Announcement

- Please make sure to start HW3
  - If you are behind, contact me about lateness
Outline

- Ruby today
  - Very brief overview of language
  - Some parts will be faster than others
  - Will work though examples in groups
  - Please ask if you have a question

Background

- You saw how useful perl was (compared to c/c++)
- You saw how many different ways there was to do everything
- There is a reason real geeks like perl
There is also a reason this compiles

```
@P=split//,".URRU\c8R";@d=split//,"\nrekcah
xinU / lreP rehtona tsuJ";sub p{
@p("r$p","u$p")=(P,P);pipe"r$p","u$p"; ++$p; ($q*=2)+=$f=!fork;map($P=$P[$f^ord
($p{$_})&6];$p{$_}=/
^$P/ix?$P:close$_}keys%p;p;p;p;map{p{$_}=$_}/^[P.]/&

until?$;map{ /^r/&&<$_}p;$_=$d[$q];sleep
rand(2)if/\S/;print
```

- Released in Dec 1994
- Everything is an object
- Similar to perl in the fact its interpreted
Interpreter

- ri
  - Ruby information (perldoc equivalent)
  - ri String
  - Ri String#chop

- start ruby interpreter
  - irb
    - Interactive ruby interpreter – good for testing small programs
  - Will be

basics

- Language basics
  - Number handling
  - Variables
  - conditions
basics

- Can get it to evaluate expressions
  - 2 + 4
    - ints
  - 3 / 2
    - ints
  - 3.0 / 2
    - floats
  - ** and %

Strings

- Anything between quotes
- Lots of built in functions
  - empty
  - length
  - reverse
  - chop
  - uppercase
  - downcase
  - swapcase
  - * n
variables

- Begin with lowercase
- Declared and assigned in same step
- Constant variables begin with upper case
  - Will just warn on changes (will still work)

- Global variables begin with $
Basic functions

- **puts**
  - Put a string to terminal
  - Return value is nil
  - `val = puts "Hello world!"

- **to_s and to_i**
  - Convert to and from String and Integer/Float objects
  - Important since to create long strings won’t automatically happen like Java does it

---

loops

- **yourINt.times do**
  - Some code

- **end**

- **Simply loops through yourInt times**
What is the sum of all the integers from 10 to 100?

```
One idea

count = 10
  sum = 0
  91.times do
    sum += count
    count += 1
  end

  puts "Sum is now " + sum.to_s
```
Question?

- So how would you count backwards??

User input

- `variable = gets`
  - Will grab user input

- Will append the newline

- `variable.chomp`
  - Will give you what you want 😊

- `variable = variable.chomp`
quotes

- Single quotes are string literals
- Can use \q and use your own delims
  \q! someth’ing!
  Or
  `someth\’ing`

- Refer to variable inside quotes using #{..}
- “this is the #{sum}”

other

- if condition
  - Something
- end

- Combine with elsif

- while condition
  - stuff
- end
Basic...

- Ok enough with basics
- Let's start with more sophisticated Data structures
- Will use it to build a program

Arrays

- Collection of objects
- numbers = [ "zero", "one", "two", "three", "four" ]
- numbers[3].reverse

- collect = []
  - collect = Array.new
- collect[20] = “surf”
Array functions

- a = %w{ ant bee cat dog elk }

- arrays.sort
- arrays.reverse
- arrays.length

- Can also combine arrays using + and *

Question

- addresses = [ [ 285, "Ontario Dr"], [ 17, "Quebec St"], [ 39, "Main St" ] ]

- What will addresses.sort do ??
Control sort

- addresses.sort do |a,b|
  - a[1] <=> b[1]
- end

Can iterate through array quite easily

- addresses.each do |item|
  - puts “this is the “ + item
- end
How would you iterate through an array with for loop?

addresses.length.times do |i|
  puts "I have: " + addresses[i]
end
hashes

- Hmm look familiar

```ruby
instSection = {
  'cello'     => 'string',
  'clarinet'  => 'woodwind',
  'drum'      => 'percussion',
  'oboe'      => 'woodwind',
  'trumpet'   => 'brass',
  'violin'    => 'string'
}

instSection['cello']
```

