



Girls Who Code At Home

गुम कोड को डीबग करें: भाग 2
लॉजिक बग्स

गतिविधि अवलोकन

भाग 1 में, हमने कंसोल विंडो का उपयोग करके बग वाली व्यक्तित्व प्रश्नोत्तरी में सिंटेक्स त्रुटियों को पहचानने और डिबग करने का तरीका सीखा। भाग 2 में, हम एक नए प्रकार के बग की जांच करने जा रहे हैं, जो एक लॉजिक बग है, और हमारी बग वाली व्यक्तित्व संहिता की मरम्मत के लिए डिबगिंग रणनीतियों का एक सेट सीखने वाले हैं। प्रोग्रामर्स अपने प्रोग्राम को यह बताने के लिए तर्क का उपयोग करते हैं कि क्या कार्रवाई करनी है। कभी-कभी, हालांकि, हमारे स्वयं के तर्क उन निर्देशों की सीध में नहीं होते हैं जिन की कंप्यूटर को सफलतापूर्वक प्रोग्राम चलाने के लिए आवश्यकता होती है। हम जावास्क्रिप्ट के साथ काम करेंगे जैसे हमने भाग 1 में किया था, लेकिन डिबगिंग पद्धति सभी भाषाओं पर लागू होती है। इस गतिविधि को पूरा करने के लिए आपको जावास्क्रिप्ट जानने की जरूरत नहीं है।

आपके विचार से आपने क्या प्रोग्राम किया:

यदि बाहर ठंड है, तो जैकेट पहनें।

आपके प्रोग्राम के रूप में **कंप्यूटर क्या पढ़ता है:**

अगर बाहर ठंड है, तो पिज्जा के लिए नाचें।

आपने इस गतिविधि को शुरू करने से पहले [भाग 1](#) के गुम कोड को डिबग पहले ही पूरा कर लिया होगा।

सीखने के लक्ष्य

इस गतिविधि को पूरा कर लेने पर आप निम्नलिखित कर सकेंगे...

- ❑ लॉजिक की त्रुटियों और सिंटेक्स की त्रुटियों के बीच अंतर का वर्णन करना।
- ❑ कोड को डिबग करने के लिए विविध रणनीतियों का वर्णन करना।
- ❑ डिबग करने की गतिविधियों को किसी टूटी हुई वेबसाइट में लॉजिक की त्रुटि को फिक्स करने के लिए लागू करना।

सामग्रियां

- [Repl.it](#) एडिटर
- [बग वाला व्यक्तित्व प्रश्नोत्तरी नमूना प्रोजेक्ट](#)
- [बग वाला व्यक्तित्व प्रश्नोत्तरी - टूटा हुआ प्रोजेक्ट](#) (केवल लॉजिक की त्रुटियों के साथ)
- [मिसिंग कोड संदर्भ मार्गदर्शिका](#)

पूर्व ज्ञान

इस प्रोजेक्ट से शुरुआत करने से पहले, हमारी सलाह है कि आप:

- किसी भी प्रोग्रामिंग भाषा में [चर राशि](#), [कार्यों](#), तथा [कंडीशनल बयान](#) सहित कोर कम्प्यूटेशनल अवधारणाओं से कुछ परिचित रहें।
- शुरुआती स्तर वालों को किसी टेक्स्ट आधारित भाषा, जैसे जावास्क्रिप्ट (JavaScript), Python, स्विफ्ट (Swift) आदि के उपयोग का अनुभव करवाएं।

यदि आप जावास्क्रिप्ट में प्रोग्रामिंग के बारे में अधिक जानना चाहते हैं, तो [Girls Who Code at Home](#) में [वर्चुअल हाइक](#) गतिविधि देखें।

यदि आप स्क्रैच में डिबगिंग का अभ्यास करना चाहते हैं, तो बाहर की जाँच करें [स्क्रैच के साथ ब्रेव नॉट परफेक्ट डिबगिंग](#) [Girls Who Code at Home](#) गतिविधि देख सकते हैं।

वुमन इन टेक स्पॉटलाइट: सिमोन गीर्टज़



स्रोत: [Mashable](#)

सिमोन गीर्टज़ के आविष्कारों में ऐसे रोबोट शामिल हैं जो बाल धोते हैं, सब्ज़ी काटते हैं, और सुबह के समय लिपस्टिक लगाते हैं। हालांकि, उसके रोबोट दुर्लभ रूप से (नहीं के बराबर) सफल होते हैं और अधिकतर बेकार हैं, और सिमोन इस विफलता को स्वीकार करती है! वह कहती है कि “बेकार चीजें बनाने की असली सुंदरता यह स्वीकृति [है] कि आप हमेशा नहीं जानते हैं कि सबसे अच्छा उत्तर क्या है।”

अपने प्रशंसकों में मिस्ट्रेस ऑफ मालफंक्शन, आविष्कारक और रोबोट की समर्थक सिमोन गीर्टज़ अपना अधिकांश समय अपने यूट्यूब चैनल पर आर्डुइनो के साथ खेलना, ऐडम सैवेज के साथ *Tested* को होस्ट करना, और श्रोताओं के सामने अपने निराले प्रोजेक्ट्स के बारे में बोलने में बिताती है।

सिमोन के निराले आविष्कारों में से एक अप्लॉज़ मशीन, और इस उत्पाद और डिजाइन के पीछे उनकी प्रेरणा के बारे में अधिक जानने के लिए [यह वीडियो](#) को देखें।

2018 में मस्तिष्क के गैर-कैंसर ट्यूमर से निदान होने के बाद, सिमोन ने उपचार और चिंतन के लिए अपने आविष्कारों से छुट्टी ली। अब जब वह उससे उबर रही है, सिमोन का आविष्कार करना जारी है, लेकिन उसने अपना ध्यान बड़े और अधिक उपयोगी उत्पादों पर केंद्रित कर लिया है। उसका सबसे हालिया बड़ा आविष्कार क्या है? एक टेस्ला को पिकअप ट्रक में बदलना। [Truckla](#) से मिलें।

झलक

याद रखें, एक कंप्यूटर वैज्ञानिक होना, कोडिंग में बेहतरीन होने से कहीं अधिक है। चर्चा करें कि सिमोन और उनका कार्य उन शक्तियों से कैसे संबंधित है, जिसके निर्माण के लिए महान कंप्यूटर वैज्ञानिक ध्यान केंद्रित करते हैं - जैसे आप अपने Girls Who Code प्रोग्राम में करती हैं।



प्रयोजन

सिमोन उसके रोबोट या प्रोजेक्ट्स के अपेक्षानुसार काम न करने पर भी खुशनुमा रवैया बनाए रखती है। असफलताओं के सामने आशावादी बना रहना आपकी मदद कैसे कर सकता है?

परिवार के किसी सदस्य या मित्र के साथ अपनी प्रतिक्रियाएँ साझा करें। चर्चा में शामिल होने हेतु दूसरों को सिमोन के बारे में अधिक पढ़ने के लिए प्रोत्साहित करें!

