EVENTS AND EVENT DRIVEN PROGRAMMING

QUESTION:

- What does a webpage do when the page finish loading, but the user hasn’t done anything yet?

- It waits for input, or other events
- Once an event happens then JavaScript code is executed.

SO FAR…

- Our programs have run as soon as the page loaded, and finished running before the user could interact the page
- So we have used prompt and alert to interact with the user
- This isn’t typical website behavior
- Typical website behavior waits for the user to do something (to take an action), and then responds in some way
- This type of action/response behavior is an Event Driven Programming model

THE EVENT LOOP

- JavaScript has a built-in event loop

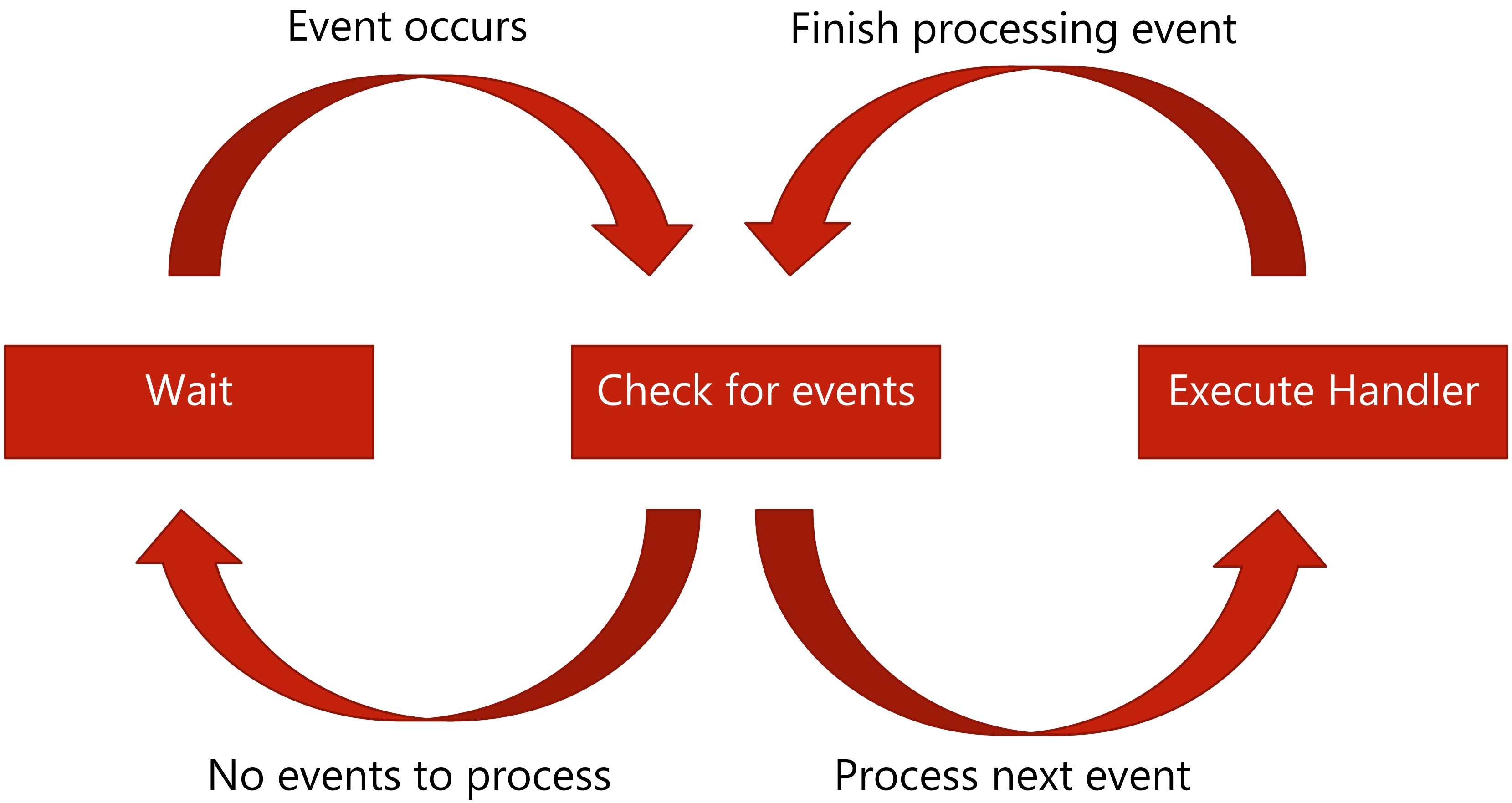
1. Once the page has loaded, any code that is outside of a function is executed
2. Then JavaScript then waits for events (action taken by user)

