```ruby
# p001hello.rb
=begin
  Ruby is an interpreted language
  Source file has .rb extension
  Free format and case sensitive
  No statement delimiters
  Two types of comments
  String literals in single or double quotes
=end
puts 'Hello'

# p002rubynumbers.rb
=begin
 Ruby Numbers
 Usual operators:
 + addition
 - subtraction
 * multiplication
 / division
=end

puts 1 + 2
puts 2 * 3
# Integer division
# When you do arithmetic with integers, you'll get integer answers
puts 3 / 2
puts 10 - 11
puts 1.5 / 2.6
# class hierarchy
# http://www.cs.mun.ca/~donald/slug/2003-10-16/presentation/img5.html

# p003rubystrings.rb
=begin
  Ruby Strings
  In Ruby, strings are mutable
=end

puts "Hello World"
# Can use " or ' for Strings, but ' is more efficient
puts 'Hello World'
# String concatenation
puts 'I like' + ' Ruby'
# Escape sequence
```

```ruby
puts 'It\'s my Ruby'
# New here, displays the string three times
puts 'Hello' * 3
# Defining a constant
# More on Constants later, here
# http://rubylearning.com/satishtalim/ruby_names.html
PI = 3.1416
puts PI

# p004stringusage.rb
# Defining a constant
PI = 3.1416
puts PI
# Defining a local variable
my_string = 'I love my city, Pune'
puts my_string
=begin
  Conversions
  .to_i, .to_f, .to_s
=end
var1 = 5;
var2 = '2'
puts var1 + var2.to_i
# << appending to a string
a = 'hello '
a<<'world.
I love this world...'
puts a

=begin
  << marks the start of the string literal and
  is followed by a delimiter of your choice.
  The string literal then starts from the next
  new line and finishes when the delimiter is
  repeated again on a line on its own.
=end
a = <<END_STR
This is the string
And a second line
END_STR
puts a

# p005methods.rb
# gets and chomp
puts "In which city do you stay?"
```

```ruby
# STDOUT - global constant - the actual standard output stream for the program
# flush - flushes any buffered data within io to the underlying operating system
STDOUT.flush
# gets - returns a string and a '\n' character
# chomp - removes this '\n'
city = gets.chomp
puts "The city is " + city

# to know which object you are in
puts self

# p006ftoc.rb
puts 'Enter temperature in Fahrenheit: '
STDOUT.flush
temp_in_fahrenheit = gets.chomp
temp_in_celsius = (((temp_in_fahrenheit.to_f - 32.0) / 9.0) * 5.0)
puts 'Temperature ' + temp_in_fahrenheit + ' degree Fahrenheit = ' + format("%.2f",
temp_in_celsius) + ' degree Celsius'

# p007dt.rb
=begin
  The first character of a name helps Ruby to distinguish its intended use
  instance variable name starts with a @ sign
  class variable name starts with a @@ sign
  global variable name starts with a $ sign
  constant name starts with an uppercase letter
  Method names should begin with a lowercase letter
  ?, ! and = are the only weird characters allowed as method name suffixes
  ! or bang labels a method as dangerous-specifically
  use underscores to separate words in a multiword method or variable name
  Class names, module names and constants use capitalization
=end

# Ruby is dynamic
x = 7           # integer
x = "house"  # string
x = 7.5        # real

# In Ruby, everything you manipulate is an object
'I love Ruby'.length

# p008mymethods.rb
# A method returns the value of the last statement
# Methods that act as queries are often named with a trailing ?
# Methods that are "dangerous," or modify the receiver, might be named with a
```

```
trailing ! (Bang methods)
# A simple method
def hello
  'Hello'
end
#use the method
puts hello

# Method with an argument - 1
def hello1(name)
  'Hello ' + name
end
puts(hello1('satish'))

# Method with an argument - 2
def hello2 name2
   'Hello ' + name2
end
puts(hello2 'talim')

# p009mymethods1.rb
# interpolation refers to the process of inserting the result of an
# expression into a string literal
# the interpolation operator #{...} gets calculated separately
def mtd(arg1="Dibya", arg2="Shashank", arg3="Shashank")
  "#{arg1}, #{arg2}, #{arg3}."
end
puts mtd
puts mtd("ruby")

# p010aliasmtd.rb
# alias new_name old_name
# When a method is aliased, the new name refers
# to a copy of the original method's body

def oldmtd
  "old method"
end
alias newmtd oldmtd
def oldmtd
  "old improved method"
end
puts oldmtd
puts newmtd
```

```ruby
# p011vararg.rb
# variable number of parameters example
# The asterisk is actually taking all arguments you send to the method
# and assigning them to an array named my_string as shown below
def foo(*my_string)
  my_string.inspect
end
puts foo('hello','world')
puts foo()

# p012mtdstack.rb
# Sequence in which the parameters are put on to the stack is left to right
def mtd(a=99, b=a+1)
  [a,b]
end
puts mtd

# p012zmm.rb
class Dummy
  def method_missing(m, *args)
    puts "There's no method called #{m} here -- please try again."
