find all code at

https://github.com/explorablesforsfi/js-d3-tutorial

@BENFMAIER AT SFI

INTRO TO JS & D3

TEXT

HOW TO MAKE SOMETHING USABLE FOR THE WEB

- Arrange things

- style things

- Connect them to code

- code

VARIABLES AND PRINTING


```
<meta charset="utf-8">

<head>
</head>

<body>

<script>
"use strict";

let a = 0;
let b = null;
let c = [989,34];
let d = {a: 0, b:c};

const yello = "yello";

console.log("a:", a);
console.log("b:", b);
console.log("c:", c);
console.log("c[0]:", c[0]);
console.log("c[2]:", c[2]);
console.log("d.a:", d.a);
console.log("d['a']:", d['a']);
console.log(yello,"world");

c[100] = 100;
console.log("c[100]:",c[100]);
console.log(c);
</script>

</body>
```

HTML basic skeleton

Starting a JS segment
"use strict" is better for debugging (gives errors when variables are misused)

"null" is like "nullptr" in C++ or "None" in Python

"let" defines a variable within the current scope (use "var" for global variable)

console.log() is like an interactive print statement

You can add things to an arbitrary position of an array

VARIABLES AND PRINTING


```html
<meta charset="utf-8">

<head>
</head>

<body>

<script>
"use strict";

let a = 0;
let b = null;
let c = [989,34];
let d = {a: 0, b:c};

const yello = "yello";

console.log("a:", a);
console.log("b:", b);
console.log("c:", c);
console.log("c[0]:", c[0]);
console.log("c[2]:", c[2]);
console.log("d.a:", d.a);
console.log("d['a']:", d['a']);
console.log(yello,"world");

c[100] = 100;
console.log("c[100]:",c[100]);
console.log(c);
</script>

</body>
```

Output

```
a: 0
b: null
c: ▶ (2) [989, 34]
c[0]: 989
c[2]: undefined
d.a: 0
d['a']: 0
yello world
c[100]: 100
▶ (101) [989, 34, empty × 98, 100]
```

FUNCTIONS

index.html

```html
<meta charset="utf-8">

<head>
</head>

<body>

<script src="function_example_1.js"></script>
<script src="function_example_2.js"></script>

</body>
```

load external scripts

function_example_1.js

```js
"use strict";

let print_text = "was asked to print the following:";

function print_simple(a) {
  console.log(print_text, a);
}

const yello = "yello";

print_simple(yello);
```

`print_text` will be available to all functions

define a function which puts stuff in the console

define a global constant

call the function

FUNCTIONS

index.html

```html
<meta charset="utf-8">

<head>
</head>

<body>

<script src="function_example_1.js"></script>
<script src="function_example_2.js"></script>

</body>
```

load external scripts

function_example_1.js

```js
"use strict";

let print_text = "was asked to print the following:";

function print_simple(a) {
  console.log(print_text, a);
}

const yello = "yello";

print_simple(yello);
```

output

```
was asked to print the following: yello
```

FUNCTIONS

index.html

```html
<meta charset="utf-8">

<head>
</head>

<body>

<script src="function_example_1.js"></script>
<script src="function_example_2.js"></script>

</body>
```

function_example_1.js

```js
"use strict";

let print_text = "was asked to print the following:";

function print_simple(a) {
  console.log(print_text, a);
}

const yello = "yello";

print_simple(yello);
```

function_example_2.js

```js
"use strict";

function get_string_representation(a) {
  return "" + a;
}

function print(a) {
  console.log(get_string_representation(a));
}

// the variable `yello` is still declared
// from function_example_1.js
print(yello);

// print a number
let number = 2;
print(number);

print_simple(yello);
```

define a function which type-casts a variable to a `String` object

this function gets a String representation of a variable and prints it

This number is type-cast to a string and printed

This function is still available from `function_example_1.js`

FUNCTIONS

index.html

```html
<meta charset="utf-8">

<head>
</head>

<body>

<script src="function_example_1.js"></script>
<script src="function_example_2.js"></script>

</body>
```

function_example_1.js

```js
"use strict";

let print_text = "was asked to print the following:";

function print_simple(a) {
  console.log(print_text, a);
}

const yello = "yello";

print_simple(yello);
```

function_example_2.js

```js
"use strict";

function get_string_representation(a) {
  return "" + a;
}

function print(a) {
  console.log(get_string_representation(a));
}

// the variable `yello` is still declared
// from function_example_1.js
print(yello);

// print a number
let number = 2;
print(number);

print_simple(yello);
```

Output

```
yello
2
was asked to print the following: yello
```

TYPECASTING

```
> 2+"2"
<· "22"
> "2"+2
<· "22"
> +"2"+2
<· 4
> +"2.2"+2
<· 4.2
```

number + string -> string

string + number -> string

+ string -> number
+ string + number -> number

TYPECASTING

```
> 2+"2"
"22"
> "2"+2
"22"
> +"2"+2
4
> +"2.2"+2
4.2
```

number + string -> string

string + number -> string

+ string -> number
+ string + number -> number

```
> console.log("see this typecast:" + 2 + 2 + " ... see?")
see this typecast:22 ... see?
undefined
> console.log("see this typecast:" + (2 + 2) + " ... see?")
see this typecast:4 ... see?
```

