Web Programming

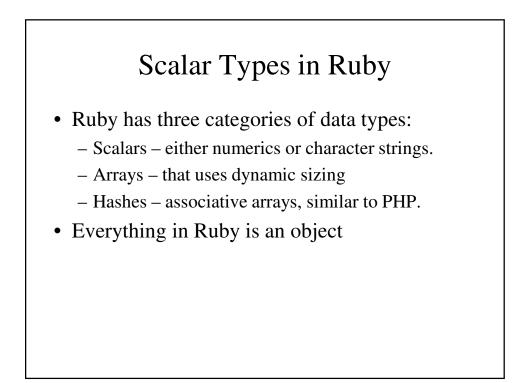
Lecture 9 – Introduction to Ruby

Origins of Ruby

- Ruby was designed by Yukihiro Matsumoto ("Matz") and released in 1996.
- It was designed to replace Perl and Python, which Matz considered inadequate.
- It grew quickly in Japan and then spread around the world.
- Its expansion was a result of the increasing popularity of Rails, a Web development framework that was written in Ruby and that uses Ruby.

Uses of Ruby

- Because Ruby is implemented by pure interpretation, it's easy to use.
- Example
 irb(main):001:0> puts "hello, world"
 hello, world
 => nil
- Ruby uses regular expressions and implicit variables like Perl, objects like JavaScript but is quite different from these languages.



Numeric and String Literals

• All numeric data types are derived from the base class Numeric, has two derived classes Float and Integer.

Integer Literals

- Integer has two derived classes:
 - **FixNum** fits the range of a machine word (usually 32 bits).
 - BigNum numbers outside the range of FixNum. (if an operation on BigNum produces a smaller value, it will be coerced into FixNum).
- Ruby ignores underscores in integer literals so they can be more readable.
 - 1_234_567_890 is more readable than 1234567890

Float Literals

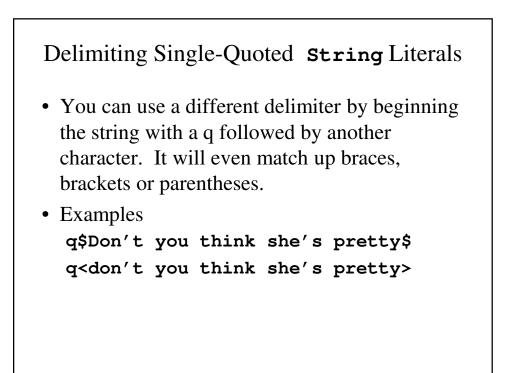
- A numeric literal with either an embedded decimal point or an exponent following it is a **Float** object.
- **Float** objects are stored as double-precision floating point numbers.
- Decimal points must have a digit on both sides of it.

String Literals

- All string literals are **String** objects, which are sequences of bytes that represent characters.
- String objects are either single-quoted or double-quoted.

Single-Quoted String Literals

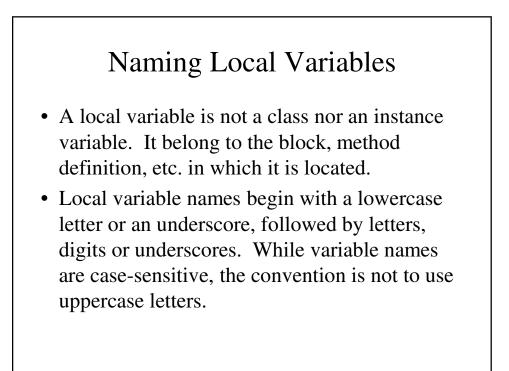
- Single quoted strings cannot have escape sequences.
- Examples
 - 'I\'ll meet you at O\'Malleys'
 - the inner apostrophes are included correctly.
 - 'Some apples are red, \n some are green.'
 contains a backslash followed by n (not a newline).



