COMP41100

EXPLORING PROGRAMMING IN RUBY

PRACTICAL ONE

21st September 2013

Andrew Doyle
Student Number: 12252388

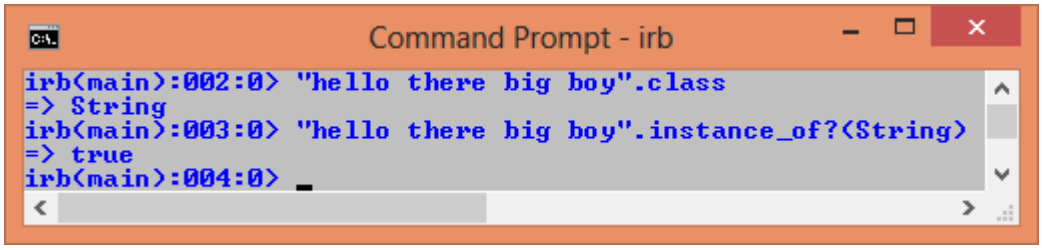
University College Dublin

QUESTION ONE

a) String

"Hello there big boy"

Figure 1a - String



"Hello there big boy", is an example of a ***string*** object. A string is a collection of characters (which includes digits, letters, symbols and whitespace) of any length. All strings in Ruby are objects of type ***String***. As shown in Figure 1a, when you append the ***class*** method to the string, it confirms that it is a string object.

When a string is embedded directly into code (i.e. pre-embedded within a program) using quotation marks, it is referred to as a string literal.

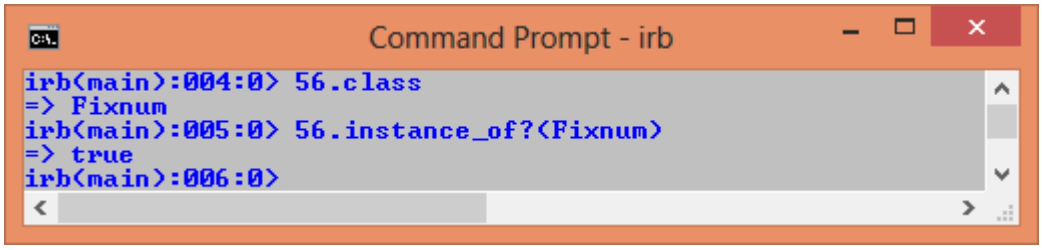
By appending ***instance_of?(String)*** to the characters typed in the string, a test is being carried out to check whether the object is an instance of a given class. This method will return true if the object is an instance of the given class. In this case the "object" is "hello there big boy", and the given class is **String**. In Figure 1a, it returns **true** as shown.

The ***instance_of?*** method will be used throughout this practical to confirm that the instance`s checked are members of a particular class.

b) Fixnum

56

Figure 1b - Fixnum



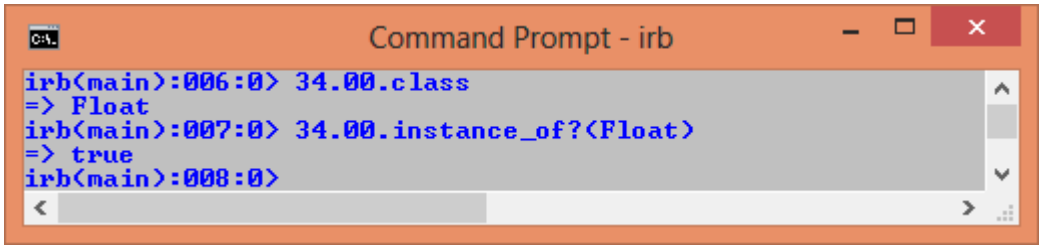
56 is an **integer** value and this is confirmed as shown in Figure 1b by appending the class method which returns **Fixnum**. As outlined by Ruby-doc.org (n.d) a **Fixnum** holds **integer** values that can be represented in a native machine word (minus 1 bit). This is in contrast to **Bignum**, which holds integers which fall outside of the range of **Fixnum**.

If an integer is too big for **Fixnum**, it is converted to **Bignum**. Conversely, if a calculation is carried out involving **Bignum** which resulted in an integer that would fall into the range of values provided by **Fixnum**, the result would be automatically converted to **Fixnum**.

c) Float

34.00

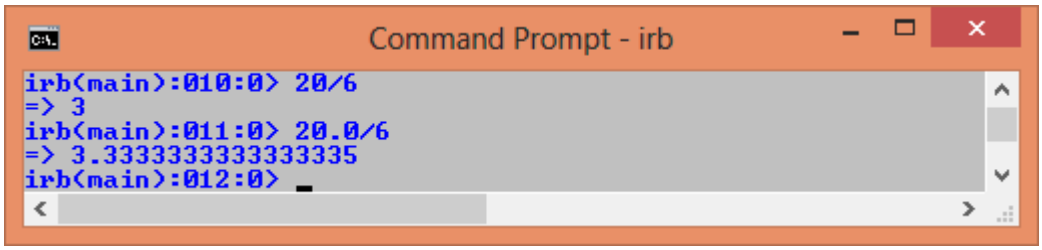
Figure 1c - Float



34.00 represents a ***Floating Point Number***. With ***Integers***, when dividing 20 by 6, it would return 3 (a whole number). As discussed by Cooper (2009, p.37), "*by default, Ruby considers any numbers without a floating point (also known as a decimal point) to be an integer—a whole number*".

In order to achieve the desired result one must work with numbers belonging to the ***Float*** class. Taking the example mentioned above, one should divide 20.0 / 6.0, as shown in Figure 1c.a. The number of decimal digits usually defaults to 15, however, in Figure 1c.a there are 16 decimal digits.

