INTRODUCCIÓN A LA PROGRAMACIÓN

Partes de una aplicación: Front-End vs Back-End vs Base de datos

Front-End (Interfaz de Usuario)

Front-end se refiere a la tecnología que permite mostrar información a un usuario. Esto incluye el diseño y las imágenes que ves, el diseño del texto y las páginas web, e incluso parte de la interactividad. Esto viene en HTML, CSS y JavaScript. Para el Pre-Bootcamp, te daremos una muestra de HTML y JavaScript.

Back-End (Soporte)

Back-end se refiere a la tecnología que mueve y administra los datos detrás de escena. Hay muchos lenguajes de programación que se pueden utilizar. Incluyen: Python, JavaScript, C #, Java, Ruby y PHP.

Base de datos

Como puede parecer, la base de datos es donde se almacenan todos los datos. ¿Te has preguntado dónde vive tu lista de Netflix? Está en una base de datos que Netflix posee y administra.

El ciclo de solicitud y respuesta

Conocer este ciclo te ayudará a contextualizar cómo se comunican el front-end, el back-end y las bases de datos. ¡Empieza a aprender más del fundador de Coding Dojo, Michael Choi, a continuación!

Como muchos saben, la web es una gran red de computadoras que interactúan y se comunican entre sí. Pero ¿cómo se comunican exactamente? ¡Qué gran pregunta! La forma más común y básica en que las computadoras se comunican entre sí es a través del ciclo de solicitud y respuesta. Generalmente, hay dos tipos de computadoras involucradas en este proceso: el cliente y el servidor.

La solicitud

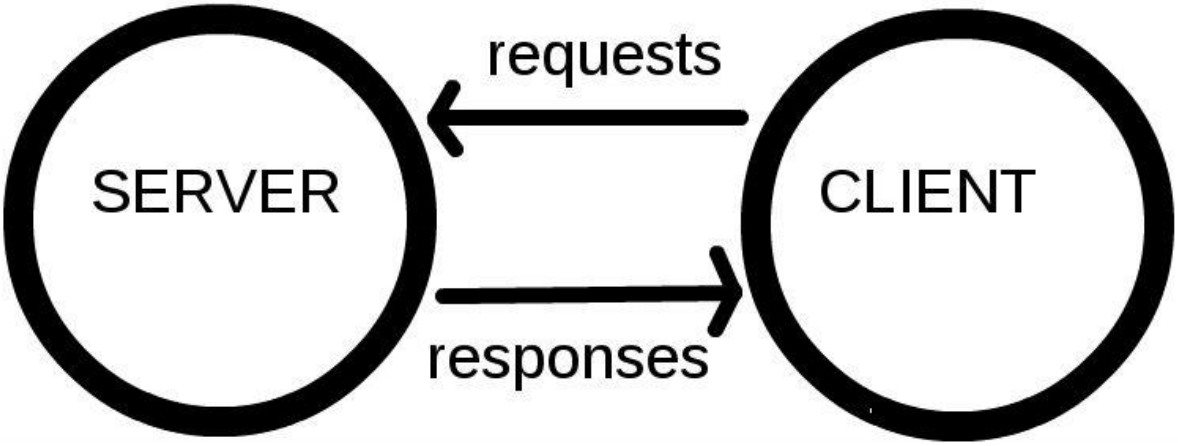
Como habrás adivinado, el ciclo de solicitud y respuesta comienza cuando la computadora cliente realiza una solicitud ingresando una dirección en un navegador web. Una vez que la solicitud está en camino, la primera parada que realiza es en el Sistema de nombres de dominio (DNS).

El DNS

La dirección real de una página web es una dirección IP que parece una serie de cuatro números separados por puntos. Digamos que soy dueño de una casa en Calle Principal 123 y elegí llamar mi casa Acres Ventosos. La dirección IP real de una página web es equivalente a mi dirección postal, y el nombre Acres Ventosos es equivalente al nombre de dominio que ingresaste en el navegador. Entonces, el DNS lee la URL y enruta tu solicitud a la dirección IP adecuada.

La respuesta

Una vez que el servidor recibe la solicitud del cliente, dado que el servidor aprueba la solicitud, se prepara una respuesta en el lado del servidor. Hay muchas formas diferentes en que el servidor puede preparar una respuesta para el cliente (consultar una base de datos, manejar la lógica, ejecutar algoritmos, etc.), pero lo cubriremos con más detalle una vez que comience el bootcamp. Una cosa importante para tener en cuenta aquí es que, aunque la respuesta se puede preparar de muchas formas diferentes, ¡siempre será una combinación de HTML, CSS y JavaScript!



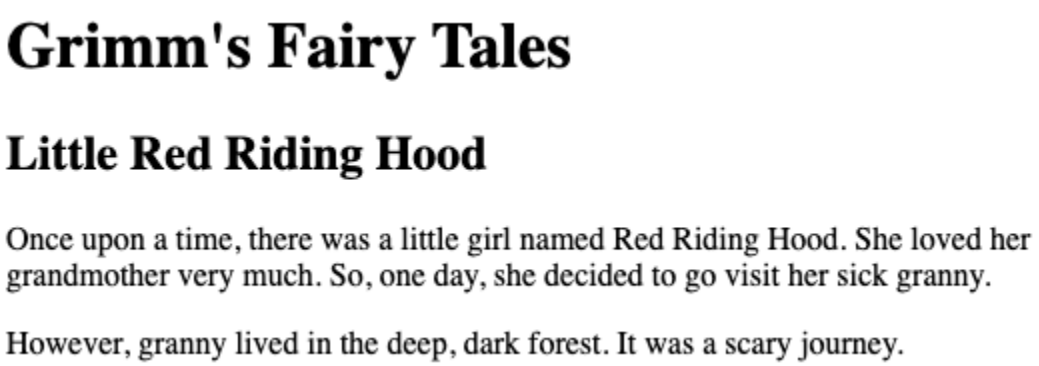
HTML

La programación es a menudo como la música: existen patrones y predictibilidad si sabes cómo escucharlos. Echa un vistazo a las dos imágenes siguientes. Uno de ellos es el código HTML real y el otro es lo que vería un usuario en base a ese código. Ve si puedes responder las preguntas a continuación.

Código HTML

```
1  <h1>Grimm's Fairy Tales</h1>
2  <h2>Little Red Riding Hood</h2>
3  <p>Once upon a time, there was a little girl named Red Riding Hood.
       She loved her grandmother very much. So, one day, she decided
       to go visit her sick granny.</p>
4  <p>However, granny lived in the deep, dark forest. It was a scary
       journey.</p>
```

Interfaz de usuario



Preguntas para reflexionar

¿Cuál es la diferencia entre h1 y h2 con respecto a cómo se muestra su texto?

Si h1 significa Título 1, ¿qué significa h2?

¿Qué crees que significa p?

¿Cuál es la relación entre <...> y </...>? ¿Cómo se relacionan entre sí?

¿Qué hace <...>?

¿Qué hace </...>?

Descripción general de HTML

Ahora que hemos captado tu atención con respecto a HTML, aprendamos más al respecto. El Lenguaje de Marcado de Hipertexto (HTML) es el lenguaje de marcado principal para todas las páginas web. Los elementos HTML son los bloques de construcción básicos de Internet.

¿Qué es HTML?

HTML es el lenguaje de la web. Significa Lenguaje de marcado de hipertexto. Su finalidad es permitirnos comunicar con el navegador el significado del contenido que deseamos colocar en una página web. Para lograr esto, HTML define una serie de etiquetas con las que podemos envolver el contenido. Este es un ejemplo de una etiqueta:

<h1>Hola, mundo</h1>

Las etiquetas HTML describen el contenido que contienen. Para este ejemplo, hay una etiqueta h1 que representa el título 1. ¡Hay muchas más de las que aprenderemos!

¿Quieres ver cómo se ve tras bambalinas?

Abre tu sitio web favorito. Haz clic derecho en tu navegador y haz clic en Inspeccionar elemento. Debería abrirse un nuevo módulo en la parte inferior de tu navegador con varias pestañas. Actualmente estamos interesados en la pestaña Elementos. Al hacer clic en él, deberías ver el código HTML

Puntos Clave

Etiquetas de apertura y cierre

Dado que se supone que las etiquetas envuelven las cosas, la mayoría de las etiquetas vienen en pares: una de apertura y una de cierre, que denotan el principio y el final del contenido. Las etiquetas sin una barra inclinada hacia adelante se denominan etiquetas de apertura, mientras que las etiquetas con barras inclinadas hacia adelante se denominan etiquetas de cierre.

```
1  <html>
2      <head>
3          <!-- we'll throw more code in here later! -->
4      </head>
5      <body>
6          <h1>Hello World Again!</h1>
7      </body>
8  </html>
```

En el ejemplo anterior, <h1> es una etiqueta de apertura; es equivalente a que le digas al navegador "Oye, voy a comenzar a incluir el contenido del encabezado 1 ahora". </h1> es una etiqueta de cierre que significa "OK, he terminado con el contenido del encabezado 1 ahora".

Anidamiento

Observa que, entre las etiquetas HTML de apertura y cierre en las líneas 1 y 8, también tenemos etiquetas de encabezado y cuerpo. Las etiquetas pueden encapsular otras etiquetas. A esto se le llama anidamiento. Los elementos anidados están indentados con un tab para facilitar la lectura del documento.

La jerarquía que surge del anidamiento se llama DOM: Document Object Model.

Comentar

El formato para comentar en un archivo HTML es el siguiente:

 Contenido de párrafo.

<!-- Este es un comentario -->

HTML: Hola, mundo

¡Juguemos con HTML!

Asignación: ve tu archivo HTML en un navegador.

Instrucciones

Abre un navegador, como Firefox o Chrome.

Busca tu archivo Apellido_Nombre_VS_Asignación_1.hmtl en tu computadora (¿Recuerdas dónde guardaste la asignación de Curso de introducción/Configuración de la computadora/Primera asignación de VS Code?)

Haz clic y arrastra el archivo hacia tu navegador o haz doble clic en el archivo. ¡Ta-Rá! Deberías ver el texto "¡Hola, mundo!” en el navegador.

Recordatorio: esta es una oportunidad para que desarrolles el hábito de valerte por ti mismo. Aunque no revisaremos tu trabajo, tu cerebro necesita este ejercicio. Recuerda, practica una habilidad sin importar si un entrenador, un maestro, un jefe o un compañero la observarán. Esta no es la universidad. Este es el Dojo.

