

Control Flow

By CODEMIND Technology

Contact us +91 96650 44698

Control Flow and Variable Scopes

- Understanding with real word example
 - Types of Statements
 - Conditional or Selection Statement
 - Looping or Iterative statements
 - Jump Statements
 - Assignments
 - Variable Scopes: Global, Local and Block scope
 - var, let and const
-

Control Flow

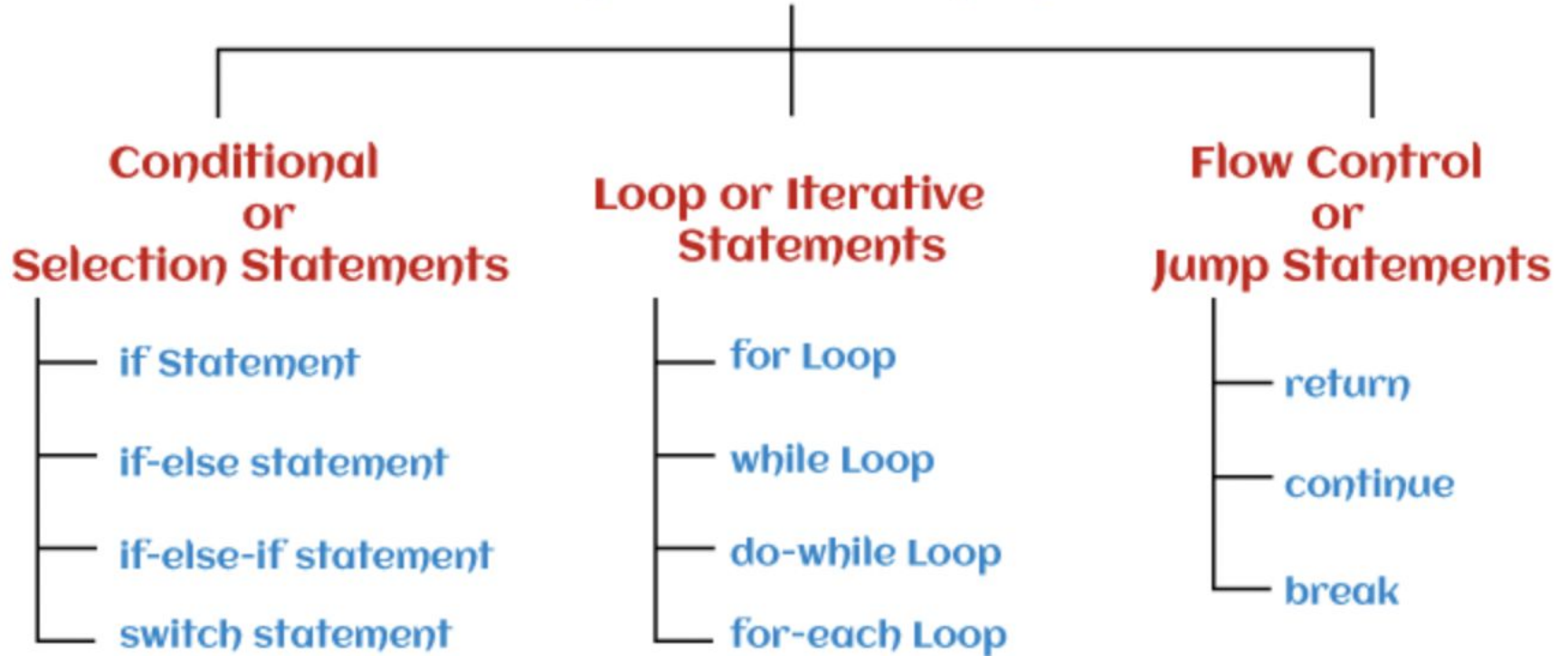
General word example: Mumbai to Pune

Control flow in JavaScript is how our JS engine runs code from top to bottom.

It starts from the first line and ends at the last line, unless it hits any statement that changes the control flow of the program such as loops, conditionals, or functions

Control Flow statements

Control Statement



if statement: Syntax and Flow chart

Condition can be checked using operators like ==, ===, !=, <, <=, >, &&, || and many more

