**Ruby Introduction**

1. Features

2. Installation

3. gets puts

4. Comments

5. Ruby Program

**Basic Programs in Ruby covering Introduction:**

1. Program to print "Hello! world".

2. Program to print "Hello" with a given name.

**Ruby Control Statements**

1. if-else

2. case

3. for

4. while

5. do-while

6. until

7. break

8. next

9. redo

10. retry

11. Assignment

**Basic Programs in Ruby covering Control Statements:**

1. Program to print a multiplication table.

2. Program to print numbers from 1 to 9 like a matrix.

3. Program to find the maximum of three numbers.

4. Program to determine if a given number is prime.

**Ruby Data Types**

1. Numbers: to\_i, to\_f, round, ceil, floor, abs, even?, odd?, zero?, positive?, negative?, to\_s, times, pred, succ, between?, to\_r, gcd, lcm, next\_float

2. Strings: length, size, empty?, upcase, downcase, reverse, strip, include, split, concat, +, to\_i, to\_f, start\_with, end\_with, gsub!, sub, slice or [, chomp, to\_sym

3. Hashes: [:key], [:key]=, keys, values, has\_key? or key? or include?, has\_value?, value?, empty?, size or length, delete, merge, each, each\_key, each\_value, fetch, to\_a, to\_h, select, reject

4. Arrays: ['index'], ['index']=, length or size, empty?, push or <<, pop, unshift, shift, concat, join, include?, index, reverse, sort, uniq, delete, each, map, collect

5. Booleans

6. Ranges

7. Date

8. Time

**Basic Programs in Ruby covering Data Types:**

1. Program to determine if a given number is even or odd.

2. Write a function to check if a number is positive, negative, or zero.

3. Write a function to calculate the square root of a number.

4. Program to check if a year is a leap year.

1. Write a function to concatenate two strings.

2. Write a function to find the length of a string.

3. Write a function to reverse a string.

4. Program to check if a string is a palindrome.

1. Write a function to create an array and access its values.

2. Write a function to iterate over an array.

3. Write a function to add two arrays.

4. Write a function to find unique elements in two arrays.

1. Write a function to create a hash and access its values.

2. Write a function to iterate over a hash.

3. Write a function to merge two hashes.

4. Write a function to find the occurrence of a character in a given string.

1. Write a function to create a range and access its values.

2. Write a function to iterate over a range of numbers.

3. Write a function to determine if a given element is within the range.

4. Write a function to find the size of the range.

1. Write a function to display the current date.

2. Write a function to calculate the age from the date of birth.

3. Write a function to calculate the number of days between two dates.

4. Write a function to display the date from two days ago.

1. Write a function to display the current time.

2. Write a function to add a given number of minutes to the current time.

3. Write a function to add a given number of hours to the current time.

4. Write a function to subtract a given number of seconds from the current time.

**Ruby Operators**

1. Unary operator

2. Arithmetic operator

3. Bitwise operator

4. Logical operator

5. Ternary operator

6. Assignment operator

7. Comparison operator

8. Range operator

**Basic Programs in Ruby covering Operators:**

1. Program to perform arithmetic operations:

- Write a function to sum two numbers.

- Write a function to subtract.

- Write a function to find the division between two numbers.

- Write a function to find the multiplication of two numbers.

- Write a function to determine if a number is even or odd.

2. Program to calculate the factorial of a number.

3. Program to find the maximum of three numbers using the Ternary operator.

4. Program to check if a year is a leap year.

5. Program to find the largest element in an array.

**Ruby Core**

1. Modules

2. Class

3. Object

4. Blocks

5. yield

6. self

7. super

8. Symbol

9. Iterators

10. File I/O

**Basic Programs in Ruby covering Class, Object, Modules, and Blocks:**

1. Create a class called "Person" with attributes name and age. Write a method that displays the person's name and age. Use a module to include additional behaviors like "walk" and "talk".

2. Create a class called "BankAccount" with attributes account\_number and balance. Implement methods to deposit and withdraw money from the account. Use a block to execute additional actions when a transaction is made.

3. Create a module called "MathHelper" that contains methods for calculating the square and cube of a number. Include this module in a class called "Calculator" to perform mathematical operations.

4. Create a class called "Student" with attributes name, grade, and subjects. Write a method that displays the student's name, grade, and a list of subjects. Use a block to iterate over the subjects and perform a specific action.

