Comment créer un diaporama d'images dans une page Web

.NET Framework 3.0 5 sur 6 ont trouvé cela utile

Les sites Web pourvus d'un grand nombre d'images, notamment les sites de galeries d'art ou de photos, peuvent bénéficier de l'affichage d'une partie (ou de l'intégralité) de leurs images au format de diaporama. Dans cette rubrique, vous découvrirez comment utiliser et créer un fichier JavaScript compatible avec plusieurs navigateurs (`slideShow.js`) et produisant in diaporama dans lequel les images apparaissent et disparaissent progressivement conformément à une liste donnée d'éléments `<img>`.

Cette rubrique se divise en deux parties principales :

- Comment utiliser le fichier slideShow.js pour créer un diaporama à partir d'une liste de balises img
- Détails de l'implémentation du fichier slideShow.js

Utilisation du fichier slideShow.js

Cet exemple décrit le balisage le plus simple nécessaire à la création d'un tel diaporama :

JavaScript

```
<!DOCTYPE html>
<html>

<head>
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
  <title>Slide Show</title>
</head>

<body>
  <div id="slideShowImages">
    <img src="images/slide1.jpg" alt="Slide 1" />
    <img src="images/slide2.jpg" alt="Slide 2" />
    <img src="images/slide3.jpg" alt="Slide 3" />
    <img src="images/slide4.jpg" alt="Slide 4" />
  </div>
  <button id="slideShowButton"></button> <!-- Optional button element. -->
  <script src="slideShow.js"></script>
</body>

</html>
```

Pour créer un diaporama à l'aide du fichier `slideShow.js`, les éléments de balisage suivants doivent être disponibles :

- Un wrapper `<div>` dont l'ID exact est `slideShowImages` et contenant des éléments `<img>` qui pointent sur les images souhaitées dans le diaporama.
- Un élément de script pointant sur `slideShow.js`.

Remarque

Cet exemple inclut un bouton bascule (facultatif) qui permet d'activer et de désactiver le diaporama. Ce bouton bascule, si vous l'utilisez, doit avoir un ID `slideShowButton`. Si vous ne l'utilisez pas, l'utilisateur peut arrêter ou démarrer le diaporama en cliquant sur l'une des images qui défilent en boucle dans le diaporama.
De plus, si vous utilisez le fichier `slideShow.js`, vous pouvez aisément définir le style du wrapper `<div>` du diaporama avec ses images enfants comme le montre cet exemple de code:

JavaScript

```
<!DOCTYPE html>
<html>

<head>
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
  <title>Slide Show</title>
  <style>
    #slideShowImages { /* The following CSS rules are optional. */
      border: 1px gray solid;
      background-color: lightgray;
    }

    #slideShowImages img { /* The following CSS rules are optional. */
      border: 0.8em black solid;
      padding: 3px;
    }
  </style>
</head>

<body>
  <div id="slideShowImages">
    <img src="images/slide1.jpg" alt="Slide 1" />
    <img src="images/slide2.jpg" alt="Slide 2" />
    <img src="images/slide3.jpg" alt="Slide 3" />
    <img src="images/slide4.jpg" alt="Slide 4" />
  </div>
  <button id="slideShowButton"></button> <!-- Optional button element. -->
  <script src="slideShow.js"></script>
</body>

</html>
```

L'une des fonctionnalités intéressantes du fichier `slideShow.js` est que des images des diapositives de dimensions variables sont automatiquement centrées dans le wrapper `<div>` et les transitions entre les images de tailles différentes semblent visiblement correctes puisque lorsqu'une image disparaît, l'autre apparaît simultanément.

Désormais, vous pouvez recourir au fichier `slideShow.js` pour créer un diaporama à partir d'une liste d'éléments `<img>`. Le reste de cette rubrique a pour but d'expliquer les détails d'implémentation du fichier `slideShow.js`.

Remarque Pour modifier la vitesse du diaporama ou le texte du bouton bascule, voir la section Globals.

Détails de l'implémentation

Comme vous pouvez le constater à l'examen de `slideShow.js`, ce fichier JavaScript est essentiellement composé de deux éléments, à savoir la ligne `window.addEventListener('load', slideShow, false)` et sa fonction de rappel annexe `slideShow`.