चरण 1: लॉजिक बग्स से मिलें (6-8 मिनट)

लॉजिक क्या है? (1 मिनट)

आप जैसा सोचते हैं, उसके विपरीत, कंप्यूटर स्मार्ट नहीं होते हैं। वे ऐसी मशीनें हैं जो (कम से कम अभी) नहीं जानती हैं कि कैसे निर्णय लेते हैं और अपने बलबूते पर कोई काम करते हैं। कंप्यूटरों को उन्हें यह बताने के लिए कि क्या करना है - निर्देशों के एक सेट से उन्हें प्रोग्राम करने के लिए - मनुष्यों की जरूरत होती है। इस निर्देश सेट को **अल्गोरिथ्म** कहते हैं। लेकिन प्रोग्राम को कैसे पता चलता है कि कौन से निर्देश का पालन करना है या कब पालन करना है? यह वह जगह है जहाँ **लॉजिक** की जरूरत पड़ती है। हम अपने प्रोग्राम को अपने कामों को पूरा करने के लिए उपयोग किए जाने वाला क्रम या **अनुक्रम** बताने के लिए लॉजिक का उपयोग करते हैं।

मैं लॉजिक का उपयोग कैसे करूँ? (3-4 मिनट)

यहाँ तीन आम संरचनाएं दी गई हैं जिनका उपयोग प्रोग्रामर अपने प्रोग्रामों में लॉजिक जोड़ने और अनुक्रम निर्धारित करने के लिए करते हैं। हमने ये उदाहरण जावास्क्रिप्ट में लिखे हैं, लेकिन इन संरचनाओं का उपयोग अधिकांश अन्य प्रोग्रामिंग भाषाओं पर लागू होता है।

कंडीशनल वक्तव्य

ये वक्तव्य हमारे प्रोग्राम को बताते हैं कि [if](#), [else if](#), [else](#) और तुलना या लॉजिकल [ऑपरेटरों](#) का उपयोग करके यह मूल्यांकन करके निर्णय कैसे लेते हैं कि कोई वक्तव्य सही है या गलत।

```
if(numPlayers >= 5){
    playAmongUs();
} else {
    playScattergories();
}
```

यह कंडीशनल इस बारे में फैसला करता है कि खिलाड़ियों की संख्या के आधार पर कौन सा गेम खेलना है। यदि खिलाड़ियों की संख्या 5 या उससे अधिक है, तो *Among Us* खेलें, अन्यथा *Scattergories* खेलें।

फंक्शन

जब भी आपके पास ऐसे कोड के टुकड़े हों जिन्हें आप फिर से इस्तेमाल करना या अपने कोड को पठनीय रखना चाहते हैं तब एक नया फंक्शन बनाएं। कुछ फंक्शन विशिष्ट मान लौटाते हैं जबकि अन्य नहीं।

```
if(score == 10){
    updateWinner();
    restartGame();
}

function updateWinner(){
    console.log("Winner!");
}

function restartGame{
    score = 0;
}
```

यह if वक्तव्य प्रोग्राम से स्कोर के 10 होने पर विजेता अपडेट करने और गेम को फिर से शुरू करने के लिए कहता है। फंक्शन कोड को अधिक पठनीय बनाते हैं क्योंकि हम उपचरणों को वर्णनात्मक नामों वाले फंक्शनों में समूहबद्ध करते हैं।

लूप

किसी शर्त के पूरा होने तक निर्देशों की एक शृंखला को दोहराने के लिए लूप। लूपों के मुख्य प्रकार हैं [for](#) लूप और [while](#) लूप।

```
while(winner == false){
    keepPlaying();
}
```

यह while लूप प्रोग्राम से तब तक keepPlaying फंक्शन चलाने के लिए कहता है जब तक कोई जीत नहीं जाता।

ध्यान दें: इस गतिविधि में लूपों के साथ काम नहीं करने वाले हैं, लेकिन उनके बारे में जानना जरूरी है!

बग्स कहाँ रहते हैं? (1 मिनट)

बग्स आपके प्रोग्राम के किसी भी बिंदु पर हो सकते हैं। आप उन्हें अपनी [चर राशियों](#), [फंक्शनों](#), [कंडीशनल्स](#), इत्यादि में छिपा हुआ पा सकते हैं।

कोड

```
if (temp <= 65){
    wearJacket();
} else if (temp >= 66){
    removeJacket();
}
```

```
function wearJacket(){
    जैकेट को अलमारी से निकालें
    अनज़िप करें
    एक हाथ अंदर डालें
    दूसरा हाथ डालें
    जैकेट को ज़िप लगाएं
}
```

```
function removeJacket(){
    जैकेट अनज़िप करें
    एक हाथ बाहर निकालें
    दूसरा हाथ बाहर निकालें
    अलमारी में लटकाएं
}
```

अनुक्रम

यदि तापमान **47 डिग्री** फारेनहीट है तो इस कोड का अनुक्रम:

परीक्षण की शर्त 1: temp <= 65?

- शर्त 1 सही के रूप में मूल्यांकन करती है
- wearJacket फंक्शन पर जाएं
- wearJacket फंक्शन में कोड की सभी पंक्तियाँ चलाएं
- बाहर निकलें

यदि तापमान **84 डिग्री** फारेनहाइट है तो इस कोड का अनुक्रम:

परीक्षण की शर्त 1: temp <= 65?

- शर्त 1 गलत के रूप में मूल्यांकन करती है

परीक्षण की शर्त 2: temp >= 66?

- शर्त 2 सही के रूप में मूल्यांकन करती है
- removeJacket फंक्शन पर जाएं
- removeJacket फंक्शन में कोड की सभी पंक्तियाँ चलाएं
- बाहर निकलें

हालांकि दोनों बार का कोड एक समान है, निर्देशों का क्रम अलग है। यहाँ कार्यों का अनुक्रम temp के मान पर निर्भर करता है।

लॉजिक त्रुटियों से समस्याएं क्यों होती हैं? (2-3 मिनट)

इन त्रुटियों को हल करना सिंटेक्स त्रुटियों से थोड़ा अधिक कठिन है। जब हमने अपनी सिंटेक्स त्रुटियों को डीबग किया, हमने अपने बग्स को पहचानने और परिवर्तन करने के लिए त्रुटि कंसोल का उपयोग किया। यह सिंटेक्स त्रुटियों के लिए अत्यंत उपयोगी रणनीति है, लेकिन आप कंसोल पर भरोसा नहीं कर सकते कि वह लॉजिक की त्रुटि होने पर आपको बताएगा। इस मामले में, आपका कोड चल सकता है, लेकिन वह वैसा नहीं चलेगा जैसे आप चाहते हैं। यदि आपका कंपाइल करते समय (यानी, आपके कोड को उस प्रारूप में बदलता है जिसे कंप्यूटर निष्पादित कर सकता है) त्रुटि की सूचना नहीं देता है, तो कंप्यूटर के अनुसार कोई त्रुटि नहीं हुई है। लॉजिक बग **प्रोग्राम के प्रवाह** - वह अनुक्रम या क्रम जिसमें आपकी कोड की पंक्तियाँ निष्पादित होती हैं - में समस्या पैदा करते हैं।