5. Create a module called "Logger" that provides logging functionality to record events. Include this module in multiple classes to log different types of activities.

6. Create a class called "Product" with attributes name, price, and quantity. Implement methods to calculate the total price and apply a discount using a block to determine the discount amount.

**Basic Programs in Ruby covering yield, self, and super:**

1. Create a class called "Counter" with an attribute called "count." Implement a method called "increment" that increases the count by 1. Use the "yield" keyword to execute a block of code after each increment.

2. Create a class called "Person" with attributes name and age. Implement a method called "introduce" that displays the person's name and age using the "self" keyword.

3. Create a class called "Vehicle" with a method called "start\_engine." Create a subclass called "Car" that inherits from "Vehicle." Override the "start\_engine" method in the "Car" class to display a message and then call the parent class's "start\_engine" method using the "super" keyword.

**Basic Programs in Ruby covering Symbol and Iterators:**

1. Create a method called "calculate" that takes two numbers and a symbol as arguments. Implement a case statement to perform different mathematical operations based on the symbol (+, -, \*, /) and return the result.

2. Create an array of fruits and use the "each" iterator to iterate over the array and display each fruit's name.

**Basic Programs in Ruby covering File I/O and Directories:**

1. Create a program that reads a text file and counts the number of lines in the file.

2. Create a program that writes a list of names to a new text file.

3. Create a program that reads a directory and lists all the files and directories within it.

**Ruby Variables**

1. Local Variables

2. Class Variables

3. Instance Variables

4. Global Variables

**Basic Programs in Ruby covering Local, Global, Class, and Instance Variables:**

1. Create a method that accepts a number as an argument and prints its square. Use a local variable to store the square value and display it.

**\*\*\***2. Create a class called "Counter" with a class variable called "count" initialized to 0. Implement instance methods to increment and decrement the count. Display the count using both the class and instance variables.

3. Create a class called "Person" with an instance variable called "name." Implement getter and setter methods for the name. Use the setter method to set the name and display it using the getter method.

4. Create a global variable called $message and assign it a string value. Display the value of the global variable in a method.

5. Create a method that accepts two numbers as arguments and multiplies them. Use a local variable to store the result and return it.

**Ruby Methods**

1. Overview

2. initialize

3. attr\_accessor

4. attr\_writer

5. attr\_reader

6. Class Method

7. Instance Method

**Basic Programs in Ruby covering initialize, attr\_reader, attr\_writer, and attr\_accessor:**

1. Create a class called "Person" with attributes name and age. Use attr\_accessor to define getter and setter methods for both attributes. Implement an initialize method to set the initial values of the attributes.

2. Create a class called "Car" with attributes make and model. Use attr\_reader to define a getter method for make and attr\_writer to define a setter method for model. Implement an initialize method to set the initial values of the attributes.

3. Create a class called "Rectangle" with attributes length and width. Use attr\_accessor to define getter and setter methods for both attributes. Implement an initialize method to set the initial values of the attributes. Also, add a method called calculate\_area that returns the area of the rectangle.

**Basic Programs in Ruby covering Class and Instance Methods:**

1. Create a class called "MathUtils" with a class method called "square" that takes a number as an argument and returns its square.

2. Create a class called "Circle" with an instance method called "area" that calculates and returns the area of a circle. The radius should be an instance variable.

**Ruby OOPs**

1. Abstraction

2. Encapsulation

3. Inheritance

4. Polymorphism

5. Class Method in Detail

6. Instance Method in Detail

7. Class Variables in Detail

8. Instance Variables in Detail

9. Mixin

**Basic Programs in Ruby covering Abstraction, Encapsulation, Inheritance, and Polymorphism:**

1. Create a class called Shape with an abstract method called calculate\_area. This class should not be instantiated directly. Then, create two subclasses called Rectangle and Circle that inherit from Shape and implement the calculate\_area method accordingly.

2. Create a class called Person with private attributes name and age. Implement getter and setter methods using encapsulation techniques to access and modify these attributes.

3. Create a base class called Vehicle with attributes brand and model. Implement a method called info that displays the brand and model of the vehicle. Then, create a subclass called Car that inherits from Vehicle and adds an additional attribute color. Override the info method in the Car class to display the brand, model, and color of the car.

4. Create a base class called Animal with a method called make\_sound. Then, create two subclasses called Cat and Dog that inherit from Animal and override the make\_sound method to produce the respective sounds of a cat and a dog.

