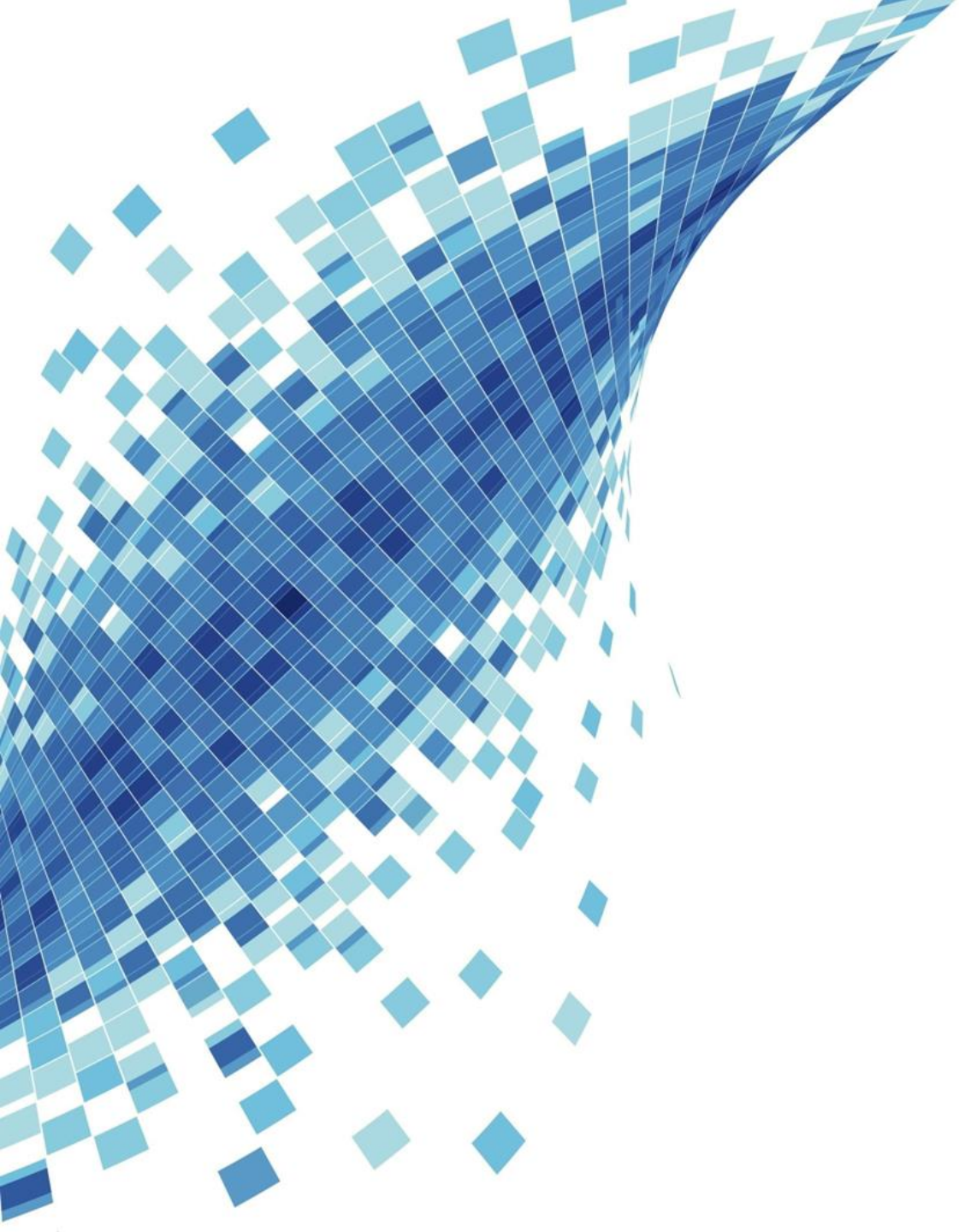
AIF233201 - Pemrograman Berbasis Web

Javascript (2)

by Raymond Chandra Putra
Pascal Alfadian Nugroho
Keenan Adiwijaya Leman

07-10-2024

INFORMATIKA
UNPAR



Basic DOM

INFORMATIKA
UNPAR

Document Object Model (DOM)

