Nick Simons
Professor Razet
csci208
12/5/16

1. Background

Ruby was created by Yukihiro Matsumoto of Japan, and first released in 1995. Matsumoto created the Ruby language to be a powerful yet relatively natural and easy-to-use object-oriented scripting language that also incorporated features of imperative and functional languages [3]. Additionally, he created Ruby to be an interpreted language rather than compiled [1, pgs 4, 6]. Although all of this sounds like a nigh-impossible task, Matsumoto evidently succeeded, because Ruby is used by many companies involved in web applications, including Amazon, GoodReads, and Hulu [13].

6. Primitive Types

```
def main
  dec = -42
  dec2 = 0d4231
  und = 53_2_9339
  bin = 0b10101
  oct = 0222
  hex = -0x32
  float = 0.4556
  puts dec
  puts dec2
  puts und
  puts bin
  puts oct
  puts hex
  puts float
  print "size of Fixnum: ", 42.size, "\n"
  print "size of Bignum 902832345524666663223557092834750982473
5: ", 90283234552466666322355709283475098247346666322355709283
4750982473.size, "\n"
  print "size of Bignum 123423422345234452312345231: ", 12342342
2345234452312345231.size, "\n"
  str = "This is a string \nThis is a second line \0"
  puts str
  print "size of string: ", str.size, "\n"
end
main()
```

Runs with and prints:

```
[nrs007@brki167-lnx-4 project]$ ruby primtypes.rb
-42
4231
5329339
21
146
-50
0.4556
size of Fixnum: 8
size of Bignum 902832345524666663223557092834750982473
5: 20
size of Bignum 123423422345234452312345231: 12
This is a string
This is a second line
size of string: 41
```

Ruby does not have primitive types in the same way that other languages do, because all data in Ruby are objects; however, Ruby does have the basic types numbers, strings, arrays, hashes, ranges, symbols, and regular expressions [1, pg 304]. Clearly, the Ruby basic types that are most similar to the primitive types of other languages are number and string, so I will briefly discuss those. Integer objects in Ruby can be of type Fixnum, whose size limit is the size of an integer that fits within the given machine's word - 1 bit, and of type Bignum, whose size limit depends on available memory in the given machine [1, pg 304]. Integers in Ruby can be written with leading characters that indicate the base, as shown in the code above. Ruby also supports floating point numbers. String objects in Ruby are, according to Thomas, Fowler, and Hunt, "stored as sequences of 8-bit bytes, and each byte may contain any of the 256 8-bit values, including null and newline" [1, pg 306]. More information on strings can be found later in this report.

8. Default parameters

```
def func1(val1 = 42, val2 = "meaning")
  puts val1
  puts val2
end
func1
```

Runs with and prints:

```
[nrs007@brki164-lnx-20 project]$ ruby params.rb
42
meaning
```

Default parameters are supported in Ruby [1, pg 332]

9. Static vs Dynamic typing

```
genericVar = 343
puts genericVar
genericVar = "now it's different!"
puts genericVar
```

Runs with and prints:

```
[nrs007@brki164-lnx-20 project]$ ruby typing.rb
343
now it's different!
```

In Ruby and other dynamically typed languages, the value holds the type [4]. The default for Ruby is dynamic typing. According to Thomas, Fowler, and Hunt, there is no static typing in Ruby, so you can't make that option happen also [1, pg 350]. Ruby completes run time type checking in the form of "duck typing" -- basically, the idea that if something in Ruby behaves like a certain type, it is recognized as being that type [9, pg 220].

10. Typing Strength

```ruby
def strongTypeTest
  begin
    strInt = "42"
    int = 42
    str = "The answer is "
    answer = str + strInt
    puts answer
    answer = str + int
    puts "if this is printed, weakly typed"
  rescue #=> e
    puts " if this prints, strongly typed"
  ensure
    puts "end"
  end
end
strongTypeTest()
```

Runs with and prints:

```
[nrs007@brki164-lnx-20 project]$ ruby strength.rb
The answer is 42
 if this prints, strongly typed
end
```

The fact that Ruby can't add an integer to a string shows that it is strongly typed [5].

