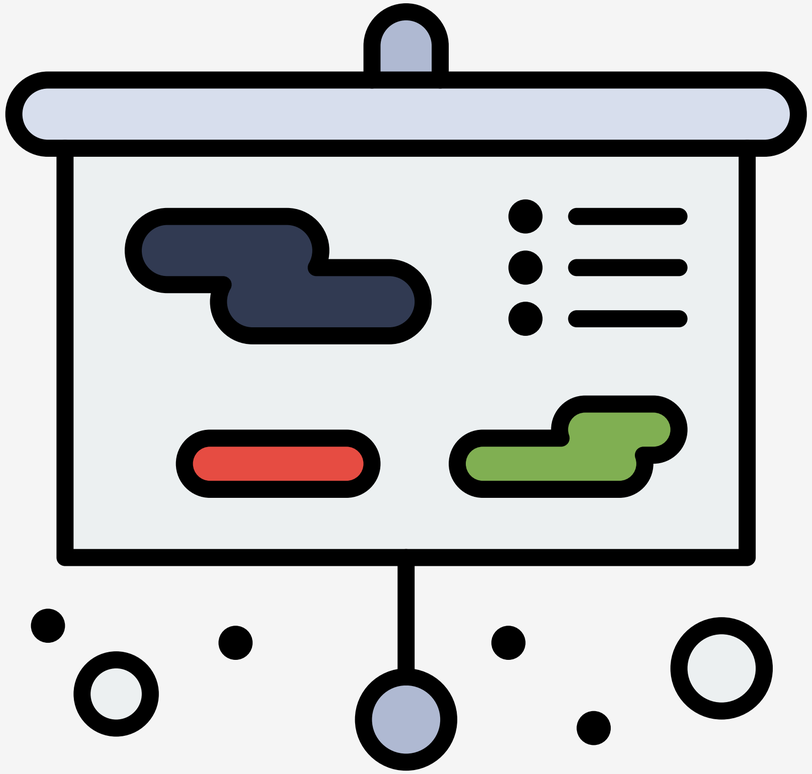
WEB PRODUCTION

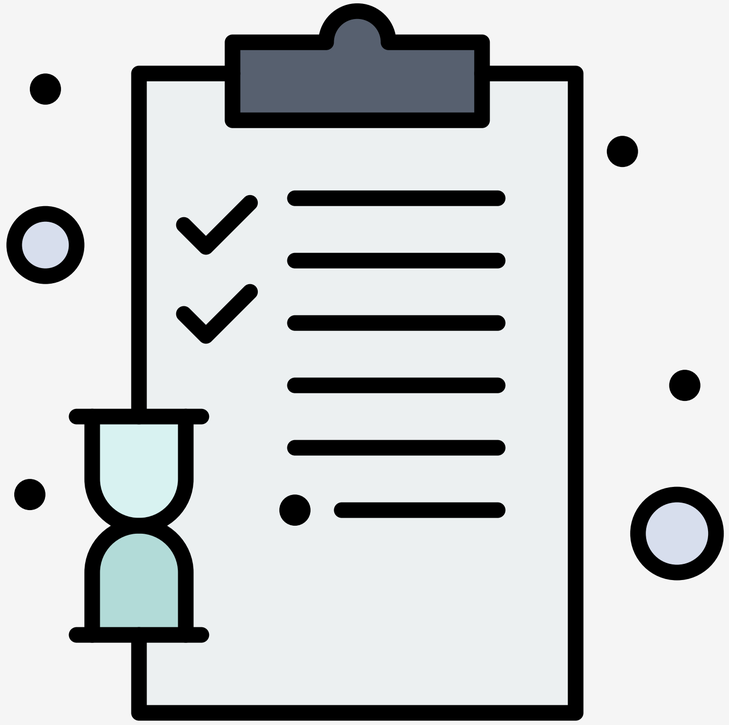
Lecture 1

TODAY'S TOPICS



- Course Introduction
- Install Development Tools
- Review Git & GitHub
- Review JavaScript
- Participation: Hybrid #1
- Exercise: Hearts