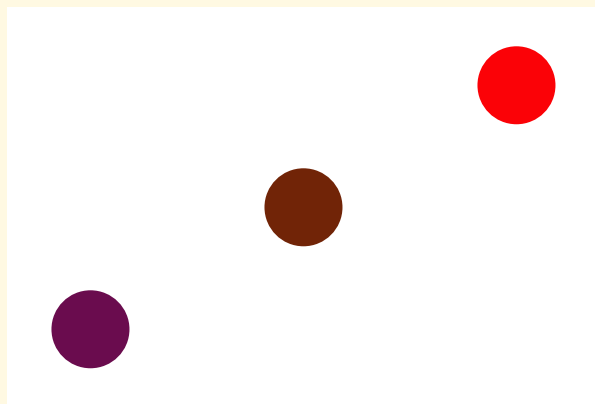
चरण 1: लॉजिक बगों से मिलें (जारी)

मान लें कि आप चमकीले लाल, हरे, और बैंगनी रंग के वृत्तों को पेंट करना चाहते हैं। अपनी पेंटिंग बनाने के लिए इस उदाहरण निर्देश सेट की जाँच करें:

कोड

ब्रश को लाल पेंट में डुबाएं
ऊपरी दायें ओर के वृत्त को पेंट करें
अपने ब्रश को हरे पेंट में डुबाएं
बीच में स्थित वृत्त को पेंट करें
अपने ब्रश को बैंगनी पेंट में डुबाएं
नीचे बायें ओर के वृत्त को पेंट करें
ब्रश को धोएं
कैनवस को सूखने दें

परिणाम



हमारे वर्तमान प्रोग्राम में, अंतिम वृत्त के मटमैले बैंगनी-भूरे रंग का होने की संभावना है। आपके विचार से वैसा क्यों होता है? आप इस “प्रोग्राम” को कैसे फिर से लिखेंगे कि जिससे अंतिम वृत्त बैंगनी हो?



संदर्भ मार्गदर्शिका के पेज 2 पर अपने विचारों की जाँच करें।

चरण 2: अपनी डीबगिंग रणनीतियों का निर्माण करें (5-10 मिनट)

भाग 1 में, हमने कुछ डीबगिंग तकनीकें सीखीं, जैसे त्रुटि संदेशों को पढ़ने के लिए कंसोल का उपयोग करना, प्रेक्षण करना, और अपने द्वारा किए गए परिवर्तन की जाँच करना। जब आप अधिक पेचीदा कोड में जाते हैं, तो एक ऐसे टूलबॉक्स का होना उपयोगी होता है, जो डीबगिंग रणनीतियों और उन्हें कार्यान्वयित करने की प्रणाली से भरा हो।

डीबगिंग में कोड की जाँच करने, समस्या को परिभाषित करने, समाधानों की अवधारणा बनाने, परिवर्तन करने, और जब तक कोड समझ न आए तब तक उसकी परीक्षा करने का एक चक्र शामिल है। सभी बग अलग होते हैं, लेकिन ऐसी आम रणनीतियाँ हैं जिनका आप समाधान खोजने में आपकी मदद करने के लिए उपयोग कर सकते हैं। अगले पेज पर रणनीतियों की एक सूची है जिसे आप अपने कोड में बग के आने पर उपयोग कर सकते हैं।



चरण 2: अपनी डीबगिंग रणनीतियों का निर्माण करें (जारी)

1. अपने आपको डीबगिंग के लिए सही मानसिकता में रखें

संभव है कि बहुत सारा समय बिताने और अपने कोड में बहुत सारी ऊर्जा का निवेश करने के बाद आपको केवल एक बग मिले। आपको इसमें डुबकी लगाने से पहले अपने कुत्ते को सहलाने, नाश्ता करने, चहलकदमी करने, या इसे टालने की जरूरत पड़ सकती है। डीबगिंग धीरे-धीरे और व्यवस्थित तरीके से करनी चाहिए। यदि आप जल्दी करते हैं, तो संभव है कि आप किसी चीज को चूक जाएं या नए बग बना डालें।

2. बुनियादी बातों में वापस जाएं

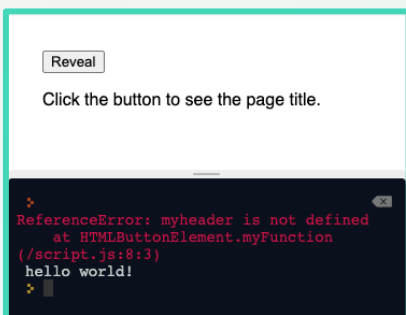
क्या आप इंटरनेट से कनेक्टेड हैं? क्या आपका कोड एडिटर या IDE (यानी इंटीग्रेटेड डेवलपर वातावरण) ठीक से काम कर रहा है? क्या आपके ब्राउज़र को अपडेट के बाद रीस्टार्ट होने की जरूरत है? जब तक कि सबकुछ फिक्स न हो जाए...यह मूर्खतापूर्ण लग सकता है।

3. एक कॉपी बनाएं

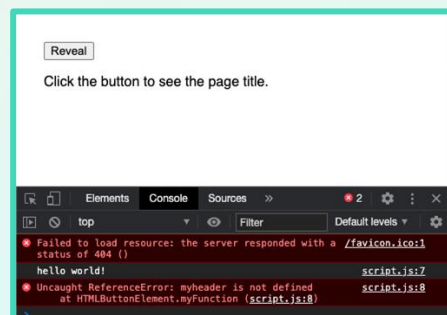
ऐसी समस्याओं को फिक्स करने की कोशिश करना जिनसे और समस्याएं पैदा हो सकती हैं। यदि आपको कोई बड़े परिवर्तन करने हैं तो हमेशा एक कॉपी बनाएं। फूंक-फूंक कर चलने में ही समझदारी है!

4. सरल ढंग से शुरू करें

सरल चीजों की पहले जाँच करें। यह उन्मूलन की प्रक्रिया है, इसलिए आसान वालों को पहले रास्ते से हटाएं। जमीनी स्तर की त्रुटियों की खोज करें और सिंटेक्स की गलतियों की जाँच करें। यदि आपका IDE डीबगिंग कंसोल है, तो त्रुटि संदेशों के लिए उसकी जाँच करें। यदि आप वेब के लिए प्रोग्रामिंग कर रहे हैं, तो आप ब्राउज़र में [डेवलपर टूल्स](#) का उपयोग भी कर सकते हैं। नीचे इस बात का एक उदाहरण दिया गया है कि बायीं ओर Repl.it कंसोल में और दायीं ओर क्रोम में डेवलपर टूल्स का उपयोग करने पर एक ही त्रुटि किस तरह से दिखाई देती है।



डिस्प्ले विंडो के नीचे मौजूद ब्लैक बॉक्स Repl.it कंसोल है। वह लाल टेक्स्ट में त्रुटि के स्थान के साथ एक त्रुटि संदेश दिखाता है, फिर स्क्रीन पर **hello world!** प्रिंट करता है।



डेवलपर टूल स्क्रीन के दायीं ओर या नीचे दिखाई देता है। **कंसोल** टैब स्क्रीन पर **hello world!** प्रिंट करता है, फिर बायीं ओर लाल टेक्स्ट में एक त्रुटि संदेश और दायीं ओर सफेद टेक्स्ट में स्थान दिखाता है।

5. पैटर्नों के लिए खोजें

अन्य मानवीय भाषाओं की तरह ही, प्रोग्रामिंग कोड को संगठित करने और उसे समझने के लिए पैटर्नों पर निर्भर करती है। हम उन पैटर्नों का उपयोग कर सकते हैं जिन्हें हमने पिछले कोड में देखा है और उन्हें वर्तमान चुनौती पर लागू कर सकते हैं। उदाहरण के लिए, मान लें कि आप एक बटन को प्रोग्राम कर रहे हैं जिससे वह हर बार क्लिक किए जाने पर स्कोर को कम करता है। आपने दो बटनों को स्कोर बढ़ाने के लिए पहले ही कोड कर लिया है, लेकिन आप नहीं जानते कि दूसरे बटन को कैसे प्रोग्राम करें। आप समाधान खोजने में मदद पाने के लिए उनके व्यवहार की समानताओं या पैटर्नों का प्रेक्षण कर सकते हैं।

1 से बढ़ाने के लिए मुझे क्लिक करें 1!

```
if(clicked == true){
  score = score + 1;
}
```

2 से बढ़ाने के लिए मुझे क्लिक करें 1!

```
if(clicked == true){
  score = score + 2;
}
```

1 से घटाने के लिए मुझे क्लिक करें 1!



