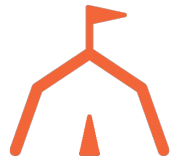
Introduction to Front-End

Today's Agenda

1. Download software
2. HTML
3. CSS
4. JavaScript



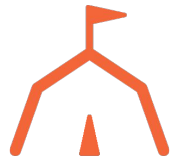
Download Development Environment

This is where we're going to write our code together. Please go to

code.visualstudio.com/download

and download VS Code.

HTML



What is HTML?

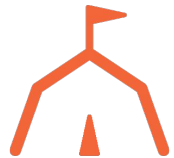
- Hypertext Markup Language
- The way in which we write (or mark up) a web page or web application.
- Developed in 1989 by Tim Berners-Lee.
- HTML can be thought of as the underlying structure and content of our webpage.

HTML

1. HTML **elements** - individual components
2. HTML **tags** - containers for elements
3. HTML **attributes** - special descriptions

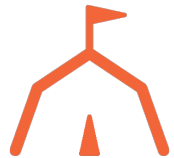
HTML Tag

HTML tags allow the webpage know what type of content to display on the browser.



HTML Tag

```
<p></p>

<img>

<div></div>
```

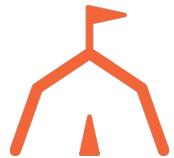


Heading Tags

- You should use a heading tag when you need a heading for a section.
- There are 6 heading tags, h1 - h6 in order of importance.

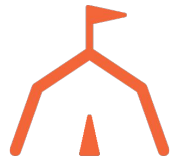
Heading Tags

```
<h1> Title </h1>
<h3> Subtitle </h3>
<h6> Description </h6>
```



Div Tag

A **div** is an empty container you can use the lay out your webpage.

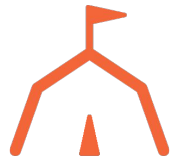


Div Tag

```
<div>
    <h1> Here's the title of this area! </h1>
    <p> Here's where I put all the content! </p>
</div>
```

Span Tag

A **span** tag is similar to a div tag but it can be used by default inline with other elements.

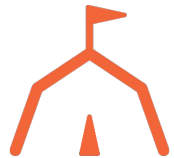


Span Tag

```
<p> Here is a paragraph tag with a nested span. The reason I’m using a span in here is so I can <span id="underline"> change the styling </span> without affecting the layout! </p>
```

A Href Tag

The **a href** tag is used to insert hyperlinks into the webpage.

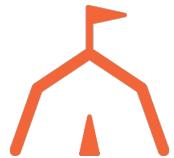


A Href Tag

```
<p> Click on the <a href="www.grandcircus.co">
link </a> for more information! </p>
```

Img Tag

The **img** tag is used to insert images into the webpage.

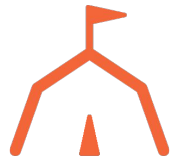


Img Tag

```
<img src="images/picture.jpg">
```

HTML Element

An HTML element is an individual component of HTML. It includes the HTML tags as well as the content.



HTML Element

Some HTML elements do not have closing tags, they are called *empty* elements.

Examples of empty elements are <img> and <input>.

HTML Element

```
<p> Hello! How are you doing today? </p>
```

Display Types

It’s important to note what display types HTML elements are by default.

Default refers to how the browser initially renders it without CSS being used to change it.

Display Types

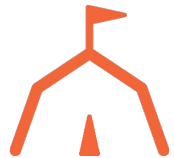
Block: Always starts on a new line and takes up the full width of the container.

Inline: Only takes up the width of the content, which cannot be adjusted.

Inline-block: Takes up the width of the content, but can also be given width and height properties.

Examples of Default Block Elements

```
<div></div>
<p></p>
<li></li>
<ul></ul>
<h1></h1>
<form></form>
```



Examples of Default Inline Elements

```
<span></span>
<input>
<a href></a>
```

HTML Attribute

1. Provides additional information about html elements.
2. Attributes include **class**, **style**, **language**, **id**, and others, including **href** and **src** which we've already seen.
3. Attributes are placed inside the opening tag.

HTML Attribute

```
<div attribute="value"> Content </div>
```

Why use HTML attributes?

They can help distinguish elements from each other. For example, if you have two paragraphs elements and you want to underline one of them, you'd need a way to target that specific element.