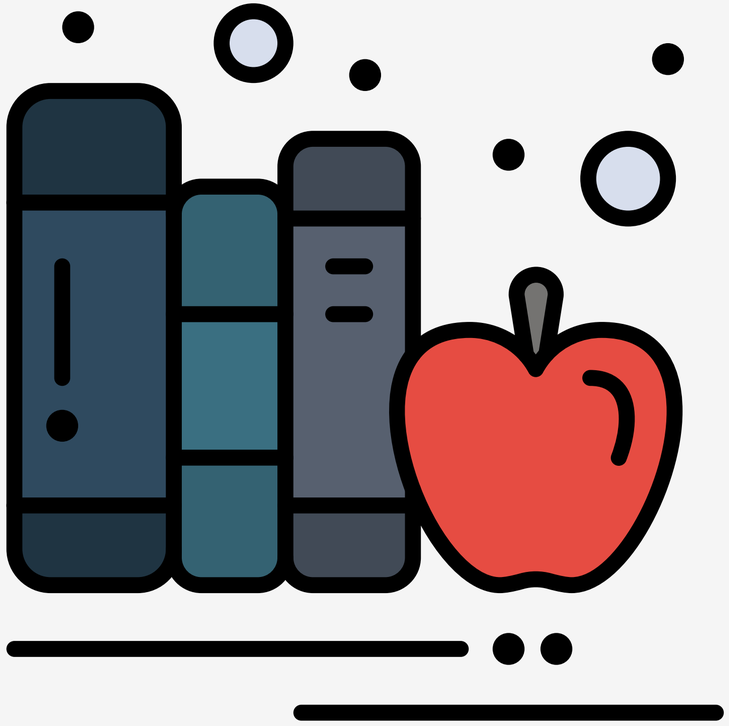
ANNOUNCEMENTS



- Sign-in Sheet
- Recordings

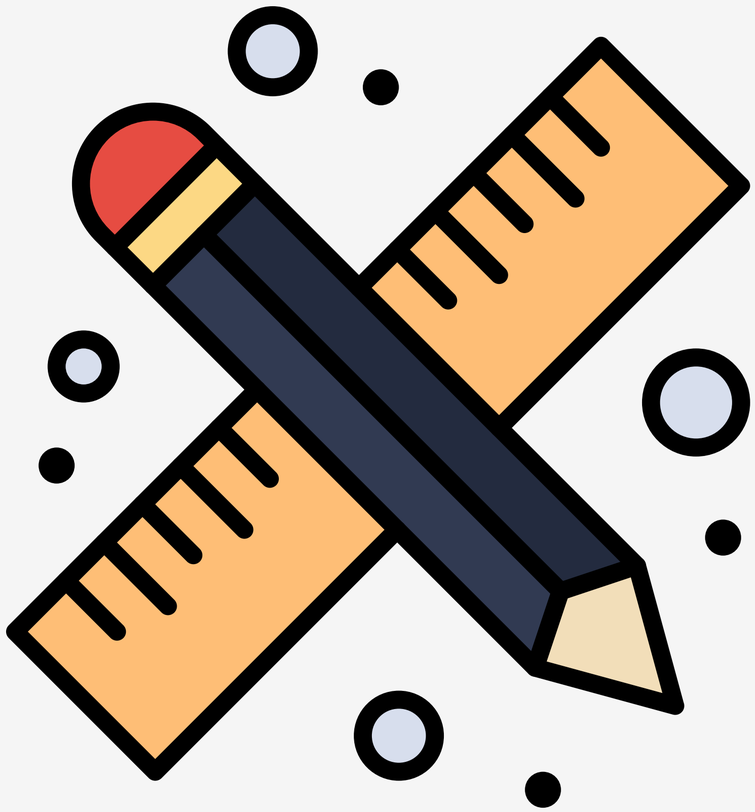
COURSE INTRODUCTION

COURSE TOPICS



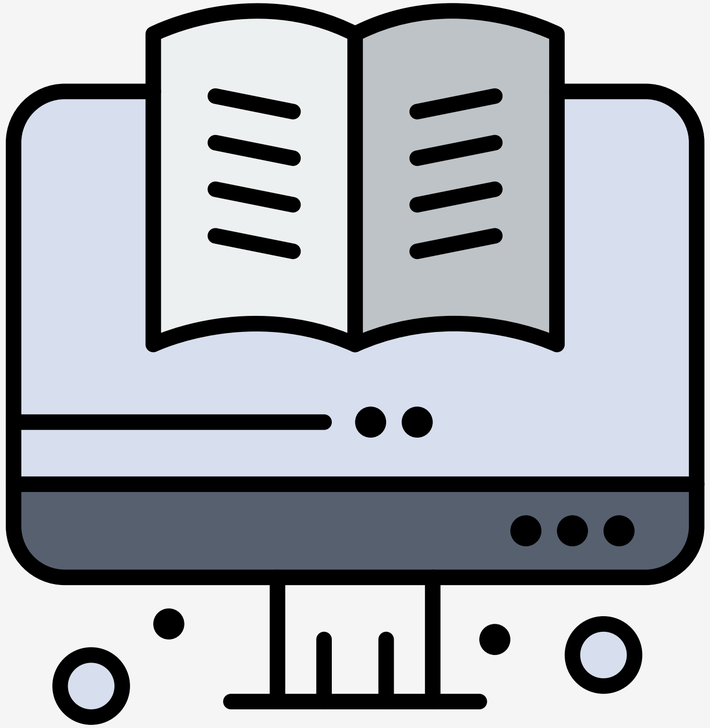
- Vue Framework
- Vue CLI
- Vue Router
- Axios
- Vuex
- Firebase

ASSIGNMENTS



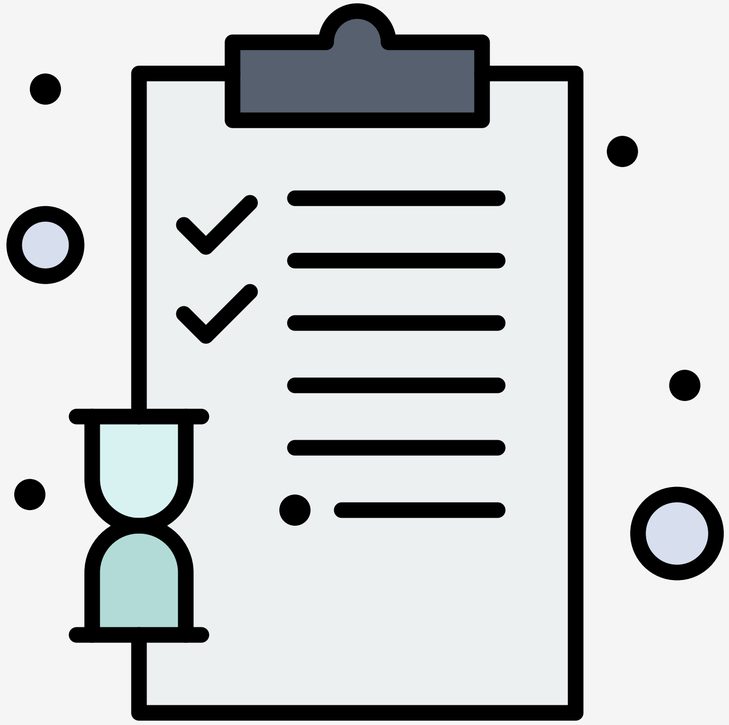
- 10 - 12 Participation (20%)
- 5 Exercises (30%)
- Midterm Project (25%)
- Final Project (25%)

COURSE CONTENT



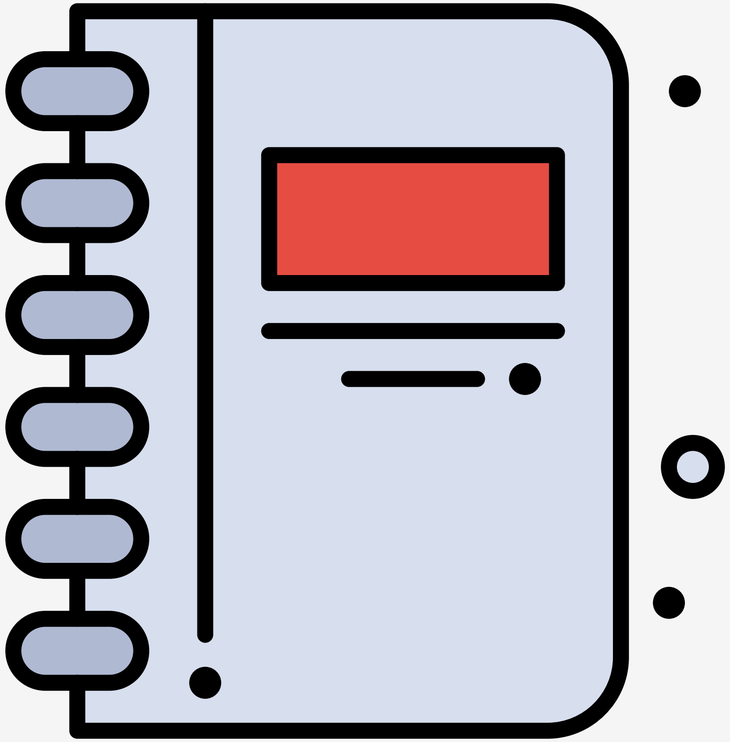
- IMDAC Website is use for content
- BrightSpace is used for submission and grades
- GitHub Classroom for submission

COURSE STRUCTURE



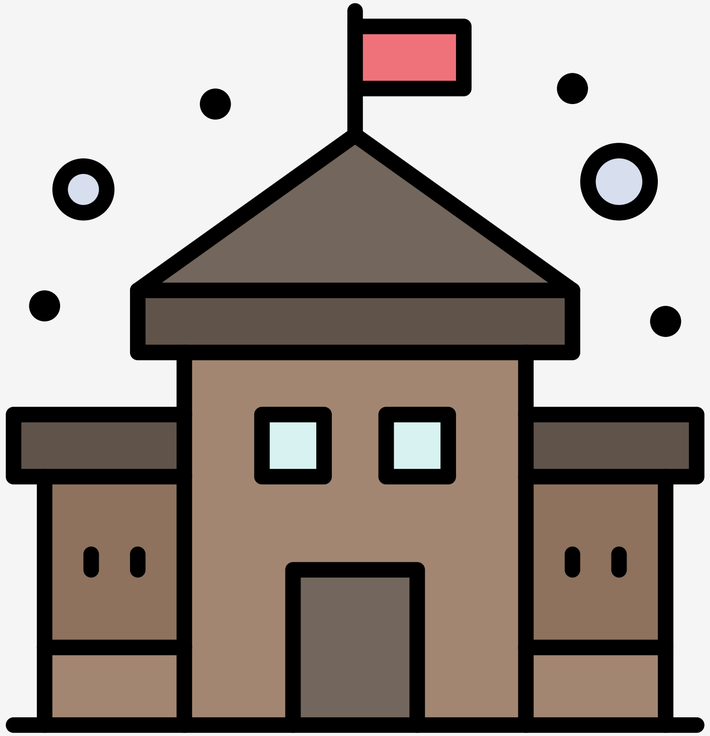
- 13 weeks (*No class on Family Day*)
- 3hrs/week lecture/lab
- 1hr/week online
- Slides and recordings will be made available

CLASS TIMES



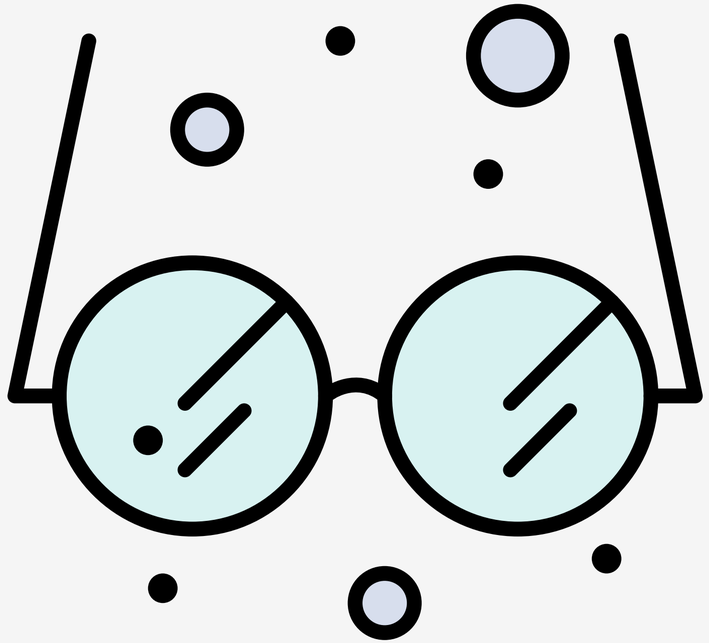
- **Section 310:**
  Mon 1:30 PM - 4:30 PM (T227)

