JS

{ JavaScript }

CRAFT KNOWLEDGE

Objectives

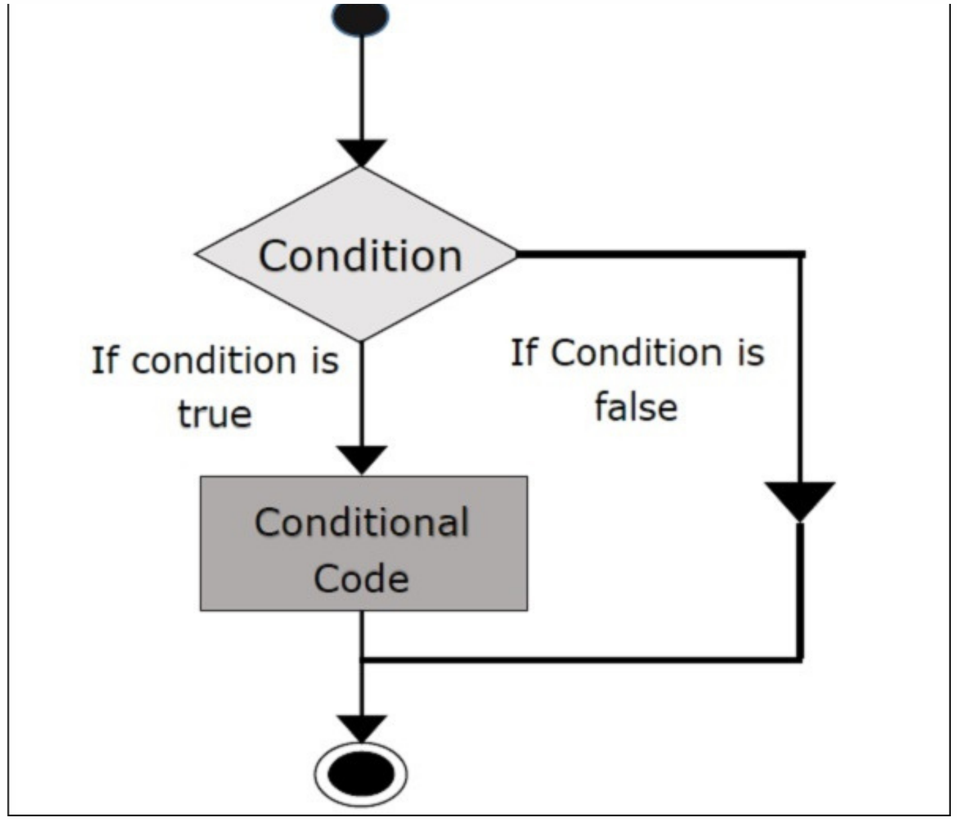
Controle Flow

- CONDITIONAL STATEMENT IN JAVASCRIPT
  - JAVASCRIPT IF CONDITIONAL
  - JAVASCRIPT IF, ELSE CONDITIONAL
  - JAVASCRIPT IF, ELSE IF, ELSE CONDITIONAL
  - JAVASCRIPT SWITCH CONDITIONAL
- TERNARY OPERTORS IN JAVASCRIPT
- ITERATION IN JAVASCRIPT
  - JAVASCRIPT FOR LOOP
  - JAVASCRIPT WHILE LOOP
  - JAVASCRIPT DO WHILE LOOP
- BREAK AND CONTINUE

Conditional statement in JavaScript

- Conditional statements are implemented when you need to perform different actions based on different conditions. And determine whether or not pieces of code can run.
- In JavaScript, the conditional statements that we have are
  - if.
  - if else.
  - if else if else.
  - switch statements.