HTML: Encabezado

La etiqueta de encabezado es una de las tres etiquetas principales necesarias en un documento HTML. Contiene todos los metadatos sobre tu sitio y no se muestra en nuestra página. Aquí hay una sección de encabezado de ejemplo:

```
<head>
<meta charset="utf-8">
<title>Mi increíble página web</title>
<meta name="description" content="El texto aquí describe de qué se trata la página web. Es lo que aparecerá en los resultados de búsqueda en motores de búsqueda como Google, bajo el título de la página web. ¡Es importante que esto sea relevante para tu página y esté bien escrito!" >
<link rel="stylesheet" href="my_css_file.css">
<script src="my_javascript_library.js"></script>
<script src="another_javascript_file.js"></script>
</head>
```

Repasemos esto línea por línea de nuevo:

- <head>

Esta es la etiqueta de encabezado de apertura que indica que estamos a punto de comenzar a hablar sobre las propiedades del documento. Es inteligente que la etiqueta <head> se aferre a las propiedades y al cerebro de la página.

- <meta charset="utf-8">

Las páginas web correctamente codificadas declaran la codificación a un navegador a través de una etiqueta meta en el encabezado. Sin esta etiqueta, es posible que un navegador no sepa cambiar a la codificación adecuada y los caracteres puedan mostrarse incorrectamente.

- <title>MI increíble página web</title>

Este es el título de tu página web, lo que significa que cuando abres esta página en el navegador, la pestaña que se abra mostrará "Mi increíble página web". Este será el nombre con el que se marcará como favorito, así como el nombre que se utilizará cuando se muestre como resultado de búsqueda en un motor de búsqueda.

- <meta name="description" content="description content">

Los motores de búsqueda utilizan la etiqueta meta de descripción al mostrar resultados.

- <link rel="stylesheet" href="my_css_file.css">

Esta línea vincula una hoja de estilo a nuestra página, que determinará cómo se muestran visualmente nuestros elementos HTML en la página. ¡Aprenderemos más sobre lo que se incluye en my_css_file.css en la sección CSS de Fundamentos de la web!

- <script src="my_javascript_library.js"></script>

Esta línea vincula un archivo JavaScript o jQuery a nuestro documento. JavaScript hace que nuestras páginas sean interactivas. Aprenderemos más sobre estos archivos en la sección jQuery.

NOTA: Puedes vincular tantas hojas de estilo o archivos JavaScript como quieras dentro de las etiquetas de encabezado.

- </head>

Esta es la etiqueta de cierre de encabezado. ¡Indica que hemos terminado de hablar sobre las propiedades de nuestra página y podemos pasar al cuerpo!

En caso de que tengas curiosidad: más sobre etiqueta metas con un ejemplo…

Como mencionamos, las etiquetas meta hacen que tu página web sea más significativa para los motores de búsqueda como Google.

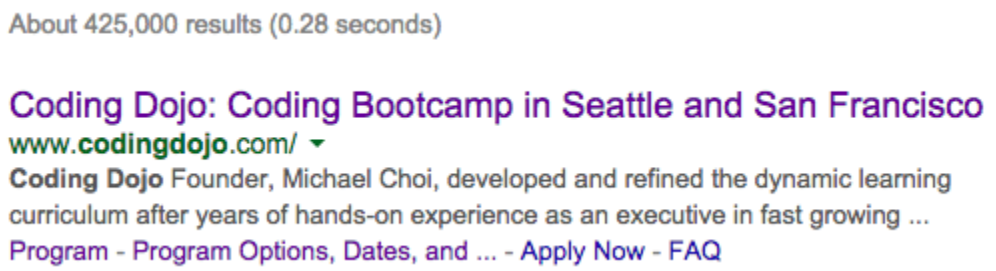
El atributo de contenido de la etiqueta meta de descripción describe el propósito básico de tu página web (un resumen de lo que contiene la página). Para cada página web, debes agregar un resumen conciso y relevante en esta sección.

Por ejemplo, esta descripción:

<title>Coding Dojo: Coding Bootcamp in Seattle and San Francisco</title>

<meta name="description" content="Coding Dojo Founder, Michael Choi, developed and refined the dynamic learning curriculum after years of hands-on experience as an executive in fast growing...">

Esto es lo que aparece en la página de resultados del motor de búsqueda de Google:



El "Coding Dojo: Bootcamp de codificación en Seattle y San Francisco" proviene de la etiqueta <title>.

HTML: Etiquetas

Es hora de aprender sobre las etiquetas más comunes que usarás dentro de la etiqueta <body>. Vamos a entrar en los elementos comunes que probablemente quieras agregar a una página web, que son:

- Encabezados y párrafos de texto
- Imágenes
- Vínculos
- Listas
- Tablas
- Formularios

En esta sección, cubriremos todo excepto formularios.

Headings

Un encabezado es un título de sección, lo que significa que a menudo (pero no siempre) cada sección de tu página tendrá un encabezado. Hay 6 niveles de encabezados que puedes usar, llamados <h1> a <h6>, cada uno indica la importancia de tu sección.

Veamos nuestra página actual para ver ejemplos. En la parte superior de esta página, verás Opcional: Etiquetas en letras grandes y en negrita. Esta es una etiqueta <h1>, escrita como…

<h1>Etiquetas de cuerpo comunes</h1>

…y es el encabezado principal de toda la página.

Párrafos

Cualquier fragmento de texto es un párrafo y, por lo tanto, debe encapsularse en etiquetas de párrafo :

* Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse cursus*

velit lectus, odales lorem id orci blandit, ac tincidunt lorem po

*rta. Sed euismod a arcu sed mollis. *

* Maecenas imperdiet risus at nisl aliquet, eu ullamcorper enim imperdiet. Sed*

id metus consectetur, sollicitudin eros at, dapibus ipsum. Morbi cursus nibh

sit amet port, pretium sagittis mi. Fusce rhoncus i

mperdiet eros, ac porta ligula ullamcorper in. Suspendisse nulla urna, facil

*isis non nunc ut, faucibus condimentum leo. *

Imágenes

Hay dos formas en que usamos las imágenes en una página web: como elementos de la página (como la carátula del álbum en Spotify o las fotos en tu feed de Facebook) o como imágenes de fondo (esto se trata en la sección CSS).

Las imágenes tienen dos atributos obligatorios: src y alt. El atributo src representa la fuente. Este es el enlace al lugar donde reside la imagen. El atributo alt significa alternativo, que son unas pocas palabras de texto para describir la imagen, en caso de que no se cargue. También lo utilizan los lectores de pantalla para personas con problemas de visión.

Enlaces

Los enlaces son cosas en las que hacemos clic y que nos redirigen a otra página. Por lo general, los enlaces están en formato de texto, pero también puedes utilizar una imagen como enlace.

La etiqueta que se usa para los enlaces es la etiqueta <a>, que significa etiqueta de anclaje. Al igual que las imágenes, los enlaces también deben tener un atributo que le indique al navegador a dónde llevará el enlace. Para los enlaces, esto se denomina atributo href, abreviatura de referencia de hipertexto.

Los valores posibles para el atributo href son:

Una URL absoluta: apunta a otro sitio web (como href=http://www.example.com/default.html)

Una URL relativa: apunta a otro archivo dentro de un sitio web (como href="default.html”)

Una URL de anclaje: apunta a un ancla dentro de una página (como href="# top”)

Ejemplo:

Click here to go to Google

 <!-- Funcionará este enlace? -->

<img src="">

Listas

La forma en que pensamos de las listas en HTML es un poco diferente de cómo las pensamos en nuestro día a día. Es cualquier colección de elementos del mismo tipo. El uso más común de las listas en HTML son los enlaces de navegación.

- Inicio
- Sobre Nosotros
- Contáctanos

Hay dos tipos de listas HTML: listas ordenadas (listas que están numeradas) y listas desordenadas. Las listas ordenadas usan la etiqueta <ol> y las listas no ordenadas usan la etiqueta <ul>. Ambas listas usan la etiqueta <li> para describir cada elemento de la lista.

Ejemplo:

```
<ul>
  <li>
    <a href="home.html">Inicio</a>
  </li>
  <li>
    <a href="about.html">Sobre Nosotros</a>
  </li>
  <li>
    <a href="contact_us.html">Contáctanos</a>
  </li>
</ul>
```

Tablas

A menudo nos encontraremos usando tablas para mostrar información. Las tablas tienen muchas etiquetas asociadas porque están formadas por muchas partes diferentes. Tienen:

Un encabezado de tabla <thead>, que contiene filas <tr> y nombres de columna <th>.

Un cuerpo de tabla <tbody>, que contiene filas <tr> llenas de datos de tabla <td>.

Las etiquetas que necesitamos son: <table>, <thead>, <th>, <tbody>, <tr> y <td>.

Ejemplo:

```
<table>
  <thead>
    <tr>
      <th>Nombre</th>
      <th>Email</th>
      <th>Número de télefono</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Nombre de ejemplo</td>
      <td>an_email@gmail.com</td>
      <td>555-5555</td>
    </tr>
    <tr>
      <td>Otro nombre</td>
      <td>another_email@gmail.com</td>
      <td>444-4444</td>
    </tr>
  </tbody>
</table>
```

Bonus: copia y pega el fragmento HTML anterior en uno de tus archivos .html y cárgalo en tu navegador. ¡Sé siempre curioso!

Funciones

Echa un vistazo al fragmento de código a continuación. Si construyéramos un programa de computadora para que sea una linterna, aquí está el fragmento que reconocería si se presionó el botón de la linterna para encender la luz.

```
function onButtonTapTurnFlashlightOn() {
   lightBulbOn = true;
}
```

Funciones

Las funciones son fragmentos de código (también conocidos como bloques de código), que cuando se activan, son un conjunto de instrucciones sobre lo que debe hacer el programa de computadora. Piensa en nuestro ejemplo de linterna: la función que viste encendía o apagaba la luz.

