# L'API Canvas en JavaScript

# Table des matières

## I. Contexte

**Durée** : 1 h

**Environnement de travail** : Repl.it

**Pré-requis** : JavaScript, HTML

***Contexte***

`<canvas>` est un élément HTML permettant d'implémenter des images, photos, graphiques et tout élément visuel. Cet élément fournit une API JavaScript permettant de traiter et de créer des rendus visuels dynamiquement.

## II. Utiliser la balise <canvas>

### Objectifs

- Implémenter une balise HTML `<canvas>`
- Initialiser un canevas avec l'API Canvas

### Mise en situation

L'élément `<canvas>` permet de dessiner des éléments graphiques à l'aide de scripts. Cet élément peut aussi recevoir tout type de média (svg, image, vidéo). Nous verrons dans ce cours comment implémenter un `canvas` et les possibilités offertes par cet élément.

***Méthode*** **La balise <canvas>**

La balise `canvas` est une balise HTML qui a deux attributs, `width` et `height`. Sa taille peut être modifiée via du CSS, mais l'image sera affichée selon la taille des attributs, puis transformée par la suite. Cette méthode n'est pas conseillée, car cela peut déformer l'image. Il faut également noter que certains navigateurs anciens ne supportent pas l'élément canevas (Internet Explorer 9, par exemple). Dans ce cas, on peut implémenter du contenu de repli en insérant du texte ou une image entre les balises.

***Exemple***

```
<div>
  <canvas id="myFirstCanevas" width="200" height="200">
  // contenu de repli
  ce texte s'affiche pour les navigateurs qui ne supportent pas canvas
  </canvas>
</div>
```

***Méthode*** **L'API Canvas**

Pour initialiser du contenu, il faut passer par l'API Canvas[1]. Via cette API, nous allons pouvoir initialiser un ou plusieurs contextes de rendus, qui vont pouvoir être en deux ou trois dimensions.

Le type de contexte sera le paramètre de la méthode `getContext()` ;.

1 https://developer.mozilla.org/fr/docs/Web/HTML/Canvas

***Exemple***

```
// élément html canvas
const canvas = document.getElementById('myFirstCanvas') ;
// initialisation du contexte
const ctx = canvas.getContext('2d') ;

```

***Remarque***

Dans le cas où le canevas ne serait pas supporté par le navigateur, ce code JavaScript retournerait une erreur. Si l'on souhaite disposer de rétrocompatibilité, il vaut mieux utiliser ce code pour initialiser le canevas.

```
const canvas = document.getElementById('myFirstCanvas');
let ctx;
if (canvas.getContext) {
  ctx = canvas.getContext('2d') ;
} else {
  // code pour les anciens navigateurs
}
```

***Syntaxe*** **À retenir**

- La balise `canvas` accepte deux attributs, `width` et `height`.
- La taille de l'image sera définie en fonction de la largeur et de la hauteur fixées.
- Pour initialiser un canevas, on doit lier la balise HTML à un script JavaScript.

## III. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

### Question

[solution n°1 p.33]

Dans une application qui devra être compatible avec tous les navigateurs, on vous demande d'intégrer un élément `canvas` dans le HTML et de fournir le code JavaScript en relation avec celui-ci. À partir du cours, créez un canevas et le script JavaScript en relation.

La taille du canevas sera de 150 px de large et 150 px de hauteur.

Le canevas devra représenter un carré rouge. Nous verrons la construction de formes dans les chapitres suivants, c'est pourquoi le code JavaScript de création de la forme vous est fourni ci-dessous :

```
// déclaration du canvas
ctx.fillStyle = 'red';
ctx.fillRect(0, 0, 150, 150)
```

Dans le cas où le navigateur ne supporterait pas le canevas, un message devra s'afficher en console : "Canevas non supporté".

Vous testerez votre code dans un environnement de travail repl.it[2].

---

1 https://repl.it/
2 https://repl.it/repls/FewNotableCookie

# IV. Dessiner des rectangles et définir des tracés et des formes

### Objectif

- Construire des formes géométriques

### Mise en situation

Maintenant que nous savons implémenter un canevas, nous allons voir comment construire des formes géométriques et tracer des lignes.

***Remarque*** **Le système de coordonnées HTML**

Les coordonnées `x` et `y` d'une page HTML se basent sur une grille dont les points 0,0 se situent en haut à gauche de la page. Donc si `x` et `y` sont égaux à 10, le point sera situé à 10 pixels du haut gauche de la page.

Un canevas fonctionne sur le même principe.

***Méthode*** **Les rectangles**

Il existe 3 fonctions permettant d'écrire des rectangles :

- `fillRect(x, y, width, height)` qui dessine un rectangle rempli.
- `strokeRect(x, y, width, height)` qui dessine un rectangle vide avec contour.
- `clearRect(x, y, width, height)` qui rend transparente une zone rectangulaire.

***Exemple*** **Un rectangle plein**

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Rectangle</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <canvas id="rectangle"></canvas>

</body>

<script>

const canvas = document.getElementById('rectangle');
let ctx;
if (canvas.getContext) {
  ctx = canvas.getContext('2d') ;
  // On indique au contexte de constuire un rectangle à 10px du top, 10px du left, 50px de largeur et 30px de hauteur
  ctx.fillRect(10, 10, 50, 30);
} else {
  // code pour les anciens navigateurs
}

</script>
</html>
```

***Méthode*** **Les tracés**

Pour construire d'autres formes géométriques que des rectangles, il faut les construire en reliant des points de tracés. On appelle cela un trajet. Il est constitué de plusieurs segments, dont le dernier point rejoint le premier. Ces segments peuvent être de toutes formes (droits, incurvés, etc.).

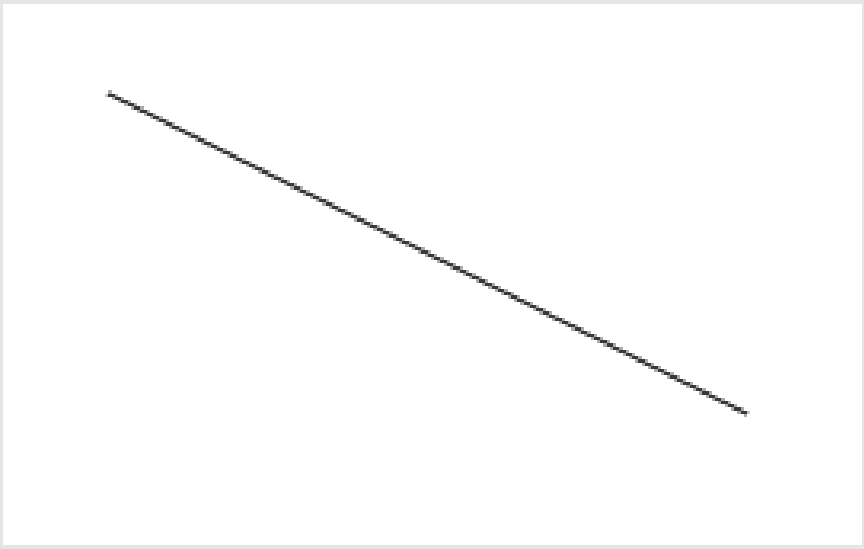
Pour réaliser un trajet, il faut l'initialiser avec la fonction `beginPath()`, et la méthode `moveTo(x, y)` permettra d'établir le point de départ de notre forme.

Les fonctions les plus couramment utilisées pour construire des lignes sont `lineTo(x, y)` qui définit une ligne droite, et `arcTo(x1, y1, x2, y2, radius)` définissant une ligne courbe.

Si le dernier point n'a pas fermé notre objet, il faudra terminer la forme par la fonction `closePath()`, qui créera une ligne droite vers ce point.