MATH

```
"use strict";

console.log( Math.exp(1) );  // exp(1)
console.log( Math.exp(0) );  // 1
console.log( Math.sin(2*Math.PI) ); // 0
console.log( Math.floor(2323.23) ); // 2323
console.log( Math.pow(2,3) );       // 8

let a = [1,2,3];

console.log(a[0])
console.log(a[4 % 3]) // 4 mod 3 = 1 ==> a[1] -> 2
console.log(a[Math.floor(2.5)]) // a[2] -> 3
console.log(Math.random()) // random number [0,1)
```

FOR-LOOPS AND MAPS


```
 1 "use strict";
 2
 3 let a = [1,2,3];
 4
 5 for(let i=0; i<a.length; ++i) {
 6   console.log(a[i]);
 7 }
 8
 9 a.forEach( function (element) {
10   console.log(element);
11 });
12
13 let c = a.map(function (element) { return Math.sin(element*Math.PI/2) });
14 let d = a.map(element => Math.sin(element*Math.PI/2) );
15
16 console.log(c);
17 console.log(d);
```

print each element

for each element in the array, call a function which prints it

for each element of the array, call a function, which manipulates it and returns the manipulation, then return an array of the whole manipulated array

SELECT DOM OBJECTS WITH D3

index.html

```html
<meta charset="utf-8">

<head>
    <script src="/libs/d3.v4.min.js"></script> <!-- load d3 -->
</head>

<body>
    <div></div>
    <script src="d3_example.js"></script>
</body>
```

d3_example.js

```js
"use strict";

let this_div = d3.selectAll("div")
                 .text("hey");



let another_div = this_div.append("p");


another_div.text("ditto");
```

selectAll elements called "div"
make " " to " hey "

"selectAll" is a function which does something and then returns the selection. "text" is a function which puts the text in the selection and then, again, returns the selection.

the selection can then further be manipulated

SELECT DOM OBJECTS WITH D3

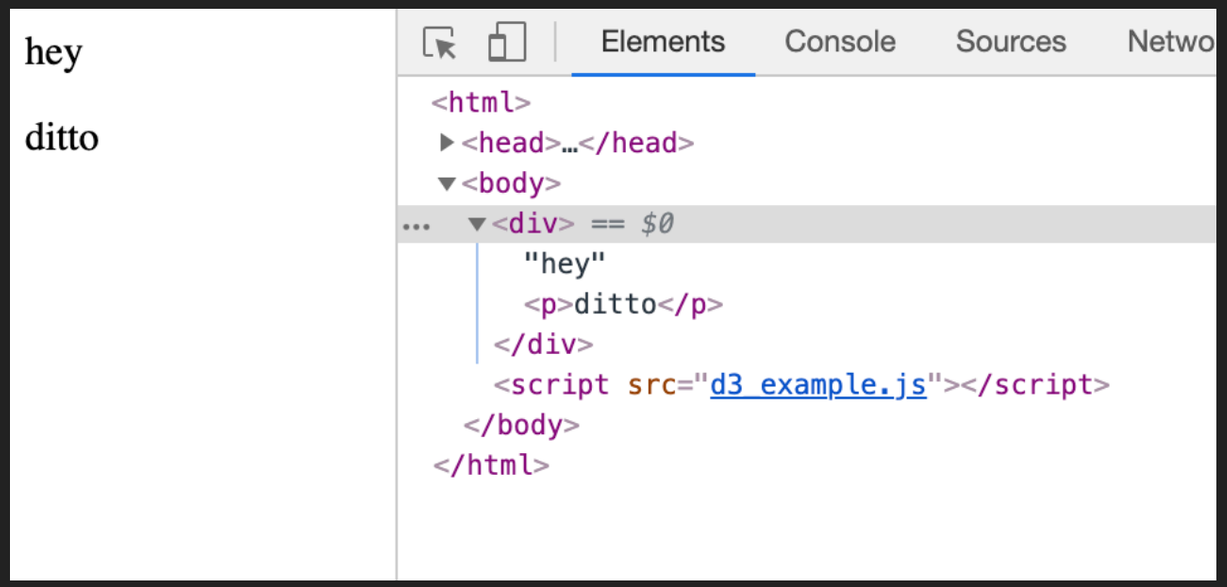
index.html

```html
<meta charset="utf-8">

<head>
    <script src="/libs/d3.v4.min.js"></script> <!-- load d3 -->
</head>

<body>
    <div></div>
    <script src="d3_example.js"></script>
</body>
```

d3_example.js

```js
"use strict";

let this_div = d3.selectAll("div")
                 .text("hey");




let another_div = this_div.append("p");


another_div.text("ditto");
```

output



CHAINING IN D3

index.html

```
<meta charset="utf-8">

<head>
    <script src="/libs/d3.v4.min.js"></script>
</head>

<body>
    <p></p>
    <script src="d3_chaining.js"></script>
</body>
```

