

# Visualización e Interfaces

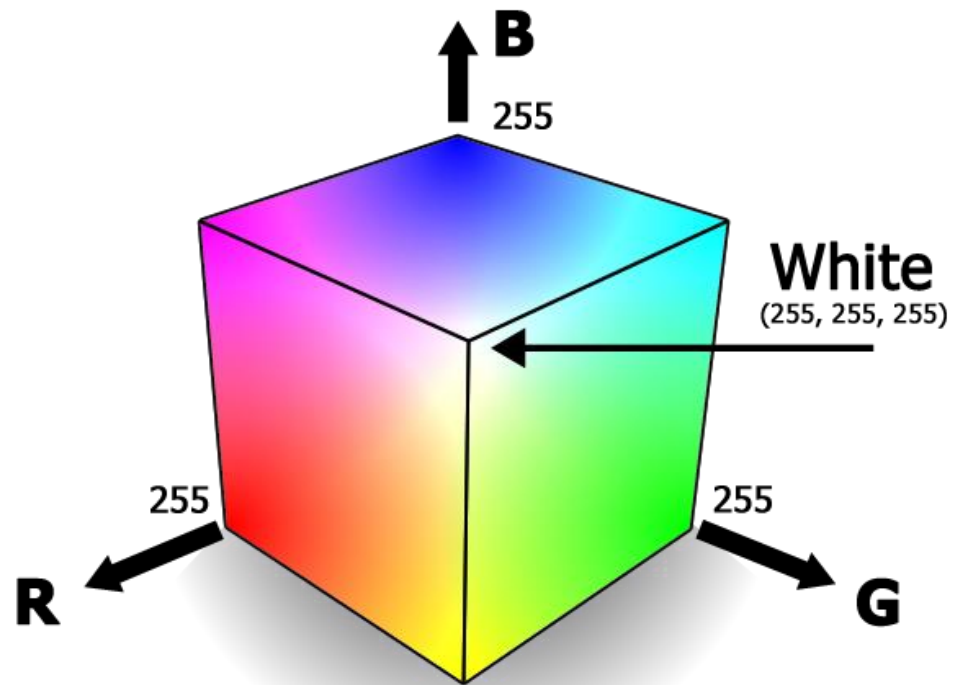
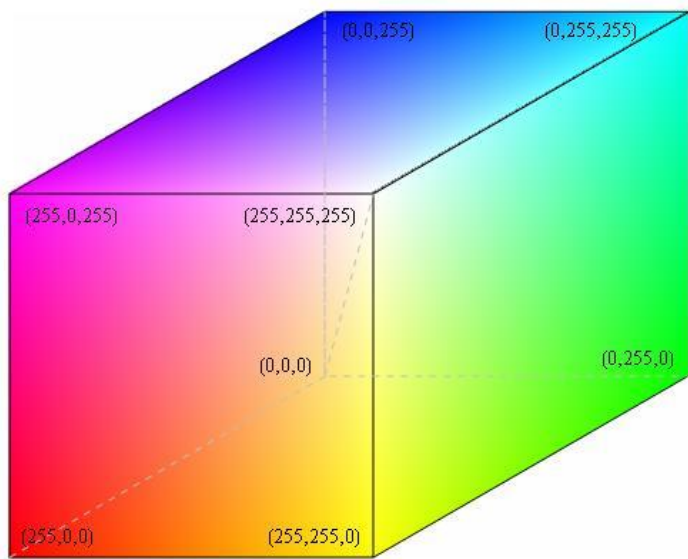
## Interfaces de usuario e Interacción

Clase 1

# Colores

- ▶ Distintos sistemas de representación (RGB, YUV, HSV, HSL, CMYK)
- ▶ RGBA utiliza 4 componentes
  - Red
  - Green
  - Blue
  - Alpha (Transparencia)
- ▶ Un pixel de una imagen se representa con el modelo RGB con 3 bytes (0..255)
- ▶ Se puede definir de dos formas:
  - Máscara hexadecimal:
    - *int color = RED << 16 || GREEN << 8 || BLUE*
    - Puede escribirse como “#RRGGBB” valores Hexa
  - Función de Javascript
    - ‘rgba(vRed, vGreen, vBlue, vAlpha)’

# Colores



# Dibujando en HTML<sub>5</sub>

- **Canvas** es el elemento donde se dibuja en HTML5
- **Context** provee la funcionalidad para dibujar
- Cada elemento visual de la página tiene asociado un **Context**.