if Statement

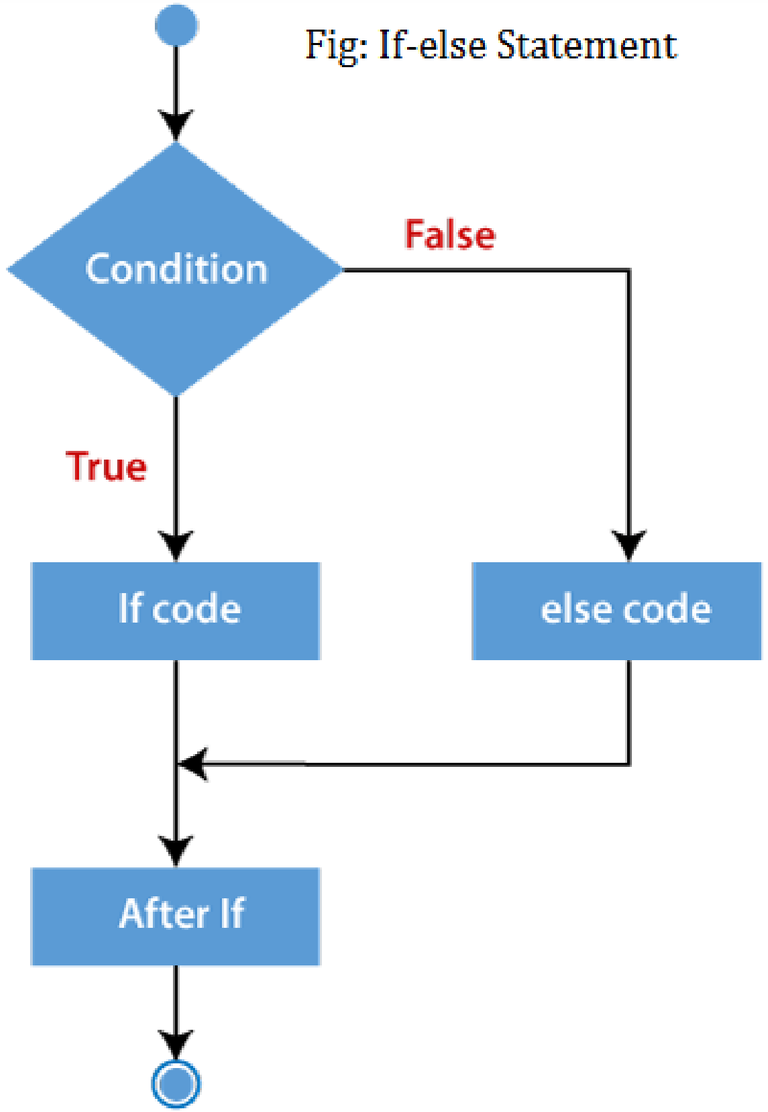


- The if statement is used when we want a block of code to be run as long as the condition it true.
- syntax

```
if (condition)
        {
            // statement
         }
```

- Example

```
if (4 * 4 = 16)
    {
        console.log('This will run!')
    }
```