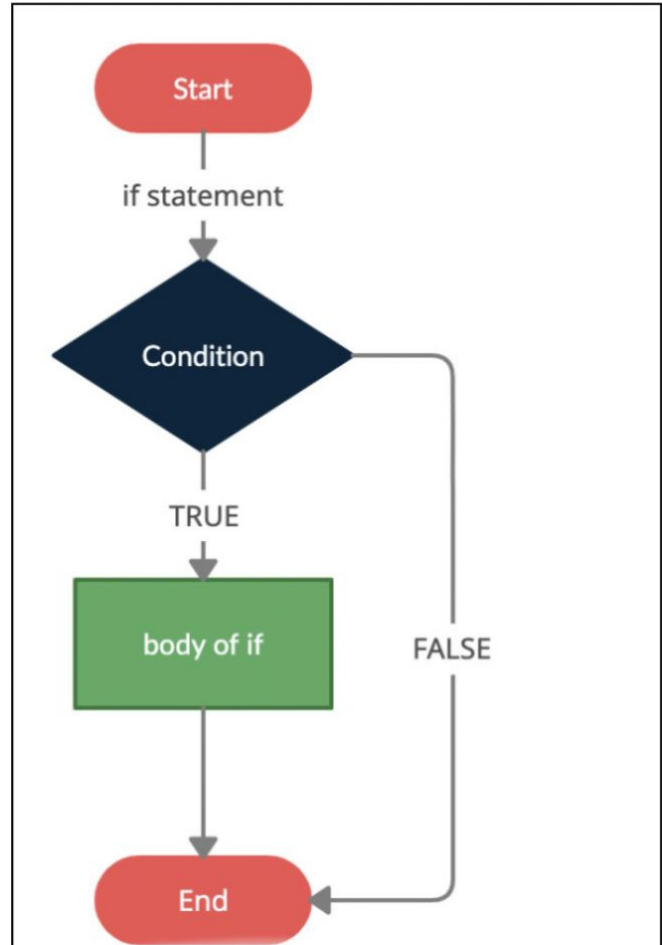
Syntax:

Condition
Any expression that evaluates to true or false

```
if (condition) {  
    statement  
    statement  
    ...  
}  
following_statement
```


True branch
This is executed if the condition is true

Flow Chart



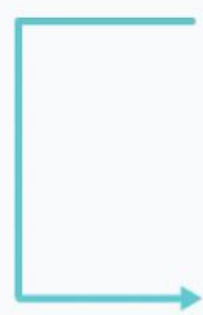
Conditional execution with true and false

Condition is true



```
let number = 2;  
if (number > 0) {  
    // code  
}  
  
//code after if
```

Condition is false



```
let number = -2;  
if (number > 0) {  
    // code  
}  
  
//code after if
```

if statement

- If stmt without { } braces
 - In this case just one line is allowed inside if stmt body
 - Ex →
- Nested if stmt: We can write if statement inside if statement

Assignment: Please make sure to write function for each step

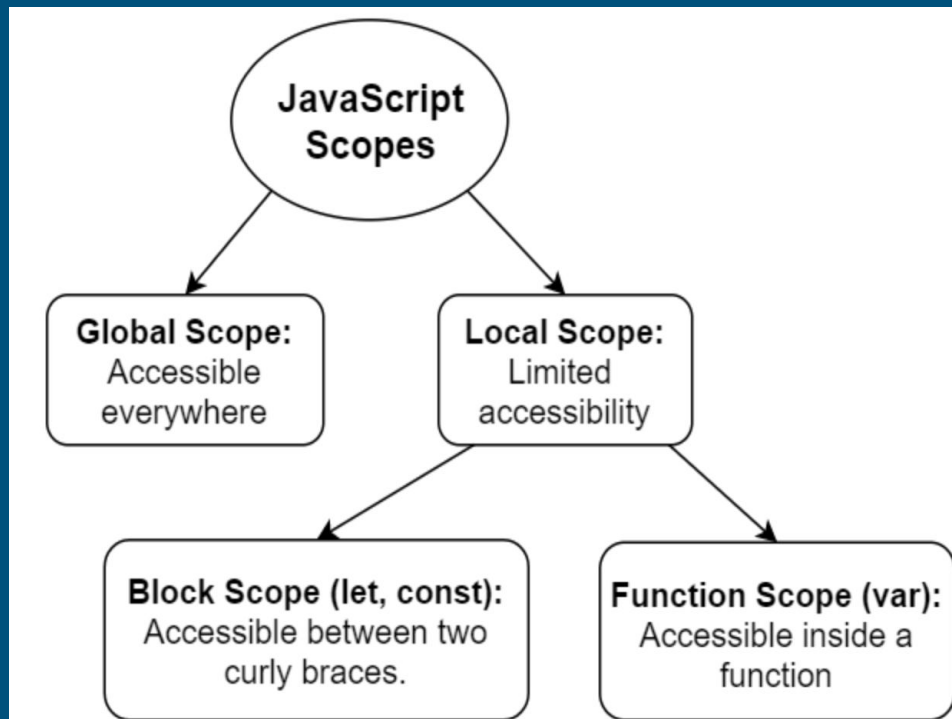
1. Create a function to check passed argument number value is even or odd and return the result as EVEN or ODD. Ex. 45, 70, 67, 98
2. Check if person is eligible for vote or not Ex→ age: 18, 20, 17, 40
3. Check if string contains more than 10 character or not Ex → “JavaScript - ES6”
4. Check if string starts with “Java” Ex→ “JavaScript Language”
5. Check is number even Ex → 2022
6. Check is number odd Ex → 123

Note: First 3 step using the normal function and remaining please define Function expression

Variable Scopes. → Gali ka Gunda

Scope determines the visibility and accessibility of a variable. JS has three scopes:

- The global Scope
- The Local Scope
 - Function Scope
 - Block Scope (started from ES6)



The global scope

- When the JS engine executes a script, it creates a global execution context.
- It also assigns variables that you declare outside of functions to the global execution context. These variables are in the global scope. They are also known as global variables.

