Topic 2 Drawing in 2D

Created	@October 11, 2022 2:49 PM
Class	CM1005-Introduction-to-Programming-I
Type	Lecture
Materials	
Reviewed	□
URL	https://www.coursera.org/learn/uol-introduction-to-programming-1/home/week/3
Week	22 Oct Week 3

2.1 Drawing functions
2.2 The console and debugging

2.1 Drawing functions

To fill our screen with a color, we can do:

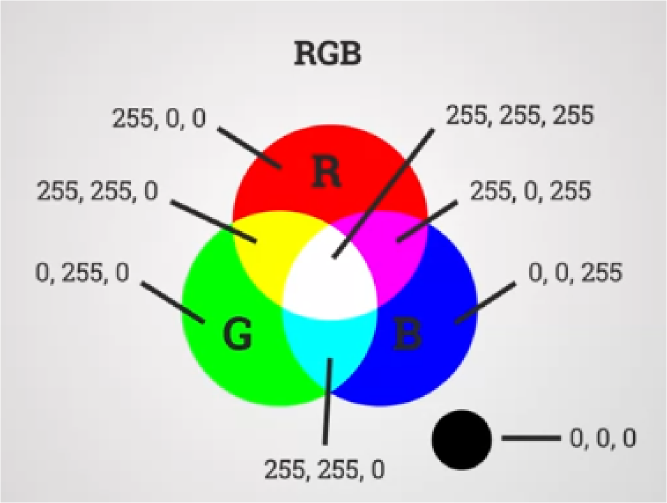
```
function draw() {
    background(rgbColorIntValue);
}
```

We can have 0..255 values for a channel of Red-Green-Blue (=**RGB**) colors each. That is **8 bits, or 1 byte** of information.

We can define the individual red, green, blue channels from darker to lighter values by providing a lower or higher value.

```
backGround(255,0,255);
```

The above gives the color of Magenta for example.



Total number of colors we have available is:

$$256^3 = 256 * 256 * 256 = 16777216$$

There are online color pickers and tables which we can use.

CSS3 module: Color

CSS (Cascading Style Sheets) is a language for describing the rendering of HTML and XML documents on screen, on paper, in speech, etc. To color elements in a document, it uses color related properties and respective values. This draft describes the properties and values that are proposed for CSS level 3.

https://www.w3.org/TR/css3-iccprof#numerical

Changing coloring of shapes in p5.js is possible via:

`fill()` change the fill color

`noFill()` do not have a fill color

`stroke()` change the outline color

`noStroke()` no outline

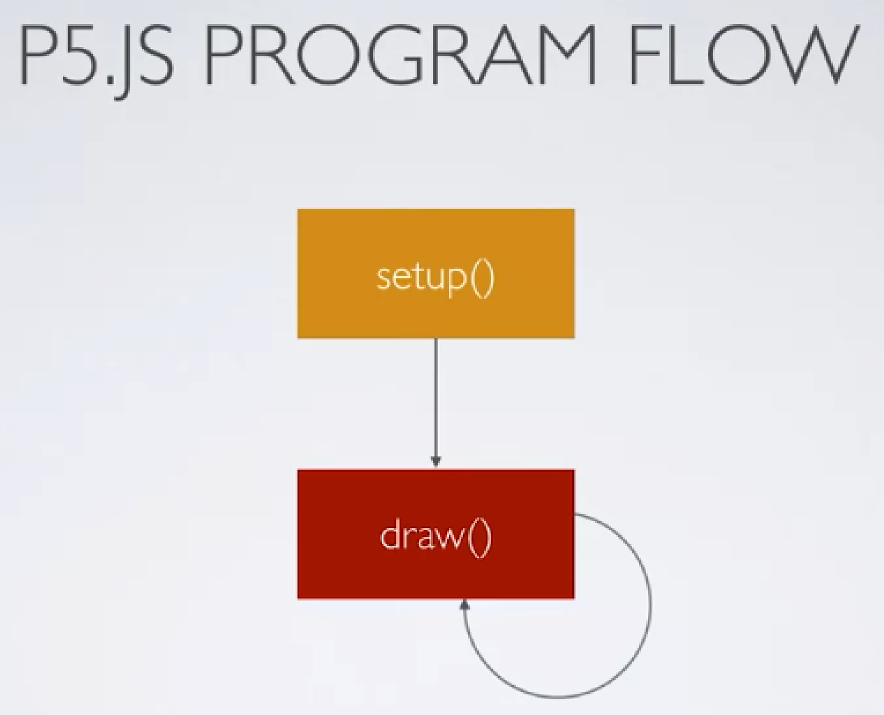
`stokeWeight()` is for the thickness of the outline

p5.js retains the last color settings until you change it.