d3_chaining.js

```
"use strict";

let p = d3.select("p")
                .text("hey")
                .attr("align", "center")
                .attr("style", "font-size: 30px; border: 1px solid black")
              ;
```

select first <p> and return selection

put text in

give new attribute

Style with CSS

CHAINING IN D3

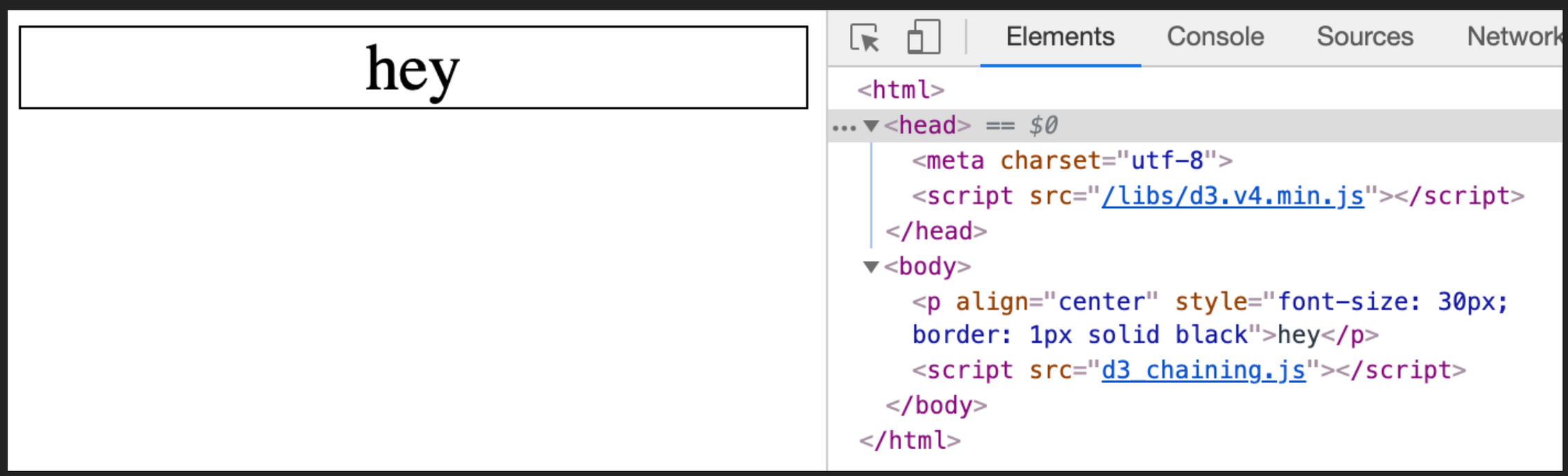
index.html

```
1 <meta charset="utf-8">
2
3 <head>
4     <script src="/libs/d3.v4.min.js"></script>
5 </head>
6
7 <body>
8     <p></p>
9     <script src="d3_chaining.js"></script>
10 </body>
```

d3_chaining.js

```
1 "use strict";
2
3 let p = d3.select("p")
4             .text("hey")
5             .attr("align", "center")
6             .attr("style", "font-size: 30px; border: 1px solid black")
7         ;
```

output



TIMERS IN D3

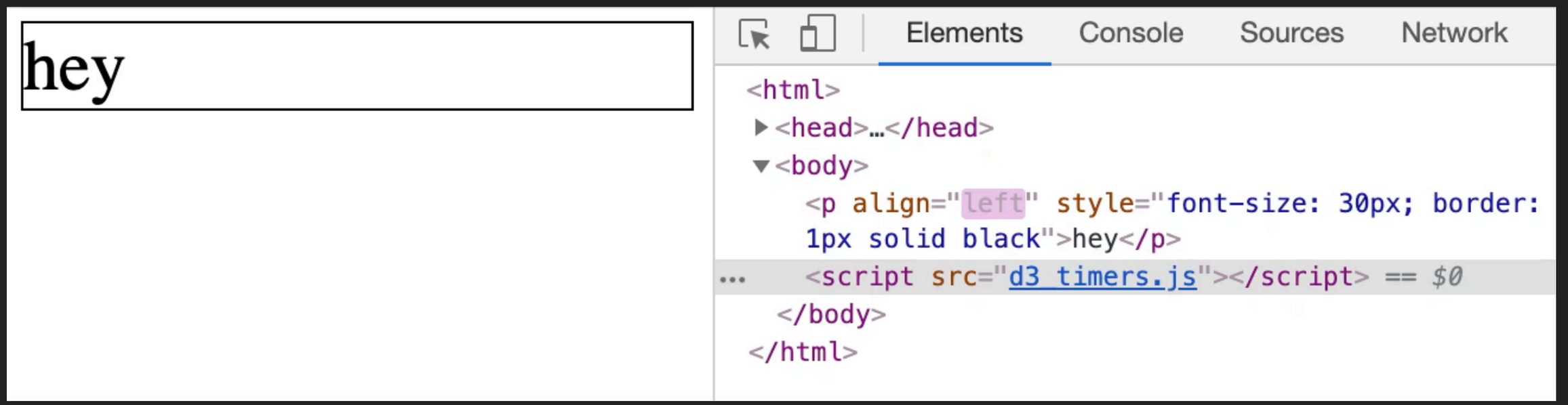
d3_timers.js

```
"use strict";

let p = d3.select("p")
                  .text("hey")
                  .attr("align", "center")
                  .attr("style", "font-size: 30px; border: 1px solid black")
                ;

let timer = d3.interval(update, 1000); // call `update` every second


let state = 0;
let aligns = ["left","center","right"]

function update() {
  p.attr("align",aligns[state % 3])

  state++; // state = state + 1
}
```