HOW EVENTS ARE HANDLED

When an event occurs

1. The event is added to the list of events to be handled
2. If JS is not currently handling an event
    1. JS looks for an event handler for the earliest event on the list
    2. The event handler is executed
    3. The event is removed from the list
    4. JS checks for the next event on the list
        - If there are no more events on the list, it goes back to waiting

EVENT LOOP


Event occurs

Finish processing event

Wait

Check for events

Execute Handler

No events to process

Process next event

EVENTS

- An event is anything that happens on a web page
  - Significant moments generated by the web browser
  - Includes any user action
  - Usually associated with an element
- Today we will examine two types of events
  - onload – event occurs once the HTML file has finished loading
  - onclick – event occurs when an element has been clicked

ONLOAD

- This event is associated with the document
- Once the HTML has loaded is great time to set up our JS
- How do we handle onload?
  - We add an onload attribute to the body of the HTML page

```
<body onload=“setup()”>
…
</body>
```

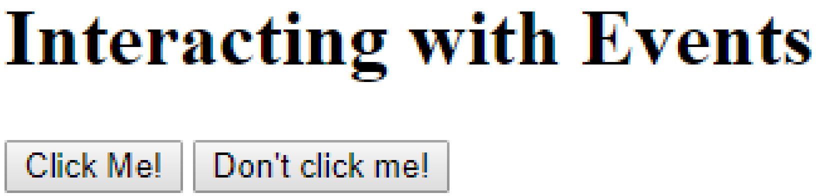
- This code calls the setup() function once the page has loaded
- Notice the JS is in quotations marks

BUTTONS

- There are two ways we can create a button in HTML

```
<button id=“b1”> Click Me </button>
```

```
<input type=“button” value=“Don’t click me”>
```

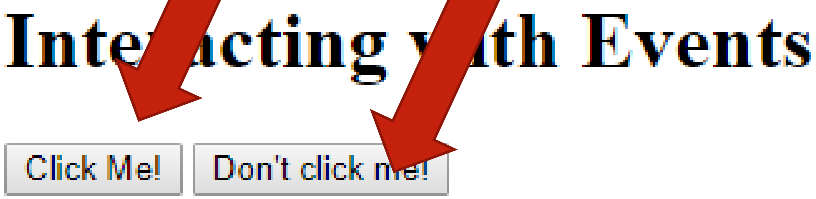


Interacting with Events

BUTTONS

- There are two ways we can create a button in HTML

```
<button id=“b1”> Click Me </button>

<input type=“button” value=“Don’t click me”>
```



Interacting with Events

ONCLICK

- This event is associated with an HTML element
    - Can click on a paragraph, a list, a div, a button
- Usually used for buttons, but can work with any element
- How do we handle onclick?
    - We add an **onclick attribute** to the target element

ONCLICK

```
<button onclick="deposit()">
    Make a deposit
</button>
```

- Then the onclick attribute determines what function runs when the button is clicked

```
<input type="button" onclick="deposit()">
```

ONCLICK

- We can have any function run when the button is clicked
- We should ALWAYS limit this code to a single function
- Technically you could place a full JS program in the quotes
- But using only one function call keeps the code tidy

ANOTHER WAY

- There is another way we can add the event handler to our code
- First we get the element from the HTML page
- Then we set the element’s event property to refer to our function

ANOTHER WAY

```
let element =
document.getElementById(“button1”);
element.onclick = deposit;
```

- Pure JavaScript
- NOTE: There are no brackets after deposit
- This is because we want to refer to the function, not invoke it at this time