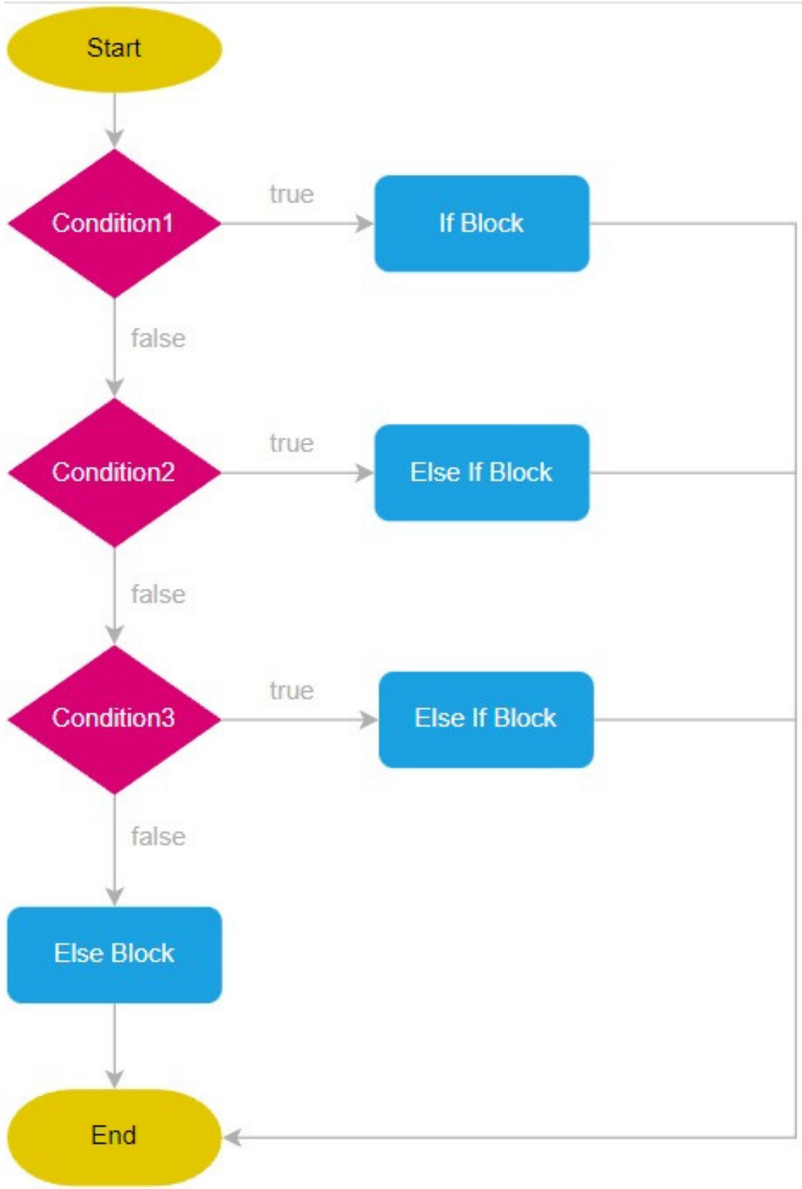
if else Statement

- The if statement executes a block if a condition is true. When the condition is false, it does nothing. But if you want to execute a statement if the condition is false, you can use an if...else statement.
- The else statement to specify a block of code to be executed if the condition is false.
- Multiple else block is NOT allowed.
- syntax

```
if (condition)
        {
            // statement
         }
else
       {
         // statement
       }
```

- Example

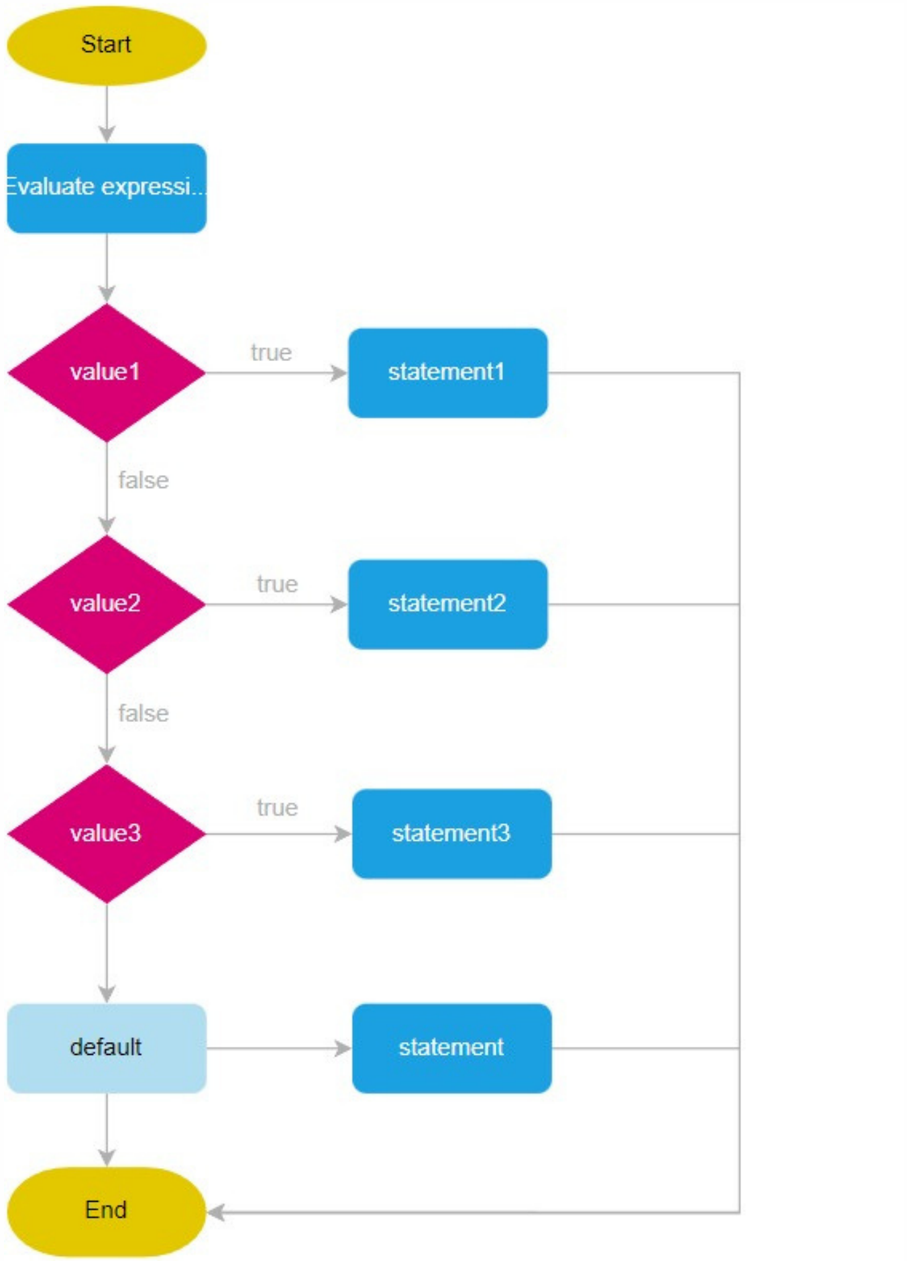
```
if (4 * 4 = 16){
     console.log('This will run!')
  }
else{
       console.log('This will run!')
     }
```