# Ejemplo de uso de Canvas

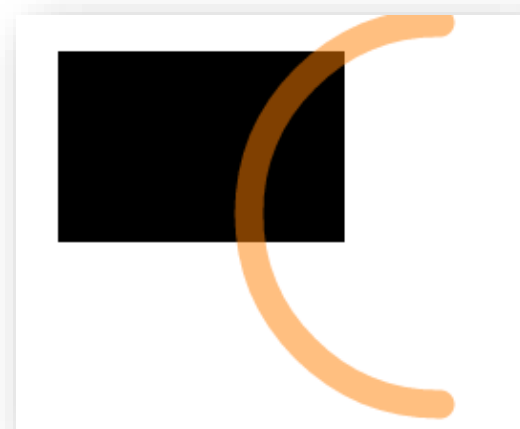
```
<canvas id="canvas" width="900" height="600" style="border:#000 solid 1px;" />
</body>
```

```
<script>
```

```
var ctx = document.getElementById("canvas").getContext("2d");
ctx.fillStyle = "#000000";
ctx.fillRect(250, 25, 150, 100);
ctx.beginPath();
ctx.arc(450, 110, 100, Math.PI * 1/2, Math.PI * 3/2);
ctx.lineWidth = 15;
ctx.lineCap = 'round';
ctx.strokeStyle = 'rgba(255, 127, 0, 0.5)';
ctx.stroke();
```

```
</script>
```

```
</html>
```



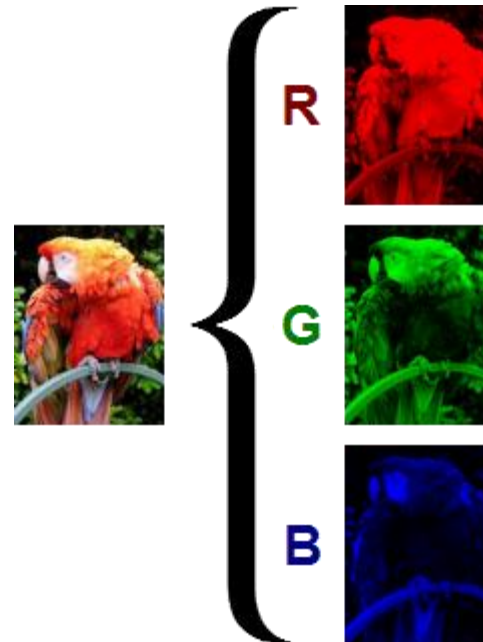
# Imágenes

- ▶ Una imagen Digital es una matriz 2D (ancho x alto) de *pixeles*.