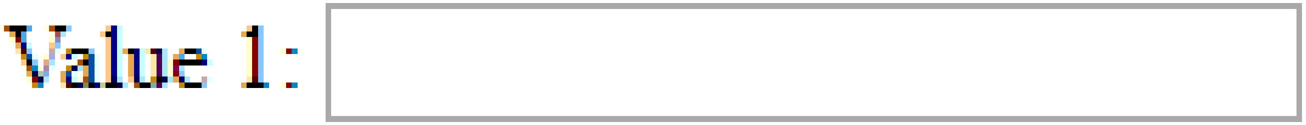
<INPUT>

- We can add inputs to the webpage
  - Like buttons and textfields
  - Both use <input> tag
- We can use the textfield rather than the prompt to accept user input!
- Much less irritating
- But in order to be able to use the contents of the textfield, we have to let the user type things, hence why we needed events

<INPUT>

- To create a textfield in our HTML

```
Value 1: <input type = “text” id=“box1”>
```



Value 1:

<INPUT>

- To create a textfield in our HTML

```
Value 1: <input type = “text” id=“box1”>
```

- To access the content of the textfield we access the element’s **value** property

```
let value = document.getElementById(“box1”).value;
```

- Keep in mind, just like prompt, that this value will **always be a string**

EXAMPLE

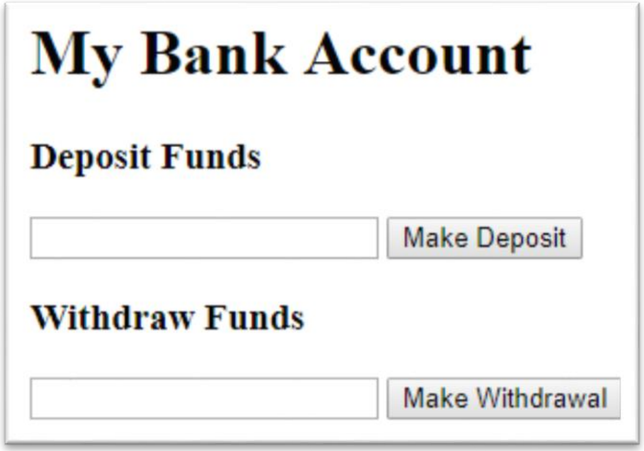
- Let’s create a web application that simulates a simple banking account application
- The account should have a balance, the ability to make deposits and withdrawals

START WITH HTML

```
<h1>My Bank Account</h1>
<h2 id="display"></h2>

<h3>Deposit Funds</h3>
<input type="text" id="dField">
<input type="button" value="Make Deposit">

<h3>Withdraw Funds</h3>
<input type="text" id="wField">
<input type="button" value="Make Withdrawal">
```

START WITH HTML

```
<h1>My Bank Account</h1>
<h2 id="display"></h2>

<h3>Deposit Funds</h3>
<input type="text" id="dField">
<input type="button" value="Make Deposit">

<h3>Withdraw Funds</h3>
<input type="text" id="wField">
<input type="button" value="Make Withdrawal">
```



My Bank Account

Deposit Funds

Withdraw Funds

NOW THE JAVASCRIPT

- We say the account should have:
    - A balance, what might that look like?

    - The ability to make a deposit

    - The ability to make a withdrawal

NOW THE JAVASCRIPT

- We say the account should have:
  - A balance, what might that look like?

  - The ability to make a deposit
    - Function to increase the balance by some amount

  - The ability to make a withdrawal
    - Function to decrease the balance by some amount

NOW THE JAVASCRIPT

- We say the account should have:
  - A balance, what might that look like?
    - A global variable that both functions can update
  - The ability to make a deposit
    - Function to increase the balance by some amount
  - The ability to make a withdrawal
    - Function to decrease the balance by some amount

LET’S CODE IT!

LAST STEP

- How do we connect the HTML buttons and textboxes to our JavaScript?