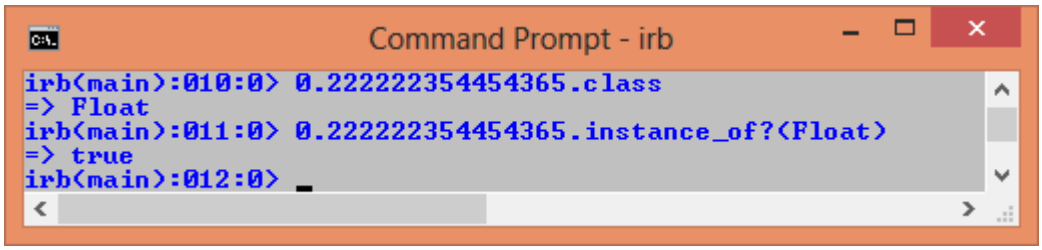
Figure 1c.a – Floating point number arithmetic



d) Float Example 2

.222222354454365

Figure 1d – Float Example 2



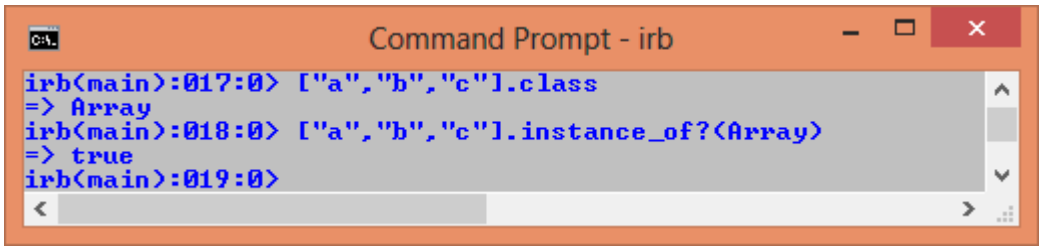
The figure 0.222222354454365 is also a floating point number, and has 15 decimal digits which is the typical maximum length of a floating point decimal number.

e) Array

["a","b","c"]

["a","b","c"] is an example of an **Array**. An array is an ordered collection of objects, which are integer indexed; starting at zero. In the given example, "a" would be at index zero, and "c" at index two.

Figure 1e - Array

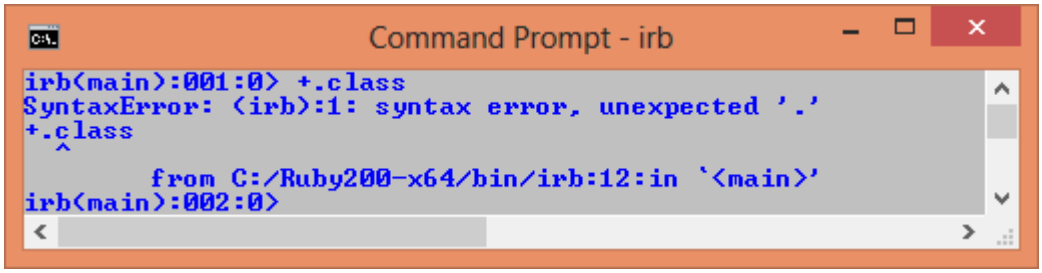


f) + operator

+

+ is an operator that is used in an expression to manipulate objects. For example it can be used to perform arithmetic on numbers, concatenate arrays and match expressions. As shown in Figure 1f, it is not an object.

Figure 1f - + operator

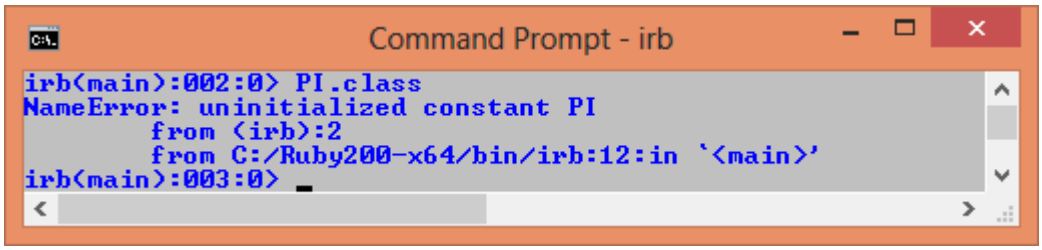


g) PI

PI

In Ruby, **PI** is a constant in the **Math** module, and represents an approximation of **π**. As shown in Figure 1g, and in part **h)**, one must prepend the Math module identifier to **PI** in order to determine it`s object type.

Figure 1g - PI

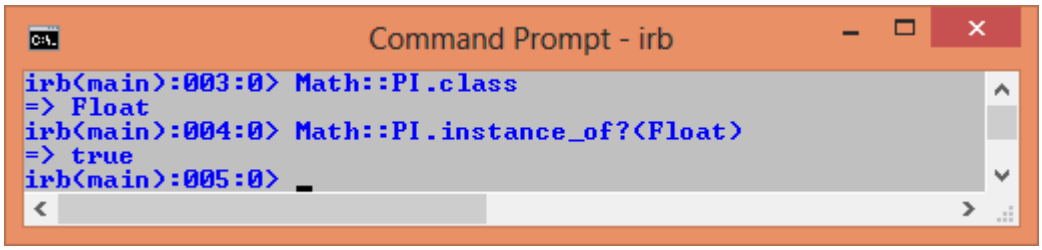


h) Math::PI

Math::PI

All constants (**PI** and **E**) and **Public Class Methods** in the **Math** module are represented by **floating point numbers**, which is why the **PI** constant belongs to the **Float** class.

Figure 1h – Math::PI

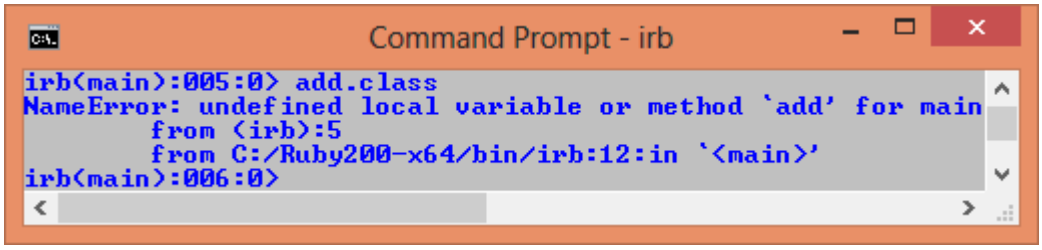


i) Add instance method

add

add is an **instance method** belonging to the **ThreadGroup** class. One can call the object.class method on an instance of a class, but not on the methods.