```
<code>
<!doctype html>
<html lang="fr">
<head>
 <meta charset="utf-8">
  <title>Rectangle</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<canvas id="trait"></canvas>
</body>
<script>
const canvas = document.getElementById('trait');
let ctx;
if (canvas.getContext) {
ctx = canvas.getContext('2d') ;
  // On indique au contexte de tracer un trait  qui démarre aux coordonnées 50px, 25px et qui fini aux coordonnées 250px, 125px
ctx.beginPath();
```

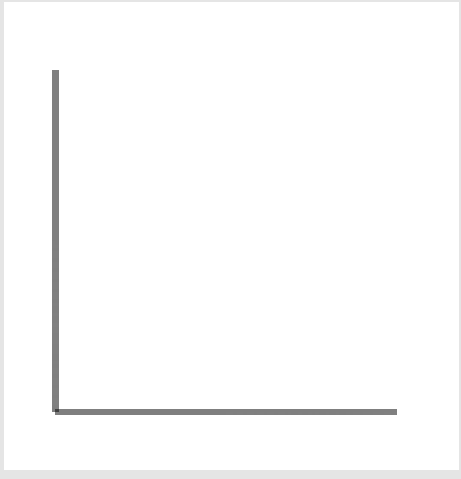
```
ctx.moveTo(50, 25);
ctx.lineTo(250, 125);
ctx.stroke();}
else {
 // code pour les anciens navigateurs
</script>
</html>
</code>

```



On va pouvoir dessiner toutes sortes de figures en dessinant plusieurs lignes à la suite dans le canevas. L'une des figures les plus simples à créer est le triangle.

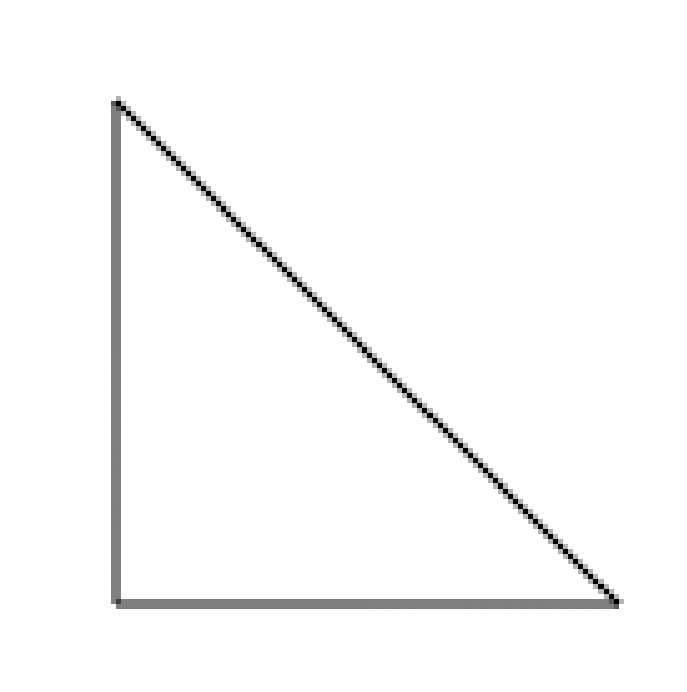
Pour dessiner plusieurs lignes à la suite, il suffit d'utiliser plusieurs fois lineTo() : les coordonnées du point défini par la première méthode lineTo() serviront de point de départ pour la ligne tracée par le deuxième appel à la méthode lineTo() et etc.

```
<code>
ctx.beginPath();
ctx.moveTo(25, 25);
ctx.lineTo(25, 125);
ctx.lineTo(125, 125);
ctx.stroke();
</code>

```



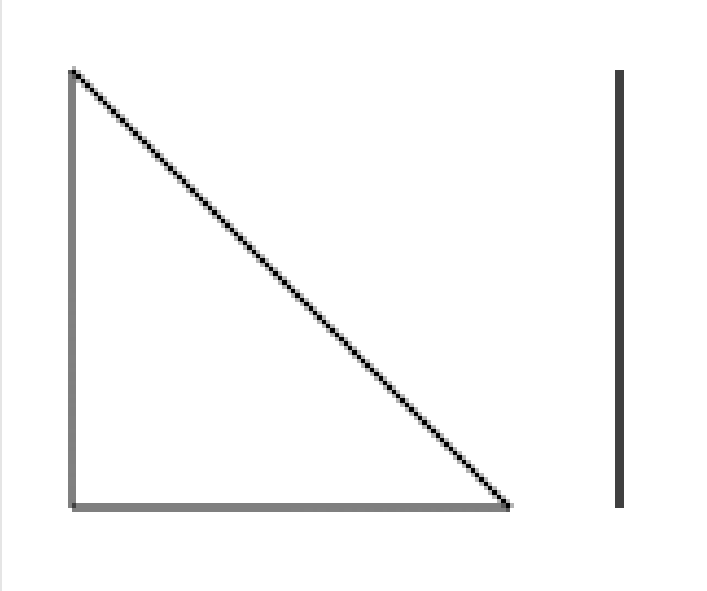
Ajout de la commande closePath() pour relier le premier et le dernier point.

```
<code>
ctx.beginPath();
ctx.moveTo(25, 25);
ctx.lineTo(25, 125);
ctx.lineTo(125, 125);
ctx.closePath();
ctx.stroke();
</code>

```



Plusieurs tracés dans un même canvas

```
<code>

ctx.beginPath();
ctx.moveTo(25, 25);
ctx.lineTo(25, 125);
ctx.lineTo(125, 125);
ctx.closePath();
ctx.stroke();

ctx.beginPath();
ctx.moveTo(150, 25);
ctx.lineTo(150, 125);
ctx.stroke();

</code>

```



***Méthode*** **arcTo()**

Construire une courbe n'est pas aisé et demandera souvent plusieurs essais et un peu de pratique. La méthode `arcTo(x1, y1, x2, y2, radius)` prend en paramètres deux points de contrôle et un radius.

Les points de contrôle forment un angle correspondant aux tangentes des deux extrémités de la courbe. Le premier point de contrôle est relié au point de départ et construit la première tangente, le second point de contrôle est relié au premier et crée la deuxième tangente. Le radius déterminera la courbure de l'arc.

***Conseil***

Avant de coder une forme géométrique, il peut être intéressant de la dessiner sur une feuille quadrillée pour avoir les repères.

***Remarque***

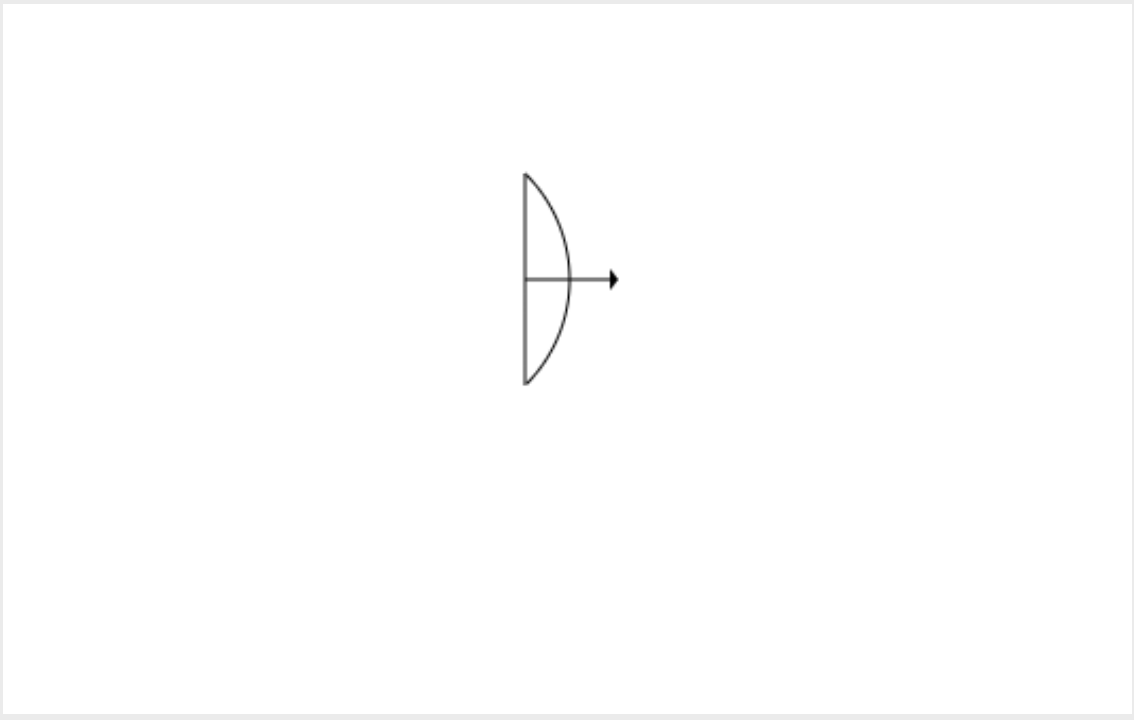
Il existe plusieurs méthodes pour créer des arcs avec l'API Canvas. Ces méthodes se trouvent dans la documentation[1].

***Exemple*** **Un arc avec une flèche**

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Cercle</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <canvas id="arc"></canvas>

</body>

```

1 https://developer.mozilla.org/fr/docs/Tutoriel_canvas/Formes_g%C3%A9om%C3%A9triques#Arcs

```
<script>

  const canvas = document.getElementById('arc');
  let ctx;
  if (canvas.getContext) {
    ctx = canvas.getContext('2d') ;
    // construction de l'arc
    ctx.beginPath();
    ctx.moveTo(20,20);
    //ctx.arcTo(70, 50, 10, 80, 30)
    ctx.arcTo(70, 70, 20, 120, 70)

    // corde de l'arc
    //ctx.moveTo(20,20);
    ctx.moveTo(20,20);
    ctx.lineTo(20,120);

    // bois de la fleche
    ctx.moveTo(20, 70);
    ctx.lineTo(60,70)

    ctx.stroke();

    // pointe de la fleche
    ctx.beginPath();
    ctx.moveTo(60, 65);
    ctx.lineTo(64,70)
    ctx.lineTo(60,75)
    ctx.fill();
  }

</script>
</html>
```

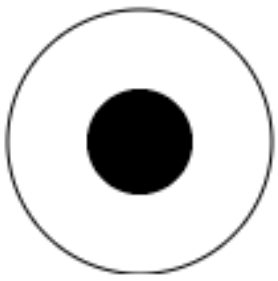


***Méthode*** **Les cercles**

Pour créer un cercle dans un canevas, on utilise la méthode `arc(x, y, rayon, angle de départ, angle de fin)`. Pour créer un cercle entier, la valeur de l'angle de départ doit être de 0 et celle de l'angle de fin égale à **Math.PI * 2**.

