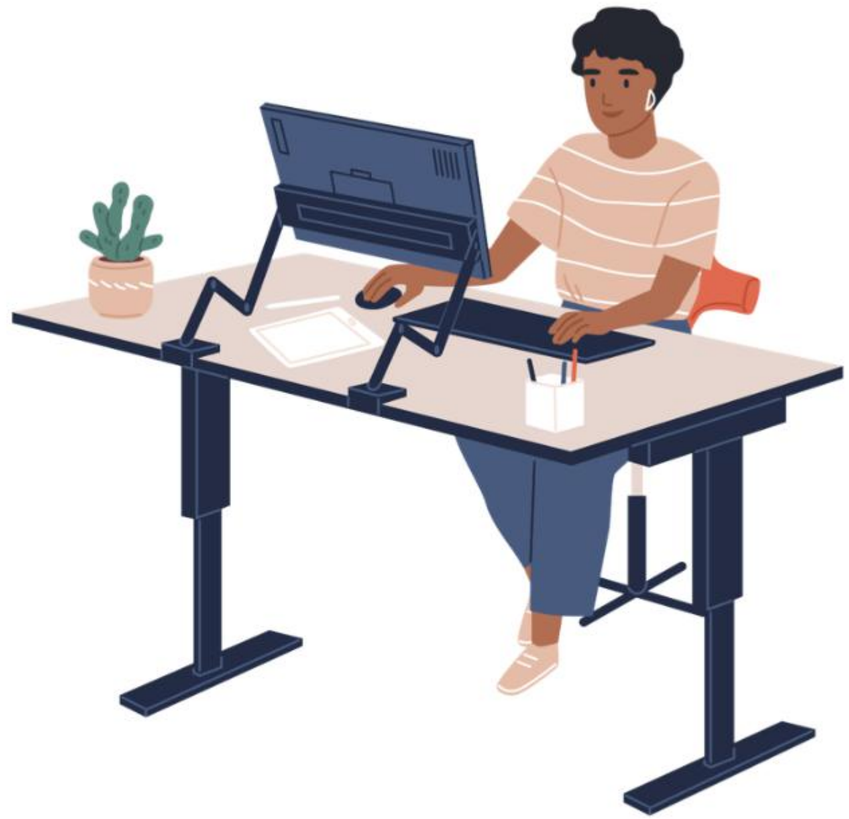
Slide Note

Learning Consolidation

Getting Started With JavaScript



Menu

00:01:00:00:34

01/ 34

Slide Note

In this Sprint, you learned to:

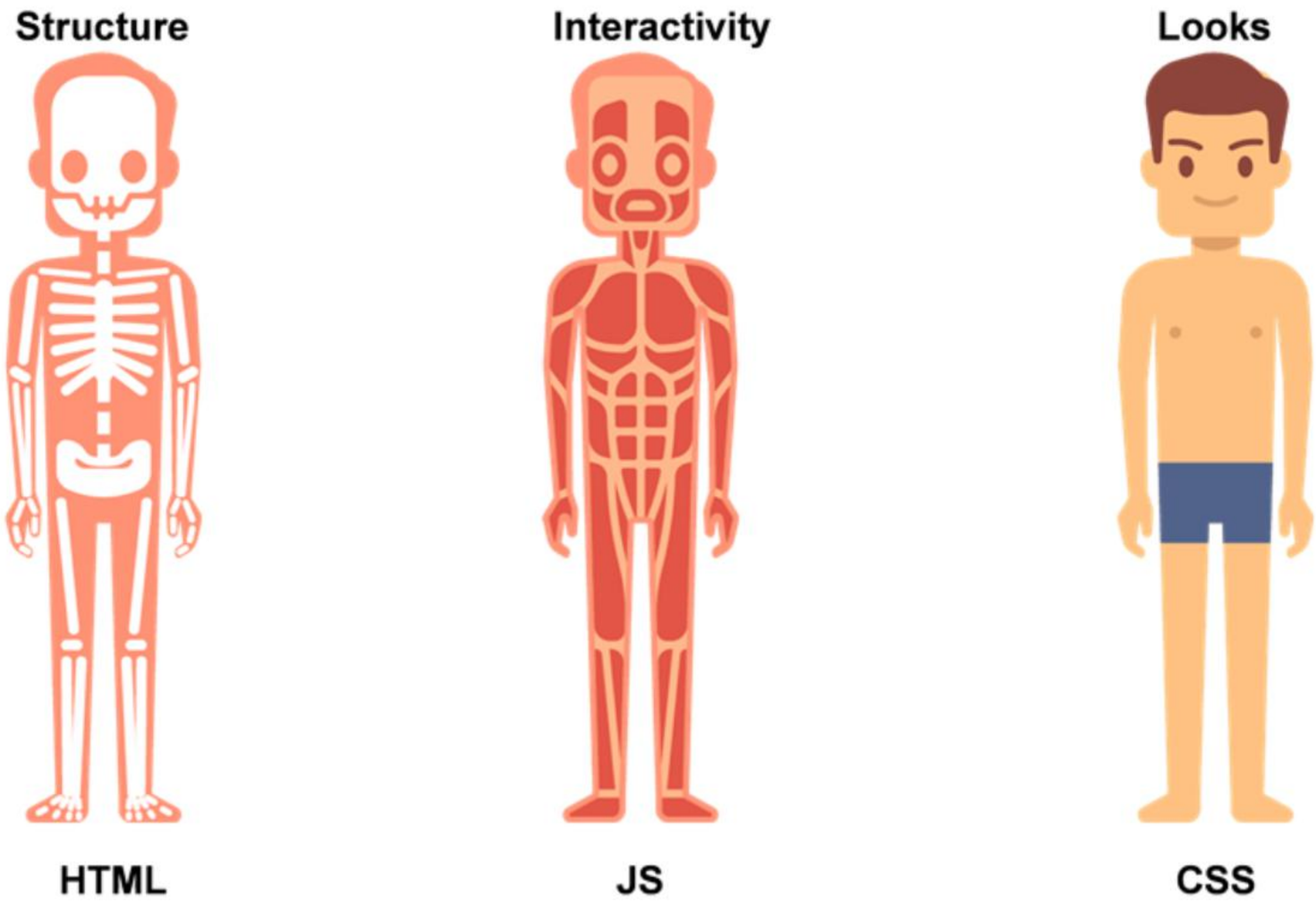
- Write Hello World program in JavaScript
- Define datatypes and literals
- Use variables and constants to store data
- Define template literals
- Explain `typeof` operator
- Implement conditional constructs for writing decision making code
- Implement loop constructions for writing iterative programs

Slide Note

While creating a dynamic or interactive web page, following tasks need to be performed

1. Structure web page: This is done with the help of HTML
2. Style web page: This is done with the help of CSS
3. Make web page interactive: This is done with the help of JavaScript – the scripting language

What Is JavaScript?



Slide Note

The two programming languages, Java and JavaScript have very different syntax, semantics, and use.

JavaScript - Introduction

- JavaScript is a tool that allows developers to write programs and make a page interactive.
  - Helps to modify content as well as styling of the web page
  - Helps to provide response to user's interaction
- JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled scripting language.
- It is popular as the scripting language for Web pages, however, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat
- **Do not confuse JavaScript with the Java programming language.**
- JavaScript engines embedded within the browser interprets JavaScript.

Slide Note

Executing Statements

- To process an instruction, an interpreter needs to know:
  - Data type of data to be processed:
    - ❖ JavaScript supports primitive and object type values
    - ❖ Primitives are immutable
    - ❖ Objects are values referred by an identifier
  - Operator to determine operation to be performed
  - How to consume results of operations:
    - ❖ The results can either be outputted or stored for later use

Slide Note

JavaScript Primitive Datatypes

Data Types	Description	Example
String	Represents textual data within single or double quotes	'hello', "hello world!" etc
Number	An integer or a floating-point number	3, 3.234, 3e-2 etc.
BigInt	An integer with arbitrary precision	900719925124740999n , 1n etc.
Boolean	Any of two values: true or false	true and false
undefined	A data type whose variable is not initialized	let a;
null	Denotes a null value	let a = null;
Symbol	Data type whose instances are unique and immutable	let value = Symbol('hello');

