

Simplified JSXGraph

JSXGraph with VSCode scaffolding



Tom Berend

board.create() constructs 120+ elements

Many of these also have overloads

```
-- '''
11
12 var f1 = board.create('glider', [-2, 0, board.defaultAxes.x], {name:"f"});
13 var f2 = board.create('glider', [2, 0, board.defaultAxes.x], {name:"f"});
14 var ell = board.create('ellipse', [f1, f2, [0,3]]);
15
16 var P = board.create('glider', [-1, 2, ell], {name: 'p'});
17 var s1 = board.create('segment', [f1,P]);
18 var s2 = board.create('segment', [f2,P]);
19
20 var txt = board.create('text', [0.2, 4,
21     () => "|pf| + |pf| = " + P.Dist(f1).toFixed(2) + ' + ' + P.Dist(f2).toFixe
22 ];
```

Workflow Requires Editor, Console, & API

How we did things 20 years ago...

```
<script>
let board = JSXGraph.initBoard('jxgbox',{boundingBox:[-5,5,5,-5]});

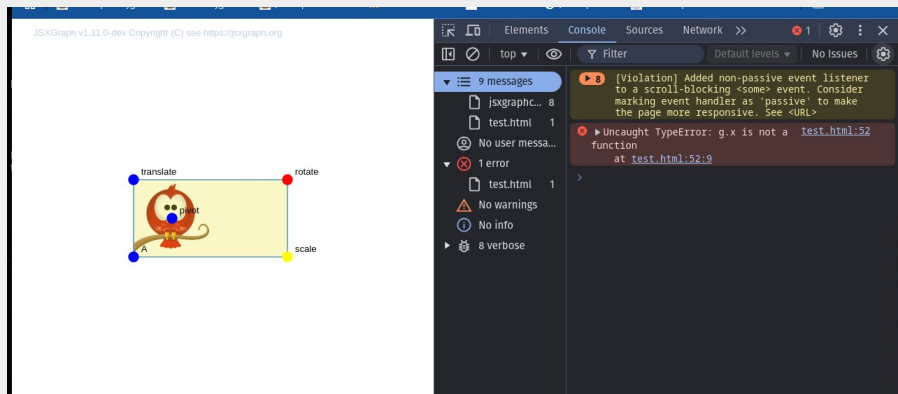
// Add an image and use the group tools to manipulate it
let urlImg = "https://jsxgraph.org/distrib/images/uccellino.jpg";

let col = 'blue';
let p = [];
p.push(board.create('point', [-2, -1], { size: 5, strokeColor: col, fillColor: col }));
p.push(board.create('point', [2, -1], { size: 5, strokeColor: 'yellow', fillColor: 'yellow', name: 'translate' }));
p.push(board.create('point', [2, 1], { size: 5, strokeColor: 'red', fillColor: 'red', name: 'rotate' }));
p.push(board.create('point', [-2, 1], { size: 5, strokeColor: col, fillColor: col, name: 'scale' }));

let pivot = board.create('point', [-1, 0], {size: 5, strokeColor:col, fillColor:col,name: 'pivot'});

let im = board.create('image', [urlImg, [-2, -1], [2, 2]]);
let pol = board.create('polygon', p, { hasInnerPoints: true });

let g = board.create('group', p.concat(im).concat(pivot));
```



JSXGraph 1.10.1 Reference

[Class Index](#) | [File Index](#)

▼ Elements

[Angle](#)
[Arc](#)
[Arrow](#)
[Arrowparallel](#)
[Axes3D](#)
[Axis](#)
[Axis3D](#)
[Bisector](#)
[Bisectorlines](#)
[Boxplot](#)
[Button](#)
[Cardinalspline](#)
[Chart](#)
[Checkbox](#)
[Circle](#)
[Circle3D](#)
[Circumcenter](#)
[Circumcircle](#)
[CircumcircleArc](#)
[CircumcircleSector](#)
[Comb](#)
[Conic](#)
[Curve](#)
[Curve3D](#)
[CurveDifference](#)
[CurveIntersection](#)
[CurveUnion](#)
[Derivative](#)
[Ellipse](#)
[ForeignObject](#)
[Functiongraph](#)
[Functiongraph3D](#)
[Glider](#)
[Grid](#)
[Group](#)
[Hatch](#)
[Hyperbola](#)
[Image](#)

Element Index

[Angle](#)

The angle element is used to denote an angle defined by three points.

[Arc](#)

An arc is a segment of the circumference of a circle.

[Arrow](#)

This element is used to provide a constructor for arrow, which is just a wrapper for element [Line](#) with [Line#straightFirst](#) and [Line#straightLast](#) properties set to false and [Line#lastArrow](#) set to true.