if, else if, else Statement

- To check multiple conditions and execute the corresponding block if a condition is true, we use the if...else...if statement like this:
- syntax

```
if (condition 1)
        {
          // statement
        }
else if (condition 2)
        {
          // statement
        }
else if (condition 3)
        {
          // statement
        }
        .
        .
        .
 else  {
          // statement
        }
```



Switch Statement

- The switch statement evaluates an expression, compares its result with case values, and executes the statement associated with the matching case value.
- syntax

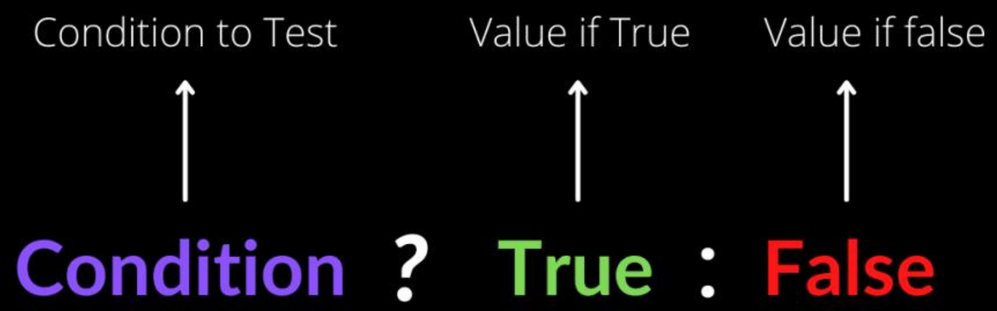
```
switch (expression)
   {

    case value1:
       statement1;
       break;
    case value2:
       statement1;
       break;
    case value2:
        statement1;
        break;
    default:
       statement3;

     }
```

How Switch Statement works

- First, evaluate the expression inside the parentheses after the switch keyword.
- Second, compare the result of the expression with the value1, value2, ... in the case branches from top to bottom. The switch statement uses the strict comparison (===).
- Third, execute the statement in the case branch where the result of the expression equals the value that follows the case keyword. then The break statement exits the switch statement.
- If the result of the expression does not strictly equal to any value, the switch statement will execute the statement in the default branch.
- That the switch statement will stop comparing the expression‘s result with the remaining case values as long as it finds a match.
- The switch statement is like the if...else...if statement. But it has more readable syntax



Ternary operators in JavaScript

- operator that takes three operands: a condition followed by a question mark (?), then an expression to execute if the condition is true followed by a colon (:), and finally the expression to execute if the condition is false.
- This operator is frequently used as an alternative to an if...else statement.
- syntax