TIMERS IN D3

d3_timers.js

```
"use strict";

let p = d3.select("p")
                  .text("hey")
                  .attr("align", "center")
                  .attr("style", "font-size: 30px; border: 1px solid black")
                ;

let timer = d3.interval(update, 1000); // call `update` every second


let state = 0;
let aligns = ["left","center","right"]

function update() {
  p.attr("align",aligns[state % 3])

  state++; // state = state + 1
}
```



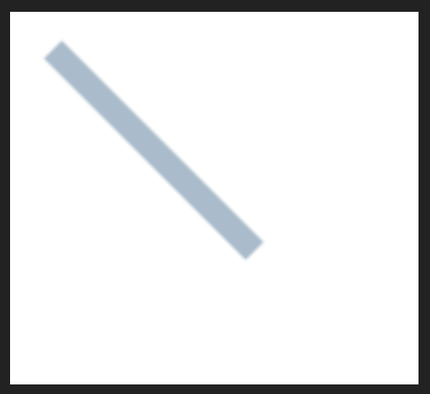
DRAWING ON CANVAS

```
1 "use strict";
2
3 let width = 200, height = 200;
4
5 let canvas = d3.select("body")
6                 .append("canvas");
7
8 canvas.attr("width",width).attr("height",height);
9
10 let context = canvas.node().getContext('2d');
11
12 context.lineWidth = 10;
13 context.strokeStyle = "#abc";
14 context.beginPath();
15 context.moveTo(10,10);
16 context.lineTo(90,90);
17 context.stroke();
```

append a `canvas` object to the body

context is for drawing

draw a 10px wide line from (10,10) to (90,90)



output

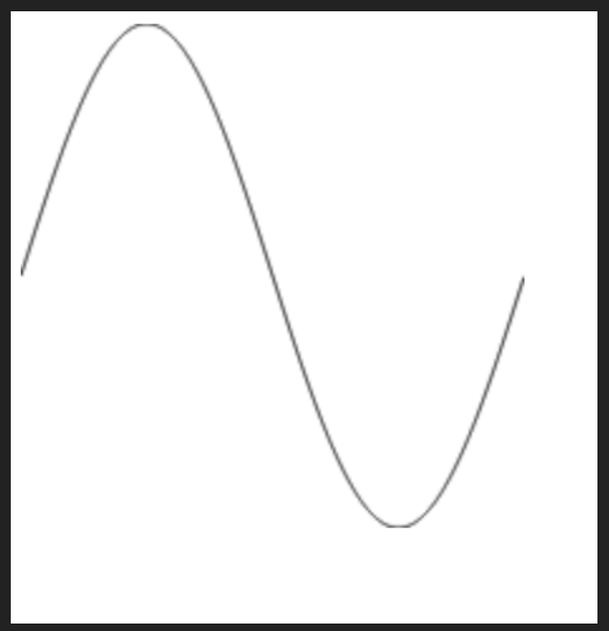
A PLOT ON CANVAS


```
"use strict";

// create canvas
let width = 200, height = 200;
let canvas = d3.select("body").append("canvas")
               .attr("width",width).attr("height",height);
let context = canvas.node().getContext('2d');

// a map function .. map [0,2pi] to the canvas [0,width]
let xScale = d3.scaleLinear().domain([0,2*Math.PI]).range([0,width]);

// a map function .. map [-1,1] to the canvas [height,0]
// (flip coordinate system)
let yScale = d3.scaleLinear().domain([-1,+1]).range([height,0]);

// create x values from 0 to 2pi
let N_values = 1000;
let x = d3.range(N_values).map( n => n/N_values*2*Math.PI);

// compute y = sin(x)
let y = x.map( _x => Math.sin(_x) );

// draw
context.lineWidth = 1;
context.strokeStyle = "#000";
context.beginPath();

context.moveTo(xScale(x[0]),yScale(y[0]));

for (let i=1; i<x.length; ++i)
  context.lineTo(xScale(x[i]),yScale(y[i]));

context.stroke();
```

