# You are building a counter application that tracks the number of times a button is clicked. Implement the counter using closure.

**Solution:**

function createCounter (){ let count = 0;

return function (){ count++;

console.log{'Click count : ${count}');

} ;

}

let clickCounter = createCounter ();

*J I* Simulate button clicks

clickCounter (); *11* Output: Click count: 1 clickCounter (); *II* Output: Click count: 2

# You have an object representing a customer order with properties orderld, productName, and quantity. Use destructuring to extract and print these properties.



**fiil.**

# Solution:

let order = { orderid: "123456",

productName: "Laptop" , quantity: 2

};

*I I* Without destructuring

console.log( "Without Destructuring:" ); console.log( "Order ID:" , order.orderid) ; console. log ("Product Name: " , order. p roductName) ; console.log( "Quantity:" , order. quantity) ;

*I I* With destructuring

console.log( "\nWith Destructuring:" );

let {orderid, productName, quantity }= order ; console.log ("Order ID:" , orderid) ; console.log{ "Product Name: " , productName) ; console.log( "Quantity:" , quantity );

fiil.

1. In this coding challenge let's try to implement the cart feature usingjavascri pt closure. Using JS closures try to create a cart array and return a function to getCartltems.



Solution:

funct ion shoppingCa r t () { canst ca r t Items = [ ] ;

r etu r n {

g etCa r t items : func t ion ( ) { r et u r n ca r t items ;

}

} ;

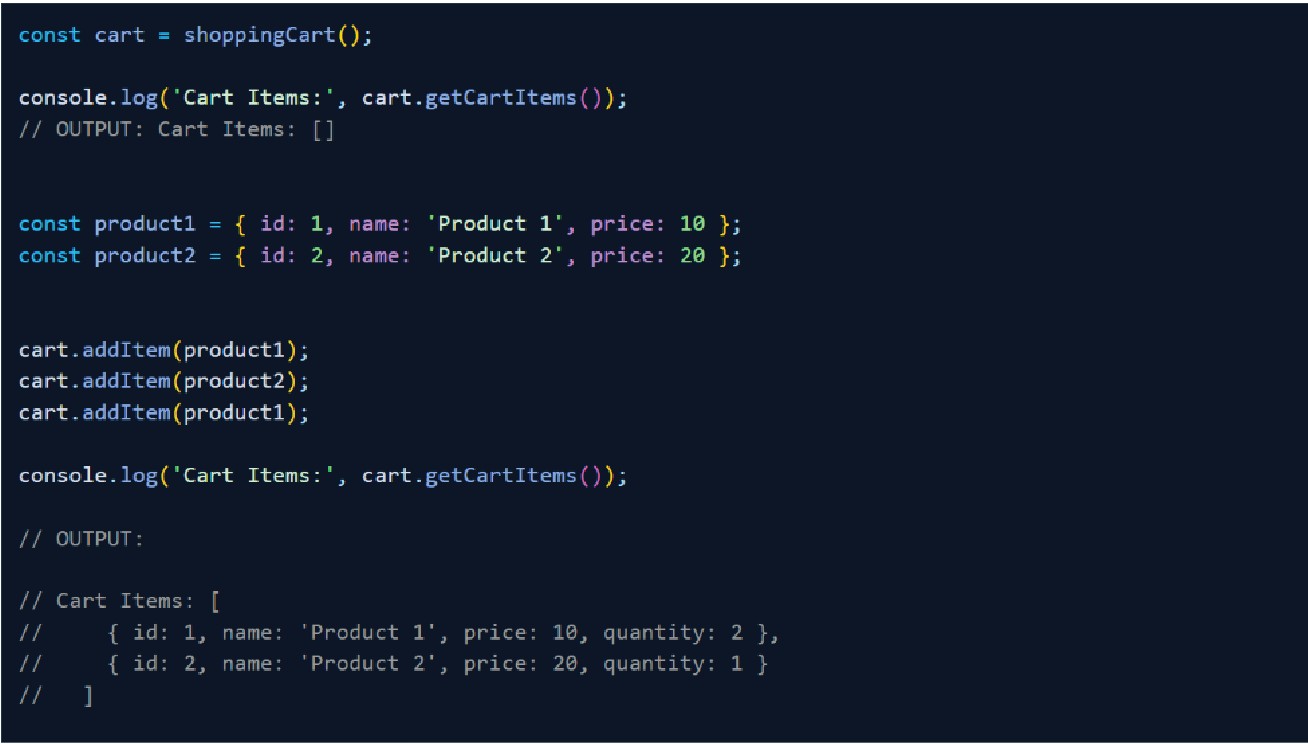
}

canst ca r t = shoppin gCa r t () ;

console . log ( ' Ca r t It em s : ' , ca r t .getCa r t items ( ) ) ;

*I I* OUTPUT : Ca r t Item s : [ ]

1. Continuing the previous codingchallenge,now let'simplement the add to cart feature. On calling add to cart closure function,the object of productld,name,quantity and price should be added to the cartltem. Note that if duplicate items with same prouductld is added,the product quantity must beincremented.Use javascript closures to achieve the output.