Double-Quoted String Literals

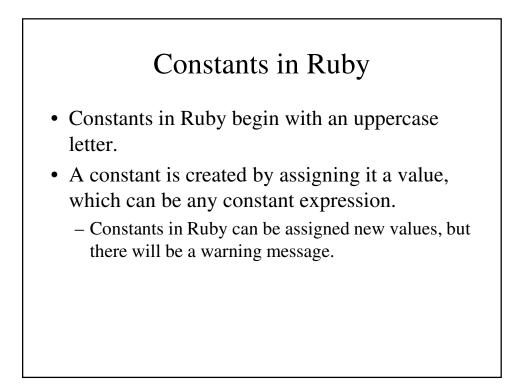
- Double-quoted strings can contain the special characters specified by escape sequences. And the values of variable names can be substituted into the string.
- Example
 - "Runs \t Hits \t Errors" will include the expected tabs
- For a different delimited for double-quotes strings, begin the string with Q:

- Q@"Why not learn Ruby", he asked.@



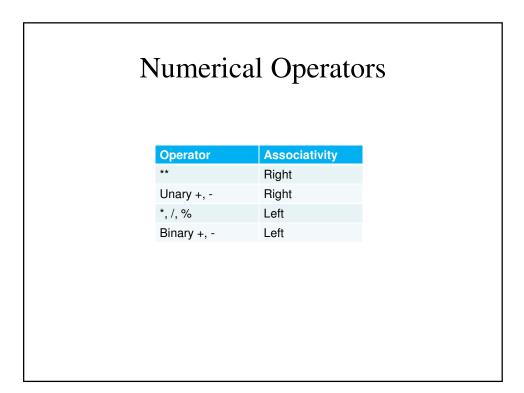
Using Variables In Strings The value associated with a local variable can be inserted in a double-quoted string: "Tuesday's high temperature was #{tue_high} " is printed as "Tuesday's high temperature was 83" Everything in Ruby is an object, so we are really

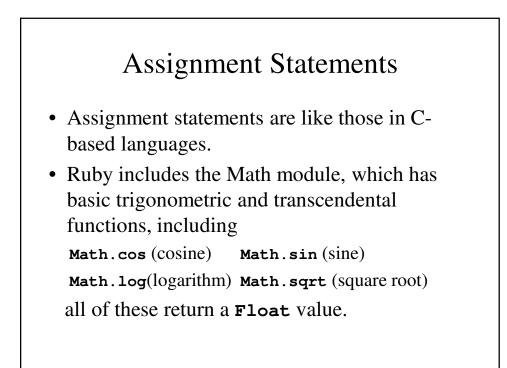
• Everything in Ruby is an object, so we are really working with their references, which are typeless. As a result, all variables are implicitly declared (how we use them determines their type).