A PLOT ON CANVAS

```
"use strict";

// create canvas
let width = 200, height = 200;
let canvas = d3.select("body").append("canvas")
               .attr("width",width).attr("height",height);
let context = canvas.node().getContext('2d');

// a map function .. map [0,2pi] to the canvas [0,width]
let xScale = d3.scaleLinear().domain([0,2*Math.PI]).range([0,width]);

// a map function .. map [-1,1] to the canvas [height,0]
// (flip coordinate system)
let yScale = d3.scaleLinear().domain([-1,+1]).range([height,0]);

// create x values from 0 to 2pi
let N_values = 1000;
let x = d3.range(N_values).map( n => n/N_values*2*Math.PI);

// compute y = sin(x)
let y = x.map( _x => Math.sin(_x) );

// draw
context.lineWidth = 1;
context.strokeStyle = "#000";
context.beginPath();

context.moveTo(xScale(x[0]),yScale(y[0]));

for (let i=1; i<x.length; ++i)
  context.lineTo(xScale(x[i]),yScale(y[i]));

context.stroke();
```

output

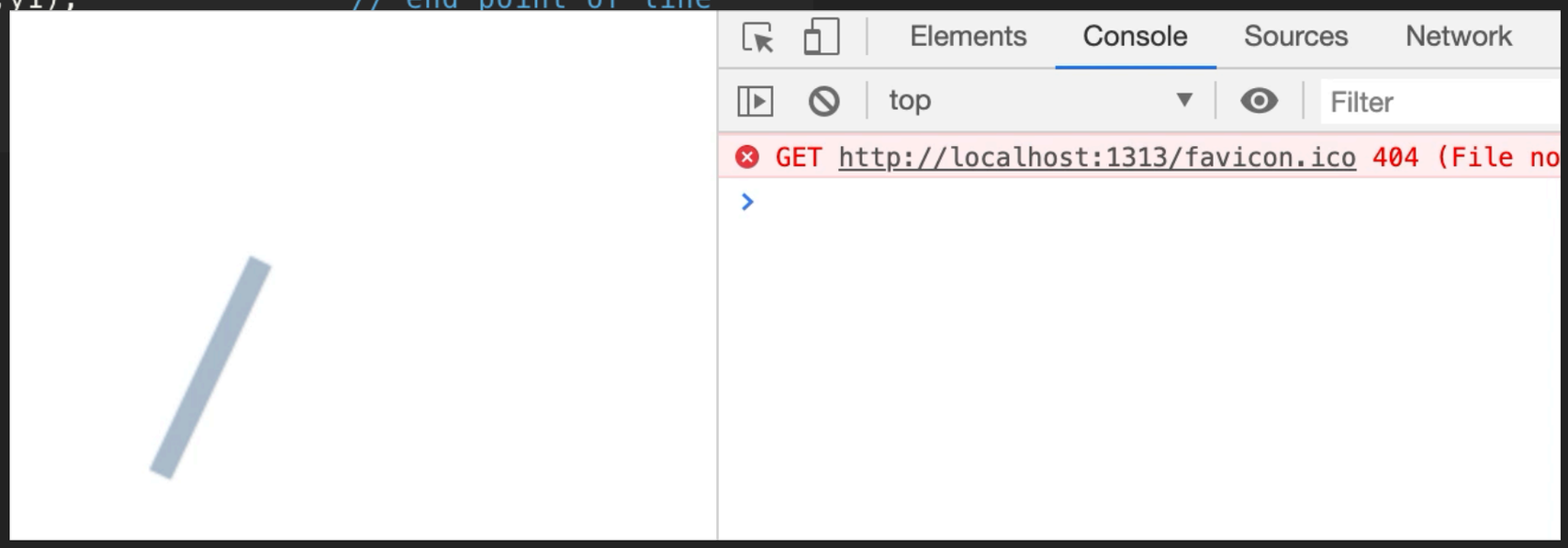


ANIMATION ON CANVAS


```
"use strict";

let width = 200, height = 200;
let canvas = d3.select("body").append("canvas")
               .attr("width",width).attr("height",height);
let context = canvas.node().getContext('2d');
let timer = d3.interval(update,25) // update drawing every 25ms

context.lineWidth = 10;         // set drawing line styles
context.strokeStyle = "#abc";

let angle = 0;                  // initial angle
let d_angle = Math.PI/25;   // increase angle by pi/25 per update
let x0 = width/2;               // origin
let y0 = height/2;
let radius = height/2;      // radius of line

function update ()
{
  context.clearRect(0,0,width,height); // clear drawing
  context.beginPath();                 // start line
  context.moveTo(x0,y0);               // move to origin

  let x1 = radius * Math.cos(angle) + x0; // compute updated
  let y1 = radius * Math.sin(angle) + y0; // position

  context.lineTo(x1,y1);               // end point of line
  context.stroke();                    // draw

  angle += d_angle;  // update angle
}
```