Du fait que la fonction `slideShow` référence `<div id="slideShowImages">` (voire `<button id="slideShowButton">`), nous devons nous assurer que ces éléments sont disponibles dans le DOM avant d'aller plus loin. C'est ce que permet d'accomplir la méthode `window.addEventListener('load', slideShow, false)`. En d'autres termes, la fonction `slideShow` est appelée uniquement lorsque le balisage de la page Web est disponible depuis le DOM. Un autre avantage à placer la majorité du code du fichier `slideShow.js` dans une fonction unique est que cela protège l'espace de noms des utilisateurs du fichier `slideShow.js`.

Nous savons donc à quel moment la fonction `slideShow` est appelée. Examinons à présent les détails de la fonction `slideShow` même en commençant par son algorithme principal :

Algorithme principal

La clef de voûte de l'algorithme de la fonction `slideShow` est la disposition CSS de `<div id="slideShowImages">` et de ses éléments `<img>` enfants. La disposition CSS est définie dans la fonction `initializedSlideShowMarkup` du fichier `slideShow.js`. C'est en particulier le cas de `initializedSlideShowMarkup` :

- Définit la `position` CSS de `<div id="slideShowImages">` sur `relative`. Ceci permet à ses enfants dont la position est absolue, de se positionner par rapport à ce conteneur. Si `<div id="slideShowImages">` était défini sur une position `static` (valeur par défaut), ses enfants positionnés de manière absolue seraient plutôt positionnés par rapport à l'élément `<body>` et non `<div id="slideShowImages">`.
- Positionne tous les éléments `<img>` du diaporama de manière absolue. Ceci permet d'empiler tous ces éléments `<img>` l'un au-dessus de l'autre dans leur conteneur `<div id="slideShowImages">`. Imaginez les diapositives comme une pile verticale d'images avec la première diapositive (le premier élément `<img>` répertorié) en bas de la pile et la dernière diapositive (le dernier élément `<img>` répertorié) en haut de la pile.
- Modifie et redéfinit l'opacité de chaque diapositive (dans la pile) sur 0 pour rendre toutes les diapositives entièrement transparentes. L'opacité du premier élément `<img>` est ensuite définie sur 1. La diapositive située en bas de la pile est alors visible dans toutes les diapositives transparentes placées au-dessus d'elle.

Cette disposition de départ achevée, nous pouvons maintenant nous déplacer dans la pile des diapositives en partant du bas (première diapositive) vers le haut (dernière diapositive) :

1. Gardez la diapositive actuelle (dans la pile) en tant que diapositive la plus basse (premier élément `<img>`).
2. Passez à la diapositive suivante dans la pile. Si aucune diapositive suivante n'existe, considérez la première diapositive comme diapositive suivante.
3. Définissez la valeur d'opacité de la diapositive actuelle sur 1 (visible) et celle de la diapositive suivante sur 0 (transparente).
4. Simultanément et lentement, changez la valeur d'opacité de la diapositive actuelle de 1 vers 0 et celle de la diapositive suivante de 0 vers 1. La diapositive actuelle disparaît alors graduellement tandis que la diapositive suivante apparaît.
5. Gardez la diapositive actuelle en tant que diapositive suivante.
6. Passez à l'étape 2 jusqu'à ce que l'utilisateur signale que le diaporama doit être arrêté.

Une fois cet algorithme principal assimilé, nous pouvons examiner les détails des composants majeurs de la fonction `slideShow`, notamment les composants suivants :

- Globals
- Main
- initializeGlobals
- insufficientSlideShowMarkup
- initializeSlideShowMarkup
- maxSlideWidth et maxSlideHeight
- startSlideShow
- haltSlideShow
- toggleSlideShow
- transitionSlides et fadeActiveSlides

Globals

Le premier élément dans `slideShow` est un littéral d'objet appelé `globals`. Cet objet est utilisé pour abriter toutes les « variables globales » de la fonction `slideShow` à un seul emplacement logique (et pratique). Par exemple, pour accéder à `slideDelay` depuis n'importe quel endroit dans `slideShow`, il vous suffit d'appeler `globals.slideDelay`.

