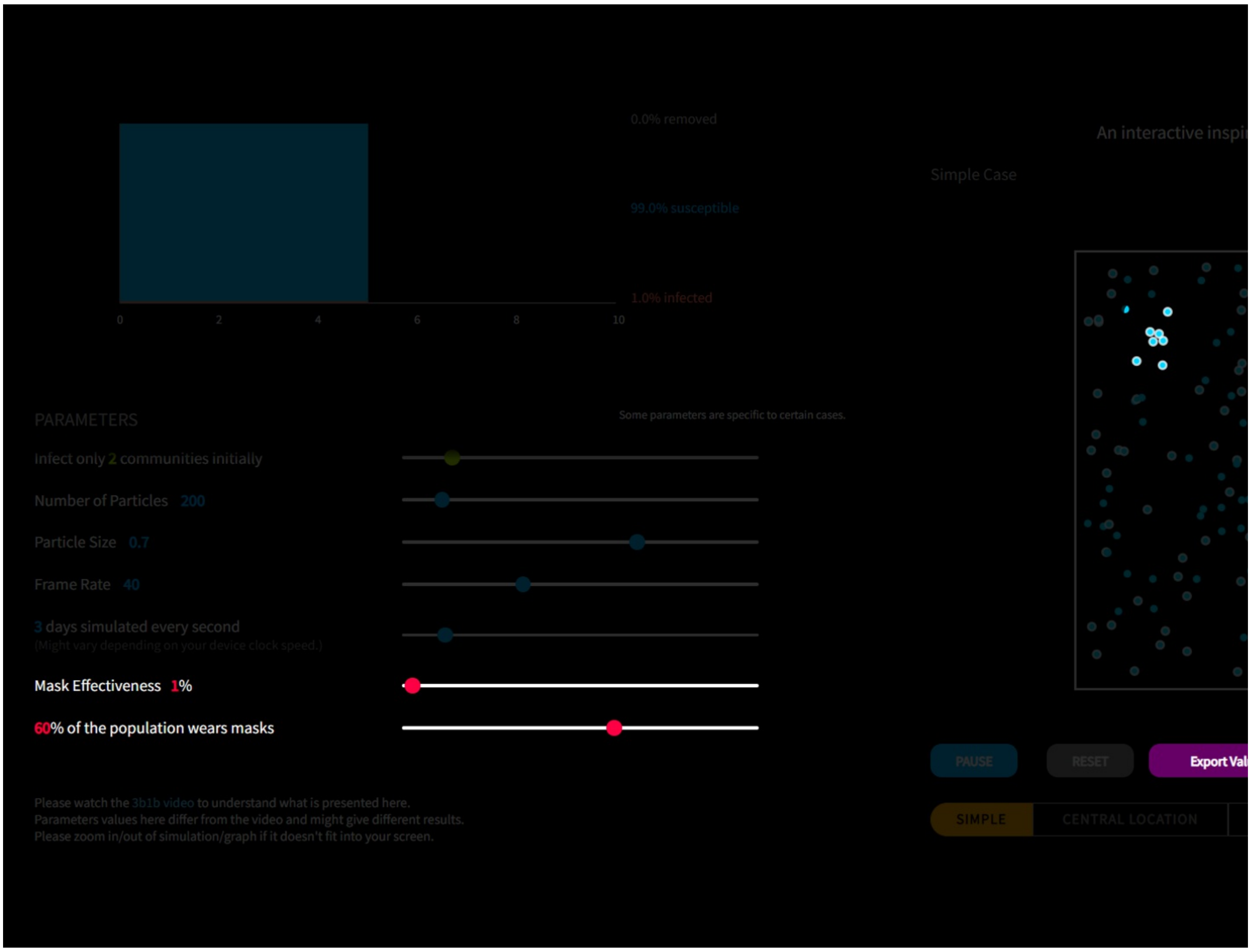
ViewX JavaScript library contains a set of functions and utilities for creating interactive simulations and animations using HTML and JavaScript. The library allows you to create and manipulate graphical objects such as points, paths, circles, and lines on an interactive graph.

Basic Demo

Basic Demo with code at viewX/template.html

Other examples

Simulating the Epidemic



PARAMETERS

Some parameters are specific to certain cases.

Infect only 2 communities initially

Number of Particles 200

Particle Size 0.7

Frame Rate 40

3 days simulated every second
(Might vary depending on your device clock speed.)

Mask Effectiveness 1%

60% of the population wears masks

Please watch the 3b1b video to understand what is presented here.
Parameters values here differ from the video and might give different results.
Please zoom in/out of simulation/graph if it doesn't fit into your screen.

viewX Documentation

Main Features

1. Interactive graph with zoom and drag capabilities.
2. Create and manipulate graphical objects such as points, paths, circles, and lines.
3. Customize the appearance of the graph and its elements, including colors, stroke width, and more.
4. Control the animation and calculation loop for the simulations.

ViewX Functions List

1. Graph/Canvas : viewX.addGraph, viewX.removeGraph
2. Line : viewX.addLine, viewX.updateLine, viewX.removeLine
3. Path : viewX.addPath, viewX.updatePath, viewX.updatePathPoints, viewX.removePath
4. Arrow : viewX.addArrow, viewX.updateArrow, viewX.removeArrow
5. Circle : viewX.addCircle, viewX.updateCircle, viewX.removeCircle
6. Ellipse : viewX.addEllipse, viewX.updateEllipse
7. Text : viewX.addText, viewX.updateText, viewX.removeText
8. Rectangle : viewX.addRectangle, viewX.updateRectangle
9. Point : viewX.addPoint, viewX.updatePoint, viewX.removePoint
10. Arc : viewX.makeArc

Demo Explained

Setting up the Graph

Importing the library.

```
<script type="text/javascript" src="https://cdn.jsdelivr.net/gh/prajwalsouza/viewX/viewx.js"></script>
```

To set up the graph, call the `setUpGraph()` function. This function creates a graph with the specified options, such as axis limits, gridlines visibility, and more. The graph is added to the specified HTML element. Uses the viewX function viewX.addGraph().

