Just enough Ruby

pluralsight
hardcore developer training

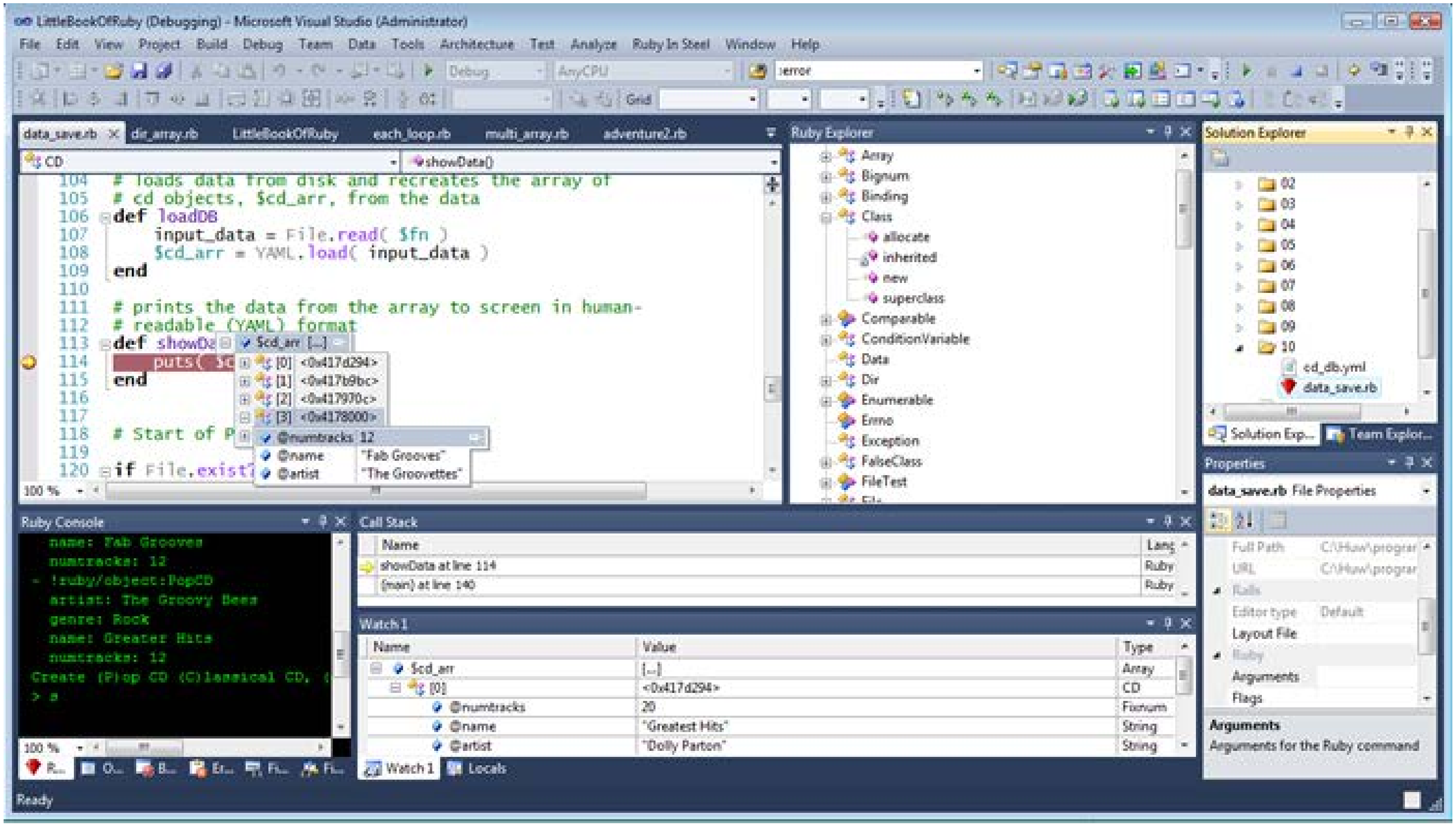
Interactive Ruby

- **“IRB” is a Ruby REPL**
  - Read-Eval-Print loop
- **Great for trying Ruby snippets**
- **Can be started from the Start Menu or from the command line**