***Exemple*** **Un cercle plein dans un cercle vide**

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Cercle</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <canvas id="cercle"></canvas>
</body>

<script>
  const canvas = document.getElementById('cercle');
  const ctx = canvas.getContext('2d');
  // arc vide
  ctx.beginPath();
  ctx.arc(100, 100, 50, 0, Math.PI * 2);
  ctx.stroke();

  // arc plein
  ctx.beginPath();
  ctx.arc(100, 100, 20, 0, Math.PI * 2);
  ctx.fill();

</script>
</html>
```



***Syntaxe*** **À retenir**

- Il existe une seule forme géométrique en canevas : le rectangle. Pour dessiner toute autre forme géométrique, il faudra créer un tracé.
- Il est conseillé d'écrire au préalable des formes complexes sur papier quadrillé afin d'avoir une idée plus précise des différents points de coordonnées.
- Il existe plusieurs méthodes pour créer des arcs : celles-ci vous sont fournies dans la documentation[1].

## V. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

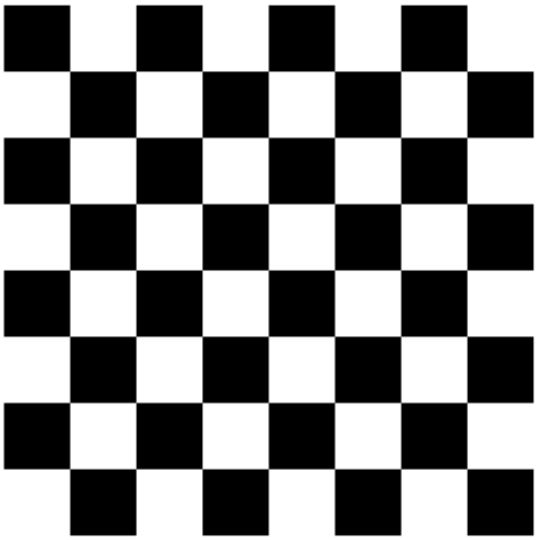
[2]

### Question

[solution n°2 p.33]

Dans le cadre d'une application de jeu en ligne, vous devez créer un échiquier. Pour faire cela, la solution du canevas a été retenue.

Écrivez le code JavaScript pour que votre canevas forme un échiquier. Les cases feront une taille de 30 px.



Vous testerez le code dans un environnement de travail repl.it[3].

## VI. Ajouter des couleurs, dégradés, ombres et transparences

1 https://developer.mozilla.org/fr/docs/Tutoriel_canvas/Formes_g%C3%A9om%C3%A9triques#Arcs
2 https://repl.it/
3 https://repl.it/repls/FewNotableCookie

### Objectif

- Ajouter des éléments de style à des canevas

### Mise en situation

Dans ce chapitre, nous allons appliquer des styles à nos formes : couleurs, dégradés, transparences et ombrages.

***Méthode*** **Les couleurs**

Il existe deux méthodes pour affecter des couleurs : `fillStyle` qui sera utilisée pour le remplissage de forme, et `strokeStyle` pour la coloration des traits.

Ces deux méthodes acceptent plusieurs nomenclatures :

- ctx.fillStyle = 'red';
- ctx.fillStyle = '#FF0000';
- ctx.fillStyle = 'rgb(255,0,0)';
- ctx.fillStyle = 'rgba(255, 0, 0, 1)';

Pour appliquer une couleur à un trait ou une forme, il faut déclarer la couleur avant la forme.

***Exemple*** **Un carré rouge**

```
1 const canvas = document.getElementById('myFirstCanvas');
2 let ctx;
3 if (canvas) {
4 ctx = canvas.getContext('2d') ;
5 ctx.fillStyle = 'red';
6 ctx.fillRect(10, 10, 130, 130);
7 }
```

***Méthode*** **Appliquer de la transparence**

La transparence est une valeur comprise entre 0 et 1. Plus on se rapproche de 1, plus le rendu est opaque.

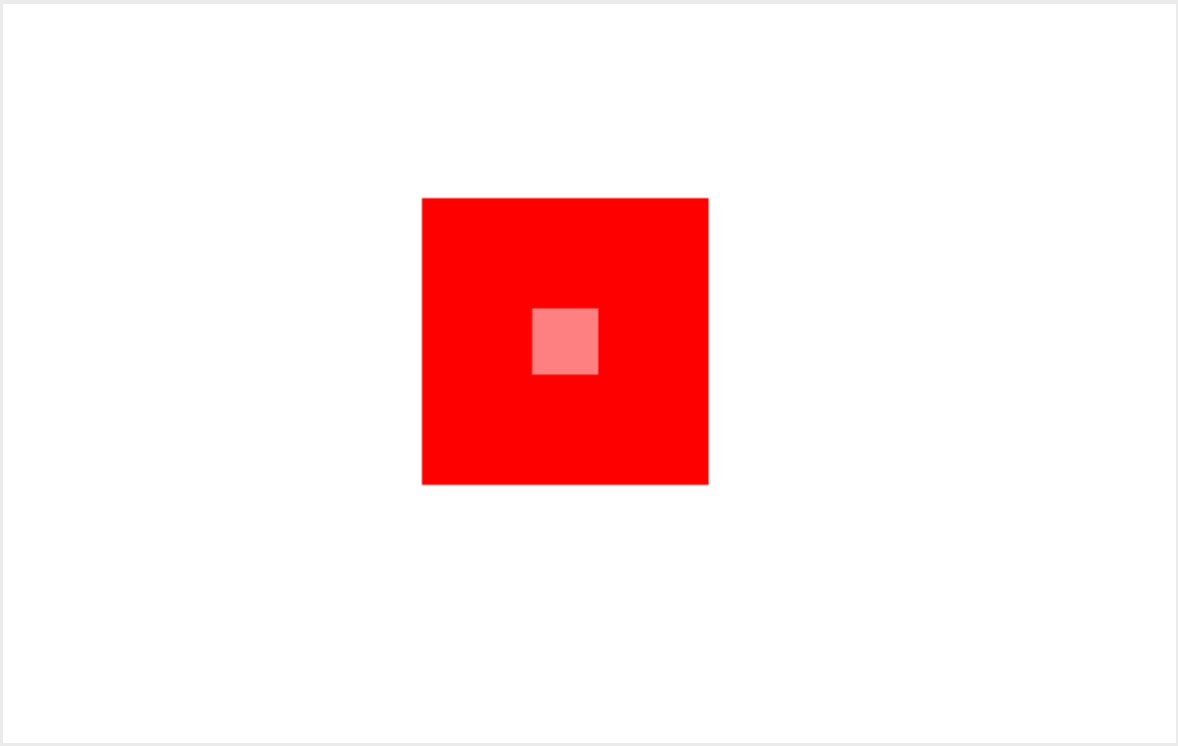
On peut affecter de la transparence à un élément de deux façons :

- `fillStyle` et `strokeStyle` acceptent la nomenclature `rgba CSS` qui permet de mettre la valeur de la transparence en dernier élément de la fonction. Par exemple : `ctx.strokeStyle = 'rgba(255, 0, 0, 1)';`.
- `globalAlpha` applique la transparence spécifiée à toutes les formes futures. Il prend comme valeur celle de la transparence.

***Exemple*** **Un carré transparent à l'intérieur d'un carré rouge**

```
1  const canvas = document.getElementById('myFirstCanvas');
2  let ctx;
3  if (canvas.getContext) {
4
5    ctx = canvas.getContext('2d') ;
6    ctx.fillStyle = 'red';
7    ctx.fillRect(10, 10, 130, 130);
8
9    ctx.fillStyle = 'white';
10   ctx.globalAlpha = 0.5;
11   ctx.fillRect(60, 60, 30, 30)
```