[Arrowparallel](#)

An arrow parallel is a segment with an arrow attached which is parallel through a given segment, given by its defining two points, through a given point.

[Axes3D](#)

This element creates the axis and plane elements of a 3D view.

[Axis](#)

This element is used to provide a constructor for an axis.

[Axis3D](#)

This element creates a 3D axis.

[Bisector](#)

A bisector is a line which divides an angle into two equal angles.

[Bisectorlines](#)

Bisector lines are similar to [Bisector](#) but take two lines as parent elements.

[Boxplot](#)

Box plot curve.

[Button](#)

This element is used to provide a constructor for special texts containing a form button element.

[Cardinalspline](#)

This element is used to provide a constructor for cardinal spline curves.

'Minimal example' is terrifying

But I just want...



Docs

Dynamic Mathematics with JavaScript

Minimal example

- Load JSXGraph from <https://jsdelivr.com>
- Optionally, include MathJax

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>JSXGraph template</title>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <link href="https://cdn.jsdelivr.net/npm/jsxgraph/distrib/jsxgraph.css"
rel="stylesheet" type="text/css" />
    <script src="https://cdn.jsdelivr.net/npm/jsxgraph/distrib/jsxgraphcor
e.js" type="text/javascript" charset="UTF-8"></script>

    <!-- The next line is optional: load MathJax -->
    <script src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-cthtml.js" i
d="MathJax-script" async></script>
  </head>
  <body>

<div id="jxgbox" class="jxgbox" style="width:500px; height:200px;"></div>

<script>
  var board = JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 2, 5, -2],
    axis:true
  });
  var fun = board.create('functiongraph', ['sin(x)']);
</script>
```

Requires webserver setup.

Requires a programmer editor.

Requires understanding file system and terminal commands.

Requires JavaScript and HTML

Requires knowing how the internet works

Same Engine – Only Replacing **create()**

Constructors for each element

```
let TSX = new TSXBoard('html03',)

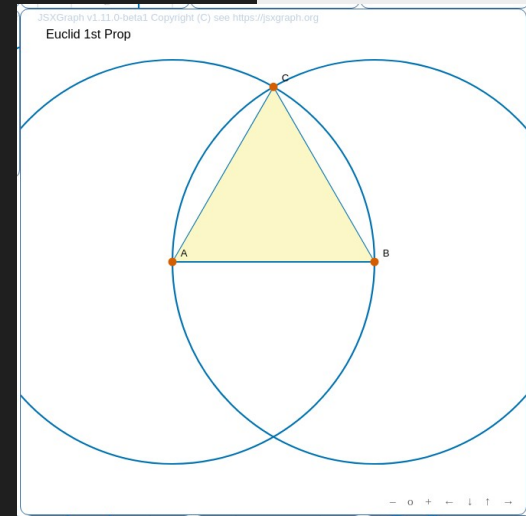
TSX.Text([-4.5, 4.5], 'Euclid 1st Prop', {fontSize: 15})

// problem - create equilateral on this segment
let a = TSX.Point([-2, 0], {name: 'A'})
let b = TSX.Point([2, 0], {name: 'B'})
TSX.Segment(a, b)

// solution
let c1 = TSX.Circle(a, b)
let c2 = TSX.Circle(b, a)

let c = TSX.Intersection(c2, c1, {name: 'C'})

TSX.Polygon([a, b, c])
```



A Thin TypeScript Wrapper over JSXGraph

Calls board.create() with types

Typed Parameters

Typed Attributes

Typed Returns

```
.....also create points with Intersection, Midpoint, TransformPoint, Circumcenter, Glider, TransformPoint
.....Point(position: pointAddr, attributes: PointAttributes = {}): Point {
.....|.....return (this._jBoard as any).create('point', position, this.defaultAttributes(attributes))
.....}
```

Modifies defaults

Keeps JSXGraph Coding Style

Works the way you expect.

Attribute is always last

```
let origin = TSX.Point([0, 0], { name: 'Origin' })
```

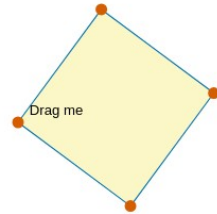
Multiple signatures

