PROYECTO DE PROGRAMACIÓN ALF

Ángel Ruiz Fernandez, G2.2

21:28 dic 11, 2024

PROYECTO DE PROGRAMACIÓN ALF: 6 files PAGE

```python
import sys
import re

import normalize
import sphone
import snif
import stime
import slocation

strippedphoneprog = re.compile("^[+]?\d{9,15}$")
nifprog = re.compile("[0-9]{8}[A-Z]")
normdateprog = re.compile("([0-9]{4})-([0-9]{2})-([0-9]{2}) ([0-9]{2}):([0-9]{2})")
normgeoprog = re.compile("^(\d{7}.\d{4}[NS])(\d{7}.\d{4}[WE])$")

def isnumeric(string):
    try:
        float(string)
        return True
    except ValueError:
        return False

if (len(sys.argv) == 1):
    print("Argumentos incorrectos")
    exit(1)

if sys.argv[1] == "-n" and len(sys.argv) == 3:
    normalize.normalize(sys.argv[2])
elif sys.argv[1] == "-sphone":
    if (len(sys.argv) != 4):
        print("Numero de argumentos invalido")
        exit(1)
    if (strippedphoneprog.match(normalize.normalize_phone(sys.argv[2])) == None):
        print("Telefono invalido")
        exit(1)
    sphone.sphone(sys.argv[3], normalize.normalize_phone(sys.argv[2]))
elif sys.argv[1] == "-snif":
    if (len(sys.argv) != 4):
        print("Numero de argumentos invalido")
        exit(1)
    if (nifprog.match(sys.argv[2]) == None):
        print("NIF invalido")
        exit(1)
    snif.snif(sys.argv[3], sys.argv[2])
elif sys.argv[1] == "-stime":
    if (len(sys.argv) != 5):
        print("Numero de argumentos invalido")
        exit(1)
    if (normdateprog.match(normalize.normalize_date(sys.argv[2])) == None or normdateprog.match(normalize.normalize_date(sys.argv[
3])) == None):
        print("Fecha invalida")
        exit(1)
    stime.stime(sys.argv[4], normdateprog, normalize.normalize_date(sys.argv[2]), normalize.normalize_date(sys.argv[3]))
elif sys.argv[1] == "-slocation":
    if (len(sys.argv) != 5):
        print("Numero de argumentos invalido")
        exit(1)
    if (normgeoprog.match(normalize.normalize_coord(sys.argv[2])) == None or not isnumeric(sys.argv[3])):
        print("Coordenadas invalidas")
        exit(1)
    slocation.slocation(sys.argv[4], normgeoprog, normalize.normalize_coord(sys.argv[2]), sys.argv[3])
```

```
else:
    print("Argumentos invalidos")
    exit(1)


```

```python
import re
import math

months = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"]

date2prog = re.compile("([a-zA-Z]+) ([0-9]{2}), ([0-9]{4}) ([0-9]{2}):([0-9]{2}) (AM|PM)")
date3prog = re.compile("([0-9]{2}):([0-9]{2}):([0-9]{2}) ([0-9]{2})\/([0-9]{2})\/([0-9]{4})")
geo2prog  = re.compile("(\d*)\° (\d*)\' (\d*)\" ([NS]), (\d*)\° (\d*)\' (\d*)\" ([WE])")
geo1prog  = re.compile("([+-]?\d*\.\d+),[ ]?([+-]?\d*\.\d+)")
extrasprog = re.compile("[ ]*;[ ]*")

def normalize_phone(line):
    phone_org = line.split(";")[0].strip()
    phone_norm = phone_org.replace(" ", "")
    if phone_norm[0] != "+":
        phone_norm = "+34" + phone_norm

    return line.replace(phone_org, phone_norm)



def normalize_date(line):
    m = re.search(date2prog, line)
    if (m != None):
        grps = m.groups()

        newdate = grps[2] + "-"
        newdate += "{:02d}".format(months.index(grps[0]) + 1) + "-"
        newdate += grps[1] + " "
        if (grps[5] == "AM"):
            newdate += str(grps[3]) + ":"
        else:
            newdate += str(int(grps[3]) + 12) + ":"
        newdate += grps[4]

        return line.replace(line[m.start():m.end()], newdate)

    m = re.search(date3prog, line)
    if (m != None):
        grps = m.groups()

        newdate = grps[5] + "-" + grps[4] + "-" + grps[3] + " " + grps[0] + ":" + grps[1]

        return line.replace(line[m.start():m.end()], newdate)

    return line

def normalize_coord(line):
    m = re.search(geo2prog, line)
    if (m != None):
        grps = m.groups()

        gps =  "{:03d}".format(int(grps[0])) + "{:02d}".format(int(grps[1]))
        gps += "{:07.4f}".format(float(grps[2])) + grps[3]
        gps += "{:03d}".format(int(grps[4])) + "{:02d}".format(int(grps[5]))
        gps += "{:07.4f}".format(float(grps[6])) + grps[7]

        line = line.replace(line[m.start():m.end()], gps)

    m = re.search(geo1prog, line)
    if (m != None):
        grps = m.groups()
        lat = float(grps[0])
        lon = float(grps[1])
        latdir = "N"
        londir = "W"
        if lat < 0:
            latdir = "S"
            lat *= -1
        if lon < 0:
            londir = "E"
            lon *= -1
        gps  = "{:03d}".format(math.floor(lat))
        lat -= math.floor(lat)
        lat *= 60
        gps += "{:02d}".format(math.floor(lat))
        lat -= math.floor(lat)
        lat *= 60
        gps += "{:07.4f}".format(math.floor(lat)) + latdir

        gps  += "{:03d}".format(math.floor(lon))
        lon  -= math.floor(lon)
        lon  *= 60
        gps  += "{:02d}".format(math.floor(lon))
        lon  -= math.floor(lon)
        lon  *= 60
        gps  += "{:07.4f}".format(math.floor(lon)) + londir

        line = line.replace(line[m.start():m.end()], gps)

    return line


def normalize(fname):
    file = open(fname, "r")

    for line in file:
        if line == "\n":
            continue
        line = re.sub(extrasprog, ";", line)
        line = normalize_date(line)
        line = normalize_coord(line)
        line = normalize_phone(line)

        print(line, end='')


    file.close()

