Interactive Ruby

- Recall from the previous slides that we will be using Ruby as an interactive programming system
  - type an expression
  - Ruby evaluates the expression
  - Ruby prints the result

Ruby prompt

Arithmetic Expressions

- The simplest expressions in Ruby are arithmetic expressions
  - built up from numbers (operands) and operators
- The symbols used for operators are commonly used in other languages and applications (e.g. spreadsheets)
  - add: +
  - subtract: -
  - multiply: *
  - divide: /


Arithmetic Expressions (cont’d)

- Use parentheses if you want to alter the standard order
  

Note! Ruby truncates, it does not round to the nearest integer
Real Numbers

- Ruby does have real numbers, in addition to integers
- To distinguish the two, use a decimal point when writing a real number:
  
  ```ruby
  >> 5.0 * 3
  => 15.0
  >> 20.0 / 3
  => 6.66666666666667
  ```

- Most CIS 170 projects will use integers
  - if a project needs real numbers we will emphasize that fact

Floating Point Numbers

- Numbers inside a computer are approximations
  - real numbers like 1/3 have an infinite number of digits
  - Ruby and other programming languages limit numbers to a finite number of digits
- From http://www.joyofpi.com/pi.html:
  
  \[
  \pi = 3.141592653589793238462643383279... \]

  - In Ruby:
    
    ```ruby
    >> include Math
    >> PI
    => 3.14159265358979
    ```

  - In computer science finite approximations of real numbers are known as
    floating point numbers (or “floats”)

Variables

- To motivate the next topic, suppose you want to compute the surface area of a counter top
  - you know the counter is a square with a triangular piece missing
  - you also know each side of the triangle is half as long as the side of the square

- Using some simple geometry will give you an equation for the area of the counter

  \[
  a = x^2 - ((x/2)^2/2)
  \]

Variables (cont’d)

- Suppose you measure one edge of the square to get \( x = 109 \)cm
- Ruby can compute the area for you:
  
  ```ruby
  >> 109*109 - ((109/2)*(109/2) / 2)
  => 10423
  ```

  - You can simplify this a little bit by using the exponentiation operator
    - to compute \( x^2 \) type \( x**2 \)
      
      ```ruby
      >> 109**2 - ((109/2)**2 / 2)
      => 10423
      ```

      \[
      a = x^2 - ((x/2)^2/2)
      \]
Variables in Ruby

- You can make the equation even simpler by using a variable
  - create a value named x, and then write the equation using x
    ```ruby
    >> x = 109
    => 109
    >> x**2 - ((x/2)**2 / 2)
    => 10423
    ```
- The first expression defines a variable named x, and the second uses the value of x to compute the area
- An expression of the form “x = ...” is known as an assignment statement
  - put the name of a variable on the left side of the assignment operator
  - put any expression on the right side
  - Ruby evaluates the expression and saves the result

Variables Can Be Updated

- You can change the value of a variable
  - simply assign it a new value in another assignment statement:
    ```ruby
    >> x = 109
    => 109
    >> x**2 - ((x/2)**2 / 2)
    => 10423
    >> x = 111
    => 111
    >> x**2 - ((x/2)**2 / 2)
    => 10809
    ```

Variable Names

- There are a few rules for defining names for variables:
  - names can be as long as you want
    ```ruby
    >> supercalifragilisticexpialadocious = 3.14
    => 3.14
    >> supercalifragilisticexpialadocious * supercalifragilisticexpialadocious
    => 9.8596
    ```
  - case is important (a is not the same as A)
  - names must start with a letter, but can have digits or underscores
  - note: variable names in Ruby start with lower case letters

Strings

- Numbers aren’t the only type of data found in programs
- We’ve already seen that Ruby can print a sentence:
  ```ruby
  >> p "hello world"
  "hello world"
  ```
- A set of letters enclosed in quotes is known as a string
- We can store strings in variables:
  ```ruby
  >> s = "hello"
  >> s
  "hello"
  ```
String Operators

- There are lots of things we can do with strings
- Example: when the operands are strings, the + operator attaches one string to the other:
  ```ruby
  >> s
  => "hello"
  >> s + ', world"
  => "hello, world"
  ```
- We’ll come back to strings later in the term, when we need them for a project
- For now, just know that Ruby has several different types of data; the ones we’ve seen so far are:
  - integers
  - reals (“floats”)
  - strings