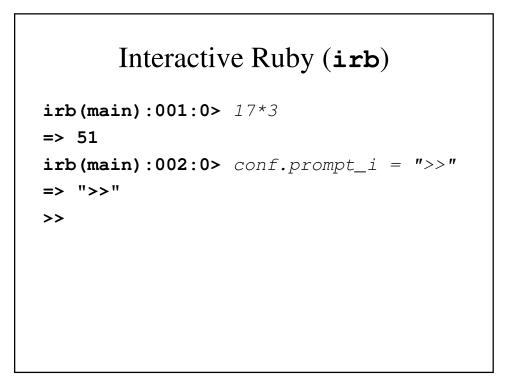


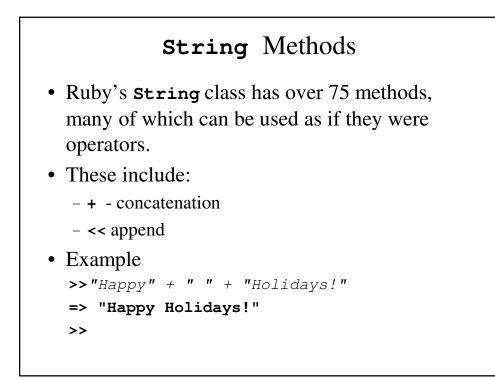


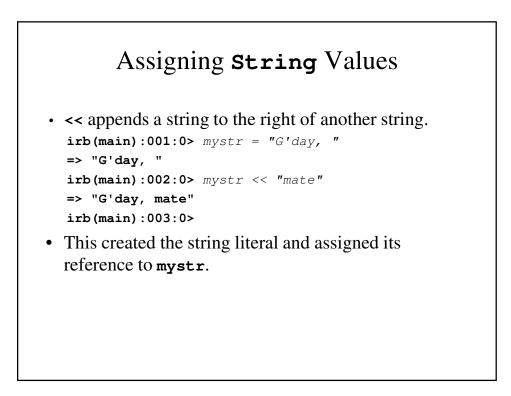
- Ruby has predefined variables (like Perl), which consist of \$ followed by a special character.
 - Examples \$_, \$^, \$\











Assigning String Values (continued)

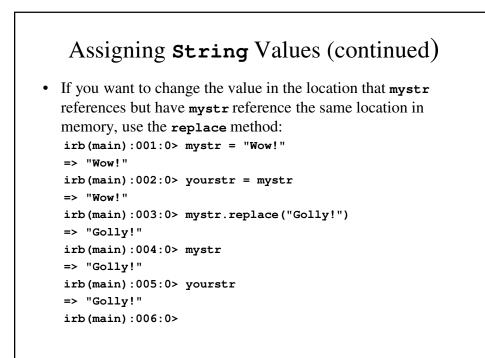
```
irb(main):003:0> mystr = "Wow!"
=> "Wow!"
irb(main):004:0> yourstr = mystr
=> "Wow!"
irb(main):005:0> yourstr
=> "Wow!"
irb(main):006:0>
```

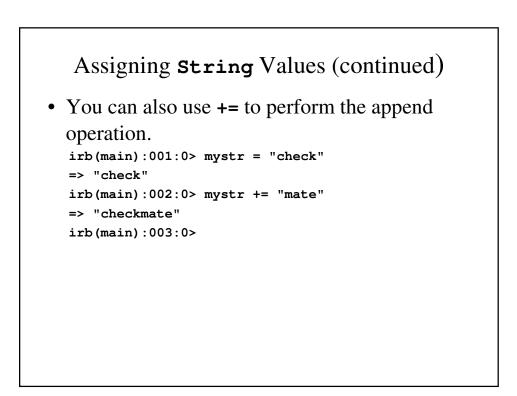
• Ruby assigned yourstr a copy of the same reference that mystr held.

Assigning String Values (continued)

```
irb(main):001:0> mystr = "Wow!"
=> "Wow!"
irb(main):002:0> yourstr = mystr
=> "Wow!"
irb(main):003:0> mystr = "What?"
=> "What?"
irb(main):004:0> yourstr
=> "Wow!"
irb(main):005:0>
```

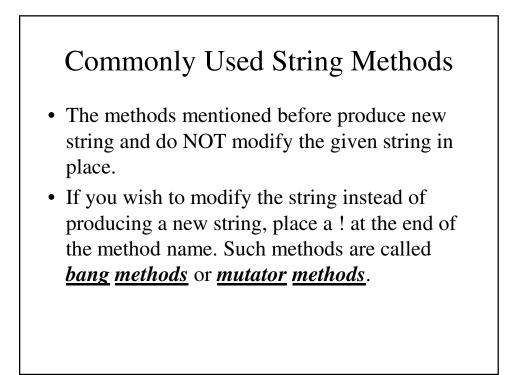
• After the assignment, yourstr has the same reference as mystr. But when mystr is assigned a different string literal, Ruby sets aside another memory location for the new literal and that is the reference that mystr now holds.





