CSC4130
Introduction to Human-Computer Interaction

Lecture 7

User Interface Technology: JavaScript

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Outline

- JavaScript
- SVG and Canvas
- Interaction

Outline

- JavaScript
- SVG and Canvas
- Interaction

JavaScript

- One of the core technologies of the World Wide Web
- Create dynamic and interactive web-based applications and systems
- Simple and easy to learn

JavaScript variables

- Create variables
  - var x (global variable)
  - let x (local variable)
  - const x = 6
- Use variables
  - x = 6
  - y = x+6
- Print variables
  - console.log(x)

x = “HCI” + 16 + 4

x = 16 + 4 + “HCI”

JavaScript array

- A collection of values
- Each value can occur multiple times in an array
- Initialization
  - const c = [1,2,3]
  - const c = [ ] c[0] = 1 c[1] = 1
  - const c = new Array(1,2,3)

Not suggested

Do these two initializations generate the same result?

const points = [40]

const points = new Array(40)

What about these two?

const points = [40,30]

const points = new Array(40,30)

JavaScript array

- Access
  - c[2]
- Add/remove element
  - c.push(1)
  - c.pop()
- Get the size
  - c.length
- Sort
  - c.sort()

JavaScript array

- Map function
  - array.map(function_name)
- Reduce function
  - array.reduce(function_name,init_value)
- Filter function
  - array.filter(function_name)

```
const numbers = [65, 44, 12, 4];
const newArr = numbers.map(myFunction)
function myFunction(num) {
  return num * 10;
}
```

```
let num = [5, 9, 12, 24, 67]
let sum = num.reduce(function (accumulator, curValue) {
  return accumulator + curValue
}, 0)
```

```
const ages = [32, 33, 12, 40];
function checkAge(age) {
  return age > document.getElementById("ageToCheck").value;
}
function myFunction() {
  document.getElementById("demo").innerHTML = ages.filter(checkAge);
}
```

JavaScript set

- A collection of unique values
- Each value can only occur once in a Set
- Initialization
  – New Set()
  – New Set([“a”,”n”,”c”])
- Add element
  – .add()
- Get the size
  – .size

JavaScript map

- Hold key-value pairs where the keys can be any datatype
- Remember the original insertion order of the keys
- Initialization
  – map = New Map()
  – map = New Map([[1,2],[3,4]])
- Access
  – map.get(1)
- Check
  – map.has(1)

JavaScript map

- Add element
  - map.set(4,5)
- Remove element
  - Map.delete(1)
  - Map.clear()
- Get the size
  - Map.size

What will happen if you pass an existing key

JavaScript object

- Dictionary type of data collection — which follows key-value stored concept like Map
- Each key in Object is also unique and associated with a single value
- Initialization
  - obj = {}
  - obj = {key1:3, key2:4}
  - obj = new Object()
  - obj = object.create(null)

Not suggested

JavaScript object

```
var obj = new Object(id: 1, name: "test") //Error - obviously
var obj1 = {id: 1, name: "test"};
var obj2 = new Object(obj1); //obj1 and obj2 points to the same one
obj2.id = 2;
console.log(obj1.id); //2
```

JavaScript object

- Access
  - obj.id
  - obj[‘id’]
- Check
  - isExist = obj.id === undefined
  - isExist = 'id' in obj
- Add element
  - obj.gender
  - obj[‘gender’]

JavaScript object

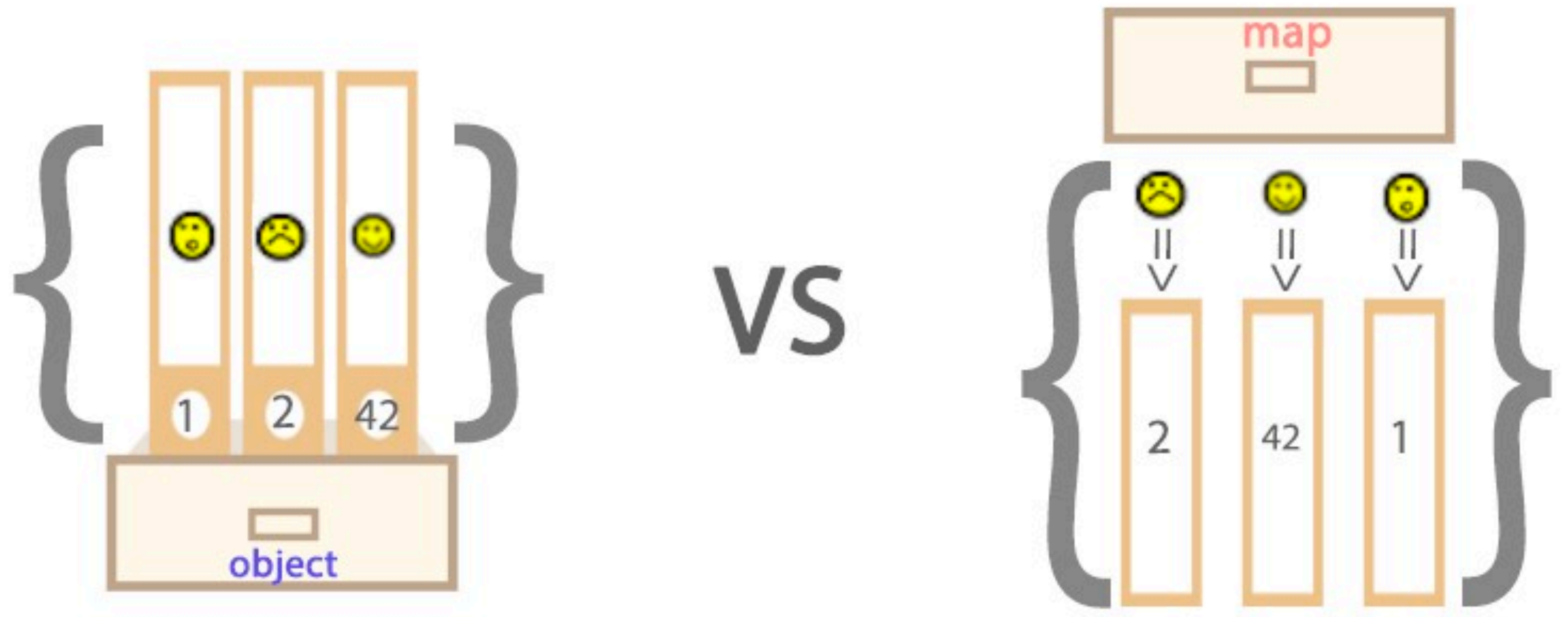
- Remove/delete element
  - delete obj.id
  - obj.id = undefined
- Get the size
  - Object.keys(obj).length

JavaScript array vs. object

- Arrays are a special type of objects
- Use objects when you want the element names to be strings (text)
- Use arrays when you want the element names to be numbers

JavaScript object vs. map

- In Object, the data-type of the key-field is restricted to integer, strings, and symbols. Whereas in Map, the key-field can be of any data-type (integer, an array, even an object)
- In the map, the original order of elements is preserved
- The map is an instance of an object but the vice-versa is not true



JavaScript object vs. map

- Object is the great choice for scenarios when we only need simple structure to store data and knew that all the keys are either strings or integers (or Symbol), because creating plain Object and accessing Object’s property with a specific key is much faster than creating a Map
- JSON has direct support for Object, but not with map (yet). So in certain situation where we have to work a lot with JSON, consider Object as preferred option
- In scenarios that requires a lot of adding and removing (especially) new pair, Map may perform much better

JavaScript object vs. map

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

- Map preserves the order of its keys — unlike object, and Map was built with iteration in mind, so in case iteration or elements order are highly significant, consider Map — it will ensure stable iteration performance in all browsers
- Map tends to perform better in storing large set of data, especially when keys are unknown until run time, and when all keys are the same type and all values are the same type

JavaScript function

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

```
let x = myFunction(4, 3);   // Function is called, return value will end up in x

function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

JavaScript function

```
let num = [5, 9, 12, 24, 67]
let sum = num.reduce(myfunction, 0)
function myfunction (accumulator, curValue) {
  return accumulator + curValue
}
```

```
let num = [5, 9, 12, 24, 67]
let sum = num.reduce(function (accumulator, curValue) {
  return accumulator + curValue
}, 0)
```

```
let num = [5, 9, 12, 24, 67]
let sum = num.reduce((accumulator, curValue) => accumulator + curValue, 0)
```

JavaScript loop

- For loop
- While loop
- Do loop

```
for (i=0; i<10; ++i) {
    console.log(i);
}
```

```
i = 3;
while (i<100) {
    console.log(i);
    i = i * 2;
}
```

```
i = 3;
do {
        console.log(i);
        i = i * 2;
} while (i<100);
```

JavaScript for loop

- For … of

```
// array
const students = ['John', 'Sara', 'Jack'];

// using for...of
for ( let element of students ) {

    // display the values
    console.log(element);
}
```

```
let map = new Map();

// inserting elements
map.set('name', 'Jack');
map.set('age', '27');

// looping through Map
for (let [key, value] of map) {
    console.log(key + '- ' + value);
}
```

JavaScript switch

```
i = "some case";
switch (i) {
case "string literals ok":
    console.log("Yes");
    break;
case "some case":
    console.log("Unlike C");
    break;
}
```

JavaScript practices

- Write a procedure that takes an array as a parameter, iterates over every object in that array, and prints the value of the field “foo” to the console

```
var objects = [{foo: 3, bar: "abc"},
               {foo: 5, bar: "def"}];
```

JavaScript practices

- Write a procedure that takes an array as a parameter, iterates over every object in that array, and prints the value of the field “foo” to the console

```
var objects = [{foo: 3, bar: "abc"},
               {foo: 5, bar: "def"}];


function printobject(array){
    for (var i=0;i<array.length;i++){
        console.log(objects[i].foo);
    }
}
```

JavaScript practices

- Write a procedure that takes an array as a parameter, iterates over every object in that array and returns a new array with all the values of the field “foo”

JavaScript practices

- Write a procedure that takes an array as a parameter, iterates over every object in that array and returns a new array with all the values of the field “foo”

```
function printobject(array){
    let new_array = [];
    for (var i=0;i<array.length;i++){
        new_array.push(array[i].foo);
    }
    return new_array;
}
```

JavaScript class

- Use the keyword class to create a class
- Always add a method named constructor()
  - It has to have the exact name “constructor"
  - It is executed automatically when a new object is created
  - It is used to initialize object properties

```
myCar1 = new Car("Ford", 2014);
```

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
```

JavaScript constructor function

```
class Animal {
  name = 'animal';

  constructor() {
    alert(this.name); //
(*)
  }
}

class Rabbit extends
Animal {
  name = 'rabbit';
}

new Animal();
new Rabbit();
```

```
class Animal {
  showName() {  // instead of this.name = 'animal'
    alert('animal');
  }

  constructor() {
    this.showName(); // instead of alert(this.name);
  }
}

class Rabbit extends Animal {
  showName() {
    alert('rabbit');
  }
}

new Animal();
new Rabbit();
```

JavaScript class

- Also, use prototype to create a class

```
// 1. Create constructor function
function Car(name, year) {
  this.name = name;
  this.year = year;
}
// a function prototype has "constructor" property by default,
// so we don't need to create it

// 2. Add the method to prototype
Car.prototype.sayHi = function() {
  alert(this.name);
};

// Usage:
let ford = new Car(“Ford”, “2014”);
ford.sayHi();
```

JavaScript inheritance

- To create a class inheritance, use the extends keyword
- A class created with a class inheritance inherits all the methods from another class

```
class Person {
    constructor(name) {
        this.name = name;
    }

    greet() {
        console.log(`Hello ${this.name}`);
    }
}
```

```
class Student extends Person {

}
```

```
Student1 = Object.create(Person);
```

