Before D3
HTML, CSS, Javascript
Data Visualization

What is D3

- D3: Data-Driven Documents
  - D3 is a **JavaScript** library for visualizing data with **HTML**, **SVG**, and **CSS**.
- D3 website: https://d3js.org/
- We will use D3.v5 in this semester
  - Although the latest version of D3 is v6

Why D3 for Data Visualization?

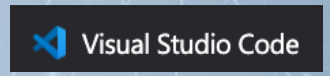
- Easy to load you own data, transform them to shapes and interact with them
- Web-based: cross-platform, your vis tool can be used on almost any other machines
- Easy to find pre-write D3 code to build your own vis tool

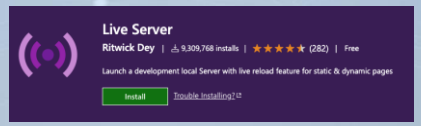
The Power of D3

- Check sample codes (a lot) on https://observablehq.com/@d3/gallery

Environment and Tool to Write D3

- Web browser – Chrome 

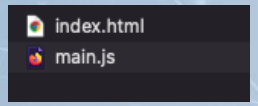
- Editor – I recommend “Visual Studio Code” (VSCode)
  - https://code.visualstudio.com/ 

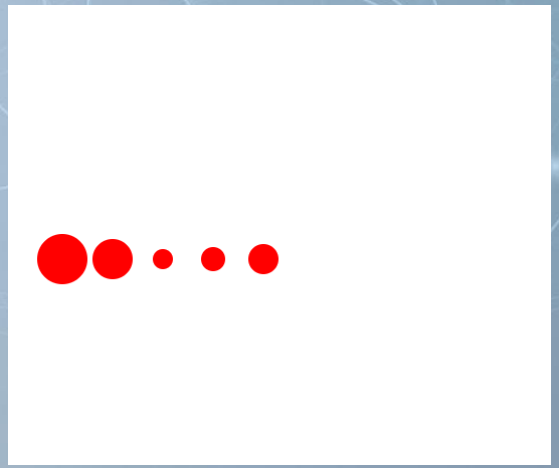
- Http-server
  - I recommend a VSCode plug-in “Live Server” for small scale test 

Our First D3 Program (Ex01-1)

- The purpose of this example is to introduce the environment to run a D3 program
- We will not introduce the syntax here

- Files





Our First D3 Program (Ex01-1)

- Index.html

To write D3 program, include D3 library

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="description" content="">
    <title>The First D3 Example</title>
</head>
<body>
    <div class="container">
        <div class="row">
            <div id="chart-area"></div>
        </div>
    </div>

    <script src="https://d3js.org/d3.v5.min.js"></script>
    <script src="main.js"></script>
</body>
</html>
```

Our First D3 Program (Ex01-1)

- Index.html

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="description" content="">
    <title>The First D3 Example</title>
</head>
<body>
    <div class="container">
        <div class="row">
            <div id="chart-area"></div>
        </div>
    </div>

    <script src="https://d3js.org/d3.v5.min.js"></script>
    <script src="main.js"></script>
</body>
</html>
```

We usually our D3 main program in another .js file
In this example, it is ”main.js” in the same folder

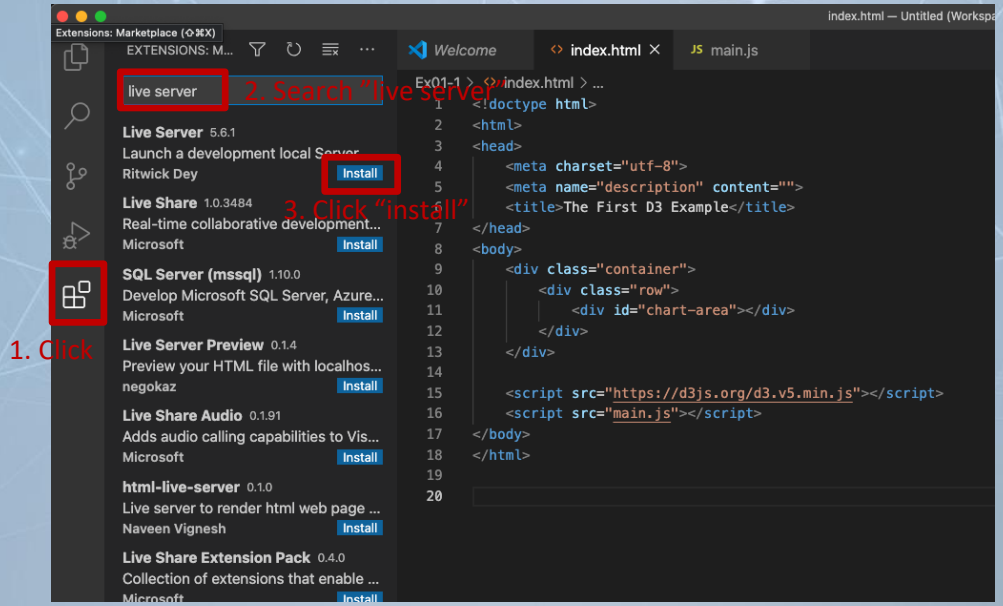
Our First D3 Program (Ex01-1)

- main.js

```
const data = [25, 20, 10, 12, 15]

const svg = d3.select("#chart-area").append("svg")
    .attr("width", 400)
    .attr("height", 400)

const circles = svg.selectAll("circle")
    .data(data)

circles.enter().append("circle")
    .attr("cx", (d, i) => (i * 50) + 50)
    .attr("cy", 250)
    .attr("r", (d) => d)
    .attr("fill", "red")
```

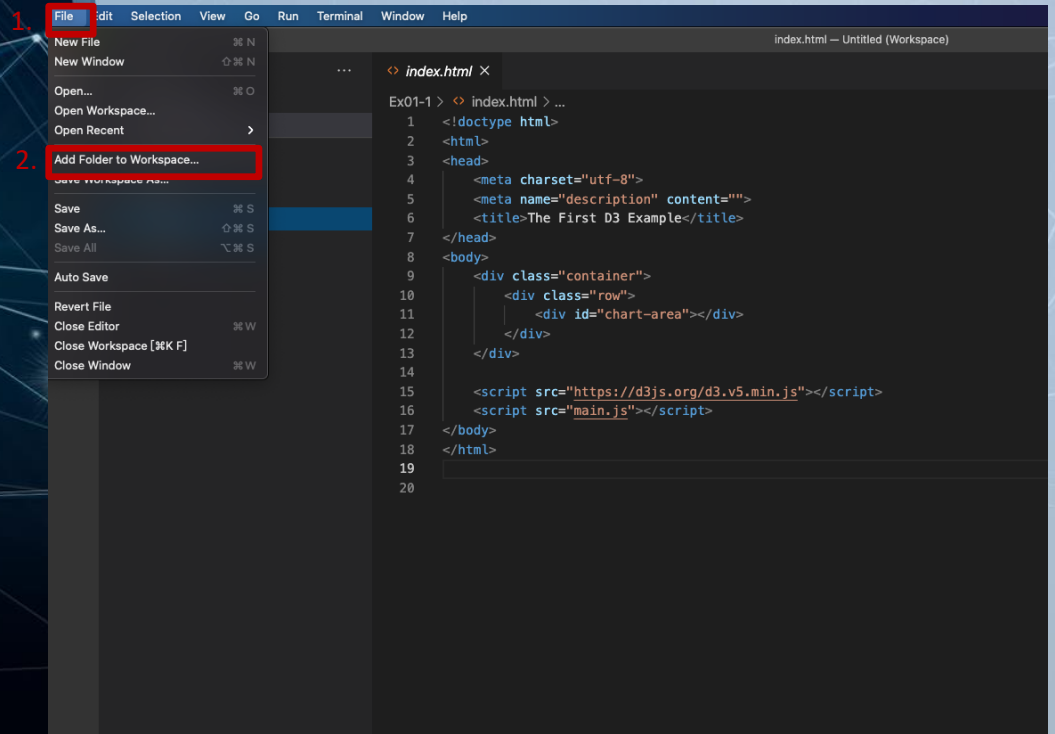
Our First D3 Program (Ex01-1)

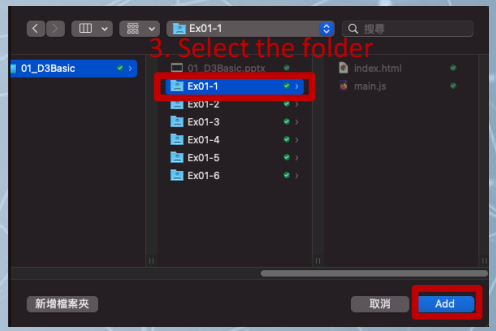
- Open “VSCode”
- Install “Live Server” extension

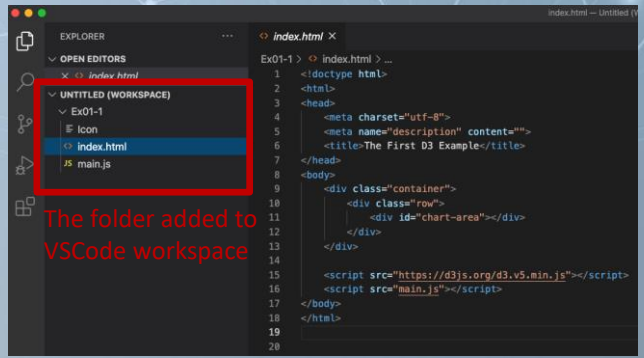


Our First D3 Program (Ex01-1)