IDE options

- **Emacs, Vim**
- **Notepad works too**
- **Ruby in Steel from SapphireSteel**
  - www.sapphiresteel.com
  - Integrates with Visual Studio
- **RubyMine from JetBrains**
  - www.jetbrains.com
  - Standalone IDE

Ruby In Steel



LittleBookOfRuby (Debugging) - Microsoft Visual Studio (Administrator)

```
# loads data from disk and recreates the array of
# cd objects, $cd_arr, from the data
def loadDB
    input_data = File.read( $fn )
    $cd_arr = YAML.load( input_data )
end

# prints the data from the array to screen in human-
# readable (YAML) format
def showDa
    puts( $c
end


# Start of P

if File.exist?
```

```
name: Fab Grooves
numtracks: 12
- !ruby/object:PopCD
artist: The Groovy Bees
genre: Rock
name: Greater Hits
numtracks: 12
Create (P)op CD (C)lassical CD, (
> s
```

Name	Lang
showData at line 114	Ruby
(main) at line 140	Ruby

Name	Value	Type
$cd_arr	[...]	Array
[0]	<0x417d294>	CD
@numtracks	20	Fixnum
@name	"Greatest Hits"	String
@artist	"Dolly Parton"	String

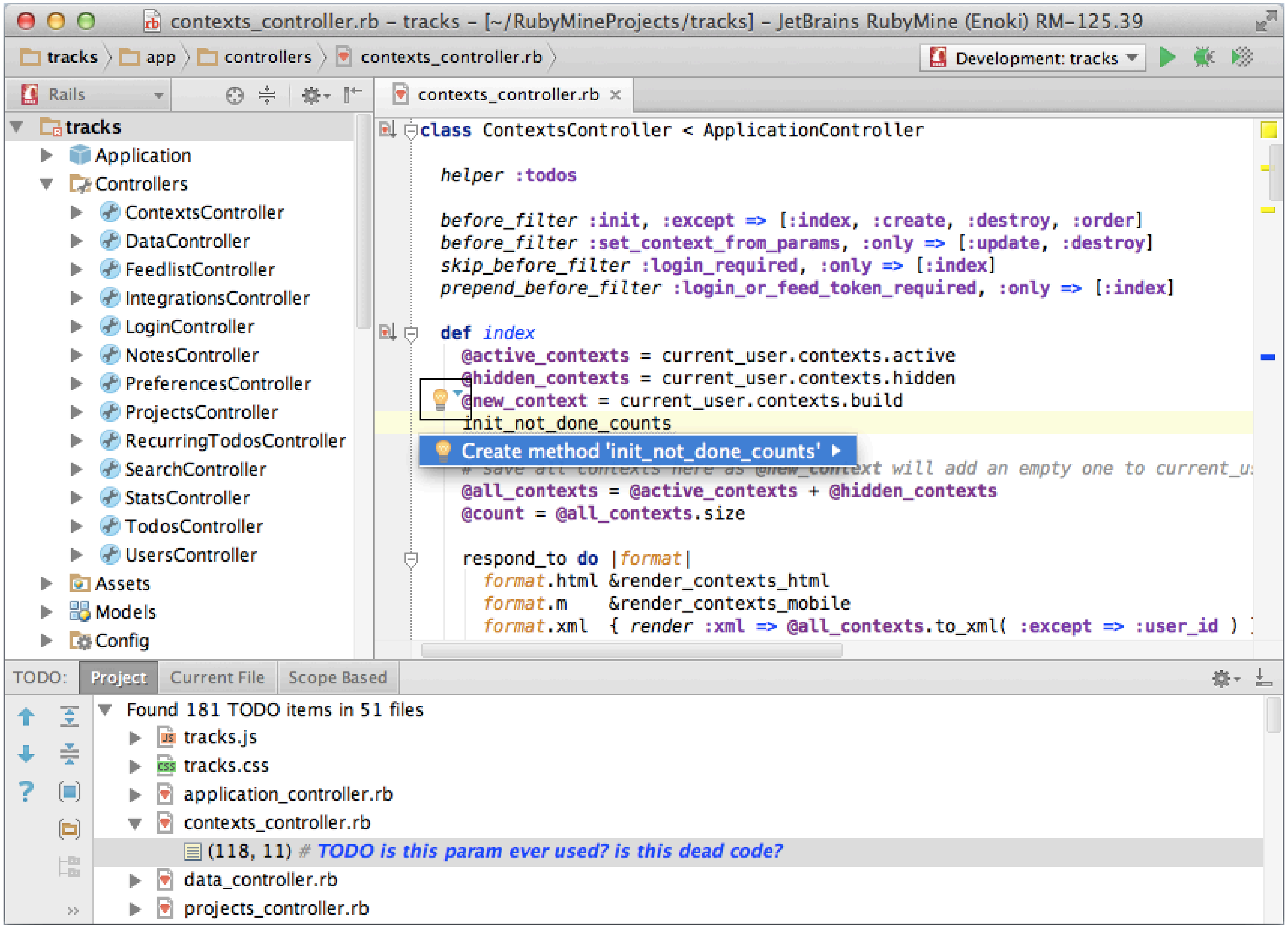
$cd_arr	[...]
[0]	<0x417d294>
[1]	<0x417b9bc>
[2]	<0x417970c>
[3]	<0x4178000>
@numtracks	12
@name	"Fab Grooves"
@artist	"The Groovettes"