```
12
13 }
```



***Méthode*** **Les dégradés**

Il y a deux types de dégradés : linéaires et radiaux.

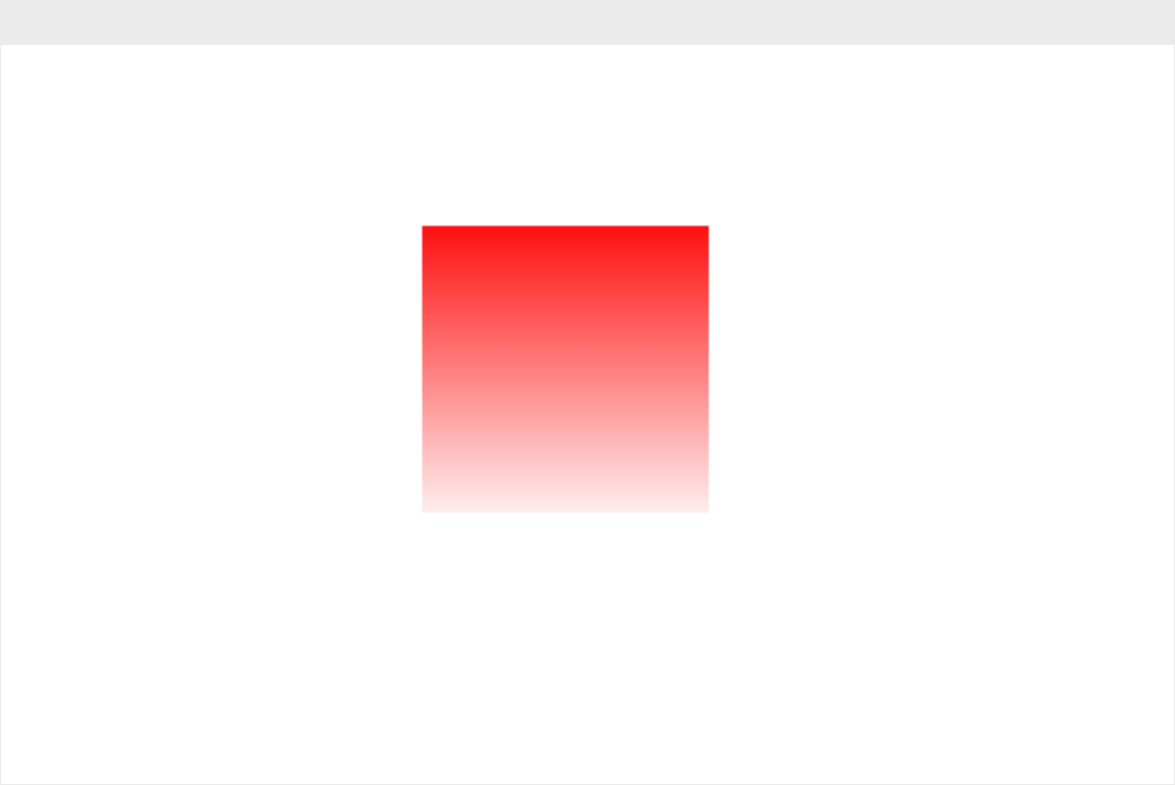
- Linaire : `createLinearGradient(x1, y1, x2, y2)` avec x1, y1 le point de départ et x2, y2 le point d'arrivée
- Radial : `createRadialGradient(x1, y1, r1, x2, y2, r2)` x, y et r représentant deux cercles ayant comme centre x, y et de rayon r

Nous pouvons assigner des couleurs aux dégradés via la méthode `addColorStop(position, color)`.

Le paramètre `position` est une valeur comprise entre 0 et 1 et indique à quel moment le dégradé va évoluer.

***Exemple*** **Un carré rouge dégradé**

```
1 const canevas = document.getElementById('myFirstCanevas');
2 let ctx;
3 if (canvas.getContext) {
4
5   ctx = canevas.getContext('2d') ;
6   const lingrad = ctx.createLinearGradient(0, 0, 0, 150);
7   lingrad.addColorStop(0, 'red');
8   lingrad.addColorStop(1, 'white');
9
10  ctx.fillStyle = lingrad;
11  ctx.fillRect(10, 10, 130, 130);
12 }
```



***Méthode*** **Les ombres**

Les ombres comportent 4 propriétés :

- `shadowOffsetX` : indique la distance horizontale sur laquelle l'ombre doit s'étendre à partir de l'objet.
- `shadowOffsetY` : indique la distance verticale sur laquelle l'ombre doit s'étendre à partir de l'objet.
- `shadowBlur` : taille du floutage.
- `shadowColor` : couleur de l'ombre.

À part `shadowColor`, dont la valeur par défaut est noir transparent, les autres propriétés ont toutes 0 comme valeur par défaut.

***Exemple*** **Un texte avec un effet d'ombre**

```
const canvas = document.getElementById('myFirstCanvas');
let ctx;
if (canvas.getContext) {

  ctx = canvas.getContext('2d') ;
  ctx.shadowOffsetX = 1;
  ctx.shadowOffsetY = 3;
  ctx.shadowBlur = 1;
  ctx.shadowColor = 'black';

  ctx.font = '30px Arial';
  ctx.fillText('Hello world', 30, 30);
}
```


***Syntaxe*** **À retenir**

- Nous avons vu dans ce chapitre qu'il est assez simple d'affecter du style à nos formes. Nous ne sommes limités que par notre imagination.
- Ce cours est une introduction au canevas, il vous fournit les bases. La documentation canevas est assez importante et vous y trouverez des compléments pour réaliser des projets plus complexes.

## VII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

### Question

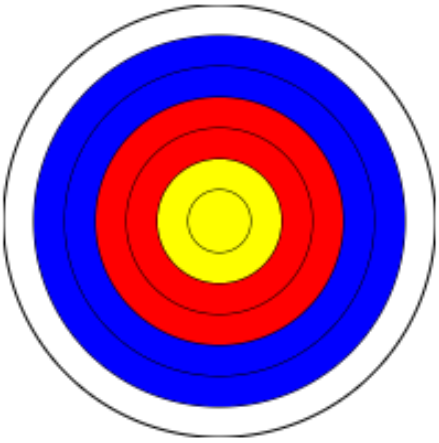
[solution n°3 p.34]

Pour les besoins d'un petit jeu en ligne, vous devez créer une cible de tir à l'arc dans un canevas. Le canevas fera 196 px de hauteur et de largeur.

- Cette cible sera composée de 7 cercles les uns dans les autres, tous séparés par la même distance.
- Le plus grand cercle fera la taille du canevas.
- Il n'est pas nécessaire que la taille du cercle soit dynamique. Le canevas sera toujours de la même taille.
- La couleur bleu est `blue`.
- La couleur rouge est `red`.

1 https://repl.it/

- La couleur jaune est `yellow`.
- La cible devra ressembler à l'image ci-dessous.



Vous copierez le code dans un environnement de travail repl.it[1].

## VIII. Ajouter du texte et des images

**Objectifs**

- Ajouter du texte
- Ajouter des images

**Mise en situation**

La balise `canvas` permet aussi d'afficher du texte et des images. Un exemple très concret de cette application est le logo de profil dans un cercle. Lorsque nous sommes connectés sur un site, si nous avons une photo de profil, le cercle affiche la photo ; sinon, le cercle affiche nos initiales.

***Méthode*** **Ajouter du texte**

Dans le chapitre précédent, nous avons déjà intégré un texte pour illustrer les ombrages. Ici, nous allons voir plus en profondeur les différentes propriétés et comment les appliquer.

Le texte d'un canevas possède 4 propriétés :

- `font` : définit la taille et le style du texte.
- `textAlign` : définit l'alignement horizontal du texte par rapport aux coordonnées de celui-ci. Elle accepte 5 valeurs : *start, end, left, right, center*.

1 https://repl.it/repls/FewNotableCookie

- `textBaseline` : définit l'alignement de base du texte. Elle accepte *top*, *middle*, *alphabetic*, *ideographic*, *bottom*.
- `direction` : direction du texte. Elle prend deux valeurs possibles : *ltr* (de gauche à droite) ou *rtl* (de droite à gauche).

***Méthode*** **Centrer un texte**

Pour centrer le texte, quelle que soit la taille du canevas, il faut récupérer les valeurs des attributs `width` et `height`. La position `x` du texte sera `width / 2` et la position `y` sera `height / 2`.

Pour cela on dispose de la méthode getAttribute(), getAttribute renvoie la valeur d'un attribut donné de l'élément spécifié. Si l'attribut n'existe pas, la valeur renvoyée sera soit null soit "" (une chaine vide) .

Il faudra alors affecter `center` à la propriété `textAlign`, et `middle` à la propriété `textBaseline` .

***Exemple*** **Un texte centré**

```
const canvas = document.getElementById('myFirstCanvas');
let ctx;

if (canvas.getContext) {

  // Ajustement de la taille du canvas. Il faut pouvoir centrer le texte quelle que soit la taille du canvas
  // Ici le canvas sera toujour un carré, la taille de la hauteur sera toujours égale à la largeur.

