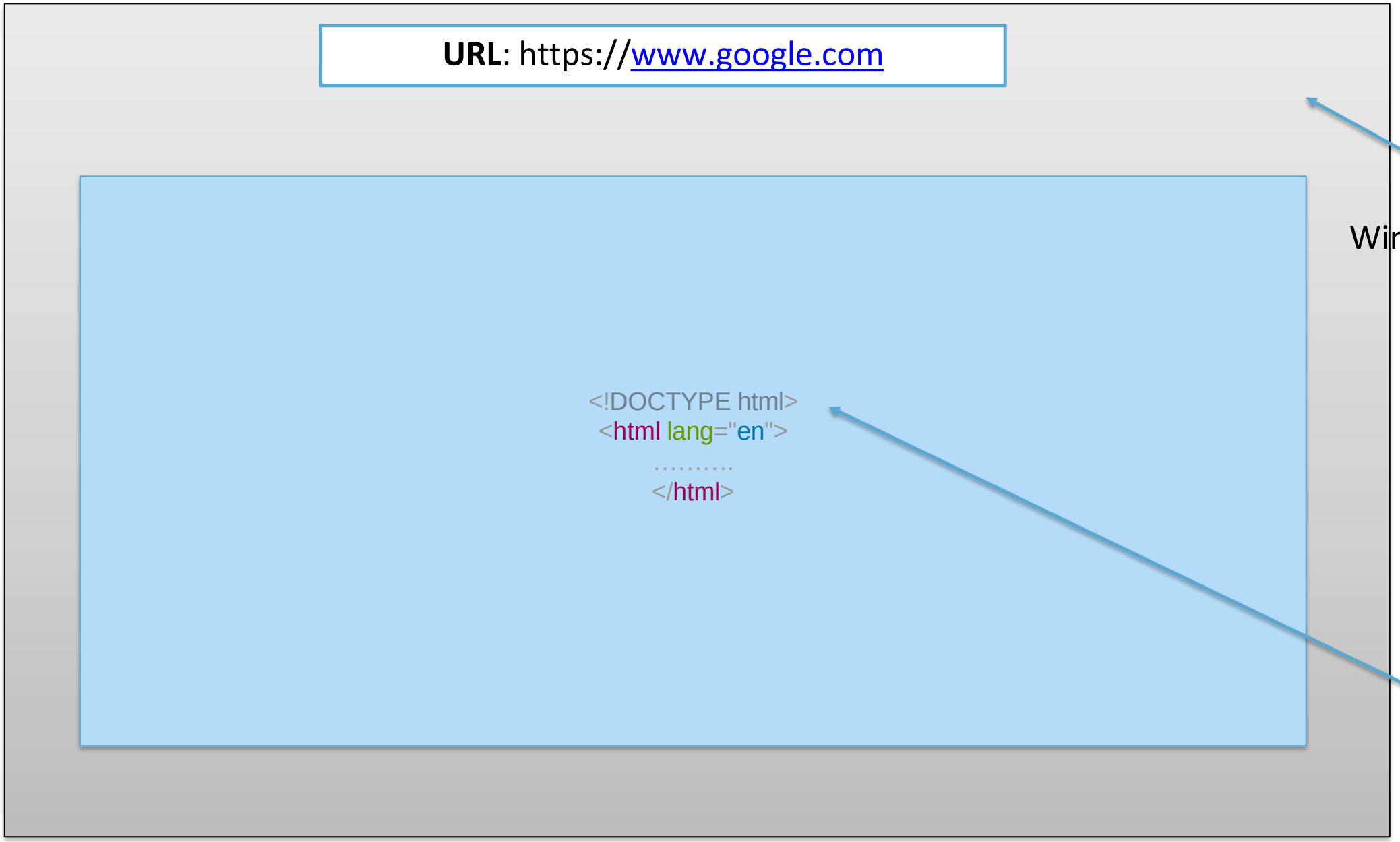
DOM - Document object model

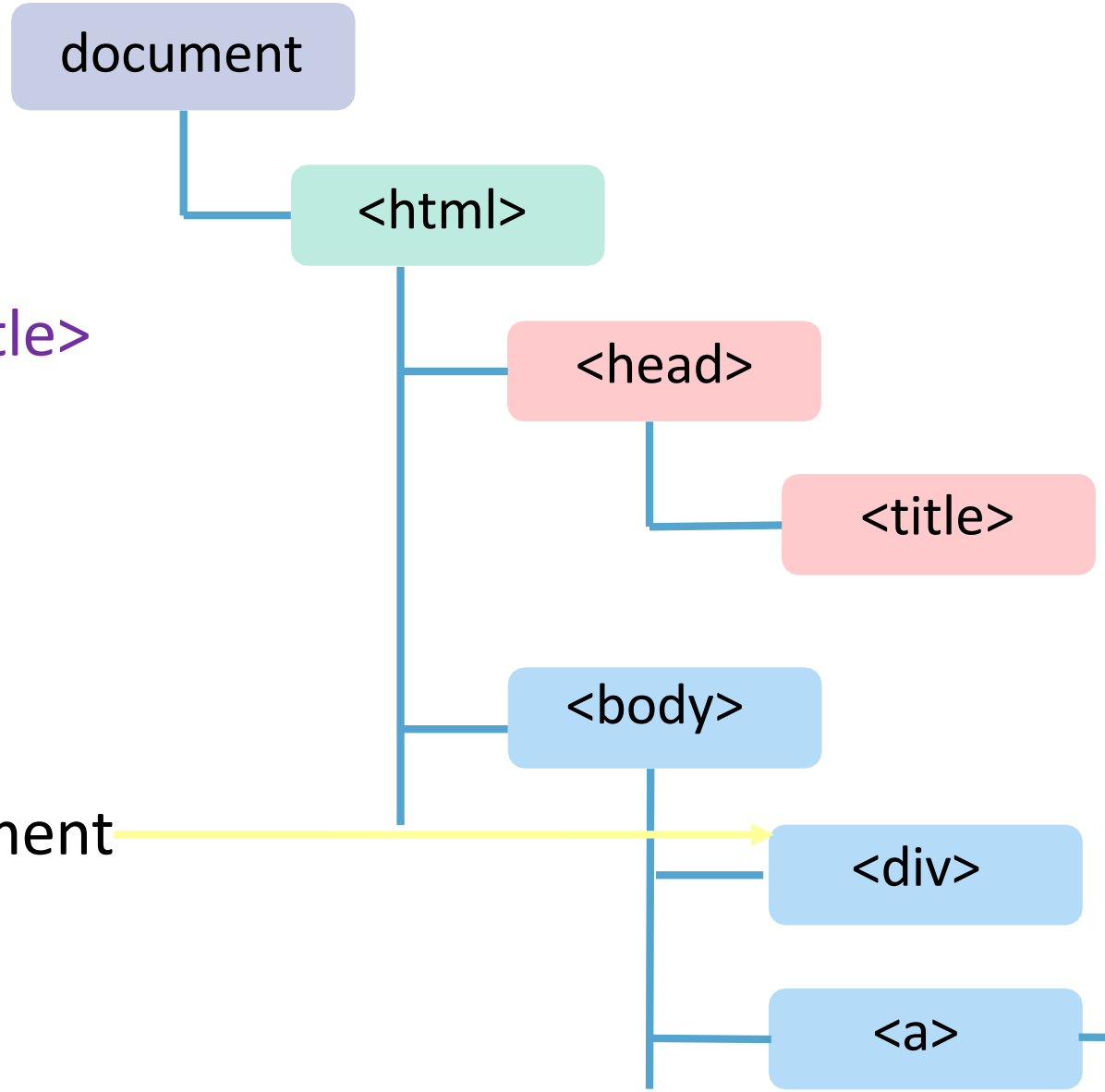


Testleaf
Always Ahead

DOM & WebElement

```
<html>
<head>
   <title> Amazon India </title>
</head>
<body>
<div>
 <span>
   <input>
   </input>
 </span>
</div>
</html>
```

WebElement



Three core components of a webpage

- ## HTML (HyperText Markup Language)

  Defines the structure and content of a webpage.
  Example: Headers, paragraphs, images, and forms.

