UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
BACHELOR OF SCIENCE IN COMPUTER SCIENCE
WEB TECHNOLOGIES COURSE – SPRING 2025
TEACHER: Luigi Libero Lucio Starace, Ph.D.

DATE: March 20, 2025
BEST DONE BEFORE: March 25, 2025

ASSIGNMENT: #04 TOPIC: JAVASCRIPT

For the exercises in this assignment, create a new HTML document with a dedicated external script for each exercise.

EXERCISE 1. VARIABLES AND FUNCTIONS

1. Declare a variable called `studentName` and assign your name as its value.
2. Create a variable named `age` and assign your age as a number.
3. Define a variable `isEnrolled` and set its value to `true`.
4. Use `console.log` to display the values of these variables along with their data types.
5. Create a function called "`studentInfo`" that takes as input the following arguments:
    a. `studentName`: the name of a student, expected to be a string
    b. `age`: the age of a student, expected to be a number
    c. `course`: the name of a course, expected to be a string. This value defaults to "`Web Technologies`".
    d. `isEnrolled`: a Boolean representing whether the student is enrolled in the course. This value defaults to `true`, unless specified otherwise;

    The function prints a message to the console along the lines of "*Student (age) is enrolled in the X course*" or "*Student (age) is NOT enrolled in the Y course*".
6. Invoke the `studentInfo` function by passing only the `studentName` and `age` variables as parameters. Observe the output.
7. Invoke the `studentInfo` function by passing a value for each parameter. Observe the output.

EXERCISE 2. HELLO OBJECTS

1. Create an object called `movie` with properties `title`, `director`, and `year`.
2. The value of the `director` property is another object, with properties `firstName`, `lastName`, `birthYear`, `deathYear`.
3. Assign appropriate values to these properties for your favourite movie.
4. Add a new property called "`is part of a saga`", and assign to it an appropriate boolean value.
5. Add a method called "`describe`" to the object. The method outputs to the console all the information about the movie.
6. Invoke the method on the object and make sure that the output is as expected.
7. Create a **Movie** constructor, and use it to create a new movie.
8. Remove the "`is part of a saga`" property from the movie you originally created.
9. Create a **Trilogy** constructor. The constructor takes as input a `title` (intended as a string), and three movie objects. Then, create a **Trilogy** object representing your favourite movie trilogy.

ASSIGNMENT: #04 TOPIC: JAVASCRIPT

EXERCISE 3. (ALMOST) PROVING THE COLLATZ CONJECTURE

The Collatz sequence was proposed by Lothar Collatz in 1937, and is defined by starting from any positive integer n and applying two simple rules. If n is even, then the next number in the sequence is obtained by dividing n by 2. If n is odd, then the next number in the sequence is obtained by multiplying n by 3 and then adding 1. The process continues until the sequence reaches 1.

The Collatz conjecture states that, for every positive integer $n > 2$, the Collatz sequence starting at n converges to 1. Proving (or disproving) the Collatz conjecture is an extraordinarily difficult problem, regarded as one of the most difficult challenges in mathematics. To date, no general proof has been found, despite the best efforts of countless excellent mathematicians.

For the Web Technologies course, you are not asked to exhibit a general proof for the Collatz conjecture (we already have our fair share of difficult stuff at hand, like mastering CSS layouts or JavaScript automatic type coercion), but we will make do with proving that the conjecture holds for all positive integer numbers up to 10'000.

In this exercise, you are asked to write a JavaScript program to show that the Collatz conjecture holds for all positive integer numbers in the range [2, 10'000]. The program should print in the standard output, for each processed starting number, a message like those shown below.

```
Collatz proved for n=2: sequence converges to 1 in 1 steps
Collatz proved for n=3: sequence converges to 1 in 7 steps
Collatz proved for n=4: sequence converges to 1 in 2 steps
Collatz proved for n=5: sequence converges to 1 in 5 steps
Collatz proved for n=6: sequence converges to 1 in 8 steps
Collatz proved for n=7: sequence converges to 1 in 16 steps
Collatz proved for n=8: sequence converges to 1 in 3 steps
...
Collatz proved for n=9998: sequence converges to 1 in 91 steps
Collatz proved for n=9999: sequence converges to 1 in 91 steps
Collatz proved for n=10000: sequence converges to 1 in 29 steps
```

ASSIGNMENT: #04 TOPIC: JAVASCRIPT

EXERCISE 4. WORKING WITH FUNCTIONS (★)

Some functions can be very computationally expensive. In such cases, when the results of an invocation are *stable* (i.e., they generally do not change between subsequent invocations), a caching mechanism can prove very useful!

Create a `cachingDecorator` function. The function should take as input another function `f`, and a number `n` representing an invocation threshold for *cache staleness*. The function should then return a new function `g`, which acts as a wrapper for `f` and includes in its implementation a caching mechanism. More in detail, the first time `g` is invoked, it invokes in turn `f` and saves the result of this invocation in its cache before returning it. Subsequently, when `g` is invoked again, the function will directly return the cached value and will not invoke `f` at all. However, `g` keeps track of the number of times its caching mechanism is used. When the cache becomes *stale* (i.e., when it is used more than `n` times), `g` updates the cached value by invoking `f` again.

The following snippet of JavaScript code demonstrates the usage of the `cachingDecorator` and the expected outputs:

```
function cachingDecorator(f, n){
  /* Write your code here! */
}

/* Do not change f() */
function f(){
  let value = Math.random();
  console.log(`f has been invoked, result is ${value}`);
  return value;
}

let g = cachingDecorator(f, 3);

//since numbers are randomly generated, you will get different numbers
console.log(g()); //f has been invoked, result is 0.5157221
console.log(g()); //0.5157221
console.log(g()); //0.5157221
console.log(g()); //f has been invoked, result is 0.8924242
console.log(g()); //0.8924242
console.log(g()); //0.8924242
console.log(g()); //f has been invoked, result is 0.1713458
```