Figure 1i – Add instance method

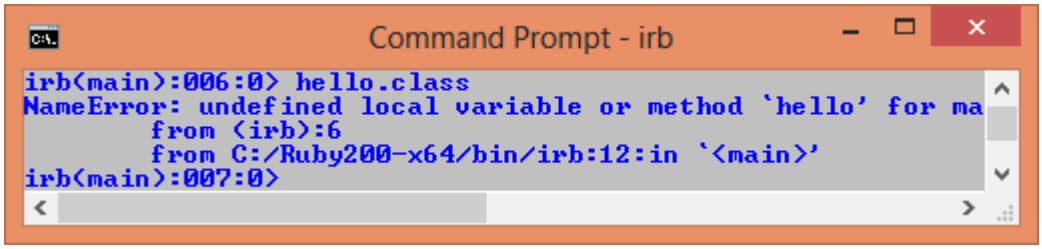


j) Unassigned Variable

Hello

In the example screenshot below, **hello** returns an error as it has not been assigned a value, and therefore could not be classified as an object. This issue is solved in part (k).

Figure 1j – unassigned variable

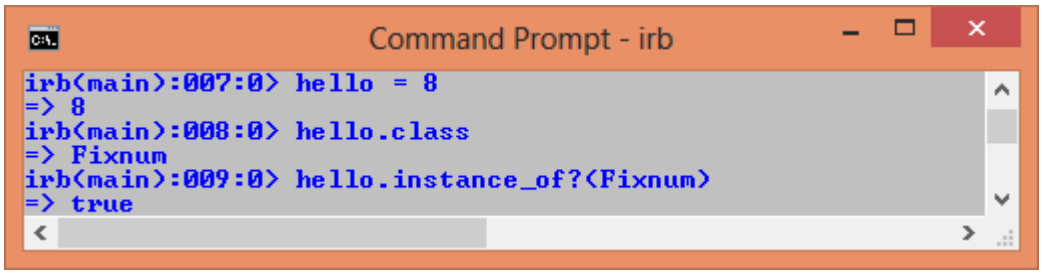


k) Assigned Variable

hello = 8 and then check hello with *class*

In this example, the **hello** variable is assigned the **integer** 8. The **hello** object, since it now stores an **integer** value, belongs to the **Fixnum** class.

Figure 1k – Assigned Variable

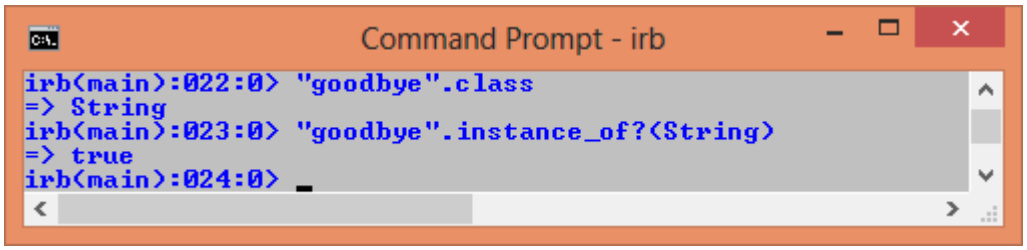


l) String Example 2

"goodbye"

This is another example of a **String**, as per "hello there big boy" in part a).

Figure 1l – String Example 2

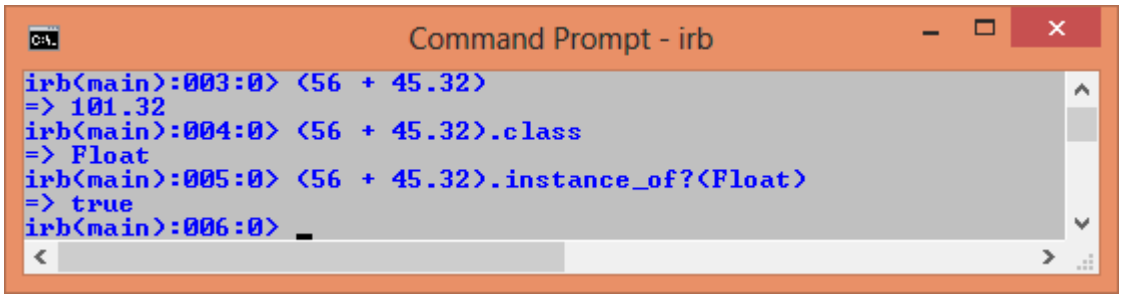


m) Floating Point Arithmetic

(56 + 45.32)

Arithmetic is taking place in this example. The **floating point number** "45.32" is added to the **integer** "56" and results in the **floating point number** "101.32". As shown in Figure 1m the result is an instance of the **Float** class.

Figure 1m – Floating Point Arithmetic

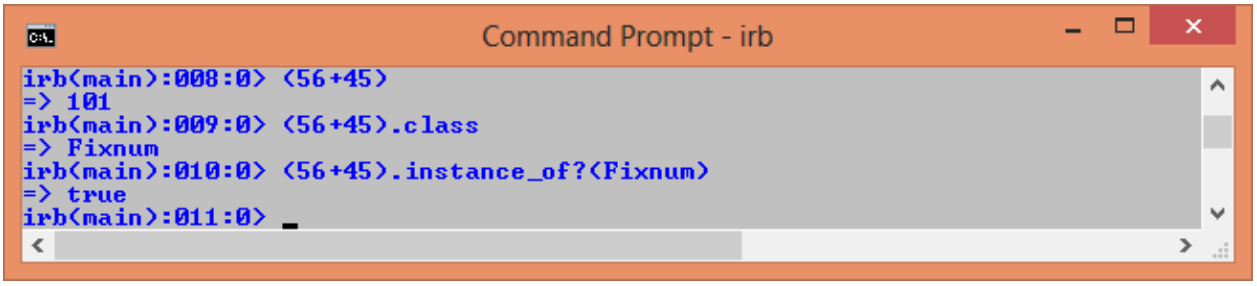


n) Integer Arithmetic

(56 + 45)

Arithmetic is taking place in this example similar to the previous example in part m). The **integer** "45" is added to the **integer** "56" and results in the **integer** "101.32". As shown in Figure 1n the result is an instance of the **Fixnum** class.