Slide Note

Escape Sequences

Escape sequence	Special Character
\0	null character
\'	single quote
\"	double quote
\\	backslash
\n	newline
\r	carriage return
\v	vertical tab
\t	tab
\b	backspace
\f	form feed

Slide Note

JavaScript – A Dynamically Typed Scripting Language

- While declaring variables or constants in JavaScript, the data type is not specified.
- The type is determined based on the value assigned.
- The value can be a literal or an evaluated result from an expression.
- Literals are fixed values:
  - String literals are specified in single or double quotes
  - Number literals could be integers or floating-point values
  - Boolean literals are true or false

Slide Note

```
console.log(typeof(34)); // prints number

console.log(typeof("34")); // prints string

console.log(typeof(false)); // prints boolean

console.log(typeof(x)); // prints undefined
```

typeof Operator

- The `typeof` is an operator that helps to determine the data type of operand.
- It returns a string which is the type of the unevaluated operand.
- When an undeclared variable is passed the `typeof` operator returns ‘undefined’.
- The developers can use this operator to validate the type of incoming data.

Slide Note

More on null and undefined

- `typeof null` // "object" (not "null" for legacy reasons)
- `typeof undefined` // "undefined"
- `null === undefined` // false
- `null == undefined` // true
- `null === null` // true
- `null == null` // true
- `!null` // true
- `isNaN(1 + null)` // false
- `isNaN(1 + undefined)` // true

Slide Note

Best Practices

- Ensure variables are declared using `let` keyword.
- Declare constants for values that do not need to be reassigned.
- Identifiers should be meaningful.
- Code should be adequately commented. Well commented codes make code easy to comprehend to the peers, mentors or reviewers.
- Code should be well indented. Well-indented codes are easy to understand.
- JavaScript is a dynamically typed language; therefore, all incoming values should be first type checked using `typeof()` operator to prevent error occurring due to type mismatch.

Slide Note

What Is ECMAScript?

- ECMAScript is a Standard for scripting languages.
  - ECMA – European Computer Manufacturer's Association
- JavaScript is a scripting language based on ECMAScript standards.
- If ECMAScript is blueprint of specification, JavaScript is the implementation of this blueprint.
- With new release of ECMAScript, the JavaScript engine running inside browsers is updated, although incrementally.
- ES6 = ECMAScript 6 = ES 2015 = ECMAScript 2015 is the 6th edition of ECMAScript
  - This version brought significant changes to the JavaScript language.
  - For example - `let` and `const` keywords, template literals, multi-line strings (with \n)
  - Refer to the site to know about the ES6 features.

Slide Note

Self-Check

What would be the output from below code?

```
let x1 = 10;

let y1 = x1++;

console.log(y1);
```

Slide Note

Self-Check: Solution

What would be the output from below code?

```
let x1 = 10;

let y1 = x1++;

console.log(y1);
```

Answer: 10

Explanation:

Here unary operator ++ is used as postfix operator and hence it evaluates post the assignment of current value 10 to y1.