- You have to add the folder into “workspace” to run it by Live Server

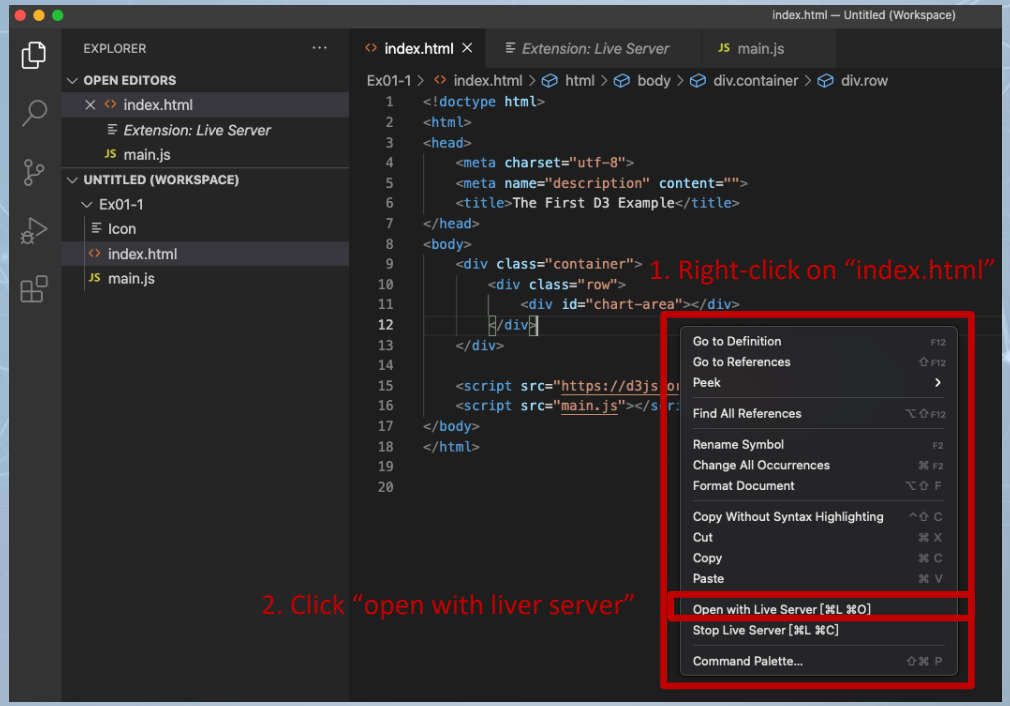




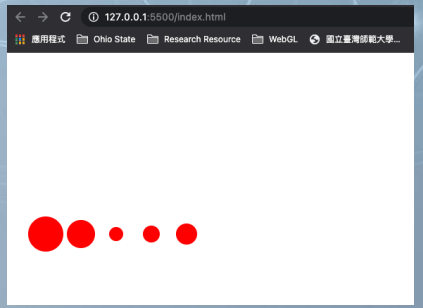


Our First D3 Program (Ex01-1)

- Run it by righ-clicking on “index.html”



3. “live server” can create a http server and run “index.html” on your browser



Try it!!!

Before D3

- D3 is web-based

- We need to know some web-based related languages and knowledge
  - HTML
  - CSS
  - SVG
  - Javascript

HTML – Hyper Text Markup Language

- HTML is the standard markup language for creating Web pages

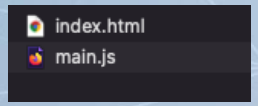
- HTML elements
  - HTML elements are the building blocks of HTML pages
  - Represented by tags

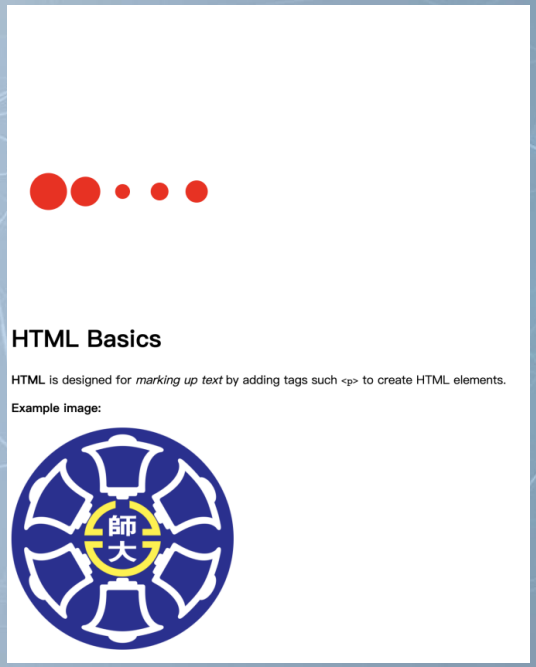
- Tag
  - HTML tags label pieces of content such as
    - <head>, <p>, <table>…..
  - Browsers do not display the HTML tags, but use them to render content of the page

Example (Ex01-2)

- Add more html tags into Ex01-1

- Files

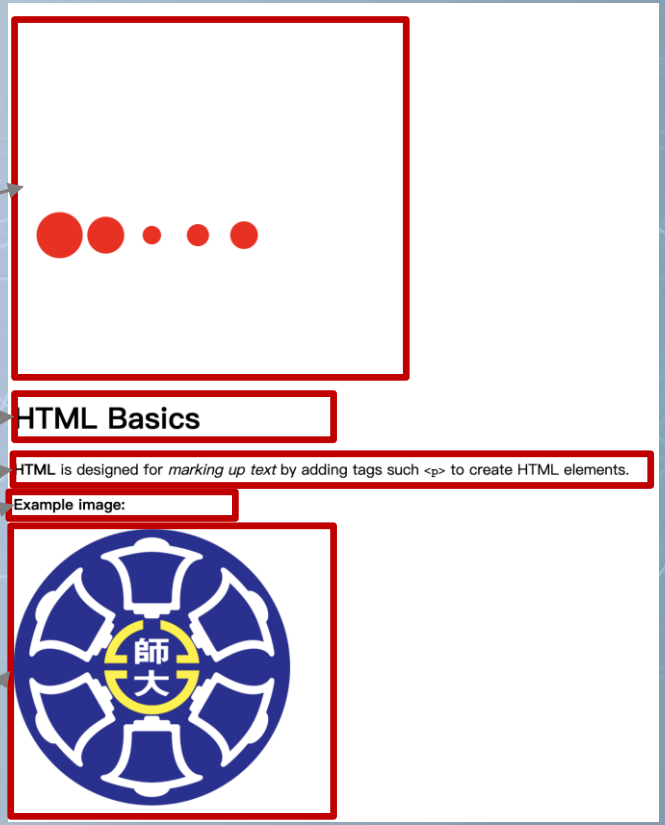




Example (Ex01-2)

- index.html

```
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>
<body>
    <div class="container">
        <div class="row">
            <div id="chart-area"></div>
        </div>
    </div>

    <h1>HTML Basics</h1>
    <p>
        <Strong>HTML</Strong>
        is designed for
        <em>marking up text</em>
        by adding tags such
        <code>&lt;p&gt;</code>
        to create HTML elements.
    </p>

    <p>
        <strong>Example image:</strong>
    </p>
    <img width="300" height="300" src="https://upload.wikimedia.org/wikipedia/en/thumb/c/c3/National_Taiwan_Norma

    <script src="https://d3js.org/d3.v5.min.js"></script>
    <script src="main.js"></script>
</body>
</html>
```

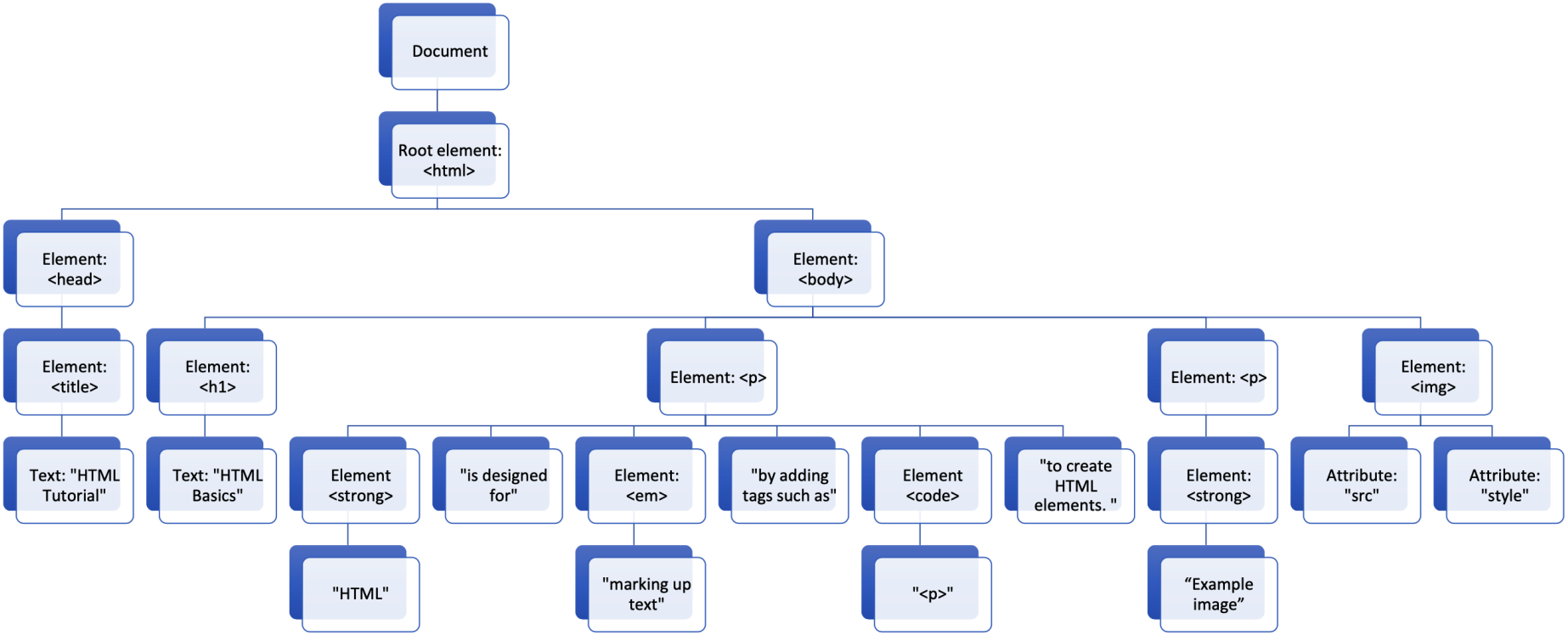


HTML - DOM

- When a web page is loaded, the browser create a Document Object Model (DOM) for the page