Figure 1n – Integer Arithmetic

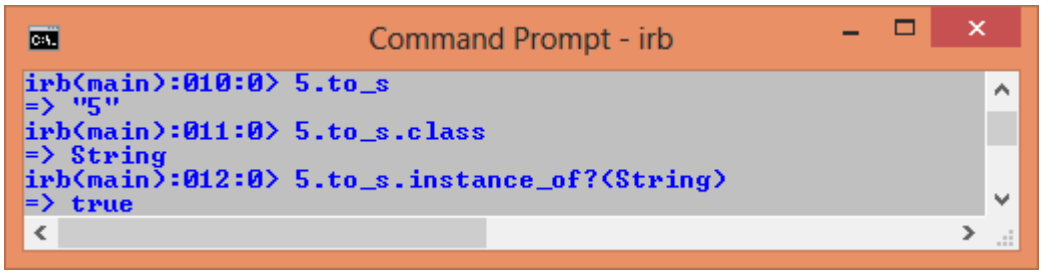


o) Converting Integer to String

5.to_s

The **to_s** method converts its receiver into a string. The **integer** 5 is the receiver, which is converted into the **string** "5".

Figure 1o – Converting Integer to String

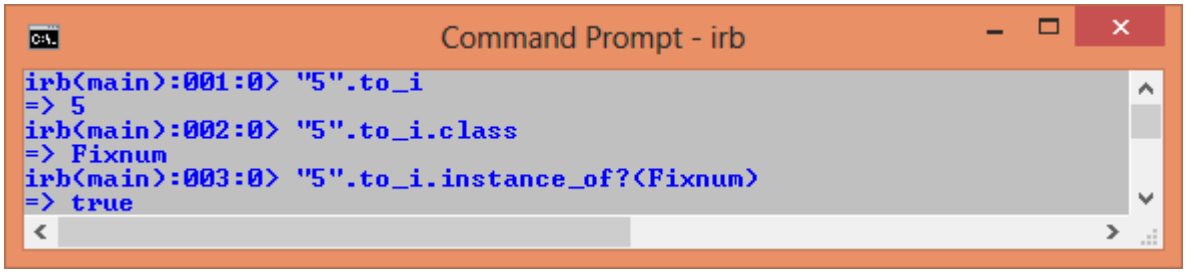


p) Converting String to Integer

"5".to_i

The **to_i** method converts its receiver into an integer. The **string** "5" is the receiver, which is converted into the **Fixnum** 5.

Figure 1p – Converting String to Integer

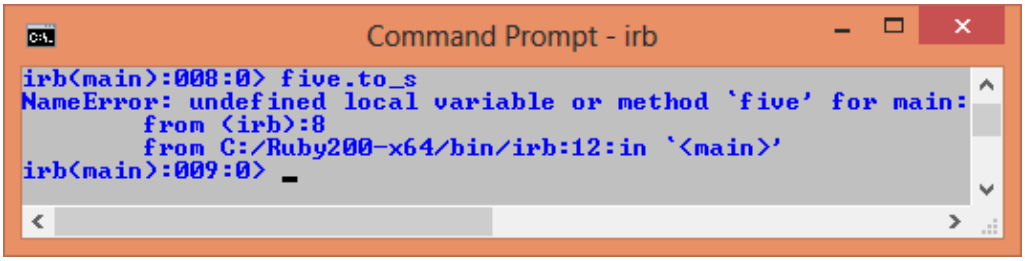


q) Attempting conversion of an undefined variable to a string

Five.to_s

In this example it appears that an attempt is taking place to convert an undefined variable to a string, which is an illegal operation.

Figure 1q – Illegal Conversion Attempt

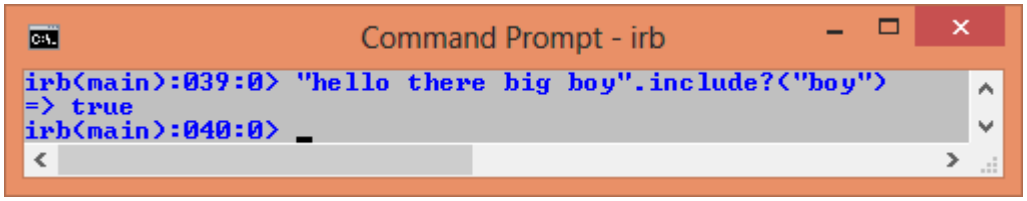


QUESTION TWO

a) String Inclusion Check

“hello there big boy”.include?(“boy”)

Figure 2a – String Inclusion Check



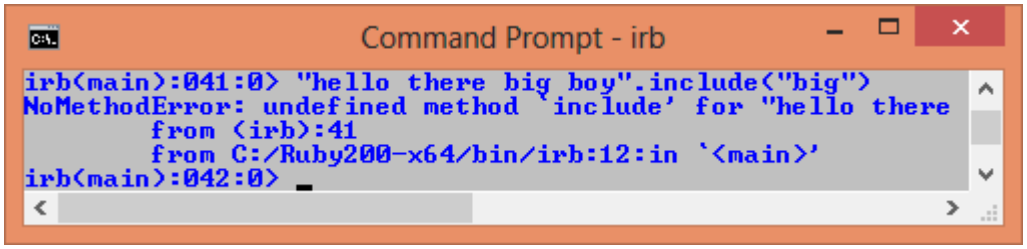
In Figure 2a example, the **include?** method is used to check whether the string “boy” is a substring of the string “hello there big boy”. The method returns true since “boy” exists within the main string.

b) String Inclusion Check (incorrect syntax)

“hello there big boy”.include(“ big”)

The above code attempts to implement the **include?** method (which was successfully implemented above in part a) with the wrong syntax (The **?** is missing). This results in an error as shown in Figure 2a.

