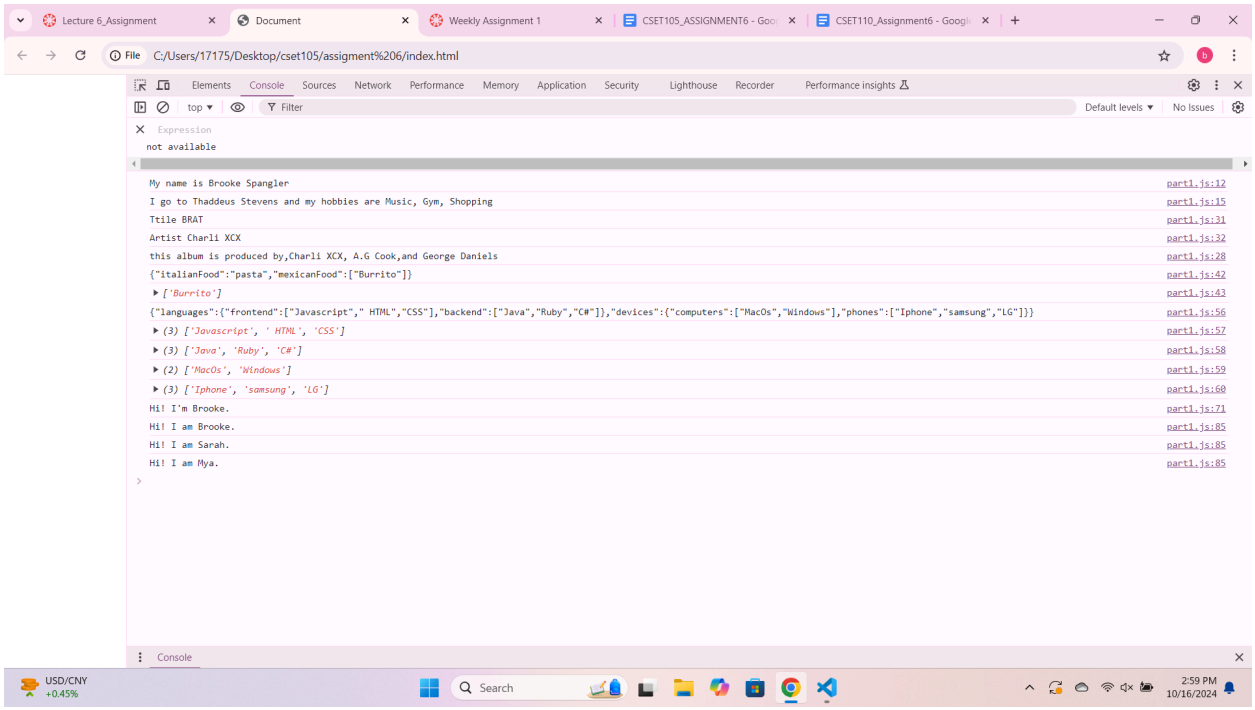
PART 1- 2 OBJECT, 2 JSON, 2 OOP

```
/*2 JS Objects*/
/*JS OBJECT 1*/
var person = {
    name: {
        firstName: "Brooke",
        lastName: "Spangler",
    },
    age: 18,
    school: "Thaddeus Stevens",
    hobby: ["Music", "Gym", "Shopping"],
    greet: function() {
        console.log("My name is " + this.name.firstName + " " +
this.name.lastName);
    },
    full: function() {
        console.log("I go to " + this.school + " and my hobbies are " +
this.hobby.join(", "));
    }
};

person.greet();
person.full();
/*JS OBJECT 2*/
let album={
    title:"BRAT",
    singer: "Charli XCX",
    songs: 15,
    producers: ["Charli XCX"," A.G Cook", "and George Daniels" ],
    play: function(){
        console.log("this album is produced by," + this.producers);
    }
};
console.log("Ttile", album.title);
console.log("Artist", album.singer);
album.play();

/*2 JS JSON*/

/*JSON 1*/
```

```
let string= JSON.stringify({
    "italianFood": "pasta",
    "mexicanFood": ["Burrito"]
});
console.log(string);
console.log(JSON.parse(string).mexicanFood);

/*JSON 2*/
let IT = JSON.stringify({
    "languages":{
        "frontend": ["Javascript", " HTML", "CSS"],
        "backend": ["Java", "Ruby", "C#"]
    },
   "devices":{
        "computers":["MacOs", "Windows",],
        "phones": ["Iphone", "samsung", "LG" ]
   }
});
console.log(IT);
console.log(JSON.parse(IT). languages.frontend);
console.log(JSON.parse(IT). languages.backend);
console.log(JSON.parse(IT). devices.computers);
console.log(JSON.parse(IT). devices.phones);

/*2 JS OOP*/

/*JS OOP 1*/
class Human {
    constructor(name) {
        this.name = name;
    }

    greeting() {
        console.log('Hi! I\'m ' + this.name + '.');
    }
}

let p = new Human("Brooke");
p.greeting();
```

```
/*JS OOP 2*/
class randomPerson {
    constructor(name) {
        this.name = name;
    }

    introduce() {
        console.log(`Hi! I am ${this.name}.`);
    }
}

let names = ["Brooke", "Sarah", "Mya"];
names.forEach(name => new randomPerson(name).introduce());
```

