API HTML 5

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

API HTML 5

API : **A**pplication **P**rogramming **I**nterfaces

- des composants permettant aux développeurs d'utiliser des fonctionnalités complexe d'une manière facile.
- ensemble de programmes complexes offrant au développeur une syntaxe plus facile
- permettant aussi l'interaction entre des programmes écrits avec des langages (de programmation) différents

API HTML 5

API : **A**pplication **P**rogramming **I**nterfaces

- des composants permettant aux développeurs d'utiliser des fonctionnalités complexe d'une manière facile.
- ensemble de programmes complexes offrant au développeur une syntaxe plus facile
- permettant aussi l'interaction entre des programmes écrits avec des langages (de programmation) différents

On dit que les programmes, qui utilisent des API, **consomment** ces API.

API Web

API Web

API accessible via les technologies Web

API Web

API Web

API accessible via les technologies Web

API REST

- **API Web** utilisant le protocole **HTTP** (**H**ypertext **T**ransfer **P**rotocol) et ses méthodes (`GET`, `POST`, `DELETE`...)
- Basée sur l'architecture client/serveur (requête/réponse)
- Fournissant des réponses dans plusieurs formats (**JSON**, **XML**, **CSV**...)

API HTML 5

Exemple d'**API Web**

- **API Google Maps** : pour afficher des cartes sur une page web
- **API YouTube** : pour afficher des vidéos sur une page web
- **API Facebook** : pour se connecter ou s'inscrire dans un système en utilisant un compte **Facebook**
- **API Weather Underground** : pour récupérer des données sur la météo selon des critères (ville, date...)
- **API Instagram** : pour accéder aux comptes utilisateurs, photos...
- ...

API HTML 5

Pour consommer une **API Web**, il faut connaître

- son adresse
- son mode de fonctionnement (valeur de retour, paramètres...)
- ...

API HTML 5

Pour consommer une **API Web**, il faut connaître

- son adresse
- son mode de fonctionnement (valeur de retour, paramètres...)
- ...

API HTML 5 ou **API HTML 5 JavaScript**

- balises **HTML 5** et/ou fonctions **JavaScript**
- simplifiant la création des pages Web
- pouvant utiliser des **API Web**

API HTML 5

Exemple d'**API HTML 5**

- Geolocation
- Canvas
- SVG
- Drag and Drop
- Web Storage
- Forms API
- ...

API HTML 5

Canvas

- une balise **HTML 5** (non-orpheline)
- manipulable avec des fonctions **JavaScript** spécifiques
- permettant de définir une zone graphique et de générer dans cette dernière des formes (cercles, rectangles...), images ou textes
- utilisable sur plusieurs navigateurs tels que **Google Chrome**, **Mozilla Firefox**, **Safari**...
- produisant des images au format matriciel (bitmap)
- ne figurant pas dans le **DOM**

Considérons la page `index.html` contenant un canvas

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Canvas</title>
</head>

<body>
    <canvas id="myCanvas" width="400" height="400" style="background-
      color:pink">

    </canvas>
    <script src="script.js"></script>
</body>

</html>
```

Considérons la page `index.html` contenant un canvas

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Canvas</title>
</head>

<body>
    <canvas id="myCanvas" width="400" height="400" style="background-
      color:pink">

    </canvas>
    <script src="script.js"></script>
</body>

</html>
```

On a attribué une couleur rose au canvas pour définir ses limites.

API HTML 5

On peut définir un message par défaut à afficher si le navigateur ne supporte pas les canvas

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Canvas</title>
</head>

<body>
    <canvas id="myCanvas" width="400" height="400" style="background-
      color:pink">
        Votre navigateur ne supporte pas la balise &lt;canvas&gt;.
    </canvas>
    <script src="script.js"></script>
</body>

</html>
```

API HTML 5

Comment utiliser ce canvas ?

Avec JavaScript,

- on récupère le canvas
- on définit le contexte
- on commence à dessiner

API HTML 5

Pour récupérer le canvas (dans `script.js`)

```
var canvas = document.getElementById('myCanvas');
```

API HTML 5

Pour récupérer le canvas (dans `script.js`)

```
var canvas = document.getElementById('myCanvas');
```

Définir un contexte 2D

```
var contexte = canvas.getContext('2d');
```

API HTML 5

Pour récupérer le canvas (dans `script.js`)

```
var canvas = document.getElementById('myCanvas');
```

Définir un contexte 2D

```
var contexte = canvas.getContext('2d');
```

Attribuer une couleur au contexte, sinon par défaut il sera noir

```
contexte.fillStyle = 'blue';
```

API HTML 5

Pour récupérer le canvas (dans `script.js`)

```
var canvas = document.getElementById('myCanvas');
```

Définir un contexte 2D

```
var contexte = canvas.getContext('2d');
```

Attribuer une couleur au contexte, sinon par défaut il sera noir

```
contexte.fillStyle = 'blue';
```

Dessiner avec `fillRect(abs, ord, largeur, hauteur)`

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

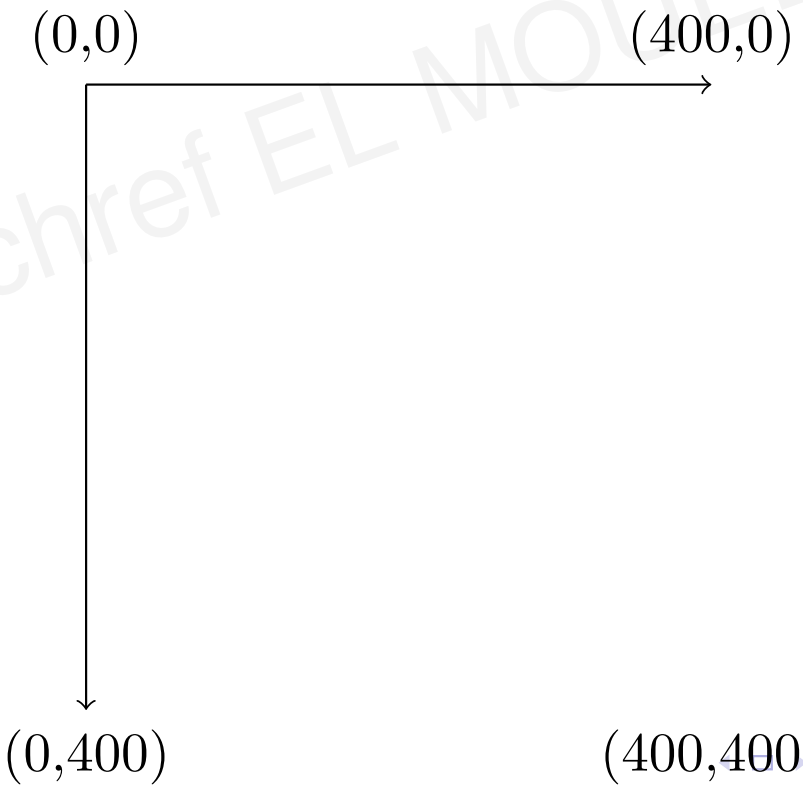
Deux formes de rectangles

- Les rectangles pleins (**fill**)
- Les rectangles vides (**stroke**)

Deux formes de rectangles

- Les rectangles pleins (**fill**)
- Les rectangles vides (**stroke**)

Le repère qu'on a défini



API HTML 5

Pour les couleurs, on peut utiliser

- Le nom en anglais
- Le code en hexadécimal (`http://colrd.com/sphere/`)
- Le code en rgb
- Le code en rgba

API HTML 5

On peut aussi utiliser la méthode `clearRect()` pour effacer le rectangle dont les données sont passées en paramètre

