08: Web Design 1

Review

Responsive CSS with Media Queries

JavaScript Document Object Model

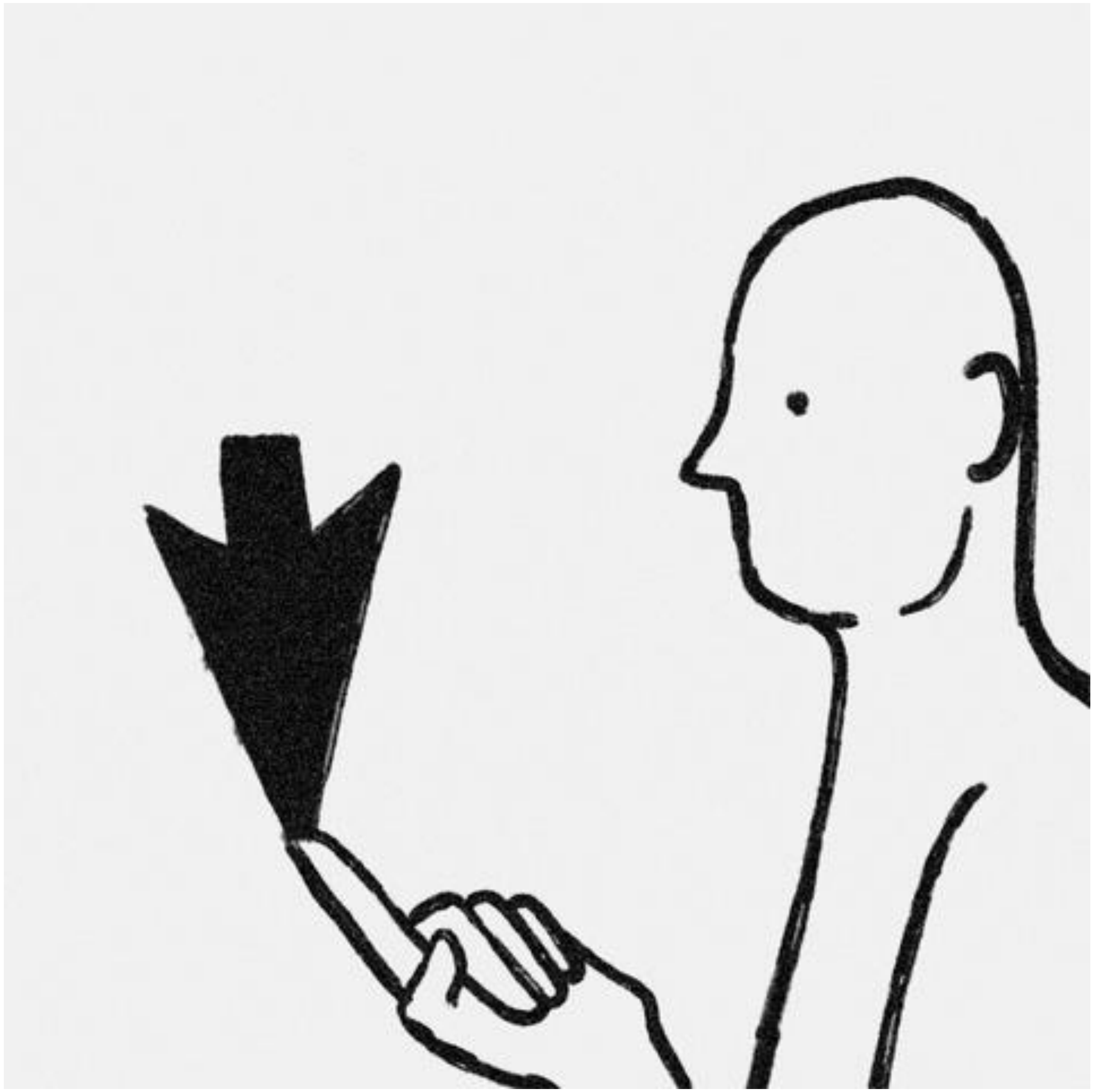
JavaScript Intro – Variables & Functions

