15.1 Origins and Uses of Ruby

- Designed by Yukihiro Matsumoto; released in 1996
- Use spread rapidly in Japan
- Use is now growing in part because of its use in Rails
- A pure object-oriented purely interpreted scripting language
- Related to JavaScript, but not closely

15.2 Scalar Types and Their Operations

- There are three categories of data types: scalars, arrays, and hashes
- Two categories of scalars, numerics and strings
- All numeric types are descendants of the Numeric class
- Integers: Fixnum (usually 32 bits) and Bignum
15.2 Scalar Types and Their Operations (continued)

- **Scalar Literals**

  - An integer literal that fits in a machine word is a **Fixnum**

    - Other integer literals are **Bignum** objects

  - Any numeric literal with an embedded decimal point or a following exponent is a **Float**

  - All string literals are **String** objects

  - Single-quoted literals cannot include characters that are specified with escape sequences

  - Double-quoted literals can include escape sequences and embedded variables *can be* interpolated
15.2 Scalar Types and Their Operations
(continued)

- **Variables**

  - A local variable is neither a class nor an instance variable

  - Names of local variables begin with lowercase letters and are case sensitive

  - Variables embedded in double-quoted literal strings are interpolated if they appear in braces and are preceded by a pound sign (#)

  - Expressions can also be embedded in braces in double-quoted literals

  - Variables do not have types—they are not declared

  - Assignment statement assign only object addresses

- **Numeric Operators**

  - Arithmetic operators are like those of other languages, except Ruby has no ++ or --

  - Division of integer objects is integer division
15.2 Scalar Types and Their Operations (continued)

- **Numeric Operators** (continued)

  - Any operation on `Bignum` objects that produces a result that fits in a `Fixnum` coerces it to a `Fixnum`

  - Any operation on `Fixnum` objects that produces a result that is too large coerces it to a `Bignum`

  - The `Math` module has methods for trig and transcendental functions
    e.g., `Math.sin(x)`

- **Interactive Ruby**

  - `irb` is an interactive interpreter for Ruby

  - Shorten the prompt with `conf.prompt_i = ">>"`

- **String Methods**

  - Some can be used as infix binary operators

  - Catenation is `+`, which can be infix

  - Append is `<<`, which can be infix
15.2 Scalar Types and Their Operations (continued)

- *String Methods* (continued)

- Remember how assignment works!

  ```python
  >> mystr = "Wow!"
  => "Wow!"
  >> yourstr = mystr
  => "Wow!"
  >> mystr = "What?"
  => "What?"
  >> yourstr
  => "Wow!"
  ```

- If you want to change the value of the location that is referenced by a variable, use `replace`

  ```python
  >> mystr.replace("Oh boy!")
  => "Oh boy!"
  ```

- More *String* methods:

  ```
  capitalize  strip
  chop        lstrip
  chomp       rstrip
  upcase      reverse
  downcase    swapcase
  ```
15.2 Scalar Types and Their Operations (continued)

- **String Methods** (continued)

  - The *bang or mutator* methods – add an `!`
  
  - Strings can be indexed – `str[3]`
  
  - Substrings can be referenced – `str[3, 5]`
  
  - Specific characters of a string can be assigned with `[]= – str[2, 3] = "gg"`
  
  - The `==` operator tests for equality of the objects
  
  - To test whether two variables reference the same object, use the `equal?` method
  
  - The `eql?` method tests whether the receiver object and the parameter have the same type and the same value
  
  - For ordering, use the `<=>` operator; it returns 0, 1, or -1
  
  - The `*` operator takes a string and a numeric expression—it replicates the string
15.3 Simple Input and Output

- **Screen Output**

  - `puts` takes a string literal as its operand
  - Implicitly attaches a newline
    - Use `print` if you don’t want the newline
  - `sprintf` is also available for conversions to string