JavaScript inheritance

- To create a object inheritance, use _proto_
- A object created with a object inheritance inherits all the methods and attributes from another object

```
let animal = {
  eats: true
};
let rabbit = {
  jumps: true
};

rabbit.__proto__ = animal; // (*)

// we can find both properties in rabbit now:
alert( rabbit.eats ); // true (**)
alert( rabbit.jumps ); // true
```

```
let animal = {
  eats: true,
  walk() {
    alert("Animal walk");
  }
};

let rabbit = {
  jumps: true,
  __proto__: animal
};

let longEar = {
  earLength: 10,
  __proto__: rabbit
};

// walk is taken from the prototype chain
longEar.walk(); // Animal walk
alert(longEar.jumps); // true (from rabbit)
```

JavaScript inheritance

```
let user = {
  name: "John",
  surname: "Smith",

  set fullName(value) {
    [this.name, this.surname] = value.split(" ");
  },

  get fullName() {
    return `${this.name} ${this.surname}`;
  }
};

let admin = {
  __proto__: user,
  isAdmin: true
};

alert(admin.fullName); // John Smith (*)

// setter triggers!
admin.fullName = "Alice Cooper"; // (**)

alert(admin.fullName); // Alice Cooper, state of admin modified
alert(user.fullName); // John Smith, state of user protected
```

JavaScript override

- If a child class has the same method or property name as that of the parent class, it will use the method and property of the child class. This concept is called method overriding

```
class Person {
    constructor(name) {
        this.name = name;
        this.occupation = "unemployed";
    }

    greet() {
        console.log(`Hello ${this.name}.`);
    }

}
```

```
class Student extends Person {

    constructor(name) {

        // call the super class constructor and pass in the name pa
        super(name);

        // Overriding an occupation property
        this.occupation = 'Student';
    }

    // overriding Person's method
    greet() {
        console.log(`Hello student ${this.name}.`);
        console.log('occupation: ' + this.occupation);
    }
}
```

JavaScript public vs. protected vs. private

- Public: accessible from anywhere
- Protected: accessible only from its inherited classes. Prefixed with an underscore _
- Private: accessible only from inside the class. Start with #

JavaScript public

- Public: accessible from anywhere

```
class CoffeeMachine {
  waterAmount = 0; // the amount of water inside

  constructor(power) {
    this.power = power;
    alert( `Created a coffee-machine, power: ${power}` );
  }

}
// create the coffee machine
let coffeeMachine = new CoffeeMachine(100);

// add water
coffeeMachine.waterAmount = 200;
```

JavaScript public vs. protected vs. private

- Protected: accessible only from the inherited classes. Prefixed with an underscore _

```
class CoffeeMachine {
  _waterAmount = 0;

  set waterAmount(value) {
    if (value < 0) {
      value = 0;
    }
    this._waterAmount = value;
  }

  get waterAmount() {
    return this._waterAmount;
  }

  constructor(power) {
    this._power = power;
  }
}
// create the coffee machine
let coffeeMachine = new CoffeeMachine(100);
// add water
coffeeMachine.waterAmount = -10; // _waterAmount will become 0, not -10
```

JavaScript public vs. protected vs. private

- Private: accessible only from inside the class. Start with #

```
class CoffeeMachine {
  #waterLimit = 200;

  #fixWaterAmount(value) {
    if (value < 0) return 0;
    if (value > this.#waterLimit) return this.#waterLimit;
  }

  setWaterAmount(value) {
    this.#waterLimit = this.#fixWaterAmount(value);
  }

}
let coffeeMachine = new CoffeeMachine();

// can't access privates from outside of the class
coffeeMachine.#fixWaterAmount(123); // Error
coffeeMachine.#waterLimit = 1000; // Error
```

Outline

- JavaScript
- SVG and Canvas
- Interaction

Scalable vector graphics (SVG)

- A procedure-based way for drawing graphics content
- “Vector” graphics refers to graphical systems that tare specified independent of coordinates, and can thus be draw and zoomed with no artifacts
- Compare with “Raster” graphics (include typical image formats like .jpg and .png) that just specify an array of pixels

SVG basics

- In html, one encodes the instructions directly with the svg

```
<svg width="…" height="…">
… instructions…
</svg>
```

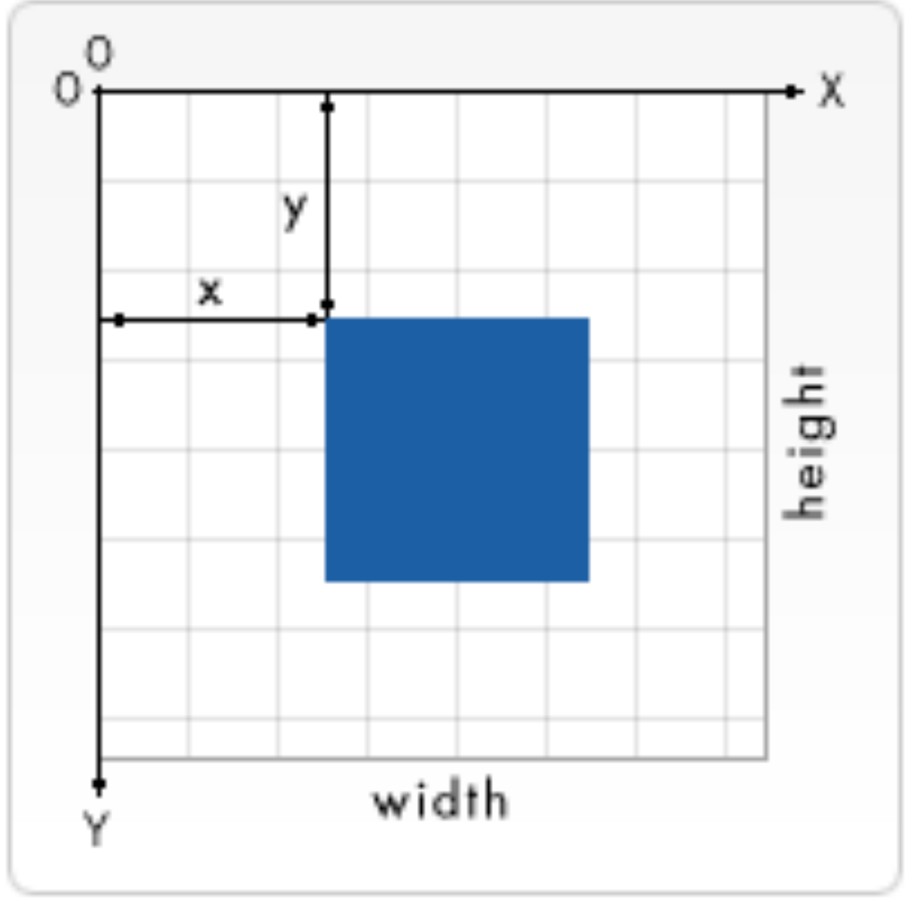
- Instructions provide commands draw many simple shapes (circles, ellipses, rectangles, lines, path, text, …) included as a set of tags (called nodes or elements)
- Each type of node has a different set of key defining attributes (e.g., a circle must define it's center position (cx,cy) and radius (r)

SVG style

- Can apply style (like with CSS type styles), but many of properties have different names from the usual ones for HTML tags
- Refer to https://oreillymedia.github.io/Using_SVG/guide/style.html for details

Drawing in SVG

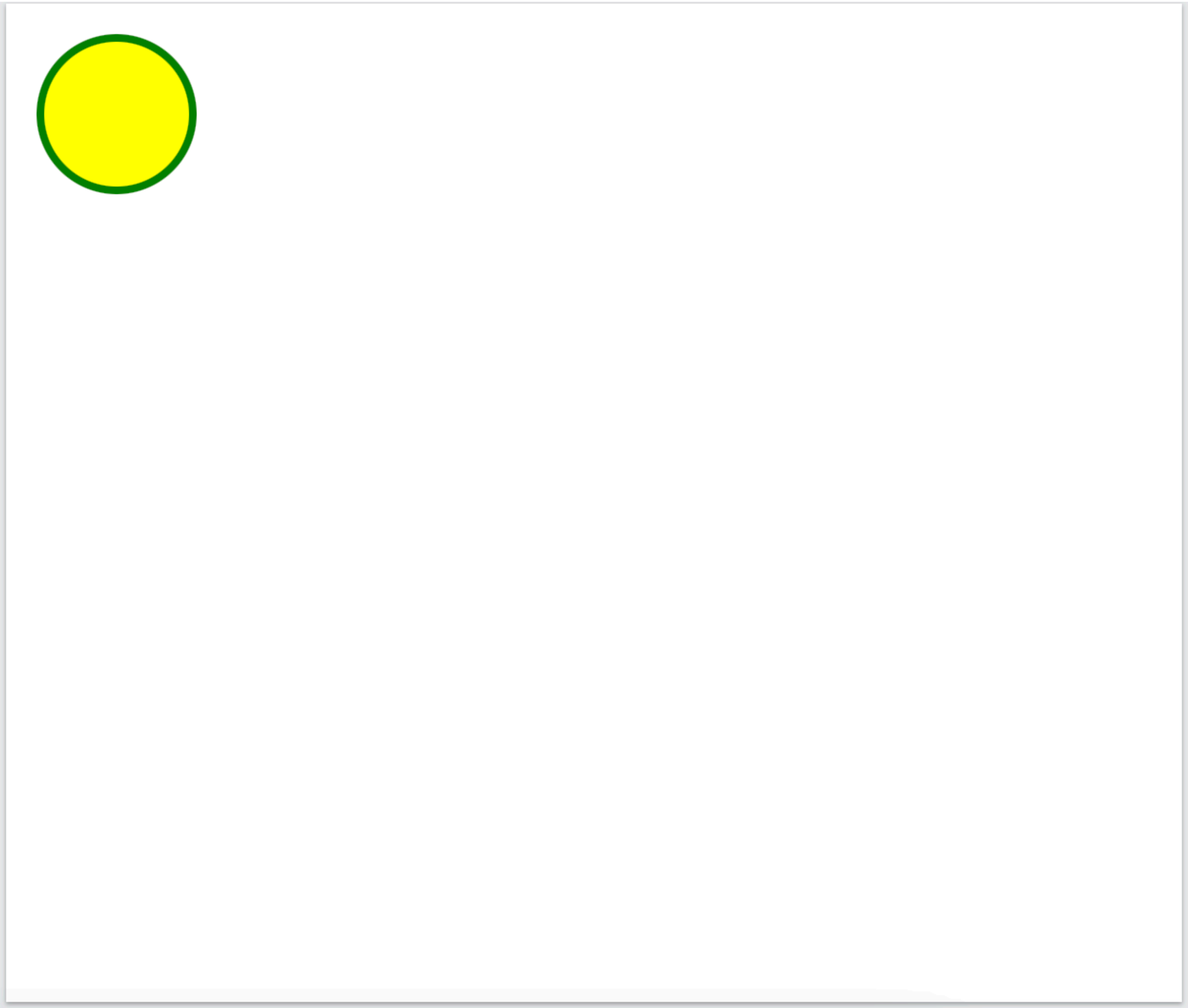
- Instruction are applied one-by-one, and new tags are drawn on top of existing ones
- Use a two-dimensional coordinate system to specify most drawing
  – Note that top-left corner is (0,0)
- Can apply various transformations using the `transform` attribute, this is particular useful if one groups elements using the svg group node <g></g>



