History of JavaScript

JavaScript was created by Brendan Eich, a programmer at a company called Netscape Communications. Originally, JavaScript was named 'Mocha', but it quickly gained popularity as “LiveScript” and later evolved into 'JavaScript'.

Initially, JavaScript was designed solely to provide scripting capabilities aimed at enhancing web pages with dynamic content and interactivity. With the growth of HTML5 and CSS3, JavaScript's capabilities expanded further, enabling developers to construct sophisticated web applications and interactive user interfaces.

Many JavaScript frameworks and libraries, such as AngularJS, ReactJS, and Vue.js, have revolutionized frontend development. Additionally, Node.js, a powerful JavaScript runtime environment, has empowered developers to build server-side applications using JavaScript.

Introduction of JavaScript

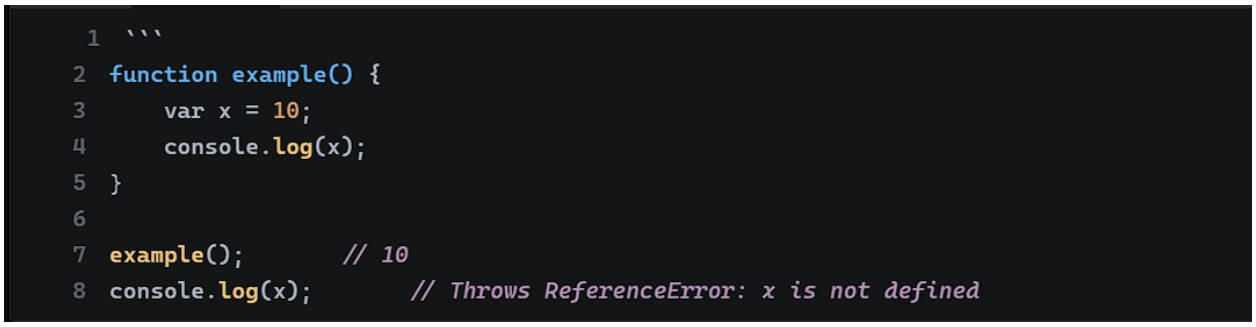
JavaScript is a high-level, interpreted programming language widely used for web development. It enables interactive and dynamic content on websites, such as animations, form validations, and real-time updates. Following the ECMAScript standard, JavaScript supports multiple programming paradigms, including functional, event-driven, and object-oriented approaches. It runs not only in web browsers but also on servers using environments like Node.js. With built-in methods and APIs, JavaScript efficiently manipulates the Document Object Model (DOM) to update web pages dynamically. Modern features like async/await, promises, and modules make JavaScript more powerful and flexible. Additionally, it has a vast ecosystem, including popular libraries and frameworks like React, Angular, and Vue.js. JavaScript can also handle HTTP requests, interact with databases, and build full-stack applications. Due to its versatility and wide adoption, it remains one of the most important programming languages in the world.

Let , Var & Const

Variable declaration is one of the easiest tasks in every programming language. However, when it comes to JavaScript, it can become a bit tricky due to the special features of the language

Var

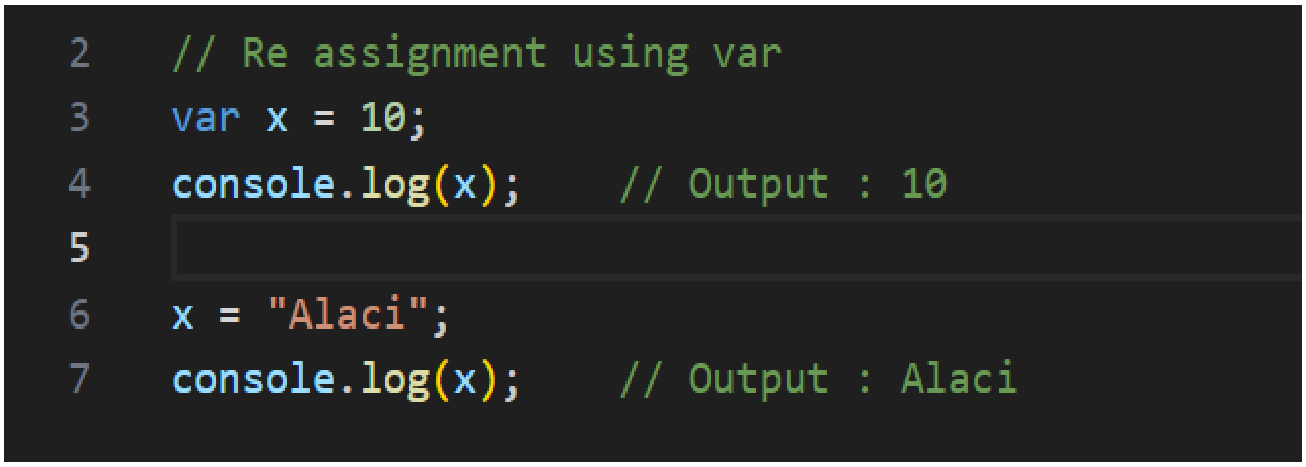
The var keyword was the traditional way of declaring variables in JavaScript. Variables declared with var are function-scoped or globally-scoped, but not block-scoped. This means that variables declared with var are hoisted to the top of their function or global scope. This hoisting behavior can sometimes lead to unexpected results and make debugging challenging.



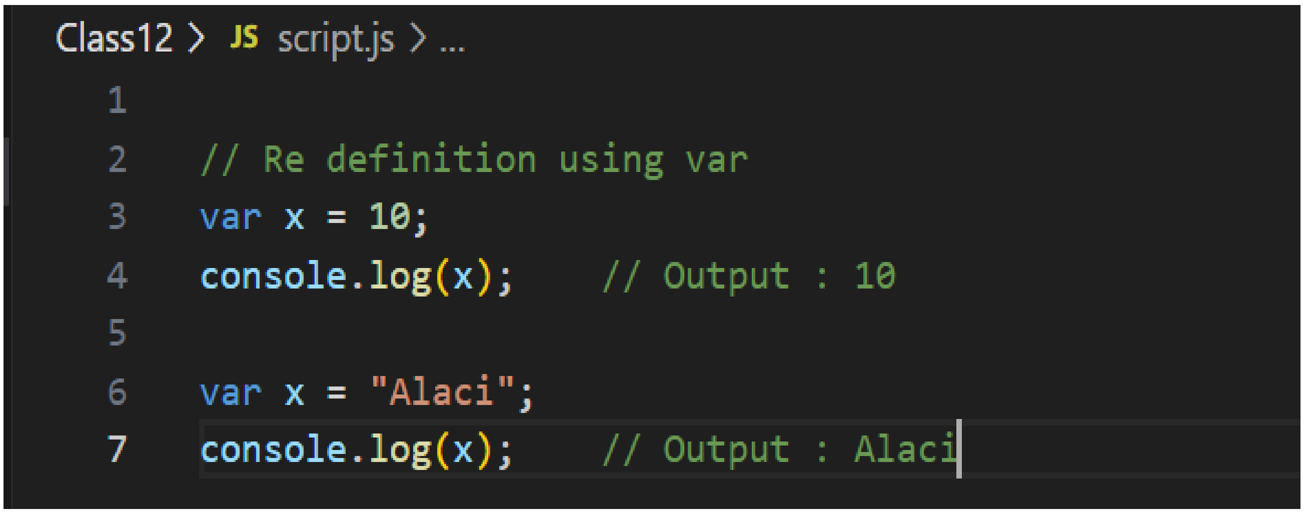
~~~
```
function example() {
    var x = 10;
    console.log(x);
}

example();        // 10
console.log(x);        // Throws ReferenceError: x is not defined
~~~

Reassignment and Redefinition using var

In JavaScript, the var keyword allows variables to be both reassigned and redefined within the same scope, unlike let which only allows reassignment. This behavior of var can sometimes lead to unexpected results and bugs in code if not used carefully.



```
// Re assignment using var
var x = 10;
console.log(x);     // Output : 10

x = "Alaci";
console.log(x);     // Output : Alaci
```

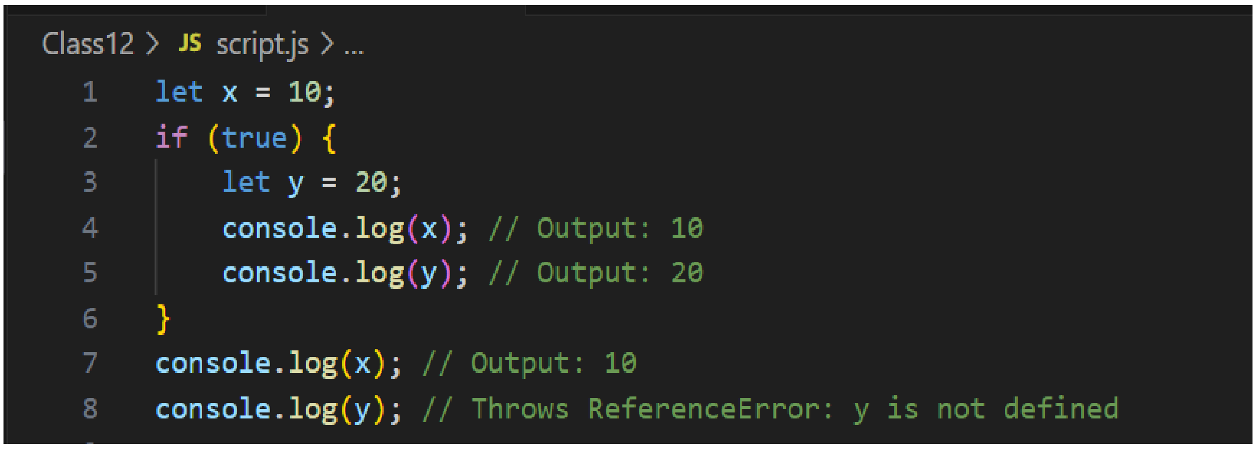


Class12 > JS script.js > ...

```

// Re definition using var
var x = 10;
console.log(x);     // Output : 10

var x = "Alaci";
console.log(x);     // Output : Alaci
```

Let

In JavaScript, the let keyword is used to declare variables that are block-scoped, meaning they are only accessible within the block in which they are defined. Block scope refers to any code block delimited by curly braces {} such as loops, conditionals, or function bodies.

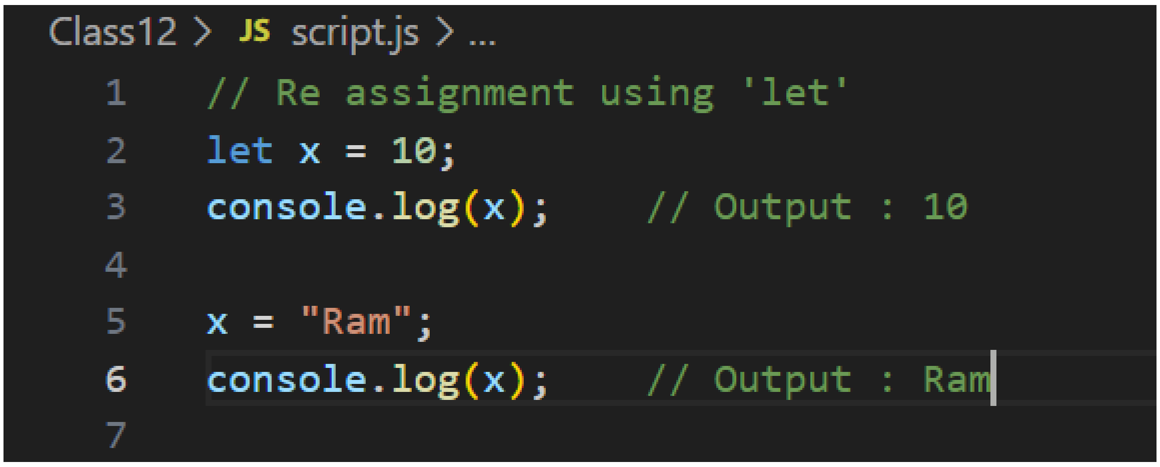


Class12 > JS script.js > ...