Veamos más a fondo la sintaxis de una función:

```
function nameOfFunction() {
   // código a ejecutar
}
```

Una función es un conjunto de instrucciones que esperan ser activadas. Sin embargo, las funciones son como actores tras bambalinas en una obra: no siempre están en el escenario repitiendo sus líneas; tienen que esperar a que los llamen. Otra metáfora es que las funciones son como una receta. Una vez que tenemos la receta de las galletas con chispas de chocolate, ahora sabemos cómo hacer las galletas, pero eso no significa que las tengamos inmediatamente una vez que obtenemos la receta. Tenemos que llamar al actor al escenario o seguir los pasos y hacer las galletas.

Llamar o invocar una función

Entonces, el ejemplo anterior de función es la receta o las líneas del actor. Para que la función funcione, tenemos que llamarla, y lo hacemos por su nombre y agregando ()

```
nameOfFunction();
```

He aquí un ejemplo:

```
function counter() {
   for (var num = 0; num <= 5; num++) {
      console.log(num);
   }
}
counter();    // ejecuta la función
counter();    // ejecuta la función otra vez
```

Ahora que hemos escrito la función counter(), ¡podemos invocarla tantas veces como queramos!

Nota: Es posible que te preguntes qué hacen todas esas cosas entre { y }. Bueno, buenas noticias: puedes aprender eso completando las Lecciones 1 a 4 de la App de algoritmos.

Las funciones son poderosas y aumentan en complejidad. A lo largo del bootcamp, aprenderás mucho más sobre las funciones.

var y tipos de datos

Los programas pueden hacer mucho más que encender y apagar cosas, como nuestra linterna. Muchas veces, requieren que se transmita información. Por ejemplo, piensa en una calculadora que calcule cuál sería la propina en la cuenta de un restaurante. El programa necesitaría saber el monto total de la cuenta, conocer el porcentaje que quieres dar de propina y luego necesitaría usar esos dos números para calcular un monto. Y tendría que darte a ti, el usuario, un lugar para ingresar esa información y un lugar para mostrarte el cálculo. ¡Uf!

Pero, ¿cómo "toma" el programa el monto de la cuenta para calcularlo?

Cada vez que se ingresa o se usa un fragmento de información en un programa, la computadora necesita extraer algo de su memoria para retenerlo. Nosotros, como desarrolladores, podemos decirle a la computadora cuándo crear espacio para un dato, cómo llamar a ese espacio para ese dato y qué se almacena en ese espacio. A este proceso lo llamamos crear variables.

Echa un vistazo a las siguientes variables y ve si puedes encontrar algunos patrones para responder las preguntas que siguen.

```
var firstName = "Gareth";

var lastName = "O'Neil";

var age = 38;

var height = "5 feet 9 inches";

console.log(firstName); //Gareth
```

Var y tipos de datos

Var

Como viste en la práctica, las variables son cosas que contienen información. Es como una carpeta de archivos: le damos un nombre a la carpeta y podemos poner cosas dentro de ella. Una vez que ponemos cosas (datos) dentro de ella, podemos acceder a esos datos en cualquier momento llamando a esa variable.

Tipos de datos

Hay muchos tipos diferentes de datos en informática; los más comunes en JavaScript son: cadena, número, undefined, null y booleano. Analicemos cada uno de ellos.

Número

Los números son exactamente lo que crees que son: ¡números! En JavaScript, los números enteros y los números flotantes (decimales) se consideran "números". Puedes realizar operaciones matemáticas utilizando variables cuando se les asignan números.

```
var myNum = 5;

var myOtherNum = 10;

console.log(myNum + myOtherNum); // 15
```

Cadena

Las cadenas son simplemente caracteres agrupados y rodeados de comillas. Las cadenas pueden tener cualquier cantidad de caracteres dentro de ellas.

```
var myStringVariable = "¡Mi nombre es Jeff!";

console.log(myStringVariable); // ¡ Jeff!
```

UNA NOTA IMPORTANTE: Es posible que hayas notado que las cadenas tienen " " que las encierran. Todo lo que se agregue a esas comillas se trata como una cadena, lo que significa que, si agregas números dentro esas comillas, se tratan como una cadena. Las computadoras no pueden hacer aritmética con cadenas. He aquí un ejemplo:

```
var exampleOfStringWithNumberInside = "Nací en  1985";

var exampleOfNumber = 1985;
```

Si quisiéramos calcular la edad de esta persona, necesitaríamos la variable exampleOfNumber, no la cadena. Aprenderás formas de evitar esto en el bootcamp, pero por ahora: recuerda que la computadora trata las cadenas de manera diferente que a los números.

Undefined

Una variable a la que aún no se le ha asignado un valor toma el valor de undefined. Mira los siguientes fragmentos de código:

```
var myVar;

console.log(myVar); // undefined se imprimirá en tu consola!
```

Null

Null y undefined a veces se usan incorrectamente de manera intercambiable. ¡Son muy diferentes! Como leíste más arriba, undefined es el valor que se le da a una variable a la que aún no se le ha asignado un valor explícitamente. Null es un valor que se asigna de manera explícita y, a menudo, se usa cuando se quiere indicar explícitamente que no se retiene nada en esta variable.

Boolean

¡El booleano es una forma muy elegante de decir verdadero o falso!

Desafío de código: Predecir funciones

Asignación: practica lo que has aprendido para predecir lo siguiente.

Instrucciones

- Abre un archivo nuevo de VS Code.
- Guárdalo como tipo .js y asígnale el nombre APELLIDO_NOMBRE_JS
- Predicción
- Usando comentarios (usa dos barras diagonales -> //), responde las preguntas de predicción.

Predicción 1: ¿Qué indicará console.log cuando se llame a esta función?

```
function myBirthYearFunc(){
   console.log("Nací en " + 1980);
}
```

Predicción 2: si necesitáramos enviar un año de nacimiento hacia la función, le diríamos a la función "oye, te enviaremos una variable llamada EntradaAñoNacimiento". Lo hacemos agregando el nombre de la variable entre paréntesis. Mira a continuación un ejemplo.

Entonces, si la variable EntradaAñoNacimiento es 1980 (también conocida como var EntradaAñoNacimiento = 1980), y se llama a esta función, ¿qué indicará console.log?

```
function myBirthYearFunc(EntradaAñoNacimiento){
   console.log("Nací en " + EntradaAñoNacimiento);
}
```

Predicción 3: Intentémoslo otra vez. Si var num1 = 10 y var num2 = 20, ¿qué indicaría console.log?

```
function add(num1, num2){
   console.log("¡Sumando números!");
   console.log("num1 is: " + num1);
   console.log("num2 is: " + num2);
   var sum = num1 + num2;
   console.log(sum);
}
```

NOTA: No se trata de hacerlo bien, se trata de practicar. Nuevamente, esto es como entrenar para un maratón: a veces el kilómetro que corres el martes apesta, pero al menos corres.

Condiciones

En JavaScript Pattern 1, hablamos sobre la creación de un programa para encender una linterna. Pero, ¿y si la luz ya está encendida? En este momento, nuestro programa de computadora no tiene el cerebro para saber si la luz ya está encendida, y si lo está, qué hacer. Este es un buen recordatorio de que las computadoras solo harán lo que se les indique. Si no está codificado explícitamente, el programa no lo hará.

Echa un vistazo al fragmento de código a continuación. Aquí, agregamos algo llamado condicional: si la luz ya está encendida, apágala; o si no, enciende la luz.

```
function onButtonTapTurnFlashlightOn() {
   if (light.bulb == true) {
      light.bulb = false;
   }
   else {
      light.bulb = true;
   }
}
```

Condicionales

¿Qué pasa si tenemos algún código que solo queremos ejecutar bajo ciertas condiciones? Piensa en nuestra linterna de ejemplo: solo queremos encenderla si ya está apagada. En programación, esto se llama sentencia condicional. La sintaxis en JavaScript se ve así:

```
if (condition) {
   // qué hacer si la condición es verdadera
}
else if (2nd condition) { // puede haber desde 0 hasta muchas de estas sentencias
   // qué hacer si la segunda condición es verdadera
}
else {  // puede haber desde 0 o 1 de estas sentencias
   // qué hacer si ninguna de las condiciones anteriores se cumple
}
```

Ten en cuenta que en una cadena if/else if/else, cuando una de las condiciones se evalúa como verdadera, nuestro código no verificará ninguna de las otras condiciones hasta que vuelva a llegar a un "si".

Algunos ejemplos:

Sentencias If/Else

```
var x = 25;
if (x > 50) {
   console.log("mayor que 50");
}
else {
   console.log("menor que 50");
}
// porque x no es mayor a 50, la segunda sentencia imprimir, "menor que 50", es la única línea que se ejecutará
```

Sentencias If/Else If/Else

```
var x = 75;

if (x > 100) {

   console.log("mayor que 100");

}

else if (x > 50) {

   console.log(" 50");

}

else if(x > 25) {

   console.log("mayor que 25");

}

else {

   console.log("número pequeño");

}

// debido a que la primera declaración no es verdadera, pero la segunda declaración es verdadera, imprimirá "mayor que 50" en nuestra consola. Sin embargo, ni siquiera intentará verificar nuestro else if (x> 25) o las declaraciones else porque ya ha encontrado algo en la cadena que es verdadero.
```

Operadores lógicos y de comparación

Las sentencias condicionales evalúan valores booleanos (verdadero o falso). Aquí hay una tabla de las formas en que puedes comparar dos valores.

NOTA: Practicarás esto en la Aplicación de algoritmos

Operador	Descripción	Ejemplos
==	Igual	1 == 2 => false; 1 == 1 => true
!=	no es igual	1 != 2 => true; 1 != 1 => false
>	mayor que	1 > 2 => false; 2 > 1 => true
<	menor que	1 < 2 => true; 1 < 2 => false
>=	mayor o igual que	1 >= 2 => false; 2 >= 2 => true
<=	menor o igual que	1 <= 2 => true; 2 <= 2 => true

Arreglos

Piensa en todas las fotografías que tienes en tu teléfono o impresas. ¿Cómo sería tu vida si no pudieras organizarlas en álbumes? ¡Qué pesadilla!

Al igual que en la vida real, tenemos cosas digitales que queremos organizar para facilitar el acceso, la revisión y cosas por el estilo. Una forma de organizar (o estructurar) estos datos, es a través de algo llamado arreglo. Echa un vistazo a lo siguiente y ve qué patrones puedes ver en la forma en que usamos los arreglos. Luego responde las preguntas a continuación.