  end
end
Dummy.new.anything

# p013expint.rb
def say_goodnight(name)
  result = "Good night, #{name}"
  return result
end
puts say_goodnight('Satish')

# modified program
def say_goodnight2(name)
  "Good night, #{name}"
end
puts say_goodnight2('Talim')

# p013strcmp.rb
# String#eql?, tests two strings for identical content. It returns the same result
as ==
# String#equal?, tests whether two strings are the same object
s1 = 'Jonathan'
s2 = 'Jonathan'
s3 = s1
```

```ruby
if s1 == s2
  puts 'Both Strings have identical content'
else
  puts 'Both Strings do not have identical content'
end
if s1.eql?(s2)
  puts 'Both Strings have identical content'
else
  puts 'Both Strings do not have identical content'
end
if s1.equal?(s2)
  puts 'Two Strings are identical objects'
else
  puts 'Two Strings are not identical objects'
end
if s1.equal?(s3)
  puts 'Two Strings are identical objects'
else
  puts 'Two Strings are not identical objects'
end

# p014constructs.rb
# In Ruby, nil and false evaluate to false, everything else (including true, 0)
means true
# nil is an actual object
# if else end
var = 5
if var > 4
  puts "Variable is greater than 4"
  puts "I can have multiple statements here"
  if var == 5
    puts "Nested if else possible"
  else
    puts "Too cool"
  end
else
  puts "Variable is not greater than 5"
  puts "I can have multiple statements here"
end

# Loops
var = 0
while var < 10
  puts var.to_s
  var += 1
```

```ruby
end

# p015elsifex.rb
# elseif example

# Original example
puts "Hello, what's your name?"
STDOUT.flush
name = gets.chomp
puts 'Hello, ' + name + '.'

if name == 'Satish'
  puts 'What a nice name!!'
else
  if name == 'Sunil'
    puts 'Another nice name!'
  end
end

# Modified example with elseif
puts "Hello, what's your name?"
STDOUT.flush
name = gets.chomp
puts 'Hello, ' + name + '.'

if name == 'Satish'
  puts 'What a nice name!!'
elsif name == 'Sunil'
    puts 'Another nice name!'
end

# Further modified
puts "Hello, what's your name?"
STDOUT.flush
name = gets.chomp
puts 'Hello, ' + name + '.'

# || is the logical or operator
if name == 'Satish' || name == 'Sunil'
  puts 'What a nice name!!'
end

# p016leapyear.rb
=begin
Program to determine if a year is a leap year.
```

```
To determine if a year is a leap year, follow these steps:
1.   If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.
2.   If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.
3.   If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.
4.   The year is a leap year (it has 366 days).
5.   The year is not a leap year (it has 365 days).

The above logic is combined into a single if check below
=end

# Get the input and determine if it is a leap year
puts "Enter the year: "
STDOUT.flush
input_year = gets.chomp.to_i
if ((input_year % 4 == 0) && (input_year % 100 > 0)) || (input_year % 400 == 0)
  puts "Year #{input_year} is a leap year"
else
  puts "Year #{input_year} is not a leap year"
end

# p017leapyearmtd.rb
def leap_year?(input_year)
  ((input_year % 4 == 0)  && (input_year % 100 > 0)) || (input_year % 400 == 0)
end

# Get the input and determine if it is a leap year
puts "Enter the year: "
STDOUT.flush
input_year = gets.chomp.to_i
if leap_year?(input_year)
  puts "Year #{input_year} is a leap year and has #{366*60*24} minutes in the year"
else
  puts "Year #{input_year} is not a leap year and has #{365*60*24} minutes in the
year"
end

# p018arrays.rb
# Arrays

# Empty array
var1 = []
# Array index starts from 0
puts var1[0]
```

```ruby
# an array holding a single number
var2 = [5]
puts var2[0]

# an array holding two strings
var3 = ['Hello', 'Goodbye']
puts var3[0]
puts var3[1]

flavour = 'mango'
# an array whose elements are pointing
# to three objects - a float, a string and an array
var4 = [80.5, flavour, [true, false]]
puts var4[2]

# a trailing comma is ignored
name = ['Satish', 'Talim', 'Ruby', 'Java',]
puts name[0]
puts name[1]
puts name[2]
puts name[3]
# the next one outputs nil
# nil is Ruby's way of saying nothing
puts name[4]
# we can add more elements too
name[4] = 'Pune'
puts name[4]
# we can add anything!
name[5] = 4.33
puts name[5]
# we can add an array to an array
name[6] = [1, 2, 3]
puts name[6]

# some methods on arrays
newarr = [45, 23, 1, 90]
puts newarr.sort
puts newarr.length
puts newarr.first
puts newarr.last

# method each (iterator) - extracts each element into lang
# do end is a block of code
# variable lang refers to each item in the array as it goes through the loop
languages = ['Pune', 'Mumbai', 'Bangalore']
```

```ruby
languages.each do |lang|
  puts 'I love ' + lang + '!'
  puts 'Don\'t you?'
end

# delete an entry in the middle and shift the remaining entries
languages.delete('Mumbai')
languages.each do |lang|
  puts 'I love ' + lang + '!'
  puts 'Don\'t you?'
end

# p019mtdarry.rb
# if you give return multiple parameters,
# the method returns them in an array
# The times method of the Integer class iterates block num times,
# passing in values from zero to num-1

def mtdarry
  10.times do |num|
    square = num * num
    return num, square
  end
end

# using parallel assignment to collect the return value
num, square = mtdarry
puts num
puts square

# p020arraysum.rb
# Write a Ruby program that, when given an array
# as collection = [1, 2, 3, 4, 5] it calculates the sum of its elements
collection = [1, 2, 3, 4, 5]
sum = 0
collection.each {|i| sum += i}
puts sum

# p021oddeven.rb
# given an array as collection = [12, 23, 456, 123, 4579] it
# displays for each number, whether it is odd or even
collection = [12, 23, 456, 123, 4579]
collection.each do |i|
  if i % 2 == 0
```

```
    puts "#{i} is even"
  else
    puts "#{i} is odd"
  end
end

# p021ranges.rb
=begin
  Sequences have a start point, an end point, and a way to produce successive values
in the sequence
  In Ruby, sequences are created using the ". ." and ". . ." range operators.