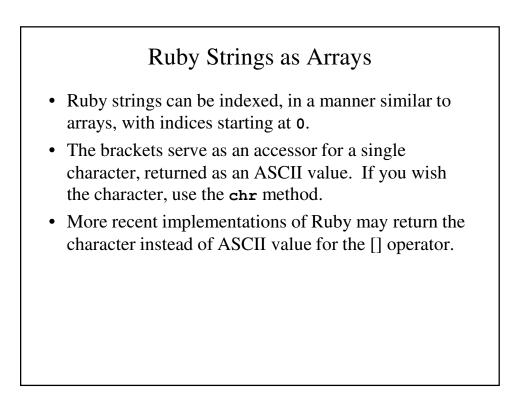
Commonly Used String Methods

Method		
capitalize	Converts the first letter to uppercase and the rest of the letters to lowercase	
chop	Removes the last character	
chomp	Removes a newline from the right end if there is one	
upcase	Converts all of the lowercase letters in the object to uppercase	
downcase	Converts all of the uppercase letters in the objects to lowercase	
strip	Removes the spaces on both ends	
lstrip	Removes the spaces on the left end	
rstrip	Removes the spaces on the right end	
reverse	Reverses the characters of the string	
swapcase	Converts all uppercase letters to lowercase and all lowercase letters to uppercase	



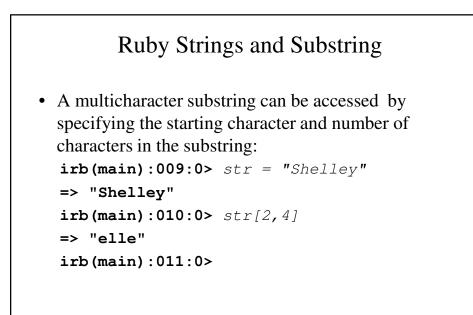
Mutator Methods – An Example

```
irb(main):001:0> str = "Frank"
=> "Frank"
irb(main):002:0> str.upcase
=> "FRANK"
irb(main):003:0> str
=> "Frank"
irb(main):004:0> str.upcase!
=> "FRANK"
irb(main):005:0> str
=> "FRANK"
irb(main):006:0>
```



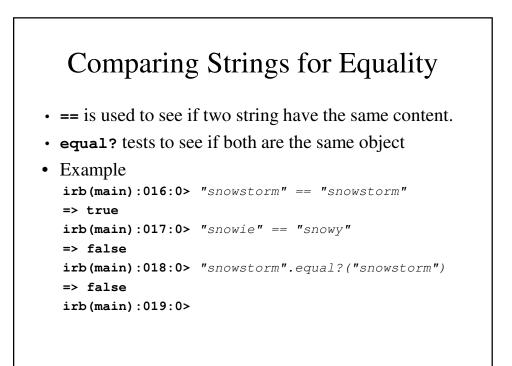
Ruby Strings as Arrays – An Example

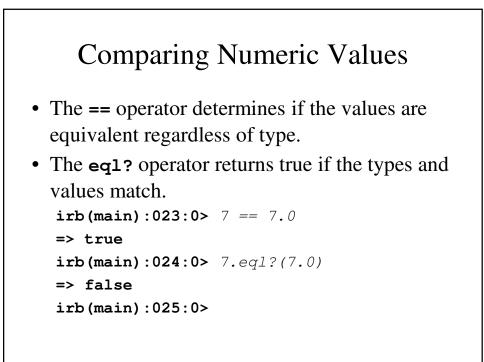
```
irb(main):006:0> str = "Shelley"
=> "Shelley"
irb(main):007:0> str[1]
=> "h"
irb(main):008:0> str[1].chr
=> "h"
```

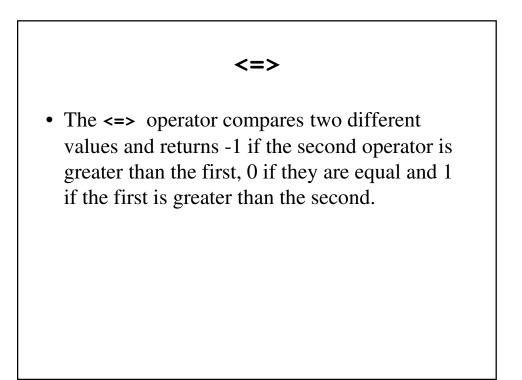


Changing a String With a Substring

The [] = operator can be used to specify characters of a substring and to what they are be changed:
irb(main):013:0> str = "Donald"
=> "Donald"
irb(main):014:0> str[3,3] = "nie"
=> "nie"
irb(main):015:0> str
=> "Donnie"
irb(main):016:0>