//Arreglo 1

var produceList = ["manzanas", "naranjas", "jalapeños"];

//Arreglo 2

var accountBalances = [5000, 10, 2500];

//Arreglo 3

var auntsContact = ["Jayne", "Smithe", "Calle Principal 123", "Springfield", "MO", 12345];

Arreglos

Hemos aprendido a capturar datos con variables y hemos estado practicando con cosas como cadenas y números. Eso es genial, ¡sabemos cómo capturar datos! ¿Pero qué pasa si necesitamos una colección de cadenas o números? Por ejemplo, si estamos administrando datos de oficina y queremos almacenar datos sobre un empleado en nuestra aplicación, el siguiente código está bien. Sin embargo, lo ideal sería almacenarlos todos juntos, ¡porque están relacionados!

```
var usersFirstName = "Dwight";
var usersLastName = "Schrute";
var usersEmail = "beetsbears@battlestar.com";
```

En su lugar, es posible que deseemos usar una estructura de datos llamada arreglo:

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];
```

Agregar: ¡Ahora solo tenemos una variable! Si queremos agregar información, podemos usar el método push (más sobre funciones y métodos más adelante) para agregar un valor al final de nuestro arreglo:

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];
user.push("jello");
console.log(user);    // ["Dwight", "Schrute", "beetsbears@battlestar.com", "jello"]
```

Eliminar:

Para eliminar un valor del final, usamos el método pop:

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];
user.pop();
console.log(user);    // ["Dwight", "Schrute"]
```

Acceder/actualizar:

Para acceder o actualizar valores en un arreglo, usamos los números de índice, con el primer valor comenzando en 0:

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];
console.log(user[0]);    // leyendo el valor -- OUTPUT: Dwight
user[1] = "Martin";    // actualizando el valor
console.log(user);    // ["Dwight", "Martin", "beetsbears@battlestar.com"]
```

Length (longitud):

Los arreglos también tienen una propiedad length, que nos dice cuántos valores están contenidos en él:

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];

console.log(user.length);   // 3

user.pop();

console.log(user.length);   // 2
```

Para practicar con arreglos, complete el nivel 2 y 3 de Algoritmos.

<u>Una herramienta: Diagramas T</u>

A medida que nuestro código crece y aumenta el número de variables, los diagramas en T nos ayudan a realizar un seguimiento del estado de todas nuestras variables. Son un activo invaluable cuando nuestro código no funciona como esperábamos porque nos ayuda a ralentizar nuestro cerebro y mirar el código exactamente como lo haría una computadora.

En una hoja de papel o una pizarra, tendremos 2 columnas. El lado izquierdo serán los nombres de las variables que tengamos. El lado derecho será el valor actual para cada una de esas variables. ¡Utiliza un diagrama T para cada ejercicio en adelante!

Ve si puedes predecir lo que está haciendo este aparentemente complicado bloque de código antes de ver el video:

```
var num1 = 20;
var num2 = 5;
if (num1 < num2) {
   num2 = num2 * num1;
} else {
   num1 = num1 / num2;
   if (num1 < num2){
      num1 = num1 * 2;
   } else if (num1 == num2){
      num2 = num1 * num2;
   } else {
      num2 = num2 * 2;
   }
}
console.log(num1);
console.log(num2);
```

Desafío de código: Predecir arreglos y condicionales

Asignación: crea diagramas T para predecir el resultado de los siguientes fragmentos de código.

Instrucciones:

Para cada fragmento de código, crea un diagrama T.

Crea un solo documento de texto o fotografía de tus diagramas T que contengan los 3 diagramas T para los fragmentos de código a continuación y envíalo.

Recordatorio: esta es una oportunidad para que desarrolles el hábito de valerte por ti mismo. No vamos a revisar tu trabajo, y tu cerebro necesita este ejercicio. Recuerda, practica una habilidad sin importar si un entrenador, un maestro, un jefe o un compañero la observarán. Este no es el colegio. Este es un Dojo.

Fragmento de código 1

```
function muestraInformaciónContacto() {
   var auntContactInfo = ["Paula", "Smith", "Calle Principal 1234", "St. Louis", "MO", 12345];
   console.log(auntContactInfo);
}
```

Fragmento de código 2

```
function muestraListaDeCompras() {
   var produce = ["manzanas", "naranjas"];
   var frozen = ["brócoli", "helado"];
   frozen.push("croqueta de papa");
   console.log(frozen);
}
```

Fragmento de código 3

```
var movieLibrary = ["Bambi", "E.T.", "Toy Story"];
movieLibrary.push("Zoro");
movieLibrary[1] = "Beetlejuice";
console.log(movieLibrary);
```

Return

Hasta ahora, hemos usado console.log() para poder visualizar cosas. Sin embargo, no es muy útil para nada más. Echa un vistazo a continuación y mira qué patrones puedes identificar que atañen a esta cosa llamada "return".

Fragmento de código 1

```
function obtenerNombreDesdeFormulario (){
   var firstName = "Juan";
   return firstName;
}
```

¡DESAFÍO! Fragmento de código 2

```
function calcularPropina (totalCuenta, porcentajePropina){
   var totalCuenta;
   var porcentajePropina;
   var propina = totalCuenta * porcentajePropina;
   return propina;
}
calcularPropina(140, .20)
```

Hemos estado usando mucho console.log porque nos permite, como desarrolladores, ver qué está pasando. Si bien es útil, además de imprimir algo en la pantalla, en realidad no hace nada.

Ahora que conocemos las funciones, es importante aprender sobre la declaración de retorno. La declaración return funciona de manera diferente a console.log() porque hace algo dentro de la función, mientras que console.log() solo muestra algo a los desarrolladores en la consola.

Supongamos que queremos tener una función que, cuando se le llame, creara un arreglo con números.

```
function createArray() {
   var newArray = [0,1,2,3,4,5];
}
// newArray solo existe dentro de la función. Acá afuera ya no existe.
```

Con este código, llamar a la función crea un arreglo [0,1,2,3,4,5], pero luego, cuando la función termina, este arreglo se pierde rápidamente porque no tenemos ninguna referencia hacia él. Incluso si lo imprimimos en la consola, no podríamos hacer nada con el arreglo más adelante. Aquí es donde los return son realmente importantes: lo que devolvemos es el resultado real de la función. Una sentencia return también sirve para finalizar la función inmediatamente.

Si nuestra función hiciera conos de helado, console.log sería el equivalente a que el heladero diga: "¡Tengo un helado!", donde una sentencia return es el heladero literalmente entregándote el cono.

Cuando se llama una función, el valor que se devuelve se envía al fragmento de código desde el que se llamó. Agreguemos una sentencia return y una variable a nuestro código de arriba:

```
function createArray() {
   var newArray = [0,1,2,3,4,5];
   return newArray;       // agregó la sentencia return
}
var needAnArray = createArray();       // ahora needAnArray es la variable que llama a createArray
```

¿Cuál crees que es el valor de needAnArray?

Si dijiste: el arreglo [0,1,2,3,4,5], ¡estás en lo correcto! Recuerda, el valor de una llamada de una función, es lo que sea que devuelva. En este caso, puesto que needAnArray llamó a createArray(), needAnArray ahora contiene el valor que devuelve createArray.

Desafío de código: Return

Asignación: crea diagramas T para predecir el resultado de los siguientes fragmentos de código.

Instrucciones:

- Para cada fragmento de código, crea un diagrama T.
- Crea un solo documento de texto o fotografía de tus diagramas T que contengan los 10 diagramas T para los fragmentos de código a continuación y envíalo.

Recordatorio: esta es una oportunidad para que desarrolles el hábito de valerte por ti mismo. No vamos a revisar tu trabajo, y tu cerebro necesita este ejercicio. Recuerda, practica una habilidad sin importar si un entrenador, un maestro, un jefe o un compañero la observarán. Este no es el colegio. Este es un Dojo.

Fragmento de código 1

```
function hello() {
   console.log('hello');
}
hello();
console.log('Dojo');
```

Fragmento de código 2

```
function hello() {
   console.log('hello');
   return 15;
}
var result = hello();
console.log('result is', result);
```

Fragmento de código 3

```
function numPlus(num) {
   console.log('num is', num);
   return num+15;
}
var result = numPlus(3);
console.log('result is', result);
```

Fragmento de código 4: hay un giro

```
var num = 15;
console.log(num);
function logAndReturn(num2){
  console.log(num2);
  return num2;
}
var result = logAndReturn(10);
console.log(result);
console.log(num);
```

Fragmento de código 5

```
var num = 15;
console.log(num);
function timesTwo(num2){
  console.log(num2);
  return num2*2;
}
var result = timesTwo(10);
console.log(result);
console.log(num);
```

Fragmento de código 6

```
function timesTwoAgain(num) {
   console.log('num is', num);
   var y = num*2;
   return y;
}
var result = timesTwoAgain(3) + timesTwoAgain(5);
console.log('result is', result);
```

Fragmento de código 7

```
function sumNums(num1, num2) {
  return num1+num2;
}
console.log(sumNums(2,3))
console.log(sumNums(3,5))
```

Fragmento de código 8

```
function printSumNums(num1, num2) {
  console.log(num1);
  return num1+num2;
}
console.log(printSumNums(2,3))
console.log(printSumNums(3,5))
```

Fragmento de código 9

```
function sumNums(num1, num2) {
   var sum = num1 + num2;
   console.log('sum is', sum);
   return sum;
}
var result = sumNums(2,3) + sumNums(3,5);
console.log('result is', result);
```

Fragmento de Código 10: un desafío arduo así que tómalo con calma

```
function sumNums(num1, num2) {
   var sum = num1 + num2;
   console.log('sum is', sum);
   return sum;
}
var result = sumNums(2,3) + sumNums(3,sumNums(2,1)) + sumNums(sumNums(2,1),sumNums(2,3));
console.log('result is', result);
```

Sugerencia: sigue el mismo orden de operaciones en matemáticas (evalúa primero los paréntesis más internos).

BUCLES

A veces, en código, queremos hacer una cosa varias veces. Comencemos con algo simple, como un contador. Podríamos escribir algo como esto:

```
console.log(0);

console.log(1);

console.log(2);

console.log(3);

console.log(4);

console.log(5);
```

Esto está bien, pero es bastante tedioso. ¿Y si quisiéramos contar hasta 10? Son otras 5 líneas de código que se ven casi iguales. Eso no es divertido. Mejor, ¡aprendamos sobre los bucles!

Bucles for

Los bucles son muy comunes y, como hacen los programadores, desarrollaron un bucle que abrevia muchos de los componentes de arriba en una línea. Un bucle for nos permite hacer la misma operación varias veces. El siguiente hará exactamente lo mismo que el código anterior:

```
for (var num = 0; num <= 5; num++) {
   console.log(num);
}
```

Este código también cuenta hasta 5, pero el código se ejecuta en este orden:

La primera parte, var num = 0, se ejecuta solo una vez

La segunda parte, num <= 5, se evalúa. Si es verdadera, ve al paso 3. Si es falsa, ve al paso 6.

Ejecuta lo que esté dentro del bloque de código del bucle for (en nuestro caso: console.log(num);

Ejecuta la tercera parte,num++

Regresa al paso 2

Continúa ejecutando el código que viene después del bucle for

Bucles + arreglos

Usando estas ideas de números de índice y longitud de arreglo, podemos usar un bucle for para iterar a través de un arreglo y mirar cada elemento. Podemos configurar nuestra variable de bucle para que comience en 0 (¡el primer índice de un arreglo!). Nos detenemos cuando alcanzamos la

longitud del arreglo. Cada vez que se ejecute el bucle, ahora podemos acceder a los elementos del arreglo por ese índice.