Output:



MATRIX CALCULATOR:

```
class Matrix2x2 {
    constructor(values = [1, 0, 0, 1]) {
      if (values.length !== 4) {
        throw new Error("Matrix must be initialized with an array of 4
elements.");
```

```
    }
    this.values = values;
  }

  get(x, y) {
    if (x < 0 || x > 1 || y < 0 || y > 1) {
      throw new Error("Index out of bounds for a 2x2 matrix.");
    }
    return this.values[x * 2 + y];
  }

  scalarMultiply(scalar) {
    return new Matrix2x2(this.values.map(val => val * scalar));
  }

  add(other) {
    if (!(other instanceof Matrix2x2)) {
      throw new Error("Can only add another Matrix2x2.");
    }
    return new Matrix2x2(this.values.map((val, i) => val +
other.values[i]));
  }

  subtract(other) {
    if (!(other instanceof Matrix2x2)) {
      throw new Error("Can only subtract another Matrix2x2.");
    }
    return new Matrix2x2(this.values.map((val, i) => val -
other.values[i]));
  }

  determinant() {
    const [a, b, c, d] = this.values;
    return a * d - b * c;
  }

  inverse() {
    const det = this.determinant();
    if (det === 0) {
      throw new Error("Matrix is singular and cannot be inverted.");
```

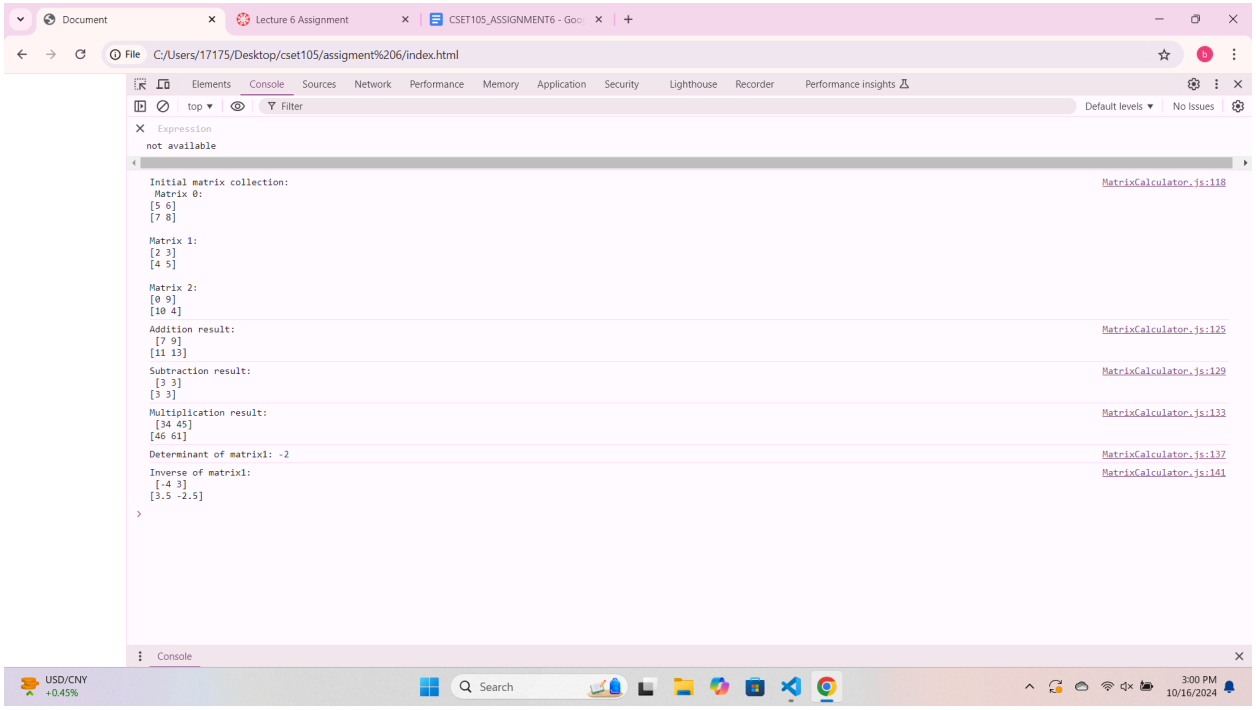
```
      }
      const [a, b, c, d] = this.values;
      return new Matrix2x2([d / det, -b / det, -c / det, a / det]);
    }

    multiply(other) {
      if (!(other instanceof Matrix2x2)) {
        throw new Error("Can only multiply by another Matrix2x2.");
      }
      const [a1, b1, c1, d1] = this.values;
      const [a2, b2, c2, d2] = other.values;
      return new Matrix2x2([
        a1 * a2 + b1 * c2, a1 * b2 + b1 * d2,
        c1 * a2 + d1 * c2, c1 * b2 + d1 * d2
      ]);
    }

    toString() {
      return `[${this.get(0, 0)} ${this.get(0, 1)}]\n[${this.get(1, 0)}
${this.get(1, 1)}]`;
    }
  }

  class MatrixCollection {
    constructor() {
      this.matrices = [];
    }

    addMatrix(matrix) {
      if (!(matrix instanceof Matrix2x2)) {
        throw new Error("Only Matrix2x2 objects can be added.");
      }
      this.matrices.push(matrix);
    }