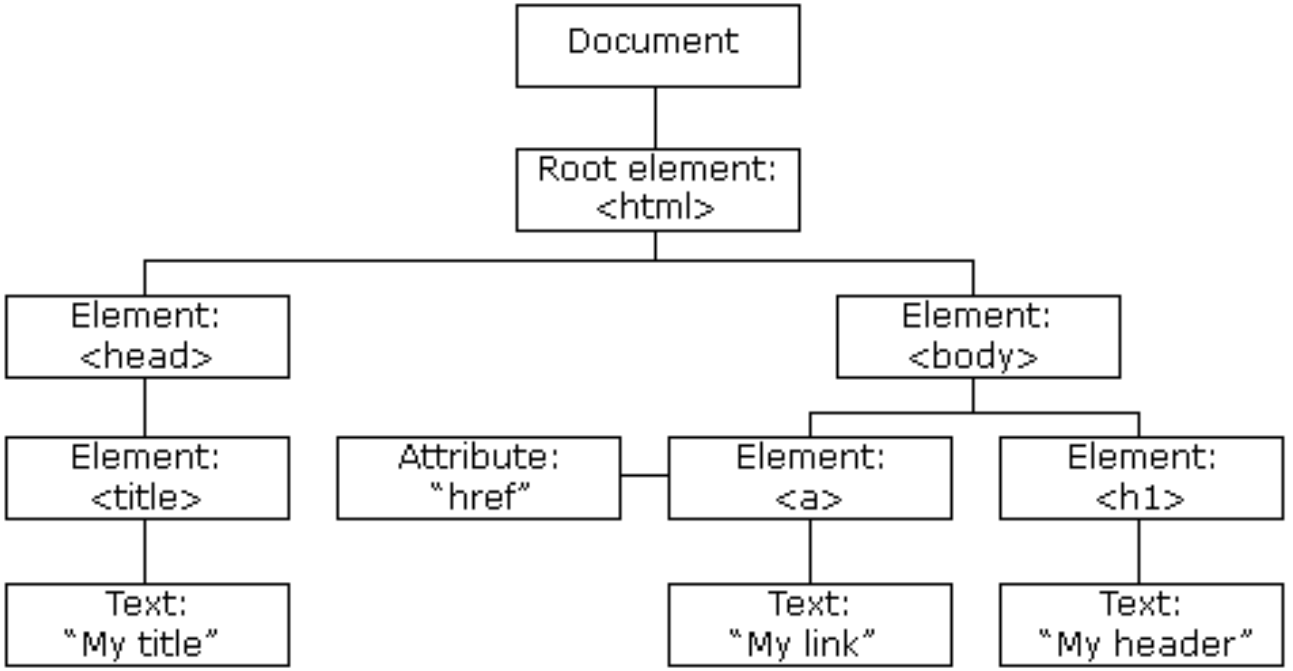
The DOM defines properties and methods to access and change each object in this model

The effect is that what the user sees in the browser window is updated

The **DOM is an API** (Application Programming Interface) - it lets the browser and your JavaScript program or CSS talk to each other

- It states what your script can ask the browser about the current page
- It states how your script can tell the browser to update what is being shown to the user

DOM Tree



INFORMATIKA
UNPAR

Find HTML Element

- Several ways to find HTML element(s):
  - by id

```
document.getElementById("intro"); //one
```

  - by class name

```
document.getElementsByClassName("intro"); //many
```

  - by tag name

```
document.getElementsByTagName("p"); //many
```

  - by selector

```
document.querySelector("p#intro"); //one
document.querySelectorAll("li.menu"); //many
```

Note: one = element, many = array of elements

Get Element

- Several ways to find HTML element(s):

```
<div>
   <a id="mylink" href="http://mysite.com">Click me</a>
   <a href="http://othersite.com">Or me<a/>
</div>
<script>
   let el = document.getElementById("mylink");
   // OR
   let el = document.querySelector("#mylink");
   // OR
   let parent = document.querySelector("div");
   let el = parent.querySelector("#mylink");
</script>
```

Note: document ← root DOM

Get Element Data

- Get element first, then get the data via element attributes.

```
<div>
   <a id="mylink" href="http://mysite.com">Click me</a>
   <a href="http://othersite.com">Or me<a/>
</div>
<script>
   // 1 element
   let url = document.querySelector("#mylink").href;
   console.log('The URL is ' + url);

   // n elements
   let links = document.getElementsByTag("a");
   url = links[1].href;
</script>
```

Modify Element Content

- To modify the **content** of an HTML element :

```
document.getElementById("p1").textContent = "New text!";
```

```
document.getElementById("p1").innerHTML = "New text!";
```

Note :

- innerHTML vs textContent
- innerHTML : potential security hole
  - Read more : https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML#Security_considerations

More Element Attributes

Attributes	JS
id	element.id
raw HTML	element.innerHTML
text	element.textContent
value	element.value
class	element.classList
attribute	element.attributes.