```
let x = 10;
if (true) {
    let y = 20;
    console.log(x); // Output: 10
    console.log(y); // Output: 20
}
console.log(x); // Output: 10
console.log(y); // Throws ReferenceError: y is not defined
```

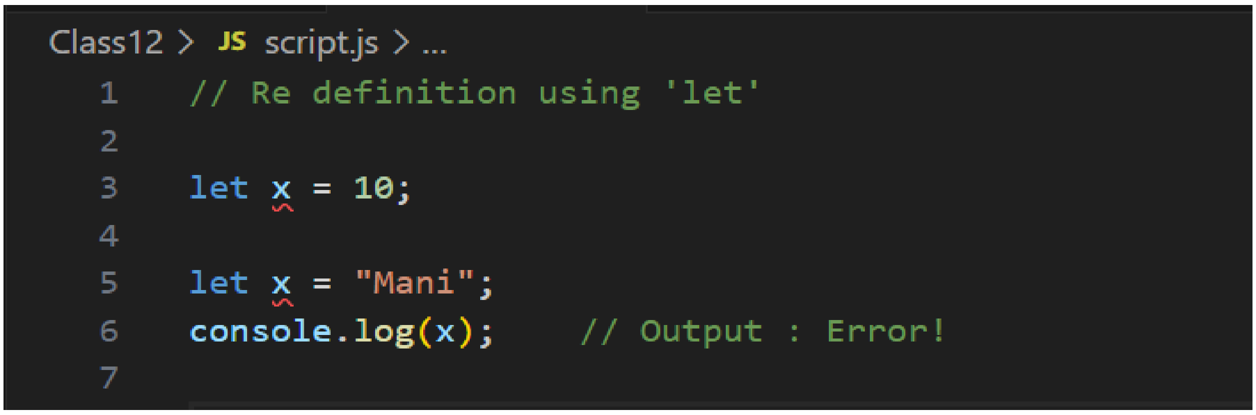
Reassignment and Redefinition using let

Variables declared with let can be reassigned, just like those declared with var, but they cannot be redeclared within the same block scope. This helps prevent accidental redeclaration of variables and can aid in writing more predictable and maintainable code.



Class12 > JS script.js > ...

```javascript
// Re assignment using 'let'
let x = 10;
console.log(x);     // Output : 10

x = "Ram";
console.log(x);     // Output : Ram

```

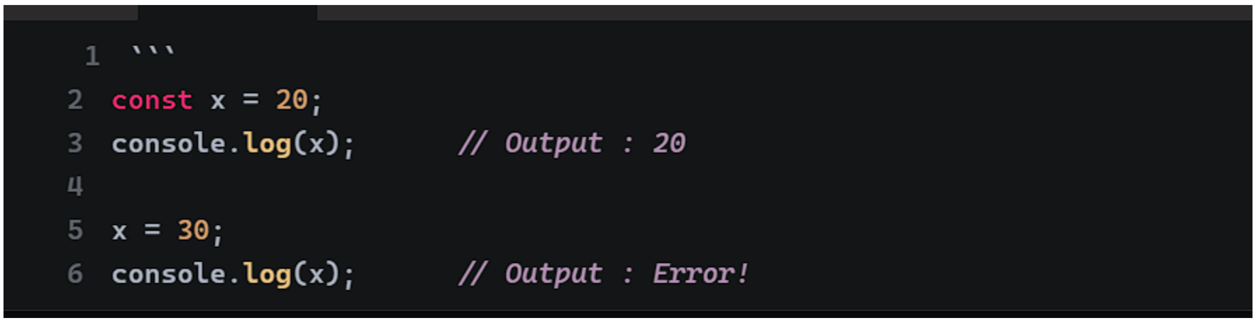


Class12 > JS script.js > ...

```javascript
// Re definition using 'let'

let x = 10;

let x = "Mani";
console.log(x);     // Output : Error!

```

Const

In JavaScript, the const keyword is used to declare constants. Constants are variables whose values cannot be reassigned once they are initialized. This means that once a value is assigned to a constant using const, it cannot be changed or reassigned throughout the execution of the script.



~~~
```
const x = 20;
console.log(x);        // Output : 20

x = 30;
console.log(x);        // Output : Error!
~~~

Constants declared with const must be initialized with a value. Unlike variables declared with var or let, which can be declared without immediately assigning a value, const requires initialization at the time of declaration.

Operators in JavaScript

JavaScript is a dynamic programming language, has various operators to perform operations on data and manipulate values.

Types of operators

1. Arithmetic Operators

- **Addition**
  The addition operator is denoted by '+'.
- **Subtraction**
  The subtraction operator is denoted by '-'.
- **Multiplication**
  The multiplication operator is denoted by '*'.
- **Exponentiation**
  he exponentiation operator is denoted by '**'.
- **Division**
  The division operator is denoted by '/'.
- **Modulus**
  The modulus operator is denoted by '%'.
- **Increment & decrement**

Increment operator is denoted by '++', and the decrement operator is denoted by '--'.

2. Assignment Operators

The assignment operator assigns value to the variables. The most common assignment operator we use is '='.

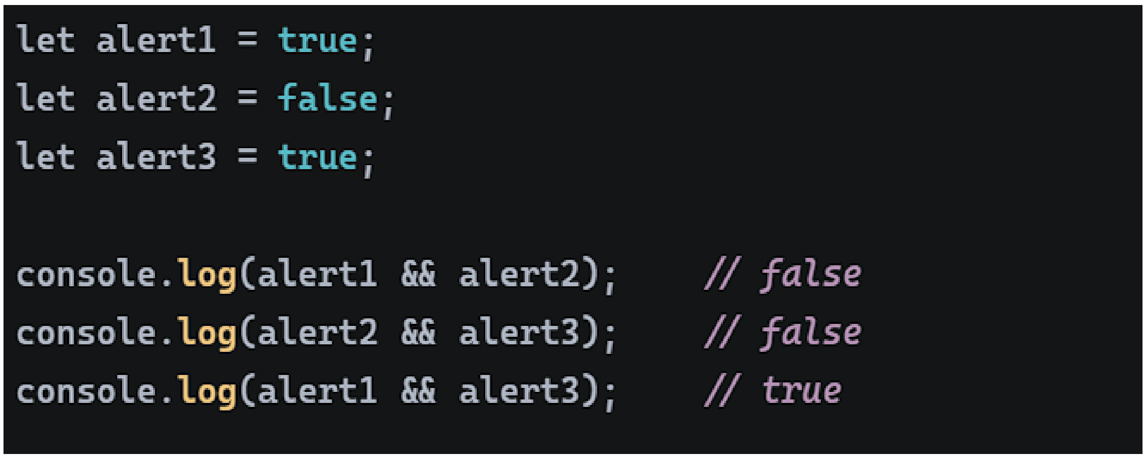
For example:

```
let FirstName = "Maniraj";
let y = 4;

console.log(FirstName);     // Maniraj
 console.log(y);                    // 4
```

3. Logical Operators

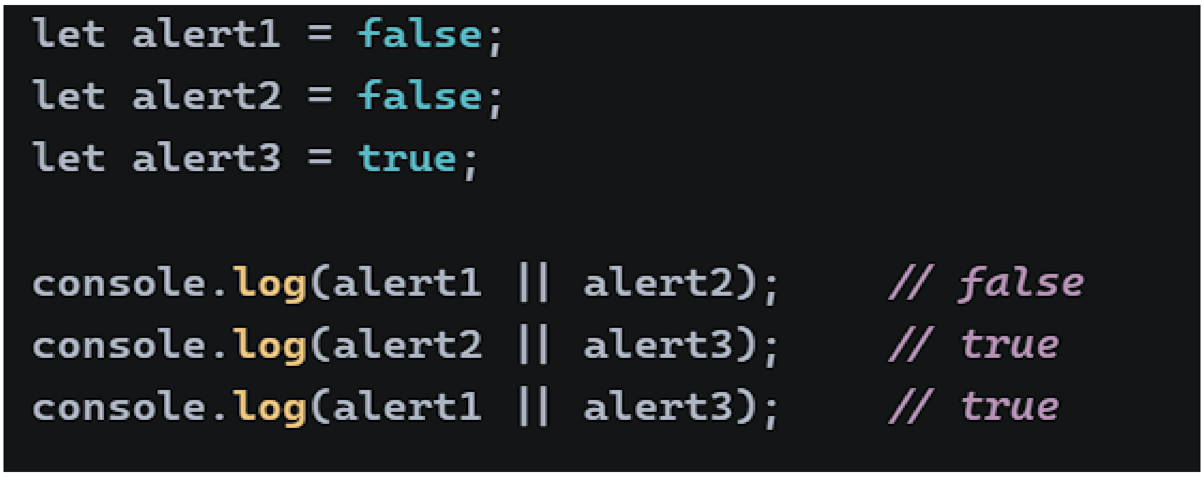
There are typically three types of logical operators :

(a) AND (&&)



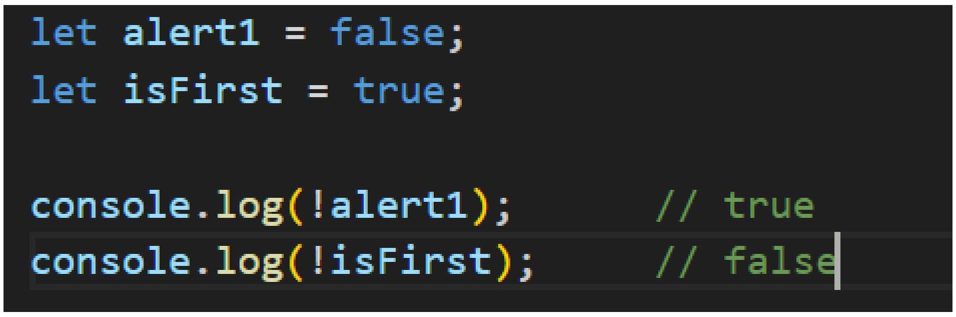
```
let alert1 = true;
let alert2 = false;
let alert3 = true;

console.log(alert1 && alert2);    // false
console.log(alert2 && alert3);    // false
console.log(alert1 && alert3);    // true
```

(b) OR (||)



```
let alert1 = false;
let alert2 = false;
let alert3 = true;

console.log(alert1 || alert2);     // false
console.log(alert2 || alert3);     // true
console.log(alert1 || alert3);     // true
```

(c) NOT (!)



```
let alert1 = false;
let isFirst = true;

console.log(!alert1);       // true
console.log(!isFirst);      // false
```

4. Comparison Operators

Comparison operators, such as greater then (>), less than (<), greater than or equal (>=), less than or equal (<=), equal to (==), not equal to (!=) are used to compare values and determine the relationship between them. These return a boolean value in the output.

If else statements

Conditional statements in JavaScript, such as 'if' and 'if-else', allow us to control the flow of our code based on certain conditions. With these statements, we can specify blocks of code to be executed only when certain conditions are met. For example, using an if statement, we can execute a piece of code if a particular condition evaluates to true. Alternatively, with an if-else statement, we can provide an alternative code block to be executed if the condition evaluates to false. These conditional statements provide powerful tools for building dynamic and responsive programs in JavaScript.

1. If Statement

In JavaScript, an 'if' statement is a fundamental conditional statement that allows us to execute a block of code only if the specified condition is true. If the condition is false, the code block associated with the 'if' statement is skipped.

Syntax:

```
if (condition){
    // Code block to be executed if the condition is true
}
```

2. If-else Statement

In JavaScript, an 'if-else' statement is a conditional statement that allows us to execute one block of code if a specified condition evaluates to true, and another block of code if the condition evaluates to false.

Syntax

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

3. If-else-if Statement

An if-else-if statement is a conditional statement that allows us to test multiple conditions and execute different blocks of code based on the outcome of those conditions.

Syntax

```
if (condition1) {
    // Code block to be executed if condition1 is true
} else if (condition2) {
    // Code block to be executed if condition2 is true
} else {
    // Code block to be executed if all conditions are false
}
```

Loops in JavaScript

At the outset of learning to code in any specific language, grasping the concept of "loops" and their functionality is paramount. Loops are integral to maintaining the readability of our code and reducing pressure on the system by minimizing the number of lines needed to accomplish a given task. This directly enhances the quality of the code, improves runtime efficiency, and saves considerable time that would otherwise be spent writing extensive code.

When discussing loops in computer programming, it's essential to note that there are primarily three types commonly used.

(a) For loop

Syntax

```
for ( initialization ; condition ; increment/decrement ) {
              // code block to be executed
}
```

(b) While loop

Syntax

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

(c) Do-while loop

Syntax

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

JavaScript Math Object

The JavaScript Math object is a built-in object that provides mathematical constants and functions for performing mathematical operations in JavaScript. It provides a set of properties and methods for mathematical computations, making it easier to work with numbers in JavaScript.There are different types of function in Math Object.