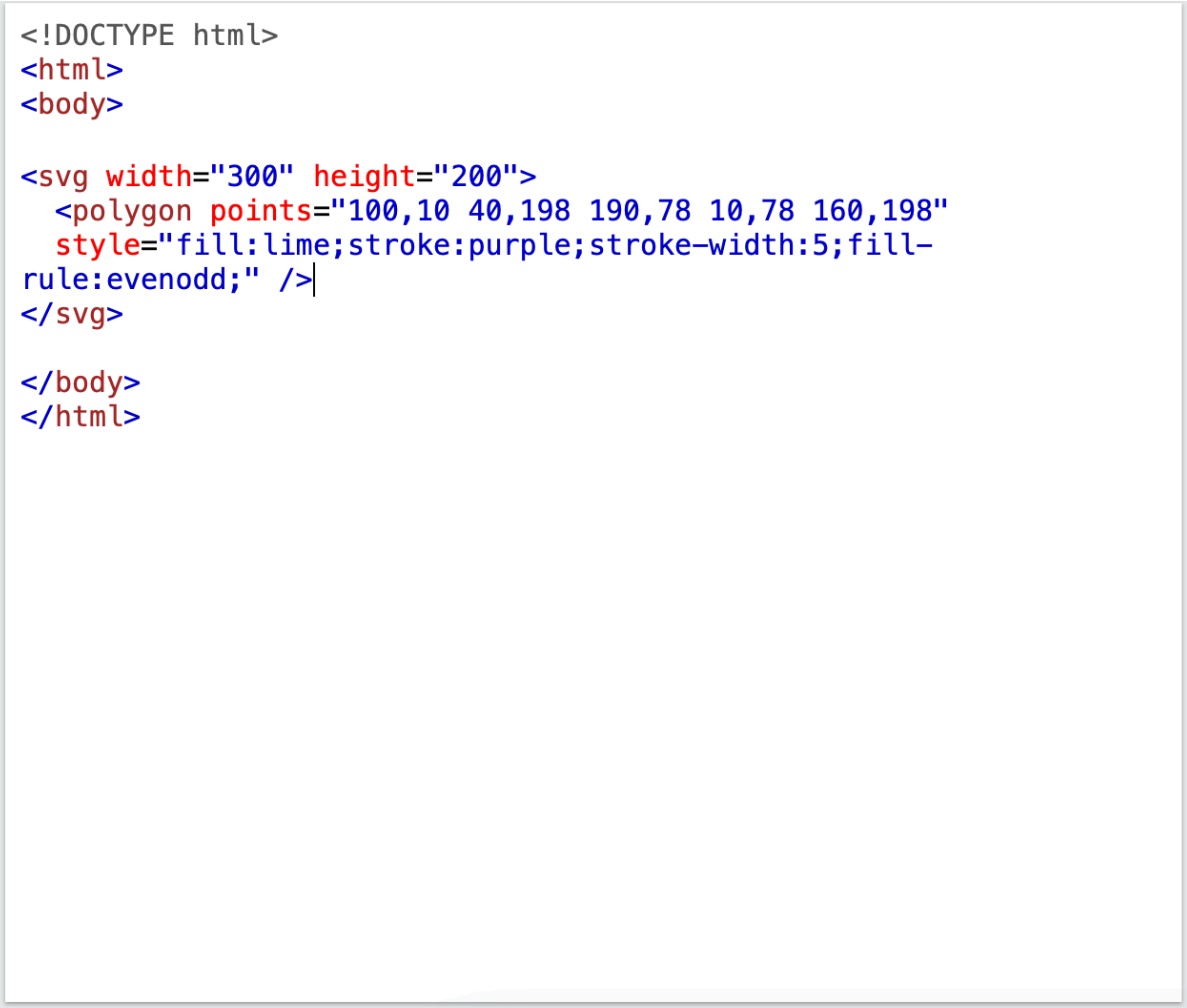
Drawing in SVG

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

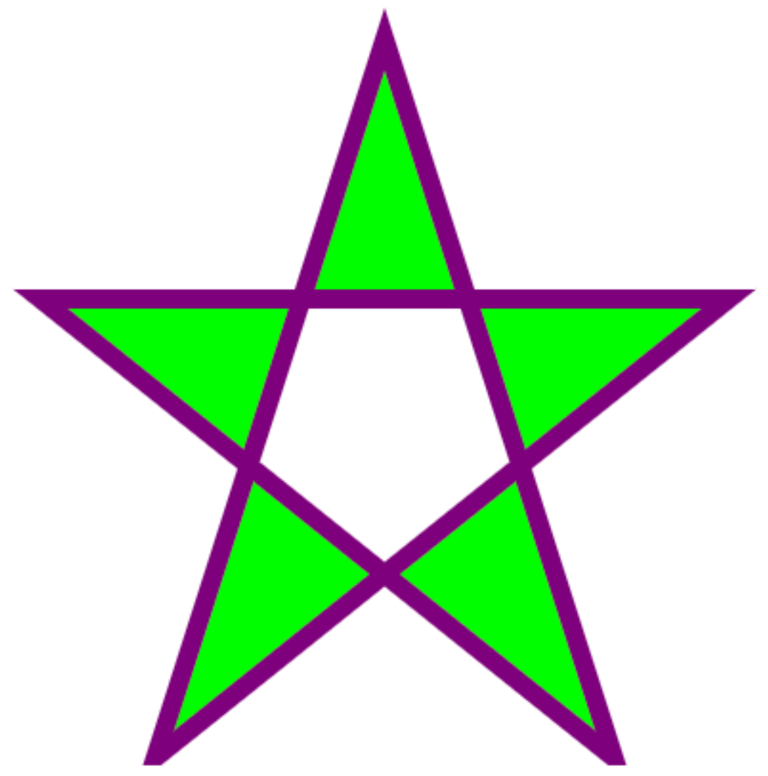
```
<!DOCTYPE html>
<html>
<body>

<svg width="100" height="100">
  <circle cx="50" cy="50" r="40"
  stroke="green" stroke-width="4" fill="yellow" />
</svg>

</body>
</html>
```



Drawing in SVG

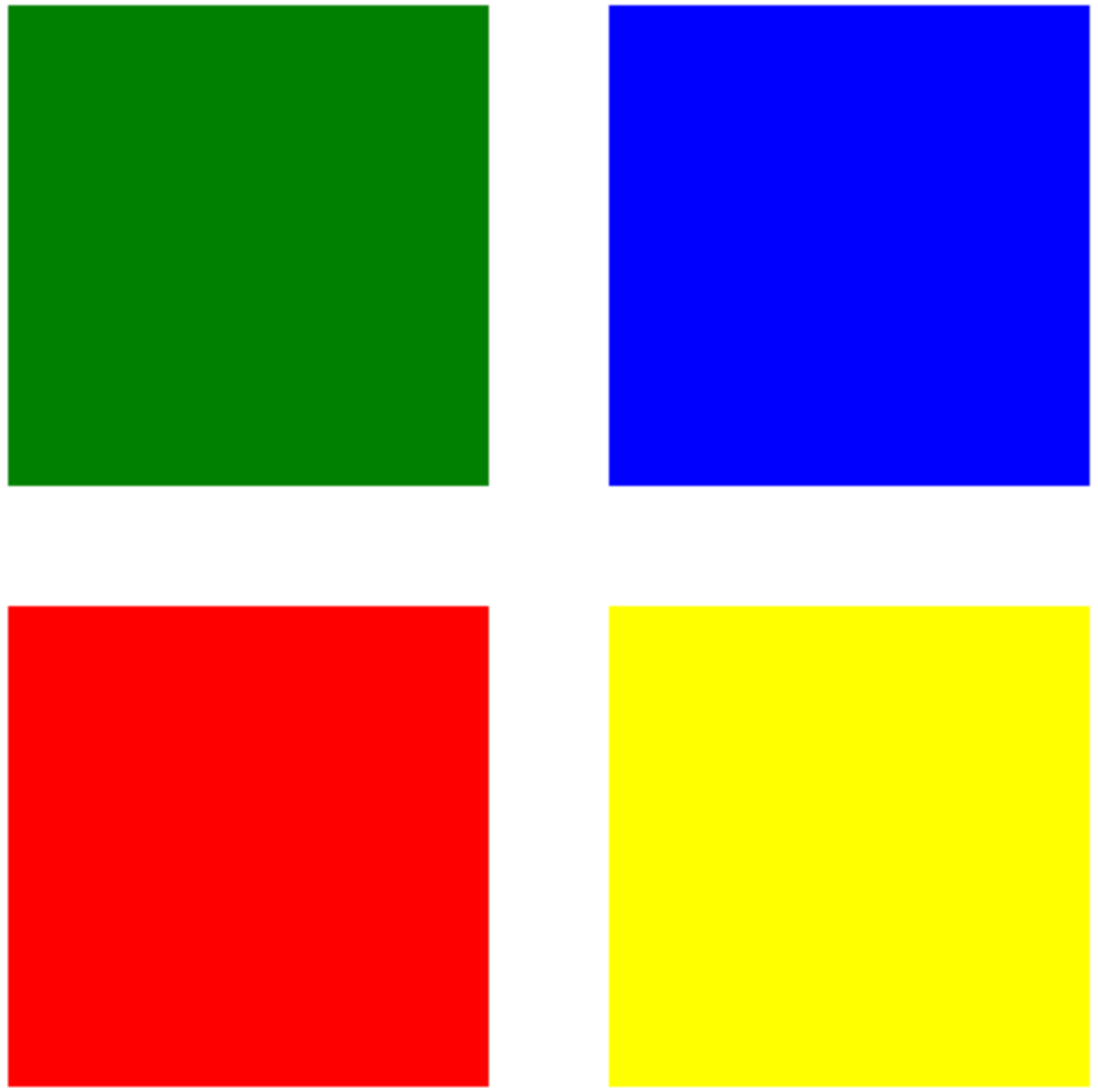


```
<!DOCTYPE html>
<html>
<body>

<svg width="300" height="200">
  <polygon points="100,10 40,198 190,78 10,78 160,198"
  style="fill:lime;stroke:purple;stroke-width:5;fill-
rule:evenodd;" />
</svg>

</body>
</html>
```



SVG transformation

- Translate

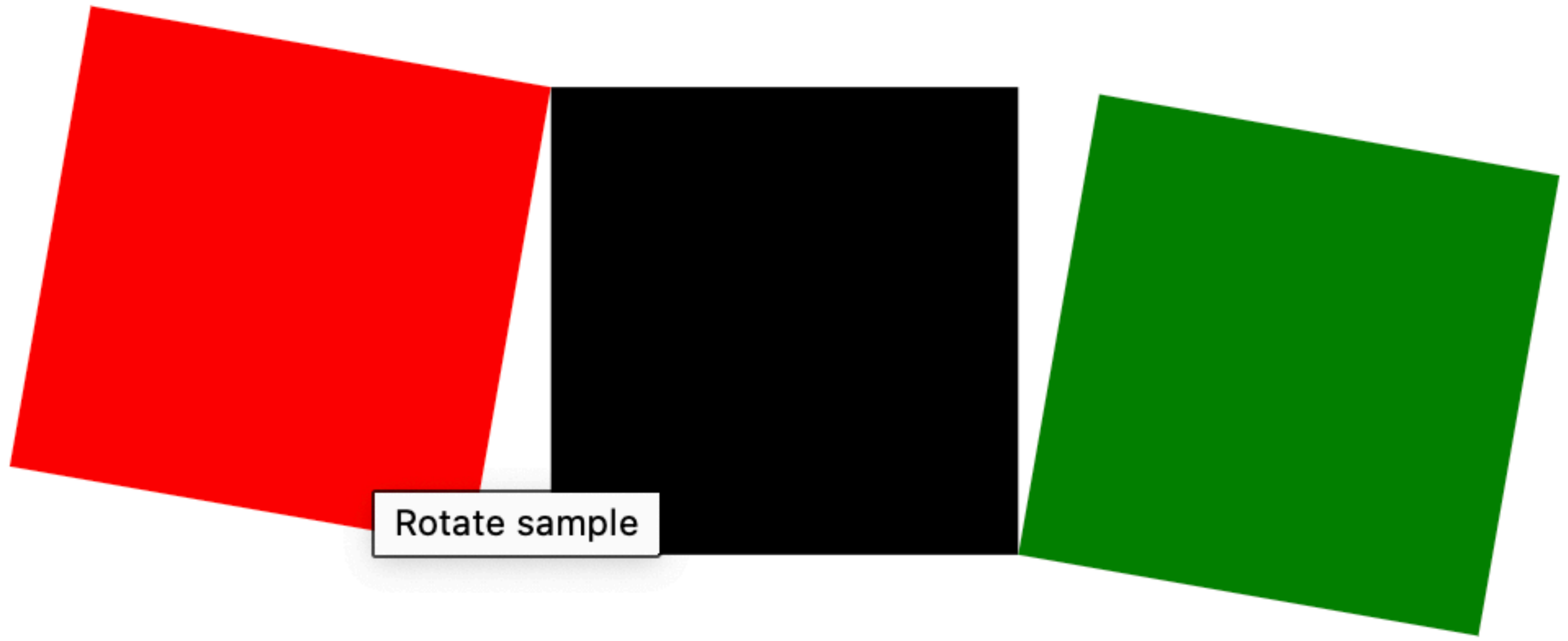


```
<rect
    x="5"
    y="5"
    width="40"
    height="40"
    fill="yellow"
    transform="translate(50 50)" />
```