STUDENT EXPECTATIONS



- Do the work
- Do your own work
- Don't be late
- Be respectful

PROFESSOR EXPECTATIONS



- Provide accurate and timely information
- Be flexible to the needs of the class
- Respond to emails within 48 hours
- Provide feedback within 2 week
- Fair and unbiased grading

PLAGIARISM & REFERENCING CODE

- Plagiarism is submitting someone else's work as your own *WITHOUT* proper reference
- Getting *ANY* code from online resource is considered plagiarism
- Sending or receiving code from a friend or classmate is considered plagiarism
- Working together on a project *MAY* fall under plagiarism

PLAGIARISM & REFERENCING CODE

- Any code that is not entirely your own should be referenced
- ONLINE: Include a description of what the code does and the source URL
- PERSON: Include a description of what the code does and the name of the person and when help was received
- EXCEPTION: Any code provided in class or in course content can be used without reference

INSTALL DEVELOPMENT TOOLS

GIT & GITHUB

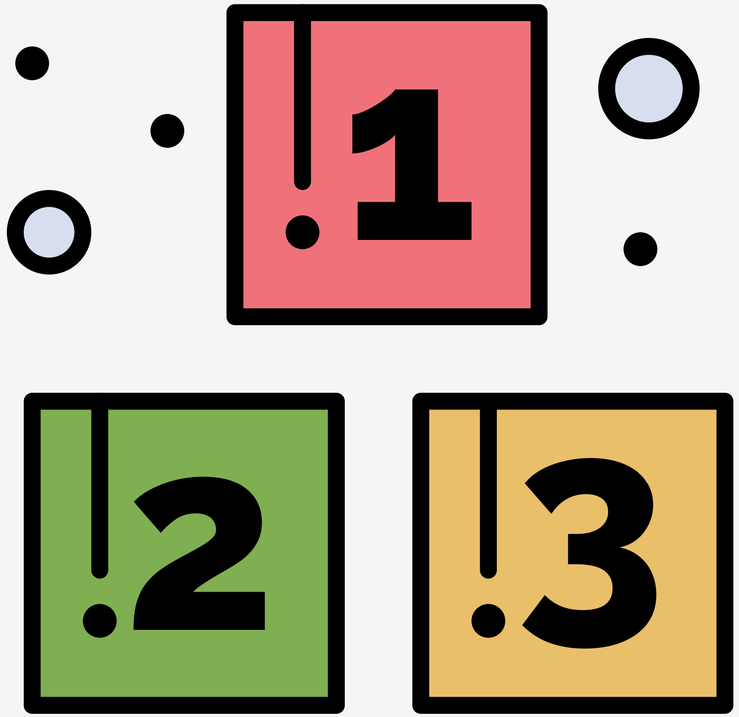
GIT & GITHUB

- Review Git & GitHub Basics
- GitHub Classroom

JAVASCRIPT REVIEW

ARRAYS

JAVASCRIPT ARRAYS

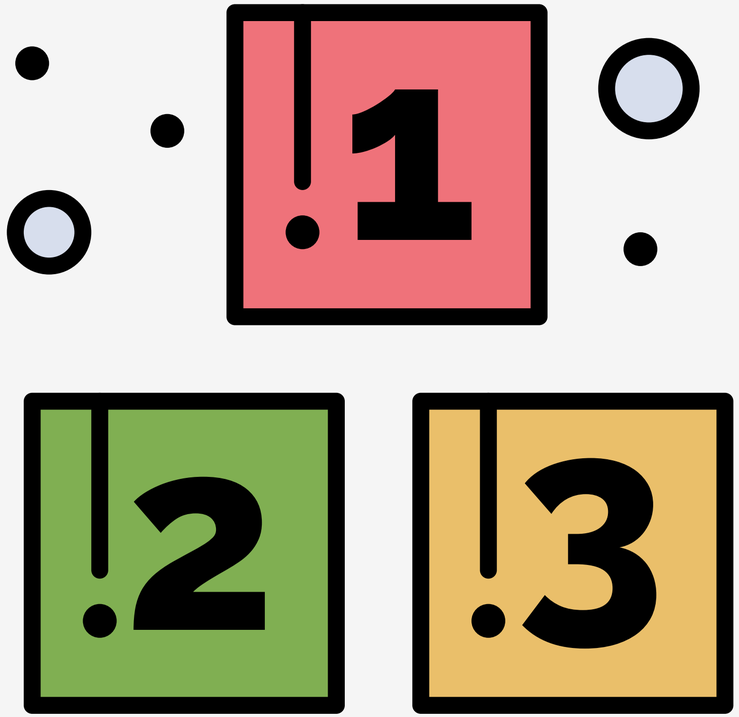


- An array is a comma separated list of items
- Each item is assigned a numbered index, which starts with `0`
- Values are retrieved using Bracket Notation
- Items can be updated using bracket notation

```
// Creates an array of colors
const colors = ['blue', 'green', 'yellow', 'red']

// Logging the FIRST color
console.log(colors[0]) // blue

// Logging the THIRD color
console.log(colors[2]) // yellow

// Logging the FIFTH color
console.log(colors[4]) // undefined
```

```
// Creates a groceries list
const groceries = ['Milk', 'Eggs']

// Replaces the FIRST item
groceries[0] = 'Juice'

// Add an item
groceries[3] = 'Bread' Bad!
```

ARRAY MANIPULATION



- Use `push()` to add items to the end of the array
- Use `pop()` to remove the last item in the array
- Use `shift()` to remove the first item in the array
- Use `unshift()` to add items to the beginning of the array
- Use `splice()` to add and / or remove items

```
// Creates a groceries list
const groceries = ['Milk', 'Eggs']

// Add an item
groceries.push('Bread')

// Add multiple items
groceries.push('Sugar', 'Flour')

// ['Milk', 'Eggs', 'Bread', 'Sugar', 'Flour']
```

```
// Creates a groceries list
const groceries = ['Milk', 'Eggs']

// Remove the last item
groceries.pop()

// ['Milk']
```

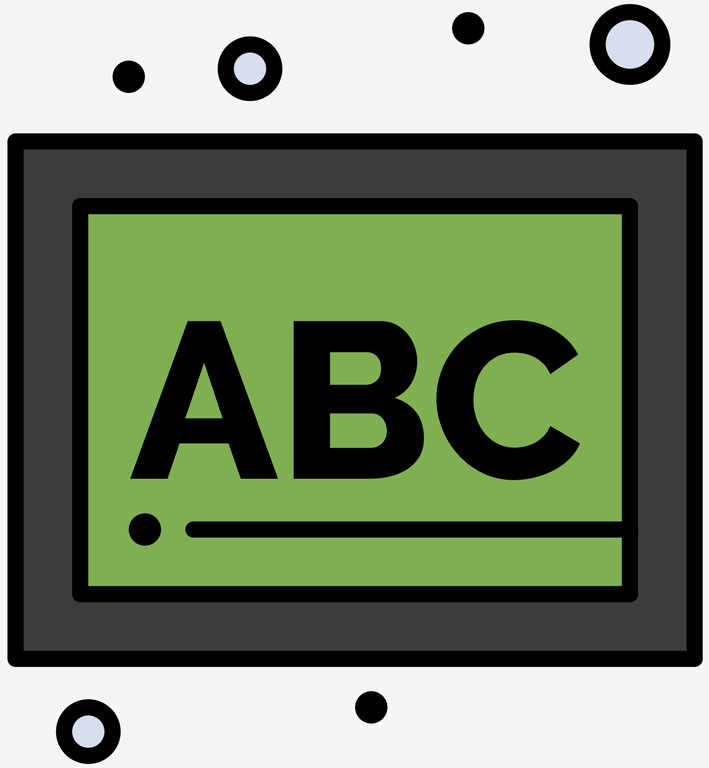
```
// Creates a groceries list
const groceries = ['Milk', 'Eggs']

// Remove the first item
groceries.shift()

// ['Eggs']
```

```
// Creates a groceries list
const groceries = ['Milk', 'Eggs']

// Add an item
groceries.unshift('Bread')

// Add multiple items
groceries.unshift('Sugar', 'Flour')

// ['Sugar', 'Flour', 'Bread', 'Milk', 'Eggs']
```

```
// Creates a groceries list
const groceries = ['Milk', 'Eggs', 'Bread']

// Add an item
groceries.splice(1, 0, 'Sugar')

// Remove an item
groceries.splice(2, 1)

// Add and Remove items
groceries.splice(0, 2, 'Flour')

// ['Flour', 'Bread']
```

