ISEN ALL IS DIGITAL! OUEST | yncréa

JavaScript
- Canvas et gestion du temps -

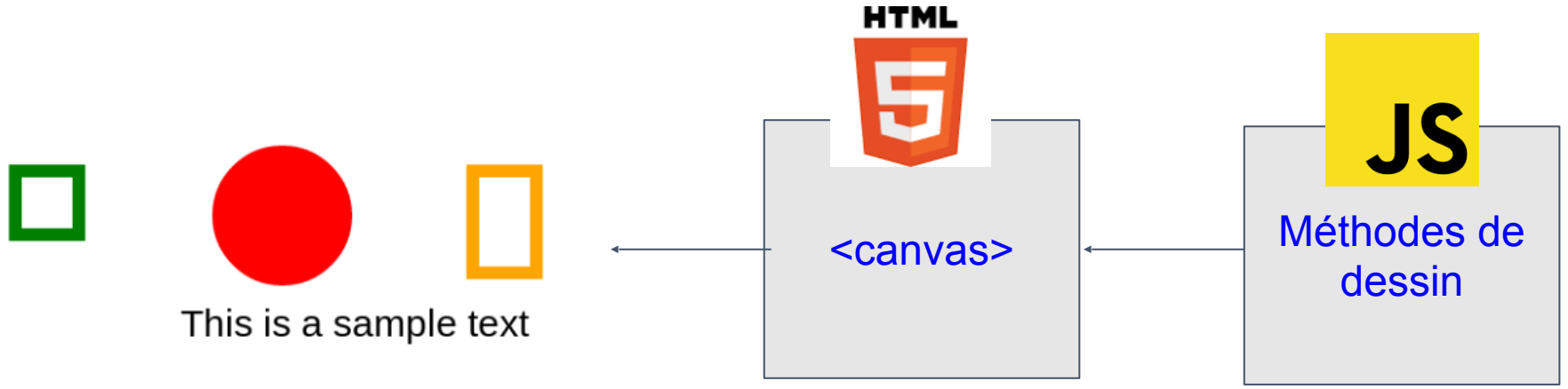
Groupe des étudiants : CIR1

Canvas

Définition

- La technologie canvas permet de dessiner dans une page web.
- Elle se décompose en deux parties :
  - un élément HTML <canvas> délimitant la zone de dessin
  - une API JavaScript : ensemble de méthodes de dessin

- Le résultat du rendu d'un canvas est une image matricielle (ensemble de pixels)



Balise <canvas>

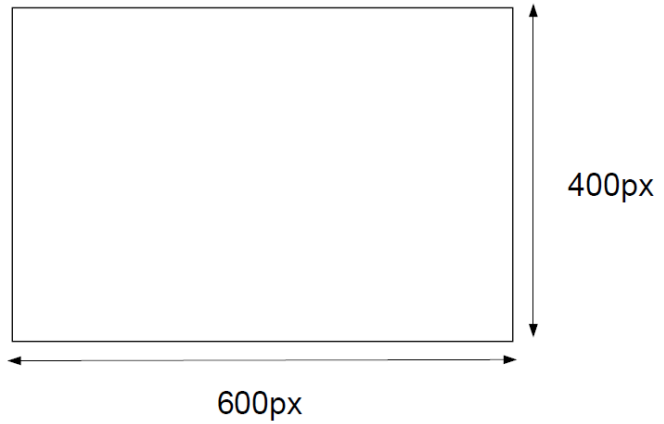
- La balise <canvas> possède des attributs width et height permettant de définir la taille (en pixels) de la zone de dessin (par défaut 300x150px)
- Par défaut, un canvas n'a pas de bords, ils peuvent être ajoutés via CSS
- Exemple :

canvasDemo.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Canvas demo</title>
    <link rel="stylesheet" type="text/css" href="canvasDemo.css">
  </head>

  <body>
    <canvas id="demo" height="400" width="600">Your browser does
not support canvas</canvas>
    </body>
</html>
```

canvasDemo.css