How do I do that?

Using attributes like class or id!

Class vs. Id

Id is an attribute used to select one specific element.

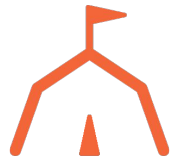
Class is an attribute used to select elements that have the same class value.

Id Attribute

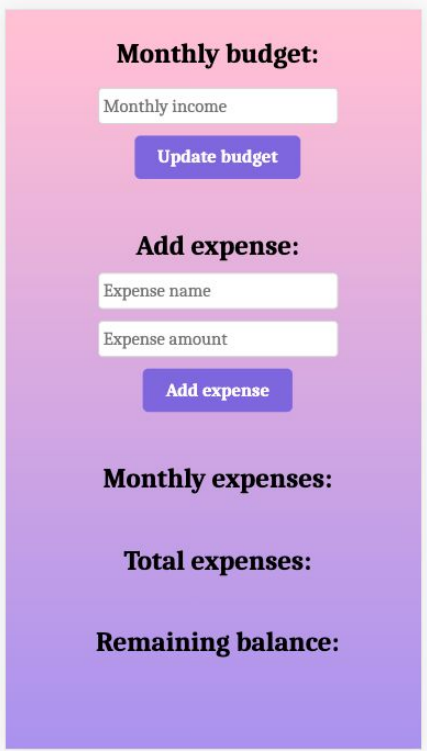
```
<div id="underline"> I want this underlined! </div>
<div> I don’t want this underlined! </div>
```

Class Attribute

```
<div class="underline"> I want this underlined! </div>
<div class="underline"> I want this underlined too! </div>
```



Exercise: Budget App Project



Budget App Project: HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>My Budget App</title>
</head>
  <body>

  </body>
</html>
```

```
<body>
  <main>
    <div class="main" id="monthly_budget_section"> </div>
    <div class="main" id="add_expense_section"> </div>
    <div class="main" id="expense_list_section"> </div>
    <div class="main" id="expense_total_section"> </div>
    <div class="main" id="remaining_balance_section"> </div>
  </main>
</body>
```

```
<div class="main" id="monthly_budget_section">
  <h3>Monthly budget:</h3>
  <p id="monthly_budget"></p>
  <form>
    <input type="number" id="income_input" placeholder="Monthly
    income" />
    <button id="update_income">Update budget</button>
  </form>
</div>
```

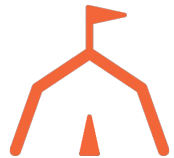
```
<div class="main" id="add_expense_section">
  <h3>Add expense:</h3>
  <form>
    <input type="text" id="name_input" placeholder="Expense name" />
    <input type="number" id="amount_input" placeholder="Expense
    amount" />
    <button id="add_expense">Add expense</button>
  </form>
</div>
```

```
<div class="main" id="expense_list_section">
  <h3>Monthly expenses:</h3>
  <div id="expense_list"></div>
</div>
```

```
<div class="main" id="expense_total_section">
  <h3>Total expenses:</h3>
  <p id="total_expenses"></p>
</div>
```

```
<div class="main" id="remaining_balance_section">
  <h3>Remaining balance:</h3>
  <p id="remaining_balance"></p>
</div>
```

CSS



What is CSS?

CSS is used to create the look and feel of a webpage. This is where you determine the colors, layout and feel of the display of the page.

You can also set styles for how you want the page to look on different devices in the CSS.

Steps to using CSS

1. Create a CSS file. This is a file within the project that ends with **.css**. Typically you'll see it named as **styles.css**.
2. Link your CSS to the HTML file.

Linking the CSS to an HTML file

```
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
</body>
```

Important Terms

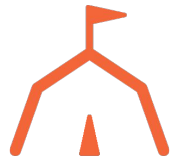
Rules: Consists of both the selector and the declaration.

Selector: The HTML element you want to style. It can be a class, id, or tag.

Declaration: Contains the specific property and value to apply to the selector.

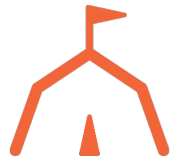
Property: The type of style you want to apply to the selector.

Value: The specific number or keyword given to the property.



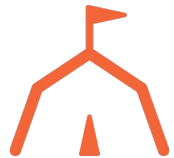
CSS Rule for Element Selector