Example:

```
function setUpGraph() {
    graphH = document.getElementById('graphHolderH');
    graphoptions = {
        xmax: 10,
        xmin: -10,
        ymax: 10,
        ymin: -10,
        axislocationX: 0,
        axislocationY: 0,
        xaxislabelvisibility: 'no',
        yaxislabelvisibility: 'no',
        fontSize: 1.6,
        unitAspectRatio: 'yes',
        fixAxisStretchCentrally: 'yes',
        scrollZoom: 'yes',
        draggability: 'yes'
    };
    viewX.addGraph(graphH, 'graphName', graphoptions)
}
setUpGraph()
```

Adding Objects to the Graph

To add objects to the graph, use the corresponding `add*` functions such as addPoint, addPath, addCircle, and addLine. These functions take the graph's name and options for the object, such as its position, color, and size.

Example:

```
function addObjects() {
    // Adding a point
    options = {
        x: 5,
        y: 5,
        pointcolor: 'hsla(40, 100%, 50%, 1)',
        pointsize: 1
    };
    viewX.addPoint('graphName', 'samplePoint', options);

    // Adding a path
    options = {
        points: [[4, 0], [0, 4], [-4, 0], [0, -4], [4, 0]],
        pathcolor: 'hsla(190, 100%, 50%, 1)',
        strokewidth: 0.3
    };
    viewX.addPath('graphName', 'samplePath', options);

    // Adding a circle
    options = {
        x: 3,
        y: 6,
        radius: 2,
        circlecolor: 'hsla(80, 100%, 50%, 0.4)',
        stroke: 'hsla(100, 100%, 50%, 1)',
        strokewidth: 1
    };
    viewX.addCircle('graphName', 'sampleCircle', options);

    // Adding a line
    options = {
        x1: 1,
        y1: 1,
        x2: 3,
        y2: 2,
        linecolor: 'hsla(280, 100%, 50%, 0.4)',
        strokewidth: 3
    };
    viewX.addLine('graphName', 'sampleLine', options);
}
addObjects()
```

Animating Objects

To animate objects in the graph, update their properties inside the `plotObjects` function. This function is called repeatedly inside the animation loop. You can update the position, color, and other properties of the objects using the `update*` functions, such as `updatePointXY`, `updateCircle`, etc.

Example:

```
function plotObjects() {
    // Update point position
    toMovePointX = 3 * Math.cos(animLoopCount * 0.01);
    toMovePointY = 3 * Math.sin(animLoopCount * 0.01);
    viewX.updatePointXY('graphName', 'samplePoint', toMovePointX, toMovePointY);

    // Update circle radius and color
    newRadius = 2 + Math.cos(animLoopCount * 0.01);
    newColor = 'hsla(' + animLoopCount / 2 + ', 100%, 50%, 0.4)';
    options = {
        radius: newRadius,
        circlecolor: newColor
    };
    viewX.updateCircle('graphName', 'sampleCircle', options);
}
```

Starting the Animation Loop

To start the animation loop, call the `playAnimationLoop()` function. This function sets up a loop that calls the `plotObjects` function at the specified frame rate.

```
function playAnimationLoop(event) {
    loopinterval = setInterval(frame, frameInterval);
    function frame() {
        animLoopCount = animLoopCount + 1
        if (playing) {
            plotObjects()
        }
    }
}

playAnimationLoop()
```

ViewX Functions

Function: viewX.addGraph

Description

This function adds a new graph to the specified parent `div` element with the given name and graph data. This graph can be customized using various options provided in the `gdata` parameter. The function returns the graph data object.

Syntax

```
viewX.addGraph(parentdiv, name, gdata);
```

Parameters

- `parentdiv` (HTMLDivElement): The parent `div` element where the graph will be added.
- `name` (String): The unique name assigned to the graph. It is used as the identifier for the graph in the library.
- `gdata` (Object): An object containing the graph data and various customization options for the graph appearance and behavior.

Return Value

This function returns an object containing the graph data (modified with the applied customizations).

Example

```
var parentDiv = document.getElementById("graph-container");
var graphName = "myGraph";
var graphData = {
  xmin: -10,
  xmax: 10,
  ymin: -10,
  ymax: 10,
  axislocationX: 0,
  axislocationY: 0,
  unitAspectRatio: "yes",
  xAxisLabel: "X Axis",
  yAxisLabel: "Y Axis"
};
viewX.addGraph(parentDiv, graphName, graphData);
```

Graph Data Options

The `gdata` object can contain various options to customize the appearance and behavior of the graph. The following options are available:

