rysujemy jakąś grafikę. Ustawiamy kilka właściwości rysowania - np. grubość linii czy kolor. Żeby nie psuć obecnych ustawień odkładamy jest na stos za pomocą metody `save()`. Ustawiamy nowe parametry. Po narysowaniu kształtów chcemy wrócić do początkowych ustawień więc korzystamy z `restore()`.

```
<canvas width="600" height="400" id="canvas"></canvas>

<script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");

    ctx.fillStyle = "gold"

    ctx.save() //zapisuje początkowy styl

    const gradient = ctx.createLinearGradient(20, 20, 300, 300)
    gradient.addColorStop(0, "orangered")
    gradient.addColorStop(0.5, "red")
    gradient.addColorStop(1, "blue")
    ctx.fillStyle = gradient
    ctx.fillRect(120, 120, 200, 200)

    ctx.restore() //wczytuje zapisany stan

    ctx.fillRect(20, 20, 200, 200)
</script>
```



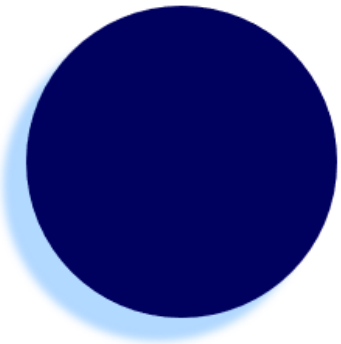
Cień

Do rysowania cienia może służyć jedna z niżej wymienionych metod:

- ✓ `shadowOffsetX` pozycja cienia w osi x
- ✓ `shadowOffsetY` pozycja cienia w osi y
- ✓ `shadowBlur` rozmycie cienia
- ✓ `shadowColor` kolor cienia

Przykładowe zastosowanie:

```
<canvas width="250" height="300" id="canvas"></canvas>

<script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");

    ctx.shadowOffsetX = -15;
    ctx.shadowOffsetY = 15;
    ctx.shadowColor = "rgba(0,128,255,0.3)"
    ctx.shadowBlur = 5;
    ctx.fillStyle = "#00005f"
    ctx.arc(canvas.width/2, canvas.height/2, 100, 0, 2*Math.PI);
    ctx.fill()
</script>
```



ĆWICZENIE 2 – zapisywanie i odczytywanie stanu, cień

Utwórz plik o nazwie imie_nazwisko_cw2.html. Utwórz prostą stronę zawierającą element canvas o dowolnych wymiarach. Spróbuj zapisać jakieś początkowe formatowanie elementu (np. grubość linii, styl łączenia, kolor, cień itp.). Utwórz dowolny element przypisując mu nowe wartości wykorzystanych wcześniej właściwości (np. zamiast koloru zielonego wykorzystaj czerwony). Następnie odzyskaj wcześniejsze ustawienia i na ich podstawie utwórz inny element. Gotowy plik zamieść na moodle.