```
h1 {
    background-color: teal;
    color: orange;
}
```



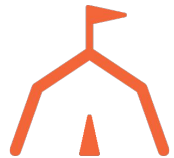
CSS Rule for Id Selector

```
#first_heading {
    font-size: 20px;
    text-align: center;
}
```



CSS Rule for Class Selector

```
.title {
    font-family: Arial;
}
```

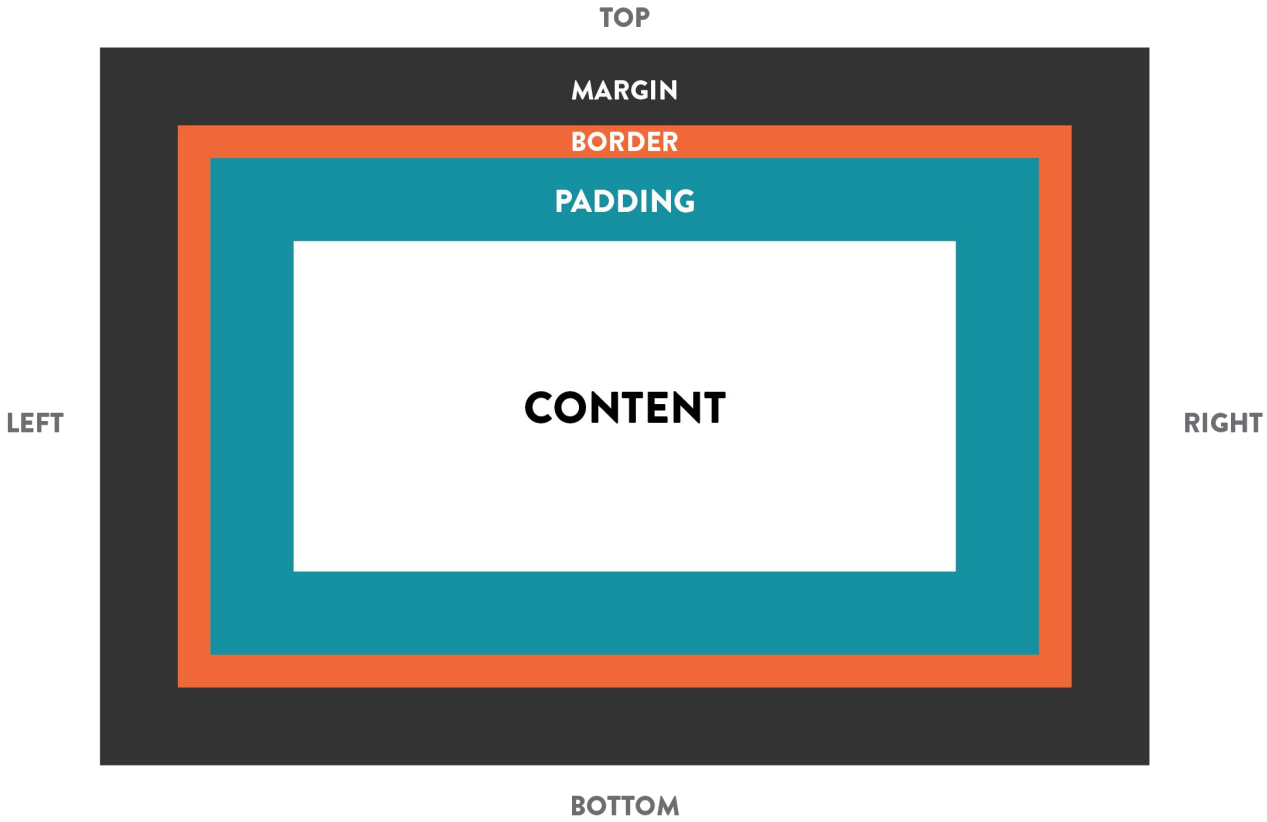


What is the Box Model?

Each HTML element is a rectangular box with a specific width and height.

The width and height are determined by the padding, margin and border of the element.

Box Model



Box Model Terms

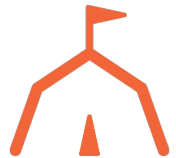
Content: Text or images represent the content of the element.

Padding: The space between the border and the content. Padding is transparent.

Border: Wraps around the padding and the content. Can be styled.

Margin: Outer layer, controls the spacing between this element and others. The margin is also transparent.

Flexbox



What is Flexbox?

- Flexbox is a set of CSS properties and values used to lay out, align and distribute space among items in a container.
- It is best used on small scale layouts, not full page layouts.

Why is it important?

It allows you to create layouts that are not restricted by direction (like inline elements are always horizontal and block elements are always vertically stacked) and easily change the order of elements while only using CSS.

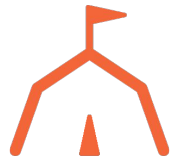
Nesting in HTML

Before we get into how to use Flexbox in CSS, we need to go over some HTML structure.

Nesting refers to putting HTML elements *within* other elements.

The parent element

The parent element is the container which other elements are nested in.

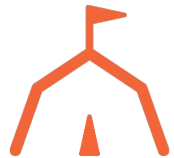


Parent element example