SVG transformation

- Rotate

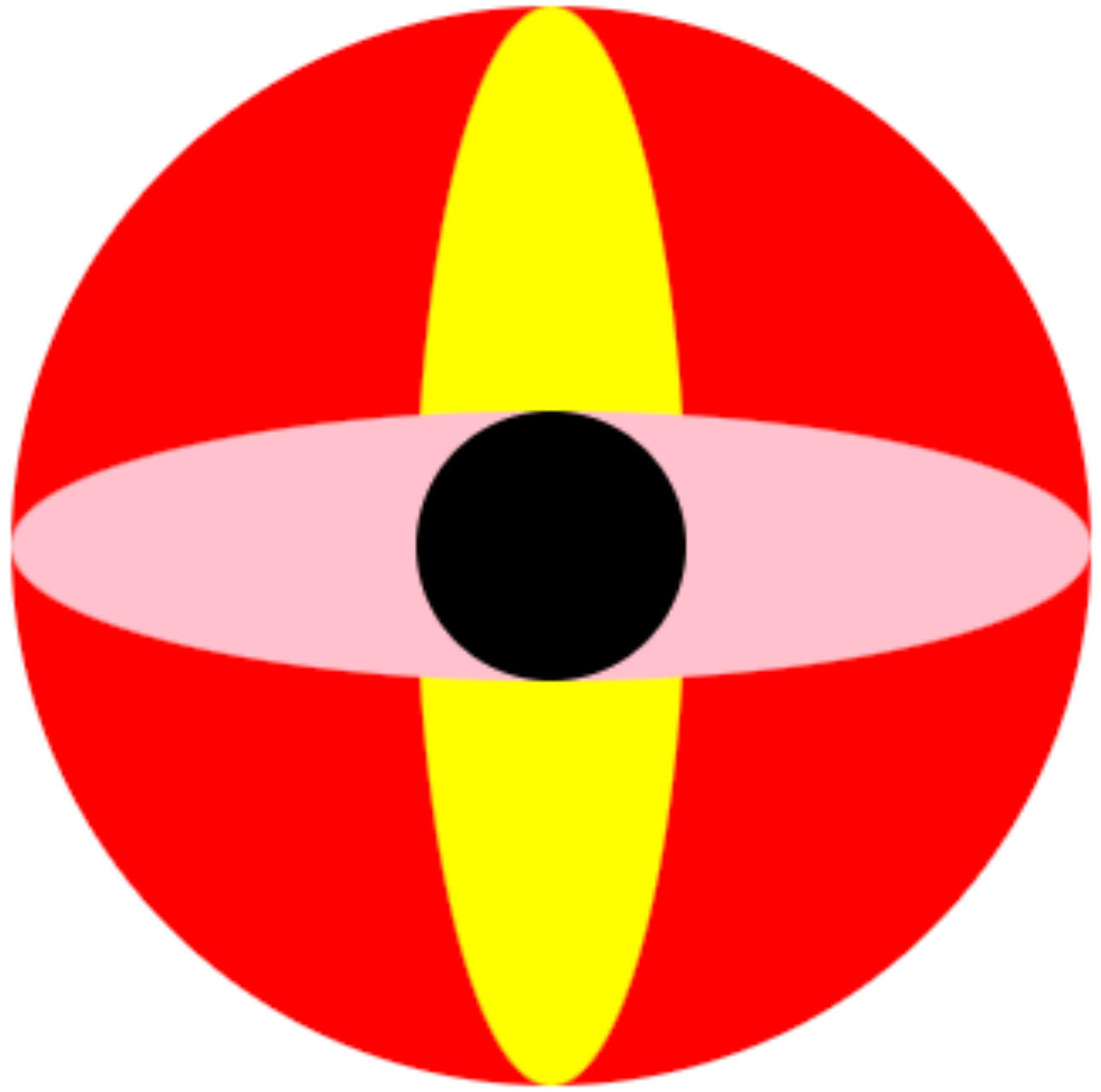
```
<rect x="0" y="0" width="10" height="10" />
  <!-- rotation is done around the point 0,0 -->
  <rect x="0" y="0" width="10" height="10" fill="red" transform="rotate(100)" />
  <!-- rotation is done around the point 10,10 -->
  <rect
    x="0"
    y="0"
    width="10"
    height="10"
    fill="green"
    transform="rotate(100, 10, 10)" />
```



https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function

SVG transformation

- Scale



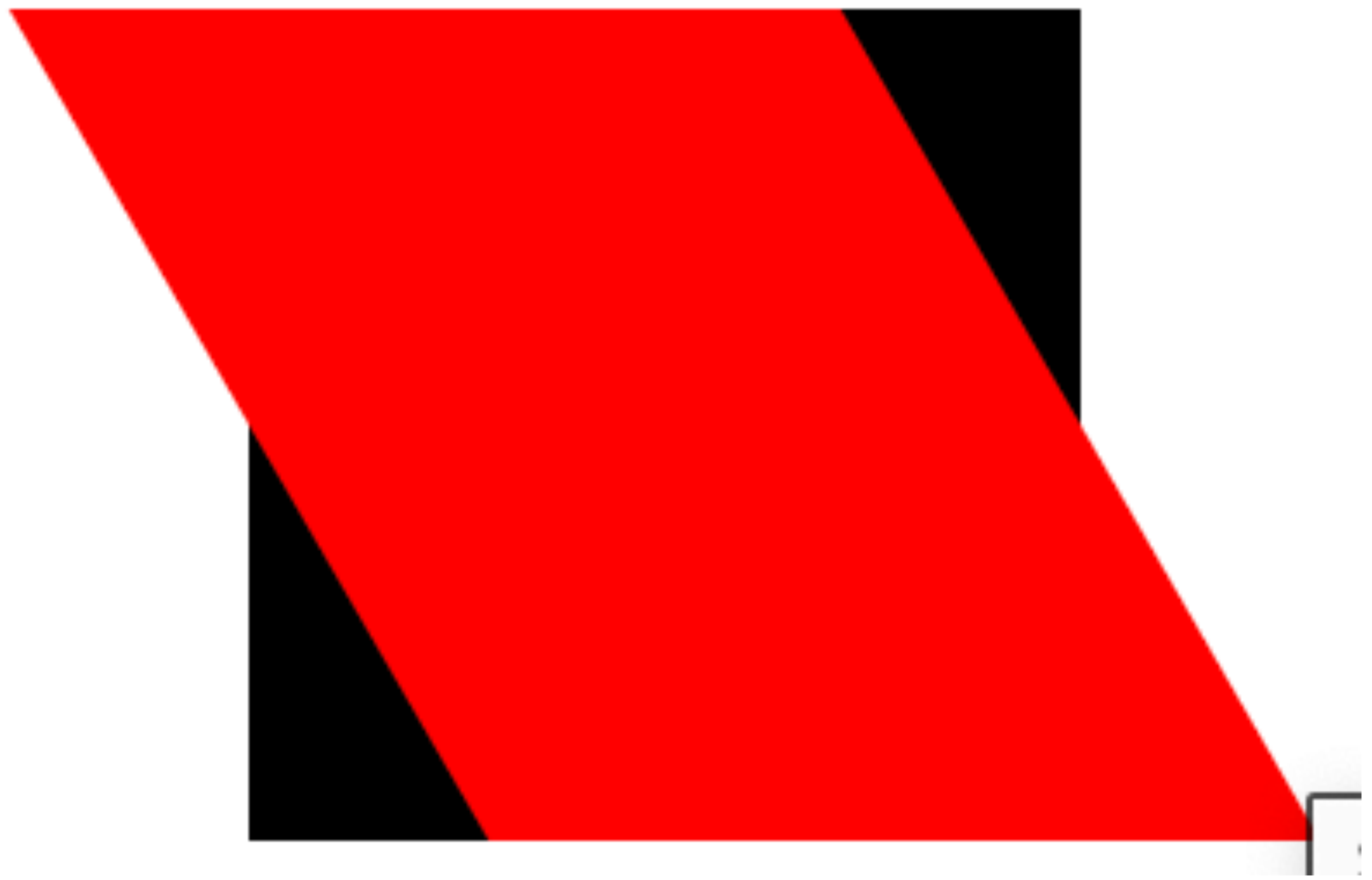
```
<circle cx="0" cy="0" r="10" fill="red" transform="scale(4)" />

  <!-- vertical scale -->
  <circle cx="0" cy="0" r="10" fill="yellow" transform="scale(1, 4)" />

  <!-- horizontal scale -->
  <circle cx="0" cy="0" r="10" fill="pink" transform="scale(4, 1)" />

  <!-- No scale -->
  <circle cx="0" cy="0" r="10" fill="black" />
```