ANIMATION ON CANVAS


```
"use strict";

let width = 200, height = 200;
let canvas = d3.select("body").append("canvas")
               .attr("width",width).attr("height",height);
let context = canvas.node().getContext('2d');
let timer = d3.interval(update,25) // update drawing every 25ms

context.lineWidth = 10;       // set drawing line styles
context.strokeStyle = "#abc";

let angle = 0;              // initial angle
let d_angle = Math.PI/25;   // increase angle by pi/25 per update
let x0 = width/2;           // origin
let y0 = height/2;
let radius = height/2;      // radius of line

function update ()
{
  context.clearRect(0,0,width,height); // clear drawing
  context.beginPath();                 // start line
  context.moveTo(x0,y0);               // move to origin

  let x1 = radius * Math.cos(angle) + x0; // compute updated
  let y1 = radius * Math.sin(angle) + y0; // position

  context.lineTo(x1,y1);               // end point of line
  context.stroke();

  angle += d_angle;
}
```



INTERACTION

index.html

```
<meta charset="utf-8">

<head>
    <script src="/libs/d3.v4.min.js"></script>
</head>

<body>
    <script src="animating_on_canvas.js"></script>
    <script src="interaction.js"></script>
</body>
```

interaction.js

```
"use strict";

let button = d3.select("body")
               .append("button")
               .text("pause")
               .on("click", function() {
                 if (button.text() == 'pause')
                 {
                   timer.stop();
                   button.text("play");
                 }
                 else
                 {
                   timer.restart(update);
                   button.text("pause");
                 }
               });
```

same as in last slide

```
let timer = d3.interval(update,25)

function update ()
{
```

add a button to the body which says "pause"

when clicked, run the following function

* button.text() is a context-sensitive function.
* when called without arguments it returns the value of the HTML-property
* when called with argument, it sets the HTML-property and returns the selection.

INTERACTION

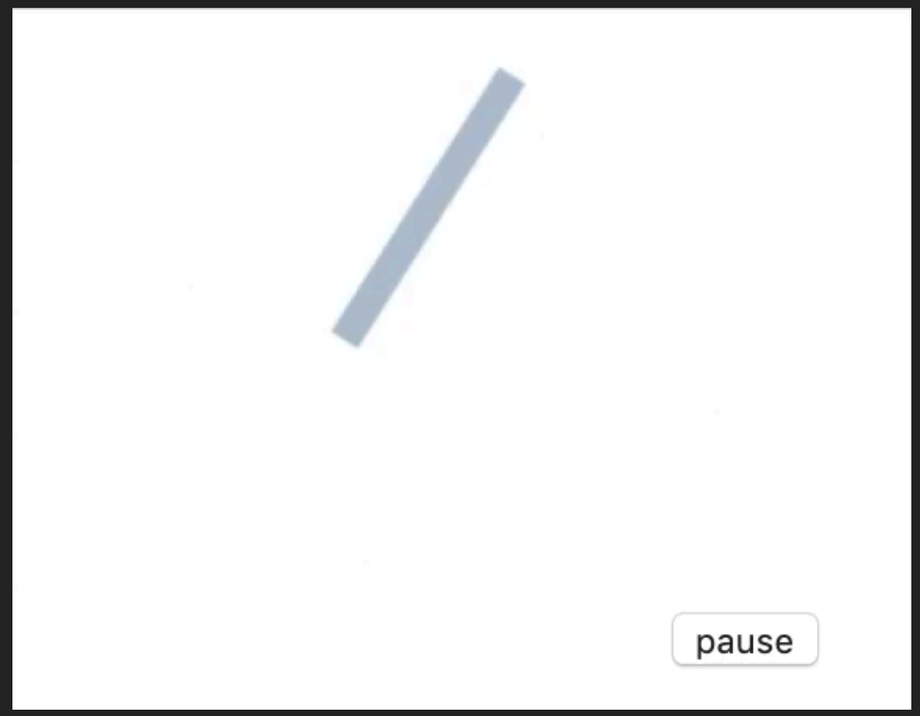
index.html

```html
<meta charset="utf-8">

<head>
    <script src="/libs/d3.v4.min.js"></script>
</head>

<body>
    <script src="animating_on_canvas.js"></script>
    <script src="interaction.js"></script>
</body>
```

interaction.js

```js
"use strict";

let button = d3.select("body")
               .append("button")
               .text("pause")
               .on("click", function() {
                 if (button.text() == 'pause')
                 {
                   timer.stop();
                   button.text("play");
                 }
                 else
                 {
                   timer.restart(update);
                   button.text("pause");
                 }
               });
```

output



CLASSES

index.html

```
<meta charset="utf-8">

<head>
    <script src="/libs/d3.v4.min.js"></script>
</head>
<body>
    <div id="container"></div>
    <p>open the console and
    type in 'hey.changecolor()'</p>
    <script src="hey_div.js"></script>
</body>
```

we want to create the container DOM
before we call the script

hey_div.js

```
"use strict";

class heyDiv {

  constructor(div_id)
  {
    this.div = d3.select(div_id);
    this.div.text("hey");
  }

  changecolor()
  {
    this.div
        .attr("style", "color: rgb(" + (255*Math.random()) +","
                                + (255*Math.random()) + ","
                                + (255*Math.random())
                            +")"
             );
  }
}

let hey = new heyDiv("#container");
```

define class
constructor is invoked upon creation

construct a "div" and let it be a
property of this object (this-keyword)

one method of this class will be that
upon request it will change the
font color

create the new object and save it to
a variable

CLASSES

index.html

```
<meta charset="utf-8">

<head>
    <script src="/libs/d3.v4.min.js"></script>
</head>
<body>
    <div id="container"></div>
    <p>open the console and
    type in 'hey.changecolor()'</p>
    <script src="hey_div.js"></script>
</body>
```

