JavaScript

```
   Global Variables List:

   statAbbr     Contains an array of two-letter state or province
abbreviations


   Functions List:

   retrieveMemberInfo()
      Retrieves membership information from the fields within the
memberInfo
      cookie

*/
function addEvent(object, evName, fnName, cap) {
   if (object.attachEvent)
       object.attachEvent("on" + evName, fnName);
   else if (object.addEventListener)
       object.addEventListener(evName, fnName, cap);
}



var stateAbbr = new Array(60);
    stateAbbr[0]="";
    stateAbbr[1]="AL";
    stateAbbr[2]="AK";
    stateAbbr[3]="AZ";
    stateAbbr[4]="AR";
    stateAbbr[5]="CA";
    stateAbbr[6]="CO";
    stateAbbr[7]="CT";
    stateAbbr[8]="DE";
    stateAbbr[9]="DC";
    stateAbbr[10]="FL";
    stateAbbr[11]="GA";
    stateAbbr[12]="HI";
    stateAbbr[13]="ID";
    stateAbbr[14]="IL";
    stateAbbr[15]="IN";
    stateAbbr[16]="IA";
    stateAbbr[17]="KS";
    stateAbbr[18]="KY";
    stateAbbr[19]="LA";
    stateAbbr[20]="ME";
    stateAbbr[21]="MD";
    stateAbbr[22]="MA";
    stateAbbr[23]="MI";
    stateAbbr[24]="MN";
    stateAbbr[25]="MS";
```

```
    stateAbbr[26]="MO";
    stateAbbr[27]="MT";
    stateAbbr[28]="NE";
    stateAbbr[29]="NV";
    stateAbbr[30]="NH";
    stateAbbr[31]="NJ";
    stateAbbr[32]="NM";
    stateAbbr[33]="NY";
    stateAbbr[34]="NC";
    stateAbbr[35]="ND";
    stateAbbr[36]="OH";
    stateAbbr[37]="OK";
    stateAbbr[38]="OR";
    stateAbbr[39]="PA";
    stateAbbr[40]="RI";
    stateAbbr[41]="SC";
    stateAbbr[42]="SD";
    stateAbbr[43]="TN";
    stateAbbr[44]="TX";
    stateAbbr[45]="UT";
    stateAbbr[46]="VT";
    stateAbbr[47]="VA";
    stateAbbr[48]="WA";
    stateAbbr[49]="WV";
    stateAbbr[50]="WI";
    stateAbbr[51]="WY";
    stateAbbr[52]="GU";
    stateAbbr[53]="MP";
    stateAbbr[54]="PR";
    stateAbbr[55]="AS";
    stateAbbr[56]="VI";
    stateAbbr[57]="AA";
    stateAbbr[58]="AE";
    stateAbbr[59]="AP";



/* Add new code below */
addEvent(window, "load", retrieveMemberInfo, false);

 function retrieveMemberInfo() {
  alert("Registration data saved");

  if (retrieveCookie("lastName")) {

   var allSpans = document.getElementsByTagName("span");
   for (var i = 0; i < allSpans.length; i++) {
     var cValue = retrieveCookie(allSpans[i].id);
     if (cValue) allSpans[i].innerHTML = cValue;

    }
```

JavaScript

```
    var stateIndex = retrieveCookie("state");
    document.getElementById("state").innerHTML =
stateAbbr[stateIndex];

    if (retrieveCookie("member1") == "true")
     document.getElementById("memberOption").innerHTML = "one year
($30)";
    else if (retrieveCookie("member2") == "true")
     document.getElementById("memberOption").innerHTML = "two year
($40)";

   } else {

    var nonMember = " you do not have an account. ";
    nonMember += "join us today <a href='register.htm'>register</a> ";
    nonMember += "with us to recieve discounts ";
    nonMember += "and merchandise. ";

    document.getElementById("intro").innerHTML = nonMember;
    document.getElementById("accountTable").style.visibility =
"hidden";

   }
  }
```

