Ruby

1. What is variables in ruby?
A. Ruby variables are locations which hold data to be used in the programs. Each variable has a different name. These variable names are based on some naming conventions. Unlike other programming languages, there is no need to declare a variable in Ruby. A prefix is needed to indicate it.

2. Types of variables provide diff and when we have to use class variable and instance specify.
A. Here are four types of variables in Ruby:

- Local variables
- Class variables
- Instance variables
- Global variables

Instance Variable	Class Variable
It is a variable whose value is instance-specific and now shared among instances.	It is a variable that defines a specific attribute or property for a class.
These variables cannot be shared between classes. Instead, they only belong to one specific class.	These variables can be shared between class and its subclasses.
It usually reserves memory for data that the class needs.	It usually maintains a single shared value for all instances of class even if no instance object of the class exists.
It is generally created when an instance of the class is created.	It is generally created when the program begins to execute.
It normally retains values as long as the object exists.	It normally retains values until the program terminates.
It can be accessed directly by calling variable names inside the class.	It can be accessed by calling with the class name.
Changes that are made to these variables through one object will not reflect in another object.	Changes that are made to these variables through one object will reflect in another object.

3. What is inheritance and how we can archive multiple inheritance?
A. Inheritance is the procedure in which one class inherits the attributes and methods of another class. The class whose properties and methods are inherited is known as the Parent class. And the class that inherits the properties from the parent class is the Child class.
When a class can inherit features from more than one parent class, the class is supposed to have multiple inheritance. But Ruby does not support multiple inheritance directly and instead uses a facility called mixin.

4. What is mixin?
A. Mixins in Ruby allows modules to access instance methods of another one using **include** method. Mixins provides a controlled way of adding functionality to classes. The code in the mixin starts to interact with code in the class. In Ruby, a code wrapped up in a module is called mixins that a class can include or extend. A class consist many mixins.

5. Methods - diff b/w instance method and class method

A. Key difference:

- Class methods can only be called on classes
- Instance methods can only be called on instances of classes
- Class methods are always defined def **self**.method_name
- Instance methods are always defined def method_name

6. Diff between each and map give any example

A. Array#each executes the given block for each element of the array, then returns the array itself.

Array#map also executes the given block for each element of the array, but returns a new array whose values are the return values of each iteration of the block.

Example: assume you have an array defined thusly:

arr = ["tokyo", "london", "rio"]
Then try executing each:

arr.each { |element| element.capitalize }
=> ["tokyo", "london", "rio"]
Note the return value is simply the same array. The code inside the each block gets executed, but the calculated values are not returned; and as the code has no side effects, this example performs no useful work.

In contrast, calling the array's map method returns a new array whose elements are the return values of each round of executing the map block:

arr.map { |element| element.capitalize }
=> ["Tokyo", "London", "Rio"]

7. Diff between map and select and collect.

A. 1) Map : Map takes the enumerable object and a block, evaluates the block for each element and then return a new array with the calculated values.

[1,2,3,4,5,6,7,8,9,10].map{ |e| e * 5 }

=> [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
2) Select : Select evaluates the block with each element for which the block returns true.

[1,2,3,4,5,6,7,8,9,10].select{ |e| e > 5 }

=> [6 ,7, 8, 9, 10]
3) Collect : Collect is similar to Map which takes the enumerable object and a block, evaluates the block for each element and then return a new array with the calculated values.

[1,2,3,4,5,6,7,8,9,10].collect{ |e| e * 5 }

=> [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

8. You have to create a string with help of array then then string should be convert into hash after that hash should be converted into previous state with single command.

A. Array to string :

```
arr = [1, 2, 3]
st = arr.join()
```

Output:

```
“123”
```

Strinh to hash :

```
require ‘json’
string=
"{\"0\"=>{\"answer\"=>\"1\", \"value\"=>\"No\"}, \"1\"=>{\"answer\"=>\"2\", \"value\"=>\"Yes\"},
\"2\"=>{\"answer\"=>\"3\", \"value\"=>\"No\"}, \"3\"=>{\"answer\"=>\"4\", \"value\"=>\"1\"}}"
JSON.parse string.gsub('=>', ':')
```

Output :

```
{"0"=>{"answer"=>"1", "value"=>"No"},
 "1"=>{"answer"=>"2", "value"=>"Yes"},
 "2"=>{"answer"=>"3", "value"=>"No"},
 "3"=>{"answer"=>"4", "value"=>"1"}}
```

Hash to array :

```
hash  = { :a => ["a", "b", "c"], :b => ["b", "c"] }
hash.values
```

Output :

```
[["a","b","c"],["b","c"]]
```

9. Take any csv file as input and print and show in row wise

A.

```
require 'csv'
  CSV.foreach('test.csv') do |row|
        puts row.inspect
  end
```

Output:

```
["Title1", "Title2", "Title3"]
["one", "two", "three"]
["example1", "example2", "example3"]
```

10. What is exception and how we can handle it, design a code to generate exception and handle it when age is less then 18 or greater then 100 of any user.

A. In Ruby, exception handling is a process which describes a way to handle the error raised in a program. Here, error means an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program’s instructions. So these types of errors were handled by the *rescue block*.

Code:

```
puts "Age: ";
age = gets.chomp.to_i
begin
   if(age<18)
      raise 'Age is under 18.'
   elsif(age>100)
      raise 'Age is more then 100.'
   else
      puts "Good to go."
   end
```

```
end
```
Output :
```
Age:21
Good to go.

Age:10
test.rb:5:in `<main>': Age is under 18. (RuntimeError)

Age:1003
test.rb:7:in `<main>': Age is more then 100. (RuntimeError)
```