  The two dot form creates an inclusive range.
  The three-dot form creates a  range that excludes the specified high value
  The sequence 1..100000 is held as a Range object
=end
digits = -1..9
puts digits.include?(5)        # true
puts digits.min                  # -1
puts digits.max                  # 9
puts digits.reject {|i| i < 5 }  # [5, 6, 7, 8, 9]

# p021rangesex.rb
s = 'key=value'
i = s.index('=')
# If range supplied to a string, as below, a new String is created
puts s[0...i]
puts s[i+1,s.length]

# p022codeblock.rb
=begin
  Ruby Code blocks are chunks of code between braces or
  between do- end that you can associate with method invocations
=end
def call_block
  puts 'Start of method'
  # you can call the block using the yield keyword
  yield
  yield
  puts 'End of method'
end
# Code blocks may appear only in the source adjacent to a method call
call_block {puts 'In the block'}

# p023codeblock2.rb
# You can provide parameters to the call to yield:
```

```ruby
# these will be passed to the block
def call_block
  yield('hello', 99)
end
call_block {|str, num| puts str + ' ' + num.to_s}

# p024proccall.rb
# Blocks are not objects
# they can be converted into objects of class Proc by calling lambda method
prc = lambda {puts 'Hello'}
# method call invokes the block
prc.call

# another example
toast = lambda do
  puts 'Cheers'
end
toast.call

# p025mtdproc.rb
=begin
  You cannot pass methods into other methods (but you can pass procs into methods),
  and methods cannot return other methods (but they can return procs)
=end

def some_mtd some_proc
  puts 'Start of mtd'
  some_proc.call
  puts 'End of mtd'
end

say = lambda do
  puts 'Hello'
end

some_mtd say

# p026phrase.rb
=begin
  If you call rand, you'll get a float greater than or equal to 0.0
  and less than 1.0. If you give it an integer parameter (by calling rand(5) ),
  you will get an integer value greater than or equal to 0 and less than 5
=end

# The program below makes three lists of words, and then randomly picks one word
```

```
# from each of the three lists and prints out the result
word_list_one = ['24/7', 'multi-Tier', '30,000 foot', 'B-to-B', 'win-win', 'front-
end',
                        'web-based', 'pervasive', 'smart', 'six-sigma', 'critical-
path', 'dynamic']
word_list_two = ['empowered', 'sticky', 'value-added', 'oriented', 'centric',
'distributed',
                        'clustered', 'branded', 'outside-the-box', 'positioned',
'networked', 'focused',
                        'leveraged', 'aligned', 'targeted', 'shared', 'cooperative',
'accelerated']
word_list_three = ['process', 'tipping-point', 'solution', 'architecture', 'core
competency',
                          'strategy', 'mindshare', 'portal', 'space', 'vision',
'paradigm', 'mission']

one_len = word_list_one.length
two_len = word_list_two.length
three_len = word_list_three.length

rand1 = rand(one_len)
rand2 = rand(two_len)
rand3 = rand(three_len)

phrase = word_list_one[rand1] + " " + word_list_two[rand2] + " " +
word_list_three[rand3]

puts phrase

# p026zdeafgm1.rb
=begin
  Write a Deaf Grandma program. Whatever you say to grandma
  (whatever you type in), she should respond with HUH?!  SPEAK UP, SONNY!,
  unless you shout it (type in all capitals). If you shout, she can hear you (
  or at least she thinks so) and yells back, NO, NOT SINCE 1938!
  To make your program really believable, have grandma shout a different year
  each time; maybe any year at random between 1930 and 1950.
  You can't stop talking to grandma until you shout BYE
=end
a = ( 1930...1951).to_a
puts 'Enter your response: '
STDOUT.flush
until (response = gets.chomp).eql?('BYE')
  if (response.eql?(response.upcase ))
    puts 'NO, NOT SINCE    ' + a[rand(a.size)].to_s  + ' !'
```

```ruby
  else
    puts 'HUH?!   SPEAK UP, SONNY!'
  end
  puts 'Enter your response: '
  STDOUT.flush
end

# p026zdeafgm2.rb
=begin
  Extend your Deaf Grandma program: What if grandma doesn't want you to leave?