```
TSX.Angle(p1, p2, p3) // angle from 3 points
TSX.Angle(l1, l2, [5, 0], [5, 5]) // two lines, two directions
TSX.Angle(l1, l2, 1, -1, { radius: 2 }) // two lines, two +/- values
```

Fix Legacy Design - eg: Translations

Some decisions didn't age well.

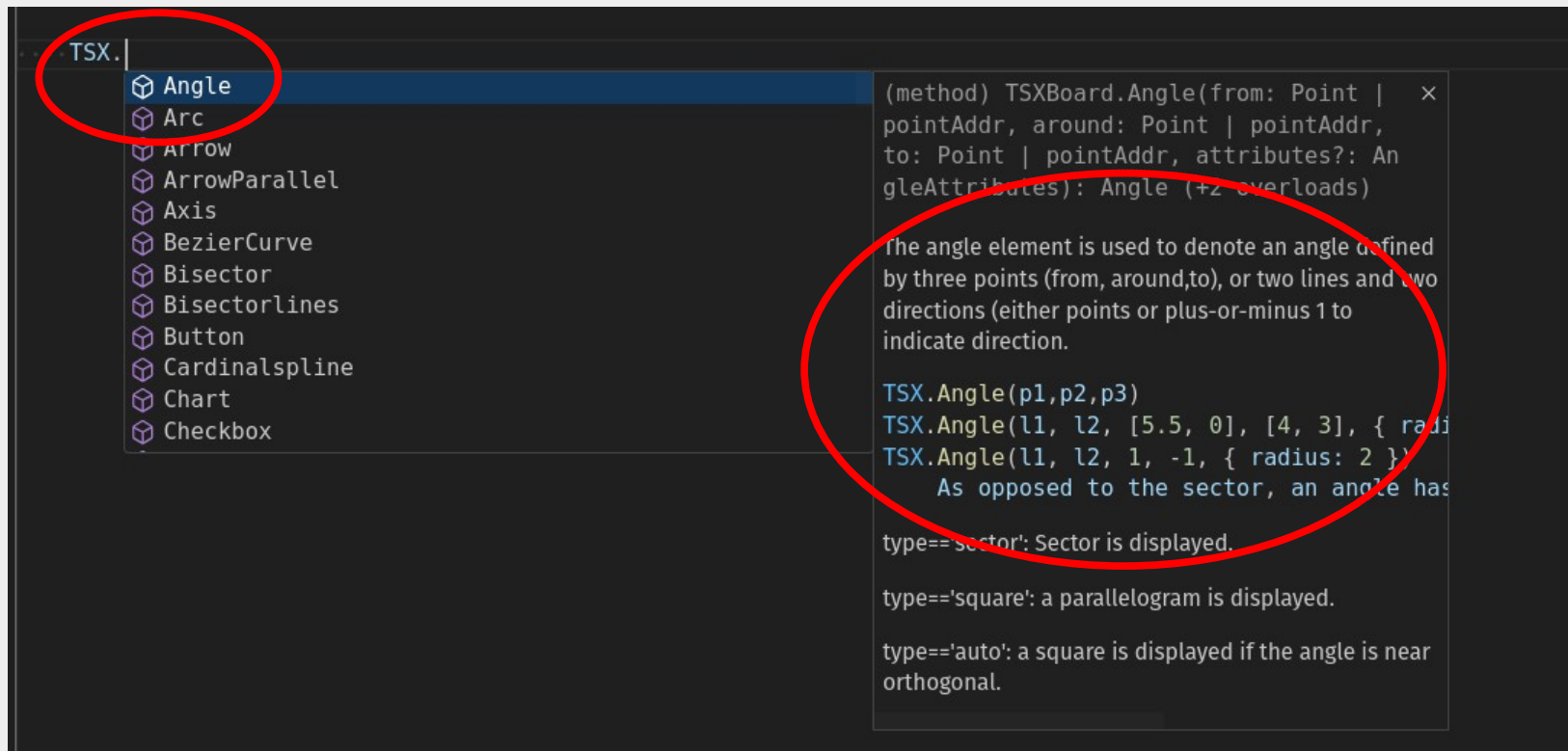
```
•//•create•a•transformation•that•rotates•around•p1,  
•let•t1•=•board.create('transform', [() => slider.Value(), p1], {type: 'rotate'})  
•let•t2•=•TSX.Rotate(() => slider.Value(), p1) ... //new
```



θ = 5.65

Completions expose JSXGraph Elements

Explore beyond the basic elements...



The image shows a dark-themed user interface for the JSXGraph library. On the left, a list of elements is displayed, each preceded by a small cube icon. The 'Angle' element is highlighted with a blue background and is also circled in red. Below the list, a detailed description of the 'Angle' element is shown, also circled in red. This description includes the method signature for `TSXBoard.Angle`, a paragraph explaining its usage with three points or two lines, and three code snippets demonstrating different ways to create an angle. The snippets show how to specify points, lines, and radius, and how to use the `type` parameter to display a sector, a parallelogram, or a square.

TSX.

- Angle
- Arc
- Arrow
- ArrowParallel
- Axis
- BezierCurve
- Bisector
- Bisectorlines
- Button
- Cardinalspline
- Chart
- Checkbox

(method) `TSXBoard.Angle`(from: Point | × pointAddr, around: Point | pointAddr, to: Point | pointAddr, attributes?: AngleAttributes): Angle (+2 overloads)

The angle element is used to denote an angle defined by three points (from, around, to), or two lines and two directions (either points or plus-or-minus 1 to indicate direction).

