Learning Ruby

Ruby Arrays and Lists

Enumerated Type – Ruby doesn’t have them
But there are cool workarounds
Implementation of Arrays

• Ruby’s Arrays are backed by actual C arrays. When you first declare a basic array ([ ] or Array.new) an instance of the RArray struct is created, and it points to a C array which is allocated with a size of 16.
• When we need more space— we increase the size by half again of the needed size and copy

\[
a = []
\]

100_000.times do
  |i| a << 'shenanigans!' ← adds to end of array
end

Only reallocates 22 times!!!
You can waste time and space, but using
Array.new(n) solves the problem.
Reallocations are really pretty fast as mem copy
Implementation

• Push/pop are $O(1)$ operations (as are appending and that is fast)
• unshift/shift are $O(n)$ operations (as moving the whole array)

Learn Ruby Conventions for capitalization and naming

• ClassNames
• method_names and variable_names
• methods_asking_a_question?
• changing_parameter_methods!
• @instance_variables (in a class)
• $global_variables
• SOME_CONSTANTS or OtherConstants
Strings

var1 = 2
var2 = '5'

puts var1.to_s + var2

Similarly, to_i gives the integer version of an object, and to_f gives the float version

gets

• Just as puts always spits out strings, gets will only retrieve strings. And whence does it get them?
• From you! Well, from your keyboard, anyway. Since your keyboard only makes strings, that works out beautifully. What actually happens is that gets just sits there, reading what you type until you press Enter. Let’s try it out:

puts 'Hello there, and what\'s your name?'
nname = getsputs 'Your name is ' + name + '? What a lovely name!
.puts 'Pleased to meet you, ' + name + '. :)'  

Hello there, and what’s your name?
Vicki
Your name is Vicki
? What a lovely name!
Pleased to meet you, Vicki
. :)

Hmmm… it looks like when I typed in the letters V i c k i, and then pressed Enter, gets got all of the letters in my name and the Enter! Fortunately, there’s a method just for this sort of thing: chomp. It takes off any Enters hanging out at the end of your string. Let’s try that program again, but with chomp to help us this time:

puts 'Hello there, and what\'s your name?'
nname = gets.chomp
.puts 'Your name is ' + name + '? What a lovely name!'  
.puts 'Pleased to meet you, ' + name + '. :)'
parts = "happy", 12.to_s, "birthday"

Treats parts as an array

puts s[0]

Objects are **strongly typed** not **statically typed**.

Strong typing means restrictions on intermixing of types.

Weakly typed – (a) implicit type conversion (b) adhoc polymorphism (overloading)

---

**Examples of typing**

- a = “this”
- a = 100 Ruby is fine, as types can change
- puts “I’m almost” + 100
  - gets mad: can't convert Fixnum into String (TypeError)
  - Strong typing, but not statically typed
- In contrast: In C when you can say if( int) that is an example of weak typing – as it doesn’t get mad it isn’t a boolean

---
Implementation
Shows how different types can be handled (array doesn’t have to have elements the same type)

Ruby Arrays are Cool!
• We’ve already seen a Ruby array – song_lines
  – Addressed from 0
  – elements don’t have to be the same type (done via pointers)
  – Using out of range index returns nil
  – Length is number of elements in array (one more than last index)
  – Last element is always length -1

• Ruby arrays shrink and grow dynamically - no more annoying array index errors.

• Ruby array elements can contain any other objects (think about the implications of this!)
How implemented?

- Details left to implementer
- Array contains addresses to actual objects
- Array addresses typically occupy adjacent memory locations (so can locate in constant time)

Nil

nil is an object - can never get a null pointer error
a = nil
a.nil? ➞ true
a.methods ➞ list of methods
a.abs ➞ NoMethodError
Playing with Ruby Arrays

all = Array.new  # Array is a class object which has methods
my_a = ['one', 'two', 'three']
puts my_a[1]   # => 'two'

other_a = ['four', 'five', 'six']
puts other_a[0]

my_a[3] = other_a  # not a copy but reference to array
                  # length extended just by assigning beyond range
my_a.each { |i| print "my_a ", i, "\n"}
puts my_a[3][2]
other_a[3] = "seven"
my_a.each { |i| print "my_a again ", i, "\n"}

#What is output?

