Creating Widgets - a Pie Menu (i)

Using libraries like JQuery extends the range of widgets and interaction techniques that can be used on web pages.

However, sometimes we wish to use a widget which is not part of the standard web widget set, nor supported by libraries like JQuery.

Usability engineers often have to create new widgets for special purposes.

Dynamic HTML is commonly used to create fully- or partly-functional prototype widgets that can be tested with users.

An increasingly popular style of widget is the *pop-up pie menu* (or *radial* or *ring* menu, as it is sometimes known).

However, there is only limited support for this type of menu in software libraries.

The aim of this exercise is to create a basic pop-up pie menu using Dynamic HTML.

This will involve creating HTML elements (the menu and the icons within it) dynamically, i.e, creating them in memory and then inserting them into the document's *node tree* after the page has been loaded.

The first step is to create a basic web-page that includes a `<script>` in the `<head>` section of the page.

Create a global variable `popupMenu` and set it to `null`; this will hold a reference to the menu when it is created.

Add an event listener to the *window* object which calls a function `setup()` when the `DOMContentLoaded` event occurs.

Create the `setup()` function, and add a line of code to this function which attaches an event listener to the `document`.

The event listener should call a function `createMenu()` when a `click` event occurs.

For example:

```
window.addEventListener('DOMContentLoaded', setup, false);
var popupMenu = null;

function setup() {

  document.addEventListener('click', createMenu, false);
}
```

Next, write the `createMenu()` function that is called by the event listener.

This function should first check the `popupMenu` global variable to make sure that the menu has not already been created.

Next, it should create a `<div>` element to serve as the background of the pop-up menu. This can be done using the `createElement()` method, e.g.:

```
var menuDiv = document.createElement('div');
```

This code creates a `<div>` element in *memory*, but it won't appear on the web-page yet.

Once an element has been created in memory, its style attributes, etc., can be set, e.g.

```
menuDiv.style.position = 'absolute';
```

In this way, set the `width`, `height`, `left` and `top` attributes to suitable values, and also give the `<div>` a distinctive `backgroundColor`.

By default, elements are nominally rectangular, but you can make your `<div>` appear (and function) as a circular element by setting the `borderRadius` property - if you set it to half the width and height of the div, the result will appear as a circle.

In order for the `<div>` to appear on the web page, it must be added into the document's node tree. This can be done using the `appendChild()` method, e.g.

```
popupMenu = document.documentElement.appendChild(menuDiv);
```

The `appendChild()` method returns a reference to the newly-created element.

Store this in the global variable you created earlier - `popupMenu`. It will be needed when the time comes to remove the pop-up menu.

```
function createMenu() {

  if(!popupMenu) {

    var menuDiv = document.createElement('div');
    menuDiv.style.position = 'absolute';
    menuDiv.style.left = '450px';
    menuDiv.style.top = '100px';
    menuDiv.style.width = '200px';
    menuDiv.style.height = '200px';
    menuDiv.style.borderRadius = '100px';
    menuDiv.style.backgroundColor = '#cccccc';
    popupMenu = document.documentElement.appendChild(menuDiv);
  }
}
```

Test your code. Clicking anywhere in the document should cause the `<div>` to appear.

Once the `<div>` has appeared, further clicks should have no effect.

As described so far, the `<div>` will always appear at a fixed position on the page.

Modify the `createMenu()` function so that it sets the `left` and `top` attributes to the current cursor position, as contained in the `clientX` and `clientY` properties of the event object.

Note that for this to work you must pass in a reference to the event object when you declare the function.

```
function createMenu(evt) {

  if(!popupMenu) {

    var menuDiv = document.createElement('div');
    menuDiv.style.position = 'absolute';

    menuDiv.style.top = parseInt(evt.clientY) + 'px';
    menuDiv.style.width = '200px';
    menuDiv.style.height = '200px';
    menuDiv.style.borderRadius = '100px';
    menuDiv.style.backgroundColor = '#cccccc';

    popupMenu = document.documentElement.appendChild(menuDiv);
  }
}
```

Test your code in a browser.

The `<div>` should now appear at the point where you click on the page, just as a pop-up menu should.

If you wish, you can add code to adjust the position of the menu relative to the cursor position, as was done in the earlier drag-and-drop exercises.

The next step is to write code to remove the menu when it is no longer needed.

Add a line of code to the `createMenu()` function which adds an event listener to the new `<div>`

The event listener should call a function `closeMenu` whenever a `click` event occurs over the menu.

```
function createMenu(evt) {

  var menuDiv = document.createElement('div');
  menuDiv.style.position = 'absolute';
  menuDiv.style.left = parseInt(evt.clientX) + 'px';
  menuDiv.style.top = parseInt(evt.clientY) + 'px';
  menuDiv.style.width = '200px';
  menuDiv.style.height = '200px';
  menuDiv.style.borderRadius = '100px';
  menuDiv.style.backgroundColor = '#cccccc';

  popupMenu = document.documentElement.appendChild(menuDiv);
  popupMenu.addEventListener('click', closeMenu, false);
}
```

Now write the `closeMenu()` function that will be called by the event listener you have just created, and pass in a reference to the event object.

The function should first cancel any default actions associated with the event and stop it propagating to other elements, using the `evt.preventDefault()` and `evt.stopPropagation()` methods.

