5. Decision and loops

5.1 Decisions and loops (for loop)

- **Loops in JavaScript**
  - A loop in programming is defined as a segment of code that executes multiple times. JavaScript loops are used to repeatedly run a block of code until a certain condition is met. If a condition evaluates to true, the loop's statement (block of code) is executed. The condition will be evaluated again and again, and the code block runs as long as the condition is true. When the condition evaluates to false, the loop stops.
  - **Loop vs iteration in programming**
    - **Loop**: As stated above, loop a block of code that runs multiple times
    - **Iteration**: It refers to the process by which a code segment is executed once. One iteration refers to 1-time execution of a loop. A loop can undergo many iterations.
  - JavaScript supports different kinds of loops. For example, for, for/in, for/of, while, do/while. However, they all essentially do the same thing: they repeat an action some number of times. We will focus on for loop and while loop as they are the most commonly used ones.
- **JavaScript For Loop**
  - For loops are commonly used to count a certain number of iterations to repeat a block of code. Meaning, we use for loop when we want to execute the code for a SPECIFIED amount of time
  - **For loop Syntax:**

```
 for (initialization; condition; post/final-
expression) {
 // statements (code block to be executed)
}
```

  - As you can see it from the syntax above, the JavaScript for loop statement allows you to create a loop with three optional expressions that are separated by semicolon (;)

- **The initialization expression**: This expression is commonly used to create a counter variable. A counter variable is a variable that keeps track of the number of times a specific piece of code is executed. A counter variable basically counts the loop. All the variable does is tell our loop where it should start counting from. This is like saying, "we are going to start counting starting from whatever value we give it in here". We usually provide 0 as value for the initial counter. However, it can be a different number too.
    - **Note 1**: The initialization expression is executed only once when the loop starts.
    - **Note 2 (Loop scope)**: If you use the var keyword to declare the counter variable, the variable will have either function or global scope. In other words, you can reference the counter variable after the loop ends. However, if you use the let keyword to declare the counter variable, the variable will have a blocked scope, which is only accessible inside the loop. See the example below:

```
for (let counter = 1; counter < 5; counter++)
{
    console.log(counter); // prints numbers 1
to 4
}
console.log(counter); prints "ReferenceError:
counter is not defined" because we used let
to define counter
```

- **Condition**: This is the expression that is checked one time prior to every iteration. If the condition evaluates to true; the loop's statement is executed. If it evaluates to false, the loop stops. The loop is terminated if the condition evaluates to false. See the example below:

```
for (var counter = 1; counter < =10; counter++)
{
    console.log(counter);
```

```
}
```

- In the above example, we want the code to be executed 10 times. What the condition expression (counter < =10) does is check if we have already run the code 10 times. If we haven't, it runs the code again
- **Note 1**: If you want to stop the loop before the condition expression evaluates to false, use break statement. In the example below, even if the condition evaluates true when j is 8 or 9, the code will not run because we used break when j is 7. Thus, the loop terminates when j reaches 7.

```
for (var j = 1; j <= 10; j++) {
if (j === 7) {
break;
}
console.log(j); // prints numbers 1 to 6
and loop stops
}
```

- **Note 2**: The condition is optional. If you omit it, the for-loop statement considers it as true. If you omit the condition expression, you need to use a break statement to terminate the loop.

```
for (let counter = 1;; counter ++) {
    console.log(counter);
    if (counter >= 10) {
        break;
    }
}
```

- **Post/final-expression**: This expression is run after every single iteration. Generally, you use the post-expression to update the counter variable (either to increment or decrement the counter)

- When the initialization expression, condition expression and final expression parts come together, the meaning they give to the following for loop will be:

```
for (var i = 0; j <= 10; i++) {
console.log(i); // prints numbers 0 to 10
}
```

  -
  - Start counting your loop starting from 0. To keep track of how many times the code inside the for loop block is executed, save that value on a variable called i. To start with, lets assign 0 to our variable i.
  - Then compare that i with 10, and if i is less than or equal 10, run the code
    - In the first loop, i is 0, which is less than 10. So, the code executes.
  - Once the code finishes executing, increase the value of i by 1
    - In the second loop, i is 1. 1 is still less than 10, so, execute the code again.
  - Once the code finishes executing, increase the value of i by 1
    - In the third loop, i is 2. 2 is still less than 10, so, execute the code again.
  - Repeat this process until i becomes greater than 10

- Look at code below to understand what a for loop does when we want to print elements of an array in the console

```
var someNumbers = [7, 58, 27];
var lengthOfArray = 3;
var loopNumber = 0;
var someMessage = "";
var i;
for (i = 0; i < 3; i++){
loopNumber = i + 1;
someMessage = "Loop " + loopNumber + ":" +
someNumbers[i];
console.log(someMessage);
}
```

5.2 Decisions and loops (while loop)

- The JavaScript while statement creates a loop that executes the code within the while block as long as the test condition evaluates to true. After every single code execution, it comes back and checks if the condition expression is still true. If the condition expression evaluates to false, execution continues with the statement after the while loop.
- **While loop Syntax**:

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

  - **Note**: The while loop evaluates the expression before each iteration, therefore, the while loop is known as a pretest loop. For this reason, it is possible that the statement inside the while loop is never executed.
  - **Example**:

```
var f = 0;
while (f < 3) {
console.log(f); // prints the numbers 0 to 2
f++;
}
```

  - **These are the steps involved in the above while loop example**:
    - First, outside of the loop, the f variable is set to 0.
    - Second, before the first iteration begins, the while statement checks if f is less than 3 and executes the statements inside the loop body.
    - Third, in each iteration, the loop increments f by 1 and after 3 iterations, the condition f < 3 is no longer true, so, code executes 3 times the loop terminates.

- **While loop vs for loop in JavaScript**
    - In for loop, the number of iterations to be done is already known and is used to obtain a certain result. While loop is used when the number of iterations is unknown. In while loop the command runs until a certain condition is reached and the statement is proved to be false.
        - While loops are often used if you want to run code for an unspecified number of times. For example, if we want to ask a user for a number between 1 and 10, we don't know how many times the user may enter a larger number, so we keep asking "while the number is not between 1 and 10".
        - JavaScript for loops are used if you already know, ahead of time, how many times we have to repeat the loop
    - In the absence of a condition, for loop iterates for an infinite number of times till it reaches break command. However, In the absence of a condition, while loop shows an error.
    - Initialization (of the counter variable) in for loop is done only once when the program starts. Initialization is done every time the loop is iterated.

5.3 Loops explained with example (function that prints sum of all numbers between 1 and the number)

- Let us now create a function that takes a number as a parameter and prints in the console the sum of all numbers between 1 and this number.
    - **Test case 1:** addUp(4) prints10 (1 + 2 + 3 + 4 = 10)
    - **Test case 2:** addUp(13) prints 91 (1 + 2 + 3+ ...+13 = 91)
- Let us use for loop to solve this problem
- **Pseudocode**
    - Check if the provided value is a number
    - Create/declare a variable sum to save the total sum.
    - Give a 0 initial value for this variable
    - Use for loop that starts from 1 by declaring a variable with a value of 1

- Make the loop end when it finds the given number by providing a condition that i is less than or equal to the given number
- Increase i's value by 1 for every iteration
- Within your for loop, add the value of i on the total sum variable
- Return the total sum outside the for loop

- **Convert your Pseudocode into JavaScript code**

```
function addUp1(num) {
let sum = 0;
for (let i = 1; i <= num; i++) {
sum = sum + i;
}
return sum;
}
console.log(addUp1(600));//prints180300
```