JavaScript

Control flows

18-05-2024

Dilakshan AT
Software Engineer, Unicom SD

Prerequisites

- **Basic JavaScript Syntax**:
  - Variables
  - Data types
  - Operators
  - Functions
  - Basic syntax

Learning objectives

- **Datatypes**
  - Boolean
- **Operators**
  - Comparison Operators
  - Logical Operators
- **Control flows**
  - Conditional statements
  - Loops

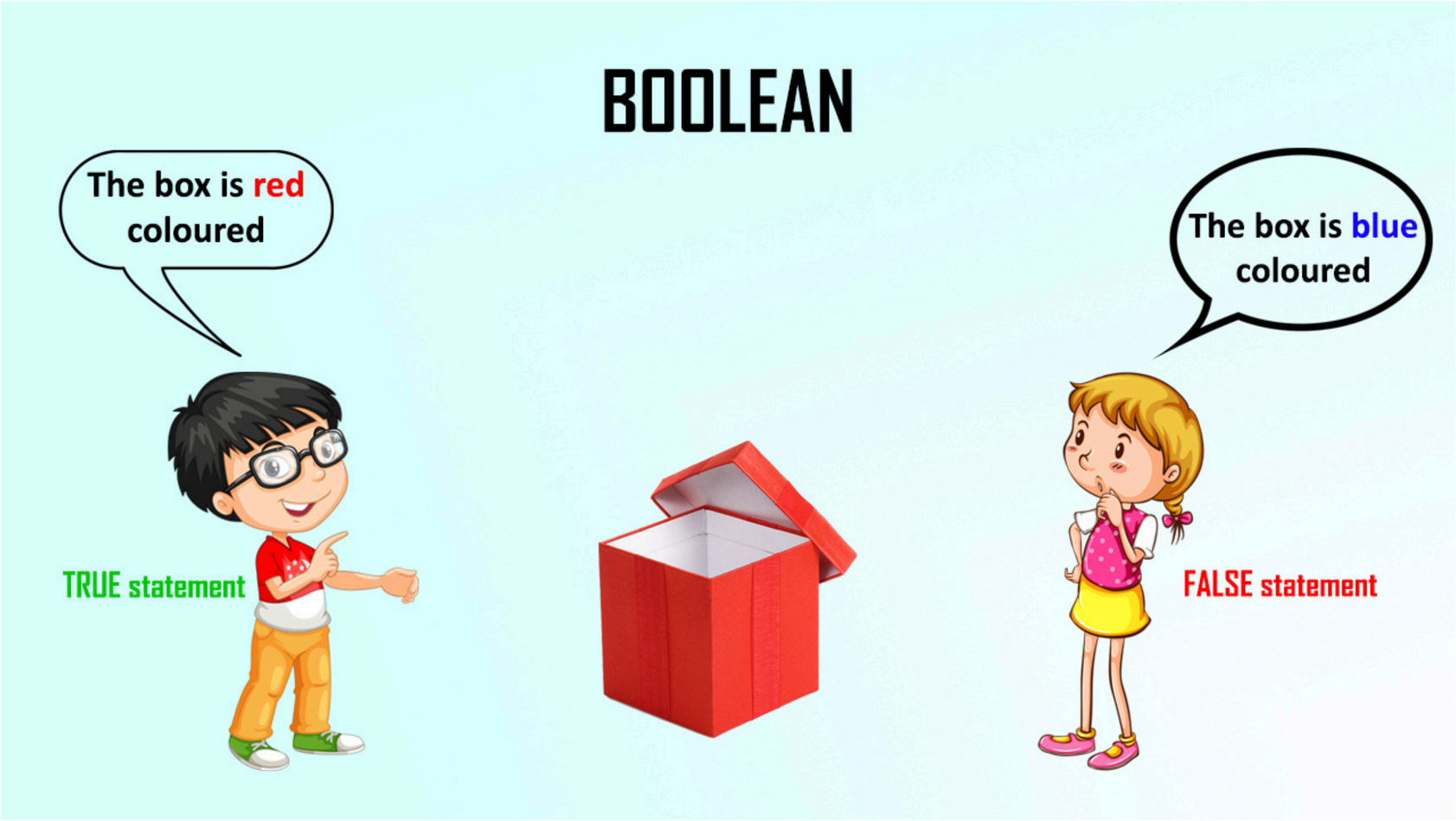
Boolean

- Boolean data type is a primitive data type that represents a logical value, either **true** or **false**.

- Booleans are commonly used in conditional statements, loops, and other control flow structures to determine the flow of execution in a program.

  - YES / NO
  - ON / OFF
  - TRUE / FALSE

Boolean


BOOLEAN
The box is red coloured
TRUE statement
The box is blue coloured
FALSE statement

Comparison Operators

- Comparison operators are used in logical statements to determine equality or difference between variables or values.

