Objects in Ruby

Lists (aka Arrays)
Objects, Classes, and Methods
Strings
Numbers
Motivation: objects are the building blocks for algorithms written in Ruby

Definitions

- A list is an ordered collection of data items
- For now we will consider only lists of numbers
  - in general lists can contain any type of data
  - lists of strings, lists of lists, ...
  - most languages also allow mixed lists
- If A is the name of a list, use the notation A_i to refer to the i-th element
  - the first element is A_0
  - when computer scientists count they start at 0
  - if the list has n elements, A_{n-1} is the last element
- This subscript notation is used in English descriptions of algorithms and often (but not always) in pseudo-code
  - we'll see a different notation for lists in Ruby and other programming languages

Aside: Lists vs Arrays

- A closely related data structure is called an array
- Arrays are also collections of elements, and we also use an index to refer to individual elements of an array
- Differences between arrays and lists:
  - lists can grow and shrink; arrays have a fixed size
  - one can insert items into the middle of a list; to insert to an array things have to be moved
  - due to the way they are stored in a computer’s memory, it is usually faster to access an element in the middle of an array

Lists

- Most algorithms work with sets of data elements
- Examples:
  - sorting, searching
  - recall the pseudo-code for insertion sort referred to items A_0, A_1, ...
  - the subscript is the “index”, or the location of an item in the list
- Counter-examples:
  - GCD, integer factorization
  - these algorithms refer to individual numbers (e.g. n, m, r)
- Programs use data structures to hold sets of values
  - sometimes called containers
Lists vs Arrays (cont’d)

- Alas, Ruby has collections of data elements that look like and act like lists, but they are called “arrays”
- In CIS 170 the distinction is probably not important, and from now on (unless it will cause confusion) we will call these things “arrays”
  - when I talk about algorithms I still may use the word “list”
  - when I talk about Ruby programs I will use “array”

Creating Arrays in Ruby

- In Ruby, elements of an array are separated by commas and enclosed in square brackets:
  ```ruby
  >> a = [2,3,5,7,11,13]
  => [2, 3, 5, 7, 11, 13]
  ```
- To refer to an element in an array, put the index in square brackets after the array name:
  ```ruby
  >> a[0]
  => 2
  >> a[5]
  => 13
  ```
- You can use a negative number for an index: Ruby interprets \(-n\) as “\(n\) elements from the right”
  ```ruby
  >> a[-1]
  => 13
  ```

Creating Arrays in Ruby (cont’d)

- There are two ways to make an empty array
  - an empty array has zero elements
  - elements will be added as the program progresses
- One way to create such an array: use the array notation with no elements between the brackets:
  ```ruby
  >> a = []
  => []
  ```
- Another way: use a special function named “new”:
  ```ruby
  >> a = Array.new
  => []
  ```

Objects

- In programming languages, an object is something that can be manipulated by an algorithm
- Important idea: types of objects are defined by the kinds of operations that can be performed
- Example: numbers
  - integers and real numbers are simple objects
  - things we can do to numbers include add, subtract, multiply, divide, ...
- Example: arrays
  - create new array objects
  - insert items into or delete items from an array
  - sort, reverse, concatenate, and many more operations
Operations on Objects

- In Ruby and other object-oriented languages, to apply an operation to an object you write the operation's name after the object's name.

Example: operations on array objects

```ruby
>> a = [2,3,5,7,11,13,17,19,23]
=> [2, 3, 5, 7, 11, 13, 17, 19, 23]
>> a.length
=> 9
>> a.max
=> 23
>> a.reverse
=> [23, 19, 17, 13, 11, 7, 5, 3, 2]
```

Terminology

- Some terminology from object-oriented programming:
  - The type of an object is called its **class**
    - Class names in Ruby start with upper case letters
    - Array, String, etc are classes in Ruby
  - Individual objects are **instances** of a class
    - The object `a` from previous slides is an instance of the Array class
  - Operations on objects are called **methods**
    - `reverse`, `sort`, and `max` are methods of the Array class
    - When we apply an operation we invoke or `call` the method