- `xmin` (Number): The minimum value on the x-axis. Default: -1.
- `xmax` (Number): The maximum value on the x-axis. Default: -1.
- `ymin` (Number): The minimum value on the y-axis. Default: 0.
- `ymax` (Number): The maximum value on the y-axis. Default: 0.
- `axislocationX` (Number): The x-coordinate of the y-axis. Default: 0.
- `axislocationY` (Number): The y-coordinate of the x-axis. Default: 0.
- `unitAspectRatio` (String): If set to 'yes', the graph will maintain a 1:1 aspect ratio between the x and y axes. Default: 'no'.
- `fixAxis` (String): Which axis to fix when maintaining a 1:1 aspect ratio. Possible values: 'xaxis', 'yaxis'. Default: 'yaxis'.
- `fixAxisStretchCentrally` (String): If set to 'yes', the graph will stretch along the fixed axis while maintaining the 1:1 aspect ratio. Default: 'no'.
- `xaxisvisibility` (String): If set to 'yes', the x-axis will be visible. Default: 'yes'.
- `yaxisvisibility` (String): If set to 'yes', the y-axis will be visible. Default: 'yes'.
- `xAxisLabel` (String): The label for the x-axis. Default: 'x axis'.
- `yAxisLabel` (String): The label for the y-axis. Default: 'y axis'.
- `xmajorgridlinesvisibility` (String): If set to 'yes', the major gridlines for the x-axis will be visible. Default: 'yes'.
- `ymajorgridlinesvisibility` (String): If set to 'yes', the major gridlines for the y-axis will be visible. Default: 'yes'.
- `xmajorgridlabelvisibility` (String): If set to 'yes', the labels for the major gridlines on the x-axis will be visible. Default: 'yes'.
- `ymajorgridlabelvisibility` (String): If set to 'yes', the labels for the major gridlines on the y-axis will be visible. Default: 'yes'.
- `gridlinenumberX` (Number): The number of gridlines on the x-axis. Default: 10.
- `gridlinenumberY` (Number): The number of gridlines on the y-axis. Default: 10.
- `scrollZoom` (String): If set to 'yes', the graph will zoom in and out with the scroll wheel. Default: 'yes'.
- `draggability` (String): If set to 'yes', the graph can be dragged. Default: 'no'.
- `dragDirection` (String): The direction in which the graph can be dragged. Possible values: 'bothXY', 'xaxis', 'yaxis'. Default: 'bothXY'.
- `dragIfCondition` (String): A condition that must be met for the graph to be draggable. Default: 'true'.
- `runFunctionOnDragEnd` (String): A function to be executed when the drag event ends. Default: ''.
- `runFunctionDuringDrag` (String): A function to be executed during the drag event. Default: ''.

Notes

- The default values for `xmin`, `xmax`, `ymin`, and `ymax` are set to create a graph with equal lengths on both axes when the aspect ratio is maintained.
- The `xmajorgridlinesvisibility`, `ymajorgridlinesvisibility`, `xmajorgridlabelvisibility`, and `ymajorgridlabelvisibility` options can be set to 'no' to hide the corresponding gridlines and labels.
- The `scrollZoom` and `draggability` options can be set to 'no' to disable zooming and dragging, respectively.
- The `dragDirection`, `dragIfCondition`, `runFunctionOnDragEnd`, and `runFunctionDuringDrag` options are only applicable when `draggability` is set to 'yes'.

Function: viewX.addLine

Description

This function adds a new line to the specified graph with the given name and line options. The line can be customized using various options provided in the `lineoptions` parameter.

Syntax

```
viewX.addLine(graphname, linename, lineoptions);
```

Parameters

- `graphname` (String): The name of the graph to which the line will be added.
- `linename` (String): The unique name assigned to the line. It is used as the identifier for the line in the library.
- `lineoptions` (Object): An object containing various customization options for the line appearance.

Return Value

This function returns an array containing the line element and the line options (modified with the applied customizations).

Example

```
 var graphName = "myGraph";
var lineName = "myLine";
var lineOptions = {
  x1: 0,
  y1: 0,
  x2: 5,
  y2: 5,
  strokedasharray: "5,5",
  strokewidth: 2,
  linecolor: "hsla(190, 100%, 50%, 1)"
};
viewX.addLine(graphName, lineName, lineOptions);
```

Line Options

The `lineoptions` object can contain various options to customize the appearance of the line. The following options are available:

- `x1` (Number): The x-coordinate of the starting point of the line. Default: 0.
- `y1` (Number): The y-coordinate of the starting point of the line. Default: 0.
- `x2` (Number): The x-coordinate of the ending point of the line. Default: 0.5.
- `y2` (Number): The y-coordinate of the ending point of the line. Default: 0.5.
- `strokedasharray` (String): A string specifying the stroke-dasharray attribute for the line. Default: "" (solid line).
- `strokewidth` (Number): The width of the line stroke. Default: 1.
- `linecolor` (String): The color of the line in CSS color format. Default: 'hsla(190, 100%, 50%, 1)'.

Function: viewX.addPath

Description

This function adds a new path to the specified graph with the given name and path options. The function returns an array containing the path element and path options.

Syntax

```
 viewX.addPath(graphname, pathname, pathoptions);
```

Parameters

- `graphname` (String): The name of the graph to which the path will be added.
- `pathname` (String): The unique name assigned to the path. It is used as the identifier for the path in the library.
- `pathoptions` (Object): An object containing the path options and various customization options for the path appearance.

Return Value

This function returns an array containing the path element and path options (modified with the applied customizations).

Example

```
 var graphName = "myGraph";
var pathName = "myPath";
var pathOptions = {
  points: [[0, 1], [1, 0]],
  strokewidth: 1,
  pathcolor: "hsla(190, 100%, 50%, 1)",
  pathfillcolor: "none"
};
viewX.addPath(graphName, pathName, pathOptions);
```

Path Options

The `pathoptions` object can contain various options to customize the appearance of the path. The following options are available:

- `points` (Array of Arrays): An array of coordinate pairs representing the points of the path. Default: `[[0, 1], [1, 0]]` .
- `strokewidth` (Number): The stroke width of the path. Default: 1.
- `pathcolor` (String): The color of the path. Default: `'hsla(190, 100%, 50%, 1)'` .
- `pathfillcolor` (String): The fill color of the path. Default: `'none'` .

Notes

- The `points` option should contain an array of coordinate pairs (e.g. `[[0, 1], [1, 0], ...]`).
- To update a path, use the `viewX.updatePath` function.

Function: viewX.addArrow

Description

This function adds a new arrow to the specified graph with the given name and arrow options. The function returns an array containing the arrow element and arrow options.