```
#demo{
    border: 1px solid black;
}
```

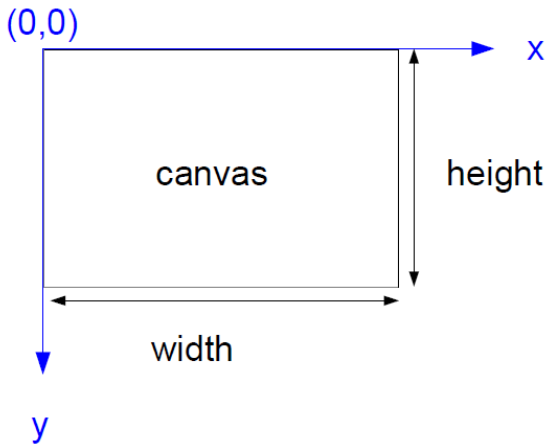


API JavaScript

Le dessin dans un canvas se fait via JavaScript

canvasDemo.js

```
// récupération de l'élément canvas
let canvas =
document.getElementById("demo");
// récupération du contexte 2D de
rendu //associé au canvas
let context = canvas.getContext("2d");
```



index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Canvas demo</title>
    <link rel="stylesheet" type="text/css" href="canvasDemo.css">
  </head>

  <body>
    <canvas id="demo" height="400" width="600">Your browser does
not support canvas</canvas>
    <script src="canvasDemo.js"></script>
    </body>
</html>
```

API JavaScript : chemins

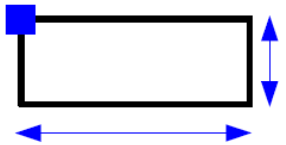
- Le dessin de formes dans un canvas se fait par l'intermédiaire de chemins (paths)

- On dessine un chemin en 4 étapes :
  1) Démarrage d'un nouveau chemin avec beginPath()
  2) Définition du chemin avec des primitives
  3) Fermeture du chemin avec closePath()
  4) Rendu du chemin

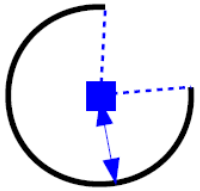
API JavaScript : chemins

- Le dessin de formes dans un canvas se fait par l'intermédiaire de chemins (paths)
- On dessine un chemin en 3 étapes :
  1) Démarrage d'un nouveau chemin avec beginPath()
  2) Définition du chemin avec des primitives

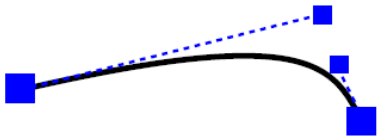
- Rectangle (coin + dimensions): `rect()`



- Arc de cercle (centre + rayon + angles début/fin): `arc()`



- Spline (points de contrôle): `quadratricCurveTo()` et `bezierCurveTo()`

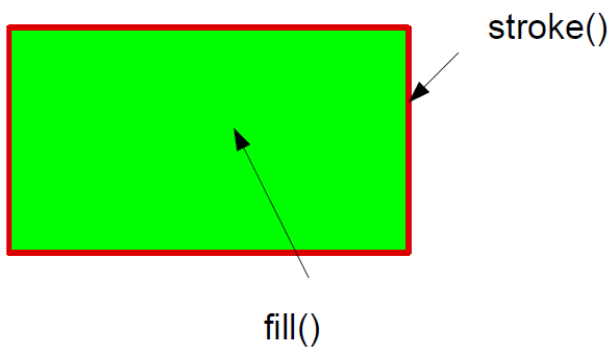


- Déplacement "crayon levé" : `moveTo()`

API JavaScript : chemins

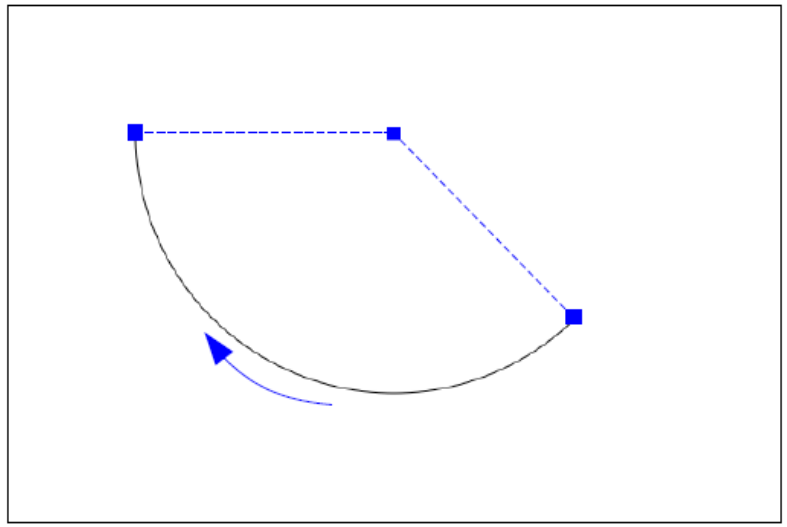
- Le dessin de formes dans un canvas se fait par l'intermédiaire de chemins (paths)