  // On récupère la valeur de l'attribut width
  let sizeCanvas = canvas.getAttribute('width');

  // Si l'attribut n'existe pas alors la valeur par défaut sera 150
  sizeCanvas  = sizeCanvas === '0' || sizeCanvas === null ? 150 : sizeCanvas;
  // On affecte ces valeurs aux attributs de l'élément HTML
  canvas.setAttribute('height', sizeCanvas)
  canvas.setAttribute('width', sizeCanvas)

  // Taille du texte par rapport à la taille du canvas
  const sizeText = sizeCanvas / 3

  ctx = canvas.getContext('2d') ;
  // Mise en place du texte
  ctx.fillStyle = 'red'
  ctx.textAlign = 'center'
  ctx.textBaseline = 'middle';
  ctx.font = `${sizeText}px Arial`;
  // On ajuste la position du texte par rapport à la taille du canvas
  ctx.fillText('JD', sizeCanvas/2, sizeCanvas/2);
}
```

***Méthode*** **Insérer des images**

À présent, nous allons voir comment gérer les images avec les canevas.

Pour insérer une image dans un canevas, il va déjà falloir se procurer une référence à cette image. Généralement, on utilisera le constructeur new Image() [1]pour créer un nouvel objet HTMLImageElement puis la propriété src pour indiquer le chemin de l'image qu'on souhaite insérer. C'est fonctionnellement équivalent à document.createElement('img').

Les images peuvent mettre un certain temps à se charger, c'est pourquoi l'ajout d'images à l'intérieur d'un canevas nécessite une fonction d'événement, `load()`.

L'intégration de l'image se fait via la méthode `drawImage(img, x, y width, height)` et le canevas doit être initialisé dans le `callback` de la méthode `load()`. En effet, si le canevas se charge avant l'image, il ne pourra pas l'afficher.

***Exemple*** **Deux méthodes d'intégration d'image**

Chargement d'une fonction via l'événement `onload` dans le le HTML :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Titre de la page</title>
    <link rel="stylesheet" href="style.css">
  </head>

  <body onload="drawCanvas()">
      <canvas id="myFirstCanvas"></canvas>
  </body>

  <script>
    function drawCanvas() {
      const img = new Image();
      let sizeCanvas = canvas.getAttribute('width');
      sizeCanvas  = sizeCanvas === '0' || sizeCanvas === null ? 150 : sizeCanvas;
      canvas.setAttribute('height', sizeCanvas)
      canvas.setAttribute('width', sizeCanvas)
      img.onload = () => {
        const newWidth = img.width * sizeCanvas / img.height // garde les proportions de
l'image.
        ctx.drawImage(img, 0, 0, newWidth, sizeCanvas);
        ctx.fillRect(0,0,sizeCanvas,sizeCanvas)
      }
      img.src = 'myImage'
    }
   </script>
</html>
```

Via la méthode `addEventListener()` :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Titre de la page</title>
    <link rel="stylesheet" href="style.css">
  </head>
```

1 https://developer.mozilla.org/fr/docs/Web/API/HTMLImageElement/Image

```
8
9   <body>
10      <canvas id="myFirstCanvas"></canvas>
11  </body>
12
13  <script>
14    const img = new Image();
15    let sizeCanvas = canvas.getAttribute('width');
16    sizeCanvas  = sizeCanvas === '0' || sizeCanvas === null ? 150 : sizeCanvas;
17    canvas.setAttribute('height', sizeCanvas)
18    canvas.setAttribute('width', sizeCanvas)
19    img.addEventListener('load', () => {
20      const newWidth = img.width * sizeCanvas / img.height
21      ctx.drawImage(img, 0, 0, newWidth, sizeCanvas);
22      ctx.rectfill(0,0,sizeCanvas,sizeCanvas)
23    })
24    img.src = 'myImage'
25   </script>
26 </html>
```

***Remarque*** **Contenir une image dans un canvas**

Pour créer un badge contenant une image, il faut que l'image se limite au cadre de notre canevas. Pour cela, il existe la méthode `globalCompositeOperation()` appelée avant et après la création du cercle.

Cette méthode peut prendre différente valeurs.

Quatre s'appliquent par rapport au contenu que nous allons créer (source) :

- source-over (place la source au dessus).
- source-in (seule la source qui chevauche ce qui est déjà dessiné est affichée).
- source-out (seule la source qui ne chevauche pas ce qui est déjà dessiné est affichée).
- source-atop (la partie de la source qui chevauche ce qui est déjà dessiné est affiché).

Quatre autres vont s'appliquer par rapport aux dessins qui ont déjà été faits (destination) :

- destination-over (la destination est au-dessus du nouvel objet).
- destination-in (la destination est affichée sous le nouvel objet).
- destination-out (seule la destination qui n'est pas chevauchée par le nouvel objet est affichée).
- destination-atop (la partie de la destination qui chevauche le nouvel objet est affichée).

```
1 <!doctype html>
2 <html lang="fr">
3 <head>
4   <meta charset="utf-8">
5   <title>Titre de la page</title>
6   <link rel="stylesheet" href="style.css">
7 </head>
8
9   <body>
10      <canvas id="myFirstCanvas"></canvas>
11  </body>
12 <script>
13
14 const canvas = document.getElementById('myFirstCanvas');
15 let sizeCanvas = canvas.getAttribute('width');
16 sizeCanvas === '0' || sizeCanvas === null ? sizeCanvas = 150 : sizeCanvas
```

```
canvas.setAttribute('height', sizeCanvas)
canvas.setAttribute('width', sizeCanvas)

let ctx;
if (canvas.getContext) {
  ctx = canvas.getContext('2d') ;
  const img = new Image();
  img.addEventListener('load', () => {
    const newWidth = img.width * sizeCanvas / img.height
    ctx.drawImage(img, 0, 0, newWidth, sizeCanvas);
    ctx.globalCompositeOperation='destination-in';
    ctx.beginPath();
    ctx.arc(sizeCanvas/2, sizeCanvas/2, sizeCanvas/2, 0, 2 * Math.PI);
    ctx.fill();
    ctx.globalCompositeOperation='source-over';
  })
  img.src="myImage";

}

</script>
</html>
```

***Syntaxe*** **À retenir**

- L'ajout de texte et d'images est relativement simple, mais nécessitera souvent des algorithmes de calculs pour définir les positions des éléments, quelle que soit la taille du canevas.

## IX. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

### Question

[solution n°4 p.35]

Dans le cadre d'un projet web, on vous demande de créer un badge permettant de voir les initiales de l'utilisateur.

Le badge devra respecter certaines conditions :

- Le badge est un cercle avec un fond gris et un texte orange.
- Le diamètre du cercle est égal à la hauteur et largeur du canevas.
- La police de texte sera `Arial` et la taille du texte devra être égale au diamètre du cercle divisé par 2.

Vous pouvez choisir les initiales que vous voulez pour vos tests : ('JD', LD', 'NL').

1 https://repl.it/

Voici les résultats attendus avec et sans image :


Vous testerez le code dans un environnement de travail repl.it[1].

## X. Appliquer des transformations

### Objectifs

- Effectuer une translation
- Effectuer des rotations
- La méthode `transform()`

### Mise en situation

Comme en CSS3, il est possible de transformer nos canevas pour créer des formes plus complexes ou des animations.

***Fondamental*** **save() et restore()**

Avant de commencer à étudier des méthodes de transformation, nous allons examiner deux méthodes indispensables à la création de formes plus complexes :

- `save()` : sauvegarde l'état du canevas dans sa globalité et enregistre les états dans une pile. `save()` peut être appelé plusieurs fois.
- `restore()` : restaure le plus récent état sauvegardé. Chaque appel à `restore()` enlève le dernier état sauvegardé et restaure tous les paramètres sauvegardés.

***Exemple*** **Créer et restaurer des carrés**

Voyons comment fonctionnent `save()` et `restore()` à partir d'un exemple simple.

