It's Coding Time!

WE'LL COVER THE FOLLOWING

Now that we've seen an English version of how a sprite animation works, it's time to convert all of that into JavaScript. Our code is going to follow these four basic steps:

1. Load the sprite sheet. This is pretty straightforward.
2. Use `drawImage` to display just the first sprite from our sprite sheet. If you recall, the `drawImage` method allows you to optionally specify the exact co-ordinates and dimension of the image you want to display instead of displaying the whole thing.
3. Shift the `drawImage` co-ordinates to display the next sprite...and the next sprite...and so on.
4. Put all of the `drawImage` logic inside a `requestAnimationFrame` loop to create our animation.

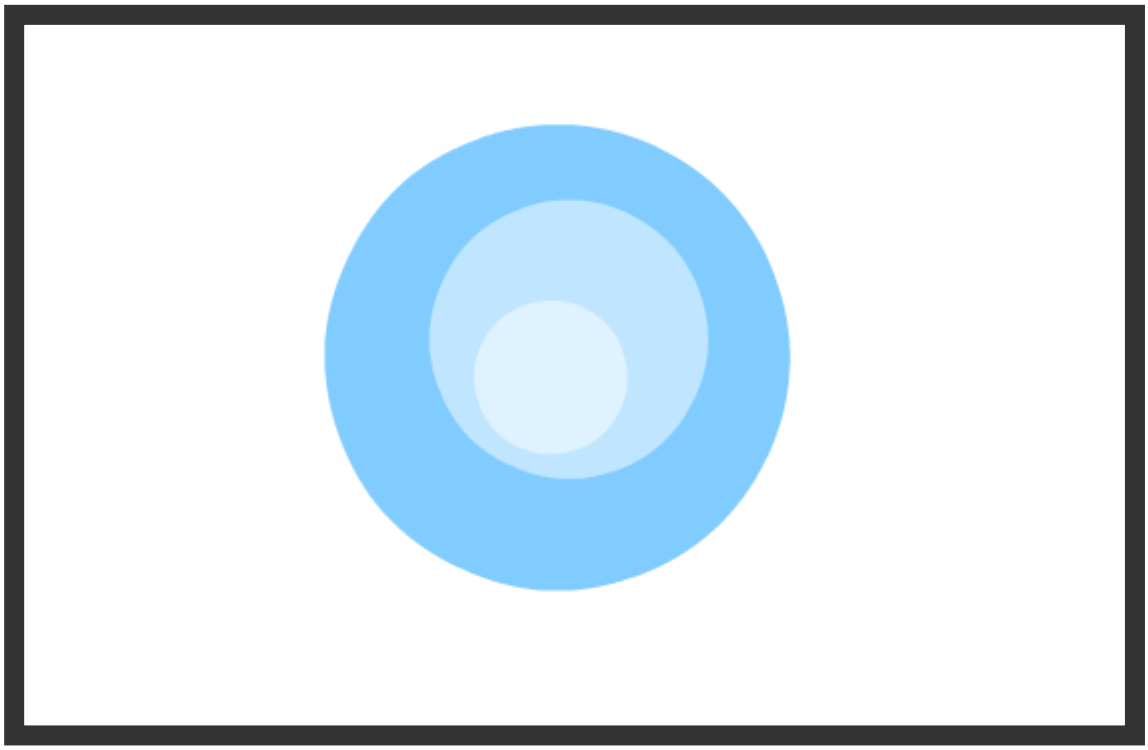
We are going to add some code that does all four of these steps next! Make sure you have our usual HTML document setup with a `canvas` element whose `id` value is `myCanvas`. This is the same type of document we've been starting off from forever, but for your reference, the content look as follows:

HTML JavaScript

```javascript
var canvas = document.querySelector("#myCanvas");
var context = canvas.getContext("2d");

var myImage = new Image();
myImage.src = "https://www.kirupa.com/stuff/sprites_blue.png";
myImage.addEventListener("load", loadImage, false);
```

```
function loadImage(e) {
  animate();
}

var shift = 0;
var frameWidth = 300;
var frameHeight = 300;
var totalFrames = 24;
var currentFrame = 0;

function animate() {
  context.clearRect(120, 25, 300, 300);

  //draw each frame + place them in the middle
  context.drawImage(myImage, shift, 0, frameWidth, frameHeight,
                    120, 25, frameWidth, frameHeight);

  shift += frameWidth + 1;

  /*
    Start at the beginning once you've reached the
    end of your sprite!
  */
  if (currentFrame == totalFrames) {
```

output



Once you've added your code, make sure everything works by previewing your document in your browser. If everything worked properly, you will see a blue circle (with a sweet circular design inside it) happily rotating. This is all a result of us animating the contents of the sprite sheet that you saw earlier. In

the next section, we'll take apart this code and learn how everything works!

Loading the Image

Before we can even think about animating our sprite sheet, we first need to load the sprite sheet image into our `canvas`. That is handled by the following chunk of code:

```
var myImage = new Image();
myImage.src = "https://www.kirupa.com/stuff/sprites_blue.png";
myImage.addEventListener("load", loadImage, false);

function loadImage(e) {
  animate();
}
```

All of this should be familiar to you if you've already seen the Drawing Images on the Canvas tutorial. We create a new `Image` object called `myImage`. We set the `src` property to the image we want to load, and then we listen for the **load** event to ensure we don't do anything until the image has fully made its way across the internet to your browser. Once our image has been loaded, we call the `animate` function via the `loadImage` event handler. The fun is about to start now!

It's Animation Time!

Before we get to the `animate` function, we have a few variables that we should look at first:

```
var shift = 0;
var frameWidth = 300;
var frameHeight = 300;
var totalFrames = 24;
var currentFrame = 0;
```

The variable names kinda hint at what they do, but for now, just know that they exist and pay attention to the default values assigned to them. We'll see all of these variables used really soon.