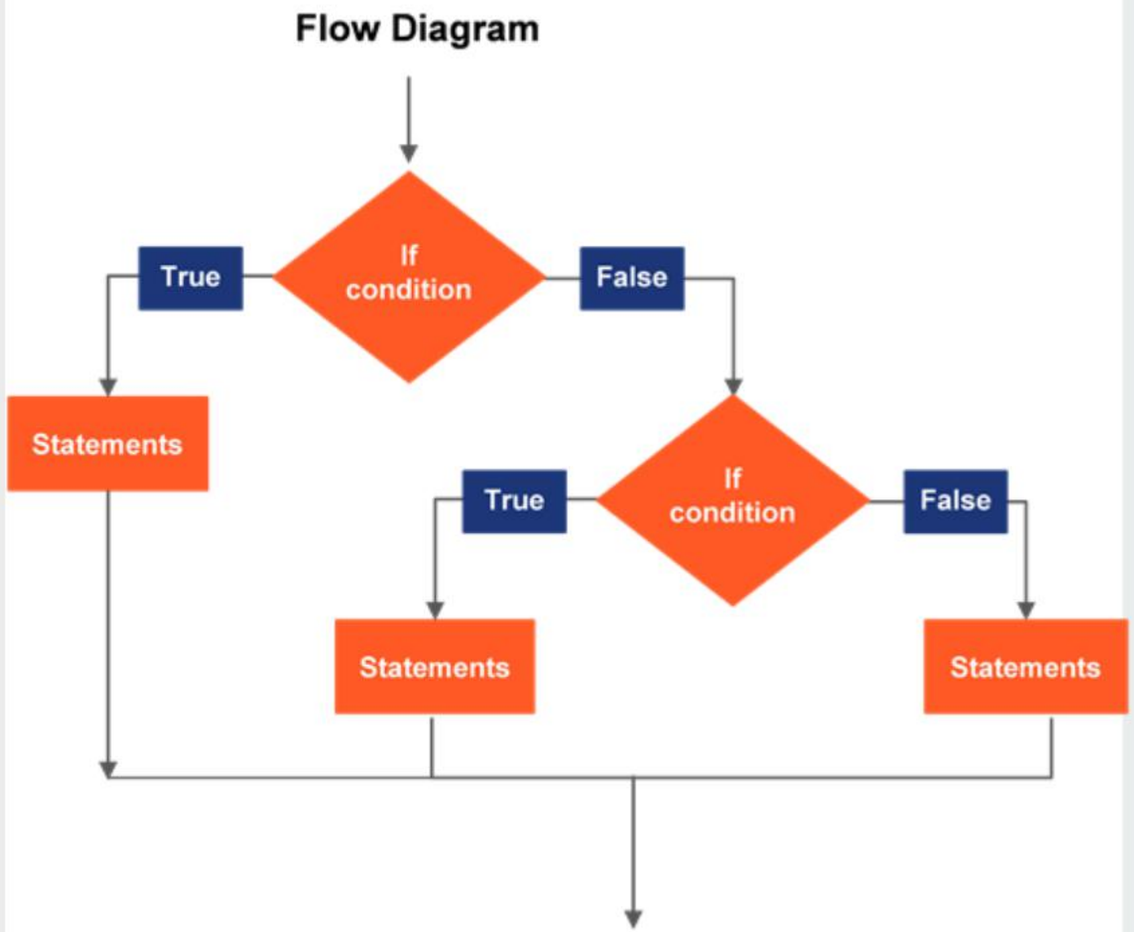
Slide Note

JavaScript Conditional Statements

- Conditional execution in JavaScript is created using:
  - Conditional Statements
    - `if`
    - `switch case`
  - Ternary Operator (? :)

Slide Note

if...else if Statements




```
if (condition-1) {
  // code to run if condition-1 is true
} else if (condition-2) {
  // code to run if condition-2 is true
} else {
  // run some other code instead
}
```