**fiil.**

**Solution:**

funct ion shoppingCa r t ( ) const ca r t Items = [ ] ;

function itemind exinCa rt ( p rod uctid ) {

ret u rn ca rt items . f indindex ( itern => it ern . id = == p rod uctid ) ;

}

ret u rn {

additem : funct ion ( p rod uct ) {

con st itemindex = itemindexinCa rt ( p rod uct . id ) ;

if ( itemindex ! == -1 ) {

ca r t it ems[ itemind ex ] . gua n itx++ ;

} else {

ca r t it ems . p ush ( { id : p rod uct.id , n ame : p rod uct .name , p rice : p rod uct.p r ice, qua nt ity : 1 }) ;

}

}.

getCa r tit ems : function ( ) { ret u rn ca r t It ems ;

}

} ;

con st ca r t = shoppingCa r t ( ) ;

console . log ( ' Ca r t Items : ' , ca r t . getCa r t items ( ) ) ;

*I I* OUTPUT : Ca r t Items : [ ]

con st p rod uct1 = { id : 1 , name : ' P r odu ct 1 ' , p r ice : 10 } ; con st p rod uct2 = { id : 2 , name : ' P r odu ct 2 ' , p r ice : 20 } ;

ca r t . additem ( p roductl ) ; ca r t . additem ( p roduct2) ; ca r t . additem ( p roductl ) ;

console . log ( ' Ca r t Items : ' , ca r t . getCa r t items ( ) ) ;

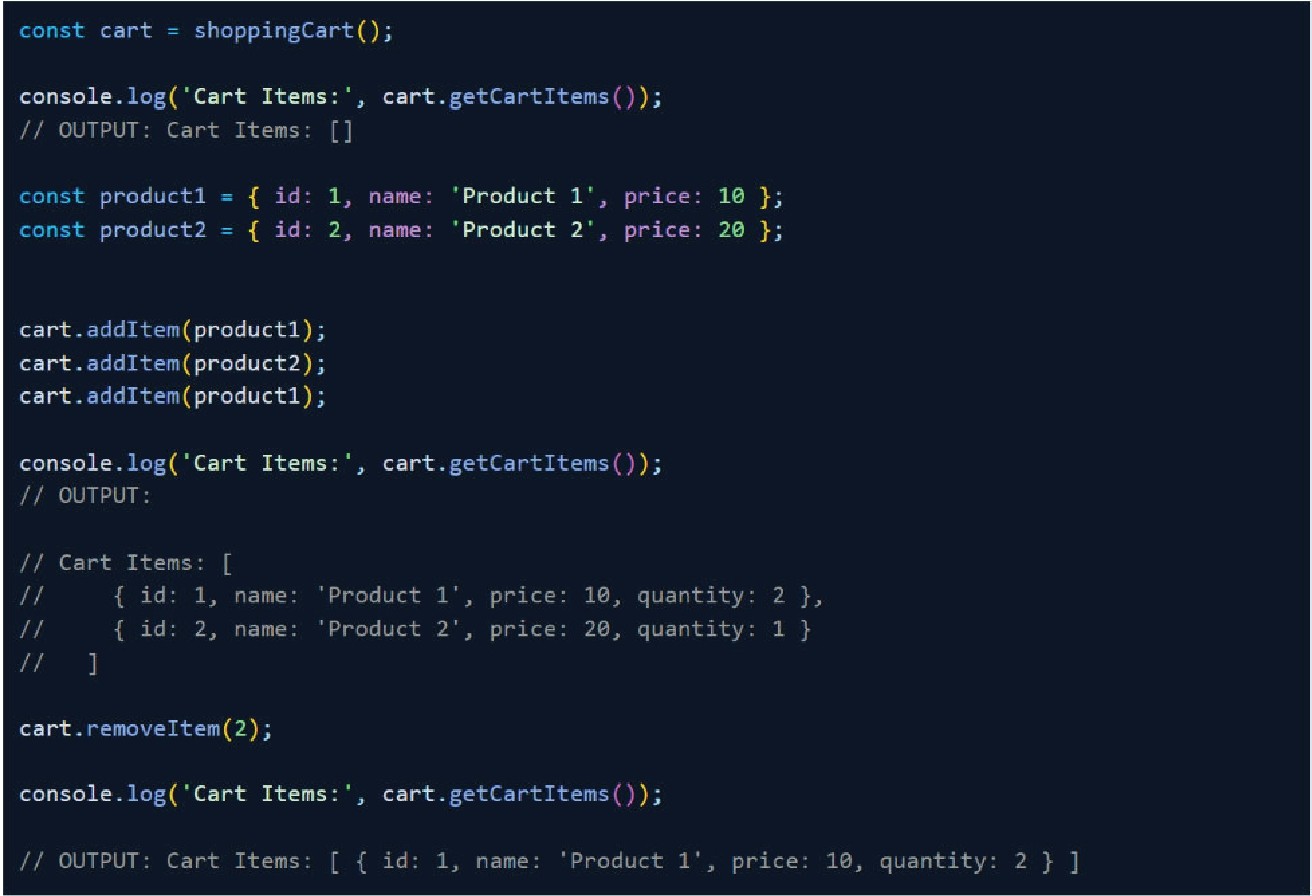
1. *I* OUTPUT :
2. Ca r t Items :

.