OBJECTS

JAVASCRIPT OBJECTS



- An object is a collection of properties
- These key / values pairs are called properties
- Values can be retrieved using Bracket or Dot notation
- Properties can be added, updated, or deleted

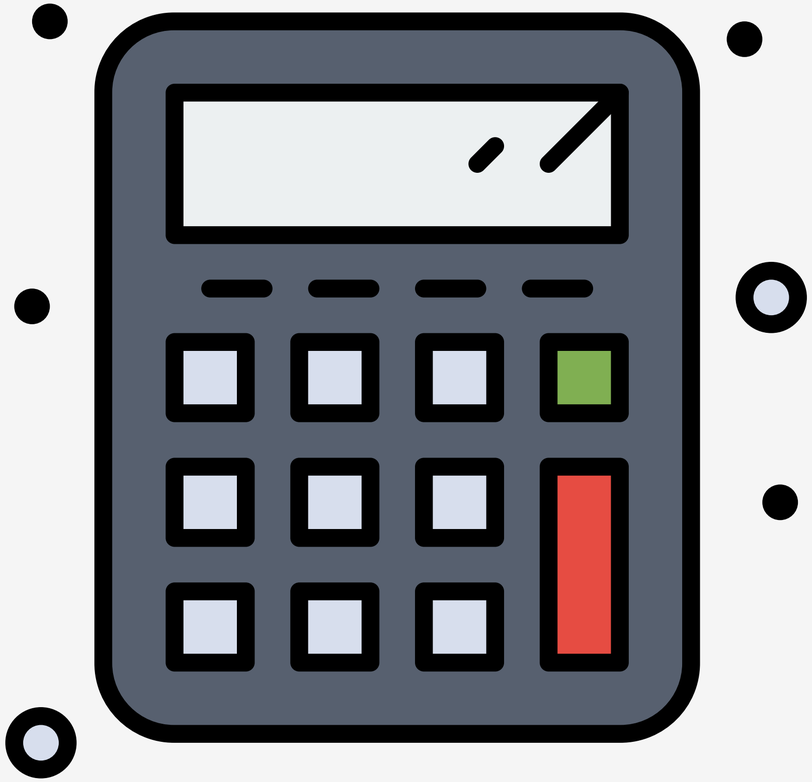
```
const car = {
  year: 2019,
  make: 'Toyota',
  model: 'Prius'
}

// dot notation
car.make // 'Toyota'

// bracket notation
car['model'] // 'Prius'
```

```
const car = {
  year: 2019,
  make: 'Toyota',
  model: 'Prius'
}

// Update properties
car.make = 'Telsa'
car['model'] = 'Model 3'

// Add properties
car.color = 'Red'
car['range'] = 500
```

```
const car = {
  year: 2019,
  make: 'Toyota',
  model: 'Prius'
}

// Remove properties
delete car.make
delete car['model']
```

IF...ELSE

IF...ELSE STATEMENTS

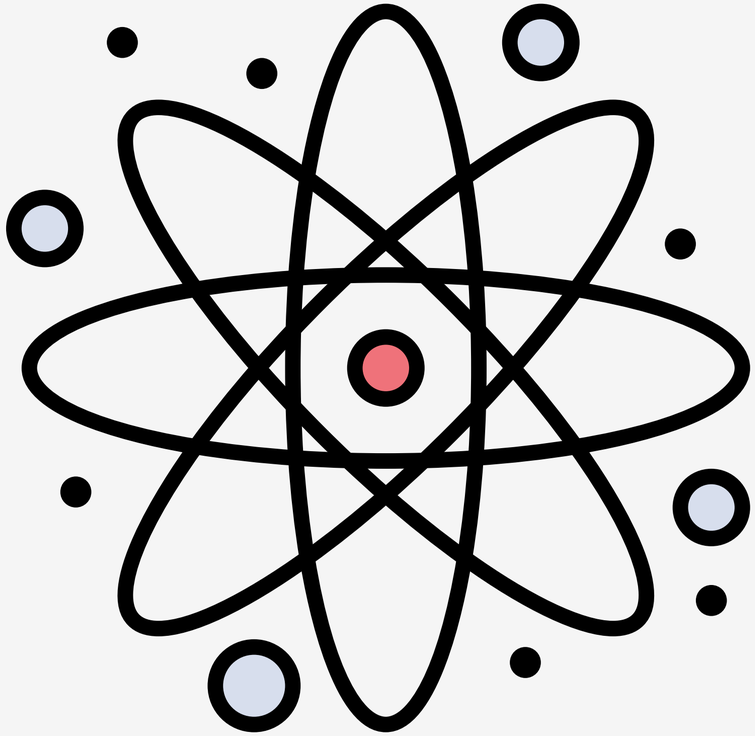


- Consists of the `if` keyword, a condition, and a statement
- If the condition is truthy, the statement will execute
- The `else` statement is use to execute an optional statement if the condition of the previous `if` statement is falsy
- The `else if` clause can be used to check for multiple conditions

```
// declare the number variable
const number = 6

if (number === 5) {
  // this block of code will NOT execute
  console.log('Yes, number is equal to 5')
} else if (number === 6) {
  // this block of code will execute
  console.log('Yes, number is equal to 6')
} else if (number === 7) {
  // this block of code will NOT execute
  console.log('Yes, number is equal to 7')
}
```

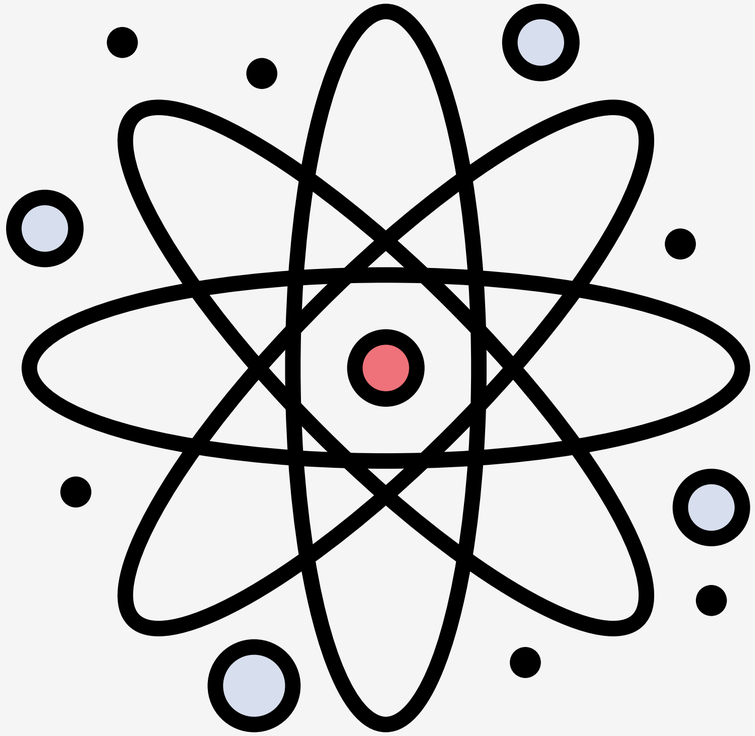
LOOPS

LOOPS



- Loops are statements that are used to repeat a block of code until a specified condition is met.
- JavaScript has following loops:
  - `for`
  - `for...of`
  - `for...in`
  - `while`