11. Static vs Dynamic Scope

```ruby
$glVar = 42
def changeVar
  glVar = 343
  if 343 == scopeTest()
    puts "Ruby has dynamic scope"
  else
    puts "Ruby has static scope"
  end
end

def scopeTest
  result = $glVar * 1
  return result
end
changeVar()
```

Runs with and prints:

```
[nrs007@brki164-lnx-20 project]$ ruby scope.rb
Ruby has static scope
```

Phil Calcado's discussion of the limitations of Ruby because of its lack of dynamic scope and his reliance on RSpec to make up for those limitations implies that there is not an easy way to make dynamic scope happen in Ruby [6].

12. The string type

Is it a primitive type or added in likely by a library? Is it made with one of the type constructors? What math operations can you do on strings? (s+s, s*3) What other operations are provided for

strings? (none, substring, index of). Is there a concept of a terminating character like the '\0' in C?

```
str1 = "string one"
str2 = String.new("string two\0")
str3 = str1 + str2
str4 = str2 * 2
str5 = str1 << "!"
puts str1
puts str2
puts str3
puts str4
puts str5
puts str1 <=> str2 # should print -1
puts str1 == str1 # should print true
puts str1 == 8448 # should print false
puts str1.capitalize # should print String one!
puts str1.length #should print 11
puts str1.chop #should print string one
```

Runs with and prints:

```
[nrs007@brki167-lnx-4 project]$ ruby string.rb
string one!
string two
string onestring two
string twostring two
string one!
-1
true
false
String one!
11
string one
```

Strings in Ruby are not necessarily primitive, instead, they are objects [7]. Since strings can be created with String.new, it seems that they are made with type constructors [8]. There can be null-terminated strings in Ruby [8]. There are many, many operations that you can do on strings in Ruby, but some of them include math operators like + and *, appending, comparison between strings, comparison between strings and other objects, capitalization, length, and chopping off the last character.

13. Math operations on numbers

```ruby
def opsTest
  begin
    puts 3 + 4
    puts 5 - 20.2
    puts 0.023 * 44
    puts 985853229238859392059034611 1/4482274942281113
    puts 9 % 2
    puts 10**12
    puts 3/'q'
    puts "Ruby allows for nonsensical math operations"
  rescue
    puts "Ruby does not allow for nonsensical math operations"
  end
end
opsTest()
```

Runs with and prints:

```
[nrs007@brki167-lnx-6 project]$ ruby ops.rb
7
-15.2
1.012
2199448364800
1
1000000000000
Ruby does not allow for nonsensical math operations
```

On numbers of types Bignum, Fixnum, Float, and Bigdecimal, addition, subtraction, multiplication, division, modulo, exponentiation, and unary minus are built in [1, pgs 420, 463, 466, 637]. Ruby does not allow for nonsensical operations.

21. Bits within an integer

```ruby
fortytwo = 42
thirtyone = 31

print "42 as binary: ", fortytwo.to_s(2)
puts
print "31 as binary: ", thirtyone.to_s(2)
puts

new = thirtyone & 17
print "11111 & 10001: ", new.to_s(2)
puts
print "new int value: ", new
puts

new2 = fortytwo >> 2
print "101010 >> 2: ", new2.to_s(2)
puts
print "new int value: ", new2
puts
```

Runs with and prints:

```
[nrs007@brki167-lnx-6 project]$ ruby bit.rb
42 as binary: 101010
31 as binary: 11111
11111 & 10001: 10001
new int value: 17
101010 >> 2: 1010
new int value: 10
```

Ruby allows you to access and manipulate the bits of an integer, using <<, >>, &, |, and ^ [1, pg 324].