    updateMatrix(index, matrix) {
      if (!(matrix instanceof Matrix2x2)) {
        throw new Error("Only Matrix2x2 objects can be used for
updates.");
      }
```

```
    if (index < 0 || index >= this.matrices.length) {
      throw new Error("Matrix index out of range.");
    }
    this.matrices[index] = matrix;
  }

  removeMatrix(index) {
    if (index < 0 || index >= this.matrices.length) {
      throw new Error("Matrix index out of range.");
    }
    this.matrices.splice(index, 1);
  }

  getMatrix(index) {
    if (index < 0 || index >= this.matrices.length) {
      throw new Error("Matrix index out of range.");
    }
    return this.matrices[index];
  }

  toString() {
    return this.matrices.map((matrix, i) => `Matrix
${i}:\n${matrix.toString()}`).join("\n\n");
  }
}

// Example usage:

const matrix1 = new Matrix2x2([5, 6, 7, 8]);
const matrix2 = new Matrix2x2([2, 3, 4, 5]);
const matrix3 = new Matrix2x2([0, 9, 10, 4]);


const collection = new MatrixCollection();
collection.addMatrix(matrix1);
collection.addMatrix(matrix2);
collection.addMatrix(matrix3);

console.log("Initial matrix collection:\n", collection.toString());
```

```
const m1 = collection.getMatrix(0);
const m2 = collection.getMatrix(1);

// addition
const resultAdd = m1.add(m2);
console.log("Addition result:\n", resultAdd.toString());

//subtraction
const resultSubtract = m1.subtract(m2);
console.log("Subtraction result:\n", resultSubtract.toString());

//multiplication
const resultMultiply = m1.multiply(m2);
console.log("Multiplication result:\n", resultMultiply.toString());

//determinant
const det = m1.determinant();
console.log("Determinant of matrix1:", det);

//inverse
const inv = m1.inverse();
console.log("Inverse of matrix1:\n", inv.toString());
```

Reflection: Without OOP, I could have used a bunch of functions, which would make everything more complicated, because the OOP has methods that are already stored into the computer, making it shorter. You can group data, methods/functions in the OOP, so it makes the code more organized and neatly-structured.

