Repetition

Lesson Objectives

- Understand repetition control logic
- Learn to use JavaScript loop syntaxes

Control logic

- Sequence
- Selection
- **Repetition**

Repetition/ Looping

- During code execution, some statements need to be executed more than one time. For achieving this, programming languages provide loop syntax, which keeps on executing the statements inside the loop until certain condition is met.
- Loop can be controlled based on a counter or based on some event (sentinel controlled).
- Based on when the looping condition is checked, loop can be pre-test or post-test loop.

while loop

- pre-test loop
- Can be used as both counter-controlled loop or sentinel-controlled loop.

```
while (condition) {
        statements;
}
```

- Counter-controlled
  - see example: *lecture_codes/lesson4/while_counter_controlled.js*
- Sentinel-controlled
  - see example: lecture_codes/lesson4/while_sentinel_controlled.js
- Careful! if you forget to update the loop counter or because of logic error loop counter is never reaching the exit condition, loop will be infinite.
  - Example - updated counter-controlled while loop example
  - Ctrl-C Ctrl-C to break out of nodejs process

do while

- post-test loop
- Better suited for sentinel-controlled loop
    - The statements inside do block is guaranteed to be executed at least once
- Example – updated sentinel controlled while loop.
    - See lecture_codes/lesson4/do_while.js
    - Notice: this version makes prompt at only one place, that is inside the do part of the do-while loop.

for

- Pre-test loop
- Better suited for counter-controlled loop
  - Updating counter is part of the syntax, hence you won't forget!

```
for (initialization; condition; update) {
        statements;
}
```

- See *lecture_codes/lesson4/for_loop.js*

Inline variable declaration

- When the “counter” variable is declared right in the loop, it is called an “inline” variable declaration.
  - Such variables are visible only inside the loop.

```
for (let i = 0; i < 3; i++) {
  alert(i); // 0, 1, 2
}
alert(i); // error, no such variable
```

- If you need to use the value of counter variable outside the loop scope, you need to declare the variable outside of the loop.

Exercise

- Write a loop that prints all even number between 1 to 20 (inclusive)
  - Write both `while` and `for` versions
- Write a loop that keeps on prompting for age until you enter age older than 18
  - Write both `while` and `do while` versions.
- Make a defining table and program to print out the balance of a savings account that compounds interest monthly. Prompt the user for the
  - initial amount
  - annual interest rate
  - number of years to compound

Nested Loops

- Loop inside of a loop

```
for(let i = 1; i<=10; i++){
    let row="";
    for(let j=1; j<=10; j++){
        row += i*j +"\t";
    }
    console.log(row);
}
```

Main Point

Looping/ repetition is the control structure that powers computation. The same operations are repeated over and over with slight changes to state information produces all the interesting effects of computation. *Science of consciousness: Self referral awareness, pure consciousness or the unified field aware of itself by its own nature is the source of all energy and matter. "Curving back on myself I create again and again."*

Exercise

- Write code to print following patterns on the console
  - Hint: see *lecture_codes/lesson4/nested_for.js*

```
11111
22222
33333
44444
55555
```

```
12345
12345
12345
12345
12345
```

Breaking the loop

- Normally, a loop exits when its condition becomes falsy.
- But we can force the exit at any time using the special break directive.

```
let sum = 0;

while (true) {
  let value = +prompt("Enter a number", '');
  if (!value) break; // (*)
  sum += value;
}
alert('Sum: ' + sum);
```

- The break directive is activated at the line (*) if the user enters an empty line or cancels the input. It stops the loop immediately

Continue to the next iteration

- The `continue` directive is a “lighter version” of `break`.
  - It doesn’t stop the whole loop.
  - Instead, it stops the current iteration and forces the loop to start a new one (if the condition allows).

```
for (let i = 0; i < 10; i++) {
  // if true, skip the remaining part of the body
  if (i % 2 == 0) continue;
  alert(i); // 1, then 3, 5, 7, 9
}
```

  - Refactor above code without using the continue statement.

break and continue

- The break statement ends a loop prematurely.
  - Example - finding if a number is prime without using break statement.
    - See *lecture_codes/lesson4/test_prime_no_break.js*
  - Example - finding if a number is prime using break statement.
    - See *lecture_codes/lesson4/test_prime_using_break.js*

- The continue statement "jumps over" one iteration in the loop.
  - See example: *lecture_codes/lesson4/continue_keyword.js*

- break and continue are rarely required in well designed code
  - best practice to avoid unless have a specific reason to use
  - typical use case involves performance optimization of stopping a loop upon some condition
  - can generally achieve same effect with a sentinel while loop
    - See *lecture_codes/lesson4/primeWhile.js*

Assignments

- Reading
  - Loops: while and for (javascript.info)

- Programming exercises in Resources > assignments