1. Math.random()

For Example,

```
let value = Math.random();
console.log(value);
value = Math.random();
console.log(value);
```

2. Math.abs()

For Example,

```
let value = Math.abs(-2.7);
console.log(value);
//Output : 2.7
```

3. Math.sqrt()

For Example,

```
let value = Math.sqrt(15);
console.log(value);
//Output : 3.872983346207417

value = Math.sqrt(36);
console.log(value);
//Output : 6
```

4. Math.pow()

For Example,

```
let value = Math.pow(2,3);
console.log(value);
//Output : 8

value = Math.pow(4,2);
console.log(value);
//Output : 16
```

5. Math.min()

For Example,

```
let value = Math.min(2,3,70,9);
console.log(value);
//Output : 2

value = Math.min(-10,-3,-9);
console.log(value);
//Output : -10
```

6. Math.max()

For Example,

```
let value = Math.max(2,3,70,9);
console.log(value);
//Output : 70

value = Math.max(-10,-3,-9);
console.log(value);
//Output : -3
```

```
value = Math.max();
console.log(value);
//Output : -Infinity
```

7. Math.sin()

For Example,

```
let value = Math.sin(Math.PI/2);
console.log(value);
//Output : 1

value = Math.sin(Math.PI/6);
console.log(value);
//Output : 0.49999999999999994

value = Math.sin(0);
console.log(value);
//Output : 0
```

8. Math.cos()

For Example,

```
let value = Math.cos(Math.PI/3);
console.log(value);
//Output : 0.5000000000000001

value = Math.cos(Math.PI/6);
console.log(value);
```

```
//Output : 0.8660254037844387

value = Math.cos(0);
console.log(value);
//Output : 1
```

9. Math.tan()

For Example,

```
let value = Math.tan(Math.PI/3);
console.log(value);
//Output : 1.7320508075688767

value = Math.tan(Math.PI/6);
console.log(value);
//Output : 0.5773502691896257

value = Math.tan(0);
console.log(value);
//Output : 0
```

JavaScript Functions

A **function** in JavaScript is a reusable block of code that performs a specific task when called. It helps in organizing code, making it more readable and reusable. A function can take **parameters** (inputs) and return **a value** or execute a set of instructions.

Syntax

```
        function  functionName(parameters) {
                // Code to execute
                  return result; // (Optional)
          }
```

Programs using Function in JavaScript

1. Multiplication Table

```
    function multiplicationTable(num) {
            for (let i = 1; i <= 10 ; i++ ) {
                  console.log(` ${num} x ${i} = ${num * i}` );
     }
     }
     multiplicationTable(5);
```

2. Fibonacci Series

```
function fibonacciSeries(n) {
  let fib = [0, 1];
  for (let i = 2; i < n; i++) {
    fib[i] = fib[i - 1] + fib[i - 2];
  }
  console.log(fib.join(', '));
}
fibonacciSeries(10);
```

3. Check Prime Number

```
function isPrime(num) {
  if (num < 2) return false;
  for (let i = 2; i <= Math.sqrt(num); i++) {
    if (num % i === 0) return false;
  }
  return true;
}
console.log(isPrime(7)); // true
console.log(isPrime(10)); // false
```

4. Check Composite Number

```
function isComposite(num) {
  if (num < 2) return false;
  for (let i = 2; i < num; i++) {
    if (num % i === 0) return true;
  }
  return false;
}
console.log(isComposite(9)); // true
console.log(isComposite(7)); // false
```

5.Factorial of a Number

```
function factorial(n) {
  if (n === 0 || n === 1) return 1;
  return n * factorial(n - 1);
}
console.log(factorial(5)); // 120
```

6. Sum of Digits

```
function sumOfDigits(num) {
  let sum = 0;
  while (num > 0) {
    sum += num % 10;
    num = Math.floor(num / 10);
  }
  return sum;
}
console.log(sumOfDigits(1234)); // 10
```

7. Reverse a Number

```
function reverseNumber(num) {
  return parseInt(num.toString().split('').reverse().join(''));
}
console.log(reverseNumber(1234)); // 4321
```

8. Check Palindrome Number

```
function isPalindrome(num) {
  return num.toString() === num.toString().split('').reverse().join('');
}
console.log(isPalindrome(121)); // true
console.log(isPalindrome(123)); // false
```

9. Check Armstrong Number

```
function isArmstrong(num) {
  let sum = 0;
  let temp = num;
  let digits = num.toString().length;

  while (temp > 0) {
    let digit = temp % 10;
```

```
        sum += Math.pow(digit, digits);
        temp = Math.floor(temp / 10);
    }
    return sum === num;
}
console.log(isArmstrong(153)); // true
console.log(isArmstrong(123)); // false
```

10. Find GCD (Greatest Common Divisor)

```
function gcd(a, b) {
    return b === 0 ? a : gcd(b, a % b);
}
console.log(gcd(36, 60)); // 12
```

Event Listeners in JavaScript

Event listeners are a fundamental aspect of JavaScript programming, enabling developers to create dynamic and interactive web applications. They allow us to respond to user actions such as clicks, mouse movements, keyboard inputs, and more

What are 'Event Listeners' ?

In JavaScript, an event listener is a function that waits for a specific event to occur and then executes a callback function in response. Event listeners are commonly used to respond to user interactions such as clicks, mouse movements, keyboard inputs, or changes in the state of an HTML element.

How do Event Listeners Work?

Event listeners are attached to HTML elements, listening for specific events such as clicks, mouse movements, key presses, form submissions, etc. When the specified event occurs, the associated callback function is executed.

addEventListener()

In JavaScript, addEventListener() is a method used to attach an event handler to an element, enabling it to listen for specific events and execute code/callback function in response to those events.

Programs using Event Listeners in JavaScript

1. Button Click Event – Display Message

```
<!DOCTYPE html>

<html>

<head>

   <title>Button Click Event</title>

</head>

<body>
```

```
  <button id="btn">Click Me</button>

  <script>

    document.getElementById("btn").addEventListener("click", function() {

      document.write("Button Clicked!!! Welcome to JavaScript.");

    });

  </script>

</body>

</html>
```

2. Window Load Event – Show Welcome Message

```
<!DOCTYPE html> <html>

<head> <title>Window Load Event</title>  </head>

<body>

  <script>

    window.addEventListener("load", function() {

      document.write("Welcome! The page has fully loaded.");

    });

  </script>
```

```
</body>

</html>
```

3. Mouse Over Event – Change Text

```
<!DOCTYPE html>

<html>

<head>

  <title>Mouse Over Event</title>

</head>

<body>

  <p id="text">Hover over me!</p>
  <script>
    document.getElementById("text").addEventListener("mouseover", function() {
      document.write("You hovered over the text!");
    });
  </script>
</body>
</html>
```

4. Keypress Event – Display Key Pressed

```
<!DOCTYPE html>
<html>
<head>
<title>Keypress Event</title>
</head>
<body>
```

```
  <input type="text" id="inputBox" placeholder="Type something...">

  <script>

      document.getElementById("inputBox").addEventListener("keypress",
function(event)             {

        document.write("You pressed: " + event.key);

    });

  </script>

 </body>

 </html>
```

5. Double Click Event – Change Message

```
<!DOCTYPE html>

<html>

<head>

  <title>Double Click Event</title>

</head>
```

```
<body>
  <button id="dblClickBtn">Double Click Me</button>
  <script>
    document.getElementById("dblClickBtn").addEventListener("dblclick", function() {
      document.write("You double-clicked the button!");
    });
  </script>
</body>
</html>
```

The End

Introduction of jQuery

jQuery is a fast, small, and feature-rich JavaScript library designed to simplify HTML document traversal, event handling, animation, and Ajax interactions. It provides an easy-to-use API that works across different web browsers, making it easier to manipulate the DOM (Document Object

Model). With jQuery, developers can write less code to achieve complex tasks, improving productivity and code readability. It offers features such as chaining, DOM manipulation, event delegation, and smooth animations. jQuery also supports Ajax calls, allowing for dynamic page updates without reloading. It has been widely used for building interactive and responsive web applications. Although newer technologies like vanilla JavaScript and modern frameworks have gained popularity, jQuery remains a valuable tool for web developers, especially for projects that require quick solutions or support older browsers.

Adding jQuery to Our Web Pages

There are several ways to start using jQuery in our web site.

- Download the jQuery library from jQuery.com
- Include jQuery from a CDN, like Google

jQuery Syntax

jQuery is a popular JavaScript library that simplifies HTML document traversal, event handling, animation, and Ajax interactions.

Basic syntax is: **$(*selector*).*action*()**

- `$` is the jQuery function.
- `selector` is used to find HTML elements.
- `action()` is the jQuery action to be performed on the selected elements.

Programs using HTML + jQuery

1. Change Text of an Element

```
<!DOCTYPE html>
<html>
<head>
  <title>Change Text</title>
</head>
<body>
  <p id="demo">This is a paragraph.</p>
  <button id="change-text">Change Text</button>

  <script src="https://code.jquery.com/jquery-
3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      $('#change-text').click(function() {
        $('#demo').text('Text has been changed!');
      });
    });
  </script>
</body>
</html>
```

2. Hide and Show an Element

```
<!DOCTYPE html>
```

```
<html>
<head>
  <title>Hide and Show</title>
</head>
<body>
  <p id="message">This is a hidden message.</p>
  <button id="hide-btn">Hide Message</button>
  <button id="show-btn">Show Message</button>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      $('#hide-btn').click(function() {
        $('#message').hide();
      });

      $('#show-btn').click(function() {
        $('#message').show();
      });
    });
  </script>
</body>
</html>
```

3. Change Background Color of an Element

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Change Background Color</title>
</head>
<body>
  <div id="color-box" style="width:200px; height:100px; background-color: lightblue;">Click the button to change my color!</div>
  <button id="change-color">Change Color</button>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      $('#change-color').click(function() {
        $('#color-box').css('background-color', 'lightgreen');
      });
    });
  </script>
</body>
</html>
```

4. Form Validation Program

```
<!DOCTYPE html>
<html>
```

```
<head>
   <title>Form Validation</title>
</head>
<body>
   <h2>Contact Form</h2>
   <form id="contact-form">
      <label for="name">Name:</label>
      <input type="text" id="name" name="name" required><br><br>

      <label for="email">Email:</label>
      <input type="email" id="email" name="email" required><br><br>

      <label for="message">Message:</label>
      <textarea id="message" name="message" required></textarea><br><br>

      <button type="submit" id="submit-btn">Submit</button>
   </form>

   <div id="error-message" style="color: red; display: none;">Please fill all the fields
correctly.</div>




   <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
   <script>
```

```
    $(document).ready(function() {
      $('#contact-form').submit(function(event) {
        event.preventDefault(); // Prevent form submission

        // Clear previous error messages
        $('#error-message').hide();

        // Validate the fields
        var name = $('#name').val();
        var email = $('#email').val();
        var message = $('#message').val();
        var emailPattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;

        if (!name || !email || !message) {
          $('#error-message').text('All fields are required!').show();
        } else if (!emailPattern.test(email)) {
          $('#error-message').text('Please enter a valid email address!').show();
        } else {
          // Form is valid, submit the form (for demo purposes, we just show a success
message)
          alert('Form submitted successfully!');
          $('#contact-form')[0].reset(); // Reset form fields
        }
      });
    });
  </script>
</body>
</html>
```

History of PHP

PHP, created by **Rasmus Lerdorf** in **1994**, initially started as a set of CGI (Common Gateway Interface) scripts for tracking visitors to his personal website. Initially called **Personal Home Page Tools** (PHP Tools), it later evolved into **PHP/FI** (Forms Interpreter). In **1997**, **Andi Gutmans** and **Zeev Suraski** joined Lerdorf to rewrite PHP, which resulted in the release of **PHP 3.0**—the first major version. This marked the shift to PHP as a robust programming language, enabling dynamic website creation. **PHP 4.0** was introduced in **2000**, powered by the **Zend Engine**, which greatly improved performance and added key features like sessions. In **2004**, **PHP 5.0** was released with enhanced **object-oriented programming (OOP)** support, and features like **PDO** for database interaction. PHP saw a major performance boost with **PHP 7.0** in **2015**, providing twice the speed compared to previous versions. Most recently, **PHP 8.0** was released in **2020**, incorporating **JIT compilation** and modern language features. Throughout its evolution, PHP has remained a critical tool for developing dynamic, data-driven websites and web applications.

Introduction to PHP

PHP (Hypertext Preprocessor) is a popular server-side scripting language used for web development. It is an open-source, general-purpose language designed for creating dynamic and interactive websites and applications. PHP is widely used for building websites that require database interactions, such as content management systems (CMS), e-commerce sites, and forums.

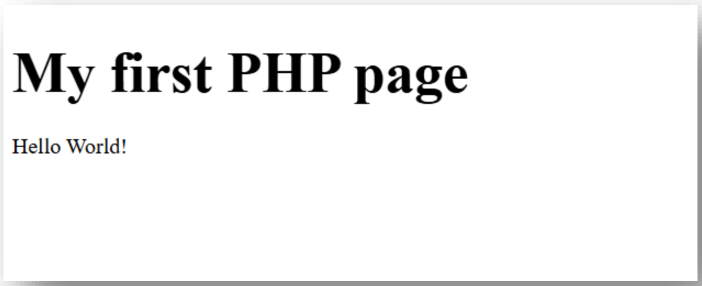
PHP code is executed on the server, and it generates HTML that is sent to the client-side (browser). This allows PHP to work seamlessly with HTML, CSS, and JavaScript to produce dynamic content.

Basic PHP Syntax

```
<?php
// PHP code goes here
?>
```

A simple `.php` file with both HTML code and PHP code:

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP
page</h1>

<?php
echo "Hello World!";
?></body></html>
```



My first PHP page

Hello World!