OTHER INPUT TYPES

- color
- password
- date
- file
- radio
- Range
- You’re free to investigate them
- The basic pattern is the same, you get the value from the input element for most input types, except buttons

PERSISTING INFORMATION

- If everything we code is inside functions
- And local variables go away once the function has executed
- Then information we want to keep around must exist inside global variables
- If everything is inside functions the our JavaScript program is now a collection of functions
- These functions can run in any order, which is usually determined by the user input
- Then these functions will change global variables to keep track of information

EXERCISE

- 1 mile = 1.60934km
- 1 km = 0.621371mi
- toKilometers() that accepts a positive number of miles and displays the equivalent number of kilometers on the page
- toMiles() that accepts a positive number of miles and displays the equivalent number of miles on the page
- Create two buttons such that when they are clicked each button triggers one of the above functions

ANOTHER WAY

```
document.addEventListener ("click" ,
     function(){
          console.log("You clicked!");
     }
);
```

- Add’s onclick listener to the entire HTML page
- This function registers its second argument to be called whenever the event described by its first argument occurs

EVENT LISTENER

- You can add an event listener to almost any HTML element
- Suppose I have a **`<p id=“p1”>`** in my HTML
- What do you think happens on this webpage?

```
function showMsg(){
     alert(“You clicked!”);
}

let para=document.getElementById(“p1”);
para.addEventListener(“click”,showMsg);
```

THE EVENT OBJECT

- Every time an event occurs an event object is generated
  - It holds additional information about the event
  - The info stored in the event object will differ, depending on what event just took place
  - We will talk about objects more in 2-3 weeks

- event.type will always hold a string of info identifying what event occurred (like “click” or “mousedown”)

KEY EVENTS

`event.key`

- Will contain the key value of the key represented by the event.
  - If the value has a printed representation, this attribute's value is the same as the char attribute
  - Otherwise, it's one of the key value strings “Alt” “CapsLock” “Control”
  - https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key/Key_Values

KEY EVENTS

`event.char`

- Will contain the character value of the key if the key corresponds to a printable character
  - If the key doesn’t correspond to a printable character (i.e., shift, alt, del) then event.code will contain the empty string

`event.code`

- Holds a string that identifies the physical key being pressed. The value is not affected by the current keyboard layout or modifier state, so a particular key will always return the same value

EXAMPLE

```
<script>
document.addEventListener("keydown", push);
function push(){
  console.log(event.code);
  if(event.key == "b")
    document.body.style.backgroundColor="blue";
}
</script>
```

- See example.html

KEYBOARD EVENTS

- There are three types of keyboard events: keydown, keypress, and keyup.
- For most keys the sequence of these key events is this:

1. When the key is first depressed, the keydown event is sent.
2. If the key is not a modifier key, the keypress event is sent and then the character is printed
3. When the user releases the key, the keyup event is sent.

KEY BOARD EVENTS

- When a key is pressed and held down, it begins to auto-repeat. This results in a sequence of events similar to the following being dispatched:

1. keydown
2. keypress
3. keydown
4. keypress
5. <<repeating until the user releases the key>>
6. keyup

IMPORTANT

- In this class we will almost always use keyup or keydown
- Only use keypress if you are trying to deal with **only** keys that actually produce characters

EXAMPLE

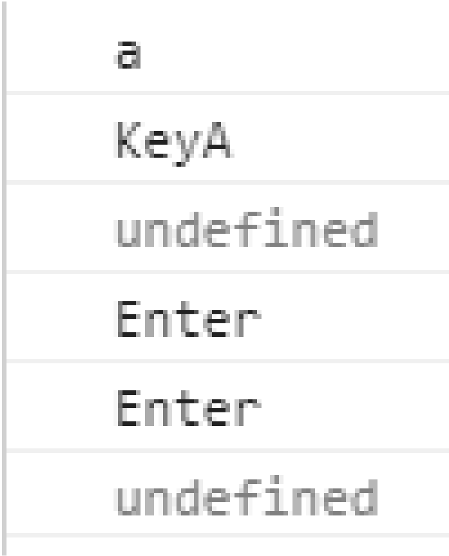
```
document.addEventListener("keypress", press);
function press(){
      console.log(event.key);
      console.log(event.code);
      console.log(event.char);
}
```

- What is printed to the console if I press a then enter then shift?

EXAMPLE

```
document.addEventListener("keypress", press);
function press(){
        console.log(event.key);
        console.log(event.code);
        console.log(event.char);
}
```

- What is printed to the console if I press a then enter then shift?