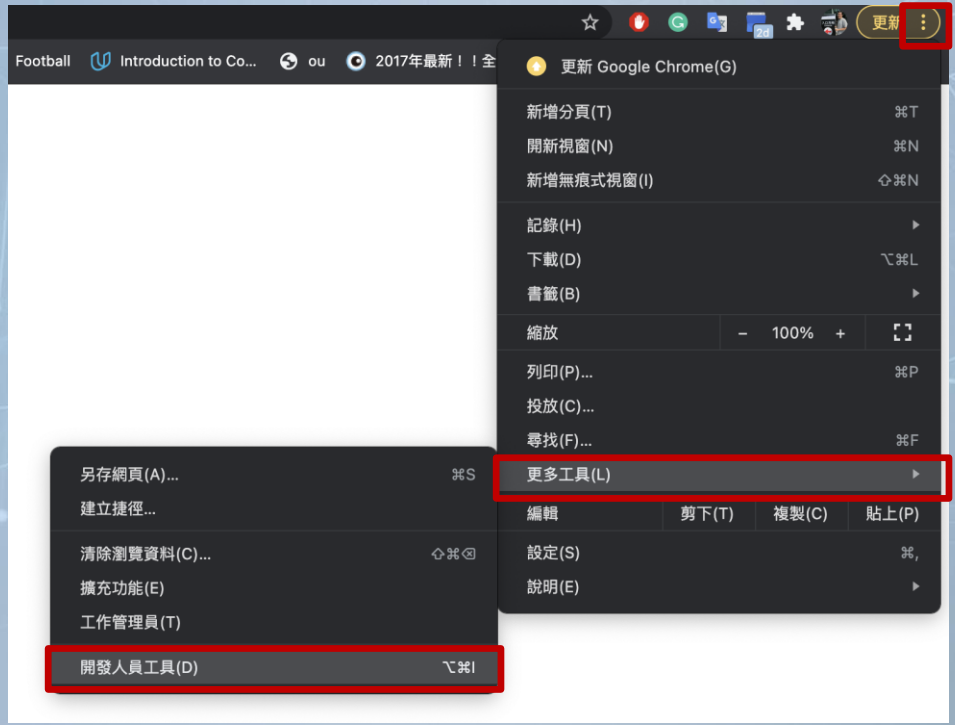
- The HTML DOM model is constructed as a **tree** of Objects

HTML - DOM


Document
Root element: <html>
Element: <head>
Element: <body>
Element: <title>
Element: <h1>
Element: <p>
Element: <p>
Element: <img>
Text: "HTML Tutorial"
Text: "HTML Basics"
Element
"is designed for"
Element:
"by adding tags such as"
Element <code>
"to create HTML elements. "
Element:
Attribute: "src"
Attribute: "style"
"HTML"
"marking up text"
"<p>"
“Example image”

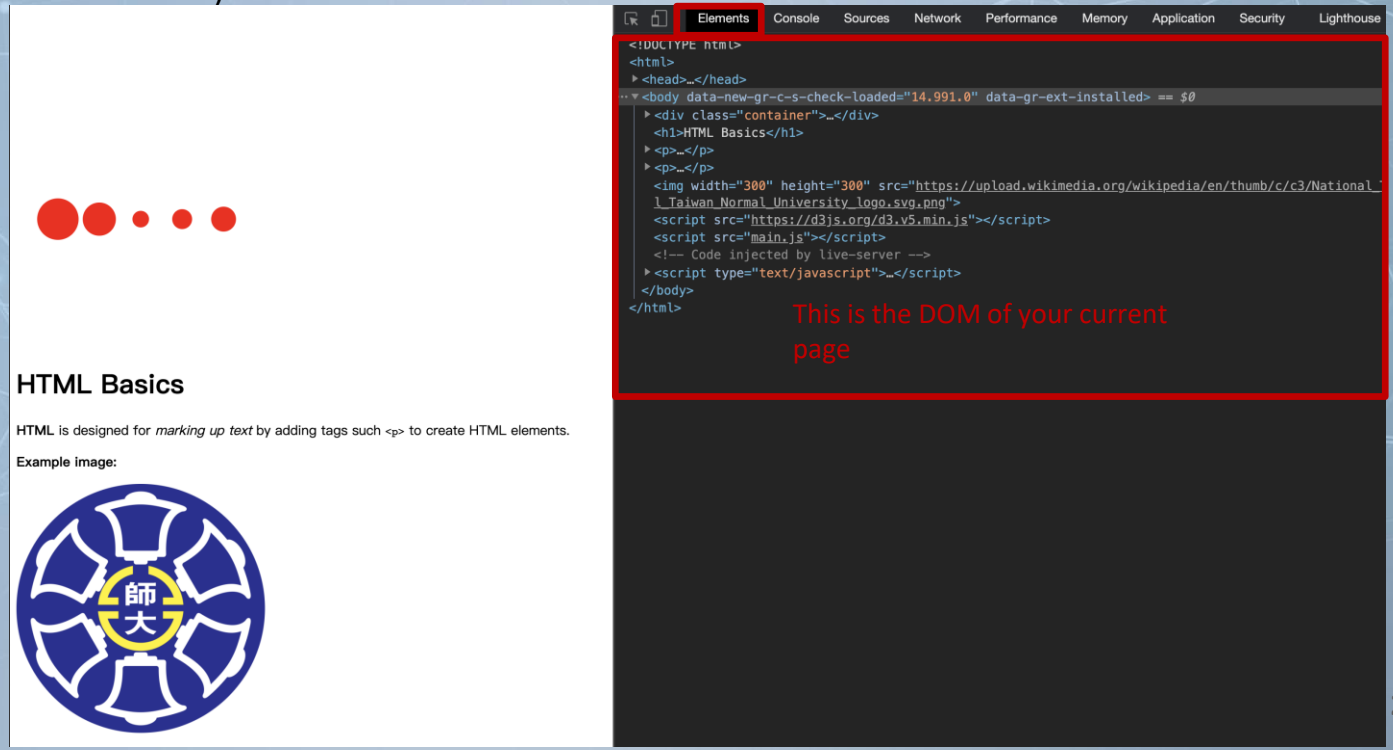
HTML - DOM

- To see the current DOM of Ex01-2 on Chrome



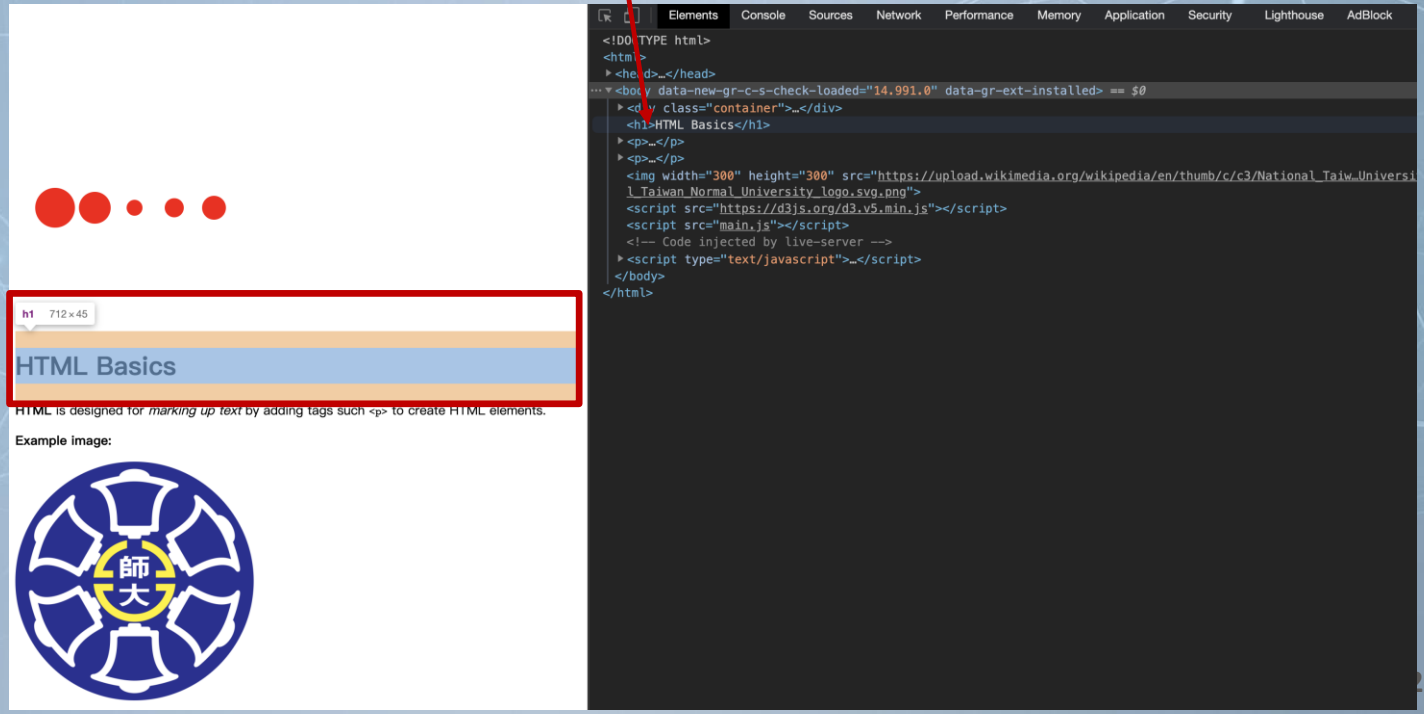
HTML - DOM

- Chrome will open a sub-window for developer (This is very useful for debugging in this semester)



HTML - DOM

- When I put my mouse cursor here, Chrome highlights the corresponding object on the page



