# ANALIZZATORE PHRASAL VERBS

## Sommario

# SCOPO

Questo programma è stato scritto come compito durante la fase di tirocinio.

Lo scopo di questo programma è l'individuazione dei phrasal verbs all'interno di un testo parserizzato da un postagger. È possibile impostare i phrasal verbs da ricercare tramite il file phrasalverbs_.csv.

## ISTRUZIONI DI COMPILAZIONE DEL FILE CSV

Il file csv rispetta le seguenti specifiche:

- nella prima colonna mettere **il verbo all'infinito (senza to)**
- nella seconda colonna mettere x se il frasale è separabile altrimenti lasciare vuota
- nella terza colonna mettere la prima preposizione
- nella quarta colonna mettere la seconda preposizione altrimenti lasciarla vuota

# SPECIFICA FILE DI INPUT

Il file in input del programma deve necessariamente essere codificato come **utf-8**

Ogni riga deve contenere tre colonne separate da un carattere di tabulazione **\t:**

1. parola
2. postag
3. lemma parola

# SPECIFICA FILE DI OUTPUT

Il file in output del programma sarà codificato in **utf-8**

Ogni riga conterrà tre colonne separate da un carattere di tabulazione **\t:**

1. parola
2. postag
3. lemma parola

# AVVIO DEL PROGRAMMA

Il programma è richiamabile a riga di comando, oppure è integrabile in altri programmi.

L'intero processo è un oggetto della classe ***AnalizzatorePhrasalVerb***

## AVVIO A RIGA DI COMANDO

Per avviare il programma a riga di comando, nella shell del sistema operativo, usare la seguente sintassi

>> python PhrasalVerbs.py [nome file input] [nome file output]

## INTEGRAZIONE PROGRAMMA IN UN ALTRO PROGRAMMA

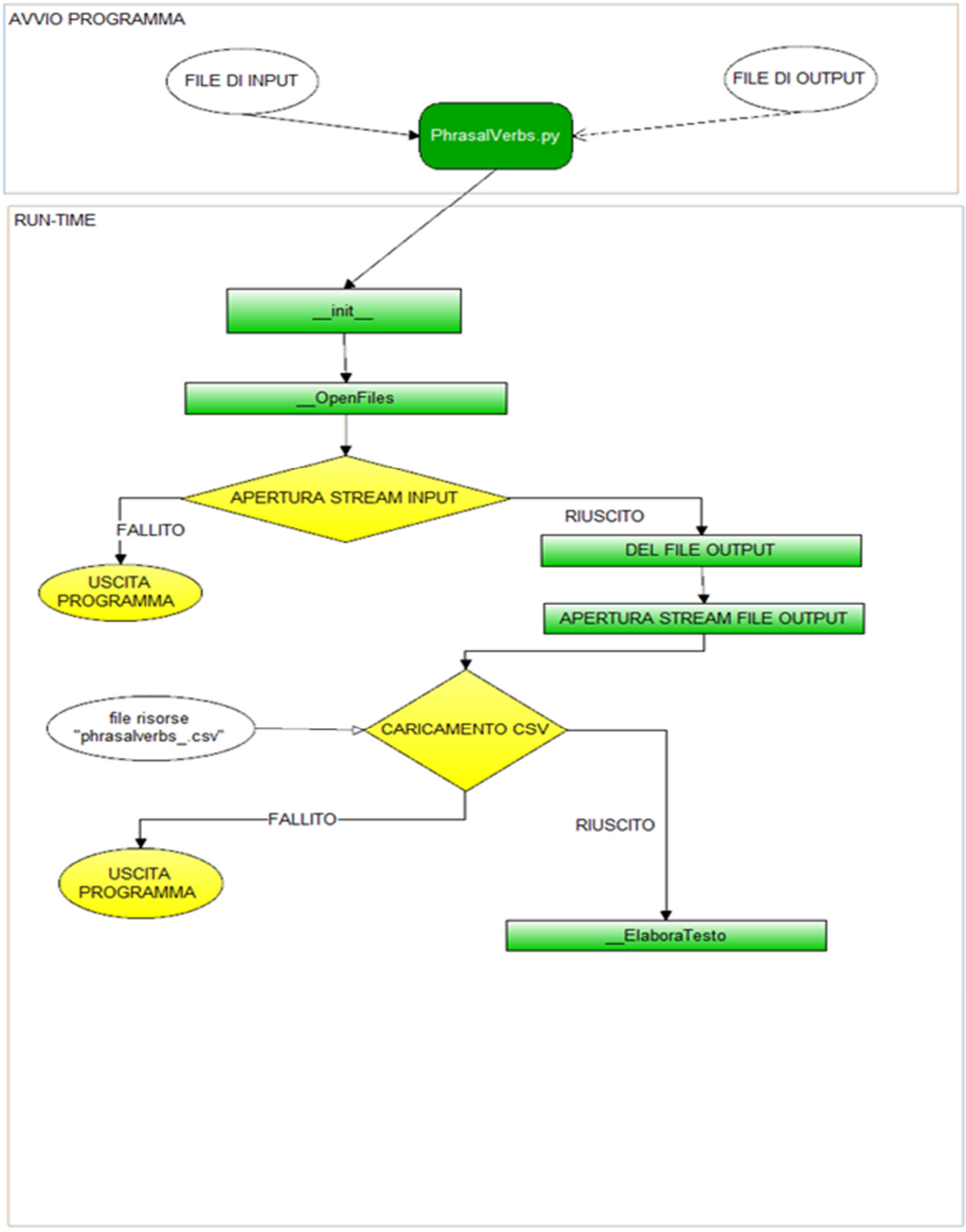
Poiché il processo crea immediatamente un file di output, si consiglia di creare l'oggetto e successivamente di leggere il file in output.