- On dessine un chemin en 3 étapes :
  1) Démarrage d'un nouveau chemin avec beginPath()
  2) Définition du chemin avec des primitives
  3) Fermeture du chemin avec closePath()
  4) Rendu du chemin
     a) stroke() : contour de la forme en utilisant la couleur courante du trait
     b) fill() : intérieur de la forme en utilisant la couleur de remplissage



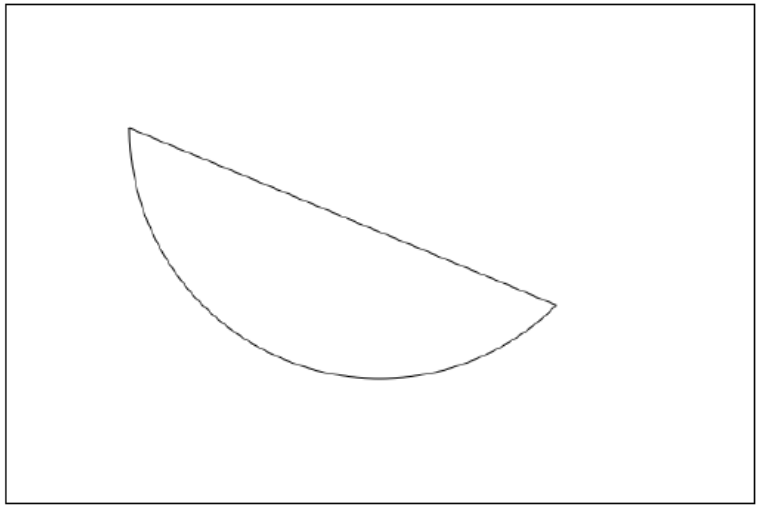
API JavaScript : Exemple

```
function get2DContext(id){
        let canvas =
document.getElementById(id);
        let context = canvas.getContext("2d");
        return context;
}
function main(){
        let context = get2DContext("demo");
        context.beginPath();
        context.arc(300, 100, 200, Math.PI/4,
Math.PI);
        context.stroke();
}
main();
```



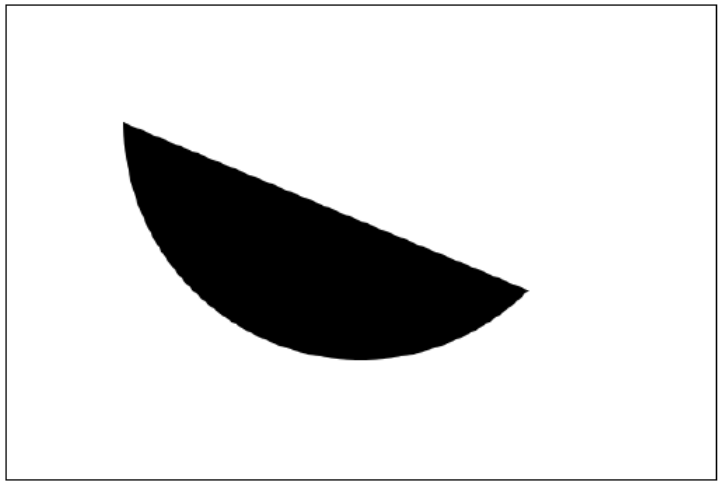
API JavaScript : Exemple

```
function get2DContext(id){
        let canvas =
document.getElementById(id);
        let context = canvas.getContext("2d");
        return context;
}
function main(){
        let context = get2DContext("demo");
        context.beginPath();
        context.arc(300, 100, 200, Math.PI/4,
Math.PI);
    context.closePath();
        context.stroke();
}
main();
```



API JavaScript : Exemple

```
function get2DContext(id){
        let canvas =
document.getElementById(id);
        let context = canvas.getContext("2d");
        return context;
}
function main(){
        let context = get2DContext("demo");
        context.beginPath();
        context.arc(300, 100, 200, Math.PI/4,
Math.PI);
    context.closePath();
        context.fill();
}
main();
```