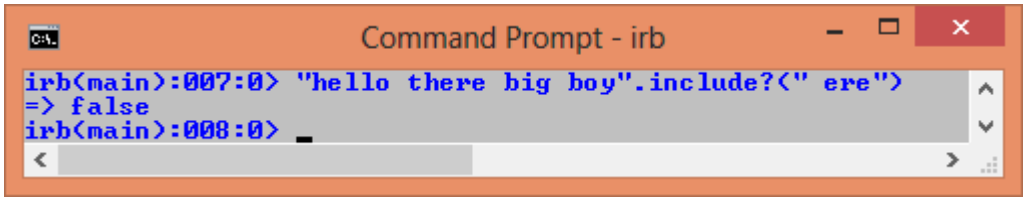
Figure 2a – String Inclusion Check (incorrect syntax)



c) String Inclusion Check (False)

“hello there big boy”.include?(“ ere”)

Figure 2c – String Inclusion Check (False)

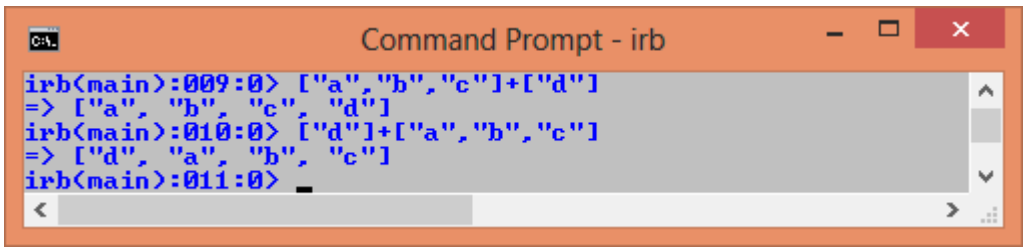


False is returned as the substring “ ere” is not found in the search string. Note the **whitespace character** at the beginning of the substring – if this character were deleted from the substring the method would return **true.**

d) Array Concatenation

What happens when you evaluate: [“a”, “b”, “c”] + [“d”]

Figure 2d – Array Concatenation

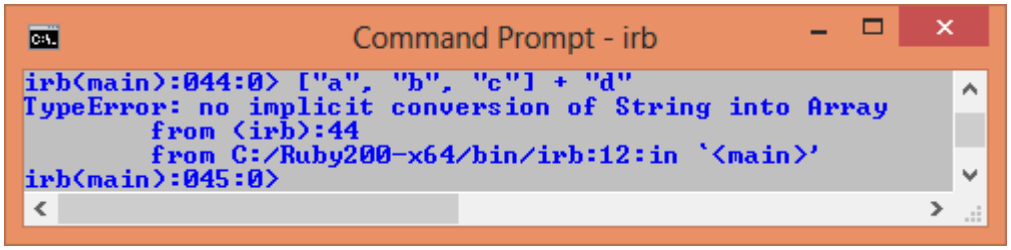


In figure 2d one array is concatenated to another array using the **+** operator (**+** operator discussed in Q1.f). As shown in the Figure 2d a new array is formed. If one reversed the order of the addition the new array will contain the letters in a different order- they will not be sorted for you alphabetically.

e) Illegal Array Concatenation attempt

What happens when you evaluate: [“a”, “b”, “c”] + “d”

Figure 2e – Illegal Array Concatenation attempt

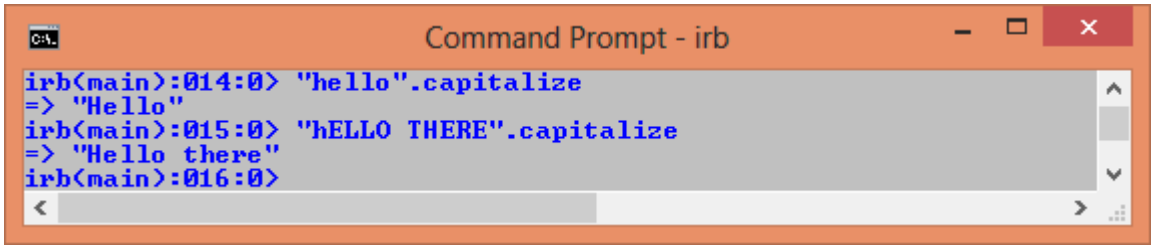


In Figure 2e an attempt is made to concatenate a **String** onto an **Array**. As shown in the error message this is an illegal operation since no implicit conversion of a String into an Array is allowed.

f) Capitalization

Is there an easy way to capitalise words, so “hello” becomes “Hello” ?

Figure 2f - Capitalize

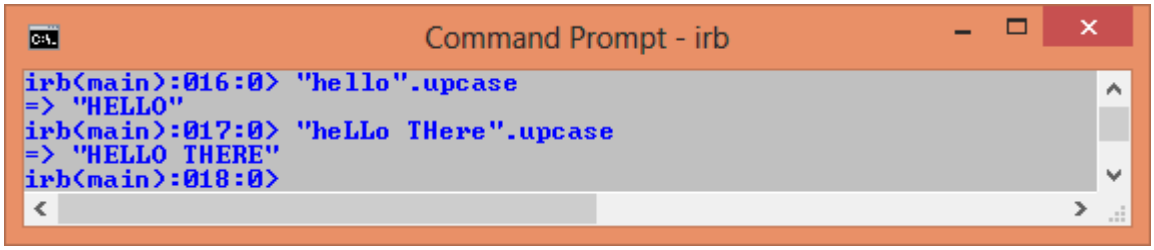


The **capitalize** method is used to capitalize the first letter of a **String Object**, with the remaining letters converted to lowercase, as shown in Figure 2f.

g) Upcase

In the same vein, make “hello” “HELLO”.