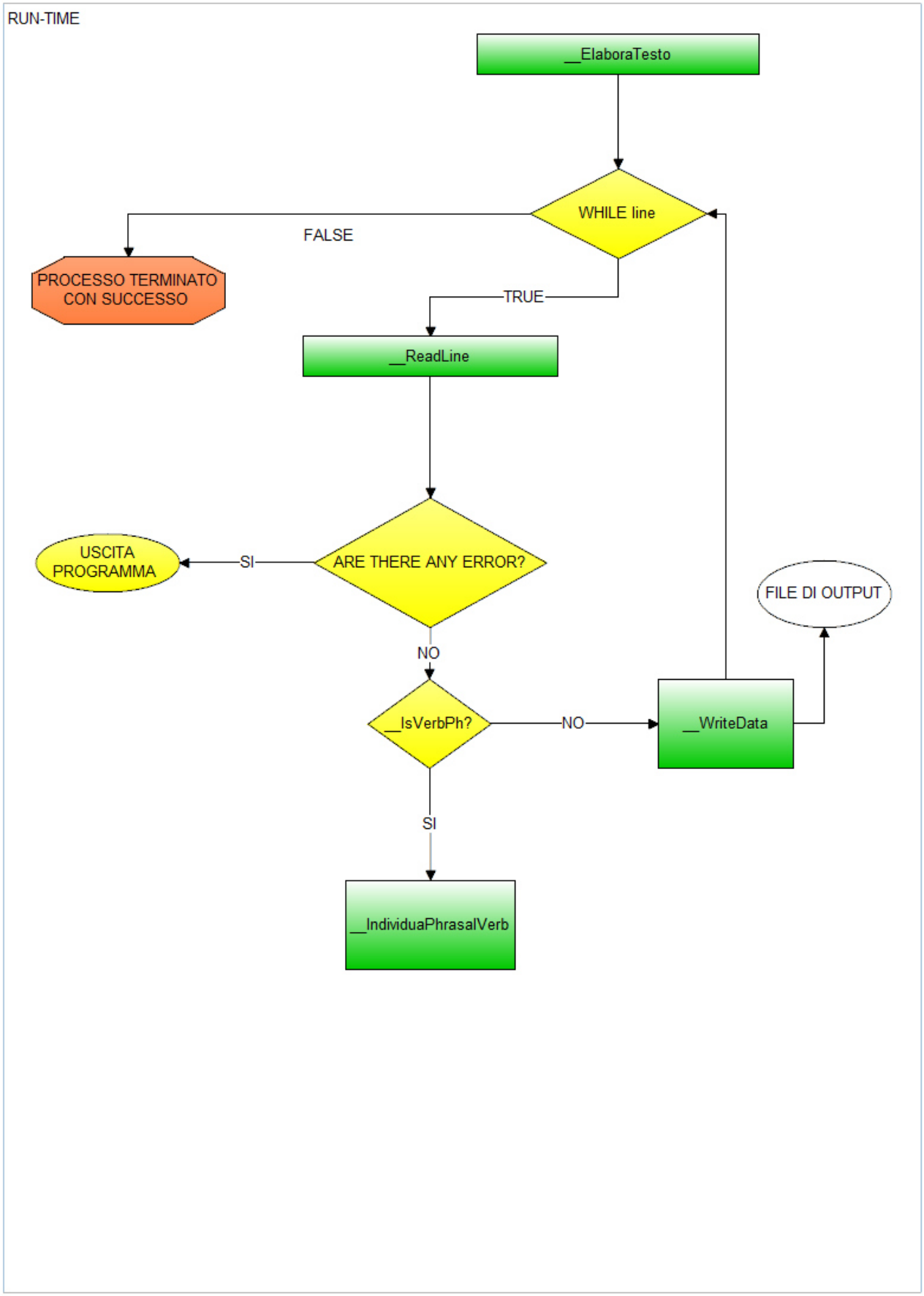
La dichiarazione dell'__init__ della funzione richiede due parametri:

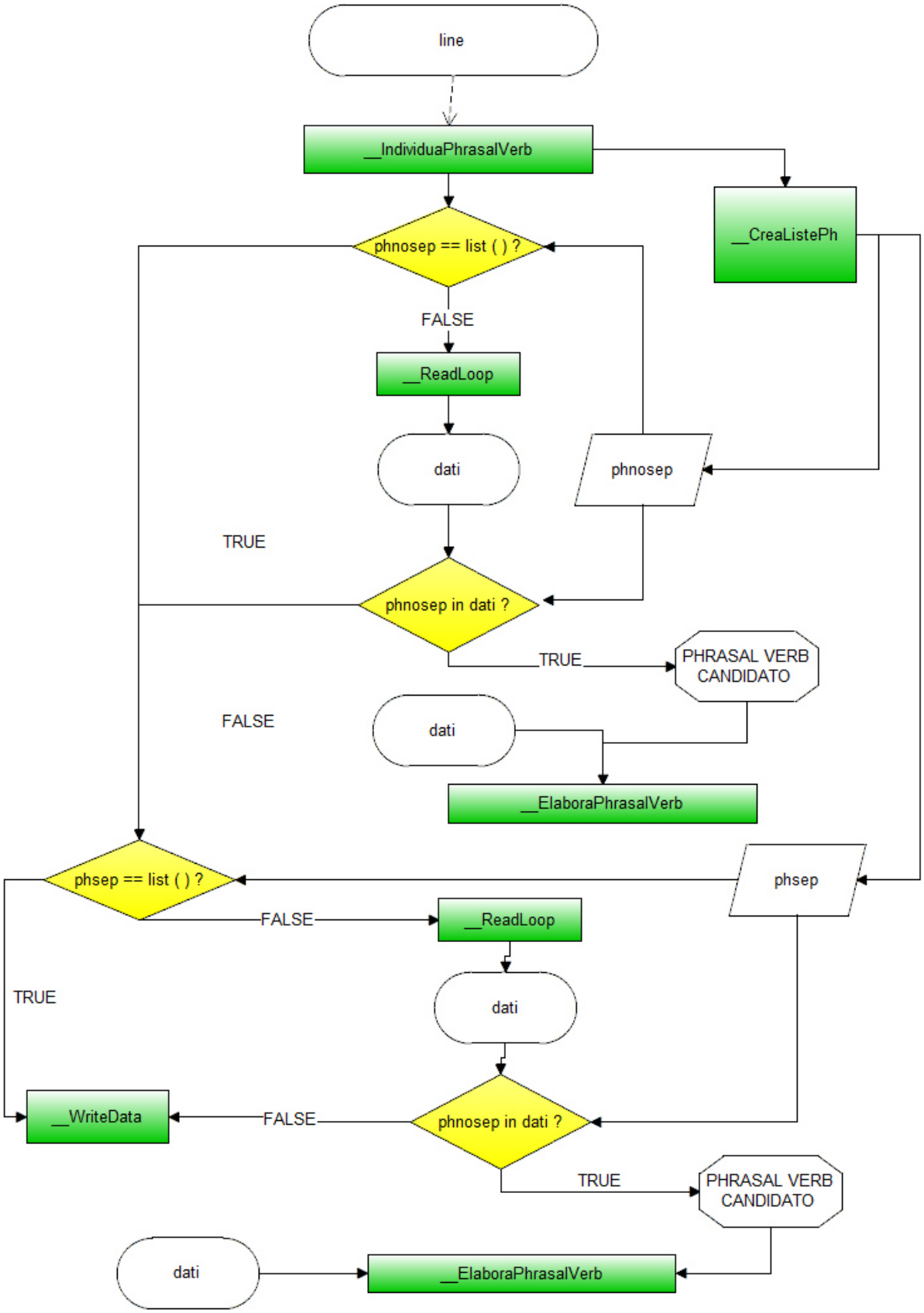
1. **filein**: il percorso completo al file in ingresso
2. **fileout**: il percorso completo al file in uscita