- **Keyboard Input**

  - The `gets` method gets a line of input from the keyboard and returns it, including the newline (returns `nil` if it gets EOF)
    (UNIX: EOF is Ctrl-D; Windows: EOF is Ctrl-Z)

    ```ruby
    name = gets
    name = gets.chomp
    ```

  - If the input is a number, it must be converted with either `to_i` or `to_f`

    ```ruby
    age = gets.to_i
    ```

  - **SHOW** `quadeval.rb`

- **Running a Ruby script**

  ```ruby
  ruby [flags] filename
  ```
  - `c` flag is compile only; `w` flag is for warnings
15.4 Control Statements

- Control Expressions

- true, false, nil (false), variables, relational expressions, or compound expressions

  - If it is a variable, it is true if its value is anything except nil

- Relational operators: usual six (like C and Java) plus <=>, eql?, and equal?

- Two sets of Boolean operators:

  - Higher precedence: &&, ||, !

  - Lower precedence: and, or, not

- Operator precedence and associativity:
  - See Table 15.3

- Selection Statements

  if control_expression
      statement_sequence
  elsif control_expression
      statement_sequence
  ... 
  else
      statement_sequence
  end
15.4 Control Statements (continued)

- Selection Statements

  - unless is inverse of if

- Two kinds of multiple selection constructs

  ```
  case expression
  when value then
    statement_sequence
  when value then
    statement_sequence
  ...
  [else
    statement_sequence]
  end
  ```

- The values could be expressions, ranges (e.g., (1 .. 10)), class names, or regular expressions
- Semantics of the `case` statement:

1. There is an implicit `break` at the end of every selectable segment

2. The value of the expression is compared with the `when` values, top to bottom, until a match is found

3. A different operator, `===`, is used for the comparisons. If the value is a class name, it is a match if its class is the same as that of the expression or one of its superclasses; if the value is a regular expression, `===` is a simple pattern match

```ruby
case in_val
  when -1 then
    neg_count += 1
  when 0 then
    zero_count += 1
  when 1 then
    pos_count += 1
  else
    puts "Error - in_val is out of range"
end
```
15.4 Control Statements (continued)

- Second form of multiple selection statement, which is actually an expression

```plaintext
case
  when Boolean_expression then expression
  when Boolean_expression then expression
  ...
  when Boolean_expression then expression
else
  expression
end
```

- The value of the whole expression is the value of the expression associated with the first true Boolean expression

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

- Logical pretest loop

```plaintext
while control_expression
  loop body
end
```

- until has the same syntax, but opposite control
15.4 Control Statements (continued)

- **Unconditional Loop**

```
loop
  body
```

- The body optionally begins with `begin` and always ends with `end`

- The `break` statement

- Control goes to the first statement after the body

- The `next` statement

- Control goes to the first statement in the body

- Ruby does not have a C-style `for` statement

- Iterators are used instead - later
15.5 Fundamentals of Arrays

- Differences between Ruby arrays and those of other common languages:

  1. Length is dynamic
  2. An array can store different kinds of data

- Array Creation

  1. Send `new` to the `Array` class

     ```ruby
     list1 = Array.new(100)
     ```
     - All elements reference `nil`

     ```ruby
     list2 = Array.new(5, "Ho")
     ```
     - `list 2` is `"Ho", "Ho", "Ho", "Ho", "Ho"`

  2. Assign a list literal to a variable

     ```ruby
     list2 = [2, 4, 3.14159, "Fred", []]
     ```
     - All array subscripts are integers and the first subscript is always zero

- The length of an array is returned by `length`
- **The for-in Statement**

  ```ruby
  for value in list
    sum += value
  end
  ```

- The `value` variable takes on the values of the elements of list (not references to the values)

- *Irb’s response to a for construct is to return an array of the values taken on by the value variable*

- The list could be an array literal

- **Methods for Arrays and Lists**

  - **Adding and deleting the end elements**
    - `unshift`, `shift`, `push`, `pop`