```
<div class="parent">
    <p>The div is the parent since it’s the
     container that wraps around me!</p>
</div>
```

The child element

The child element is the next direct element within a container.



Child element example

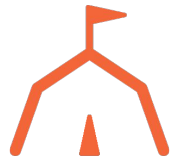
```
<div>
    <p class="child">I’m the child element
     since I’m the direct descendant of the
     div container.</p>
</div>
```

Another display type

- Just like the block, inline and inline-block display types, there is one specifically used for Flexbox.
- In order to access Flexbox properties, you need to use the flex display type.

Another display type

Unlike the other display types, the flex display type needs to be put on the *parent* element.



HTML structure

```
<div class="flex-container">
    <p>First paragraph</p>
    <p>Second paragraph</p>
    <p>Third paragraph</p>
</div>
```

CSS structure

```
.flex-container {
    display: flex;
}
```

Flex-direction

Flex-direction is a property used on the parent element in order to know whether to lay out the child elements horizontally or vertically.

Flex-direction values

Row: Default value, lays out elements horizontally.

Column: Stacks elements vertically.

Row-reverse: Reverses the order of how the elements are written in HTML horizontally.

Column-reverse: Reverses the order of how the elements are written in HTML vertically.

HTML structure

```
<div class="flex-container">
    <p>First paragraph</p>
    <p>Second paragraph</p>
    <p>Third paragraph</p>
</div>
```

CSS structure

```
.flex-container {
    display: flex;
    flex-direction: column-reverse;
}
```

Justify-content

The justify-content property is also used on the parent element to adjust the alignment of the flex items horizontally.

Justify-content values

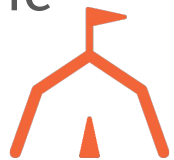
Flex-start: Items moved to the left of the container

Flex-end: Items moved to the right of the container

Center: Items are horizontally centered

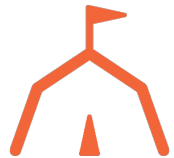
Space-between: Even spacing between items with the left and right most element at the edge of the container.

Space-around: Even spacing around the items, including on the left and rightmost elements.



HTML structure

```
<div class="flex-container">
    <p>First paragraph</p>
    <p>Second paragraph</p>
    <p>Third paragraph</p>
</div>
```



CSS structure

```
.flex-container {
    display: flex;
    justify-content: space-between;
}
```

Align-items

The align-items property is also used on the parent element to adjust the alignment of the flex items vertically within the container.

Align-items values

Flex-start: Items moved to the top of the container

Flex-end: Items moved to the bottom of the container

Center: Items are vertically centered

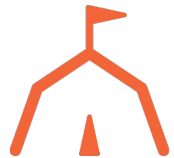
Stretch: Items' heights fill the vertical space of the container.

HTML structure

```
<div class="flex-container">
    <p>First paragraph</p>
    <p>Second paragraph</p>
    <p>Third paragraph</p>
</div>
```

CSS structure

```
.flex-container {
    display: flex;
    align-items: center;
    justify-content: flex-end;
}
```



Let’s try it

How would you write the HTML and CSS to do the below?

1. Create a *parent* div container
2. With 4 *child* span items
3. Displayed in a column
4. That’s vertically centered

Follow these steps

1. Go to CodePen.io and create a new “pen”
2. Write the HTML. Look up the tags for divs and spans.
3. Write a CSS rule for the span elements: give them a border, background color, margin and padding so you can visualize the box model.
4. Write a CSS rule for the div element.
5. Add the Flexbox properties and values necessary to complete the desired outcome.

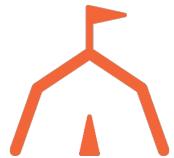
HTML structure