- To change the value of an HTML **attribute**, use this syntax:

  element.attribute = new value

- Example:

```
document.getElementById("myImage").src="landscape.jpg";
```

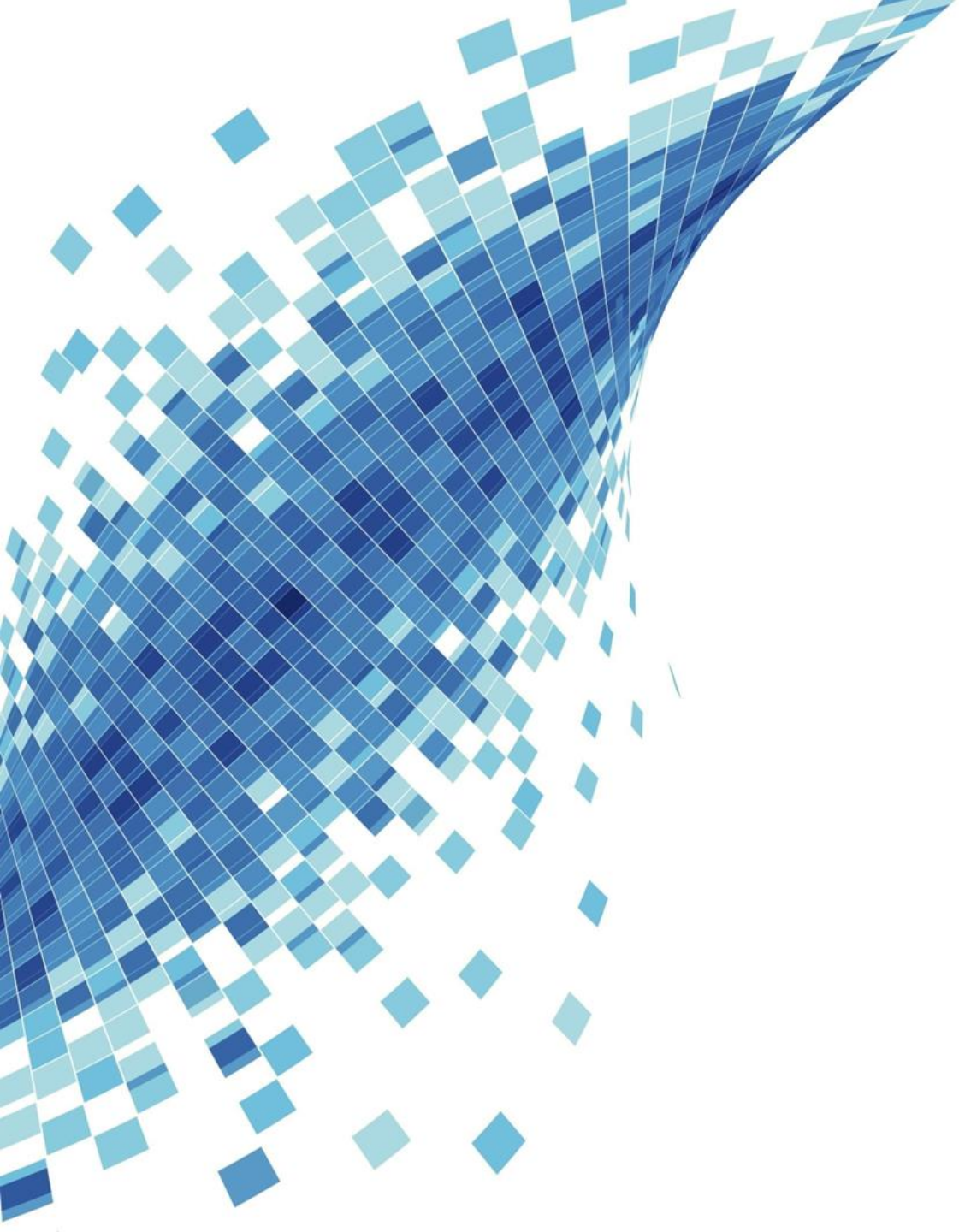
Modify CSS Element

- To change the **style** of an HTML element, use this syntax:

  element.style.*property = new style*

- Example:

```
document.getElementById("p2").style.color = "blue";
```

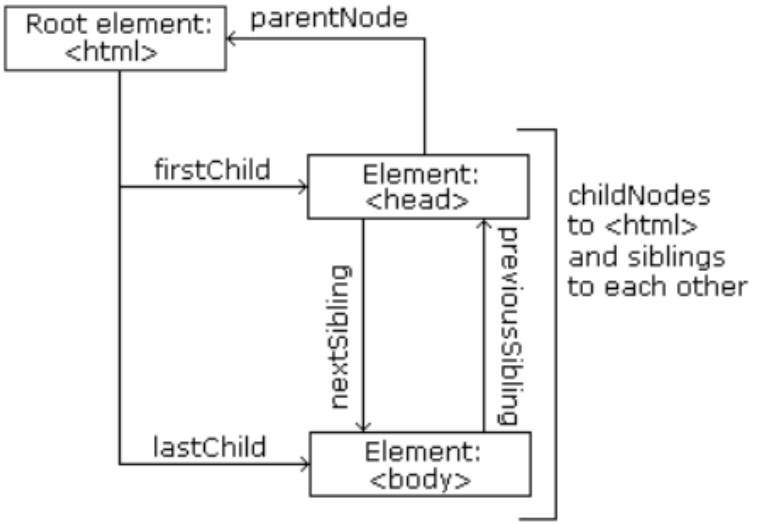


DOM Manipulation

INFORMATIKA
UNPAR

DOM Navigation

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



Method :

- parentNode
- childNodes[*nodenumber*]
- firstChild
- lastChild
- nextSibling
- previousSibling
- children

INFORMATIKA UNPAR

DOM Manipulation

- Several strategies for manipulating HTML elements in JS:
  1. Change content of existing HTML elements in page:
     - Good for simple text updates or value
       - element.textContent = “new content”;
       - element.value = “new content”;
       - element.attributes.href.value = “new content”;

DOM Manipulation (2)

2. Add elements via createElement and appendChild:

```
let element = document.createElement(tag)
parent.appendChild(element);
```

- Note : appendChild preferable than innerHTML
- Remove : element.remove()

DOM Manipulation (3)