```
TSX.Angle(p1, p2, p3)
TSX.Angle(l1, l2, [5.5, 0], [4, 3], { radius: 2 })
TSX.Angle(l1, l2, 1, -1, { radius: 2 })
```

As opposed to the sector, an angle has

`type=='sector'`: Sector is displayed.

`type=='square'`: a parallelogram is displayed.

`type=='auto'`: a square is displayed if the angle is near orthogonal.

Hover for Context Help with Examples

Also: Hover over variables to see their type

```
(method) TSXBoard.Point(position: pointAddr, attributes?: PointAttributes): Point
```

Create a point. If any parent elements are functions or the attribute 'fixed' is true then point will be constrained.

```
TSX.Point([3,2],{strokeColor:'blue',strokeWidth:5,strokeOpacity:.5})
```

```
TSX.Point([3,3]),{fixed:true, showInfobox:true}
```

```
TSX.Point([()=>p1.X()+2,()=>p1.Y()+2]) // 2 up 2 right from p1
```

```
TSX.Point([1,2,2]) // three axis definition - [z,x,y]
```

+

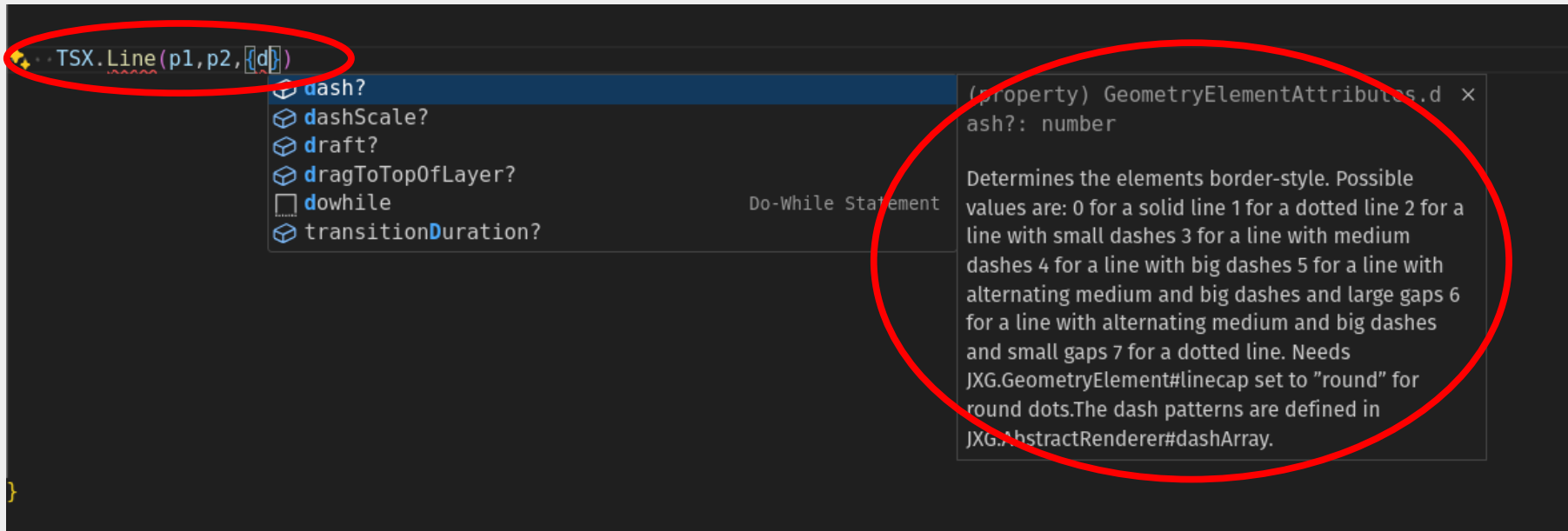
also create points with Intersection, Midpoint, TransformPoint, Circumcenter, Glider, TransformPoint, and others.