```
var arr = [2,4,6,8,10];

for (var i = 0; i < arr.length; i = i + 1) {
   console.log(i);          // imprime el índice
   console.log(arr[i]);     // imprime el valor del arreglo en ese índice
}
```

¡Intenta hacer un diagrama T del código anterior para ver cómo funciona!

Nota: Ten en cuenta que la condición final de nuestros bucles lleva < en lugar de <=. ¿Por qué es esto? Debido a que los arreglos comienzan en el índice 0, el último índice del arreglo en realidad será uno menos que la longitud del arreglo. Por lo tanto, queremos que nuestro bucle deje de incrementar un valor antes de llegar a la longitud del arreglo.

Para practicar con arreglos, completa el nivel 5 de la Aplicación de algoritmos.

Bucles While

Hay varias formas diferentes de escribir bucles. Ahora que hemos visto los bucles for, vamos a aprender sobre los bucles while, que lucen como lo siguiente:

```
while (condition) {
   // qué seguir haciendo mientras la condición sea verdadera
}
// llegamos a esta línea cuando la condición en el ciclo es falsa
```

En nuestro contador de ejemplo, queremos seguir imprimiendo en la consola mientras (while) el número es menor o igual a 5. Escribámoslo:

```
var num = 0;   // comienza una variable en 0
while (num <= 5) {
   console.log(num);   // imprime el valor actual de num
   num++;      // incrementa num en 1
}
```

Entonces, ¿por qué necesitamos bucles for y bucles while?

Es posible que te preguntes, si los bucles solo están diseñados para hacer algo de forma repetitiva, ¿por qué necesitamos dos tipos? Los bucles for y while son buenos para diferentes situaciones.

Por lo general, se usa un bucle for cuando quieres repetir un proceso un número conocido de veces.

Por otro lado, se usa un bucle while cuando se quiere repetir un proceso hasta que se cumpla alguna condición.

Para practicar con bucles while, completa el Nivel 9 de la Aplicación de algoritmos.

Desafío de código: JavaScript

El propósito de esta tarea es que te acostumbres a crear y manipular código JavaScript.

Asignación: Utilizando VS Code, crea un proyecto, comprímelo y envíalo.

Instrucciones:

- Abrir VS Code y crear un archivo nuevo.
- Guarda el archivo nuevo con el tipo de archivo, esta vez es .js, y la siguiente convención en cuanto a la nomenclatura: Apellido_Nombre_JS_Jugueteo
- Crea lo siguiente usando lo que aprendiste acerca de las funciones, los condicionales, variables y tipos de datos, y envía la asignación.

La historia: queremos una aplicación que una vez que mida la altura de un niño, pueda mostrar si el niño es lo suficientemente alto como para subirse a la montaña rusa. Nos encargaron la construcción de la función que mostrará el mensaje correcto al niño.

Paso 1: crea la siguiente variable

una variable llamada alturaNiño que es un número. Dale cualquier número entero positivo que quieras.

Paso 2: crea una función llamada muestraSiElNiñoPuedeSubirALaMontañaRusa.

Paso 3: dentro de la función, crea la siguiente condicional

Si el alturaNiño es mayor a 52, el console.log debe decir "¡Súbete, chico!". Si no, console.log debe mostrar "Lo siento, chico. Tal vez el próximo año".

SUGERENCIA: Si te atrasas, ¡está bien! Sigue jugando con eso. Utiliza las páginas de este curso para ayudarte.

Desafío de código: Depuración

Aún no has aprendido sobre el término "return", pero básicamente devuelve lo que está a su derecha.

Usando lo que sabes sobre las funciones, variables, console.log y ahora return, arregla el siguiente código.

```
function greeting(){
   return "Hola, mundo";
}
var word = greeting();
console.log();
```

Operadores

Si bien no profundizamos mucho en las matemáticas al programar, nosotros sí utilizamos algunos de sus fundamentos. Podemos utilizar cualquiera de las operaciones matemáticas básicas que podrías pensar como suma (+), resta (-), división (/), y multiplicación (*). Sin embargo, hay algunas opciones que tenemos que podrían no ser tan intuitivas:

Módulo(%)

Esto también se conoce como cálculo del resto. Básicamente, podemos usarlo para ver si dividimos dos números, ¿qué quedaría?

```
console.log(5 % 2)   //Esto imprimiría 1
console.log(6 % 2)   //Esto imprimiría 0
console.log(8 % 3)   //Esto imprimiría 2
```

Esto es especialmente útil para ver si un número es divisible uniformemente por otro. En el ejemplo anterior, dado que había un resto de 1 con 5%2, significa que 5 no es divisible uniformemente por 2. Sin embargo, dado que 6%2 da un resto de 0, podemos decir que 6 es divisible uniformemente por 2.

Eliminar lugares decimales

Al dividir números, a menudo nos encontramos con algún decimal. Sin embargo, muchas veces no queremos un decimal en nuestro número resultante. JavaScript nos da varias formas de eliminar el decimal y volver a un número entero.

Math.round(num)

Esto redondea un número como estamos acostumbrados. Si el decimal es 0,5 o más, se redondeará al siguiente número; mientras que, si el número es 0,49 o menos, se redondeará a la baja:

```
var num1 = Math.round(2.5)   //num1 es 3

var num2 = Math.round(2.4)   //num2 es 2

var num3 = Math.round(-2.9)  //num3 es -3

var num4 = Math.round(-2.1)  //num4 es -2
```

Math.floor(num)

Esto siempre redondea un número hacia abajo:

```
var num1 = Math.floor(2.5)   //num1 es 2

var num2 = Math.floor(2.4)   //num2 es 2

var num3 = Math.floor(-2.9)  //num3 es -3

var num4 = Math.floor(-2.1)  //num4 es -3
```

Math.ceil(num)

Esto siempre redondea un número hacia arriba:

```
var num1 = Math.ceil(2.5)   //num1 es 3

var num2 = Math.ceil(2.4)   //num2 es 3

var num3 = Math.ceil(-2.9)  //num3 es -2

var num4 = Math.ceil(-2.1)  //num4 es -2
```

Math.trunc(num)

Esto siempre trunca el número, cortando el decimal independientemente de cuál sea el número:

```
var num1 = Math.trunc(2.5)   //num1 es 2

var num2 = Math.trunc(2.4)   //num2 es 2

var num3 = Math.trunc(-2.9)  //num3 es -2

var num4 = Math.trunc(-2.1)  //num4 es -2
```

Funciones integradas y denominación de funciones

Las funciones floor(), ceil() y trunc() resaltan dos claves muy importantes sobre las funciones. La primera es el nombre de las funciones. Puedes inferir lo que floor(), ceil() y trunc() van a hacer con los números basándote únicamente en sus nombres. floor() lleva los números al suelo, ceil() levanta los números hasta el techo y trunc() trunca los lugares decimales. La segunda clave es que se denominan funciones integradas ... ¿por qué "integradas"? Ciertamente no construimos estas funciones nosotros mismos, alguien más lo hizo para que pudiéramos reutilizarlas cuando las necesitáramos. Eso es lo mejor de la mayoría de los lenguajes de programación: vienen con muchas herramientas útiles listas para usar. A medida que avanzas en el programa, desafíate a usar las funciones integradas con moderación. Queremos que dediques tu tiempo especialmente a los algoritmos y a pensar en los problemas que presentan y no memorizar una función interesante que hace todo el trabajo pesado por ti. Si tienes preguntas sobre las funciones que puedes usar, ¡pregunta a tus instructores!

¿Qué es Full Stack?

Full Stack es un conjunto de tecnologías necesarias para ejecutar una aplicación. Hay 3 partes principales en una aplicación full stack:

Front End: HTML, CSS, JavaScript, por lo general, las abreviaturas de full stack indican los marcos de JavaScript utilizados, pero no es necesario incluirlos.

Back End: Python, Ruby, C#, Java, PHP, generalmente especificando el uso del lenguaje de programación y/o el marco para el lenguaje

Base de datos: SQL, MongoDB, CouchDB, Redis, especificando la base de datos utilizada para el stack, pero no es requerida.

Full stack se refiere a la serie de tecnologías utilizadas en un "stack", pero es posible que el nombre del stack no indique directamente todas las tecnologías. Dicho esto, es bueno saber que un desarrollador full-stack es aquel que puede construir una aplicación web completa. No necesitan una persona de base de datos, una persona de back-end o una persona de front-end para realizar su proyecto.

Los 4 niveles de un programador

Subir de nivel como programador es como ganar rangos en el ejército. Se necesita tiempo, práctica y cada nivel se presenta con nuevos conceptos necesarios para aprender. Hay 4 niveles principales de programación que identificamos universalmente para casi todos los lenguajes de programación web.

- Programación procedimental
- Programación orientada a objetos (O.O.P.)
- M.V.C.
- O.R.M.

Procedimental

Comenzamos con la procedimental, escribiendo código en 1 archivo. Lo llamamos procedimental porque el código se ejecuta de arriba a abajo con poca complejidad. Nuestros juegos de ejemplo (Ninja Game, Pacman, Avión) están escritos en programación procedimental. Este es un gran comienzo en la programación, simple y aquí es donde todo comenzó inicialmente.

Programación orientada a objetos (O.O.P.)

Después de la procedimental, viene la Programación orientada a objetos, que brinda a los desarrolladores una herramienta para poder recrear objetos o bloques de código con 1 comando, reduciendo el código y aumentando la eficiencia. Es importante aprender esta luego de la procedimental, para que puedas apreciar los beneficios que ofrece.

M.V.C.

Luego viene la M.V.C.: esta es la Programación orientada a objetos con más estructura, tenemos ciertos archivos o bloques de código que están dedicados a un trabajo determinado. Esta es ideal principalmente para equipos más grandes o proyectos grandes, o si simplemente quieres una estructura más determinada.

M - Modelos: archivos o bloques de código que se comunican con la base de datos, recuperando o enviándole información.

V - Vistas: archivos o bloques de código que se cargan como plantillas, generalmente son archivos HTML.

C - Controladores: archivos o bloques de código que dirigen al servidor para obtener información de Modelos o cargar Vistas. Los controladores son como controladores de tráfico aéreo, toman todas las decisiones por el servidor.

O.R.M.

ORM es una forma de estructurar nuestros modelos, y más específicamente, las relaciones entre modelos, de manera que podamos usar los comandos ORM para consultar la base de datos de manera más eficiente. Con unas pocas palabras, podemos recuperar o agregar datos a una base de datos que normalmente tomaría 1-2 líneas de sentencias SQL. Esto ofrece mayormente eficiencia en la escritura de código y no eficiencia en la velocidad del programa. Sin embargo, es muy beneficioso para los equipos que construyen MVP. (Modelo-Vista-Controlador) y equipos que necesitan construir cosas de forma rápida.

Conclusión

Como puedes ver, te beneficia aprender a programar desde el nivel 1 y e ir subiendo de nivel, no solo para comprender mejor la programación, sino también para comprender por qué usarías cada uno de estos diferentes niveles y poder identificar cuándo usarlos y cuándo no. Esto es lo que te diferenciará de la mayoría de los desarrolladores, la comprensión fundamental de la programación.

Entonces, ¿cuál es el mejor nivel para codificar? ¿Y cuál es el mejor lenguaje de programación? ¿O el mejor marco? Depende, siempre. Queremos que aprendas qué problemas resuelven estas herramientas para que tú, “el desarrollador”, pueda tomar decisiones informadas al elegir el lenguaje, el marco (básicamente el nivel de programación en el que codificas) al crear una aplicación.

Plan de estudios de Coding Dojo

El plan de estudios de Coding Dojo comenzó hace casi 10 años en un esfuerzo por capacitar a los empleados internos en una empresa de desarrollo. En ese momento, el desafío era que incluso los graduados de CS necesitaban de 6 a 9 meses de capacitación para estar listos para escribir código de producción. Aquí es cuando comenzó el plan de estudios de Coding Dojo. Nuestro fundador (Michael Choi) creó un plan de estudios para ayudar a sus empleados a ponerse al día a un ritmo mucho más rápido. Avanza rápidamente 10 años, 5 de los cuales Coding Dojo lleva operando, con revisiones continuas a nuestro contenido, y llega a lo que hemos dedicado tanto tiempo, esfuerzo pasión: el Bootcamp de Coding Dojo.

Lo que nos distingue es nuestro enfoque de la enseñanza. Nos esforzamos por enseñarte los fundamentos de la programación (que son universales) y lo hacemos a través de un programa enfocado e intenso, que ofrece orientación para aprender el material en el orden y ritmo correctos. En última instancia, nuestro plan de estudios se esfuerza por convertirte en un desarrollador autosuficiente. Omitimos deliberadamente ciertos códigos e instrucciones. Esto te da la oportunidad de pensar críticamente y unir las cosas. Cuando pasas por este proceso, aprendes y retienes la información a un ritmo mucho más rápido.

El siguiente patrón será lo que encontrarás en nuestro plan de estudios:

- Vídeos de descripción general
- Contenido escrito
- Cuestionarios
- Asignaciones
- Asignaciones opcionales

Esto se hace por diseño, esto te ofrece la oportunidad de escuchar a nuestros instructores hablar sobre por qué estás aprendiendo los nuevos temas, cómo son importantes e incluso darte una muestra rápida del contenido. A esto le sigue contenido escrito que cubre los temas con suficiente detalle para que empieces a codificar. Código, código, código, no hay sustituto para esto. Verás que damos muchos proyectos, pequeños a grandes, y te alentamos a asociarte con 1 o 2 personas para trabajar juntos y discutir las asignaciones mientras lo construyes. Este es un aspecto clave para tener éxito en Coding Dojo y, en última instancia, ser un desarrollador autosuficiente; un desarrollador que no necesita ayuda para crear nuevas aplicaciones, sin importar la complejidad.

Descripción general

El objetivo de este curso es presentar los conceptos básicos de programación de computadoras. El curso está diseñado específicamente para aquellos que son nuevos en programación.

Cuando se trata de codificación/programación, todos los programas (web, móvil, escritorio, robots, etc.) se construyen utilizando esencialmente algunos componentes fundamentales: variables, declaraciones if/else, arreglos, bucles for y funciones.

La combinación de estos cinco componentes esenciales permite a los desarrolladores crear todo lo que puedan imaginar (por ejemplo, una aplicación móvil, una aplicación de escritorio, una aplicación web, sistemas operativos, aplicaciones AR/VR, programas informáticos utilizados para ejecutar robots). Un buen programador sabe cómo combinar bien estos componentes y un programador que todavía no es bueno aún no está capacitado para combinarlos de la manera correcta.

El objetivo de este minicurso es ayudarte a ver cómo estos componentes básicos se combinan para crear algo. Para permitirte ver visualmente cómo se pueden usar estas partes esenciales para crear programas, usaremos el navegador (que ya entiende cómo ejecutar programas JavaScript) y un editor de texto simple para crear tu aplicación. Aunque te vamos a enseñar algo de JavaScript, HTML y CSS básico para mostrarte cómo puedes combinar estos componentes, la lógica/habilidades que aprendas en este curso se pueden aplicar a cualquier tipo de programación (para crear cualquier tipo de software para escritorio, dispositivos móviles o web).

Lo que aprenderás será lo siguiente

(5 minutos) Introducción a JavaScript

(10 minutos) Introducción a HTML, CSS y JavaScript básicos - Ninja Walking Game

(15 minutos) ¡Crear PacMan desde cero!

(15 minutos) ¡Crear un juego de aviones que disparan desde cero!

Para aquellos que estén interesados, habrá otros capítulos opcionales que les permitirán profundizar. ¡El objetivo es que en 30 minutos aprendas lo suficiente para poder crear un juego simple!

Para aquellos interesados en ser aceptados en nuestros programas, pueden trabajar en los desafíos de estos juegos y, durante su entrevista, mostrarnos lo que han construido.

Programación Básica con JavaScript

```
var x = 15;
var y = "CodingDojo";
var x_array = [1,5,10,15];
var y_array = ["Coding", "Dojo", "es", "asombroso!"];
document.write('hola');  //escribe "hola" en el documento
document.write('hola ', y);  //escribe 'hola CodingDojo' en los documentos
console.log(x);  //registra 15 en la consola
console.log(y);  //registra "CodingDojo" en la consola
console.log(y_array[3]);  //registra el valor en y_array índice 3 que es 'asombroso!';
//esto imprime 0 a 99
for(var i=0; i<100; i++){
  console.log(i);
}
//recorre de 1 a 100 e imprime si el número es par o impar
for(var j=1; j<=100; j++){
    if(j%2 == 0) {
        console.log(j, ' es un número Par');
    }
    else {
        console.log(j, 'es un número Impar');
    }
}
function sum(x,y){
    console.log('Tengo los dos siguientes parámetros', x, y);
    return x+y;
}
console.log(sum(5,3));  //registra 8
console.log(sum(10,15) + sum(3,5));  //registra 33
```

Aplicación de algoritmos

Recomendamos encarecidamente a todos los estudiantes que dediquen un tiempo a revisar la Aplicación del algoritmos (debería tomar entre 30 minutos y algunas horas, según tu experiencia).

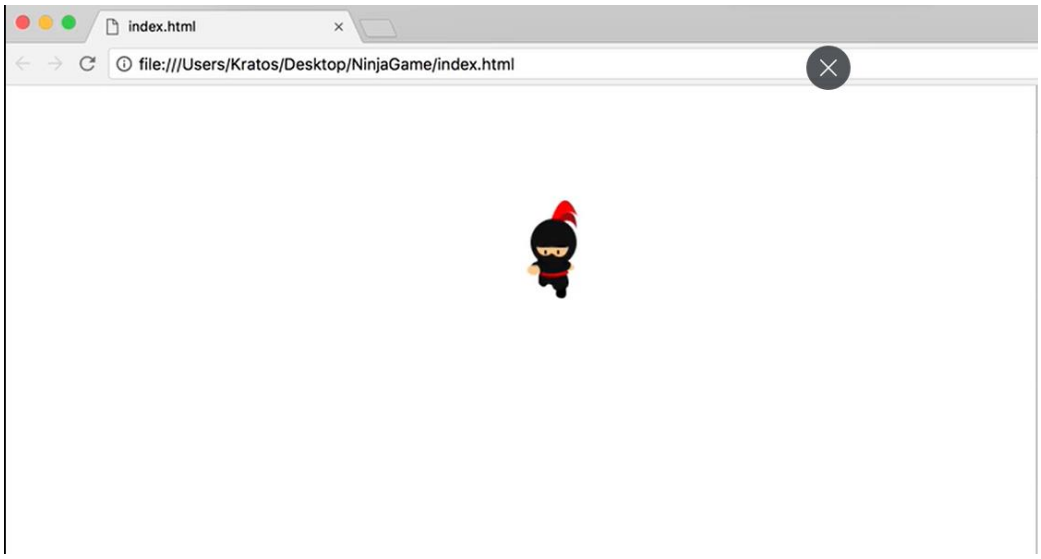
Juego del Ninja

¡Bienvenido a tu primer juego de JavaScript! ¡Codificar es MUY divertido! En el mundo del código, podemos construir cualquier cosa que imaginemos, realmente no hay límites dentro del código. Este juego te ofrecerá la oportunidad de construir algo con nuestra ayuda para que comiences y para ayudarte a ver los patrones de programación. Recuerda, todo se deriva de variables, funciones, bucles y arreglos. Por ahora, usaremos variables y funciones.

Para este juego, usaremos variables y funciones para mover un Ninja en nuestra pantalla. Es importante mantenerlo simple al principio, por lo que te proporcionaremos parte del código, lo construiremos frente a ti y te daremos la oportunidad de terminar el código tú mismo. Una vez que hayas terminado, puedes volver a recrearlo codificando al mismo tiempo que yo. Después de un par de repeticiones, podrás construir esto por tu cuenta mirando poco o nada el código dado.

Conceptos básicos de JavaScript

Los navegadores tienen un intérprete de JavaScript integrado donde podemos ejecutar JavaScript para decirle al navegador qué hacer. Por ejemplo, creemos el siguiente archivo llamado ninja.html donde agregaremos HTML, CSS y JavaScript simples. Para esta tarea, puedes seguir este enlace y descargar una carpeta prefabricada con el archivo index.html y las imágenes. Puedes hacerlo haciendo clic aquí.



