RUBY INTRODUCTION

TOPICS

- Download and Install
- Test Cases
- Basic Programming
- References

TESTING

- Test-Driven Development
  - Write test cases and then enough application code to make the test pass
- Java has JUnit and TestNG
- Ruby has Test::Unit and RSpec
- Mike Clark - Using Test Cases to learn a language
  - http://clarkware.com/cgi/blosxom/2005/03/18

TEST:UNIT

```
require 'test/unit'

class StringTest < Test::Unit::TestCase

  def test_length
    s = "Hello, World!"
    assert_equal(13, s.length)
  end

end
```

EVERYTHING IN RUBY IS AN OBJECT

```
class Fib
  # Recursive Version
  def fib(n)
    return 1 if n == 0 || n == 1
    return fib(n-1) + fib(n-2)
  end

  # or
  def fib_non_recursive(n)
     return 1 if n == 0 || n == 1
     f1, f2 = 1,1
     while f1 <= n
        f1, f2 = f2, f1+f2
     end
  end
end
```

RSPEC

"RSpec is a framework which provides programmers with a Domain Specific Language to describe the behaviour of Ruby code with readable, executable examples that guide you in the design process and serve well as both documentation and tests."

RSpec is for 'Behavior-Driven Development' (BDD)
Examples of expected behavior are specified

http://rspec.rubyforge.org/

TESTING FIBONACCI

```
require 'Fib'

describe Fib do
  before(:each) do
    @fib = Fib.new
  end

  it "should equal 1 for Fib(0)" do
    @fib.fib(0).should == 1
  end

  it "should equal 1 for Fib(1)" do
    @fib.fib(1).should == 1
  end

  it "should equal 2 for Fib(2)" do
    @fib.fib(2).should == 2
  end

end
```

EXECUTE RSPEC

```
mpro1:~/projects/code/Dan$ spec fib_spec.rb --format specdoc

Fib
- should equal 1 for Fib(0)
- should equal 1 for Fib(1)
- should equal 2 for Fib(2)

Finished in 0.006859 seconds

3 examples, 0 failures
```

BENCHMARKING

```
require 'benchmark'
require 'Fib'

fib=Fib.new

Benchmark.bmbm do |b|
  b.report("recurse") { fib.fib(30) }
  b.report("loop") { fib.fib_non_recursive(30) }
end
```

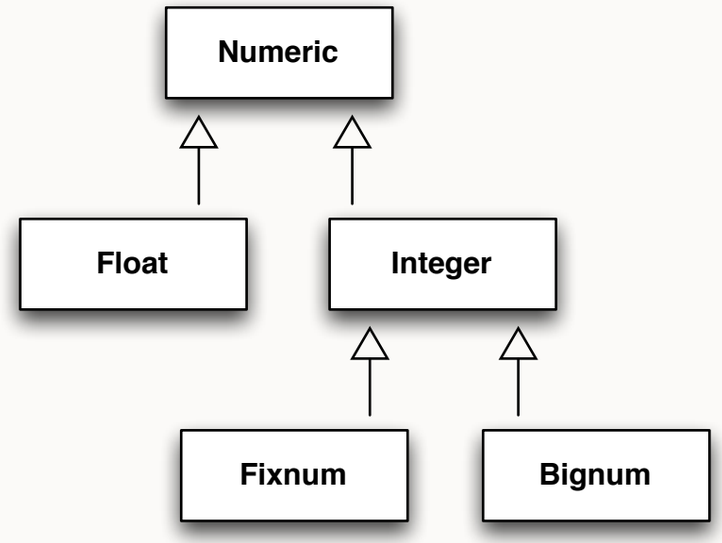
STRINGS

http://github.com/deweime/jhu-484-code/blob/4bffb6354fa8a8a9ad2adac0dd696059b4a66441/learn/string_test.rb

```ruby
require 'test/unit'

class StringTest < Test::Unit::TestCase

  def test_length
    s = "Hello, World!"
    assert_equal(13, s.length)
  end

  def test_plus
    s = "Hello"
    h = s + " World"
    assert_equal(h,"Hello World")
    assert_equal(s,"Hello")
  end

  def test_append
    s="Hello"
    s << " World"
    assert_equal(s,"Hello World")
  end

  def test_interpolation
      val="AddMeIn"
      str="This is a string #{val}"
      assert_equal(str,"This is a string AddMeIn")
  end

  def test_content_equality
      str1="one"
      str2="one"
      assert(str1==str2,"Content equality test")
  end

  def test_object_equality
      str1="one"
      str2="one"
      assert(! (str1.equal?(str2)),"object equality test")
  end
```

NUMBERS

- All Numbers are objects
- Operators are actually methods
- large numbers are automatically converted to Bignums
- Hex: ‘0x’; Octal has leading ‘0’
- Strings have a to_i method that



NUMBERS (CONT)

```
puts 5.class             #=> Fixnum
puts 3.0.class           #=> Float
puts 2_000_000_000.class  #=> Bignum
puts "============"

puts Fixnum.ancestors
puts "============"
puts Float.ancestors
puts "============"
puts Bignum.ancestors
puts "============"


puts 5.methods
puts "============"
```

COLLECTIONS AND CONTAINERS

- Arrays
  - ordered collections; addressable via an index
- Hashes (Dictionaries or Associative Arrays)
  - addressable via a key
- Enumerations (Mixin)

CLASSES AND OBJECTS

```
require 'test/unit'

class HashTest < Test::Unit::TestCase
  def test_hash_creation
    assert_equal( @h["Maryland"], "MD")
  end
  .....
end
```

METHOD ACCESS

- public - default access level
- private - can only be called internally
- protected - can call on objects of the same class instance

VARIABLE CONVENTIONS

- All caps is a constant
- @var is an instance variable
- @@var is a class variable
- var in a method is a local variable

VARIABLES

- come into existence when declared. untyped.

```
class Attributes
  attr_accessor :name    # Could also have just attr_reader and attr_writer

  def change_name(n)
    @name=n
  end
end

attr=Attributes.new
attr.name="Fred"
puts attr.name

attr.change_name("Dan")
puts attr.name
```

NO MULTIPLE INHERITANCE BUT....

- We can 'mixin' Modules e.g Comparable and Enumerable

```
class MixinExample
  include Enumerable

  ITEMS=["One", "Two", "Three", "Ocho"]

  def each
    ITEMS.each { |item| yield item}
  end
end

m=MixinExample.new
a=m.grep(/^[Oo]/)
puts a
```

REFERENCES

- Ruby logo is copyright © 2006, Yukihiro Matsumoto