```
contexte.fillStyle = 'blue';
contexte.fillRect(10, 10, 100, 100);
contexte.clearRect(40, 40, 60, 60);
```

API HTML 5

Exercice

1. Ajouter un bouton associé à une fonction qui permet de déplacer le carré bleu à droite (sans qu'il quitte le canvas (la zone rose)
2. Ajouter ensuite un bouton pour déplacer le carré à gauche, un pour le déplacer vers le haut et un dernier vers le bas (bien sûr sans quitter le canvas)
3. Changer la couleur du carrée après chaque déplacement

API HTML 5

Pour dessiner une ou plusieurs lignes, il faut

- démarrer le chemin
- préciser la position de départ
- préciser la position d'arrivé
- refaire l'étape précédente si on veut dessiner d'autres lignes
- dessiner en choisissant le motif (plein ou vide)
- fermer le chemin

API HTML 5

On démarre le chemin

```
contexte.beginPath();
```

API HTML 5

On démarre le chemin

```
contexte.beginPath();
```

On se place sur la position de départ

```
contexte.moveTo(10, 10); // sinon par défaut (0,0)
```

API HTML 5

On démarre le chemin

```
contexte.beginPath();
```

On se place sur la position de départ

```
contexte.moveTo(10, 10); // sinon par défaut (0,0)
```

On précise la position d'arrivée

```
contexte.lineTo(210, 10);
```

API HTML 5

On démarre le chemin

```
contexte.beginPath();
```

On se place sur la position de départ

```
contexte.moveTo(10, 10); // sinon par défaut (0,0)
```

On précise la position d'arrivée

```
contexte.lineTo(210, 10);
```

On trace seulement les lignes

```
contexte.stroke();
```

API HTML 5

On démarre le chemin

```
contexte.beginPath();
```

On se place sur la position de départ

```
contexte.moveTo(10, 10); // sinon par défaut (0,0)
```

On précise la position d'arrivée

```
contexte.lineTo(210, 10);
```

On trace seulement les lignes

```
contexte.stroke();
```

On déclare la fin

```
contexte.closePath();
```

API HTML 5

Pour dessiner un triangle

```
contexte.beginPath();
contexte.moveTo(10, 10);
contexte.lineTo(210, 10);
contexte.lineTo(110, 110);
contexte.lineTo(10, 10);
contexte.stroke();
contexte.closePath();
```

API HTML 5

Pour dessiner un triangle

```
contexte.beginPath();
contexte.moveTo(10, 10);
contexte.lineTo(210, 10);
contexte.lineTo(110, 110);
contexte.lineTo(10, 10);
contexte.stroke();
contexte.closePath();
```


Si on veut que le triangle soit plein, il faut remplacer `contexte.stroke()` par `contexte.fill()`

API HTML 5

Autres méthodes pour les lignes

- `setLineDash([tableau])` : définit le motif de tiret utilisé pour tracer des lignes (tableau : spécifie la longueur d'un tiret et l'espace juste après)
  Exemple : avec `contexte.setLineDash([5, 20, 15, 5])` on aura un premier tiret de longueur 5, un espace de longueur 20, un tiret de longueur 15 et un espace de longueur 5.
- `lineWidth` : précise l'épaisseur de la ligne
- `lineCap` : détermine la forme utilisée pour tracer les extrémités des lignes (valeurs possibles : round (arrondi), square (carré) ou butt (par défaut))
- ...

API HTML 5

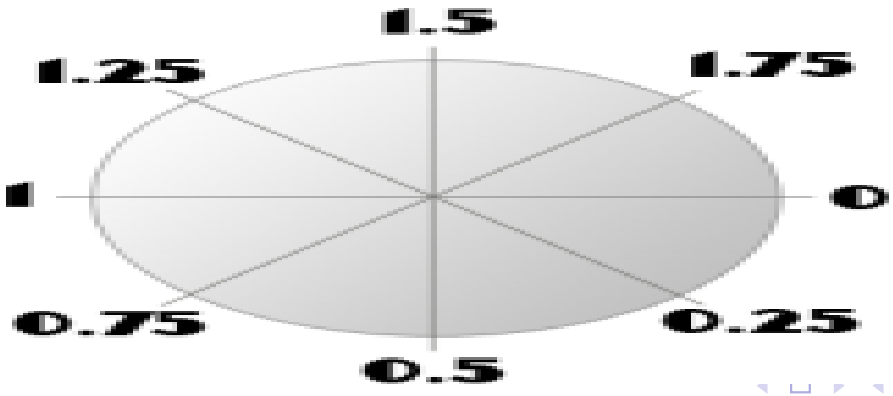
Pour dessiner (ou une portion d') un cercle, il faut préciser

- les coordonnées (abscisse et ordonné) du centre
- le rayon
- angle de départ
- angle d'arrivé
- le sens (false : [par défaut] le sens des aiguilles d'une montre)

API HTML 5

Pour dessiner (ou une portion d') un cercle, il faut préciser

- les coordonnées (abscisse et ordonné) du centre
- le rayon
- angle de départ
- angle d'arrivé
- le sens (false : [par défaut] le sens des aiguilles d'une montre)



API HTML 5

Exemples de trois arcs

```
contexte.beginPath();
contexte.arc(100, 100, 50, 0, Math.PI, true);
contexte.fill();
contexte.closePath();

contexte.beginPath();
contexte.arc(200, 200, 50, Math.PI*1.5, Math.PI*0.5, true);
contexte.fill();
contexte.closePath();

contexte.beginPath();
contexte.arc(300, 300, 50, Math.PI*0.5, 0);
contexte.fill();
contexte.closePath();
```

API HTML 5

On peut aussi dessiner un arc entre deux points (10,10) et (210,210) par rapport au point (210,10). Le dernier paramètre 180 est l'angle de l'arc

```
contexte.beginPath();
contexte.moveTo(10, 10);
contexte.arcTo(210, 10, 210, 210, 180);
contexte.fill();
contexte.closePath();
```

API HTML 5

On peut aussi dessiner une ellipse

```
contexte.beginPath();
contexte.ellipse(200, 200, 50, 100, 0, 0, 2 * Math.PI);
contexte.stroke();
contexte.closePath();
```

API HTML 5

On peut aussi dessiner une ellipse

```
contexte.beginPath();
contexte.ellipse(200, 200, 50, 100, 0, 0, 2 * Math.PI);
contexte.stroke();
contexte.closePath();
```

Explication

- (200, 200) : coordonnées du centre
- 50 : rayon horizontal
- 100 : rayon vertical
- 0 : angle de rotation de l'ellipse
- (0, 2 * Math.PI) : angles de départ et d'arrivé
- comme pour les arcs, on peut aussi ajouter le sens

API HTML 5

Pour afficher une image dans un canvas, il faut

- utiliser un objet de type `Iamge`
- préciser la source de l'image
- dessiner l'image dans le contexte

API HTML 5

Instancier un objet de type `Iamge`

```
var myImage = new Image();
```

API HTML 5

Instancier un objet de type `Iamge`

```
var myImage = new Image();
```

Préciser la source de l'image

```
myImage.src = "http://www.lsis.org/elmouelhia/images/avatar.jpg";
```

API HTML 5

Instancier un objet de type `Iamge`

