RUBY

#### WHY

* powerfull syntax
* easily syntax
* dynamic typing
* multi environment
* interpretive language

#### VARIABLES

* NIL = object = nothing
* var.nil? => true / false
* gets => always string, confert to integer / float
* print => same line no new line
* puts => with extra new line

#### CLASS

Class Employee

Def initialize(name, pay)

@name = name

@pay = pay

end

def to\_S

return Emplyee

end

end

#### STATEMENTS

### IF

if

elsif

end

### CASE

case grade

when “A”

when”B”,”C”

else

end

### ITERATOR

colors.each {|color| puts color}

### WHILE and UNTILL loops

while number !=-1

….

end

### FOR loops

Range = > (1..10)

For i in (1..5)

For num in nums

### Operators

+ - \* / % \*\* && || and or

### Objects Initialization and Self Objects

### Initialisation

Example: time.rb

class Time

def initialize (hr, min, sec)

@hour = hr

@minutes = min

@seconds = sec

end

def initialize(hr,min)

@hour = hr

@minutes = min

@seconds = 0

end

def initialize(min)

@hour = 0

@minutes = min

@seconds = 0

end

def initialize

@hour = 0

@minutes = 0

@seconds = 0

end

end

tehTime = Time.new(3,2,0)

anotherTime = Time.new(12,1)

someTime = Time.new(58)

blankTime = Time.new

### SELF OBJECTS

Example: student.rb

Class Student

Attr\_reader :name, :id, :grades

Def initialize(name, id, grades)

@name = name

@id = id

@grades = grades

end

def equals?(aStudent)

if self.name == aStudent.name && self.id == aStiudent.id then

return true

else

reurn false

end

end

end

compare.rb

require\_relative ‘student.rb’

s1 = Student.new(“Jane Doe”, ”12345”, [90])

s1 = Student.new(“Jane Doe”, ”12346”, [90])

if s1.equals?(s2) then

puts “same student”

else

puts “different student”

end

### Completing Ruby Applications

### Comments

# single line comment at begin or at the end of a line

=begin

this is a multiline

comment in Ruby

=end

### Code organization

Rcommended build of Ruby program:

Sample.rb

=begin

Program name :

Author :

Description :

=end

# require statements

# include statements

# module definition

# class defentition

# main program section

# testing code

### RubyGEMS

Packages of libraries

Install Gem rake

*Gem install –r rake*

Controlling good install rake

Rake –version

Gem overview:

*gem search –r*

Gem installed

*Gem list –local*

### MINI TEST

testconvert.rb

require ‘minitest/autorun’

class Convert

def ftoc(f)

return (5.0/9.0) \* (f – 32.00)

end

end

class TestConvert < MiniTest::Unit::Testcase

def setup

@c = Convert.new

end

def test\_ftoc\_convert

assert\_equal 0, @c.ftoc(32)

end

end

### Distribution Ruby Application

Install

*Gem install ocra*

Convert.rb

print “Enter a fahrenheit temperatue : “

temp = gets.chomp

temp = temp.to\_i

puts (5.0/9.0) \* (temp – 32.00)

run

*ruby converttemp.rb*

make executable

*ocra converttemp.rb*

Run

*converttemp*

**Exercise: Use Basic Ruby Structures to Create a Program**

**Solution**

The model solution to this task is given in the Analysis section that follows. You can use the **Print** feature at the top of the page to create a hard copy.

**Analysis**

The solution for each of the tasks in the exercise is analyzed in the following table.