Les six premières variables globales sont particulièrement dignes d'intérêt :

JavaScript

```
slideDelay: 4000,
fadeDelay: 35,
wrapperID: "slideShowImages",
buttonID: "slideShowButton",
buttonStartText: "Start Slides",
buttonStopText: "Stop Slides",
```

Le premier élément, `slideDelay`, détermine le temps (en millisecondes) pendant lequel une diapositive précise est visible à l'écran. Dans ce cas, chaque diapositive est visible pendant un laps de temps total de 4 secondes.

Pour décrire `fadeDelay`, il faut se reporter à l'algorithme principal utilisé dans le fichier `slideShow.js`. Prenons notre pile de diapositives. La diapositive située en bas de la pile affiche une valeur **opacity** égale à 1 (elle est visible) alors que les autres ont une valeur d'opacité égale à 0 (elles sont transparentes). Pour effectuer une opération de transition progressive entre la première diapositive et la diapositive suivante, nous devons diminuer l'opacité de la première diapositive de 1 à 0 et, simultanément, augmenter l'opacité de la deuxième de 0 à 1. À mi-parcours précisément de cette opération, les deux diapositives affichent une valeur d'opacité de 0,5. Une fois l'opération terminée, la première diapositive affiche une opacité égale à 0 (transparente) et la deuxième une opacité définie sur 1 (visible). Ce principe acquis, on comprend alors que `fadeDelay` représente le temps écoulé entre chaque changement individuel d'opacité. Par exemple, si `fadeDelay` est 35, 35 millisecondes s'écouleront entre chaque changement d'opacité. Le délai de changement d'opacité est la réciproque de cette valeur qui, dans le cas présent, est approximativement 0,0286. La valeur incrémentielle de l'opacité étant 1/35, 35 valeurs de ce type seront nécessaires pour passer de 0 à 1 (de même, 35 valeurs seront utiles pour diminuer l'opacité de 1 à 0). Le temps global nécessaire pour une opération de transition progressive est donc de 35 millisecondes multipliées par 35 ou 35^2 = 1 225 millisecondes = 1,225 secondes. De la même manière, si la valeur de `fadeDelay` est 45, cela prend 45^2 = 2,025 secondes pour accomplir une seule opération de transition progressive entre deux diapositives consécutives. Pour résumer : paramétrez les fonctions `slideDelay` et `fadeDelay` comme bon vous semble, mais gardez toujours une valeur

`fadeDelay` largement inférieure à `slideDelay`.

Pour continuer, `wrapperID` et `buttonID` doivent correspondre aux ID employés dans le balisage : `<div id="slideShowImages">` et `<button id="slideShowButton">`.

`buttonStartText` et `buttonStopText` définissent le texte à utiliser dans le bouton bascule (facultatif) du diaporama. Vous pouvez librement changer ces libellés (chaînes) selon les besoins de votre application.

Pour obtenir des informations sur les « variables globales » restantes, voir les commentaires dans le fichier `slideShow.js`.

Main

Examinons à présent le bloc de code « principal » de la fonction `slideShow` :

JavaScript

```
initializeGlobals();

if ( insufficientSlideShowMarkup() ) {
  return;
}

 // Assert: there's at least one slide image.

if (globals.slideImages.length == 1) {
  return;
}

// Assert: there are at least two slide images.

initializeSlideShowMarkup();

globals.wrapperObject.addEventListener('click', toggleSlideShow, false);

if (globals.buttonObject) {
  globals.buttonObject.addEventListener('click', toggleSlideShow, false);
}

startSlideShow();
```

Ce bloc de code principal peut se décrire comme suit :