```
<div>
  <span>Item 1</span>
  <span>Item 2</span>
  <span>Item 3</span>
  <span>Item 4</span>
</div>
```

CSS structure

```
div {
  display: flex;
  flex-direction: column;
  align-items: center;
}
```

Budget App Project: CSS



```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  initial-scale=1.0">

  <link href="https://fonts.googleapis.com/css?family=Caladea:400,
  700&display=swap” rel="stylesheet" />
  <link rel="stylesheet" href="styles.css" />

  <title>My Budget App</title>
</head>
```

```css
body {
 font-family: "Caladea";
 background: linear-gradient(#fbc7d4, #9796f0);
 background-attachment: fixed;
}
```

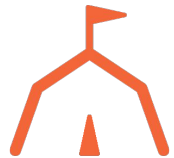
```css
main {
 width: 60%;
 margin: auto;
 text-align: center;
}

.main {
 margin-bottom: 6vh;
}

h3 {
 font-size: 1.5em;
 margin-bottom: 0px;
}
```

```
input {
 display: block;
 width: 200px;
 height: 30px;
 border-radius: 4px;
 border: 1px solid lightgrey;
 margin: 10px auto;
 padding: 0 4px;
 font-size: 1em;
 font-family: "Caladea";
}
```

```css
button {
 padding: 10px 20px;
 border-radius: 5px;
 border: none;
 background-color: #7574db;
 color: white;
 font-size: 1em;
 font-family: "Caladea";
}
```

```
button:hover {
 color: #7574db;
 background-color: white;
}

.red {
 color: red;
}

.green {
 color: green;
}
```

JavaScript

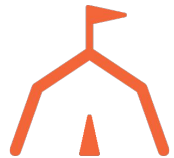


Steps to using JavaScript

1. Create a JavaScript file. This is a file within the project that ends with **.js**. Typically you'll see it named as **script.js**.
2. Link your JS in the HTML file.

Linking the JS to an HTML file

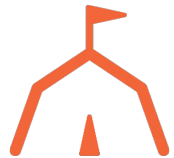
```
<html>
<head>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <script src="script.js"></script>
</body>
</html>
```



Variables

Variables are containers that store data values.

Data values are things like numbers and strings.



How to create variables

1. Use the keyword **let**.
2. Name your variable. It should be as clear to read as possible, using `camelCase`.
3. Assign your variable with an **=**.
4. Write the data value you want to store.
5. End with a semicolon.

Camelcase

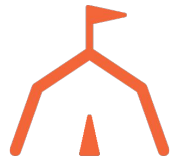
Camelcase is a way to format your variable names so they are the most readable without breaking them with spaces.

The first letter of every word should be capitalized except the first word.

Example: totalOrder vs totalorder

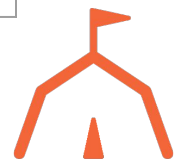
Variables

```
let name = "Name";

let total = 100;

let codingBootcamp = "Grand Circus";

let amountOfPeopleAttending = 20;
```



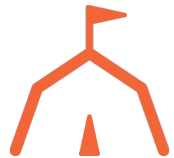
Data Types

string	A sequence of text
number	A number
boolean	A true/false value
array	A structure that allows you to store multiple values in one single reference.
object	Technically, everything in JavaScript is an object, and can be stored in a variable. Objects have properties and values.



Data Types Examples

```
let string = "hello!";
let number = 30;
let boolean = true;
let array = ["item one", "item two", "item three"];
let object = {
  propertyOne: "value one",
  propertyTwo: "value two"
};
```



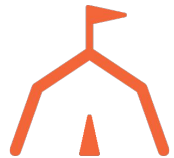
Other Important Keywords

return: determines the value of the function

console.log(): allows you to display JavaScript values in the console window

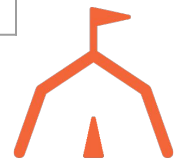
console.log()