- ## CSS (Cascading Style Sheets)

  Controls the visual styling of the webpage.
  Example: Fonts, colors, layouts, and animations.

- ## JavaScript

  Makes the webpage dynamic and interactive.
  Example: Form validation, animations, and real-time updates.

Locators & Selectors

```
        else{
            awa
            awa
            awa
            con
            exp
            con
        }
    }
})

test('Locators'
    await page.locator('#user-name')
});
```

```
(method) Page.locator(selector: string, options?: {
    has?: Locator | undefined;
    hasText?: string | RegExp | undefined;
} | undefined): Locator
```

The method returns an element locator that can be used to perform actions on this page / frame. Locator is resolved to the element immediately before performing an action, so a series of actions on the same locator can in fact be performed on different DOM elements. That would happen if the DOM structure between those actions has changed.

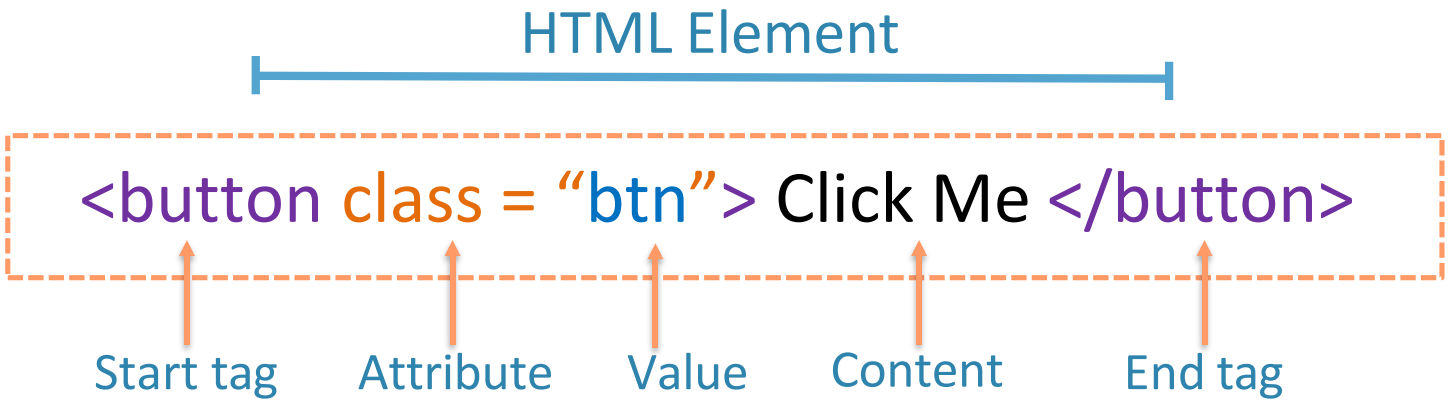
Learn more about locators.