Slide Note

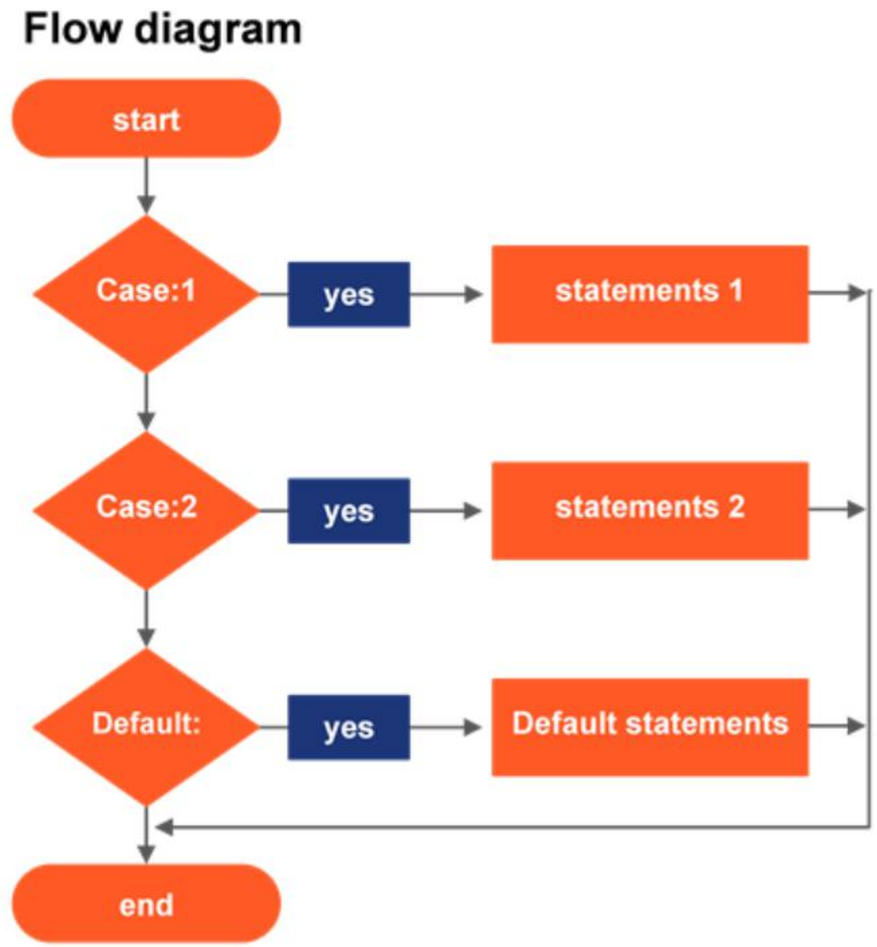
```
switch (expression) {
  case choice1:
    // run this code
    break;

  case choice2:
    // run this code instead
    break;

    // include as many cases as you like

  default:
    // actually, just run this code
}
```

switch case Statements



Slide Note

Key Points on `switch case` Construct

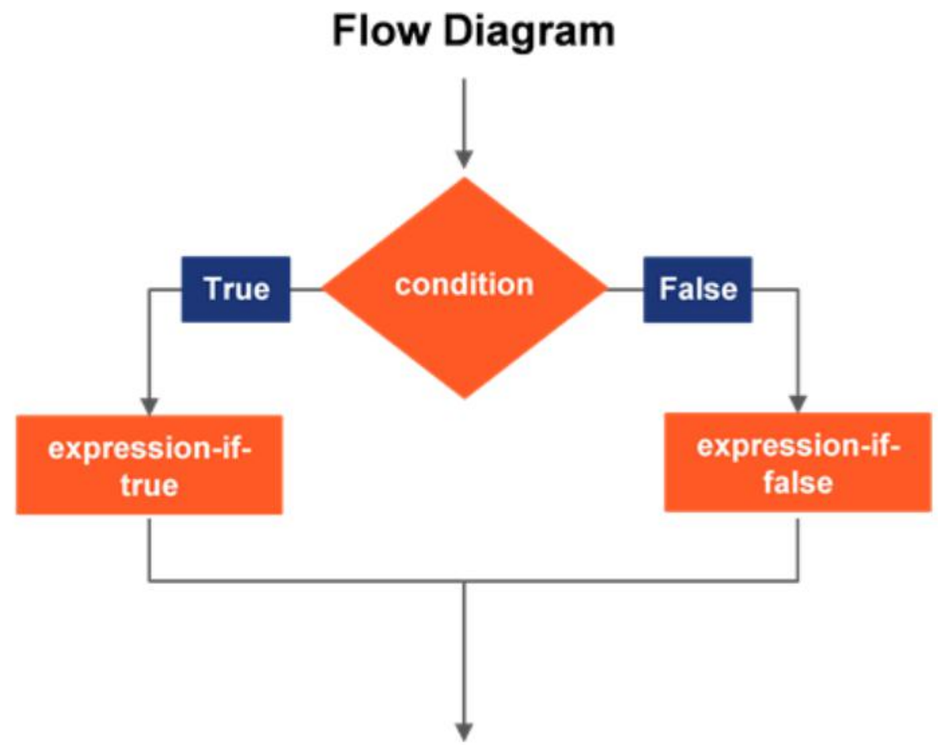
- `default` and `break` are optional statements in switch case construct.
- If `default` statement is not found, the program resumes execution at the statement following the end of switch.
- If `break` is omitted, the program continues execution inside the switch statement and evaluates the subsequent case statements.
- By convention, the `default` clause is written as the last clause, but it is not mandatory.

