



Girls Who Code en casa

Depuración de código faltante, parte 2
Guía de referencia

Depuración de código faltante, parte 2 Guía de referencia



En este documento encontrará todas las respuestas a algunas de las preguntas de la actividad. Siga la actividad y cuando vea este ícono, deténgase y revise sus conocimientos aquí.

Paso 1: Conocer los errores lógicos

¿Por qué los errores lógicos causan problemas?

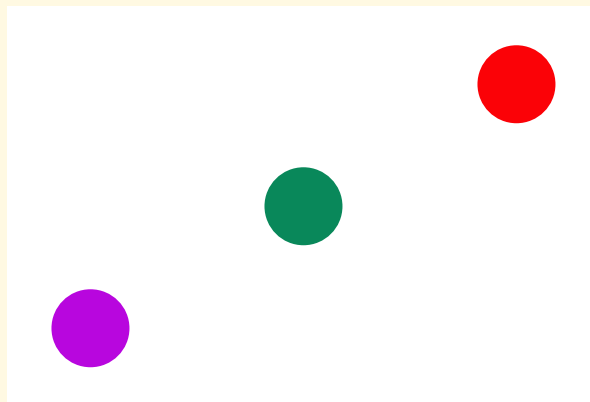
El color será amarronado porque el programa no le indica al pintor que debe limpiar el pincel después de cada uso. Solo le indica que limpie el pincel al final, cuando termine de pintar.

Para que el último círculo sea de color púrpura vibrante, usted debería añadir la instrucción **Rinse brush** (Enjuagar pincel) después de cada color:

CÓDIGO

```
Mojar el pincel en pintura roja
Pintar un círculo
Enjuagar el pincel
Mojar el pincel en pintura verde
Pintar un círculo
Enjuagar el pincel
Mojar el pincel en pintura púrpura
Pintar un círculo
Enjuagar el pincel
Dejar secar el lienzo
```

RESULTADOS



Paso 4: Revisar el código

A continuación, se incluye una explicación detallada del [código roto](#) al comienzo del paso.

Declarar variables

En primer lugar, declaramos las variables para almacenar la puntuación de cada bicho (p. ej., **butterflyScore**). Luego, declaramos una serie de variables para almacenar el elemento HTML de cada uno de los botones de respuesta en nuestras preguntas (p. ej., **q2a3**). También declaramos variables para almacenar en cuántas preguntas ha hecho clic una persona (**questionCount**) y para almacenar el resultado y reiniciar los elementos HTML.

Escuchar los clics de los botones

A continuación, agregaremos agentes de escucha de eventos a todos nuestros botones. [Los agentes de escucha de eventos](#) escuchan determinadas entradas en una página web, como un clic del mouse, la pulsación de una tecla o un desplazamiento. Si el navegador “oye” o detecta la entrada que está esperando, el programa realiza una acción. Esta acción la definimos en una función que va adentro del agente de escucha de eventos. Por ejemplo, si la persona elige **Mochi** en la pregunta 1, hace clic en el botón **q1a1**. Cuando hace clic en el botón, el navegador ejecuta el código en la función **butterfly**.

JAVASCRIPT	DESCRIPCIÓN
<code>q1a1.addEventListener("click", butterfly);</code>	<ul style="list-style-type: none">→ q1a1. Variable que almacena la entrada del botón de mochi (la primera respuesta a la primera pregunta).→ <code>.</code>. El punto le indica a nuestro programa que adjunte la variable al método que le sigue. Aquí el método es <code>addEventListener</code>.→ addEventListener. La palabra clave para llamar al método (o la función) de agente de escucha de eventos.→ "click". El evento que se está escuchando.→ butterfly. La función que se ejecutará si se produce el evento.→ <code>;</code>. Todas las sentencias de JavaScript terminan con un punto y coma.

Registrar y actualizar la puntuación

Ahora tenemos que crear las funciones que hemos enumerado dentro de nuestros agentes de escucha de eventos. Cuatro de ellos registran la puntuación de cada bicho: **abeja**, **mariposa**, **saltamontes** y **mariquita**. Si se llama a una de estas funciones (es decir, cuando una persona hace clic en un botón con esa función asociada), aumentamos la puntuación del bicho en 1 y aumentamos la variable **questionCount** en 1.

A continuación, tenemos una sentencia condicional para evaluar cuando el cuestionario esté completo. Si **questionCount** es igual a 3 (usamos el [operador de comparaciones](#) `==` aquí, en lugar del operador de asignaciones `=`), entonces ejecutamos la función **updateResult** que sigue. Esta función comprueba cada puntuación para ver si es mayor o igual a 2. Si una cumple esa condición, esta función actualiza el texto de la parte inferior con el resultado. Si ninguna lo hace, entonces le pedimos a la persona que lo intente de nuevo.

Reiniciar el cuestionario

La función final se ejecuta si se hace clic en el botón de reinicio (*Restart*). En este caso, reiniciar significa restablecer a cero todas las puntuaciones y el valor **questionCount** en 0. De lo contrario, los números seguirán aumentando.

Paso 5: Describir el problema

A continuación, se incluye un ejemplo de una forma de describir lo que queremos que ocurra en el programa.

