

PROYECTO DE PROGRAMACIÓN ALF
Ángel Ruiz Fernandez, G2.2
21:28 dic 11, 2024

PROYECTO DE PROGRAMACIÓN ALF: 6 files	PAGE
main.py 64L 2073B	3
normalize.py 109L 3173B	5
slocation.py 59L 1506B	7
snif.py 9L 192B	8
sphone.py 14L 323B	9
stime.py 57L 1533B	10

```
1 import sys
2 import re
3
4 import normalize
5 import sphone
6 import sniff
7 import stime
8 import slocation
9
10 strippedphoneprog = re.compile("^[+]?\\d{9,15}$")
11 nifprog = re.compile("[0-9]{8}[A-Z]")
12 normdateprog = re.compile("([0-9]{4})-([0-9]{2})-([0-9]{2}) ([0-9]{2}):([0-9]{2})")
13 normgeoprogram = re.compile("^\\d{7}\\.\\d{4}[NS]) (\\d{7}\\.\\d{4}[WE])$")
14
15 def isnumeric(string):
16     try:
17         float(string)
18         return True
19     except ValueError:
20         return False
21
22 if len(sys.argv) == 1:
23     print("Argumentos incorrectos")
24     exit(1)
25
26 if sys.argv[1] == "-n" and len(sys.argv) == 3:
27     normalize.normalize(sys.argv[2])
28 elif sys.argv[1] == "-sphone":
29     if len(sys.argv) != 4:
30         print("Numero de argumentos invalido")
31         exit(1)
32     if (strippedphoneprog.match(normalize.normalize_phone(sys.argv[2])) == None):
33         print("Telefono invalido")
34         exit(1)
35     sphone.sphone(sys.argv[3], normalize.normalize_phone(sys.argv[2]))
36 elif sys.argv[1] == "-snif":
37     if len(sys.argv) != 4:
38         print("Numero de argumentos invalido")
39         exit(1)
40     if (nifprog.match(sys.argv[2]) == None):
41         print("NIF invalido")
42         exit(1)
43     sniff.snif(sys.argv[3], sys.argv[2])
44 elif sys.argv[1] == "-stime":
45     if len(sys.argv) != 5:
46         print("Numero de argumentos invalido")
47         exit(1)
48     if (normdateprog.match(normalize.normalize_date(sys.argv[2])) == None or normdateprog.match(normalize.normalize_date(sys.argv[
49 3]))) == None):
50         print("Fecha invalida")
51         exit(1)
52     stime.stime(sys.argv[4], normdateprog, normalize.normalize_date(sys.argv[2]), normalize.normalize_date(sys.argv[3]))
53 elif sys.argv[1] == "-slocation":
54     if len(sys.argv) != 5:
55         print("Numero de argumentos invalido")
56         exit(1)
57     if (normgeoprogram.match(normalize.normalize_coord(sys.argv[2])) == None or not isnumeric(sys.argv[3])):
58         print("Coordenadas invalidas")
59         exit(1)
60     slocation.slocation(sys.argv[4], normgeoprogram, normalize.normalize_coord(sys.argv[2]), sys.argv[3])
```

```
60 else:
61     print("Argumentos invalidos")
62     exit(1)
63
64
```

```

1 import re
2 import math
3
4 months = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciemb
re"]
5
6 date2prog = re.compile("([a-zA-Z]+) ([0-9]{2}), ([0-9]{4}) ([0-9]{2}):([0-9]{2}) (AM|PM)")
7 date3prog = re.compile("([0-9]{2}):([0-9]{2}):([0-9]{2}) ([0-9]{2})\\/([0-9]{2})\\/([0-9]{4})")
8 geo2prog = re.compile("(\\d*)\\° (\\d*)\\' (\\d*)\\\" ([NS]), (\\d*)\\° (\\d*)\\' (\\d*)\\\" ([WE])")
9 geo1prog = re.compile("([+-]?\\d*\\.\\d+), [ ]?([+-]?\\d*\\.\\d+)")
10 extrasprog = re.compile("[ ]*;*[ ]*")
11
12 def normalize_phone(line):
13     phone_org = line.split(";")[0].strip()
14     phone_norm = phone_org.replace(" ", "")
15     if phone_norm[0] != "+":
16         phone_norm = "+34" + phone_norm
17
18     return line.replace(phone_org, phone_norm)
19
20
21
22 def normalize_date(line):
23     m = re.search(date2prog, line)
24     if (m != None):
25         grps = m.groups()
26
27         newdate = grps[2] + "-"
28         newdate += "{:02d}".format(months.index(grps[0]) + 1) + "-"
29         newdate += grps[1] + " "
30         if (grps[5] == "AM"):
31             newdate += str(grps[3]) + ":"
32         else:
33             newdate += str(int(grps[3]) + 12) + ":"
34         newdate += grps[4]
35
36         return line.replace(line[m.start():m.end()], newdate)
37
38     m = re.search(date3prog, line)
39     if (m != None):
40         grps = m.groups()
41
42         newdate = grps[5] + "-" + grps[4] + "-" + grps[3] + " " + grps[0] + ":" + grps[1]
43
44         return line.replace(line[m.start():m.end()], newdate)
45
46     return line
47
48 def normalize_coord(line):
49     m = re.search(geo2prog, line)
50     if (m != None):
51         grps = m.groups()
52
53         gps = "{:03d}".format(int(grps[0])) + "{:02d}".format(int(grps[1]))
54         gps += "{:07.4f}".format(float(grps[2])) + grps[3]
55         gps += "{:03d}".format(int(grps[4])) + "{:02d}".format(int(grps[5]))
56         gps += "{:07.4f}".format(float(grps[6])) + grps[7]
57
58         line = line.replace(line[m.start():m.end()], gps)
59