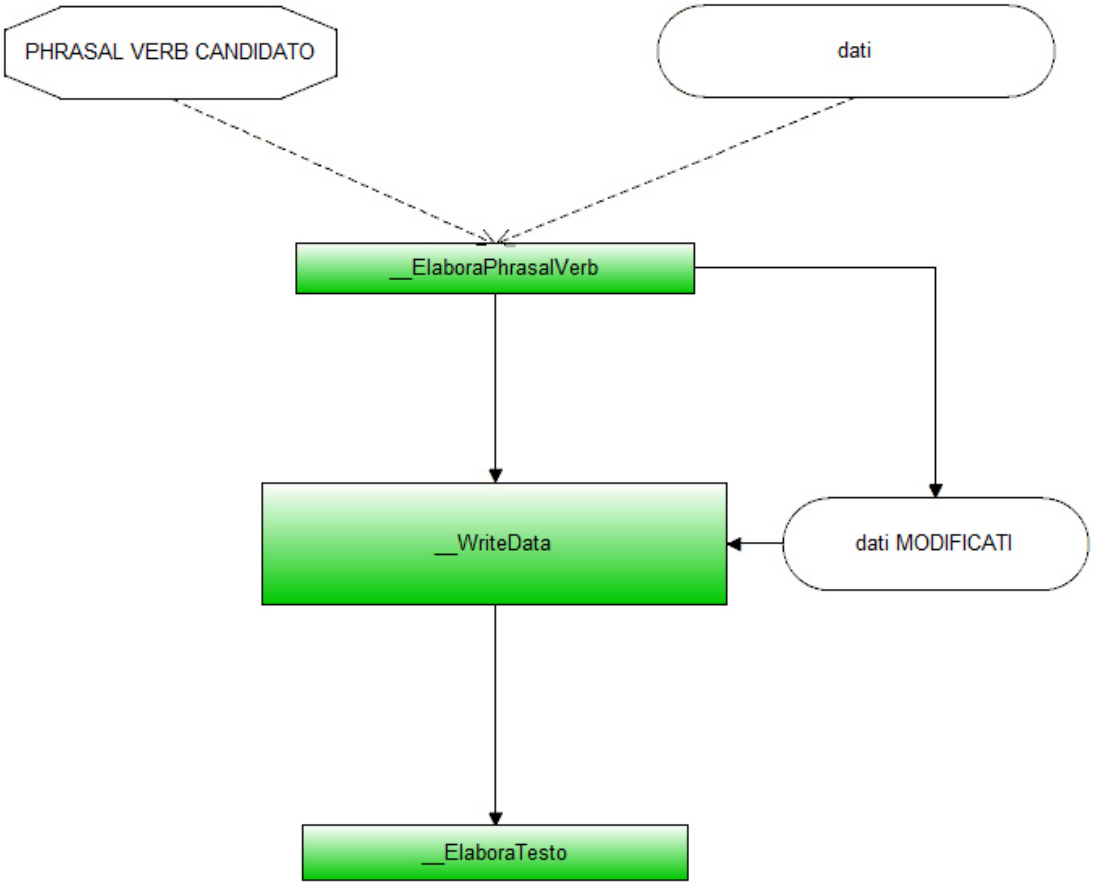
# SCHEMA LOGICO DI FUNZIONAMENTO DEL PROGRAMMA