DOM Manipulation

- Adding/Removing classes
- Creating elements

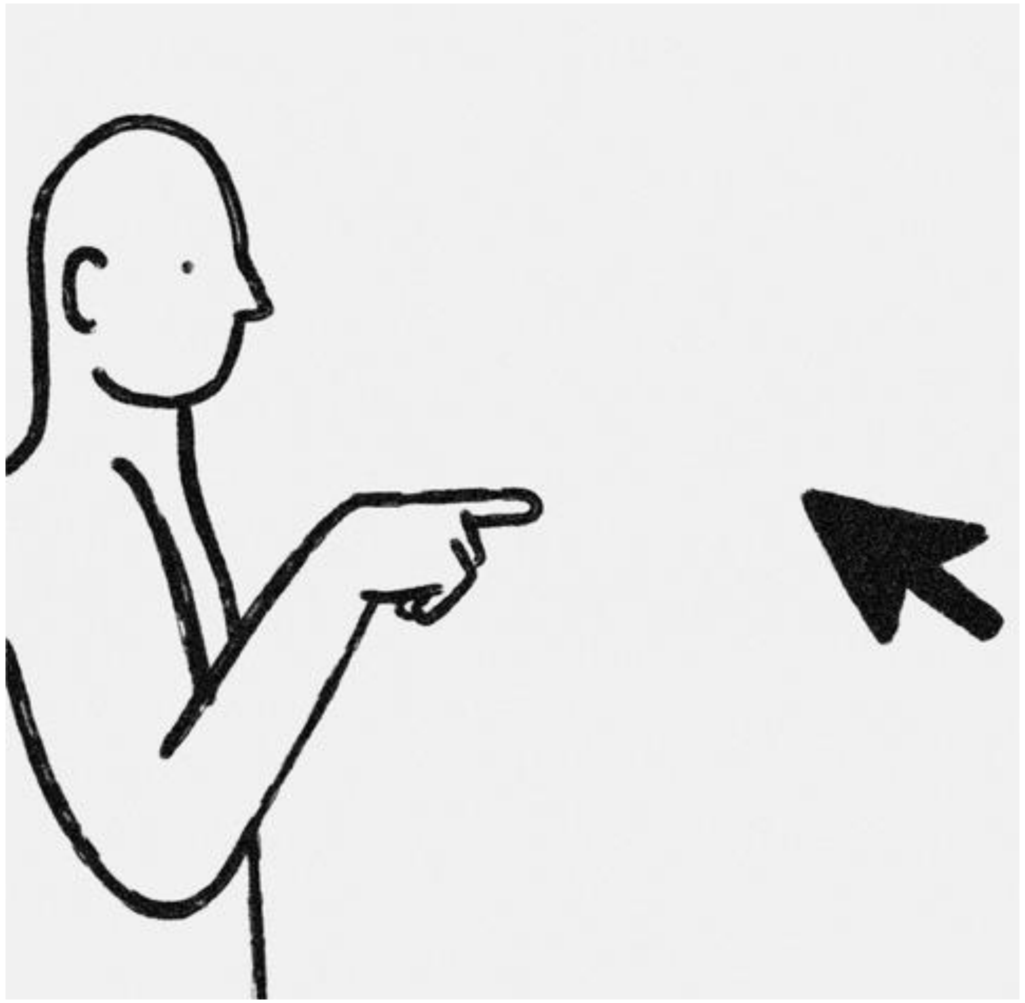
Timing Functions

JavaScript Events



JavaScript Events

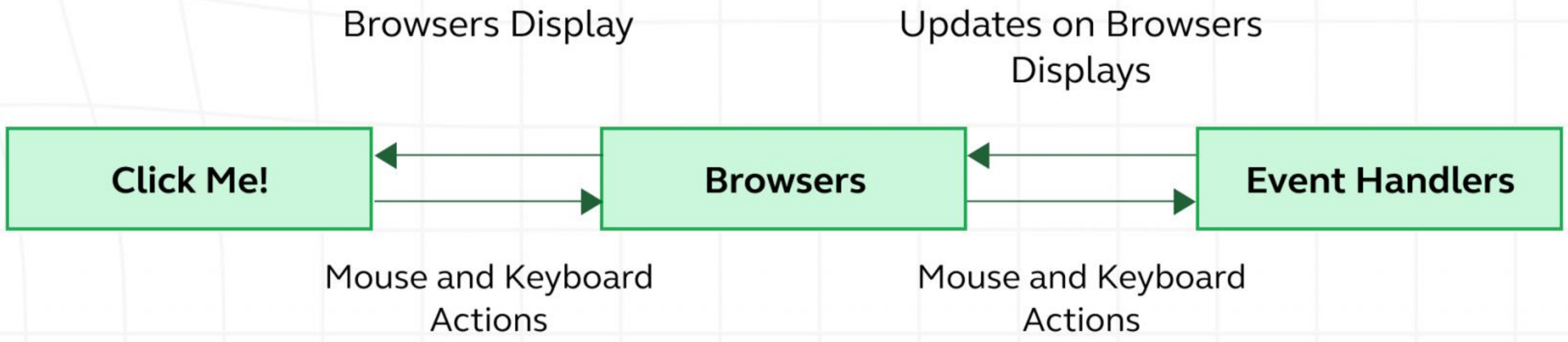
- Events are actions or occurrences that happen in the browser, often triggered by user interactions
- They allow JavaScript to react to user inputs, making web apps more dynamic.



JavaScript Events

- **Mouse Events**
  - click: User clicks on an element.
  - dblclick: User double-clicks.
  - mouseover: Mouse hovers over an element.
  - mouseout: Mouse leaves an element.
- **Keyboard Events**
  - keydown: Key pressed down.
  - keyup: Key released.
- **Form Events**
  - input: User interacts with form elements.
  - submit: Form submitted.

Event and Event Handlers


Browsers Display
Updates on Browsers Displays
Click Me!
Browsers
Event Handlers
Mouse and Keyboard Actions
Mouse and Keyboard Actions

Event Cycle

- **Event Trigger**: An event occurs (like clicking a button).
- **Event Listener**: JavaScript "listens" for the event.
- **Event Handler**: A function executes in response to the event.

Listening to Events

- Events that occur on an element can be ‘listened’ to with addEventListener method

```
myElement.addEventListener(event, eventHandler)
```

Listening to Events

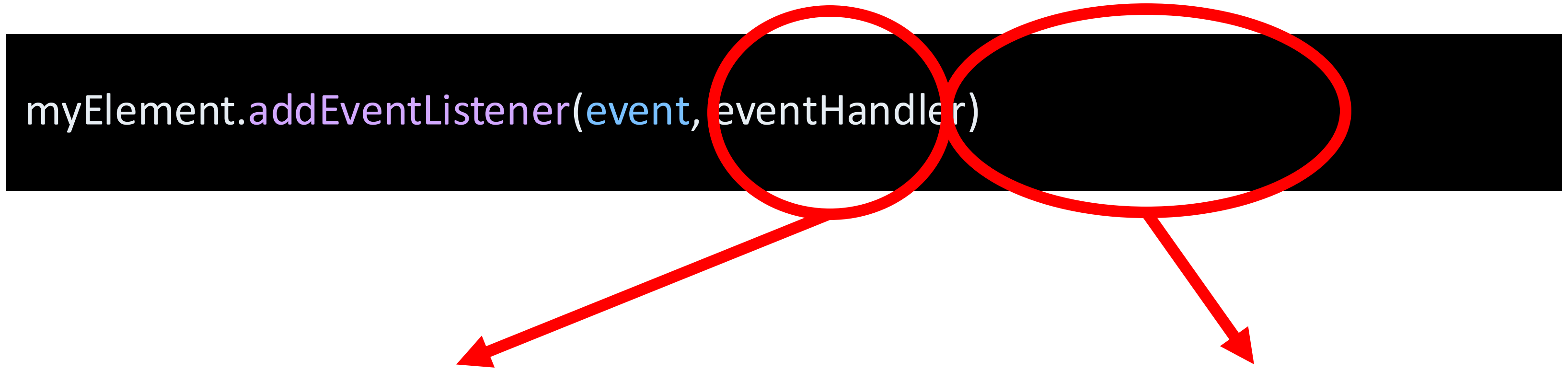
- Events that occur on an element can be ‘listened’ to with addEventListener method

```
myElement.addEventListener(event, eventHandler)
```

The event we want to listen to:
click, dblclick, keypress etc

Listening to Events

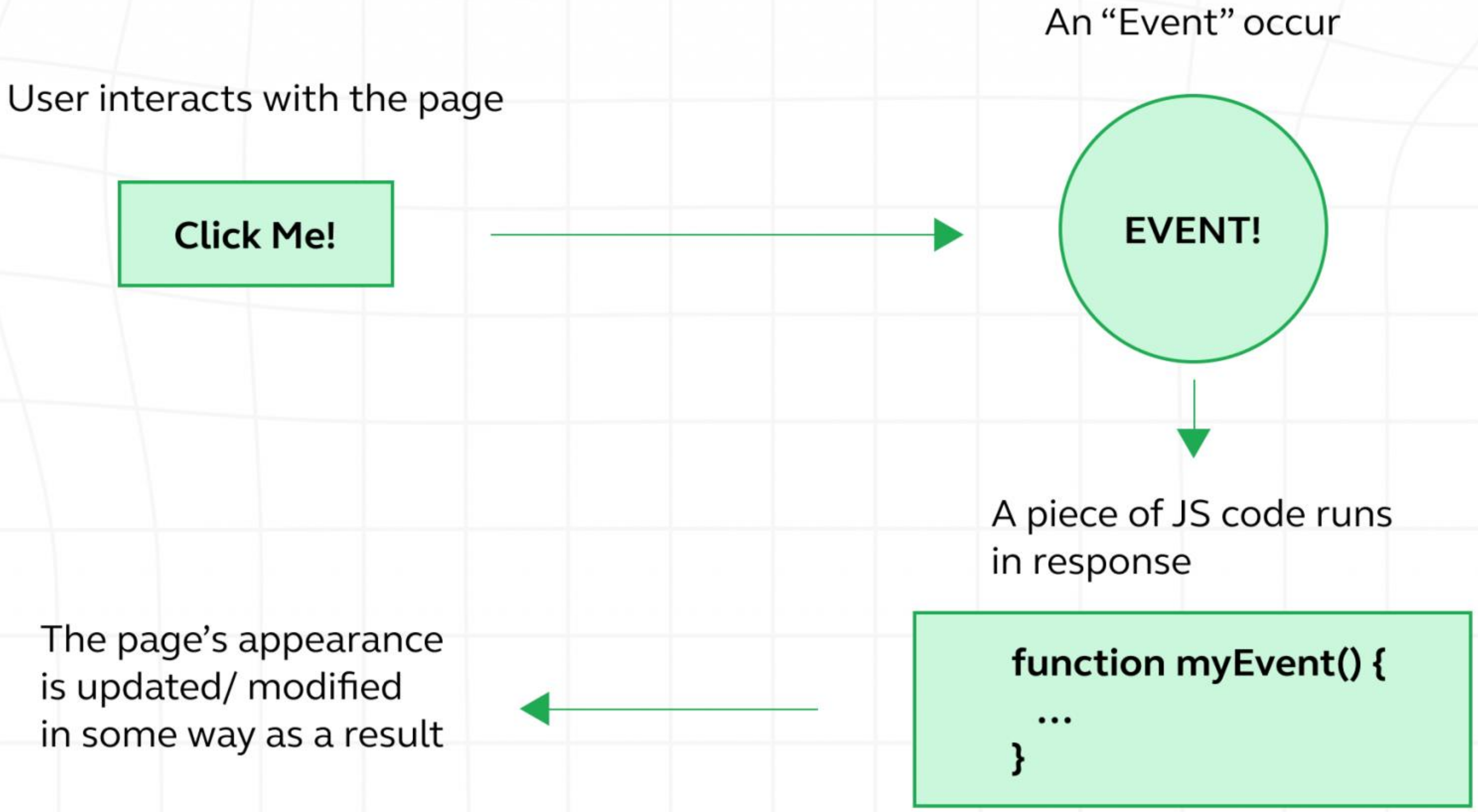
- Events that occur on an element can be ‘listened’ to with addEventListener method



```
myElement.addEventListener(event, eventHandler)
```

