

avionschool

Lesson 8.0 Control Flow in Javascript

Before controlling the flow...

- If coercion (implicit or explicit type conversion) concerns the *type* of the variable,
- how about the *immutability* of the variable?

`var` - Declares a variable, optionally initializing it to a value.

`let` - Declares a block-scoped, local variable, optionally initializing it to a value.

`const` Declares a block-scoped, read-only named constant.

What is Control Flow?

- The control flow is the order in which the computer executes statements in a script.

Comparison Operators

Comparison operators (`>` `>=` `<` `<=` `==` `!=`). These operators work just as they would in a math class: greater than, less than, and etc. We use these operators to evaluate two expressions. As the computer runs the code, the operator will return either a true (if the statement is true) or a false (if the statement is not true).

Logical Operators

- We can also combine two equality expressions and ask if either of them are true, both of them are true, or neither of them is true.

`&&` - This will evaluate both expressions and will return true if BOTH expressions are true. If one (or both) of them is false this operator will return false

```
1 if (100 > 10 && 10 === 10) {  
2     console.log('Both are true, so run this');  
3 }
```

|| - It will determine if one of the expressions is true. It will return true if one (or both) of the expressions is true. It will return false if BOTH expressions are false

```
1 if (10 === 9 || 10 > 9) {  
2   console.log('One of them is true, so run this');  
3 }
```

! - The NOT operator will return the opposite boolean value of what is passed to it

```
1 if (!false) {  
2   console.log('This will return true.');
```

if statements

- These are two simple if statements: one with just a “then” branch and one with both a “then” branch and an “else” branch

```
● ● ●  
  
if (cond) {  
    // then branch  
}  
  
if (cond) {  
    // then branch  
} else {  
    // else branch  
}
```

- Instead of the block, else can also be followed by another if statement:

```
● ● ●  
  
if (cond1) {  
    // ...  
} else if (cond2) {  
    // ...  
}  
  
if (cond1) {  
    // ...  
} else if (cond2) {  
    // ...  
} else {  
    // ...  
}
```

- You can continue this chain with more else ifs.

switch statements

A switch statement looks as follows:



```
switch («switch_expression») {  
    «switch_body»  
}
```

The body of switch consists of zero or more case clauses:



```
case «case_expression»:  
    «statements»
```


And, optionally, a default clause:



```
default:  
    «statements»
```

A SWITCH IS EXECUTED AS FOLLOWS:

- It evaluates the switch expression.
- It jumps to the first case clause whose expression has the same result as the switch expression.
- Otherwise, if there is no such clause, it jumps to the default clause.
- Otherwise, if there is no default clause, it does nothing.



```
switch (num) {  
    case 1:  
        return 'Monday';  
    case 2:  
        return 'Tuesday';  
    case 3:  
        return 'Wednesday';  
    case 4:  
        return 'Thursday';  
    case 5:  
        return 'Friday';  
    case 6:  
        return 'Saturday';  
    case 7:  
        return 'Sunday';  
}
```


while loops

A while loop has the following syntax:




```
while («condition») {  
  «statements»  
}
```

Before each loop iteration, while evaluates condition:

If the result is falsy, the loop is finished.

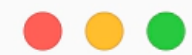
If the result is truthy, the `while` body is executed one more time.



```
const arr = ['a', 'b', 'c'];  
while (arr.length > 0) {  
  const elem = arr.shift(); // remove first element  
  console.log(elem);  
}  
// Output:  
// 'a'  
// 'b'  
// 'c'
```

do-while loops

- The do-while loop works much like while, but it checks its condition after each loop iteration, not before.



```
let input;  
do {  
  input = prompt('Enter text:');  
  console.log(input);  
} while (input !== 'q');
```

`prompt()` is a global function that is available in web browsers. It prompts the user to input text and returns it.

for loops

A for loop has the following syntax:



```
for («initialization»; «condition»; «post_iteration») {  
  «statements»  
}
```

It has three parts and each of them is optional:

- **initialization:** sets up variables, etc. for the loop. Variables declared here via `let` or `const` only exist inside the loop.
- **condition:** This condition is checked before each loop iteration. If it is falsy, the loop stops.
- **post_iteration:** This code is executed after each loop iteration.

FOR LOOPS SAMPLE:



```
for (let i=0; i<3; i++) {  
  console.log(i);  
}
```

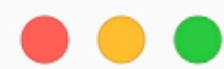
// Output:

// 0

// 1

// 2

THIS IS HOW TO LOG THE CONTENTS OF AN ARRAY VIA A FOR LOOP:



```
const arr = ['a', 'b', 'c'];  
for (let i=0; i<arr.length; i++) {  
  console.log(arr[i]);  
}
```

// Output:

// 'a'

// 'b'

// 'c'

for-of loops

- A for-of loop iterates over an iterable – a data container that supports the iteration protocol. Each iterated value is stored in a variable, as specified in the head:

```
for («iteration_variable» of «iterable») {  
  «statements»  
}
```

- The iteration variable is usually created via a variable declaration:

```
const iterable = ['hello', 'world'];  
for (const elem of iterable) {  
  console.log(elem);  
}  
// Output:  
// 'hello'  
// 'world'
```

- But you can also use a (mutable) variable that already exists:

```
const iterable = ['hello', 'world'];  
let elem;  
for (elem of iterable) {  
  console.log(elem);  
}
```

for-in loops

- A for-of loop iterates over an iterable – a data container that supports the iteration protocol. Each iterated value is stored in a variable, as specified in the head:

```
for («iteration_variable» of «iterable») {  
  «statements»  
}
```

- The iteration variable is usually created via a variable declaration:

```
const iterable = ['hello', 'world'];  
for (const elem of iterable) {  
  console.log(elem);  
}  
  
// Output:  
// 'hello'  
// 'world'
```

- But you can also use a (mutable) variable that already exists:

```
const iterable = ['hello', 'world'];  
let elem;  
for (elem of iterable) {  
  console.log(elem);  
}
```

const: for-of vs. for

Note that in `for-of` loops you can use `const`. The iteration variable can still be different for each iteration (it just can't change during the iteration). Think of it as a new `const` declaration being executed each time in a fresh scope.

In contrast, in `for` loops you must declare variables via `let` or `var` if their values change.

Coding Challenge - Playing Cards

- Using an array data structure, create a standard deck of cards(52) represented by strings: (e.g. '♠', '♥', '♦', '♣', 'A', '2')
- Print the content of the cards whenever there is a change in the deck after the following tasks:
- Create a function that accepts the array of cards and returns the shuffled cards
- Create a function that accepts the array of cards and returns arranged by suit
- Create a function that accepts the array of cards and returns arranged by face or value in ascending / descending order (e.g. Ace, 2.. 10, Jack, Queen, King)
- Create a function that deals a card (printed name should be like 'Ace of Spades', 'Two of Diamonds')
- After a shuffle, loop your deck by dealing five cards until the deck is exhausted. On each deal, print what kind of poker hand is dealt! https://en.wikipedia.org/wiki/List_of_poker_hands