

Client-Side Web Development

Class 10.1

Today's Topics

- Functions
- Event Handling
- **Exercise:** Event Horizon

Announcements

Web Portfolio

Any Questions?

Functions

A *function* is set of statements that can be used to perform a task some time in the future

A *function* can be executed multiple times

A *function* should be declared
before it can be called.

Declaring a Function

A *function definition* starts with the function keyword followed by a name, set of parentheses, and a set of curly braces.

**The `return` statement is used to
specify the value a function will
return after complete its task**

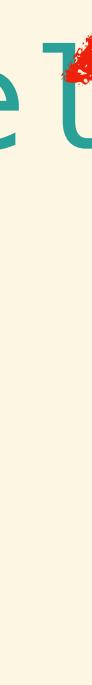
```
// Function definition
function greeting () {
    return `Hello World`
}
```

// Function definition

```
function greeting () {  
    return `Hello World`  
}
```


The function keyword

```
// Function definition  
function greeting () {  
    return `Hello World`  
}
```



The name of the function

```
// Function definition  
function greeting () {  
    return `Hello World`  
}
```



Set of parenthesis used to
hold parameters

// Function definition

function greeting () {

 return 'Hello World'

}

Set of curly braces used to
hold the code block

// Function definition

```
function greeting () {
```

```
    return
```

```
        `Hello World`
```

```
}
```

Returns any value that proceeds it,
when the function is called

Calling a Function

A *function* does not execute until it
is called

To *call a function*, use the function's name or variable followed by a set of parenthesis

```
// Function definition
function greeting () {
    return `Hello World`  

}
```

```
// Calling the function
greeting() // Hello World
```

```
// Function definition  
function greeting () {  
    return 'Hello World'  
}
```

Use the function name to
call the function

// Calling the function
greeting() // Hello World

Function Scope

**Variables declared inside a function
can only be accessed by the
function and its children**

**Variables declared outside of any
function or block are accessible
from inside a function**

```
// Creating a global variable
const who = `World`


// Function definition
function greeting () {
    // Creating a function variable
    const salutation = `Hello`
    return `${salutation} ${who}`
}

console.log(greeting()) // Hello World
console.log(salutation) // Error
```

// Creating a global variable

```
const who = `World`
```

A global variable is one that
is declared outside of a
block or function

// Function definition

```
function greeting () {
```

// Creating a function variable

```
    const salutation = `Hello`
```

```
    return `${salutation} ${who}`
```

```
}
```

```
console.log(greeting()) // Hello World
```

```
console.log(salutation) // Error
```

// Creating a global variable

```
const who = `World`
```

// Function definition

```
function greeting () {
```

// Creating a function variable

```
    const salutation = `Hello`
```

```
    return `${salutation} ${who}`
```

```
}
```

Global variables can be
used inside of a function

```
console.log(greeting()) // Hello World
```

```
console.log(salutation) // Error
```

```
// Creating a global variable
const who = `World`  
  
// Function definition
function greeting () {  
  
    // Creating a function variable  
    const salutation = `Hello`  
  
    return `${salutation} ${who}`  
}
```

Variables declared inside a function have function scope

```
console.log(greeting()) // Hello World
console.log(salutation) // Error
```

```
// Creating a global variable
```

```
const who = `World`
```

```
// Function definition
```

```
function greeting () {
```

```
    // Creating a function variable
```

```
    const salutation ← `Hello`
```

```
    return `${salutation} ${who}`
```

```
}
```

Variables declared inside a
function cannot be used outside
of the function

```
console.log(greeting()) // Hello World
```

```
console.log(salutation) // Error
```

Event Handling

DOM Events

DOM Events are notifications that some action has occurred on the page.

DOM Events can represent a basic user action or the status of the render model.