```
var myImage = new Image();
```

Préciser la source de l'image

```
myImage.src = "http://www.lsis.org/elmouelhia/images/avatar.jpg";
```

Dessiner l'image dans le contexte (avec sa taille réelle)

```
contexte.drawImage(myImage, 10, 10);
// (10, 10) les coordonnées du premier point de l'image
```

API HTML 5

Pour afficher l'image avec une taille personnalisée

```
var myImage = new Image();
myImage.src = "http://www.lsis.org/elmouelhia/images/avatar.jpg";
contexte.drawImage(myImage, 10, 10, 300, 200);
// 300 : largeur
// 200 : hauteur
```

API HTML 5

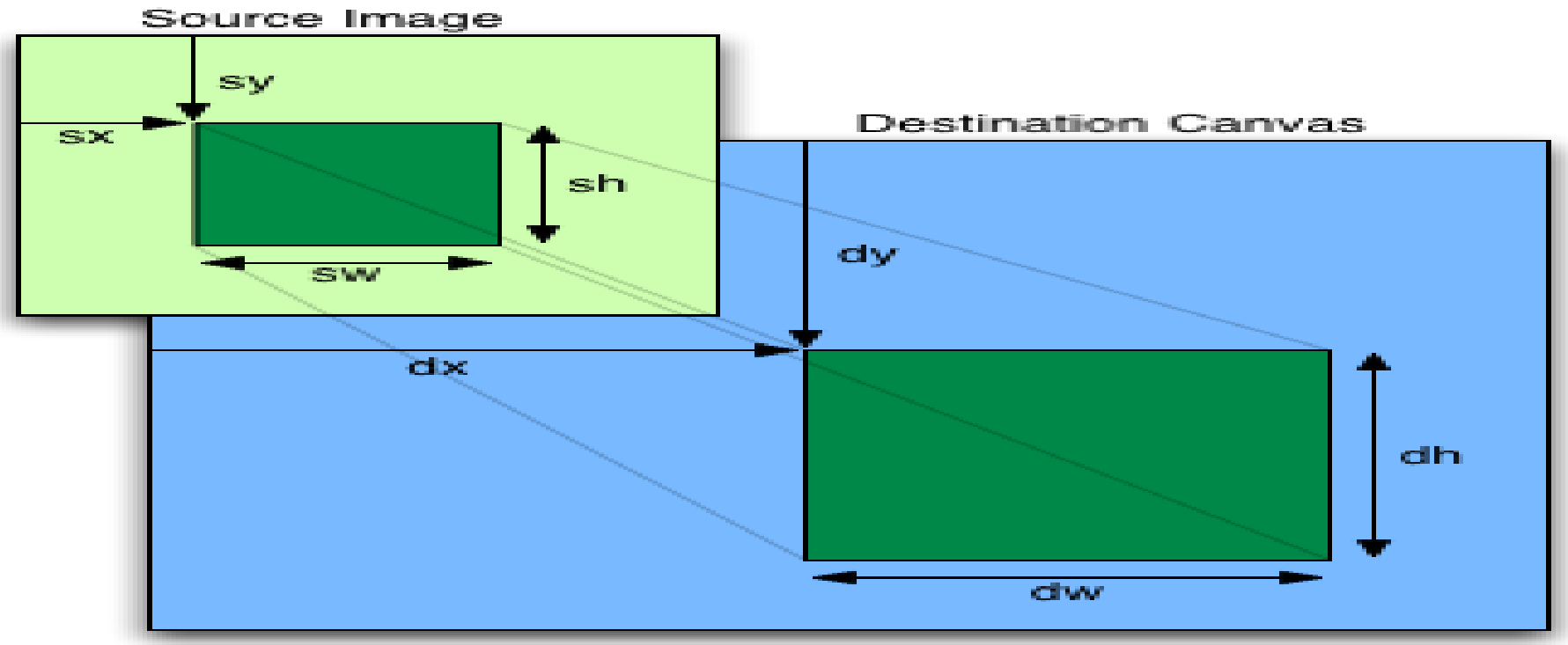
Les trois surcharges de `drawImage` (source whatwg)

```
context.drawImage(image, dx, dy)
context.drawImage(image, dx, dy, dw, dh)
context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)
```

API HTML 5

Les trois surcharges de `drawImage` (source whatwg)

```
context.drawImage(image, dx, dy)
context.drawImage(image, dx, dy, dw, dh)
context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)
```



API HTML 5

On peut aussi créer un apttern et lui associer une image pour la répéter

```
var myImage = new Image();
myImage.src = "http://www.lsis.org/elmouelhia/images/avatar.jpg";
var pattern = contexte.createPattern(myImage, 'repeat');
// repeat-x permet de le répéter horizontalement
// repeat-y permet de le répéter horizontalement
// no-repeat permet de ne pas répéter
contexte.fillStyle = pattern;
contexte.fillRect(0, 0, canvas.height, canvas.width);
```

API HTML 5

Pour dessiner du texte dans un canvas, il faut

- préciser le font
- ajouter le texte

API HTML 5

Pour dessiner du texte dans un canvas, il faut

- préciser le font
- ajouter le texte

Exemple

```
contexte.font = "40px Arial";
// 40 : taille du texte
// Arial : nom de la police

contexte.strokeText("Hello World", 50, 50);
// on peut aussi utiliser fillText
```

API HTML 5

Il est possible de modifier la couleur et l'alignement du texte

```
contexte.font = "bold italic 40px Comic Sans MS";
contexte.strokeStyle = "blue";

// alignement horizontal
contexte.textAlign = "center";

// alignement vertical
contexte.textBaseline = "bottom";

contexte.strokeText("Hello World", 200, 50);
```

API HTML 5

Avec la méthode `translate()`, l'origine du repère est modifiée. Elle est maintenant au point (100, 100).

```
contexte.translate(100, 100);
contexte.fillStyle = 'blue';
contexte.fillRect(0, 0, 100, 100);
```

API HTML 5

Le point d'origine du canvas n'est plus accessible car le point (0,0) est ce qui était précédemment (100, 100), pour récupérer l'origine (0,0) du canvas, il faut utiliser les méthodes `save()` et `restore()`

```
contexte.save();
contexte.translate(100, 100);
contexte.fillStyle = 'blue';
contexte.fillRect(0, 0, 100, 100);
contexte.restore();
contexte.fillStyle = 'gold';
contexte.fillRect(0, 0, 100, 100);
```

API HTML 5

Le point d'origine du canvas n'est plus accessible car le point (0,0) est ce qui était précédemment (100, 100), pour récupérer l'origine (0,0) du canvas, il faut utiliser les méthodes `save()` et `restore()`

```
contexte.save();
contexte.translate(100, 100);
contexte.fillStyle = 'blue';
contexte.fillRect(0, 0, 100, 100);
contexte.restore();
contexte.fillStyle = 'gold';
contexte.fillRect(0, 0, 100, 100);
```

En supprimant les deux méthodes `save()` et `restore()`, le rectangle doré sera dessiné au-dessus du bleu.

API HTML 5

Avec la méthode `rotate()`, on fait la rotation en précisant l'angle de rotation

```
contexte.fillStyle = 'olive';
contexte.rotate(Math.PI/12);
contexte.fillRect(50, 50, 100, 100);
```

API HTML 5

Avec la méthode `rotate()`, on fait la rotation en précisant l'angle de rotation