```
<html>
  <body>
    <div id='background'>
      <div id='character' style='position:absolute; top:100px; left:450px; background-image:
url("img/down1.png"); width:59px; height:86px;'></div>
    </div>
    <script type="text/javascript">
      var leftValue = 450, topValue = 100;
      function update(){
        document.getElementById("character").style.left = leftValue+"px";
        document.getElementById("character").style.top = topValue+"px";
      }
      document.onkeydown = function(e) {
        console.log('e: ', e);
        console.log('e.keyCode: ', e.keyCode);
        if (e.keyCode == 37) { // LEFT
          leftValue = leftValue - 10;
        }
        else if (e.keyCode == 39) { // RIGHT
          leftValue = leftValue + 10;
        }
        else if (e.keyCode == 40) { // DOWN
          topValue = topValue + 10;
        }
        update();
      }
    </script>
  </body>
</html>
```

Desafíos

¿Cómo conseguirías que el personaje ascendiera?

Actualmente, el personaje puede ir a todas partes, incluso donde leftValue es negativo o topValue es negativo. ¿Cómo ajustarías el código anterior para que el personaje permanezca entre 0 y 500 para las variables leftValue y topValue?

Puedes utilizar la siguiente carpeta para comenzar, descárgala haciendo clic aquí.

Juego del Ninja II

¡Ahora hagamos que el juego sea un poco más emocionante haciendo que el personaje parezca que realmente camina!