They can have additional properties such as **Alpha**, which controls **opacity** (see-through).

The opaqueness is also a value between 0..255.

Program execution in p5.js is rendering frames in a repetitive manner:



P5.JS PROGRAM FLOW

setup()

draw()

Within the specific function the commands are executed in sequence.

We can draw all kind of different geometric shapes using the following:

`rect(x, y, width, height)` rectangle

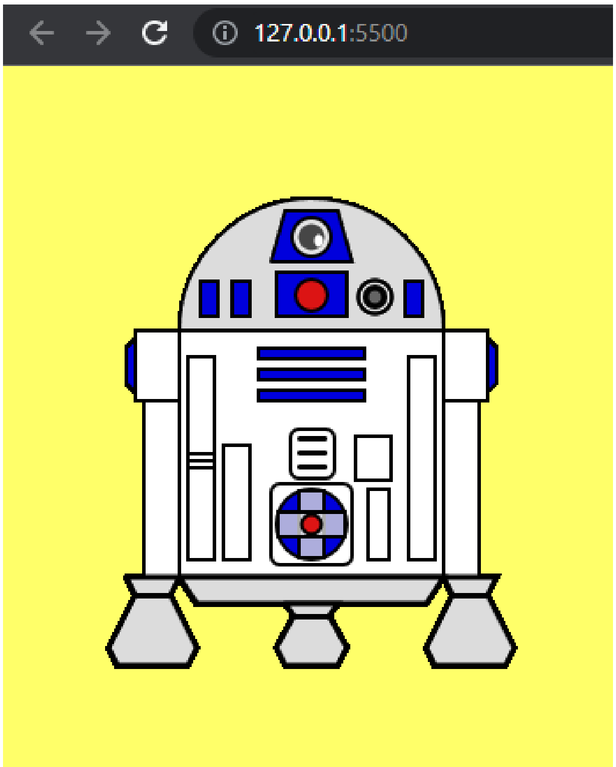
`ellipse(x, y, r1, r2)` ellipse or circle

