

const g = svg

            .attr("height", 500)

            .attr("width", 500)

            .append("g");

        let rect = g.selectAll("rect").data([0,1,2,3,4,5]);

        console.log(rect);

Output:

\_enter: Array(1)

0: (6) [EnterNode, EnterNode, EnterNode, EnterNode, EnterNode, EnterNode]

length: 1

\_exit: Array(1)

0: []

length: 1

\_groups: Array(1)

0: (6) [empty × 6]

length: 1

\_parents: Array(1)

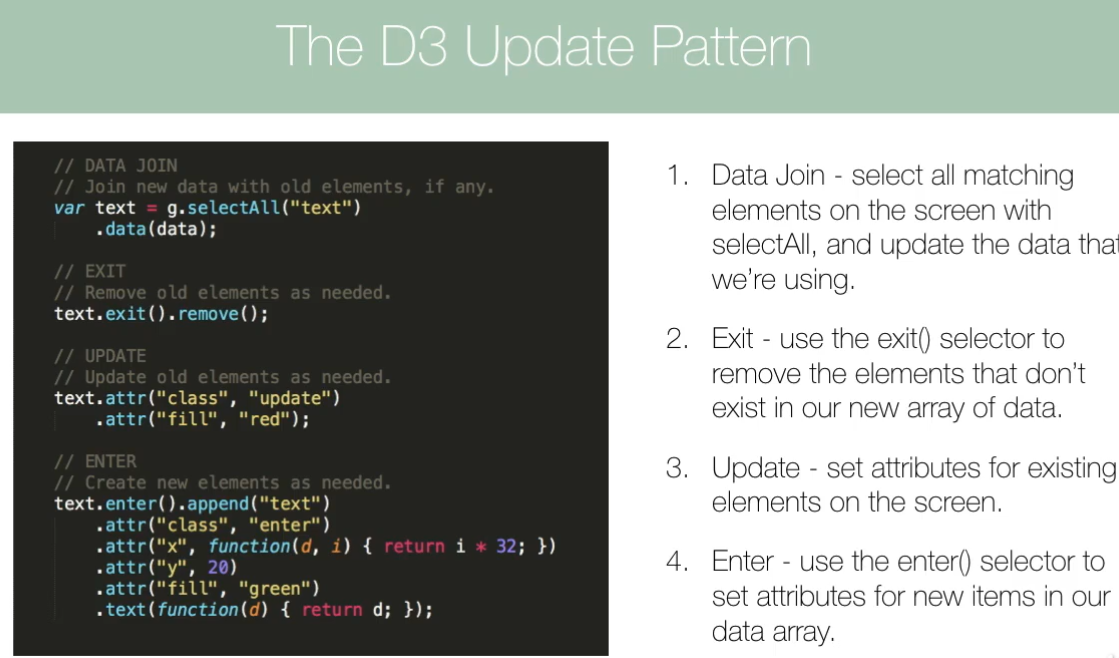
0: g

length: 1

\_enter: represents all of the elements in a data array that don't exist on the page

\_exist: represents the elements that are on the page that don't exist in the data array

\_groups:  representing all the elements that currently exist on the screen



**Update initial pattern:**

**Static content:**

    constructor() {

        super();

        this.toggleInc = true;

        this.bars = null;

        this.digitBar = null;

    }

    componentDidMount() {

        const svg = $D.select("#playground").append("svg");

        $("#playground svg").css({

            "margin-left": "auto",

            "margin-right": "auto",

            display: "block"

        });

        const height = 500;

        const width = 600;

        const g = svg

            .attr("height", height)

            .attr("width", width)

            .append("g");

        g.append("text")

            .attr("x", width / 2)

            .attr("y", height \* 0.9)

            .attr("text-anchor", "middle")

            .attr("font-size", "20px")

            .text("Time line");

        g.append("text")

            .attr("x", -width \* 0.1)

            .attr("y", height / 15)

            .attr("text-anchor", "middle")

            .attr("font-size", "20px")

            .attr("transform", "rotate(-90)")

            .text("Revenue");

        const gx = g

            .append("g")

            .attr(

                "transform",

                "translate(" + width \* 0.15 + "," + height \* 0.8 + ")"

            );

        const gy = g

            .append("g")

            .attr(

                "transform",

                "translate(" + width \* 0.15 + "," + -height \* 0.01 + ")"

            );

**// insert virtual rectangles inside an object**

**// no DOM manipulation**

        this.bars = g.selectAll("rect");

        this.update(height, width, g, gx, gy);

// **Interval updating:**

        $D.interval(() => {

            this.update(height, width, g, gx, gy);

        }, 2000);

    }

**Dynimic content:**

    update = (height, width, g, gx, gy) => {

        $.getJSON("./revenues.json", (data) => {

**// clean data to prevent unexpected behavior**

            data.forEach((d) => {

                d.revenue = parseInt(d.revenue);

                d.profit = parseInt(d.profit);

            });

            this.toggleInc = !this.toggleInc;

            let flag = this.toggleInc === true ? "revenue" : "profit";

            let xrange = $D

                .scaleBand()

                .domain(

                    data.map((d) => {

                        return d.month;

                    })

                )

                .range([0, width \* 0.8])

                .paddingOuter(0.5)

                .paddingInner(0.2);

**// apply a horizontal axis faces downward in range**

**// no DOM manipulation**

**let xAxisCall = $D.axisBottom(xrange);**

**// a group DOM apply the axis rules**

**// DOM has been manipulated**

**gx.call(xAxisCall);**

            let yrange = $D

                .scaleLinear()

                .domain([

                    0,

                    $D.max(

                        data.map((d) => {

                            return d[flag];

                        })

                    )

                ])

                .range([height \* 0.8, 0]);

            let yAxisCall = $D

                .axisLeft(yrange)

                .ticks(data.length)

                .tickFormat((d) => {

                    return d + "$";

                });

            gy.call(yAxisCall);

**// apply data to virtual rectangles**

            this.digitBar = this.bars.**data**(data);

**// extremly important to clean the old data caused DOMs**

**g.selectAll('rect').remove();**

**// append actual rectangle based on the data**

this.digitBar

                .**enter**()

                .append("rect")

                .attr("x", (d) => {

                    return xrange(d.month)+height\*0.2;

                })

                .attr("y", (d) => {

                    return yrange(d[flag]);

                })

                .attr("width", xrange.bandwidth)

                .attr("height", (d) => {

                    return height \* 0.8 - yrange(d[flag]);

                })

                .attr("fill", "aliceblue");

        });

    };