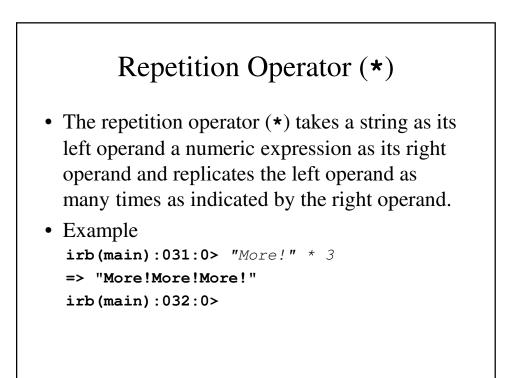






<=> - Examples

```
irb(main):025:0> 7 <=> 5
=> 1
irb(main):026:0> "grape" <=> "grape"
=> 0
irb(main):027:0> "grape" <=> "apple"
=> 1
irb(main):030:0> "apple" <=> "prune"
=> -1
irb(main):031:0>
```



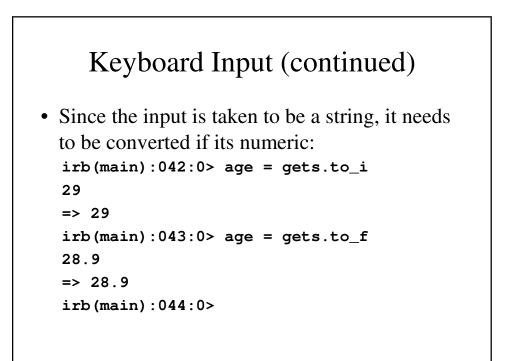
Screen Output

- Output is directed to the screen using the puts method (or operator).
- The operand for puts is a string literal with a newline implicitly appended to the end.
- A variable's value can be included in the string by writing #{variableName}
- **print** works in the same way except with the included newline.
- **sprintf** works as it does in C, allowing for formatted output.

Screen Output – An Example

```
irb(main):032:0> name = "Pudgy"
=> "Pudgy"
irb(main):033:0> puts "My name is #{name}"
My name is Pudgy
=> nil
irb(main):034:0> print "My name is #{name}"
My name is Pudgy=> nil
irb(main):035:0> total = 10
=> 10
irb(main):036:0> str = sprintf("%5.2f", total)
=> "10.00"
irb(main):037:0>
```

Example Content of the set o



```
quadeval.rb
```

```
#quadeval.rb - A simple Ruby program
# Input:
           Four numbers, representing the values of
            a, b, c, and x
#
            The value of the expression
# output:
            a*x**2 _ b*x + c
#
# Get input
puts "please input the value of a"
a = gets.to_i
puts "please input the value of b"
b = gets.to_i
puts "please input the value of c"
c = gets.to_i
```

```
# compute and display the result
result = a * x ** 2 + b * x + c
puts "The value of the expression is #{result}"
```

Running quadeval.rb

```
C:\>ruby quadeval.rb

please input the value of a

1

please input the value of b

2

please input the value of c

1

Please input the value of x

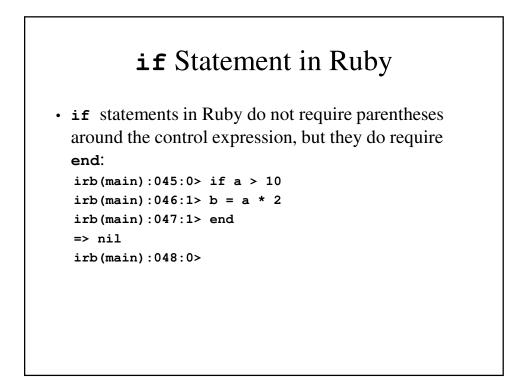
5

The value of the expression is 36

C:\>
```