  When you shout BYE, she could pretend not to hear you. Change your previous
  program so that you have to shout BYE three times in a row. Make sure to test
  your program: if you shout BYE three times, but not in a row, you should still
  be talking to grandma
=end
a = ( 1930...1951).to_a
puts 'Enter your response: '
STDOUT.flush
until (response = gets.chomp).eql?('BYE BYE BYE')
  if response.eql?('BYE')
    # do nothing
  elsif response.eql?(response.upcase)
    puts 'NO, NOT SINCE    ' + a[rand(a.size)].to_s  + ' !'
  else
    puts 'HUH?!   SPEAK UP, SONNY!'
  end
  puts 'Enter your response: '
  STDOUT.flush
end

# p027readwrite.rb
# Open and read from a text file
# Note that since a block is given, file will automatically be closed when the block
terminates
File.open('p014constructs.rb', 'r') do |f1|
  while line = f1.gets
    puts line
  end
end

# Create a new file and write to it
File.open('test.rb', 'w') do |f2|
  # use "" for two lines of text
  f2.puts "Created by Satish\nThank God!"
end
```

```ruby
# p028swapcontents.rb - Program to swap the contents of 2 text files
# Asuumptions: The two files exist in the same folder as the program

# Function to read contents of one file and write them to another file
# Accepts 2 file names - file1 and file2
# Reads from file1 and writes to file2
def filereadwrite(file1, file2)
  f2 = File.open(file2, "w")
  f1 = File.open(file1, "r")
  while line = f1.gets
    f2.puts line
  end
  f1.close
  f2.close
end

filereadwrite("file1", "file1.tmp")
filereadwrite("file2", "file1")
filereadwrite("file1.tmp", "file2")

File.delete('file1.tmp')

# p028xrandom.rb
# We now need to display the contents of the file from the word USA
f = File.new("hellousa.rb")

# SEEK_CUR - Seeks to first integer number parameter plus current position
# SEEK_END - Seeks to first integer number parameter plus end of stream
#   (you probably want a negative value for first integer number parameter)
# SEEK_SET - Seeks to the absolute location given by first integer number parameter
# :: is the scope operator - more on this later
f.seek(12, IO::SEEK_SET)
print f.readline
f.close

# p029dog.rb
# define class Dog
class Dog
  def initialize(breed, name)
    # Instance variables
    @breed = breed
    @name = name
  end
```

```ruby
  def bark
    puts 'Ruff! Ruff!'
  end

  def display
    puts "I am of #{@breed} breed and my name is #{@name}"
  end
end

=begin
  Classes in Ruby are first-class objects - each is an instance of class Class.
  When a new class is defined (typically using class Name ... end), an object of
  type Class is created and assigned to a constant (Name. in this case).
  When Name.new is called to create a new object, the new instance method in Class
  is run by default, which in turn invokes allocate to allocate memory for the
object,
  before finally calling the new object's initialize method. The constructing and
  initializing phases of an object are separate and both can be over-ridden.
  The initialization is done via the initialize instance method while the
construction
  is done via the new class method. initialize is not a constructor!
  Class Hierarchy - http://www.cs.mun.ca/%7Edonald/slug/2003-10-
16/presentation/img5.html
=end

# make an object
# Objects are created on the heap
d = Dog.new('Labrador', 'Benzy')

=begin
  Every object is "born" with certain innate abilities.
  To see a list of innate methods, you can call the methods
  method (and throw in a sort operation, to make it
  easier to browse visually)
=end
puts d.methods.sort

# Amongst these many methods, the methods object_id and respond_to? are important.
# Every object in Ruby has a unique id number associated with it
puts "The id of obj is #{d.object_id}."

# To know whether the object knows how to handle the message you want
# to send it, by using the respond_to? method.
if d.respond_to?("talk")
  d.talk
```

```ruby
else
  puts "Sorry, the object doesn't understand the 'talk' message."
end

d.bark
d.display

# making d and d1 point to the same object
d1 = d
d1.display

# making d a nil reference, meaning it does not refer to anything
d = nil
d.display

# If I now say
d1 = nil
# then the Dog object is abandoned and eligible for Garbage Collection (GC)
=begin
  The Ruby object heap allocates a minimum of 8 megabytes.
  Ruby's GC is called mark-and-sweep. The "mark" stage checks objects
  to see if they are still in use. If an object is in a variable that
  can still be used in the current scope, the object (and any object
  inside that object) is marked for keeping. If the variable is long gone,
  off in another method, the object isn't marked. The "sweep" stage then
  frees objects which haven't been marked.
  If you stuff something in an array and you happen to keep that array around,
  it's all marked. If you stuff something in a constant or global variable,
  it's forever marked.
=end

# p030motorcycle.rb
class MotorCycle
  def initialize(make, color)
    # Instance variables
    @make = make
    @color = color
    end
  def startEngine
    if (@engineState)
      puts 'Engine is already Running'
    else
      @engineState = true
      puts 'Engine Idle'
    end
```

```
  end
end

# p031motorcycletest.rb
require_relative 'p030motorcycle'
m = MotorCycle.new('Yamaha', 'red')
m.startEngine

class MotorCycle
  def dispAttr
    puts 'Color of MotorCycle is ' + @color
    puts 'Make  of MotorCycle is ' + @make
  end
end
m.dispAttr
m.startEngine
puts self.class
puts self
puts MotorCycle.instance_methods(false).sort

# p031xdognext.rb
require 'p029dog'
# define class Dog
class Dog
  def big_bark
    puts 'Woof! Woof!'