```

```python
import re
import math

import normalize

def sextodec(deg, min, sec):
    return (sec + min * 60 + deg * 3600)/3600

def coordtofloat(gps):
    lat = sextodec(float(gps[0:2]), float(gps[3:4]), float(gps[5:11]))
    if (gps[12] == "S"):
        lat = -lat
    lon = sextodec(float(gps[13:16]), float(gps[17:18]), float(gps[19:25]))
    if (gps[25] == "S"):
        lon = -lon

    return lat, lon

def distance(lat1, lon1, lat2, lon2):
    lat1 = math.radians(lat1)
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = math.sin(dlat/2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon/2)**2
    c = 2 * math.atan2(math.sqrt(a),  math.sqrt(1-a))
    return 6371 * c


def slocation(fname, normgeoprog, orggps, range):
    file = open(fname, "r")

    orggps = normalize.normalize_coord(orggps)

    if re.match(normgeoprog, orggps) == None or len(orggps) != 26:
        print("Invalid location argument")
        return

    org = coordtofloat(orggps)
    range = float(range)

    for line in file:
        if line == "\n":
            continue

        normline = normalize.normalize_coord(line)
        locgps = normline.split(";")[3].strip()
        if re.match(normgeoprog, locgps) == None or len(locgps) != 26:
            continue

        loc = coordtofloat(locgps)

        d = distance(loc[0], loc[1], org[0], org[1])

        if d < range:
            print(line, end='')

    file.close()
```

```python
def snif(fname, nif):
    file = open(fname, "r")
    for line in file:
        if line == "\n":
            continue

        if nif in line:
            print(line, end='')
    file.close()
```

```python
import normalize

def sphone(fname, phone):
    filter = normalize.normalize_phone(phone)

    file = open(fname, "r")
    for line in file:
        if line == "\n":
            continue

        num = normalize.normalize_phone(line.split(";")[0])
        if filter in num:
            print(line, end='')
    file.close()
```

```python
import re

import normalize


def comparar(A1, M1, D1, h1, m1, A2, M2, D2, h2, m2):
    if A1 > A2:
        return 1
    elif A1 < A2:
        return -1
    elif A1 == A2:
        if M1 > M2:
            return 1
        elif M1 < M2:
            return -1
        elif M1 == M2:
            if D1 > D2:
                return 1
            elif D1 < D2:
                return -1
            elif D1 == D2:
                if h1 > h2:
                    return 1
                elif h1 < h2:
                    return -1
                elif h1 == h2:
                    if m1 > m2:
                        return 1
                    elif m1 < m2:
                        return -1
                    elif m1 == m2:
                        return 0


def stime(fname, normdateprog, t1, t2):
    t1grps = re.search(normdateprog, t1).groups()
    t2grps = re.search(normdateprog, t2).groups()

    file = open(fname, "r")
    for line in file:
        if line == "\n":
            continue

        normline = normalize.normalize_date(line)

        m = re.search(normdateprog, normline)

        if (m != None):
            grps = m.groups()

            if ((comparar(grps[0], grps[1], grps[2], grps[3], grps[4],
                         t1grps[0], t1grps[1], t1grps[2], t1grps[3], t1grps[4]) == 1) and
                (comparar(grps[0], grps[1], grps[2], grps[3], grps[4],
                         t2grps[0], t2grps[1], t2grps[2], t2grps[3], t2grps[4]) == -1)):
                print(line, end='')
    file.close()

```