CMSC 330: Organization of Programming Languages

Array, Hashes, Code Blocks, Equality
Arrays and Hashes

- Ruby data structures are typically constructed from Arrays and Hashes
  - Built-in syntax for both
  - Each has a rich set of standard library methods
  - They are integrated/used by methods of other classes
Arrays of objects are instances of class **Array**

- Arrays may be **heterogeneous**
  
  ```ruby
  a = [1, "foo", 2.14]
  ```

- **C-like syntax for accessing elements**
  
  - indexed from 0
  
  - return **nil** if no element at given index

  ```ruby
  irb(main):001:0> b = []; b[0] = 0; b[0]
  => 0
  
  irb(main):002:0> b[1]  # no element at this index
  => nil
  ```
Arrays Grow and Shrink

- Arrays are growable
  - Increase in size automatically as you access elements
    ```ruby
    irb(main):001> b = []; b[0] = 0; b[5] = 0; b
    => [0, nil, nil, nil, nil, 0]
    - `[ ]` is the empty array, same as `Array.new`

- Arrays can also shrink
  - Contents shift left when you delete elements
    ```ruby
    a = [1, 2, 3, 4, 5]
    a.delete_at(3)  # delete at position 3; a = [1,2,3,5]
    a.delete(2)    # delete element = 2; a = [1,3,5]
    ```
Iterating Through Arrays

- It's easy to iterate over an array with `while`
  - `length` method returns array’s current length

```ruby
a = [1,2,3,4,5]
i = 0
while i < a.length
  puts a[i]
i = i + 1
end
```

- Looping through elements of an array is common
  - We’ll see a better way soon, using code blocks
Arrays as Stacks and Queues

- Arrays can model stacks and queues

  ```javascript
  a = [1, 2, 3]
  a.push("a")    # a = [1, 2, 3, "a"]
  x = a.pop       # x = "a"
  a.unshift("b") # a = ["b", 1, 2, 3]
  y = a.shift    # y = "b"
  ```

Note that `push`, `pop`, `shift`, and `unshift` all permanently modify the array.
Hash

- A hash acts like an associative array
  - Elements can be indexed by *any kind* of values
  - Every Ruby object can be used as a hash key, because the `Object` class has a `hash` method

- Elements are referred to like array elements

```ruby
italy = Hash.new
italy["population"] = 58103033
italy["continent"] = "europe"
italy[1861] = "independence"
pop = italy["population"]  # pop is 58103033
planet = italy["planet"]   # planet is nil
```
Hash methods

- **new(o)** returns hash whose default value is o
  - h = Hash.new(“fish”); h[“go”]  # returns “fish”

- **values** returns array of a hash’s values

- **keys** returns an array of a hash’s keys

- **delete(k)** deletes mapping with key k

- **has_key?(k)** is true if mapping with key k present
  - has_value?(v) is similar
Hash creation

Convenient syntax for creating literal hashes

- Use \{ key => value, ... \} to create hash table

```ruby
credits = {
    "cmsc131" => 4,
    "cmsc330" => 3,
}

x = credits["cmsc330"]  # x now 3
credits["cmsc311"] = 3
```

- Use {} for the empty hash
Quiz 1: What is the output

```python
a = {"foo" => "bar"}
a[0] = "baz"
print a[0]
print a[1]
print a["foo"]
```
Quiz 1: What is the output

```plaintext
a = {"foo" => "bar"}
```a[0] = "baz"
```
print a[0]
print a[1]
print a["foo"]
```

A. Error
B. barbaz
C. bazbar
D. baznilbar
Quiz 2: What is the output

```ruby
a = { "Yellow" => [] }
a["Yellow"] = {}
a["Yellow"]["Red"] = ["Green", "Blue"]
puts a["Yellow"]["Red"][1]
```

A. Green
B. *(nothing)*
C. Error
D. Blue
Quiz 2: What is the output

```ruby
a = { "Yellow" => [] }
a["Yellow"] = {}
a["Yellow"]["Red"] = ["Green", "Blue"]
puts a["Yellow"]["Red"][1]
```

A. Green  
B. (nothing)  
C. Error  
D. Blue
Quiz 3: What is the output

a = [1,2,3]
a[1] = 0
a.push(1)
print a[1]

A. 2
B. 1
C. 0
D. (nothing)
Quiz 3: What is the output

A. 2
B. 1
C. 0
D. (nothing)