```
1 const canvas = document.getElementById('canvas');
2 let ctx;
3 if (canvas.getContext) {
4   ctx = canvas.getContext('2d') ;
5   ctx.fillStyle = 'red'
6   ctx.fillRect(30, 30, 30, 30)
7   ctx.save() // Sauvegarde l'atat
8
```

1 https://repl.it/repls/FewNotableCookie

```
  ctx.fillStyle = 'blue'
  ctx.fillRect(70, 30, 30, 30)
  ctx.save() // Sauvegarde l'état

  ctx.fillStyle = 'yellow'
  ctx.fillRect(110, 30, 30, 30)

  ctx.restore()// restaure le dernier état sauvegardé et efface la sauvegarde
  ctx.fillRect(150, 30, 30, 30)

  ctx.restore()// restaure le premier état sauvegardé et efface la sauvegarde
  ctx.fillRect(190, 30, 30, 30)

} else {
  // code pour les anciens navigateurs
}

</script>
</html>
```

**Méthode** **La translation**

La translation correspond au déplacement des coordonnées de l'origine. On utilise cette méthode via `translate(x, y)`.

**Exemple** **Translation de deux carrés**

```
let ctx;
if (canvas.getContext) {

  ctx = canvas.getContext('2d') ;
  ctx.fillStyle = 'red';
  ctx.fillRect(10, 10, 50, 50);

  // On déplace le point d'origine du canevas
  ctx.translate(90, 60)

  ctx.fillStyle = 'blue';
  ctx.fillRect(10, 10, 50, 50);
}
```

***Méthode*** **Les rotations**

Pour effectuer une rotation, on va utiliser la méthode `rotate(angle)`. La rotation va tourner les éléments du canevas dans le sens des aiguilles d'une montre, à partir du point d'origine. Les rotations se cumulent : si l'on crée plusieurs rotations successives, elles se cumuleront entre elles.

***Exemple*** **Du texte vertical**

```
const canvas = document.getElementById('myFirstCanevas');
let ctx;

if (canvas.getContext) {

  ctx = canvas.getContext('2d') ;
  // Rotation
  ctx.rotate(Math.PI/2);
  ctx.font = '30px Arial';
  ctx.fillText('Hello world', 0, -30);
}
```

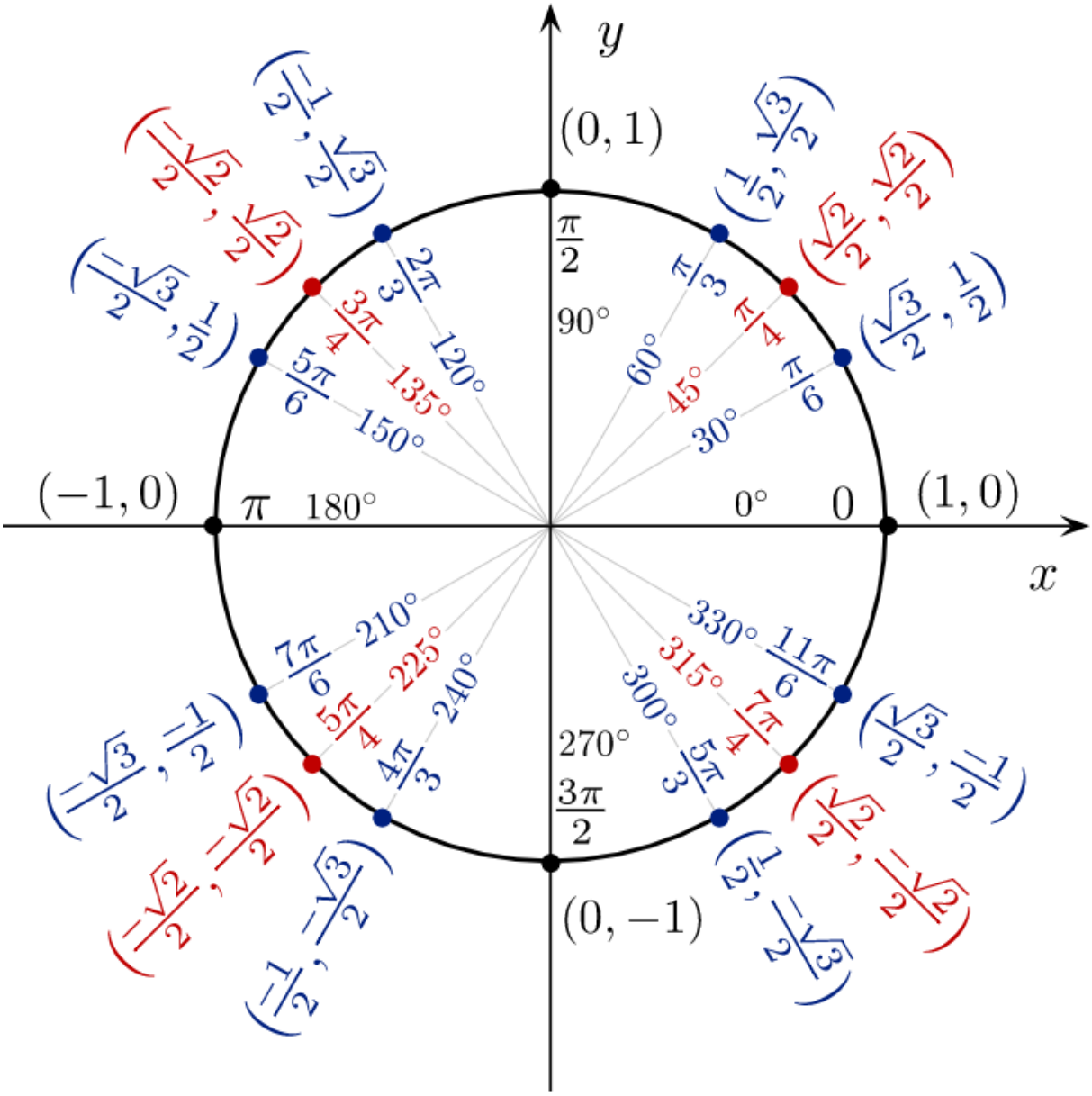
***Remarque***

Les rotations modifient l'emplacement visuel du point d'origine. Cela peut être assez désorientant. Pour se repérer plus facilement, il peut être intéressant d'utiliser cette méthode avec `translate`.

**Quelques notions de trigonométrie**

En mathématiques, le cercle trigonométrique est un cercle qui permet d'illustrer et de définir des notions comme celles d'angle, de radian et les fonctions trigonométriques : cosinus, sinus, tangente.

Dans notre cas , il nous servira à travailler sur les cercles , demi-cercles, angles et les rotations. De manière très simplifiée, ce cercle nous permet d'associer un angle à une valeur de Pi afin d'utiliser cette valeur dans nos formules.




*Image*

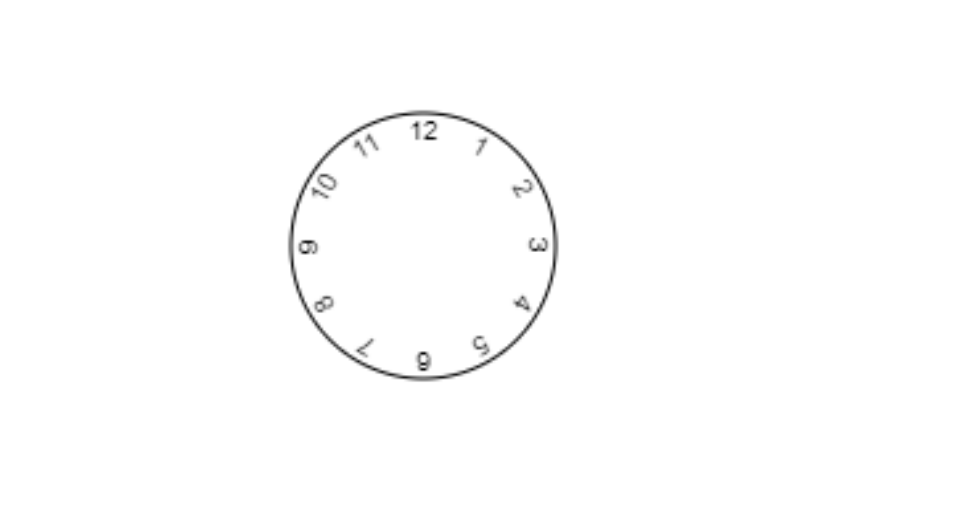
Tableau issu du cercle trigonométrique.

| Angle sexagésimal | 0 | 30° | 45° | 60° | 90° | 120° | 135° | 150° | 180° | 210° | 225° | 240° | 270° | 300° | 315° | 330° | 360° |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Angle en radians | 0 | $\frac{\pi}{6}$ | $\frac{\pi}{4}$ | $\frac{\pi}{3}$ | $\frac{\pi}{2}$ | $\frac{2\pi}{3}$ | $\frac{3\pi}{4}$ | $\frac{5\pi}{6}$ | $\pi$ | $\frac{7\pi}{6}$ | $\frac{5\pi}{4}$ | $\frac{4\pi}{3}$ | $\frac{3\pi}{2}$ | $\frac{5\pi}{3}$ | $\frac{7\pi}{4}$ | $\frac{11\pi}{6}$ | $2\pi$ |

***Exemple*** **Une horloge**

```
const canvas = document.getElementById('cible');
let ctx;
if (canvas.getContext) {
  ctx = canvas.getContext('2d') ;

  // Déplacement du point d'origine
  ctx.translate(150, 75)

```

```
9   // cercle de l'horloge
10  ctx.beginPath();
11  ctx.arc(0, 0, 50, 0, 2 * Math.PI);
12  ctx.stroke();
13
14  // Mise en place des heures
15  ctx.textAlign = 'center';
16  ctx.fillText('12', 0, -40)
17  for (let i = 1; i < 12; i++) {
18    ctx.rotate(Math.PI/6)
19    ctx.fillText(`${i}`, 0, -40)
20  }
21 }
```




***Méthode*** **transform()**

La méthode `transform()` permet de :

- Mettre à l'échelle,
- Pivoter,
- Déplacer,
- Incliner.

La méthode `transform()` prend 6 paramètres. Ces paramètres forment une matrice qui correspond à celle de transformation de l'API Canvas. Lorsque l'on utilise `transfom(p1, p2, p3, p4, p5, p6)`, on multiplie la matrice de transformation courante par la matrice décrite par les arguments de cette méthode.

- `p1` : Échelle horizontale.
- `p2` : Inclinaison horizontale.
- `p3` : Inclinaison verticale.
- `p4` : Échelle verticale.
- `p5` : Déplacement horizontal.
- `p6` : Déplacement vertical.

***Exemple*** **Les carrés**

Reprenons l'exemple des carrés vu précédemment. Ce code fournira exactement le même rendu :