Figure 2g - Upcase



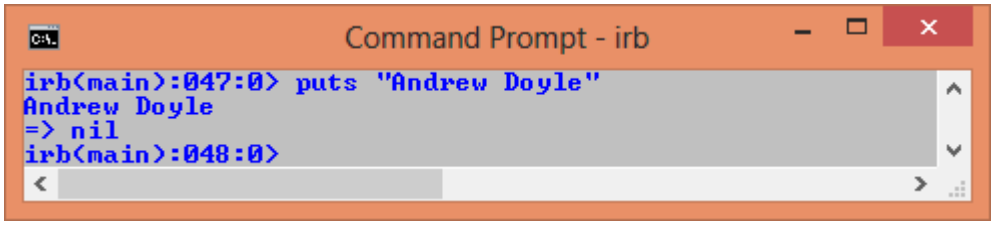
The **upcase** method converts all characters in a string to upper case, as shown in Figure 2g.

h) Printing to command line

Write a command to print out your name.

Ruby`s **puts** command sends a line of output to the console. As shown in Figure 2h, the **String** “Andrew Doyle” is sent to the **irb** command line.

Figure 2h – puts command



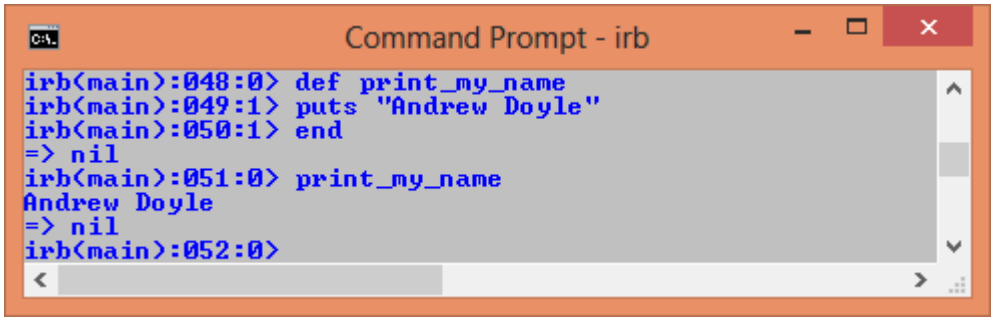
i) Defining a method

Write a method to print out your name.

Following on from the previous example, it is possible to define a method that will carry out the same task whenever it is called. As shown in Figure 2i, **def** starts the definition of the method, which is named as **print_my_name** (it could be called anything you want – although it is best to provide meaningful names which aid in the understanding of the code).

After the definition, the actions to be carried out when the method is called are outlined (in this example, the **String** "Andrew Doyle" will be put). To complete the method, **end** is called. As shown in Figure 2i calling the method results in the desired output.

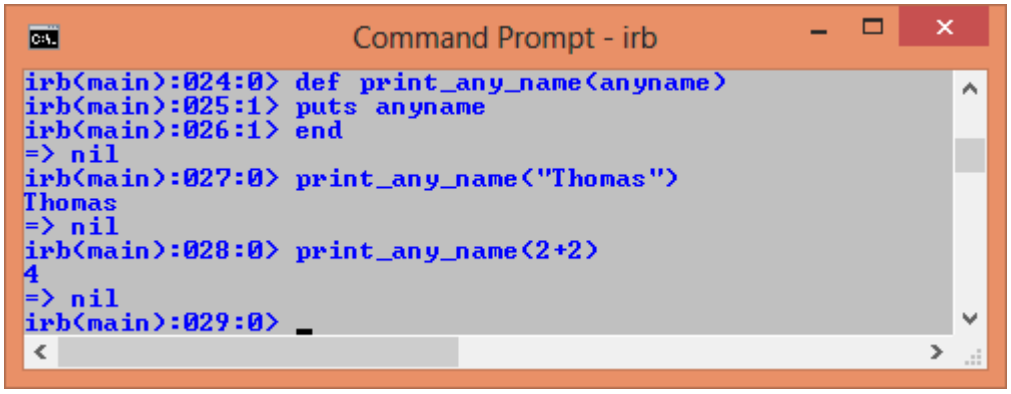
Figure 2i – defining a method



j) Method with Parameter

Write a method to print out any name.

Figure 2j – method with parameter

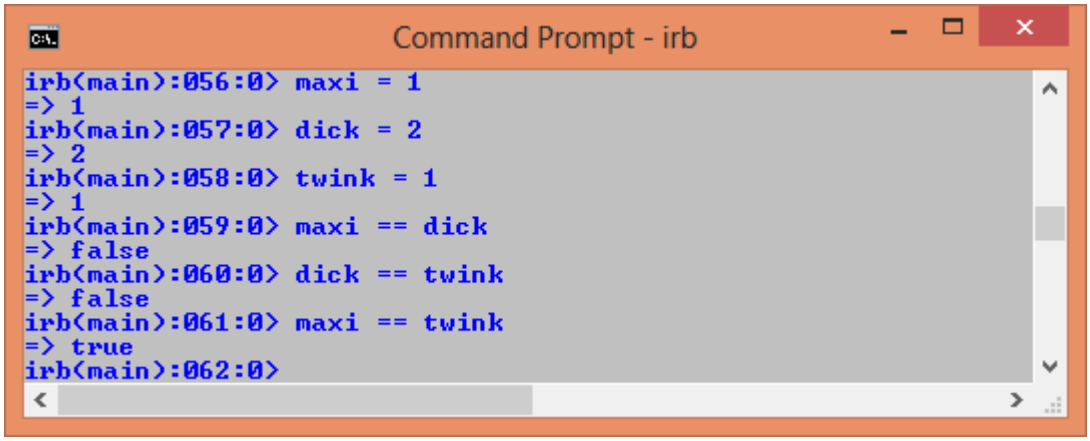


This method allows the user to specify any string to be output when the method is called. This is done in the definition by providing a parameter, which is "put" within the function. To call the method, the user enters the string which is assigned to the **anyname** variable passed into the function. As shown in Figure 2j, you could also carry out other tasks in the parameter, such as arithmetic.

k) Assignment with equality tests

Set up the variables, maxi, dick and twink so that they are all assigned numbers but two of them are assigned to the same numbers. Then show with a series of equality tests which ones actually have the same value.