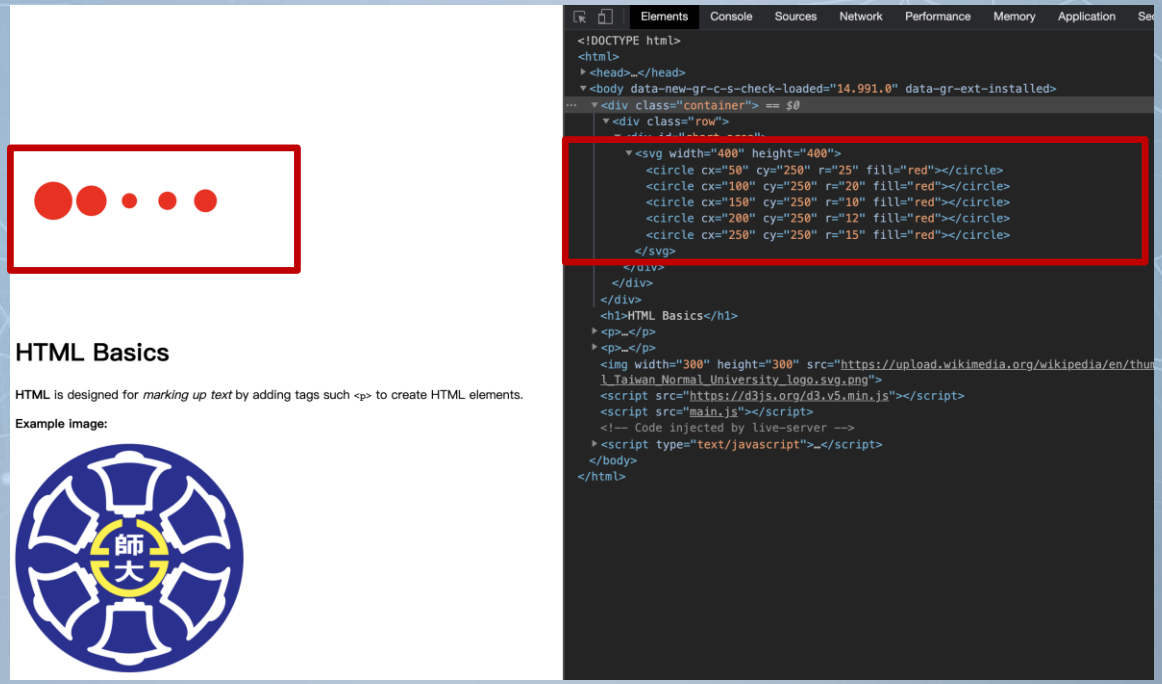
HTML - DOM

- With the object model, JavaScript can create dynamic HTML by manipulating the objects
  - Javascript can change all the HTML element in the page
  - Change all the HTML attributes in the page
  - Change all the CSS styles
  - Remove existing HTML elements and attributes
  - Add new HTML elements and attributes
  - React to all existing HTML elements in the page
  - Create new HTML events in the page

- In short, Javascript (D3) can dynamically (by the data or user input) to change the appearance of the page

HTML - DOM

- We do not have these circle tags in index.html, these are added by main.js (D3)



Try Ex01-2

- This course will not teach HTML.
- Even if you do not know HTML very well, it is ok because HTML is very easy to understand.
- If you really want a tutorial, you can check this website
  - https://www.w3schools.com/html/

CSS – Cascading Style Sheets

- CSS describe how HTML elements are to be displayed on screen
- CSS saves a lot of work
  - It can control the appearance of multiple elements and web pages all at once

Example (Ex01-3)

- index.html

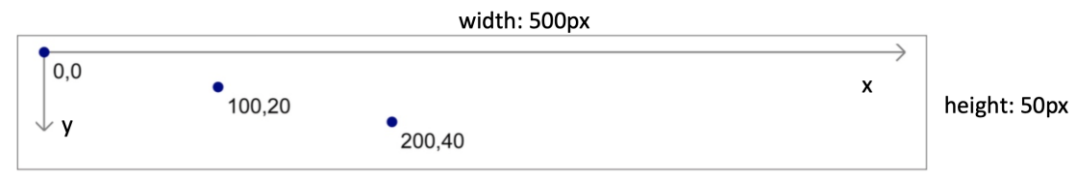

```
<html>
<head>
    <title>Example</title>
</head>
<style>
    p{
        color: red;
        font-family: verdana;
        font-size: 20px;
    }
    img{
        width: 200px;
        border-radius:50%;
    }
</style>
<body>
    <div class="container">
        <div class="row">
            <div id="chart-area"></div>
        </div>
    </div>

    <h1>HTML Basics</h1>
    <p>
        <Strong>HTML</Strong>
        is designed for
        <em>marking up text</em>
        by adding tags such
        <code>&lt;p&gt</code>
        to create HTML elements.
    </p>

    <p>
        <strong>Example image:</strong>
    </p>
    <img width="300" height="300" src="https://upload.wikimedia.org/wikipedia/en/thumb/c/c3/Nati

    <script src="https://d3js.org/d3.v5.min.js"></script>
    <script src="main.js"></script>
</body>
</html>
```

Try Ex01-3

- This course will not teach CSS.
- If you really want a tutorial, you can check this website
  - https://www.w3schools.com/css/default.asp

SVG – Scalable Vector Graphics

- SVG defines vector-based graphics for the web
- Svg HTML tag
  - <svg width="500" height="50"></svg>
  - Create a SVG canvas with 500x50pixels
- SVG coordinates system **(important!!!)**



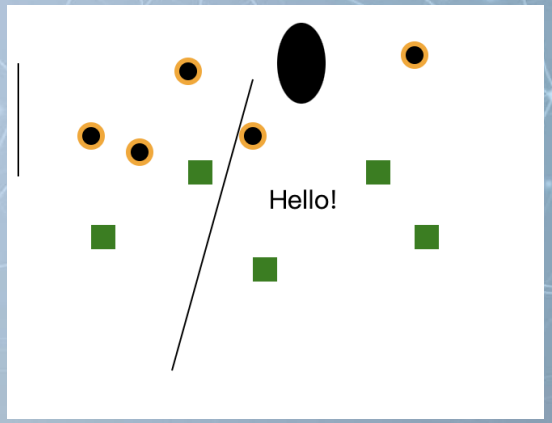
SVG - Shape

- On SVG, we can draw
  - Lines: <line>
  - Rectangles: <rect>
  - Circles: <circle>
  - Ellipse: <ellipse>
  - Text: <text>
  - Polygon: <polygon> 
  - Path: <path> (curve defined by points) 
  - …….

Example (Ex01-4)

- Draw lines, rectangles, circles, ellipses and text on SVG
- Use CSS

- Files:
  - Index.html



Example (Ex01-4)

- index.html
- Modify index.html to learn

```
<!doctype html>
<html>
<head>
    <title>Example</title>
    <style>
        rect{
            fill: green;
        }

        circle{
            stroke: orange;
            stroke-width: 3;
        }
    </style>
</head>

<body>
    <svg width="500" height="300">
        <line x1="5" x2="5" y1="100" y2="30" stroke="black"/>
          <line x1="100" x2="150" y1="220" y2="40" stroke="black"/>

          <rect x="150" y="150" width="15" height="15"/>
          <rect x="220" y="90" width="15" height="15"/>
          <rect x="110" y="90" width="15" height="15"/>
          <rect x="220" y="90" width="15" height="15"/>
          <rect x="50" y="130" width="15" height="15"/>
          <rect x="250" y="130" width="15" height="15"/>

          <circle cx="250" cy="25" r="7"/>
          <circle cx="150" cy="75" r="7"/>
          <circle cx="80" cy="85" r="7"/>
          <circle cx="110" cy="35" r="7"/>
          <circle cx="50" cy="75" r="7"/>

          <ellipse cx="180" cy="30" rx="15" ry="25"/>

          <text x="160" y="120">Hello!</text>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

Try it

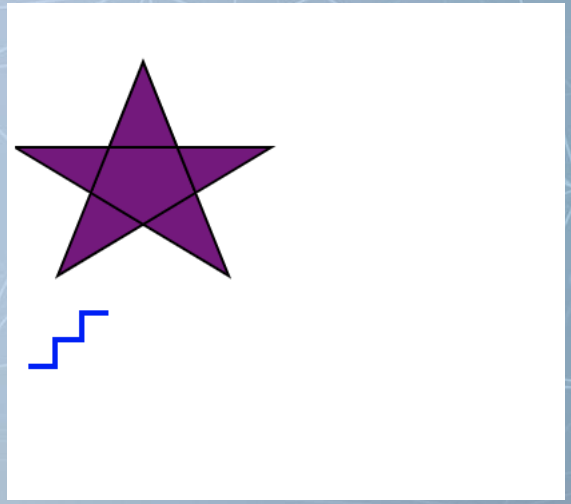
- What you should try
  - Open the developer tool to check the DOM structure first
  - What if you remove <style>…</style>
  - What if you change svg height to 100?
  - Modify the value of each parameter in line, rect, circle, ellipse, text, check result and guess the meaning of each parameters

- If you want to learn more about SVG
  - Check here, https://www.w3schools.com/graphics/svg_intro.asp

Example (Ex01-5)

- <polygon>
- <polyline>

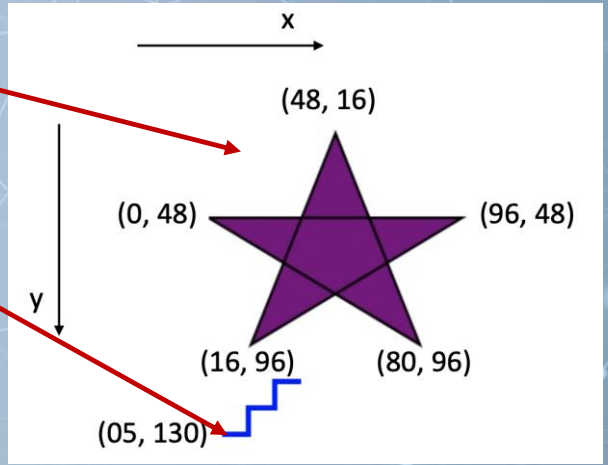
- File
  - Index.html



Example (Ex01-5)

- Check the coordinates