data_save.rb File Properties

Full Path	C:\Huw\progra
URL	C:\Huw\progra
Editor type	Default
Layout File	
Arguments	
Flags	

Arguments
Arguments for the Ruby command

Rubymine



contexts_controller.rb – tracks – [~/RubyMineProjects/tracks] – JetBrains RubyMine (Enoki) RM-125.39

tracks 〉 app 〉 controllers 〉 contexts_controller.rb 〉   Development: tracks

Rails

- tracks
  - Application
  - Controllers
    - ContextsController
    - DataController
    - FeedlistController
    - IntegrationsController
    - LoginController
    - NotesController
    - PreferencesController
    - ProjectsController
    - RecurringTodosController
    - SearchController
    - StatsController
    - TodosController
    - UsersController
  - Assets
  - Models
  - Config

contexts_controller.rb ×

```
class ContextsController < ApplicationController

  helper :todos

  before_filter :init, :except => [:index, :create, :destroy, :order]
  before_filter :set_context_from_params, :only => [:update, :destroy]
  skip_before_filter :login_required, :only => [:index]
  prepend_before_filter :login_or_feed_token_required, :only => [:index]

  def index
    @active_contexts = current_user.contexts.active
    @hidden_contexts = current_user.contexts.hidden
    @new_context = current_user.contexts.build
    init_not_done_counts
    # save all contexts here as @new_context will add an empty one to current_u
    @all_contexts = @active_contexts + @hidden_contexts
    @count = @all_contexts.size

    respond_to do |format|
      format.html &render_contexts_html
      format.m    &render_contexts_mobile
      format.xml  { render :xml => @all_contexts.to_xml( :except => :user_id ) }
```

TODO: Project | Current File | Scope Based

- Found 181 TODO items in 51 files
  - tracks.js
  - tracks.css
  - application_controller.rb
  - contexts_controller.rb
    - (118, 11) # TODO is this param ever used? is this dead code?
  - data_controller.rb
  - projects_controller.rb

Identifiers

- **Names for**
  - Variables
  - Classes
  - Methods
- **Person != person**
- **Camel case**
  - thisIsAMethod
- **Pascal case**
  - ThisIsAMethod
- **Snake case**
  - This_Is_A_Method

