CS ACTIVITY FORM

Student Name	PANAGSAGAN, JESSIE P.	**Year & Section**	1Y., 1B	**Activity No.**	DSA - A2
Class Instructor	CO Pascua	**Class Subject**	DSA	**Class Schedule**	
Activity Proper:					

A. Discuss the following in not less than 5 Sentences EACH.

- What are the Control Flow Structures in JavaScript (Conditional)?

Control flow structures, this will include the `if`, `else if`, `else`, and `switch`, this will guide us on how to code and executes based on specific conditions. An `if` statement assesses that whether a condition is true and runs the appropriate code, while `else if` and `else` serve as alternatives for different cases. And the `switch` statement offers a systematic way to manage multiple values linked to a single variable, making the code more comprehensible. Together, these structures allow developers to create codes that can adapt to user interactions and a range of scenarios.

- What are the Control Flow Structures in JavaScript (Loop)?

Control flow structures in JavaScript, especially loops, allow for the repeated execution of code based on defined conditions. The primary loop types are the for loop, while loop, and do...while loop, each catering to specific needs. The for loop is particularly useful for iterating over arrays or executing a set number of times through its initialization, condition, and increment statements. In contrast, the while loop continues to run as long as its condition remains true, making it ideal for cases where the number of iterations isn't fixed. Finally, the do...while loop ensures that the code block runs at least once before evaluating its condition, providing versatility for situations that require an initial execution.

- For each Structure: present the Syntax and Sample implementations.

Control flow structures in JavaScript, particularly conditional statements, enable developers to execute specific pieces of code based on particular conditions. The main conditional statements include if, else if, else, and switch, which together provide the flexibility to create dynamic code.

The *if* statement evaluates a condition, and if true, executes a block of code.

```
let temperature = 30;

if (temperature > 25) {

   console.log("It's a hot day!");

}
```

The *else if* statement allows additional conditions to be checked if the initial if condition is false.

```
let temperature = 30;

if (temperature > 25) {

   console.log("It's a hot day!");
```

```
}
```

The *else* statement provides a fallback option when none of the preceding conditions are met.

```
let temperature = 10;

if (temperature > 25) {
   console.log("It's a hot day!");
} else {
   console.log("It's not a hot day.");
}
```

The *switch* statement provides a way to execute different code blocks based on the value of a single variable.

```
let fruit = "apple";

switch (fruit) {
   case "banana":
      console.log("This is a banana.");
      break;
   case "apple":
      console.log("This is an apple.");
      break;
   default:
      console.log("Fruit not recognized.");
}
```

The for loop is useful for iterating over arrays or executing a block of code a specific number of times.

```
for (let i = 0; i < 5; i++) {
   console.log(`Iteration number: ${i}`);
}
```

The *while* loop runs as long as its condition remains true, making it suitable for cases where the number of iterations isn't predetermined.

```
let count = 0;
```

```
while (count < 5) {
   console.log(`Count is: ${count}`);
   count++;
}
```

The *do...while* loop guarantees that the code block runs at least once before evaluating its condition.

```
let num = 0;
do {
   console.log(`Number is: ${num}`);
   num++;
} while (num < 5);
```

B. Discuss what is an Infinite Loop

- What is an infinite Loop

An *infinite loop* is a sequence of instructions in computer programming that loops endlessly without a terminating condition or a break to exit the loop. This results in the loop running indefinitely, which can cause the program to freeze or crash. These loops often occur due to logical errors, such as a missing condition that would normally allow for termination. Debugging infinite loops can be challenging, as they may require programmers to analyze the code meticulously to identify the root cause. Ultimately, understanding and preventing infinite loops is crucial for efficient program execution and maintaining system stability.

Ex.

```
while (true) {
   console.log("This loop will run forever!");
}
```

In this example are demonstrating the infinite loop that continuously logs a message to the console.

- How is an infinite Loop triggered and how to avoid the accidental implementation of an infinite loop?

The infinite loop is typically triggered when a loop's condition is set in such a way that it never evaluates to false, or when there is no condition specified to exit the loop. like for example the Condition Mistakes, If the loop condition is incorrectly set using a condition that is always true the loop will never terminate. And also on how to avoid the accidental implementation of an infinite loop is to carefully plan loop conditions, that before writing a loop, outline the logic and conditions that should be met for termination. Ensure that these conditions can realistically and be satisfied. And always use a comments and documentation clearly document

that the intended logic and exit conditions within the code. This not only helps others understand the code but also makes it easier for us to spot potential issues.