```
<body>
    <svg>
    <polygon style="fill: purple; stroke: black;"
        points="48,16  16,96  96,48  0,48  80,96" />

    <polyline fill="none" stroke="blue" stroke-width="2"
        points="05,130
                15,130
                15,120
                25,120
                25,110
                35,110" />
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```



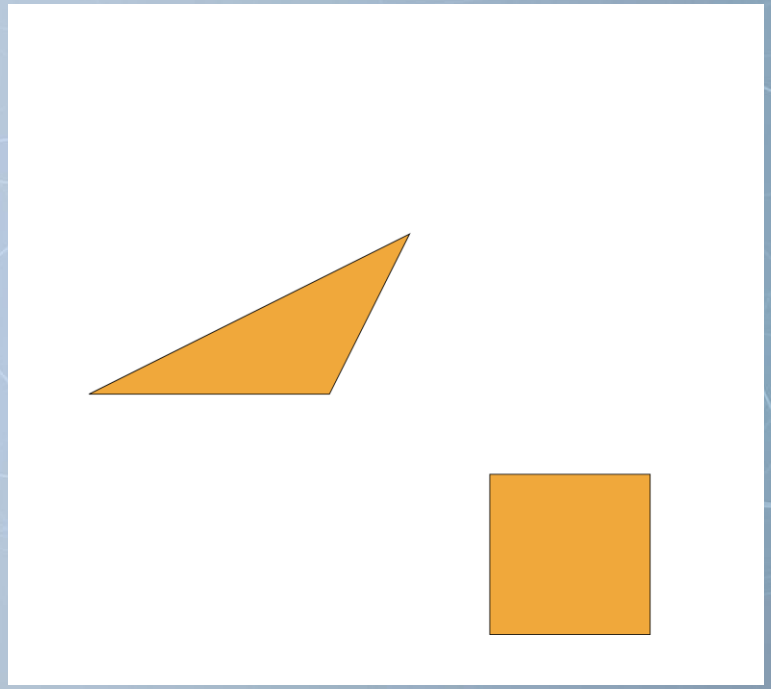
Try it

- And, you can learn more from here,
  - https://www.w3schools.com/graphics/svg_polygon.asp
  - https://www.w3schools.com/graphics/svg_polyline.asp

Example (Ex01-6)

- <path>

- File:
  - Index.html



Example (Ex01-6)

- Path is useful and important in D3 visualization
  - You can use it to make almost any shape
  - And, understanding it well helps you to read codes

Example (Ex01-6)

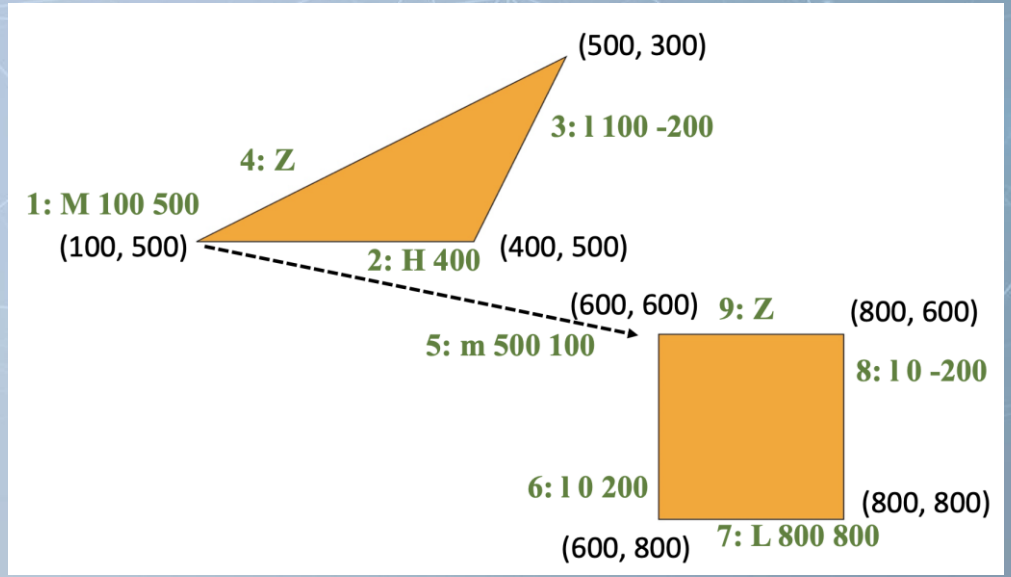
- M = move (pen) to
  - m = move by
- L = line to
  - l = line by
- H = horizontal line to
- V = vertical line to
- C = curve to
- S = smooth curve to
- Q = quadratic Bézier curve
- T = smooth quadratic Bézier curve to
- A = elliptical Arc
- Z = close path

- More details: check https://www.w3schools.com/graphics/svg_path.asp

Example (Ex01-6)

- Let's use the above table to explain Ex01-6

```
<body>
    <svg width="1000" height="1000">
        <path d="
                M 100 500
                H 400
                l 100 -200
                Z

                m 500 100
                l 0 200
                L 800 800
                l 0 -200
                Z"
        fill="orange" stroke="black"/>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

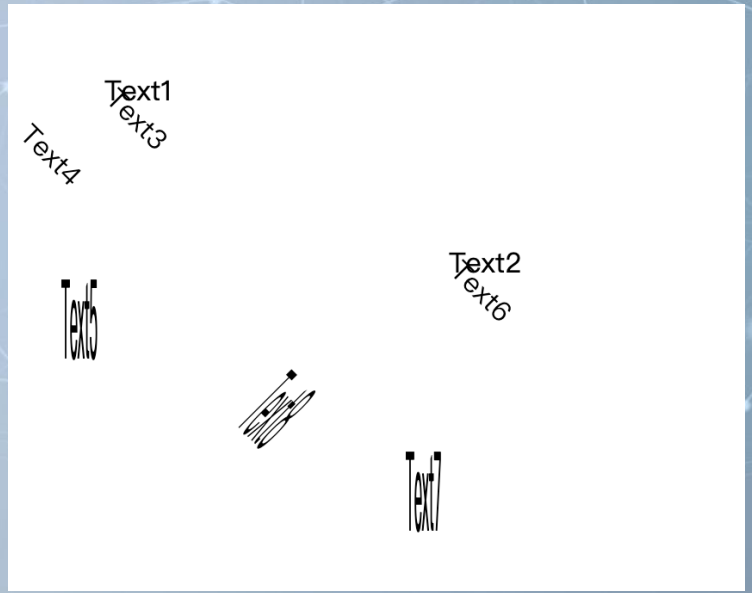


Try it

- Can you draw your own shape?

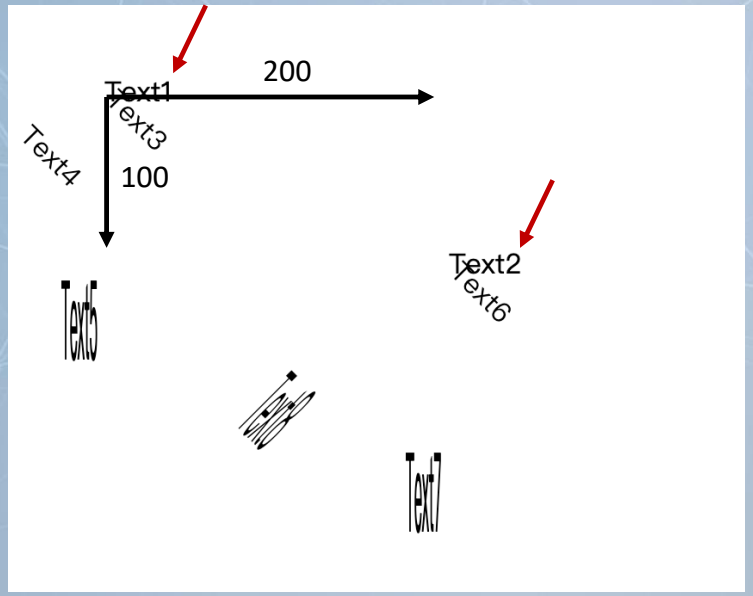
Example (Ex01-7)

- SVG – Transformation
  - translate
  - scale
  - rotate
- File
  - index.html



Example (Ex01-7)

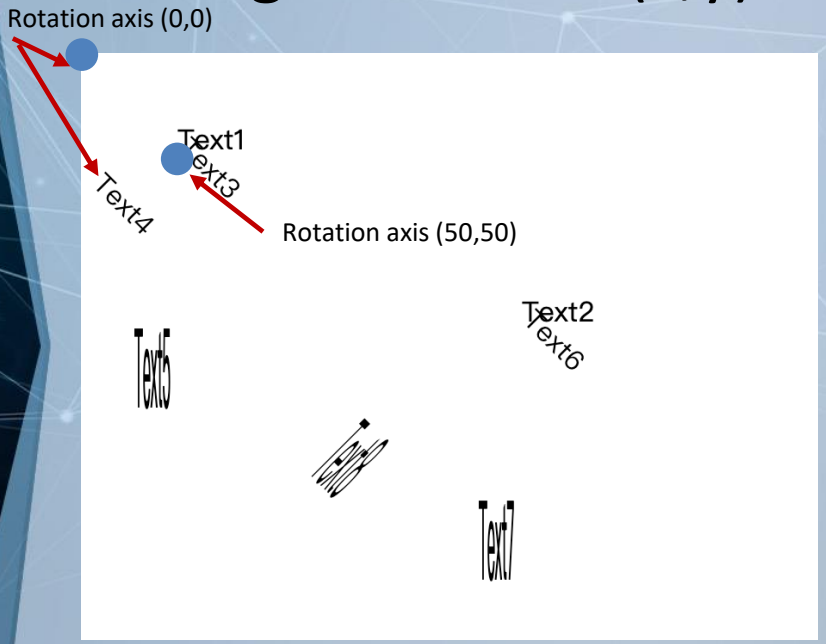
- translate – move by (dx, dy)