```
const canvas = document.getElementById('canvas');
let ctx;
if (canvas.getContext) {
  ctx = canvas.getContext('2d') ;
  ctx.fillStyle = 'red'
  ctx.fillRect(30, 30, 30, 30)
  ctx.save()

  ctx.fillStyle = 'blue'
  ctx.transform(1, 0, 0, 1, 40, 0) // déplace de 40px sur l'axe des x
  ctx.fillRect(30, 30, 30, 30)
  ctx.save()

  ctx.fillStyle = 'yellow'
  ctx.transform(1, 0, 0, 1, 40, 0) // déplace de 40px sur l'axe des x
  ctx.fillRect(30, 30, 30, 30)

  ctx.restore()
  ctx.transform(1, 0, 0, 1, 80, 0) // déplace de 80px sur l'axe des x à partir du dernier état sauvegardé
  ctx.fillRect(30, 30, 30, 30)

  ctx.restore()
  ctx.transform(1, 0, 0, 1, 160, 0) // déplace de 160px sur l'axe des x à partir du premier état sauvegardé
  ctx.fillRect(30, 30, 30, 30)

} else {
  // code pour les anciens navigateurs
}
```

***Syntaxe*** **À retenir**

Les transformations permettent de modifier l'état des éléments du canevas. Il est possible de :

- Déplacer l'axe d'origine,
- Effectuer une rotation,
- Changer d'échelle,
- Incliner.

Il est possible de sauvegarder l'état d'un élément et de le restaurer.

## XI. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

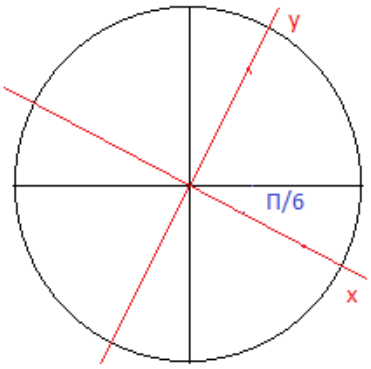
1

1 https://repl.it/

Dans le cours, nous avons créé une horloge pour illustrer la rotation couplée avec la translation. Comme vous l'aurez sûrement remarqué dans l'exemple, les chiffres suivent l'angle de rotation. Dans cet exercice, il va falloir remettre les chiffres dans le bon sens.

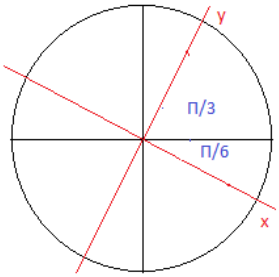
Rappel : nous avons effectué une translation au préalable qui a placé les coordonnées (0, 0) au centre du cercle.

Pour construire une horloge, chaque chiffre est positionné à un angle de 30 ° par rapport au chiffre précédent, soit Π/6. On va donc effectuer une rotation de notre axe de 30 °.




Ensuite, nous allons effectuer une translation au croisement du cercle et de l'axe des `y`. En effet, nous allons devoir effectuer une rotation à partir de ce point pour rétablir l'angle d'écriture. Sinon cela donnera le rendu de l'exemple du cours : les chiffres suivront l'axe de rotation.

Pour connaître la position en `y`, il faut multiplier le rayon par l'angle, soit 50 x Π/3. Comme nos coordonnées (0,0) correspondent au centre du cercle, la valeur de `y` sera négative (dans un `canvas`, les coordonnées positives de `y` descendent). Donc, pour connaître la position de `y`, il faut faire -50 x Π/3.




Une fois cela fait, nous allons pouvoir rétablir l'angle en appliquant la formule inverse de la rotation du cercle, soit -Π/6. Maintenant, nous pouvons écrire notre heure.

### Question

[solution n°5 p.35]

Voici le code JavaScript de création du cercle. Complétez ce code pour obtenir le résultat ci-dessous :


```
const canvas = document.getElementById('hour');
let ctx;
if (canvas.getContext) {
  ctx = canvas.getContext('2d') ;

  // Déplacement du point d'origine
  ctx.translate(150, 75)

  // cercle de l'horloge
  ctx.beginPath();
  ctx.arc(0, 0, 50, 0, 2 * Math.PI);
  ctx.stroke();

  // Votre code ici
} else {
   // code ancien navigateur
}
```

### Indice :

Une fois que vous aurez écrit l'heure, n'oubliez pas de rétablir la rotation ainsi que la translation vers le centre du cercle avant de commencer une nouvelle rotation de l'axe.

## XII. Auto-évaluation

### A. Exercice final

#### Exercice 1

[solution n°6 p.36]

Exercice

Un élément HTML `canvas` possède seulement deux attributs : `width` et `height`.

- ○ Vrai
- ○ Faux

Exercice

Pour initialiser un canevas en JavaScript, on utilise la méthode `getContext(context)`.

- ○ Vrai
- ○ Faux

Exercice

Parmi les réponses suivantes, lesquelles permettent de créer une forme rectangulaire ?

- ☐ fillRect(x, y, width, height)
- ☐ createRect(x, y, width, height)
- ☐ strokeRect(x, y, width, height)
- ☐ clearRect(x, y, width, height)

Exercice

Lors de la création d'un trajet, quelle méthode permet d'initialiser le point de départ ?

- ○ begin(x, y)
- ○ moveTo(x, y)
- ○ entryPoint(x, y)
- ○ start(àx, y)

Exercice

Quelle méthode utilise-t-on pour créer un cercle ? (N'indiquez pas les paramètres)

Exercice

`strokeStyle` permet de modifier les traits de contours et `fillStyle` permet de modifier les couleurs de remplissage d'une forme.

- ○ Vrai
- ○ Faux

Exercice

À quoi sert `globalAlpha` ?

- ○ Modifier le contraste d'une forme
- ○ Modifier la transparence d'une forme
- ○ Transformer les caractères de texte
- ○ Modifier la luminosité d'une forme

Exercice

Quelle méthode permet d'intégrer une image ? (Ne marquez pas les paramètres)

Exercice

À quoi sert la propriété `textAlign` ?

- ○ Définir la position verticale d'un texte
- ○ Définir la position horizontale d'un texte
- ○ Aligner un texte avec une forme
- ○ Changer l'orientation du texte

Exercice

Quelle méthode permet de restaurer le dernier état sauvegardé ?

## B. Exercice : Défi

Dans le cadre d'un projet web, on vous demande de créer un badge permettant de voir :

- Soit les initiales de l'utilisateur s'il n'a pas de photo de profil,
- Soit sa photo de profil si elle existe.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

### Question

[solution n°7 p.37]

Créez un badge respectant ces conditions :

- Le badge est un cercle avec un fond gris et un texte orange,
- Le diamètre du cercle est égal à la hauteur et largeur du canevas,
- La police de texte sera `Arial` et la taille du texte devra être égale au diamètre du cercle divisé par 2.

Créez la fonction `createBadge(canvas, urlImg = null)` dont les paramètres sont :

- `canvas` : élément `canvas`,
- `urlImg` : paramètre optionnel, si l'on ne met pas ce paramètre, alors il sera automatiquement initialisé à `null`.

Pour être sûr que votre fonction marche, il faudra tester si l'image existe avant de la charger. Pour cela, voici la fonction que l'on peut utiliser :

```
const checkImageExists = (imageUrl, callBack) => {
  const imageData = new Image();
  imageData.onload = () => {
    callBack(true);
  };
  imageData.onerror = () => {
    callBack(false);
  };
  imageData.src = imageUrl;
}
```

1 https://repl.it/

Pour utiliser cette fonction :

```
const img = new Image();
checkImageExists(img , (imageExist) => {
   if (imageExist) {
     // On charge l'image et le canvas
   } else {
     // On charge les initials et le canvas
   }
)
```

Voici l'URL d'une image valide que vous pouvez utiliser pour vos tests : https://fotomelia.com/wp-content/uploads/edd/2015/05/main-fleur-paquerette-ciel-bleu-images-photos-gratuites.jpg.

Vous pouvez choisir les initiales que vous voulez pour vos tests : ('JD', LD', 'NL').

Les résultats attendus avec et sans image sont :


Vous testerez le code dans un environnement de travail repl.it[1].

## Solutions des exercices

1 https://repl.it/repls/FewNotableCookie

## p. 4 Solution n°1