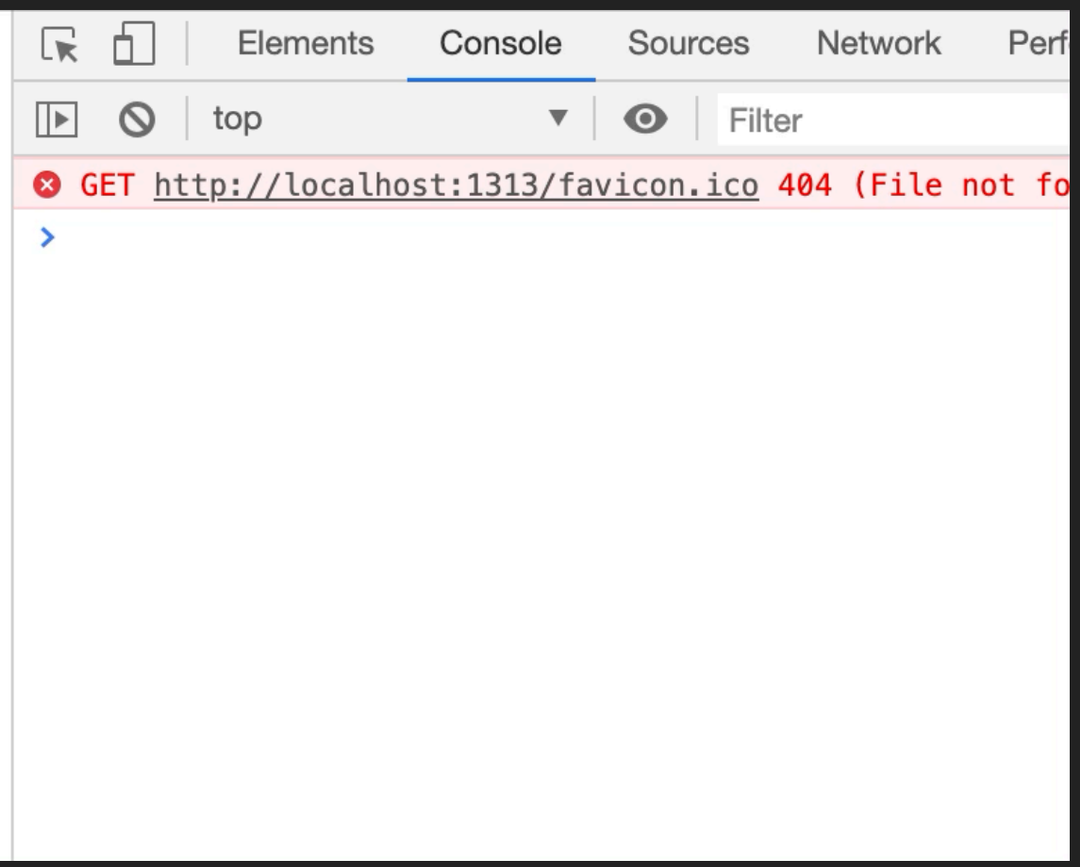
hey_div.js

```
"use strict";

class heyDiv {

  constructor(div_id)
  {
    this.div = d3.select(div_id);
    this.div.text("hey");
  }

  changecolor()
  {
    this.div
        .attr("style", "color: rgb(" + (255*Math.random()) +","
                                     + (255*Math.random()) + ","
                                     + (255*Math.random())
                                   +")"
              );
  }
}

let hey = new heyDiv("#container");
```

output

hey

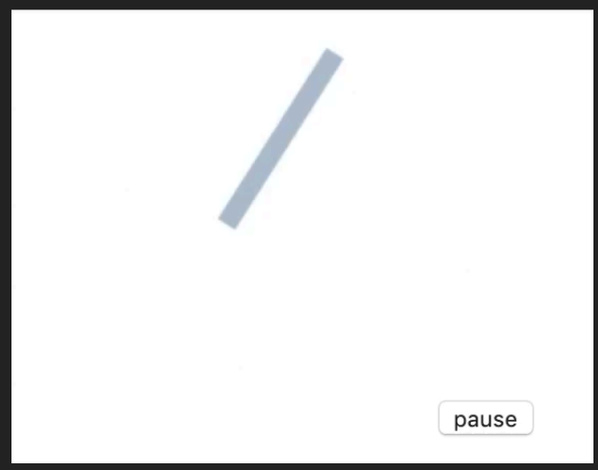
open the console and type in 'hey.changecolor()'