Creating (Declaring) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example:

```
$x = 5;
$y = "John";
```

PHP echo and print Statements

`echo` and **`print`** are more or less the same. They are both used to output data to the screen.

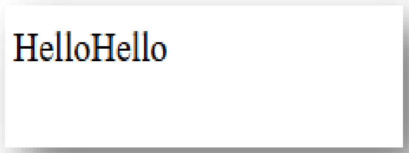
The differences are small: **`echo`** has no return value while **`print`** has a return value of 1 so it can be used in expressions. **`echo`** can take multiple parameters (although such usage is rare) while **`print`** can take one argument. `echo` is marginally faster than `print`.

The PHP echo Statement

The `echo` statement can be used with or without parentheses: `echo` or `echo()`.

Example:

```
echo "Hello";
  //same as:
echo("Hello");
```



```
HelloHello
```

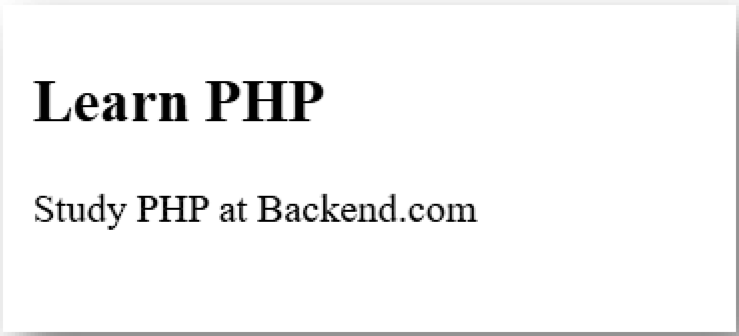
The PHP print Statement

The `print` statement can be used with or without parentheses: `print` or `print()`.

Example:

```
$txt1 = "Learn PHP";
$txt2 = "Backend.com";
```

```
print '<h2>' . $txt1 . '</h2>';
print '<p>Study PHP at ' . $txt2 . '</p>';
```



```
Learn PHP

Study PHP at Backend.com
```

PHP Data Types

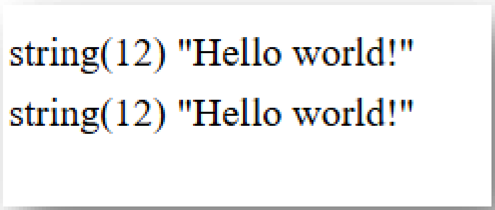
Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- **String**
  A string is a sequence of characters, like "Hello world!".
  A string can be any text inside quotes.

  Example:

  ```
  <!DOCTYPE html>
  <html>
  <body>
  <?php
  $x = "Hello world!";
  $y = 'Hello world!';
  ```



```
string(12) "Hello world!"
string(12) "Hello world!"
```

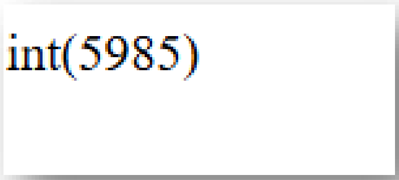
```
var_dump($x);
echo "<br>";
var_dump($y);
?>
</body>
</html>
```

- **Integer**

  An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

  Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5985;
var_dump($x);
?>
</body>
</html>
```



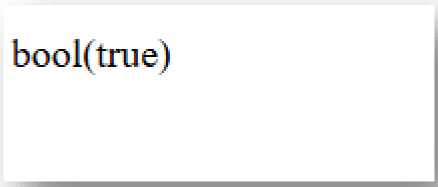
```
int(5985)
```

- **Boolean**

  A Boolean represents two possible states: TRUE or FALSE.

  Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = true;
var_dump($x);
```



```
bool(true)
```

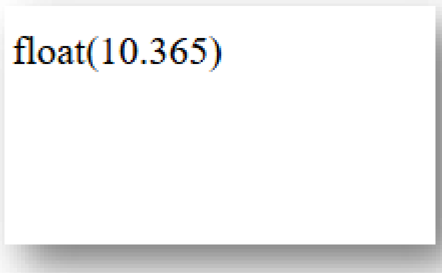
```
?>
</body>
</html>
```

- **Float**

  A float (floating point number) is a number with a decimal point or a number in exponential form.

  Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 10.365;
var_dump($x);
?>

</body>
</html>
```



```
float(10.365)
```

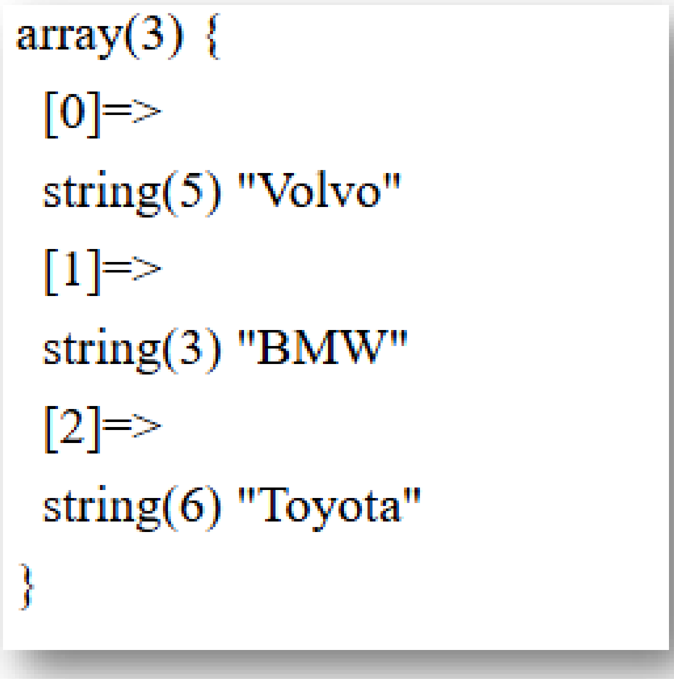
- **Array**

  An array stores multiple values in one single variable.

  Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
```

```
?>

</body>
</html>
```



```
array(3) {
  [0]=>
  string(5) "Volvo"
  [1]=>
  string(3) "BMW"
  [2]=>
  string(6) "Toyota"
}
```

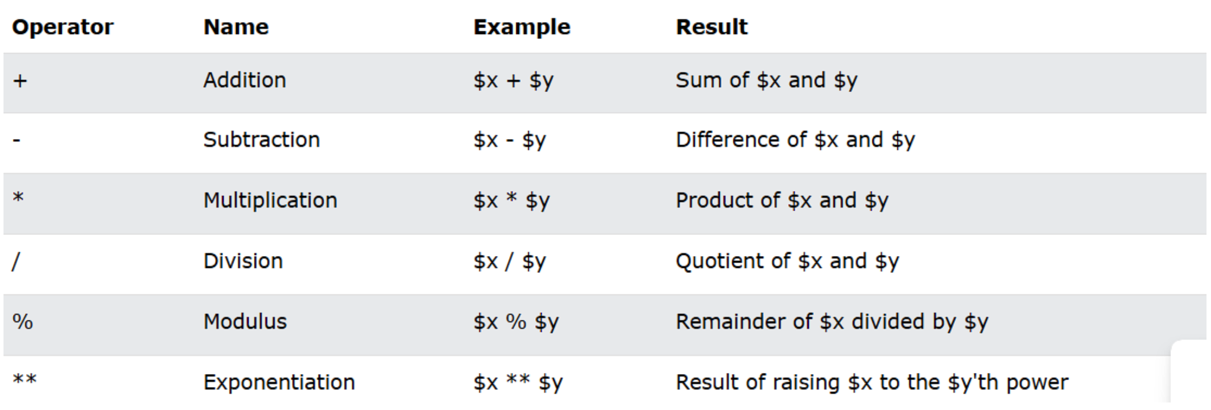
PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

1. Arithmetic Operators

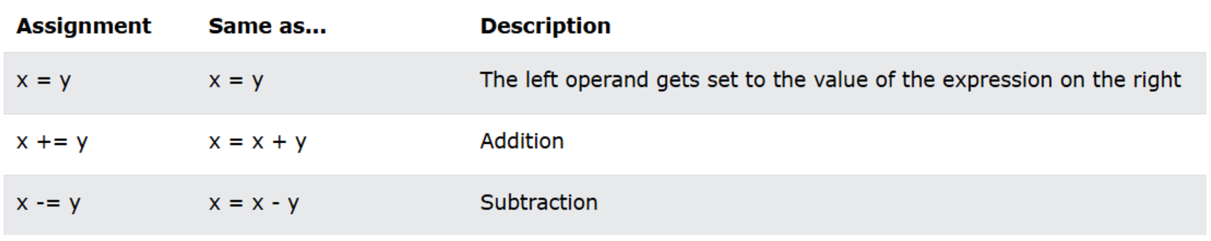
The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.



Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

2. Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.The basic assignment operator in PHP is "=".



Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction

3. Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.

4. Increment / Decrement Operators

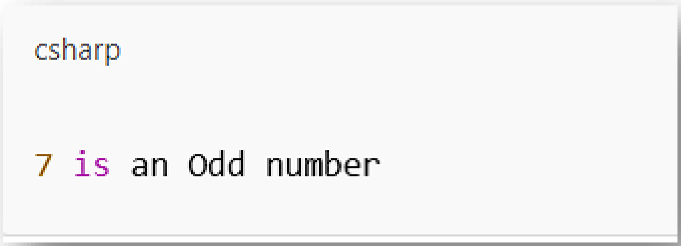
The PHP increment operators are used to increment a variable's value while PHP decrement operators are used to decrement a variable's value.

Operator	Same as...	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

Basic Program using PHP

1. Check Even or Odd Number

```
<?php

$num = 7;
```

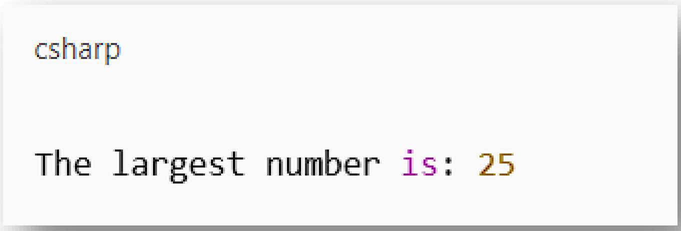


```
csharp

7 is an Odd number
```

```
 if ($num % 2 == 0)
 {
echo "$num is an Even number";
 } else {
echo "$num is an Odd number";
 }
?>
```

2. Find the Largest of Three Numbers

```
<?php
$a = 15;
$b = 25;
$c = 20;

$largest = max($a, $b, $c);
echo "The largest number is: $largest";
?>
```



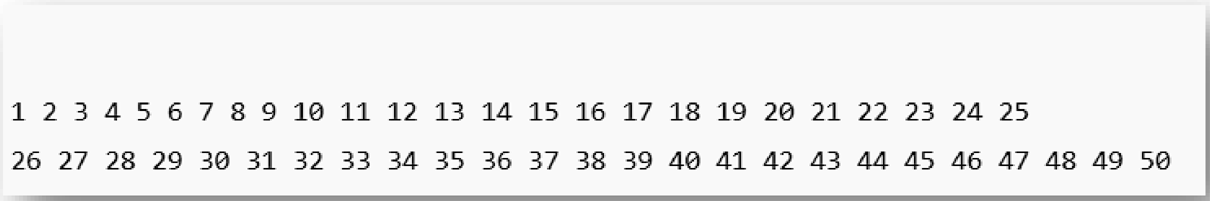
csharp

```
The largest number is: 25
```