The variable `message` is global-scoped. It can be accessible everywhere in the script.

```
var message = 'Hi';
```

Global Execution Context

Execution Phase (Web Browser)

Global Object: window

this: window

message: 'hi'

The Local Scope

- The variables that you declare inside a function are local to the function. They are called local variables.
- When the JavaScript engine executes the say() function, it creates a function execution context. The variable message declared inside the say() function is bound to the function execution context of the function, not the global execution context.

Explained in next sheet.

Local Scope

```
var message = 'Hi';

function say() {
  var message = 'Hello';
  console.log('message');
}

say();
console.log(message);
```

Global Execution Context

Execution Phase (Web Browser)

Global Object: window

this: window

message: 'Hi'

say: function(){...}



Function Execution Context

Execution Phase

Global Object: arguments

this: window

message: 'Hello'

Block Scope

ES6 provides the `let` and `const` keywords that allow you to declare variables in block scope.

What is block ?

Generally, whenever you see curly brackets { }, it is a block.

It can be the area within the if, else, switch conditions or for, do while, and while loops.

```
function say(message) {  
    if(!message) {  
        let greeting = 'Hello'; // block scope  
        console.log(greeting);  
    }  
    // say it again ?  
    console.log(greeting); // ReferenceError  
}  
  
say();
```

const introduced in ES6

- It cannot be declared without initialization.
- `const PI; // Not allowed`

Note: The `const` keyword specifies that a variable's value is constant and tells the compiler to prevent the programmer from modifying it

Diff. between var, let, const

var	let	const
It is available right from the beginning when the JavaScript was introduced.	It is a new way to declare variables in JavaScript, starting from ES6.	const is used to store a value that will not be changed throughout the execution of the script. It is also introduced recently in ES6.
It has a global/function scope.	It has block scope.	It also has block scope.
Can be updated or re-declared in its scope.	We can't re-declare them.	const represents a constant value, so it can't be updated or re-declared.

If else statement

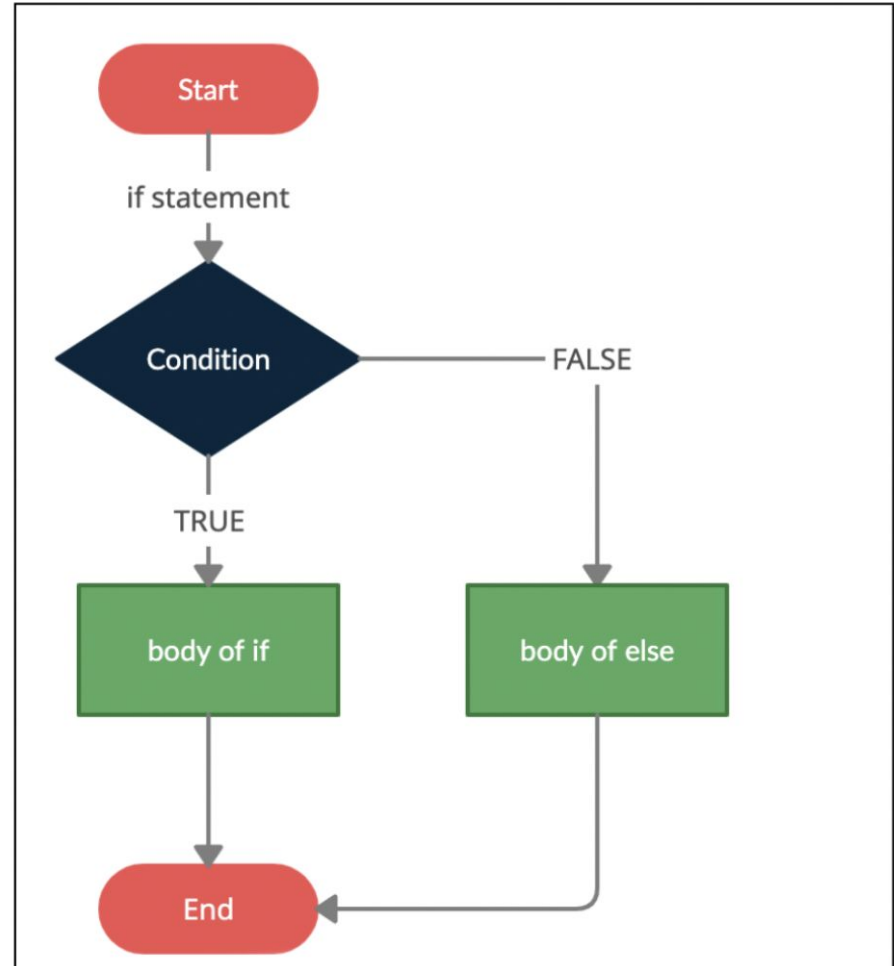
Syntax

```
if (condition) {  
    statement  
    statement  
    ...  
} else {  
    statement  
    statement  
    ...  
}  
following_statement
```

True branch
This is executed if the condition is true

False branch
This is executed if the condition is false

Flow Chart



Example with both the condition flow

Condition is true

```
let number = 2;  
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if
```

The diagram illustrates the execution flow for the 'Condition is true' scenario. A teal arrow originates from the 'if' statement, points to the first code block ('// code'), and then continues down to the '// code after if' line, bypassing the 'else' block.