```
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>

<body>
    <svg width="1000" height="1000">
        <text x="50" y="50">
            Text1
        </text>
        <text x="50" y="50" transform="translate(200,100)">
            Text2
        </text>
        <text x="50" y="50" transform="rotate(45, 50, 50)">
            Text3
        </text>
        <text x="50" y="50" transform="rotate(45, 0, 0)">
            Text4
        </text>
        <text x="50" y="50" transform="scale(0.5,4.0)">
            Text5
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50)">
            Text6
        </text>
        <text x="50" y="50" transform="translate(200,100) scale(0.5,4.0)">
            Text7
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50) scale(0.5,4.0) ">
            Text8
        </text>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

Example (Ex01-7)

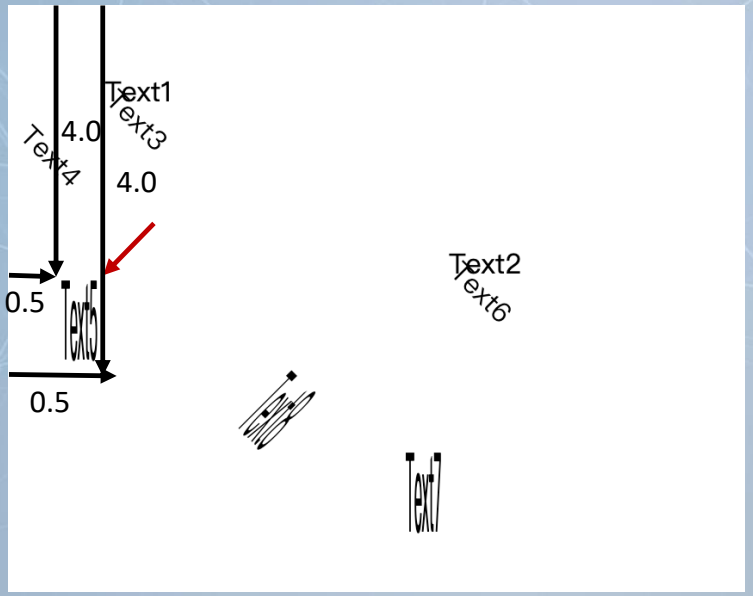
- rotate(a, x, y) – rotate a shape by ‘a’ degrees about (x,y)




```
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>

<body>
    <svg width="1000" height="1000">
        <text x="50" y="50">
            Text1
        </text>
        <text x="50" y="50" transform="translate(200,100)">
            Text2
        </text>
        <text x="50" y="50" transform="rotate(45, 50, 50)">
            Text3
        </text>
        <text x="50" y="50" transform="rotate(45, 0, 0)">
            Text4
        </text>
        <text x="50" y="50" transform="scale(0.5,4.0)">
            Text5
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50)">
            Text6
        </text>
        <text x="50" y="50" transform="translate(200,100) scale(0.5,4.0)">
            Text7
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50) scale(0.5,4.0) ">
            Text8
        </text>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

Example (Ex01-7)

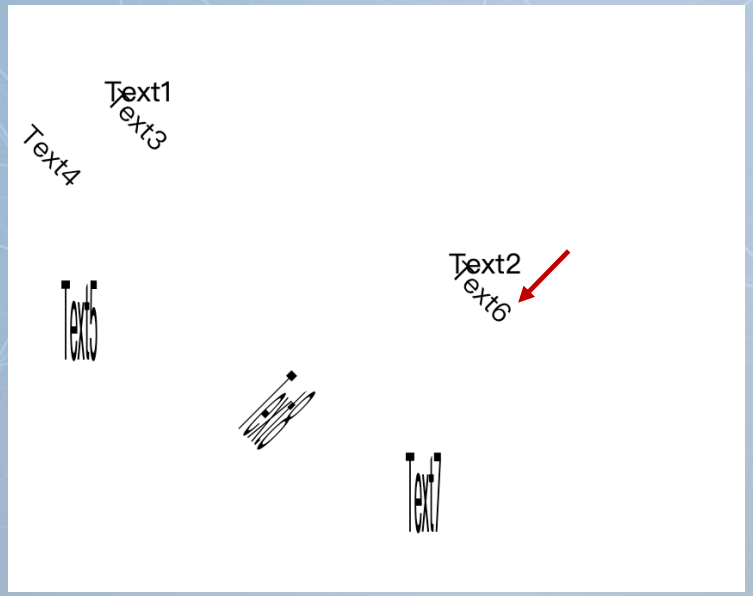
- scale(sx, sy)




```
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>

<body>
    <svg width="1000" height="1000">
        <text x="50" y="50">
            Text1
        </text>
        <text x="50" y="50" transform="translate(200,100)">
            Text2
        </text>
        <text x="50" y="50" transform="rotate(45, 50, 50)">
            Text3
        </text>
        <text x="50" y="50" transform="rotate(45, 0, 0)">
            Text4
        </text>
        <text x="50" y="50" transform="scale(0.5,4.0)">
            Text5
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50)">
            Text6
        </text>
        <text x="50" y="50" transform="translate(200,100) scale(0.5,4.0)">
            Text7
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50) scale(0.5,4.0) ">
            Text8
        </text>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

Example (Ex01-7)

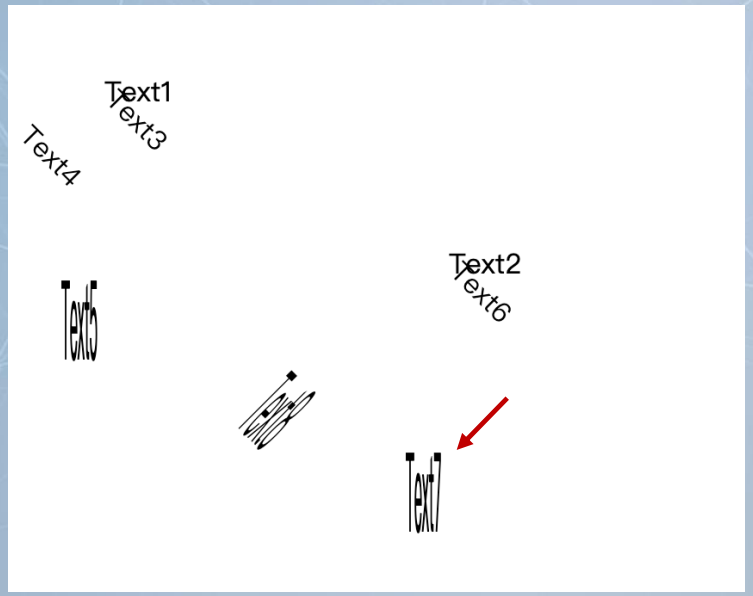
- Multiple transformations - from left to right




```
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>

<body>
    <svg width="1000" height="1000">
        <text x="50" y="50">
            Text1
        </text>
        <text x="50" y="50" transform="translate(200,100)">
            Text2
        </text>
        <text x="50" y="50" transform="rotate(45, 50, 50)">
            Text3
        </text>
        <text x="50" y="50" transform="rotate(45, 0, 0)">
            Text4
        </text>
        <text x="50" y="50" transform="scale(0.5,4.0)">
            Text5
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50)">
            Text6
        </text>
        <text x="50" y="50" transform="translate(200,100) scale(0.5,4.0)">
            Text7
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50) scale(0.5,4.0) ">
            Text8
        </text>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

Example (Ex01-7)

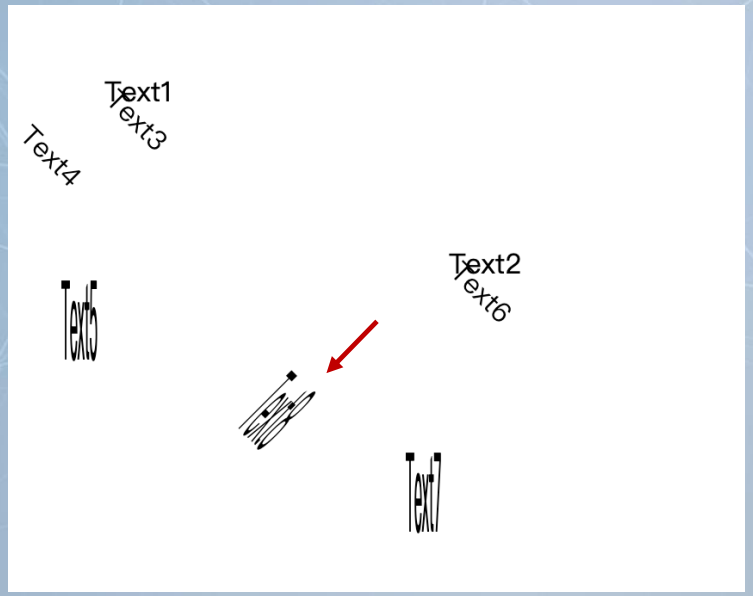
- Multiple transformations - from left to right




```
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>

<body>
    <svg width="1000" height="1000">
        <text x="50" y="50">
            Text1
        </text>
        <text x="50" y="50" transform="translate(200,100)">
            Text2
        </text>
        <text x="50" y="50" transform="rotate(45, 50, 50)">
            Text3
        </text>
        <text x="50" y="50" transform="rotate(45, 0, 0)">
            Text4
        </text>
        <text x="50" y="50" transform="scale(0.5,4.0)">
            Text5
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50)">
            Text6
        </text>
        <text x="50" y="50" transform="translate(200,100) scale(0.5,4.0)">
            Text7
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50) scale(0.5,4.0) ">
            Text8
        </text>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

Example (Ex01-7)