  end
end
# make an object
d = Dog.new('Labrador', 'Benzy')
d.bark
d.big_bark
d.display

# p032mystring.rb
class String
  def writesize
    puts self.size
  end
end
size_writer = "Tell me my size!"
size_writer.writesize

# p033mammal.rb
class Mammal
```

```ruby
  def breathe
    puts "inhale and exhale"
  end
end

class Cat<Mammal
  def speak
    puts "Meow"
  end
end

rani = Cat.new
rani.breathe
rani.speak

# p034bird.rb
class Bird
  def preen
    puts "I am cleaning my feathers."
  end
  def fly
    puts "I am flying."
  end
end

class Penguin<Bird
  def fly
    puts "Sorry. I'd rather swim."
  end
end

p = Penguin.new
p.preen
p.fly

# p035inherit.rb
class GF
  @m =10
  puts @m.object_id
  def initialize
    puts 'In GF class'
  end
  def gfmethod
    puts 'GF method call'
  end
```

```ruby
end

# class F sub-class of GF
class F < GF
  def initialize
    super
    puts 'In F class'
  end
end

# class S sub-class of F
class S < F
 def initialize
   super
   puts @m
   puts @m.object_id
   puts 'In S class'
  end
end
son = S.new
son.gfmethod

# p036duck.rb
class Duck
  def quack
    'Quack!'
  end

  def swim
    'Paddle paddle paddle...'
  end
end

class Goose
  def honk
    'Honk!'
  end
  def swim
    'Splash splash splash...'
  end
end

class DuckRecording
  def quack
    play
```

```ruby
end

  def play
    'Quack!'
  end
end

def make_it_quack(duck)
  duck.quack
end
puts make_it_quack(Duck.new)
puts make_it_quack(DuckRecording.new)

def make_it_swim(duck)
  duck.swim
end
puts make_it_swim(Duck.new)
puts make_it_swim(Goose.new)

# p037rectangle.rb
# The Rectangle constructor accepts arguments in either
# of the following forms:
#   Rectangle.new([x_top, y_left], length, width)
#   Rectangle.new([x_top, y_left], [x_bottom, y_right])
class Rectangle
  def initialize(*args)
    if args.size < 2  || args.size > 3
      # modify this to raise exception, later
      puts 'This method takes either 2 or 3 arguments'
    else
      if args.size == 2
        puts 'Two arguments'
      else
        puts 'Three arguments'
      end
    end
  end
end
Rectangle.new([10, 23], 4, 10)
Rectangle.new([10, 23], [14, 13])

# p037xmtdovride.rb
class A
  def a
    puts 'In class A'
```

```
  end
end

class B < A
  def a
    puts 'In class B'
  end
end

b = B.new
b.a

# p038bicycle.rb
class Bicycle
  attr_reader :gears, :wheels, :seats
  def initialize(gears = 1)
    @wheels = 2
    @seats = 1
    @gears = gears
  end
end

class Tandem < Bicycle
  def initialize(gears)
    super
    @seats = 2
  end
end
t = Tandem.new(2)
puts t.gears
puts t.wheels
puts t.seats
b = Bicycle.new
puts b.gears
puts b.wheels
puts b.seats

# p038or.rb
class OR
  def mtd
    puts "First definition of method mtd"
  end
  def mtd
    puts "Second definition of method mtd"
  end
```

```ruby
end
OR.new.mtd

# p039symbol.rb
puts "string".object_id
puts "string".object_id
puts :symbol.object_id
puts :symbol.object_id

# p039xsymbol.rb
class Test
  puts :Test.object_id.to_s
  def test
    puts :test.object_id.to_s
    @test = 10
    puts :test.object_id.to_s
  end
end
t = Test.new
t.test

# p039xyzsymbol.rb
know_ruby = 'yes'
if know_ruby == 'yes'
  puts 'You are a Rubyist'
else
  puts 'Start learning Ruby'
end

# p040myhash.rb
h = {'dog' => 'canine', 'cat' => 'feline', 'donkey' => 'asinine', 12 => 'dodecine'}
puts h.length  # 3
puts h['dog']   # 'canine'
puts h
puts h[12]

# p041symbolhash.rb
people = Hash.new
people[:nickname] = 'IndianGuru'
people[:language] = 'Marathi'
people[:lastname] = 'Talim'

puts people[:lastname] # Talim

# p042time.rb
```

```ruby
t = Time.now
# to get day, month and year with century
# also hour, minute and second
puts t.strftime("%d/%m/%Y %H:%M:%S")

# You can use the upper case A and B to get the full
# name of the weekday and month, respectively
puts t.strftime("%A")
puts t.strftime("%B")

# You can use the lower case a and b to get the abbreviated
# name of the weekday and month, respectively
puts t.strftime("%a")
puts t.strftime("%b")

# 24 hour clock and Time zone name
puts t.strftime("at %H:%M %Z")

# p043raise.rb
def raise_exception
  puts 'I am before the raise.'
  raise 'An error has occured'
  puts 'I am after the raise'
end
raise_exception

# p044inverse.rb
def inverse(x)
    raise ArgumentError, 'Argument is not numeric' unless x.is_a? Numeric
    1.0 / x
end
puts inverse(2)
puts inverse('not a number')

# p045handexcp.rb
def raise_and_rescue
  begin
    puts 'I am before the raise.'