```
... let a = TSX.Point([1,0])
```

Hover cursor

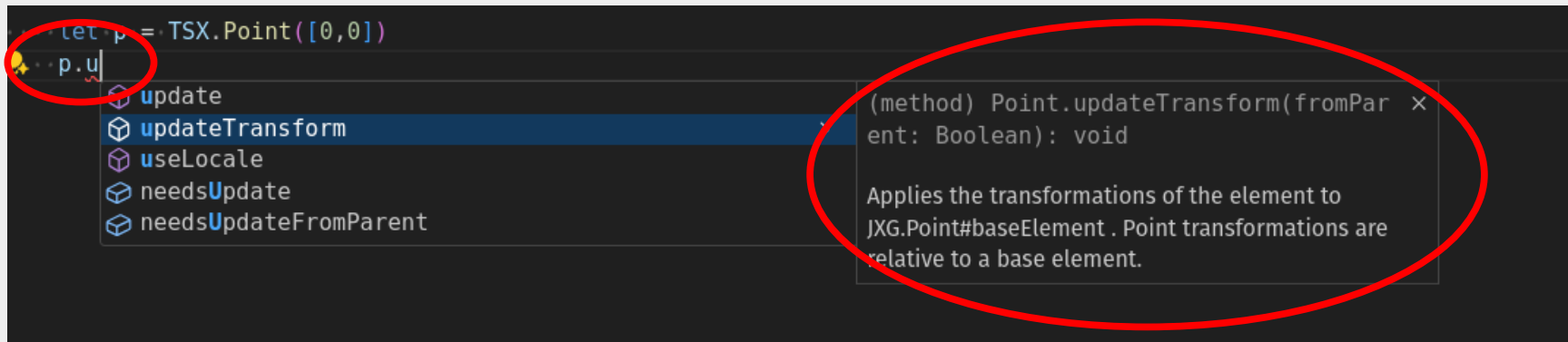
Completions expose Attributes

Was it 'dashscale' or 'dashScale'?



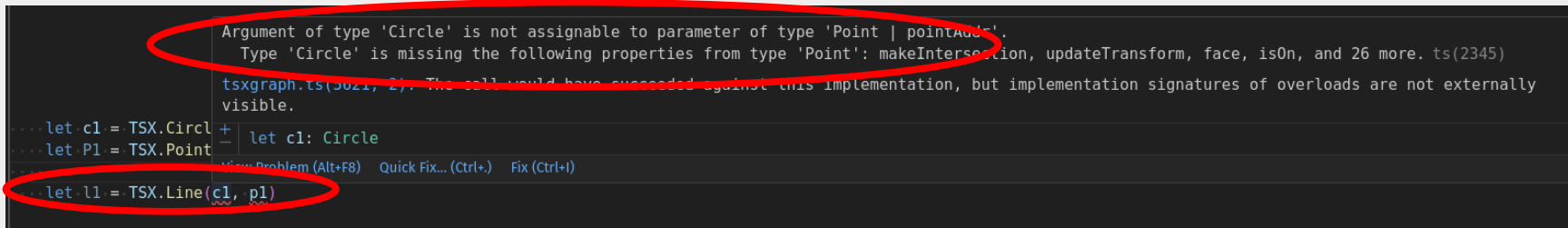
Completions expose Methods and Fields

The function you need is already there



Catch Most Errors during Edit

Almost never need console debug



Interactive syntax checking and type checking.
Also linting, refactoring, prettifying.

Don't need 'View' anymore

Don't need 'View' anymore

```

let p12D = TSX.Point([-3, 1], { name: '2D' })
let p13D = TSX.Point3D([-3, 1.5, 0], { name: '3D' })

let s12D = TSX.Segment(p12D, [3, 1], { strokeColor: 'blue', name: 'li' })
let s13D = TSX.Line3D(p13D, [3, 1.5, 0], { strokeColor: 'red', name: 'li' })

TSX.Circle(p12D, 1, { withLabel: true, name: 'Circle2D' })
TSX.Circle3D(p13D, [0, 0, 1], 1, { withLabel: true, name: 'Circle3D' })

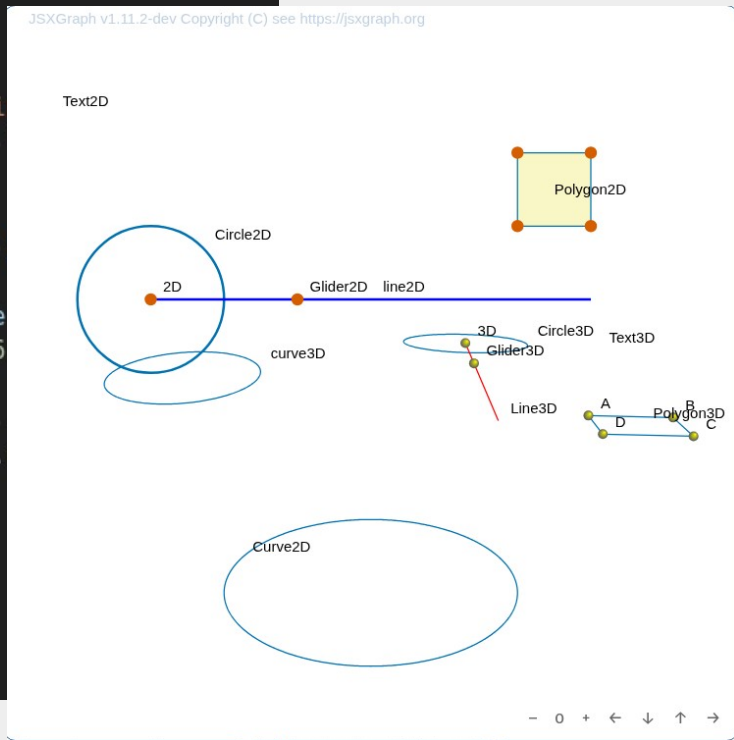
TSX.Polygon([[2, 2], [2, 3], [3, 3], [3, 2]], { withLabel: true, name: 'li' })
TSX.Polygon3D([2.6, 2.6, 0], [2.6, 3.6, 0], [3.6, 3.6, 0], [3.6, 2.6, 0], { withLabel: true, name: 'li' })