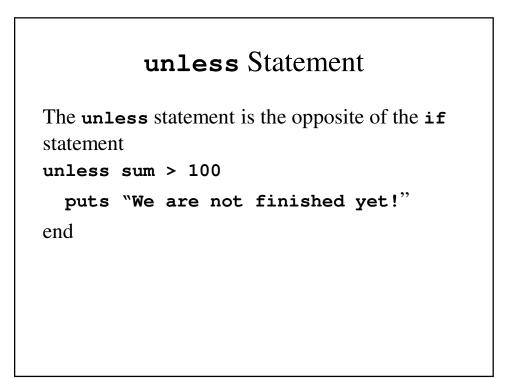
Relational Operators			
Operator	Operation		
==	Is equal to		
!=	Is not equal to		
<	Is less than		
>	Is greater than		
<=	Is less than or equal to		
>=	Is greater than or equal to		
<=>	Compare, returning -1, 0 or +1		
eql?	True if the receiver object and the parameter have the same type and equal values		
equal?	True if the receiver object and the parameter have the same object ID		

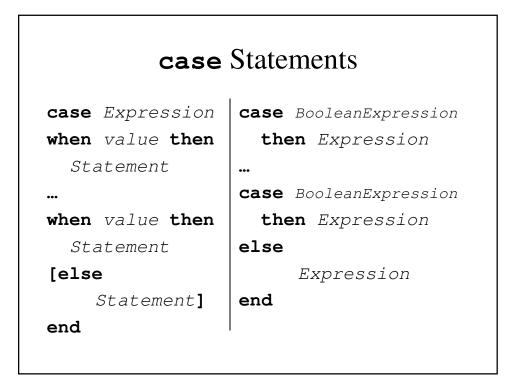
Operator Precedence			
Operator	Associativity		
**	Right		
!, unary + and –	Right		
*, /, %	Left		
+	Left		
æ	Left		
+, -	Left		
>, <, >=, <=	Nonassociative		
==, !=, <=>	Nonassociative		
& &	Left		
11	Left		
=, +==, *=, **=, /=, %=, &=, &&=, =	Right		
not	Right		
or, and	Left		

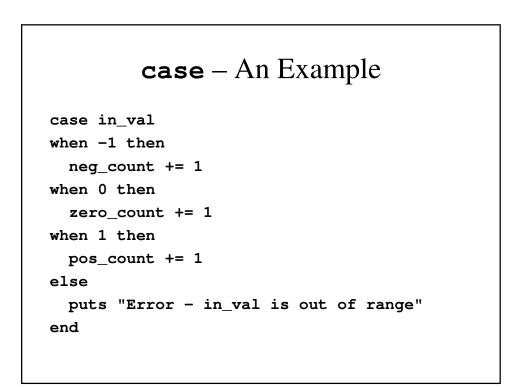


if..elsif..else

```
if snowrate < 1
   puts "Light snow"
elsif snowrate < 2
   puts "Moderate snow"
else
   puts "Heavy snow"
end</pre>
```