Figure 2k – Assignment and equality tests

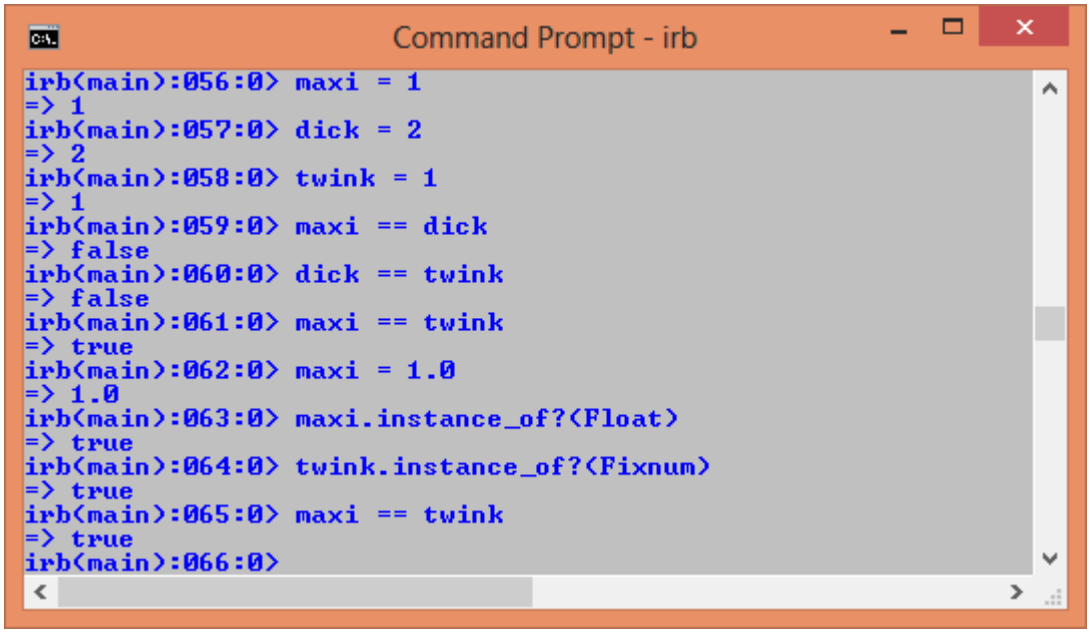


As shown in Figure 2k, the three variables are assigned values. They are then compared using the equality test syntax **==**. It is very important to differentiate between equality tests (using **==**) and assignment (using **=**). Maxi (which was assigned the **Integer** 1) and twink (which was assigned 2) are shown to be equal since the equality test returns **true**.

l) Comparing a Float and an Integer

If you change the variables with the same number to be a Float and Fixnum does it change the results of the equality tests?

Figure 2l – Comparing a Float and an Integer



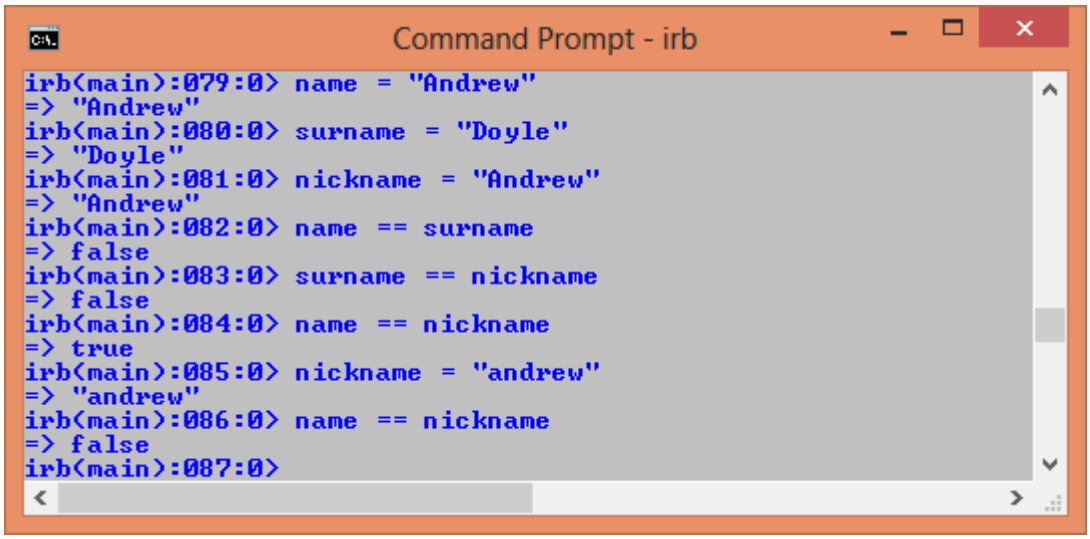
As shown in Figure 2l, assigning the variable **maxi** a **floating point number** does not change the result of the equality test.

m) String equality test

Do a version of these test using strings rather than numbers.

Figure 2m demonstrates how the equality tests can be carried out on strings in the same manner as the tests were carried out on numbers. As a further test, the first character of one variable was changed from a capital letter to a lowercase letter. This changes the result of the equality test from previously being **true**, to now returning **false**.