Syntax

```
 viewX.addArrow(graphname, arrowname, arrowoptions);
```

Parameters

- `graphname` (String): The name of the graph to which the arrow will be added.
- `arrowname` (String): The unique name assigned to the arrow. It is used as the identifier for the arrow in the library.
- `arrowoptions` (Object): An object containing the arrow options and various customization options for the arrow appearance.

Return Value

This function returns an array containing the arrow element and arrow options (modified with the applied customizations).

Example

```
var graphName = "myGraph";
var arrowName = "myArrow";
var arrowOptions = {
  from: [0, 0],
  to: [1, 1],
  strokewidth: 0.4,
  arrowcolor: "hsla(0, 0%, 0%, 1)"
};
viewX.addArrow(graphName, arrowName, arrowOptions);
```

Arrow Options

The `arrowoptions` object can contain various options to customize the appearance of the arrow. The following options are available:

- `from` (Array): A coordinate pair representing the starting point of the arrow. Default: `[0, 0]`.
- `to` (Array): A coordinate pair representing the end point of the arrow. Default: `[1, 1]`.
- `strokewidth` (Number): The stroke width of the arrow. Default: 0.4.
- `arrowcolor` (String): The color of the arrow. Default: `'hsla(0, 0%, 0%, 1)'`.

Notes

- To update an arrow, use the `viewX.updateArrow` function.

Function: viewX.updateArrow

Description

This function updates an existing arrow in the specified graph with new arrow options. The function modifies the arrow element and arrow options in the graph data.

Syntax

```
viewX.updateArrow(graphname, arrowname, newarrowoptions);
```

Parameters

- `graphname` (String): The name of the graph containing the arrow to be updated.
- `arrowname` (String): The name of the arrow to be updated.
- `newarrowoptions` (Object): An object containing the new arrow options to be applied to the arrow.

Example

```
var graphName = "myGraph";
var arrowName = "myArrow";
var newArrowOptions = {
  from: [0, 0],
  to: [2, 2],
  strokewidth: 0.6,
  arrowcolor: "hsla(0, 0%, 0%, 1)"
};
viewX.updateArrow(graphName, arrowName, newArrowOptions);
```

Notes

- The `newarrowoptions` object can contain any of the arrow options available in the `viewX.addArrow` function.

Function: viewX.addCircle

Description

This function adds a circle to the specified graph with the given name and circle options. The circle appearance can be customized using various options provided in the `circleoptions` parameter.

Syntax

```
viewX.addCircle(graphname, circlename, circleoptions);
```

Parameters

- `graphname` (String): The name of the graph to which the circle will be added.
- `circlename` (String): The unique name assigned to the circle. It is used as the identifier for the circle in the library.
- `circleoptions` (Object): An object containing the circle options for customizing the appearance of the circle.

Return Value

This function returns an array containing the circle SVG element and the circle options object.

Example

```
var graphName = "myGraph";
var circleName = "myCircle";
var circleOptions = {
  x: 2,
  y: 3,
  radius: 1,
  stroke: 'hsla(190, 100%, 50%, 0.5)',
  strokewidth: 0.1,
  circlecolor: 'hsla(190, 100%, 50%, 1)'
};
viewX.addCircle(graphName, circleName, circleOptions);
```

Function: viewX.addEllipse

Description

This function adds an ellipse to the specified graph with the given name and ellipse options. The ellipse appearance can be customized using various options provided in the `ellipseoptions` parameter.

Syntax

```
viewX.addEllipse(graphname, ellipsename, ellipseoptions);
```

Parameters

- `graphname` (String): The name of the graph to which the ellipse will be added.
- `ellipsename` (String): The unique name assigned to the ellipse. It is used as the identifier for the ellipse in the library.
- `ellipseoptions` (Object): An object containing the ellipse options for customizing the appearance of the ellipse.

Return Value

This function returns an array containing the ellipse SVG element and the ellipse options object.

Example

```
 var graphName = "myGraph";
var ellipseName = "myEllipse";
var ellipseOptions = {
  x: 2,
  y: 3,
  rx: 1,
  ry: 2,
  stroke: 'hsla(190, 100%, 50%, 0.5)',
  strokewidth: 0.1,
  ellipsecolor: 'hsla(190, 100%, 50%, 1)'
};
viewX.addEllipse(graphName, ellipseName, ellipseOptions);
```

Function: viewX.addText

Description

This function adds a new text element to the specified graph with the given name and text options. The text element can be customized using various options provided in the `textoptions` parameter.

Syntax

```
 viewX.addText(graphname, textname, textoptions);
```

Parameters

- `graphname` (String): The name of the graph to which the text element will be added.
- `textname` (String): The unique name assigned to the text element. It is used as the identifier for the text element in the library.
- `textoptions` (Object): An object containing the text options and various customization options for the text element appearance and behavior.

Return Value

This function returns an array containing the text element and the text options (modified with the applied customizations).

Example

```
 var graphName = "myGraph";
var textName = "myText";
var textOptions = {
  x: 5,
  y: 5,
  text: "Hello, world!",
  textAlign: "center",
  fontSize: 14,
  fontFamily: "Arial",
  textcolor: "hsla(190, 100%, 0%, 1)"
};
viewX.addText(graphName, textName, textOptions);
```

Text Options

The `textoptions` object can contain various options to customize the appearance and behavior of the text element. The following options are available:

- `x` (Number): The x-coordinate of the text element's position. Default: 0.
- `y` (Number): The y-coordinate of the text element's position. Default: 0.
- `text` (String): The text content of the text element. Default: ''.
- `textAlign` (String): The text alignment of the text element. Possible values are 'left', 'center', and 'right'. Default: 'left'.
- `fontSize` (Number): The font size of the text element. Default: 12.
- `fontFamily` (String): The font family of the text element. Default: 'Source Sans Pro'.
- `textcolor` (String): The text color of the text element. Default: 'hsla(190, 100%, 0%, 1)'.