The event we want to listen to:
click, dblclick, keypress etc

The function that runs when the event occurs


User interacts with the page
Click Me!
An "Event" occur
EVENT!
A piece of JS code runs in response
The page's appearance is updated/ modified in some way as a result

```
function myEvent() {
  ...
}
```

Listening to Events

```
// Select the element
const btn = document.querySelector("#my-btn");

// Event Handler
function handleClick() {
  alert("Hurrah!")
}

// Listening to the event
btn.addEventListener('click', handleClick)
```

Listening to Events

```
// Select the element
const btn = document.querySelector("#my-btn");

// Event Handler
function handleClick() {
  alert("Hurrah!")
}

// Listening to the event
btn.addEventListener('click', handleClick)
```

Passing only a reference to the function

No parenthesis() after the function name

Runs only when the event is triggered

Aside: Writing Anonymous Functions

```
function handleClick() {
  alert("Hurrah!")
}

btn.addEventListener('click', handleClick)
```

```
// Inline function
btn.addEventListener('click', function() {
  alert("Hurrah from the inline function")
})
```

What's this***?***

- this refers to the element that triggered the event when using traditional function syntax in an event handler.

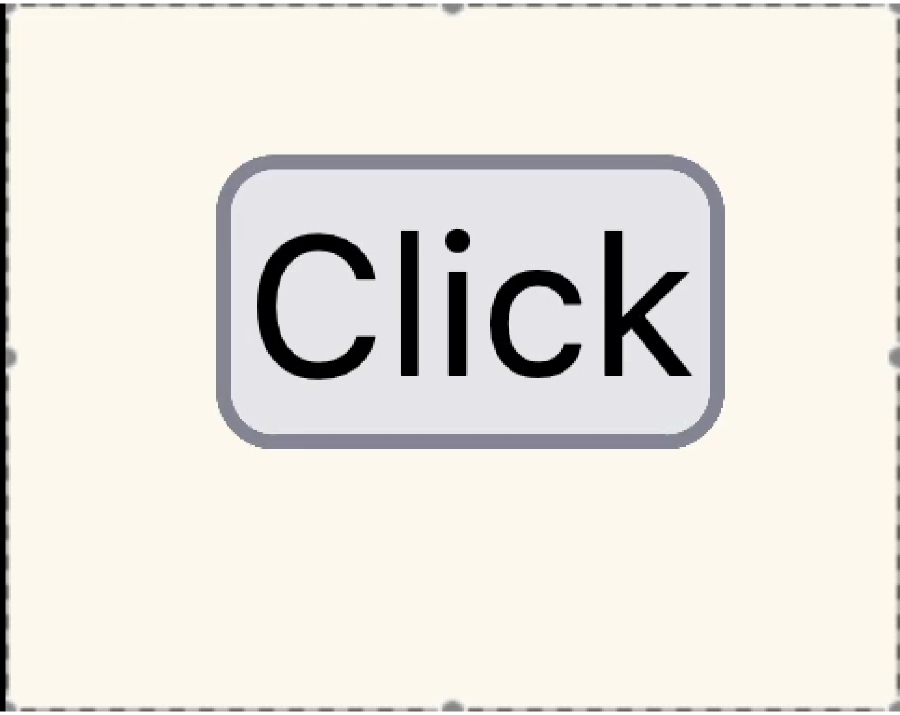
Note: With arrow functions, this inherits its value from the outer scope instead of the event target.

Using this

```
btn.addEventListener("click", function () {
// `this` refers to the element itself (the button)


// adds background color pink style on button press
  this.style.backgroundColor = "pink";
});
```

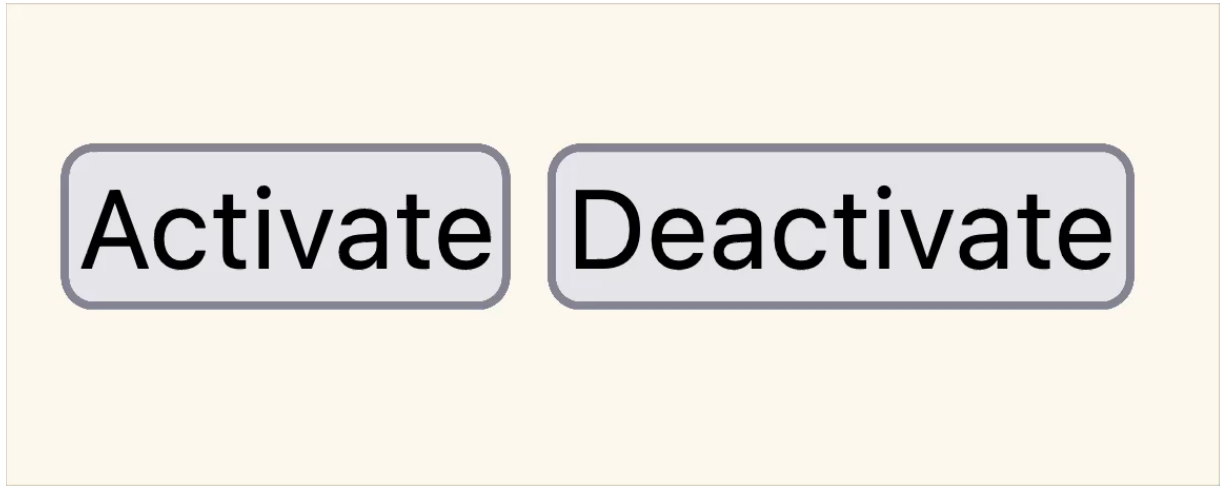
Using this

```
btn.addEventListener("click", function () {
// `this` refers to the element itself (the button)


// adds background color pink style on button press
  this.style.backgroundColor = "pink";
});
```



In-Class Exercise 1

- In a new web project, define a CSS class that changes the background colour of the body.
- Create two buttons labelled "**Activate**" and "**Deactivate**".
- Attach JavaScript event listeners to each button to dynamically add and remove the CSS class on the page when the buttons are clicked.