    raise 'An error has occured.'
    puts 'I am after the raise.'
  rescue
    puts 'I am rescued.'
  end
  puts 'I am after the begin block.'
end
```

```
raise_and_rescue

# p046excpvar.rb
begin
  raise 'A test exception.'
rescue Exception => e
  puts e.message
    puts e.backtrace.inspect
end

# p046xreadwrite.rb
# Open and read from a text file
# Note that since a block is given, file will automatically be closed when the block
terminates
begin
  File.open('p014constructs.rb', 'r') do |f1|
    while line = f1.gets
      puts line
    end
  end

  # Create a new file and write to it
  File.open('test.rb', 'w') do |f2|
    # use "" for two lines of text
    f2.puts "Created by Satish\nThank God!"
  end
rescue Exception => msg
  # display the system generated error message
  puts msg.message
end

# p047classaccess.rb
class ClassAccess
  def m1      # this method is public
  end
  protected
  def m2      # this method is protected
  end
  private
  def m3      # this method is private
  end
end
ca = ClassAccess.new
ca.m1
#ca.m2
```

```ruby
#ca.m3

# p047zclassaccess.rb
class Person
  def initialize(age)
    @age = age
  end
  def age
    @age
  end
  def compare_age(c)
    if c.age > age
      "The other object's age is bigger."
    else
      "The other object's age is the same or smaller."
    end
  end
  protected :age
end

chris = Person.new(25)
marcos = Person.new(34)
puts chris.compare_age(marcos)
#puts chris.age

# p048accessor.rb
# First without accessor methods
class Song
  def initialize(name, artist)
    @name     = name
    @artist   = artist
  end
  def name
    @name
  end
  def artist
    @artist
  end
end

song = Song.new("Brazil", "Ivete Sangalo")
puts song.name
puts song.artist

# Now, with accessor methods
```

```ruby
class Song
  def initialize(name, artist)
    @name     = name
    @artist   = artist
  end
  attr_reader :name, :artist  # create reader only
  # For creating reader and writer methods
  # attr_accessor :name
  # For creating writer methods
  # attr_writer :name

end

song = Song.new("Brazil", "Ivete Sangalo")
puts song.name
puts song.artist

# p049instvarinherit.rb
class C
  def initialize
    @n = 100
  end

  def increase_n
    @n *= 20
  end
end

class D < C
  def show_n
    puts "n is #{@n}"
  end
end

d = D.new
d.increase_n
d.show_n

# p050newdog.rb
class NewDog
  def initialize(breed)
    @breed = breed
  end
  attr_reader :breed, :name # create reader only
```

```ruby
  # setter method
  def name=(nm)
    @name = nm
  end
end

nd = NewDog.new('Doberman')
#nd.name=('Benzy')
nd.name = 'Benzy'
puts nd.name

# p050xfreeze.rb
str = 'A simple string. '
str.freeze
begin
  str << 'An attempt to modify.'
rescue => err
  puts "#{err.class} #{err}"
end
# The output is - TypeError can't modify frozen string

# p051gamecharacters.rb
class GameCharacter
  def initialize(power, type, weapons)
    @power = power
    @type = type
    @weapons = weapons
  end
  attr_reader :power, :type, :weapons
end

# p052dumpgc.rb
require 'p051gamecharacters'
gc = GameCharacter.new(120, 'Magician', ['spells', 'invisibility'])
puts gc.power.to_s + ' ' + gc.type + ' '
gc.weapons.each do |w|
  puts w + ' '
end

File.open('game', 'w+') do |f|
  Marshal.dump(gc, f)
end

# p053loadgc.rb
require 'p051gamecharacters'
```

```ruby
File.open('game') do |f|
  @gc = Marshal.load(f)
end

puts @gc.power.to_s + ' ' + @gc.type + ' '
@gc.weapons.each do |w|
  puts w + ' '
end

# p054constwarn.rb
A_CONST = 10
A_CONST = 20

# p055constalter.rb
A_CONST = "Doshi"
B_CONST = A_CONST
A_CONST[0] = "J" # alter string referenced by constant
puts A_CONST # displays Joshi
puts B_CONST

# p056const.rb
OUTER_CONST = 99

class Const
  def get_const
    CONST
  end
  CONST = OUTER_CONST + 1
end

puts Const.new.get_const
puts Const::CONST
puts ::OUTER_CONST
puts Const::NEW_CONST = 123

# p057mymethods2.rb
# variables and methods start lowercase
$glob = 5             # global variables start with $
class TestVar         # class name constant, start uppercase
 @@cla = 6            # class variables start with @@
 CONST_VAL = 7        # constant style, all caps, underscore
 def initialize(x)    # constructor
  @inst = x          # instance variables start with @
  @@cla += 1         # each object shares @@cla
 end
```

```
 def self.cla          # class method, getter
  @@cla
 end
 def self.cla=(y)      # class method, setter, also TestVar.
