Associative Arrays

Another container object: Hash

Arrays in Ruby are collections of objects:

```ruby
>> a = [ "hello", "bonjour", "konichi-wa" ]
=> ["hello", "bonjour", "konichi-wa"]
```

Elements in arrays are accessed by their position (index):

```ruby
>> a[2]
=> "konichi-wa"
```

Arrays are examples of a more general concept of a container

Today: another type of container

Tables

- Recall how we implemented the mapping between grades (strings) and their numeric values:

  ```ruby
  val = case grade
  when "A": 4
  when "B": 3
  when "C": 2
  when "D": 1
  else 0
  end
  ```

  This code is clear and easy to understand, but it is inefficient.
  - Ruby scans the labels from top to bottom until it finds a match.

Tables (cont’d)

- For another example, consider a program that translates a DNA string into the corresponding string of amino acid letters:

  ```ruby
  aa = case codon
  when "AAA": "K"
  when "AAG": "K"
  when "AAC": "N"
  when "AGA": "R"
  ...
  when "AGA": "R"
  else 0
  end
  ```

  The translation is defined by the genetic code.
  - There are 64 (4³) combinations to consider.
What we’d really like to do is look up a value in a table. There are lots of ways of looking things up in tables:

- time to find an item can be (nearly) constant, does not depend on table size
- for more info see books on data structures and algorithms (CIS 211/313/315)

<table>
<thead>
<tr>
<th>Grade</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Codon</th>
<th>Amino Acid</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>K</td>
</tr>
<tr>
<td>AAT</td>
<td>K</td>
</tr>
<tr>
<td>AAC</td>
<td>N</td>
</tr>
<tr>
<td>AAG</td>
<td>N</td>
</tr>
<tr>
<td>AGA</td>
<td>R</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

There are lots of ways of looking things up in tables:

- for more info see books on data structures and algorithms (CIS 211/313/315)

Inserting New Items

To add a new key/value pair to a Hash object just use an assignment:

```ruby
>> gc = Hash.new
=> {}
>> gc.length
=> 0
>> gc["AAA"] = "K"
=> "K"
>> gc["AAT"] = "K"
=> "K"
>> gc
=> {"AAA"=>"K", "AAT"=>"K"}
>> gc.length
=> 2
```

Note: a real program would probably read the genetic code keys and values from a file instead of having a sequence of 64 assignment statements...

Hashes

- Tables like these are examples of **associative arrays**
  - a collection of associations between **keys** and **values**
  - another term: **map**, or **mapping**

An associative container in Ruby is called a “hash” (from the CS term “hash table”)

- A simple way to make a Hash object is to just list the key/value pairs between curly braces:
  ```ruby
  >> gmap = { "A" => 4, "B" => 3, "C" => 2, "D" => 1, "F" => 0 }
  => {"A"=>4, "B"=>3, "C"=>2, "D"=>1, "F"=>0}
  ```

- To access an element, use the same square brackets you use for regular arrays, but instead of a position use a key:
  ```ruby
  >> gmap["B"]
  => 3
  ```

Keys

- What do you suppose will happen if you try to access an item in a Hash object with a position instead of a key?
  ```ruby
  >> gmap.class
  => Hash
  >> gmap[0]
  => nil
  ```

Ruby didn’t generate an error message

The nil means “there is no object in this table with that key”

- If you want to see a list of keys of objects currently in a Hash object use the keys method:
  ```ruby
  >> gmap.keys
  => ["A", "B", "C", "D", "F"]
  ```
Keys (cont’d)

- Ruby lets you mix the types of items used for keys:
  ```ruby
  gmap[0] = "use a letter grade"
  #=> "use a letter grade"
  gmap.keys
  #=> [0, "A", "B", "C", "D", "F"]
  gmap[0]
  #=> "use a letter grade"
  ```

Explore

- As usual, I won’t try to give a list of methods of the Hash class
- Read the documentation, get a general sense of the types of methods
  - constructors (ways of creating new Hash objects)
  - adding elements
  - deleting elements
  - finding keys or values

One Last Example

- One of the benefits of using a Hash object for the GPA program is that it will help do error-checking for inputs
- we should be checking to make sure each item in ARGV is a valid grade
- Use a method of the Hash class to get an Array of keys
- Use a method of the Array class to see if a string is in the array of keys
  ```ruby
  gmap
  #=> {"A"=>4, "B"=>3, "C"=>2, "D"=>1, "F"=>0}
  gmap.keys
  #=> ["A", "B", "C", "D", "F"]
  gmap.keys.index("C")
  #=> 2
  gmap.keys.index("P")
  #=> nil
  ```

Symbols

- In idiomatic Ruby programmers often use symbols for keys in a hash
- A symbol is a name that starts with a colon
  ```ruby
  x = :green
  #=> :green
  x.class
  #=> Symbol
  x.to_s
  #=> "green"
  ```
Symbols

What's the difference between a string and a symbol? Why use symbols?
+ symbols are constants, strings are mutable
+ symbols are very compact -- represented internally as integers

```ruby
>> x.to_i
=> 24081
>> y = :yellow
=> :yellow
>> y.to_i
=> 24089
```

When should you consider using a symbol?
Think of symbols as labels
+ convenient, mnemonic name for something that could also be represented by an integer

Example: create a histogram of test scores
+ read scores, assign to one of three categories
+ category labels could be numbers (0, 1, 2) but the program is easier to follow if the categories are given symbolic names

```ruby
>> hist = {:good => 0, :bad => 0, :ugly => 0}
=> {:good=>0, :bad=>0, :ugly=>0}
>> hist.keys
=> [:good, :bad, :ugly]
>> hist[:good] += 1
=> 1
```