6. समस्या का वर्णन करें

कभी-कभी जब हम अपने बलबूते पर किसी समस्या की बहुत देर तक तलाश करते हैं, तो हम सरल चीजों को छोड़ने लग सकते हैं। लेकिन यदि आपको किसी और को समस्या का वर्णन करना है, तो आपको अधिक विस्तृत होना पड़ेगा। दिखावा करें कि आप किसी मित्र को अपनी समस्या बता रहे या दिखा रहे हैं और वर्णन करें:

- ❑ आपने पहले से क्या कर लिया है।
- ❑ क्या गलत हो रहा है।
- ❑ आपने क्या अपेक्षा की थी या क्या करवाना चाहते हैं।
- ❑ पिछली बार इसकी परीक्षा करने के बाद आपने अपने कोड में क्या जोड़ा या बदला है।

अपने प्लानिंग नोट्स, सूडोकोड, और/या रेखा-चित्रों को भी फिर से देखें। संभव है कि आप खुद ही गलती को पहचान लेंगे।

7. इसे प्रेक्षण-योग्य बनाएं

यह देखना बहुत उपयोगी होता है कि समय के साथ चर राशियों के मान कैसे बदलते हैं या विभिन्न घटनाएं कब होती हैं, जैसे बटन का क्लिक या टेक्स्ट इनपुट। आप अपने प्रोग्राम के प्रवाह के बारे में फैसले करने के लिए इन चर राशियों का अक्सर उपयोग करेंगे। उदाहरण के लिए:

```
if(zoomCalls >= 5){  
    takeWalk();  
}
```

लेकिन आप कैसे जान सकते हैं कि किसी चर राशि का क्या मान है? मुझे `zoomCalls` की संख्या का पता कैसे चलेगा? आप `console.log(variableName)` फंक्शन का उपयोग IDE की कंसोल विंडो में चर राशि का मान प्रिंट करने के लिए कर सकते हैं या उसे [developer tools](#) का उपयोग करके देख सकते हैं।

कोड

```
console.log("zoomCalls = " + zoomCalls);  
if(zoomCalls >= 5){  
    takeWalk();  
}
```

कंसोल

```
zoomCalls = 3  
zoomCalls = 4
```

ध्यान दें: यदि आप जानना चाहते हैं कि आप कौन सा मान प्रिंट कर रहे हैं, तो आप टेक्स्ट के साथ अपने मानों में संदर्भ जोड़ सकते हैं। जिस टेक्स्ट को आप प्रिंट करना चाहते हैं उसे `" "` या एकल `' '` कोटेशन चिह्नों में रखें जिसके बाद प्लस संकेत `+` लगाएं और चर राशि का नाम लिखें। यह खास तौर पर उपयोगी होता है यदि आप एक ही बार में एक से अधिक को प्रिंट करना चाहते हैं!

8. एक बार में एक चीज की जाँच करें

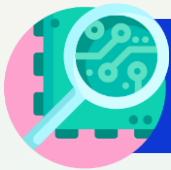
यहाँ एक परिदृश्य है जिसे हम निश्चित रूप से नहीं चाहते: अपने कोड में कुछ “छोटी-छोटी चीजों” को बदल कर, आप एक नई समस्या पैदा करते हैं और अब आप पता नहीं लगा सकते कि वह पुरानी समस्या है या वह नए परिवर्तनों के कारण हो रही है। आपका लक्ष्य समस्या को पहचानना है। एक बार में एक चीज को बदलें, फिर उसकी जाँच करें।

9. इंटरवेब खोजें

ऐसी ढेर सारी वेबसाइटें हैं जो कोड की त्रुटियों के बारे में प्रश्नों का उत्तर देने के लिए पूरी तरह से समर्पित हैं। इस बात की काफी अच्छी संभावना है कि किसी और के सामने भी आपकी वाली समस्या आई होगी। अपनी समस्या की विस्तृत वर्ण करने के बाद, अपने पसंदीदा सर्च इंजन का उपयोग करके कुछ उत्तर खोजें। यदि कुछ भी सामने नहीं आता है, तो अपने प्रश्न को अलग ढंग से या अलग शब्दों में रखने की कोशिश करें - किसी समाधान की कुशलतापूर्वक खोज करने में सक्षम होना भी एक कौशल है! आप किसी भी त्रुटि संदेश को कॉपी भी कर सकते हैं और वहाँ से अपनी खोज शुरू कर सकते हैं। यदि आप वेब के लिए प्रोग्रामिंग कर रहे हैं, तो [W3Schools](#) और [MDN](#) शानदार संसाधन हैं। [StackOverflow](#) सभी भाषाओं में काम करने वाले डेवलपर्स के लिए प्रश्न पूछने और अन्य लोगों के समाधानों से सीखने की एक अच्छी साइट है। आप चाहे जिस भाषा में प्रोग्राम कर रहे हों, संभव है कि उसे सीखने के लिए समर्पित कोई फोरम और वेबसाइटें या पोस्ट होंगी।

10. किसी मित्र से पूछें

यदि आप उपरोक्त सब चीजों को आजमाने के बाद भी उसे समझ नहीं पा रहे हैं, तो किसी मित्र, मेंटर, या शिक्षक को फोन करें।



अधिक डीबगिंग संसाधनों के लिए, NYU में क्ले शिर्की की [यह वीडियो सिरीज](#) और p5.js समुदाय की यह शानदार [Field Guide to Debugging](#) देखें।

चरण 3: प्रश्नोत्तरी की जाँच करें (3-4 मिनट)

अब जबकि हमारे पास डीबगिंग रणनीतियों की उपयोग में आसान टूलकिट है, अब हमारे अंतिम बग - लॉजिक बग को हल करने का समय है। सबसे पहले, इससे पहले कि हम उसे हल करने के बारे में सोचना शुरू करें हमें पता लगाना है कि त्रुटि क्या है और कहाँ पर है।

