Arrays in Ruby

1. Introduction

Brief Introduction to Data Structures in Ruby

Data structures are essential components of any programming language, allowing developers to store, manipulate, and retrieve data efficiently. In Ruby, data structures like arrays, hashes, and ranges provide flexible and powerful ways to work with collections of data.

Importance of Choosing the Right Data Structure

Choosing the right data structure can significantly impact the performance and readability of your code. Some compelling reasons include:

- **Performance Optimization**: Different structures have different time complexities for operations like searching, inserting, and deleting.
- **Memory Efficiency**: Some structures consume more memory than others; selecting the right one can improve memory usage.
- **Code Simplicity & Maintainability**: The right structure simplifies logic, making code easier to understand and maintain.
- **Scalability**: Proper data structures ensure applications remain efficient as data grows.

Brief Overview of Collections in Ruby (Arrays, Hashes, Ranges)

- **Arrays**: Ordered, indexed collections of elements.
- **Hashes**: Key-value pairs for quick lookups.
- **Ranges**: Represent sequences of numbers or characters.

---

2. Arrays in Ruby (40 min)

What is an Array?

An **array** is an ordered collection of objects, indexed starting from zero.

```ruby
arr = [1, 2, 3, "hello", :symbol]
```

Creating Arrays

Literal Syntax (`[]`)

```ruby
numbers = [1, 2, 3, 4, 5]
```

`%w[]` and `%i[]` shorthand

```ruby
words = %w[apple banana cherry]  # ["apple", "banana", "cherry"]
symbols = %i[red green blue]     # [:red, :green, :blue]
```

Space Complexity and Code Complexity of Arrays

- **Space Complexity**: Arrays require contiguous memory allocation and have a complexity of **O(n)**.
- **Code Complexity**: Searching in an array is **O(n)**, while accessing elements by index is **O(1)**.

Common Array Methods

`push`, `pop`, `shift`, `unshift`

```ruby
arr = [1, 2, 3]
arr.push(4)      # => [1, 2, 3, 4]
arr.pop          # => 4, arr becomes [1, 2, 3]
```

```
arr.shift        # => 1, arr becomes [2, 3]
arr.unshift(0)   # => [0, 2, 3]
```

`first`, `last`, `include?`

```
arr = [10, 20, 30, 40]
arr.first       # => 10
arr.last        # => 40
arr.include?(20) # => true
```

`map`, `select`, `reject`, `compact`

```
numbers = [1, 2, 3, 4, 5]

doubled = numbers.map { |n| n * 2 }   # [2, 4, 6, 8, 10]

evens = numbers.select { |n| n.even? }  # [2, 4]

odds = numbers.reject { |n| n.even? }  # [1, 3, 5]

arr = [1, nil, 2, nil, 3]
arr.compact  # => [1, 2, 3] (removes nil values)
```

Accessing Elements

Using Indexes and Negative Indexes

```
arr = [10, 20, 30, 40, 50]
arr[0]   # => 10
arr[-1]  # => 50
arr[-2]  # => 40
```

Ranges Inside Arrays (`array[1..3]`)

```
arr[1..3] # => [20, 30, 40]
```

Iterating Over Arrays

`each`, `map`, `reduce`

```
arr = [1, 2, 3, 4]

arr.each { |n| puts n }  # Prints each number

squared = arr.map { |n| n**2 }  # [1, 4, 9, 16]

sum = arr.reduce(0) { |sum, n| sum + n }  # 10 (sum of all elements)
```

Intermediate/Advanced Topics

1. Using `zip` to Merge Two Arrays

```
arr1 = [1, 2, 3]
arr2 = ['a', 'b', 'c']
merged = arr1.zip(arr2)  # => [[1, "a"], [2, "b"], [3, "c"]]
```

2. `group_by` for Categorization

```
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
grouped = arr.group_by { |n| n.even? }  # {false=>[1, 3, 5, 7, 9], true=>[2, 4
```