TSX.Curve((x: number) => 2 * Math.cos(x), (y: number) => Math.sin(y))
TSX.Curve3D((x: number) => 2 * Math.cos(x) + .5, (y: number) => Math.sin(y))

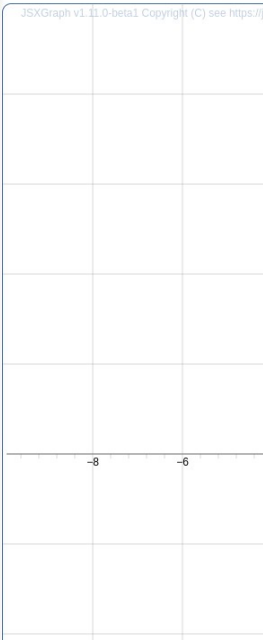
TSX.Glider(s12D, [-1, 0], { name: 'Glider2D' })
TSX.Glider3D(s13D, [-1, 0, 0], { name: 'Glider3D' })

TSX.Text([-4.2, 3.7], 'Text2D')
TSX.Text3D([-4, 4, 0], 'Text3D')

```



Stop using console.log()



Supports JavaScript and TypeScript

Same syntax check, discovery, completions...

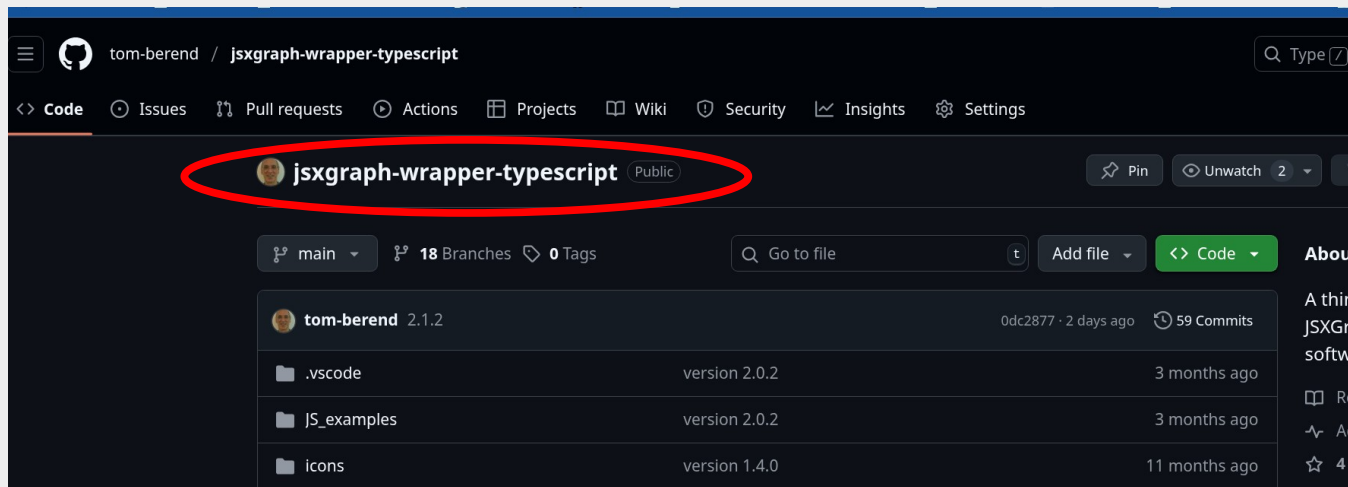
This code is identical in
JavaScript and TypeScript

```
import { TSXBoard } from '../src/tsxgraph.js';  
const TSX = new TSXBoard('jxgbox', { axis: true })  
  
let p12D = TSX.Point([-3, 1], { name: '2D' })  
let p13D = TSX.Point3D([-3, 1.5, 0], { name: '3D' })
```

TypeScript adds type safety, interfaces, and improved tooling. Better for larger projects.

Github Quick-Start includes Webserver

...ready to go in 10 seconds