```
contexte.fillStyle = 'olive';
contexte.rotate(Math.PI/12);
contexte.fillRect(50, 50, 100, 100);
```

On peut combiner rotation et translation

```
contexte.translate(100,100);
contexte.fillStyle = 'olive';
contexte.rotate(Math.PI/12);
contexte.fillRect(50, 50, 100, 100);
```

API HTML 5

SVG : **S**calable **V**ector **G**raphics

- format d'images vectorielles : pouvant être redimensionnées sans perte de qualité
- normalisée par **W3C** depuis 1999
- basé sur le format **XML** (balisage)
- S'intégrant facilement dans un document **HTML 5** et figurant dans le **DOM**

API HTML 5

Créons un fichier `svg.xml`

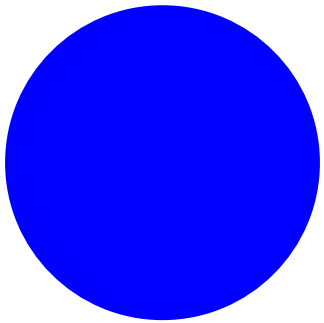
```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="
   400" height="400">
  <circle cx="50" cy="50" r="40" fill="blue" />
  <text x="10" y="120">SVG dans un fichier XML</text>
</svg>
```

API HTML 5

Créons un fichier `svg.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="
   400" height="400">
  <circle cx="50" cy="50" r="40" fill="blue" />
  <text x="10" y="120">SVG dans un fichier XML</text>
</svg>
```

En ouvrant `svg.xml` **dans un navigateur (par exemple), voici le résultat affiché**



SVG dans un fichier XML

API HTML 5

Le corps du fichier `svg.xml` peut être intégré dans un fichier HTML (pas besoin de préciser le `namespace`

```
<!DOCTYPE html>
<html>
<head>
  <title>SVG</title>
</head>
<body>
  <svg width="400" height="400">
    <circle cx="50" cy="50" r="40" fill="blue" />
    <text x="10" y="120">SVG dans un fichier XML</text>
  </svg>
</body>
</html>
```

API HTML 5

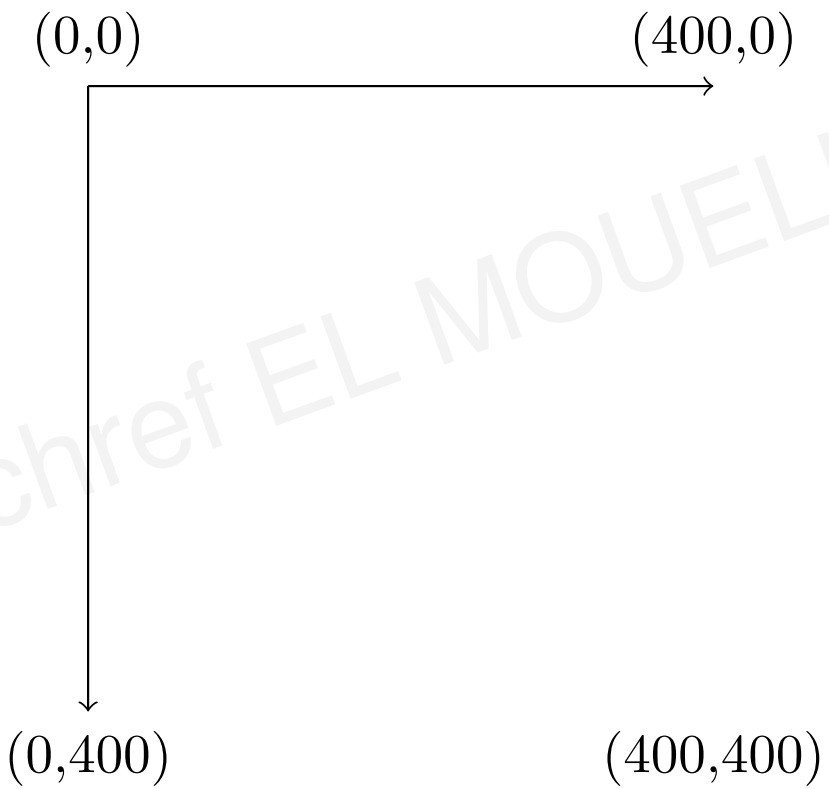
Le corps du fichier `svg.xml` peut être intégré dans un fichier HTML (pas besoin de préciser le `namespace`

```
<!DOCTYPE html>
<html>
<head>
  <title>SVG</title>
</head>
<body>
  <svg width="400" height="400">
    <circle cx="50" cy="50" r="40" fill="blue" />
    <text x="10" y="120">SVG dans un fichier XML</text>
  </svg>
</body>
</html>
```

Le résultat affiché est le même.

API HTML 5

Le repère qu'on a défini (comme pour les canvas)



API HTML 5

Une ligne a un point de départ et un point d'arrivé

- Si les coordonnées ne sont pas précisées, par défaut c'est (0, 0)

API HTML 5

Une ligne a un point de départ et un point d'arrivé

- Si les coordonnées ne sont pas précisées, par défaut c'est (0, 0)

On peut aussi préciser

- la couleur du contour [par défaut noir]
- l'épaisseur du contour [par défaut 1]

API HTML 5

Les trois codes suivants permettent de dessiner la même ligne

```
<svg width="400" height="400">
  <line x1="0" y1="0" x2="100" y2="100" stroke="red"
     stroke-width="2" />
</svg>
```

API HTML 5

Les trois codes suivants permettent de dessiner la même ligne

```
<svg width="400" height="400">
  <line x1="0" y1="0" x2="100" y2="100" stroke="red"
    stroke-width="2" />
</svg>
```

```
<svg width="400" height="400">
  <line x2="100" y2="100" stroke="red" stroke-width="2"
    />
</svg>
```

API HTML 5

Les trois codes suivants permettent de dessiner la même ligne

```
<svg width="400" height="400">
  <line x1="0" y1="0" x2="100" y2="100" stroke="red"
    stroke-width="2" />
</svg>
```

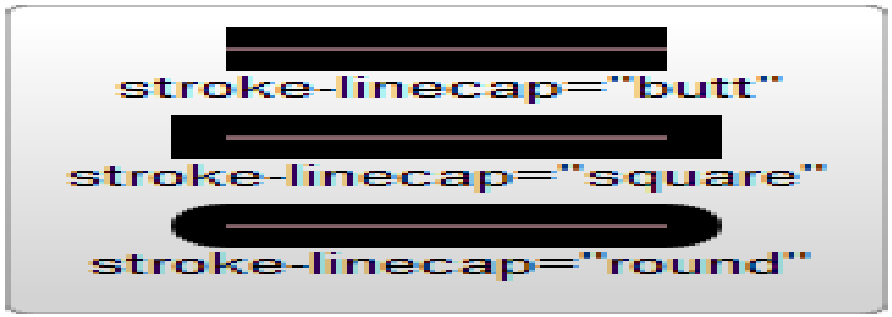
```
<svg width="400" height="400">
  <line x2="100" y2="100" stroke="red" stroke-width="2"
    />
</svg>
```

```
<svg width="400" height="400">
  <line x1="100" y1="100" stroke="red" stroke-width="2"
    />
</svg>
```

API HTML 5

Valeurs de la propriété `stroke-linecap`

- `butt` (valeur par défaut) ferme la ligne avec un angle droit à l'endroit où la ligne se termine.
- `square` termine au delà de la ligne avec un angle droit. La distance ajoutée est la moitié de `stroke-width`.
- `round` ajoute un effet arrondi au dernier trait de la ligne. Le rayon du cercle dépend de `stroke-width`.