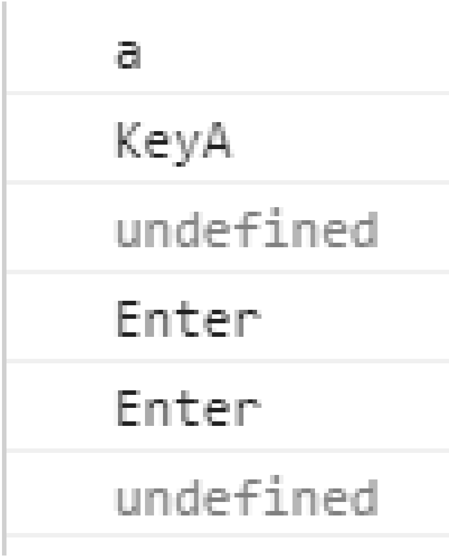


```
a
KeyA
undefined
Enter
Enter
undefined
```

EXAMPLE

```
document.addEventListener("keypress", press);
function press(){
        console.log(event.key);
        console.log(event.code);
        console.log(event.char);
}
```

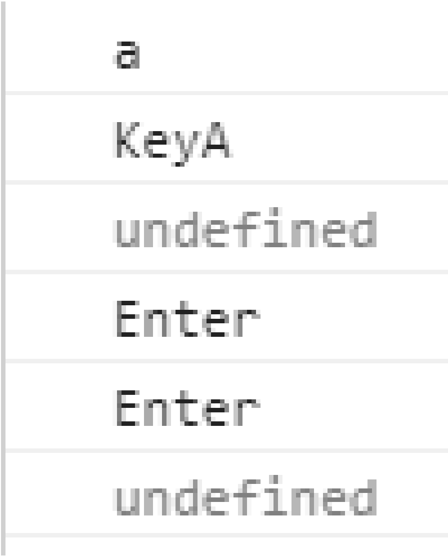
- What is printed to the console if I press a then enter then shift?
- Why did shift not appear?



```
a
KeyA
undefined
Enter
Enter
undefined
```

EXAMPLE

```
document.addEventListener("keypress", press);
function press(){
        console.log(event.key);
        console.log(event.code);
        console.log(event.char);
}
```

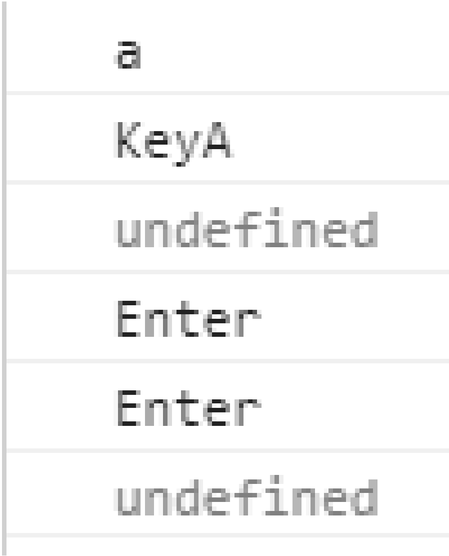


```
a
KeyA
undefined
Enter
Enter
undefined
```

- What is printed to the console if I press a then enter then shift?
- Why did shift not appear?
- Keypress only works for keys that produce characters

EXAMPLE

```
document.addEventListener("keypress", press);
function press(){
        console.log(event.key);
        console.log(event.code);
        console.log(event.char);
}
```

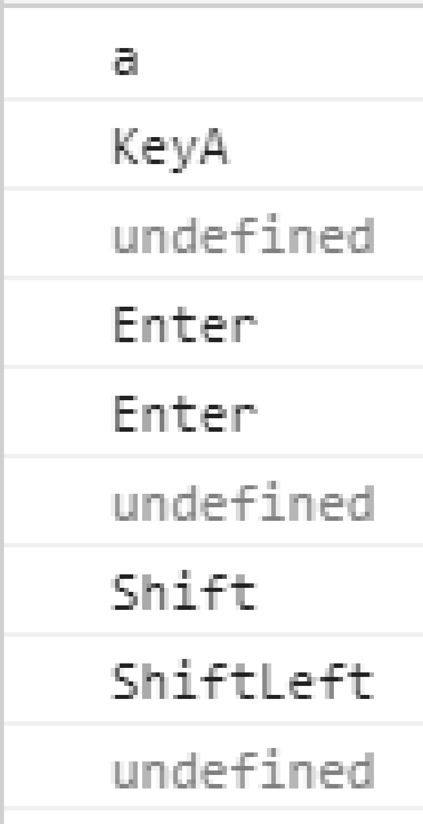


```
a
KeyA
undefined
Enter
Enter
undefined
```

- What is printed to the console if I press a then enter then shift?
- How would we fix this example so that the shift press is logged?