3. Print Numbers from 1 to 50 Using Loop

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Loop in PHP</title>
</head>
<body>
  <h2>Numbers from 1 to 50:</h2>
  <p>
  <?php
    for ($i = 1; $i <= 50; $i++) {
      echo "$i ";
    }
  ?>
  </p>
</body>
</html>
```

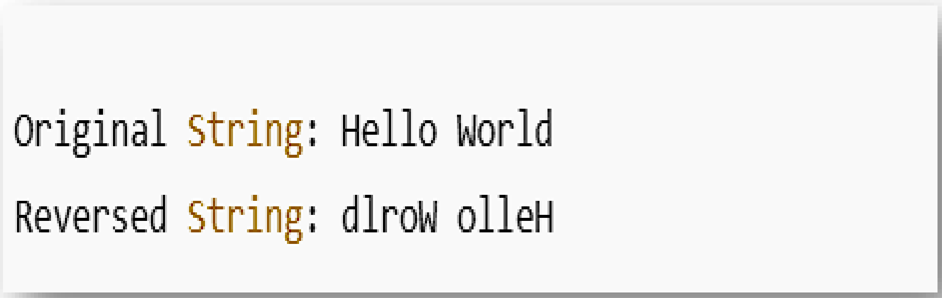


```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

4. Reverse a String

```
<!DOCTYPE html>
<html>
<head>
    <title>Reverse a String</title>
```

```
</head>
<body>
    <h2>Reverse a String</h2>
    <?php
        $str = "Hello World";
        $rev_str = strrev($str);
        echo "<p>Original String: <b>$str</b></p>";
        echo "<p>Reversed String: <b>$rev_str</b></p>";
    ?>
</body>
</html>
```

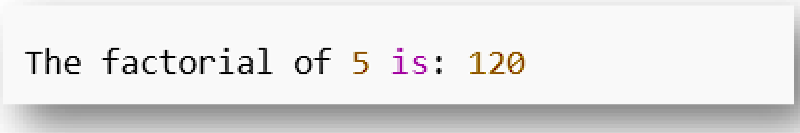


```
Original String: Hello World
Reversed String: dlroW olleH
```

5. Factorial of a Number

```
<!DOCTYPE html>
<html>
<head>
    <title>Factorial Calculation</title>
</head>
```

```
<body>
   <h2>Factorial of a Number</h2>
   <?php
      function factorial($num) {
         if ($num == 0 || $num == 1) {
            return 1;
         }
         return $num * factorial($num - 1);
      }
      $number = 5; // Change this number for different results
      $fact = factorial($number);
      echo "<p>The factorial of $number is:
<b>$fact</b></p>";
   ?>
</body>
</html>
```



```
The factorial of 5 is: 120
```

PHP + MySQL Database System

PHP and MySQL together form a powerful **backend system** for web applications. **PHP** is a **server-side scripting language** used to create dynamic web pages, while **MySQL** is a **relational database management system (RDBMS)** used to store and manage data efficiently. PHP can connect to MySQL using built-in functions like `mysqli_connect()` or PDO, allowing developers to perform **CRUD operations** (Create, Read, Update, Delete) on the database. This combination is widely used in **CMS platforms**, e-commerce sites, and user authentication systems. With PHP and MySQL, developers can build **secure, scalable, and interactive web applications**.

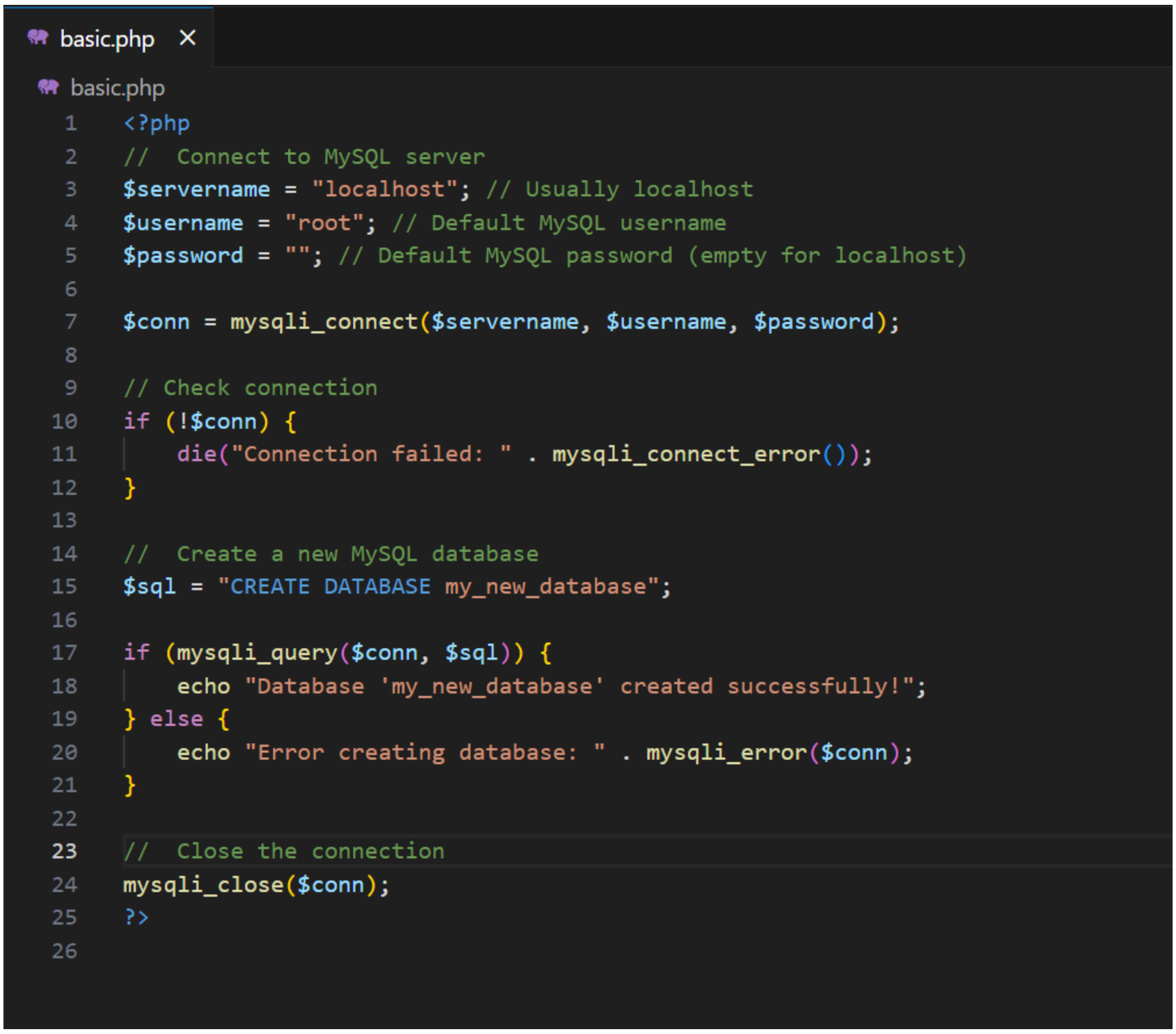
PHP Create a MySQL Database

PHP can be used to create a MySQL database using the **`mysqli`** extension. The `mysqli` (MySQL Improved) extension provides a set of functions that allow you to interact with the MySQL database server. To create a database, PHP connects to the MySQL server, sends an SQL query to create the database, and checks for successful execution.

Steps to Create a MySQL Database with PHP:

I. **Connect to the MySQL server** using PHP.
II. **Send the SQL query** to create a new database.
III. **Check for success** and handle errors if any.

Example:



basic.php

```php
<?php
//  Connect to MySQL server
$servername = "localhost"; // Usually localhost
$username = "root"; // Default MySQL username
$password = ""; // Default MySQL password (empty for localhost)

$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

//  Create a new MySQL database
$sql = "CREATE DATABASE my_new_database";

if (mysqli_query($conn, $sql)) {
    echo "Database 'my_new_database' created successfully!";
} else {
    echo "Error creating database: " . mysqli_error($conn);
}

//  Close the connection
mysqli_close($conn);
?>
```

Explanation of Code:

1. **Connect to MySQL Server**: We use `mysqli_connect()` to establish a connection with the MySQL server using the server name, username, and password.
2. **SQL Query to Create Database**: The SQL query `"CREATE DATABASE my_new_database"` tells MySQL to create a new database named `my_new_database`.
3. **Error Handling**: After executing the query, the code checks whether the database was created successfully. If not, it displays the error message.
4. **Close Connection**: Finally, the connection to the MySQL server is closed using `mysqli_close()`.

Expected Output:

If the database is created successfully, the output will be:

```nginx
Database 'my_new_database' created successfully!
```

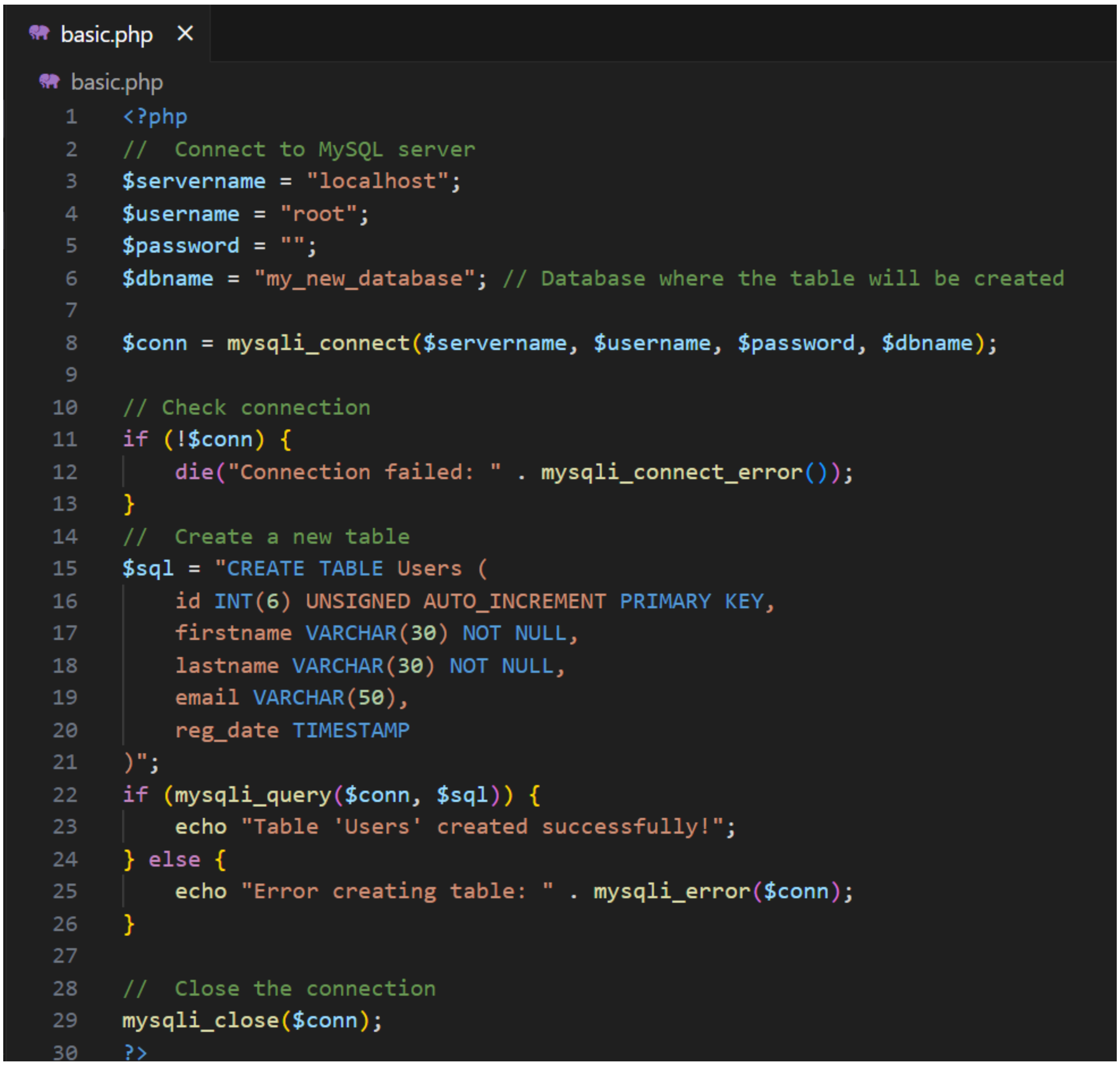
If there is any issue with the SQL query or connection, the error message will be displayed, such as:

```pgsql
Error creating database: Database 'my_new_database' already exists
```

PHP MySQL Create Table

We can use PHP to create a table in a MySQL database by executing a `CREATE TABLE` query. This involves connecting to the MySQL database and running an SQL statement to define the structure of the table, such as columns, data types, and constraints.