**There are *DOM Events* for the
keyboard, mouse, touch, clipboard,
media, view, printing, drag & drop,
animation, forms, and more**

To have JavaScript respond to ***DOM Events*** you must add an event listener to an element

Event Listeners

Event Listeners are JavaScript objects that
listens for a specific DOM Event to occur and
executes a function when it does

The `addEventListener()` Method

This method listens for a specified event to occur on a specified element and then executes the provided function

**This method requires an event type
and a function**

```
const $button = document.getElementById('button')

$button.addEventListener('click', function () {
  alert(`You pressed the button!`)
})
```

Event Types

Mouse Events

Mouse Events

- mouseenter
- *mouseover*
- mousemove
- mousedown
- mouseup
- auxclick
- *click*
- dblclick
- contextmenu
- wheel
- mouseleave
- *mouseout*
- select

```
const $button = document.getElementById('button')

$button.addEventListener('mouseover', function () {
    alert(`Don't you do it!`)
})

$button.addEventListener('click', function () {
    alert(`You pressed the button!`)
})

$button.addEventListener('mouseout', function () {
    alert(`Don't do it again!`)
})
```

```
const $button = document.getElementById('button')
```

```
$button.addEventListener('mouseover', function () {  
    alert(`Don't you do it!`)  
})
```

```
$button.addEventListener('click', function () {  
    alert(`You pressed the button!`)  
})
```

```
$button.addEventListener('mouseout', function () {  
    alert(`Don't do it again!`)  
})
```

Listening for three different events to occur

Keyboard Events

Keyboard Events

- `keydown`
- `keypress` (ignores modifier keys)
- `keyup`

Getting Key Codes

Deprecated

- `event.keyCode` (100%)
- `event.charCode`
- `event.which`

Not Fully Supported

- `event.key` (85%)
- `event.code` (48%)

