ruby

Features of Ruby

- Object Oriented
- Portable
- Interpreted
- Garbage Collection
- Exception Handling
- Dynamic Typing
- Case Sensitive
- Flexible
- Visual appearance
- Mixins

Data Types in ruby

- Data types describes type of data stored and how to handle them.
- Ruby has the following data types:
  - Numbers
  - Booleans
  - Strings
  - Symbols
  - Arrays
  - Hashes

Numbers

- Numbers are defined as sequnce of digits seperated by underscore(optional)
- Numbers are further divided as
  - Fixnum
  - Bignum
  - Float
  - Complex
  - Rational
  - BigDecimal

- In ruby generally two type of numbers are used:
  - Finxnum (Integer) or
  - Float

```
distance = 11.2  # float type
time = 2        # integer type
```

Boolen

- Boolean data type represents only one bit of information either **true** or **false**

```
night = true # boolena type
```

Strings

- String represents sequence of character
- Defined by enclosing a text within single (' ') or double (“ ”) quote

```
name = "Arun"
puts name
```

Symbols

- Symbols are light-weight immutable strings.
- A symbol is preceded by a colon [:]
- They take less space and have better performance than strings

```
 person = {:name => "John"}
 puts person[:name]
```

Arrays

- An array is a list of variables enclosed in square brackets and separated by commas.
- They can hold objects like integer, number, hash, string, symbol or any other array.
- Ways to Initialize an array:
    - array = Array.new
    - array = Array.new(5)
    - array = [1,2,3]
    - array = Array(0..9)

```
array = Array(0..10)
puts(array)
```

Hashes

- Hash is a collection of key-value enclosed in curly braces {} and separated by commas.
- The key in hashes are unique otherwise later overrides the former.
- keys and value is separated by **=>** .

```
employee = { "John" => 31,"David" =>25,"Mike" => 23 }
puts(employee)
```

String Concatenation

- String concatenation is used to concatinate two strings
- Methods of String Concatenation:
    - **Using '+' operator**

      ```
      str1 = "hello"
      str2 = "world!"
      str1 = str1 + str2
      ```

    - **Using '+=' operator**

      ```
      str1 = "hello"
      str2 = "world!"
      str1 += str2
      ```

    - **Using concat()**

      ```
      str1 = "hello"
      str2 = "world"
      str1.concat(str2,33)
      ```

Common array methods in ruby

arr=[1,2,3,4,5,6]

Methos	Function	Example
index()	Gives index of an element	`arr.index(1)`
empty ?	Returns whether array is empty or not	`arr.empty?`
push()	Add an element to end of array	`arr.push(7)`
insert(a,b)	Adds b at posion a	`arr.insert(0,10)`
pop()	Removes element from end	`arr.pop()`
delete(a)	Removes element from position a	`arr.delete(1)`
each{}	Iterates over array	`arr.each{ |a| puts a }`
reverse_each{}	Reverse iteration	`arr.each{ |a| puts a }`
include?()	Checks if the element is in array or not	`arr.include?(1)`
flatten	Converts Multidimentional array to 1-D	`arr.flatten`

nil vs false

nil	false
It is not a value	It is a value
It is not a data type	It is a boolean data type
It is an object of NilClass	Object of FalseClass
nil is true in .nil?	false is false in .nil?
nil is false in if condition	false is false in if condition

Ranges in ruby

- Ruby range represents a set of values with a beginning and an end. They can be constructed using s..e and s...e.
  Ruby has a variety of ways to define ranges.
  - Ranges as sequences
  - Ranges as conditions
  - Ranges as intervals
- **.to_a.reverse** is used to return the reverse range

```
a = Array(1..5)
# a= [1, 2, 3, 4, 5]
```

Roles of Modules and Mixins

Modules

- Modules are used for grouping together classes, methods and constants.
- It provides with namespace and removes namespace clashing.
- It implements the mixin facility.
- Module name must start with Uppercase.
- More than 1 module can have function with same name
- Syntax:

```
module Identifier
    statement1
    statement2
    ..........
end
```

- Example:

```
module Moral
    BAD=1
    VERY_BAD=0
    def Moral.sin(badness)
    ..........
    end
end
```

Mixins

- ruby does not support multiple inheritance
- but ruby modules eliminate the need of multiple inheritance by providing **Mixins**
- It gives a controlled way to add multiple inheritance to the classes

```
module A
    def a1
    statement1
    statement2
    ..........
    end
end


module B
    def b1
    statement1
    statement2
    ..........
    end
end

class Example
    include A
    include B

    def e1
    statement
    end
end

Ex = Example.new
Ex.a1
Ex.b1
Ex.e1
```

Programs

1. Prime Number or not

```
def is_prime(num)
    return false if num <= 1
    (2..Math.sqrt(num)).none? {|i| (num%i).zero?}
end
puts is_prime(2)
```

2. All pairs of co-prime number

```
require "set"
def gcd(a,b)
   return b==0?a:gcd(b,a%b)
end

def coPrime(num)
    pairs=Set[]
    (2..num).each do |i|
       (1..i).each do |j|
        gcd(i,j)==1?pairs.add([i,j]):nil
       end
    end
    pairs.each{|i| print i }
end

coPrime(5)
```

3. Fibonacci Series upto first n terms

```
def fibo(n)
    i, j=0, 1
    while i<= n
        puts i
        i,j = j, i+j
    end
end
fibo(6)
```

Reverse a string without method

```
def rev_str(string)
    string2=[]
    string.each_char{|i| string2.insert(0,i)}
    return string2.join("")
end

puts rev_str("raushan")
```