```
# -*- coding: UTF-8 -*-

##
## Data:  24-04-2011
##

from Util.Util import cls, stop, creditos
from Logic.Interpretadores import ExpressionInterpreter, InferenceInterpreter

def wiriHelp():
    ''' Funcaozinha de ajuda desse modulo do LogSent '''
    cls()
    print " REGRAS DE INFERENCIA: + ------------------------------ +"
    print
    print
    print "   (Modus Ponens)   (Modus Tollens)   (Silogismo Hip.)     "
    print "   [1] p -> q       [1] p -> q        [1] p -> q           "
    print "   [2] p            [2] !q            [2] q -> r           "
    print "       ------           ------            ------           "
    print "   [:] q            [:] !p            [:] p -> r           "
    print
    print
    print "   (Adicao)         (Simplificacao)   (Silogismo Dis.)     "
    print "   [1] p            [1] p & q         [1] p | q            "
    print "                                      [2] !p               "
    print "       -----            -----             ------           "
    print "   [:] p | q        [:] p             [:] q                "
    print
    print
    print "   (Conjuncao)      (Contraposicao)   (Resolucao)          "
    print "   [1] p            [1] p -> q        [1]  p | q           "
    print "   [2] q                              [2] !p | r           "
    print "       -----            --------          ------           "
    print "   [:] p & q        [:] !q -> !p      [:] q | r            "
    print
    print
    print "   (Dilema Const.)  (Dilema Dest.)                         "
    print "   [1] p -> q       [1] p -> q                             "
    print "   [2] r -> s       [2] r -> s                             "
    print "   [3] p | r        [3] !q | !s                            "
    print "       -----            --------                           "
    print "   [:] q | s        [:] !p | !r                            "
    print
    print
    print "   OBS: Sempre formule corretamente sua inferencia, pois o "
    print "  LogSent ainda nao possui um analisador sintatico.        "
    print
    print
    print " OPERADORES + ---------------------------------------- +"
    print
    print "   !   (not)"
    print "   &   (and)"
    print "   |   (or)"
```

```
    print "   *   (xor)"
    print "   ->  (implica)"
    print "   <-> (bi implica)"
    print
    print
    print " EXPRESSOES + ---------------------------------------- +"
    print
    print "   O LogSent opera com expressoes binarias, ou seja, o u-  "
    print "  suario deve explicitar as precedencias usando parenteses."
    print
    print "  CUIDADO:  - Formule bem a sua expressao para que nao haja"
    print "            erros no resultado, ja que o LogSent nao tem um"
    print "            analisador sintatico."
    print
    print "  Exemplos de proposicoes validas:                         "
    print
    print "    (p -> q) &  r                                          "
    print
    print "    !t <-> ((p & q) -> a)                                  "
    print
    print " + ------------------------------------------------------ +"
    print
    print
    stop()




def analisador():
    ''' Recebe premissas e retorna o nome do tipo de R.I.
        (Regra de Inferencia). '''

    sair = False

    while not sair:
        cls()
        print " (<) Main Menu [R]                              [A] Ajuda"
        print
        print " + Analisador de R.I. + ---------------------------- +"
        print
        print " Digite [ENTER] para iniciar o analisador              "
        print

        opcao = raw_input(" >> ")

        if opcao == '':
            cls()
            expInterpreter = ExpressionInterpreter()
            infEval = InferenceInterpreter()

            quant = int(input(" Quantas premissas tem a R.I.?\n >> "))
            print " \n" * 3
            premissas = []
```

```
            for i in range(quant):
                p = raw_input(" [" + str(i + 1) + "] ")
                premissas.append( expInterpreter.eval(p) )
            print "     --------"
            conclusao = expInterpreter.eval( raw_input(" [Q] ") )

            try:
                inferenceRule = infEval.identify(premissas, conclusao)

                print "\n"
                print " Resultado + ------------------------------------- +"
                print
                print " [" + inferenceRule.nome + "]"
                print
                stop()

            except Exception as e:
                print
                print " [ERRO] Problemas ao avaliar esta R.I.!"