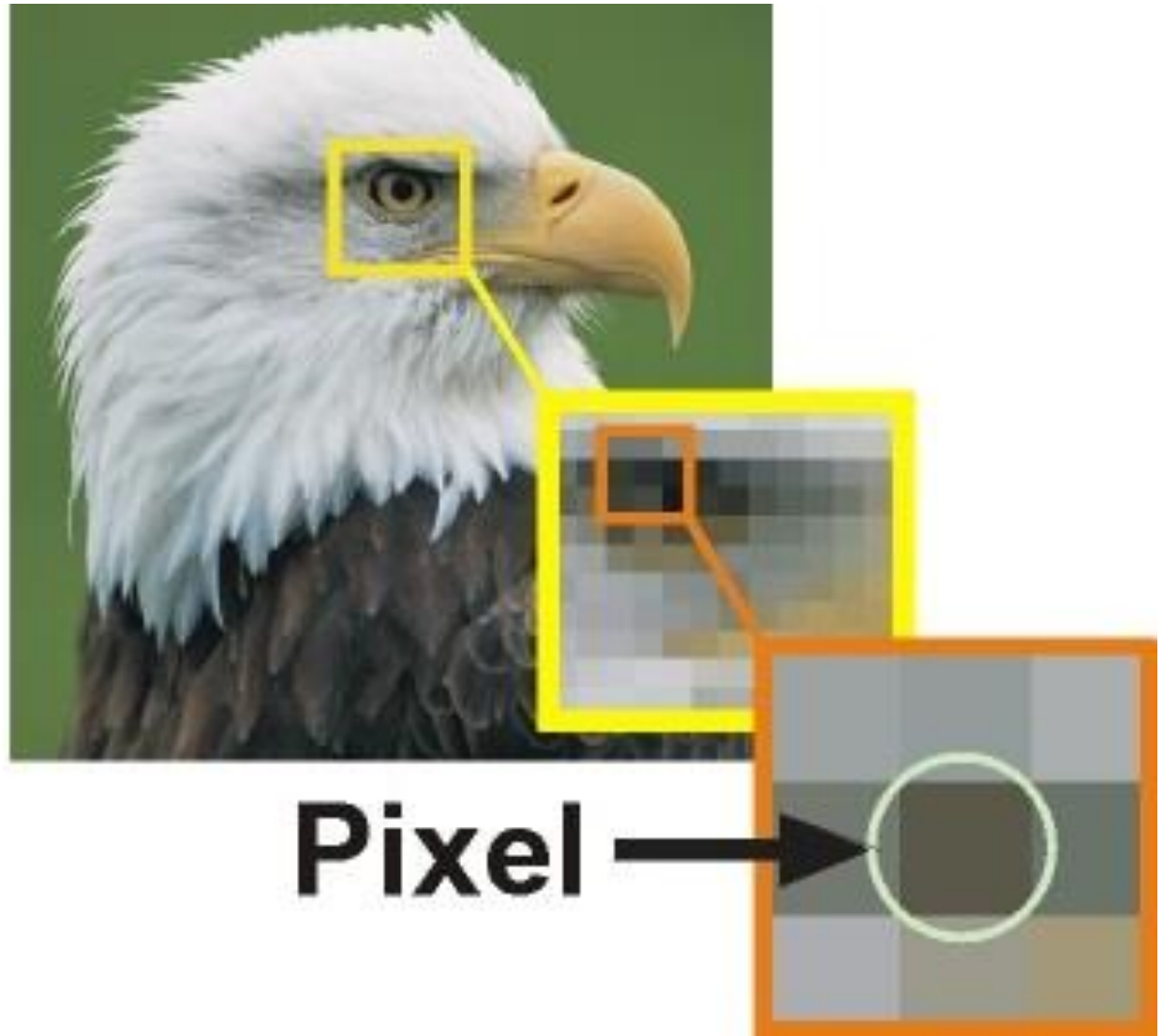
## *Escala grises*



## *Color*



# Imágenes





# Representar imágenes

- ▶ HTML5 provee un “*ImageData*”
- ▶ La imagen se recorre en *Ancho* y *Alto*
- ▶ “*ImageData*” almacena los pixeles en un arreglo de 1d
- ▶ Los colores se almacenan en un *Array de enteros* y pueden acceder como si fueran una matriz
- ▶ Para dibujar la Imagen en Pantalla
  - `ctx.putImageData ( Imagen, x, y ) ;`
- ▶ Para convertir de matriz a arreglo?