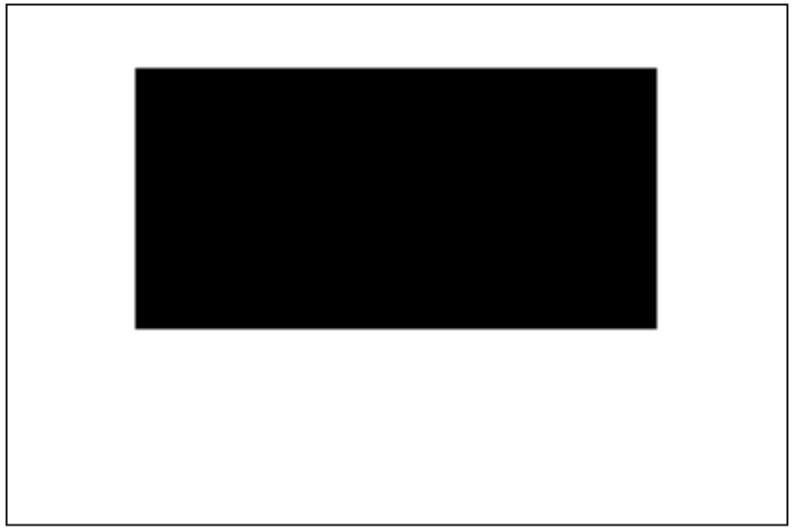


API JavaScript : cas particulier des rectangles

- Pour les rectangles, il existe des méthodes toutes faites qui évitent de définir explicitement des trajets et déclencher le rendu :
  - strokeRect() dessine le contour d'un rectangle
  - fillRect() dessine l'intérieur d'un rectangle

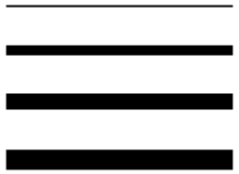
- Exemple avec fillRect()

```
context.fillRect(100, 50, 400, 200);
```

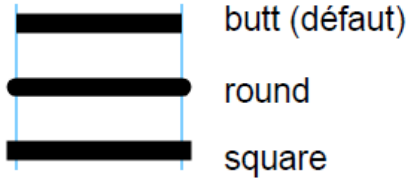


API JavaScript : Modification du style

- Largeur du trait : lineWidth



- Extrémités du trait : lineCap



- Style de contour : strokeStyle
- ….

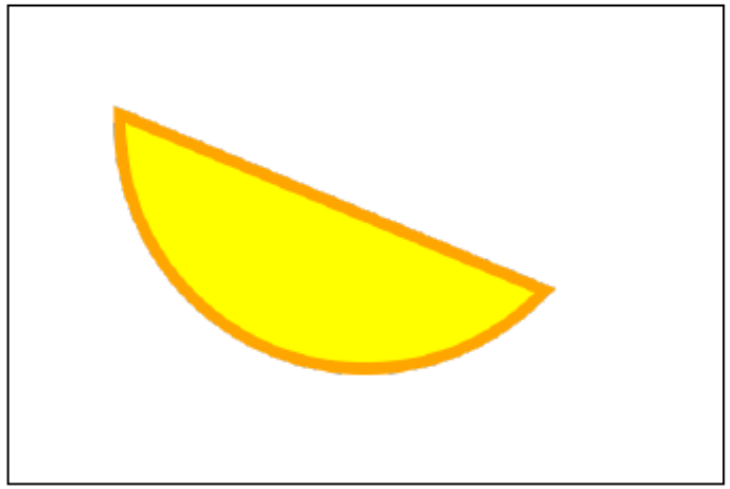
couleur
(green)

degradé
(linear gradient)

API JavaScript : Modification du style

- Exemple :

```
...

function main(){
    let context = get2DContext("demo");
    context.beginPath();
    context.lineWidth = 20;
    context.fillStyle = "yellow";
    context.strokeStyle = "#ffa500"; // orange
    context.arc(300, 100, 200, Math.PI/4, Math.PI);
    context.closePath();
    context.stroke();
    context.fill();
}

main();
```



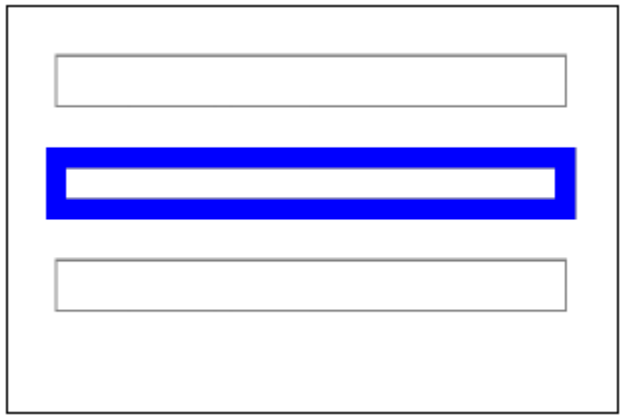
API JavaScript : Sauvegarde et restauration de contexte