  - **The `concat` method**
    - Takes one or more parameters, which are pushed onto the end of the array

  - **The `reverse` method**
    - Does what its name implies
15.5 Fundamentals of Arrays (continued)

- Methods for Arrays and Lists (continued)

- The `include?` predicate method
  - Searches the array for the given element

- The `sort` method
  - For arrays of a single element type
  - Works for any type elements for which it has a comparison operation
  - Returns a new array; does not affect the array object to which it is sent

→ SHOW `process_names.rb`
15.6 Hashes

- a.k.a. Associative Arrays

- Two fundamental differences between arrays and hashes:

  1. Arrays use numeric subscripts; hashes use string values

  2. Elements of arrays are ordered and are stored in contiguous memory; elements of hashes are not

- Hash Creation

  1. Send new to the Hash class

     ```ruby
     my_hash = Hash.new
     ```

  2. Assign a hash literal to a variable

     ```ruby
     ages = ("Mike" => 14, "Mary" => 12)
     ```

- Element references – through subscripting

  ```ruby
  ages["Mary"]
  ```

- Element are added by assignment

  ```ruby
  ages["Fred"] = 9
  ```
15.6 Hashes (continued)

- Element removal

```ruby
ages.delete("Mike")
```

- Hash deletion

```ruby
ages = () or
ages.clear
```

- Testing for the presence of a particular element

```ruby
ages.has_key?("Scooter")
```

- Extracting the keys or values

```ruby
ages.keys
ages.values
```
15.7 Methods

- All Ruby subprograms are methods, but they can be defined outside classes

  \[
  \text{def method\_name[ (formal\_parameters) ]}
  \]
  
  \[
  \text{statement\_sequence}
  \]
  
  \[
  \text{end}
  \]

- When a method is called from outside the class in which it is defined, it must be called through an object of that class

- When a method is called without an object reference, the default object is self

- When a method is defined outside any class, it is called without an object reference

- Method names must begin with lowercase letters

- The parentheses around the formal parameters are optional

- Neither the types of the formal parameters nor that of the return type is given

- If the caller uses the returned value of the method, the call is in the place of an operand in an expression
15.7 Methods (continued)

- If the caller does not use the returned value, the method is called with a standalone statement

- The `return` statement is often used to specify the return value

- If a `return` is not executed as the last statement of the method, the value returned is that of the last expression evaluated in the method

- **Local Variables**

  - Local variables in methods are either formal parameters or variables created in the method

  - A local variable hides a global variable with the same name

  - The names of local variables must begin with either a lowercase letter or an underscore

  - The lifetime of a local variable is from the time it is created until execution of the method is completed
15.7 Methods (continued)

- **Parameters**

- Formal parameters that correspond to scalar actual parameters are local references to new objects that are initialized to the values of the corresponding actual parameters.

- Actual parameters that are arrays or hashes are in effect passed by reference → `SHOW median` method.

- If a method has only normal parameters, the number of actual parameters must match the number of formal parameters.

- The normal scalar parameters in a method can be followed by an asterisk parameter, which is a reference to an array, which accepts any number of parameters, including none.

- Formal parameters can have default values.

- Keyword parameters can be implemented using hash literals as actual parameters.

- If the hash literal follows all normal scalar parameters and precedes any array or block parameters, the braces can be omitted.

```ruby
find(age, {'first' => 'Davy', 'last' => 'Jones'})
```
15.7 Methods (continued)

- *Parameters* (continued)

- Symbols are created by preceding an unquoted string with a colon

- A symbol made from a variable’s name can be thought of as being that variable’s name

- Does not refer to the variable’s value—it refers to its name

- A symbol is not related to any particular instance of the variable

- All symbols are instances of the `Symbol` class

- Symbols can be used to specify the keys in hash literals used as actual parameters

\[
\text{find}(\text{age}, :\text{first} => 'Davy', \\
:\text{last} => 'Jones')
\]
15.8 Classes

class class_name
...
end

- Class names must begin with uppercase letters

- The names of instance variables must begin with at signs (@)