A RATHER COMPLICATED CLASS

```
<meta charset="utf-8">

<head>
    <script src="/libs/d3.v4.min.js"></script>
    <script src="rotator.js"></script>
</head>

<body>
    <script>

        // define an array of any for elements
        let a = [1,2,3,4];
        let rotators = [];  // an array which will contain the rotator objects

        // for each element
        a.forEach(function (element) {
            // define the div id
            let this_div_id = "div_" + element;

            // create the div
            d3.select("body")
              .append("div")
              .attr("id",this_div_id);

            // create the rotator and push to other array
            let rotator = new Rotator('#'+this_div_id, 200,200);
            rotators.push(rotator);

            // make the rotator go
            rotator.playpause()

            // note that we could've used `map` here instead of
            // rotators = []
            // and rotators.push()
        });


        // make the first rotator green
        rotators[0].context.strokeStyle = "#afa";

        // make the second rotator thinner
        rotators[1].context.lineWidth = 3;

        // make the third rotator go faster
        rotators[2].d_angle = Math.PI/10;

        // make the last rotator go in the other direction
        rotators[3].d_angle *= -1;
    </script>
    </script>
</body>
```

idea: generalize the button-rotator concept and allow for individual manipulation of these objects



A RATHER COMPLICATED CLASS


```
"use strict";

class Rotator {

  constructor(div_id, width, height)
  {
    this.width = width;
    this.height = height;
    this.canvas = d3.select(div_id).append("canvas")
                    .attr("width",width).attr("height",height);

    this.context = this.canvas.node().getContext('2d');
    this.context.lineWidth = 10;       // set drawing line styles
    this.context.strokeStyle = "#abc";

    this.angle = 0;             // initial angle
    this.d_angle = Math.PI/25;  // increase angle by pi/25 per update
    this.x0 = this.width/2;          // origin
    this.y0 = this.height/2;
    this.radius = this.height/2;     // radius of line

    // initial situation is that it is not running
    this.is_running = false;

    this.init_button(div_id);

  }

  init_button(div_id)
  {
    // `this` usually refers to the last defined function in the call stack,
    // i.e. in this case to the class (which is syntactic sugar for a function
    // we're saving this reference to the class in a local variable
    let self = this;

    this.button = d3.select(div_id)
                    .append("button")
                    .text("pause")
                    .on("click", function() {
                      // here, `this` would refer to the button and
                      // therefore we use `self` which we declared ealrier
                      // and which refers to the Rotator class
                      console.log(this,self);
                      self.playpause(self);
                    });
  }

  playpause(self=null){

    //if playpause is just called as playpause(), assume that the call came
    //from within the class and as such, `this` refers to the class.
    //otherwise, `self` is assumed to be explicitly given when called
    //(e.g. when a button is pressed)
    if (self === null)
      self = this;

    if (!self.is_running)
    {
      self.timer = d3.interval( function () {
                        // here, `this` would refer to the timer and
                        // therefore we use `self` which we declared ealrier
                        // and which refers to the Rotator object
                         self.update(self);
                      } ,25); // update drawing every 25ms
      self.button.text('pause');
    }
    else
    {
      self.button.text('play');
      self.timer.stop();
    }

    // change running status
    self.is_running = !self.is_running;
  }

  update(self=null)
  {
    //if playpause is just called as update(), assume that the call came
    //from within the class and as such, `this` refers to the class.
    //otherwise, `self` is assumed to be explicitly given when called
    //(e.g. when the timer calls this function)
    if (self === null)
      self = this;

    self.context.clearRect(0,0,self.width,self.height); // clear drawing
    self.context.beginPath();                  // start line
    self.context.moveTo(self.x0,self.y0);               // move to origin

    let x1 = self.radius * Math.cos(self.angle) + self.x0; // compute updated
    let y1 = self.radius * Math.sin(self.angle) + self.y0; // position

    self.context.lineTo(x1,y1);                // end point of line
    self.context.stroke();                     // draw

    self.angle += self.d_angle;  // update angle
  }

}
```

A RATHER COMPLICATED CLASS

```html
<meta charset="utf-8">

<head>
    <script src="/libs/d3.v4.min.js"></script>
    <script src="rotator.js"></script>
</head>

<body>
    <script>

        // define an array of any for elements
        let a = [1,2,3,4];
        let rotators = [];  // an array which will contain the rotator objects

        // for each element
        a.forEach(function (element) {
            // define the div id
            let this_div_id = "div_" + element;

            // create the div
            d3.select("body")
              .append("div")
              .attr("id",this_div_id);

            // create the rotator and push to other array
            let rotator = new Rotator('#'+this_div_id, 200,200);
            rotators.push(rotator);

            // make the rotator go
            rotator.playpause()

            // note that we could've used `map` here instead of
            // rotators = []
            // and rotators.push()
        });


        // make the first rotator green
        rotators[0].context.strokeStyle = "#afa";

        // make the second rotator thinner
        rotators[1].context.lineWidth = 3;

        // make the third rotator go faster
        rotators[2].d_angle = Math.PI/10;

        // make the last rotator go in the other direction
        rotators[3].d_angle *= -1;
    </script>
    </script>
</body>
```