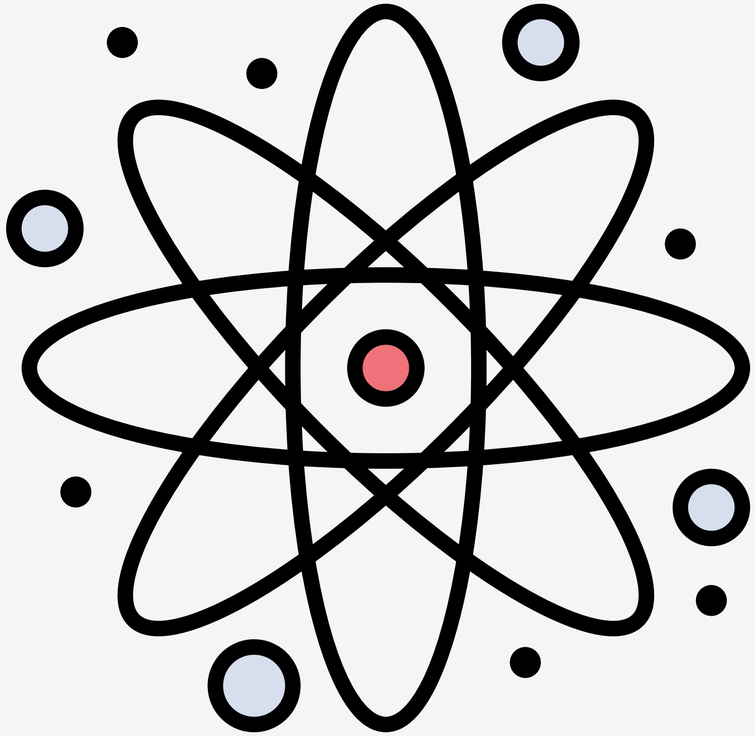
FOR LOOP



- The `for` loop is used when the number of iterations is knowable
- Consists of three expressions separated by semi-colons
    1. The initialization of the iterator
    2. The condition that is check before each loop to see if the loop should continue
    3. The iteration of the iterator

```javascript
// will loop 5 times
for (let i = 0; i < 5; i++) {
  console.log(i) // Logs 0 to 4
}

// iterating over an array
const animals = ['cat', 'dog', 'mouse']

for (let i = 0; i < animals.length; i++) {
  // Logs all the animals in the array
  console.log(animals[i])
}
```

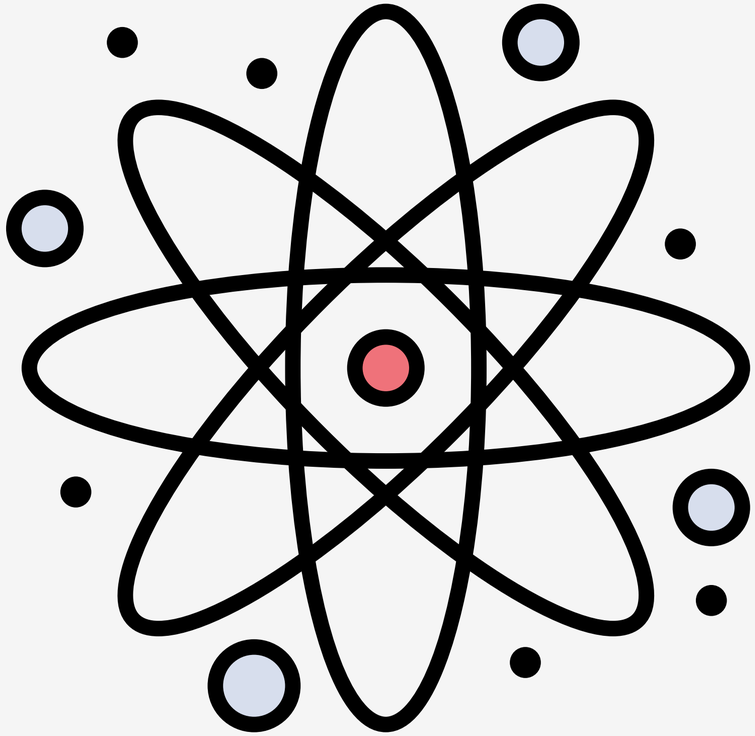
FOR...OF LOOP



- The `for...of` loop is used iterate over iterable objects, like strings and arrays
- The expression starts with the initialization of a variable, which hold the value of each item
- This followed by the `of` keyword
- The expression ends with the iterable object

```
// iterating over an array
const animals = ['cat', 'dog', 'mouse']

for (const animal of animals) {
  // Logs all the animals in the array
  console.log(animal)
}

// iterate over a string
const name = 'Ted Mosby'

for (const char of name) {
  // Logs each character of the name
  console.log(char)
}
```

FOR...IN LOOP

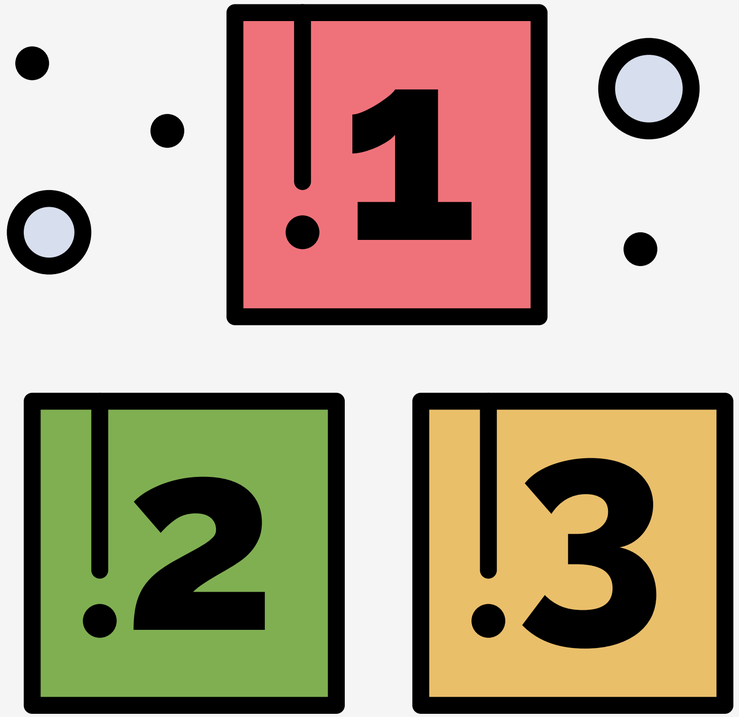


- The `for...in` loop is used iterate over properties of an object
- The expression starts with the initialization of a variable, which hold the key of each property
- This followed by the `in` keyword
- The expression ends with the object
- When retrieving values, bracket notation *MUST* be used

```
// iterate over properties
const sounds = {
  cow: 'moo',
  duck: 'quack',
  horse: 'nay'
}

for (const animal in sounds) {
  // Logs each animals' sound
  console.log(sounds[animal])
}
```