Example:



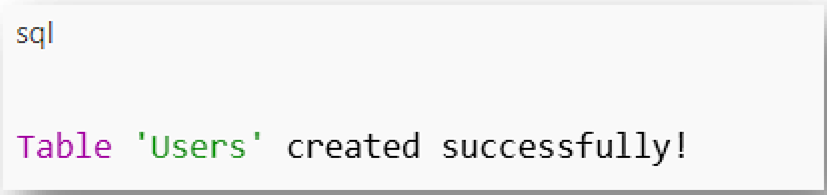
basic.php ×

basic.php

```php
<?php
//  Connect to MySQL server
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "my_new_database"; // Database where the table will be created

$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
//  Create a new table
$sql = "CREATE TABLE Users (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    email VARCHAR(50),
    reg_date TIMESTAMP
)";
if (mysqli_query($conn, $sql)) {
    echo "Table 'Users' created successfully!";
} else {
    echo "Error creating table: " . mysqli_error($conn);
}

//  Close the connection
mysqli_close($conn);
?>
```

Expected Output:

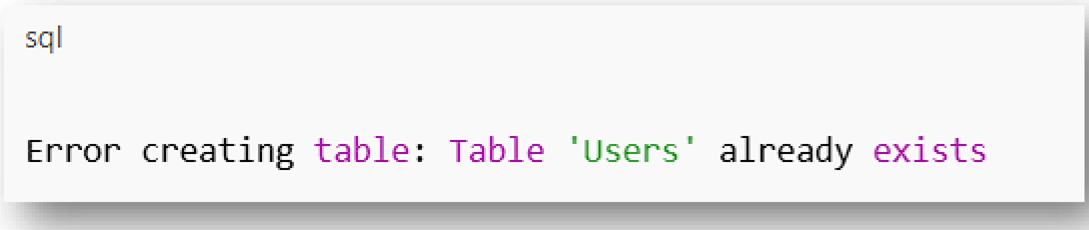
If the table is created successfully:



sql

```sql
Table 'Users' created successfully!
```

If there is an issue (e.g., the table already exists):



```sql
Error creating table: Table 'Users' already exists
```

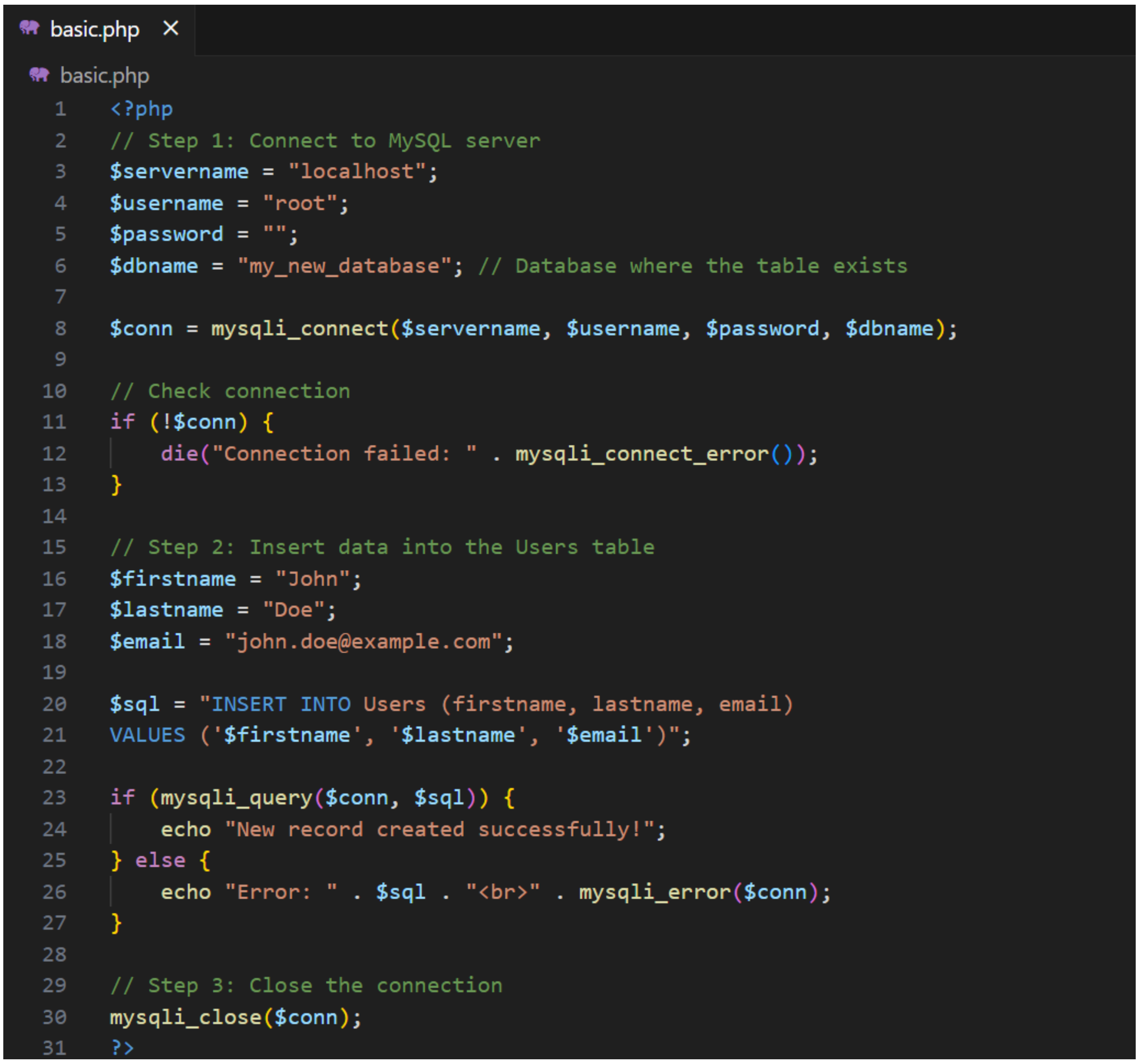
PHP MySQL Insert Data

We can use PHP to insert data into a MySQL table by executing an **`INSERT INTO`** SQL query. This involves connecting to the MySQL database, preparing the **INSERT** statement, and executing it with the data we want to insert.

Steps to Insert Data into a MySQL Table using PHP:

1. **Connect to the MySQL database**.
2. **Prepare an SQL `INSERT INTO` query** with the data.
3. **Execute the query** and check for success.
4. **Close the database connection**.

Example:



```php
<?php
// Step 1: Connect to MySQL server
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "my_new_database"; // Database where the table exists

$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Step 2: Insert data into the Users table
$firstname = "John";
$lastname = "Doe";
$email = "john.doe@example.com";

$sql = "INSERT INTO Users (firstname, lastname, email)
VALUES ('$firstname', '$lastname', '$email')";

if (mysqli_query($conn, $sql)) {
    echo "New record created successfully!";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

// Step 3: Close the connection
mysqli_close($conn);
?>
```

PHP MySQL Delete Data

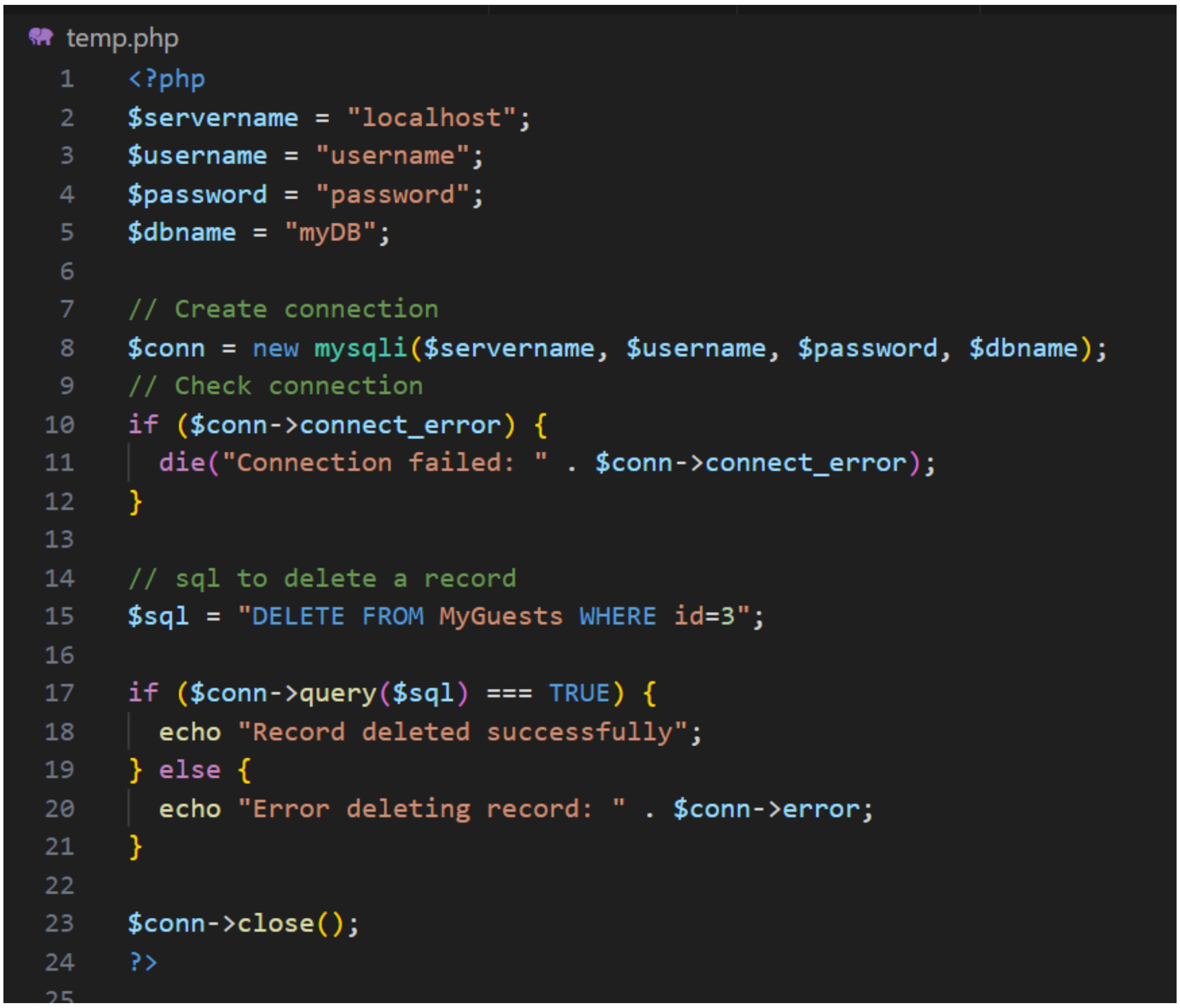
The **DELETE** statement is used to delete records from a table:

```sql
DELETE FROM table_name
WHERE some_column = some_value
```

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

The following examples delete the record with id=3 in the "MyGuests" table:



temp.php

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
  echo "Record deleted successfully";
} else {
  echo "Error deleting record: " . $conn->error;
}

$conn->close();
?>
```

After the record is deleted, the table will look like this:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

Introduction of C Programming

C is a general-purpose, procedural programming language that was developed in the early 1970s by **Dennis Ritchie** at **Bell**

Labs. It is one of the most widely used programming languages and serves as the foundation for many modern languages like C++, Java, and Python.

Why Learn C?

1. **Fast and Efficient** – C provides low-level access to memory and is highly efficient, making it ideal for system programming.
2. **Portable** – Programs written in C can run on different platforms with minimal or no modifications.
3. **Foundation for Other Languages** – Many programming languages derive syntax and concepts from C.
4. **Used in Various Applications** – C is widely used in operating systems, embedded systems, databases, and game development.

Features of C Language

1. **Simple and Structured** – C follows a structured programming approach, making it easy to understand and modify.
2. **Rich Library** – It has a standard library with built-in functions for operations like I/O, math, and string handling.
3. **Fast Execution** – C programs compile and execute quickly due to its minimal runtime overhead.
4. **Low-Level Programming Support** – C allows direct interaction with hardware through pointers and memory management.
5. **Modular Approach** – Large programs can be divided into functions, improving reusability and readability.

Introduction of Function

A **function** in C is a self-contained block of code that performs a specific task and can be reused multiple times. It helps in modular programming by breaking complex problems into smaller, manageable parts. A function in C consists of a function name, return type, parameters (optional), and a function body enclosed in curly braces **`{}`**. The **`main()`** function is the entry point of every C program, while user-defined functions can be created using the **`returnType functionName(parameters) { //code }`** format. Functions improve code readability, reusability, and debugging efficiency.

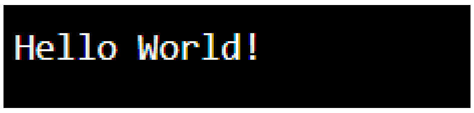
Predefined Functions

Predefined functions in C are built-in functions provided by standard libraries to perform common operations like input/output, mathematical calculations, and string handling. These functions help programmers avoid writing complex code from scratch.

For Example : main() is a function, which is used to execute code, and **printf()** is a function; used to output/print text to the screen,

Example