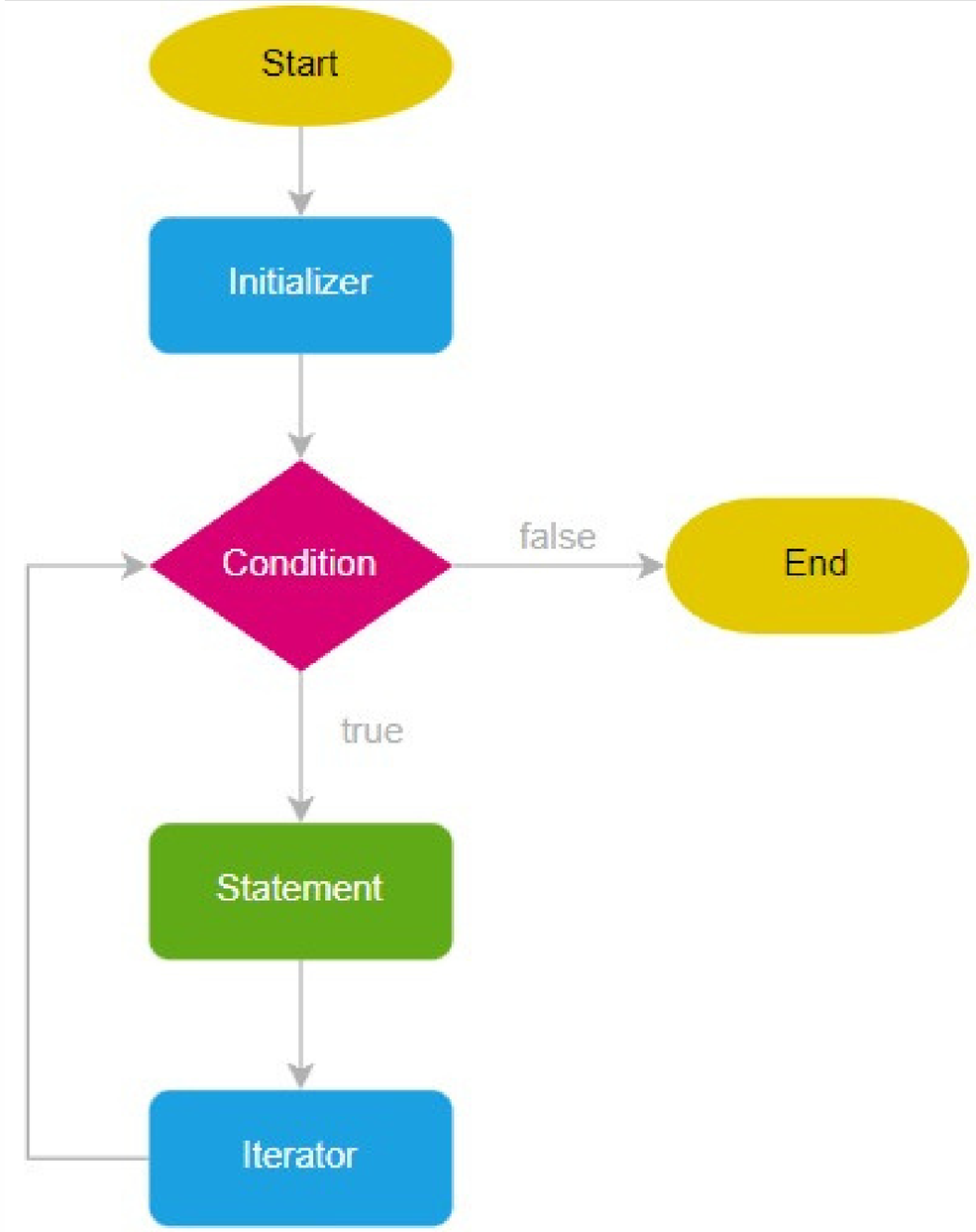
```
condition ? exprIfTrue : exprIfFalse
```

Example

```
const Drive_License = age >= 21 ? "Yes" : "No";
```

loop (Iteration) statement in JavaScript

- Loops offer a quick and easy way to do something repeatedly.
- In JavaScript, there are many different kinds of loops, but they all essentially do the same thing:
- The various loop mechanisms offer different ways to determine the start and end points of the loop.
  - for loop
  - while loop
  - do while loop



JavaScript for loop

- Loops are handy, if you want to run the same code over and over again, each time with a different value.
- The for statement creates a loop with 3 optional expressions:

syntax:

```
for (initializer; condition; iterator)
    {
    // statements
    }
```