- Multiple transformations - from left to right




```
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>

<body>
    <svg width="1000" height="1000">
        <text x="50" y="50">
            Text1
        </text>
        <text x="50" y="50" transform="translate(200,100)">
            Text2
        </text>
        <text x="50" y="50" transform="rotate(45, 50, 50)">
            Text3
        </text>
        <text x="50" y="50" transform="rotate(45, 0, 0)">
            Text4
        </text>
        <text x="50" y="50" transform="scale(0.5,4.0)">
            Text5
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50)">
            Text6
        </text>
        <text x="50" y="50" transform="translate(200,100) scale(0.5,4.0)">
            Text7
        </text>
        <text x="50" y="50" transform="translate(200,100) rotate(45, 50, 50) scale(0.5,4.0) ">
            Text8
        </text>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

Try it

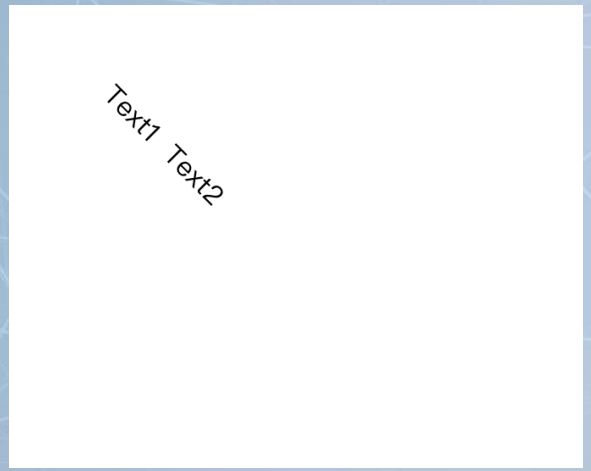
- Make sure you really understand every examples

Example (Ex01-8)

- <g> - group multiple shapes
- Why? You can transform them together

- File
  - index.html

Example (Ex01-8)




```
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>

<body>
    <svg width="1000" height="1000">
        <g transform="rotate(45,50,50)">
            <text x="50" y="50">
                Text1
            </text>
            <text x="100" y="50">
                Text2
            </text>
        </g>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

Try it

- Try to modify the transformation in <g> and see what happens

Example (Ex01-9)

- SVG has no layer
  - Any pixel-paint applied later obscure any earlier paint and therefore appear to be “in front”

```
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>

<body>
    <svg>
        <rect x="0" y="0" width="30" height="30" fill="purple"/>
        <rect x="20" y="5" width="30" height="30" fill="blue"/>
        <rect x="40" y="10" width="30" height="30" fill="green"/>
        <rect x="60" y="15" width="30" height="30" fill="yellow"/>
        <rect x="80" y="20" width="30" height="30" fill="red"/>
    </svg>

    <script src="https://d3js.org/d3.v5.min.js"></script>
</body>
</html>
```

SVG Elements Reference

- https://developer.mozilla.org/en-US/docs/Web/SVG/Element

SVG elements A to Z

A

- <a>
- <animate>
- <animateMotion>
- <animateTransform>

C

- <circle>
- <clipPath>
- <color-profile>

D

- <defs>
- <desc>
- <discard>

E

- <ellipse>

F

- <feBlend>
- <feColorMatrix>
- <feComponentTransfer>
- <feComposite>
- <feConvolveMatrix>
- <feDiffuseLighting>
- <feDisplacementMap>
- <feDistantLight>
- <feDropShadow>
- <feFlood>
- <feFuncA>
- <feFuncB>
- <feFuncG>
- <feFuncR>
- <feGaussianBlur>
- <feImage>
- <feMerge>
- <feMergeNode>
- <feMorphology>
- <feOffset>
- <fePointLight>
- <feSpecularLighting>
- <feSpotLight>
- <feTile>
- <feTurbulence>
- <filter>
- <foreignObject>

G

- <g>

H

- <hatch>
- <hatchpath>

I

- <image>

L

- <line>
- <linearGradient>

M

- <marker>
- <mask>
- <mesh>
- <meshgradient>
- <meshpatch>
- <meshrow>
- <metadata>
- <mpath>

P

- <path>
- <pattern>
- <polygon>
- <polyline>

R

- <radialGradient>
- <rect>

S

- <script>
- <set>
- <solidcolor>
- <stop>
- <style>
- <svg>
- <switch>
- <symbol>

T

- <text>
- <textPath>
- <title>
- <tspan>

U

- <unknown>
- <use>

V

- <view>

Brief Introduction to Javascript

High-level definition

HTML

- Define paragraphs, headings, data tables, or embed pictures and videos on the page.

CSS

- Set the background color, font, or make the content presented in multiple columns 。

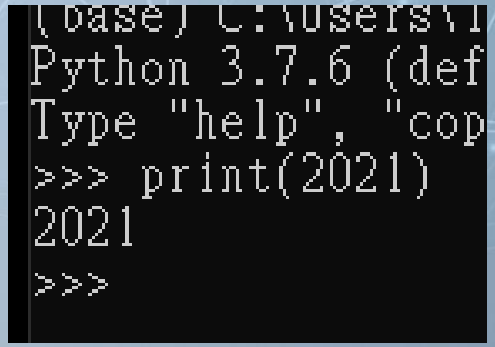
JavaScript

- Allow you to dynamically update content, control multimedia, animation --- almost everything.

Console

- The console is a panel that displays important messages, like errors, for developers. Much of the work the computer does with our code is invisible to us by default. If we want to see things appear on our screen, we can print, or log, to our console directly.

```
> console.log(2021)
  2021
```



semi-colon(;)

- In JavaScript, all code instructions should end with a semi-colon (;)
- Your code may work correctly for single lines, but probably won't when you are writing multiple lines of code together. Try to get into the habit of including it.

```
> console.log(2021)
  2021
```

Comments

- Single line
- Multiple line

```
> // Single line comment
< undefined
> /* multiple line comment
  multiple line comment
  multiple line comment */
< undefined
```

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Sep 17 15:21:36 2019
4
5  @author: HaoYi
6  """
```

Declaring variables

var

- Declare local or global writable variables.

let

- Declare local writable variables.

const

- Declare read-only variables.

var

- **var** is used in pre-EC6 versions of JavaScript.
- There is no reason to use **var**, unless you need to support old versions of Internet Explorer with your code.

```
> var x;
< undefined
> x = 2021;
< 2021
> console.log(x);
  2021
```

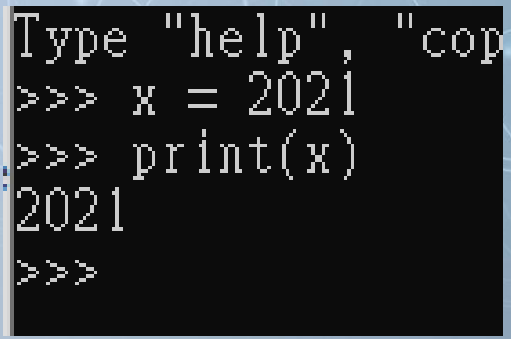
```
> var x = 2021;
< undefined
> console.log(x);
  2021
```

```
Type "help", "cop
>>> x = 2021
>>> print(x)
2021
>>>
```

let

- **let** is the preferred way to declare a variable when it can be reassigned.
- A **let** variable will contain **undefined** if nothing is assigned to it.

```
> let x;
< undefined
> x = 2021;
< 2021
> console.log(x);
  2021
```



const

- **const** is the preferred way to declare a variable with a constant value.
- It must have an assignment.
- Any attempt of re-assigning a **const** variable will result in JS runtime error.

```
> const a;
Uncaught SyntaxError: Missing initializer in const declaration
> const a = 50;
undefined
> a = 80;
Uncaught TypeError: Assignment to constant variable.
    at <anonymous>:1:3
```

The difference between var and let

- why do we need two keywords for defining variables? Why have **var** and **let**?
- https://www.geeksforgeeks.org/difference-between-var-and-let-in-javascript/#:~:text=var%20and%20let%20are%20both,with%20var%20keyword%20are%20hoisted.

```
var myName = 'Chris';
var myName = 'Bob';
```

```
let myName = 'Chris';
let myName = 'Bob';
```

```
let myName = 'Chris';
myName = 'Bob';
```

Variable types

- Numbers
- Strings
- Booleans
- Dictionary
- Arrays
- Objects

Array

- Arrays consist of square brackets and elements that are separated by commas.

```
> let x = [1,2,3,'string',[5,6,7]];
<- undefined
```

```
>>> x = [1,2,3,'string',[5,6,7]]
```

- You can then access individual items in the array using bracket notation.

```
> console.log(x[0]);
  1
```

```
>>> print(x[0])
1
```

- You can find out the length of an array (how many items are in it) by using the **length** property.

```
> console.log(x.length);
  5
```

```
>>> len(x)
5
```

Converting between strings and arrays

- split()

```
> let data = "facebook,youtube,google,ptt";
< undefined
> let arr = data.split();
< undefined
> console.log(arr);
  ▶ ["facebook,youtube,google,ptt"]
```

```
>>> data = "facebook,youtube,google,ptt"
>>> arr = data.split()
>>> print(arr)
['facebook,youtube,google,ptt']
```

- join()

```
> let str = arr.join();
< undefined
> console.log(str);
  facebook,youtube,google,ptt
```

```
>>> str = "".join(arr)
>>> print(str)
facebook,youtube,google,ptt
```

- toString()

```
> console.log(arr.toString());
  facebook,youtube,google,ptt
```

Dictionary

```
> let dic = {'name': 'Taipei', 'population': 2000000, 'Region': 'north'};
< undefined
> console.log(dic);
  ▶ {name: "Taipei", population: 2000000, Region: "north"}
< undefined
> console.log(dic.name);
  Taipei
< undefined
> console.log(dic['name']);
  Taipei
< undefined
> console.log(dic.population);
  2000000
< undefined
> console.log(dic['population']);
  2000000
< undefined
```

Array of Dictionary

```
> let dic = [{'name': 'Taipei', 'population': 2000000, 'Region': 'north'},
             {'name': 'Kaohsiung', 'population': 1700000, 'Region': 'south'},
             {'name': 'Tainan', 'population': 1500000, 'Region': 'south'}

            ];
< undefined
> console.log(dic[1].population);
  1700000
< undefined
> console.log(dic[2].name);
  Tainan
< undefined
> console.log(dic[0]['Region']);
  north
< undefined
```

Conditions

- Logical Operators
- if...else statements
- else if clause
- switch statements

Logical operators

- Or ||

```
> (3 < 5) || (3 > 5)
<- true
```

```
>>> (3 < 5) or (3 > 5)
True
```

- And &&

```
> (3 < 5) && (3 > 5)
<- false
```

```
>>> (3 < 5) and (3 > 5)
False
```

- Not !