Playing with Ruby Arrays (output)

my_a one
my_a two
my_a three
my_a fourfivesix
six
my_a again one
my_a again two
my_a again three
my_a again fourfivesixseven
Fun with Indices

puts other_a[-1] → last element
another_a = other_a[1..2] → slices (inclusive), Deep copy
t = (10..20).to_a
puts t[-1] → last element
t[-11] = 28 → out of range, but just inserts before first
t[-15] = 28 → out of range, and gives an error
another_a = t[1..2] → slices (inclusive), Deep copy

t.values_at(0, 2, 7) → array of values at those subscripts
variable number of arguments
t.values_at(1..6) → array of values at subscript range 1 through 6

Ruby Array Methods

puts another_a.length

Array.new
Array.new(100)
Array.new(10, 'x') → sets size and initial values

other_a.sort → other_a.sort! if want to change
other_a.delete("five") → find and delete all occurrences
other_a → gone from here
my_a → and from here (no surprise, right?)
Working with "each" Element

Elegant iterators and conditional execution

```ruby
my_a = ['elements', 'are', 'of', 'varying', 'lengths', 'so', 'check', 'them']
my_a.each { |element| p element if element.length > 4 }
  # polymorphic - depends on type of element
my_a = Array.new(10, [1,2,3]) # assignment
my_a[4] = [41,42,43,44,45,46]
my_a[10] = [1,2,3,4,5]
my_a.each { |element| p element if element.length > 4 }

my_a[14] = [3,4,5]
my_a.each { |element| p element if element.length > 4 }
-> undefined method `length' for nil:NilClass (NoMethodError)

elements
varying
lengths
check
[41, 42, 43, 44, 45, 46]
[1, 2, 3, 4, 5]
**************************
After my_a[14] = [1,2,3,4,5]

: undefined method `length' for nil:NilClass
(NoMethodError)
in `each'
\tryit.rb:11
```
Working with "each" Element

my_a.each { |element| puts element if element > 3 }  
my_a.each { |element| if element > 3 then  
        puts element  
    end}  
my_a.each { |element| puts element}  
my_a.each_index { |index| puts my_a[index] if index> 2 }  
my_a.each_with_index { |element,index|  
    print " ", index, ": ", element}

At seats

- Create an array of friends  
- Print out the friends as  
  - friend 0 is sally  
  - friend 1 is ben  
  - friend 2 is sam
friends = ["sally", "ben", "ted", "joey", "mary"]

friends.each_with_index{|element,index| print "friend ", index, " is ", element + "\n"}

OR
for ss in 0...friends.length
  print “friend ", ss, " is ", friends[ss], "\n";
end

At seats

• Create an array of friends
• Print out the friends backwards as
  – sam
  – ben
  – sally
for ss in 0...friends.length
    print " friend ", ss, " is ", friends[-(ss+1)], "\n";
end

OR
puts friends.reverse

---

**flatten**

my_a.flatten  → no subarray – flatten as single dimension
my_a.flatten! → changes m_a

```ruby
a = [1,2,3,4,5]
b = ["6a", "6b", "6c"]
a<<b
p a -* [1, 2, 3, 4, 5, ["6a", "6b", "6c"]]
a.flatten!
p a -* [1, 2, 3, 4, 5, "6a", "6b", "6c"]
```
Ruby Array Stacks

my_a = [1,2,3,4,5,6,7]
last = my_a.pop → removes last and returns it
first = my_a.shift → removes first and returns it

my_a.each{|item| puts item}
my_a.unshift( last ) → prepends to first
my_a.push( first ) → appends to end
my_a.each{|item| puts "then " + item.to_s}

#What is output?

