
## Nested Loops to make Deck of Cards

**nested loop to make a deck of playing cards**

In this lesson we will use a nested loop to make a deck of cards from which we will deal a 5-card hand of poker.

1. Open **05.04-Deck-of-Cards.html** and see that it has a div of five image tags, under which is a DEAL CARDS button:

```
    <div id="card-box">
        <img src="images/10-of-Diamonds.png">
        <img src="images/Jack-of-Diamonds.png">
        <img src="images/Queen-of-Diamonds.png">
        <img src="images/King-of-Diamonds.png">
        <img src="images/Ace-of-Diamonds.png">
    </div>

    <button>DEAL CARDS</button>
```

2. Preview the html file in the browser. It's a 5-card hand of poker--a royal flush, no less.
3. Click the DEAL CARDS button. Five random cards replace the royal flush.
4. Open the **images** folder and click a card to see it. Each card has two identifiers: a **kind** and a **suit** for a file name structure of **kind-of-suit.png**:

- 2-of-Clubs.png - Queen-of-Spades.png

5. Switch the html file to use START .js.
6. Open **05.04-Deck-of-Cards-START.js**. The **kind** and **suit** arrays are given, but we also need a new empty array to store the cards.
7. Declare a new empty array, called **deck**:

```
    const kinds = [2, 3, 4, 5, 6, 7, 8, 9, 10, 'Jack', 'Queen', 'King',
'Ace'];
    const suits = ['Diamonds', 'Hearts', 'Spades', 'Clubs'];
    const deck = [];
```

We will begin by just making a deck of file names. Once we get that to work, we will upgrade the deck so that each card is an object with several properties.

8. Set up a nested loop, where the outer loop iterates over the **kinds** array, and the inner loop iterates over **suits**:

```
    for (let i = 0; i < kinds.length; i++) {
        for (let j = 0; j < suits.length; j++) {
        }
    }
```

The outer loop runs 13 times, and the inner loop runs four times, for a total of 52 iterations, one for each card.

9. In the inner loop, concatenate the card file name, and push the result into the deck:

```
        let cardFileName = `${kinds[i]}-of-${suits[j]}.png`;
        deck.push(cardFileName);
```

10. After the loop ends, log the deck to see what we get.

```
    console.log(deck);
    /*
    ['2-of-Diamonds.png', '2-of-Hearts.png', '2-of-Spades.png',
    '2-of-Clubs.png', '3-of-Diamonds.png', '3-of-Hearts.png',
        -- ETC. --
    'King-of-Spades.png', 'King-of-Clubs.png', 'Ace-of-Diamonds.png',
'Ace-of-Hearts.png', 'Ace-of-Spades.png', 'Ace-of-Clubs.png']
    */
```

**deck of cards as an array of objects**

Let's run the nested loop again, but this time we will make each card an object, in which the file name is just one of several properties. The five properties will be:

- **name** ("2 of Diamonds", etc.)
- **file** ("2-of-Diamonds.png", etc.)
- **kind** (2, 3, 4..Jack, Queen, King, Ace)
- **suit** (Diamonds', 'Hearts', 'Spades', 'Clubs')
- **valu** (numeric value; face cards = 10, Ace = 11)

**valu** is for storing the numeric value of each card, from 1-11, using the blackjack scoring system, where aces start with a value of 11, face cards are worth 10 and the numbered cards are their respective numeric values.

11. Declare a new, empty array to hold our 52 card objects.

```
const deckOfCards = [];
```

12. Set up the nested loop;

```
for (let i = 0; i < kinds.length; i++) {
    for (let j = 0; j < suits.length; j++) {
    }
}
```

13. Simplify the current array items by passing them to variables:

```
for (let i = 0; i < kinds.length; i++) {
    for (let j = 0; j < suits.length; j++) {
        let kind = kinds[i];
        let suit = suits[j];
    }
}
```

14. Concatenate the card and image file names. The "Queen of Diamonds" has a file name of "Queen-of-Diamonds.png":

```
        let kind = kinds[i];
        let suit = suits[j];
        let name = `${kind} of ${suit}`;
        let file = `${kind}-of-${suit}.png`;
```