- Iterate through hash

```ruby
somehash.each do |key, value|
  puts key + " => " + value
end

somehash.each_key do |key|
  puts key
end
```
Reg Expression

```ruby
if line =~ /Perl|Python/
  puts "Scripting language mentioned: #{line}"
end

line.sub(/Perl/, 'Ruby')
line.gsub(/Python/, 'Ruby')
```

functions

- Beyond basic usage, now lets talk about how to program functions

```ruby
def say_hi
  puts "Hey, How are you?"
end
```
Pass args

```ruby
def say_hi(name)
    puts "Hello " + name + ", How are you?"
end
```

Classes

- As mentioned ruby is pure object oriented language
- So need to define classes if you want your own objects
- Start with upper case letter
- Initialize function always called first
- Instance variables in class start with @
class Address
    def initialize(street)
        @street = street
    end
end

test = Address.new("120 st")
test.inspect

---

Another class

class Song
    def initialize(name, artist, duration)
        @name = name
        @artist = artist
        @duration = duration
    end
end
Always open

- Can always add a new function to a class by saying:

```ruby
class Song
  def to_s
    "Song: #{@name}--#{@artist} (@#{@duration})"
  end
end
```

Inheritance

```ruby
class KaraokeSong < Song
  def initialize(name, artist, duration, lyrics)
    super(name, artist, duration)
    @lyrics = lyrics
  end
end
```
question

- So what will happen when you call KaraokeSong to_s ??

functions

- Would be useful to be able to ask the song class for individual items

```ruby
class Song
  def name
    @name
  end
  def artist
    @artist
  end
  def duration
    @duration
  end
end
```
shortcut

class Song
  attr_reader :name, :artist, :duration
End

class Song
  attr_writer :duration
end

Calls wide stuff

- @@
  - Needs to be initialized
- Can also have class specific functions

class Example

  def instMeth  # instance method
    end

  def Example.classMeth  # class method
    end

end
Levels of privacy

- Can control how much access you want to share for each part of the class
  - Public
    - No privacy
  - Private
    - Complete private, no one except own class
  - Protected
    - Own class and subclasses

```ruby
class MyClass
  def method1    # default is 'public'
    #...
  end
  protected     # subsequent methods will be 'protected'
  def method2    # will be 'protected'
    #...
  end
  private       # subsequent methods will be 'private'
  def method3    # will be 'private'
    #...
  end
  public        # subsequent methods will be 'public'
  def method4    # and this will be 'public'
    #...
  end
end
```
Method II

class MyClass

  def method1
  end

  # ... and so on

  public :method1, :method4
  protected :method2
  private :method3
end

What is this?

class SongList
  def [](key)
    if key.kind_of?(Integer)
      return @songs[key]
    else
      for i in 0...@songs.length
        return @songs[i] if key == @songs[i].name
      end
    end
    return nil
  end
end
- yield allows a block of code to run inside a function
- Can pass it args

```python
def fibUpTo(max):
    i1, i2 = 1, 1        # parallel assignment
    while i1 <= max
        yield i1
        i1, i2 = i2, i1+i2
    end
end
fibUpTo(1000) { |f| print f, " " }  
```
So Iterators unlike c++ and Java are part of the class not independent

Natural extension of object manipulation

```ruby
f = File.open("testfile")
f.each do |line|
  print line
end
f.close
```
closure

- Proc object contains all current environment snapshot of running state for use later

```ruby
def nTimes(aThing)
    return proc { |n| aThing * n }
end

p1 = nTimes(23)
p1.call(3) \» 69
p1.call(4) \» 92
p2 = nTimes("Hello ")
p2.call(3) \» "Hello Hello Hello 
```

Ranges

- 1..10
- 'a'..'z'
- 0...anArray.length

- Allows you to specify range of values
- Internal data structure with start and end values
(1..10).to_a
digits = 0..9
digits.include?(5) » true
digits.min » 0
digits.max » 9
digits.reject {|i| i < 5 } » [5, 6, 7, 8, 9]
digits.each do |digit|
  dial(digit)
end

More ranges

while gets
  print if /start/../end/
end
Exception Handling

```ruby
opFile = File.open(opName, "w")
begind
  # Exceptions raised by this code will
  # be caught by the following rescue clause
  while data = socket.read(512)
    opFile.write(data)
  end
rescue SystemCallError
  $stderr.print "IO failed: " + $!
  opFile.close
  File.delete(opName)
  raise
end
```

More info

- [http://dev.rubycentral.com/ref/](http://dev.rubycentral.com/ref/)
next

- Read up more on ruby on the web
- Install ruby-rails on your local machine
- Will meet on Wednesday and cover rails
- End of week, you should be creating cool web based stuff (if server runs ruby 😊 )