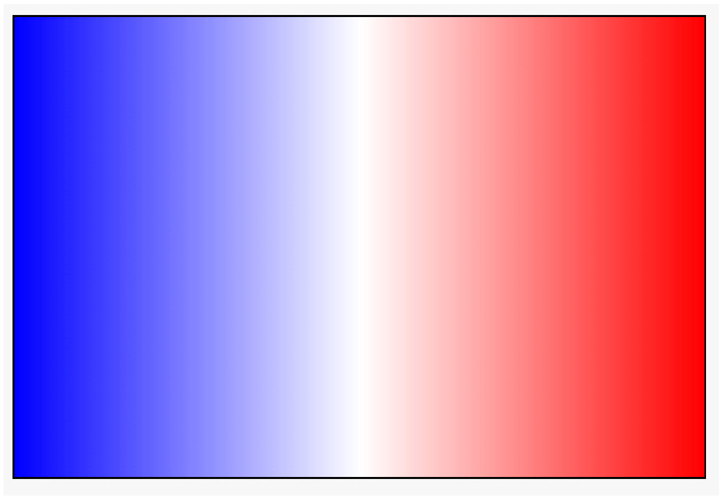
Linear gradients

It is possible to define the stroke or the fill style as a "gradient", a set of interpolated colors, like in this example below (try it online at: http://jsbin.com/ruguxi/2/edit)



The concept of linear gradient is seen as an "invisible" rectangle in which a set of colors are interpolated along a line.

The gradient becomes visible when we draw shapes on top of the invisible gradient, and when the `fillStyle` or `strokeStyle` property has for value this gradient.

Here are the different steps needed:

Step 1: define a linear gradient

Syntax:

```
ctx.createLinearGradient(x0,y0,x1,y1);
```

... where the `(x0, y0)` and `(x1, y1)` parameters define "the direction of the gradient" (as a vector with a starting and an ending point). This direction is an invisible line along which the colors that compose the gradient will be interpolated.

Let's detail an example:

```
grdFrenchFlag = ctx.createLinearGradient(0, 0, 300, 0);
```

This line defines the direction of the gradient: a virtual, invisible line that goes from the top left corner of the canvas (0, 0) to the top right corner of the canvas (300, 0). The interpolated colors will propagate along this line.

If this gradient is going to be reused by different functions, it is a good practice to create/initialize it in a function called when the page is loaded and to store it in a global variable.

Step 2: add a number of "color stops" to this gradient

We will add a set of "colors" and "stops" to this gradient. The stops go from 0 (beginning of the virtual line defined just above), to 1 (end of the virtual line). A color associated with a value of 0.5 will be right in the middle of the virtual line.

Here is an example that corresponds to an interpolated version of the French flag, going from blue to white, then to red, with proportional intervals. We define three colors, blue at position 0, white at position 0.5 and red at position 1:

```
grdFrenchFlag.addColorStop(0, "blue");
grdFrenchFlag.addColorStop(0.5, "white");
grdFrenchFlag.addColorStop(1, "red");
```

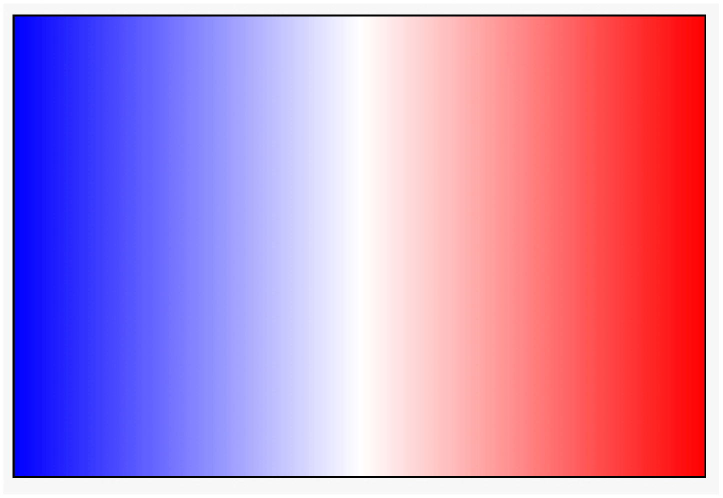
Step 3: draw some shapes

First, let's set the `fillStyle` or `strokeStyle` of the context with this gradient, then let's draw some shapes "on top of the gradient".