15. Declare a variable, **valu**, with an inital value of 0:

```
        let name = `${kind} of ${suit}`;
        let file = `${kind}-of-${suit}.png`;
        let valu = 0;
```

16. Run conditional logic to set the **valu** property. Only "Jack", "Queen" and "King" have more than three characters, which is what **(kind.length > 3** checks for:

```
        let valu = 0;
        if(kind == 'Ace')  {
            valu = 11;
        } else if(kind.length > 3) {
            valu = 10;
        } else {
```

```
                valu = kind;
            }
```

17. Declare an object called **card**. Its properties equal the variables we have made:

```
            let card = { name: name, file: file, kind: kind,
                         suit: suit, valu: valu };
```

18. Push the card object into the **deckOfCards** array:

```
            deckOfCards.push(card);
```

19. Below the nested loop, output the array of 52 objects:

```
    console.log(deckOfCards);
    /*
    0: {name: '2 of Diamonds', file: '2-of-Diamonds.png', kind: 2, suit:
'Diamonds', valu: 2}
    51: {name: 'Ace of Clubs', file: 'Ace-of-Clubs.png', kind: 'Ace',
suit: 'Clubs', valu: 11}
    */
```

**ouputting cards to the DOM**

Let's deal a hand of 5-card poker. Clicking the DEAL CARDS button will:

- call a function called **dealCards**
- the function will get all five of the card img tags from the DOM
- the function will run a loop five times, once for each card image
- each iteration of the loop will generate a random int from 0-51
- the random integer will be used to get a card by its array index
- the item's file name will be concatenated into an image path
- the img object will have its src property set to the image path

This will be the simplest implementation possible, meaning that there will only be one player hand dealt and all five cards will appear at once, without the benefit of a realistic-looking time delay between cards.

20. Get the button and tell it to run the function when clicked:

```
    const btn = document.querySelector('button');
    btn.addEventListener('click', dealCards);
```

21. Get the five images. The **querySelectorAll** method will get all img tags and make an array of them called a Node List:

```
const imgArr = document.querySelectorAll('img');
```

22. Now for the function, which loops through the array of images:

```
function dealCards() {
    for (let i = 0; i < imgArr.length; i++) {
    }
}
```

23. Each time the loop runs, generate a random number from 0-51:

```
function dealCard() {
    for (let i = 0; i < imgArr.length; i++) {
        let r = Math.floor(Math.random() * deckOfCards.length);
    }
}
```

24. Get a random card from the deck by index:

```
    for (let i = 0; i < imgArr.length; i++) {
        let r = Math.floor(Math.random() * deckOfCards.length);
        let card = deckOfCards[r];
    }
```

25. Set the src of the image in the imgArray by concatenating the file pate using the file property of the card object:

```
        let r = Math.floor(Math.random() * deckOfCards.length);
        let card = deckOfCards[r];
        imgArr[i].src = "images/" + card.file;
```

26. Reload the page and click the DEAL CARDS button. Five new cards should appear.

If you keep clicking the DEAL CARDS button, you will notice a bug: the same card showing up more than once in the same hand. To fix this, remove cards from the deck as they are dealt.

27. Using the splice() method, remove the dealt card from the deck:

```

```

```
            let card = deckOfCards[r];
            imgArr[i].src = "images/" + card.file;
            deckOfCards.splice(r, 1);
```

Now we have a new bug: after 10 hands we have used 50 cards and are down to our last two. To fix this, we can use conditional logic to replenish the deck once the supply runs too low. To do this, we need to make a "backup" -- a copy -- of the original deck.

28. Outside of the function, copy the array using **slice(0)**, which returns a copy from index 0 to the end of the array:

```
    let deckCopy = deckOfCards.slice(0);
```

29. Comment out the dealCards function, and rewrite it. Start with an if statement that makes a fresh copy if there are fewer than 5 cards left:

```
    function dealCard() {

        if(deckCopy.length < 5) {
            deckCopy = deckOfCards.slice(0);
        }

    }
```

30. Switch to using deckCopy throughout; after each hand, log how many cards are left, so you can see when the card supply is replenished:

31. Rerun the page. Now you should be able to deal hands indefinitely, without ever running out of cards.

**END: Lesson 05.04 NEXT: Lesson 05.05**