Slide Note

```
condition ? expression-if-true : expression-if-false
```

Ternary Operator

- The conditional (ternary) operator is the only JavaScript operator that takes three operands.



Slide Note

Truthy and Falsy

- All conditional constructs that contain a condition execute a statement if the specified condition is truthy.
- If the condition is falsy, another statement can be executed.
- In JavaScript, a truthy value is a true value in Boolean context.
- Falsy values are:
  - false, 0, empty string (“”), null, undefined and NaN.
- If a value is not falsy, it is truthy.

Slide Note

JavaScript Guidelines With Conditional Constructs

- Ternary operators should be put on a single line.
- Use shortcuts for Boolean tests, for example use `x` and `!x`, and not `x === true` and `x === false`.
- There should be no space between a control statement keyword and its opening parenthesis.
- There should be a space between the parentheses and the opening curly brace.
- It's good practice to always use block statements.

Slide Note

Self-Check

Can there be a more optimized way to write the below code?

```
let character = "d";

switch(character) {
    case "a":
        console.log(`Character 'a' is a vowel`);
        break;
    case "e":
        console.log(`Character 'e' is a vowel`);
        break;
    case "i":
        console.log(`Character 'i' is a vowel`);
        break;
    case "o":
        console.log(`Character 'o' is a vowel`);
        break;
    case "u":
        console.log(`Character 'u' is a vowel`);
        break;
    default:
        console.log(`Character is not a vowel`);
}
```

Slide Note

Self Check: Solution

Answer: Optimize the code by removing breaks from each case statement.
And putting it only after the last matching case statement.

```
let character = "d";

switch(character) {
    case "a":

    case "e":

    case "i":

    case "o":

    case "u":
        console.log(`Character '${character}' is a vowel`);
        break;

    default:
        console.log(`Character '${character}' is not a vowel`);
}
```