*II* id : 1. nam e : ' P rod uct 1 . p rice : 10 . qua ntity : 2 } *II* { id : 2 , nam e : ' P rod uct 2 ' p rice : 20 , qua ntity : } *II*

**fiil.**

# Continuing the previous coding challenge, now let's implement the remove item from cart feature. On calling the remove item closure function, the specified productld item must be removed from cartltems array.Usejavascri pt closures to achieve the output.



**Solution:**

.....

funct ion shoppin g Ca r t ( ) { con st ca r t items = [ ] ;

fu nc t ion itemind exinCa r t ( p rod u ctid ) {

r etu rn ca r t item s .find ind ex ( it ern = > it ern . id == = p rod uc t id ) ;

}

ret u r n {

ad d item : funct ion ( p r od uct ) {

can st itemi nd ex = itemind exinCa rt ( p rod uct . id ) ;

if ( item in d ex ! == -1 ) {

ca r t items[ itemi nd ex ] . ;

} else {

ca r t items . pu sh ( { id : p r od uct .. id , name : p rod u ct . n ame , p r ice : p rod uc t . p r ice , q ua n t ity : 1 }) ;

}

},

removeitem: function (productid) {

const indexToRemove = itemindexinCa rt (productld) :

if ( indexToRemove l== -1 ){

cartitems .splice (indexTaRemave, 1 );

}

},

getCartitems: function (){ return cartItems;

}

} ;

canst ca rt = shoppingCa rt ()*:*

cansale.lag( " Cart Items:' , cart.getCartitems ());

*J I* OUTPUT: Cart Items: []

canst p roductl = { id: 1 , name : ·Product 1 · price: 10 }; const product2 " {id:2, name : ·Product 2· price :20 };

cart.additem{product 1 ); cart.additern(product2) ; cart.additem( product 1 );

console.log( " Cart Items:' , ca rt.getCartitems ());

*I I* OUTPUT:

const product1 = { id: 1 , name : ·Product 1 · , price: 10 }; canst product2 = {id:2, name : 'Prod uct 2' , price: 20 };

cart.additem(prod uctl); cart.additem( product2) ; cart.additem(prod uct1);

console.log( 'Cart Items:', cart.getCartitems ()):

*I I* OUTPUT:

/ / Cart Items:

*II* id: 1• nam e: ·Product 1 ' ' price: 10, quantity: 2 }'

/ / { id: 2, nam e: 'Product 2' ' price: 20, quantity: }

*J I*

cart.removeitem( 2);

console.log( 'Cart Items:', cart.getCartltems ()):

*J I* OUTPUT: Cart Items: [ { id: 1 , nam e: ' Product 1' , price: 10, quantity: 2 } ]

**fiil.**

# You are developing a music playlist management system. Implement functions that leverage closures and higher-order functions to perform common playlist operations.

**Task 1:**Create a function createPlaylist that takes a playlist name as a parameter and returns a closure. This closure should allow adding and listing songs for the given playlist.

**Task 2:**Create a function addSong that takes a song name and artist as parameters and adds the song to the specified playlist. Use the closure created in TASK 1.

**Task 3:** Create a function listSongs that lists all the songs in a specified playlist. Use the closure created in the Task l



# Solution:

*I I* Task 1 : C r eate Playlist

function c r eatePla ylist ( p laylist Na me ) { let playlist = [ I ;

ret u rn {

addSon g : funct ion ( songName , a r t ist ) { playlist . push( { song Nia me , a r t ist }) ;

},

listSongs : function ( ) {

if ( pla ylist . lengtll *:= ==* 0 ) {

con sole . log ( " ${playlist N ame} Playlist is empty . ·) ;

} else {

con sole . log ( " ${playlist N ame} Playlist : ${pla y list .ma p ( song >

·${song .songName} by ${son g . a r t ist }·) .jo in ( · , · ) }·) ;

}

**fiil.**

}

} ;

}

*I I* Task 2 : Add Song to Pla ylist

function addSong ( pla ylist , songNa me , a r t ist ) { p lay list . add Song ( song Na me , a r t ist ) ;

}

*I I* Task 3 : List Songs in Pla ylist function listSong s ( p laylist ) {

p laylist . listSong s ( ) ;

}

const myPla ylist ;;; c rea tePla ylist ( "My Favo r ites " ) ; addSong (myPla ylist , "Sorig1 " , "A rtist1 " ) ;

addSong ( myPla ylist , "Song2 " , "A r tist2" ) ; add Song ( my Pla ylist , "Song3 " , "A r t ist3" ) ;

listSongs ( my Pla ylist ) ; *I I* Out p ut : My avorites Pla ylist : Song1 by A r t ist1 , Song2

by A r tist2 , by Ar t ist3