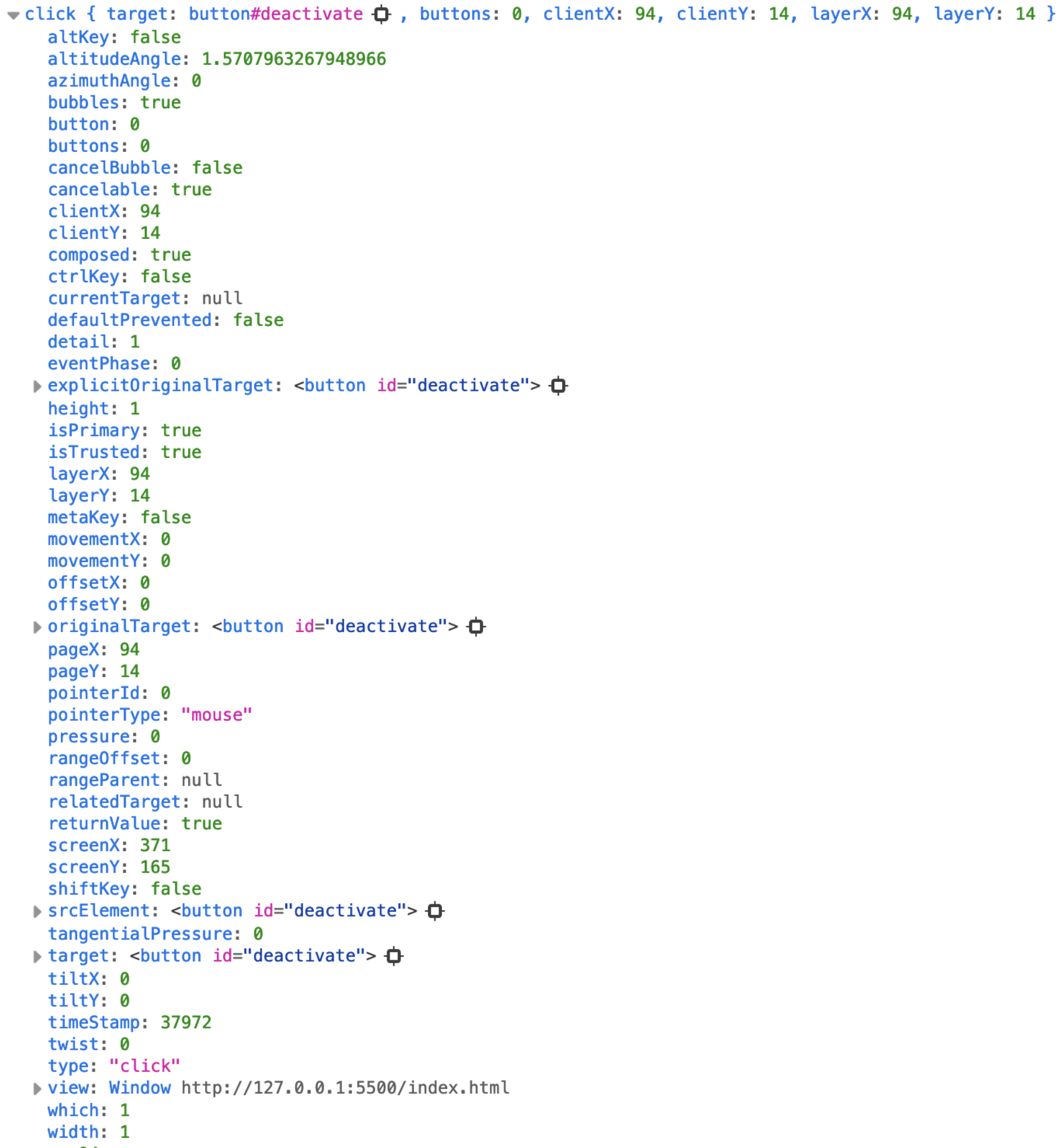
The Event Object

- The event object is automatically passed to event handler functions, containing information about the event that occurred.
- It provides useful details like which element triggered the event, the type of event, and more, which allows us respond to the event effectively.

The Event Object

Receiving with the event argument

```
btn.addEventListener("click", function (e) {
  console.log(e)
});
```



```
click { target: button#deactivate , buttons: 0, clientX: 94, clientY: 14, layerX: 94, layerY: 14 }
    altKey: false
    altitudeAngle: 1.5707963267948966
    azimuthAngle: 0
    bubbles: true
    button: 0
    buttons: 0
    cancelBubble: false
    cancelable: true
    clientX: 94
    clientY: 14
    composed: true
    ctrlKey: false
    currentTarget: null
    defaultPrevented: false
    detail: 1
    eventPhase: 0
    explicitOriginalTarget: <button id="deactivate">
    height: 1
    isPrimary: true
    isTrusted: true
    layerX: 94
    layerY: 14
    metaKey: false
    movementX: 0
    movementY: 0
    offsetX: 0
    offsetY: 0
    originalTarget: <button id="deactivate">
    pageX: 94
    pageY: 14
    pointerId: 0
    pointerType: "mouse"
    pressure: 0
    rangeOffset: 0
    rangeParent: null
    relatedTarget: null
    returnValue: true
    screenX: 371
    screenY: 165
    shiftKey: false
    srcElement: <button id="deactivate">
    tangentialPressure: 0
    target: <button id="deactivate">
    tiltX: 0
    tiltY: 0
    timeStamp: 37972
    twist: 0
    type: "click"
    view: Window http://127.0.0.1:5500/index.html
    which: 1
    width: 1
```

JavaScript Objects

- An object is a collection of properties, where each property is a key-value pair

```
let person = {
        name: "John",
        age: 30,
        greet: function () {
                console.log("Hello world!");
        },
};
```

JavaScript Objects

```
let person = {
		name: "John",
		age: 30,
		greet: function () {
				console.log("Hello world!");
		},
};
```

Accessing values:

```
person.name
person.age
person.greet()
```

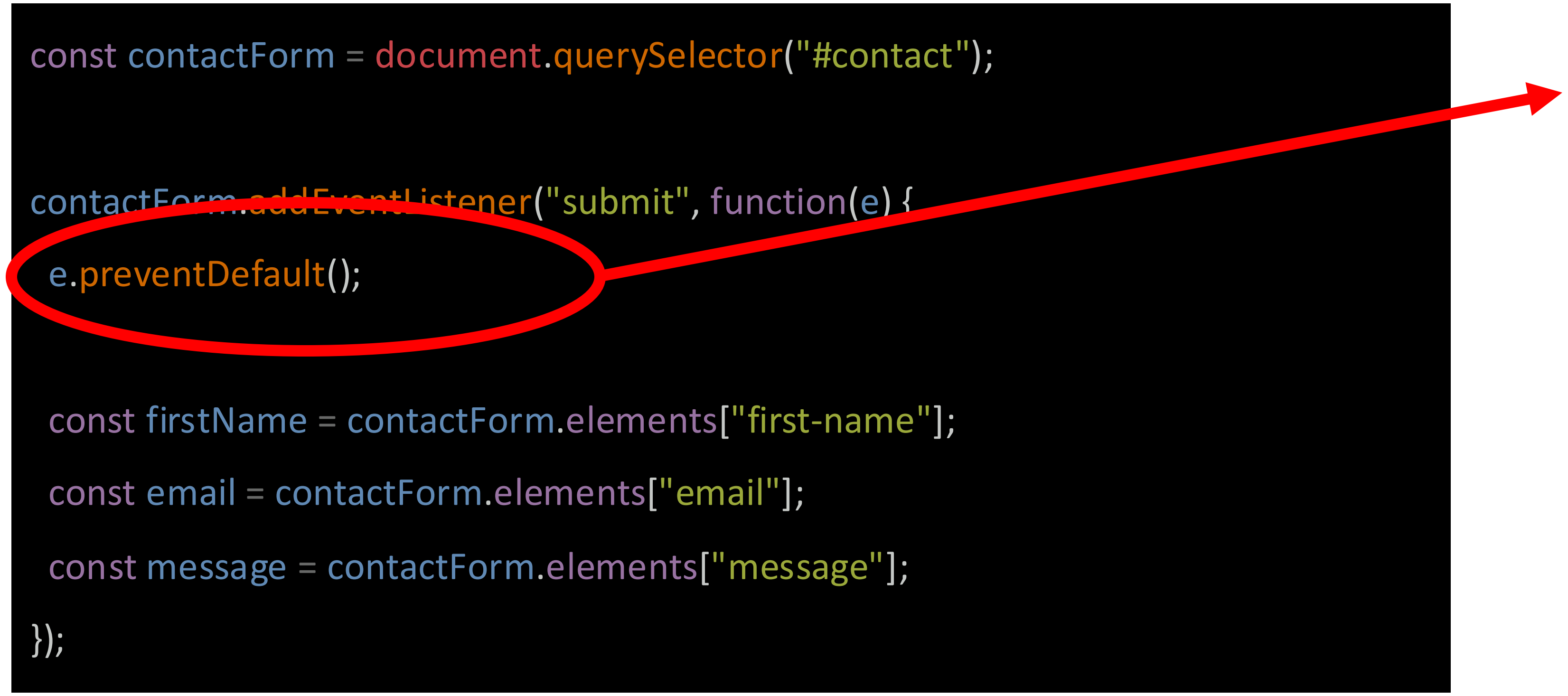
Accessing Event Details

- **event.target**: The element that triggered the event (e.g., the clicked element).
- **event.type**: The type of the event (e.g., "click", "keydown").
- **clientX**: The horizontal coordinate of the mouse pointer relative to the viewport.
- **clientY**: The vertical coordinate of the mouse pointer relative to the viewport.
- **pageX**: The horizontal coordinate of the mouse pointer relative to the entire document (including scroll offsets).
- **pageY**: The vertical coordinate of the mouse pointer relative to the entire document (including scroll offsets).