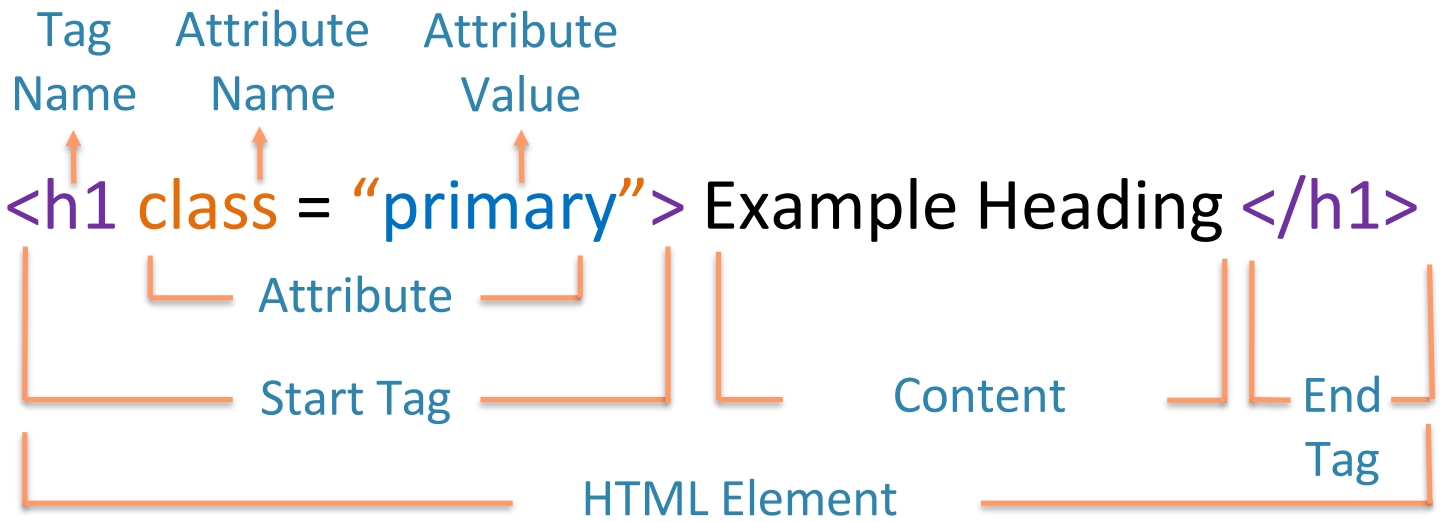
@param `selector` — A selector to use when resolving DOM element. See working with selectors for more details

Agenda – Part 3

- Locators
- Selectors
- Diving into Selector Strategies
- Which strategy to prefer?
- Industry best practices

Let’s understand structure of an element





- Tags
- Attribute Name
- Attribute Value
- Visible Text

Diving into Selector Strategies

- Xpath Locators
- CSS Selectors
- Playwright recommended selectors

Playwright recommended Locator Strategy

- getByRole()
- getByText()
- getByLabel()
- getByPlaceholder()
- getByAltText()
- getByTitle()
- getByTestId()

Playwright recommended Locator Strategy

- getByRole()

To locate a web element by its role (button, link, checkbox, alert)

Syntax :

```
await page.getByRole('button', { name: 'Login' }).click();
```

- getByText()

To locate a web element by the text content.

Syntax:

```
await page.getByText('Username').click();
```

Playwright recommended Locator Strategy

- ## getByLabel()

To locate a web element by the label's text

Syntax :

```
await page.getByLabel('Username').click();
```

- ## getByPlaceholder()

to locate an input by its placeholder value

Syntax:

```
await page.getByPlaceholder('Search Amazon.in').click();
```

Playwright recommended Locator Strategy

- getByAltText()

  To locate a web element by its images/logos (text alternatives)

  Syntax :

  ```
  await page.getByAltText("Playwright logo").isVisible();
  ```

- getByTitle()

To locate a web element by its title attribute

<title>Home | Salesforce</title>

Syntax :

```
await page.getByTitle("Home | Salesforce").isVisible();
```

Playwright recommended Locator Strategy

- getByTestId()

  To locate a web element based on its data-testid attribute.(check : amazon)

  Syntax :