API HTML 5

Exemple (les deux dernières lignes permettent de voir le dépassement) de lignes avec les valeurs `round` et `square`)

```
<svg width="400" height="400">

  <line x1="10" y1="10" x2="10" y2="110" stroke="red" stroke-
    width="5" stroke-linecap="butt" />

  <line x1="110" y1="10" x2="110" y2="110" stroke="red" stroke-
    width="5" stroke-linecap="square" />

  <line x1="210" y1="10" x2="210" y2="110" stroke="red" stroke-
    width="5" stroke-linecap="round" />

  <line x1="0" y1="10" x2="220" y2="10" stroke="black" />

  <line x1="0" y1="110" x2="220" y2="110" stroke="black" />
</svg>
```

API HTML 5

La propriété `stroke-dasharray`

- permet de définir comment présenter une ligne non continue
- utilise un tableau de valeurs : la longueur du trait et la longueur de l'espace entre deux traits
- peut aussi être utilisée pour les autres formes graphiques comme cercle, rectangle...

API HTML 5

Exemple

```
<svg width="400" height="400">

  <line x1="10" y1="10" x2="10" y2="310" stroke="red"  stroke-
    dasharray="30, 10"  />

  <line x1="110" y1="10" x2="110" y2="310" stroke="red" stroke-
    dasharray="30, 10, 50" />

  <line x1="210" y1="10" x2="210" y2="310" stroke="red"  stroke
    -dasharray="30, 10, 50, 30" />
</svg>
```

Remarque

Si le nombre d'éléments du tableau est impair (comme pour la deuxième ligne de notre exemple), alors ce dernier sera répartie afin de créer un nombre pair de valeurs par répétition. `"30, 10, 50"` $\Rightarrow$ `"30, 10, 50, 30, 10, 50"`

API HTML 5

Polyline (ligne brisée)

lignes entre plusieurs points (généralement utilisée pour les tracés ouverts)

API HTML 5

Polyline (ligne brisée)

lignes entre plusieurs points (généralement utilisée pour les tracés ouverts)

Exemple de polyline

```
<svg width="400" height="400">
  <polyline points="0 0,  30 20, 100 20, 100 300"
    fill="none" stroke="black"  />
</svg>
```

API HTML 5

Polygone

polyline avec deux extrémités connectées

API HTML 5

Polygone

polyline avec deux extrémités connectées

Exemple de polygone

```
<svg width="400" height="400">
  <polygon points="0 0,  30 20, 100 20, 100 300"
    fill="none" stroke="black"  />
</svg>
```

API HTML 5

Pour dessiner un cercle

il faut préciser le rayon

API HTML 5

Pour dessiner un cercle

il faut préciser le rayon

On peut aussi préciser

- les coordonnées du centre [par défaut (0, 0)]
- la couleur du fond [par défaut noir]
- la couleur du contour [par défaut `none`]
- l'épaisseur du contour [par défaut 1]
- l'opacité prenant une valeur entre 0 et 1 (`stroke-opacity` et `fill-opacity`)

API HTML 5

Exemple

```
<svg width="400" height="400">
  <circle cx="50" cy="50" r="40" stroke="red" stroke-width="2" fill="
    blue" />
</svg>
```

API HTML 5

Exemple

```
<svg width="400" height="400">
  <circle cx="50" cy="50" r="40" stroke="red" stroke-width="2" fill="
    blue" />
</svg>
```

Explication

- les coordonnées du centre : (50, 50)
- la couleur du fond : bleu
- la couleur du contour : rouge
- l'épaisseur du contour 2

API HTML 5

Exemple

```
<svg width="400" height="400">
  <circle cx="50" cy="50" r="40" stroke="red" stroke-width="2" fill="
    blue" />
</svg>
```

Explication

- les coordonnées du centre : (50, 50)
- la couleur du fond : bleu
- la couleur du contour : rouge
- l'épaisseur du contour 2

Ce code est équivalent au précédent

```
<svg width="400" height="400">
  <circle cx="50" cy="50" r="40"  style="fill:blue;stroke:red;stroke-
    width:2"/>
</svg>
```

API HTML 5

Pour les ellipse, ce qui change par rapport aux cercles

- rayon vertical (`ry`)
- rayon horizontal (`rx`)

API HTML 5

Pour les ellipse, ce qui change par rapport aux cercles

- rayon vertical (`ry`)
- rayon horizontal (`rx`)

Exemple

```
<svg width="400" height="400">
  <ellipse cx="100" cy="100" rx="70" ry="30"  fill="
    blue" />
</svg>
```

API HTML 5

Pour dessiner un rectangle

il faut préciser ses longueur et largeur

API HTML 5

Pour dessiner un rectangle

il faut préciser ses longueur et largeur

On peut aussi préciser

- les coordonnées du centre (sommet en haut à gauche) [par défaut (0, 0)]
- la couleur du fond [par défaut noir]
- la couleur du contour [par défaut `none`]
- l'épaisseur du contour [par défaut 1]

API HTML 5

Exemple

```
<svg width="400" height="400">
  <rect x="20" y="20" width="250" height="100" stroke="red" stroke-
    width="2" fill="blue"  />
</svg>
```

API HTML 5

Exemple

```
<svg width="400" height="400">
  <rect x="20" y="20" width="250" height="100" stroke="red" stroke-
     width="2" fill="blue"  />
</svg>
```

Explication

- les coordonnées du centre : (20, 20)
- la largeur : 250
- la hauteur : 100
- la couleur du fond : bleu
- la couleur du contour : rouge
- l'épaisseur du contour 2

API HTML 5

Autres attributs

- l'opacité prenant une valeur entre 0 et 1 (`stroke-opacity` et `fill-opacity`)
- `rx` étant le rayon horizontal pour arrondir les coins du rectangle
- `ry` étant le rayon vertical pour arrondir les coins du rectangle

API HTML 5

Chemin (`path`)

- Permettant de créer des lignes lignes, courbes, arcs, formes graphiques...
- Ayant un paramètre `d` contenant l'ensemble de commandes à appliquer
- Chaque commande commence par une lettre indiquant le type d'opération suivi de paramètres de cette commande
  - Si la lettre est écrite en minuscule, les coordonnées qui suivent sont relatives
  - Sinon les coordonnées sont absolues

API HTML 5

Quelques commandes

- `M x y` : MoveTo
- `L x y` : LineTo ou
  - `H x` (ligne horizontale) ou
  - `V x` (ligne verticale)
- `Z` : ClosePath (lier le dernier point au premier, pas de différence entre `z` et `Z`)

API HTML 5

Exemple : dessiner un rectangle avec les quatre points (10,10), (90,10), (90,90) et (10,90)

```
<svg width="400" height="400">
  <path d="M 10 10 H 90 V 90 H 10 L 10 10"/>
</svg>
```

API HTML 5

Exemple : dessiner un rectangle avec les quatre points (10,10), (90,10), (90,90) et (10,90)

```
<svg width="400" height="400">
  <path d="M 10 10 H 90 V 90 H 10 L 10 10"/>
</svg>
```