1. `initializeGlobals` initialise `globals` offrant un accès à l'élément wrapper `<div>`, ses enfants `<img>` et le bouton bascule (si disponible).
2. `insufficientSlideShowMarkup` renvoie la valeur `false` si le balisage attendu par `slideShow.js` apparaît présent ; sinon, il renvoie la valeur `true`. Comme vous pouvez le constater dans le code fourni à titre d'exemple, si la valeur renvoyée est `true`, la fonction `slideShow` est immédiatement désactivée et met discrètement fin au fichier `slideShow.js` (par conception).
3. Si `globals.slideImages.length == 1` a la valeur `true`, un seul élément `<img>` de diaporama est présent dans le balisage et il est déjà affiché à l'écran ; il nous suffit de mettre fin à `slideShow.js` (en laissant cette image seule sur l'écran).
4. Le composant `initializeSlideShowMarkup` prépare le balisage spécifique (voir description dans la section consacrée à l'algorithme principal) pour le diaporama à venir.
5. Nous ajoutons ensuite un détecteur d'événements, `toggleSlideShow`, pour désactiver le diaporama (s'il est activé) et l'activer (dans le cas inverse) lorsque vous cliquez sur l'élément wrapper `<div>`.
6. Ensuite, si `globals.buttonObject` a la valeur `true`, le bouton bascule du diaporama est présent dans le diaporama et nous y associons la même fonction de détection d'événements (`toggleSlideShow`).
7. Enfin, nous appelons `startSlideShow` pour lancer définitivement le diaporama.

Chacune des fonctions décrites ici, tout comme les fonctions restantes figurant dans `slideShow`, sont décrites en détail dans la section suivante.

initializeGlobals

Voici le code de la fonction imbriquée `initializeGlobals` :

JavaScript

```
function initializeGlobals() {
  globals.wrapperObject = (document.getElementById(globals.wrapperID) ? document.getElementById
  globals.buttonObject = (document.getElementById(globals.buttonID) ? document.getElementById(g

  if (globals.wrapperObject) {
    globals.slideImages = (globals.wrapperObject.querySelectorAll('img') ? globals.wrapperObjec
  }
}
```

En nous basant sur les valeurs d'ID spécifiques contenues dans `globals` (soit `gloabals.wrapperID` et `globals.butonID`), nous recherchons une référence à ces valeurs si elles sont présentes dans le balisage. Puis, à l'aide de `querySelectorAll`, nous remplissons un tableau avec des objets `<img>` de diapositive. Ce tableau, `globals.slideImages`, agit en guise de pile métaphorique comme le décrit la section consacrée à l'algorithme principal.

insufficientSlideShowMarkup

Voici le code de la fonction imbriquée `insufficientSlideShowMarkup` :

JavaScript

```
function insufficientSlideShowMarkup() {
  if (!globals.wrapperObject) {
    if (globals.buttonObject) {
      globals.buttonObject.style.display = "none";
    }
    return true;
  }

  if (!globals.slideImages.length) {
    if (globals.wrapperObject) {
      globals.wrapperObject.style.display = "none";
    }

    if (globals.buttonObject) {
      globals.buttonObject.style.display = "none";
    }

    return true;
  }

  return false;
}
```

Cette fonction a pour principal objectif de renvoyer la valeur `false` si l'ensemble du balisage prévu est disponible et la valeur `true` dans le cas inverse. Comme vous le constatez dans le code ci-dessus, s'il s'avère que tout le balisage prévu n'est pas présent, le code tente de nettoyer le balisage associé du diaporama (il définit alors la valeur de la propriété CSS `display` sur `"none"`) avant de renvoyer la valeur `true`.