https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function

SVG transformation

- Skew

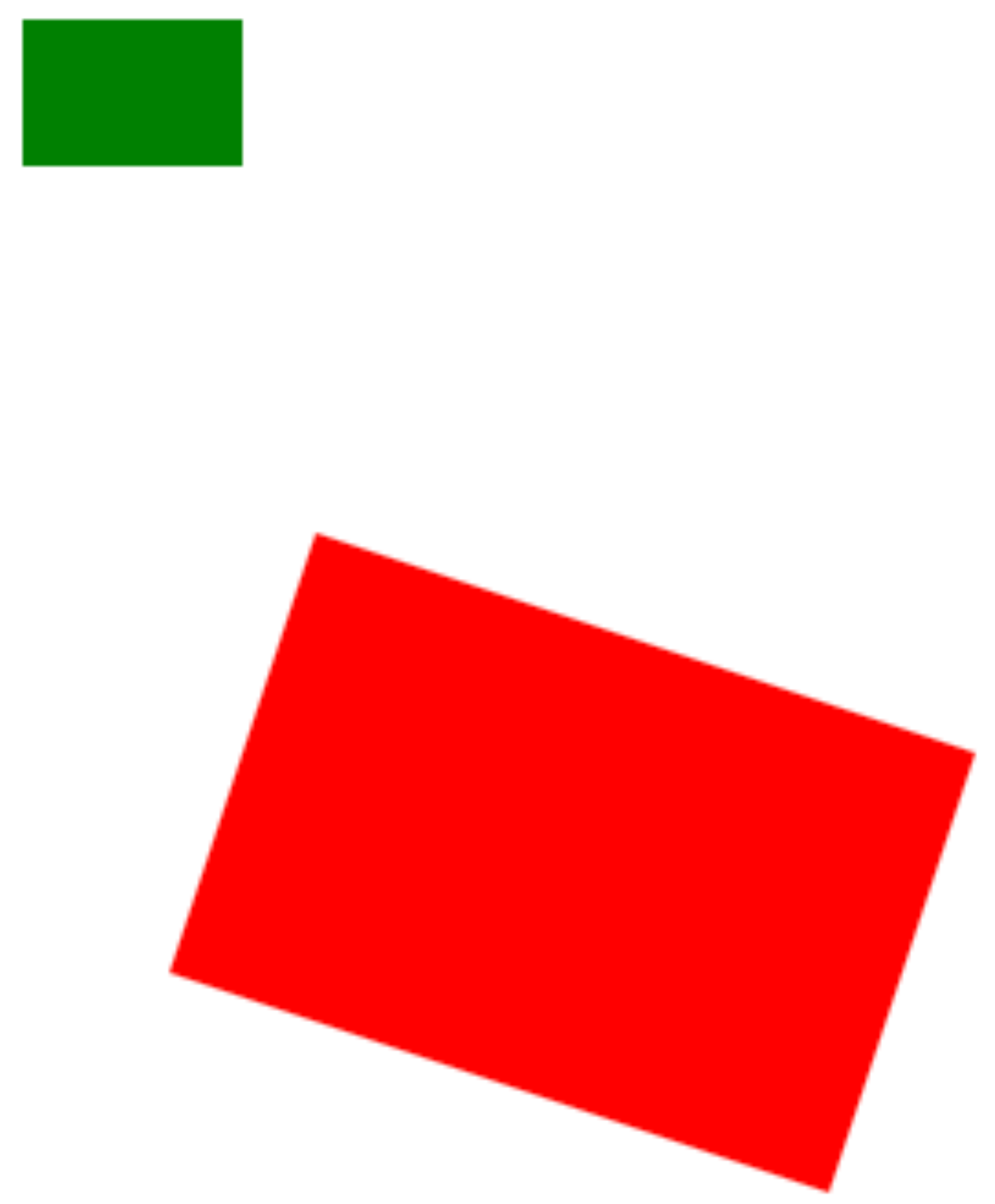
```
<rect x="-3" y="-3" width="6" height="6" />

<rect x="-3" y="-3" width="6" height="6" fill="red" transform="skewX(30)" />
```



SVG transformation

- Matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



```
<rect
    x="10"
    y="10"
    width="30"
    height="20"
    fill="red"
    transform="matrix(3 1 -1 3 30 40)" />
```

SVG transformation

- Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function

SVG transformation

- Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function

SVG transformation

- Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function

SVG transformation

- Matrix

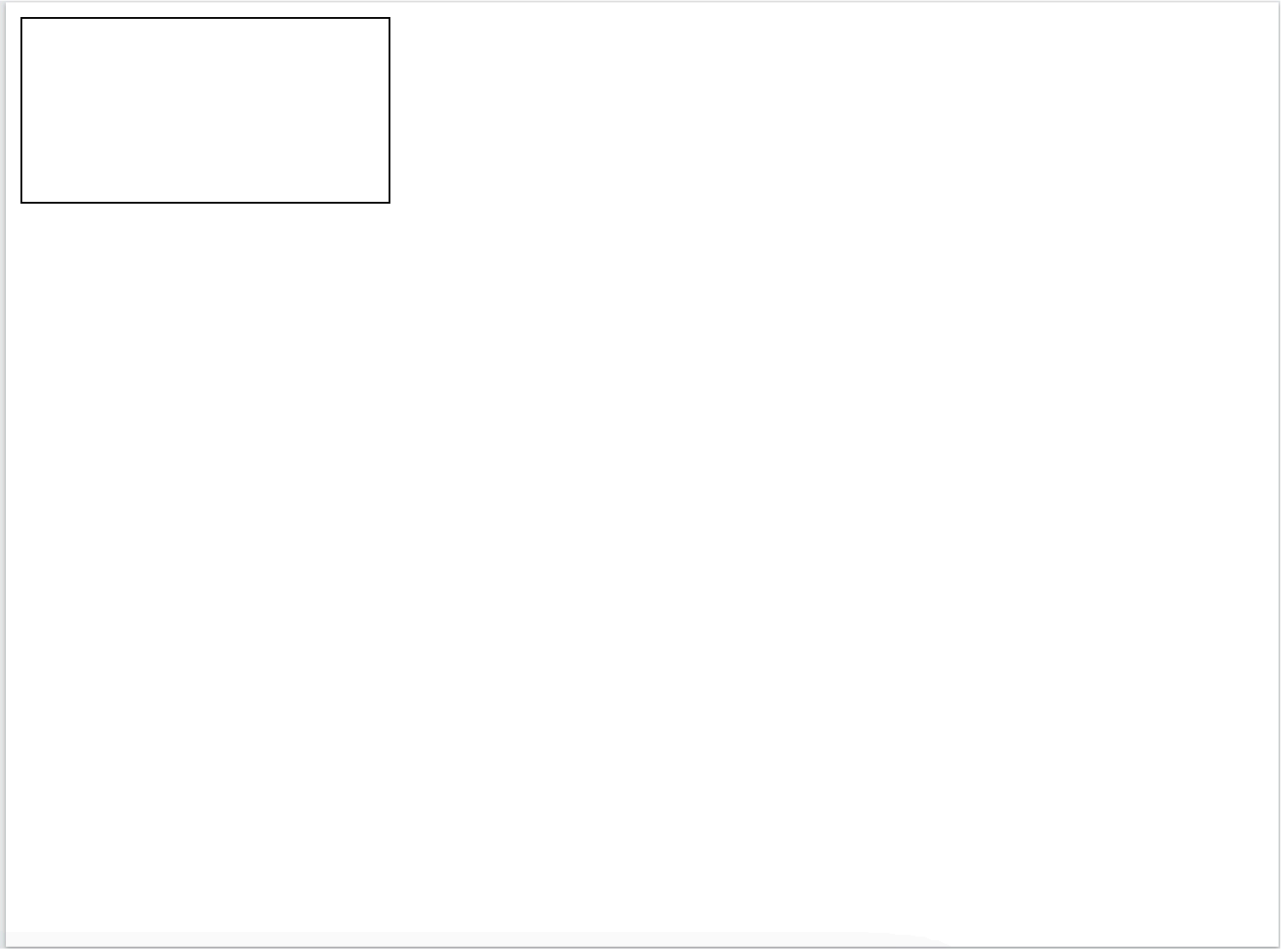
$$\begin{array}{ccccccc}
\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} & = & \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} & & \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} & & \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \\
\text{matrix} & & \text{rotation} & & \text{scale} & & \text{translation}
\end{array}$$

Canvas

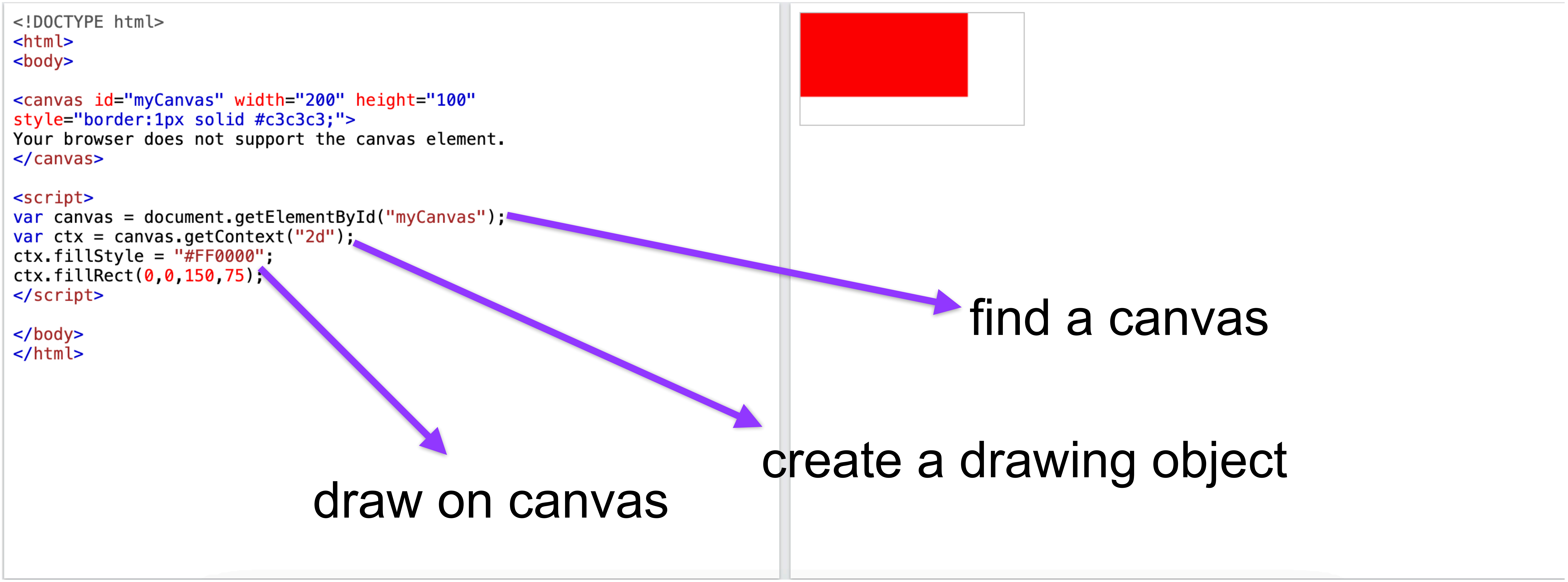
- Used for draw graphics
- Only a container and must use a script to draw graphics
- Only support two shapes: rectangles and paths

Canvas

```html
<!DOCTYPE html>
<html>
<body>

<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #000000;">
</canvas>

</body>
</html>
```



Drawing in canvas



```
<!DOCTYPE html>
<html>
<body>

<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #c3c3c3;">
Your browser does not support the canvas element.
</canvas>

<script>
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(0,0,150,75);
</script>

</body>
</html>
```

Drawing with canvas

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

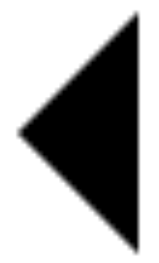
- Draw a triangle

```
function draw() {
  const canvas = document.getElementById('canvas');
  if (canvas.getContext) {
    const ctx = canvas.getContext('2d');

    ctx.beginPath();
    ctx.moveTo(75, 50);
    ctx.lineTo(100, 75);
    ctx.lineTo(100, 25);
    ctx.fill();
  }
}
```