`line(x, y, a, b)` line

`triangle(x, y, a, b, v, w)` triangle

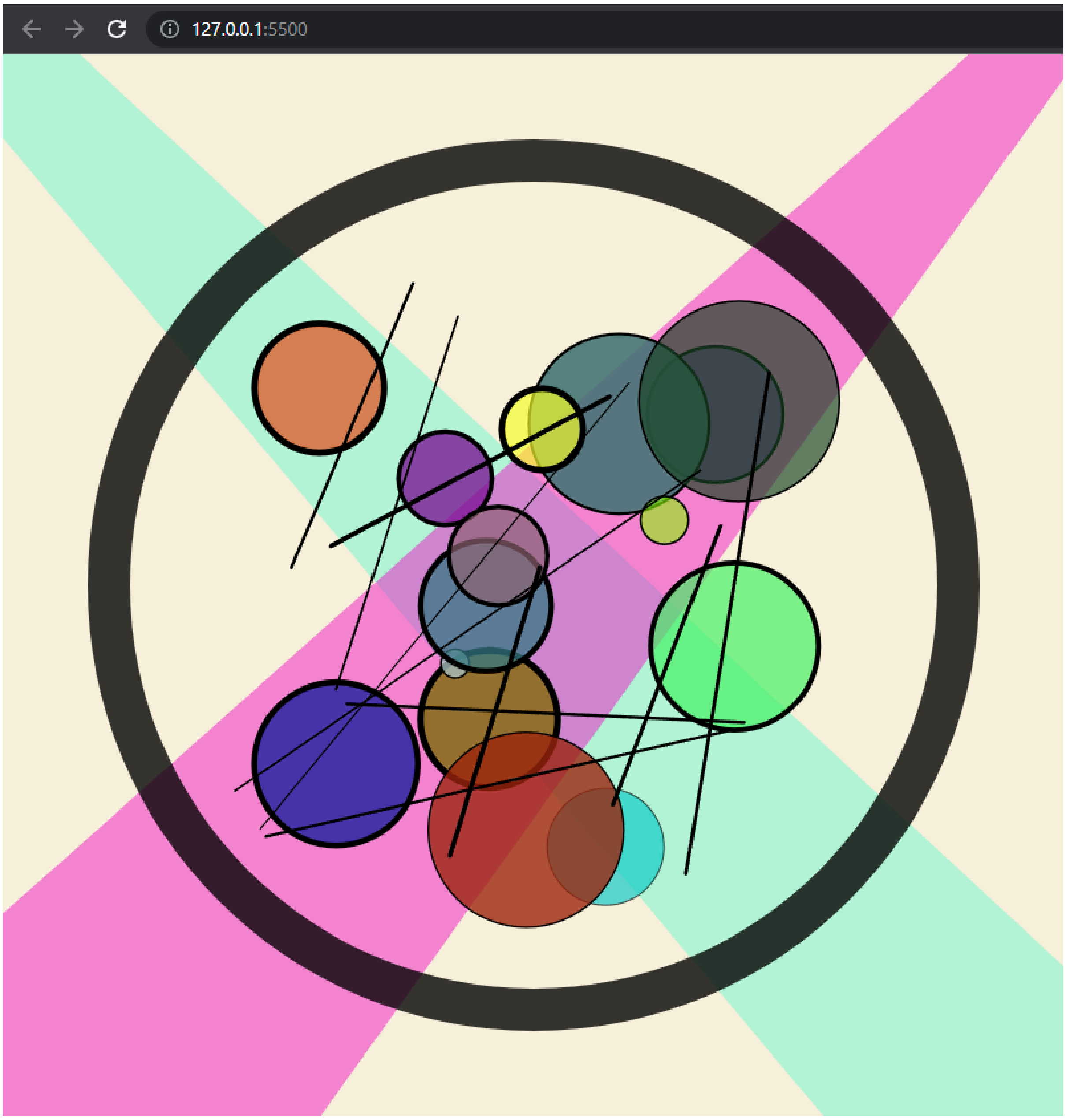
`point(x, y)` point

Hack it robot parade, my solution:



```
function setup()
{
  //create a canvas for the robot
  createCanvas(350, 400);
  background(255,255,0, 150);
}

function draw()
{
  // R2-D2

  stroke(1);
  strokeWeight(2);
  // body
  fill(220);
  arc(175, 150, 150, 150, PI, HALF_PI+HALF_PI);
  fill(255);
  rect(100, 150, 150, 140);

  // legs
  rect(75, 150, 25, 40);
  rect(250, 150, 25, 40);
  rect(80, 190, 20, 100);
  rect(250, 190, 20, 100);

  // leg caps
  fill(0,0,220);
  beginShape();
    vertex(75, 155)
    vertex(75, 185)
    vertex(70, 180)
    vertex(70, 160)
  endShape(CLOSE);
  beginShape();
    vertex(275, 155)
    vertex(275, 185)
    vertex(280, 180)
    vertex(280, 160)
  endShape(CLOSE);

  // blue paint body
  rect(145, 160, 60, 6);
  rect(145, 172, 60, 6);
  rect(145, 184, 60, 6);
  ellipse(175, 260, 40, 40);

  // blue paint head
  rect(112, 122, 10, 20);
  rect(130, 122, 10, 20);
  rect(228, 122, 10, 20);
  rect(155, 117, 40, 25);
  beginShape();
    vertex(152, 111)
    vertex(160, 82)
    vertex(190, 82)
    vertex(198, 111)
  endShape(CLOSE);

  // cross and eye
  fill(220, 200);
  rect(168, 241, 14, 38);
  rect(156, 253, 38, 14);
  ellipse(211, 131, 19, 19);
  fill(100)
  ellipse(211, 131, 12, 12);
  noFill();
  ellipse(211, 131, 9, 9);
  ellipse(211, 131, 19, 19);
  fill(230)
  ellipse(175, 97, 22, 22);
  fill(70);
  noStroke();