Working with HTML Forms

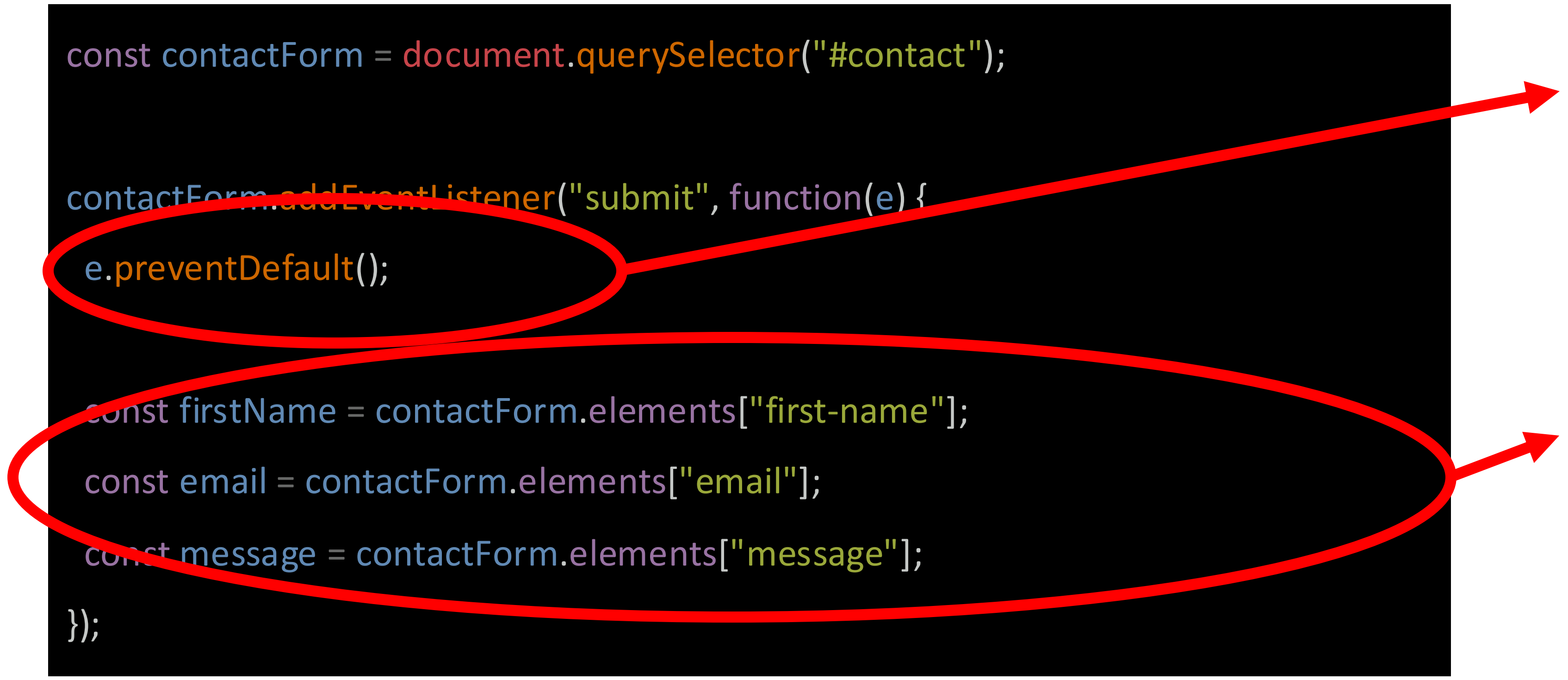
```html
<form id="contact">
  <label for="first-name">First Name</label>
  <input type="text" name="first-name" id="first-name" placeholder="First Name" />
  <label for="email">Email</label>
  <input type="email" name="email" id="email" />
  <label for="message">Message</label>
  <textarea name="message" id="message"></textarea>
  <input type="submit" />
</form>
```

```
const contactForm = document.querySelector("#contact");

contactForm.addEventListener("submit", function(e) {
  e.preventDefault();

  const firstName = contactForm.elements["first-name"];
  const email = contactForm.elements["email"];
  const message = contactForm.elements["message"];
});
```



```
const contactForm = document.querySelector("#contact");

contactForm.addEventListener("submit", function(e) {
  e.preventDefault();

  const firstName = contactForm.elements["first-name"];
  const email = contactForm.elements["email"];
  const message = contactForm.elements["message"];
});
```

Prevents the default action from happening
(page reload in this case)



```
const contactForm = document.querySelector("#contact");

contactForm.addEventListener("submit", function(e) {
  e.preventDefault();

  const firstName = contactForm.elements["first-name"];
  const email = contactForm.elements["email"];
  const message = contactForm.elements["message"];
});
```

Prevents the default action from happening
(page reload in this case)

Accessing data for each form field

Working with Individual Inputs & Keyboard Events

- Keyboard events can be listened to for the following events:
  - keydown: when a key is pressed down
  - keyup: when a key is released

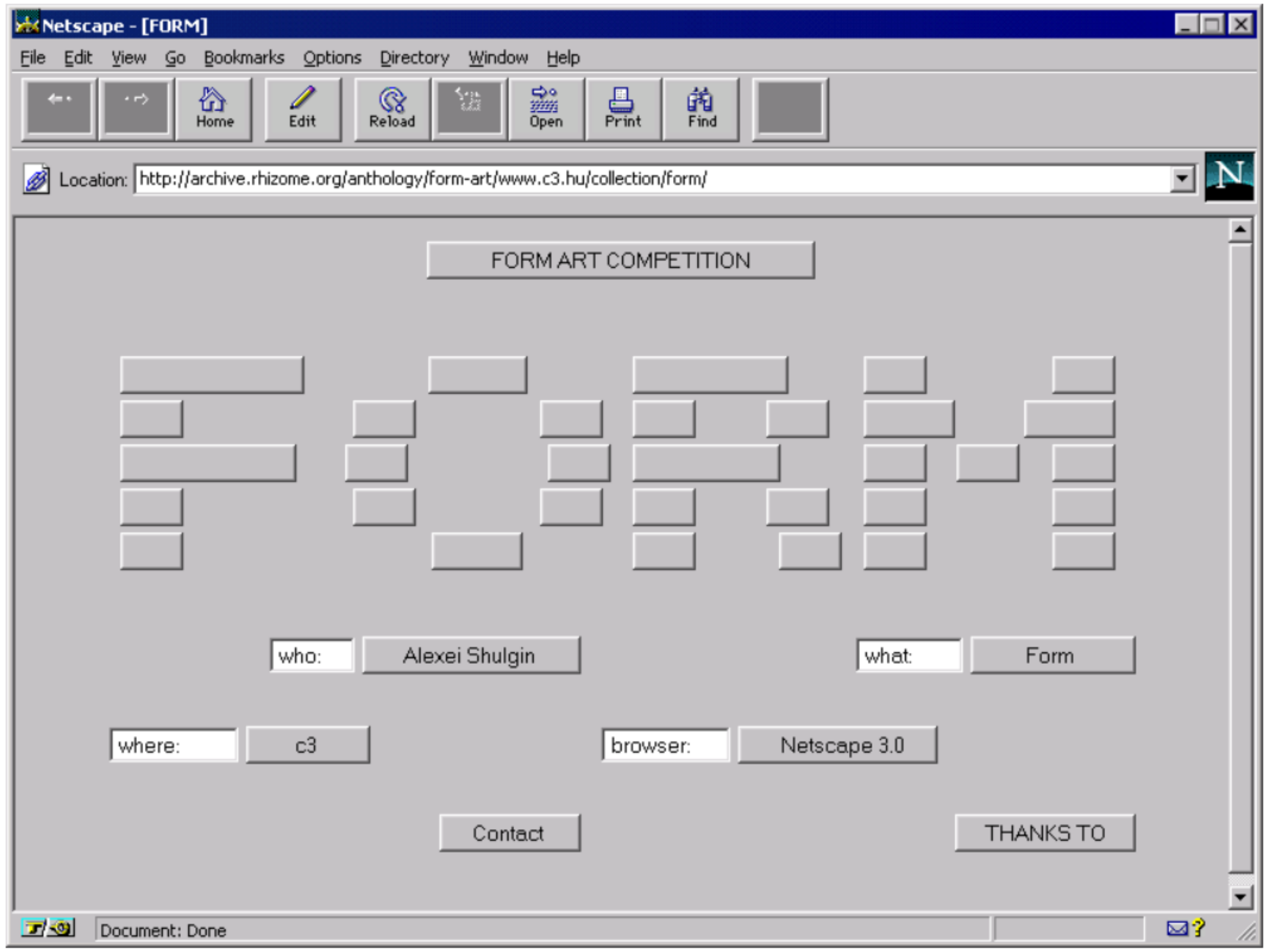
```
<input type="text" id="message" />



const message = document.querySelector("#message");


message.addEventListener("keydown", function (e) {
  console.log(e.key); // prints the input key character
});
```