Creates a new path

Moves the pen to the coordinates specified by x and y

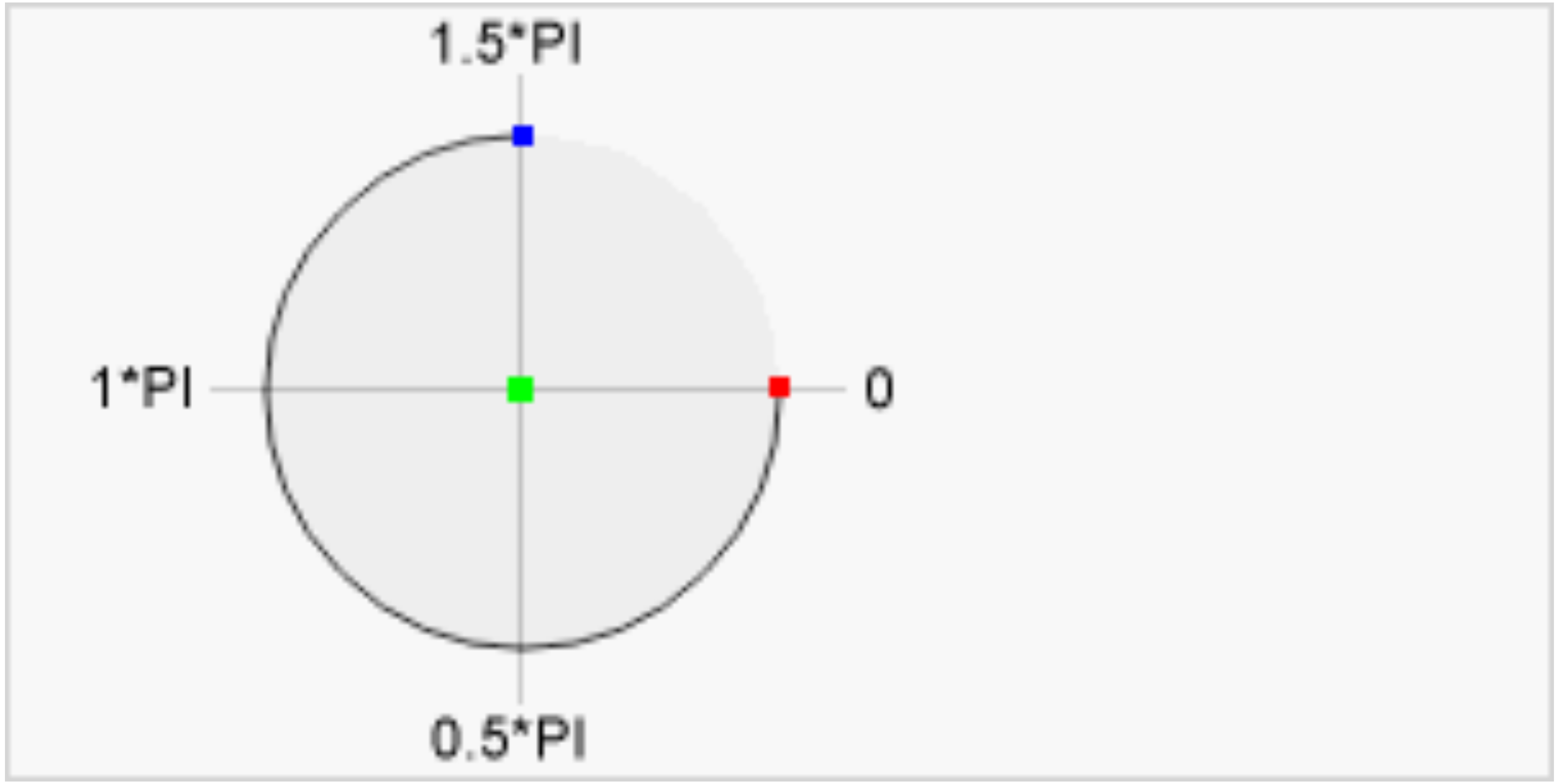
Draws a line from the current drawing position to the position specified by x and y

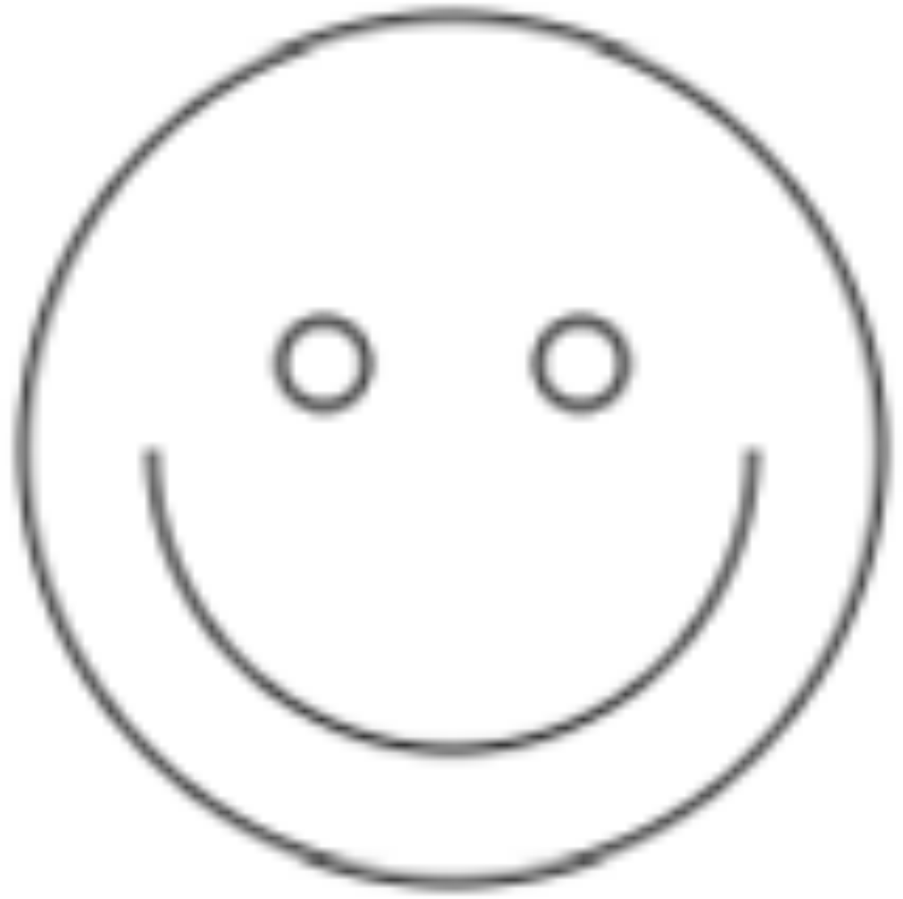


Drawing with canvas

- Draw a smile face

```
function draw() {
  const canvas = document.getElementById("canvas");
  if (canvas.getContext) {
    const ctx = canvas.getContext("2d");

    ctx.beginPath();
    ctx.arc(75, 75, 50, 0, Math.PI * 2, true); // Outer circle
    ctx.moveTo(110, 75);
    ctx.arc(75, 75, 35, 0, Math.PI, false); // Mouth (clockwise)
    ctx.moveTo(65, 65);
    ctx.arc(60, 65, 5, 0, Math.PI * 2, true); // Left eye
    ctx.moveTo(95, 65);
    ctx.arc(90, 65, 5, 0, Math.PI * 2, true); // Right eye
    ctx.stroke();
  }
}
```





Canvas exercise

- Write a JavaScript program to draw two intersecting rectangles, one of which has alpha transparency

Canvas exercise

- Write a JavaScript program to draw two intersecting rectangles, one of which has alpha transparency

```
function draw()
 {
     var canvas = document.getElementById(“canvas");
     if (canvas.getContext) {
      var context = canvas.getContext("2d");

       context.fillStyle = "rgb(256,0,0)";
       context.fillRect (15, 10, 55, 50);

       context.fillStyle = "rgba(0, 0, 200, 0.6)";
       context.fillRect (35, 30, 55, 50);
   }
 }
```

Outline

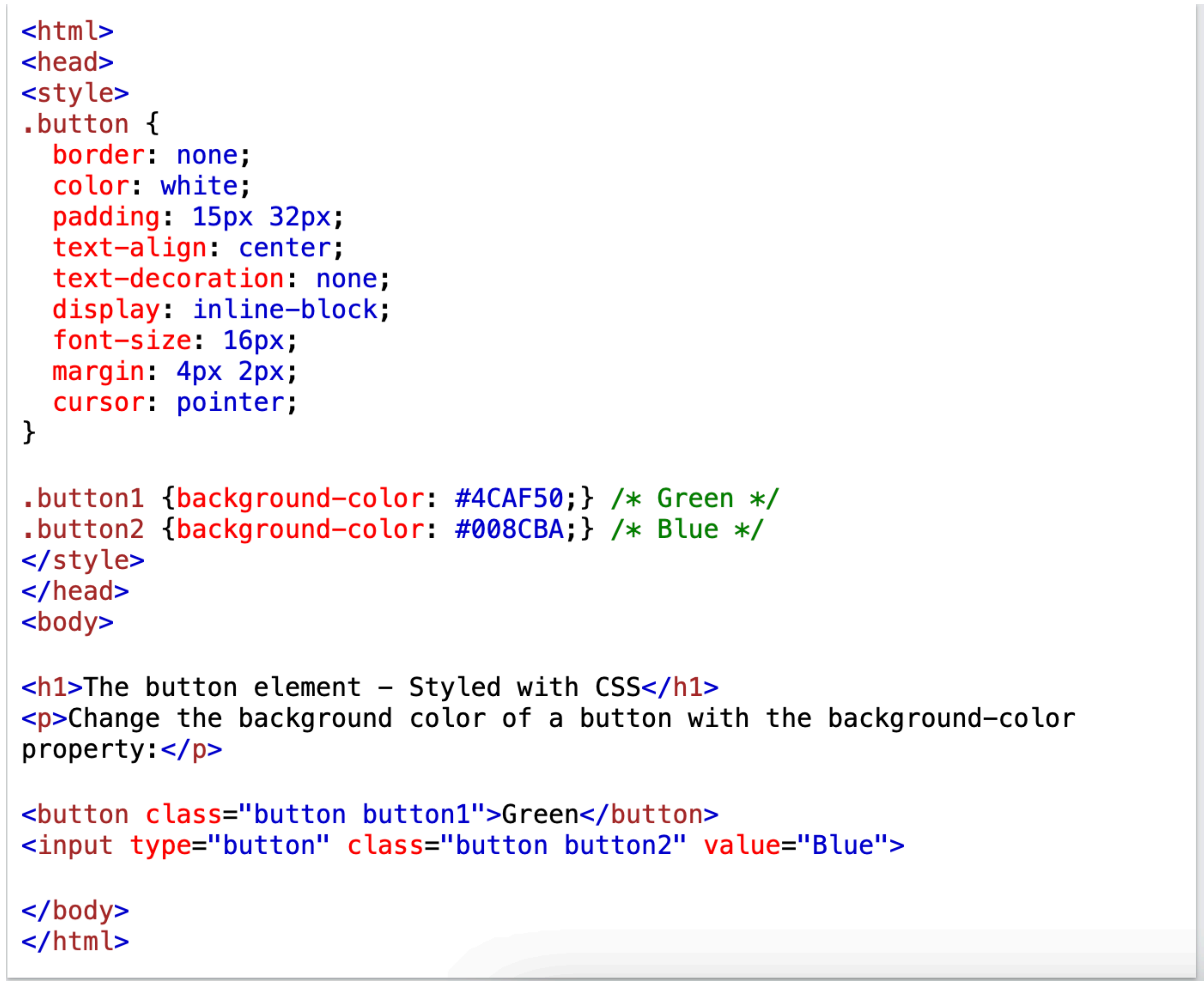
- JavaScript
- SVG and Canvas
- Interaction