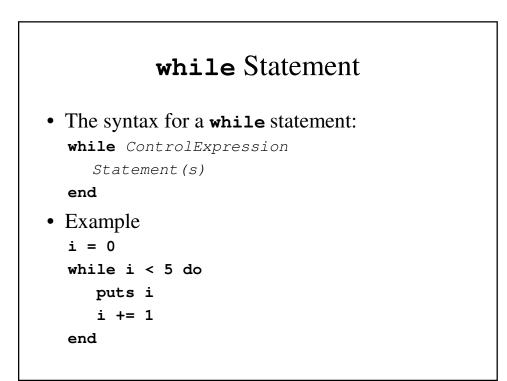


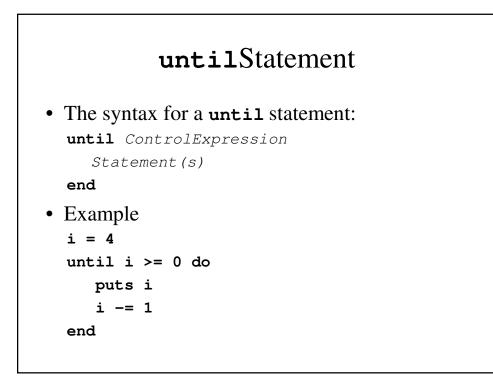


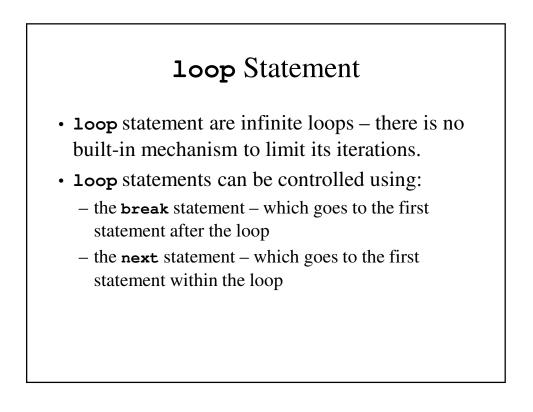


case - An Example

leap = case
 when year % 400 then true
 when year % 100 then false
 else year %4 == 0
 end







loop Statement - Examples

```
sum = 0
loop do
  dat = gets.to_i
  if dat < 0 break
  sum += dat
end</pre>
```

```
sum = 0
loop do
  dat = gets.to_i
  if dat < 0 next
  sum += dat
end</pre>
```

Arrays in Ruby In Ruby, array size is dynamic, growing and shrinking as necessary Arrays in Ruby can store different types of data in the same array. Arrays can be created by: Using the predefined Array class. Assign a list literal to a variable.

Initializing Arrays - Examples

```
irb(main):001:0> list1 = Array.new(5)
=> [nil, nil, nil, nil]
irb(main):002:0> list2 = [2, 4, 3.14159, "Fred", [] ]
=> [2, 4, 3.14159, "Fred", []]
irb(main):003:0> list3 = Array.new(5, "Ho")
=> ["Ho", "Ho", "Ho", "Ho", "Ho"]
irb(main):004:0>
```

Working With Arrays - Examples

```
irb(main):004:0> list = [2, 4, 6, 8]
=> [2, 4, 6, 8]
irb(main):005:0> second = list[1]
=> 4
irb(main):006:0> list[3] = 9
=> 9
irb(main):007:0> list
=> [2, 4, 6, 9]
irb(main):009:0> list[2.99999] # indices are
truncated
=> 6
irb(main):010:0> len = list.length
=> 4
irb(main):011:0>
```

for-in Statement

- The for-in statement is used to process elements of an array.
- The scalar variable takes on the values in the array one at a time.
- The scalar variable gets the <u>value</u>, <u>not</u> a reference to a value. Therefore, operations on the scalar variable do not affect the array.

for-in Statement – An Example

```
irb(main):001:0> sum = 0
=> 0
irb(main):002:0> list = [2, 4, 6, 8]
=> [2, 4, 6, 8]
irb(main):003:0> for value in list
irb(main):004:1> sum += value
irb(main):005:1> end
=> [2, 4, 6, 8]
irb(main):006:0> sum
=> 20
irb(main):007:0>
```

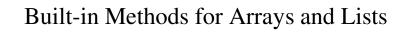
for-in Statement – Another Example

```
irb(main):001:0> list = [1, 3, 5, 7]
=> [1, 3, 5, 7]
irb(main):002:0> for value in list
irb(main):003:1> value += 2
irb(main):004:1> end
=> [1, 3, 5, 7]
irb(main):005:0> list
=> [1, 3, 5, 7]
irb(main):006:0>
```

for-in Statement - Another Example irb(main):001:0> list = [2, 4, 6] => [2, 4, 6] irb(main):002:0> for index in [0, 1, 2] irb(main):003:1> puts "For index = #{index}, the value is #{list[index]}" irb(main):004:1> end For index = 0, the value is 2 For index = 1, the value is 4 For index = 2, the value is 6 => [0, 1, 2] irb(main):005:0>