Notes

- The `x` and `y` coordinates are specified in graph coordinates, not pixel coordinates.
- The text element is added as an SVG element to the graph, allowing it to scale with the graph dimensions.

Function: viewX.addRectangle

Description

This function adds a new rectangle element to the specified graph with the given name and rectangle options. The rectangle element can be customized using various options provided in the `rectoptions` parameter.

Syntax

```
 viewX.addRectangle(graphname, rectname, rectoptions);
```

Parameters

- `graphname` (String): The name of the graph to which the rectangle element will be added.
- `rectname` (String): The unique name assigned to the rectangle element. It is used as the identifier for the rectangle element in the library.
- `rectoptions` (Object): An object containing the rectangle options and various customization options for the rectangle element appearance and behavior.

Return Value

This function returns an array containing the rectangle element and the rectangle options (modified with the applied customizations).

Example

```
 var graphName = "myGraph";
var rectName = "myRectangle";
var rectOptions = {
  x: 2,
  y: 2,
  w: 4,
  h: 3,
  stroke: "hsla(190, 100%, 50%, 0.5)",
  strokewidth: 0.1,
  rectcolor: "hsla(190, 100%, 50%, 1)"
};
viewX.addRectangle(graphName, rectName, rectOptions);
```

Rectangle Options

The `rectoptions` object can contain various options to customize the appearance and behavior of the rectangle element. The following options are available:

- `x` (Number): The x-coordinate of the top-left corner of the rectangle. Default: 0.
- `y` (Number): The y-coordinate of the top-left corner of the rectangle. Default: 0.
- `w` (Number): The width of the rectangle. Default: 1.
- `h` (Number): The height of the rectangle. Default: 1.
- `stroke` (String): The color of the rectangle's stroke. Default: 'hsla(190, 100%, 50%, 0.5)'.
- `strokewidth` (Number): The width of the rectangle's stroke. Default: 0.1.
- `strokedasharray` (String): The dash array pattern for the rectangle's stroke. Default: "".
- `rectcolor` (String): The fill color of the rectangle. Default: 'hsla(190, 100%, 50%, 1)'.

Notes

- The `x`, `y`, `w`, and `h` values are specified in graph coordinates, not pixel coordinates.
- The rectangle element is added as an SVG element to the graph, allowing it to scale with the graph dimensions.

Function: viewX.addPoint

Description

This function adds a new point to the specified graph with the given point name and point options. The point can be customized using various options provided in the `pointoptions` parameter.

Syntax

```
 viewX.addPoint(graphname, pointname, pointoptions);
```

Parameters

- `graphname` (String): The name of the graph in which the point will be added.
- `pointname` (String): The unique name assigned to the point. It is used as the identifier for the point in the library.
- `pointoptions` (Object): An object containing various customization options for the point appearance and behavior.

Return Value

This function returns an array containing the point element (SVG Ellipse) and the point options (modified with the applied customizations).

Example

```
 var graphName = "myGraph";
var pointName = "myPoint";
var pointOptions = {
  x: 2,
  y: 3,
  pointsize: 0.7,
  pointcolor: 'hsla(190, 100%, 50%, 1)',
  draggability: 'yes'
};
viewX.addPoint(graphName, pointName, pointOptions);
```

Point Options

The `pointoptions` object can contain various options to customize the appearance and behavior of the point. The following options are available:

- `x` (Number): The x-coordinate of the point. Default: 0.3.
- `y` (Number): The y-coordinate of the point. Default: 0.3.
- `pointsize` (Number): The size of the point. Default: 0.7.
- `pointcolor` (String): The color of the point. Default: 'hsla(190, 100%, 50%, 1)'.
- `draggability` (String): Whether the point can be dragged ('yes' or 'no'). Default: 'no'.
- `currentlyDraggable` (String): Whether the point is currently draggable ('yes' or 'no'). Default depends on the `draggability` option.
- `runFunctionOnDragEnd` (String): The function to run when the point drag ends. Default: ''.
- `runFunctionDuringDrag` (String): The function to run during the point drag. Default: ''.
- `dragDirection` (String): The direction in which the point can be dragged ('bothXY', 'xOnly', or 'yOnly'). Default: 'bothXY'.
- `dragIfCondition` (String): The condition for allowing the point to be dragged. Default: 'true'.

Notes

- The `draggability` option can be set to 'yes' to enable dragging for the point. The `currentlyDraggable` option determines whether the point is draggable at the moment.
- The `dragDirection` option can be set to 'xOnly' or 'yOnly' to restrict the dragging to only one axis.
- The `dragIfCondition` option can be set to a custom condition to allow dragging only when the condition is met.

Function: viewX.makeArc

Description

This function creates an arc with the specified parameters and adds it to the graph with the given `ringname`. The function returns an array of points representing the arc.

Syntax

```
 viewX.makeArc(arcradius, arcthickness, arccolor, startanglepercent, endanglepercent, ringname);
```

Parameters

- `arcradius` (Number): The radius of the arc.
- `arcthickness` (Number): The thickness of the arc.
- `arccolor` (String): The color of the arc (e.g., `"red"` or `"#FF0000"`).
- `startanglepercent` (Number): The starting angle of the arc as a percentage of 360 degrees (e.g., `0.25` for 90 degrees).
- `endanglepercent` (Number): The ending angle of the arc as a percentage of 360 degrees (e.g., `0.5` for 180 degrees).
- `ringname` (String): The name of the graph to which the arc will be added.

Return Value

This function returns an array of points representing the arc.