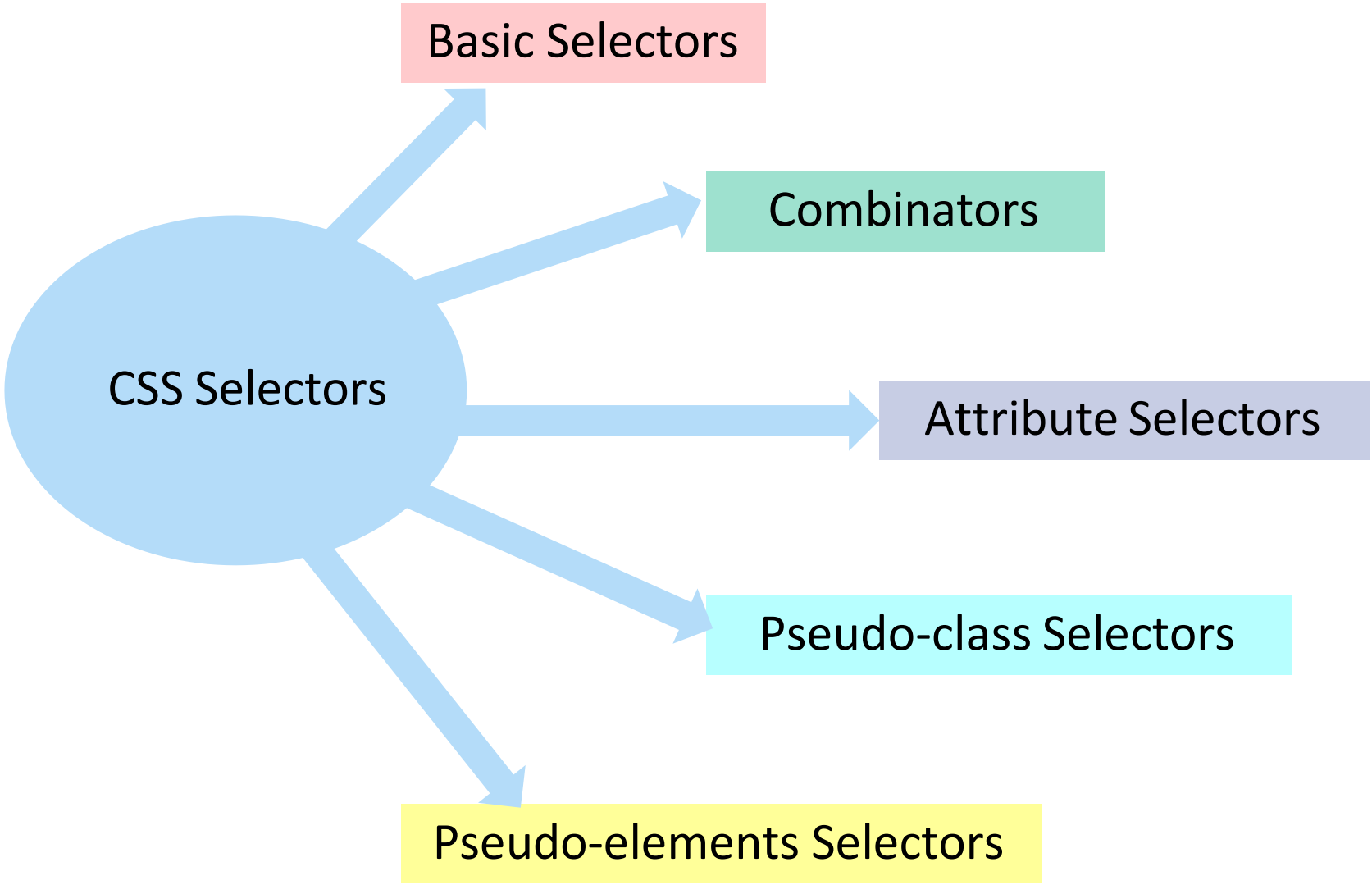
  ```
  page.getByTestId("overlay").click();
  ```

Sample code with getByMethod()

```
import { test, expect } from '@playwright/test';

test('test', async ({ page }) => {
  await page.goto('http://leaftaps.com/opentaps/control/main');
  await page.getByLabel('Username').click();
  await page.getByLabel('Username').fill('democsr');
  await page.getByLabel('Password').click();
  await page.getByLabel('Password').fill('crmsfa');
  await page.getByText('Username').click();
  await page.getByRole('button', { name: 'Login' }).click();
  await page.getByRole('link', { name: 'CRM/SFA' }).click();
});
```

CSS Selectors



CSS most common used strategies

- Find an element by tag name

  Syntax: tagName[attribute=value]

