Common Programming Errors

Introduction

- We look at some common programming errors from assignment submissions from previous years.
- In many cases, you haven't seen the assignment
  - the advice is generally applicable however.
- Remember, the most important thing is that your program works
  - the issues we look at here will help you turn a very good solution into an excellent one.

Some Basics

- Follow the input specification exactly
  - Use the given I/O specifications
- Submit .rb files only
  - Folders, Rails files, IDE project files etc. are not .rb files
- Provide a comment on submission
  - If it works, tell me.
- Now we look at some common programming errors...

Don’t print from inside a class

```
class Die
  ....
  def print_stats
    puts "{ "
    @sides_stats.each do |key,value
      print "#{key}=>#{value} "
    end
    puts "}"
  end
  ...
end # end of Die class
```

Prefer a `to_s` method that returns a string

Why is this bad?

In general, keep I/O out of your classes!

Avoid Duplication

```
if @dice.die_face == :East
   @location.x += 1
   @path_taken.push(self.location_of_kangaroo)
end
if @dice.die_face == :West
   @location.x -= 1
   @path_taken.push(self.location_of_kangaroo)
end
if @dice.die_face == :North
   @location.y += 1
   @path_taken.push(self.location_of_kangaroo)
end
if @dice.die_face == :South
   @location.y -= 1
   @path_taken.push(self.location_of_kangaroo)
end
```

How to improve this code?

Fixing the Duplication

```
case @dice.die_face
  when :NORTH @location.y += 1
  when :SOUTH @location.y -= 1
  when :EAST @location.x += 1
  when :WEST @location.x -= 1
end
@path_taken.push(self.location_of_kangaroo)
```

This code is shorter and clearer than the original.

More Duplication

```
class Shop
   def total_price
     total_price = 0
     @items.each do |line|
       total_price = total_price + line.price
     end
   end

   def average_price
     average_price = 0.0
     @items.each do |line|
       average_price = average_price + line.price
     end
     average_price = average_price/@items.size
   end
end
```

`average_price` should call **`total_price`**, not repeat its code.
`line` is a poor name, and the += operator should be used

Write clear methods; use clear method names

A method in a Point class:

```
def point a, b                 #"a" and "b" are arguments
  @p = Point.new @x, @y
  @x += a                      #Adds argument "a" to "@x"
  @y += b                      #Adds argument "b" to "@y"
  @q = Point.new @x, @y
  puts "#{@p} -> #{@q}"        #Prints out both points
  if beyond_dimensions?        #Calls method below
    puts "^invalid move^"      #Prints out
    @p
  else
    @q
  end
end
```

- method too complicated
- bad name
- uninformative comments

How to improve this?

Improved Version

```
# increment x and y coords by a and  b
def incr_point a, b
  @x += a
  @y += b
  if beyond_dimensions?
    puts "^invalid move^"  # roll back changes
    @x -= a
     @y -= b
  end
  p
end
```

Method name and comments are clearer.
Code is shorter and easier to read.
I'd like to remove the line that returns **`p`** as well.

Make lower level classes do the work

A method in the Kangaroo class:

```
def hop! # Make hop in random direction
    direction = @die.throw
    if direction == :EAST
        if @point.x==@gridsize
            hop!
        else
            @point.east
            @die.stats[:EAST]+=1
            @die.stats[:TOTAL]+=1
        end
    elsif direction == :WEST
        if @point.x==0
            hop!
        else
            @point.west
            @die.stats[:WEST]+=1
            @die.stats[:TOTAL]+=1
        end
```

Similar code for NORTH and SOUTH omitted...

- too much detail!
- case statement better
- avoid recursion

Similar method from sample solution

```
# Hop to random location inside the grid
def hop!
  @location.move(@die.throw)
  while @grid.lies_outside?(@location) do
    @location.undo
    @location.move(@die.throw)
  end
  @num_of_hops += 1
  @locations_visited.push @location.clone
end
```

The detailed work is delegated to the Die and Grid classes. The algorithm in the **`hop`** method is now much clearer.