Ou avec des données relatives

```
<path d="m 10 10 h 80 v 80 h -80 v -80"/>
```

API HTML 5

Exemple : dessiner un rectangle avec les quatre points (10,10), (90,10), (90,90) et (10,90)

```
<svg width="400" height="400">
  <path d="M 10 10 H 90 V 90 H 10 L 10 10"/>
</svg>
```

Ou avec des données relatives

```
<path d="m 10 10 h 80 v 80 h -80 v -80"/>
```

Ou en utilisant la commande Z

```
<path d="m 10 10 h 80 v 80 h -80 z"/>
```

API HTML 5

Pour dessiner une courbe de Bézier cubique, il faut préciser

- la commande `C` ou `c`
- les coordonnées de deux points de contrôle
- les coordonnées du dernier point de la courbe

API HTML 5

Pour dessiner une courbe de Bézier cubique, il faut préciser

- la commande `C` ou `c`
- les coordonnées de deux points de contrôle
- les coordonnées du dernier point de la courbe

Exemple (les cercles rouges sont visualisés pour montrer les quatre points indispensables pour dessiner la courbe)

```
<svg width="400" height="400">
  <circle cx="10" cy="10" r="2" fill="red"/>
  <circle cx="50" cy="50" r="2" fill="red"/>
  <circle cx="150" cy="50" r="2" fill="red"/>
  <circle cx="200" cy="10" r="2" fill="red"/>
  <path d="M10 10 C 50 50, 150 50, 200 10" stroke="black" fill=
    "none"/>
</svg>
```

API HTML 5

Pour dessiner une courbe de Bézier quadratique, il faut préciser

- la commande `Q` ou `q`
- les coordonnées d'un seul point de contrôle
- les coordonnées du dernier point de la courbe

API HTML 5

Pour dessiner une courbe de Bézier quadratique, il faut préciser

- la commande `Q` ou `q`
- les coordonnées d'un seul point de contrôle
- les coordonnées du dernier point de la courbe

Exemple (les cercles rouges sont visualisés pour montrer les trois points indispensables pour dessiner la courbe)

```
<svg width="400" height="400">
   <circle cx="10" cy="10" r="2" fill="red"/>
   <circle cx="100" cy="100" r="2" fill="red"/>
   <circle cx="200" cy="10" r="2" fill="red"/>
  <path d="M10 10 Q 100 100, 200 10" stroke="black" fill="none"
    />
</svg>
```

API HTML 5

Pour écrire un texte

- il faut préciser son emplacement
- On peut aussi préciser la couleur, le sens de rotation...

API HTML 5

Pour écrire un texte

- il faut préciser son emplacement
- On peut aussi préciser la couleur, le sens de rotation...

Exemple (affiché en noir : couleur par défaut)

```
<svg width="400" height="400">
  <text x="50" y="50" > John Wick </text>
</svg>
```

API HTML 5

Pour écrire un texte

- il faut préciser son emplacement
- On peut aussi préciser la couleur, le sens de rotation...

Exemple (affiché en noir : couleur par défaut)

```
<svg width="400" height="400">
  <text x="50" y="50" > John Wick </text>
</svg>
```

On peut marquer une partie du texte avec la balise `tspan` en utilisant les propriétés CSS

```
<svg width="400" height="400">
  <text x="50" y="50" >
    <tspan font-weight="bold" fill="red">
      Ce n'est pas moi
    </tspan>
    John Wick
  </text>
</svg>
```

API HTML 5

Il est possible de définir un path (ici une courbe de Bézier) et l'utiliser pour écrire le texte

```
<svg width="400" height="400">
  <path id="chemin" d="M 20,20 C 80,60 100,40 120,20
     " fill="none" />
  <text>
    <textPath href="#chemin">
      Hello John Wick
    </textPath>
  </text>
</svg>
```

API HTML 5

Pour définir un texte cliquable

```
<svg width="400" height="400">
  <a xlink:href="http://www.lsis.org/elmouelhia/"
     target="_blank">
    <text x="50" y="50" fill="red">
      Visiter ma page
    </text>
  </a>
</svg>
```

Étan donné le rectangle suivant

```
<svg width="400" height="500" style="background-color:#bff;">
  <rect x="0" y="0" width="10" height="10"  />
</svg>
```

Étan donné le rectangle suivant

```
<svg width="400" height="500" style="background-color:#bff;">
  <rect x="0" y="0" width="10" height="10"  />
</svg>
```

Pour appliquer une translation, on ajoute l'attribut `transform="translate(x,y)"`

```
<svg width="400" height="400" style="background-color:#bff;">
  <rect width="10" height="10" transform="translate(100,100)" />
</svg>
```

Étan donné le rectangle suivant

```
<svg width="400" height="500" style="background-color:#bff;">
  <rect x="0" y="0" width="10" height="10"  />
</svg>
```

Pour appliquer une translation, on ajoute l'attribut `transform="translate(x,y)"`

```
<svg width="400" height="400" style="background-color:#bff;">
  <rect width="10" height="10" transform="translate(100,100)" />
</svg>
```

Pour appliquer une rotation, on ajoute l'attribut `transform="rotate(a)"`

```
<svg width="400" height="400" style="background-color:#bff;">
  <rect width="10" height="10" transform="rotate(60)" />
</svg>
```

Étan donné le rectangle suivant

```
<svg width="400" height="500" style="background-color:#bff;">
  <rect x="0" y="0" width="10" height="10"  />
</svg>
```

Pour appliquer une translation, on ajoute l'attribut `transform="translate(x,y)"`

```
<svg width="400" height="400" style="background-color:#bff;">
  <rect width="10" height="10" transform="translate(100,100)" />
</svg>
```

Pour appliquer une rotation, on ajoute l'attribut `transform="rotate(a)"`

```
<svg width="400" height="400" style="background-color:#bff;">
  <rect width="10" height="10" transform="rotate(60)" />
</svg>
```

Il est possible de faire les deux

```
<svg width="400" height="400" style="background-color:#bff;">
  <rect  width="10" height="10" transform="translate(100,100) rotate
     (60)" />
</svg>
```

API HTML 5

Web History API

- API offrant la possibilité de consulter l'historique d'un visiteur
- Historique = **URL** visitées par le visiteur

API HTML 5

Considérons la page `index.html` suivante

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Historique</title>
</head>

<body>
    <h1>Historique</h1>
    <button onclick="retour()">Retour</button>
    <script src="script.js"></script>
</body>

</html>
```

En cliquant sur le bouton `retour`, on retourne à la dernière page visitée dans ce même onglet

```
function retour() {
    window.history.back();
}
```

En cliquant sur le bouton `retour`, on retourne à la dernière page visitée dans ce même onglet

```
function retour() {
    window.history.back();
}
```

Pour consulter l'avant dernière page de l'historique, on utilise `go`

```
function retour() {
    window.history.go(-2);
}
```

En cliquant sur le bouton `retour`, on retourne à la dernière page visitée dans ce même onglet

```
function retour() {
    window.history.back();
}
```

Pour consulter l'avant dernière page de l'historique, on utilise `go`

```
function retour() {
    window.history.go(-2);
}
```


Autres méthodes/attributs

- `length` : contenant le nombre d'éléments dans l'historique
- `forward` : pour aller à la page suivante dans l'historique

API HTML 5

Drag and drop (glisser-déposer)

- un concept permettant de déplacer des éléments d'un conteneur vers un autre
- existant depuis quelques temps dans les **OS** (déplacer un fichier en le glissant avec la souris d'un dossier vers un autre)
- existant dans le web bien avant le **HTML 5** mais très compliqué

Considérons la page `index.html`