- Find an element by ID

  Syntax: tagName[id=value] OR *#idvalue*

  Hash "#" to denote id

- Find an element by class

  Syntax: tagName[class=value] OR *.classname*

  Dot "." to denote class

Testleaf
Always Ahead

CSS selectors for different situations

Type Selector

This selects the first <button> element on the page using the tag name button. In CSS, selecting by tag name means it will target all elements of that type unless further specified.
tagname <button>

Syntax :

```
page.locator("button").click();
//or use this → page.locator("button#submitBtn").click();
```

ID - #idValue

To locate a web element by its ID
<input class="inputLogin" type="text" id="username" name="USERNAME" size="50">

Syntax :

```
page.locator("#username");
//or use this → page.locator(“input[id=username]”)
```

CSS selectors for different situations

- Class - .classValue

To locate a web element by its class

<input class="inputLogin" type="text" id="username" name="USERNAME" size="50">

Syntax :

```
page.locator(".inputLogin").fill("demosalesmanager");
// or use this → page.locator(“input[class=inputLogin]”). fill("demosalesmanager");
```

- Attribute selector - [attributeName = 'attributeValue']

To locate by its HTML tag

<input class="inputLogin" type="text" id="username" name="USERNAME" size="50">

Syntax :

```
page.locator("[name='USERNAME']")
// or use this → page.locator(“input[name=USERNAME]”). fill("demosalesmanager");
```

CSS selectors for different situations

Descendant Combinator (ancestor descendant)

Selects all elements that are *descendants* (not just direct children) of a specified ancestor.

Example :

selects all <input> elements inside a <form>, regardless of nesting level.

Syntax : form input

Child Combinator (parent>child)

Selects elements that are direct children of a specified parent.

Example :

selects <input> elements that are directly inside a <p> tag, not deeper nested inputs

Syntax :

p > input.

CSS selectors for different situations

Adjacent Sibling Combinator (element + adjacent)

Selects an element that is the immediate next sibling of another element.

Example :
selects the first <input> immediately following a <label>

Syntax : label + input

General Sibling Combinator (element ~ sibling)

Selects all siblings that come after a specified element, not necessarily immediately.

Example :
selects all <p> elements that follow an <h2> in the same parent, even if other elements are in between.

Syntax : h2 ~ p

Playwright recommended Locator Strategy

Substring

- ### Starts with attribute selector (Carrot ^)

Starts with attribute selector (Carrot ^)
Select an elememt with an attribute that starts with a specific value

Syntax :

tagname[attribute^='value']

div[class^='inventory']
page.locator("div[class^='inventory']")

Playwright recommended Locator Strategy

Ends with attribute selector(Dollar - $)

Select an element with an attribute that ends with a specific value

Syntax :

tagname[attribute$='value']

```
<button class="btn_primary btn_inventory">ADD TO CART</button>
<button class="btn_secondary btn_inventory">ADD TO CART</button>
<button class="btn btn_inventory">ADD TO CART</button>
```

```
page.locator("button[class$='btn_inventory']")
```

Contains attribute selector

Select an element with an attribute containing a specific substring

tagname[attribute*='value']

button[class*='btn']

Industry Best Practices

- Writing locator which are unique
- Writing locator which are readable and can provide some context
- Using text based locator
- Using a unique attribute dedicated for testing like data test id etc.,

Playwright *config* file walkthrough

testDir	Contains dir path where all test cases resides
fullyParallel	Decides if to run all files in parallel or all. Files and test cases inside them in parallel
Test timeout	Max time allowed for a test case to run
retries	Number of time a test case to retry on failure
workers	Number of workers to split the test cases on
trace	On what conditions, we should keep the trace
screenshots	On what conditions, we should keep the screenshot
video	On what conditions, we should keep the video
Expect timeout	Max time allowed for assertions we are doing using expect

Auto Wait

Playwright when performing any relevant action first ensures that

- All applied checks for that actions are passed
- Within a set timeout which can be overridden by
- If the criteria is not full filled , it will raise ***TimeoutError***

page.click() ensures

- element is attached
- element is visible
- element is stable
- element is receiving events
- element is enabled