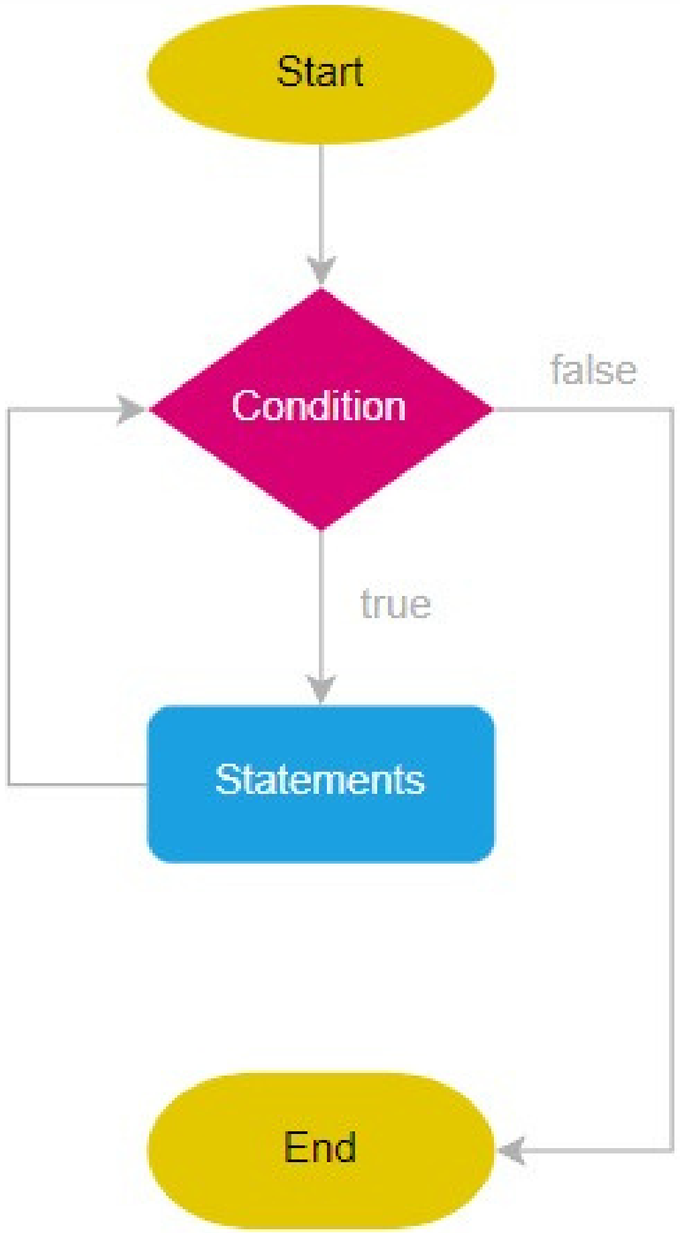
1) **initializer**: The for statement executes the initializer only once the loop starts. Typically, you declare and initialize a local loop variable in the initializer.

2) **condition**: The condition is a boolean expression that determines whether the for should execute the next iteration. The for statement evaluates the condition before each iteration. If the condition is true, it executes the next iteration. Otherwise, it'll end the loop.

3) **iterator**: The for statement executes the iterator after each iteration.

Example

```
for (let i=0; i<5; i++)
    {
      console.log("hi");
    }
```



JavaScript while loop

- The JavaScript while statement creates a loop that executes a block as long as a condition evaluates to true.