```
console.log(2 + 2);
// will print 4 in the console window

console.log("Hello!")
// will print "Hello!" in the console window

let bootcamp = "Front-End Development";
console.log(bootcamp);
// will print "Front-End Development" in the console window
```



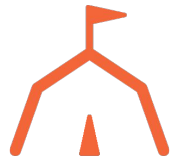
JavaScript Arithmetic Operators

+	addition
-	subtraction
*	multiplication
/	division
%	modulus (remainder)
++	increment
--	decrement



JavaScript Arithmetic Operators

```
let addition = 1 + 2; // 3

let subtraction = 10 - 8; // 2

let multiplication = 4 * 11; // 44

let division = 100 / 5; // 20

let modulus = 4 % 2; // 0
```



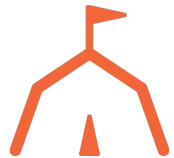
Conditionals

Conditionals control the behavior of whether or not a specific piece of code should execute or not.

They use **logical or comparison operators** to determine if something is true or not.

Conditionals

```
if (condition is true) {
    // execute this piece of code
}
```

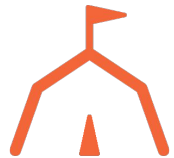


JavaScript Comparison Operators

==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

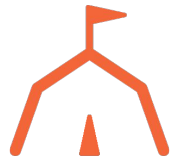
JavaScript Logical Operators

&&	and
||	or
!	not



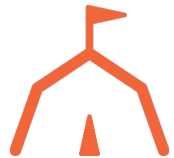
Conditionals

```
let name = "Grand Circus";
if (name === "Grand Circus") {
    return true;
} else {
    return false;
}
```



Conditionals

```
let temperature = 70;
if (temperature > 60) {
    console.log("You don't need a jacket!");
} else {
    console.log("Wear a jacket!");
}
```



Loops

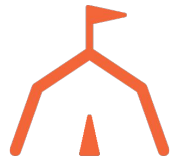
Loops offer a quick and easy way to do something repeatedly.

There are many different kinds of loops, but they all essentially do the same thing.

For Loops

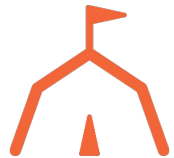
We’re going to take a look at the common “for” loop.

A for loop repeats until a specified condition evaluates to false.



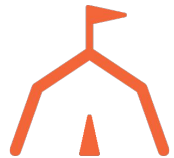
For Loops

```
for (initialExpression; condition;
incrementExpression) {
  // code block
}
```



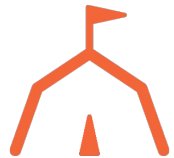
For Loops

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
// 0
// 1
// 2
// 3
// 4
```



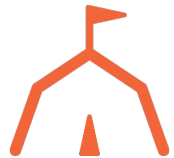
Functions

Functions are used to easily package code and execute it multiple times without having to write the same code over and over.



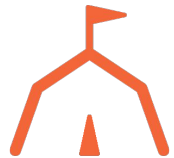
Functions

You can use **parameters** and **arguments** to pass through data to a function.



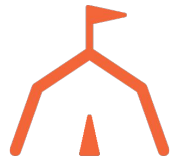
Functions

```
function addition (x, y){
    return x + y; // Will return 10
}

addition(6, 4)

// Parameters are "x and y" inside the parentheses of the function definition
// Arguments are the values "6 and 4" passed through the function when
triggered
```



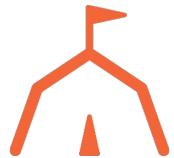
Functions

You can also use functions to make HTML interactive.



Functions

```
// JavaScript Code
function submitAnswer (){
    console.log("Your answer was submitted!");
}

<!-- HTML Code -->
<button onclick="submitAnswer()" id="submission-button">Done</button>
```



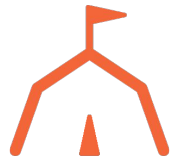
Functions

Let’s say in the previous example we want it to return a specific statement.

This is where we would use parameters and arguments.

Functions

```
// JavaScript Code
function submitAnswer(answer){
    console.log(answer); // Will print out "Done!" when the button is clicked
}

<!-- HTML Code -->
<button onclick="submitAnswer('Done!')" id="submission-button">Done</button>
```



Targeting Specific Elements with JavaScript

There are other ways to run more interactivity with HTML elements using JavaScript.

