| **BASIS FOR COMPARISON** | **FOR** | **WHILE** |
| --- | --- | --- |
| Declaration | for(initialization; condition; iteration){ //body of 'for' loop } | while ( condition) { statements; //body of loop } |
| Format | Initialization, condition checking, iteration statement are written at the top of the loop. | Only initialization and condition checking is done at the top of the loop. |
| Use | The 'for' loop used only when we already knew the number of iterations. | The 'while' loop used only when the number of iteration are not exactly known. |
| Condition | If the condition is not put up in 'for' loop, then loop iterates infinite times. | If the condition is not put up in 'while' loop, it provides compilation error. |
| Initialization | In 'for' loop the initialization once done is never repeated. | In while loop if initialization is done during condition checking, then initialization is done each time the loop iterate. |
| Iteration statement | In 'for' loop iteration statement is written at top, hence, executes only after all statements in loop are executed. | In 'while' loop, the iteration statement can be written anywhere in the loop. |

In general, you should use a for loop when you know how many times the loop should run. If you want the loop to break based on a condition other than the number of times it runs, you should use a while loop.

If we just use < or > it will not be inclusive to the number in the condition check

You must use the = sign too

++ increment by one

--decrement by one

Console.log & return

Console.log prints for the developer to see

Return gives the value back to the function

T Diagrams

The point of a T Diagram is to keep track of your variables, as our algorithms get longer it can be difficult to keep up with the changes. A computer only runs one line at a time, t diagrams help us visualize one line at a time exactly what we’re doing. Whenever you call a new function, it is a new TDiagram.

Function Call

A function call is where we invoke the function.

You use the keyword function & then name it whatever you want

Always two parenthesis after & then the code is within two {}

A function does nothing until you call it.

You call it by Name();

Parenthesis is what calls the function, runs it.

Parameters

Inputs you provide to the function.

They are placed within the parenthesis in the function.

They can be named anything.

Put the input the function needs inside of the parenthesis when calling it.

(keyword) (name of the function)

function sayHello() {

console.log("hello");

}

sayHello()

(calling the function)

(parameters)

function banana(num) {

console.log(num);

}

banana("word")

(passing in the input the function needs)

// printing up to 255

function print255() {

//starting while loop off at 1

var num = 1;

//while number is less than or equal to 255, console.log

while(num <= 255) {

//logging num

console.log(num);

// num++

num = num + 1;

}

}

print255();

// Print sum 0-255

function sum() {

// making a variable named sum to add to

var sum = 0;

//start for loop off at 0(initialization); run until we hit 255(condition checking); adding one to num every time we go through the for loop(iteration)

for(var num = 0; num <=255; num++) {

//add num to sum

sum += num;

console.log("New number:", num, "Sum:", sum);

}

}

sum()