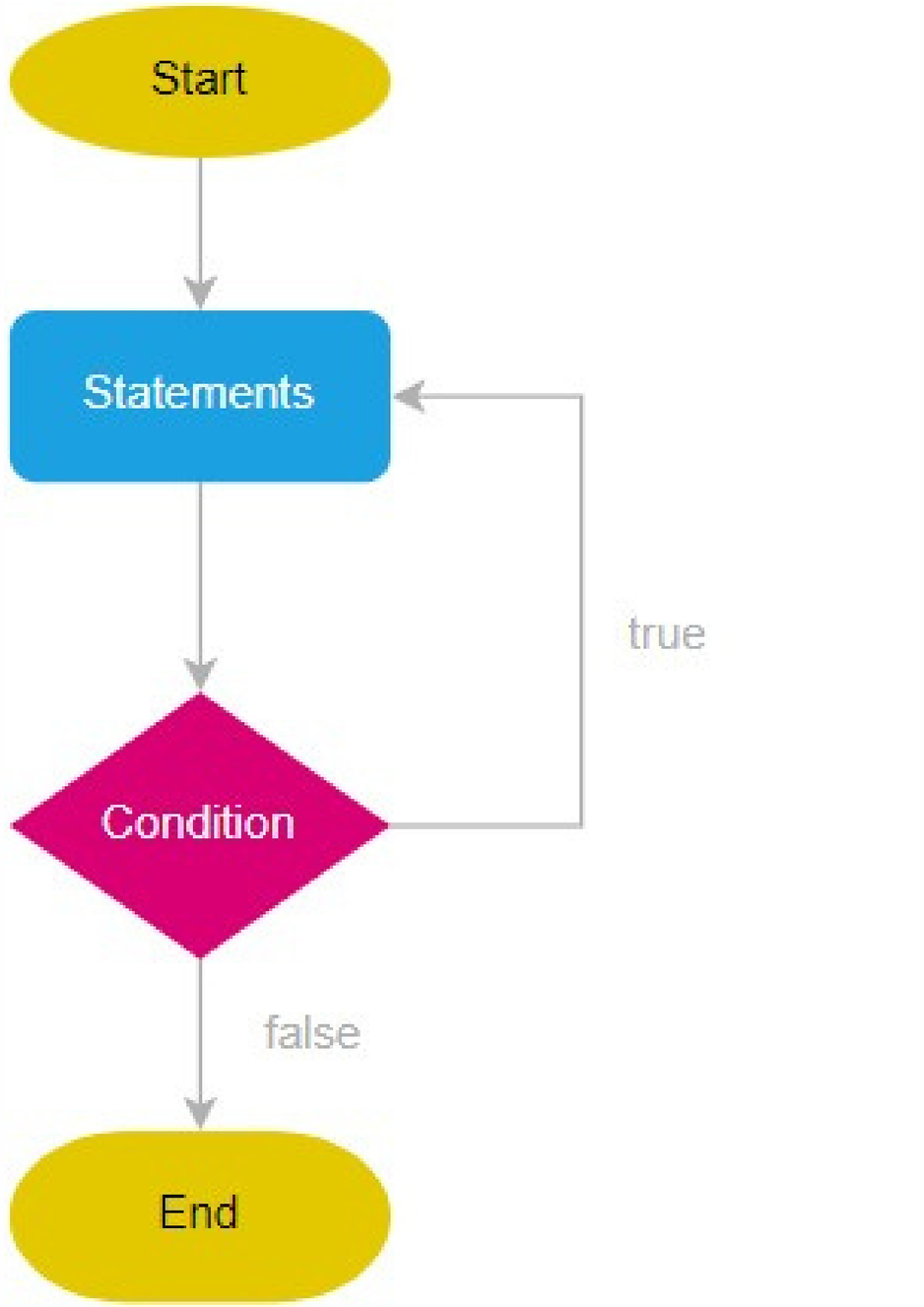
syntax:

```
while (expression)
  {
    // statement
  }
```

- The while statement evaluates the expression **before each iteration** of the loop.
- If the expression **evaluates to true,** the while **statement executes** the statement. Otherwise, the while loop exits.
- Because the while loop evaluates the expression before each iteration, it is known as a **pretest loop.**
- If the expression evaluates to **false** before the **loop enters**, the while loop will never execute.

Example

```
let i=0
while(i<5)
    {
      console.log("hi");
      i++
    }
```



JavaScript do...while Loop

- The do...while loop statement creates a loop that executes a block until a condition evaluates to false.

syntax:

```
do {
    statement;
} while(expression);
```

- The do-while loop always executes the statement at least once before evaluating the expression.
- Because the do...while loop evaluates expression after each iteration, it's often referred to as a **post-test loop**.
- Inside the loop body, you need to make changes to some variables to ensure that the expression is false after some iterations. Otherwise, you'll have an indefinite loop.

Example

```
let i=0
do {
    console.log("hi");
    i++
}while(i<5);
```

Break and Continue

- The **break** statement "jumps out" of a loop.

  Example

```
// program to print the value of i
for (let i = 1; i <= 5; i++) {
        if (i == 3) {
            break;
        }
        console.log(i);
        }
```

- when i is equal to 3, the break statement terminates the loop. Hence, the output doesn't include values greater than or equal to 3.
  - Output
    - 1
    - 2

- Note: The break statement is almost always used with decision-making statements.

- The **continue** statement: is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.
  Example.

```
for (let i = 1; i <= 5; i++) {
    // condition to continue
    if (i == 3){
        continue;
        }
    console.log(i);
        }
```

- This means
  - When i is equal to 3, the continue statement skips the third iteration.
  - Then, i becomes 4 and the test condition and continue statement is evaluated again.
  - Hence, 4 and 5 are printed in the next two iterations.
    - Output
      - 1
      - 2
      - 4
      - 5
- Note: The continue statement is almost always used with decision making statements.