Name _____ Period _____ Role (Circle one) Programmer/Driver

Name _____ Period _____ Role (Circle one) Programmer/Driver

Event Handlers

Your Tasks (Mark these off as you go)

- ☐ Explain the purpose of an event
- ☐ Register an event handler
- ☐ Register multiple handlers to a single event
- ☐ Have Ms. Pluska check off the above tasks
- ☐ Remove an event handler
- ☐ Explore mouse events
- ☐ Have Ms. Pluska check off the above tasks
- ☐ Explore keyboard events
- ☐ Have Ms. Pluska check off the above tasks
- ☐ Brainstorm a program
- ☐ Receive credit for the group portion of this lab

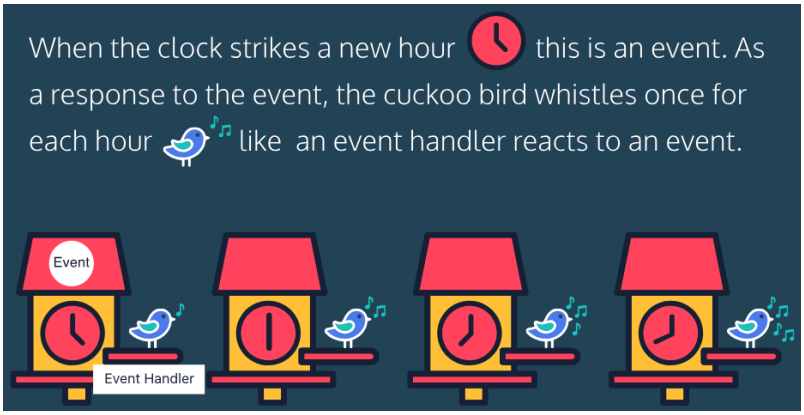
☐ Explain the purpose of an event

Events on the web are user interactions and browser manipulations that you can program to trigger functionality. Some examples of events are:

- A mouse clicking on a button
- Webpage files loading in the browser
- A user swiping right on an image

As an example, consider a cuckoo clock depicted below. When the clock hands strike a new hour, the cuckoo bird responds with a whistle for each hour. For example, the cuckoo bird will whistle twice when the clock strikes 2 o'clock.

As you can see in the diagram, the clock striking an hour is the specific event that causes a specific response from the cuckoo bird. Event handler functions wait for their specific events to fire like the cuckoo bird in the clock awaiting the next hour. These functions can be used to change a DOM element's color, text and much more!



In this lesson, you'll learn to use JavaScript with events to create dynamic websites.

Think about the objects you interact with in your life: your phone, car, etc. Provide a few examples of event – response interactions you encounter.

□ Register an event handler

HTML events are "**things**" that happen to HTML elements. Using javascript you can write functions to handle these interactions.

When an event handler function is created, it is attached to the DOM element being interacted with, or the *event target*. Check out the syntax:

```
var myButton = document.createElement("button");
myButton.innerHTML = "Click Me!";
document.body.append(myButton);

myButton.onclick = function() {
     myButton.innerHTML = "You clicked me!";
}
```

Let's break the code above,

1. First we created a button element and assigned it the variable *myButton*.
2. Next, we appended the button the body of the html document.
3. Then we created the event handler property which consists of the event target (*myButton*) followed by the event name (the prefix *on-* and the event type.) In this example, we're using the *click* event which fires when the user presses and releases the mouse on a DOM element, *myButton.*
4. Lastly, we assigned an event handler function to the property.

Check out the excerpt about JavaScript below. There is more information, or *moreInfo*, to be read when the user clicks the *readMore* element, but it does not appear to be working. You are going to create an event handler to fix this!

(a) First, create an event handler property for a click event that uses the *readMore* button as the event target.