- Each class implicitly has a constructor, new, which is called to create an instance
  - The new constructor calls the class initializer

- A class may have a single initializer, initialize

  - Initializes the instance variables

  - Parameters for initialize are passed to new

SHOW Stack2_class.rb

- Classes are dynamic – subsequent definitions can include new members; methods can be removed with remove_method in subsequent definitions

- Access Control

  - All instance data has private access by default, and it cannot be changed
15.8 Classes (continued)

- Access Control (continued)

- If needed, external access to instance variables is provided with getters and setters

```ruby
class My_class
  # Constructor
  def initialize
    @one = 1
    @two = 2
  end
  # A getter for @one
  def one
    @one
  end
  # A setter for @one
  def one=(my_one)
    @one = my_one
  end
end

- Note the "=" in the setter name

- When an instance variable is referenced inside the class, the @ is included in the name; when referenced outside the class, it is excluded

- Shortcuts for getters and setters

  attr_reader :one, :two
  attr_writer :one
```
15.8 Classes (continued)

- Access Control (continued)

- Method access control: public, private, and protected

- Public and protected are as in Java, etc.

- Private means the method cannot be called through an object reference

- There are two ways to specify access

  1. Call the access function without parameters – resets the default (which is public)

  2. Following all method definitions in a class, call the access function, passing the method names as symbols

     private :meth7, ...

- Access control of methods is dynamic

- Subsequent class definitions can specify different access levels

- Class variables are private to the class and its instances
15.8 Classes (continued)

- **Inheritance**

  ```ruby
  class My_Subclass < Base_class
  end
  ```

- **Modules**
  - A way to collect related methods into an encapsulation
  - Access to a module is with `include`
    ```ruby
    include Math
    ```
  - The methods in a module are mixed into those of the class that includes it – `mixins`
  - Provides the benefits of multiple inheritance
15.9 Blocks and Iterators

- A block is a segment of code, delimited by either braces or `do` and `end`

- By itself, a block does nothing

- Blocks can be sent to methods by attaching them to calls

  - This construct is used to build iterators

- **Built-in Iterators**

  `times` – blocks are sent to number objects to build simple counting loops

    ```ruby
    5.times {puts "hey!
    }```

- Blocks can have parameters; they appear at the beginning of the block in vertical bars (`|`)

  `each` – blocks are sent to array objects; the block is applied to each element of the array

    ```ruby
    list.each {|value| puts value}
    ```

- If `each` is used on a hash, two block parameters are required (key and value)
15.9 Code Blocks and Iterators  
(continued)

**upto** – blocks are sent to number objects, including a parameter for the last value

```ruby
5.upto(8) { |value| puts value }
```

**step** – blocks are sent to number objects, including two parameters for the last value and a step size

```ruby
0.step(6, 2) { |value| puts value }
```

**collect** – sent to an array, like each, but the block constructs and returns a new array

```ruby
list.collect { |value| value -= 5 }
```

- The mutator version of this is often useful – `collect!`

- Blocks can be passed as parameters

- Example – see Chapter 16
- **User-defined blocks**

- Blocks are called in methods with `yield`, which can include parameters to the block

```ruby
def get_name
  puts "Your name:"
  name = gets
  yield(name)
end

get_name {|name| puts "Hello, " + name}
```
15.10 Pattern Matching

- Patterns are placed between slashes

- Use the =~ operator between a string and a pattern

  name = "Freddie"
  name =~ /d/  

→ SHOW word_table.rb

- Remembering matches

  $1, $2, ... have the matches from the first parenthesized subpatterns

- Substitutions

  - Ruby has four methods of the String class for string substitutions

    str.sub(pattern, replacement)

    str.gsub(pattern, replacement)

    - There are also mutator versions

    - The i modifier can follow the pattern