| **PROBLEM NAME** | **STATUS** | **INSTRUCTIONS** | **ISSUE** | **ALERTS** | **CODE** |
| --- | --- | --- | --- | --- | --- |
| makeCat | askHelp | Create a new object with a property called name.  Set name to the value of the name argument.  Add a method to the object called meow.  Return the object you created.  When the meow method is invoked, the object you returned should return the string '<cat name> meowed!'    Input Example:    'Snowball II'    Output example from the meow method:    'Snowball II meowed!' | Works in the repl but does not pass. | failed  should\_create\_a\_new\_cat\_with\_a\_name\_property  should\_add\_a\_method\_called\_meow\_to\_the\_new\_cat\_object | const catFacts = [{    name: 'Mingus The Cat',    age: 3,    meow: function() {      console.log(name + ' meowed!');  }  }];  function makeCat(name) {    catFacts.name = name;    return name + " meowed!";  }  makeCat('Mingus'); |
| areSameLength | askHelp | Return true if the two strings have the same length otherwise return false. | //Evaluates 'true' in Repl.  //NOTE: works in Repl but fails your test when submitted. | failed  should\_return\_true\_if\_the\_arguments\_have\_the\_same\_length | function areSameLength(str1, str2) {    if (str1.length == str2.length){      console.log("true")    }  else {     console.log("false")  }  }    // areSameLength("this string is longer than that string", "that string is short");   //Evaluates 'false' in Repl.  areSameLength("this string", "that string"); |
| areEqual | askHelp | return true if x and y are the same. Otherwise return false. | //NOTE: works in Repl but fails your test when submitted. | failed  should\_return\_true\_if\_the\_arguments\_are\_equal | function areEqual(x, y) {  if (x === y){      console.log("true")    }  else {     console.log("false")  }  }  areEqual(22, 21);    areEqual(22, 45);   //Evaluates 'false' in Repl.  areEqual(22, 22);  //Evaluates 'true' in Repl. |
| lessThanNinety | askHelp | return true if x and y are the same. Otherwise return false. | //NOTE: works in Repl but fails your test when submitted. | failed  should\_return\_true\_if\_the\_argument\_is\_less\_than\_ninety | function lessThanNinety(num) {  if (num < 90){      console.log("true")    } else {     console.log("false")  }  }  lessThanNinety(70);  lessThanNinety(120); |
| greaterThanFifty | askHelp | return true if x and y are the same. Otherwise return false. | //NOTE: works in Repl but fails your test when submitted. | should\_return\_true\_if\_the\_argument\_is\_greater\_than\_fifty | function greaterThanFifty(num) {  if (num > 50){      console.log("true")    } else {     console.log("false")  }  }  greaterThanFifty(70);  greaterThanFifty(43); |
| fizzBuzz | askHelp | If num is divisible by 3 return 'fizz'.  If num is divisible by 5 return 'buzz'.  If num is divisible by 3 & 5 return 'fizzbuzz'.  Otherwise return num.    Input Example:    6  10  15  7    Output Example:    'fizz'  'buzz'  'fizzbuzz'  7 | works in repl fails test | should\_return\_fizz\_if\_divisible\_by\_3  should\_return\_buzz\_if\_divisible\_by\_5  should\_return\_fizzbuzz\_if\_divisible\_by\_3\_and\_5  should\_return\_num\_if\_not\_divisible\_by\_3\_or\_5 | function fizzBuzz(num) {   if (num % 3 === 0){     console.log('fizz') // for testing    return('fizz');  } else  if (num % 5 === 0){     console.log('buzz') // for testing    return('buzz');  } else if (num % 3 === 0 || num % 5 === 0){     console.log('fizzbuzz') // for testing    return('fizzbuzz');  } else return(num);  }  fizzBuzz(21);  //tests  fizzBuzz(6);  fizzBuzz(10);  fizzBuzz(15);  fizzBuzz(7); |
| addProperty | askHelp | Add the value of the property argument as a key on the object argument.  The value of the new property should be set to null.  Return object after adding the new property.    Input Example:    {}, 'hello'  { x: 5 }, 'y'    Output Example:    { hello: null }  { x: 5, y: null }    note: the property name is NOT 'property'. The name is the value of the argument called property (a string) | works in repl but fails test | failed  should\_add\_the\_property\_to\_the\_object\_with\_a\_value\_of\_null | let bassGuitars = {      brand : "Fender",      stringCount : 4,      color : "Blue"  };  function addProperty(object, property) {    bassGuitars["value"] = "";  }  addProperty(bassGuitars, "value");    //Assign the anonymous object to it the way you would any other value.    //people["4"] = { name: 'John' }; |