  @@cla = y
 end
 def inst              # instance method, getter
  @inst
 end
 def inst=(i)          # instance method, setter
  @inst = i
 end
end
puts $glob
test = TestVar.new(3) # calls constructor
puts TestVar.cla      # calls getter
puts test.inspect     # gives object ID and instance vars
TestVar.cla = 4       # calls setter
test.inst=8           # calls setter
puts TestVar.cla
puts test.inst        # calls getter
other = TestVar.new(17)
puts other.inspect
puts TestVar.cla

# p058mytrig.rb
module Trig
  PI = 3.1416
  # class methods
  def Trig.sin(x)
    # ...
  end
  def Trig.cos(x)
    # ...
  end
end

# p059mymoral.rb
module Moral
  VERY_BAD = 0
  BAD      = 1
  def Moral.sin(badness)
    # ...
  end
end
```

```ruby
# p060usemodule.rb
require 'p058mytrig'
require 'p059mymoral'
Trig.sin(Trig::PI/4)
Moral.sin(Moral::VERY_BAD)

# p061mixins.rb
module D
  def initialize(name)
    @name =name
  end
  def to_s
    @name
  end
end

module Debug
  include D
  # Methods that act as queries are often
  # named with a trailing ?
  def who_am_i?
    "#{self.class.name} (\##{self.object_id}): #{self.to_s}"
  end
end

class Phonograph
  # the include statement simply makes a reference to a named module
  # If that module is in a separate file, use require to drag the file in
  # before using include
  include Debug
  # ...
end

class EightTrack
  include Debug
  # ...
end

ph = Phonograph.new("West End Blues")
et = EightTrack.new("Real Pillow")
puts ph.who_am_i?
puts et.who_am_i?

#  p062stuff.rb
```

```ruby
#  A module may contain constants, methods and classes.
#  No instances

module Stuff
  C = 10
  def Stuff.m(x)  # prefix with the module name for a class method
    C*x
  end
  def p(x)        # an instance method, mixin for other classes
    C + x
  end
  class T
    @t = 2
  end
end
puts Stuff::C     # Stuff namespace
puts Stuff.m(3)   # like a class method
x = Stuff::T.new
# uninitialized constant error, if you try the following
# puts C

# p063stuffusage.rb
require 'p062stuff'     # loads Stuff module from Stuff.rb
                    # $: is a system variable -- contains the path for loads
class D
  include Stuff     # refers to the loaded module
  puts Stuff.m(4)
end

d = D.new
puts d.p(5)         # method p from Stuff
puts $:             # array of folders to search for load
$: << "c:/"         # add a folder to the load path
puts $:
puts Stuff.m(5)     # Stuff class methods not called from D object

# p063xself1.rb
class S
  puts 'Just started class S'
  puts self
  module M
    puts 'Nested module S::M'
    puts self
  end
  puts 'Back in the outer level of S'
```

```
  puts self
end

# p063xself2.rb
class S
  def m
    puts 'Class S method m:'
    puts self
  end
end
s = S.new
s.m

# p063xself3.rb
obj = Object.new
def obj.show
  print 'I am an object: '
  puts "here's self inside a singleton method of mine:"
  puts self
end
obj.show
print 'And inspecting obj from outside, '
puts "to be sure it's the same object:"
puts obj

# p063xself4.rb
class S
  def S.x
    puts "Class method of class S"
    puts self
  end
end
S.x

# p064regexp.rb
string = "My phone number is (123) 555-1234."
phone_re = /\((\d{3})\)\s+(\d{3})-(\d{4})/
m = phone_re.match(string)
unless m
  puts "There was no match..."
  exit
end
print "The whole string we started with: "
puts m.string
print "The entire part of the string that matched: "
```

```ruby
puts m[0]
puts "The three captures: "
3.times do |index|
  puts "Capture ##{index + 1}: #{m.captures[index]}"
end
puts "Here's another way to get at the first capture:"
print "Capture #1: "
puts m[1]

# p065my_first_test.rb
require 'test/unit'

class MyFirstTest < Test::Unit::TestCase
  def test_for_truth
    assert true
  end
end

# p066testradius.rb
require 'test/unit'
require 'p067radius'
class TestRadius < Test::Unit::TestCase
=begin
  def test_key
    robj = Radius.new('78')
    assert_equal('78', robj.key)
    robj = Radius.new(78)
    assert_equal('78', robj.key)
      robj = Radius.new([78])
    assert_nil(robj.key)
  end
end
=end

  def setup
    @robj = Radius.new('78')
  end
  def test_key
    assert_equal('78', @robj.key)
    @robj = Radius.new(78)
    assert_equal('78', @robj.key)
    @robj = Radius.new([78])
    assert_nil(@robj.key)
  end
end
```

```ruby
# p067radius.rb
class Radius
  attr_reader :key
  def initialize(key)
    @key = key
    if @key.class == Fixnum then
      @key = @key.to_s
    end
    if @key.class != String then
      @key = nil
    end
  end
end

# p068dtserver.rb
# Date Time Server - server side using thread
# usage: ruby p068dtserver.rb

require "socket"

dts = TCPServer.new('localhost', 20000)
loop do
  Thread.start(dts.accept) do |s|
    print(s, " is accepted\n")
    s.write(Time.now)
    print(s, " is gone\n")
    s.close
  end
end

# p069dtclient.rb
require 'socket'

streamSock = TCPSocket.new( "127.0.0.1", 20000 )
#streamSock.send( "Hello\n" )
str = streamSock.recv( 100 )
print str
streamSock.close

# p070rubysmtp.rb
require 'net/smtp'
user_from = "superman@world.com"
user_to = "batman@world.com"
the_email = "From: superman@world.com\nSubject: Hello\n\nEmail by Ruby.\n\n"
```

```ruby
# handling exceptions
begin
  Net::SMTP.start('localhost', 25) do |smtpclient|
    smtpclient.send_message(the_email, user_from, user_to)
  end
rescue Exception => e
  print "Exception occured: " + e
end

# p070thread1.rb
puts Thread.main
puts ""
t1 = Thread.new {sleep 100}
Thread.list.each {|thr| p thr }
puts "Current thread = " + Thread.current.to_s
puts ""

t2 = Thread.new {sleep 100}
Thread.list.each {|thr| p thr }
puts Thread.current
puts ""

Thread.kill(t1)
Thread.pass                         # pass execution to t2 now
t3 = Thread.new do
  sleep 20
  Thread.exit                        # exit the thread
end
Thread.kill(t2)                      # now kill t2
Thread.list.each {|thr| p thr }

# now exit the main thread (killing any others)
Thread.exit

# p070thread.rb
x = Thread.new { sleep 0.1; print "x"; print "y"; print "z" }
a = Thread.new { print "a"; print "b"; sleep 0.2; print "c" }
x.join # Let the threads finish before
a.join # main thread exits...