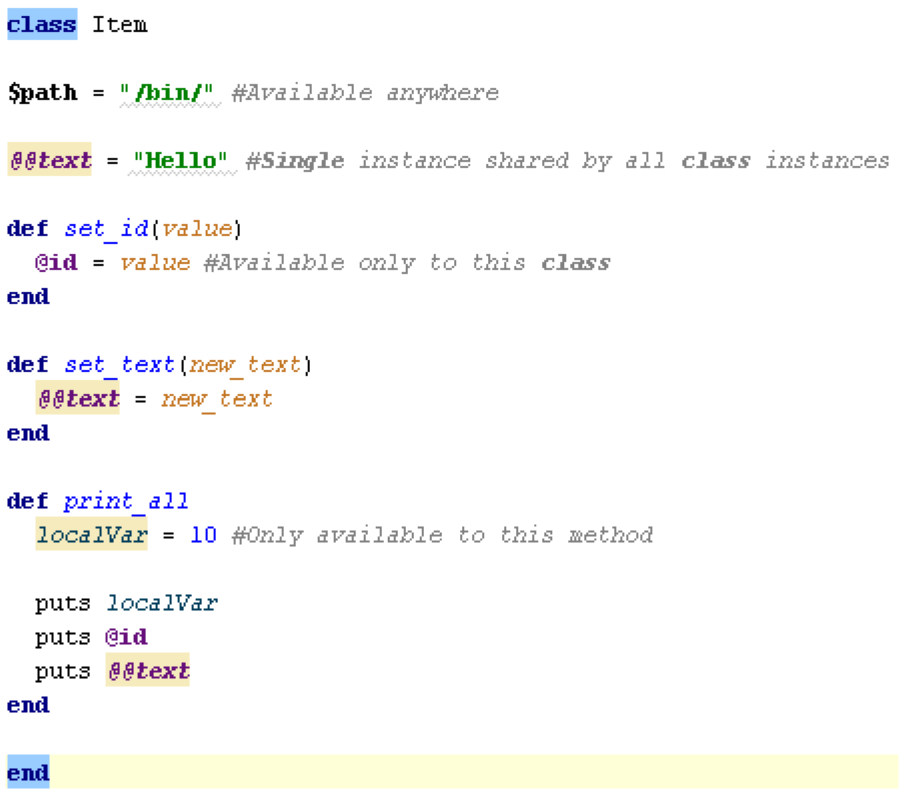
Scope

```
public class Class1
{
    int x = 10; //Available to all class members

    public Class1()
    {
        int y = 20; //Available only to this method
    }
}
```

```
class Item

$path = "/bin/" #Available anywhere

@@text = "Hello" #Single instance shared by all class instances

def set_id(value)
  @id = value #Available only to this class
end

def set_text(new_text)
  @@text = new_text
end

def print_all
  localVar = 10 #Only available to this method

  puts localVar
  puts @id
  puts @@text
end

end
```

Scope

- Locals only available in the scope they were defined
- Instance variables are only available to a specific instance of the class
- Instance variables are declareded inside of methods, not at the class level
- Class level variables are a single instance, available to all class instances
- Instance and class level variables are private. Not avilable outside of the class
- Globals are available from anywhere



```
class Item

$path = "/bin/" #Available anywhere

@@text = "Hello" #Single instance shared by all class instances

def set_id(value)
  @id = value #Available only to this class
end

def set_text(new_text)
  @@text = new_text
end

def print_all
  localVar = 10 #Only available to this method

  puts localVar
  puts @id
  puts @@text
end

end
```

Strings & substitution

- **“Text” and ‘Text’ are both valid**
- **“Text\n” != ‘Text\n’**
- **String interpolation**
  - “Text #{expression} more text”
  - Console.Write(“{0}, {1}”, varX, varY)