In our example, the gradient corresponds to an invisible rectangle that fills the canvas. If we draw a rectangle of the canvas size, it should be filled with the entire gradient:

```
ctx.fillStyle = grdFrenchFlag;
ctx.fillRect(0, 0, 300, 200);
```

The result is shown below: a big rectangle that fills the whole canvas, with colors going from blue (left) to white (middle) to red (right).

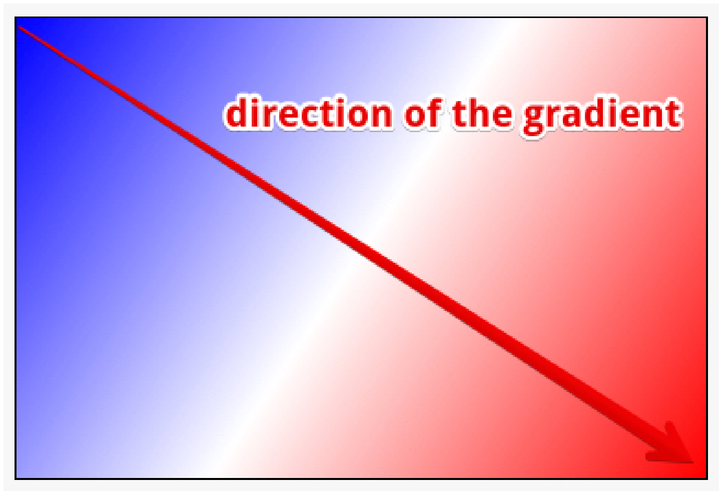


CHANGING THE DIRECTION OF THE GRADIENT

If you modify the source code that defines the direction of the gradient as follows...

```
    grdFrenchFlag = ctx.createLinearGradient(0, 0, 300, 200);
```

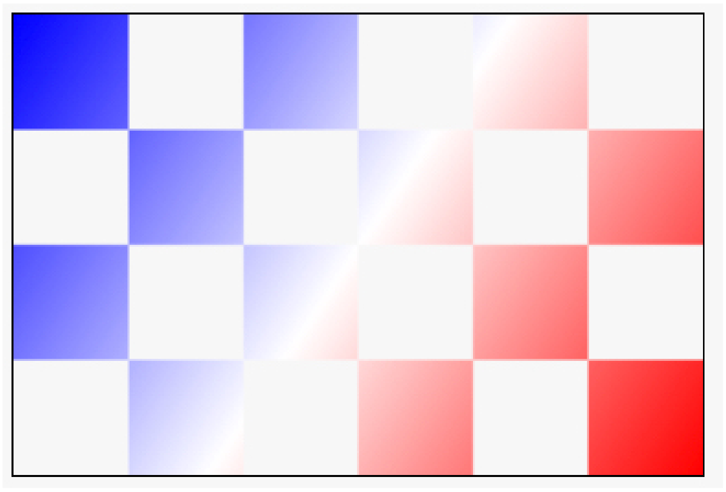
... then you will define a gradient that goes from the top left corner of the canvas to the bottom right of the canvas. Let's see what it does (online version here:http://jsbin.com/ruguxi/4/edit):



DRAWING SHAPES THAT DO NOT COVER THE WHOLE GRADIENT

Instead of drawing a filled rectangle that covers the whole surface of the canvas, let's draw several smaller rectangles.

Online example: http://jsbin.com/baluxa/3/edit



Note that the canvas has its default background color where we did not draw anything. And where we drawn rectangles, we can see "through" and the colors from the gradient are visible.

Here is the code that draws the checkboard:

```
ctx.fillStyle = grdFrenchFlag;
ctx.fillRect(0, 0, 50, 50);
ctx.fillRect(100, 0, 50, 50);
ctx.fillRect(200, 0, 50, 50);
ctx.fillRect(50, 50, 50, 50);
ctx.fillRect(150, 50, 50, 50);
ctx.fillRect(250, 50, 50, 50);
ctx.fillRect(0, 100, 50, 50);
ctx.fillRect(100, 100, 50, 50);
ctx.fillRect(200, 100, 50, 50);
ctx.fillRect(50, 150, 50, 50);
ctx.fillRect(150, 150, 50, 50);
ctx.fillRect(250, 150, 50, 50);
```

This code is rather ugly isn't it? It would have been better to use a loop...