14. Variables

```ruby
$school = "Bucknell University"  # global variable

class Student
  @@newStudent # class variable
  NUM = 2  # constant
  def initialize(name, gradYear, major)
    @name = name     #--|
    @year = gradYear     #--|== instance variables
    @major = major   #--|
    @@newStudent = name
  end
  def prntStudent
    print "#@name is majoring in #@major and graduates in #@year\n"
  end
  def double(x)
    product = NUM * x # local variable
    return product
  end
end



stdnt = Student.new("Nick", 2018, "CSCI")
stdnt.prntStudent()
puts stdnt.double(3) # should print 6
```

Runs with and prints:

```
[nrs007@brki167-lnx-6 project]$ ruby vars.rb
Nick is majoring in CSCI and graduates in 2018
6
```

Ruby has global, class, constant, instance, and local variables [10].

15. Dangling Else

```ruby
def elseTest(val)
  if 0 < val
  if val % 2 == 0
  puts "even"
  puts "positive"
  else
  puts "not positive"
  end  # if one of these ends were not here,
  end  # there would be a syntax error
end
elseTest(4)
```

Runs with and prints:

```
[nrs007@brki167-lnx-6 project]$ ruby else.rb
even
positive
```

The dangling else is not a problem in Ruby, because Ruby if-else statements must have a terminating “end” keyword.

16. Precedence and Associativity

The order of precedence for ALL Ruby operators is as follows: element reference/element set > Exponents > Not/complement/unary plus/unary minus > multiplication/division/mod > plus and minus > right/left shift > bitwise AND > bitwise exclusive or/regular or > all comparison operators (<=, <, >, >=) > all equality and pattern matching operators (<=>, ==, ===, !=, =~, !~) > logical AND > logical OR > exclusive/inclusive range > if-then-else > all assignments > check if symbol defined > logical negation (not) > logical composition (or, and) > expression modifiers (if, unless, while, until) > block expression (begin/end) [1, 324].
Right-Associative operators: !, ~, (unary)+, **, -, ?, :, all assignments, not
Left-Associative operators: *, /, %, +, -, <<, >>, &, |, ^, <, <=, >=, >, &&, ||, rescue, and, or
Non-Associative operators: ==, ===, !=, =~, !~, <=>, .., …, defined?, if, unless, while, until
Associativity information comes from [9, pgs 102, 103].

23. Garbage Collecting

```
puts GC.enable
GC.start
puts GC.disable
puts GC.disable
```

Runs with and prints:

```
[nrs007@brki164-lnx-20 project]$ ruby gc.rb
false
false
true
```

Prints false once because the garbage collector is automatically enabled, then false again because the gc was already enabled, then “true” because the garbage collector was already disabled

Ruby has a garbage collector. It closes file variables when they are out of scope [1, pg 120]. It is “mark-and-sweep” garbage collection [1, pg 271]. With the GC module, programmers can manually call some garbage collection methods, including disable, enable, start, and garbage_collect [1, 470].

17. Short Circuit Evaluation

```ruby
def shortTest(x)
  if 0 == x || func2(x)
    puts "If this is the only line printed, short circuit eval"
  end
end

def func2(x)
  puts "If this is printed, not short circuit eval"
  return true
end

shortTest(0)
```

Runs with and prints:

```
[nrs007@brki167-lnx-6 project]$ ruby short.rb
If this is the only line printed, short circuit eval
```

So, Ruby used short-circuit evaluation. None of the built-in operators allow you to avoid short circuit [1, pg 88].

18. BNF Grammar

stmt-list ::= stmt | stmt stmt-list
stmt ::= if-stmt | ret-stmt | assign-stmt | comp-stmt
if-stmt ::= IF exp THEN stmt END
| IF exp THEN stmt ELSE stmt END
| IF exp THEN stmt ELSIF stmt ELSE stmt END
ret-stmt ::= RETURN exp | RETURN
assign-stmt ::= ID '=' exp
exp ::= exp2 relop exp2 | exp2
relop ::= '<' | '>' | '<=' | '>=' | '!=' | '=='
exp2 ::= exp2 addop term | term
addop ::= '+' | '-'
term ::= term mulop factor | factor
mulop ::= '*' | '/' | '%'
factor ::= exp | NUMBER | ID