Condition is false

```
let number = -2;  
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if
```

The diagram illustrates the execution flow for the 'Condition is false' scenario. A teal arrow originates from the 'if' statement, points to the second code block ('// code' under the 'else' block), and then continues down to the '// code after if' line, bypassing the first code block.

Assignment 02: if else stmt, Pls have the separate function for each main step

1. A Function to check the number is even or odd using if statement & return result
 - 1.1. Ex → 2, 45, null, 13, 0
 - 1.2. Please call the same function for all numbers and log result on console
2. Design a grade system with following steps
 - 2.1. score less than 35 then Failed
 - 2.2. score more than equal to 35 then Passed
 - 2.3. score is greater than equal 35 and less than 60 then GRADE is 'C'
 - 2.4. score is greater than equal to 60 and less than 75 then GRADE is 'B'
 - 2.5. score is greater than equal to 75 and less than equal to 90 then GRADE is 'A'
 - 2.6. score is greater than equal to 90 and less than equal to 100 then GRADE is 'A+'
 - 2.7. Make sure to handle unhappy path scenarios like
 - 2.7.1. invalid score like -ve score or more than 100
 - 2.7.2. what if invalid input like "Fourty Five Score". (Hint → Return with invalid input)
3. Write a function expression to check the type of given argument
 - 3.1. Ex → "50", 100, 60.89, null, undefined, NaN, "Hello"

Ternary Operator or Conditional Operator

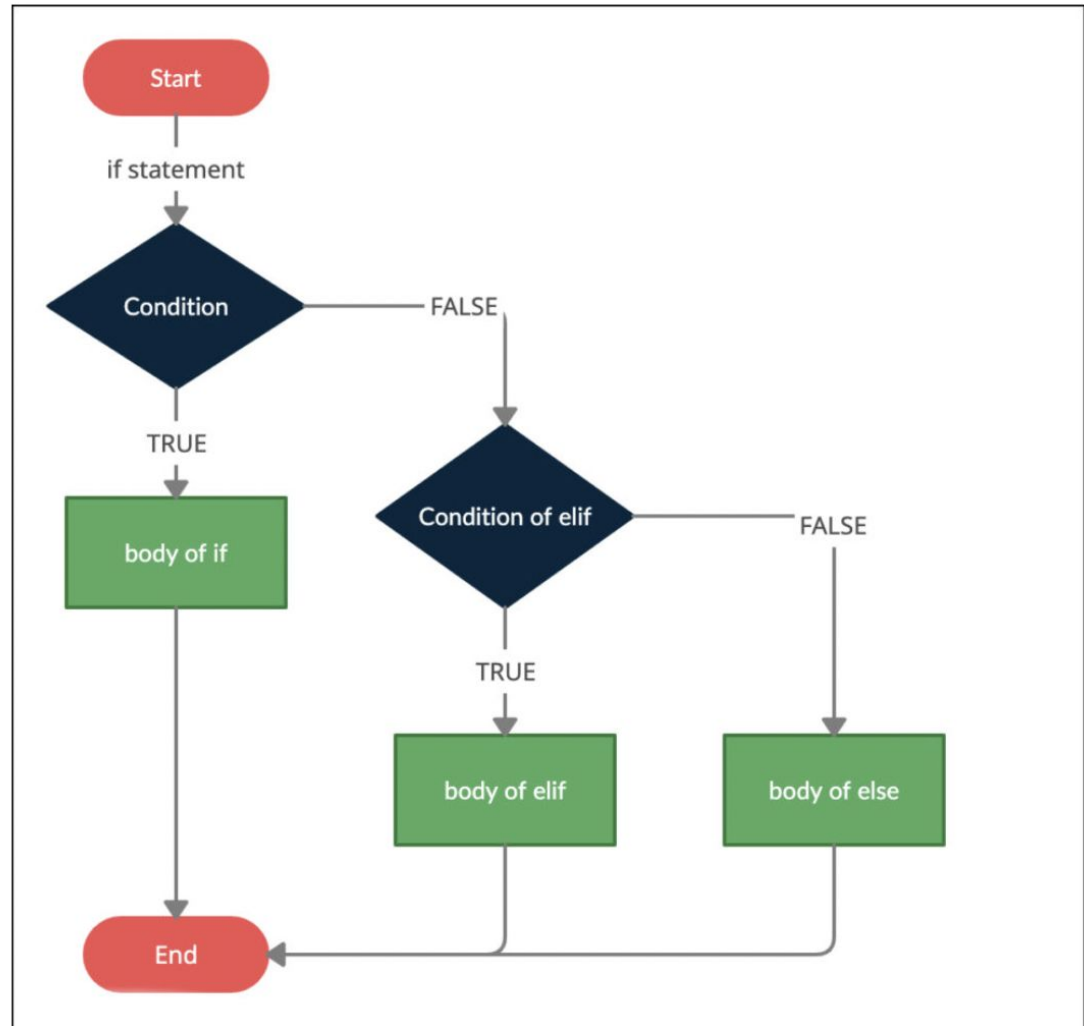
Ternary operator is similar to if else statement

```
variable <- if (condition) Statement else Statement;
```