```

```
60     m = re.search(geolprog, line)
61     if (m != None):
62         grps = m.groups()
63         lat = float(grps[0])
64         lon = float(grps[1])
65         latdir = "N"
66         londir = "W"
67         if lat < 0:
68             latdir = "S"
69             lat *= -1
70         if lon < 0:
71             londir = "E"
72             lon *= -1
73         gps = "{:03d}".format(math.floor(lat))
74         lat -= math.floor(lat)
75         lat *= 60
76         gps += "{:02d}".format(math.floor(lat))
77         lat -= math.floor(lat)
78         lat *= 60
79         gps += "{:07.4f}".format(math.floor(lat)) + latdir
80
81         gps += "{:03d}".format(math.floor(lon))
82         lon -= math.floor(lon)
83         lon *= 60
84         gps += "{:02d}".format(math.floor(lon))
85         lon -= math.floor(lon)
86         lon *= 60
87         gps += "{:07.4f}".format(math.floor(lon)) + londir
88
89         line = line.replace(line[m.start():m.end()], gps)
90
91     return line
92
93
94 def normalize(fname):
95     file = open(fname, "r")
96
97     for line in file:
98         if line == "\n":
99             continue
100         line = re.sub(extrasprog, ";", line)
101         line = normalize_date(line)
102         line = normalize_coord(line)
103         line = normalize_phone(line)
104
105         print(line, end='')
106
107
108     file.close()
109
```

```
1 import re
2 import math
3
4 import normalize
5
6 def sextodec(deg, min, sec):
7     return (sec + min * 60 + deg * 3600)/3600
8
9 def coordtofloat(gps):
10     lat = sextodec(float(gps[0:2]), float(gps[3:4]), float(gps[5:11]))
11     if (gps[12] == "S"):
12         lat = -lat
13     lon = sextodec(float(gps[13:16]), float(gps[17:18]), float(gps[19:25]))
14     if (gps[25] == "S"):
15         lon = -lon
16
17     return lat, lon
18
19 def distance(lat1, lon1, lat2, lon2):
20     lat1 = math.radians(lat1)
21     lon1 = math.radians(lon1)
22     lat2 = math.radians(lat2)
23     lon2 = math.radians(lon2)
24     dlat = lat2 - lat1
25     dlon = lon2 - lon1
26     a = math.sin(dlat/2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon/2)**2
27     c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
28     return 6371 * c
29
30
31 def slocation(fname, normgeoprogram, orggps, range):
32     file = open(fname, "r")
33
34     orggps = normalize.normalize_coord(orggps)
35
36     if re.match(normgeoprogram, orggps) == None or len(orggps) != 26:
37         print("Invalid location argument")
38         return
39
40     org = coordtofloat(orggps)
41     range = float(range)
42
43     for line in file:
44         if line == "\n":
45             continue
46
47         normline = normalize.normalize_coord(line)
48         locgps = normline.split(";")[3].strip()
49         if re.match(normgeoprogram, locgps) == None or len(locgps) != 26:
50             continue
51
52         loc = coordtofloat(locgps)
53
54         d = distance(loc[0], loc[1], org[0], org[1])
55
56         if d < range:
57             print(line, end='')
58
59     file.close()
```

```
1 def snif(fname, nif):
2     file = open(fname, "r")
3     for line in file:
4         if line == "\n":
5             continue
6
7         if nif in line:
8             print(line, end='')
9     file.close()
```



```
1 import normalize
2
3 def sphone(fname, phone):
4     filter = normalize.normalize_phone(phone)
5
6     file = open(fname, "r")
7     for line in file:
8         if line == "\n":
9             continue
10
11         num = normalize.normalize_phone(line.split(";")[0])
12         if filter in num:
13             print(line, end='')
14     file.close()
```

```
1 import re
2
3 import normalize
4
5
6 def comparar(A1, M1, D1, h1, m1, A2, M2, D2, h2, m2):
7     if A1 > A2:
8         return 1
9     elif A1 < A2:
10         return -1
11     elif A1 == A2:
12         if M1 > M2:
13             return 1
14         elif M1 < M2:
15             return -1
16         elif M1 == M2:
17             if D1 > D2:
18                 return 1
19             elif D1 < D2:
20                 return -1
21             elif D1 == D2:
22                 if h1 > h2:
23                     return 1
24                 elif h1 < h2:
25                     return -1
26                 elif h1 == h2:
27                     if m1 > m2:
28                         return 1
29                     elif m1 < m2:
30                         return -1
31                     elif m1 == m2:
32                         return 0
33
34
35 def stime(fname, normdateprog, t1, t2):
36     t1grps = re.search(normdateprog, t1).groups()
37     t2grps = re.search(normdateprog, t2).groups()
38
39     file = open(fname, "r")
40     for line in file:
41         if line == "\n":
42             continue
43
44         normline = normalize.normalize_date(line)
45
46         m = re.search(normdateprog, normline)
47
48         if (m != None):
49             grps = m.groups()
50
51             if ((comparar(grps[0], grps[1], grps[2], grps[3], grps[4],
52                 t1grps[0], t1grps[1], t1grps[2], t1grps[3], t1grps[4]) == 1) and
53                 (comparar(grps[0], grps[1], grps[2], grps[3], grps[4],
54                 t2grps[0], t2grps[1], t2grps[2], t2grps[3], t2grps[4]) == -1)):
55                 print(line, end='')
56     file.close()
57
```