ARTIFICIAL INTELLIGENCE

SUPINFO International University

Sign of Success

3AIT – Graded Exercise

Functional programming

The analysis of a guided world

Functional programming
Lisp
Python

Version 1.0
Last update: 04/19/2018
Use: Students/Staff
Author: Cyril Alexandre Pachon

Table of contents

1 PREAMBLE: GENERAL INSTRUCTIONS

Your folder should conform to the following:

1. The folder should be named: **[3AIT]-IDOpenCampus-NameOfCampus-SURNAME-FirstName-TP**.
2. In the folder, files **.pdf**, **.l** and **.py** are acceptable
3. Use an archive tool to **ZIP** your folder: **[3AIT]-IDOpenCampus-NameOfCampus-SURNAME-FirstName-TP.zip**, (e.g. **[3AIT]-123456-Paris-Lupin-Marc-TP.zip**)
4. Your archive must be placed in website: **sce.sad.supinfo.com**.

For this examination, you can use the support slides (.ppt, LABs), pen and paper (these are not provided by the school) for your draft response to the questions. You can use **LispWorks** and **Python** Tools. You are **NOT ALLOWED** to use **Internet** for this examination.

If you are found cheating by your supervisor your examination session will be **CANCELLED**. Your transcript will be marked as “cheater”.

2 THE FUNCTIONAL PROGRAMMING (6 POINTS)

Question1.1 (1 point): With selectors or constructors, write the recursive function named **NB** which counts the number of atoms in the list of atoms. Specify all the elements of your productions.

We have the following list L = ((1 2 3) (4 5 6) (7 8 9))

Question1.2 (3 points): With selectors or constructors, write a recursive function named **VF** which checks if a list contains sub lists (only depth 1, like list L) containing the same number of atoms in each sub list (like list L). Specify all the elements of your productions.

Question1.3 (2 points): With selectors or constructors, write a recursive function named VLA which checks if the number of atoms in each sub list of L is equal to the number of sub lists in L. Specify all the elements of your productions.

3 THE LISP PROBLEM (8 POINTS)

An Expert System contains the following expression:

```
(and (defun E()
(lambda (A B)
(cond
  ((< A 20) (expt B 10))
  (T  "Confirm me the result obtained?"))))
  (funcall (E) *read-base* *print-base*))
```

Question 2.1 (1 point): Give the result of the expression.

Question2.2 (2 points): Explain in detail (line by line) the given expression. How is the result produced?

Question2.3 (1 point): Make single change in the given expression such as the character string is interpreted and displayed. It is not required to redo the expression, but just to modify one instruction of the expression to obtain the result: "Confirm me the result obtained?". Explain your choice.

Question 2.4 (1.5 points): Write a Lisp function named **F** without argument which includes the initial expression to correspond to the following results:

(funcall (F) 0) → 0
(funcall (F) 5) → 9765625
(funcall (F) 3) → 59049
(funcall (F) 19) → 6131066257801
(funcall (F) 20) → "Confirm me the result obtained?"
"Yes it is 20"
(funcall (F) 25) → "Confirm me the result obtained?"
"Yes it is 25"

Question2.5 (1.5 points): The Lisp **expt** function was given in the initial expression. Now we have to do it. Write a Lisp recursive function named **expt2** which produces the same interpretation as **expt**. Define all the settings you will use.

Question 2.6 (1 point): Modify the function **F** by now including **expt2**. Your function **F** must be checked. Give a set of tests to check the functioning.

4 THE PYTHON SYTEM (6 POINTS)

We have the following Python function **F**:
F = lambda L : L.reverse()

Question 3.1 (1 point): What does the function **F** do, how to edit the result?

You have the following Python functions:
car = lambda liste: liste[0]
cdr = lambda liste: liste[1:]
membre = lambda a, liste : a in liste

Question 3.2 (2 points): With the functions given, and keeping the principle of functional programming, write a function in Python named **EG** with 2 arities, which eliminates all the values on the left of a given value.

>>> EG('a', ['d','c','b']) → ['d', 'c', 'b']
>>> EG('a', ['d','c','a','b']) → ['a', 'b']
>>> EG('1', ['d','a','b']) → ['d', 'a', 'b']

Question 3.3 (3 points): You have the results of a suite **S**:

>>> S([]) → 1
>>> S(['1']) → 1
>>> S(['1','2']) → 2
>>> S(['1','2','3']) → 2
>>> S(['1','2','3','4']) → 24
>>> S(['1','2','3','4','5']) → 24
>>> S(['1','2','3','4','5','6']) → 720
>>> S(['1','2','3','4','5','6','7']) → 720
>>> S(['1','2','3','4','5','6','7','8']) → 40320
>>> S(['1','2','3','4','5','6','7','8','9']) → 40320
>>> S(['1','2','3','4','5','6','7','8','9','10']) → 3628800
…….

Using the functions **car** and **cdr**, write in Python the function **S** with 1 arity, which give the result of the suite.