  ellipse(175, 97, 15, 15);
  fill(255);
  ellipse(179, 99, 5, 7);
  stroke(1);

  // red circles
  fill(220,20,20);
  ellipse(175, 260, 11, 11);
  ellipse(175, 130, 18, 18);

  // white body paint
  noFill();
  rect(105, 165, 15, 115);
  rect(230, 165, 15, 115);
  rect(125, 215, 15, 65);
  rect(200, 210, 20, 25);
  rect(207, 240, 12, 40);
  rect(163, 206, 25, 28, 5);
  rect(152, 237, 46, 46, 5);

  line(105, 220, 120, 220);
  line(105, 224, 120, 224);
  line(105, 228, 120, 228);
  strokeWeight(3);
  line(168, 211, 182, 211);
  line(168, 219, 182, 219);
  line(168, 227, 182, 227);

  // wheels
  strokeWeight(3);
  fill(220);
  beginShape();
    // right leg
    vertex(70, 290);
    vertex(280, 290);
    vertex(275, 300);
    vertex(290, 330);
    vertex(285, 340);
    vertex(245, 340);
    vertex(240, 330);
    vertex(255, 300);
    vertex(250, 290);
    // left leg
    vertex(100, 290);
    vertex(95, 300);
    vertex(110, 330);
    vertex(105, 340);
    vertex(65, 340);
    vertex(60, 330);
    vertex(75, 300);
  endShape(CLOSE);
  line(75, 300, 95, 300);
  line(255, 300, 275, 300)
  beginShape();
    // belly
    vertex(100, 290);
    vertex(250, 290);
    vertex(240, 305);
    vertex(110, 305);
  endShape(CLOSE);
  beginShape();
    // middle leg
    vertex(160, 305);
    vertex(165, 312);
    vertex(155, 330);
    vertex(160, 340);
    vertex(190, 340);
    vertex(195, 330);
    vertex(185, 312);
    vertex(190, 305);
  endShape(CLOSE);
  line(165, 312, 185, 312);
}
```