True branch
Execute this if the condition is true

False branch
Execute this if the condition is false

If else if else statement



Syntax

```
if (condition) {  
    statement  
    statement  
    ...  
} else if (condition) {  
    statement  
    statement  
    ...  
} else {  
    statement  
    statement  
    ...  
}  
following_statement
```

First condition
This is executed if the first condition is true


New condition
A new condition to test if previous condition isn't true

False branch
This is executed if none of the conditions are true

Example with all possible condition execution

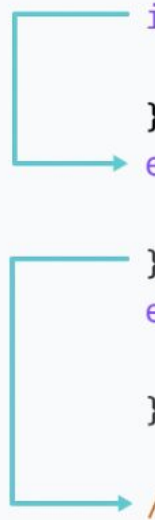
1st Condition is true

```
let number = 2;  
if (number > 0) {  
  // code  
}  
else if (number == 0){  
  // code  
}  
else {  
  //code  
}  
//code after if
```



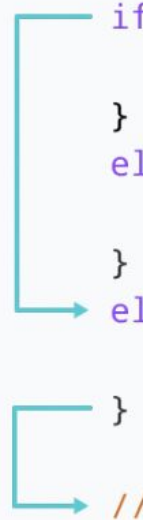
2nd Condition is true

```
let number = 0;  
if (number > 0) {  
  // code  
}  
else if (number == 0){  
  // code  
}  
else {  
  //code  
}  
//code after if
```



All Conditions are false

```
let number = -2;  
if (number > 0) {  
  // code  
}  
else if (number == 0){  
  // code  
}  
else {  
  //code  
}  
//code after if
```



Assignment 03:

1. Please design a marriage eligibility checker system to decide whether he or —she is eligible for marriage or not based on two input values gender and age

1.1 “Male”, 17

1.2 “Male”, 25

1.3 “Female”, 28

1.4 “Female”, 16

1.5 “Other”, 35

1.6 “Other”, 41

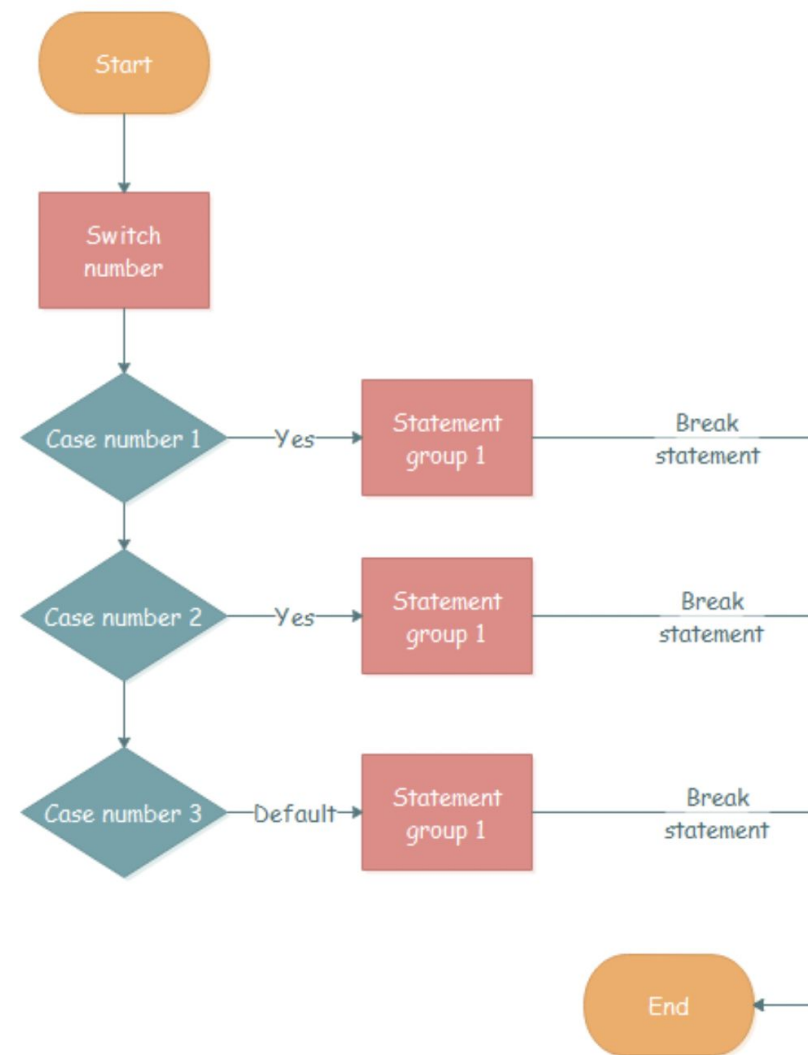
Note: Consider the age criteria defined by our Indian constitution.

