Lesson:

Loops

PW SKILLS

What we'll learn:

- For, while and do-while loop
- break and continue
- switch statement with & without break & with return

Loops are used in programming to repeatedly run a block of code.

Let us look at different types of loops we have in JavaScript

Loops:

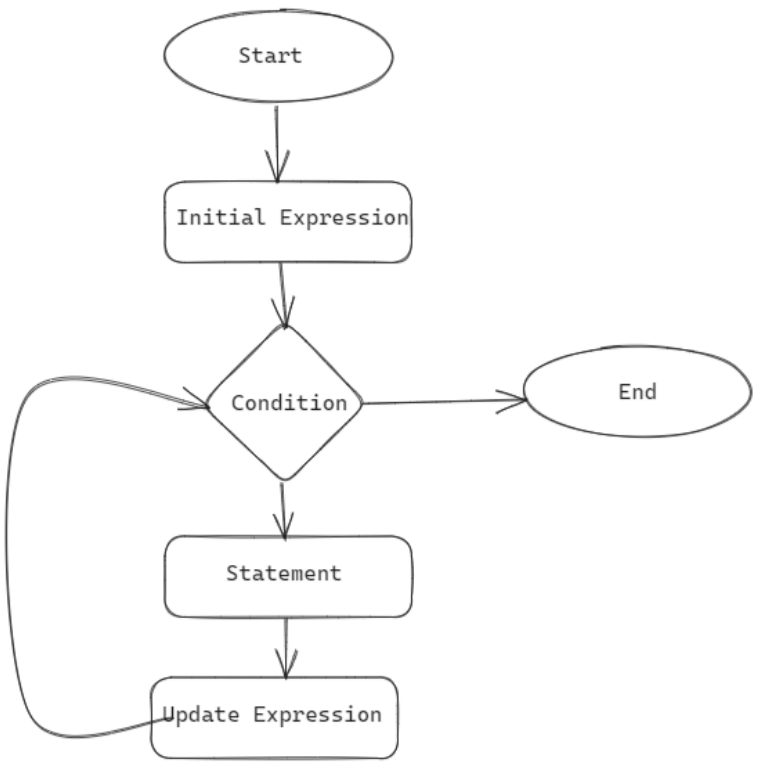
1. For Loop:

We use for loop to run a piece of code until the set condition turns false**.**

Syntax:

```javascript
for (initial expression; condition; update expression) {
  // for loop body
}
```

Understanding flow



- **Initial Expression** only ever executes once while initializing and declaring variables.
- The **Condition** is assessed.
- If the condition is **false,** the for loop will end.
- If the condition is **true,**
  - The for loop's code block will be executed
  - **Update expression** changes the **initial expression's** value.
- The condition is once more assessed and the loop continues until the condition becomes false.

Example:

```javascript
for (let i = 0; i < 3; i++)
{
    let name = "PW Skills";
    console.log(name);
}
// Output
/*
PW Skills
PW Skills
PW Skills
*/
```

In the above loop we run the loop until the value of i becomes false, and print PW Skills as long as the loop runs(i is true)

Example:

```javascript
for (let i = 2; i <= 20; i+=2) {
    console.log(i);
}

// Output
/*
2
```

```
4
8
10
12
14
16
18
20
*/
```

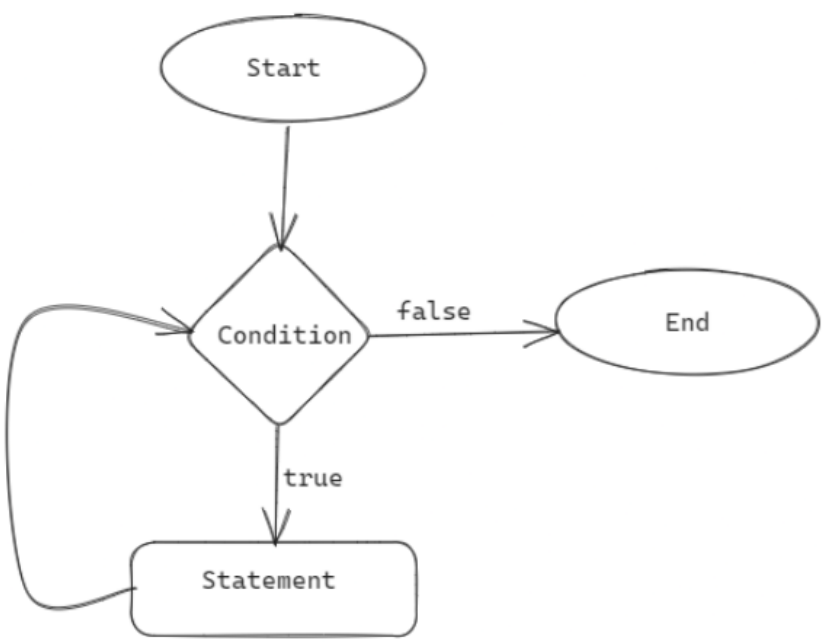
Here we are printing the even numbers till 20