***indice = (x + y \* imageData.width) \* 4;***



# Generar una imagen

```
var imageData = ctx.createImageData(width, height);

for (x=0; x<width; x++){
    for (y=0; y<height; y++){
        setPixel(imageData, x, y, r, g, b, a);
    }
}

ctx.putImageData(imageData, 0, 0);
```

.....

```
function setPixel(imageData, x, y, r, g, b, a)
{
    index = (x + y * imageData.width) * 4;
    imageData.data[index+0] = r;
    imageData.data[index+1] = g;
    imageData.data[index+2] = b;
    imageData.data[index+3] = a;
}
```

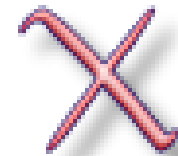
# Cómo cargar una Imagen

- ▶ La carga de imágenes es **asincrónica**
- ▶ El script se ejecuta secuencialmente línea por línea
- ▶ El tiempo de demora depende del tamaño de la imagen y de la latencia del servidor que la tiene

```
var imagen1 = new Image();
```

```
imagen1.src = "imagen.jpg";
```

```
ctx.drawImage(imagen1, 0, 0);
```



Puede que la imagen no esté cargada en memoria al momento de dibujarla

# Cargar una imagen y dibujarla


```
var image1 = new Image();  
image1.src = "imagen.jpg";
```

Ubicación  
del recurso




```
image1.onload = function() {  
    myDrawImageMethod(this);  
}
```

Evento de  
finalización de  
la carga del  
recurso



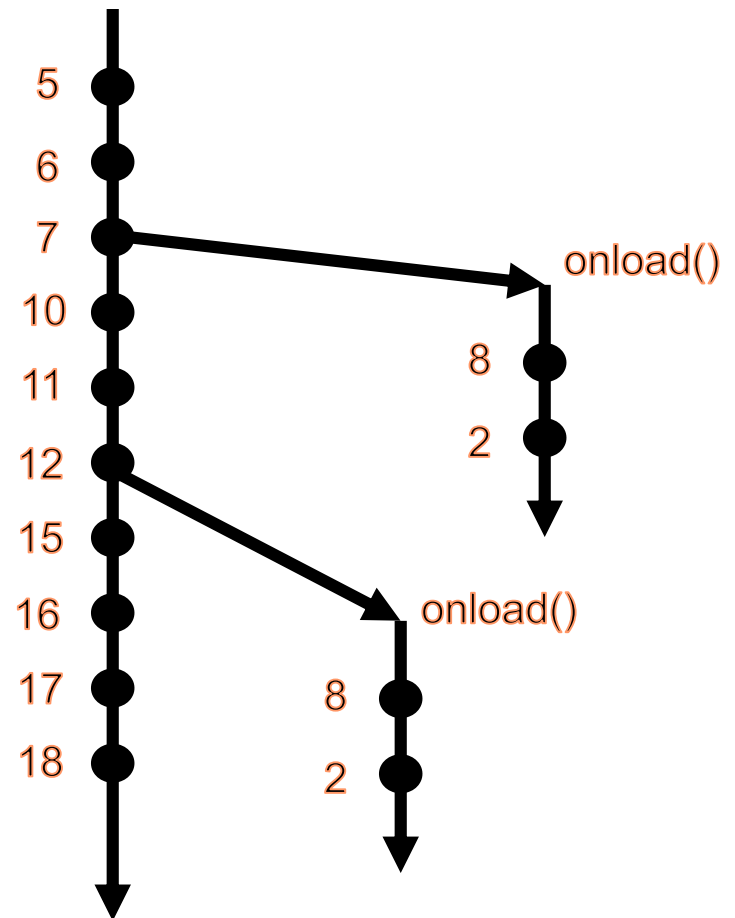
```
function myDrawImageMethod(image) {  
    ctx.drawImage(image, 0, 0);  
}
```