In-Class Exercise 2

- Create an input field and an empty `<p>` tag in your HTML.
- Use JavaScript to attach an event listener to the input field that listens for the input event.
- When you type in the input field, display the typed text inside the `<p>` tag using `textContent`.
- Test it by typing in the input field and see the text appear in the `<p>` tag.

Exploring Friction on the Web

Forms as form


Netscape - [FORM]
File Edit View Go Bookmarks Options Directory Window Help
Home Edit Reload Open Print Find

Location: http://archive.rhizome.org/anthology/form-art/www.c3.hu/collection/form/

FORM ART COMPETITION

who:	Alexei Shulgin	what:	Form
where:	c3	browser:	Netscape 3.0

Contact

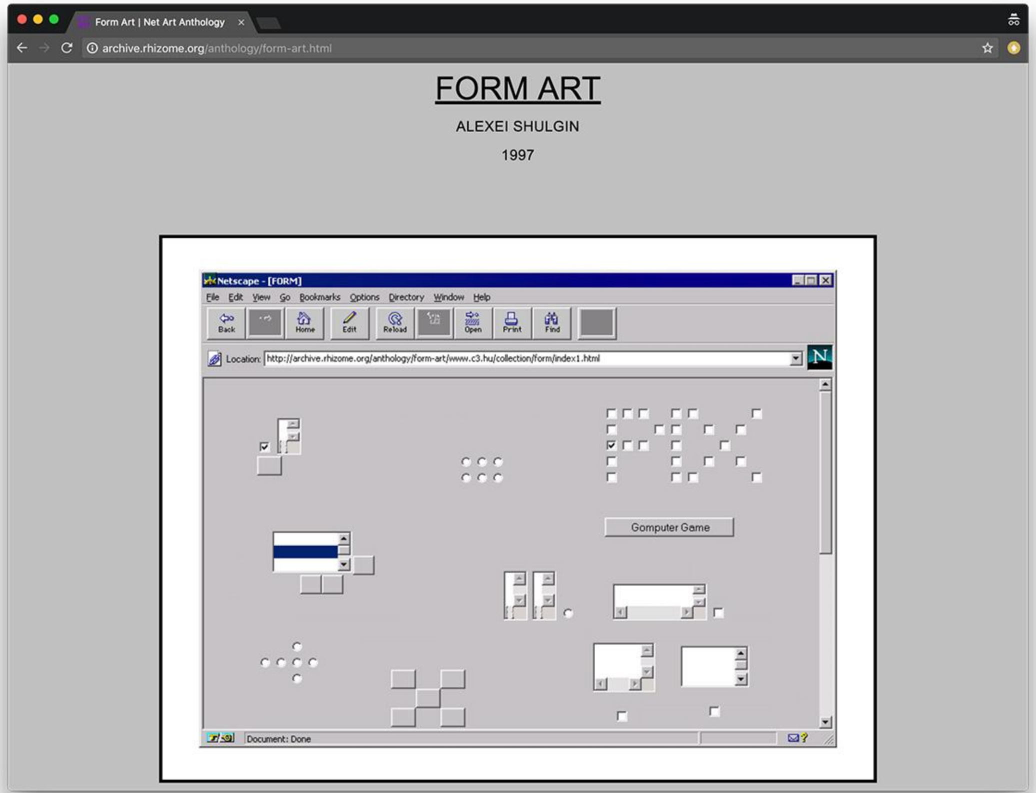
THANKS TO

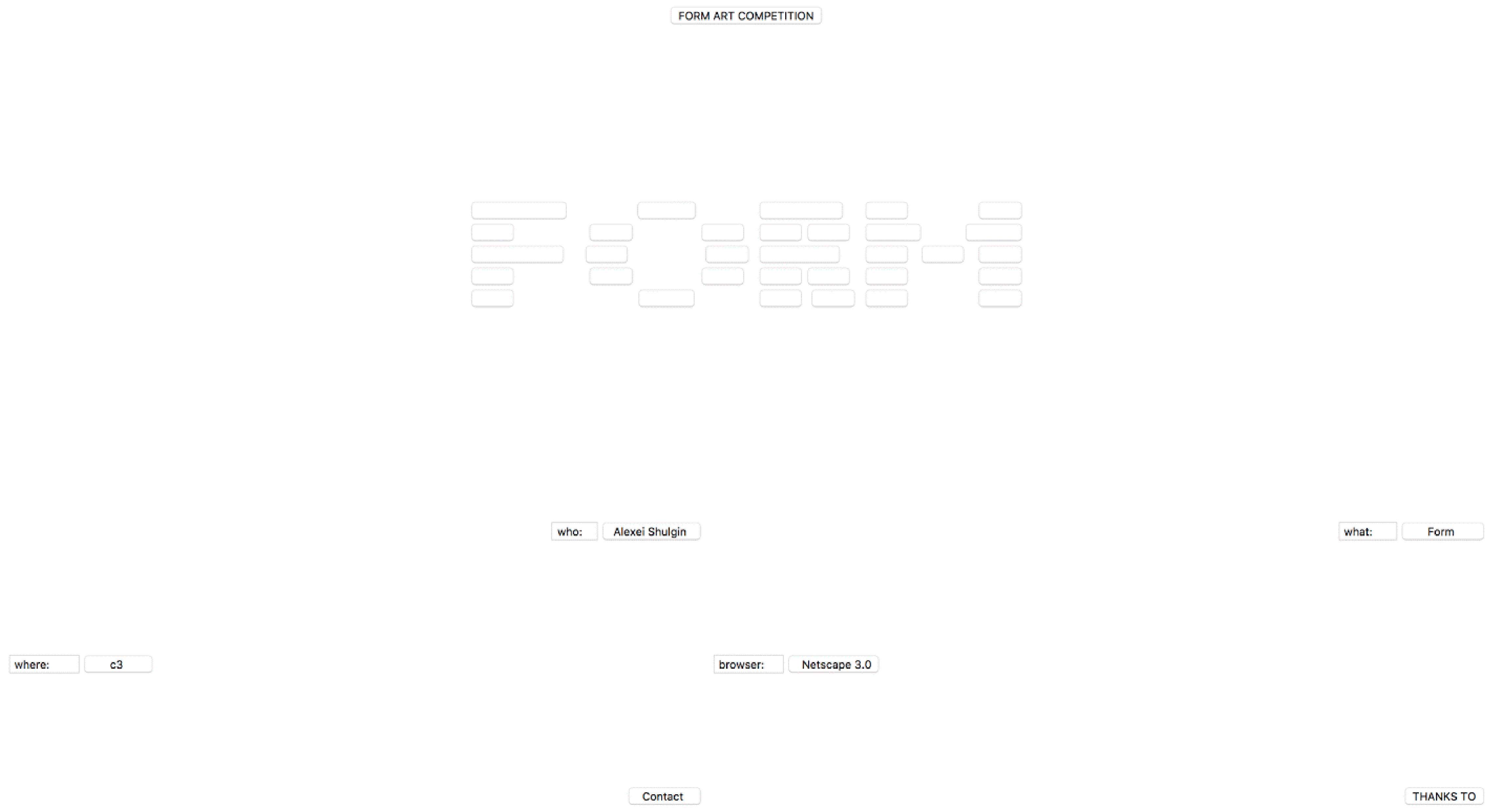
Document: Done

Form Art 1997

Russian artist Alexei Shulgin's *Form Art* (1997), which used HTML buttons and boxes as the raw material for monochromatic compositions, is at first glance a purely formal study of certain aspects of HTML. But it was also absurd: *Form Art* transformed the most bureaucratic, functional, and unloved aspects of the web into aesthetic, ludic elements.