```
> !(3 < 5)
<- false
```

```
>>> not (3 < 5)
False
```

if...else

```
> let x = 2021;
< undefined
> if(x===2021){
      console.log(x+100);
  }
  else{
      console.log(x);
  }
  2121
```

```
>>> x = 2021
>>> if(x==2021):
...     print(x+100)
... else:
...     print(x)
...
2121
```

== -> Convert 2 variable to same types then compare

=== -> If variable types are different, false is returned

else if

```
> if(x===2021){
      console.log(x+100);
  }
  else if(x===2022){
      console.log(x+200);
  }
  else if(x===2023){
      console.log(x+300);
  }
  else{
      console.log(x);
  }
```

```
>>> if(x==2021):
...     print(x+100)
... elif(x==2022):
...     print(x+200)
... elif(x==2023):
...     print(x+300)
... else:
...     print(x)
```

function

```
> function foo(){
      console.log("hello world!");
  }
< undefined
> foo();
  hello world!
```

```
>>> def foo():
...     print("hello world!")
...
>>> foo()
hello world!
```

```
> let foo2 = foo;
< undefined
> foo2();
  hello world!
```

```
>>> foo2 = foo
>>> foo2()
hello world!
```

Arrow function

```
> function foo(){
      console.log("hello world!");
  }
< undefined
> foo();
  hello world!
```

```
> let foo = () => {
      console.log("hello world!");
  }
< undefined
> foo();
  hello world!
```

Arrow function

```
> let foo = function(d){ return d*2; }
<· undefined
> foo(3)
<· 6
```

```
> let foo = (d) => (d*2);
<· undefined
> foo(3)
<· 6
```

++ and --

- ++

```
> let x = 2021;
< undefined
> console.log(x++);
  2021
< undefined
> console.log(x);
  2022
```

- --

```
> let x = 2021;
< undefined
> console.log(--x);
  2020
< undefined
> console.log(x);
  2020
```

Loops

- while loop
- do...while loop
- for loop

while

```
> while(x>0){
      console.log(x);
      x = x - 1;
  }
  5
  4
  3
  2
  1
```

```
>>> x = 5
>>> while(x>0):
...     print(x)
...     x = x - 1
...
5
4
3
2
1
```

do...while

```
> let x = 3;
< undefined
> do{
      console.log(x);
      x = x - 1;
  }while(x>3);
```

for

```
> let x = [2020,2021,2023,2024];
< undefined
> for(let i = 0 ; i < x.length ; i++){
      console.log(x[i]);
  }
  2020
  2021
  2023
  2024
```

```
>>> x = [2020,2021,2023,2024]
>>> for i in range(len(x)):
...     print(x[i])
...
2020
2021
2023
2024
```

Javascritp Functions for Light-weight Data Processing

- forEach()
- find()
- every()
- some()
- map()
- reduce()

Array.prototype.forEach()

```
> let x = [2020,2021,2023,2024];
< undefined
> for(let i = 0 ; i < x.length ; i++){
      console.log(x[i]);
  }
  2020
  2021
  2023
  2024
```

```
>>> x = [2020,2021,2023,2024]
>>> for i in range(len(x)):
...     print(x[i])
...
2020
2021
2023
2024
```

```
> function foo(a){
      console.log(a);
  }
< undefined
> x.forEach(foo);
  2020
  2021
  2023
  2024
```

```
> x.forEach(function(a){
      console.log(a);
  });
  2020
  2021
  2023
  2024
```

```
> x.forEach((a) => {
      console.log(a);
  });
  2020
  2021
  2023
  2024
```

```
> x.forEach(a => console.log(a));
  2020
  2021
  2023
  2024
```

Array.prototype.forEach()

```
> let x = [2021, 2022, 2023, 2024]
<- undefined
> x.forEach(function(d, i)
            {
                console.log(d, i);
            });
  2021 0
  2022 1
  2023 2
  2024 3
```

```
> let x = [2021, 2022, 2023, 2024];
<- undefined
> x.forEach((d,i)=>{
                console.log(d,i);
            });
  2021 0
  2022 1
  2023 2
  2024 3
```

Array.prototype.forEach()

Iterate through all element of the array to modify the value (if you want)

Array.prototype.filter()

```
let data = [{name: "Ben", age:16}, {name: "Casey", age:22}, {name: "Stella", age:19}, {name: "James", age:20}]
undefined
filterData = data.filter( d => (d.age > 19.5) )
▼(2) [{…}, {…}]
  ▶0: {name: 'Casey', age: 22}
  ▶1: {name: 'James', age: 20}
   length: 2
  ▶[[Prototype]]: Array(0)
```

Return an new array that only keep the elements that meet the condition you give

Array.prototype.find()

```
let data = [{name: "Ben", age:16}, {name: "Casey", age:22}, {name: "Stella", age:19}, {name: "James", age:20}]
undefined
findItem = data.find( d => d.age>19.5 )
▼{name: 'Casey', age: 22} i
    age: 22
    name: 22 asey"
  ▶ [[Prototype]]: Object
```

find() is very similar to filter, but is only return the first item with true return value from the function you give

Array.prototype.find()

```
let data = [{name: "Ben", age:16}, {name: "Casey", age:22}, {name: "Stella", age:19}, {name: "James", age:20}]
undefined
findItem = data.find( d => d.age>19.5 )
▼{name: 'Casey', age: 22} i
    age: 22
    name: 22 asey"
  ▶ [[Prototype]]: Object
```

find() is very similar to filter, but is only return the first item with true return value from the function you give

Array.prototype.every()

```
> let data = [{name: "Ben", age:16}, {name: "Casey", age:22}, {name: "Stella", age:19}, {name: "James", age:20}]
< undefined
> everyResult = data.every( d => d.age>19.5 )
< false
> everyResult = data.every( d => d.age>15 )
< true
```

every() only return one value, true or false.
If “all” data items meet the condition you give, it returns true.
Otherwise, it returns false.

Array.prototype.some()

```
> let data = [{name: "Ben", age:16}, {name: "Casey", age:22}, {name: "Stella", age:19}, {name: "James", age:20}]
< undefined
> someResult = data.some( d => d.age>19.5 )
< true
> someResult = data.some( d => d.age>23 )
< false
```

some() only return one value, true or false.
If “all” the data items do “not” meet the condition you give, it returns “false”.
Otherwise, it returns true.

Array.prototype.map()

```
let data = [{name: "Ben", age:16}, {name: "Casey", age:22}, {name: "Stella", age:19}, {name: "James", age:20}]
undefined
mapNewArray = data.map( (d, i) => { return {name: d.name, age:(d.age+10), newString: "new"} } )
▼(4) [{…}, {…}, {…}, {…}] ℹ
  ▶0: {name: 'Ben', age: 26, newString: 'new'}
  ▶1: {name: 'Casey', age: 32, newString: 'new'}
  ▶2: {name: 'Stella', age: 29, newString: 'new'}
  ▶3: {name: 'James', age: 30, newString: 'new'}
   length: 4
  ▶[[Prototype]]: Array(0)
```

map() returns a new array whose size is the same as the array you give.
map() process each item to generate one corresponding new item in the new array

This example returns a new array.
We keep same “name” for each one.
We add 10 to age for each one and we create a new key-value pair newString: “new” for each item.

Array.prototype.reduce()

```
> let data = [{name: "Ben", age:16}, {name: "Casey", age:22}, {name: "Stella", age:19}, {name: "James", age:20}]
< undefined
> value = data.reduce(function(accumulator, currentValue, currentIndex, array){
                        console.log(accumulator, currentValue, currentIndex);
                        return accumulator + currentValue.age;
                      }, 0);
  0 ▸ {name: 'Ben', age: 16} 0                                          VM2973:2
  16 ▸ {name: 'Casey', age: 22} 1                                       VM2973:2
  38 ▸ {name: 'Stella', age: 19} 2                                      VM2973:2
  57 ▸ {name: 'James', age: 20} 3                                       VM2973:2
< 77
```

reduce() can iterate through all items in the array and combine them to a value
reduce() takes two input arguments. The first one is a function to tell reduce() how to “reduce” data in your array. The second argument is the initial value before reduce() starts to run.

The first argument (accumulator) of the function you give: the accumulated value so far
The second argument (currentValue) of the function you give: the input data item of this iteration
The third argument (currentIndex) of the function you give: the index of the input data item of this interation

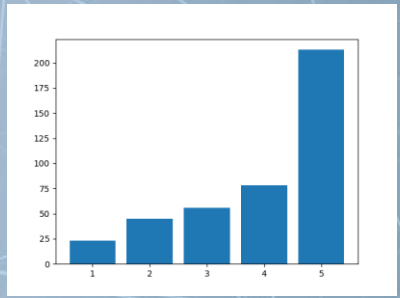
“return” of the function: output of the current iteration

Reference

- https://developer.mozilla.org/en-US/docs/Web/JavaScript
- https://www.codecademy.com/catalog/language/javascript

D3

- OK, so again, what D3 can do?
- D3 is going to “visualize data”
  - Visualize data: convert data to shapes with different attribute
- Example:
  - If we have data: [24, 50, 55, 75, 200]
  - Visualize the data to bar chart
  - 24, 50, 55, 75, 200 can be the height of the bars
  - <rect x=”100” y=“20” width=“30” height=“75”>



- D3 visualize the data by adding/removing/modifying those shape tags to DOM