# Cheat Sheet: Chap 4\_5

# Conditions, Logic, Flow Control and Loops:

Conditions:

We can make decisions in our code using if and if else statements. It is very much like this template:

***“if \*some condition is true\*, then \*a certain action will happen\*, else \*another action will happen\*”***

Take a look on the above statement and analyze the Syntax. It started with an “IF” indicating some kind of a yes/no is going to happen. You will be dealing almost entirely in your coding life, where you are going to evaluate conditions in a business problem, or control the flow of your logic using these open and close gates.

//IF and Else

let rain = true;//lets create a boolean variable

if(rain)//check if the value witing the parethesis is true, if yes, open the gate for true value

{

console.log("\*\* Taking my umbrella when I need to go outside \*\*");

} else //check if the value in the previous parethesis is false, if false, open the gate for false value

{

console.log("\*\* I can leave my umbrella at home \*\*");

}

Off course the word is not always white and black, but there are a lot of grey shades in between white and black. You are not always sure, if you are getting a black or white. So lets address those grey areas in your code.

In the above example, there are only 2 values in the truth table. Either its raining, or its not. But what if, the sky is cloudy, and its not raining, but a heavy chances of rain. In that case, logic cannot be written as an open and shut case, but rather have a lot of other parameters involved to make an informed decision

Let’s look at the below snippet

//If Else IF and ELSE

rain = true;//lets create a boolean variable for rain

let isCloudy=true;//a variable to denote if the weater is cloudy

if(rain && isCloudy)//check if the weather is cloudy and its rainig

{

console.log("\*\* Taking my umbrella when I need to go outside \*\*");

} else if(!rain && isCloudy)//may be the weather is cloudy but no rain

{

console.log("\*\* I shouldnt leave my umbrella at home as it is cloudy \*\*");

}else if(rain && !isCloudy)//may be the weather is not cloudy but its rain

{

console.log("\*\* Its raining, but not cloudy. May be the rain stops, but i should take my umbrella \*\*");

}else //No Rain No Clouds

{

console.log("\*\* Sunny Day Sweepin' the clouds away. On my way to where the air is sweet. \*\*");

}

By Adding one additional Boolean variable, our possibilities in our truth table suddenly got double. Instead of a clear open and shut statement, our log may varies between 2 more possibilities. The possibilities are mutualy exclusive and will be handled by an IF-ELSE IF-ELSE statement. You can have as many ELSE-IF in a conditional flow, but there may only be one ELSE at the end of the statement.

Switch-Case:

If else statements are great for evaluating Boolean conditions. There are many things you can do with them, but in some cases, it is better to replace them with a switch statement. This is especially the case when evaluating more than four or five values.

switch (activity) {

    case "Get up":

        console.log("It is 6:30AM");

        break;

    case "Breakfast":

        console.log("It is 7:00AM");

        break;

    case "Drive to work":

        console.log("It is 8:00AM");

        break;

    case "Lunch":

        console.log("It is 12:00PM");

        break;

    case "Drive home":

        console.log("It is 5:00PM");

        break;

    case "Dinner":

        console.log("It is 6:30PM");

        break;

    default:

        // code to be executed when no cases match

        console.log("So jaa nahi tu gabbar ajayayga :p")

        break;

}

If there is a case matched within these evaluation the code in the corresponding code block will be executed and terminated the case by using a break statement. It is not mandatory to add a break statement as there will be no interpreter error, but it often acts as a logic bomb. There should be a rule of thumb defining the default case , for a gracefully execution of the switch-case block.

Loops:

Loops execute a code block a certain number of times. We can use loops to do many things, such as repeating operations a number of times and iterating over data sets, arrays, and objects.

* while loop
* do while loop
* for loop
* for in
* for of loop

While Loop:

A while loop executes a certain block of code as long as an expression evaluates to true.

//While Loop

//Simple While Loop

let i = 0;

while (i < 10) {//Boolean Condition Evaluated, Execute, Not Evaluated, Exit

    console.log(i);

    i++;

}

//A Little Complex One

let someArray = ["Mike", "Antal", "Marc", "Emir", "Louiza", "Jacky"];

let notFound = true;

while (notFound && someArray.length > 0) {

    if (someArray[0] === "Louiza") {

        console.log("Found her!");

        notFound = false;

    } else {

        someArray.shift();

    }

}

Do-While

In some cases, you really need the code block to be executed at least once. For

example, if you need valid user input, you need to ask at least once. The same goes for trying to connect with a database or some other external source: you will have to do so at least once in order for it to be successful. And you will probably need to do so as long as you did not get the result you needed. In these cases, you can use a do while loop.

//Do-While

let number;

do {

    number = prompt("Please enter a number between 0 and 100: ");

} while (!(number >= 0 && number < 100));

For Loops:

for loops are special loops. The syntax might be a little bit confusing at first, but you will find yourself using them soon, because they are very useful.

1. Initialize the variables.
2. Check the condition.
3. If the condition is true, execute the code block. If the condition is false, the loop will end here.
4. Perform the statement (the third part, for example, i++).
5. Go back to step 2.
6. //For Loop
7. let arr = [];
8. for (let i = 0; i < 100; i++) {
9. arr.push(i);
10. }

Nested Loops:

A loop inside a loop is

called a nested loop. Often it is not the best solution to the problem. It could even be a sign of poorly written code (sometimes called "code smell" among programmers), but every now and then it is a perfectly fine solution to a problem.

let arrOfArrays = [];

for (let i = 0; i < 3; i++) {

    arrOfArrays.push([]);

    for (let j = 0; j < 7; j++) {

        arrOfArrays[i].push(j);

    }

}

For Of Loop (Iterator)

There is another loop we can use to iterate over the elements of an array: the for of loop. It cannot be used to change the value associated with the index as we can do with the regular loop, but for processing values it is a very nice and readable loop.

//For Of

let names = ["Chantal", "John", "Maxime", "Bobbi", "Jair"];

for (let name of names) {

    console.log(name);

}

Break and Continue:

* When the break statement is executed, the loop will end, even when the condition is still true.
* Break can be used to quit the loop, and continue can be used to move on to the next iteration of the loop. It quits the current iteration and moves back up to check the condition and start a new iteration