19. Control structures

```ruby
def ctrlTest(x, y, z)
  if x > 0
    puts "positive"
  elsif x == 0
    puts "zero" #should print first time
  else
    puts "negative" #should print second time
  end
  unless y == 3
    puts "y is not 3" #should print first time but not second
  end
  zVal = case
           when z % 2 == 0 then "even" #should print first time
           when z % 2 == 1 then "odd" #should print second time
           else "this shouldn't have happened!"
           end
  puts zVal
  i = 0
  until i >= 3
    print "i is ", i, "\n"
    i = i + 1
  end
  bananaList = [1, 33, 700]
  for item in bananaList do
    print item, " "
  end
  print "\n"
  print bananaList.map{|x| x+1}, "\n"
  2.upto(5){|x| puts x*2}
  puts
end
ctrlTest(0, 4, 52)
ctrlTest(-1, 3, 11)
```

Runs with and prints:

```
[nrs007@brki167-lnx-6 project]$ ruby ctrl.rb
zero
y is not 3
even
i is 0
i is 1
i is 2
1 33 700
[2, 34, 701]
4
6
8
10

negative
odd
i is 0
i is 1
i is 2
1 33 700
[2, 34, 701]
4
6
8
10
```

Ruby includes the following control structures: conditionals (e.g. if-then statements), loops (for/while/until), iterators and enumerable objects (times,map,etc.), and blocks [9]. Control is transferred thanks to boolean values and statements like return, break, next, redo, retry, throw, and catch [9, 146].

20. Comment styles

```
=begin
this is
how you
do multiline
comments in Ruby
=end

def commentTest
  puts "that's all!" # This is an in-line comment in Ruby
end

# This is a single-line comment in Ruby

commentTest()
```

Runs with and prints:

```
[nrs007@brki167-lnx-6 project]$ ruby comment.rb
that's all!
```

Ruby has support for line, multiline, and in-line comments [10].

1. Exceptions

```
class CustomError < ZeroDivisionError
end

def tryCatchTest
  begin

    raise CustomError
    y = 3/0
    puts "In begin block"
    return
  rescue CustomError
    puts "Custom Error was raised, rescued"
  rescue ZeroDivisionError
    puts "zero division error"
  ensure
    puts "In the finally block"
  end
end
tryCatchTest()
```

Runs with and prints:

```
[nrs007@brki164-lnx-24 project]$ ruby tryCatch.rb
Custom Error was raised, rescued
In the finally block
```

Ruby does have exceptions. The Ruby equivalent to Java's finally is "ensure," and there can be multiple "rescue" blocks, which are the equivalent to Java's catch blocks. Additionally, you can create one of your own exceptions to be thrown [1, pgs 101-109].

3. How inheritance works

```ruby
class Cat
  LEGS = 4
  @@isCat
  def initialize(name, color, age)
    @name = name
    @color = color
    @age = age
    @@isCat = "true"
  end
  def getLegs
    return LEGS
  end
  def getIsCat
    return @@isCat
  end
end

class Tiger < Cat
  def isBig
    puts "yes"
  end
  def toStr
    puts "name: #@name, color: #@color, age: #@age"
  end
end

  hobbes = Tiger.new("Hobbes", "Orange", 12)
  hobbes.toStr()
  puts hobbes.getIsCat()
  puts hobbes.getLegs()
  puts
  puts Tiger.ancestors
  puts
  puts String.ancestors
  puts
  puts Integer.ancestors()
```

Runs with and prints:

```
[nrs007@brki164-lnx-24 project]$ ruby inheritance.rb
name: Hobbes, color: Orange, age: 12
true
4

Tiger
Cat
Object
Kernel
BasicObject

String
Comparable
Object
Kernel
BasicObject

Integer
Numeric
Comparable
Object
Kernel
BasicObject
```

There are access controls in the form of public, protected, and private methods. Ruby does not support multiple inheritance [1, pg 28]. Constructors work as shown in the code example, and destructors work with a call to define_finalizer(), as described in Thomas, Fowler, and Hunt [1, pg 557]. A Ruby subclass needs accessor methods to access the constants and class variables of its parent class, but not the instance variables. All objects in Ruby do inherit from a base class, called BasicObject.