                print "", e
                print
                stop()
                return


        elif opcao in ['R', 'r']:
            sair = True

        elif opcao in ['A', 'a']:
            wiriHelp()



def autocompletar():
    ''' Interface de usuario para a ferramenta de que completa regras de
        inferencia'''

    sair = False

    while not sair:
        cls()
        print " (<) Main Menu [R]                              [A] Ajuda"
        print
        print " + Completador de R.I. + ---------------------------- +"
        print
        print " Digite [ENTER] para iniciar                             "
        print

        opcao = raw_input(" >> ")

        if opcao == '':
            cls()
            expInterpreter = ExpressionInterpreter()
```

```
            infInterpreter = InferenceInterpreter()

            print " HINT: Deixe um campo vazio quando quiser omitir algo.\n"
            quant = int(input(" Quantas premissas tem a R.I.?\n >> "))
            print " \n" * 3

            premissas = []
            for i in range(quant):
                p = raw_input(" [" + str(i + 1) + "] ")
                if p == '':
                    premissas.append(None)
                else:
                    premissas.append( expInterpreter.eval(p) )

            print "     -----------"
            conclusao = expInterpreter.eval( raw_input(" [Q] ") )

            try:
                inferenceRule = infInterpreter.complete(premissas, conclusao)

                print "\n"
                print " Resultado + ---------------------------------------- +"
                print
                print " [" + inferenceRule.nome + "]"
                print
                print inferenceRule
                stop()

            except Exception as e:
                print
                print " [ERRO] Problemas ao completar esta R.I.!"
                print "", e
                print
                stop()
                return

        elif opcao in ['R', 'r']:
            sair = True

        elif opcao in ['A', 'a']:
            wiriHelp()




def provar():
    ''' Interface para o modulo de provas de argumentos. '''
    sair = False

    while not sair:
        cls()
        print " (<) Main Menu [R]                              [A] Ajuda"
        print
```

```
print " + Avaliador de Argumentos + ------------------------ +"
print
print " Digite [ENTER] para iniciar                            "
print

opcao = raw_input(" >> ")

if opcao == '':
    cls()
    expInterpreter = ExpressionInterpreter()
    infInterpreter = InferenceInterpreter()

    print " HINT: Deixe um campo vazio quando terminar.\n"
    print

    premissas = []
    objetivo = expInterpreter.eval( raw_input(" [OBJETIVO] ") )
    print " \n" * 3

    p = None
    i = 1
    while p != '':
        p = raw_input(" [" + str(i) + "] ")
        if p != '':
            premissas.append( expInterpreter.eval(p) )
            i += 1

    try:
        logs, conclusao = infInterpreter.proof(premissas, objetivo)

        print "\n"
        print " Resultado + ------------------------------------- +"
        print
        diference = len(conclusao) - len(logs)
        j = 0
        for i, premissa in enumerate(conclusao):
            if diference - 1 < i:
                string = " [ " + str(i) + " ] " + str(premissa)
                print string, " " * (30 - len(string)), logs[j]
                j += 1
            else:
                print " [", i, "]", premissa
        stop()

    except Exception as e:
        print
        print " [ERRO] Problemas ao provar este argumento!"
        print "", e
        print
        stop()
        return

elif opcao in ['R', 'r']:
```

```
            sair = True

        elif opcao in ['A', 'a']:
            wiriHelp()




def main():
    sair = False
    opcoes = {  '1' : analisador,
                '2' : autocompletar,
                '3' : provar,
                '4' : creditos }

    while not sair:
        cls()
        print "       LogSent [W.I.R.I.]"
        print
        print "    [Who Inference Rule Is?]   "
        print
        print
        print " + Main Menu + ------------- +"
        print
        print "  [1] Analisador de R.I.       "
        print
        print "  [2] Autocompletador de R.I.  "
        print
        print "  [3] Provar argumento         "
        print
        print "  [4] Creditos                 "
        print
        print "  [S] Sair                     "
        print
        print " + ------------------ v 1.0 - +"
        print

        opcao = raw_input(" >> ")

        if opcao in ['1', '2', '3', '4']:
            acao = opcoes[opcao]
            acao()

        elif opcao in ['S', 's']:
            sair = True

        else:
            print
            print " [ERRO] Opcao invalida!"
            stop()
```