EXAMPLE

```
document.addEventListener("keydown", press);
function press(){
        console.log(event.key);
        console.log(event.code);
        console.log(event.char);
}
```



```
a
KeyA
undefined
Enter
Enter
undefined
Shift
ShiftLeft
undefined
```

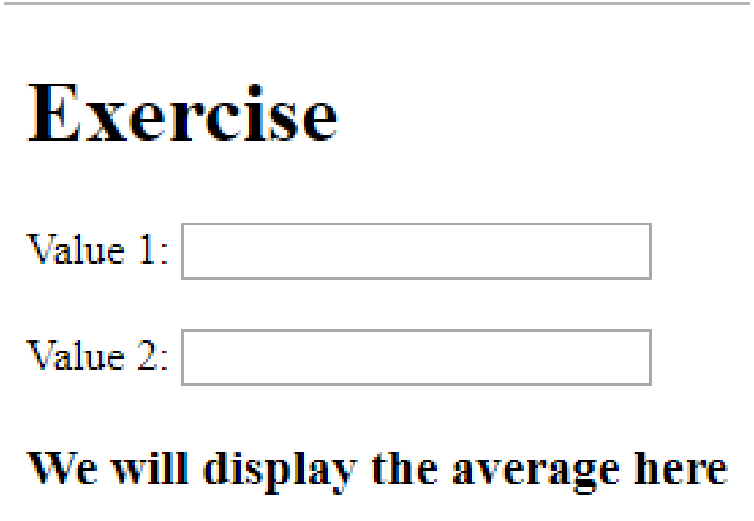
EVENT PROPAGATION

- Event handlers that are set on elements with children will also receive some events that happen in the children
  - Ex. If a button inside a paragraph is clicked, event handlers on the paragraph will also receive the click event
- In this example the outer element, the paragraph, is considered the parent element
- The inner element is considered the child element
- if both the paragraph and the button have a handler, the more specific handler, the one on the button, gets to go first

EVENT PROPAGATION

- The event is then said to **propagate** outward, from the element where it happened to that element’s parent element and so on until there are no more parent elements
- At any point, an event handler can call the stopPropagation method on the event object to prevent handlers “further up” from receiving the event
  - event.stopPropagation();
- This can be useful when, for example, you have a button inside another clickable element and you don’t want clicks on the button to activate the outer element’s click behavior

- Write the HTML to reproduce the page. Give each text field & the h3 a unique id name
- Write a keydown event handler that runs a function when a key is used in the second textfield
- When enter is pressed, the function should get the two numbers, lets call them val1 and val2, from the text fields
- The function should then display on the average of the two numbers on the webpage

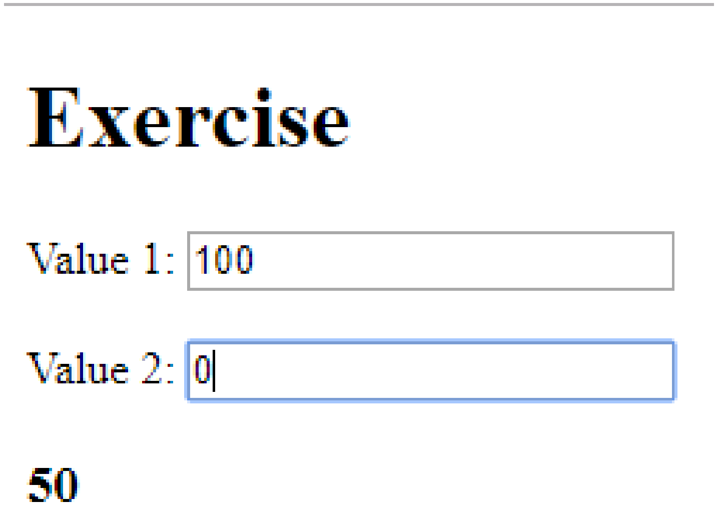


Exercise

Value 1:

Value 2:

We will display the average here



Exercise

Value 1: 100

Value 2: 0

50