```
<html>
  <body>
    <div id="background" style="width: 900px; height: 750px; background-image:
url('img/white_bg.jpg');">
      <div id="character" style="width: 59px; height: 86px; background-image: url('img/down1.png');
position: absolute; left: 150px; top: 150px;"></div>
    </div>
    <script type="text/javascript">
      var leftValue = 150;
      var topValue = 150;
      var walkValue = 1;
      document.onkeydown = function(e){
        console.log(e);
        if (walkValue == 1){
          walkValue = 2;
        }
        else if (walkValue == 2){
          walkValue = 1;
        }
        if(e.keyCode == 37){  // Left
          leftValue = leftValue - 25;
          document.getElementById("character").style.left = leftValue+"px";
          document.getElementById("character").style.backgroundImage =
"url('img/left"+walkValue+".png')";
        }
        else if (e.keyCode == 39){  // Right
          leftValue = leftValue + 25;
          document.getElementById("character").style.left = leftValue+"px";
          document.getElementById("character").style.backgroundImage =
"url('img/right"+walkValue+".png')";
        }
        else if (e.keyCode == 38){  // Top
          topValue = topValue - 25;
          document.getElementById("character").style.top = topValue+"px";
        }
        else if (e.keyCode == 40){  // Bottom
          topValue = topValue + 25;
          document.getElementById("character").style.top = topValue+"px";
        }
      }
    </script>
  </body>
</html>
```

Desafíos

¿Cómo agregarías la instrucción “arriba”?

Nuestro Ninja se está moviendo, pero no parece que esté caminando. ¿Qué cambiarías para que pareciera que el ninja camina?

Ninja Man

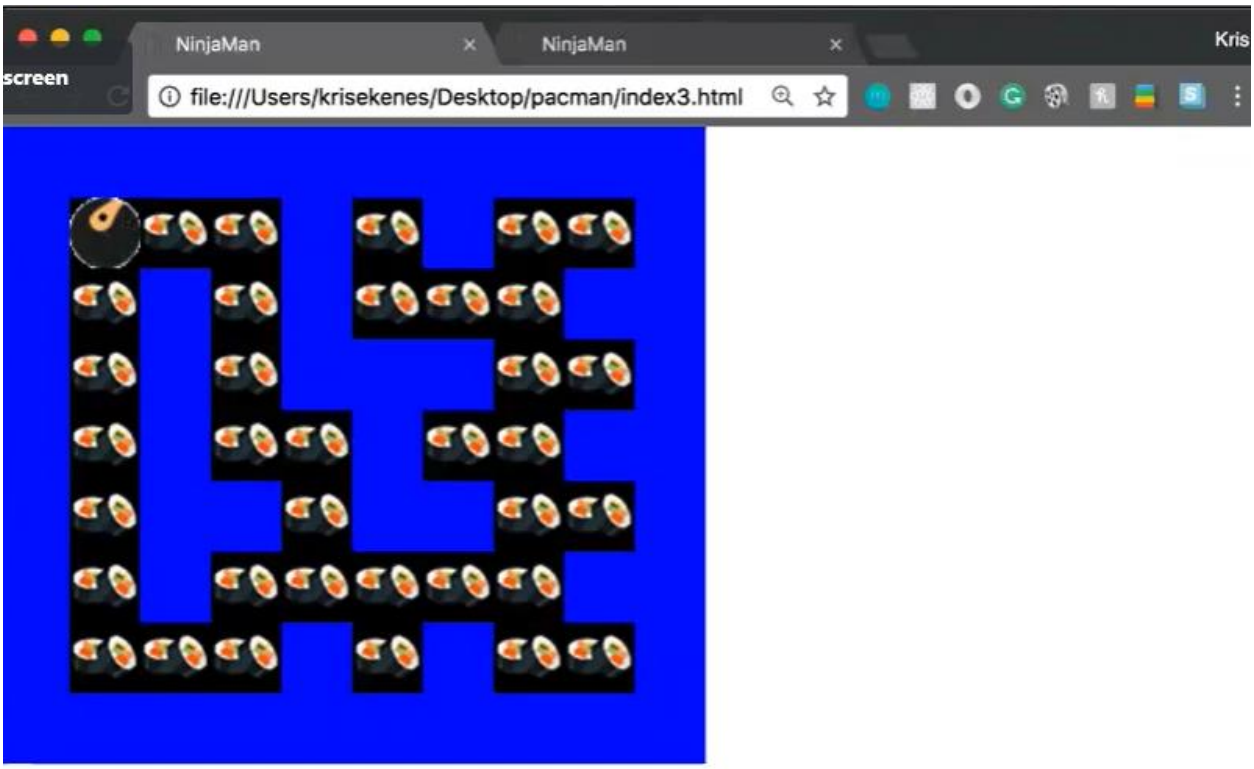
En esta sección, crearás el juego NinjaMan basado en el popular juego PacMan. Desde el principio, esto suena complicado, pero ya has creado todos los componentes que necesitas para comprender cómo crear este juego. Para abordar esta aplicación, dividiremos NinjaMan en algunas partes. Primero, la elaboración de HTML y CSS como herramientas para el diseño de nuestro algoritmo. En segundo lugar, creemos un algoritmo para representar el HTML apropiado para el mundo de nuestro NinjaMan. Por último, necesitaremos posicionar a NinjaMan en este mundo y, al presionar una tecla, mover a NinjaMan en la dirección apropiada.

También habrá algunos desafíos para que completes cuando hayamos terminado.

Cubriremos lo siguiente:

- Descripción general de CSS
- Elaboración de HTML y CSS para crear contenido estático
- Usar un algoritmo para crear HTML para nosotros
- Posicionar a nuestro NinjaMan
- Hacer que nuestro NinjaMan se mueva al presionar una tecla
- Detectar paredes
- Comer sushi

Puedes encontrar las imágenes aquí.



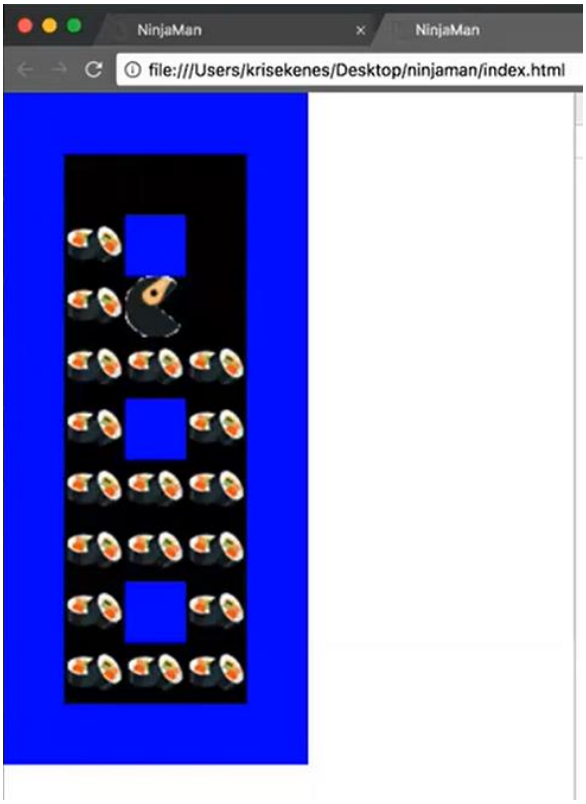
Recreando NinjaMan con contenido estático

Ahora que sabemos cómo debería ser el mundo de NinjaMan, ¿cómo haríamos para crearlo usando un algoritmo?

Crear el mundo de NinjaMan

Así que tenemos nuestro mundo NinjaMan, pero nuestro NinjaMan no se mueve ni interactúa con este mundo. Conocemos la base de lo que debería estar ocurriendo, así que asegúrate de intentar mover a NinjaMan.

Hacer que NinjaMan se mueva



Tu desafío será:

- (Básico) Lleva el puntaje de cuántas piezas de sushi come NinjaMan
- (Básico) Agrega Onigiri como alimento alternativo para comer
- (Intermedio) Genera un mundo aleatorio cuando se carga la página
- (Avanzado) Agrega fantasmas que persigan a NinjaMan
- (Avanzado) Dale a NinjaMan 3 vidas. El Juego termina cuando un fantasma toca a NinjaMan 3 veces.

Juego del Avión

¡Bienvenido al nivel final de nuestra Introducción a la programación! ¡Ahora es el momento de tomar lo que hemos aprendido y crear un avión que podrás personalizar y jugar tú mismo!

Usaremos los mismos conceptos que hemos aprendido hasta ahora:

- Variables
- Condicionales
- Bucles
- Funciones
- Bucle de juego

Si no tuviste mucha práctica en el bucle de juego en Pacman, lo repasaremos juntos en esta sección, pero en resumen, un bucle de juego es simplemente una función que queremos llamar cada X cantidad de segundos. Esto se puede usar para mover enemigos, misiles y también verificar colisiones.

Construyamos juntos un juego de aviones básico y, a lo largo del camino, te daremos desafíos para completar. Una vez que hayas pasado por todo el juego, te animamos a que lo sigas construyendo y, más tarde, intentes personalizar lo que tienes, o incluso crearlo tú mismo, y lo más importante, ¡divertirte con él!

Parte 1

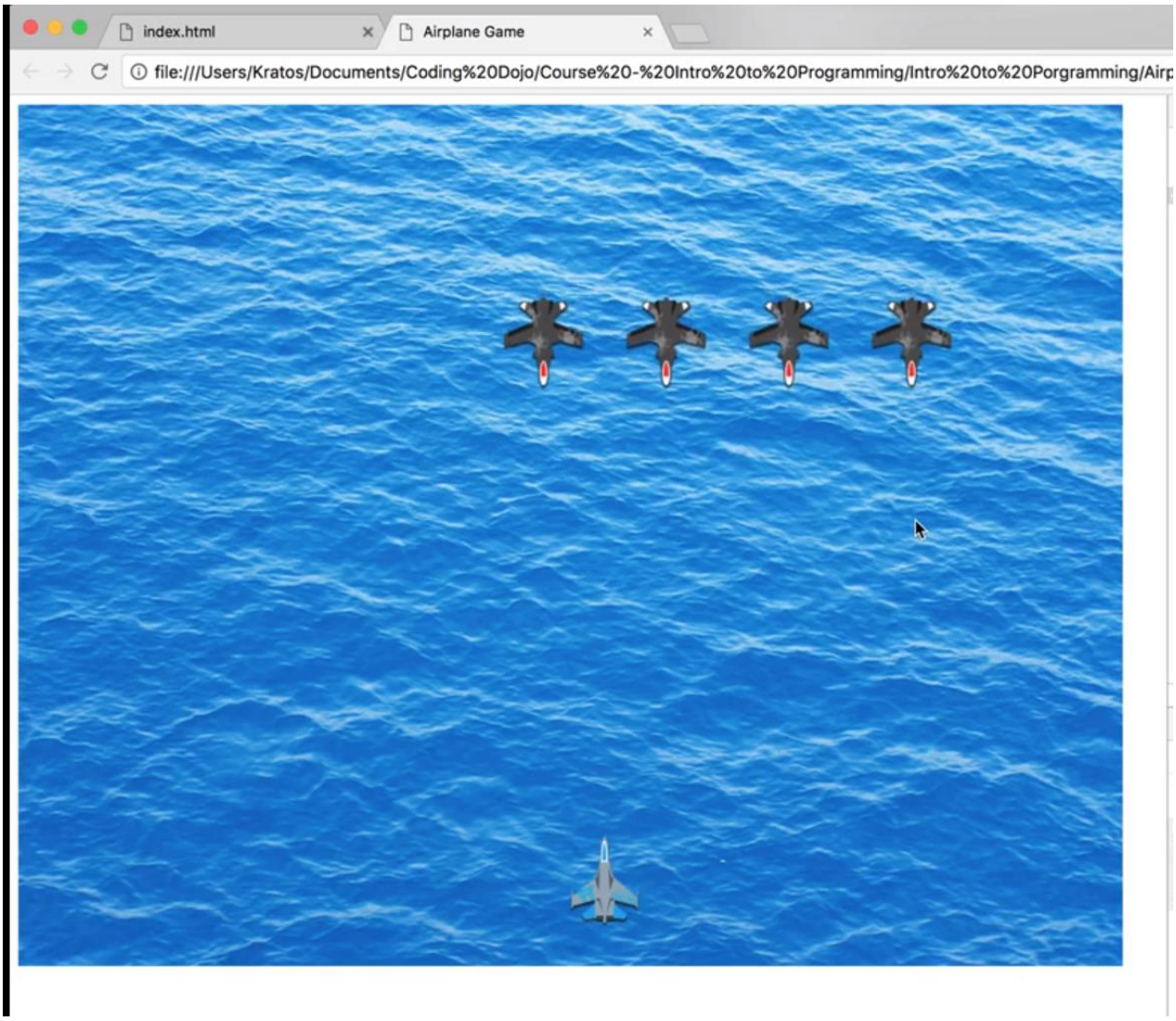
Cubriremos lo siguiente:

- Construir divisiones para el océano, los jugadores y los enemigos.
- Darle estilo al océano.
- Crear un objeto de jugador para mantener su posición.
- Crea un arreglo de enemigos, y objetos para las posiciones.
- Crear la función de drawPlayer para dibujar jugadores.
- Crear la función de drawEnemies para dibujar enemigos.
- Crear un disparador de onkeydown para moverse hacia la izquierda y hacia la derecha.

Tu desafío será:

- (básico) Agregar característica: Mover al jugador hacia arriba y hacia abajo.
- (básico) Agregar más enemigos (agrega 4 más en el océano).
- (intermedio) Cambiar el fondo (diferentes colores o tus propias imágenes).
- (intermedio) Cambiar avión enemigo (tus propias imágenes).
- (intermedio) Cambiar avión del héroe (tus propias imágenes).
- (avanzado) Modificación de característica: limita el rango de subida/bajada del jugador a aproximadamente 1/3 de la parte inferior de la pantalla.

Puedes tomar los elementos y un archivo index.html inicial descargando la siguiente carpeta aquí.



Juego del Avión

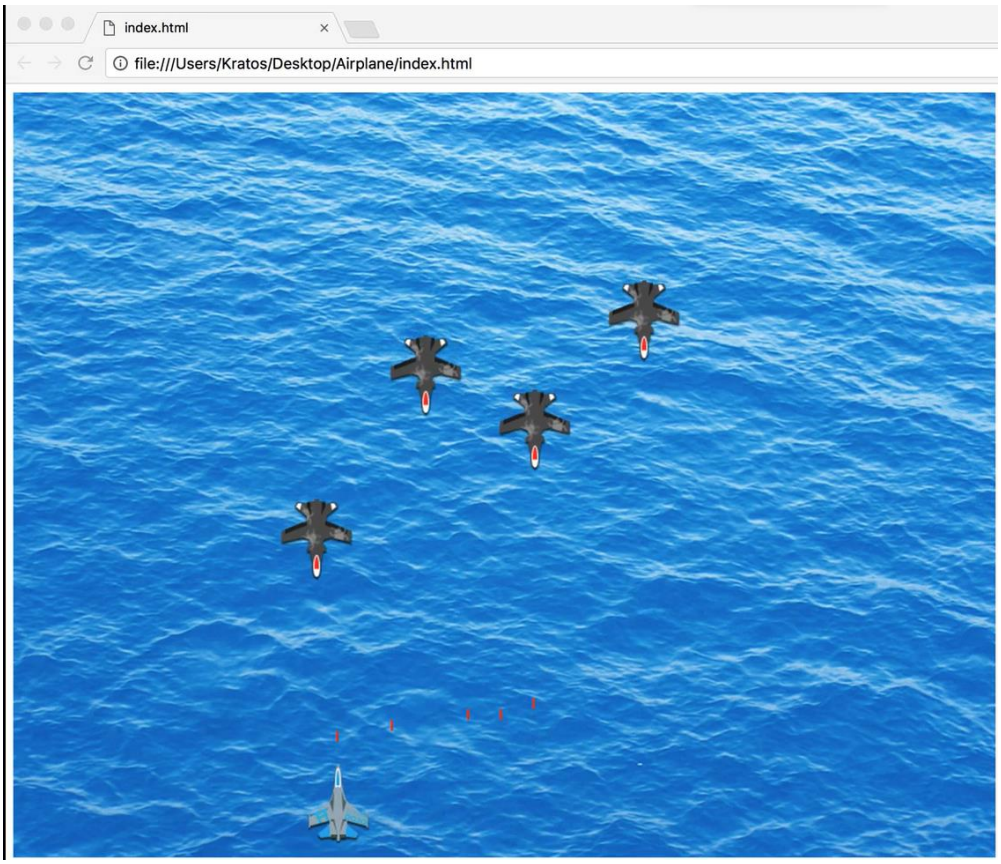
Parte 2

Cubriremos lo siguiente:

- Agregar una función de bucle de juego.
- Crear la función moveEnemies: para mover a los enemigos en cada ciclo de gameLoop
- Construye una división para contener los misiles del jugador
- Haz que nuestro héroe dispare misiles.
- Mover enemigos
- Mover misiles

Tu desafío será:

- (básico) Cambiar el color de los misiles.
- (intermedio) Haz que los enemigos se muevan más rápido.
- (intermedio) Haz que los misiles se muevan más rápido.
- (avanzado) Haz que el juego parezca más fluido (pista: velocidad del bucle de juego)



¡Misión cumplida!

¡Buen trabajo por llegar tan lejos! Esta misión está cumplida, pero con 1 logro, ¡se te han abierto muchas otras oportunidades! ¡Deberías estar orgulloso de tu logro y de haber llegado tan lejos! Esto apenas es un rasguño de la superficie de la programación, pero esperamos que te haya dado una idea de lo divertido y emocionante que puede ser escribir código, ¡y esperamos que te hayas divertido tanto como nosotros!

Desafíos Obligatorios

HTML
Predecir I
Predecir II
Return
Ninja I