Dibuja la  
imagen usando  
el contexto del  
HTML5



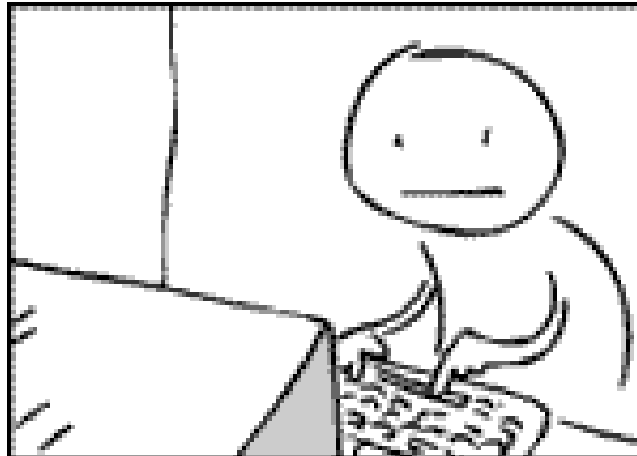
# Por qué se usa .onload() ?

```
1  function myDrawImageMethod(image) {  
2      ctx.drawImage(image, 0, 0);  
3  }  
4  
5  var image1 = new Image();  
6  image1.src = "imagen1.jpg";  
7  image1.onload = function() {  
8      myDrawImageMethod(this);  
9  }  
10  
11 var image2 = new Image();  
12 image2.src = "imagen2.jpg";  
13 image2.onload = function() {  
14     myDrawImageMethod(this);  
15 }  
16  
17 // Otra linea de código 1  
18 // Otra linea de código 2  
19 // Otra linea de código 3  
20 // Otra linea de código 4
```



# Ejercicio

- ▶ Cargar una imagen almacenada en la PC
- ▶ Dibujar la imagen sobre el canvas



# Acceder a los pixeles

- ▶ **ImageData** almacena los pixeles en un arreglo de 1D
- ▶ El **Context** provee métodos para acceder a los pixeles que ya se dibujaron en el contexto:
- ▶ Entonces, para acceder a los pixeles de una imagen...  
¡Primero tengo que dibujar la imagen!

# Acceder a los pixeles

```
var imageData;  
  
var image1 = new Image();  
image1.src = "image.jpg";  
  
image1.onload = function() {  
    ctx.drawImage(this, 0, 0);  
    imageData = ctx.getImageData(0, 0, this.width, this.height);  
    // Código para modificar pixeles ...  
    ctx.putImageData(imageData, 0, 0);  
}
```

**IMPORTANTE:** Por políticas de seguridad, Chrome **NO** permite acceder directamente a los datos de la imagen.



# Acceder a los pixeles

- ▶ ¿Cómo accedo a cada pixel?
- ▶ ¿Cómo obtengo las componentes R, G y B?

```
function getRed(imageData, x, y) {  
    // ?????????????????????????????????????????????  
    // ?????????????????????????????????????????????  
}
```

```
function getGreen(imageData, x, y) {  
    // ?????????????????????????????????????????????  
    // ?????????????????????????????????????????????  
}
```

```
function getBlue(imageData, x, y) {  
    // ?????????????????????????????????????????????  
    // ?????????????????????????????????????????????  
}
```

# Acceder a los pixeles

- ▶ ¿Cómo accedo a cada pixel?
- ▶ ¿Cómo obtengo las componentes R, G y B?

```
function getRed(imageData, x, y) {  
    index = (x + y * imageData.width) * 4;  
    return imageData.data[index+0];  
}
```

```
function getGreen(imageData, x, y) {  
    index = (x + y * imageData.width) * 4;  
    return imageData.data[index+1];  
}
```

```
function getBlue(imageData, x, y) {  
    index = (x + y * imageData.width) * 4;  
    return imageData.data[index+2];  
}
```

# Filtros

- ▶ Modifica el valor de un pixel dado una ecuación matemática
- ▶ Este valor puede ser simplemente un coeficiente

$$\textit{Blanco y Negro} = (R + G + B) / 3$$