- Cuando una persona hace clic en un botón de respuesta, el programa llama a la función de bicho especificada (p. ej., **abeja**, **mariposa**, **saltamontes** o **mariquita**) asociada en el agente de escucha de eventos.
- La función suma 1 a la puntuación actual de ese bicho y añade 1 al valor actual **questionCount**.
- Pruebe si el valor **questionCount** es igual a 3 cada vez que el valor **questionCount** aumenta.
Nota: Dado que el cuestionario tiene tres preguntas, sabemos que la persona ha terminado si este valor es igual a 3. Si tuviéramos cuatro preguntas, lo cambiaríamos a 4.
- Si **questionCount** no es igual a 3, no haga nada y deje que la persona continúe el cuestionario.
- Si **questionCount** es igual a 3, llame a la función **updateResult** para probar el valor de la puntuación de cada bicho.
- Evalúe la puntuación de cada bicho y devuelva un resultado en la función **updateResult**:
 - ◆ Si **beeScore** es mayor o igual a 2, actualice el texto de abajo para que diga "¡Usted es una abeja!".
 - ◆ De otra manera, si **butterflyScore** es mayor o igual a 2, actualice el texto de abajo para que diga "¡Usted es una mariposa!".
 - ◆ De otra manera, si **grasshopperScore** es mayor o igual a 2, actualice el texto de abajo para que diga "¡Usted es un saltamontes!".
 - ◆ De otra manera, si **ladybugScore** es mayor o igual a 2, actualice el texto de abajo para que diga "¡Usted es una mariquita!".
 - ◆ De otra manera, actualice el texto de abajo para que diga "Mmm... tengo dudas. Inténtelo de nuevo más tarde".
- Cuando se hace clic en el botón de reinicio (*Restart*), se llama a la función **restartQuiz** para que establezca todas las variables en 0.

Paso 6: Hacerlo observable

Codifique con **console.log()**, además de cada función de puntaje:

```
58 // Registrar el puntaje de la abeja
59 function bee() {
60     beeScore += 1;
61     questionCount += 1;
62     console.log("questionCount = " + questionCount + "\t" + "beeScore = " +
63     beeScore);
64 }
```

```

65 // Registrar el puntaje de la mariposa
66 function butterfly() {
67     butterflyScore += 1;
68     questionCount += 1;
69     console.log("questionCount = " + questionCount + "\t" + "butterflyScore = "
70         + butterflyScore);
71 }
72 // Registrar el puntaje del saltamontes
73 function grasshopper() {
74     grasshopperScore += 1;
75     questionCount += 1;
76     console.log("questionCount = " + questionCount + "\t" + "grasshopperScore = "
77         + grasshopperScore);
78 }
79 // Registrar el puntaje de la mariquita
80 function ladybug() {
81     ladybugScore += 1;
82     questionCount += 1;
83     console.log("questionCount = " + questionCount + "\t" + "ladybugScore = "
84         + ladybugScore);
85 }

```

Respuestas a nuestras pistas:

- ☐ ¿Qué controla el texto resultante?
 - ☐ La función `updateResult`.
- ☐ ¿Cuándo llamamos (o usamos) a la función `updateResult`?
 - ☐ En la sentencia condicional que evalúa cuando `questionCount` es igual a 3.
- ☐ ¿La sentencia condicional se ejecuta cuando la necesitamos?
 - ☐ ¡No!

Paso 8: Probar una cosa a la vez

Hipótesis 1: Prueba de la sentencia condicional questionCount en la función de la abeja

```
58 // Registrar el puntaje de la abeja
59 function bee() {
60     beeScore += 1;
61     questionCount += 1;
62     console.log("questionCount = " + questionCount + "\t" + "beeScore = " +
        beeScore);
63     // Agregar sentencia condicional
64     if (questionCount == 3){
65         updateResult();
66     }
67 }
```

Hipótesis 2: Prueba de la sentencia condicional questionCount en las demás funciones

```
69 // Registrar el puntaje de la mariposa
70 function butterfly() {
71     butterflyScore += 1;
72     questionCount += 1;
73     console.log("questionCount = " + questionCount + "\t" + "butterflyScore = "
        + butterflyScore);
74     // Agregar sentencia condicional
75     if (questionCount == 3){
76         updateResult();
77     }
78 }
79
80 // Registrar el puntaje del saltamontes
81 function grasshopper() {
82     grasshopperScore += 1;
83     questionCount += 1;
84     console.log("questionCount = " + questionCount + "\t" + "grasshopperScore = "
        + grasshopperScore);
85     // Agregar sentencia condicional
86     if (questionCount == 3){
87         updateResult();
88     }
89 }
```

```

91 // Registrar el puntaje de la mariquita
92 function ladybug() {
93     ladybugScore += 1;
94     questionCount += 1;
95     console.log("questionCount = " + questionCount + "\t" + "ladybugScore = "
96               + ladybugScore);
97     // Agregar sentencia condicional
98     if (questionCount == 3){
99         updateResult();
100     }
101 }

```



¡Error arreglado!

En este error, el programa nunca evaluó la sentencia condicional que llamaba a la función `updateResult`. Aunque pueda parecer lógico probar el valor de `questionCount` después de todas las funciones de puntuación de bichos, en realidad debemos probarlo cada vez que se ejecuta una función de puntuación de bichos. Resolvimos el problema colocando la sentencia condicional dentro de cada función de puntuación de bichos para que el programa pueda comprobar el valor de `questionCount` después de que la persona haga clic en un botón.