```
> git clone ....
> npm i
> npm run start
```

Then browse to **<http://localhost:3000>**

Or Just Run in the Playground

...ready to go in 0 seconds

[Simplified JSXGraph](#) Playground V2.1.2

[JSXGraph](#) with VSCode scaffolding !!


Editor Keys

Run

Share

Download HTML

Mobile

Space Icons by [Good Stuff No Nonsense](#) is licensed under 
[TSX.Image](#)(then CTRL+i to list available images [preview](#)

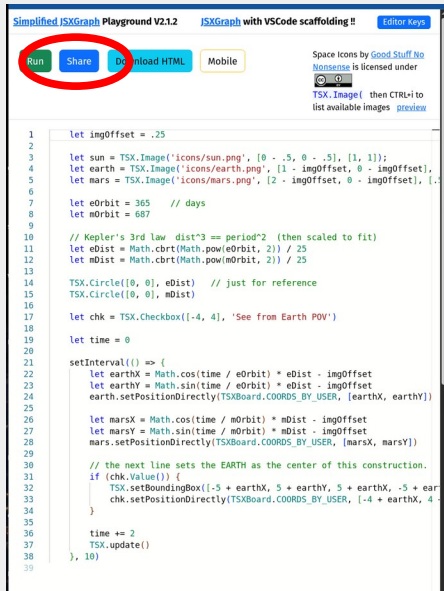
```
1 let center = TSX.Point([0,0],{name:'A'})  
2 TSX.Circle(center,2) // try running this...
```

Try it here..

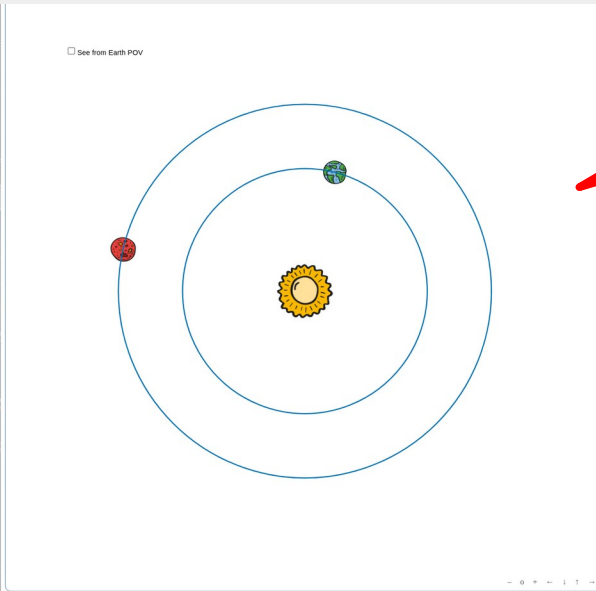
<https://cheeseandcrackers.ca/playground/>

Explore, Create, Share, Get Feedback

...lets students show off their awesomeness !



```
1 let imgOffset = .25
2
3 let sun = TSX.Image('icons/sun.png', [0 - .5, 0 - .5], [1, 1]);
4 let earth = TSX.Image('icons/earth.png', [1 - imgOffset, 0 - imgOffset], [1, 1]);
5 let mars = TSX.Image('icons/mars.png', [2 - imgOffset, 0 - imgOffset], [1, 1]);
6
7 let eOrbit = 365 // days
8 let mOrbit = 687
9
10 // Kepler's 3rd law dist^3 == period^2 (then scaled to fit)
11 let eDist = Math.cbrt(Math.pow(eOrbit, 2)) / 25
12 let mDist = Math.cbrt(Math.pow(mOrbit, 2)) / 25
13
14 TSX.Circle([0, 0], eDist) // just for reference
15 TSX.Circle([0, 0], mDist)
16
17 let chk = TSX.Checkbox([-4, 4], 'See from Earth POV')
18
19 let time = 0
20
21 setInterval(() => {
22   let earthX = Math.cos(time / eOrbit) * eDist - imgOffset
23   let earthY = Math.sin(time / eOrbit) * eDist - imgOffset
24   earth.setPositionDirectly(TSXBoard.COORDS_BY_USER, [earthX, earthY])
25
26   let marsX = Math.cos(time / mOrbit) * mDist - imgOffset
27   let marsY = Math.sin(time / mOrbit) * mDist - imgOffset
28   mars.setPositionDirectly(TSXBoard.COORDS_BY_USER, [marsX, marsY])
29
30   // the next line sets the EARTH as the center of this construction.
31   if (chk.Value()) {
32     TSX.setBoundingBox([-5 + earthX, 5 + earthY, 5 + earthX, -5 + earthY])
33     chk.setPositionDirectly(TSXBoard.COORDS_BY_USER, [-4 + earthX, 4 + earthY])
34   }
35
36   time += 2
37   TSX.update()
38 }, 10)
```



Playful Space Icons

Try it here..

<https://cheeseandcrackers.ca/playground/?script=TPHOG9C07>


Simplified JSXGraph Interactive Textbook

Assumes basic JavaScript


DevSite Home Dashboard My courses Site administration

☰

Points, Lines, and Circles

 The point is the first principle of Geometry.

—
Leonardo da Vinci

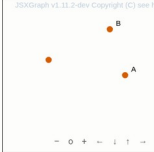
 This chapter will outline some simple elements for creating JSXGraph constructions.


Creating Points

Almost every **construction** in JSXGraph starts with points.

JSXGraph uses Cartesian coordinates to position on the **Board**. We describe the point location with array notation, so we might describe a point as being at [3,2].

We always draw points on a JSXGraph **Board** object. In this textbook, we always name the board **TSX**.




 To draw a point, we call the board method **TSX.Point()** which returns a **Point** object. This first call to **TSX.Point()** has one parameter - the coordinate array [3,2].