Another example

Part of the main script:

```
# Print die stats
total_throws = 0
rw.stats.each do |direction, num_of_throws|
  total_throws += num_of_throws
end
puts "Total hops the kangaroo took: #{rw.num_of_hops}"
puts "Die stats are as follows:"
[:NORTH, :SOUTH, :EAST, :WEST].each do |direction|
  percentage = ...
  puts "#{direction}: #{percentage}%"
end
```

First 4 lines better moved to a more suitable class.

Minimise Commenting (yes, they lied to you)

```
# while loop to run until the final_location?
# method returns true meaning the kangaroo has
# reached its destination
while (!final_location?)
# get a random direction by calling the throw method
# of the die class
direction = @die.throw
# store the current x and y points of the kangaroo's
# location in local variables
x = @skippy.location.x
y = @skippy.location.y
```

This code is ruined by comments.

Same Code without Comments

```
while (!final_location?)
direction = @die.throw
   x = @skippy.location.x
   y = @skippy.location.y
   ...
end
```

Nice, clear code, no comments required.

Avoid long methods

```
# Move the kangaroo in direction given by dir

 def hop! die,dim
  dir = die.throw
  @dim = dim
  @xval = @pos.x1
  @yval = @pos.y1

  # Checks to see if kangaroo object is at a boundary,
  # given by the dim argument.
  # If true it checks to see what boundary, movement is restricted.
  # If false it is allowed move in any direction
  if at_boundary? == true
    if @yval == 0 && @xval==0
      case dir
      when ":North" then @pos.x1 += 1
      when ":East" then @pos.y1 += 1
      end
    elsif @yval ==0
      @hops += 1
      case dir
      when ":North" then @pos.x1 += 1
      when ":East" then @pos.y1 += 1
      when ":South" then @pos.x1 -= 1
      end
    elsif @xval == 0
      @hops += 1
      case dir
      when ":North" then @pos.x1 += 1
      when ":East" then @pos.y1 += 1
      when ":West" then @pos.y1 -=1
      end
    elsif @yval == (@dim-1)
      @hops += 1
      case dir
      when ":North" then @pos.x1 += 1
      when ":South" then @pos.x1 -= 1
      when ":West" then @pos.y1 -=1
      end
    elsif @xval == (@dim-1)
      @hops += 1
      case dir
      when ":East" then @pos.y1 += 1
      when ":South" then @pos.x1 -= 1
      when ":West" then @pos.y1 -=1
      end
    end
  else
    @hops += 1
    case dir
    when ":North" then @pos.x1 += 1
    when ":East" then @pos.y1 += 1
    when ":South" then @pos.x1 -= 1
    when ":West" then @pos.y1 -=1
    end
  end
end
```

- Rewrite long methods
- Consider splitting

Poor layout shows you don’t care

```
class Die

  # attr_accessor :die

  $die = { :NORTH=>0, :SOUTH=>0,:EAST=>0,:WEST=>0 }

  def throws

    @num =  rand(4)

#    print " rolled a: #{@num} == "

       ret_num = @num

       @num = case
           when @num == 0

#      puts "\\n @num is: "; puts $@num
#      print :NORTH
            $die[:NORTH] +=1

           when @num == 1
#             print :SOUTH
              $die[:SOUTH] += 1

           when @num == 2
#             print :EAST
             $die[:EAST] += 1
       when @num == 3
#            print :WEST
              $die[:WEST] += 1
       end

    return ret_num

  end

end
```

Code that looks well shows you care about:
- the code
- your colleagues

Don’t define instance variables you don’t need

```
class RandomWalk
  def initialize dimension
    @grid = Grid.new dimension
    @skippy = Kangaroo.new @grid
  end

  def start
    while !@skippy.at_home?
      @skippy.hop!
    end
  end

  def stats
    @skippy.die_stats
  end

  def num_of_hops
    @skippy.num_of_hops
  end

  def locations_visited
    @skippy.locations_visited
  end
end
```

What’s wrong here?

