Looping in JavaScript

Knowledge of how to loop through objects in JavaScript as well as which type of loop to use in a particular situation is very important in order to be proficient in JavaScript. Following are five types of loops available and examples of when to use each.

For-in Loop:

Used to iterate through objects.

Example:

```
const pet = { name: "Rex", breed: "Pug", color: "brown" }
for (const key in pet) { console.log(pet[key]) } // => Rex / Pug / brown
```

Note: This works on any object with enumerable properties (non-symbols). So it works on an array, but it is much better to use a loop designed for arrays.

Caution: Does not guarantee that the properties will be visited in the same order as the object. Should be avoided if you are modifying properties during the loop. A better option would be to use Object.keys(pet) which returns an array that can be used in a loop.

For-of Loop:

Used to iterate through arrays.

Example:

```
const petArray = [
   { name: "Rex", breed: "Pug", color: "brown" },
   { name: "Pebbles", breed: "Dalmatian", color: "white with black spots" },
]
for (const element of petArray) { console.log(element['name']) } //=> Rex / Pebbles
```

Note: This works on strings, arrays, and array-like objects.

Example of looping through a string:

```
const stringArray = "this is a string"
for (const letter of stringArray) { console.log(letter) } //=> t / h / i / s / ...
```

Caution: This is best used when need to visit each element and are not changing the elements. If you want to perform an action on each element, there are better methods such as map and reduce. If you want to control the iteration or access the index of the element in the loop then a basic for loop will work better.

For-Each Loop:

Loops through an array using a callback function. Optional index, array, and thisArg arguments.

```
petArray.forEach( (element, index) => console.log(index, element.name)) //=> 0 "Rex" / 1 "Pebbles"
```

Note: This is similar to for-of loops, but requires a callback function. This is useful if you need access to the index or need to perform more complex actions. Again, this is best used when you need to visit each element in the array. For more complex situations a regular for loop might be better.

For Loop:

Used for a loop with three optional expressions.

Example:

```
let numbers = []

for (let i=0; i<5; i++) { numbers.push(i) }

console.log(numbers) //=> [0,1,2,3,4]
```

Note: The expressions in parenthesis are optional and very flexible. It is easy to customize how they are handled in the loop. This makes the for loop a great option for more complicated situations.

Caution: If you don't specify the optional expressions, be sure to evaluate and update them in the loop. Otherwise you will get infinite loops and/or unexpected results.

While / Do-While:

Used to loop until the specified condition evaluates to false.

Example of while loop:

```
let i = 0


while (i < 5) {

   console.log('Hi!', i)

   i++

} //=> Hi! 0 / Hi! 1 / Hi! 2 / Hi! 3 / Hi! 4
```

Note: The condition is checked at the beginning of the loop so if the condition immediately evaluates to false the while loop will not execute.

Example of do-while loop:

```
let k = 5

do {

  console.log('Hi Again!', k)

  k++

} while (k < 5) //=> Hi Again! 5
```

Note: This condition is checked at the end of the loop so the loop will always execute at least one time.

Continue / Break:

These are two useful keywords when working with loops.

Continue: Skip to the next iteration of the loop.

Break: Exits the loop

This was a short summary of the loops available in JavaScript. Knowing these important concepts will help you become a better JavaScript programmer.