codecademy

Loops

While Loop

The `while` loop creates a loop that is executed as long as a specified condition evaluates to `true`. The loop will continue to run until the condition evaluates to `false`. The condition is specified before the loop, and usually, some variable is incremented or altered in the `while` loop body to determine when the loop should stop.

```
while (condition) {
  // code block to be executed
}

let i = 0;

while (i < 5) {
  console.log(i);
  i++;
}
```

Reverse Loop

A `for` loop can iterate “in reverse” by initializing the loop variable to the starting value, testing for when the variable hits the ending value, and decrementing (subtracting from) the loop variable at each iteration.

```
const items = ['apricot', 'banana',
'cherry'];

for (let i = items.length - 1; i >= 0; i
-= 1) {
  console.log(`${i}. ${items[i]}`);
}

// Prints: 2. cherry
// Prints: 1. banana
// Prints: 0. apricot
```

Do...While Statement

A `do...while` statement creates a loop that executes a block of code once, checks if a condition is true, and then repeats the loop as long as the condition is true. They are used when you want the code to always execute at least once. The loop ends when the condition evaluates to false.

```
x = 0
i = 0

do {
  x = x + i;
  console.log(x)
  i++;
} while (i < 5);

// Prints: 0 1 3 6 10
```

For vs. While loops:

while loops are easier to reason about, so should always be preferred.
For loops are good when things are simple and straightforward.
However, in those situations, it's probably better to use functions like .forEach() or .filter()

So the moral is: use functions for simple task, use while loops and invariants for complex task, avoid for loops

For Loop

A `for` loop declares looping instructions, with three important pieces of information separated by semicolons `;` :

- The *initialization* defines where to begin the loop by declaring (or referencing) the iterator variable
- The *stopping condition* determines when to stop looping (when the expression evaluates to `false`)
- The *iteration statement* updates the iterator each time the loop is completed

```
for (let i = 0; i < 4; i += 1) {
  console.log(i);
};

// Output: 0, 1, 2, 3
```

Initialize variable:

1. check if condition satisfied
Yes => run body, run iteration statement, back to 1
No => exists loop

Looping Through Arrays

An array's length can be evaluated with the `.length` property. This is extremely helpful for looping through arrays, as the `.length` of the array can be used as the stopping condition in the loop.

```
for (let i = 0; i < array.length; i++){
  console.log(array[i]);
}

// Output: Every item in the array
```

Break Keyword

Within a loop, the `break` keyword may be used to exit the loop immediately, continuing execution after the loop body. exists the loop IT IS IN

Here, the `break` keyword is used to exit the loop when `i` is greater than 5.

not recommended.

```
for (let i = 0; i < 99; i += 1) {
  if (i > 5) {
     break;
  }
  console.log(i)
}

// Output: 0 1 2 3 4 5
```

Nested For Loop

A nested `for` loop is when a `for` loop runs inside another `for` loop.
The inner loop will run all its iterations for *each* iteration of the outer loop.

```
for (let outer = 0; outer < 2; outer += 1) {
  for (let inner = 0; inner < 3; inner += 1) {
    console.log(`${outer}-${inner}`);
  }
}

/*
Output:
0-0
0-1
0-2
1-0
1-1
1-2
*/
```

Loops

A *loop* is a programming tool that is used to repeat a set of instructions. *Iterate* is a generic term that means “to repeat” in the context of *loops*. A *loop* will continue to *iterate* until a specified condition, commonly known as a *stopping condition*, is met.