```
<head>
  <title>Drag and drop</title>
  <style media="screen">
    #conteneur{
      display: flex;
      flex-wrap: wrap;
    }
    #conteneur>div{
      width:50%;
      height:100px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div id="conteneur">
    <div style="background-color:teal" id=draggable>Hello</div>
    <div style="background-color:gold" id=dropper>World</div>
    <div>draggable</div>
    <div>dropper</div>
  </div>
  <script src="script.js"></script>
</body>
```

API HTML 5

Objectif

Déplacer `Hello` dans la même div que `world`

API HTML 5

Objectif

Déplacer `Hello` dans la même div que `world`

Premier problème

Le `Hello` ne peut être déplacé

API HTML 5

Objectif

Déplacer `Hello` dans la même div que `world`

Premier problème

Le `Hello` ne peut être déplacé

Solution

Il faut activer la propriété `draggable` pour pouvoir le déplacer

API HTML 5

Activons le déplacement pour la première `div`

```
<div id="conteneur">
    <div style="background-color:teal" draggable="true" id=draggable>
        Hello
    </div>
    <div style="background-color:gold" id=dropper>World</div>
    <div>draggable</div>
    <div>dropper</div>
</div>
```

API HTML 5

Activons le déplacement pour la première `div`

```
<div id="conteneur">
    <div style="background-color:teal" draggable="true" id=draggable>
        Hello
    </div>
    <div style="background-color:gold" id=dropper>World</div>
    <div>draggable</div>
    <div>dropper</div>
</div>
```


Constats

- La `div` du `Hello` peut être déplacée
- En essayant de le déposer sur la deuxième `div`, un signe d'interdiction est affiché

API HTML 5

Pour arrêter cette interdiction, on ajoute un script JavaScript dans `script.js` pour annuler l'interdiction de drop

```
var draggable  = document.querySelector('#draggable');
var dropper  = document.querySelector('#dropper');

dropper.addEventListener('dragover', function(e) {
    e.preventDefault();
});
```

API HTML 5

Pour arrêter cette interdiction, on ajoute un script JavaScript dans `script.js` pour annuler l'interdiction de drop

```
var draggable  = document.querySelector('#draggable');
var dropper  = document.querySelector('#dropper');

dropper.addEventListener('dragover', function(e) {
    e.preventDefault();
});
```


Constats

- Le signe d'interdiction n'est plus
- `Hello` n'apparaît toujours pas dans la deuxième `div`

API HTML 5

Quelques évènements importants

- `dragstart` : se déclenche lorsqu'on commence le déplacement et possède un objet `dataTransfer` permettant de :
  - transmettre une chaîne de caractère à l'élément `dropper`
  - préciser le chemin d'une image à utiliser pendant le déplacement
  - ...
- `dragend` : signale à l'objet déplacé que son déplacement est terminé
- `dragover` : se déclenche lorsqu'un élément se déplace dans la zone `dropper`
- `dragenter` : se déclenche lorsqu'un élément entre dans la zone `dropper`
- `dragleave` : se déclenche lorsqu'un élément quitte la zone `dropper`
- `drop` : se déclenche lorsqu'un élément est déposé dans `dropper`

API HTML 5

Afin d'aider l'utilisateur à savoir qu'il est dans une zone de drop, on peut ajouter

```
dropper.addEventListener('dragenter', function() {
    dropper.style.backgroundColor = 'red';
});

dropper.addEventListener('dragleave', function() {
    dropper.style.backgroundColor = 'gold';
});
```

API HTML 5

Pour envoyer des données au démarrage du déplacement

```
draggable.addEventListener('dragstart', function(e) {
    e.dataTransfer.setData('text/plain', draggable.innerHTML);
});
```

API HTML 5

Pour envoyer des données au démarrage du déplacement

```
draggable.addEventListener('dragstart', function(e) {
    e.dataTransfer.setData('text/plain', draggable.innerHTML);
});
```

Pour récupérer les données envoyées

```
dropper.addEventListener('drop', function(e) {
  var texte = e.dataTransfer.getData('text/plain');
  dropper.innerHTML = texte + " " + dropper.innerHTML;
  dropper.style.backgroundColor = 'gold';
});
```

API HTML 5

Formats de données autorisés

- `text/plain` : représente des données textuelles brutes. C'est le format par défaut pour du texte simple.
- `text/html` : permet de transférer du contenu **HTML**. Par exemple, vous pouvez stocker du code **HTML** pour qu'il soit interprété lors du dépôt.
- `text/uri-list` : représente une liste de ressources, comme des **URL**. Cela peut être utilisé pour faire glisser des liens ou des fichiers.
- `application/json` : permet de transférer des données au format **JSON**, utile pour partager des objets ou des structures de données plus complexes.
- `application/xml` ou `text/xml` : pour le transfert de données **XML**.
- `Formats personnalisés` : permet de définir des types de données personnalisés. Par exemple, `application/x-custom-data`, où `x-custom-data` est un type de données spécifique à votre application.

API HTML 5

Lors du transfert de données, on peut aussi afficher une image

```
var img = new Image();
img.src = 'http://elmouelhia.free.fr/images/ok.png';

draggable.addEventListener('dragstart', function(e) {
  e.dataTransfer.setDragImage(img, 50, 50);
  // (50, 50) : position du curseur par rapport à l'image de
     taille 80px * 80px

  e.dataTransfer.setData('text/plain', draggable.innerHTML);
});
```

API HTML 5

Forms API

- une API facilitant la validation de formulaires **HTML 5**
- définissant de nouvelles méthodes (`checkValidity()`) et propriétés (`rangeOverflow`...) pour faciliter et accélérer la validation

Considérons la page `index.html` suivante

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Forms API</title>
</head>

<body>
    <div>
        Somme à envoyer (comprise entre 100 et 1 000) :
        <input id="somme" type="number" min="100" max="1000" required>
        <span id="msg"></span>
    </div>
    <button onclick="envoyer()">envoyer</button>
    <script src="script.js"></script>
</body>

</html>
```

API HTML 5

Deux contraintes définies

- `required` : une valeur doit être saisie
- `min` et `max` : un intervalle doit être respecté

API HTML 5

En cliquant sur le bouton `envoyer`, on vérifie si ces deux contraintes ont été respectées

```
function envoyer() {
  var somme = document.getElementById("somme");
  if (!somme.checkValidity()) {
    document.getElementById("msg").innerHTML = somme.validationMessage;
  }
}
```

API HTML 5

En cliquant sur le bouton `envoyer`, on vérifie si ces deux contraintes ont été respectées

```
function envoyer() {
  var somme = document.getElementById("somme");
  if (!somme.checkValidity()) {
    document.getElementById("msg").innerHTML = somme.validationMessage;
  }
}
```


Explication

- `checkValidity()` retourne `true` si les contraintes sont respectées
- `validationMessage` contiendra le message d'erreur si une contrainte est violée

API HTML 5

Il est aussi possible de vérifier contrainte par contrainte en utilisant les propriétés `valueMissing`, `rangeOverflow` **et** `rangeUnderflow`

```
function envoyer() {

  var somme = document.getElementById("somme");
  var msg = document.getElementById("msg");

  if (somme.validity.valueMissing) {
    msg.innerHTML = "une valeur doit être saisie";
    return;
  }
  if (somme.validity.rangeOverflow) {
    msg.innerHTML = somme.value + " est supérieure au max";
    return;
  }
  if (somme.validity.rangeUnderflow) {
    msg.innerHTML = somme.value + " est inférieure au min";
  }
}
```