(b) Now, you need to make the *moreInfo* element display when the click event fires. Create a statement that changes the *.display* style property of the *moreInfo* element to 'block'.

```
var readMore = document.getElementById('read-more');
var moreInfo = document.getElementById('more-info');
```

□ Register multiple handlers to a single event

It's best practice to create named event handler functions, instead of anonymous functions. Your code will remain organized and reusable this way, even if your code gets complex. Check out the syntax:

```
var eventHandlerFunction = function() {
      // this block of code will run
}

eventTarget.onclick = eventHandlerFunction;
```

The .addEventListener() method is another common syntax for registering event handlers. An event listener waits for a specific event to occur and calls a named event handler function to respond to it. This method requires two arguments:

1. The event type as a string
2. The event handler function

Check out the syntax of an .addEventListener() method with a clickevent:

```
eventTarget.addEventListener('click', eventHandlerFunction);
```

You'll want to use the *.addEventListener()* method to allow multiple event handlers to be registered to a single event without changing its other event handlers.

In the previous example, you wrote code to display additional information when the *readMore* button was clicked. But, now we want more to happen when *readMore* is clicked. To register more than one handler to be assigned to this click event, you will need to use the .addEventListener() method.

(a) Write a function that changes the .display style property of the *moreInfo* element to 'block'. Assign this function to a variable called *showInfo*.
(b) Write another function that changes the text of readMore to "Read Less". Assign this function to readLess.
(c) Now, use the addEventListener() method to register both of these handlers to the readMore button when it is clicked.

```
var readMore = document.getElementById('read-more');
var moreInfo = document.getElementById('more-info');
```

☐ Have Ms. Pluska check off the above tasks

Before you continue have Ms. Pluska check off the above tasks

Do not continue until you have Ms. Pluska's (or her designated TA's) signature _____

☐ Remove an event handler

The *.removeEventListener()* method is used to reverse the .addEventListener() method. This method stops the code from "listening" for an event to fire when it no longer needs to. *.removeEventListener* also passes two arguments:

1. The event type as a string
2. The event handler function

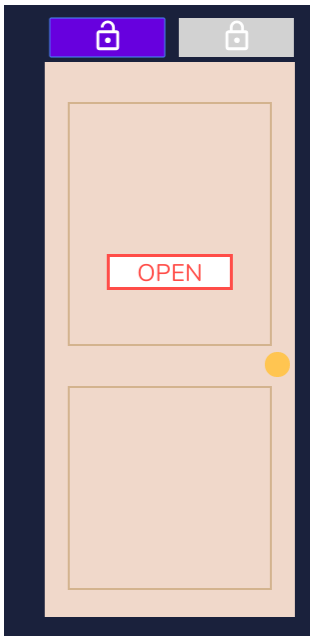
Check out the syntax of a *.removeEventListener()* method with a clickevent:

```
eventTarget.removeEventListener('click', eventHandlerFunction);
```

Because this method unregisters event handlers, it needs to identify which function to remove from the event. The event handler function passed to the *.removeEventListener()* method **must** be the same function of the corresponding *.addEventListener()*.

This digital restaurant needs your help because the lock on the door is broken! People are able to get in even when the restaurant is closed. You need to use the *.removeEventListener()* to keep the door locked. **First**, you must add an event listener to the *lock* element when a click event is fired with an anonymous function. **Next**, inside the function, add a *.removeEventListener()* to turn off the *openDoor* function when a user tries to click the *door* element. Then run your code and fire the event to test out your event handlers.

```
var door = document.getElementById('door');
var unlock = document.getElementById('unlock');
var lock = document.getElementById('lock');
var sign = document.getElementById('sign');

var openDoor = function() {
  door.hidden = true;
}

var closeDoor = function(){
  door.hidden = false;
}

unlock.onclick = function() {
  sign.innerHTML = 'OPEN';
}

lock.onclick = function() {
  sign.innerHTML = 'CLOSED';
}

unlock.addEventListener('click', function(){
  door.addEventListener('click', openDoor);
});
//Add your code below
```



□ Explore mouse events

Beyond the click event, there are all types of DOM events that can fire in a browser! It's important to know most events in the DOM take place without being noticed because there are no event handlers connected to them.

It's also important to know some registered events don't depend on user interactions to fire. For instance, the *load* event fires after website files completely load in the browser. Browsers can fire many other events without a user — you can check out a list of events on the MDN Events Reference page.