https://anthology.rhizome.org/form-art


Form Art | Net Art Anthology
archive.rhizome.org/anthology/form-art.html
FORM ART
ALEXEI SHULGIN
1997
Netscape - [FORM]
Location: http://archive.rhizome.org/anthology/form-art/www.c3.hu/collection/form/index1.html
Computer Game
Document: Done



FORM ART COMPETITION

who: Alexei Shulgin

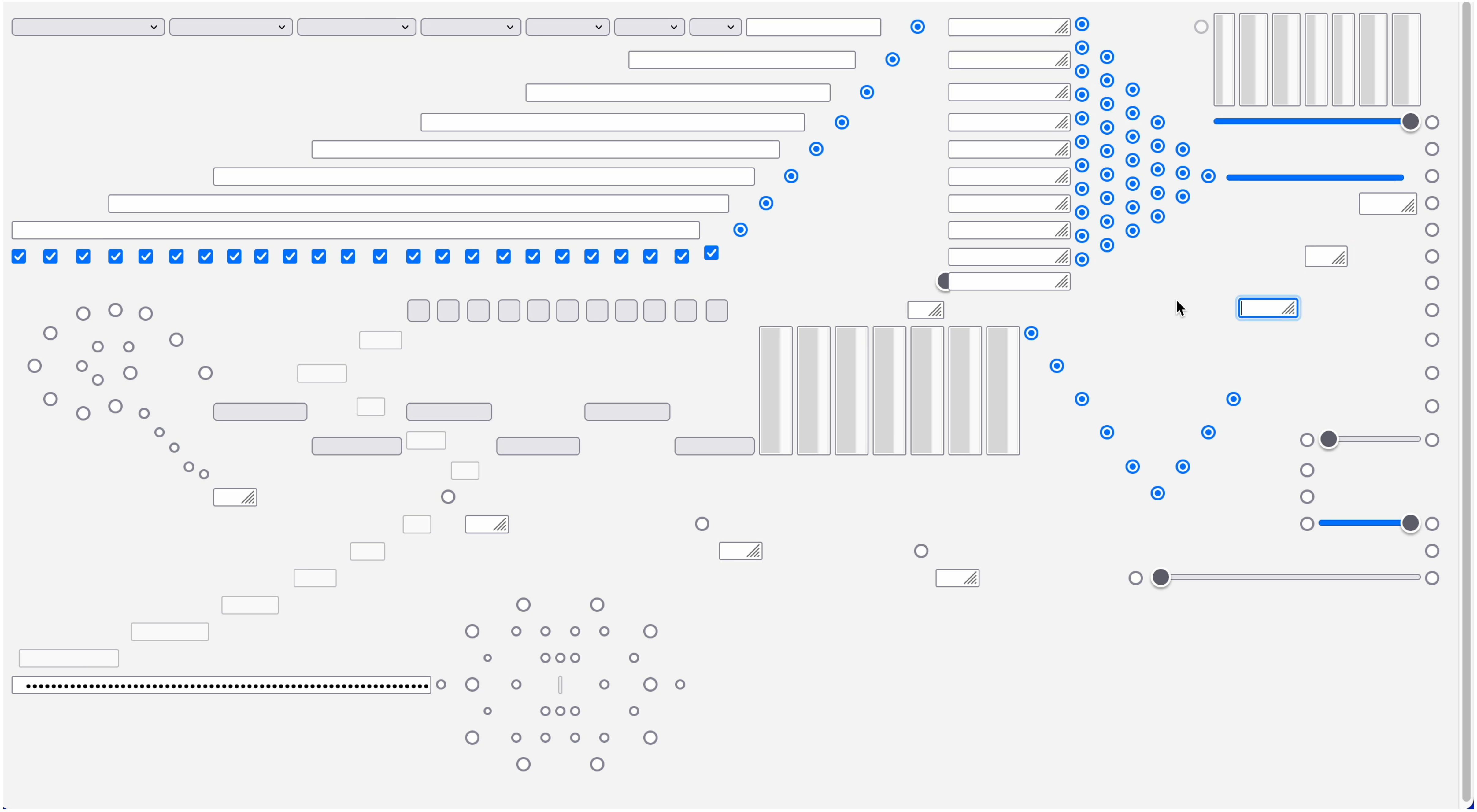
what: Form

where: c3

browser: Netscape 3.0

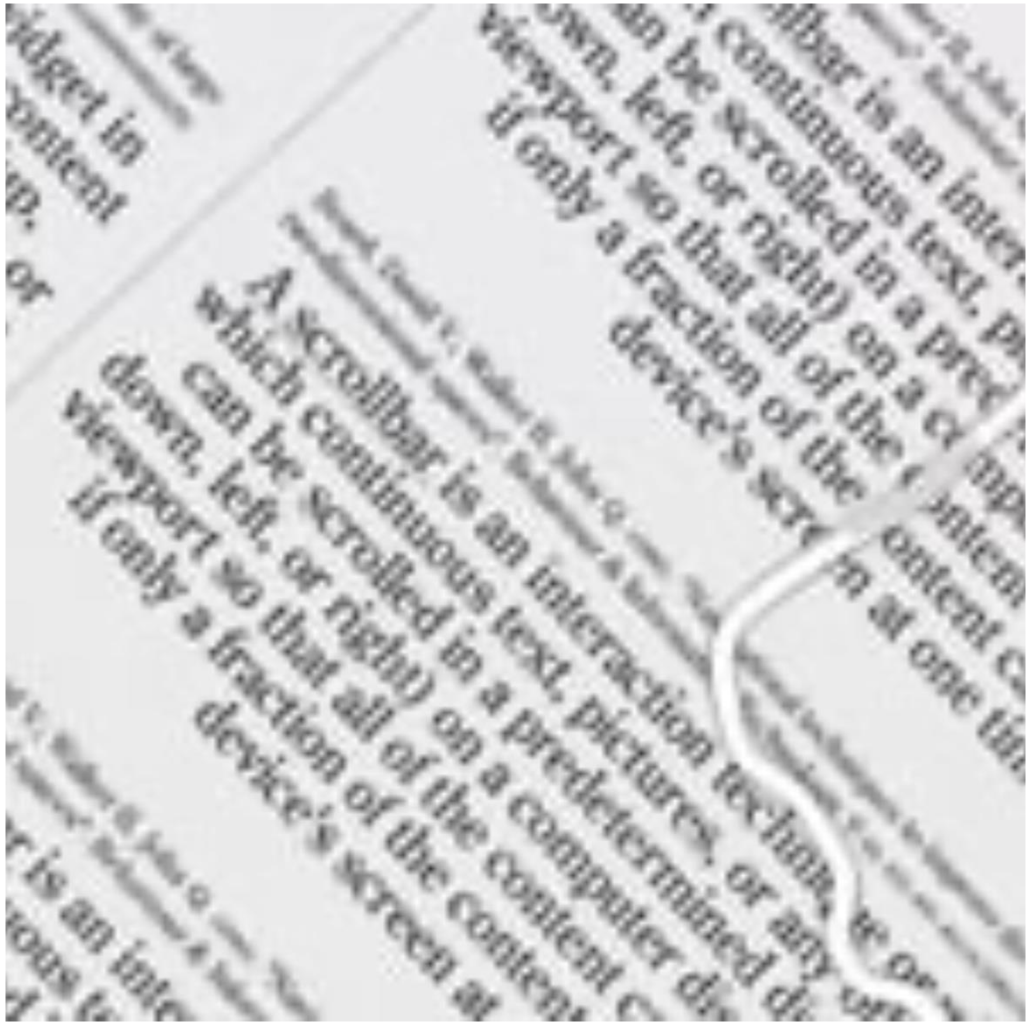
Contact

THANKS TO

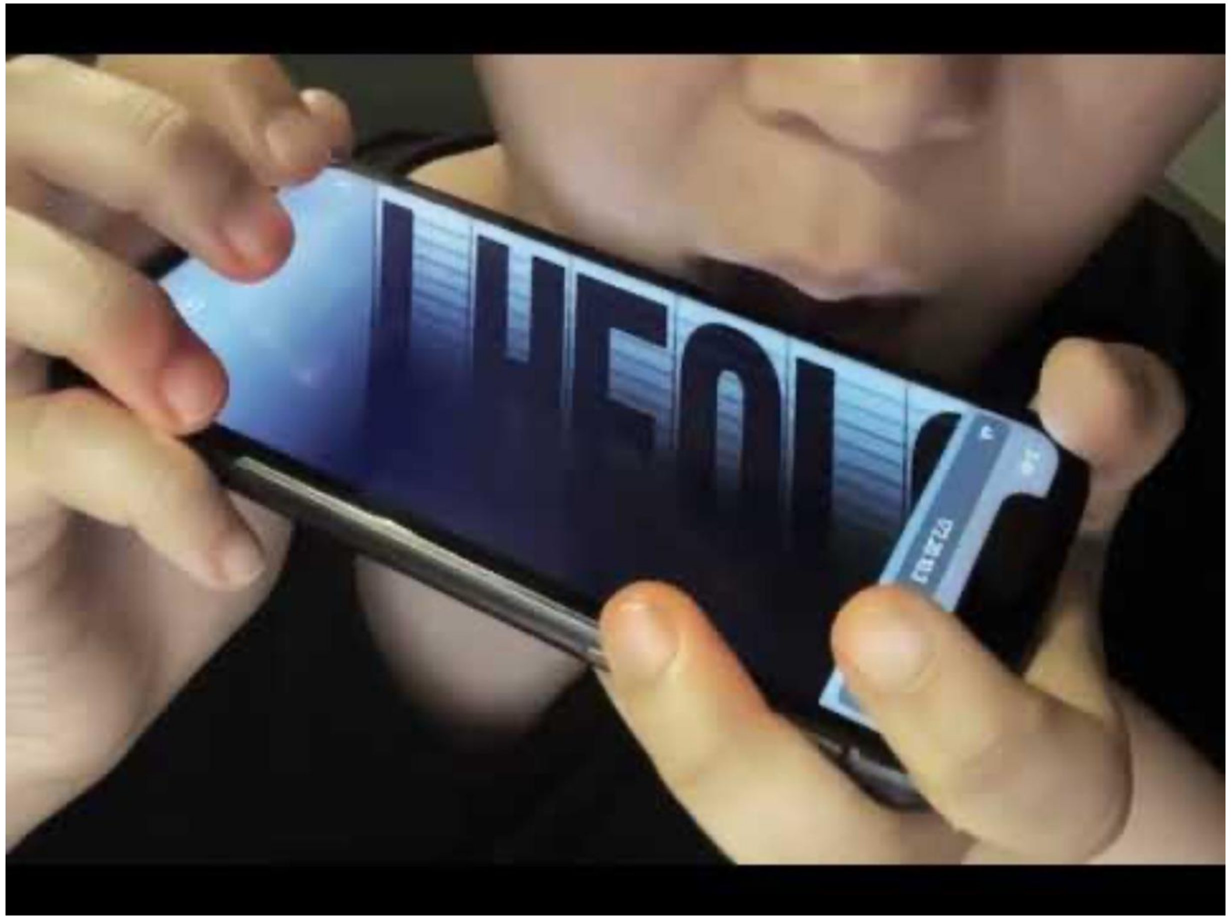