# p072soapserver.rb
require 'logger'
require 'soap/rpc/standaloneServer'
class MyServer < SOAP::RPC::StandaloneServer
  def initialize(* args)
```

```
    super
    add_method(self, 'sayhelloto', 'username')
    @log = Logger.new("soapserver.log", 5, 10*1024)
  end
  def sayhelloto(username)
    t = Time.now
    @log.info("#{username} logged on #{t}")
    "Hello, #{username} on #{t}."
  end
end

server = MyServer.new('RubyLearningServer','urn:mySoapServer','localhost',12321)
trap('INT') {server.shutdown}
server.start

# p073prclient.rb
require 'soap/rpc/driver'
driver = SOAP::RPC::Driver.new('http://127.0.0.1:12321/', 'urn:mySoapServer')
driver.add_method('sayhelloto', 'username')
puts driver.sayhelloto('RubyLearning')

# p074hellotk.rb
require 'tk'
hello = TkRoot.new {title "Hello World"}
Tk.mainloop

# p075hellotk1.rb
require 'tk'
hello = TkRoot.new do
  title "Hello World"
  # the min size of window
  minsize(400,400)
end
TkLabel.new(hello) do
  text 'Hello World'
  foreground 'red'
  pack { padx 15; pady 15; side 'left'}
end
Tk.mainloop

# p076hellotk2.rb
require 'tk'
TkButton.new do
  text "EXIT"
  command { exit }
```

```
  pack('side'=>'left', 'padx'=>10, 'pady'=>10)
end
Tk.mainloop

# p077soapguiclient.rb
require 'soap/rpc/driver'
require 'tk'

class SOAPGuiClient
  def connect
    @buttonconnect.configure('text' => 'Reset')
    @buttonconnect.command { reset }
    begin
      driver = SOAP::RPC::Driver.new('http://217.160.200.122:12321/',
'urn:mySoapServer')
      driver.add_method('sayhelloto', 'username')
      s = driver.sayhelloto('Satish Talim')
    rescue Exception => e
      s =  "Exception occured: " + e
    ensure
      @label.configure('text' => s)
    end
  end #connect

  def reset
    @label.configure('text' => "")
    @buttonconnect.configure('text' => 'Connect')
    @buttonconnect.command { connect }
  end # reset

#---
  def initialize
    root = TkRoot.new do
      title 'SOAP Client'
      # the min size of window
      minsize(535, 100)
  end
#---
    @label = TkLabel.new(root) do
      pack
    end
#---
    @buttonconnect = TkButton.new(root)
    @buttonconnect.configure('text' => 'Connect')
    @buttonconnect.command { connect }
```

```ruby
    @buttonconnect.pack('side'=>'bottom')
#---
    Tk.mainloop
  end #initialize
end # class

SOAPGuiClient.new
#---

# p078rubymysql.rb
require 'mysql'

#my = Mysql.new(hostname, username, password, databasename)
con = Mysql.new('localhost', 'root', '', 'ruby')
rs = con.query('select * from student')
rs.each_hash { |h| puts h['name']}
con.close

# p079rubyquirk1.rb
class MotorCycle
  def initialize(make, color)
    @make, @color = make, color
  end
end

m = MotorCycle.new('Honda', 'blue')
m.instance_variable_set(:@make, 'Kawasaki')
m.instance_variable_set(:@gears, 4)
puts m.inspect

# p080dbconnect.rb
require 'rubygems'
require 'active_record'
ActiveRecord::Base.establish_connection(
:adapter=> "mysql",
:host => "localhost",
:database=> "students"
)

class Rubyist < ActiveRecord::Base
end

Rubyist.create(:name => 'Mitali Talim', :city => "Nashville, Tenessee")
Rubyist.create(:name => 'Sunil Kelkar', :city => "Pune, India")
Rubyist.create(:name => 'Adam Smith', :city => "San Fransisco, USA")
```

```
participant = Rubyist.find(:first)
puts %{#{participant.name} stays in #{participant.city}}

Rubyist.find(:first).destroy
```