Kandinsky task solution

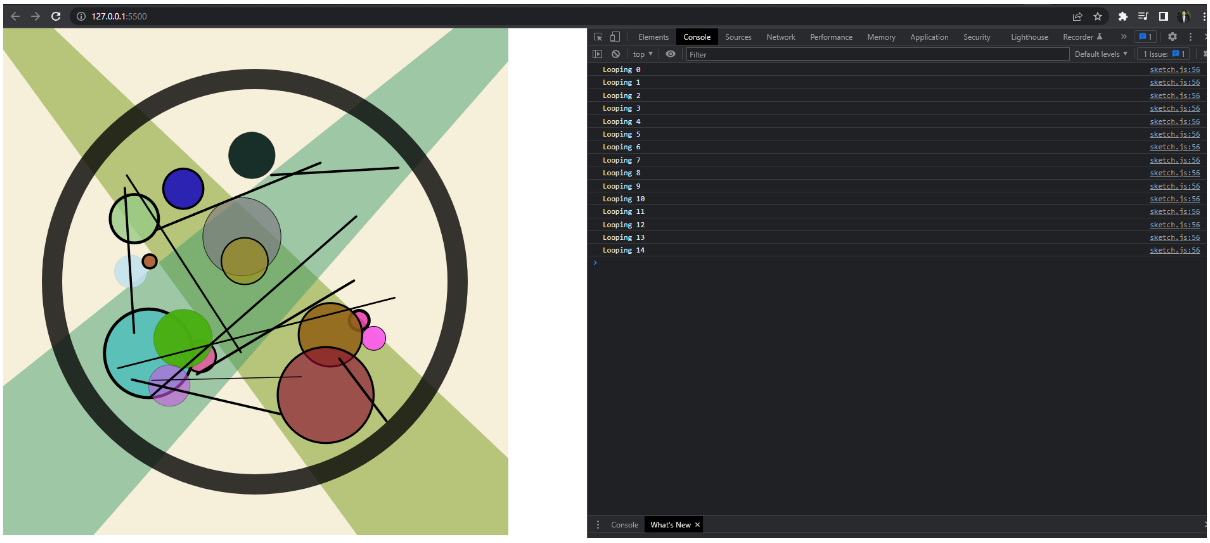


```
let width = 800;
let height = 800;

function doBackgroundLine(mirror)
{
  stroke(0);
  noStroke();
  fill(random(256),random(256),random(256), random(50)+105);
  beginShape();
    let wValue = random(width/10)+10;
    let hValue = random(height/10)+10;
    vertex(mirror ? width : 0, hValue);
    vertex(mirror ? width : 0, 0);
    vertex(mirror ? width - wValue : wValue, 0);
    wValue = random(width/10)*3+20;
    hValue = random(height/10)*3+20;
    vertex(mirror ? 0 : width, height-hValue);
    vertex(mirror ? 0 : width, height);
    vertex(mirror ? wValue: width-wValue, height);
  endShape(CLOSE);
}

function doRandomCircle()
{
  fill(random(256),random(256),random(256), random(100)+150);
  radius = random(width / 6)+20;
  wValue = random(width - width/2) + width / 4;
  hValue = random(height - height/2) + height / 4;
  stroke(0);
  strokeWeight(random(0, 5));
  ellipse(wValue, hValue, radius, radius);
}

function doRandomLine()
{
  stroke(0);
  strokeWeight(random(1, 4));
  wValue = width / 2 + (Math.random()<0.5?1:-1) * random(width/3.5);
  hValue = height / 2 + (Math.random()<0.5?1:-1) * random(height/3.5);
  wValue2 = width / 2 + (Math.random()<0.5?1:-1) * random(width/3.5);
  hValue2 = height / 2 + (Math.random()<0.5?1:-1) * random(height/3.5);
  line(wValue, hValue, wValue2, hValue2);
}

function setup()
{
  // init
  createCanvas(width, height);
  background(200,155,10,40);

  doBackgroundLine(false);
  doBackgroundLine(true);

  for (let i = 0; i < 15; i++)
  {
    doRandomCircle();
  }

  for (let i = 0; i < 10; i++)
  {
    doRandomLine();
  }

  // black circle
  noFill();
  stroke(0, 200);
  strokeWeight(width / 25);
  ellipse(width/2, height/2, width/5*4, height/5*4);

}

function draw()
{
  //do your drawing here
}
```

2.2 The console and debugging

The console is a powerful tool that allows us to see what is going on under the hood.

We can view the console in **Chrome browser** by pressing **<F12>** and navigating to the *Console tab*.



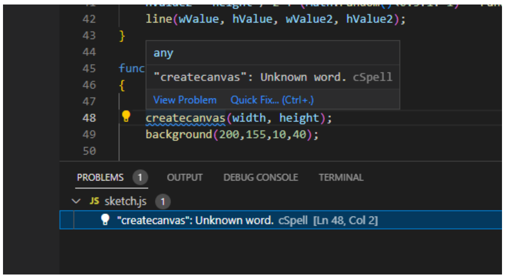
```
Looping 0                                        sketch.js:56
Looping 1                                        sketch.js:56
Looping 2                                        sketch.js:56
Looping 3                                        sketch.js:56
Looping 4                                        sketch.js:56
Looping 5                                        sketch.js:56
Looping 6                                        sketch.js:56
Looping 7                                        sketch.js:56
Looping 8                                        sketch.js:56
Looping 9                                        sketch.js:56
Looping 10                                       sketch.js:56
Looping 11                                       sketch.js:56
Looping 12                                       sketch.js:56
Looping 13                                       sketch.js:56
Looping 14                                       sketch.js:56
```

Sending messages to the console is possible via:

```
console.log("Message with variable i " + i);
```

The console also contains error messages, for example if we did not do capital C in `createCanvas()`. This is called **Camel-casing**.