@grid should be
a normal variable

Proper classes have instance variables

```
class Product

  def real_price(input)
    @price = input.split.last
    incl_vat = @price.to_f*1.2
  end

  def output_name(input)
    @name = input.split.first
  end

end
```

What’s wrong here?

This isn’t a class, it’s a pair of stand-alone functions.

Use Inheritance Appropriately

```
class Shop
  ...
  ...
  ...
end


class Item < Shop
  ...
  ...
  ...
end
```

The **is a** test

To test if X should be a subclass of Y,
ask if this statement is true:
"an instance of X **is a**n instance of Y"

Simple assignment, simple solution

```
class Shop

  def print     #prints all the items in ths shop together with their post VAT prices
    a=[]
    names=[]                                      #an array to save all the names of the items
    prices=[]                                     #an array to save all pre-VAT prices of the items
    f= File.open('shop.dat').each_line{ |s|
      a =s.split("\t")
      names << a[0]
      prices << a[1]
    }
    v=-1
    tax = []                                      #an array to save all post-VAT prices of the
items
    while (v+2 <= prices.count)

      v+=1
      tax << prices[v].to_f + ((prices[v].to_f) * 20/100)
    end
    k=0
    puts "#{names[k]} #{tax[k]} "
    while(k<=names.count)
      k+=1
      puts "#{names[k]} #{tax[k]} "
    end
  end

  def total_value         #returns total value of all the items in the shop combined (post-VAT)
    a=[]
    prices=[]
    f= File.open('shop.dat').each_line{ |s|
      a =s.split("\t")
      prices << a[1]
    }
    v=-1
    tax = []
    while (v<= prices.count)
      v+=1
      tax << prices[v].to_f + ((prices[v].to_f) * 20/100)
    end
    total_tax = tax.inject {|result, element| result + element}  #add all elements in array together
    return total_tax
  end

  def average_value       #returns average value of all items in shop (post-VAT)
    a=[]
    prices=[]
    f= File.open('shop.dat').each_line{ |s|
      a =s.split("\t")
      prices << a[1]
    }
    num_of_els = prices.count
    avg_val = total_value/num_of_els
    return avg_val
  end

end

s = Shop.new
s.print
puts "Total value of items in the shop: #{s.total_value.round(2)}"
puts "Average value of items in the shop: #{s.average_value.round(2)}"
```

If the problem is simple, the solution should be simple too.

Sample Solution: Shop Class

```
class Shop
  def initialize
    @products = []
  end

  def add_product product
    @products.push product
  end

  def to_s
    @products.join', '
  end

  def total_net_value # total net value of the items in the shop
    sum=0
    @products.each {|product| sum+=(product.net_price*product.count) }
    return sum
  end

  def average_price # average price (incl. VAT) of the products in the shop
    total_product_gross=0
    @products.each {|product| total_product_gross+=(product.gross_price)}
    total_product_gross / @products.length
  end
end
```

Simple code. No method more than 3 lines long.

Sample Solution: Item Class

```
VAT = 20 # Rate of Value Added Tax (a percentage)

class Product
  attr_reader :net_price, :count
  def initialize name, net_price, count
    @name = name
    @net_price = net_price
    @count = count
  end

  def gross_price
    @net_price * (1.0 + VAT/100.0)
  end

  def to_s
    "#{@name} #{@net_price} #{@count}"
  end
end
```

Sample Solution: main script

```
# Read input file and create Product objects
shop = Shop.new
IO.foreach('shop.dat') do |line|
  data =  line.split # so data is an array of strings
  name = data[0]
  price = data[1].to_f
  count = data[2].to_i
  product = Product.new(name, price, count)
  shop.add_product product
end

puts 'Here are the contents of the shop:'
puts shop.to_s
puts sprintf "total value in shop: %.2f", shop.total_net_value
puts sprintf "average product price: %.2f", shop.average_price
```

Summary

We've looked at several common errors from past programming assignments.

Try to absorb these and avoid the same mistakes in your work.

Don't forget that first and foremost, your code has to work correctly.