Checking for Modifier Keys

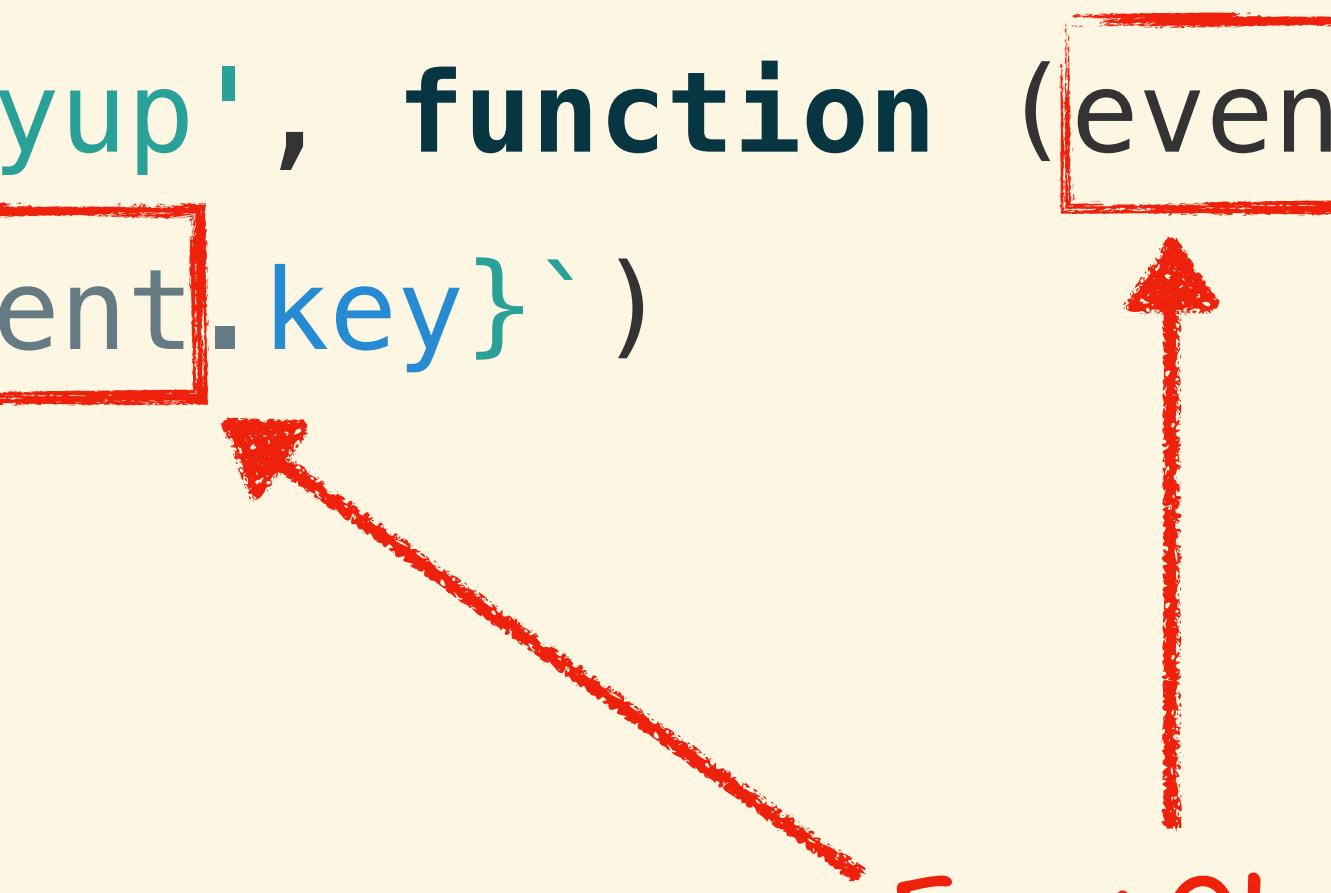
- `event.ctrlKey`
- `event.shiftKey`
- `event.altKey`
- `event.metaKey` (Not Fully Supported)

```
const textbox = document.getElementById('textbox')

textbox.addEventListener('keyup', function (event) {
    console.log(`You typed ${event.key}`)
})
```

```
const $textbox = document.getElementById('textbox')
```

```
$textbox.addEventListener('keyup', function (event) {  
    console.log(`You typed ${event.key}`)  
})
```



Event Object

```
const $textbox = document.getElementById('textbox')

$textbox.addEventListener('keyup', function (event) {
    console.log(`You typed ${event.key}`)
})
```



event.key



Represents the key
pressed as a string

Other Standard Events

Other Standard Events

- blur
- change
- copy
- cut
- focus
- invalid
- load
- paste
- reset
- resize
- select
- submit

Adding to Multiple Elements

**There will be times when you will need to add
an event listener to multiple elements**

This can be done using a loop

```
<div id="box"></div>  
  
<button class="button">red</button>  
<button class="button">green</button>  
<button class="button">blue</button>
```

```
const $box = document.getElementById('box')
const $buttons = document.querySelectorAll('.button')

for (const $button of $buttons) {
    $button.addEventListener('click', function (e) {
        $box.style.background = e.target.textContent
    })
}

}
```

```
const $box = document.getElementById('box')
const $buttons = document.querySelectorAll('.button')

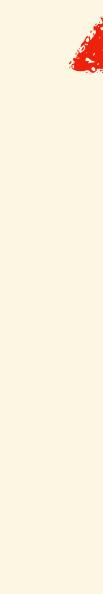
for (const $button of $buttons) {
    $button.addEventListener('click', function (e) {
        $box.style.background = e.target.textContent
    })
}
```



Event Object

```
const $box = document.getElementById('box')
const $buttons = document.querySelectorAll('.button')

for (const $button of $buttons) {
    $button.addEventListener('click', function (e) {
        $box.style.background = e.target.textContent
    })
}
```



The Button that was clicked

Examples

Exercise: Event Horizon

For next class....

- **Review:** Dominoes
- **Lab:** Domino's