Example

```
 var arcRadius = 5;
var arcThickness = 2;
var arcColor = "red";
var startAnglePercent = 0;
var endAnglePercent = 0.5;
var ringName = "myGraph";
viewX.makeArc(arcRadius, arcThickness, arcColor, startAnglePercent, endAnglePercent, ringName);
```

Function: viewX.updateLine

Description

This function updates the specified line in the specified graph with the new line values provided in the `linevalues` parameter.

Syntax

```
 viewX.updateLine(graphname, linename, linevalues);
```

Parameters

- `graphname` (String): The name of the graph containing the line to be updated.
- `linename` (String): The name of the line to be updated.
- `linevalues` (Object): An object containing the new line values to be applied.

Return Value

This function does not return a value.

Example

```
 var graphName = "myGraph";
var lineName = "myLine";
var updatedLineValues = {
  x1: 1,
  y1: 1,
  x2: 6,
  y2: 6,
  strokedasharray: "10,10",
  strokewidth: 3,
  linecolor: "hsla(200, 100%, 50%, 1)"
};
viewX.updateLine(graphName, lineName, updatedLineValues);
```

Updated Line Values

The `linevalues` object can contain new values for the line to be updated. The following options are available:

- `x1` (Number): The new x-coordinate of the starting point of the line.
- `y1` (Number): The new y-coordinate of the starting point of the line.
- `x2` (Number): The new x-coordinate of the ending point of the line.
- `y2` (Number): The new y-coordinate of the ending point of the line.
- `strokedasharray` (String): A string specifying the new stroke-dasharray attribute for the line.
- `strokewidth` (Number): The new width of the line stroke.
- `linecolor` (String): The new color of the line in CSS color format.

Function: viewX.updatePath

Description

This function updates an existing path in the specified graph with new path options. The function modifies the path element and path options in the graph data.

Syntax

```
 viewX.updatePath(graphname, pathname, newpathoptions);
```

Parameters

- `graphname` (String): The name of the graph containing the path to be updated.
- `pathname` (String): The name of the path to be updated.
- `newpathoptions` (Object): An object containing the new path options to be applied to the path.

Example

```
 var graphName = "myGraph";
var pathName = "myPath";
var newPathOptions = {
  points: [[0, 1], [1, 0], [2, 2]],
  strokewidth: 1,
  pathcolor: "hsla(190, 100%, 50%, 1)",
  pathfillcolor: "none"
};
viewX.updatePath(graphName, pathName, newPathOptions);
```

Notes

- The `newpathoptions` object can contain any of the path options available in the `viewX.addPath` function.

Function: viewX.updatePathPoints

Description

This function updates the points of an existing path in the specified graph. The function modifies the path element and path options in the graph data.

Syntax

```
 viewX.updatePathPoints(graphname, pathname, npathpoints);
```

Parameters

- `graphname` (String): The name of the graph containing the path to be updated.
- `pathname` (String): The name of the path to be updated.
- `npathpoints` (Array of Arrays): An array of coordinate pairs representing the new points

Function: viewX.updateCircle

Description

This function updates the appearance and position of the circle with the specified name in the specified graph. The circle appearance can be customized using various options provided in the `circlenewvalues` parameter.

Syntax

```
viewX.updateCircle(graphname, circlename, circlenewvalues);
```

Parameters

- `graphname` (String): The name of the graph to which the circle belongs.
- `circlename` (String): The unique name assigned to the circle. It is used as the identifier for the circle in the library.
- `circlenewvalues` (Object): An object containing the new values for the circle options.

Example

```
var graphName = "myGraph";
var circleName = "myCircle";
var circleNewValues = {
  x: 4,
  y: 5,
  radius: 2,
  stroke: 'hsla(190, 100%, 50%, 0.7)',
  strokewidth: 0.15,
  circlecolor: 'hsla(190, 100%, 50%, 0.8)'
};
viewX.updateCircle(graphName, circleName, circleNewValues);
```

Function: viewX.updateEllipse

Description

This function updates the appearance and position of the ellipse with the specified name in the specified graph. The ellipse appearance can be customized using various options provided in the `ellipsenewvalues` parameter.

Syntax

```
viewX.updateEllipse(graphname, ellipsename, ellipsenewvalues);
```

Parameters

- `graphname` (String): The name of the graph to which the ellipse belongs.
- `ellipsename` (String): The unique name assigned to the ellipse. It is used as the identifier for the ellipse in the library.
- `ellipsenewvalues` (Object): An object containing the new values for the ellipse options.

Example

```
var graphName = "myGraph";
var ellipseName = "myEllipse";
var ellipseNewValues = {
  x: 4,
  y: 5,
  rx: 2,
  ry: 3,
  stroke: 'hsla(190, 100%, 50%, 0.7)',
  strokewidth: 0.15,
  ellipsecolor: 'hsla(190, 100%, 50%, 0.8)'
};
viewX.updateEllipse(graphName, ellipseName, ellipseNewValues);
```

Function: viewX.updateText

Description

This function updates an existing text element on the specified graph with the given name and text values. The text element can be updated using various options provided in the `textvalues` parameter.

Syntax

```
viewX.updateText(graphname, textname, textvalues);
```

Parameters

- `graphname` (String): The name of the graph containing the text element to be updated.
- `textname` (String): The unique name assigned to the text element. It is used as the identifier for the text element in the library.
- `textvalues` (Object): An object containing the updated text values and various customization options for the text element appearance and behavior.

Return Value

This function returns an array containing the updated text element and the text options (modified with the applied customizations).

Example

```
var graphName = "myGraph";
var textName = "myText";
var newTextValues = {
  x: 7,
  y: 7,
  text: "Updated text!",
  textcolor: "hsla(190, 100%, 50%, 1)"
};
viewX.updateText(graphName, textName, newTextValues);
```

Function: viewX.updateRectangle

Description

This function updates an existing rectangle element on the specified graph with the given name and rectangle value updates. The rectangle element can be updated using various options provided in the `rectvalueupdate` parameter.