Artist Spotlight: Yehwan Song

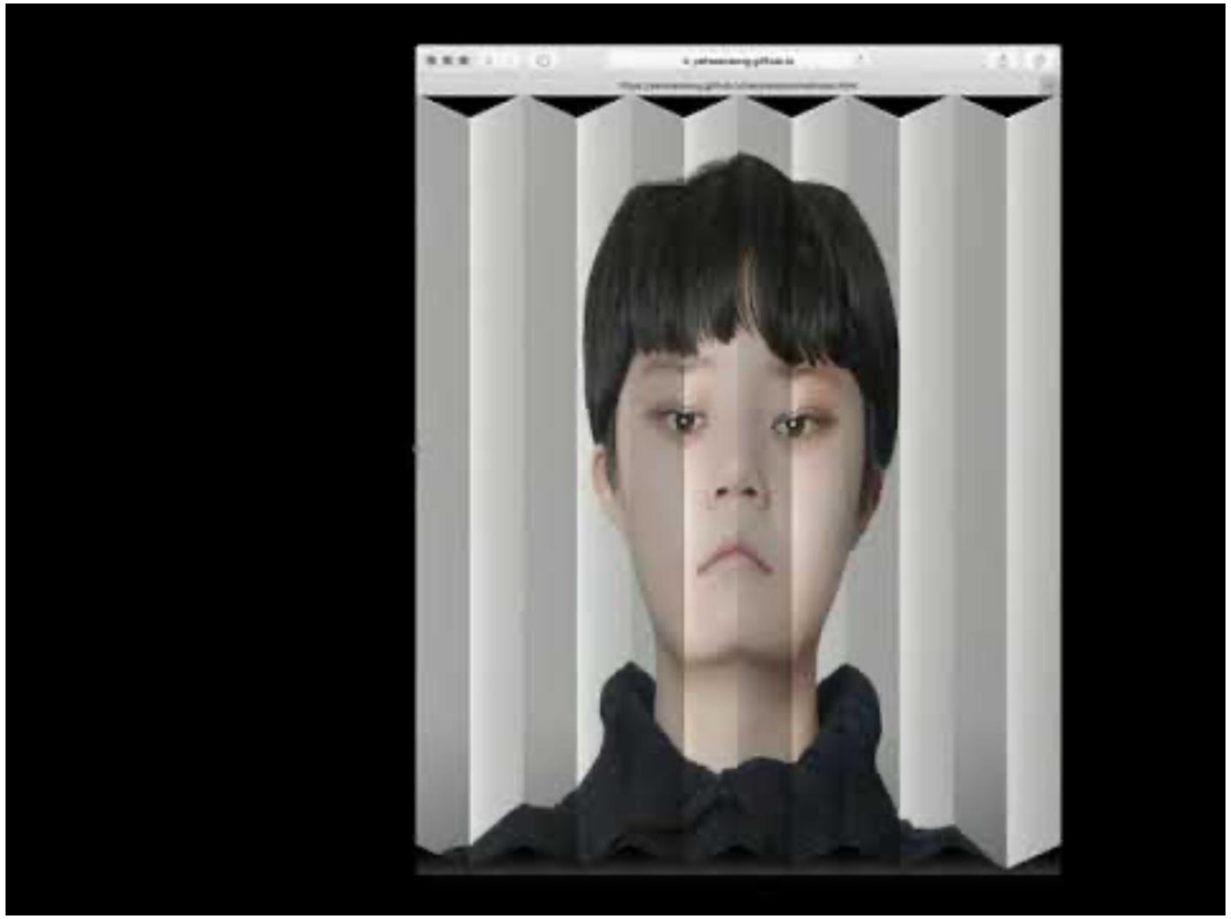
https://yhsong.com


A scrollbar is an interaction technique or
which continuous text, pictures, or any
can be scrolled in a predetermined di
down, left, or right) on a computer
viewport so that all of the content c
if only a fraction of the content
device's screen at

MT Everest Scroll Bar



HELLO



Very Responsive

Fountain Sculpture

Some more

- https://chongkiu33.github.io/ARECACEAE/
- https://otheroffice.net/
- https://amelieknopper.de/