Ruby Lab Exercise Sheet

Work through the examples given on the sheet and make sure you understand what the programs are doing. When you have this completed attempt the 5 questions listed at the end of this document!

A sample solution will be made available on Moodle next week.

Example 1: Create a program to ask the user to enter their name. When they have entered their name output a greeting!

```
print ("Enter your name: ")

name = gets()

puts ("Hello #{name}")
```

`gets()` *is the opposite of puts, it reads in information!!*

Example 2: Product tax calculator

We want to be able to calculate the selling price or grand total of some item based on its pre-tax value or subtotal. To do this multiply the original value of the item by the tax rate and then add the result to the value of the item.

```
itemprice = 100

taxrate = 0.175

taxdue = itemprice * taxrate

puts " Tax due on €#{itemprice} is €#{taxdue}, so the overall price will
be €#{itemprice+taxdue}"
```

Example 3: if statements

```
itemprice = 1200

if (itemprice > 1000) then

    puts (" This item is expensive!! ")

end
```

Example 4: Classes and Objects

```
class Person
  def set_name( aName )
```

```
        @name = aName
      end

  def get_name
      return @name
  end
end
class Book
  def initialize( aName, aDescription )
      @name         = aName
      @description  = aDescription
  end
  def to_s # override default to_s method
      "The #{@name} book is #{@description}\n"
  end
end

person1 = Person.new
person1.set_name( "John Mc" )

puts person1.get_name

b1 = Book.new("Ruby", "this is a great Ruby book!!")
b2 = Book.new("Rails", "A great introduction to Rails!!") puts
b1.to_s
puts b2.to_s
# The inspect method lets you look inside an object
puts "Inspecting 1st Book: #{b1.inspect}"
```

The Person class has get and set methods. The object will be created first and its values will be assigned. The Book class contains a method named initialize, which takes two arguments. These two values will be assigned to the book name and description as the object is being created. The Book class also contains a to_s method which will override the default to_s method. The to_s method is usually used to convert a different type to a string. For our example we have overridden this method and gave it some new functionality! Using the .inspect method gives a human-readable representation of the object!

Example 5: If you want to inspect an object and print its details:

```
p(anobject)
```

where anobject is some object in your program!

Eg. From Example 4

```
p (person1)
```

Example 6: In Ruby, a class can only inherit from a single other class. Some other languages support multiple inheritance, a feature that allows classes to inherit features from multiple classes, but Ruby doesn't support this.

```
class Mammal
    def breathe
        puts "inhale and exhale"
    end
end

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

class Lion < Mammal
    def speak
        puts "Roar!!!"
    end
end

dodger = Cat.new
dodger.breathe
dodger.speak

lion1 = Lion.new
lion1.breathe
lion1.speak
```

Though we didn't specify how a Cat should breathe, every cat will inherit that behaviour from the Mammal class since Cat was defined as a subclass of Mammal. (In OO terminology, the smaller class is a subclass and the larger class is a super-class. The subclass is sometimes also known as a derived or child class and the super-class as base or parent class). Hence from a programmer's standpoint, cats get the ability to breathe for free; after we add a speak method, our cats can both breathe and speak.

Example 7: Inheritance – Passing arguments to the superclass

```
class Mammal

  def initialize(planet)
      @planet = planet
  end

  def breathe
      puts "inhale and exhale"
  end

end

class Cat < Mammal

  def initialize(age,planet)
      @age = age
      super(planet)
  end

  def speak
      puts "Meow"
  end

  def age

      puts @age

  end

  def planet

      puts @planet
  end
end

dodger = Cat.new(21,"Earth")
dodger.breathe
dodger.speak
dodger.age
dodger.planet
```

To pass arguments to a superclass, use the super() method in the child class to pass the required arguments to the parent class. Run the above example to understand how this works!

Questions

Question 1: Create a Hello World program named helloworld.rb that outputs the string "Hello World!" to the terminal.

Hint – use the puts command

Question 2: See if you can determine what the output would be for the following:

```
puts ("\n\t#{(1+2)*3}\nGoodbye")
```

Create a program to see if you were correct!!

Question 3: Create a program to ask the user to enter the price for a specific item before tax. Calculate the overall price of the item using a tax rate of 21%

Display all relevant information back to the user!

Hint: See Example 2

Question 4: Create a class named person with a name and an age.

Use the `initialize()` method.

Use `attr_accessor` to create the get and set methods.

Question 5: Create a class named student that inherits from the person class created in Question 4

A student should have a college name and a student ID.

Create two student objects and output their data!!