Syntax

```
viewX.updateRectangle(graphname, rectname, rectvalueupdate);
```

Parameters

- `graphname` (String): The name of the graph containing the rectangle element to be updated.
- `rectname` (String): The unique name assigned to the rectangle element. It is used as the identifier for the rectangle element in the library.
- `rectvalueupdate` (Object): An object containing the updated rectangle values and various customization options for the rectangle element appearance and behavior.

Return Value

This function returns an array containing the updated rectangle element and the rectangle options (modified with the applied customizations).

Example

```
 var graphName = "myGraph";
var rectName = "myRectangle";
var newRectValues = {
  x: 3,
  y: 3,
  w: 5,
  h: 4,
  stroke: "hsla(190, 100%, 70%, 0.5)",
  strokewidth: 0.2,
  rectcolor: "hsla(190, 100%, 70%, 1)"
};
viewX.updateRectangle(graphName, rectName, newRectValues);
```

Notes

- The `x`, `y`, `w`, and `h` values can be updated independently, allowing for flexible customization of the rectangle element.
- The rectangle element is updated as an SVG element on the graph, allowing it to scale with the graph dimensions.

Function: viewX.updatePoint

Description

This function updates the specified point in the graph with new point options.

Syntax

```
viewX.updatePoint(graphname, pointname, newpointoptions);
```

Parameters

- `graphname` (String): The name of the graph containing the point.
- `pointname` (String): The name of the point to be updated.
- `newpointoptions` (Object): An object containing the new point options to be applied.

Example

```
 var graphName = "myGraph";
var pointName = "myPoint";
var newPointOptions = {
  x: 5,
  y: 7,
  pointsize: 1,
  pointcolor: 'hsla(250, 100%, 50%, 1)',
  draggability: 'no'
};
viewX.updatePoint(graphName, pointName, newPointOptions);
```

Notes

- The `newpointoptions` object can contain any of the point options available in the `pointoptions` object of the `viewX.addPoint` function. Any options not provided will maintain their previous values.

Function: viewX.updatePointXY

Description

This function updates the x and y coordinates of the specified point in the graph.

Syntax

```
viewX.updatePointXY(graphname, pointname, xvalue, yvalue);
```

Parameters

- `graphname` (String): The name of the graph containing the point.
- `pointname` (String): The name of the point to be updated.
- `xvalue` (Number): The new x-coordinate of the point.
- `yvalue` (Number): The new y-coordinate of the point.

Example

```
 var graphName = "myGraph";
var pointName = "myPoint";
var newX = 8;
var newY = 4;
viewX.updatePointXY(graphName, pointName, newX, newY);
```

Function: viewX.removePoint

Description

This function removes the specified point from the graph.

Syntax

```
viewX.removePoint(graphname, pointname);
```

Parameters

- `graphname` (String): The name of the graph containing the point.
- `pointname` (String): The name of the point to be removed.

Example

```
 var graphName = "myGraph";
var pointName = "myPoint";
viewX.removePoint(graphName, pointName);
```

Notes

- After removing a point using this function, it cannot be restored. To add a new point, use the `viewX.addPoint` function.

Function: viewX.removeLine

Description

This function removes a line from the specified graph by its name.

Syntax

```
viewX.removeLine(graphname, linename);
```

Parameters

- `graphname` (String): The name of the graph from which the line will be removed.
- `linename` (String): The unique name of the line to remove from the graph.

Example

```
viewX.removeLine("myGraph", "myLine");
```

Function: viewX.removeCircle

Description

This function removes a circle from the specified graph by its name.

Syntax

```
viewX.removeCircle(graphname, circlename);
```

Parameters

- `graphname` (String): The name of the graph from which the circle will be removed.
- `circlename` (String): The unique name of the circle to remove from the graph.

Example

```
viewX.removeCircle("myGraph", "myCircle");
```

Function: viewX.removeText

Description

This function removes a text element from the specified graph by its name.

Syntax

```
viewX.removeText(graphname, textname);
```

Parameters

- `graphname` (String): The name of the graph from which the text element will be removed.
- `textname` (String): The unique name of the text element to remove from the graph.

Example

```
viewX.removeText("myGraph", "myText");
```

Function: viewX.removePath

Description

This function removes a path from the specified graph by its name.

Syntax

```
viewX.removePath(graphname, pathname);
```

Parameters

- `graphname` (String): The name of the graph from which the path will be removed.
- `pathname` (String): The unique name of the path to remove from the graph.

Example

```
viewX.removePath("myGraph", "myPath");
```

Function: viewX.removeArrow

Description

This function removes an arrow from the specified graph by its name.

Syntax

```
viewX.removeArrow(graphname, arrowname);
```

Parameters

- `graphname` (String): The name of the graph from which the arrow will be removed.
- `arrowname` (String): The unique name of the arrow to remove from the graph.

Example

```
viewX.removeArrow("myGraph", "myArrow");
```

Notes

- The specified element (line, circle, text, path, or arrow) will be removed from the graph by its unique name.
- Removing an element that does not exist in the graph will not throw any errors, but it will have no effect.

Function: viewX.removeGraph

Description

This function removes a graph with the specified `graphname` from the DOM and deletes its data from the `viewX.graphData` object.

Syntax

```
viewX.removeGraph(graphname);
```

Parameters

- `graphname` (String): The unique name assigned to the graph that you want to remove.

Example

```
viewX.removeGraph("myGraph");
```

Function: viewX.deleteSegments

Description

This function deletes a collection of SVG segments by setting their `outerHTML` to an empty string.

Syntax

```
viewX.deleteSegments(collection);
```

Parameters

- `collection` (Array): An array of SVG elements to be deleted.

Example

```
 var segments = document.querySelectorAll(".segments-to-delete");
viewX.deleteSegments(segments);
```

Function: viewX.randomChoice

Description

This function returns a random element from the given `choicearray`.