Challenges

1. Add Elements to an Array

Write a method that adds elements 4 and 5 to an existing array [1, 2, 3] using push, then returns the modified array.

```ruby
def add_elements(arr)
  # Your code here
end

p add_elements([1, 2, 3]) # Expected output: [1, 2, 3, 4, 5]
```

2. Remove the Last Element

Write a method that removes and returns the last element of an array using pop.

```ruby
def remove_last(arr)
  # Your code here
end

p remove_last([10, 20, 30]) # Expected output: 30
```

3. Remove the first element

Write a method that removes the first element from an array using shift and returns the modified array.

```ruby
def remove_first(arr)
  # Your code here
end

p remove_first(["apple", "banana", "cherry"]) # Expected output: ["banana", "
```

4. Unshift Elements into an Array

Write a method that inserts “start” at the beginning of an array using unshift.

```ruby
def add_at_start(arr)
  # Your code here
end

p add_at_start(["middle", "end"]) # Expected output: ["start", "middle", "end'
```

5. Remove Duplicate Elements Using uniq

Write a method that removes duplicate elements from an array using uniq.

```ruby
def remove_duplicates(arr)
  # Your code here
end

p remove_duplicates([1, 2, 2, 3, 4, 4, 5]) # Expected output: [1, 2, 3, 4, 5]
```

6. Get the First Element

Write a method that returns the first element of an array using first.

```ruby
def get_first(arr)
  # Your code here
end

p get_first([100, 200, 300]) # Expected output: 100
```

7. Get the Last Element

Write a method that returns the last element of an array using last.

```ruby
def get_last(arr)
  # Your code here
end

p get_last(["red", "blue", "green"]) # Expected output: "green"
```

8. Access an Element by Index

Write a method that returns the second element of an array using [].

```ruby
def get_second(arr)
  # Your code here
end

p get_second(["a", "b", "c", "d"]) # Expected output: "b"
```

9. Check if an Element Exists

Write a method that checks if the number 7 is in an array using include?.

```ruby
def contains_seven?(arr)
  # Your code here
end

p contains_seven?([1, 3, 5, 7, 9]) # Expected output: true
p contains_seven?([2, 4, 6, 8])    # Expected output: false
```

10. Double Each Element Using map

Write a method that takes an array of numbers and returns a new array where each number is doubled.

```ruby
def double_numbers(arr)
  # Your code here
end

p double_numbers([1, 2, 3, 4]) # Expected output: [2, 4, 6, 8]
```

11. Select Even Numbers

Write a method that selects only even numbers from an array using select.

```ruby
def get_evens(arr)
  # Your code here
end

p get_evens([1, 2, 3, 4, 5, 6]) # Expected output: [2, 4, 6]
```

12. Reject Odd Numbers

Write a method that removes odd numbers from an array using reject.

```
def remove_odds(arr)
  # Your code here
end

p remove_odds([1, 2, 3, 4, 5, 6]) # Expected output: [2, 4, 6]
```

13. Remove nil Values from an Array

Write a method that removes all nil values from an array using compact.

```
def remove_nils(arr)
  # Your code here
end

p remove_nils([1, nil, 2, nil, 3]) # Expected output: [1, 2, 3]
```

14. Create a Unique Sorted List

Write a method that takes two arrays, merges them, removes duplicates, and returns a sorted array.

```
def merge_unique_sorted(arr1, arr2)
  # Your code here
end

p merge_unique_sorted([3, 1, 4], [4, 5, 6, 1]) # Expected output: [1, 3, 4, 5,
```

15. Find the Most Frequent Element

Write a method that returns the most frequently occurring element in an array. If multiple elements have the same frequency, return the first one encountered.

```
def most_frequent(arr)
  # Your code here
end

p most_frequent([1, 2, 3, 2, 4, 2, 5]) # Expected output: 2
p most_frequent(["apple", "banana", "apple", "cherry"]) # Expected output: "ap
```