Objects

- The slides that introduced Ruby said it belongs to a family of languages known as **object-oriented languages**
- The term **object** simply means “generic piece of data”
- Integers, floating point numbers, and strings are some of the types of objects that Ruby can deal with
- Some other types of objects include:
  - lists of numbers
  - dates
  - objects used to build graphical interfaces, e.g. windows and menus
- Ruby has several dozen pre-defined types of objects
- It also allows programmers to define new types

Variables Are Labels for Objects

- Think of the memory of your computer (RAM) as being one big “object storage unit”
- When Ruby evaluates an assignment statement, it creates a new variable if it does not already exist:
  ```ruby
  >> x = 109
  => 109
  >> y = x**2 - ((x/2)**2 / 2)
  => 10423
  ```
- Variables can refer to any type of object
  - we’ll see examples of strings and other more complex objects later

Object Store

- In pictures like the one at right objects are drawn as small “clouds”
  - the clouds mean the objects are abstract ideas
  - as programmers we don’t (usually) care how the computer represents objects in memory
  - all we care about is that fact that we have created a variable and assigned it a value
  - we let the computer figure out how to store the value internally
Object Store (cont’d)

- When we change the value of a variable, Ruby throws away the old object (the previous value) and replaces it with a new one
  
  ```ruby
  >> s
  => "hello"
  >> s = 3 * 9
  => 27
  ```

- Since variable names are just labels, Ruby doesn’t care if the new value is the same type of object

Review

- Ruby is an interactive language
  - IRB has a read-eval-print loop
  - type an expression, Ruby evaluates it and prints the result

- Expressions are made up of
  - constants (e.g., 26 or 3.14159)
  - operators (e.g., + * - / **) for expressions involving numbers
  - variables (labels on objects)

- When you ask Ruby to evaluate an expression containing a variable, the current value of the variable is plugged in to the expression
  - Ruby fetches the value from the “object store”
  - A variable’s value can be changed

Functions

- In math, a function is a way to relate sets of values
  - example: the sin function in trigonometry

- Ruby has an extensive library of math functions

- We need to tell Ruby we want to use the library:
  
  ```ruby
  >> include Math
  >> sin(3.0)
  => 0.141120008059867
  ```

Methods

- In Ruby (and other object-oriented languages) functions are known as methods

- Here is some important terminology:
  - when a method is used in an expression we call the method
  - a method call can include parameter values
  - we say the parameters are passed into the method
  - methods return values that can be used in the original expression

- Example: when the sin method is passed the value 1 it returns 0.84...
  
  ```ruby
  >> sin(1.0)
  => 0.841470984807897
  ```
Defining New Methods

- One of the most powerful features of any programming language is the ability to define new methods.
- Example: suppose we need to calculate the size of several countertops of the same shape, different edge length.
- One approach: make a spreadsheet to enter sizes in one column, a formula for the area in a second column.

```
<table>
<thead>
<tr>
<th>x</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>7088</td>
</tr>
<tr>
<td>95</td>
<td>7897</td>
</tr>
<tr>
<td>100</td>
<td>8750</td>
</tr>
<tr>
<td>105</td>
<td>9647</td>
</tr>
<tr>
<td>110</td>
<td>10588</td>
</tr>
<tr>
<td>115</td>
<td>11572</td>
</tr>
</tbody>
</table>
```

Defining New Methods (cont’d)

- The equivalent idea in Ruby is to write a new method.
- After we define the method:

  ```ruby
  >> countertop(109)
  => 10423
  ```

```
def countertop(x)
  x**2 - ((x/2)**2)/2
end
```

- Ruby saves the definition of the method.
- We can call it whenever we want to compute the area of a countertop.
- When the method is called, Ruby creates the variable `x` and assigns it the value passed as a parameter.
- Ruby evaluates the expression and returns the result of the expression as the value of the method.
Abstraction

- One of the most important concepts in computer science is the idea of abstraction.
  - When we define a method like `countertop` we're making a small "package".
  - Inside the package are all the details of how to compute the area of the counter.
  - From outside we forget about the details -- all we care about is the fact that we can call this method and it will do a computation.

```
def countertop(x):
    return x**2 - ((x/2)**2)/2
```

```
>>> countertop(109)
10423
```

Keywords

- The words `def` and `end` are keywords.
  - Keywords are words that have special meaning in a programming language.
  - They cannot be used as the names of variables or methods.
  ```ruby
  end = 14
  SyntaxError: compile error
  ```
- Ruby has several other keywords -- we'll introduce them as they are needed.