Syntax

```
viewX.randomChoice(choicearray);
```

Parameters

- `choicearray` (Array): An array of elements from which a random choice will be made.

Return Value

This function returns a random element from the `choicearray`.

Example

```
 var choices = ["apple", "banana", "cherry"];
var randomFruit = viewX.randomChoice(choices);
console.log(randomFruit); // Output: "apple" (or "banana" or "cherry")
```

Function: viewX.randomWeightedChoice

Description

This function returns a random element from the given `choicearray` based on the weights provided in the `weightArray`.

Syntax

```
viewX.randomWeightedChoice(choicearray, weightArray);
```

Parameters

- `choicearray` (Array): An array of elements from which a random choice will be made.
- `weightArray` (Array): An array of weights corresponding to the elements in the `choicearray`.

Return Value

This function returns a random element from the `choicearray` based on the weights provided in the `weightArray`.

Example

```
 var choices = ["apple", "banana", "cherry"];
var weights = [1, 2, 1];
var randomFruit = viewX.randomWeightedChoice(choices, weights);
console.log(randomFruit); // Output: "banana" (or "apple" or "cherry" with the specified weights)
```

Function: viewX.linearValue

Description

This function calculates the linear interpolation between two points `(xv1, yv1)` and `(xv2, yv2)` for a given input value `inputvl`.

Syntax

```
viewX.linearValue(xv1, xv2, yv1, yv2, inputvl);
```

Parameters

- `xv1` (Number): The x-coordinate of the first point.
- `xv2` (Number): The x-coordinate of the second point.
- `yv1` (Number): The y-coordinate of the first point.
- `yv2` (Number): The y-coordinate of the second point.
- `inputvl` (Number): The input value for which the interpolated value should be calculated.

Return Value

This function returns the interpolated value for the given input value `inputvl`.

Example

```
 var result = viewX.linearValue(0, 10, 0, 100, 5);
console.log(result); // Output: 50
```

Function: viewX.setFont

Description

This function sets the font size of a collection of HTML elements with the specified `fontval`.

Syntax

```
viewX.setFont(divCollection, fontval);
```

Parameters

- `divCollection` (Array): An array of element IDs for which the font size should be set.
- `fontval` (String): The font size value to be set (e.g., `"16px"` or `"1.5em"`).

Example

```
 var elements = ["element1", "element2", "element3"];
var fontSize = "18px";
viewX.setFont(elements, fontSize);
```

Function: viewX.distF

Description

This function calculates the Euclidean distance between two points `pt1` and `pt2`.

Syntax

```
viewX.distF(pt1, pt2);
```

Parameters

- `pt1` (Array): An array of two numbers representing the x and y coordinates of the first point.
- `pt2` (Array): An array of two numbers representing the x and y coordinates of the second point.

Return Value

This function returns the Euclidean distance between the two points.

Example

```
 var point1 = [0, 0];
var point2 = [3, 4];
var distance = viewX.distF(point1, point2);
console.log(distance); // Output: 5
```

Function: viewX.addVec

Description

This function calculates the sum of two vectors `pt1` and `pt2`.

Syntax

```
viewX.addVec(pt1, pt2);
```

Parameters

- `pt1` (Array): An array of two numbers representing the x and y components of the first vector.
- `pt2` (Array): An array of two numbers representing the x and y components of the second vector.

Return Value

This function returns an array representing the sum of the two input vectors.

Example

```
 var vector1 = [1, 2];
var vector2 = [3, 4];
var sum = viewX.addVec(vector1, vector2);
console.log(sum); // Output: [4, 6]
```

Function: viewX.directionVec

Description

This function calculates the unit direction vector from point `pt1` to point `pt2`.

Syntax

```
viewX.directionVec(pt1, pt2);
```

Parameters

- `pt1` (Array): An array of two numbers representing the x and y coordinates of the first point.
- `pt2` (Array): An array of two numbers representing the x and y coordinates of the second point.

Return Value

This function returns an array representing the unit direction vector from `pt1` to `pt2`.

Example

```
 var point1 = [0, 0];
var point2 = [3, 4];
var direction = viewX.directionVec(point1, point2);
console.log(direction); // Output: [0.6, 0.8]
```

Function: viewX.rotatedVec

Description

This function calculates the vector obtained by rotating the input vector `ofVector` by the specified angle (in degrees).

Syntax

```
viewX.rotatedVec(ofVector, angle);
```

Parameters

- `ofVector` (Array): An array of two numbers representing the x and y components of the input vector.
- `angle` (Number): The angle in degrees by which the vector should be rotated.

Return Value

This function returns an array representing the rotated vector.

Example

```
 var vector = [1, 0];
var angle = 90;
var rotated = viewX.rotatedVec(vector, angle);
console.log(rotated); // Output: [0, 1]
```

Function: viewX.mod

Description

This function calculates the magnitude (or modulus) of the input vector `ofVector`.

Syntax

```
viewX.mod(ofVector);
```

Parameters

- `ofVector` (Array): An array of two numbers representing the x and y components of the input vector.

Return Value

This function returns the magnitude of the input vector.

Example

```
 var vector = [3, 4];
var magnitude = viewX.mod(vector);
console.log(magnitude); // Output: 5
```

Function: viewX.shuffle

Description

This function shuffles the elements of an input array `array` using the Fisher-Yates algorithm.

Syntax

```
viewX.shuffle(array);
```

Parameters

- `array` (Array): An array of elements to be shuffled.

Return Value

This function returns the shuffled array.

Example

```
 var numbers = [1, 2, 3, 4, 5];
var shuffled = viewX.shuffle(numbers);
console.log(shuffled); // Output: [3, 1, 5, 4, 2] (or any other random permutation)
```