Si riporta lo schema logico di funzionamento:




RUN-TIME
__ElaboraTesto
WHILE line
FALSE
PROCESSO TERMINATO CON SUCCESSO
TRUE
__ReadLine
ARE THERE ANY ERROR?
SI
USCITA PROGRAMMA
NO
__IsVerbPh?
NO
__WriteData
FILE DI OUTPUT
SI
__IndividuaPhrasalVerb


line
__IndividuaPhrasalVerb
__CreaListePh
phnosep == list ( ) ?
FALSE
__ReadLoop
dati
phnosep
TRUE
phnosep in dati ?
TRUE
PHRASAL VERB CANDIDATO
FALSE
dati
__ElaboraPhrasalVerb
phsep == list ( ) ?
phsep
FALSE
__ReadLoop
TRUE
dati
__WriteData
FALSE
phnosep in dati ?
TRUE
PHRASAL VERB CANDIDATO
dati
__ElaboraPhrasalVerb


PHRASAL VERB CANDIDATO
dati
__ElaboraPhrasalVerb
__WriteData
dati MODIFICATI
__ElaboraTesto

# IL CODICE DEL PROGRAMMA

si riporta il codice integrale del programma:

```
# -*- coding: utf-8 -*-
"""

@author: Patrizio



"""
from __future__ import unicode_literals

import codecs
import os
import csv
import sys

from collections import defaultdict

class AnalizzatorePhrasalVerb:
    r"""
        Questa classe opera sul file elaborato dal postagger
        individua i phrasal verbs, modifica la 3° colonna del frasale aggiungendogli la particella frasale
        ed elimina la riga della particella frasale

    """

    def VERSION (self):
        return "0.5.1.b"
    def AUTHOR (self):
        return "Patrizio Bellan\npatrizio.bellan@gmail.com"

    def __init__ (self, filein, fileout):
        r"""
            :param filein: description
            :param fileout: description
            :type str: path + filename
            :type str: path + filename

        """

        #variabili interne di classe
        self.__filecsv_phrasalverb = u"phrasalverbs_.csv"     #path + filename al csv contenente i parametri dei frasali
        self.__tagforend = list ([u',',u':',u'SENT'])         #le tag che mi identificano fine frase durante la ricerca dei frasali
        self.__tagprep = list ([u'RP'])                       #tag delle particelle frasali da ricercare -> must be type == list
        self.phs = defaultdict ()                             #dizionario contentente i frasali

        self.__fin = None  #puntatore file input
        self.__fout = None #puntatore file output

        if self.__OpenFiles (filein,fileout):
            if self.__ElaboraTesto ():
                print "Processo completato con successo"
            else:
                print "Impossibile completare il processo a causa di errori di elaborazione"
        else:
            print "Impossibile accedere ai files\nProcesso non completato"


    def __OpenFiles (self, filein, fileout):
        r"""
            Questo metodo apre lo stream ai files e carica le risorse necessarie

            :param filein: description
            :param fileout: description
            :type str: path + filename
            :type str: path + filename
```

```
        :return: True - tutto ok False - ci sono stati errori di caricamento/apertura files
        :rtype: bool

    """
    try:
        self.__fin = codecs.open (filein, 'r','utf-8')
        #rimuovo l'eventuale file di output precedente
        try:
            os.remove (fileout)
        except:
            pass

        self.__fout = codecs.open (fileout, 'a','utf-8')

        if self.__LoadPhrasalVerbs ():
            return True
        else:
            return False
    except:
        return False


def __CloseFiles(self):
    r"""
        Questo metodo chiude gli stream dei files
    """
    self.__fin.close ()
    self.__fout.close ()


def __WriteData (self, dati):
    r"""
        Questo metodo scrive i dati sul file di uscita

        :param dati: array di dati da scrivere
        :type list:

        :return: True - tutto ok False - ci sono stati errori
        :rtype: bool

    """

    try:
        #scrivo i dati sul file
        for line in dati:
            if line:
                out = u'\t'.join (line) + u'\n'
                self.__fout.write (out)
                self.__fout.flush ()

        return True

    except:
        print "Errore durante la scrittura del file di output"
        self.__CloseFiles ()

    return False


def __ReadLine (self):
    r"""
        Questo metodo legge una riga dal file di input
        passa la riga al metodo __SeparaColonne e ne ritorna il risultato

        :return: True - tutto ok False - ci sono stati errori
        :rtype: bool

    """

    try:
        line = self.__fin.next ()