- On peut sauvegarder un contexte de rendu avec la méthode save(), afin de le restaurer plus tard avec restore()

- La restauration de contexte permet de retrouver le style initial sans avoir à faire toutes les modifications inverses

- Exemple :

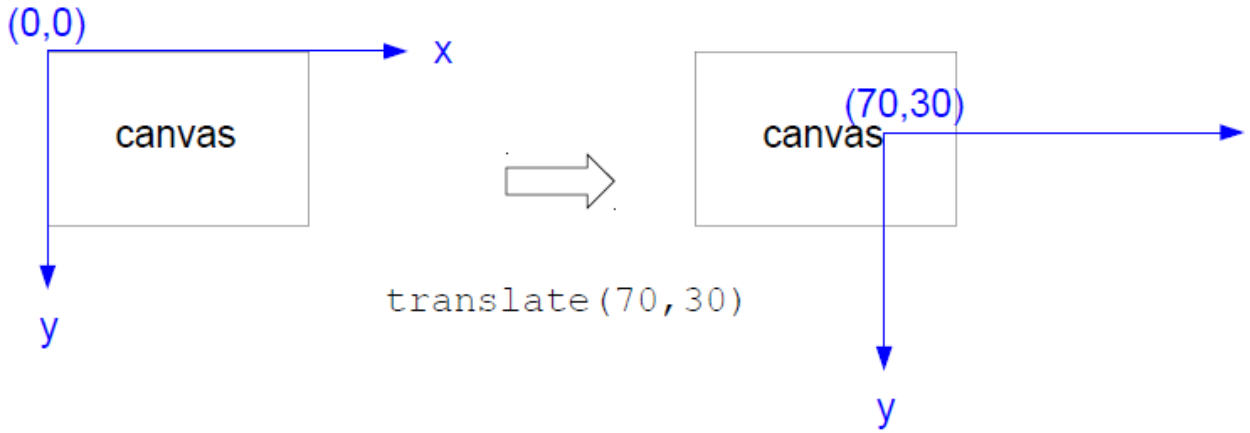
```
let context = get2DContext("demo");
context.strokeRect(50,50,500,50);
context.save();
context.lineWidth = 20;
context.strokeStyle = "blue";
context.strokeRect(50,150,500,50);
context.restore();
context.strokeRect(50,250,500,50);
```

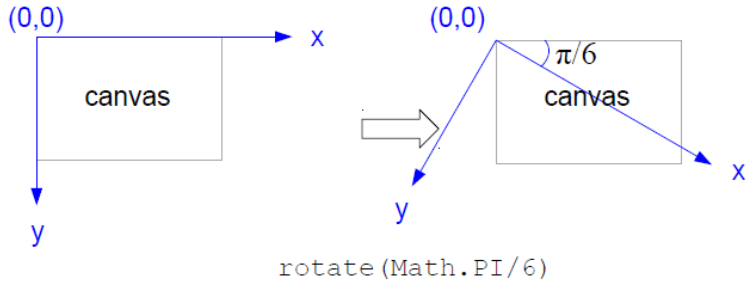


API JavaScript : Méthodes de transformation

- Les méthodes de transformation agissent sur le repère et non sur les formes

- **Exemple** :
  - translate(70, 30)
  - rotate(Math.PI/6)





API JavaScript : Dessin du texte

- On peut dessiner du texte avec les méthodes strokeText() et fillText()

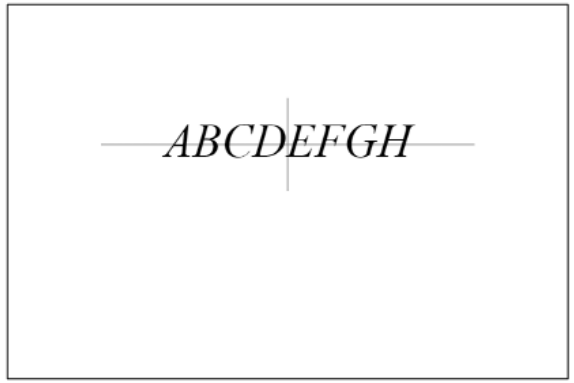
- On peut jouer sur l'apparence du texte en modifiant les propriétés associées du contexte de rendu, notamment :
  - la police : font
  - l'alignement (horizontal) du texte : textAlign
  - la ligne de base : textBaseLine