याद रखें: अधिकांश समय पर, प्रोग्राम गलत नहीं होता है क्योंकि वह आपके निर्देशों का पालन करता है। इसका मतलब है कि लॉजिक बग वास्तव में हमारे प्रोग्राम को कैसे काम करना चाहिए इस बारे में हमारी अपनी गलत धारणाओं बनाम वह वास्तव में कैसे काम करता है, इसका परिणाम होते हैं। लॉजिक त्रुटियों का संबंध हमारे प्रोग्राम के निष्पादन करने के तरीके से होता है। इससे पहले कि हम शुरू करें, चलिए किसी भी सुराग के लिए प्रश्नोत्तरी की जाँच करते हैं।

प्रश्नोत्तरी लें (2-3 मिनट)

जब हमने इस गतिविधि के आरंभ में बगी संस्करण की परीक्षा की थी, तब प्रश्नोत्तरी ने कभी भी हमें परिणाम नहीं दिया था। अब जबकि हमने अपने सिटेक्स बग फिक्स कर लिए हैं, यह सुनिश्चित करने के लिए कि कोई और चीज नहीं बदली है, चलिए इसकी फिर से परीक्षा करते हैं।

- ☐ प्रश्नोत्तरी 3-4 बार लें।
- ☐ प्रश्नों का उत्तर अलग-अलग कन्फिगरेशनों में दें।
- ☐ प्रश्नोत्तरी को रीसेट करने का काम पूरा करने के बाद रीस्टार्ट बटन का उपयोग करें।

चरण 3: प्रश्नोत्तरी की जाँच करें (जारी)

और लीजिए - अब भी कोई परिणाम नहीं है! और कोई त्रुटि संदेश भी नहीं है। इस बिंदु पर, सबसे सही चीज है पीछे जाना, शाब्दिक रूप से और लाक्षणिक रूप से। यदि आप एक क्विक ब्रेक और स्नैक लेना चाहते हैं, तो अब उसका समय है। अगले खंड में, हम यह समीक्षा करके कि कोड कैसे काम करता है, सुरागों की खोज शुरू करेंगे।

You are a...

Restart

तल में परिणाम टेक्स्ट का स्क्रीनशॉट जो कहता है "You are a..." और उसके नीचे एक रीस्टार्ट बटन।

चरण 4: कोड की समीक्षा करें (3-7 मिनट)

जब आप पहले बग्स को फिक्स कर रहे थे तब आपने शायद कोड और टिप्पणियों को पढ़ा होगा, लेकिन यदि आपने ऐसा नहीं किया था तो वापस जाएं और अभी यह काम करें। यदि आपको जो कुछ चल रहा है वह पूरी तरह से समझ में नहीं आता है तो कोई बात नहीं। हमारा लक्ष्य प्रोग्राम के प्रवाह को नियंत्रित करना है, इसलिए हमें इस बिंदु पर उच्च स्तर की समझ चाहिए।



पेज 2 पर संदर्भ मार्गदर्शिका में अपने विचारों की जाँच करें।

चरण 5: समस्या का वर्णन करें (5-7 मिनट)

चलिए यह वर्णन करके शुरू करते हैं कि हम क्या करवाना चाहते हैं। पेन और कागज लें या टेक्स्ट एडिटर खोलें और लिखें कि जब कोई व्यक्ति प्रश्नोत्तरी ले तब प्रोग्राम को क्या करना चाहिए। हमने पहला चरण नीचे शामिल किया है।

- जब कोई व्यक्ति किसी उत्तर बटन पर क्लिक करता है, तो प्रोग्राम उस विशिष्ट बग फंक्शन को कॉल करता है (यानी **bee**, **butterfly**, **grasshopper** या **ladybug**) जो इवेंट लिसेनर में उससे संलग्न होता है।



संदर्भ मार्गदर्शिका के पेज 4 पर अपने विचारों की जाँच करें।

इसे लिखने के बाद, हम देख सकते हैं कि अपने प्रोग्राम के दौरान हमारी स्कोरिंग चर राशियों के मान को जानना कितना महत्वपूर्ण है। उदाहरण के लिए, यदि **grasshopperScore** 4 से अधिक है और कोई परिणाम नहीं मिलता है, वह हमारे लिए एक बड़ा सुराग है!

चरण 6: इसे प्रेक्षण-योग्य बनाएं (10-15 मिनट)

हम अपनी चर राशियों के मान को कैसे एक्सेस करते हैं? बेशक `console.log()` से! हम प्रत्येक बग स्कोर चर राशि और `questionCount` के मान को जानना चाहते हैं जब भी उन्हें अपडेट किया जाता है। यह अंतिम भाग महत्वपूर्ण है क्योंकि इससे हमें यह जानने में मदद मिलती है कि `console.log()` फंक्शन को कहाँ रखना है। हम इन मानों को किसी व्यक्ति के द्वारा बटन पर क्लिक करने के बाद जानना चाहते हैं, इसलिए हम `console.log()` को प्रत्येक बग स्कोर फंक्शन के भीतर रखेंगे।

- ❑ नीचे फंक्शनों के भीतर स्कोर चर राशि और `questionCount` मान प्रिंट करने के लिए `console.log()` का उपयोग करें। स्पष्ट करने वाला टेक्स्ट जोड़ने के लिए सिंगल या डबल कोटेशन का उपयोग करके अपने संदेशों में संदर्भ जोड़ें ताकि आपको पता रहे कि कौन सा मान किस स्कोर के साथ जाता है (जरूरत हो तो उदाहरण के लिए चरण 1 देखें)।
 - ❑ `bee` फंक्शन
 - ❑ `butterfly` फंक्शन
 - ❑ `grasshopper` फंक्शन
 - ❑ `ladybug` फंक्शन

सुझाव: आप दो मानों के बीच स्पेस बनाने के लिए “\t” का उपयोग कर सकते हैं।



संदर्भ मार्गदर्शिका के पेज 4 पर अपने विचारों की जाँच करें।

आपके फंक्शनों को `console.log()` में जोड़ने के बाद, अब समय है आपके कोड की जाँच करने का!

- ❑ **चलाएं** बटन पर क्लिक करें।
- ❑ प्रश्न एक के लिए *Mochi* पर क्लिक करें, फिर कंसोल विंडो की जाँच करें।

आखिरकार! हमारे पास अपने प्रोग्राम से कुछ आउटपुट है! कंसोल में निम्नलिखित टेक्स्ट प्रदर्शित होगा:

```
questionCount = 1
```

```
butterflyScore = 1
```

- ❑ अब प्रश्न दो के लिए *शहर* पर क्लिक करें और कंसोल की जाँच करें। निम्नलिखित टेक्स्ट प्रदर्शित होना चाहिए:

```
questionCount = 1
```

```
butterflyScore = 1
```

```
questionCount = 2
```

```
beeScore = 1
```

अभी तक तो सब अच्छा दिख रहा है! प्रत्येक क्लिक के साथ हमारे मान बढ़ रहे हैं, इसलिए हम जानते हैं कि हमारे बग फंक्शन काम कर रहे हैं।

- ❑ तीसरे प्रश्न के लिए, 2018 पर क्लिक करें और कंसोल की जाँच करें। निम्नलिखित टेक्स्ट प्रदर्शित होना चाहिए:

```
questionCount = 1      butterflyScore = 1
questionCount = 2      beeScore = 1
questionCount = 3      butterflyScore = 2
```

इस बिंदु पर, हमें एक परिणाम प्राप्त हो चुकना चाहिए। `questionCount` 3 के बराबर है और हमारे पास एक स्कोर है जो 2 से अधिक या उसके बराबर है (`butterflyScore`), लेकिन परिणाम टेक्स्ट नहीं बदला है।

चलिए इन सुरागों के बारे में सोचते हैं और खुद से कुछ प्रश्न पूछते हैं:

- ❑ परिणाम टेक्स्ट को क्या नियंत्रित करता है?
- ❑ हम `updateResult` फंक्शन को कब कॉल (या उपयोग) करते हैं?
- ❑ क्या कंडीशनल वक्तव्य तब चलता है जब हमें उसके चलने की जरूरत होती है?