Male ≥ 21

Female ≥ 18

Switch case statement

- A switch stmt can replace multiple if checks
- It gives a more descriptive way to compare a value With multiple variants
- The switch has one or more case blocks and an optional default
- Break is imp, otherwise will check all conditions



Syntax

```
                                EXPRESSION
switch ( $variable )
{
    case 0:
        //code;
        break;
    case 1:
        //code;
        break;
    case 2:
        //code;
        break;
    default:
        //code;
}
```

It is important to use **break**;
at the end of each case statement.
Otherwise the following statements
will all be executed!

Statement following the keyword
default: will only be executed if
no other cases have been matched.

If **\$variable** is 0 (case: 0) that code
will be executed. If none of the cases
match, default: code will be executed.

Assignment

Given a day in number and then make

Make a decision and log on console

- Week day
- Working or non working day

1	→	Monday
2	→	Tuesday
3	→	Wednesday
4	→	Thursday
5	→	Friday
6	→	Saturday
7	→	Sunday

Tricky Points

Case statements without break statement

Different values (types) that we can pass in switch()

Switch case without default

Assignment 04

1. Given a month number then log the name of the month

1 --- January

2 --- February

.....

12 --- December

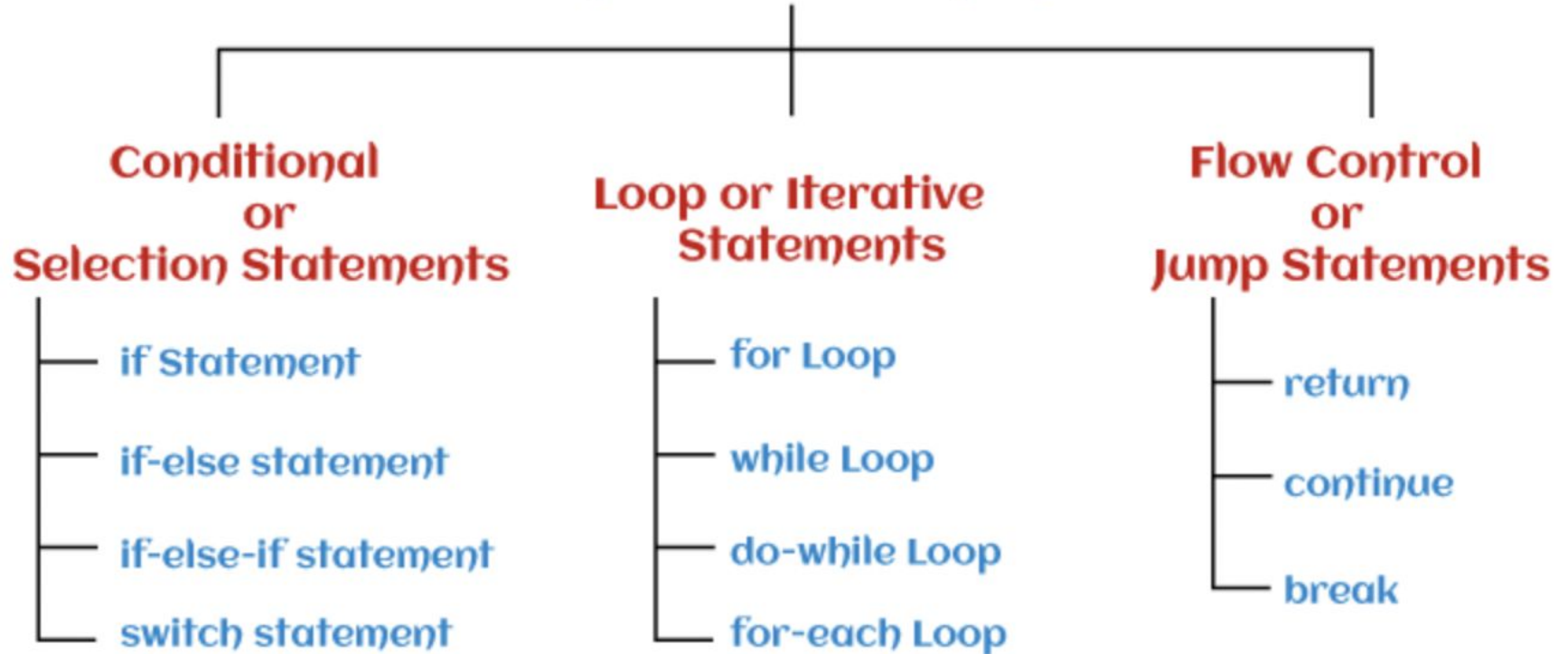
Note: Handle the invalid input value correctly

Can you please write what are the different invalid input values ? Please mention few.

Think what is the problem in case you would implement solution using multiple if checks?

