college CDI

Scripting language I (LS1)

Course 4

Loops

Although they aren't always absolutely needed, you will soon discover loops can save you a lot of work. They allow to repeat actions many times without having to write the same codes over and over again.

Incrementing (++) / decrementing (--)

- Incrementing (using ++) adds 1 to a numerical value.
- Decrementing (using --) subtracts 1 from a numerical value.

Operators order

Incrementation and decrementation can be used in two ways : before or after the variable. Depending on the position, the result won't be the same. If the operator is placed before the variable, the value is incrimented or decrimented before the expression's evaluation. If the operator is placed after the variable, the value is incrimented or decrimented after the expression's evaluation.

Traditional method:

```
<script>
        let number = 0;
        number = number + 1;

        alert(number);          // Displays : « 1 »
</script>
```

Incrementing :

```
<script>
        let number = 0;
        number++;

        alert(number);          // Displays : « 1 »
</script>
```

Explanation:

a = b++;

The variable « a » takes the value of the variable « b », then be is incremented. If initially b=10, after evaluating the expression, *a*=10 and *b*=11.

a = ++b;

The variable *b* is first incremented, then variable *a* takes the value of variable b. If initially *b*=10, after the evaluation of the expression, *a*=11 and *b*=11.

The while while loop

The while loop is a lot like conditional structures we've already learned. Although, the purpose of a loop is to repeat an instruction or a sequence of instructions a certain number of times; each cycle is called iteration (repetition). The instructions will be repeated as long as a condition hasn't been verified. The loop stops when the condition returns a *false* Booleans.

While loop syntax :

```
while (condition) {
        instruction_1;
        instruction_2;
        instruction_3;
}
```

It is of course necessary to define a condition that will eventually stop the loop. Otherwise, the loop would continue indefinitely.

Example : Incrementing a value from 1 to 10

```
<script>
let number = 1;

while (number < 10) {
            number++;
}

alert(number);                                  // Displays : « 10 »
</script>
```

Explanation :

Initially, *number* equals 1.

The loop then evaluates if *number* is lower than 10.

As long as it's true, *number* is incremented (+1) and the loop evaluates the condition again. This cycle repeats as long as *number* is lower than 10.

When the condition is not verified (when it returns false), when number reaches 10, the loop stops and the following instructions are executed.

Example :
Let's create a program that will a user to supply us with his name and those of his siblings.

```
<script>
let list = "", name;
keepon = true;
while (keepon) {
        name = prompt("Write a name :");

        if (name) {
                list += name + "";
        } else {
                keepon = false;
        }
}
alert(list);
</script>
```

To force a loop to stop : break;

A loop can be stop by replacing *keepon = false;* by *break;*

To force a loop to stop : continue;

Not as commonly used, this instructions doesn't stop a loop to execute the next instructions. It rather stop a loop to execute the next loop.

The « do while » loop

Not very commonly used, this loop, although quite similar to the *while* loop, executes the instructions first and then evaluates de condition.

Syntax :

```
do {
      instruction_1;
      instruction_2;
      instruction_3;
} while (condition);
```

The « for » loop

Even if this loop structure may seem complicated at first, it i a lot like a *while* loop. It is very much used in JavaScript; much more than *while*. You will soon discover that the for loop is absolutely essential in JavaScript to manipulate elements like matrix or objects.

Syntax :

```
for (declaration; condition; incrementation) {
      instruction_1;
      instruction_2;
      instruction_3;
}
```

Example :

```
<script>
let i;                          // on déclare notre variable
for(i=0; i<10; i++)             //  initialisation; condition; incrémentation
alert(i);
</script>
```

Explanation:

The loop parenthesis contains three blocks (sort of instructions) ending with a colon.

The first block assigns a value to the variable. The second block provides a condition and the third block takes care of incrementing the value each time the loop is repeated.

In the following example, the loop creates five consecutive message windows displaying the current iteration number..

Exemple :

```
<script>
for (let iter = 1; iter < 5; iter++) {

alert("Itération numéro : " + iter);
}
</script>
```

Explanation :

The first block declares the variable and its initial value. The second block provides a condition (as long as the variable is lower than 5). The third block increments the variable's value on each iteration.

Finally, the instruction creates a message window containing a string followed by the current variable' value.

Important note

In JavaScript, it is better not to declare a variable within the conditional structure (between brace brackets). This can cause performance and logical issues.

It is way safer to declare variables in the initial block (within parenthesis), like shown in the examples.

One the loop is done, the variable are still available to use elsewhere in the program.

Execution priority

- The for loop three blocks aren't executed simultaneously, but in sequence :

    1. **Initializations :**
       Just before the loop starts

    2. **Condition :**
       Before each loop iteration

    3. **Incrementation :**
       After each loop iteration (if break is used, loop won't be incremented).

Generating a random number

Math.random()

Math.random() generates a random number between 0 and 1 including zero and excluding one; that means a number between 0 and 0.999999.

Getting a whole number

Since the result of ***Math.random()*** generates a decimal number, it is necessary to multiply this result by 10 to get a whole number:

```
Math.random() * 10;
```

But this means the result will be between 0 and 9.999999, so it need to be rounded.

Rounding numbers

To round a number, ***Math.floor()*** can be used. It rounds down numbers, for instance, 5.72321 would be rounded to 5.

```
Math.floor(Math.random() * 10);
```

Then, the possible results of random number generation would be between 0 and 9. To make it possible between 1 and 10, 1 must be added.

```
Math.floor((Math.random() * 10) + 1);
```

Assignment 5

Create a program where the computer chooses a random number between 1 and 100.

The, the user is asked to choose a number within the same range. If the number chosen by the user is lower or higher than the one defined by the computer, a message tells the user and ask to try again.

When the user finally finds the answer, a congratulation message is displayed as well as the number of tries needed to get the good answer.