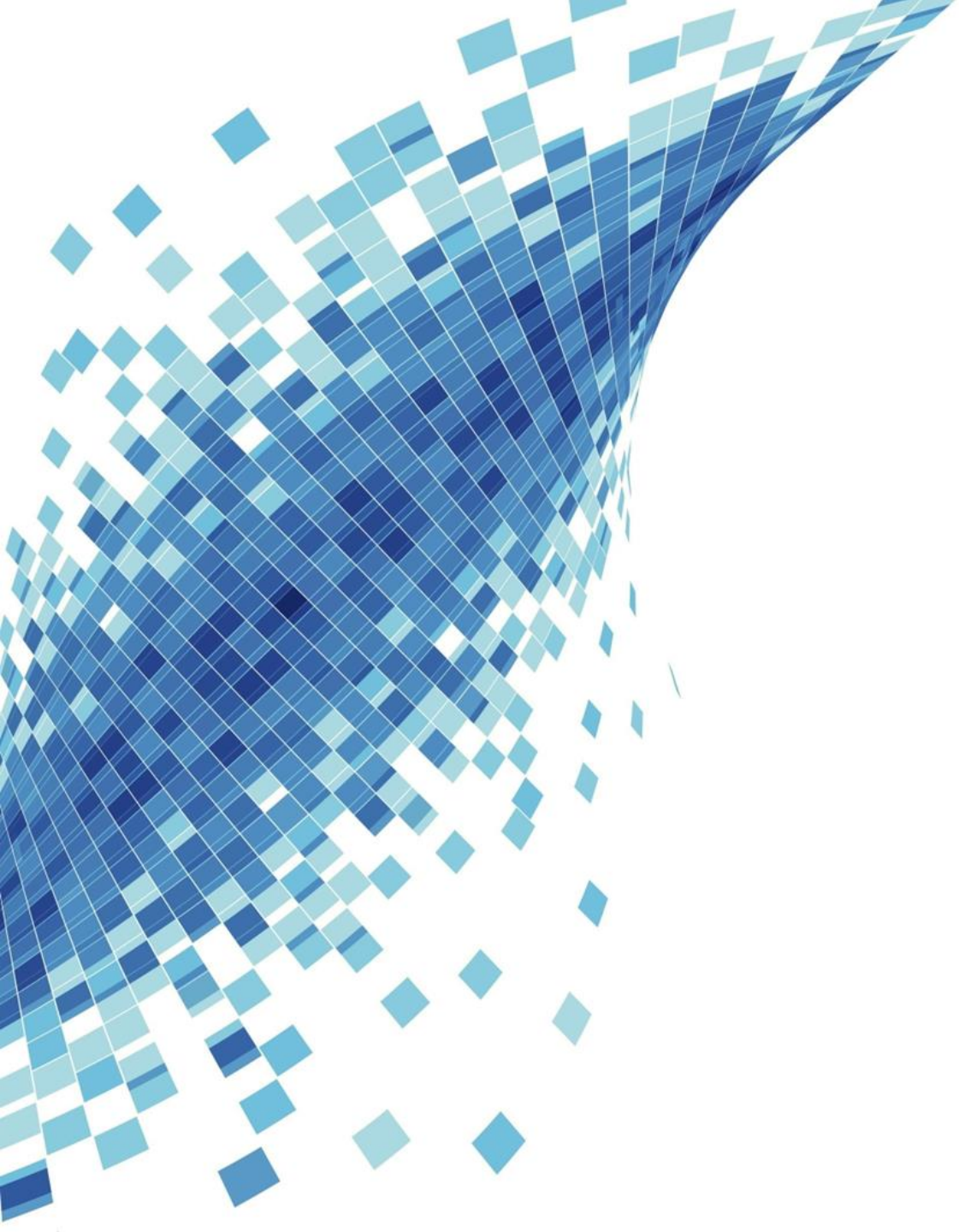
3. Put all "views" in the HTML but set inactive ones to hidden, then update display state as necessary.
   - JS :

```
el.classList.add('hidden')
el.classList.remove('hidden')
el.classList.toggle('hidden')
```

DOM Manipulation - Summary

- Common Strategy:

1. Change content of existing HTML elements in page:
   - Good for simple text updates.
2. Add elements via createElement and appendChild
   - Needed if you're adding a variable number of elements.
3. Put all "views" in the HTML but set inactive ones to hidden, then update display state as necessary.
   - Good when you know ahead of time what element(s) you want to display.
4. Combination two or three of them.



Callback

INFORMATIKA
UNPAR

Review Function

- Show the addition result to screen (label):

```
function show(text){
  let el = document.getElementById("label");
  el.textContent = text;
}

function add(a,b){
  return a + b;
}

let result = add(2,3);
show(result);
```

Problem :
Need to call 2 function

INFORMATIKA UNPAR

Review Function (2)

- Alternative:

```
function show(text){
  let el = document.getElementById("label");
  el.textContent = text;
}

function add(a,b){
  let result = a + b;
  show(result);
}

add(2,3);
```

Problem :
Cannot add without show.

Callback to The Rescue

- A callback is a function passed as an argument to another function.
- Example :