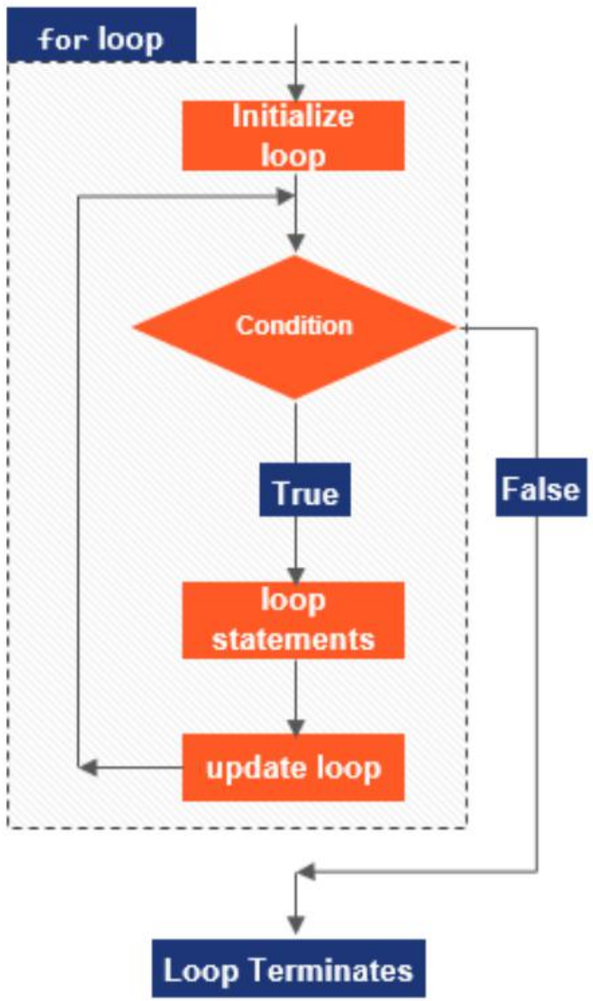
Slide Note

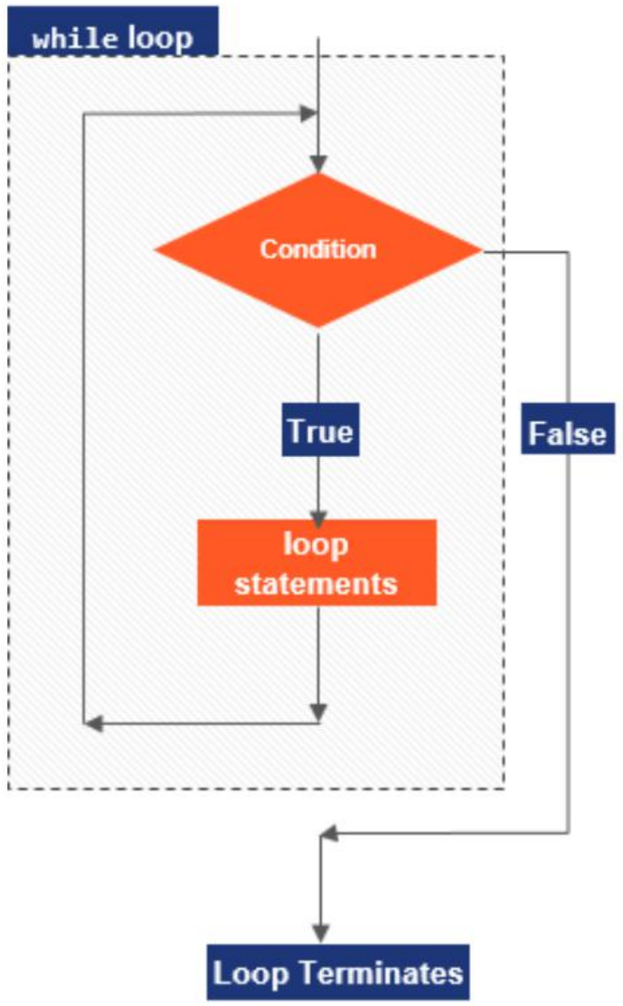
Loops

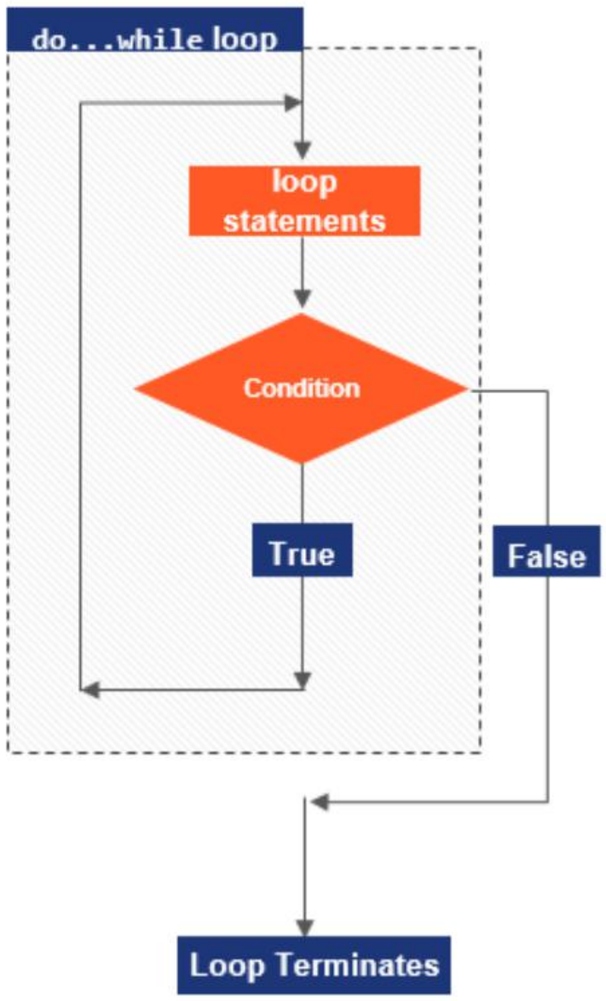
- Loops are programming constructs.
- The statements inside the loop are specified once but executed multiple times.
- Loops provide code re-usability.
- Re-usable code has a fewer number of code lines making it easy to maintain and debug.
- Loop constructs are comprised of:
  - Loop statement that controls the loop iterations
    - A loop statement is defined using a loop keyword and a condition expression.
  - Loop body that contains one or more statements that execute every time a loop repeats
    - The loop body defines a block of statements and are defined using delimiters like curly braces ({ }).