API JavaScript : Dessin du texte

- Exemple

```
function drawCross(context){
    context.save();
    context.strokeStyle = "grey";
    context.beginPath();
    context.moveTo(100,150);
    context.lineTo(500,150);
    context.moveTo(300,100);
    context.lineTo(300,200);
    context.stroke();
    context.restore();

}
```

```
...

function main(){

    let context = get2DContext("demo");
    drawCross(context);
    context.textBaseline = "middle";
    context.textAlign = "center";
    context.font = "italic 50px times";
    context.fillText("ABCDEFGH", 300,150);
}

main();
```



Gestion de temps

L'objet Date

- L'objet date mesure le nombre millisecondes depuis minuit le 01 janvier 1970 UTC (epoch Unix) → il permet de manipuler le temps et l'heure

- L'objet date a plusieurs méthodes, parmi elles :
  - toLocaleDateString() : renvoie une chaîne de caractères correspondant à l'heure
  - toLocaleTimeString() : renvoie une chaîne de caractères correspondant à la date (jour, mois, année)

- Exemple :

```
const event = new Date();

const options = { weekday: 'long', year: 'numeric', month: 'long', day:
'numeric' };

console.log(event.toLocaleDateString('ar-EG', options));

console.log(event.toLocaleTimeString('en-US'));
```

```
"الاثنين، ١١ مايو ٢٠٢٠"
"6:40:41 PM"
```

L'objet Date

- L'objet date mesure le nombre millisecondes depuis minuit le 01 janvier 1970 UTC (epoch Unix) → il permet de manipuler le temps et l'heure

- L'objet date a plusieurs méthodes, parmi elles :
  - toLocaleDateString() : renvoie une chaîne de caractères correspondant à la date(jour, mois, année)
  - toLocaleTimeString() : renvoie une chaîne de caractères correspondant à l'heure

- Exemple :

Pour utiliser le langage local par défaut du navigateur, on précise **undefined**

```
const event = new Date();

const options = { weekday: 'long', year: 'numeric', month:
'long', day: 'numeric' };

console.log(event.toLocaleDateString('ar-EG', options));

console.log(event.toLocaleTimeString('en-US'));
```

```
"الاثنين، ١١ مايو ٢٠٢٠"
"6:40:41 PM"
```

Méthodes de gestion du temps

- L'objet global window possède plusieurs méthodes de gestion du temps, parmi elles :

`setTimeout(f, delay)`	Appelle (une seule fois) la fonction `f` après `delay` ms
`setInterval(f, delay)`	Appelle la fonction `f` toutes les `delay` ms

- Pour chaque méthode m, l'appel explicite est window.m() mais on peut omettre l'objet window et écrire simplement m()

- La fonction f donnée en argument est une callback (comme pour les méthodes d'itération)

- Exemple

- Code :

```
function hello(){

   console.log("hello");
}

function main(){

   setTimeout(hello, 1000);
}
```

- Exécution (console de Chrome) :

```
> 10:15:33.385 main()
< 10:15:33.387 undefined
  10:15:34.387 hello
>
```

- Exemple

- Code :

```
function hello(){

  console.log("hello");
}

function main(){

  setInterval(hello, 1000);
}
```

- Exécution (console de Chrome) :

```
10:17:36.417 main()
10:17:36.420 undefined
10:17:37.420 hello
10:17:38.419 hello
10:17:39.419 hello
10:17:40.419 hello
10:17:41.420 hello
10:17:42.420 hello
...
```

- On peut annuler l'exécution d'une fonction via setTimeout
- **Exemple** :

```
function hello(){
  console.log("hello");
}

function main(){
    let helloId = setInterval(hello, 200);
    setTimeout(function(){
            console.log("Stop hello()");
            clearInterval(helloId);
        }
        , 1000);
}
```

hello() est appelée toutes les 200ms sous l'effet de setInterval()

Au bout de 1s, l'effet de setInterval() est annulé par clearInterval()

Il ne se passe plus rien ici

```
> 10:34:00.289 main()
< 10:34:00.291 undefined
  10:34:00.491 hello
  10:34:00.690 hello
  10:34:00.890 hello
  10:34:01.092 hello
  10:34:01.291 hello
  10:34:01.292 Stop hello()
>
```