  - Equal to: **== or ===**
  - Not equal to: **!= or !==**
  - Greater than: **>**
  - Less than: **<**
  - Greater than or equal to: **>=**
  - Less than or equal to: **<=**

Logical Operators

- Used to combine conditional statements.
    - Logical AND: **&&**
    - Logical OR: **||**
    - Logical NOT: **!**

Conditional (Ternary) Operator

- The conditional (ternary) operator in JavaScript is a concise shorthand for an if-else statement.

- It assigns values to variables based on a condition, returning one of two expressions depending on whether the condition evaluates to true or false.

Conditional (Ternary) Operator

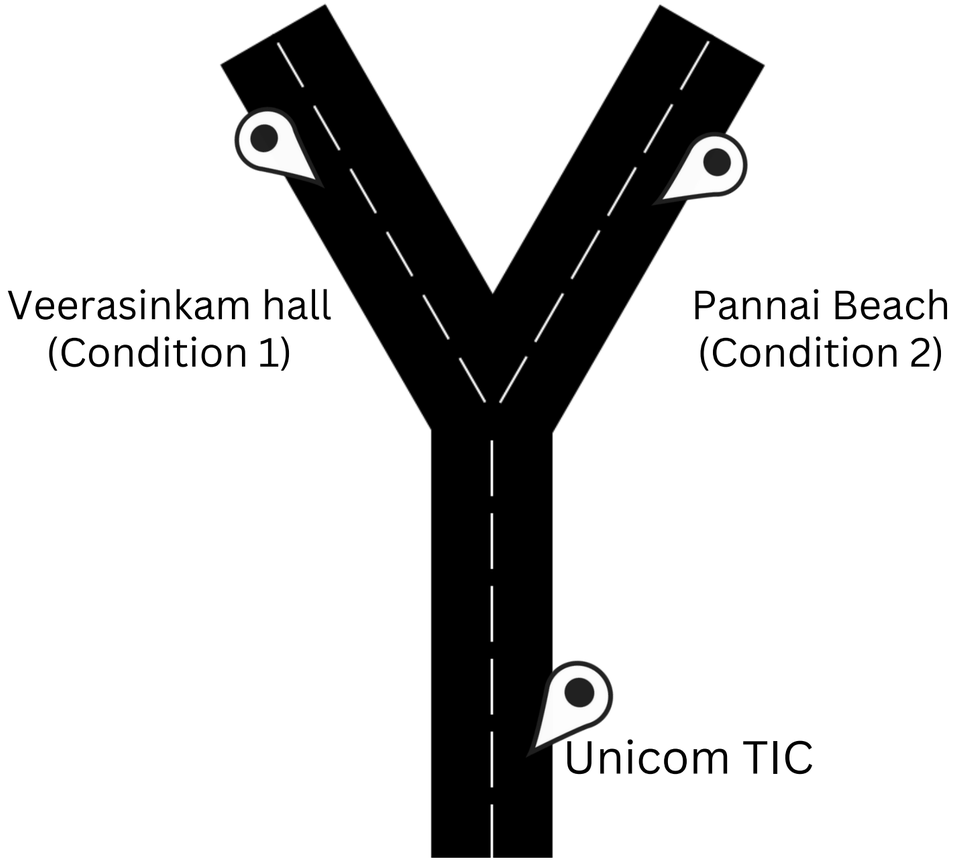
Syntax

```
var variable = condition ? valueIfTrue : valueIfFalse;
```

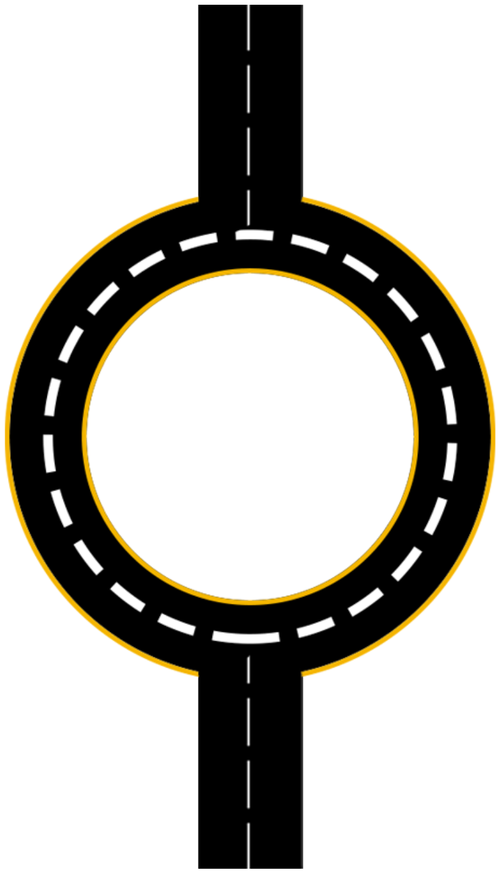
Control flows

- Control flow refers to the order in which statements in a program are executed.
- It determines the path a program takes based on different conditions and decisions made during runtime.
- In simpler terms, control flow directs the flow of execution within a program, allowing it to respond dynamically to various situations.