        line = self.__SeparaColonne (line)
```

```
            return line

        except StopIteration:
            return False


    def __SeparaColonne (self, line):
        r"""
            Questo metodo, data una riga di input, la splitta e la ripulisce

            :param line: la linea letta
            :type str:

            :return: lista a 3 colonne dei dati in input
            :rtype: list

        """
        return [l.strip() for l in line.split(u'\t')]


    def __LoadPhrasalVerbs (self):
        r"""
            Questo metodo carica le risorse dal file csv in memoria nel dizionario di classe self.phs

            :return: True - tutto ok False - ci sono stati errori
            :rtype: bool

        """
        try:
            #carico i phrasal v. dal file csv - la prima è riga di initestazione quindi la salto
            with open (self.__filecsv_phrasalverb, 'rb') as f:
                reader = csv.reader (f)
                intestazione = True
                for line in reader:
                    if intestazione:
                        intestazione = False
                    else:
                        line = line[0].split (u';')
                        if len (line[0]) > 0 and len (line) == 4:
                            prep1 = line[2]
                            if len (line[3]) > 0:
                                prep2 = line[3]
                            else:
                                prep2 = False

                            if len (line[1]) > 0:
                                separable = True
                            else:
                                separable = False

                            verb = line[0].lower ()
                            try:
                                self.phs[verb].append ([separable, prep1, prep2])
                            except KeyError:
                                self.phs[verb] = [[separable, prep1, prep2]]
                        else:
                            print "ATTENZIONE FORMATO ERRATO NEL FILE CSV:", line
                            return False
                return True
        except :
            print "ERROR LINE:", line

            return False


    def getListVerb (self):
        r"""
            Ritorna tutti i frasali con i parametri

            :return: dizionario contenente tutti i phrasal verbs individuabili
            :rtype: defaultdict(list)
```

```
    """
    return self.phs.keys ()


def __IsVerbPh (self, line):
    r"""
        Questa funzone ritorna True se il verbo è un possibile frasale

        :param line: l'array della linea letta
        :type list:

        :return: True - lineè un verbo, False altrimenti
        :rtype: bool

    """

    if u"V" in line[1] and line[2].lower() in self.phs.keys():
        return True
    else:
        return False


def __ElaboraTesto (self):
    r"""
        Questo metodo elabora tutto il testo

        legge una riga
        la elabora
        scrive la riga sul file di uscita

        :return: True - tutto ok False - ci sono stati errori
        :rtype: bool

    """

    line = True

    try:
        while line:
            line = self.__ReadLine ()
            if not line:
                return self.__WriteData ([line])
            if self.__IsVerbPh(line):
                if not self.__IndividuaPhrasalVerb (line):
                    return False
            else:
                if not self.__WriteData ([line]):
                    return False

        return True
    except:
        return False


def __IndividuaPhrasalVerb (self, line):
    r"""
        Questo metodo ricerca se siamo in presenza di un phrasal verbs

        :param line: l'array della linea letta
        :type list:

        :return: True - tutto ok False - ci sono stati errori
        :rtype: bool

    """
    candidato = False
    dati = [line]

    #elaboro le liste dei candidati
    sep, nosep = self.__CreaListePh (line)
```

```
#prima controllo se è nei nosep
if nosep:
    n = self.__ReadLoop (dati, 2)
    for ph in nosep:
        if dati[1][2] == ph[0]:
            candidato = ph
            if n>2:
                if dati[2][2] == ph[1]:
                    #phrasal Trovato!
                    #ora devo elaborare il phrasal in uscita
                    if self.__ElaboraPhrasalVerb (dati,ph):
                        return True
                    else:
                        return False
    if n == -1:
        return self.__WriteData (dati)
    if candidato:
        #phrasal Trovato!
        #ora devo elaborare il phrasal in uscita
        if self.__ElaboraPhrasalVerb(dati, candidato):
            return True
        else:
            return False

if not sep:
    #controllo che non vi siano preposizioni
    if len (dati) > 1:
        if self.__IsTherePrep (dati, 1):
            return True
        if dati[1][1] in self.__tagforend:
            return self.__WriteData (dati)

    #controllo che non vi siano preposizioni
    if len (dati) == 3:
        if self.__IsTherePrep (dati, 2):
            return True

if u'V' in dati[-1][1]:
    return self.__WriteData (dati)
if len (dati) > 2:
    if u'V' in dati[-2][1]:
        return self.__WriteData (dati)

#poi o ho già risolto e quindi non sono arrivato qui oppure
#o è un ph separabile (se la lista non è vuota) e quindi cerco di risolverlo
#oppure no lo è e quindi copio i dati nel file di out
if sep:
    if len (dati) != 3:
        n = self.__ReadLoop (dati,2)
        if n and n == 1:
            if self.__WriteData (dati):
                return True
            else:
                return False
    #per prima cosa verifico i candidati con una sola preposizione
    for ph in sep:
        if dati[2][2] == ph[0] and not ph[1]:
            candidato = ph
            break

    #procedo nella lettura della frase

    #se trovo un candidato a 2 lo elaboro
    #se non trovo un candidato migliore ma la variabile candidato ne contiene
    #già uno allora utilizzo candidato
    #altrimenti procedo nella ricerca fino a che non trovo o una fine frase o un altra prep

    #per prima cosa leggo la prossima riga
    #se è un fine frase:
    #           elaboro come frasale il candidato e scrivo i dati e l'ultima riga letta
    #se ho un candidato, lo uso per filtrare la prep2,
    #se lo trovo nell'ultima riga:
```

```
            #                   scrivo il frasale
            #altrimenti registro il candidato trovato precedentemente
            #se non ho un candidato, ricerco partendo dalla prep1

            while True:
                nextline = self.__ReadLine ()
                #verifico di non essere a fine file
                if not nextline and not candidato:
                    if self.__WriteData ([nextline]) and self.__WriteData (dati):
                        return True
                    else:
                        return False
                elif not nextline and candidato:
                    if self.__ElaboraPhrasalVerb (dati,candidato) and self.__WriteData ([nextline]):
                        return True
                    else:
                        return False

                if nextline[1] in self.__tagforend:
                    if candidato:
                        if self.__ElaboraPhrasalVerb (dati, candidato) and self.__WriteData ([nextline]):
                            return True
                        else:
                            return False

                #verifico se ho un candidato cerco prep2 in nextline
                if candidato:
                    for ph in sep:
                        #questo è il filtro dei ph
                        if ph[0] == candidato[0]:
                            if ph[1] == nextline[2]:
                                dati.append (nextline)
                                if self.__ElaboraPhrasalVerb (dati, ph):
                                    return True
                                else:
                                    return False
                    #se sono qui devo utilizzare il candidato come phrasal
                    if self.__ElaboraPhrasalVerb (dati,ph) and self.__WriteData ([nextline]):
                        return True
                    else:
                        return False

                #se sono qui è perchè non ho un candidato e quindi devo effettuare la ricerca
                #partendo dalla prima colonna della matrice sep
                dati.append (nextline)

                #verifico tutti i candidati presenti a due prep in sep
                for ph in sep:
                    if ph[1] == dati[-1][2] and ph[0] == dati[-2][2]:
                        return self.__ElaboraPhrasalVerb (dati, ph)


                if self.__IsTherePrep (dati, -1):
                    return True