initializeSlideShowMarkup

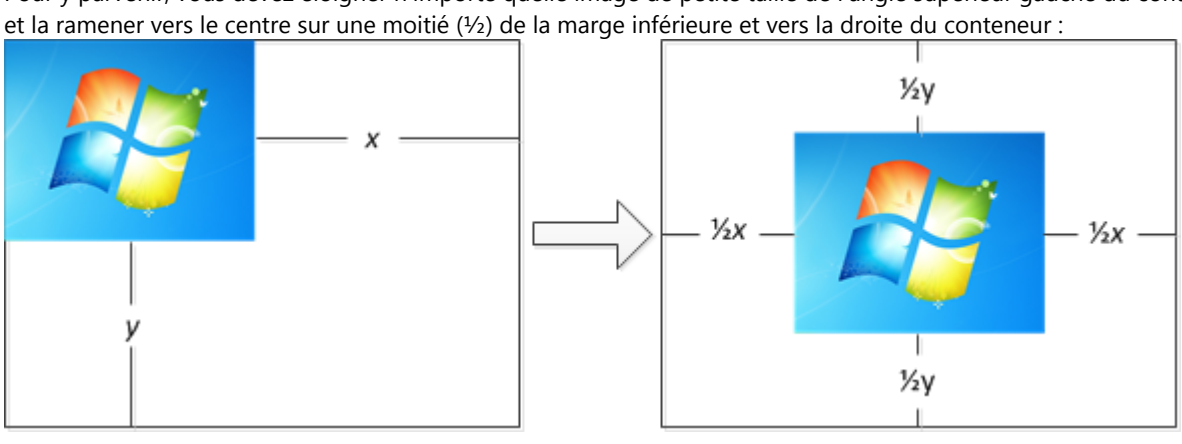
Voici le code de la fonction imbriquée `initializeSlideShowMarkup` :

JavaScript

```
function initializeSlideShowMarkup() {
  var slideWidthMax = maxSlideWidth();
  var slideHeightMax = maxSlideHeight();

  globals.wrapperObject.style.position = "relative";
  globals.wrapperObject.style.overflow = "hidden";
  globals.wrapperObject.style.width = slideWidthMax + "px";
  globals.wrapperObject.style.height = slideHeightMax + "px";

  var slideCount = globals.slideImages.length;
  for (var i = 0; i < slideCount; i++) {
    globals.slideImages[i].style.opacity = 0;
    globals.slideImages[i].style.position = "absolute";
    globals.slideImages[i].style.top = (slideHeightMax - globals.slideImages[i].getBoundingClie
    globals.slideImages[i].style.left = (slideWidthMax - globals.slideImages[i].getBoundingClie
  }

  globals.slideImages[0].style.opacity = 1;

  if (globals.buttonObject) {
    globals.buttonObject.textContent = globals.buttonStopText;
  }
}
```

Cette fonction permet de réaliser de multiples opérations, notamment les suivantes :

- Définit les valeurs CSS `position` de `<div id="slideShowImages">` et ses enfants `<img>` comme cela est décrit dans la section consacrée à l'algorithme principal.
- Règle les valeurs `width` et `height` du conteneur `<div id="slideShowImages">` afin que les images des diapositives dotées de la largeur et de la hauteur les plus élevées s'y intègrent parfaitement (si les images des diapositives ont des dimensions différentes).
- Règle la position `absolute` des images des diapositives afin qu'elles soient centrées dans le conteneur `<div id="slideShowImages">` (ici encore, au cas où les images des diapositives présentent des dimensions hétérogènes). Pour y parvenir, vous devez éloigner n'importe quelle image de petite taille de l'angle supérieur gauche du conteneur et la ramener vers le centre sur une moitié (½) de la marge inférieure et vers la droite du conteneur :



- Enfin, si le bouton bascule est présent dans le balisage, la fonction définit le texte « arrêter le diaporama » du bouton bascule (dans la mesure où, par défaut, le diaporama est initialement en cours d'exécution).

maxSlideWidth et maxSlideHeight

Ces deux fonctions renvoient la largeur de l'image la plus large et la hauteur de l'image la plus haute dans le diaporama (il peut s'agir de deux images différentes). Ces deux fonctions étant quasiment identiques, nous examinerons seulement la fonction `maxSlideWidth` :

JavaScript

```
function maxSlideWidth() {
  var maxWidth = 0;
  var maxSlideIndex = 0;
  var slideCount = globals.slideImages.length;

  for (var i = 0; i < slideCount; i++) {
    if (globals.slideImages[i].width > maxWidth) {
      maxWidth = globals.slideImages[i].width;
      maxSlideIndex = i;
    }
  }

  return globals.slideImages[maxSlideIndex].getBoundingClientRect().width;
}
```

La méthode `getBoundingClientRect` renvoie toujours une valeur exprimée en pixels et inclut toutes les propriétés CSS spécifiées, telles que les bordures, les marges, etc. Ceci permet de s'assurer que la largeur et la hauteur de `<div id="slideShowImages">` sont suffisantes pour intégrer tous les ajouts CSS réalisés dans les images du diaporama.