FUNCTIONS

FUNCTIONS

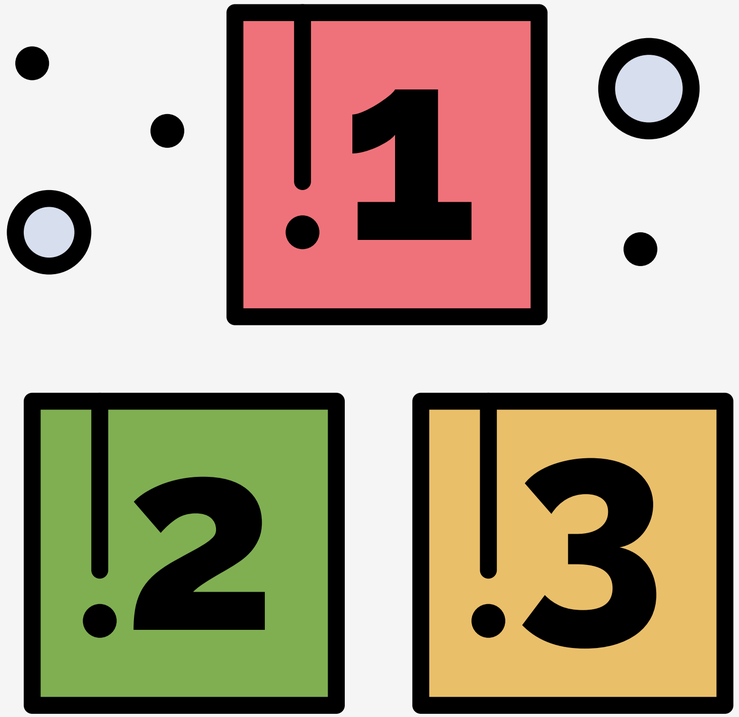


- Function declaration is just one way of defining a function
- Other methods includes function expressions and arrow functions
- The `return` statement is used to end a function and provide the function's value
- Parameters are declared when a function is defined and can be used anywhere inside the function

```
// a function with parameters
function add (a, b) {
  return a + b
}

// invoking with arguments
console.log(add(3, 5)) // 8
```

THE DOM

FINDING DOM ELEMENTS



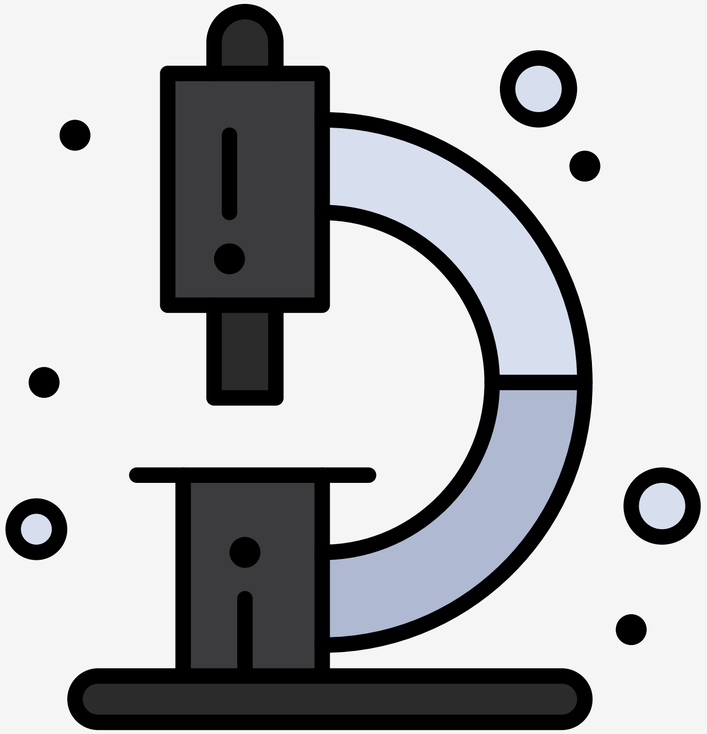
- JavaScript has many methods for retrieving elements from the DOM
- The most common methods are:
  - `getElementById()`
  - `querySelector()`
  - `querySelectorAll()`

```
// finds first element with an id of box
const $box = document.getElementById('box')

console.log($box) // Element Object
```

```
// Find element by tag name
const $header = document.querySelector('header')
console.log($header) // Element Object

// Find element by class name
const $button = document.querySelector('.button')
console.log($button) // Element Object

// Find element by id
const $box = document.querySelector('#box')  Bad!
console.log($button) // Element Object
```

```
// Find elements by class name
const $buttons = document.querySelectorAll('.button')
console.log($buttons) // NodeList

for (const button of $buttons) {
  console.log(button.textContent)
}
```

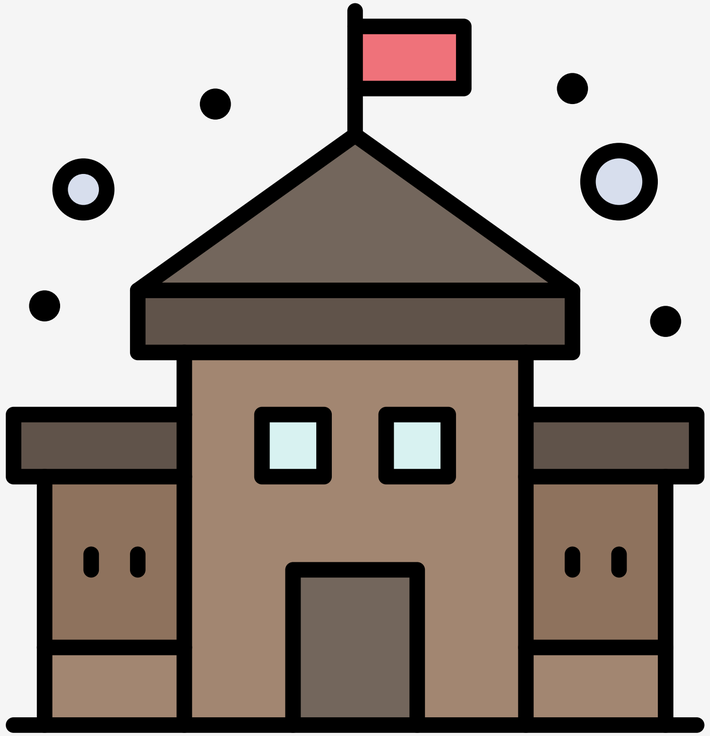
MANIPULATING ATTRIBUTES



- Most standard attributes have a corresponding property in the Element object
- These properties can be access using dot or bracket notation.
- The methods include:
  - `getAttribute()`
  - `setAttribute()`
  - `removeAttribute()`

```
const $link = document.getElementById('link')

// Reading the id attribute
console.log($link.getAttribute('id')) // link
console.log($link.id) // link

// Setting the href attribute
$link.setAttribute('href', 'https://google.ca')
$link.href = 'https://google.ca'

// Removing the target attribute
$link.removeAttribute('target')
```

MANIPULATING CLASSES



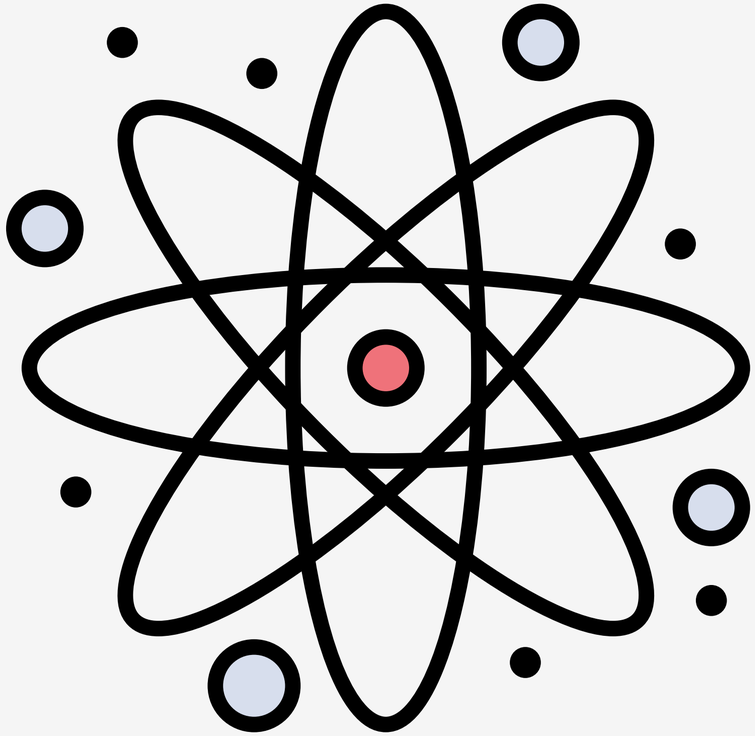
- Manipulating classes is different than other attributes
- There are two properties for manipulating classes `className` and `classList`
- The classList property contains the following methods:
  - `add()`
  - `remove()`
  - `toggle()`

```
const $link = document.getElementById('link')

// Reading the class attribute
console.log($link.className) // link

// Updating the class attribute
$link.className = 'red active'
console.log($link.className) // red active
```

```
const $link = document.getElementById('link')

// Reading the class attribute
console.log($link.className) // link

// Adding classes
$link.classList.add('red', 'active')
console.log($link.className) // link red active

// Removing classes
$link.classList.remove('active')
console.log($link.className) // link red

// Toggling classes
$link.classList.toggle('active')
console.log($link.className) // link red active
```