JavaScript

```
Filename: cookies.js



   Functions List:

   addEvent(object, evName, fnName, cap)
      Adds an event handler to object where object is the object
      reference, evName is the name of the event, fnName is the
      reference to the function, and cap specifies the
      capture phase.

   writeDateString(dateObj)
      Returns the date contained in the dateObj date object as
      a text string in the format mmm. dd, yyyy

   writeCookie(cName, cValue, expDate, cPath, cDomain, cSecure)
      Write a cookie named cName with a value, cValue. The expDate
parameter
      contains a date object specifying the expiration date. The cPath
and
      cDomain parameters specify the path and domain for the cookie.
The
      cSecure parameter is a Boolean that specifies whether or not to
include
      the secure flag. The expDate, cPath, cDomain, and cSecure
parameters
      are all optional.

   retrieveCookie(cName)
      Retrieves the value of the cookie, cName.

   deleteCookie(cName)
      Deletes the cName cookie.

   writeMCookie(cName, fName, fValue, expDate, cPath, cDomain,
cSecure)
      Writes a multi-valued cookie where cName is the name of the
cookie,
      fName is a field name in the cookie, fValue is the field's
value,
      and expDate, cPath, cDomain, and cSecure are optional parameters
that
      specify the cookie's expiration date, path, domain, and secure
flag.

   retrieveMCookie(cName, fName)
      Retrieves the field value for the fName field from the multi-
valued
      cookie, cName.

   cookiesEnabled()
```

```
      Returns a Boolean value indicating whether cookies have been
enabled
      under the current browser.

*/

function addEvent(object, evName, fnName, cap) {
   if (object.attachEvent)
       object.attachEvent("on" + evName, fnName);
   else if (object.addEventListener)
       object.addEventListener(evName, fnName, cap);
}


function writeDateString(dateObj) {

   var monthName = new Array("Jan", "Feb", "Mar",
  "Apr", "May", "Jun", "Jul", "Aug", "Sep",
  "Oct", "Nov", "Dec");

   var thisMonth = dateObj.getMonth();
   var thisYear = dateObj.getFullYear();

   return monthName[thisMonth] + " " + dateObj.getDate() + ", " +
thisYear;
}



/* Add new code below */
 function writeCookie(cName, cValue,expDate, cPath, cDomain, cSecure)
{
   if (cName && cValue != "") {
     var cString = cName + "=" + escape(cValue);

     if (expDate) cString += ";expires=" + expDate.toGMTString();
     if (cPath) cString += ";path=" + cPath;
     if (cDomain) cString += ";domain=" + cDomain;
     if (cSecure) cString += ";secure";

     document.cookie = cString;
   }

 }

 function retrieveCookie(cName) {

  if (document.cookie) {

   var cookiesArray = document.cookie.split("; ");
   for (var i = 0; i < cookiesArray.length; i++) {
    if (cookiesArray[i].split("=")[0] == cName) {
```

```
     return unescape(cookiesArray[i].split("=")[1]);
     }
    }
   }
  }

  function deleteCookie(cName) {

   if (document.cookie) {

    var cookiesArray = document.cookie.split("; ");
    for (var i = 0; i < cookiesArray.length; i++) {
     if (cookiesArray[i].split("=")[0] == cName) {
      document.cookie = cName + "=;expires=Thu, 01-Jan-1970 00:00:01
GMT";
      }
     }
    }
}

function writeMCookie(cName, fName, fValue, expDate, cPath, cDomain,
cSecure) {

 if (cName && fName && fValue != "") {

  var subkey = fName + "=" + escape(fValue);

  var cValue = null;
  var cookiesArray = document.cookie.split("; ");

  for (var i = 0; i < cookiesArray.length; i++) {
   if (cookiesArray[i].split("=")[0] == cName) {
    var valueIndex = cookiesArray[i].indexOf("=") + 1;
    cValue = cookiesArray[i].slice(valueIndex);
    break;
    }
    }
    if (cValue) {

     var fieldExists = false;
     var cValueArray = cValue.split("&");
     for (var i =0; i < cValueArray.length; i++) {
      if (cValueArray[i].split("=")[0] == fName) {
       fieldExists = true;
       cValueArray[i] = subkey;
       }
      }
      if (fieldExists) cValue = cValueArray.join("&")
      else cValue += "&" + subkey;
      } else {
       cValue = subkey;
       }
```

JavaScript

```
  var cString = cName + "=" + cValue;

      if (expDate) cString += ";expires=" + expDate.toGMTString();
      if (cPath) cString += ";Path=" + cPath;
      if (cDomain) cString += ";domain=" + cDomain;
      if (cSecure) cString += ";secure";

document.cookie = cString;
  }
 }

 function retrieveMCookie(cName, fName) {

  if (document.cookie) {

  }
 }
```

JavaScript

```
Global Variables:
   styleList
      An array containing the link elements whose rel attribute equals
"stylesheet"
      or "alternate stylesheet" and whose title attribute is not empty

   Functions List:

   addEvent(object, evName, fnName, cap)
      Adds an event hander to object where evName is the name of the
event,
      fnName is the function assigned to the event, and cap indicates
whether
      event handler occurs during the capture phase (true) or bubbling
      phase (false)

   makeStyleListBox()
      Generate selection list for each preferred and alternate style
sheet linked
      to the current Web document

   changeStyle()
      Changes style sheet from the currently active sheet to whatever
the
      user changes the selected style from the selection list

*/

function addEvent(object, evName, fnName, cap) {
   if (object.attachEvent)
       object.attachEvent("on" + evName, fnName);
   else if (object.addEventListener)
       object.addEventListener(evName, fnName, cap);
}
```