A sample solution to the exercise is given here Solution

| **Checkpoints** | |
| --- | --- |
| **Number** | **Details** |
| **1** | The initialize method can be used when creating new instances of the Cardclass. The price and note parameters are assigned default values within the method signature which are then assigned to the class variables.  By calling .to\_f the price is stored as a float type.  class Card     attr\_reader :name, :price, :action, :note      def initialize(name, action, price=nil, note="")         @name = name         @price = price.to\_f         @action = action         @note = note     end end |
| **2** | The begin statement starts a block that will handle any exceptions that occur. It should surround the command to open the file passed in by the file name.  The rescue statement is where you handle particular exceptions, in this case the puts statement will print out the warning message.  def loadData(filename)     begin         file = open("#{filename}")     rescue Exception => e           puts "Error loading file #{filename}"       end end |
| **3** | An if statement can use the file variable to determine if it is true or false. Within the if statement block a puts statement will send the output that the file was opened. Then you call the readInput method and pass the file as a parameter, followed by a call to the displayToScreen method.  file = open("#{filename}") if file     puts "File opened successfully"     readInput(file)     displayToScreen end |
| **4** | A while loop can be used to read each line of a file when calling the .getsmethod.  def readInput(file)     first\_line = true     cards = []     while line = file.gets      end end |
| **5** | An if statement can check to see if first\_line equals true and if so, you assign the value to the variable @user. You then need to set the first\_line variable to false.  def readInput(file)     first\_line = true     @cards = []     while line = file.gets         if first\_line == true             @user = line             first\_line = false         end     end end |
| **6** | You add an else statement to handle when first\_line is false. Within the elseblock a case statement is used to handle cases when the array count is 4, 3, or 2.  Another else statement within the case block will print an error to the screen.  The Card class can be instantiated using 2, 3, or 4 parameters as the method will assign default values when any parameters are missing.  while line = file.gets     if first\_line == true         @user = line         first\_line = false     else         properties = line.split(',')         case properties.count         when 4             @cards << Card.new(properties[0],properties[1],properties[2],properties[3])         when 3             @cards << Card.new(properties[0],properties[1],properties[2])         when 2             @cards << Card.new(properties[0],properties[1])         else             puts "Error processing line"           end     end end |
| **7** | The puts command can be used to display the user name on the first line.  The next line is shown with a second puts command that separates the headings using \t to represent a tab.  You can call 60.times to perform a command that many times and in this case the print command will print the – character 60 times on the same line. Printing a \nwill then cause any following print statements to occur on the new line.  The .each method will allow you to iterate through each card in the @cards array.  Then you need to print out the properties of the individual card. When using a putsstatement, within the quotations you can refer to an instance variable by enclosing them with # { }.  def displayToScreen     puts @user     puts "Action\tPrice\tName\t\tNotes"     60.times{ print "-"}     print "\n"     @cards.each do |card|         puts "#{card.action}\t#{card.price}\t#{card.name}\t#{card.note}"     end end |

If you'd like to review the scenario and assignment, return to the [**exercise**](https://cdnlibrary.skillport.com/courseware/Content/cca/pg_ruby_a01_it_enus/output/t107/lp1/pages/me.html) page.

**Table of Contents**

| [**Top of page**](https://cdnlibrary.skillport.com/courseware/Content/cca/pg_ruby_a01_it_enus/output/t107/lp1/pages/solution.html#pagetop) |

| [**Solution**](https://cdnlibrary.skillport.com/courseware/Content/cca/pg_ruby_a01_it_enus/output/t107/lp1/pages/solution.html#section_1) |

| [**Analysis**](https://cdnlibrary.skillport.com/courseware/Content/cca/pg_ruby_a01_it_enus/output/t107/lp1/pages/solution.html#section_2) |

© 2014 Skillsoft Ireland Limited

| Print | [**Contents**](https://cdnlibrary.skillport.com/courseware/Content/cca/pg_ruby_a01_it_enus/output/t107/lp1/pages/solution.html#contents) |

**Exercise: Use Basic Ruby Structures to Create a Program**

**Solution**

The model solution to this task is given in the Analysis section that follows. You can use the **Print** feature at the top of the page to create a hard copy.

**Analysis**

The solution for each of the tasks in the exercise is analyzed in the following table.