पेज 5 पर संदर्भ मार्गदर्शिका में अपने विचारों की जाँच करें।

चरण 7: पैटर्नों की खोज करें (1-2 मिनट)

हम समस्या को पंक्ति 82 पर कंडीशनल वक्तव्य सीमित करना शुरू कर रहे हैं:

```
81 // प्रश्नोत्तरी की समाप्ति के लिए ट्रैक करें
82 if (questionCount == 3){
83     updateResult();
84 }
```

इस वक्त, यह कंडीशनल सभी बग स्कोर ट्रैकिंग फंक्शनों के बाद `questionCount` मान की जाँच करता है। आपने एक पैटर्न देखा होगा कि प्रत्येक बग फंक्शन के भीतर स्कोर और `questionCount` चर राशियाँ अपडेट होती हैं। शायद हमें प्रत्येक फंक्शन के भीतर `questionCount` के मान की जाँच करने की जरूरत है!

चरण 8: एक बार में एक चीज का परीक्षण करें (5-10 मिनट)

चलिए इस अवधारणा का परीक्षण करते हैं। सभी फंक्शनों को बदलने की बजाय, चलिए पहले `bee` फंक्शन को बदलने की कोशिश करते हैं। यदि वह काम करता है, तो हम शेष फंक्शनों को बदल सकते हैं।

चरण 8: एक बार में एक चीज की जाँच करें (जारी)

- ❑ पूरे `questionCount` कंडीशनल वक्तव्य को कॉपी करें। सभी कर्ली ब्रैकेट लगाना सुनिश्चित करें!

```
if (questionCount == 3){
    updateResult();
}
```

- ❑ इसे `bee` फंक्शन के भीतर पेस्ट करें। उसे `console.log()` फंक्शन के नीचे और बंद करने वाले कर्ली ब्रैकेट के ऊपर होना चाहिए।
- ❑ अपना कोड चलाएं।
- ❑ सभी `bee` प्रतिक्रियाओं का चयन करके अपनी अवधारणा की जाँच करें। प्रत्येक प्रश्न के लिए `bee` प्रतिक्रियाओं के बटन पर क्लिक करें:
 - ❑ Q1: फल
 - ❑ Q2: शहर
 - ❑ Q3: 2019
- ❑ **परिणाम टेक्स्ट की जाँच करें।** क्या परिणाम दिखाई देते हैं या क्या हमारे पास अब भी बग है? आप देखेंगे कि हमें `bee` परिणाम मिला था जो हमें यह सोचने के लिए प्रेरित करता है कि शायद हमने समस्या की पहचान कर ली है! चलिए जाँच करके देखते हैं कि क्या हमारी अवधारणा अन्य परिणामों के लिए काम करती है।

You are a bee!

Restart

तल पर परिणाम टेक्स्ट का स्क्रीनशॉट जो कहता है "आप एक bee हैं!" और उसके नीचे एक रीस्टार्ट बटन।

हमें `bee` प्रतिक्रियाओं पर हमारी अवधारणा की जाँच करते समय हमारे विकल्पों से मेल खाने वाला एक परिणाम मिला है! लगता है हमने त्रुटि पैदा करने वाली समस्या की पहचान कर ली है।

- ❑ **शेष फंक्शनों के लिए उसी स्थान में कंडीशनल वक्तव्य जोड़ें।** एक बार में एक अवधारणा की जाँच करना याद रखें। निम्नलिखित फंक्शनों की जाँच करने के लिए एक जैसे चरणों का अनुसरण करें:
 - ❑ `butterfly` फंक्शन
 - ❑ `grasshopper` फंक्शन
 - ❑ `ladybug` फंक्शन

प्रोग्राम की फिर से जाँच करें। प्रश्नों का उत्तर विभिन्न कन्फिगरेशनों में करें और देखें कि क्या परिणाम मिलता है। जाँच करने में आपकी मदद करने के लिए अगले पेज पर एक स्कोरिंग कुंजी है।



पेज 6 पर संदर्भ मार्गदर्शिका में अपने विचारों की जाँच करें।

चरण 8: एक बार में एक चीज की जाँच करें (जारी)

बग का परिणाम	bee	butterfly	grasshopper	ladybug
Q1: Dessert	फल q1a4	Mochi q1a1	कुकीज़ q1a2	केक q1a3
Q2: छुट्टी	शहर q2a2	समुद्रतट q2a3	जंगल q2a1	पर्वत q2a4
Q3: रंग	2019 q3a4	2018 q3a2	2020 q3a3	2017 q3a2



बधाई!

आपने बग वाले व्यक्तित्व प्रश्नोत्तरी को सफलतापूर्वक डीबग कर दिया! अब जब हमारे पास एक पूरी तरह से कार्यशाला प्रश्नोत्तरी है, आप अधिक जानने के लिए एक्सटेंशनों में जा सकते हैं।

सुझाव: भविष्य में हमेशा के लिए लॉजिक त्रुटियों से बचने की कोशिश करना चाहते हैं? मनुष्य के लिए पठनीय भाषा (यानी सूडोकोड) में अपने कोड की योजना, रेखा-चित्र बनाने और लिखने में समय लगाएं।

चरण 9: एक्सटेंशन (5-75 मिनट)

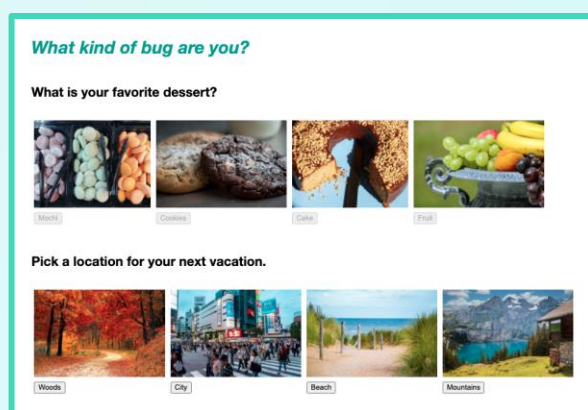
एक्सटेंशन 1: अपनी व्यक्तिगत डीबगिंग जाँचसूची लिखें (5-10 मिनट)

आपके लिए उपयोगी तरीके की खोज करने के लिए उपरोक्त रणनीतियों को आवश्यकतानुसार बदलें! जब आप पहली बार प्रोग्रामिंग करना और विभिन्न त्रुटियों का सामना करना शुरू करते हैं, तो यह याद रखना आसान नहीं होता कि कहाँ से शुरू करें। एक पेन और कागज लें या अपना पसंदीदा टेक्स्ट एडिटर खोलें और उन रणनीतियों की एक सूची बनाएं जिन्हें आप याद रखना चाहते हैं। उसे किसी ऐसी जगह रखें जहाँ से आप आसानी से एक्सेस कर सकते हैं ताकि जब आपका सामना अगले बग से हो पता हो कि कहाँ जाना चाहिए!

