Writing Own Ruby Methods

```
# 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')
```

```ruby
# 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.class
puts mtd << "Ruby"
```

```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

puts oldmtd

#alias newmtd oldmtd

def oldmtd
  "old improved method"
end

alias newmtd oldmtd
puts oldmtd
puts newmtd
```

```ruby
# p011vararg.rb
# Variable number of parameters example:
# The asterisk(*splat) is actually taking all method arguments sent
# 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()

def opt_args(a,*x,b)
  "#{a}, #{x}, #{b}"
end

puts opt_args("first_arg", "second_arg", "third_arg")
```

```ruby
# p012zmm.rb

class Dummy
  def method_missing(m, *args)
    txt = "There's no method called #{m} here.\n-- please try again."
    puts txt
  end
end

Dummy.new.anything
# There's no method called anything here.
# -- please try again.
# p012mtdstack.rb

# The sequence the parameters are put on the stack is left to right.

def mtd(a=99, b=a+1)
  [a,b]
end

puts mtd

# Are the parameters passed by value or reference?
#  Observe the following example:

def downer(string)
  string.downcase
end

a = "HELLO"
downer(a)      # -> "hello"
puts a         # -> "HELLO"

def downer(string)
  string.downcase!
end

a = "HELLO"
downer(a)      # -> "hello"
puts a         # -> "hello"
```