1. document.querySelector()*
2. document.getElementById()*

*The document object represents the HTML document of the browser window

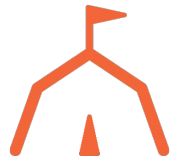
document.querySelector()

document.querySelector() allows you to target the first element in the document that contains the specified selector.

You can use it for both a class or id using the proper symbol.

document.querySelector()

```
<!-- HTML Code -->
<p class="paragraph">This is a paragraph in HTML</p>

// JavaScript Code
var paragraph = document.querySelector(".paragraph");
```

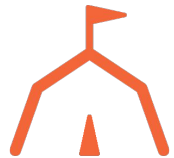


document.getElementById()

document.getElementById() allows you to target the element in the document that contains the specified id.

document.getElementById

```
<!-- HTML Code -->
<p id="paragraph">This is a paragraph in HTML</p>

// JavaScript Code
var paragraph = document.getElementById("paragraph");
```



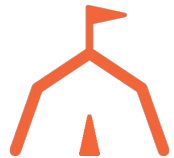
Events

Events are what happens in JavaScript when the HTML document is interacted with.

For example, if an element is clicked or the browser window is resized.

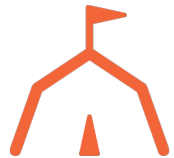
JavaScript Events

onclick()	Triggers when element is clicked
onsubmit()	Triggers when something is submitted
onmouseover()	Triggers when mouse goes over element
onresize()	Triggers when the element or window size is changed
onkeypress()	Triggers when a key on keyboard is pressed



Event Example

```
<!-- HTML Code -->
<button id="btn"> Click me! </button>

// JavaScript Code
let btn = document.getElementById('btn');
btn.onclick = function(){
  console.log("Hello!");
}
```



Budget App Project: JavaScript

```
<!DOCTYPE html>
<html lang="en">
...
  <body>

   <script src="script.js"></script>
  </body>
</html>
```

```
document
  .querySelector("#update_income")
  .addEventListener("click", updateBudget);

document
  .querySelector("#add_expense")
  .addEventListener("click", addExpense);
```

```
let monthlyBudget = document.querySelector("#monthly_budget");
let incomeInput = document.querySelector("#income_input");
let remainingBalance = document.querySelector("#remaining_balance");
let amountInput = document.querySelector("#amount_input");
let nameInput = document.querySelector("#name_input");
let expenseList = document.querySelector("#expense_list");
let totalExpenses = document.querySelector("#total_expenses");
```

```
let monthlyIncome = 0;
let expenses = [];
let expenseTotal = 0;
let balance = 0;
```

```
function updateBudget(event) {
 event.preventDefault();
 monthlyIncome = incomeInput.value;
 monthlyBudget.innerText = "$" + monthlyIncome;
 incomeInput.value = "";
 updateBalance();
}
```

```
function updateBalance() {
 balance = monthlyIncome - expenseTotal;
 remainingBalance.innerText = "$" + balance;
 if (balance < 0) {
   remainingBalance.classList.remove("green");
   remainingBalance.classList.add("red");
 } else {
   remainingBalance.classList.remove("red");
   remainingBalance.classList.add("green");
 }
}
```

```
function addExpense(event) {
 event.preventDefault();
 let expense = {
   expenseName: nameInput.value,
   expenseAmount: amountInput.value
 };
 let newExpense = document.createElement("p");
...
```

```
...
 newExpense.innerText = expense.expenseName + ": $" +
 expense.expenseAmount;
 expenseList.appendChild(newExpense);
 expenseAmount = parseInt(amountInput.value);
 expenses.push(expenseAmount);
 nameInput.value = "";
 amountInput.value = "";
 updateExpenseTotal();
}
```

```
function updateExpenseTotal() {
 expenseTotal = 0;
 for (let i = 0; i < expenses.length; i++) {
   expenseTotal += expenses[i];
 }
 totalExpenses.innerText = "$" + expenseTotal;
 updateBalance();
}
```

Budget App Project: Hosting on GitHub

GitHub

Want to host your project on GitHub pages?

First, get set up on GitHub:

1. Create a GitHub account
2. Create a New Repository
3. Add your files

GitHub Pages

Then, set up your project site:

1. Go to the Settings tab in your repository
2. Scroll down to the GitHub Pages section
3. Select master branch under Source
4. Visit *username*.github.io/*repository*

** If site does not load, add /index.html to end of URL*