startSlideShow

Le code de la fonction imbriquée `startSlideShow` est très simple :

JavaScript

```
function startSlideShow() {
  globals.slideShowID = setInterval(transitionSlides, globals.slideDelay);
}
```

Il s'agit simplement de demander que la fonction `transitionSlides` soit appelée à chaque intervalle `gloabls.slideDelay` (en millisecondes) jusqu'à ce que les appels répétitifs soient arrêtés en appelant `clearInterval(globals.slideShowID)`.

haltSlideShow

Comme vous vous y attendiez peut-être, le code du composant `haltSlideShow` est constitué d'une seule ligne :

JavaScript

```
function haltSlideShow() {
  clearInterval(globals.slideShowID);
}
```

Ceci a pour effet de supprimer les appels de `transitionSlides` précédemment demandés à chaque intervalle `globals.slideDelay` (en millisecondes) et d'arrêter le diaporama.

toggleSlideShow

Voici le code de la fonction imbriquée `toggleSlideShow` :

JavaScript

```
function toggleSlideShow() {
  if (globals.slideShowRunning) {
    haltSlideShow();
    if (globals.buttonObject) {
      globals.buttonObject.textContent = globals.buttonStartText;
    }
  }
  else {
    startSlideShow();
    if (globals.buttonObject) {
      globals.buttonObject.textContent = globals.buttonStopText;
    }
  }
  globals.slideShowRunning = !(globals.slideShowRunning);
}
```

Exécutez de nouveau les deux appels `addEventListener` effectués dans le bloc de code principal :

JavaScript

```
globals.wrapperObject.addEventListener('click', toggleSlideShow, false);

if (globals.buttonObject) {
  globals.buttonObject.addEventListener('click', toggleSlideShow, false);
}
```

La fonction `toggleSlideShow` est donc appelée lorsque vous cliquez sur `<div id="slideShowImages">` ou `<button id="slideShowButton">` (si disponibles).

Ensuite, parce que `globals.slideShowRunning` est défini au départ sur `true`, lors du premier appel de `toggleSlideShow`, `haltSlideShow` est appelé et le texte du bouton bascule (le cas échéant) change et reprend sa forme de démarrage du diaporama. Les effets de `globals.slideShowRunning` sont ensuite annulés indiquant l'état actuel du diaporama (désactivé dans le cas présent).

Lorsque le composant `toggleSlideShow` est rappelé (quand vous cliquez sur une image du diaporama ou sur le bouton bascule), `globals.slideShowRunning` prend la valeur `false`, de sorte que `startSlideShow` est appelé, le texte du bouton change de forme pour indiquer que le diaporama est désactivé et les effets de `globals.sideShowRunning` sont annulés (indiquant que le diaporama est désormais activé/en cours d'exécution).

Pour résumer, le principe du composant `toggleSlideShow` est le suivant : il active et désactive le diaporama en appelant `startSlideShow` et `haltSlideShow` conformément à l'état défini pour `globals.slideShowRunning`.

transitionSlides et fadeActiveSlides

L'algorithme principal est principalement implémenté dans la fonction `transitionSlides` :

JavaScript

```
function transitionSlides() {
  var currentSlide = globals.slideImages[globals.slideIndex];

  ++(globals.slideIndex);
  if (globals.slideIndex >= globals.slideImages.length) {
    globals.slideIndex = 0;
  }

  var nextSlide = globals.slideImages[globals.slideIndex];

  var currentSlideOpacity = 1;
  var nextSlideOpacity = 0;
  var opacityLevelIncrement = 1 / globals.fadeDelay;
  var fadeActiveSlidesID = setInterval(fadeActiveSlides, globals.fadeDelay);

  function fadeActiveSlides() {
    currentSlideOpacity -= opacityLevelIncrement;
    nextSlideOpacity += opacityLevelIncrement;

    // console.log(currentSlideOpacity + nextSlideOpacity);

    if (currentSlideOpacity >= 0 && nextSlideOpacity <= 1) {
      currentSlide.style.opacity = currentSlideOpacity;
      nextSlide.style.opacity = nextSlideOpacity;
    }
    else {
      currentSlide.style.opacity = 0;
      nextSlide.style.opacity = 1;
      clearInterval(fadeActiveSlidesID);
    }
  }
}
```