```
#include <stdio.h>


int main() {
  printf("Hello World!");
  return 0;
}
```

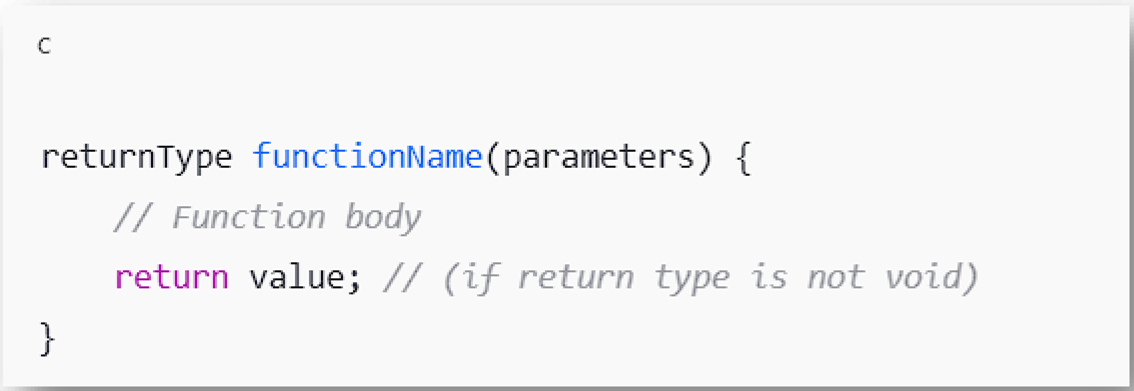


```
Hello World!
```

User-Defined Function

A **user-defined function** in C is a function that is created by the programmer to perform a specific task. It helps in modular programming, making the code more structured, reusable, and easy to debug.

Syntax:

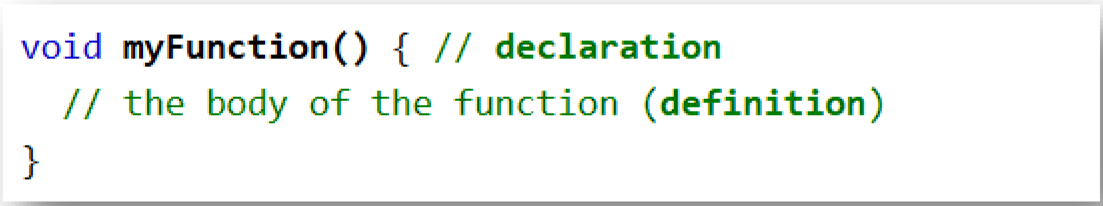


```c
c

returnType functionName(parameters) {
    // Function body
    return value; // (if return type is not void)
}
```

Function Declaration and Definition

A function consist of two parts:

- **Declaration:** the function's name, return type, and parameters (if any)
- **Definition:** the body of the function (code to be executed)



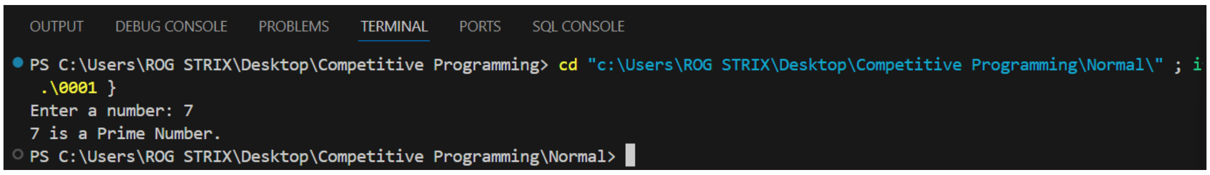
```c
void myFunction() { // declaration
  // the body of the function (definition)
}
```

Programs Using Function:

1. Prime Number Check

Code:

```
#include <stdio.h>

// Function to check if a number is prime
int isPrime(int num) {
    if (num < 2) return 0;
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) return 0;
    }
    return 1;
}


int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (isPrime(num))
        printf("%d is a Prime Number.\n", num);
    else
        printf("%d is not a Prime Number.\n", num);

    return 0;
}
```

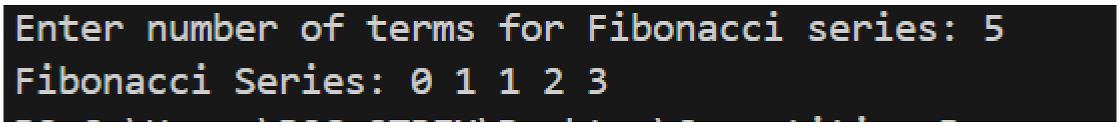


```
PS C:\Users\ROG STRIX\Desktop\Competitive Programming> cd "c:\Users\ROG STRIX\Desktop\Competitive Programming\Normal\" ; i
 .\0001 }
Enter a number: 7
7 is a Prime Number.
PS C:\Users\ROG STRIX\Desktop\Competitive Programming\Normal>
```

2. Fibonacci Series

Code:

```c
#include <stdio.h>

// Function to print Fibonacci series
void fibonacci(int n) {
    int a = 0, b = 1, next;
    printf("Fibonacci Series: %d %d", a, b);
    for (int i = 2; i < n; i++) {
        next = a + b;
        printf(" %d", next);
        a = b;
        b = next;
    }
    printf("\n");
}

int main() {
    int n;
    printf("Enter number of terms for Fibonacci series: ");
    scanf("%d", &n);

    if (n < 2)
        printf("Fibonacci series needs at least 2 terms.\n");
    else
        fibonacci(n);

    return 0;
}
```



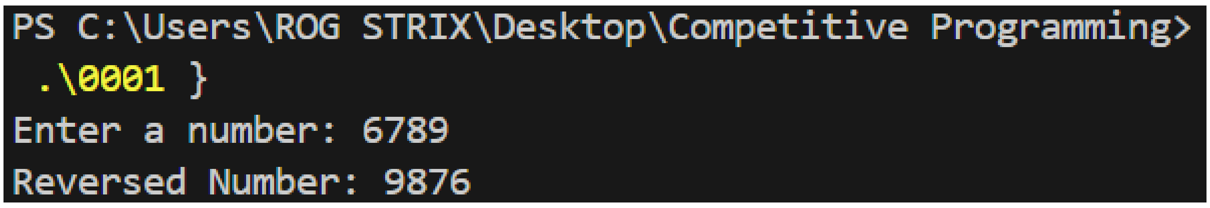
```
Enter number of terms for Fibonacci series: 5
Fibonacci Series: 0 1 1 2 3
```

3. Reverse a Number

Code:

```c
#include <stdio.h>
```

```
// Function to reverse a number
int reverseNumber(int num) {
   int rev = 0;
   while (num > 0) {
      rev = rev * 10 + (num % 10);
      num /= 10;
   }
   return rev;
}

int main() {
   int num;
   printf("Enter a number: ");
   scanf("%d", &num);
   printf("Reversed Number: %d\n", reverseNumber(num));
   return 0;
}
```



```
PS C:\Users\ROG STRIX\Desktop\Competitive Programming>
 .\0001 }
Enter a number: 6789
Reversed Number: 9876
```

Definition of Recursion in C

Recursion in C is a programming technique where a function **calls itself** to solve a problem. It continues calling itself until it reaches a **base case**, which stops further recursive calls.

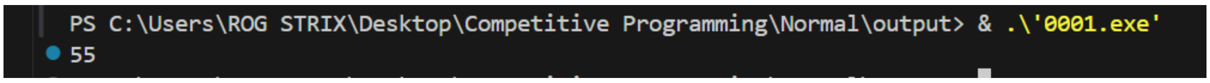
Key Components of Recursion:

1. **Base Case:** The condition that stops recursion.
2. **Recursive Case:** The function calls itself with a modified argument.

Example: Sum of K Number

Code:

```
#include <stdio.h>
int sum(int k);

int main() {
  int result = sum(10);
  printf("%d", result);
  return 0;
}

int sum(int k) {
  if (k > 0) {
    return k + sum(k - 1);
  } else {
    return 0;
  }
}
```

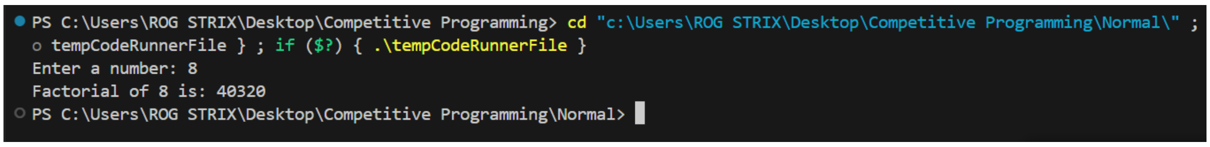


```
PS C:\Users\ROG STRIX\Desktop\Competitive Programming\Normal\output> & .\'0001.exe'
55
```

Example: Find Factorial Number

Code:

```
#include <stdio.h>
long long factorial(int num) {
   if (num == 0 || num == 1)  // Base Case
      return 1;
   return num * factorial(num - 1);  // Recursive Case
}

int main() {
   int num;
   printf("Enter a number: ");
   scanf("%d", &num);

   printf("Factorial of %d is: %lld\n", num, factorial(num));
   return 0;
}
```



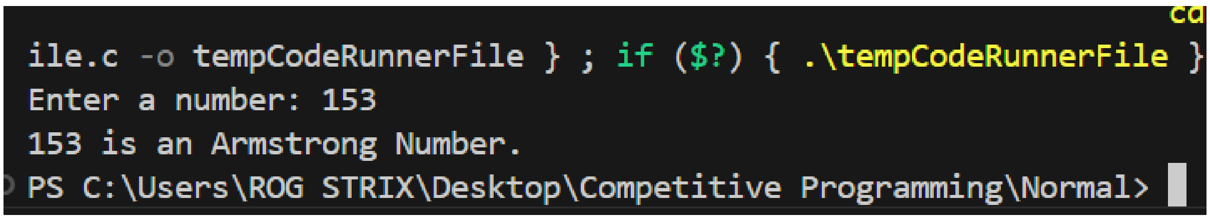
```
● PS C:\Users\ROG STRIX\Desktop\Competitive Programming> cd "c:\Users\ROG STRIX\Desktop\Competitive Programming\Normal\" ;
  o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
  Enter a number: 8
  Factorial of 8 is: 40320
○ PS C:\Users\ROG STRIX\Desktop\Competitive Programming\Normal>
```

Example: Armstrong Number Check using Recursion

Code:

```
#include <stdio.h>

#include <math.h>
// Function to count the number of digits (Recursive)
int countDigits(int num) {
   if (num == 0)
      return 0;
   return 1 + countDigits(num / 10);
}

// Function to calculate Armstrong sum (Recursive)
int armstrongSum(int num, int digits) {
   if (num == 0)
      return 0;
   int digit = num % 10;
   return pow(digit, digits) + armstrongSum(num / 10, digits);
}
```

```
// Function to check if a number is Armstrong
int isArmstrong(int num) {
  int digits = countDigits(num);
  return num == armstrongSum(num, digits);
}

int main() {
  int num;
  printf("Enter a number: ");
  scanf("%d", &num);

  if (isArmstrong(num))
    printf("%d is an Armstrong Number.\n", num);
  else
    printf("%d is not an Armstrong Number.\n", num);

  return 0;
}
```



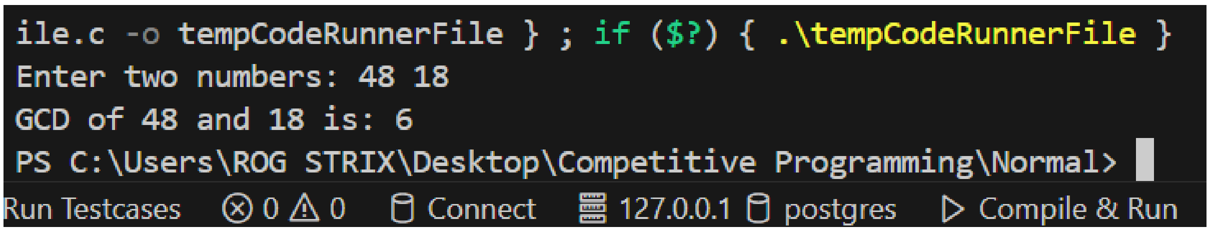
```
ile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a number: 153
153 is an Armstrong Number.
PS C:\Users\ROG STRIX\Desktop\Competitive Programming\Normal>
```

Greatest Common Divisor (GCD) using Recursion

The **GCD** of two integers is the largest integer that divides both numbers without leaving a remainder. It is also known as the **Greatest Common Factor (GCF)** or **Highest Common Factor (HCF)**.

Code:

```
#include <stdio.h>
// Function to find GCD using recursion
int gcd(int a, int b) {
    if (b == 0)
        return a;  // Base case
    return gcd(b, a % b);  // Recursive case
}

int main() {
    int num1, num2;
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    printf("GCD of %d and %d is: %d\n", num1, num2, gcd(num1, num2));
    return 0;
}
```

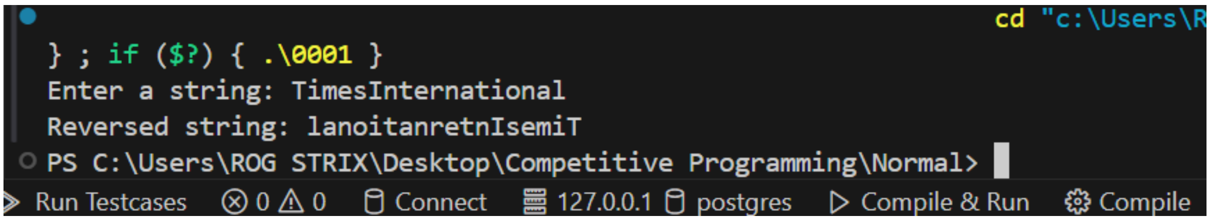


```
ile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter two numbers: 48 18
GCD of 48 and 18 is: 6
PS C:\Users\ROG STRIX\Desktop\Competitive Programming\Normal>
```

Reverse a String using Recursion

Code:

```
#include <stdio.h>
#include <string.h>
// Function to reverse a string using recursion
void reverseString(char str[], int start, int end) {
    if (start >= end)
        return; // Base case
    // Swap characters at start and end
    char temp = str[start];
    str[start] = str[end];
    str[end] = temp;
    // Recursive call
    reverseString(str, start + 1, end - 1);
}
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int len = strlen(str);
    reverseString(str, 0, len - 1);
    printf("Reversed string: %s\n", str);
    return 0;
}
```



```
                                                              cd "c:\Users\R
} ; if ($?) { .\0001 }
Enter a string: TimesInternational
Reversed string: lanoitanretnIsemiT
PS C:\Users\ROG STRIX\Desktop\Competitive Programming\Normal>
```

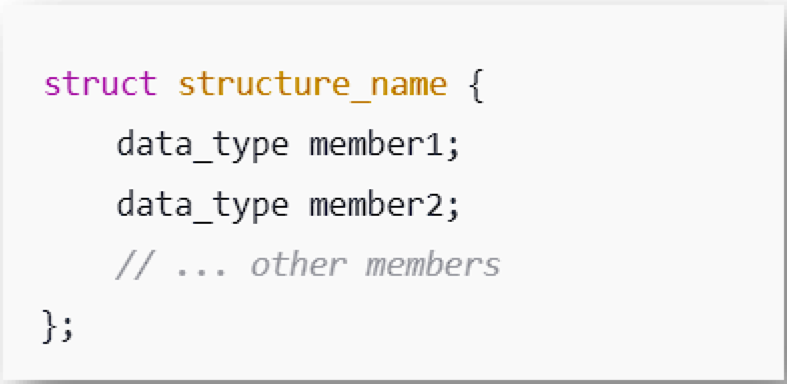
Introduction of Structures

In C programming, a **structure** is a user-defined data type that allows grouping of variables of different data types under a single name. Structures are used to represent **a record**, such as a student's information, an employee's details, etc. Unlike arrays, structures can store different types of data (integers, floats, strings, etc.) in one unit.

Why Use Structures?

- **Organizes data:** Helps store related data together.

- **Modular programming:** Makes the code more readable and manageable by grouping different types of data.
- **Flexible:** Allows grouping of different data types into a single unit.



```
struct structure_name {
    data_type member1;
    data_type member2;
    // ... other members
};
```

Structure Memory Allocation:

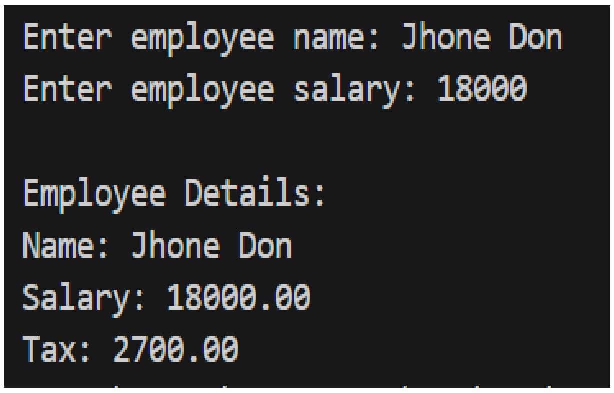
- Memory for a structure is allocated as the sum of the memory required for each member.
- The structure can store data of various types, and each member occupies the appropriate amount of memory depending on its type.

Calculate Tax for Each Employee Using Structure

Code:

```
#include <stdio.h>
// Define a structure to store employee details
struct Employee {
  char name[50];
  float salary;
  float tax;
};
```

```
// Function to calculate tax based on salary
float calculateTax(float salary) {
  if (salary <= 5000)
    return salary * 0.05; // 5% tax for salary <= 5000
  else if (salary <= 10000)
    return salary * 0.10; // 10% tax for salary between 5001 and 10000
  else
    return salary * 0.15; // 15% tax for salary > 10000
}

int main() {
  struct Employee emp;
  // Input employee details
  printf("Enter employee name: ");
  fgets(emp.name, sizeof(emp.name), stdin);
  printf("Enter employee salary: ");
  scanf("%f", &emp.salary);
  // Calculate tax
  emp.tax = calculateTax(emp.salary);
  // Display employee details and tax
  printf("\nEmployee Details:\n");
  printf("Name: %s", emp.name);
  printf("Salary: %.2f\n", emp.salary);
  printf("Tax: %.2f\n", emp.tax);

  return 0;
}
```



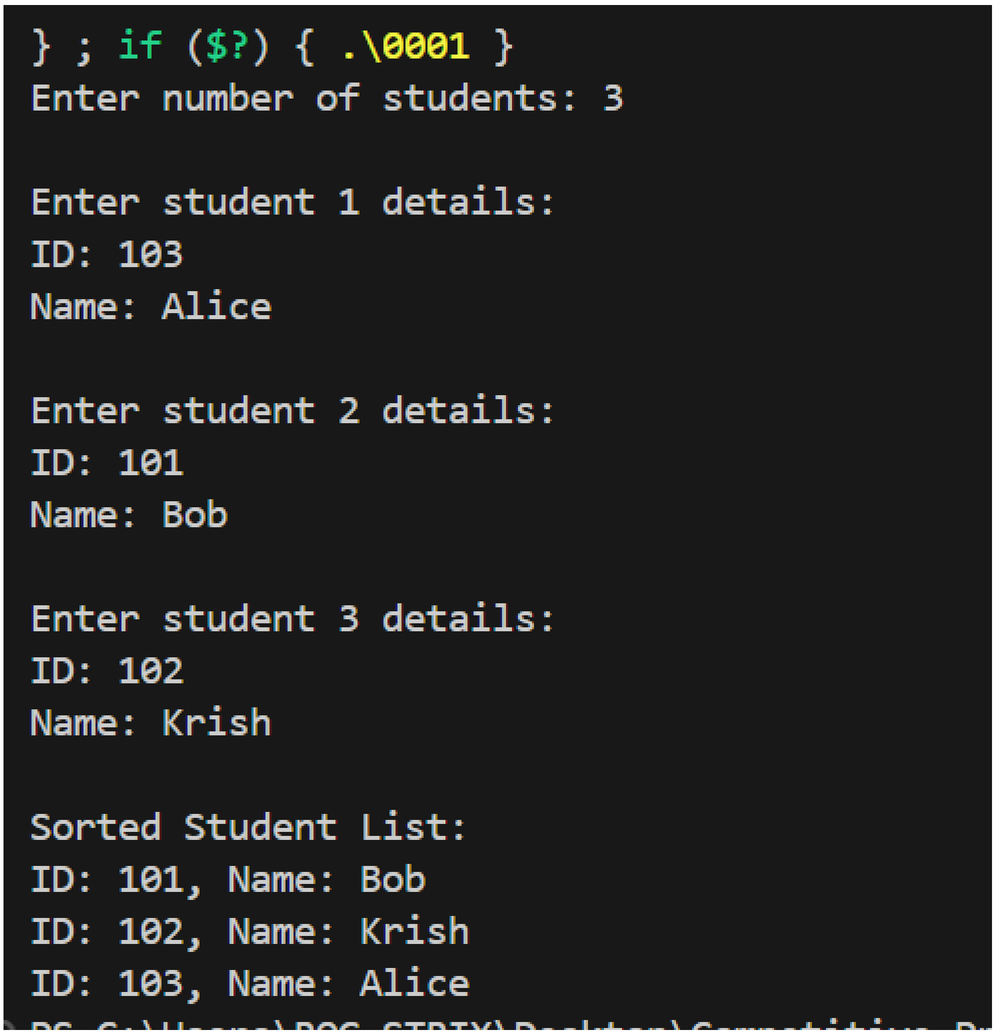
```
Enter employee name: Jhone Don
Enter employee salary: 18000

Employee Details:
Name: Jhone Don
Salary: 18000.00
Tax: 2700.00
```

Sort Students by Their ID Using Structure

Code:

```
#include <stdio.h>
// Define a structure to store student details
struct Student {
  int id;
```

```
    char name[50];
};

// Function to swap two students
void swap(struct Student* a, struct Student* b) {
    struct Student temp = *a;
    *a = *b;
    *b = temp;
}

// Function to sort students by their ID
void sortStudents(struct Student students[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (students[j].id > students[j+1].id) {
                swap(&students[j], &students[j+1]);
            }
        }
    }
}
int main() {
    int n;
    printf("Enter number of students: ");
    scanf("%d", &n);
    struct Student students[n];
    // Input student details
    for (int i = 0; i < n; i++) {
        printf("\nEnter student %d details:\n", i+1);
        printf("ID: ");
        scanf("%d", &students[i].id);
        getchar();  // To consume the newline character
        printf("Name: ");
        fgets(students[i].name, sizeof(students[i].name), stdin);
    }
    // Sort students by ID
    sortStudents(students, n);

    // Display sorted student details
    printf("\nSorted Student List:\n");
    for (int i = 0; i < n; i++) {
        printf("ID: %d, Name: %s", students[i].id, students[i].name);
    }

    return 0;
}
```



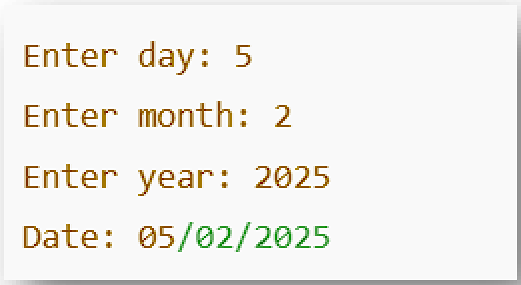
```
} ; if ($?) { .\0001 }
Enter number of students: 3

Enter student 1 details:
ID: 103
Name: Alice

Enter student 2 details:
ID: 101
Name: Bob

Enter student 3 details:
ID: 102
Name: Krish

Sorted Student List:
ID: 101, Name: Bob
ID: 102, Name: Krish
ID: 103, Name: Alice
```

Store and Display Dates Using Structure

Code:

```
#include <stdio.h>
// Define a structure to store date
```

```
struct Date {
    int day;
    int month;
    int year;
};

// Function to display the date
void displayDate(struct Date date) {
    printf("Date: %02d/%02d/%04d\n", date.day, date.month, date.year);
}

int main() {
    struct Date today;
    // Input date details
    printf("Enter day: ");
    scanf("%d", &today.day);
    printf("Enter month: ");
    scanf("%d", &today.month);
    printf("Enter year: ");
    scanf("%d", &today.year);

    // Display the date
    displayDate(today);

    return 0;
}
```



```
Enter day: 5
Enter month: 2
Enter year: 2025
Date: 05/02/2025
```