Output:



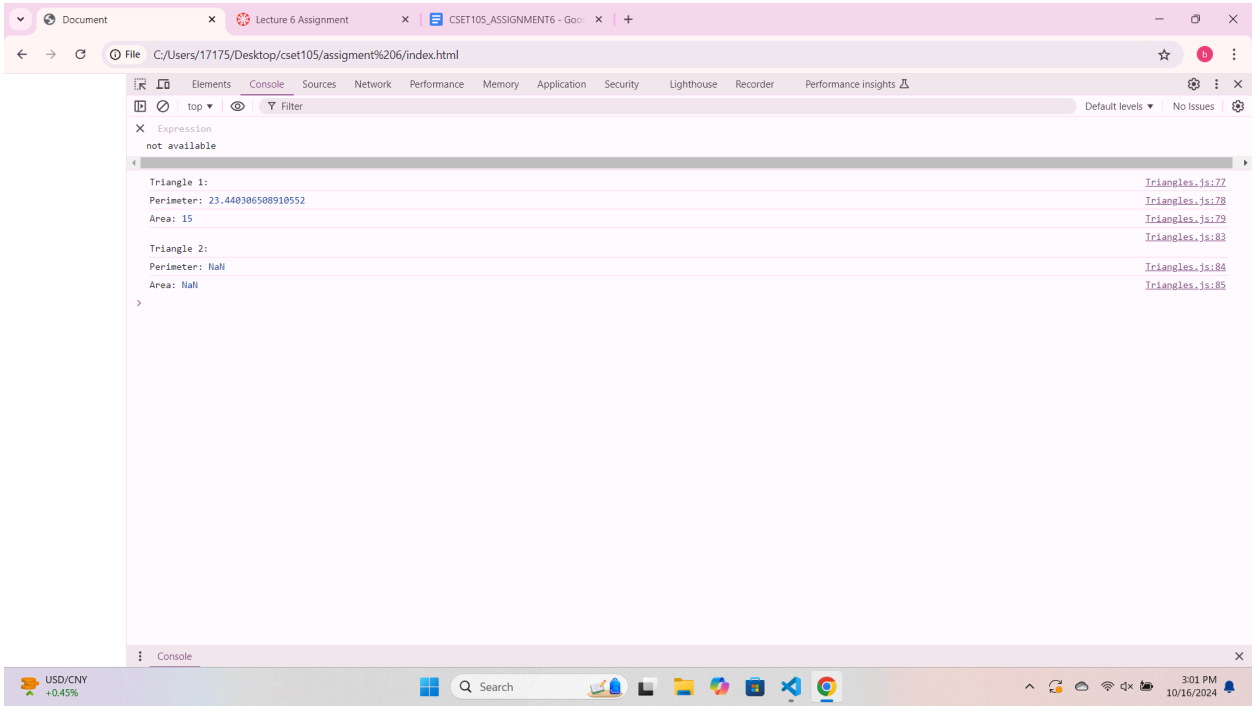
TRIANGLE

```
// 2 Triangles
class Point {
    constructor(x, y) {
      this.x = x;
      this.y = y;
    }
  }
  class Line {
    constructor(startPoint, endPoint) {
      if (!(startPoint instanceof Point) || !(endPoint instanceof Point))
{
        throw new Error("Both startPoint and endPoint must be instances of
the Point class.");
      }
      this.startPoint = startPoint;
      this.endPoint = endPoint;
```

```
    }

    // distance formula
    getLength() {
      const dx = this.endPoint.x - this.startPoint.x;
      const dy = this.endPoint.y - this.startPoint.y;
      return Math.sqrt(dx * dx + dy * dy);
    }
  }
  class Triangle {
    constructor(pointA, pointB, pointC) {
      if (!(pointA instanceof Point) || !(pointB instanceof Point) ||
!(pointC instanceof Point)) {
        throw new Error("All points must be instances of the Point
class.");
      }
      this.pointA = pointA;
      this.pointB = pointB;
      this.pointC = pointC;

 // the three sides of the triangle
      this.sideAB = new Line(this.pointA, this.pointB);
      this.sideBC = new Line(this.pointB, this.pointC);
      this.sideCA = new Line(this.pointC, this.pointA);
    }

   // will check if it is a valid or invalid triangle.
   /*will check if the largest side's length is smaller
   than the other two lengths of the sides combined. */
    isValidTriangle() {
      const lengthAB = this.sideAB.getLength();
      const lengthBC = this.sideBC.getLength();
      const lengthCA = this.sideCA.getLength();

      return (lengthAB + lengthBC > lengthCA) &&
             (lengthBC + lengthCA > lengthAB) &&