```html
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Canvas</title>
  </head>
  <body>
    <canvas id="canvas" width="150" height="150"></canvas>
    <script src="script.js"></script>
  </body>
</html>
```

```javascript
const canvas = document.getElementById('canvas');
let ctx;
if (canvas.getContext) {
   ctx = canvas.getContext('2d') ;
   ctx.fillStyle = 'red';
   ctx.fillRect(0, 0, 150, 150)
} else {
   console.log('canevas non supporté')
 }

```

## p. 12 Solution n°2

```html
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Badge</title>
</head>

  <body>
      <canvas id="canvas" width="240" height="240"></canvas>
  </body>
<script>


  const canvas = document.getElementById('canvas');
  let ctx;
  if (canvas.getContext) {
    ctx = canvas.getContext('2d') ;
    for (let j = 0; j < 8; j++) {
      if (j % 2 === 0) {
        for (let i = 0; i < 8; i++) {
          if (i % 2 === 0) {
          ctx.fillRect(i * 30, j * 30, 30, 30);
          }
        }
      } else {
        for (let i = 0; i < 8; i++) {
          if (i % 2 !== 0) {
          ctx.fillRect(i * 30 , j * 30, 30, 30);
          }
```

```
        }
      }
    }
  }

</script>
</html>
```

p. 16 **Solution n°3**

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Cible</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <canvas id="cible" width="196" height="196"></canvas>

</body>

<script>

const canvas = document.getElementById('cible');
let ctx;
if (canvas.getContext) {
  ctx = canvas.getContext('2d') ;
  let size = 196 / 2;
  for (let cercle = 0; cercle < 8; cercle++) {
    ctx.beginPath();
    ctx.arc(98, 98, size - cercle*14, 0, Math.PI * 2);
    ctx.stroke();
    switch(cercle) {
      case 1:
      ctx.fillStyle = 'blue';
      ctx.fill()
      break;
      case 3:
      ctx.fillStyle = 'red';
      ctx.fill()
      break;
      case 5:
      ctx.fillStyle = 'yellow';
      ctx.fill()
      break;
    }
  }

} else {
  // code pour les anciens navigateurs
}

</script>
</html>
```

p. 21 **Solution n°4**

```html
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Badge</title>
</head>

  <body>
      <canvas id="canvas"></canvas>
  </body>
<script>


  const canvas = document.getElementById('canvas');
  let sizeCanvas = canvas.getAttribute('width');
  sizeCanvas === '0' || sizeCanvas === null ? sizeCanvas = 150 : sizeCanvas
  canvas.setAttribute('height', sizeCanvas)
  canvas.setAttribute('width', sizeCanvas)
  let ctx;
  if (canvas.getContext) {
    ctx = canvas.getContext('2d') ;

    // création du badge avec texte
    ctx.fillStyle = 'grey'
    ctx.beginPath();
    ctx.arc(sizeCanvas/2, sizeCanvas/2, sizeCanvas/2, 0, 2 * Math.PI);
    ctx.fill();

    const sizeText = sizeCanvas / 2
    ctx.fillStyle = 'orange'
    ctx.textAlign = 'center'
    ctx.textBaseline = 'middle';
    ctx.font = `${sizeText}px Arial`;
    // On ajuste la position du texte par rapport à la taille du canvas
    ctx.fillText('JD', sizeCanvas/2, sizeCanvas/2);
  }

</script>
</html>
```

p. 29 **Solution n°5**

```
// Déplacement du point d'origine
  ctx.translate(150, 75)

  // cercle de l'horloge
  ctx.beginPath();
  ctx.arc(0, 0, 50, 0, 2 * Math.PI);
  ctx.stroke();

  // Mise en place des heures
  // On donne à la police une taille de 15% par rapport à la taille du rayon de l'horloge donc on multiplie le rayon par 0.15.
  ctx.font = 50*0.15 + "px arial";
  ctx.textBaseline="middle";
```

```
ctx.textAlign="center";
for(num = 1; num < 13; num++){
  ang = num * Math.PI / 6;
  ctx.rotate(ang);
  ctx.translate(0, -45); // +45 pour revenir au centre
  ctx.rotate(-ang);
  ctx.fillText(num.toString(), 0, 0);
  ctx.rotate(ang);
  ctx.translate(0, 50*((Math.PI / 3)-0.15));
  ctx.rotate(-ang);
}
```

**Exercice** p. 29 **Solution n°6**

**Exercice**

Un élément HTML `canvas` possède seulement deux attributs : `width` et `height`.

- (●) Vrai
- ( ) Faux

**Exercice**

Pour initialiser un canevas en JavaScript, on utilise la méthode `getContext(context)`.

- (●) Vrai
- ( ) Faux

**Exercice**

Parmi les réponses suivantes, lesquelles permettent de créer une forme rectangulaire ?

- [x] fillRect(x, y, width, height)
  *Dessine un rectangle rempli.*
- [ ] createRect(x, y, width, height)
- [x] strokeRect(x, y, width, height)
  *Dessine un rectangle vide avec contour.*
- [x] clearRect(x, y, width, height)
  *Rend transparente une zone rectangulaire.*

**Exercice**

Lors de la création d'un trajet, quelle méthode permet d'initialiser le point de départ ?

- ( ) begin(x, y)
- (●) moveTo(x, y)
- ( ) entryPoint(x, y)
- ( ) start(àx, y)

**Exercice**

Quelle méthode utilise-t-on pour créer un cercle ? (N'indiquez pas les paramètres)

arc()

**Exercice**

`strokeStyle` permet de modifier les traits de contours et `fillStyle` permet de modifier les couleurs de remplissage d'une forme.

- (●) Vrai
- ( ) Faux

**Exercice**

À quoi sert `globalAlpha` ?

- ( ) Modifier le contraste d'une forme
- (●) Modifier la transparence d'une forme
- ( ) Transformer les caractères de texte
- ( ) Modifier la luminosité d'une forme

**Exercice**

Quelle méthode permet d'intégrer une image ? (Ne marquez pas les paramètres)

drawImage

**Exercice**

À quoi sert la propriété `textAlign` ?

- ( ) Définir la position verticale d'un texte
- (●) Définir la position horizontale d'un texte
- ( ) Aligner un texte avec une forme
- ( ) Changer l'orientation du texte

**Exercice**

Quelle méthode permet de restaurer le dernier état sauvegardé ?

restore()

p. 31 **Solution n°7**

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Badge</title>
</head>

  <body>
      <canvas id="badgeWithImage"></canvas>
      <canvas id="badgeWithText"></canvas>
  </body>
<script>

```

```
// fonction qui teste l'image
const checkImageExists = (imageUrl, callBack) => {
  const imageData = new Image();
  imageData.onload = () => {
    callBack(true);
  };
  imageData.onerror = () => {
    callBack(false);
  };
  imageData.src = imageUrl;
}

// fonction de création du badge
const createBadge = (canvas, urlImg = null) => {
  let sizeCanvas = canvas.getAttribute('width');
  sizeCanvas === '0' || sizeCanvas === null ? sizeCanvas = 150 : sizeCanvas
  canvas.setAttribute('height', sizeCanvas)
  canvas.setAttribute('width', sizeCanvas)
  let ctx;
  if (canvas.getContext) {
    ctx = canvas.getContext('2d') ;
    // On verifie si l'image existe
    checkImageExists(urlImg, (imageExit) => {
      if (imageExit) {
        // création du badge avec image
        const img = new Image();
        img.addEventListener('load', () => {
          const newWidth = img.width * sizeCanvas / img.height
          ctx.drawImage(img, 0, 0, newWidth, sizeCanvas);
          ctx.globalCompositeOperation='destination-in';
          ctx.beginPath();
          ctx.arc(sizeCanvas/2, sizeCanvas/2, sizeCanvas/2, 0, 2 * Math.PI);
          ctx.fill();
          ctx.globalCompositeOperation='source-over';
        })
        img.src = urlImg;
      } else {
        // création du badge avec texte
        ctx.fillStyle = 'grey'
        ctx.beginPath();
        ctx.arc(sizeCanvas/2, sizeCanvas/2, sizeCanvas/2, 0, 2 * Math.PI);
        ctx.fill();

        const sizeText = sizeCanvas / 2
        ctx.fillStyle = 'orange'
        ctx.textAlign = 'center'
        ctx.textBaseline = 'middle';
        ctx.font = `${sizeText}px Arial`;
        // On ajuste la position du texte par rapport à la taille du canvas
        ctx.fillText('JD', sizeCanvas/2, sizeCanvas/2);
      }
    })
  }
}

const canvas1 = document.getElementById('badgeWithImage');
const img = "https://fotomelia.com/wp-content/uploads/edd/2015/05/main-fleur-paquerette-ciel-bleu-images-photos-gratuites.jpg";
```

```
createBadge(canvas1, img);

const canvas2 = document.getElementById('badgeWithText');
createBadge(canvas2);

</script>
</html>
```