(a) Navigate to the MDN Events Reference page – you can do this on your phone or computer – https://developer.mozilla.org/en-US/docs/Web/Events. Once there, scroll to the section on mouse events.
(b) The code below creates a simple div element on the screen. Write three different functions. Assign each function to the following variables: changeColor, changeSize, changeText. In the body of each function, write code to change the backgroundColor, the size, and font color.
(c) Write three event listeners for each of the above functions. Each should fire for different mouse events.

```
var width = 200;
var height = 200;
var moreInfo = document.createElement("div");
moreInfo.innerHTML = "JavaScript can also handle requests and responses
on a website. It's a great language to master for front-end and back-end
web development.";
moreInfo.style.width = width + "px";
moreInfo.style.height = height + "px";
moreInfo.style.backgroundColor = "red";
moreInfo.style.textAlign = "center";
moreInfo.style.padding = "10px";
moreInfo.style.margin="10px";
document.body.append(moreInfo);
```

□ Have Ms. Pluska check off the above tasks

Before you continue have Ms. Pluska check off the above tasks

Do not continue until you have Ms. Pluska's (or her designated TA's) signature

_____

□ Explore keyboard events

Another popular type of event is the keyboard event! keyboard events are triggered by user interaction with keyboard keys in the browser.

The *keydown* event is fired while a user presses a key down.

The *keyup* event is fired while a user releases a key.

The *keypress* event is fired when a user presses a key down and releases it. This is different from using *keydown* and *keyup* events together, because those are two complete events and *keypress* is one complete event

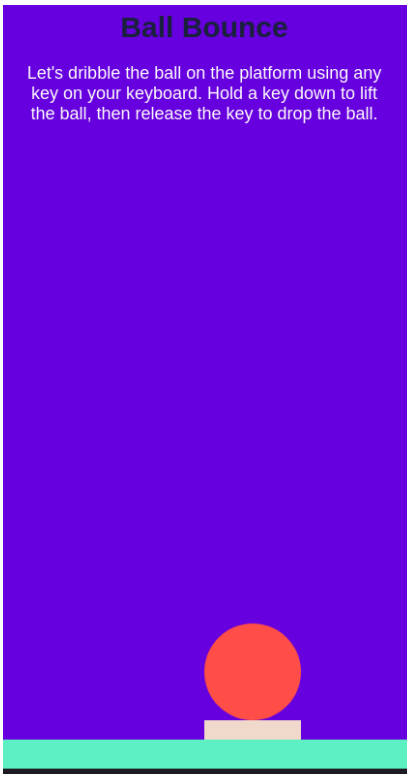
.

Keyboard events have unique properties assigned to their event objects like the *.key* property that stores the values of the key pressed by the user. You can program the event handler function to react to a specific key, or react to any interaction with the keyboard.

Now it’s time to create a game! Program this code to dribble the ball on the platform using any key on a keyboard. When a user presses a key down, it should lift the ball up. When the user releases the key, the ball should drop.

First, make a function named *up* that will raise the *ball* to '250px' from the bottom of the page. **Next**, make a function named *down* that will change the position of the ball to ‘50px’ from the bottom of the page. **Finally**, add two even listeners. The first event handler should run the *up* function when a *keydown* event fires, the second should run the *down* function when a *keyup* even fires anywhere on the document.

```
var ball = document.getElementById('circle);
```



☐ Have Ms. Pluska check off the above tasks

Before you continue have Ms. Pluska check off the above tasks

Do not continue until you have Ms. Pluska’s (or her designated TA’s) signature

_____

□ Brainstorm a program

With your partner brain storm a “hide and seek” game. Your game will have a grid of 9 boxes, one box will contain a monster that is initially hidden from the user. Your game will also have a score that initially should be set to 10.

Each box will have an event handler which will respond when the user clicks on it. If the box the user clicks on does not contain the monster you must substract 1 from the score, you will also change the color of the box so the user knows they have already clicked on that box. When the user clicks on the box with the monster, the monster should appear!

Compare your ideas with your partner, then obtain a large piece of butcher paper and a marker. Write your final code on this paper.

□ Receive Credit for the group portion of this lab

- Indicate the names of all group members.
- Make sure both you and your partner have completed the above tasks
- Have Ms. Pluska check off the group tasks
- Submit your lab to the needs to be graded folder to receive credit for the group portion of this lab.
- Do not submit your lab until you have Ms. Pluska’s (or her designated TA’s) signature

_____