एक्सटेंशन 2: बटनों को अक्षम करें (15-20 मिनट)

इस वक्त, प्रयोक्ता किसी दिए गए प्रश्न के लिए एक से अधिक उत्तर विकल्प पर क्लिक कर सकता है और प्रश्नोत्तरी के परिणामों को मटियामेट कर सकता है। विचार करें कि उनमें से एक पर क्लिक करके आप कैसे उत्तर विकल्प बटनों को अक्षम कर सकते हैं।

आप [इस एक्सटेंशन के एक उदाहरण](#) को यहाँ देख सकते हैं।



शुरू करने के लिए आप:

- ❑ ऐसे फंक्शन बना सकते हैं जो किसी दिए गए प्रश्न के लिए उत्तर बटनों को अक्षम करते हैं।
- ❑ फिर आप प्रत्येक उत्तर विकल्प बटन के लिए एक और इवेंट लिसेनर जोड़ सकते हैं जो जब भी किसी उत्तर विकल्प बटन को क्लिक किया जाता है तो आपके अक्षम करें फंक्शन को कॉल करता है।
- ❑ अंत में, यदि आपने गेम को रीस्टार्ट करने के लिए कोई एक्सटेंशन जोड़ा था, तो आप यह भी सुनिश्चित करना चाहेंगे कि आप अपने रीस्टार्ट फंक्शन के हिस्से के रूप में अपने सभी बटनों को सक्षम कर लें।

एक्सटेंशन संसाधन

- [HTML DOM बटन डिसएबल प्रॉपर्टी](#)
- [इवेंट लिसेनर](#)
- [जावास्क्रिप्ट फंक्शन](#)
- [जावास्क्रिप्ट HTML DOM](#)

एक्सटेंशन 3: व्यक्तित्व प्रश्नोत्तरी को आवश्यकतानुसार बदलें (25-45 मिनट)

अपना खुद की व्यक्तित्व प्रश्नोत्तरी बनाएं! यह काफी दुर्लभ बात है कि प्रोग्रामर स्कैच से कोई चीज पूरी तरह से बनाते हैं। वास्तव में, ऐसा कोड लिखना बेहतर है जिसे आप दोबारा इस्तेमाल कर सकते हैं। अपनी प्रश्नोत्तरी को आवश्यकतानुसार बदलने के लिए इस कोड का दोबारा इस्तेमाल करने की कोशिश करें। आपको इस एक्सटेंशन में **index.html** फाइल को अपडेट करना होगा, ताकि यदि आपको HTML की थोड़ी सी जानकारी है तो वह उपयोगी हो। यदि ऐसा नहीं है, तो कोई बात नहीं - सहायता के लिए एक्सटेंशन संसाधन देखें। नीचे कुछ कदम दिए गए हैं जिन्हें आप शुरुआत करने के लिए उठा सकते हैं।

योजना

इस अनुभाग में प्रश्नों का उत्तर देते समय इस योजना मार्गदर्शिका को भरें:

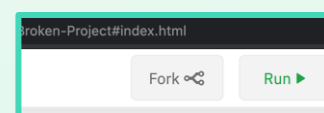
प्रश्न 1				
उत्तर	1.	2.	3.	4.
परिणाम स्कोर करना	+1	+1	+1	+1
प्रश्न 2				
उत्तर	1.	2.	3.	4.
परिणाम स्कोर करना	+1	+1	+1	+1
प्रश्न 3				
उत्तर	1.	2.	3.	4.
परिणाम स्कोर करना	+1	+1	+1	+1

शुरू करने के लिए आप:

- ❑ चार संभव अंतिम परिणाम चुन सकते हैं। इस प्रश्नोत्तरी के लिए केवल चार संभव परिणाम चुनें जो सबसे अंत में आपके प्रयोक्ता के पास होंगे। बाद में, आपको प्रत्येक प्रश्न के उत्तर को संभव परिणामों में से एक के साथ मैप करने की जरूरत पड़ेगी।
 - ❑ उदाहरण के लिए: Bee, Butterfly, Grasshopper, और Ladybug
- ❑ पूछने के लिए तीन प्रश्न चुनें। मैप करें कि अपनी प्रश्नोत्तरी में कौन से प्रश्न पूछेंगे। प्रत्येक प्रश्न को एक दूसरे से संबंधित होना चाहिए और उसके चार संभव उत्तर होने चाहिए।
 - ❑ उदाहरण के लिए: आपकी पसंदीदा मिठाई क्या है?
- ❑ प्रत्येक प्रश्न के लिए चार संभव उत्तर चुनें। प्रत्येक उत्तर को प्रश्नोत्तरी के अंतिम परिणामों में से एक के साथ सीधे मैप होना चाहिए।
 - ❑ उदाहरण के लिए: प्रश्न 1 के लिए उत्तर 1 Mochi है। Mochi, butterfly के साथ मैप होता है, इसलिए Mochi को चुनने से, **butterflyScore** में 1 की वृद्धि हो सकती है।
- ❑ वह फोटो चुनें जिससे आप प्रत्येक उत्तर का प्रतिनिधित्व करवाना चाहते हैं। उसे स्थानीय रूप से अपने कंप्यूटर पर सहेजें। यदि इसका नाम लंबा है, तो आप इसे कोई छोटा और अधिक वर्णनात्मक **q1a3-cake.jpg** जैसा नाम दे सकते हैं।
 - ❑ याद रखें कि काम करते समय निःशुल्क स्टॉक छवियाँ या [Creative Commons](#) छवियाँ बहुत उपयोगी होती हैं। [Unsplash](#), [Pixabay](#), और [Burst](#) साइट्स पर उपलब्ध विकल्पों को एक नज़र डालें।

index.html फाइल में टेक्स्ट और छवियों को अपडेट करें

- ❑ अपने वर्किंग Repl.it को फोर्क करें। इसे नया नाम और वर्णन दें।
- ❑ बायीं ओर फाइल्स पेन में **आस्तियाँ** फोल्डर में अपनी छवियाँ अपलोड करें।
- ❑ **<h1>** टैग के भीतर टेक्स्ट को बदलकर अपनी प्रश्नोत्तरी के शीर्षक को अपडेट करें।
- ❑ उपरोक्त योजना मार्गदर्शिका का उपयोग करके प्रश्न 1 और उसके उत्तरों के लिए टेक्स्ट और छवियों को अपलोड करें।
 - ❑ **<h2>** टैग के पहले सेट के भीतर वर्तमान प्रश्न को अपने पहले प्रश्न से प्रतिस्थापित करें।
 - ❑ पहले प्रश्न के लिए छवि को अपडेट करें। **<div class="answer-choice">** के ठीक नीचे एक **** टैग है जो वह छवि प्रदर्शित करता है जिसे आप प्रश्न 1 के अंतिम उत्तर के लिए उपयोग करना चाहते हैं। मौजूदा लिंक को आप जिस छवि का उपयोग करना चाहते हैं उसके लिंक से प्रतिस्थापित करें। यह लिंक **assets/imageName.jpg** होगा।
 - ❑ बटन टेक्स्ट को अपडेट करें। **** टैग के नीचे आपको उत्तर टेक्स्ट वाला एक **<button>** टैग दिखना चाहिए। उदाहरण के लिए, **<button id="q1a1">Mochi</button>**। मौजूदा उत्तर को डिलीट करें और अपना खुद का जोड़ें।
 - ❑ प्रश्न 1 के शेष उत्तरों के लिए यही प्रक्रिया दोहराएं।
- ❑ उपरोक्त योजना मार्गदर्शिका का उपयोग करके **index.html** फाइल में शेष प्रश्नों और उत्तरों के लिए टेक्स्ट और छवियाँ अपडेट करें।