Control Flow statements

Control Statement



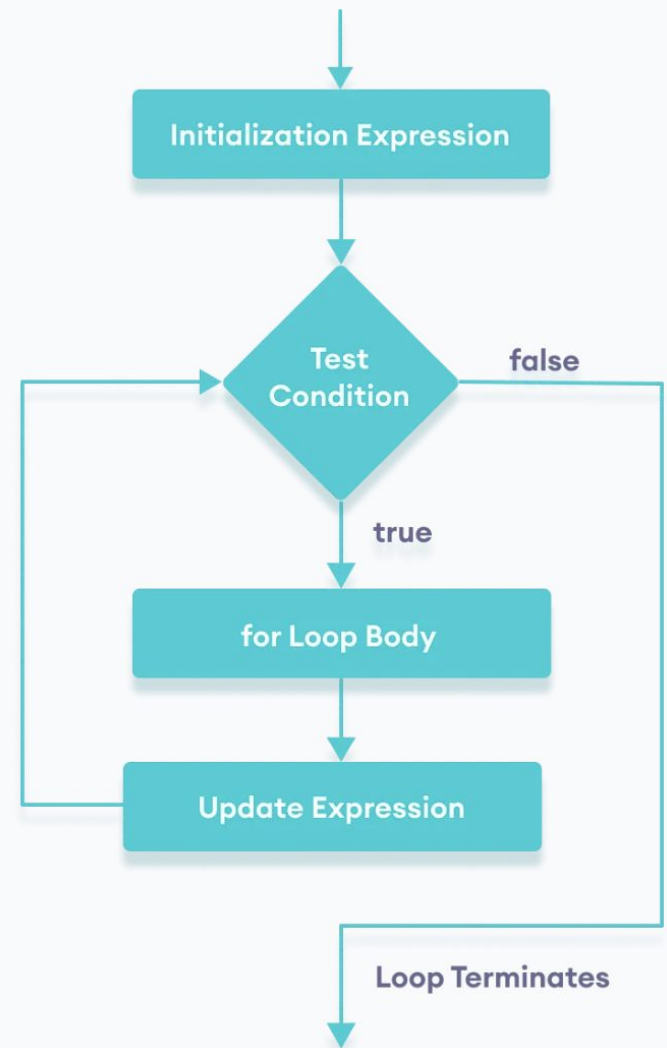
For loop flow chart

Loops are used to repeat a block of code

Ex → print numbers from 1 to 10

The syntax of the `for` loop is:

```
for (initialExpression; condition; updateExpression) {  
    // for loop body  
}
```

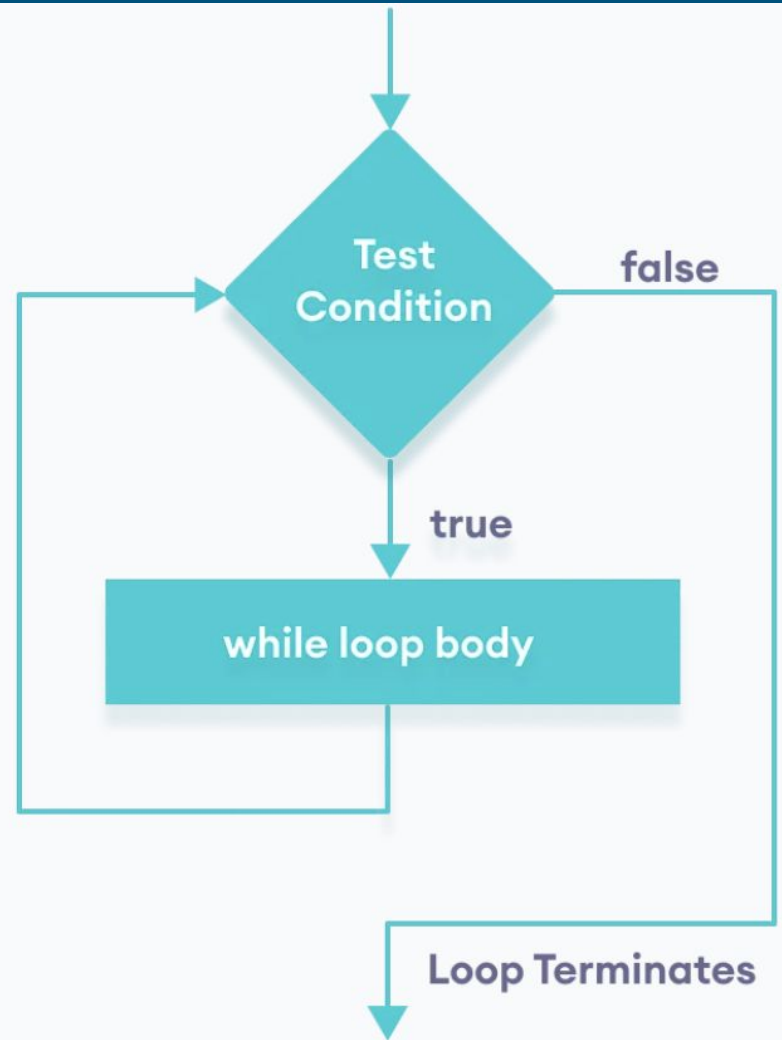


While loop

Ex → print numbers from 1 to 10

The syntax of the `while` loop is:

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



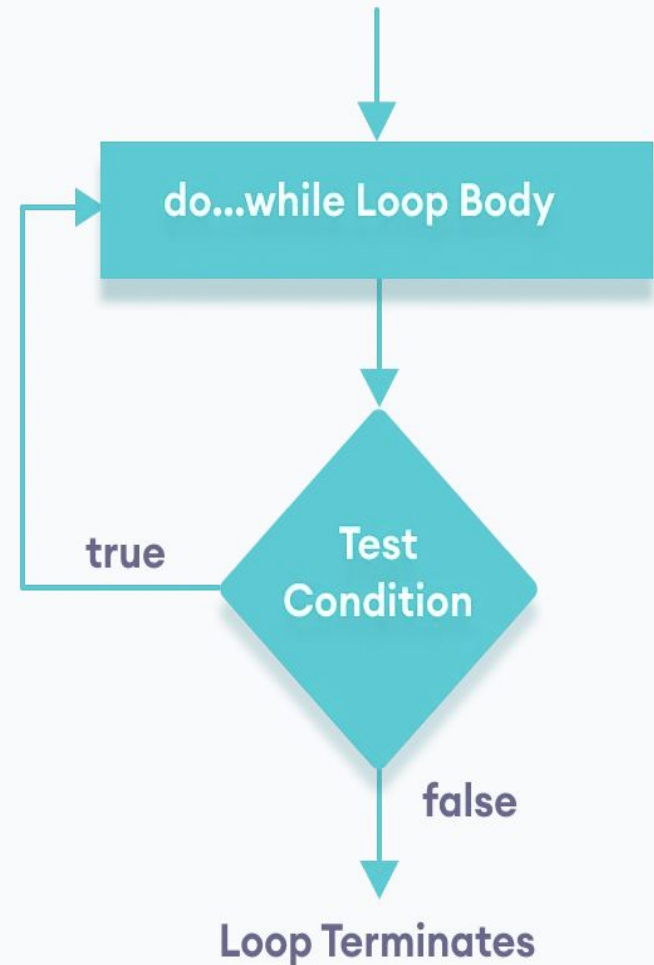
Assignments

1. WAP to print numbers from 10 to 1
2. WAP to find first 15 odd numbers
3. WAP to find all prime numbers from 1 to 50
4. WAP to print numbers from 30 to 20

do while loop

The syntax of `do...while` loop is:

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



Infinite loop

If **the condition** of a loop is always `true`, the loop runs for infinite times (until the memory is full). For example,

```
// infinite while loop
while(true){
    // body of loop
}
```

Here is an example of an infinite `do...while` loop.

```
// infinite do...while loop
const count = 1;
do {
    // body of loop
} while(count == 1)
```

Diff. between for loop and while loop

A `for` loop is usually used when the number of iterations is known. For example,

```
// this loop is iterated 5 times
for (let i = 1; i <=5; ++i) {
    // body of loop
}
```

And `while` and `do...while` loops are usually used when the number of iterations are unknown. For example,

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

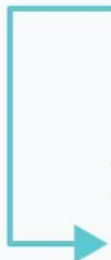
Jump statements

'break' statement is used to break the loop immediately

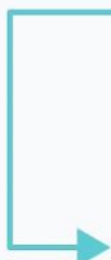
Example 1: break with for Loop

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

```
for (init; condition; update) {
  // code
  if (condition to break) {
    break;
  }
  // code
}
```

A teal line starts from the 'break;' statement, goes left, then down, and finally right as an arrow pointing to the closing brace of the for loop, indicating an immediate exit from the loop.

```
while (condition) {
  // code
  if (condition to break) {
    break;
  }
  // code
}
```

A teal line starts from the 'break;' statement, goes left, then down, and finally right as an arrow pointing to the closing brace of the while loop, indicating an immediate exit from the loop.

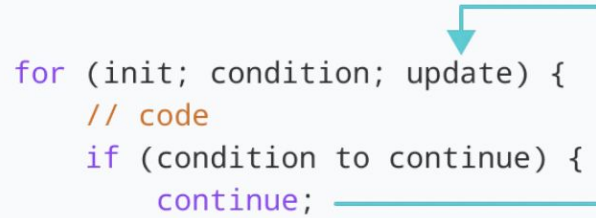
'continue' statement

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.

```
// program to print the value of i
for (let i = 1; i <= 5; i++) {

    // condition to continue
    if (i == 3) {
        continue;
    }

    console.log(i);
}
```



```
for (init; condition; update) {
    // code
    if (condition to continue) {
        continue;
    }
    // code
}

-----

while (condition) {
    // code
    if (condition to continue) {
        continue;
    }
    // code
}
```

Assignment:

1. Write a function to find vowels from the given string. Ex → 'I love JavaScript'
2. Write a function expression to sum all numbers from 1 to 10
3. Write a function to print table for 5, 7
4. Write a function to get the sum of square numbers from 1 to 5.

Ex→ $1*1 + 2*2 + 3*3 + 4*4 + 5*5$

5. WAF to print string's even index character Ex → "Hard work always pays back"
 - 5.1. Log on console only even index characters
 - 5.2. Log on console odd characters but make sure not consider " " empty spaces