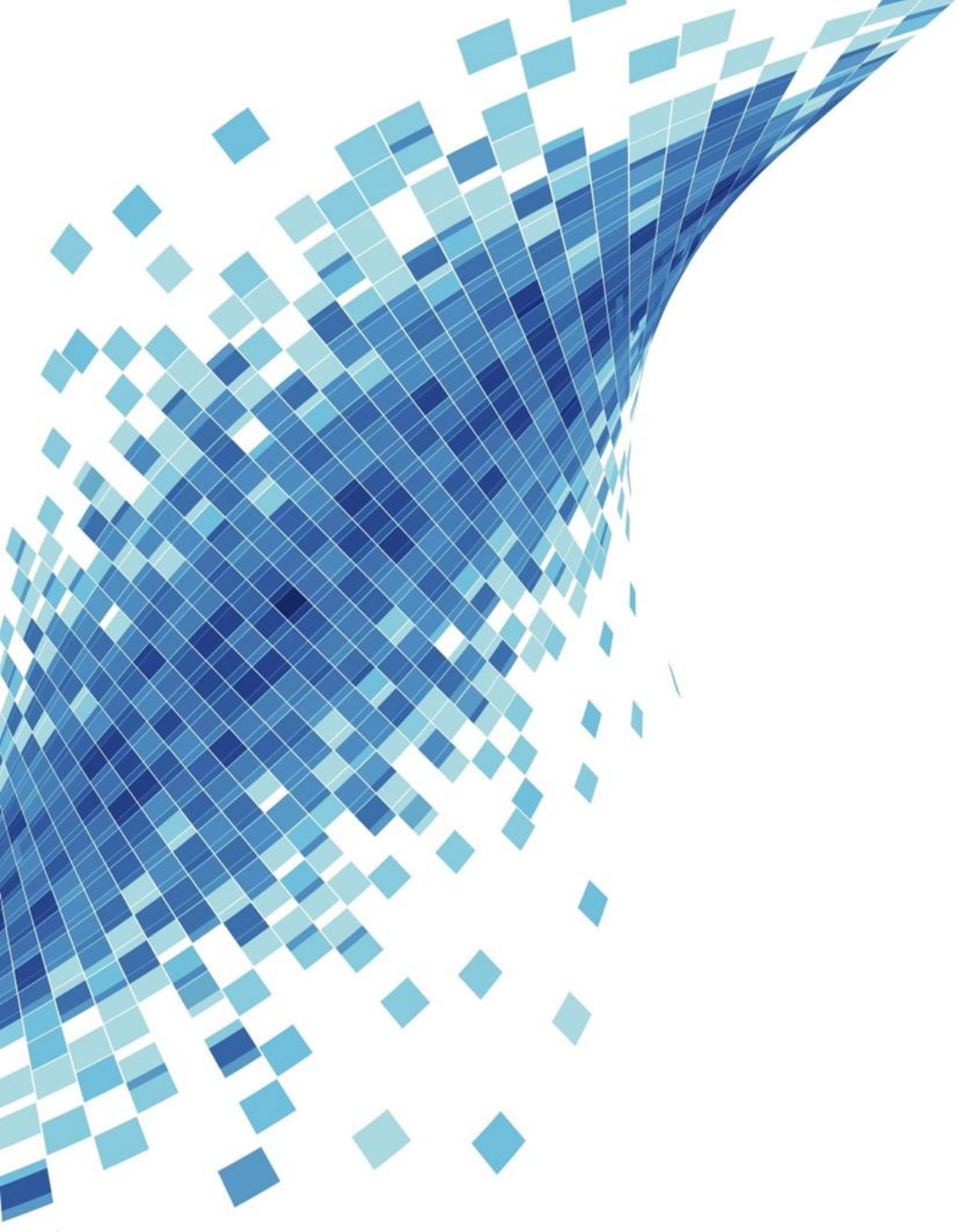
```
function fA(name){
      console.log("Hello " + name);
}

function fB(callback){
      callback("World")
}

fB(fA);
```

Prev problem solution using callback

```
function show(text){
  let el = document.getElementById("label");
  el.textContent = text;
}

function add(a,b,callback){
  let result = a + b;
  if (callback) callback(result);
  return result;
}

add(2,3,show); //change label to "5"

//Other usage 1
let result = add(2,3); //result = 5
console.log(result);
//Other usage 2
add(2,3,console.log); //print 5 on console
```

Callback summary

- These examples doesn’t show much significant change on codes whether we use callback or not. (simple case)
- But it show basic ability of callback.
- Later on, callback function truly shine on asynchronous javascript.



Event-Driven

INFORMATIKA
UNPAR

Event-driven programming

- Some examples of HTML **events**:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked

- JavaScript lets you execute code when events are detected.