Here is function that draws a chessboard (online example here:http://jsbin.com/dekiji/2/edit):

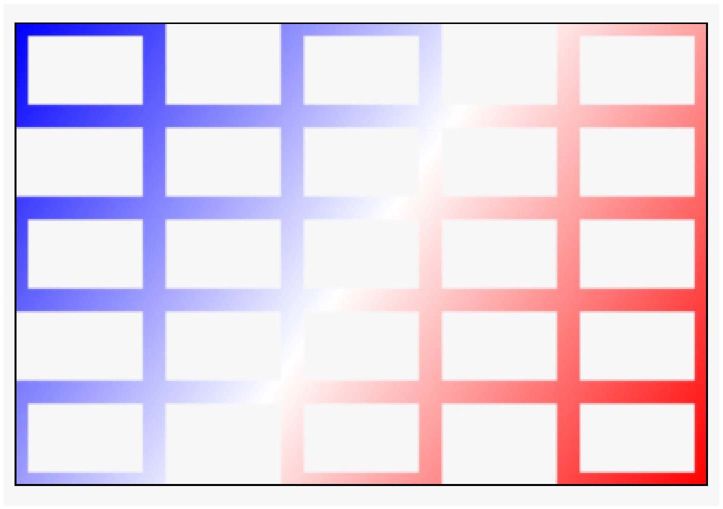
```
// n = number of cells per row/column
function drawCheckboard(n) {
    ctx.fillStyle = grdFrenchFlag;
    var l = canvas.width;
    var h = canvas.height;
    var cellWidth = l / n;
    var cellHeight = h / n;
10.
    for(i = 0; i < n; i+=2) {
      for(j = 0; j < n; j++) {
        ctx.fillRect(cellWidth*(i+j%2), cellHeight*j, cellWidth, cellHeight);
      }
    }
}
```

The two loops (lines 11-15) draw only one cell out of two (see the `i+=2` at line 11), and shift them one cell on the right if the current line is an odd number (the formula`cellWidth * (I+j%2)` does that, when computing the X coordinate of each cell).

This code is much more complex that the previous one, takes 16 lines instead of 13, but is much more powerful. Try to call the function with a value of 10, 20, or 2...

DRAWING OUTLINED SHAPES WITH GRADIENTS

As we used `fillStyle` and `fillRect` for drawing rectangles filled with a gradient, we can also use `strokeStyle` and `strokeRect` in order to draw wireframed rectangles. The next example is just a variation of the previous one, where we just used the `lineWidth`property to set the outline of the rectangles at 5 pixels: http://jsbin.com/dekiji/4/edit



Extract from source code:

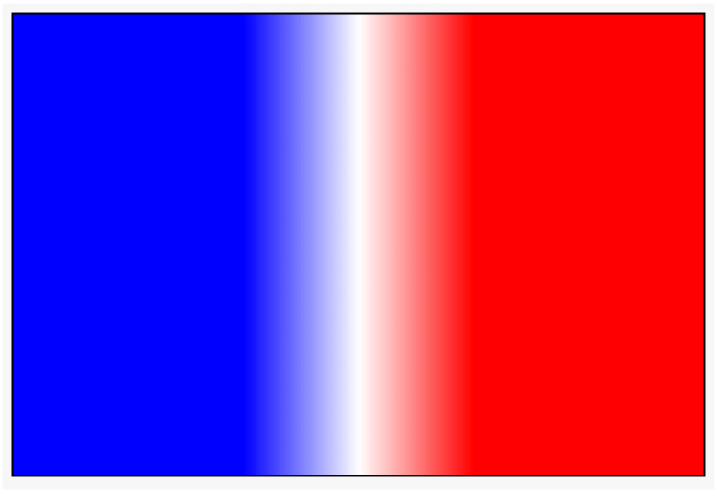
```
function drawCheckboard(n) {
    ctx.strokeStyle = grdFrenchFlag;
    ctx.lineWidth=10;
      ...
    for(i = 0; i < n; i+=2) {
       for(j = 0; j < n; j++) {
          ctx.strokeRect(cellWidth*(i+j%2), cellHeight*j, cellWidth, cellHeight);
       }
    }
}
```

WHAT HAPPENS IF WE DEFINE A GRADIENT SMALLER THAN THE CANVAS?