Interaction with JavaScript and HTML

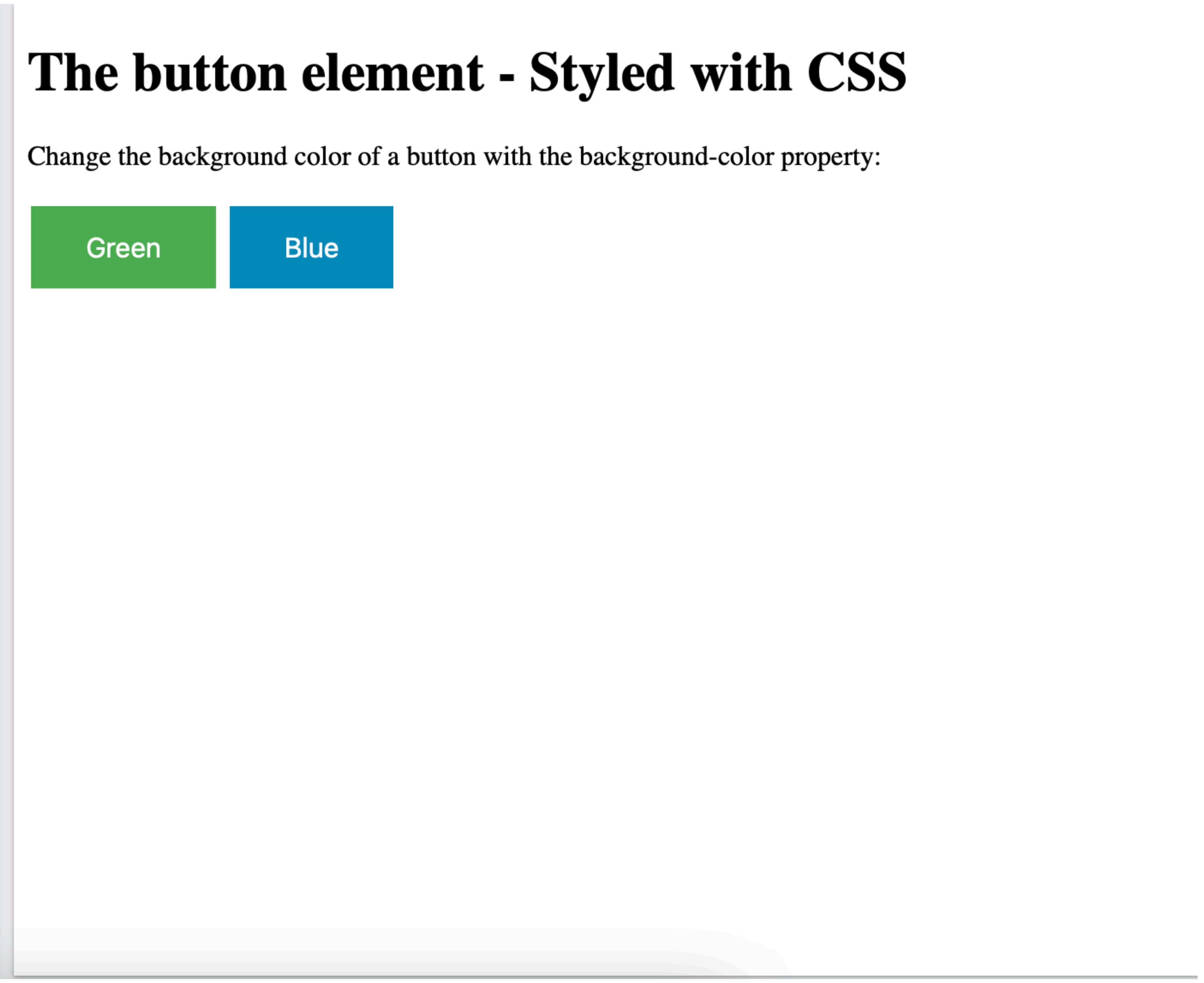
- Interaction tags
  - Button
  - Text
  - Radio
  - Option
  - ect.

Button

- Syntax
  - <button>content</button>
  - <input type=‘button’ value=“content”> </input>



```
<html>
<head>
<style>
.button {
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 4px 2px;
  cursor: pointer;
}

.button1 {background-color: #4CAF50;} /* Green */
.button2 {background-color: #008CBA;} /* Blue */
</style>
</head>
<body>

<h1>The button element - Styled with CSS</h1>
<p>Change the background color of a button with the background-color
property:</p>

<button class="button button1">Green</button>
<input type="button" class="button button2" value="Blue">

</body>
</html>
```

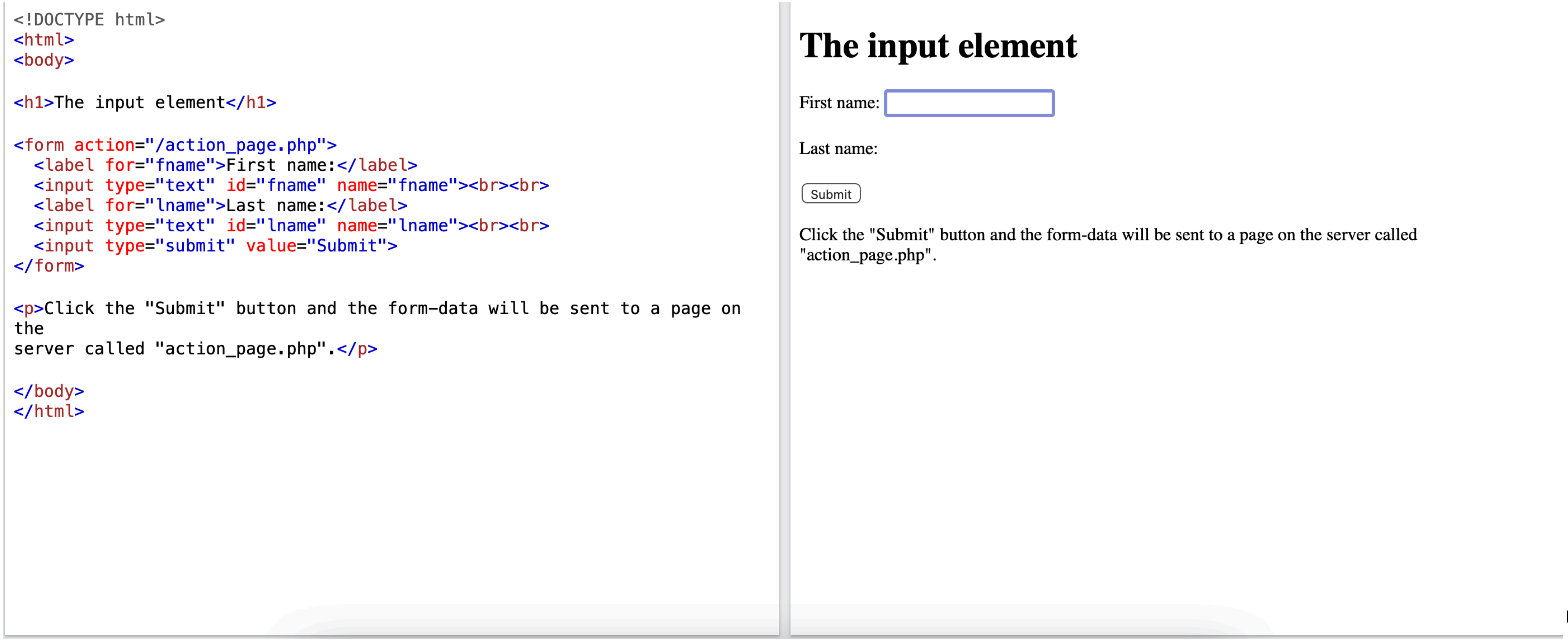


The button element - Styled with CSS

Change the background color of a button with the background-color property:

Text

- Syntax
  - <input type=“text”>



```
<!DOCTYPE html>
<html>
<body>

<h1>The input element</h1>

<form action="/action_page.php">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit">
</form>

<p>Click the "Submit" button and the form-data will be sent to a page on
the
server called "action_page.php".</p>

</body>
</html>
```

The input element

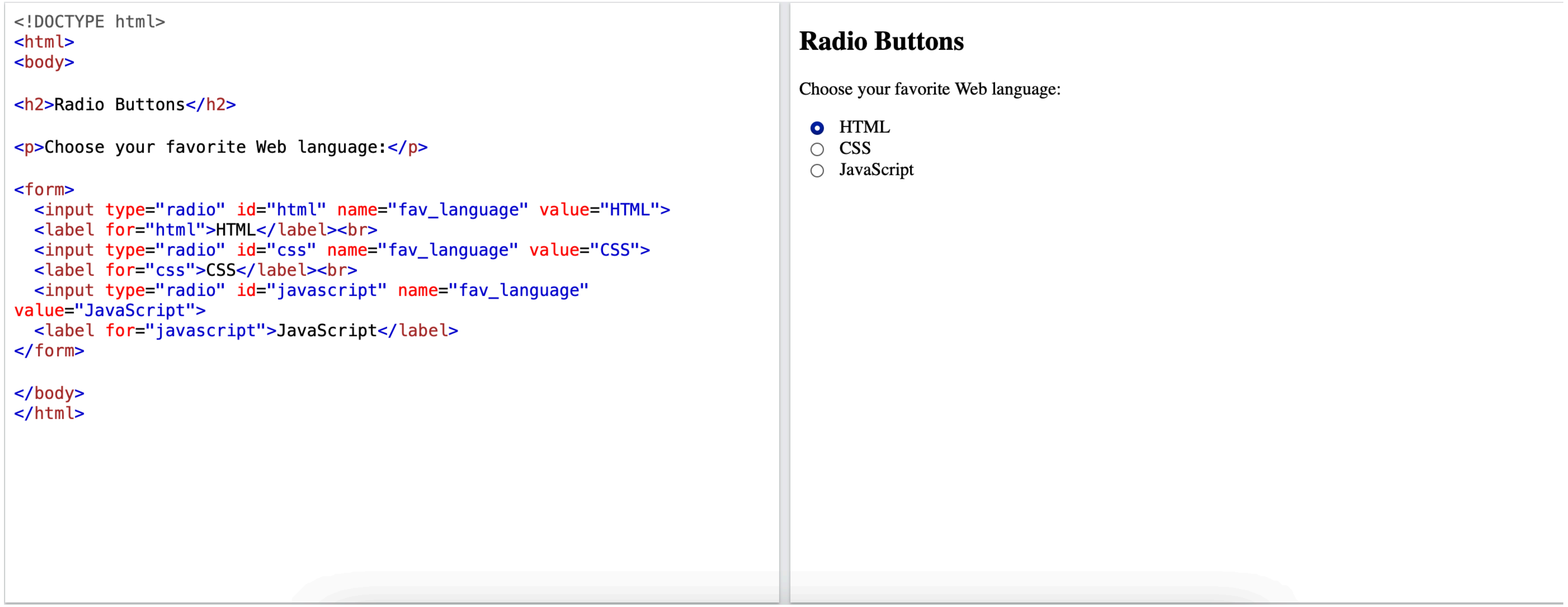
First name:

Last name:

Click the "Submit" button and the form-data will be sent to a page on the server called "action_page.php".

