Containers Blocks and Iterators in Ruby

Vikas Kedia

November 29, 2004

Ruby

- Developed in 1995 in Japan
- Dynamic Language like PERL, Python
- Object Oriented Language
- Pure OO like smalltalk
  - No funtions Only Method calls
  - Everything is an object in Ruby

What are Containers?

- Containers are objects that hold reference to other objects
- Commonly used containers

  String Ordered container of characters

  Array Like strings but can hold objects of any type

Why use Containers?

- Occur naturally in many applications
  - In mp3 player playlist is a container of songs
  - In GUI window is container of widgets (buttons,textbox etc)
- Do not need to keep track of related objects individually

Advantages of Containers

- Easy to operate on all the related objects simultaneously
- Examples
  - Shuffle the songs in a playlist
  - Display all the widgets in a GUI
- Easy to add, destroy, maintain the objects in container

Make your own Container

- Defined as a class

```
class Containername
     def methodname
       methodbody
     end
     def methodname2
       methodbody2
    end
end
```

Methods inside Container

- Common method in container

initialise	to initialise a container
append	to add an object reference to container
delete	to remove an object reference
iterator	to iterate through the objects in conatiner

Iterator

- We have all used for loop to list the elements in an array
- For loop the simplest iterator

Iterator

- We have all used for loop to list the elements in an array
- For loop the simplest iterator

For loop not the way of the Samurai oops Ruby!

Iterators in Ruby

- Iterator generalises the concept of loop
- Iterator is just a method though a bit different
- When calling an iterator tell it what code you want executed in each iteration
- Same iterator can be used for multiple purpose due to this

Example

- An iterator to find a student record from database
- Can search based on name, roll number etc etc ...
- Tell the search criteria while calling the iterator
- To look for a student whose name is "name"
- @students.find|student| name=student.name
- To loof for a student whose rollnumber is "rollno"
- @students.find|student| rollno=student.rollno

A Simple Iterator

```
class Array
    def find
      for i in 0...len
        currval=self[i]
        yield(currval)
      end
   end
end
```

- To print the square of each element
- Array.find|num| print num*num

Block

- Block is a way to group statements together as in conventional languages
- Already seen a couple of blocks
- Code which is passed to the iterator is a block
- Parameters can be passed to a block: |num| is a parameter
- Blocks also return values: the last expression evaluated

Executing a Block

- A block is not executed when it is encountered
- Interpreter remembers the state when it encounters a block
- Block executed inside the method
- Yield statement used to execute the block

Yield Statement

yield	causes the block to be executed
yield(param)	passed the parameter "param" to block and executes it
num=yield(param)	store the value of last statement executed in the block in num

Another Example

- An iterator to accumulate the elements in an array
- Block tells the accumulation criteria

```
class MyArray
   def Accumulate(inival)
      val=inival
      for i in 0 ... len
         val=yield(val,self[i])
      end
      return val
  end
end
```

Using Accumulate

- To add the elements of the array
- MyArray.Accumulate(0)|sum,element|sum+element
- To multiply the elements of the array
- MyArray.Accumulate(1)|prod,element|prod*element
- Counting the elements
- MyArray.Accumulate(0)|count,element|count+1

Conclusion

- Containers used to group objects together
- Iterators used to act on the objects in container
- Block tells iterator how to act