2. **While Loop**: This loop keeps on running as long as the condition is true

Syntax:

```javascript
while (condition) {
  // body of the loop
}
```

Understanding Flow:



1. Loop starts
2. Condition is evaluated
    a. If the condition is false, the while loop will terminate.
    b. If the condition is true, the while loop's statements run, and continue until the condition evaluates to false.
3. Loop Ends

Example 1: To print numbers from 1 to 10 using while loop

```javascript
let i = 1, n = 10;
while (i <= n)
{
    console.log(i);
    i=i+1;
}

/*
Output
1
2
3
4
5
6
7
8
9
10
*/
```

Example 2 : To print numbers from 10 to 1 in decreasing order.

JavaScript

```
let i = 1, n = 10;
while (n>=i)
{
    console.log(n);
    n=n-1;
}

 /* Output
 10
 9
 8
 7
 6
 5
 4
 3
 2
 1
 */
```

Example 3: Draw following pattern using while loop,

Unset

```
*
**
***
****
*****
```

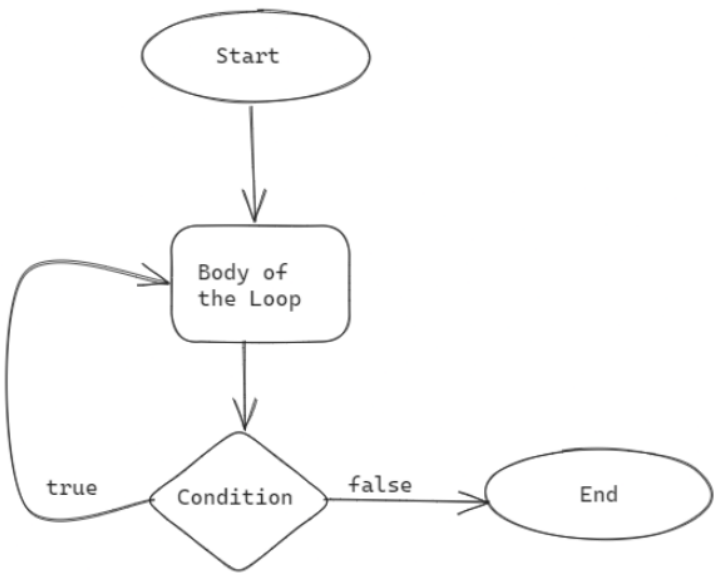
```javascript
let row = 1;
while (row <= 5) {
  let pattern = '';
  let col = 1;
  while (col <= row) {
    pattern += '*';
    col++;
  }
 console.log(pattern);
 row++;
    }
```

3. **Do While Loop:** Do while is similar to while loop with one difference that the 1st iteration runs always, and later iterations run after the condition evaluates to true.

Syntax:

```javascript
do {
  // body of the loop
} while (condition);
```

Understanding Flow:



1. Loop Starts
2. Loop Body is executed first.
3. Condition is accessed
   c. If Condition is false
   d. If the condition is true, the while loop's statements run, and continue until the condition evaluates to false.
4. Loop Ends

Example:

```javascript
let result = '';

let i = 0;
do {
  i = i + 1;
  result = result + i;
} while (i < 7);
console.log(result);

/* Output
1234567
*/
```

Here we write a program to print numbers from 1 to 7 in a line using do-while loop

For and while loop are used interchangeably, although they have different syntax they have equal capability, developer use different loops in different scenarios.

For vs While vs Do While

- Use a for loop when:
  - Used when you know the number of iterations or the range of values you want to iterate over.
  - The loop variable is automatically updated by the loop structure.
- Use a while loop when:
  - Scenarios where the number of iterations is not known in advance or when the termination condition is based on a dynamic condition.
  - You want to execute the code block only if the initial condition is true.
  - The loop may not need to run at all if the condition is initially false.
  - The loop termination depends on the condition being evaluated at the beginning of each iteration.

Use a do-while loop when:

- You want to execute the code block at least once, regardless of the initial condition.
- The loop termination depends on the condition being evaluated at the end of each iteration.

Break and Continue:

- **Break:**

We can terminate the loop explicitly using the break statement.

Example:

```javascript
for (let i = 0; i < 4; i++) {
  console.log(i);
  if (i == 2) {
    break;
  }
}
/* Output
0
1
2
*/
```

Here, we use an if statement inside the loop. If the current value of i becomes 2, the if statement will execute and the break statement will terminate the loop.

That is why we only see numbers till 2 in the output.

- **Continue:**

Continue statement skips the current iteration of loop, and moves on to the subsequent iteration.

Example:

```javascript
for (let i = 0; i < 20; i++) {
   if (i % 2 === 0) {
     continue;
   }
   console.log(i);
}
```

```
/* Output
1
3
5
7
9
11
13
15
17
19
*/
```

Here, the for loop iterates through the values from 0 to 20.