JavaScript

```
Global Variables
   nextAd
      Used to track the next ad to be displayed in the banner box


   Functions

   addEvent(object, evName, fnName, cap)
      Run the function fnName when the event evName occurs in object.

   makeBannerAds()
      Create the banner box and stacked collection of banner ads

   changeBannerAd()
      Change the ad displayed in the banner box by changing the
stacking
      order of the ads

   moveNextAd(top)
      Move the nextAd object down top pixels from its current
location.

*/


function addEvent(object, evName, fnName, cap) {
   if (object.attachEvent)
       object.attachEvent("on" + evName, fnName);
   else if (object.addEventListener)
       object.addEventListener(evName, fnName, cap);
}

/* Add new code below */

var nextAd = null;

addEvent(window, "load", makeBannerAds, false);

function makeBannerAds()  {
    var bannerBox = document.createElement('div');
    bannerBox.id = "bannerBox";

    var bannerAd = null;
    var bannerAnchor = null;
    var bannerImage = null;
    for(var i = 0; i < adsURL.length; i++)  {
        bannerAd = document.createElement('div');
        bannerAd.className = "bannerAd";
        bannerAd.style.zIndex = i;

        bannerAnchor = document.createElement('a');
        bannerAnchor.href = adsURL[i];
```

JavaScript

```
        bannerImage = document.createElement('img');
        bannerImage.src = "banner" + i + ".png";

        bannerAnchor.appendChild(bannerImage);
        bannerAd.appendChild(bannerAnchor);
        bannerBox.appendChild(bannerAd);
    }

    document.body.appendChild(bannerBox);
    setInterval("changeBannerAd();", 5000);
}

function changeBannerAd()  {
    var allAds = document.getElementById("bannerBox").childNodes;

    for(var i=0; i<allAds.length; i++)  {
        if(allAds[i].style.zIndex == 0)  {
            allAds[i].style.top = "-50px";
            nextAd = allAds[i];
        }
    }

    for(var i=0; i<allAds.length; i++)  {
        allAds[i].style.zIndex--;

        if(allAds[i].style.zIndex < 0)
            allAds[i].style.zIndex = allAds.length-1;
    }

    var timeDelay = 0;
    for(var i=-350; i<=0; i++)  {
        setTimeout("moveNextAd(" + i + ")", timeDelay);
        timeDelay += 3;
    }
}

function moveNextAd(top)  {
    nextAd.style.top = top + "px";
}
```

JavaScript

```
   Custom Object Classes

   pokerGame
      The pokerGame object contains properties and methods for the
game
      of draw poker

   pokerDeck
      The pokerDeck object contains an array of poker cards and
methods
      for shuffling and drawing cards from the deck.

   pokerHand
      The pokerHand object contains an array of poker cards drawn from
a
      poker deck. The methods associated with the object include the
ability
      to calculate the value of the hand and to mark cards to be
discarded,
      replaced with new cards from a poker deck.

   pokerCard
      The pokerCard object contains properties and methods associated
with
      individual poker cards including the card rank, suit, and value.


*/




/*    The pokerGame Object  */

 var pokerGame = {
  currentPot: null,
  currentBet: null,

  placeBet : function() {
                  this.currentPot -= this.currentBet;
                  },
             payout :   function(odds) {
               this.currentPot += this.currentBet*odds
               }
  }




/*    The pokerDeck Object constructor  */
```

JavaScript

```
function pokerDeck() {

 this.cards = new Array(52);
 var suits = new Array("Club", "Diamond", "Heart", "Spade");
 var ranks = new Array("2", "3", "4", "5", "6", "7", "8", "9", "10",
"Jack", "Queen", "King", "Ace");

 // generate the array of pokerCards object
  var cardCount = 0;
  for (var i = 0; i < 4; i++) {
    for (var j = 0; j < 13; j++) {
      this.cards[cardCount] = new pokerCard(suits[i], ranks[j]);
      cardCount++;
      }
   }
   //shuffle method to randomize the card order
   this.shuffle = function() {
     this.cards.sort(function() {
      return 0.5 - Math.random();
      })
    }
    //dealTo method to deal cards from the deck
     this.dealTo = function(pokerHand) {
      for (var i = 0; i < pokerHand.cards.length; i++) {
        pokerHand.cards[i] = this.cards.shift();
        }
      }
}




/*    The pokerHand Object constructor   */

function pokerHand(handLength) {
//verify that only one parameter value is sent
 if (arguments.length != 1) {
  throw new Error ("Enter a single parameter value");
  }
//verify that the value is a number
 if (arguments[0].constructor != Number) {
  throw new Error ("handLength must be a number")
 }
 this.cards = new Array(handLength);

 }

//return the maximum value

pokerHand.prototype.maxValue = function() {
 var values = new Array();
```