script.js फाइल में लॉजिक को अपडेट करें

प्रश्नोत्तरी में लॉजिक पूर्व-स्थापित है जो इस आधार पर किसी व्यक्ति के परिणाम को निर्धारित करता है कि वह प्रत्येक उत्तर का कैसे उत्तर देता है। जैसा कि आपने कोड की समीक्षा करते समय देखा, प्रत्येक उत्तर एक इवेंट लिसेनर का उपयोग करने वाले एक विशिष्ट परिणाम के साथ मैप होता है। आपको अपने प्रश्नों, उत्तरों, और परिणामों के आधार पर इस लॉजिक को अपडेट करने की जरूरत पड़ेगी। लॉजिक और इसका निर्माण कैसे किया जाता है, इसकी पूरी समीक्षा के लिए, संदर्भ मार्गदर्शिका का चरण 4 देखें जो पेज 4 पर है।

- ❑ शीर्ष पर `/* Global Variables */` टिप्पणी के नीचे, अपनी खुद की प्रश्नोत्तरी के चार संभव परिणामों के स्कोर चर राशि नामों को अपडेट करें।+
- ❑ अपने प्रोग्राम में प्रत्येक चर राशि के नामों को **हर जगह** अपडेट करें। *यदि आपका प्रोग्राम काम नहीं करता है, तो आपको इस स्थान को सबसे पहले जाँचना चाहिए।*
- ❑ प्रत्येक इवेंट लिसेनर में फंक्शन के नामों को अपडेट करें। नाम को वह परिणाम होना चाहिए जिससे उत्तर मैप होता है। उदाहरण के लिए, यदि `q1a1` का उत्तर आइस स्क्रीम परिणाम से मैप होता है, तो आपको फंक्शन का नाम `iceCream` रखना चाहिए।
- ❑ फंक्शन के वर्तमान नामों को इवेंट लिसेनर्स में आपके द्वारा अभी-अभी बनाए गए नामों से प्रतिस्थापित करें। *सुनिश्चित करें कि वे सही स्कोर चर राशियों से मेल खाते हैं।*

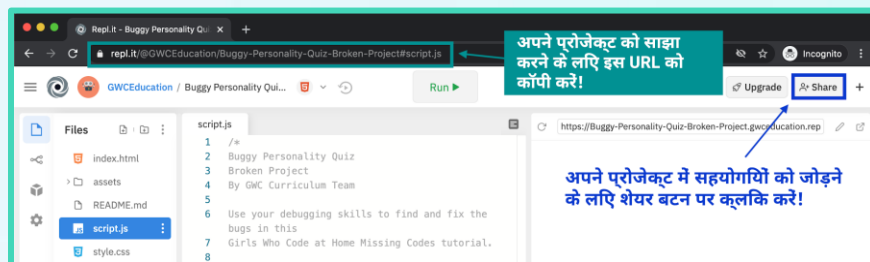
एक्सटेंशन संसाधन

- [HTML का परिचय](#)
- [HTML के व्यावहारिक परिचय के लिए Your Skillz Girls Who Code at Home](#) को साझा करें
- [हेडलाइन टैग्स \(जैसे, <h1>\)](#)
- [टैग](#)
- [<button> टैग](#)
- [इवेंट लिसेनर](#)
- [जावास्क्रिप्ट फंक्शन](#)
- [जावास्क्रिप्ट HTML DOM](#)

चरण 10: अपने Girls Who Code at Home परियोजना को साझा करें! (5 मिनट)

केवल देखने का एक्सेस (1-2 मिनट)

Repl.it पर आपके काम को साझा करना आसान है! बस अपने Repl प्रोजेक्ट के URL पते को शीर्ष पर वेब एड्रेस बार में कॉपी और पेस्ट करें। इससे अन्य लोग आपके प्रोजेक्ट को चलाने, आपका कोड देखने, और आपके प्रोजेक्ट को फोर्क करने तथा अपने खुद के प्रोजेक्ट के साथ रीमिक्स करने में सक्षम होंगे। हम आपके डीबगिंग काम को देखना पसंद करेंगे और हम जानते हैं कि दूसरे भी ऐसा करेंगे। अपने फिक्स्ड कोड को हमारे साथ साझा करें!



इस लिंक को अपने सोशल मीडिया कार्टों पर साझा करें तथा [@girlswhocode](#) [#codefromhome](#) को टैग करना मत भूलें और हो सकता है कि हम आपको अपने खाते में शामिल कर लें!

चरण 10: अपने Girls Who Code at Home प्रोजेक्ट को साझा करें! (जारी)

सहयोगी जोड़ना (2 मिनट)

यदि आप किसी प्रोजेक्ट पर मित्रों के समूह के साथ काम करना चाहते हैं, तो आप विंडो के ऊपरी-दाएं भाग में स्थित **साझा करें** बटन का उपयोग करके उन्हें सहयोग करने के लिए आसानी से आमंत्रित कर सकते हैं। इससे एक नई विंडो खुलेगी जिसमें आपके प्रोजेक्ट पर सहयोग करने के लिए अन्य लोगों को आमंत्रित करने के दो विकल्प होंगे।

- ❑ **ईमेल या Repl.it यूजरनेम के द्वारा आमंत्रित करें।** यह विकल्प आपको विशिष्ट लोगों के साथ अपना प्रोजेक्ट साझा करने के लिए उनका ईमेल पता या यदि उनके पास पहले से ही Repl.it के साथ खाता है तो Repl.it यूजरनेम टाइप करके आमंत्रित करने देता है। हम यह सुनिश्चित करने के लिए कि आप अपने प्रोजेक्ट को सही लोगों के साथ साझा कर रहे हैं, इस विकल्प की अनुशंसा करते हैं!
- ❑ **आमंत्रण लिंक साझा करें।** विंडो के तल पर एक अद्वितीय आमंत्रण लिंक है। आप इस लिंक को मित्रों को कॉपी और पेस्ट कर सकते हैं ताकि वे आपके प्रोजेक्ट को एक्सेस कर सकें।

सहयोगियों के बारे में टिप्पणी: याद रखें कि सहयोगी जोड़ने से अन्य लोगों को आपके प्रोजेक्ट को संपादित करने की सुविधा मिलती है। इससे वे आपके कोड, नाम, और विवरण को बदलने में सक्षम होते हैं। **अपने आमंत्रण लिंक को सोशल मीडिया पर साझा मत करें!** इन संपादन अधिकारों को आप किनके साथ साझा करेंगे इस बारे में चयनशील रहें।



और Girls Who Code at Home गतिविधियों के लिए तैयार रहें!

