|  |
| --- |
| Marc Laffan (15202421) |
| Exploring Programming in Ruby |
| Practical 1 |

|  |
| --- |
| Marc Laffan  9-17-2015 |

**1.a** This data is of the String class:

irb(main):010:0> "hello there big boy".class

=> String

Which we can prove using the “instance\_of?” method:

irb(main):009:0> "hello there big boy".instance\_of?(String)

=> true

This is due to the data being wrapped in double quotes.

**1.b** This data is of the Fixnum class:

irb(main):011:0> 56.class

=> Fixnum

Which we can prove using the “instance\_of?” method:

irb(main):013:0> 56.instance\_of?(Fixnum)

=> true

**1.c** This data is of the Float class:

irb(main):014:0> 34.00.class

=> Float

Which we can prove using the “instance\_of?” method:

irb(main):015:0> 34.00.instance\_of?(Float)

=> true

**1.d** This data is of the Float class:

irb(main):016:0> 0.222222354454365.class

=> Float

Which we can prove using the “instance\_of?” method:

irb(main):017:0> 0.222222354454365.instance\_of?(Float)

=> true

**1.e** This data is of the Array class due to the Array syntax used:

irb(main):019:0> ["a", "b", "c"].class

=> Array

Which we can prove using the “instance\_of?” method:

irb(main):020:0> ["a", "b", "c"].instance\_of?(Array)

=> true

**1.f** This data is not of any class, as it is an operator. :

irb(main):026:0> +.class

SyntaxError: (irb):26: syntax error, unexpected '.'

+.class

^

from C:/Ruby200-x64/bin/irb:12:in `<main>'

Similarly, the “instance\_of?” method returns another syntactical error:

irb(main):027:0> +.instanceOf(String)

SyntaxError: (irb):27: syntax error, unexpected '.'

+.instanceOf(String)

^

from C:/Ruby200-x64/bin/irb:12:in `<main>'

**1.g** This data is not of any class, but Ruby interprets it as an uninitialized constant due to it beginning with an uppercase letter:

irb(main):028:0> PI.class

NameError: uninitialized constant PI

from (irb):28

from C:/Ruby200-x64/bin/irb:12:in `<main>'

We receive similar results with the “instance\_of?” method:

irb(main):029:0> PI.instanceOf(Fixnum)

NameError: uninitialized constant PI

from (irb):29

from C:/Ruby200-x64/bin/irb:12:in `<main>'

**1.h** This data is of the Float class. The variable itself is a static constant variable, meaning that it never changes and is stored on a class basis, rather than an instance basis:

irb(main):031:0> Math::PI.class

=> Float

We can prove that the PI constant is of the Float type using the “instance\_of?” method:

irb(main):032:0> Math::PI.instance\_of?(Float)

=> true

**1.i** This data is not of any class, but Ruby interprets it as a local variable/method due to it beginning with an lowercase letter:

irb(main):033:0> add.class

NameError: undefined local variable or method `add' for main:Object

from (irb):33

from C:/Ruby200-x64/bin/irb:12:in `<main>'

The “instance\_of?” method has trouble making sense of the data due to it not being initialised. If we wrapped the data in quotes, it would be interpreted as a String:

irb(main):035:0> add.instance\_of?(String)

NameError: undefined local variable or method `add' for main:Object

from (irb):35

from C:/Ruby200-x64/bin/irb:12:in `<main>'

**1.j** Similarly to 1.i, without being wrapped in quotes this data is interpreted as a local variable/method:

irb(main):036:0> hellow.class

NameError: undefined local variable or method `hellow' for main:Object

from (irb):36

from C:/Ruby200-x64/bin/irb:12:in `<main>'

And again, the “instance\_of?” method returns an error:

irb(main):038:0> hellow.instance\_of?(String)

NameError: undefined local variable or method `hellow' for main:Object

from (irb):38

from C:/Ruby200-x64/bin/irb:12:in `<main>'

**1.k** In this case, we first assign a value to the “hello” variable, then ask for its class type. As the variable will have a value assigned to it, the methods work as they should have originally:

irb(main):039:0> hello = 8

=> 8

irb(main):040:0> hello.class

=> Fixnum

The “instance\_of?” method shows us the value of the number assigned to the “hello” variable, which is Fixnum for the number 8:

irb(main):003:0> hello.instance\_of?(Fixnum)

=> true

**1.l** This data is of the class String:

irb(main):004:0> "goodbye".class

=> String

Which we can prove using the “instance\_of?” method:

irb(main):005:0> "goodbye".instance\_of?(String)

=> true

**1.m** This data is first evaluated by Ruby to the new value of 101.32:

irb(main):007:0> (56 + 45.32)

=> 101.32

101.32 is of class Float:

irb(main):006:0> (56 + 45.32).class

=> Float

Which we can prove using the “instance\_of?” method:

irb(main):008:0> (56 + 45.32).instance\_of?(Float)