                #cerco di individuare il possibile prossimo candidato
                for ph in sep:
                    if dati[-1][2] == ph[0] and not ph[1]:
                        candidato = ph
                        break

        else:
            if self.__WriteData (dati):
                return True
            else:
                return False


    def __IsTherePrep(self, dati, index):
        r"""
            Questa funziona ritorna true se ha trovato una preposizione o un verbo,
            dopo aver registrato i dati
```

```
        :param dati: array di dati da scrivere
        :type list:
        :param index: l'indice di ricerca
        :type list:

        :return: True - preposizione del phrasal trovata, False altirmenti
        :rtype: bool

    """
    if dati[index][1] in self.__tagprep or u'V' in dati[index][1]:
        self.__WriteData (dati)
        return True
    else:
        return False


def __ElaboraPhrasalVerb(self, dati, ph):
    r"""
        Questo metodo modifica le righe in corrispondenza del frasale nell'array dati
        elimina la riga della particella/e del frasale
        e scrive i risultati su file

        :param dati: array di dati da scrivere
        :type list:
        :param ph: il phrasal trovato
        :type list:

        :return: True - tutto ok (riga scritta sul fileout) False - ci sono stati errori
        :rtype: bool

    """

    phrasal = dati[0][2] + u'_' + ph[0]
    #elimino la riga della preposiz. del frasale
    for i in dati:
        if i[2]  ==  ph[0]:
            dati.remove (i)
            break
    if ph[1]:
        phrasal = u'_' + phrasal + u'_' + ph[1]
        #elimino la riga della preposiz. del frasale
        for i in dati:
            if i[2]  ==  ph[1]:
                dati.remove (i)
                break

    #Aggiungo la tag alla seconda colonna (postag)
    dati[0][1] = dati[0][1] + u'_PHRASAL_' + phrasal

    #aggiungo le informazioni alla 3° colonna del verbo e salvo i dati
    dati[0][2]  =  phrasal

    return self.__WriteData (dati)


def __ReadLoop (self, dati, n):
    r"""
        questa funzione legge un numero n di righe
        se trova una riga con tag di fine esce

        :param dati: array di dati da scrivere
        :type list:
        :param n: il numero di righe da leggere
        :type int:

        :return: il numero di righe effettivamente lette
        :rtype: int
        :return: Modifica l'array dati in ingresso
        :rtype: None

    """
```

```
            for i in xrange (n):
                line = self.__ReadLine ()
                dati.append (line)
                if line:
                    if line[1] in self.__tagforend:
                        return i + 1
                else:
                    #sono a fine file
                    return -1
            return i + 1


        def __CreaListePh (self, line):
            r"""
                Questo metodo crea due liste
                una per i candidati separabili
                l'altra per i candidati non separabili

                :param line: l'array della linea letta
                :type list:

                :return: phsep di tutti i possibili frasali separabili
                :rtype: list
                :return: phnosep di tutti i possibili frasali non separabili
                :rtype: list

            """

            phsep, phnosep = list(),list()
            #creo due liste, una contenente i separabili e l'altra non separabili
            for ph in self.phs[line[2].lower ()]:
                if ph[0]:
                    phsep.append ([ph[1], ph[2]])
                else:
                    phnosep.append ([ph[1], ph[2]])
            return phsep, phnosep


    if __name__ == '__main__':
        if len (sys.argv) == 3:
            #avvio il programma
            print "avvio programma"
            fileinp = sys.argv[1]
            fileout = sys.argv[2]
        else:
            #test mode
            print "Test Mode"
            print """ specificare i file di input e di output come nell'esempio\n\n
            >>>python PhrasalVerbsPipeline input.txt output.txt\n
            Test Mode files:
            fileinp = 'input_pulito.txt'
            fileout = 'output_2.txt'
            """

            fileinp = 'input_pulito.txt'
            fileout = 'output_2.txt'

        a = AnalizzatorePhrasalVerb(fileinp,fileout)
```