Radio

- Syntax
  - <input type=“radio” value= “content”>



```
<!DOCTYPE html>
<html>
<body>

<h2>Radio Buttons</h2>

<p>Choose your favorite Web language:</p>

<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language"
value="JavaScript">
  <label for="javascript">JavaScript</label>
</form>

</body>
</html>
```

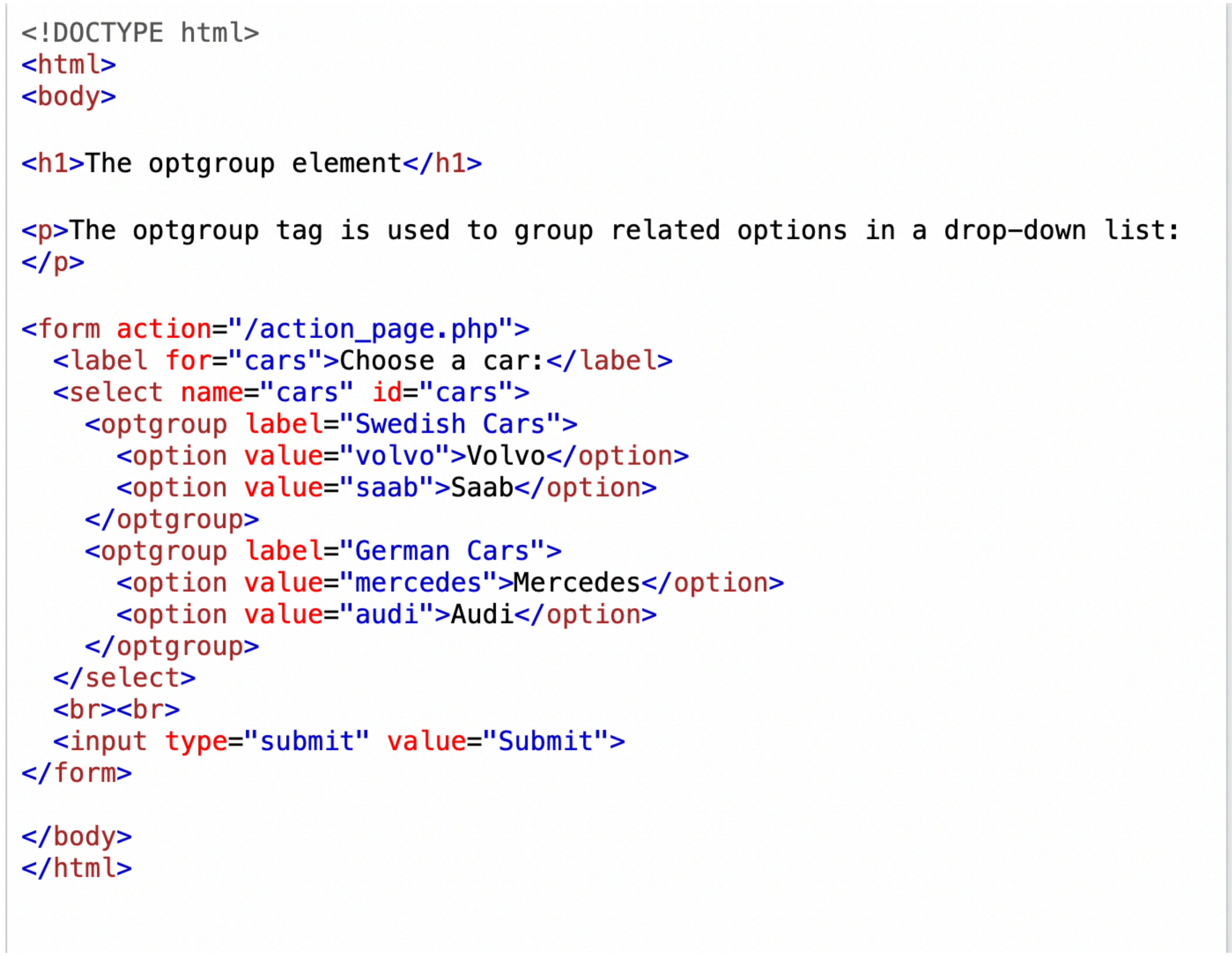
Radio Buttons

Choose your favorite Web language:

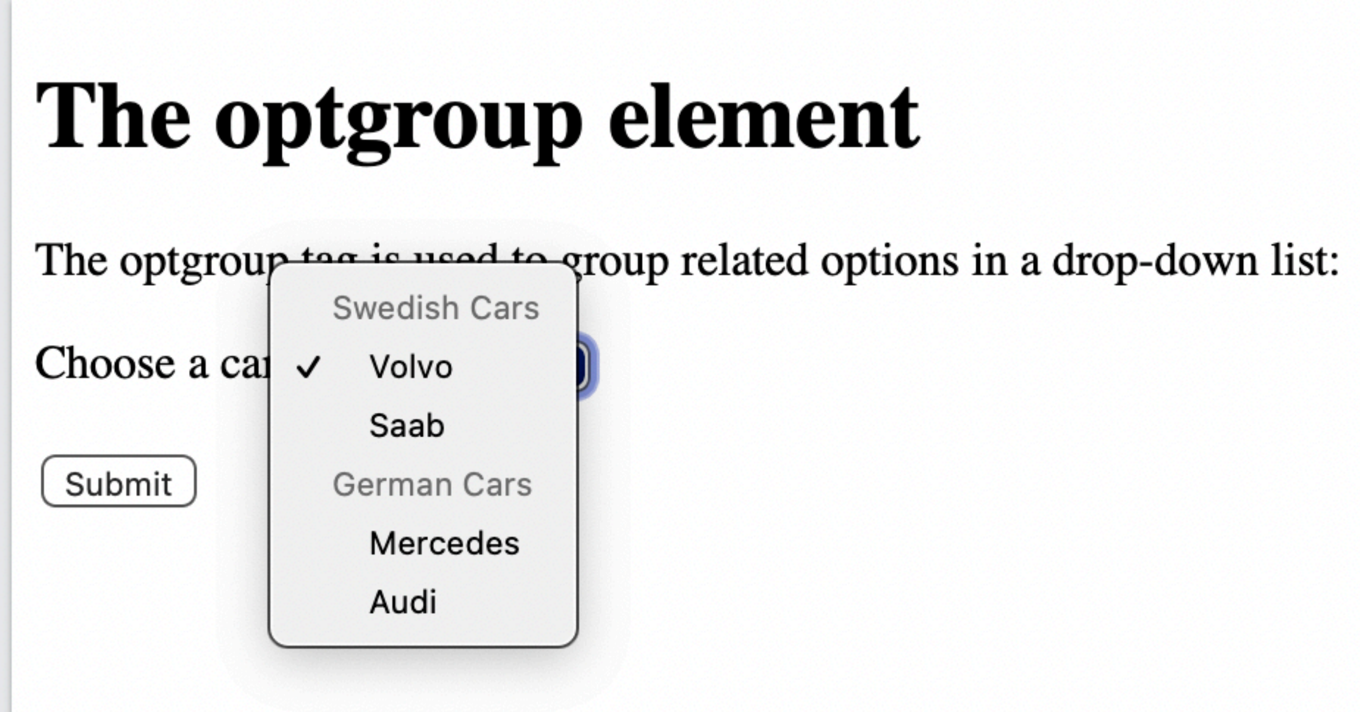
[✓ HTML]
○ CSS
○ JavaScript

Option

- Syntax
  – <select name = “name”> <option value=“value”>value</option> </select>



```
<!DOCTYPE html>
<html>
<body>

<h1>The optgroup element</h1>

<p>The optgroup tag is used to group related options in a drop-down list:
</p>

<form action="/action_page.php">
  <label for="cars">Choose a car:</label>
  <select name="cars" id="cars">
    <optgroup label="Swedish Cars">
      <option value="volvo">Volvo</option>
      <option value="saab">Saab</option>
    </optgroup>
    <optgroup label="German Cars">
      <option value="mercedes">Mercedes</option>
      <option value="audi">Audi</option>
    </optgroup>
  </select>
  <br><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```



The optgroup element

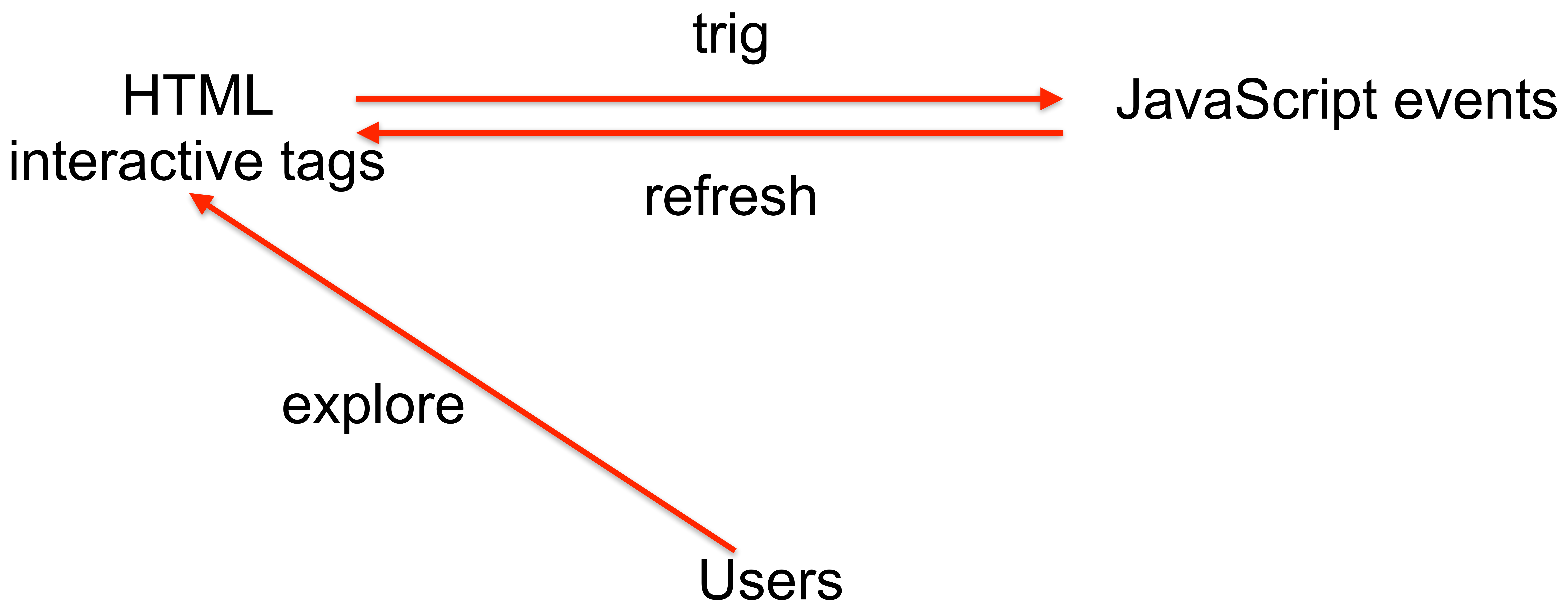
The optgroup tag is used to group related options in a drop-down list:

Choose a car

More interactive tags

- Checkbox
- Color
- Passward
- Search
- Date/time/month/number
- Examples can be found at https://www.w3schools.com/tags/att_input_type.asp

Interactive tags and JavaScript events



JavaScript events

- Something a user does
  - An HTML input field is changed
  - An HTML button is clicked
  - An mouse is moved
  - etc.

JavaScript events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Button and JavaScript event

- Create a button tag in HTML

```
<button class="button button1">Green</button>
```

$\downarrow$

```
<button class="button button1" onclick="click_button('green');">Green</button>
```

- Complete click_button function on JavaScript

```
click_button = function(c){
  color = c;
  console.log("You click "+color);
}
```

Slider and JavaScript event

- Create a slider tag in HTML

```
<button onclick="Click()">Display the chosen value</button>
<label for="vol">Volume (between 0 and 50):</label>
<input type="range" id="vol" name="vol" min="0" max="50" value="25" step="1">
<p id="demo"> </p>
```

- Get slider value in Javascript and show in HTML

```
function Click(){
        var slider = document.getElementById("vol");
        var value = slider.value;
        document.getElementById("demo").innerHTML = value;
}
```

EventListener

- Add or remove an event to a element
  - Mousemove
  - Mousedown
  - Mouseup
  - etc.

Mousmove event

```
window.addEventListener('load', ()=>{
    document.addEventListener('mousemove', display);
});

function display(event){
    let x = Math.random();
    document.getElementById("demo").innerHTML = x;
}
```

Interaction with SVG

- Interact with one object
- Interact with multiple objects and multiple listeners

Interaction practice

- Create a button in HTML and once users click the button, a rectangle with random color will be shown

Additional resources

- JavaScript: https://www.w3schools.com/js/default.asp
- JavaScript: https://javascript.info/
- SVG: https://www.w3schools.com/graphics/svg_intro.asp
- Canvas: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API