Events

- HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.
- Syntax:

  <HTML_tag on(*event_name*)="*JavaScript*">
- Example:

```
<html>
<body>
    <button onclick="displayDate()">
    The time is?</button>
</body>
</html>
```

Common Events

- **`click`**
  The user clicks an HTML element
- **`change`**
  An HTML element has been changed
- **`mouseover`**
  The user moves the mouse over an HTML element
- **`mouseout`**
  The user moves the mouse away from an HTML element
- **`keydown / keyup`**
  The user pushes / releases a keyboard key
- **`load`**
  The browser has finished loading the page

Event listener

- Attach event handler to specific element.
- No need overwriting existing event handlers.
- You can add many event handlers to one element.
- You can add many event handlers of the same type to one element, i.e two "click" events.
- the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup

Event listener

- Add Event :
  - addEventListener(event, function);

```
element.addEventListener("click", function(){
    alert("Hello World!");
});
element.addEventListener("click", myFunction);
```

- Remove Event :
  - removeEventListener(event, function);

```
element.removeEventListener("click",myFunction);
```

Alternative: Arrow Function

- Arrow function is the alternative for declaring function.

```
function add(a, b) {
  return a + b;
}

const add = (a, b) => {
  return a + b;
};
```

Alternative: Arrow Function

- Advantages:
  - Arrow function do not bind to *this* context. Means it is useful when you do not want to deal with changing context of *this* keyword.
  - Usually, the code is shorter.

```
button.addEventListener('click', event => {
  //Do something here
});
```

Multiple event

- What if you have event listeners set on both an element and a child of that element?

```
<div id="outer">
  Click me!
  <div id="inner">
    No, click me!
  </div>
</div>
```

- Event bubbling : ordering event from inner to outer.
- Event capturing: ordering event from outer to inner.

```
element.addEventListener('click',onClick,{capture : true});
```

- Can be stopped using:

```
event.stopPropagation();
```

INFORMATIKA UNPAR

Finding the element twice...

```
function openPresent() {
  const image = document.querySelector('img');
  image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

- Solution:
  - JS : event.currentTarget

```
function openPresent(event) {
  const image = event.currentTarget;
  image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
```

Intermezzo : Important Notes

INFORMATIKA
UNPAR

HTML ready state

```
<html>
▼<head>
    <meta charset="utf-8">
    <title>First JS Example</title>
    <script src="script.js"></script>
  </head>
▼<body>
    <button>Click Me!</button>
  </body>
</html>
```

```
function onClick() {
  console.log('clicked');
}

const button = document.querySelector('button');
button.addEventListener('click', onClick);
```

What option do we have?

- Put the <script> tag at the bottom of the page
  - Not “silver-bullet” solution
- Run js after html load
  - onload event : `<body onload="onLoad()">`
  - window.onload : `window.onload = onLoad;`
  - DOMContentLoaded eventlistener :

```
document.addEventListener("DOMContentLoaded",onLoad);
```

  - ~~jQuery ready : `$(document).ready(onLoad);`~~
- Use defer: `<script src="script.js" defer></script>`
  - Newer and recommended

Extra

INFORMATIKA
UNPAR

1. Don't query UI for state

Example :

Making all blue buttons unclickable.

```
1  let btns = document.querySelectorAll(".blue");
2
3  for(let item in btns){
4      item.disabled = true;
5  }
```

- This is not a great software engineering technique
  - Couples your "view" and your "model"
  - Track your state separately from UI

2. Data attributes

```
<div id="grid">
  <div id="one"></div>
  <div id="two"></div>
  <div id="three"></div>

  <div id="four"></div>
  <div id="five"></div>
  <div id="six"></div>

  <div id="seven"></div>
  <div id="eight"></div>
  <div id="nine"></div>
</div>
```

- Problem :
  - multiple id
  - cannot use number as id

- New way

```
<div id="grid">
    <div data-index="0"></div>
    <div data-index="1"></div>
    <div data-index="2"></div>

    <div data-index="3"></div>
    <div data-index="4"></div>
    <div data-index="5"></div>

    <div data-index="6"></div>
    <div data-index="7"></div>
    <div data-index="8"></div>
</div>
```

INFORMATIKA UNPAR

2. Data attributes (2)

- Use data attributes
- Syntax:
  - data-attr_name =”Your value”

```
<article
  id="electriccars"
  data-columns="3"
  data-index-number="12314"
  data-parent="cars">
...
</article>
```

- Access via JS