JavaScript

```
 for (var i = 0; i < this.cards.length; i++){
  values[i] = this.cards[i].value();
  }

  return values.max();
 }

 //return true if the hand is a flush

 pokerHand.prototype.hasFlush = function() {

  for (var i = 1; i < this.cards.length; i++) {
   if (this.cards[i].suit != this.cards[i - 1].suit) return false;
   }
   return true;
   }

   //straight
   pokerHand.prototype.hasStraight = function() {

   var values = new Array();


   for (var i = 0; i < this.cards.length; i++) {
     values[i] = this.cards[i].value();
     }

     values.numericSort(true);

     for (var i = 1; i < values.length; i++) {
      if (values[i] != values[i - 1] + 1) return false;
      }
      return true;
     }

     //royal flush

     pokerHand.prototype.hasRoyalFlush = function() {
      return this.hasFlush() && this.hasStraight() && this.maxValue()
== 14;
      }
      //return true if straight flush

      pokerHand.prototype.hasStraightFlush = function() {
       return this.hasFlush() && this.hasStraight() && this.maxValue()
!= 14;
       }

       //return duplicates

       pokerHand.prototype.matches = function() {
```

JavaScript

```
        var values = new Array();
        for (var i = 0; i < this.cards.length; i++) {
         values[i] = this.cards[i].value();
       }
        values.numericSort(true);

        var duplicates = new Array();
        for (var i = 1; i < this.cards.length; i++) {
         if (values[i] == values[i - 1]) duplicates.push(values[i]);
         }

         //check for full house & 4 of a kind
         if (duplicates.length == 3) {
          for (var i = 1; i < 3; i++) {
           if (duplicates[i] != duplicates[i - 1]) return "Full
House";
           }
            return "Four of a Kind";
            }
           //check 3 kind vs 2 pair
           if (duplicates.length == 2) {
            if (duplicates[0] == duplicates[1]) return "Three of a
Kind";
            else return "Two Pair";
           }
          //check for jack or better
          if (duplicates.length == 1 && duplicates[0] >= 11) return
"Jacks or Better";

//no winning hand
return "";

        }
   //return the value of the hand

   pokerHand.prototype.handValue = function() {

   if (this.hasRoyalFlush()) return "Royal Flush"
   else if (this.hasStraightFlush()) return "Straight Flush"
   else if (this.hasFlush()) return "Flush";
   else if (this.hasStraight()) return "Straight";
   else return this.matches();
   }

 //return the odds multuplyr

 pokerHand.prototype.handOdds = function() {

  switch (this.handValue()) {
   case "Royal Flush" : return 250;
   case "Straight Flush" : return 50;
   case "Four of a Kind" : return 25;
```

```
   case "Full House" : return 9;
   case "Flush" : return 6;
   case "Straight" : return 4;
   case "Three of a Kind" : return 3;
   case "Two Pair" : return 2;
   case "Jacks or Better" : return 1;
   default: return 0;

  }
 }
```

```
/*    The pokerCard Object constructor   */
function pokerCard(suit, rank) {

 this.suit = suit;
 this.rank = rank;

}
//img source

pokerCard.prototype.imageSrc = function() {
 var fileName = this.suit.substring(0,1);

 if (this.rank == "10") fileName += "10"
  else fileName += this.rank.substring(0,1);

  fileName += ".png";
  return fileName.toLowerCase();
  }
  //discard the propert marks cards to be replaced

  pokerCard.prototype.discard = false;

  //raplcefromdeck

  pokerCard.prototype.replaceFromDeck = function(pokerDeck) {
   this.rank = pokerDeck.cards[0].rank;
   this.suit = pokerDeck.cards[0].suit;
   pokerDeck.cards.shift();
  }
//value method return the numeric rank

pokerCard.prototype.value = function() {
 switch(this.rank) {
  case "Jack": return 11;
  case "Queen": return 12;
  case "King": return 13;
```

```
 case "Ace": return 14;
 default: return parseInt(this.rank);
 }
}
```