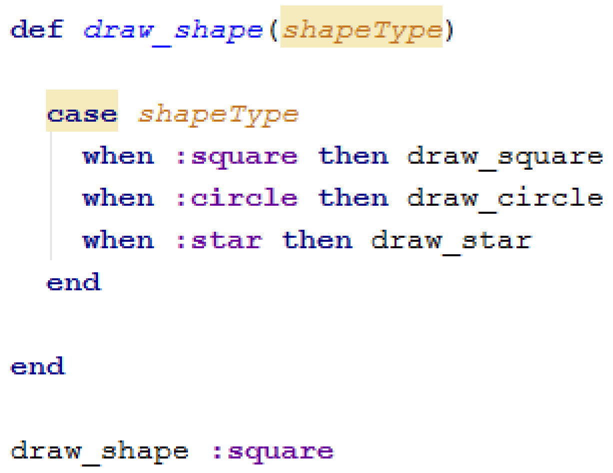
Symbols

```
const int SQUARE = 1;
const int CIRCLE = 2;
const int STAR = 3;

public void Draw()
{
    DrawShape(CIRCLE);
    DrawShape(2);
}

private void DrawShape(int shapeType)
{
    switch (shapeType)
    {
        case SQUARE:
            {
                break;
            }
        case CIRCLE:
            {
                break;
            }
        case STAR:
            {
                break;
            }
        default:
            {
                throw new Exception();
            }
    }
}
```

```
enum ShapeType
{
    Square,
    Circle,
    Star
}

public void Draw()
{
    DrawShape(ShapeType.Circle);
}

private void DrawShape(ShapeType shapeType)
{
    switch (shapeType)
    {
        case ShapeType.Square:
            {
                break;
            }
        case ShapeType.Circle:
            {
                break;
            }
        case ShapeType.Star:
            {
                break;
            }
    }
}
```

Symbols

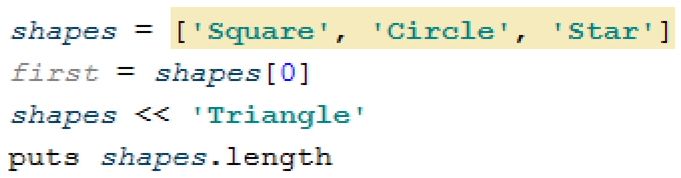
- Globally unique
- Named representation of a memory location
- Symbols are denoted with a colon
  - :symbol_name
- Symbols are unique instances
- Use symbols in place of string identifiers
  - Collection[:key] vs Collection["key"]



```
def draw_shape(shapeType)

  case shapeType
    when :square then draw_square
    when :circle then draw_circle
    when :star then draw_star
  end

end

draw_shape :square
```

Arrays

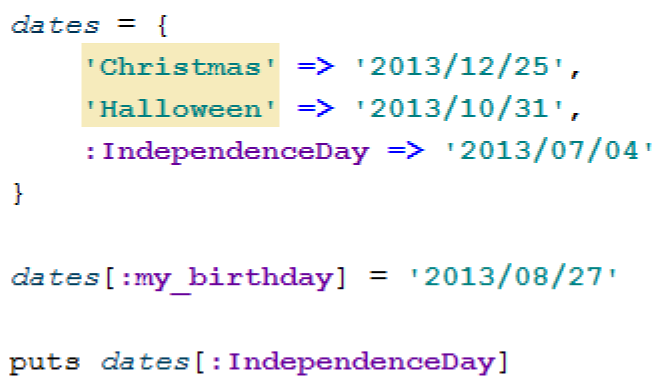
- **Arrays are dynamic**
- **Will grow automatically**
  - No need to manually resize them
- **Arrays can contain any combination of types**
  - Same as `object[]` in C#



```
shapes = ['Square', 'Circle', 'Star']
first = shapes[0]
shapes << 'Triangle'
puts shapes.length
```

Hashes

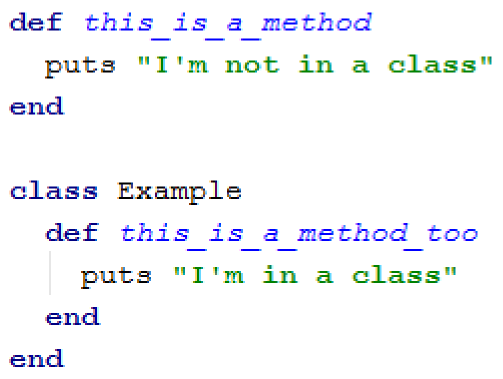
- **Similar to .NET Dictionaries**
- **Collection[“MyKey”]**
  - Collection[4567] also works
  - Collection[:key] does too