```
let p = TSX.Point([3, 2])
```

TSX.Point() has an optional second parameter that lets you set the point's attributes, which are usually decorative.

```
let p = TSX.Point([3, 2], {name: 'A'})
```

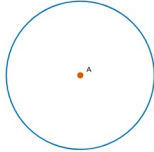
 The code below creates three points in slightly different ways.

The **Point** object is created whether you assign it to a variable or not. You only need the variable if you intend to refer to it later.

```
01 TSX.Point([-2,1]) // no label
02
03 let A = TSX.Point([3,0], {name: 'A'})
04
05 let B = [2, 3]
06 TSX.Point(B, {name: 'B'})
```

```
1 let center = TSX.Point([0,0],{name:'A'}) // try running this...
2 TSX.Circle(center,2)
```

run



Try it here..

<https://cheeseandcrackers.ca>

Simplified JSXGraph Interactive Textbook

Assumes basic JavaScript

DevSite

Home

Dashboard

My courses

Site administration

x

⋮

General

Simplified JSXGraph

Simplified JSXGraph

| Topic | Activity | Step | Description |
|---------------------|----------|----------------------------|--|
| Simplified JSXGraph | | | JSXGraph constructions are composed of basic elements like points and lines. We add relations to the construction, and we add events if we want additional actions to be triggered. |
| <div>open</div> | | Introduction | Overview of JSXGraph, and introduction to the features of this interactive textbook. |
| <div>open</div> | | Points and Lines | Building our first constructions with Points and Lines. |
| <div>open</div> | | Animating | Animating your construction. |
| <div>open</div> | | Interactive Controls | Controls and inputs make your construction interactive. |
| <div>open</div> | | Polygons and Curves | A surprisingly rich set of tools for building constructions. |
| <div>open</div> | | Transformations | Using linear algebra to animate complex models. |
| <div>open</div> | | JSXGraph in 3D | Explore 3D drawing with JSXGraph |
| <div>open</div> | | Setting Up the Environment | Moving beyond this textbook environment. |

Try it here..

<https://cheeseandcrackers.ca>

Try Simplified JSXGraph

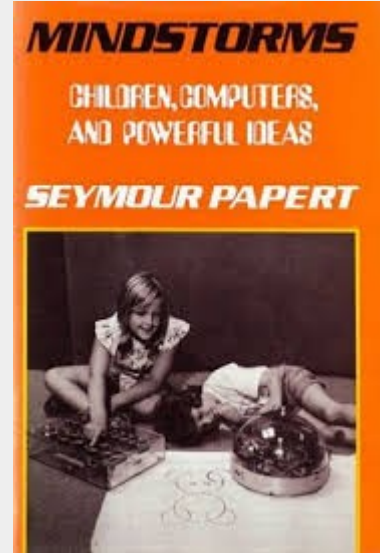
for your next project

- Delivers modern tooling
- 100% compatible – wraps `board.create()`
- 100% coverage – attributes & methods from documentation
- Works with JavaScript or TypeScript

JSXGraph = “LOGO for High-School”

Why I wrote this package

- ‘Math-like’ – leads students to math ideas
- Basic Elements controlled by Functions
- Simple 2D graphics – ideal for writing games
- ‘Real Programming’ – not Scratch
- Interactive Geometry – unique capabilities
- GUI elements included



Missing: “Low Floor – High Ceiling”

Simplified JSXGraph

JSXGraph with VSCode scaffolding



Tom Berend