1.What are variables in ruby?

ANS:- 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.

There are four types of variables in Ruby:

- **Local variables** :- A local variable name starts with a lowercase letter or underscore (_). It is only accessible or have its scope within the block of its initialization. Once the code block completes, variable has no scope.
- **class variables :-** A class variable name starts with @@ sign. They need to be initialized before use. A class variable belongs to the whole class and can be accessible from anywhere inside the class. If the value will be changed at one instance, it will be changed at every instance.
- **Instance variables:-**An instance variable name starts with a @ sign. It belongs to one instance of the class and can be accessed from any instance of the class within a method. They only have limited access to a particular instance of a class.

- **global variables:-** A global variable name starts with a $ sign. Its scope is globally, means it can be accessed from any where in a program.An uninitialized global variable will have a nil value.

2. What is inheritance and how we can achieve multiple inheritance

ANS:- inheritance:- inheritance is one of the most important features. Inheritance allows the programmer to inherit the characteristics of one class into another class.

We can achieve multiple inheritance:- Ruby supports only *single class inheritance*, it does not support multiple class inheritance but it supports *mixins*. The *mixins* are designed to implement multiple inheritances in Ruby, but it only inherits the interface part.

3.What is mixin?

ANS:-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. 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.

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

ANS:-Methods:-*Method* is a collection of statements that perform some specific task and return the result. Methods allow the user to reuse the code without retyping the code. Methods are time savers and help the user to reuse the code without retyping the code.

Defining & Calling the method: In Ruby, the method defines with the help of def keyword followed by *method_name* and end with end keyword. A method must be defined before calling and the name of the method should be in lowercase. Methods are simply called by its name. You can simply write the name of method whenever you call a method

instance method and class method:-

In Ruby, a *method* provides functionality to an Object. A *class method* provides functionality to a class itself, while an *instance method* provides functionality to one instance of a class.

We cannot call an instance method on the class itself, and we cannot directly call a class method on an instance.

6.Diff between each and map give any example

ANS:- 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:-

```
arr = ["tokyo", "london", "rio"]
a=arr.each { |element| element.capitalize }
b= arr.map { |element| element.capitalize }
```

```
puts  a
puts  b
```

Output– tokyo, london, rio
Tokyo, London, Rio

7.Diff between map and select and collect.

ANS:-

1.map:-Map takes the enumerable object and a block, evaluates the block for each element and then returns a new array with the calculated values.

2. Select:-Select evaluates the block with each element for which the block returns true.and also, select would return a element for which the statement is true.

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.

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

ANS:-

```
#create csv file
file = File.new("file_.csv", "a")
puts "enter 0 to exit"
while true
            puts "Enter your first name"
            name=gets.chomp
            if name=="0"
           puts "file created successfully!"
          break
End
 puts "Enter your last name"
 last_name=gets.chomp
file.puts "\n#{name},#{last_name}"
 end
file.close
line_num=0

#open csv file
File.open('file_.csv').each do |line|
print "Line #{line_num} is :-  #{line}"
line_num+=1
End
```

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

ANS:- What is an exception and how we can handle it :-The execution and the exception always go together. If you are opening a file, which does not exist, then if you did not handle this situation properly, then your program is considered to be of bad quality.

The program stops if an exception occurs. So exceptions are used to handle various type of errors, which may occur during a program execution and take appropriate action instead of halting program completely.

Ruby provide a nice mechanism to handle exceptions. We enclose the code that could raise an exception in a *begin/end* block and use *rescue* clauses to tell Ruby the types of exceptions we want to handle.

```
begin
  puts "Enter age "
  age=gets.chomp.to_i
  if age<18 || age>100
     raise "Invalid_Age_Exception"
  else
      puts "valid age"
 end
rescue Exception =>  e
puts e
end
```