             (lengthCA + lengthAB > lengthBC);
    }

    // Method for perimeter
```

```
    getPerimeter() {
      if (!this.isValidTriangle()) {
        return NaN;
      }

      return this.sideAB.getLength() + this.sideBC.getLength() +
this.sideCA.getLength();
    }

    // Shoelace formula
    getArea() {
      if (!this.isValidTriangle()) {
        return NaN;
      }

      const x1 = this.pointA.x, y1 = this.pointA.y;
      const x2 = this.pointB.x, y2 = this.pointB.y;
      const x3 = this.pointC.x, y3 = this.pointC.y;

      return Math.abs((x1 * y2 + x2 * y3 + x3 * y1 - y1 * x2 - y2 * x3 -
y3 * x1) / 2);
    }
  }

// Triangle 1: Valid triangle
const triangle1 = new Triangle(new Point(0, 0), new Point(10, 0), new
Point(0, 3));
console.log("Triangle 1:");
console.log("Perimeter:", triangle1.getPerimeter());
console.log("Area:", triangle1.getArea());

// Triangle 2: Invalid triangle (collinear points)
const triangle2 = new Triangle(new Point(0, 0), new Point(2, 0), new
Point(800, 0));
console.log("\nTriangle 2:");
console.log("Perimeter:", triangle2.getPerimeter());
console.log("Area:", triangle2.getArea());
```

Output :


Document
Lecture 6 Assignment
CSET105_ASSIGNMENT6 - Goo
File
C:/Users/17175/Desktop/cset105/assigment%206/index.html
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
top
Filter
Default levels
No Issues
Expression
not available
Triangle 1:
Triangles.js:77
Perimeter: 23.440306508910552
Triangles.js:78
Area: 15
Triangles.js:79
Triangles.js:83
Triangle 2:
Perimeter: NaN
Triangles.js:84
Area: NaN
Triangles.js:85
Console
USD/CNY
+0.45%
Search
3:01 PM
10/16/2024