Figure 2m – equality tests on strings

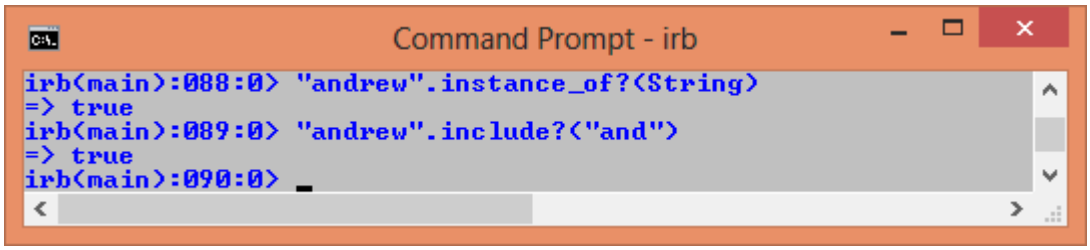


QUESTION 3

What's a predicate?

A method that returns true or false (i.e. a **Boolean** result) is called a **predicate**. As shown in Figure 3 below, the **instance_of?** And **include?** methods (used earlier in this paper) are tests that return a **Boolean** true or false result.

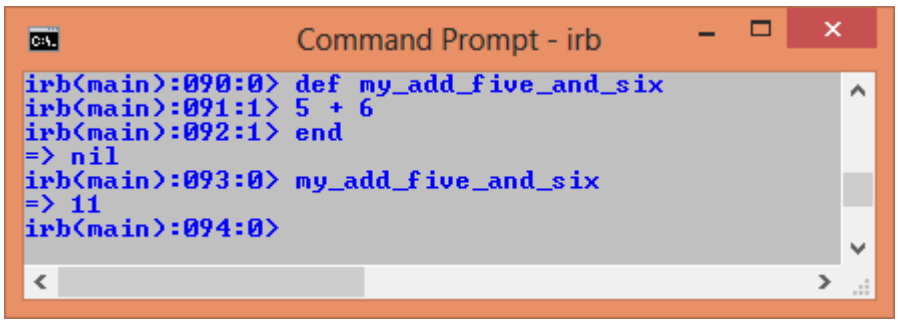
Figure 3 – Example of a predicate



QUESTION 4

Define your own adding method that always adds 5 and 6 together. So, my_add_five_and_six => 11.

Figure 4 – Method to perform addition

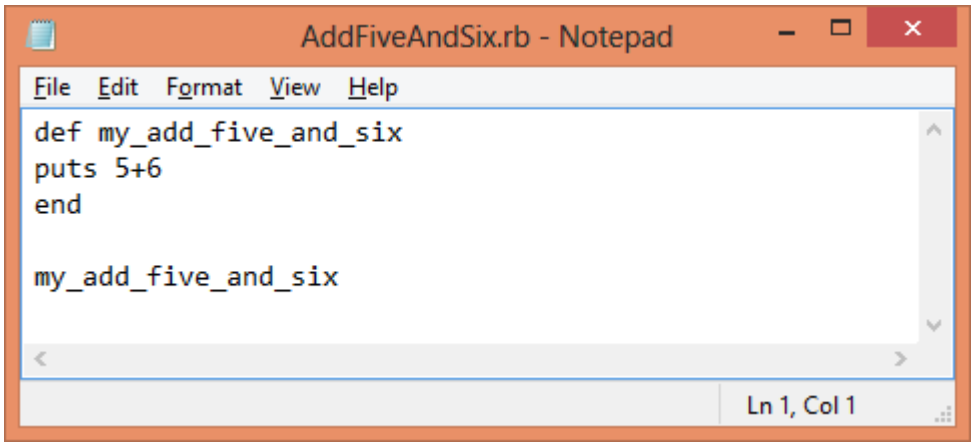


This method is declared using **def** following by the chosen method name. The addition is specified on the next line. The method is completed using **end**. As shown in Figure 4, calling the method returns the result of the addition.

QUESTION 5

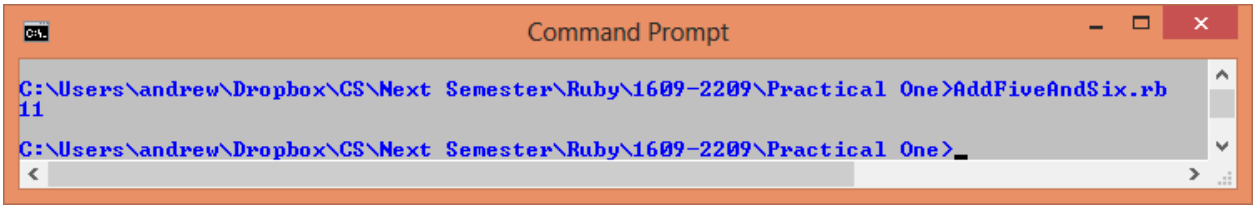
Put this defined method in a file and call it using the ruby command outside of irb.

Figure 5.1 – Method Defined in Ruby File



The method is defined and called in a **.rb** file as shown in Figure 5.1. This is similar to how it was completed in **irb** with one major exception: one must use the **puts** command within the method. To call the ruby file outside of irb in the command line, one must navigate to the directory where the file was stored and enter in the file name (including ruby`s **.rb** extension); as shown in Figure 5.2.

Figure 5.2 – Running Ruby File from Command Prompt



REFERENCES

1. Cooper, P. (2009). Beginning Ruby From Novice to Professional, Second Edition. New York, United States of America: Apress.

2. Ruby-doc.org (n.d). Fixnum. *Ruby-doc.org*. Retrieved 20th September 2013, from http://www.ruby-doc.org/core-2.0.0/Fixnum.html.

3. Ruby-doc.org (n.d). Object. *Ruby-doc.org*. Retrieved 21st September 2013, from http://www.ruby-doc.org/core-2.0.0/Object.html#method-i-class.

LIST OF FIGURES