A sample solution to the exercise is given here Solution

| **Checkpoints** | |
| --- | --- |
| **Number** | **Details** |
| **1** | The initialize method can be used when creating new instances of the Cardclass. The price and note parameters are assigned default values within the method signature which are then assigned to the class variables.  By calling .to\_f the price is stored as a float type.  class Card     attr\_reader :name, :price, :action, :note      def initialize(name, action, price=nil, note="")         @name = name         @price = price.to\_f         @action = action         @note = note     end end |
| **2** | The begin statement starts a block that will handle any exceptions that occur. It should surround the command to open the file passed in by the file name.  The rescue statement is where you handle particular exceptions, in this case the puts statement will print out the warning message.  def loadData(filename)     begin         file = open("#{filename}")     rescue Exception => e           puts "Error loading file #{filename}"       end end |
| **3** | An if statement can use the file variable to determine if it is true or false. Within the if statement block a puts statement will send the output that the file was opened. Then you call the readInput method and pass the file as a parameter, followed by a call to the displayToScreen method.  file = open("#{filename}") if file     puts "File opened successfully"     readInput(file)     displayToScreen end |
| **4** | A while loop can be used to read each line of a file when calling the .getsmethod.  def readInput(file)     first\_line = true     cards = []     while line = file.gets      end end |
| **5** | An if statement can check to see if first\_line equals true and if so, you assign the value to the variable @user. You then need to set the first\_line variable to false.  def readInput(file)     first\_line = true     @cards = []     while line = file.gets         if first\_line == true             @user = line             first\_line = false         end     end end |
| **6** | You add an else statement to handle when first\_line is false. Within the elseblock a case statement is used to handle cases when the array count is 4, 3, or 2.  Another else statement within the case block will print an error to the screen.  The Card class can be instantiated using 2, 3, or 4 parameters as the method will assign default values when any parameters are missing.  while line = file.gets     if first\_line == true         @user = line         first\_line = false     else         properties = line.split(',')         case properties.count         when 4             @cards << Card.new(properties[0],properties[1],properties[2],properties[3])         when 3             @cards << Card.new(properties[0],properties[1],properties[2])         when 2             @cards << Card.new(properties[0],properties[1])         else             puts "Error processing line"           end     end end |
| **7** | The puts command can be used to display the user name on the first line.  The next line is shown with a second puts command that separates the headings using \t to represent a tab.  You can call 60.times to perform a command that many times and in this case the print command will print the – character 60 times on the same line. Printing a \nwill then cause any following print statements to occur on the new line.  The .each method will allow you to iterate through each card in the @cards array.  Then you need to print out the properties of the individual card. When using a putsstatement, within the quotations you can refer to an instance variable by enclosing them with # { }.  def displayToScreen     puts @user     puts "Action\tPrice\tName\t\tNotes"     60.times{ print "-"}     print "\n"     @cards.each do |card|         puts "#{card.action}\t#{card.price}\t#{card.name}\t#{card.note}"     end end |

If you'd like to review the scenario and assignment, return to the [**exercise**](https://cdnlibrary.skillport.com/courseware/Content/cca/pg_ruby_a01_it_enus/output/t107/lp1/pages/me.html) page.

**Table of Contents**

| [**Top of page**](https://cdnlibrary.skillport.com/courseware/Content/cca/pg_ruby_a01_it_enus/output/t107/lp1/pages/solution.html#pagetop) |

| [**Solution**](https://cdnlibrary.skillport.com/courseware/Content/cca/pg_ruby_a01_it_enus/output/t107/lp1/pages/solution.html#section_1) |

| [**Analysis**](https://cdnlibrary.skillport.com/courseware/Content/cca/pg_ruby_a01_it_enus/output/t107/lp1/pages/solution.html#section_2) |