EVENTS

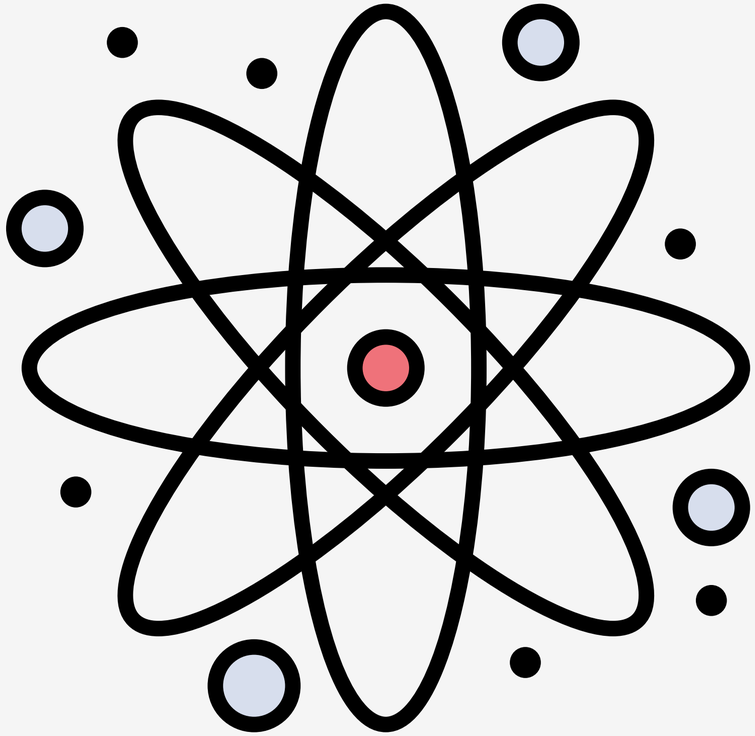
CREATING DOM ELEMENTS



- Utilizes template literals and the `innerHTML` property
- Template literals will be used to create HTML templates
- The `innerHTML` property can be used to read and update the HTML inside of an element

```
const $list = document.getElementById('list')

console.log($list.innerHTML) // <li>Milk</li>

$list.innerHTML += '<li>Eggs</li>'
console.log($list.innerHTML)
// <li>Milk</li><li>Eggs</li>
```

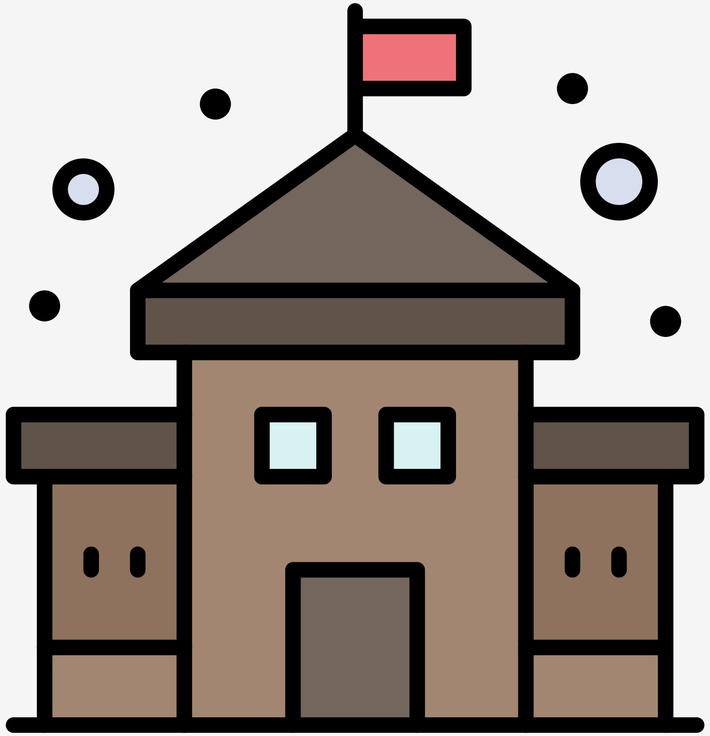
CREATING DOM ELEMENTS



- Retrieving and updating the DOM is process intensive
- Do NOT insert DOM elements inside a loop
- Store all the elements in an array or string
- Add all of them to the DOM at once

```
const $list = document.getElementById('list')

const items = ['milk', 'bread', 'eggs']
const listItems = []

for (const item of items) {
  listItems.push(`<li>${item}</li>`) // add to array
}

$list.innerHTML += listItems.join('') // insert items
```

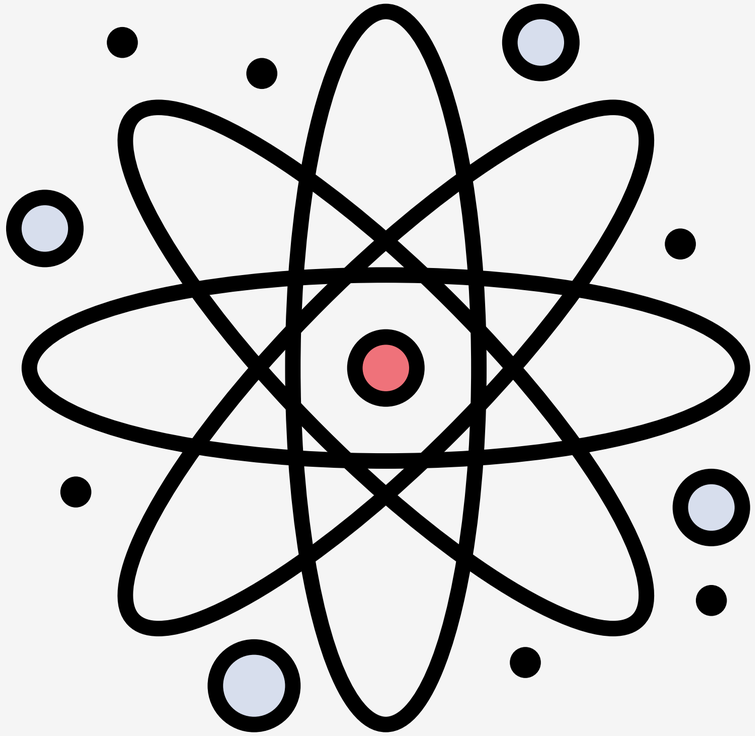
EVENT LISTENERS



- Event Listeners is the preferred method for responding to DOM Events
- Event Listeners are created by using the `addEventListener()` method
- The method takes two arguments, an event type and a function

```
const $button = document.getElementById('button')

// using an anonymous function
$button.addEventListener('click', function () {
  console.log(`The button was clicked`)
})
```