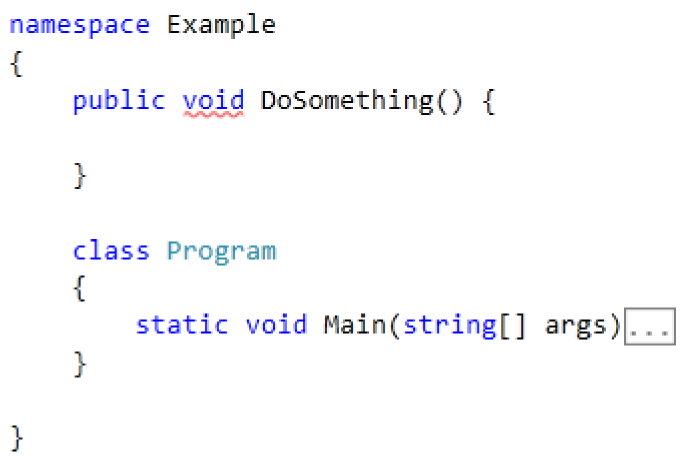
```
dates = {
    'Christmas' => '2013/12/25',
    'Halloween' => '2013/10/31',
    :IndependenceDay => '2013/07/04'
}

dates[:my_birthday] = '2013/08/27'

puts dates[:IndependenceDay]
```

Methods

- **Methods are named blocks of code**
- **Methods can be defined anywhere**
- **No void methods, must return a value**
    - Result of the last line of method is the return value



```ruby
def this_is_a_method
  puts "I'm not in a class"
end

class Example
  def this_is_a_method_too
    puts "I'm in a class"
  end
end
```



```csharp
namespace Example
{
    public void DoSomething() {

    }

    class Program
    {
        static void Main(string[] args)...
    }

}
```

Modules

- **Ruby only supports single inheritance**
- **Modules are similar to classes, except…**
  - Cannot be instantiated
  - Cannot inherit or be derived from
  - Can contain classes, methods, attributes and other modules
- **Modules must be included before any members can be accessed**

```ruby
module MyModule
  def say_hello
    puts "Hello!"
  end
end

class Example
  include MyModule
end
```

Summary

- IRB is a Ruby REPL (Read-Eval-Print-Loop)
- Everything in Ruby is an object, even true, false and nil
- Identifiers in Ruby are case sensitive
- Classes and constants must start with capital letter
- Local variables and methods must start with a lower case letter
- Scope is defined using modifiers, @, @@, $
- Double quoted string will interpret escape sequences
- Symbols are similar to enums in .NET
- Use symbols when the underlying value is unimportant
- As identifiers, symbols are better for performance than strings

Summary

- **Arrays in Ruby are like** `object[]` **in .NET**
- **Arrays are dynamic, automatically growing as we need them too**
- **Hashes are like** `Dictionary<object, object>` **in .NET**
- **Methods can be defined anywhere**
- **Every method will return a value, implicitly or explicitly**
- **Modifiers can be applied to methods to change their meaning**
  - ? – The method will answer a question posed by the invocation
  - ! – Method should be used with caution; makes an in-place change
  - = - Method becomes a setter

Summary

- **Classes don't have constructors, they have an initialize method**
  - Gets called automatically when using `new` to create an instance
- **Instance members are not available outside of the class**
- **Quickly define getters and setters using**
  - `attr_accessor`
  - `att_reader`
  - `att_writer`
- **Ruby only supports single inheritance, but can "mix-in" functionality**
- **Modules can be used like namespaces**

Summary

- Loops and conditional statements can be used as modifiers
  - `puts “Hello” if x == 10`
  - `do_work until x == 10`
- Handle errors using begin/rescue
- Throw and catch mean different things in Ruby