The remainder of the division of the current value of i by 2 is returned by the i%2 expression.

The continue statement, which skips the current iteration of the loop and moves to the iterator expression i++, is executed if the remainder is zero. If not, the value of i is output to the console.

switch statement with & without break & with return

- In the past we have looked at conditionals and loops, lets us now look at switch statement which is similar to conditionals
- Let's look at a real-life scenario, assume it's lunchtime and you walk to your favorite restaurant. The attendant offers you the menu.
- On the menu are different delicious items made for special people like you.
- You go through the menu and choose one or more meals from the menu and have yourself a good lunch.
- That is what switch statements help us do in JavaScript.

Syntax:

```javascript
switch (expression) {
```

```
  case value1:
    // code to execute;
    break;
  case value2:
    // code to execute;
    break;
  case value3:
    // code to execute;
    break;
  default:
  // default code;
}
```

- switch keyword defines a switch block.
- case keyword defines different case blocks, which will be executed only when case value matches with expression value. Case values can be a number, string or boolean etc.
- break keyword is used at the end of every case block to terminate switch case evaluation further.
- default keyword defines a default case block. When no case value is matched then the default block will be executed. It is not mandatory to include it.

Let us compare switch with if statement

JavaScript

```
switch (expression) {
   case value1: // if (expression === value1 then execute
this block)
   // code to execute;
   break;
   case value2: // if (expression === value2 then execute
this block)
   // code to execute;
   break;
```

```
  case value3: // if (expression === value3 then execute
this block)
  // code to execute;
  break;

  default: // if (expression === none of the previous
values matched execute this block)
 // default code;
}
```

Example:
Let's now implement a switch statement considering an example.

We represent our weekdays in both number notation and text notation like 1 for Sunday, 2 for Monday, and so on.

Let's take the number notation as input and provide text notation as output using the switch statement.

In this case, the only condition is that the number notation of the day must be between 1 to 7. If any other input is given then it will be considered invalid input.

Let's look at the code.

JavaScript

```
var day = 5;

switch (day) {
  case 1: // if (day === 1) then execute this block)
    console.log("Sunday");
    break;


  case 2: // if (day === 2) then execute this block)
    console.log("Monday");
    break;
```

```
  case 3: // if (day === 3) then execute this block)
    console.log("Tuesday");
    break;

  case 4: // if (day === 4) then execute this block)
    console.log("Wednesday");
    break;

  case 5: // if (day === 5) then execute this block)
    console.log("Thursday");
    break;
  case 6: // if (day === 6) then execute this block)
    console.log("Friday");
    break;

  case 7: // if (day === 7) then execute this block)
    console.log("Saturday");
    break;

  default: // if (expression === none of the previous
conditions then execute this block)
    console.log("Day doesn't exist");
}

// Output: Thursday
```

This code will output Thursday to the console because the value of the variable day is 5.

As we have passed the day to the switch statement, the day matches case 5 in the switch statement. When this case is executed, it will run the code block associated with it, which is console.log(Thursday).

The break statement at the end of the case ensures that the code execution exits the switch statement after the matching case has been executed, so the code in the other cases and the default block will not be executed.

In the above example, we have used numbers as case values, 1,2,3,4, but be careful if you use “1” as your case value it will be treated as a different case because “1” is a string and 1 is a number datatype and switch cases use strict comparison (===). The values must be of the same type to match. A strict comparison can only be true if the operands are of the same type.

```javascript
console.log(1 === '1') // false
```

What happens when we skip the break keyword?
In JavaScript, when you skip the break keyword in a switch statement, the program will continue executing the statements in the following case(s) without any further evaluation of the case conditions. This behavior is known as "fall-through."

Example:

```javascript
let color = "red";

switch (color) {
  case "red":
    console.log("The color is red.");
  case "blue":
    console.log("The color is blue.");
    break;
 case "green":
   console.log("The color is green.");
    break;
 default:
   console.log("The color is unknown.");
}
```

```
//Output
/*
In this case, if the value of color is "red," but the
program will print:

The color is red.
The color is blue.
*/
```

Since the break keyword is not used after the first case, the program will continue executing the statements in the subsequent cases. This is because JavaScript does not automatically exit the switch statement after a matching case unless a break statement is encountered.

- **Switch with return statement**

We can use the **return** keyword instead of **break** for every case. It will be helpful, if we are using switch statements inside the **function**, because it will serve two purposes, one to come out of the **switch** block (like a **break)** and at the same time it returns the result from the function.

Lets see one example using return statement,

JavaScript

```
let grade = 'B';

function getValue(grade){
   switch (grade) {
    case 'A':
       return "Excellent";
    case 'B':
       return "Average"
    case 'C':
       return "Below than average";
    default:
       return "No Grade";
   }
}
console.log(getValue(grade)); // Average
```

In the above example, the getValue function returns keywords “Excellent”, “Average”, “Below Average” based on grades A, B, C respectively. It uses switch statements and instead of the break keyword in every case we are returning value using the return keyword.