If this is not done, the `click` event may be received by the document and trigger the creation of a new menu every time you try to remove a menu.

Now add a line of code to this function to remove the menu `<div>`. This can done using the `removeChild()` method, e.g.:

```
document.documentElement.removeChild(popupMenu);
```

Once the `<div>` has been removed, reset the `popupMenu` global variable to `null`.

```
function closeMenu(evt) {

  evt.preventDefault();
  evt.stopPropagation();
  document.documentElement.removeChild(popupMenu);
  popupMenu = null;
}
```

Test your code in a browser. Clicking anywhere in the document should cause the `<div>` to appear, and clicking on it should remove it.

You now need to add some 'icons' to your menu.

Write another function, `addIcon()` that creates a `<div>` to serve as the first icon.

It can be created in the same way you created the menu `<div>`, using `document.createElement()`.

This `<div>` should be small enough to fit inside the menu `<div>`, and have a different colour.

Set its `position` attribute to `absolute`, and set the `left` and `top` attributes to suitable values.

Instead of appending this `<div>` to the `documentElement` as before, append it to the pop-up menu `<div>`, e.g.

```
newIcon = popupMenu.appendChild(iconDiv);
```

Because the new `<div>` is appended as a child of the menu `<div>`:

- It will be removed when the menu `<div>` is removed - there is no need to remove it separately.
- The `left` and `top` style attributes are calculated relative to the `left` and `top` edges of the menu `<div>` rather than to the edges of the browser window.
- Clicking the 'icon' should also close the menu, so add a call to the `closeMenu()` function.

```
function addIcon() {

  var iconDiv = document.createElement('div');
  iconDiv.id = 'icon1';
  iconDiv.style.position = 'absolute';
  iconDiv.style.left = '80px';
  iconDiv.style.top = '10px';
  iconDiv.style.width = '40px';
  iconDiv.style.height = '40px';
  iconDiv.style.backgroundColor = '#ff0000';
  iconDiv.style.borderRadius = '10px';

  newIcon = popupMenu.appendChild(iconDiv);
  newIcon.addEventListener('click', closeMenu, false);
}
```

Add a line to the `createMenu()` funtion that calls the `createMenu()` function.

```
function createMenu(evt) {

  var menuDiv = document.createElement('div');
  menuDiv.style.position = 'absolute';
  menuDiv.style.left = parseInt(evt.clientX) + 'px';
  menuDiv.style.top = parseInt(evt.clientY) + 'px';
  menuDiv.style.width = '200px';
  menuDiv.style.height = '200px';
  menuDiv.style.borderRadius = '100px';
  menuDiv.style.backgroundColor = '#cccccc';
  popupMenu = document.documentElement.appendChild(menuDiv);
  popupMenu.addEventListener('click', closeMenu, false);
  addIcon();
}
```

Test your code in a browser. When you click on the page, the 'menu' should appear with the 'icon' inside it.

Clicking on either the 'menu' or the 'icon' should remove the menu.

Adjust the position of the icon `<div>` if necessary so that it appears in an appropriate position within the menu.

You now need to extend the code so that clicking on the 'icon' does something - ideally it would call a function, but for the moment it will just display a dialog-box.

Modify the `closeMenu()` function so that it:

- obtains the `id` of the icon which received the 'click' event.
- displays the `id` in an `alert()` dialog-box.

```
function closeMenu(evt) {

  evt.preventDefault();
  evt.stopPropagation();
  document.documentElement.removeChild(popupMenu);
  if (evt.target.id) alert('Selected icon = ' + evt.target.id);
  popupMenu = null;
}
```

Test your code in a browser.

Clicking on either the menu or the icon should close the menu, but clicking on the icon should also display an `alert()` showing the `id` of the icon.

When this is working correctly, add more icons to the menu.

You could simply create several `addIcon()` functions, one for each icon.

A more elegant approach would be to use a single `addIcon()` function, but modify it so that the position, etc., of each icon can be passed to it as parameters.

```
function addIcon(xCoord, yCoord, iconNumber) {

  var iconDiv = document.createElement('div');
  iconDiv.id = 'icon' + iconNumber;
  iconDiv.style.position = 'absolute';
  iconDiv.style.left = xCoord + 'px';
  iconDiv.style.top = yCoord + 'px';
  iconDiv.style.width = '40px';
  iconDiv.style.height = '40px';
  iconDiv.style.backgroundColor = '#ff0000';
  iconDiv.style.borderRadius = '10px';

  newIcon = popupMenu.appendChild(iconDiv);
  newIcon.addEventListener('click', closeMenu, false);
}
```

Multiple icons can now be created by calling the `addIcon()` function multiple times with appropriate parameters, e.g.:

```
addIcon(80, 10, 0);
addIcon(150, 80, 1);
addIcon(80, 150, 2);
addIcon(10, 80, 3);
```

etc.

The widget as described is very basic but provides the essential features of a pie menu.

Once it is working you can easily extend and customise it.

An obvious next step is to replace each 'icon' `<div>` with an image.

You may also wish to add 'rollover' highlighting to icons by adding `mouseover` and `mouseout` event listeners which trigger a change in the background colour when the cursor is over an icon.