Review on Ruby

Ruby Properties

- Everything is ruby is an object even literals and classes
- (A **literal**is a special syntax in the **Ruby** language that creates an object of a specific type, like Number literals, array Literals, String literals)

- Variables need not to be declared
  Example: x=5

- Ruby is a dynamic language
  X=6
  X="hi"
- No special main() function

- Parallel assignment is possible in Ruby:
  x=0, y=3
  x,y =x+2,y+2

? What is the value of x and y if we print them?

Ruby Properties

- Inheritance exists in Ruby
- Methods are defined with def and end in ruby, the method can return something
- The new method creates an object. S=Classneme.new
- All the control flow blocks need end
- ***Initialize*** method in Ruby works as constructor

Instance variables

Instance variables (with @) can be directly accessed only by instance methods (private). Outside the class they require accessors

getter	setter
def x	def x= (value)
@x	@x=value
end	end

Note: in setter we use x= format because when later for example you create an object obj of a class A you write obj.x=10 to set the value for x

attr_accessor

- Ruby provides a Shortcut:

Calss A

attr_accessor :x, :y #generates x= and x also y= and y

So if we create an object of class A

Obj=A.new

A.x => works and we do not need getter/setter methods defines

- attr_accessor provides both attr_reader and attr_writer
- attr_reader=>shortcut for getter
  attr_writer =>shortcut for setter

No Method overloading in Ruby

- Thus there is just one initialize method in Ruby and unlike Java it does not have different constructors.
- Ruby does not issue an exception or warning if a class defines more than one initialize method
- But last initialize method defined is the valid one

An Example of Class and Subclass in Ruby

(original ruby file is available on Course Webpage)

class **Time**

```
def initialize (hour,minutes, seconds)
@hour=hour
@minutes=minutes
@seconds=seconds
end

def sethour (hour)
@hour = hour
end

def setminutes (minutes)
@minutes = minutes
end

def gethour
@hour
end

def getminutes
@minutes
end

def setseconds (seconds)
@seconds = seconds
end

def getseconds
@seconds
end
end
```

SubClass

class **LunchTime < Time**

by using attr_accessor you do not need setter and getter for lunchhour, even for the hour in the parrent class that was private before

attr_accessor :lunchhour, :hour

```
  def initialize (hour,minutes, seconds,lunchhour )
    super(hour,minutes, seconds)
    @lunchhour=lunchhour
  end

   def islunchtime
     if(hour==lunchhour)
        return true
      else
        return false
      end    #if
     end    #method
end    #class
```

Z and K are two object passed to precede method

```
def precede(z,k)
 if(z.gethour<k.gethour)
return 1
elsif(z.gethour==k.gethour && z.getminutes<k.getminutes)
return 1
elsif(z.getminutes==k.getminutes && z.getseconds<k.getseconds)
return 1
else
return 0
 end
end
```

Test

```
t1=Time.new(2,5,6)
#puts p.hour # this gives error since hour is private
puts t1.gethour

 t2=Time.new(2,5,10)

#If t1 is before t2
puts precede(t1,t2)

#creating an abject of subclass
 t3=LunchTime.new(2,5,6,2)
puts t3. islunchtime()
```

Sort and Sort! Built-in Functions In Ruby

- The important difference between **.sort** and **.sort!** is that the first one returns a **copy** of an original array, while the second one modifies the original data.

Example:

A= [1, 25, 5, 15, 10, 20]

print A.sort
=> [1, 5, 10, 15, 20, 25]

However

Print A=> [1, 25, 5, 15, 10, 20] (the original array has not changed)

But sort! Changes the original array (In-Place Sorting)

Mixin include, require

- Require and include are different.
- How include mixin (module) in ruby? Using include

  Example:

  include Debugger #Debugger is the name of a module

- If we want to import another ruby file in the current file we use require

  Example:

  require “time.rb”