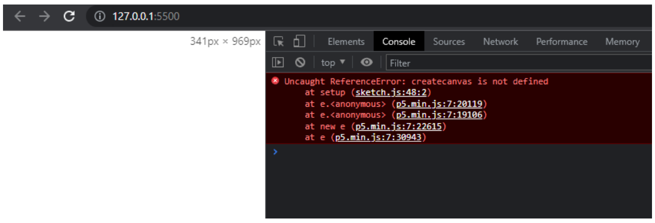
The *Problems view* in VSCode can display the cause of the error, and also hovering the mouse can reveal the issue and provide suggestions.



```
    line(wValue, hValue, wValue2, hValue2);
}

func
{

    createcanvas(width, height);
    background(200,155,10,40);
```

"createcanvas": Unknown word. cSpell

"createcanvas": Unknown word. cSpell [Ln 48, Col 2]



```
Uncaught ReferenceError: createcanvas is not defined
    at setup (sketch.js:48:2)
    at e.<anonymous> (p5.min.js:7:20119)
    at e.<anonymous> (p5.min.js:7:19106)
    at new e (p5.min.js:7:22615)
    at e (p5.min.js:7:30943)
```

A coding mistake is called a **bug**.

When we fix a bug, it is called debugging.

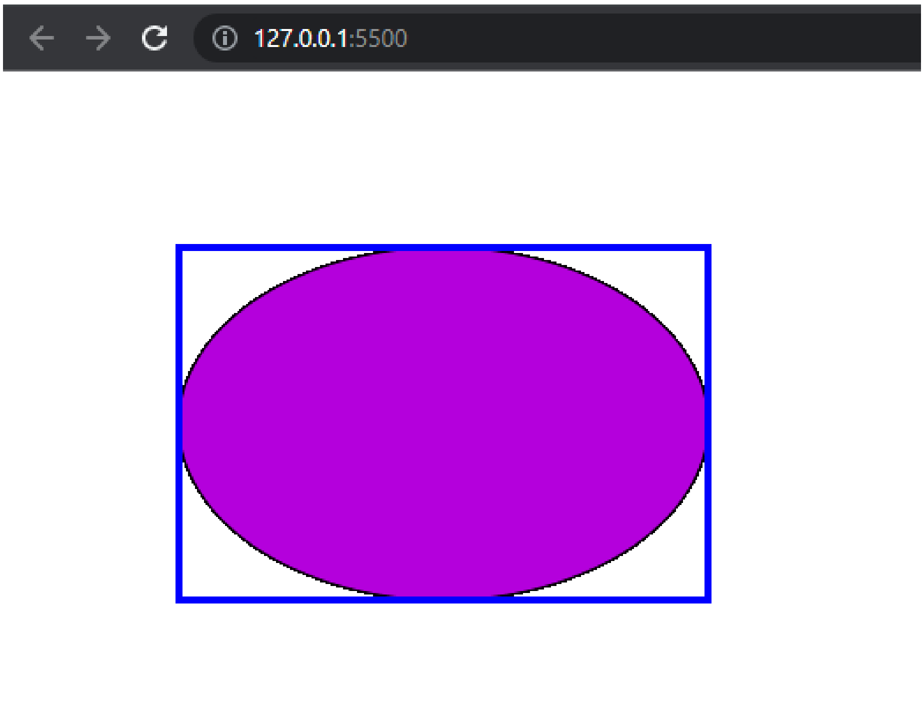
A typo-kind of problem is referred as **syntax error**.

An **argument error** would mean that we did not provide the current number or type of parameters for a method call.

A **semantic error** is a problem with the logic of the code, where it is compile error-free, however the expected result is wrong.

Debugging fix of errors:

```
function setup()
{
  createCanvas(500, 500);
}

function draw()
{
  fill(180, 0, 220);
  strokeWeight(1); // syntax error strokeWeigth
  stroke(0);

  ellipse(250, 200, 300, 200); // semantic error due to wrong position of ellipse at 100, 100

  noFill();
  strokeWeight(4);
  stroke(0, 0, 255);
  rect(100, 100, 300, 200); // syntax error due to missing comma, syntax error due to Camel-case Rect
}
```


127.0.0.1:5500