Built-in Methods for Arrays and Lists

- There are many built-in methods that are a part of Ruby. They include:
 - shift removes and returns the first element of the list
 - pop removes and return the last element of the list
 - **unshift** takes a scalar or an array literal and appends it to the beginning of the array.
 - **push** takes a scalar or an array literal and appends it to the end of the array.



- There are many built-in methods that are a part of Ruby. They include:
 - + catenates two arrays
 - **reverse** returns an array with the order of elements of the array reversed
 - include? returns true if the specific object is in the array.
 - sort sorts elements as long as Ruby has a way to compare them.

shift - An Example

irb(main):001:0> list = [3, 7, 13, 17]
=> [3, 7, 13, 17]
irb(main):002:0> first = list.shift
=> 3
irb(main):003:0> list
=> [7, 13, 17]
irb(main):004:0>

pop – An Example

```
irb(main):004:0> list = [2, 4, 6]
=> [2, 4, 6]
irb(main):005:0> last = list.pop
=> 6
irb(main):006:0> list
=> [2, 4]
irb(main):007:0>
```

unshift – An Example

- irb(main):009:0> list = [2, 4, 6]
- => [2, 4, 6]
- irb(main):010:0> list.unshift(8, 10)
- => [8, 10, 2, 4, 6]
- irb(main):011:0>

push - An Example . irb(main):007:0> list = [2, 4, 6] . => [2, 4, 6] . irb(main):008:0> list.push(8, 10) . => [2, 4, 6, 8, 10] . irb(main):009:0>

concat - An Example

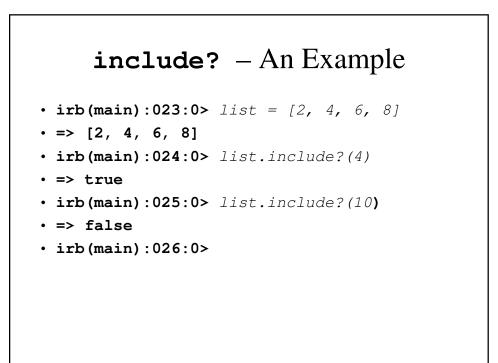
irb(main):011:0> list1 = [1, 3, 5, 7]
=> [1, 3, 5, 7]
irb(main):012:0> list2 = [2, 4, 6, 8]
=> [2, 4, 6, 8]
irb(main):013:0> list1.concat(list2)
=> [1, 3, 5, 7, 2, 4, 6, 8]
irb(main):014:0>

+ - An Example

irb(main):014:0> list1 = [1, 3, 5, 7]
=> [1, 3, 5, 7]
irb(main):015:0> list2 = [2, 4, 6, 8]
=> [2, 4, 6, 8]
irb(main):016:0> list3 = list1 + list2
=> [1, 3, 5, 7, 2, 4, 6, 8]
irb(main):017:0>

reverse – An Example

irb(main):018:0> list = [2, 4, 6, 8] => [2, 4, 6, 8] irb(main):019:0> list.reverse => [8, 6, 4, 2] irb(main):020:0> list => [2, 4, 6, 8] irb(main):021:0> list.reverse! => [8, 6, 4, 2] irb(main):022:0> list => [8, 6, 4, 2] irb(main):023:0>



sort – An Example

```
irb(main):028:0> list = [16, 8, 2, 4]
=> [16, 8, 2, 4]
irb(main):029:0> list.sort
=> [2, 4, 8, 16]
irb(main):030:0> list2 = ["jo", "fred", "mike",
"larry"]
=> ["jo", "fred", "mike", "larry"]
irb(main):031:0> list2.sort
=> ["fred", "jo", "larry", "mike"]
irb(main):032:0>
```