Gardez à l'esprit que `globals.slideImages` est un tableau d'objets d'images de diapositives et que `globals.slideIndex` est initialement défini sur 0. Voici par conséquent le pseudo-code pour `transitionSlides` :

1. Gardez la diapositive actuelle indiquée par `globals.slideIndex` :

   JavaScript

   ```
   var currentSlide = globals.slideImages[globals.slideIndex];
   ```

2. Passez à la diapositive suivante et s'il n'en existe aucune, commencez depuis le début :

   JavaScript

   ```
   ++(globals.slideIndex);
   if (globals.slideIndex >= globals.slideImages.length) {
     globals.slideIndex = 0;
   }
   ```

3. Gardez la diapositive suivante indiquée par `globals.slideIndex` :

   JavaScript

   ```
   var nextSlide = globals.slideImages[globals.slideIndex];
   ```

4. Préparez les variables locales utilisées pour faire disparaître la diapositive actuelle et apparaître la diapositive suivante :

   JavaScript

   ```
   var currentSlideOpacity = 1;
   var nextSlideOpacity = 0;
   var opacityLevelIncrement = 1 / globals.fadeDelay;
   ```

5. Appelez la fonction locale `fadeActiveSlides` à chaque intervalle `globals.fadeDelay` (en millisecondes) jusqu'à ce que les deux diapositives soient complètement estompées. La fonction `fadeActiveSlides` est donc appelée à chaque intervalle `globals.fadeDelay` (en millisecondes) tandis que `currentSlideOpacity >= 0` et `nextSlideOpacity <= 1`. Si l'une de ces variables prend la valeur `false`, les deux diapositives apparaissent entièrement estompées pour des raisons pratiques. Leurs valeurs d'opacité sont alors explicitement définies et les appels répétés de `fadeActiveSlides` sont arrêtés dans la clause `else` :

   JavaScript

   ```
   currentSlide.style.opacity = 0;
   nextSlide.style.opacity = 1;
   clearInterval(fadeActiveSlidesID);
   ```

Comme le montre le code de `fadeActiveSlides`, les proportions dans lesquelles l'opacité de la diapositive actuelle est diminuée et celle de la diapositive suivante est augmentée sont les mêmes et, du fait que les valeurs d'opacité sont toujours comprises entre 0 et 1, il s'ensuit que la somme de ces deux valeurs d'opacité doit toujours être égale 1 (ou au moins très proche de 1 en raison des erreurs d'arrondi). En réalité, cette situation est vraie et peut se vérifier en appelant `console.log(currentSlideOpacity + nextSlideOpacity)` comme le suggère le commentaire fourni dans l'exemple de code précédent et répété ici :

JavaScript

```
// console.log(currentSlideOpacity + nextSlideOpacity);
```

Pour plus d'informations sur `console.log`, voir Comment utiliser les outils de développement F12 pour déboguer vos pages Web.

Rôle de CSS dans l'implémentation du fichier slideShow.js

Paradoxalement, c'est la technologie CSS qui permet d'implémenter cette bibliothèque de diaporamas JavaScript (`slideShow.js`). C'est notamment le cas si vous définissez la valeur `position` CSS de `<div id="slideShowImages">` sur `relative`, ainsi que la totalité de ses enfants `<img>` sur une valeur `position` de type `absolute`. Cette opération permet alors de créer une « pile » d'images de diapositives transparentes. Le but recherché est alors de modifier les valeurs d'opacité des diapositives adjacentes pour créer un diaporama en style fondu.

Enfin, il existe de nombreuses manières différentes d'implémenter une bibliothèque de diaporamas JavaScript, notamment divers plug-ins jQuery. Avec une description détaillée expliquant au moins une implémentation de ce type, il devient plus facile d'écrire votre propre bibliothèque de diaporamas personnalisée ou de modifier le fichier `slideShow.js` pour répondre à vos besoins spécifiques (si nécessaire).

Rubriques associées

Galeries de photos Contoso Images