Parameters

- Some operations require operands
  - These are called **parameters** in programming languages
- Supply parameters in a list enclosed in parentheses after the method name
- The operation that adds an item to the end of an array is called **push**

```ruby
>> a = [2,3,5,7,11,13,17,19,23]
=> [2, 3, 5, 7, 11, 13, 17, 19, 23]
>> a.push(29)
=> [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Results of Expressions

- Consider this example of a series of expression evaluations involving integers:

```ruby
>> x = 3
=> 3
>> y = 4
=> 4
>> x**2 + y**2
=> 25
```

- It seems obvious, but it's worth pointing out that evaluation of \( x^2 + y^2 \) does not modify `x` or `y`:

```ruby
>> x
=> 3
>> y
=> 4
```

- Ruby uses the value of `x` and `y` as inputs to a calculation and then it prints the results.
Results of Expressions (cont’d)

- The same is true of most expressions involving arrays
- Calling a method tells Ruby to use the value of an object in a calculation, and the result of the calculation is then printed

```ruby
>> a
= [{2, 3, 5, 7, 11, 13, 17, 19}
>> a.reverse
= [{19, 17, 13, 11, 7, 5, 3, 2}
>> a
= [{2, 3, 5, 7, 11, 13, 17, 19}
```

- The `reverse` method does not modify the object
- The call `a.reverse` means “create a new array by copying the elements of `a` in reverse order”

Chains of Operations

- The fact that method calls return new objects makes it easy to write very concise expressions
- Example: the `uniq` method returns a copy of an array with all duplicates removed

```ruby
>> a = [2,3,5,3,2,6,2,4,3]
>> a
= [{2, 3, 5, 3, 2, 6, 2, 4, 3]
>> a.uniq
= [{2, 3, 5, 6, 4}]
```

- This expression asks how many unique values there are in `a`:

```ruby
>> a.uniq.length
= 5
```

- This sorts the unique values:

```ruby
>> a.uniq.sort
= [{2, 3, 4, 5, 6}]
```

Q: what would I get if I typed `a.sort.uniq`?

More Conventions

- Ruby allows one to use `!` in names
- By convention, a method that modifies an object has `!` at the end of its name
- The `Array` class has `reverse!` and `sort!` and others

```ruby
>> a
= [{2, 3, 5, 3, 2, 6, 2, 4, 3}
>> a.sort
= [{2, 2, 2, 3, 3, 4, 5, 6}
>> a
= [{2, 3, 5, 3, 2, 6, 2, 4, 3}
>> a.sort!
= [{2, 2, 2, 3, 3, 4, 5, 6}
>> a
= [{2, 2, 2, 3, 3, 4, 5, 6}
```

- But note `push`, `pop`, and some other “destructive” methods don’t have `!` in their names...

Additional Methods

- For programming assignments in CIS 170 you are welcome to use any methods you discover
- To learn more about the `Array` class and all the methods that are available:
  - look in the appendix of Programming Ruby and other books
  - look on-line, e.g. http://www.ruby-doc.org/core/
- But be forewarned: these documents are written for experienced programmers; the organization and descriptions might be hard for novices
- For CIS 170 projects all the information you need for projects will be available on our class web site
  - lecture notes
  - project descriptions
  - special handouts on specific topics
Strings

- A string in a programming language is a collection of characters.
- You can think of it as a list of characters.
- The notation for writing strings is to enclose the letters in quotes.

```ruby
>> s = "hello"
=> "hello"
>> x = "fred"
=> "fred"
>> address = "1477 E. 13th Ave, Eugene"
=> "1477 E. 13th Ave, Eugene"
```

- Note that any character can be included in a string.
- A topic for some other time: UNICODE and other methods for creating strings with "international" characters like ñ, ç, é, å, ...

Strings (cont’d)

- You can use either single or double quotes.

```ruby
>> name = 'ethel'
=> "ethel"
>> greeting = "Hi, Ethel"
=> "Hi, Ethel"
```

- The greeting example shows how special characters are marked inside strings.
- There are many of these, e.g. "\n" means "return" or "newline".
- There are reasons to prefer one form or the other, but we’ll save that discussion for another day -- for now use whichever form you prefer.

Strings of Digits

- Important note: a string of digits is not the same thing as an integer!