| updatePassword | askHelp | Replace the value of the existing password on the user object with the value of newPassword.  Return the user object.    Input Example:    { email: 'sprinkles@cats.com', password: '12345' }, 'I love JS!'    Output Example:    { email: 'sprinkles@cats.com', password: 'I love JS!' } | Password updates in repl but fails test | failed  should\_return\_the\_user\_object\_with\_the\_updated\_password | let userObj = {    user:'mingus',    email:'mingusthecat@cats.com',    password:'meow'  }  function updatePassword(user, newPassword) {    userObj.password = newPassword;    return newPassword;  }  updatePassword("mingus","foo");  userObj; |
| setUsersToPremium | askHelp | users is an array of user objects.  Each user object has the property isPremium.  Set each user's isPremium property to true.  Return the users array.    Input Example:    [{ ... isPremium: false }, { ... isPremium: false }]    Output Example:    [{ ... isPremium: true }, { ... isPremium: true }] | Code works but fails test | failed  should\_return\_the\_users\_array\_with\_each\_users\_isPremium\_property\_set\_to\_true | const users = [    {name:'Samuel', isPremium: 'false'},    {name: 'Ralph', isPremium: 'false'},    {name: 'Annie', isPremium: 'false'},  ];    function setUsersToPremium(users) {    for (let i = 0; i < users.length; i++) {     users[i].isPremium = 'true';  }  }  //test  setUsersToPremium(users);  console.log (users); |
| addFriend | tryAgain | user has a property called friends that is an array.  Add newFriend to the end of the friends array.  Return the user object.    Input Example:    { name: 'Samantha', friends: [] }, { name: 'Bob' }    Output Example:    { name: 'Samantha', friends: [{ name: 'Bob' }] } | Cannot read property 'push' of undefined      failed  should\_add\_a\_new\_friend\_to\_the\_end\_of\_the\_friends\_array\_property |  | let userObj = [    {name:'Samuel', friends:[]},    {name: 'Ralph', friends: []},    {name: 'Annie', friends: []}  ];  function addFriend(user, newFriend)  {    // userObj.name.push(newFriend);    if (userObj.name == "user") userObj.friends.push(newFriend);    return userObj;  }  addFriend("Samuel","Ralph");  userObj; |
| invokeMethod | askHelp | method is a string that contains the name of a method on the object  Invoke this method using bracket notation.  Nothing needs to be returned.    Input Example:    { foo: function() {} }, 'foo' | lost on this | failed  should\_invoke\_the\_method\_on\_the\_object    What's Wrong?  TypeError: Cannot read property 'playNote' of undefined at invokeMethod (eval at n.evaluate (<https://replit.org/public/replbox_javascript.9411632bd490dec2b542.bundle.js:1:4058>), <anonymous>:9:9) at eval (eval at n.evaluate (<https://replit.org/public/replbox_javascript.9411632bd490dec2b542.bundle.js:1:4058>), <anonymous>:11:1) at eval (<anonymous>) at n.evaluate (<https://replit.org/public/replbox_javascript.9411632bd490dec2b542.bundle.js:1:4058>) at n.<anonymous> (<https://replit.org/public/replbox_javascript.9411632bd490dec2b542.bundle.js:1:2136>) at n.emit (<https://replit.org/public/replbox_javascript.9411632bd490dec2b542.bundle.js:1:9782>) at r (<https://replit.org/public/replbox_javascript.9411632bd490dec2b542.bundle.js:1:4474>) at Object.\_custom\_runSingleUnitTests (<https://replit.org/public/replbox_javascript.9411632bd490dec2b542.bundle.js:1:5796>) at <https://replit.org/public/secure/runner.js:1:1323> | let notes = [    {notesAvailable : ['A','Bb','B','C','C#','D','Eb','E','F'],   playNote : function(note){    console.log('I just played a ' + note);   }  }  ]  function invokeMethod(notes, method) {    notes.playNote(note);  }  invokeMethod(notes.playNote,'Bb'); |
| createObject | askHelp | Create and return an object that has the following properties: name, age, hobbies  name should be a string.  age should be a number.  hobbies should be an array that has three strings in it.    You can set the values of these properties to whatever you would like.    Example Output:    {  name: 'Alan',  age: 35,  hobbies: ['reading', 'cooking', 'horseback riding']  } | ?? | Failed tests  should\_return\_an\_object\_with\_name\_age\_and\_hobbies\_properties  should have the required properties with the required value types  RangeError: Maximum call stack size exceeded  More info | function createObject() {   const myObject = {      name: 'David',      age: 46,      hobbies: ['music', 'camping','boating']    };    console.log(createObject());    return myObject;    }  //tests |