**Basic Programs in Ruby covering Class Method and Instance Method:**

1. Create a class called MathUtils that provides both class methods and instance methods for mathematical operations.

Basic Programs in Ruby covering Class Variables and Instance Variables:

1. Create a class called Car that tracks the total number of cars created using a class variable @@total\_cars and stores the details of each car using instance variables.

**Basic Programs in Ruby covering Mixin:**

1. Create a module called Greeting that defines a method called greet. Then, create two classes called Person and Animal. Include the Greeting module in both classes to provide the greeting behavior.

**Ruby Advance**

1. load vs require

2. include vs extend

3. Exceptions

4. Regex

5. Lambda Proc Block

**Basic Programs in Ruby covering load, require, include, and extend:**

1. Create two Ruby files: file1.rb and file2.rb. In file1.rb, define a method called greeting that prints a greeting message. In file2.rb, use load to load file1.rb and invoke the greeting method.

2. Create two Ruby files: module1.rb and module2.rb. In module1.rb, define a module called Greeting with a method called say\_hello. In module2.rb, use require to load module1.rb and include the Greeting module in a class called Person. Instantiate an object of the Person class and invoke the say\_hello method.

3. Create a class called Printer with an instance method called print\_document. In another class called Formatter, define a module called Format. In the Format module, define a class method called format\_document. Use extend to extend the Format module in the Printer class and invoke the format\_document method from within the print\_document method.

**Basic Programs in Ruby covering Exceptions:**

1. Create a method called calculate\_age that takes a birth year as an argument and calculates the age. If the birth year is in the future or if an invalid input is provided, raise a RuntimeError exception with a custom error message.

2. Create a method called divide that takes two numbers as arguments and performs division. Handle the ZeroDivisionError exception and display a custom error message when division by zero occurs.

**Basic Programs in Ruby covering Regex:**

1. Create a method called validate\_email that takes an email address as an argument and checks if it is valid using a regular expression. Display a message indicating whether the email is valid or not.

2. Create a method called extract\_phone\_numbers that takes a string as an argument and extracts all phone numbers from it using regular expressions. Return an array of the extracted phone numbers.

**Basic Programs in Ruby covering Lambda, Proc, and Block:**

1. Create a lambda that takes two numbers as arguments and returns their sum. Invoke the lambda and display the result.

2. Create a method called multiply\_numbers that takes a block as an argument. Inside the method, prompt the user to enter two numbers and yield them to the block. The block should multiply the numbers and return the result. Call the method and display the product.

3. Create a proc that takes an array as an argument and returns a new array with the elements squared. Invoke the proc and display the new array.

**Ruby Misc**

1. RVM

2. Gems

3. Linter

4. Debugging

**Ruby Assignment**

1. Questions

2. Scenarios

3. Live Coding

**Write a program for a bank system in Ruby:**

Write a Ruby program for banking operations like creating accounts, crediting, debiting, showing details, and deleting accounts. Create a User class with attributes name, address, balance (default 0), and account\_number. Create another class, BankingOperation, where all operation methods (create, credit, debit, delete) will take place. Use a hash as a class variable to store the user object, with the account number as the key.

When the user creates an account, ask for their name and address, set the balance to 0, generate a random account number, and store user info in the hash. For credit and debit operations, ask for the account number and amount from the user. For deletion, ask for the account number and confirmation.

Use a switch case to ask which operation needs to be performed. Users can choose the operation to perform:

1. Open a new account

2. Credit amount

3. Debit amount

4. Account info

5. Exit

1. When the user chooses option 1, we need to:

- Ask for the following information:

- Name

- Address

- Store user info in a hash with a default balance of 0.

- Generate a random account number and use it as a key in the hash.

- Return this account number to the user.

- Display all these options on the console for the next operation.

- The program should not exit until the user chooses the exit operation.

2. To credit the amount in the user's account:

- Ask the user to enter their account number (returned in the first step).

- Add the entered amount to the user's balance.

3. For debit the amount from the user's account:

- Ask the user to enter their account number (returned in the first step).

- Deduct the entered amount from the user's balance.

- Before deducting the balance, check if the user has a sufficient balance. If not, return the message "insufficient balance."

4. To show account details:

- Ask the user to enter the account number (from the first step).

- Display details from the hash that match the entered account number.

5. When the user chooses to exit, terminate the program.