Entering Definitions in IRB

- For short definitions we can just type the entire definition on one line.
  ```ruby
  def f(x) x**2 - x end
  >>> f(10)
  90
  >>> f(5)
  20
  ```
- For more complicated definitions it is more convenient to put the definition in a file, and then load the file into IRB.
  ```ruby
  load "countertop.rb"
  ```
- The string following the word `load` is the name of the file that contains the definition.
  ```ruby
  convention: text files containing Ruby code should have names ending .rb
  ```

Using a Text Editor

- You can use any application that operates on text files.
  - If you use a word processor make sure the files are saved as "plain text".
  - These applications often create `.doc` (MS Word format) or `.rtf` (rich text) files.
  - These formats include extra information such as font type, text size, etc.
- Programmers use applications known as text editors.
  - There are lots of them out there.
  - Many good ones are free.
  - BBEdit (OS/X), emacs (Linux and OS/X), ...
  - The `fxri` and `FreeRide` applications on Windows include a text editor.
Workflow

- For labs and tutorial projects in CIS 170 a typical session with Ruby will involve the following steps:
  - Download a program from the class website
  - Load the program into IRB
  - Call methods defined in the program (following instructions supplied with the lab)
  - Use your text editor to modify the program or enter some new data
  - Re-load the program after making the changes
  - Run the program again
  - Cut and paste results from your terminal window into the document you will submit as part of your lab report

Tutorial Project: Temperature Converter

- To get some experience with the “Ruby Workbench” do the tutorial project in Ch 2 of Science of Computing
- Project: write a method that converts temperature values from Fahrenheit to Celsius
  - When you have completed the tutorial, you will have a method definition in a file named temps.rb
  ```ruby
  load "temps.rb"
  true
  celsius(72)
  22
  ```

Temperature Converter (cont’d)

- Start by learning how Ruby evaluates arithmetic expressions

1. Type a simple expression, e.g., 5 * 6 and hit the return key. Did you see the result you expected?
2. Try some simple expressions involving other operators:
   - 6 - 3
   - 3 * 7
   - 8 / 4
3. Try some expressions with and without parentheses:
   - 3 + 4 * 5
   - (3 + 4) * 5
   - 8 - 4 / 2
   - (8 - 4) / 2

Temperature Converter (cont’d)

- Next try some expressions where operands are floating point numbers

The equation you learned in school for converting Fahrenheit to Celsius is often written as $C = (F - 32) \times \frac{5}{9}$. We can use Ruby to help us convert 80°F to Celsius:

```ruby
(80 - 32) * 5 / 9
=> 26
```

10. Use IRB to evaluate a simple expression such as 3 * 5 where both operands are integers.
11. Repeat the previous expression, but replace the integer operands with floating point numbers. Can you see the difference between the outputs for these two expressions?
12. Use Ruby to convert the following temperature values to Celsius: 100°F, 90°F, 70°F, 212°F, 32°F.
13. The formula for converting from Celsius to Fahrenheit is $F = C \times \frac{9}{5} + 32$. Use this formula to convert the following temperatures to Fahrenheit: 0°C, 10°C, 20°C, 30°C, 100°C.
Temperature Converter (cont’d)

- Next try some expressions that include variable names

19. Start IRB, and type two expressions that define variables x and y:
   ```ruby
   >> x = 6
   => 6
   >> y = 5
   => 5
   ```

20. Try out a few expressions using these variables:
    ```ruby
    >> x + 3
    >> x * y
    >> (x + 3) * y
    Did you get what you expected?
    ```

Temperature Converter (cont’d)

- Enter the method definition in IRB, to see how `def` works

28. Type the following three lines into IRB to create a new method named `celsius`:
    ```ruby
    >> def celsius(f)
    return (f - 32) * 5 / 9
    end
    ```
    You can either type these as three separate lines, as shown above, hitting the return key at the end of each line, or you can type them all together on one line.

29. Try calling your new method:
    ```ruby
    >> celsius(50)
    => 10
    >> celsius(90)
    => 35
    ```

Extra Credit Challenge

- Start a Ruby session and load `countertop.rb` (I will put the file on the web, or you can make your own)
  ```ruby
  >> load "countertop.rb"
  ```

- Call the method a couple times:
  ```ruby
  >> countertop(109)
  => 10423
  >> countertop(115)
  => 11601
  ```

- Can you explain why the value printed by Ruby is not the same as the value in the spreadsheet?