© 2014 Skillsoft Ireland Limited

OEFENING 2

| **Checkpoints** | |
| --- | --- |
| **Number** | **Details** |
| **1** | In the new file ScientificAnalysis.rb, you create the basic structure of a Ruby application code file.  In the sample solution, the multi-line header is,  =begin Program name: ScientificAnalysis.rb Author: <your name> Description: Program for scientfic analysis =end  The individual lines showing the requires statement, module, class, main program, and testing code sections are commented:  # requires statements # module definitions # class definitions # main program section # testing code |
| **2** | The any\_results\_below\_freezing method should have a ? to indicate that it returns a Boolean result.  def any\_results\_below\_freezing? (results)  You can use the .each\_value method to iterate over the values in the hash.  Results.each\_value do |result|  When determining if the response should change, the ternary operator can be used to set the variable, which would be similar to this code:  response = result\_below\_freezing?(result) ? true : response |
| **3** | In order to create a private method, ensure the private keyword is on a line above the method. All methods below this line within the module would have a private scope.  private def result\_below\_freezing?(value)     return value < 0 end |
| **4** | Use the attr\_reader and attr\_writer accessors to create the three properties for the class. The include keyword is used to add the Analyze module to the class as a mixin.  #class definitions  class CollectedData     attr\_reader :researcher\_name, :date\_entered, :data     attr\_writer :researcher\_name, :date\_entered, :data     include Analyze |
| **5** | The initialize method will be executed when you create an instance of the class. If the code does not pass in a value for the date\_entered parameter, it can be assigned a default value of DateTime.now in the method signature. Within the method the @data instance can be given a value with Hash.new or {}.      def initialize(researcher, date\_entered=DateTime.now)         @researcher\_name = researcher     @date\_entered = date\_entered         @data = Hash.new()     end  In order to use the DateTime.now call, the program requires the date module.  # requires statements require 'date' |
| **6** | The add method should take the value and assign it directly to the @data instance variable.  def add(value)     @data = value end |
| **7** | The puts statement will send the output to the screen.  #main program section  puts "Starting program" |
| **8** | You create an instance of the class  mydata = CollectedData.new("Matt White"). |
| **9** | The data must be entered in the form of a proper hash with key and value pairs. This code will pass the hash in the proper format to the add method.  mydata.add({"boiling water"=> 100, "frying pan"=> 75, "pencil"=> 20}) |
| **10** | The puts statement can be used to display the result of  mydata.any\_results\_below\_freezing?(mydata.data) |
| **11** | In order to run the unit tests, this line must be below the header section of the file:  require 'minitest/autorun' |
| **12** | The < character denotes that a class inherits another class which gives this class signature:  class TestCollectedData < MiniTest::Unit::TestCase |
| **13** | The setup method will call CollectedData.new and pass the string "John Smith". Then the add method is called to add the hash.  def setup     @data1 = CollectedData.new("John Smith")     @data1.add( {"Iceberg1"=>-5, "Deep freeze1" =>-10, "Driveway" => 35}) end |
| **14** | The test\_below\_freezing method must use an assert\_equal call that tests the equality of the value true to the result of a call to the any\_results\_below\_freezing? method.  def test\_below\_freezing     assert\_equal true, @data1.any\_results\_below\_freezing?(@data1.data) end |
| **15** | To package the code into an executable, you can use the ocra gem by running ocra ScientificResults.rb at the command prompt. |

### WORKING WITH DATA

### Higher Math

Absolute value of -2 : -2.abs => 2

4.1.div(2) => 2

4.modulo(2) => 0

3.modulo(2) => 1

1.2.floor => 1

1.2.ceil => 2

1.5.round => 2

square route

Math.sqrt(9) => 3

2.to\_f => 2.0

1.4.to\_i => 1

### Random Numbers

rand

rand(5)

rand(10)

rand(100)

rand(60..100)

srand(123) (sead of random numbers)

for i in 1..10

puts rand(10)

end

rarray.be

numbers = []

for i in 0..9

numbers[i] = rand(60..100)

end

print numbers

puts numbers.sample

puts numbers.sample

### Advanced Arrays

nums1 = [1,2,3]

nums2 = [4,5]

Array together:

nums3 = nums1+nums2

Array add values:

nums3 << 6 << 7 << 8

find differences between 2 arrays:

nums1 – nums2

intersection

nums1 & nums2

union

nums1 | nums2

unique

nums1.uniq

push and pop

nums1.push 4 (add at end new value)

nums1.pop (remove last value)

### Regular Expressions

line = “the quick brown fox”

check if string conains ‘quick”

if (line =~ /quick/)

puts “line contains quick”

end

check begin of the line:

line =~/^quick/ (postion 0)

line = “123 Main”

check for numbers, fiirstplace

line =~/[0-9]/ => 0

check for alfabet

line =~/[a-z]/ => 4

substitute (only first)

line.sub(“o”, “O”) => the quick brOwn fox (only first)

substitute (all)

line.gsub(“o”, “O”) => the quick brOwn fOx (all)

### Accessors and Virtual Attributes

time.rb

class Time

attr\_reader :hours, :minutes

attr\_writer :hours, :minutes

def time\_in\_minutes

@hours \* 60 + @minutes

end

def time\_in\_minutes=(new\_minutes)

mins = new\_minutes.to\_i

if mins > 59

@hours = mins / 60

@minutes = mins – 60

else

@hours = 0

@minutes = mins

end

end

end

theTime =Time.new

theTime.hours = 3

theTime.minutes = 45

puts theTime.hours, theTime.minutes

theTime.time\_in\_minutes = 30

puts theTime.time\_in\_minutes

### Working with Methods

### Overriding Methods