```ruby
>> n = "12"
=> "12"
>> n + 4
TypeError: can’t convert Fixnum into String
from (irb):10:in `+'
```

- What Ruby is telling us here is that "12" is a string object and 4 is a number object, and that it won’t add a string and a number.
- If you have experience with scripting languages (e.g. Perl) you may find this surprising -- scripting languages automatically convert strings to numbers and vice versa.

Overloaded Operators

- It is possible to use + in expressions involving strings.

```ruby
>> greeting = "hola"
=> "hola"
>> phrase = greeting + "! que tal?"
=> "hola! que tal?"
```

- In computer science jargon, + is an overloaded operator.
- It is the name of many different methods.
- One method operates on integers, another on strings.
- Ruby selects the method to use depending on the context.
- In 4 + 5 the + symbol denotes the method that adds integers.
- In "hola" + "! que tal?" the + symbol denotes a method of the String class.
Methods of the String Class

- The String class has many useful methods
  
  ```ruby
  >> phrase.reverse
  => "?lat euq !aloh"
  >> phrase.index("que")
  => 6
  >> phrase += "?"
  => "hola! que tal??"
  >> phrase.capitalize
  => "Hola! que tal??"
  >> phrase.crypt("key")
  => "ke2WNahUbVTkJ6"
  
  - Note String and Array both have `reverse` methods -- which one is called depends on the type of the object

Methods of the String Class (cont’d)

- The `tr` method translates letters
  - Takes two parameters: a list of letters to translate, and a list of translations
    - both “lists” are actually strings
  - Example: replace every vowel by its upper case equivalent
    ```ruby
    >> s = "hello there"
    => "hello there"
    >> s.tr("aeiou", "AEIOU")
    => "HElLO thErE"
    
    - If the second list is shorter than the first, Ruby just reuses the last letter:
      ```ruby
      >> s.tr("aeiou", "*")
      => "h*ll* th*r*"
      ```

Split and Join

- Two very useful methods of the string class are named `split` and `join`
  - `split` divides a string into several smaller strings and returns an array of these smaller strings
  - The default is to split the array into words (substrings separated by spaces):
    ```ruby
    >> s = "Fred Ethel Ricky Lucy"
    => "Fred Ethel Ricky Lucy"
    >> s.split
    => ["Fred", "Ethel", "Ricky", "Lucy"]
    
    - If you pass a parameter to `split`, Ruby breaks the string at places where the parameter value is found:
      ```ruby
      >> s.split("e")
      => ["Fr", "d Eth", "l Ricky Lucy"]
      ```

Split and Join (cont’d)

- `join` is the opposite of `split` -- it is a method of the Array class, and it creates a new string by combining all the elements in the array into a string
  ```ruby
  >> a = ["one", "two", "three"]
  => ["one", "two", "three"]
  >> a.join
  => "onetwothree"
  
  - You can pass a parameter to `join` to give it a string to put between the elements:
    ```ruby
    >> a.join("--")
    => "one--two--three"
    ```
  - `split` and `join` have many other options -- read about them in the documentation
Numbers Are Objects

- Numbers are objects in Ruby
- Previous slides hinted at this idea
  - + refers to a method that is implemented in at least two classes
  - the error message said something about combining a Fixnum and a String
- Fixnum is the name of the Ruby class for integers
  - 2, 13, 52, etc are all instances of the Fixnum class
- Another way to write an expression that adds two numbers:
  
  ```ruby
  >> n = 5
  => 5
  >> n.+(8)
  => 13
  ```
  - emphasizes that + is the name of a method in the Fixnum class

Methods for Fixnum Objects

- Ruby has several methods for Fixnum objects and other numbers
- A glimpse at one of the topics for the next set of slides on Ruby:
  
  ```ruby
  5.times do |i|
    puts i
  end
  0
  1
  2
  3
  4
  ```
  - times is a method defined for Fixnum objects
  - puts stands for "put string" -- Ruby's name for a method that prints strings
  - do..end is a block -- a loop that is executed the specified number of times
  - i is an index variable, a value that changes for each loop iteration