Slide Note

JavaScript Loops







Slide Note

for Loop

- Syntax of for Loop is given below:

```
for ([initialization]; [condition]; [increment-expression])
   loop-statement
```

- for loop is used as counter loop that repeats itself for a specified count.
- The count is initialized with initialization expression.
- The count value is tested by the condition expression.
  - The loop statement executes if condition evaluates to true.
  - If condition evaluates to false, loop terminates.
- The increment-expression updates count value, and the loop repeats, until the condition evaluates to true.

Slide Note

`while` Loop

- Syntax of `while` loop is given below:

```
while (condition)
   loop-statement
```

- `while` loop is usually used when the count of iterations is not known.
- A condition expression determines for how long the loop will repeat.
  - The loop statement executes if condition evaluates to true.
  - If condition evaluates to false, loop terminates.

Slide Note

do...while Loop

- Syntax of `do...while` loop is given below:

```
do
   loop-statement
while (condition);
```

- `do...while` loop is similar to the `while` loop.
  - Like `while` loop it is used when the count of iterations is unknown, and a condition is used to determine end of loop.
- However, with `do...while` loop, loop statements first execute and then the condition is checked.
  - If condition evaluates to true, the next iteration of loop repeats.
  - If condition is false, the loop terminates.

Slide Note

Controlling Iterations in Loops

`break` Statement

- The `break` statement in a loop is used to terminate the loop early - before the condition evaluates to false.
- It is also used with a `switch` statement to exit from `switch` block after the matching case statement is encountered.

`continue` Statement

- The `continue` statement in a loop is used to terminate the current iteration and skip the loop to the next iteration.
- In `for` loop, the `continue` statement, transfers the control to increment-expression.
- In `while` loop, the control is transferred to while condition.

Slide Note

Labelled Statement

- Syntax of labeled statement is given below:

```
label :
   statement
```

- In the syntax above, label is an identifier that is used as a label to refer to a statement.
- The statement with label can be referred to in the code somewhere else.
- For example:
  - A loop can be labeled by a label and the label can be referred to transfer the control using `break` or `continue` statement

Slide Note

Code Using Label

break with label

```
for (var i = 0; i < 5; i++) {
  innerLoop:
  for (var j = 0; j < 5; j++) {
    if (i == j) break innerLoop;
    console.log(i, j);
  }
}
```

Output:
1 0
2 0
2 1
3 0
3 1
3 2
4 0
4 1
4 2
4 3

continue with label

```
for (var i = 0; i < 5; i++) {
  innerLoop:
  for (var j = 0; j < 5; j++) {
    if (i == j) continue innerLoop;
    console.log(i, j);
  }
}
```

Output:
0 1
0 2
0 3
0 4
1 0
1 2
1 3
1 4
2 0
2 1
2 3
2 4
3 0
3 1
3 2
3 4
4 0
4 1
4 2
4 3

Slide Note

Self-Check

Predict the output for the given code.

```
let x = 1;
do {
  if(x<1) {
    break;
    console.log(x);
  }
}while(++x <= 10);
console.log(x);
```

1. Program will print numbers 1 to 11
2. Program will print numbers 1 to 10
3. Program will print only 11
4. No output

Slide Note

Self-Check: Solution

Predict the output for the given code.

```
let x = 1;
do {
  if(x<1) {
    break;
    console.log(x);
  }
}while(++x <= 10);
console.log(x);
```

1. Program will print numbers 1 to 11
2. Program will print numbers 1 to 10
3. **Program will print only 11**
4. No output