2. Static vs Dynamic method binding

```ruby
class Animal
  def sound
    puts "generic sound (static)"
  end

  def callSound
    self.sound()
  end
end
class Dog < Animal
  def sound
    puts "Bark! (dynamic)"
  end
end
spot = Dog.new()
spot.callSound()
```

Runs with and prints:

```
[nrs007@brki164-lnx-24 project]$ ruby bindingTest.rb
Bark! (dynamic)
```

Ruby's default is Dynamic type binding. You can't make the other option occur.

4. Deep vs Shallow copying of objects

```ruby
class Sheep
  def initialize(name)
    @name = name
  end
  def getName
    return @name
  end
end
molly = Sheep.new("Molly")
dolly = molly.dup
if molly.getName.object_id == dolly.getName.object_id
  puts "shallow"
else
  puts "deep"
end
```

Runs with and prints:

```
[nrs007@brki164-lnx-24 project]$ ruby copying.rb
shallow
```

Ruby automatically performs shallow copying. In order to achieve the functionality of a

deep copy, you could use initialize_copy, which serves as something similar to a constructor for copying objects, as described on thingsaaronmade.com [2].

5.Namespaces

```ruby
module Human
  def Human.walk
    puts "two legs"
  end
end
module Wombat
  def Wombat.walk
    puts "four legs"
  end
end
Human.walk()
Wombat.walk()
```

Runs with and prints:

```
[nrs007@brki164-lnx-24 project]$ ruby namespace.rb
two legs
four legs
```

There are namespaces in Ruby in the form of modules, which can be implemented using ‘require’ if the modules are in a separate file [1, pgs 110-111]

22. Super

```ruby
class Guitar
  attr_reader :strings, :type, :whammy
  def initialize(type = "acoustic")
    @strings = 6
    @type = type
    @whammy = false
  end
end

class Gibson < Guitar
  def initialize(type)
    super
    @whammy = true
  end
end

elect = Gibson.new("electric")
acoust = Guitar.new
puts elect.strings # 6
puts elect.type    # electric
puts elect.whammy  # true

puts acoust.strings# 6
puts acoust.type   # acoustic
puts acoust.whammy # false
```

Runs with and prints:

```
[nrs007@brki167-lnx-6 project]$ ruby super.rb
6
electric
true
6
acoustic
false
```

The above code was written using rubylearning.com as a reference [12]. Super works in Ruby as expected. It allows a child class to have methods with the same name but different implementations as methods in its parent class. In this way, the child class's method overrides that of the parent class.

Works Cited

[1] Thomas, Dave,et al. Programming Ruby: The Pragmatic Programmers' Guide, Second Edition. 2004
[2] http://thingsaaronmade.com/blog/ruby-shallow-copy-surprise.html, Things Aaron Made
[3] https://www.ruby-lang.org/en/about/ Ruby: A Programmer's Best Friend
[4] https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html, Oracle
[5] http://www.rubyfleebie.com/ruby-is-dynamically-and-strongly-typed/ Ruby Fleebie
[6] http://philcalcado.com/2012/07/01/annoying_rspec_dynamic_scoping_in_example_groups.html Phil Calcado
[7] https://ruby-doc.org/core-1.8.7/String.html Ruby-Doc
[8] http://www.techotopia.com/index.php/Ruby_Strings_-_Creation_and_Basics Techotopia
[9] Flanagan, David, and Yukihiro Matsumoto. The Ruby Programming Language. First Edition, 2008
[10] https://www.tutorialspoint.com/ruby/ruby_variables.htm Tutorials Point: Simply Learning
[11] http://rubylearning.com/blog/2015/03/20/comment-types-in-ruby/ Ruby Learning
[12] http://rubylearning.com/satishtalim/ruby_overriding_methods.html Ruby Learning
[13] http://tech.co/ruby-on-rails-2014-08 Tech Co