Now, we get to the `animate` function. This function is responsible for quickly cycling through each sprite in the sprite sheet to create the animation. A bulk of this responsibility lies in the following line that handles which part of the

sprite sheet to display, and it handles where to display it:

```
context.drawImage(myImage, shift, 0, frameWidth, frameHeight,
                  120, 25, frameWidth, frameHeight);
```

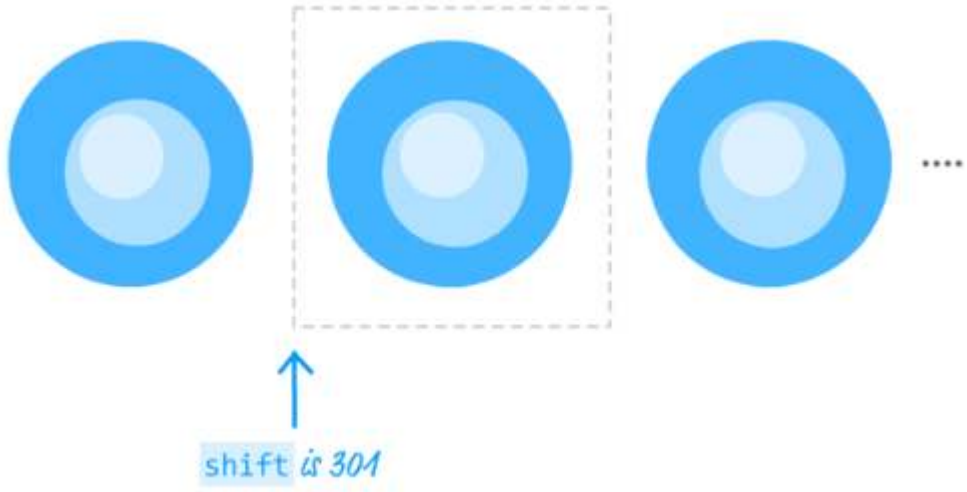
Each sprite is a square that is 300 pixels on each side. There are a bunch of sprites just like this arranged side-by-side, and what we are doing is taking just the first image and displaying it on the screen. To see how, let's look at our `drawImage` code with all of the variables replaced with their actual numerical values:

```
context.drawImage(myImage, 0, 0, 300, 300,
                  120, 25, 300, 300);
```

From our sprite sheet, we grab a 300px by 300px square starting at the (0, 0) mark. That is our first sprite and handled by the first line in our `drawImage` call. We place that grabbed sprite at its original size of 300px by 300px on our `canvas` at the (120, 25) mark.

Because of how `drawImage` works, we don't see any part of the rest of the sprite sheet. We only see the 300 by 300 pixel square we cut out from the sprite sheet and placed in our `canvas`.

To display the next sprite, we tell our `drawImage` method to grab the next sprite from our sprite sheet. Our `shift` variable is responsible for telling `drawImage` where to start looking for the next sprite, so what we need to do is simply adjust the value stored by the `shift` variable:



To move to our next frame, we increase our `shift` variable by 301. The

number of pixels you need to shift to get to the next frame depends entirely

on your sprite sheet. In our case, our sprite sheet has a 1 pixel gap between each sprite. That is why we increment our `shift` variable really awkwardly by both the `frameWidth` and a 1 value:

```
shift += frameWidth + 1;
```

Ignoring the rest of the code for a second, you can now see how our drawImage function works to create the animation. For the second frame, looking at our `drawImage` call with all of the variables replaced with their stored values, you'll see something that looks like the following:

```
context.drawImage(myImage, 301, 0, 300, 300,
                  120, 25, 300, 300);
```

This ensures that our next sprite is taken from the (301, 0) position at the same 300 by 300 pixel size. This taken sprite is then placed at the (120, 25) mark at the original 300 by 300 pixel size. To our users, this will look like a direct replacement of the earlier sprite with a new one that is slightly more rotated.

With every `requestAnimationFrame` call to the `animate` function, we shift over to the next frame in our sprite sheet. We do this shifting by increasing the value of the `shift` variable by 301 each frame. That's it! This automatically ensures `drawImage` is looking at the right part of our sprite sheet and displays the correct sprite. This is all done very rapidly, so what you end up seeing is each frame played back to create a smooth animation.

Ok, we are almost done here. The last thing we are going to look at is some of the code we skipped:

```
if (currentFrame == totalFrames) {
  shift = 0;
  currentFrame = 0;
}
```

The code we skipped is kinda important. We need a way to know when we have reached the end of our sprite sheet so that we can restart our animation from the beginning. The `currentFrame` variable acts as a counter, and the

`totalFrames` variable specifies the number of frames in our sprite sheet. The way you can figure out how many frames you have is by simply counting the number of sprites you have. Some image tools may provide you with that information. If your particular sprite sheet doesn't come with that information, you'll have to manually count...like an animal :P

Anyway, we determine the end by constantly checking when the value of `currentFrame` is the same as `totalFrames`. When both of those variables are equal, it means that we've reached the end of our sprite sheet and it's time to reset everything:

```
if (currentFrame == totalFrames) {
  shift = 0;
  currentFrame = 0;
}
```

By setting the value of `shift` to 0, is we ensure our next `drawImage` call looks at the first frame in our sprite sheet. Setting `currentFrame` to 0 simply resets our counter.

If we are not at the last frame where `currentFrame` is equal to `totalFrames`, then we should go right on and increment the value of `currentFrame`:

```
currentFrame++;
```

And...that's exactly what we do! This ensures we keep an accurate tally of where in the sprite sheet we are, and that helps us pull the plug and reset everything back to the beginning when we've reached the end of our sprite sheet.