=> true

**1.n** Similar to the 1.m, the data is first evaluated by Ruby:

irb(main):009:0> (56 + 45)

=> 101

However this time, no fractions are involved. The class has changed from a Float to a Fixnum:

irb(main):010:0> (56 + 45).class

=> Fixnum

Which we can prove using the “instance\_of?” method:

irb(main):011:0> (56 + 45).instance\_of?(Fixnum)

=> true

**1.o** This data is originally of the Fixnum class, but the “to\_s”, or “to string” method changes the data type to that of a string. Note the quote-marks on the results from the first example:

irb(main):012:0> 5.to\_s

=> "5"

irb(main):013:0> 5.to\_s.class

=> String

We can again prove the data types accuracy using the “instance\_of?” method:

irb(main):014:0> 5.to\_s.instance\_of?(String)

=> true

**1.p** This data is doing the opposite of the previous example. Instead of converting the Fixnum to a string, we are converting a string to a Fixnum (Again, note the quotation marks). The end results is a Fixnum with a value of 5:

irb(main):015:0> "5".to\_i

=> 5

irb(main):016:0> "5".to\_i.class

=> Fixnum

Which we can prove using the “instance\_of?” method:

irb(main):017:0> "5".to\_i.instance\_of?(Fixnum)

=> true

**1.q** This data is not of any class, but Ruby interprets it as a local variable/method due to it beginning with an lowercase letter:

irb(main):018:0> five.to\_s

NameError: undefined local variable or method `five' for main:Object

from (irb):18

from C:/Ruby200-x64/bin/irb:12:in `<main>'

irb(main):019:0> five.to\_s.class

NameError: undefined local variable or method `five' for main:Object

from (irb):19

from C:/Ruby200-x64/bin/irb:12:in `<main>'

irb(main):020:0> five.to\_s.instance\_of?(String)

NameError: undefined local variable or method `five' for main:Object

from (irb):20

from C:/Ruby200-x64/bin/irb:12:in `<main>'

If the data was wrapped in quotation marks, the “instance\_of?” and “class” methods would work correctly.

**2.a** The value returned is a Boolean value of “true”, as the “includes?” predicate tests to see if the String includes the text “boy”, which it does.

**2.b** This line of code does not include a question mark (?), so the method does not work as predicates usually contain a question mark. If any part of a method signature is missing, Ruby will not recognise the method.

**2.c** This line of code does work, but returns “false”. This is due to the space preceding the “ere”, as Ruby is case sensitive when checking if a character sequence is contained within a String. As "hello there big boy" does not contain “ ere” anywhere, false is returned.

**2.d** The String Array containing the string “d” is added to the String Array that already contains the Strings “a”, “b” and “c”. This is very similar to concatenation for strings using the plus (+) operator.

**2.e** In this case, we are actually trying to add a string to an array object. Despite the array being of type String, we cannot add a String to a String Array in the literal sense. We can insert a String into a String Array (See previous example), but we cannot add them in the same sense. They are nnot compatible data types.

**2.f** Yes, we can use the “capitalize!” String method to convert the first letter of the String to uppercase:

irb(main):030:0> "hello".capitalize!

=> "Hello"

**2.g** Yes, we can use the “upcase” String method to convert the String entirely to uppercase:

irb(main):031:0> "hello".upcase

=> "HELLO"

**2.h**

p "Marc Laffan"

**2.i**

def name

p "Marc Laffan"

end

name

**2.j**

def name(yourname)

p yourname

end

name "Test"

**2.k**

maxi = 1

dick = 1

twinko = 3

p maxi == dick

p maxi == twinko

p dick == maxi

p dick == twinko

p twinko == maxi

p twinko == dick

C:\Users\Marc\Downloads>ruby test1.rb

true

false

true

false

false

false

**2.l** No, although the type of the data has changed, the numeric values themselves are still the same. In other words 1.0 is the same value as 1:

C:\Users\Marc\Downloads>ruby test1.rb

true

false

true

false

false

false

3

1.0

1

maxi = 1

dick = 1.0

twinko = 3

p maxi == dick

p maxi == twinko

p dick == maxi

p dick == twinko

p twinko == maxi

p twinko == dick

p twinko

p dick

p maxi

**2.m**

C:\Users\Marc\Downloads>ruby test1.rb

true

false

true

false

false

false

"Twinko"

"Dick"

"Dick"

maxi = "Dick"

dick = "Dick"

twinko = "Twinko"

p maxi == dick

p maxi == twinko

p dick == maxi

p dick == twinko

p twinko == maxi

p twinko == dick

p twinko

p dick

p maxi

**3.** A predicate is a method in Ruby that ends with a question mark and returns either a true or false value. Examples include “instance\_of?”.

**4.**

def my\_add\_five\_to\_six

p 5+6

end

my\_add\_five\_to\_six

**5.** Please see attached file for the actual file (test1.rb) I used for this task:

C:\Users\Marc\Downloads>ruby test1.rb

11