Ruby Array Stacks (output)

2
3
4
5
6
then 7
then 2
then 3
then 4
then 5
then 6
then 1
collect: Returns a new array by invoking block once for every element. The result of block is used as the given element in the new array.

\[
a = [ "a", "b", "c", "d" ]
\]

\[
b = a.collect { |x| x + "!" } \Rightarrow ["a!", "b!", "c!", "d!"]
\]

\[
a \Rightarrow ["a", "b", "c", "d"]
\]

Ruby Exercises

1. Question 1

\[
a = [34, 45, 56, 2, 13, 54]
\]

\[
a.sort!
\]

\[
a.reverse
\]

Puts a[4] gives,

\[
a) 13 \ b) 54 \ c) 45 \ d) 56
\]
Join returns a string formed by joining array elements

2. Question 2
a=[34,45,56,2,13,54]
t = a.join
puts a.class
puts t.class
a) fixNum, Array b) Array String c) String String d) ERROR

rindex Returns the index of the last object in arr such that the object == anObject.

3. Question 3
a=[34,45,56,2,13,54, 45, 10, 100]
i = a.rindex(45)
puts i gives,
a) 2 b) 6 c) 45 d) 56
Remember zero addressing

4. Question 4
a=\{34,45,[56, 57],2,13,54\}
a= a.flatten

puts "Array is a[3] = #{a[3]} #{a}"

a) Array is a[3] = 2 3445565721354
b) Array is a[3] = NIL
c) Array is a[3] = NULL
d) Array is a[3] = 57 3445565721354

5. Question 5
a=\{34,45,56,2,13,54\}
b= a.min + a.max + a.first + a.last
puts b gives,
a) 92 b) 144 c) 146 d) 112
6. Question 6

a=[34,45,56,2,13,54]
b= a[2].value+a[3].value
puts @b gives,
a) Argument Error  b) 58  c) NomethodError  d) 0

Question 7

a=[34,45,56,2,13,54]
a << [137,89]
B=a.length
puts b gives,
a) Error  b) 7  c) 6  d) 8
Question 8: What does this do?

```ruby
all = [1,2,3].collect do |element|
  element*2
end.reverse

puts all
```

Question 9

```ruby
presidents.pop
presidents.pop
presidents.pop
presidents.unshift("Nixon")
presidents.unshift("Johnson")
presidents.unshift("Kennedy")
presidents.each { |i| print i, "\n"}
```
Answer

- Kennedy
- Johnson
- Nixon
- Ford
- Carter
- Reagan

Question 10

```ruby
presidents = []
presidents[2] = "Adams"
presidents[4] = "Madison"
presidents[6] = "Adams"
presidents.each { |i| print i, "\n"}
print "=================
"
presidents[6] = "John Quincy Adams"
presidents.each { |i| print i, "\n"}
print "\n"
```
Question 10

- nil
  - nil
  - Adams
  - nil
  - Madison
  - nil
  - Adams

Question 11

[1..3] is an array slice

p123=presidents[1..3]
p123.each { |i| print i, "\n"}
Question 11

Carter
Reagan
Bush1

Question 12. Another way of getting a slice: starting subscript of 1 and a count of 3, instead of a range 1 through 3.

p123=presidents[1,3]
p123.each { |i| print "part ", i, "\n"}
Question 12

part Carter
part Reagan
part Bush1

Question 13  [2,5] is beginning point and length

- numbers = ["one", "two", "buckle", "my", "shoe", "three", "four", "shut", "the", "door"]
- numbers.each { |i| print i, "\n"}
- print " ===================\n"
- numbers[2,5]=[]
- numbers.each { |i| print i, "\n"}
one
two
buckle
my
shoe
three
four
shut
the
doork

Question 14

numbers = ["one", "two", "3", "4", "5", "seven"]
numbers.each { |i| print i, "\n"}
print "====================\n" numbers[2,3]=["three", "four", "five", "six"]
numbers.each { |i| print i, "\n"}
• one
• two
• 3
• 4
• 5
• seven
• ================
• one
• two
• three
• four
• five
• six
• seven