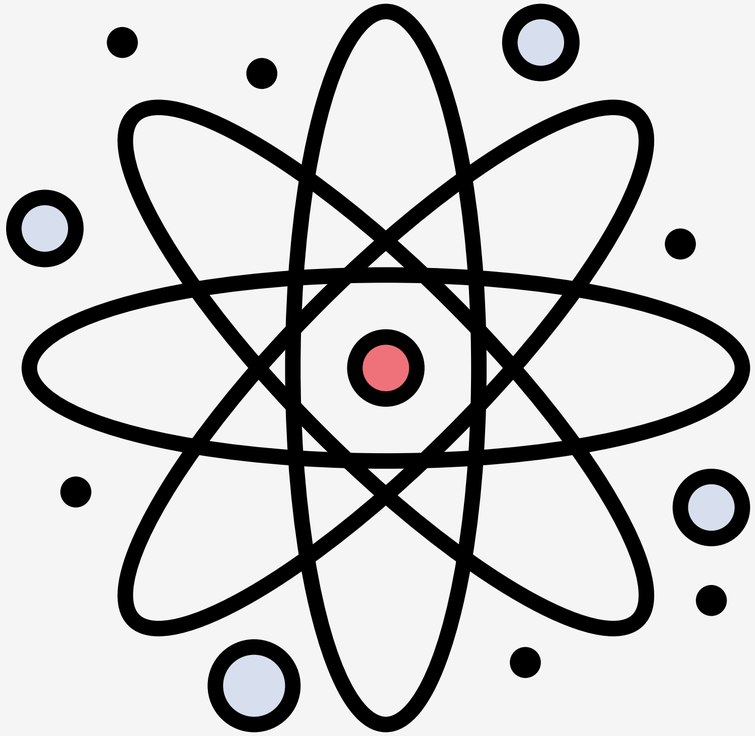
USING A LOOP



- Retrieve all the elements as a NodeList or HTMLCollection.
- Predefined the event handler function
- Use a loop to add the event listener to each element

```
// getting all .button elements
const $buttons = document.querySelectorAll('.button')

// predefine button action
function buttonAction (e) {
  console.log(e.target.textContent)
}

// loop over buttons
// add the event listener
for ($button of $buttons) {
  $button.addEventListener('click', buttonAction)
}
```

EVENT DELEGATION

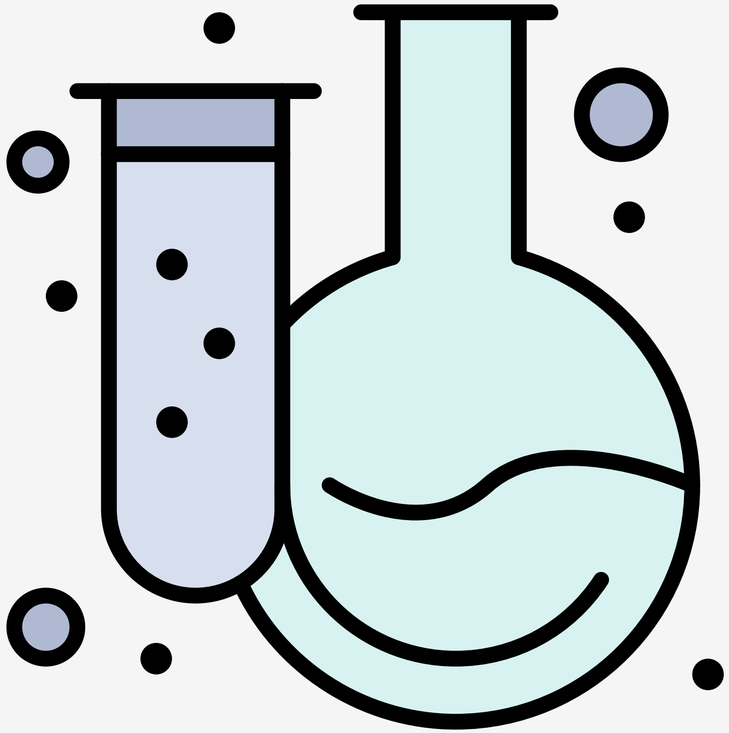


- Utilize the process of event propagation
- Adds event listeners to a parent element of the target element
- Use the `target` property of the Event object to identify which event originated
- Use `classList.contains()` or `closest()` to help identify if `target` is the desired element

```
// parent of the .button elements
const $buttons = document.querySelector('.buttons')

// predefine button action
function buttonAction (e) {
  if (e.target.classList.contains('button')) {
    console.log(e.target.textContent)
  }
}

// use event delegation
$buttons.addEventListener('click', buttonAction)
```

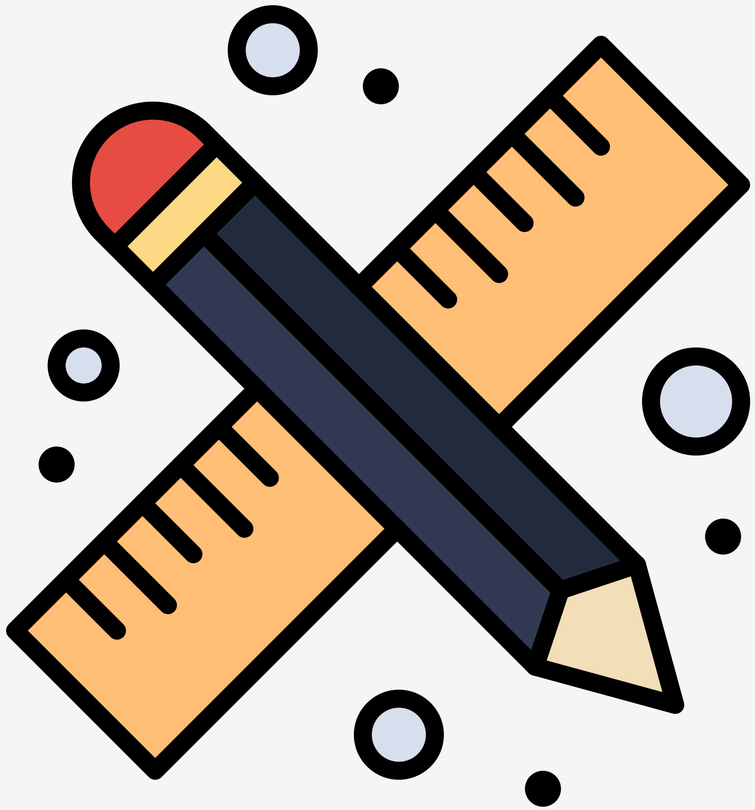
HANDS-ON

HYBRID #1



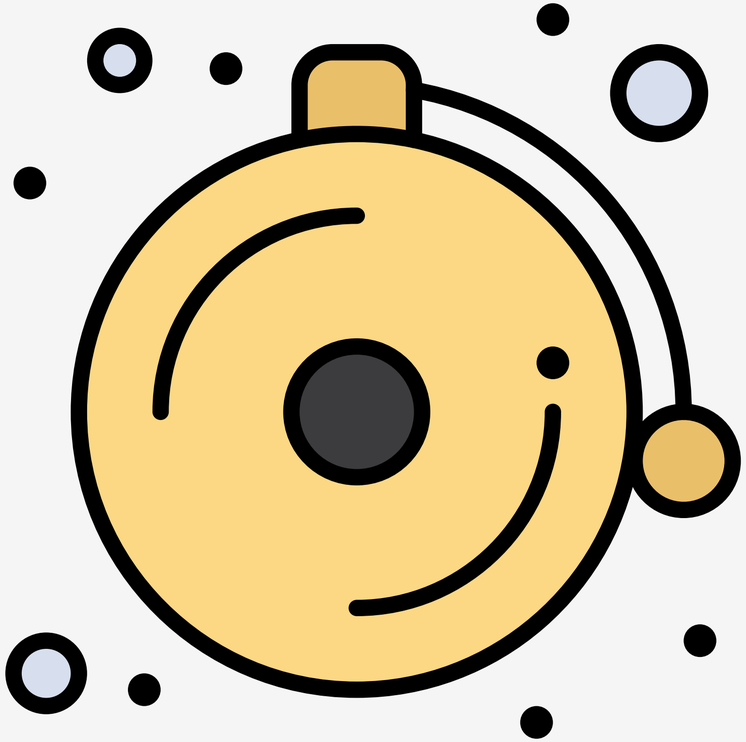
- Watch the first three sections of Learning Vue.js on LinkedIn Learning
- Write 2 to 3 sentences for each section
- *DUE:* **Mon. Jan. 20 @ 11:59 PM**

HEARTS



- *GITHUB CLASSROOM ASSIGNMENT*
- Create a deck of cards and deal out the cards to 4 players
- Create 4 buttons to display each players hand
- Submit the URL to your repository
- *DUE:* **Mon. Jan. 13 @ 11:59 PM**

NEXT TIME...



- Vue Basics
- Participation: Yahtzee
- Exercise: Hearts Vue