Ruby Assignment 1
by
Aliasger Kanchwala

Q1.what is 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.

Q2.types of variables provide diff and when we have to use class variable when instance specify.

Ans.

There are four types of variables in Ruby:

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

1.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.When uninitialized local variables are called, they are interpreted as call to a method that has no arguments.

2.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.A class variable is shared by all the descendents of the class. An uninitialized class variable will result in an error.

3.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.They don't need to be initialize. An uninitialized instance variable will have a nil value.

4.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. It is advised not to use them as they make programs cryptic and complex.There are a number of predefined global variables in Ruby.

Difference between class variables and class instance variables :-

The main difference is the behavior concerning inheritance: class variables are shared between a class and all its subclasses, while class instance variables only belong to one specific class.

Q3.what is inheritance and how we can archive multiple inheritance ?

Ans.
Inheritance: In this one object acquires the properties of another object. It creates new classes from pre defined classes. New class inherit behaviors of its parent class which is referred as superclass. In this way, pre defined classes can be made more reusable and useful.

We can achieve multiple inheritance in ruby by mixins .Ruby does not support multiple inheritance directly but Ruby Modules have another wonderful use. At a stroke, they pretty much eliminate the need for multiple inheritance, providing a facility called a *mixin*.

Mixins give you a wonderfully controlled way of adding functionality to classes. However, their true power comes out when the code in the mixin starts to interact with code in the class that uses it.

Ex:-

```
module A
   def a1
   end
   def a2
   end
end
module B
   def b1
   end
   def b2
   end
end

class Sample
include A
include B
```

```
    def s1
    end
end

samp = Sample.new
samp.a1
samp.a2
samp.b1
samp.b2
samp.s1
```

Q4.what is mixin?

Ans.

Ruby doesn't support multiple inheritance. Modules eliminate the need of multiple inheritance using mixin in Ruby.

A module doesn't have instances because it is not a class. However, a module can be included within a class.

When you include a module within a class, the class will have access to the methods of the module.

Ex:-

```
module Name
  def bella
  end
  def ana
  end
end
```

```
module Job
  def editor
  end
  def writer
  end
end

class Combo
include Name
include Job
  def f
  end
end

final=Combo.new
final.bella
final.ana
final.editor
final.writer
final.f
```

Q5.methods - diff b/w instance method and class method

Ans.

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.

Ex:-

```
class SayHello
  def self.from_the_class
    "Hello, from a class method"
  end

  def from_an_instance
    "Hello, from an instance method"
  end
end
```

When we call

```
>> SayHello.from_the_class
```

It gives output:-

```
Hello, from a class method
```

And when we call

```
>> SayHello.from_an_instance
```

It gives error

Similary

For calling from_an_instance method we have to create object first and then call.

Q6.diff between each and map give any example

Ans.

Ruby each Iterator

The each iterator returns all the elements of an array or a hash.

Ex:-

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

ary.each do |i|

  puts i

End
```

Ruby map function

map() is a Array class method which returns a new array containing the values returned by the block.

Ex:-

```
a = [18, 22, 33, 3, 5, 6]

b = [1, 4, 1, 1, 88, 9]

puts "map method : #{a.map {|num| num > 10 }}"

puts "map method : #{b.map {|x| x.odd? }}"
```

Output:-

```
map method : [true, true, true, false, false, false]
```

map method : [true, false, true, true, false, true]

Q7.diff between map and select and correct?

Ans.

1.select :- **select()** is a Array class method which returns a new array containing all elements of array for which the given block returns a true value.

Ex:-

```
a = [18, 22, 33, 3, 5, 6]

b = [1, 4, 1, 1, 88, 9]

puts "select method : #{a.select {|num| num > 10
}}"

puts "select method : #{b.select {|x| x.odd? }}"
```

Output:-

```
select method : [18, 22, 33]

select method : [1, 1, 1, 9]
```

2.map :- map() is a Array class method which returns a new array containing the values returned by the block.

Ex:-

```
a = [18, 22, 33, 3, 5, 6]

b = [1, 4, 1, 1, 88, 9]

puts "map method : #{a.map {|num| num > 10 }}"

puts "map method : #{b.map {|x| x.odd? }}"
```

Output:-

```
map method : [true, true, true, false, false, false]
map method : [true, false, true, true, false, true]
```

3.collect :- **collect()** is an Array class method which invokes the argument block once for each element of the array. A new array is returned which has the value returned by the block.

Ex:-

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

puts "collect a : #{a.collect {|x| x + 1 }}"
```

Q8.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?

Ans.

For example :-

```
# we have  an array
a=[1,2,3,4]
# to convert it into string we use
a.join("")
# now string to hash
s= "{\"a\"=>\"ali\"}" # we will get this using h.to_s
JSON.parse d.gsub('=>', ':')


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

Q9.take any csv file as input and print and show in row wise?

Ans.

```
require 'csv'
CSV.foreach('c1.csv') do |row|
   puts row.inspect
end

# it will show csv file data in row format
```

Q10.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 ?

Ans.

An exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at runtime, that disrupts the normal flow of the program's instructions. In Ruby, descendants of an Exception class are used to interface between raise methods and rescue statements in the begin or end blocks.

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.

Everything from begin to rescue is protected. If an exception occurs during the execution of this block of code, control is passed to the block between rescue and end.

For each rescue clause in the begin block, Ruby compares the raised Exception against each of the parameters in turn. The match will succeed if the exception named in the rescue clause is the same as the type of the currently thrown exception, or is a superclass of that exception

Program:-

```
begin
   age=gets.chomp.to_i
   if age<18 or age>100
```

```
        raise 'not valid for vote'
    end

rescue Exception => e
    puts e.message
else
    puts "go for vote"
ensure
    puts "try another one"
end
```

.