Example



Conditional



Loops

Types of Control Flows

- Control flows primarily consist of **conditional statements** and **loops**, which allow control of the flow of execution in their code based on different conditions and to iterate over data structures.
    - **Conditional statements**
    - **Loops**
    - **Iterating over Data Structures**

Conditional Statements

- Conditional statements are used to execute different blocks of code based on specified conditions.
- Types of Conditional Statements:
  - **if statement**
  - **if...else statement**
  - **nested if...else statement**
  - **switch statement**

if statement

- Executes a block of code if a specified condition is true. If the condition evaluates to false, the code block is skipped.

if statement

Syntax

```
if (condition) {
  // code to be executed if condition is true
}

```

if statement

```
1  let temperature = 25;
2
3  if (temperature > 30) {
4    console.log("It's hot outside!");
5  }
6
```

if...else statement

- Executes one block of code if a specified condition is true and another block of code if the condition is false.

if...else statement

Syntax

```
if (condition) {
  // code to be executed if condition is true
} else {
  // code to be executed if condition is false
}

```

if...else statement

```
let hour = 14;

if (hour < 12) {
  console.log("Good morning!");
} else {
  console.log("Good afternoon!");
}
```

nested if...else statement

- Allows for multiple conditions to be evaluated hierarchically, enabling more complex decision-making scenarios.

nested if...else statement

Syntax

```
if (condition1) {
  // code to be executed if condition1 is true
} else if (condition2) {
  // code to be executed if condition1 is false and condition2 is true
} else {
  // code to be executed if both condition1 and condition2 are false
}
```

nested if...else statement

```
let num = 0;

if (num > 0) {
  console.log("Positive");
} else if (num < 0) {
  console.log("Negative");
} else {
  console.log("Zero");
}
```

switch statement

- Use the switch statement to select one of many code blocks to be executed.

switch statement

Syntax

```
switch (expression) {
  case value1:
    // code to be executed if expression equals value1
    break;
  case value2:
    // code to be executed if expression equals value2
    break;
  default:
  // code to be executed if expression doesn't match any case
}
```

switch statement

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

switch statement

```
let day = "Monday";

switch (day) {
  case "Monday":
    console.log("It's the start of the week.");
    break;
  case "Friday":
    console.log("It's finally Friday!");
    break;
  default:
    console.log("Enjoy your day!");
}

```

Loops

- Loops are used to execute a block of code repeatedly.
- Loops execute a block of code repeatedly until a specified condition is met.

- Types of Loops:
  - **for loop**
  - **while loop**
  - **do...while loop**

for loop

- The for loop is used to iterate over a block of code a **specified number** of times.
- It consists of an **initialization**, a **condition**, and an **iteration expression**.

for loop

Syntax

```
for (initialization; condition; iteration) {
  // code to be executed
}

```

for loop

```
1   for (let i = 0; i < 5; i++) {
2     console.log(i);
3   }
4
```

while loop

- The while loop executes a block of code as long as a **specified condition** is true.
- It only has a condition expression.

while loop

Syntax

```
while (condition) {
  // code to be executed
}

```

while loop

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

do...while loop

- The do...while loop is similar to the **while** loop but ensures that the block of code is executed **at least once before the condition** is checked.

do...while loop

Syntax

```
do {
  // code to be executed
} while (condition);

```

do...while loop

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

break

- The **break** statement is used to exit the current loop **prematurely**, regardless of whether the loop's condition has been fulfilled.

break

```
for (let i = 0; i < 10; i++) {
  if (i == 5) {
    break; // exit loop when i equals 5
  }
  console.log(i);
}
```

continue

- The **continue** statement is used to **skip** the current iteration of the loop and proceed to the next iteration.

continue

```
1  for (let i = 0; i < 10; i++) {
2    if (i == 5) {
3      continue; // skip iteration when i equals 5
4    }
5    console.log(i);
6  }
7
```

Takeaway

- Understanding **Boolean** data types and **logical operations** is crucial for making decisions and controlling program flow.

- **Control flows** are fundamental for directing program execution and managing data effectively.

- Conditional statements like **if**, **if...else**, **nested if...else**, and **switch** allow developers to execute specific code blocks based on varying conditions.

- Loops like **for**, **while**, and **do...while** help developers automate repetitive tasks and navigate through data structures with ease.

- Keywords like '**break**' and '**continue**' facilitate the **interruption** of loops or **skipping** of specific iterations as needed.

Any Questions?

Happy Coding!