```
var article = document.getElementById('electriccars');

article.dataset.columns // "3"
article.dataset.indexNumber // "12314"
article.dataset.parent // "cars"
```

3. jQuery

- From official website:
  jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.
- Initial release : 2006

Accessing the DOM using jQuery

Using Vanilla JS

- **getElementById(“blog”)**
- **getElementsByTagName(“p”)**
- **innerHTML**
- **value**

Using jQuery

- **$(“#blog”)**
- **$(“p”)**
- **.html()**
- **.val()**

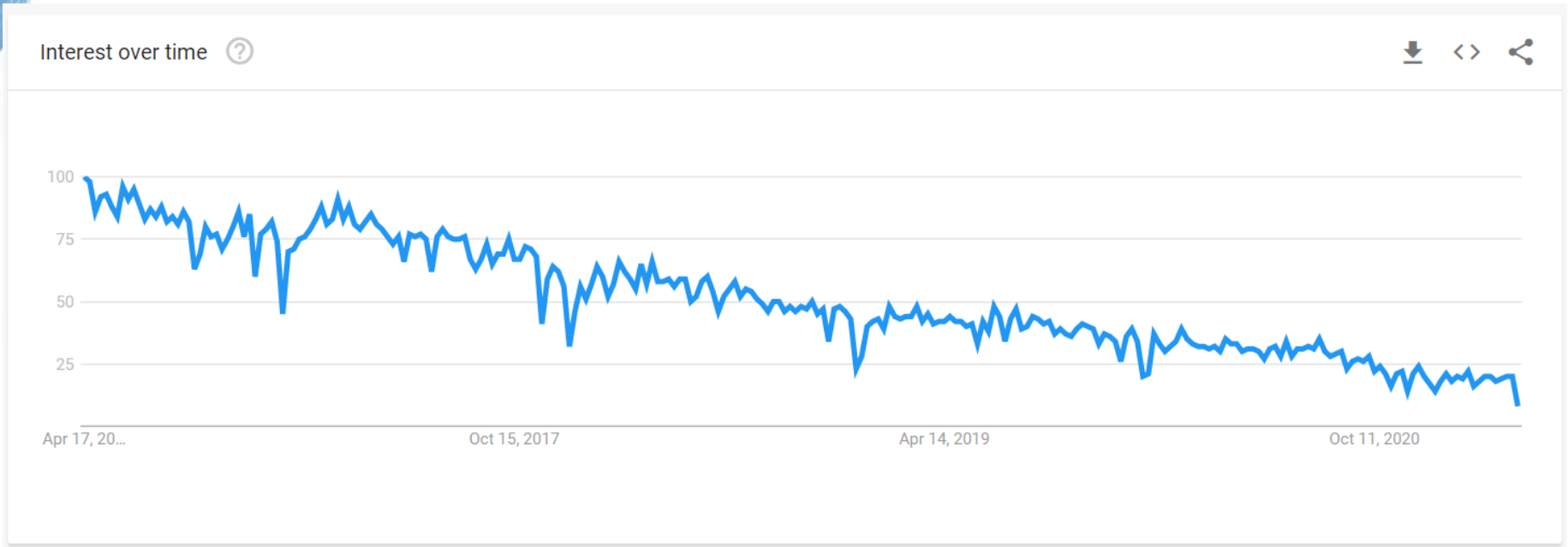
Example

```
var secondaryHeadings = $("h2");
alert(secondaryHeadings.html());
secondaryHeadings.html("Now we've changed the content");
```

jQuery Breakdown

- jQuery is a lightweight, "write less, do more", JavaScript library.
  - true. But, compared to non library?
  - jQuery line of code is more than your weekly assignments.
- jQuery greatly simplifies JavaScript programming.
  - true. But, what if js has improved?
  - Modern js can do most of jQuery features.
- jQuery is easy to learn.
  - half true :)

jQuery trends in past 5 years



INFORMATIKA UNPAR

jQuery Summary

- jQuery has already past its glory many years ago.
- There is still tons of tutorials on internet using jQuery. (please browse wisely)
- Usage is still high because of legacy code, but slowly decreasing from time to time.
- For newer developer better not starting learn jQuery.

Thanks

INFORMATIKA
UNPAR

INFORMATIKA
UNPAR

informatika.unpar.ac.id
informatika@unpar.ac.id
if.unpar
if.unpar