Let's go back to the very first example of this page. The one with the blue-white-red interpolated French flag. This time we will define a gradient that is smaller. Instead of going from `(0, 0)` to `(300, 0)`, it will go from `(100, 0)` to `(200, 0)`, while the canvas remains the same (`width=300`, `height=200`).

```
grdFrenchFlag = ctx.createLinearGradient(100, 0, 200, 0);
```

Like in the first example we will draw a filled rectangle that is the same size of the canvas. Here is the online version: http://jsbin.com/ruvuta/1/edit, and here is a screenshot of the result:



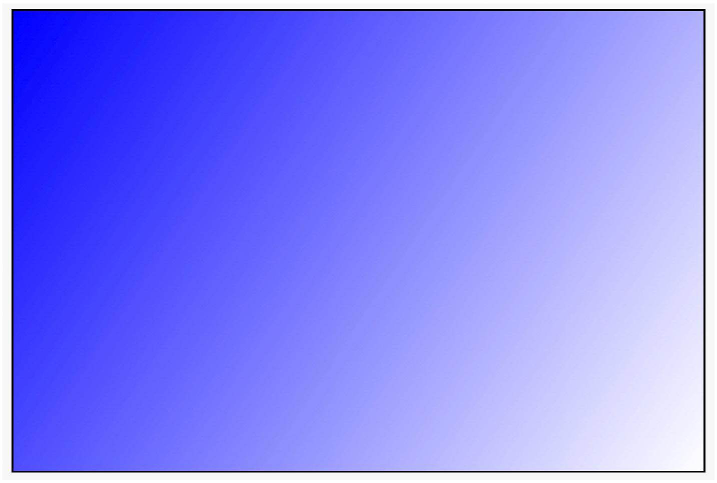
We notice that "before" the gradient start, the first color of the gradient is repeated without any interpolation (columns 0-100 are all blue), then we "see through" and the gradient is drawn (columns 100-200), then the last color of the gradient is repeated without any interpolation (columns 200-300 are red).

WHAT HAPPENS IF WE DEFINE A GRADIENT BIGGER THAN THE CANVAS?

Nothing special, we will "see through the drawn shapes" and parts of the gradient that are located in the canvas area will be shown. You can try this example that defines a gradient twice bigger than the canvas:

```
    grdFrenchFlag = ctx.createLinearGradient(0, 0, 600, 400);
```

And if we draw the same rectangle with the canvas size, here is the result:



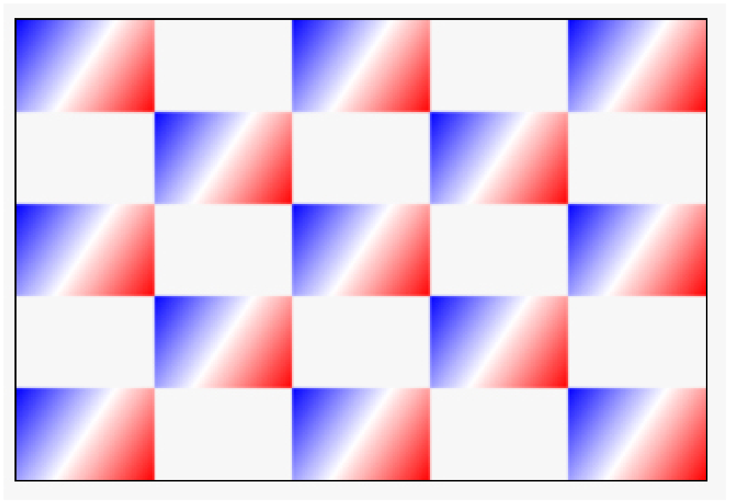
The red color is far away further than the bottom right corner.... we see only the first top left quarter of the gradient.

DRAW SHAPES THAT SHARE THE SAME GRADIENT AS A WHOLE

This time, we would like to draw the chessboard with the gradient in each cell. How can we do this with one single gradient?

We can't! At least we can't without recreating it for each cell!

It suffices to create a new gradient before drawing each filled rectangle, and set it with the starting and ending point of its direction/virtual line accordingly to the rectangle coordinates. Here is an online example and the resulting display:



Extract from source code:

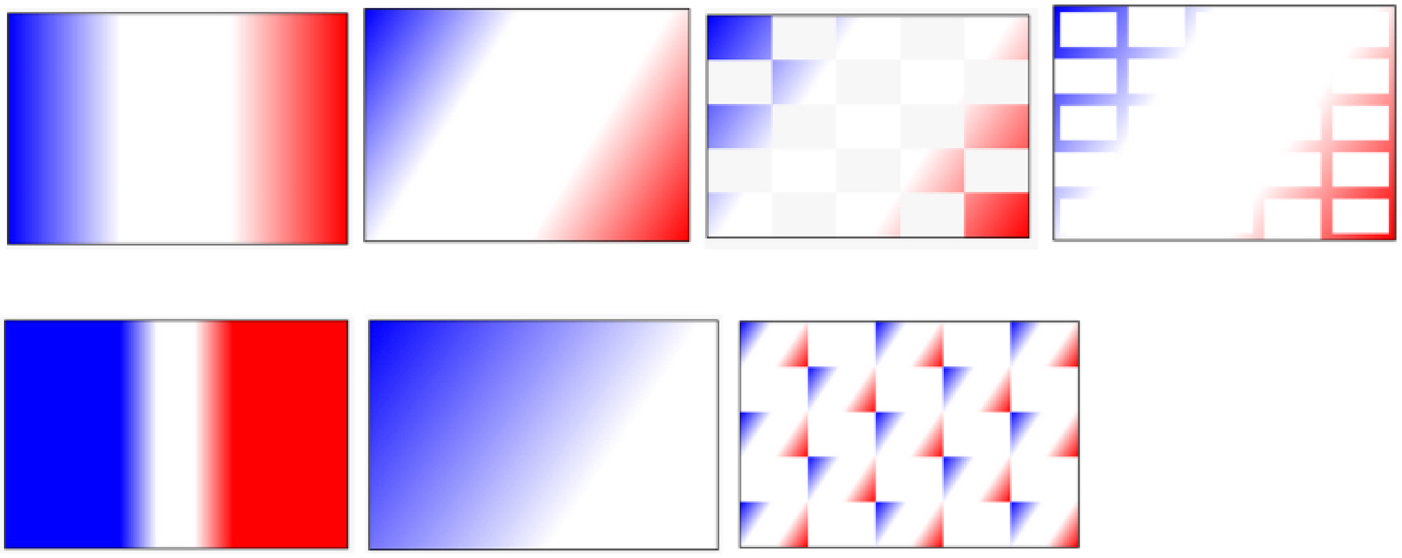
```
function setGradient(x, y, width, height) {
   grdFrenchFlag = ctx.createLinearGradient(x, y, width, height);
   grdFrenchFlag.addColorStop(0, "blue");
   grdFrenchFlag.addColorStop(0.5, "white");
   grdFrenchFlag.addColorStop(1, "red");
   // set the new gradient to the current fillStyle
   ctx.fillStyle = grdFrenchFlag;
}

// n = number of cells per row/column
function drawCheckboard(n) {
  var l = canvas.width;
  var h = canvas.height;

  var cellWidth = l / n;
  var cellHeight = h / n;
  for(i = 0; i < n; i+=2) {
    for(j = 0; j < n; j++) {
      var x = cellWidth*(i+j%2);
      var y = cellHeight*j;
      setGradient(x, y, x+cellWidth, y+cellHeight);
      ctx.fillRect(x, y, cellWidth, cellHeight);
    }
  }
}
```

We wrote a function `setGradient(startX, startY, endX, endY)` that creates a gradient and set the `fillStyle` context property so that any filled shape drawn will be with this gradient.

In the `drawCheckBoard(...)` function we call it just before drawing rectangles. Like that, each rectangle is drawn using its own gradient.

THIS FLAG IS NOT REALLY LOOKING LIKE THE FRENCH FLAG, ISN'T IT?

Indeed the French flag in these images is more accurate:



We slightly modified the examples of this chapter so that the flag looks more like the French flag. Look at these modified versions below, and try to find out what has changed in the gradient definitions:

- http://jsbin.com/zefogoneko/1/edit
- http://jsbin.com/bocogowuku/1/edit
- http://jsbin.com/dafufuwunu/1/edit
- http://jsbin.com/runuzamiva/1/edit
- http://jsbin.com/yocosocowe/1/edit
- http://jsbin.com/tuwogokovi/1/edit
- http://jsbin.com/jemarudica/1/edit

KNOWLEDGE CHECK 3.5.2

```
grdFrenchFlag = ctx.createLinearGradient(0, 0, 300, 0);
```

```
grdFrenchFlag.addColorStop(0, "blue");
grdFrenchFlag.addColorStop(0.5, "white");
grdFrenchFlag.addColorStop(1, "red");
```

The gradient above defines...

- ○ A gradient that goes from (0, 0) to (300, 0), defining an invisible line. Colors will be interpolated horizontally along this line. Shapes drawn in the canvas between X=0 and X=300 will be drawn using interpolated colors defined by this gradient.
- ○ Same as above but colors will be interpolated diagonally.