API HTML 5

Autres propriétés

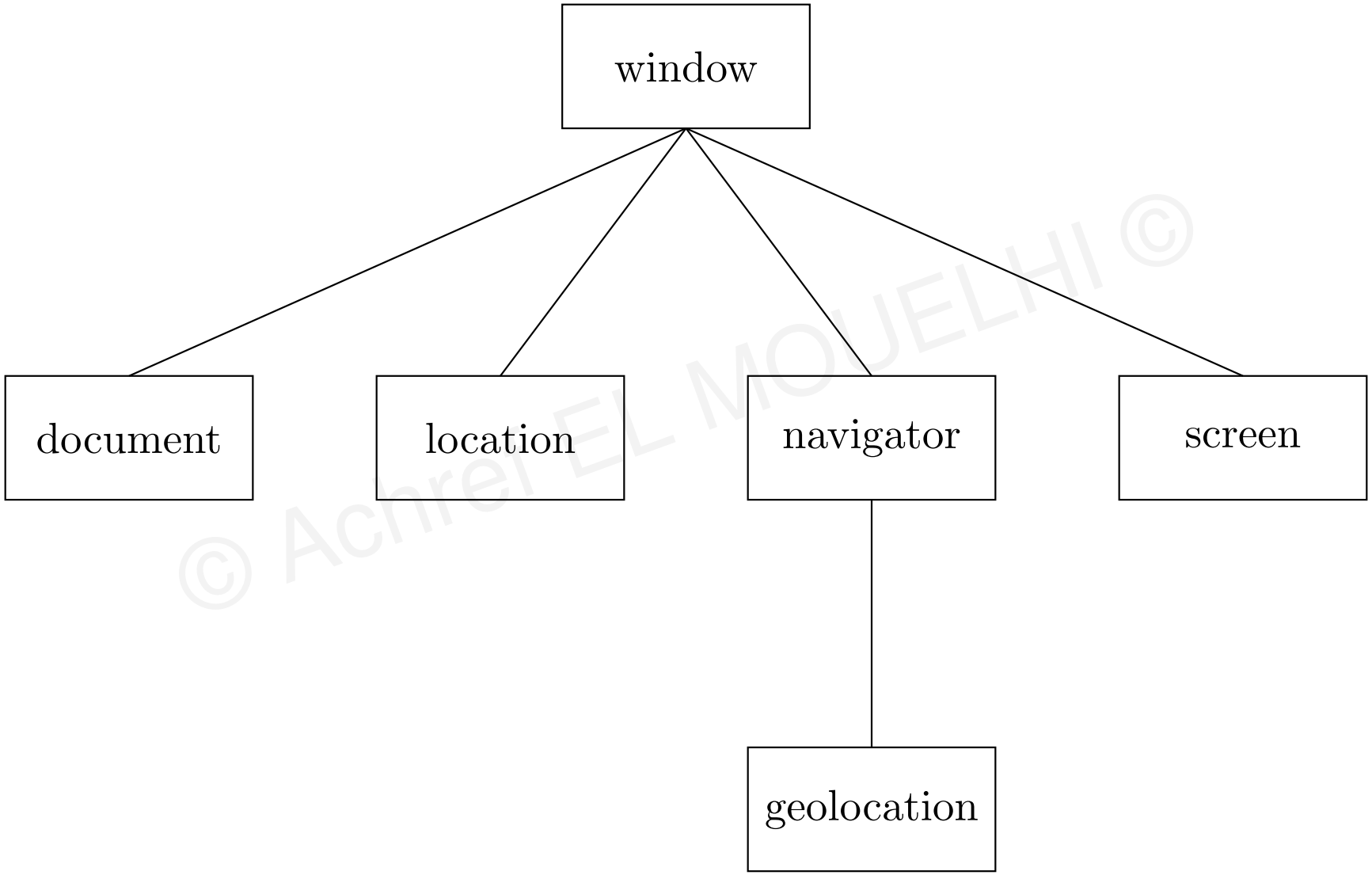
- `tooLong`
- `patternMismatch` : l'attribut `pattern` prend comme valeur une expression régulière et fonctionne uniquement avec les `input` de type `text`, `tel`, `email`, `url`, `password` et `search`
- `valid`
- `...`

API HTML 5

Geolocation

- { fonctions } définies dans un objet
- permettant de récupérer des données sur la position de l'utilisateur
- résultat plus précis pour les appareils avec **GPS**, comme les smartphones.

API HTML 5



Considérons la page `index.html`

```
<!DOCTYPE html>
<html>
<head>
  <title>Geolocation</title>
</head>
<body>
  <div id=mapholder></div>
  <button onclick=getLocation()>show</button>
  <script src="script.js"></script>
</body>
</html>
```

Considérons la page `index.html`

```
<!DOCTYPE html>
<html>
<head>
  <title>Geolocation</title>
</head>
<body>
  <div id=mapholder></div>
  <button onclick=getLocation()>show</button>
  <script src="script.js"></script>
</body>
</html>
```


Objectif

- Utiliser l'objet `geolocation`
- Récupérer les coordonnées (Longitude et latitude) et les afficher dans la `div` ayant l'identifiant `mapholder`

API HTML 5

Dans `script.js`, on commence par récupérer l'emplacement du résultat

```
var x = document.getElementById("mapholder");
```

API HTML 5

Dans `script.js`, on commence par récupérer l'emplacement du résultat

```
var x = document.getElementById("mapholder");
```

Code de la fonction `getLocation()` qui sera appelée en cliquant sur le bouton `show`

```
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser.";
    }
}
```

API HTML 5

Dans `script.js`, on commence par récupérer l'emplacement du résultat

```
var x = document.getElementById("mapholder");
```

Code de la fonction `getLocation()` qui sera appelée en cliquant sur le bouton `show`

```
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser.";
    }
}
```

La fonction `getLocation()` prend en paramètre une fonction callback qui sera exécutée après exécution de `getLocation()` et en récupérant la valeur de retour de cette dernière (position).

API HTML 5

La méthode `showPosition()`

```
function showPosition(position) {
  console.log(position.coords);
  x.innerHTML = "Latitude : " + position.coords.latitude +
                "<br>" +
                "Longitude : " + position.coords.longitude;
}
```

API HTML 5

Si on veut afficher la position sur une carte **Google Maps** ?

- Créer un projet **Google Developers Console** dans `https://console.developers.google.com/flows/enableapi?apiid=maps_backend&keyType=CLIENT_SIDE&reusekey=true&hl=fr&pli=1`
- Utiliser la même page pour créer une clé (sans aucune restriction sur les protocoles)
- Activer la clé en cherchant (dans la même interface en utilisant la zone de recherche) `Geocoding` et `Geolocating` et en cliquant chaque fois sur Activer
- Copier la clé pour l'utiliser

API HTML 5

La méthode `showPosition()` devient

```
function showPosition(position) {
  var latlon = position.coords.latitude + "," + position.coords.
     longitude;
  var cle = // coller ici la clé
  var img_url = "https://maps.googleapis.com/maps/api/staticmap?center=
     "
  +latlon+"&zoom=14&size=400x300&sensor=false&key=" + cle;
  x.innerHTML = "<img src='"+img_url+"'>";
}
```

Le code de la méthode `getLocation()` ne change pas.