Pythonra

Programozzunk Python nyelven III

Ebben a részben néhány egyszerűbb python scriptet mutatok be, majd egy-két gyakran használt algoritmus python megvalósítását fogjuk tárgyalni.

Biztosan sokan hallottak arról hogy az ember olvasáskor elsősorban az utolsó néhány karakterét figyeli meg a szavaknak, és az ezek között található karakterek sorrendje nem olyan fontos az olvashatóság szempontjából.

„Valuózínsleg ezt a szöevget is el tjdua osavlni mndneiki.”

Készítsünk egy olyan python scriptet ami a megadott file szavait beolvassa, és összekeveri bennük a betűket, majd kiírja az eredményt egy másik file-ba.

A python filekezelése:

Pythonban, csakúgy mint más nyelvekben, ahhoz hogy egy file-ból olvasni tudjunk előbb meg kell nyitnunk. Erre az open() függvény szolgál.
Az open() nek 2 string paramétere van, az első a file neve, a használt operációs rendszernek megfelelő elérési úttal. A második pedig a file hozzáférési mód,
hogy írásra, olvasásra vagy hozzáfűzésre akarjuk használni, a szöveg vagy bináris file-t.

Hozzáférési mód	Leírás
r	Csak olvasásra nyitja, meg. Ilyenkor írni nem tudunk a file-ba
w	Csak írásra nyitja meg.
a	Hozzáfűzésre nyitja meg, vagy a file végéhez pozicionál, és onnan kezdve írhatunk a file-ba.
rb	Binárisként kezeli a file-t, és olvasásra nyitja meg.
wb	Írásra nyitja meg a bináris file-t.
ab	Hozzáfűzésre nyitja meg, és binárisként kezeli.
r+	Olvasásra és írásra nyitja meg.
w+	Írásra és olvasásra nyitja meg.
a+	Hozzáfűzésre és olvasásra.
rb+	Olvasásra és írásra nyitja meg a bináris file-t.
wb+	Írásra és olvasásra nyitja meg a bináris file-t.
ab+	Hozzáfűzésre és olvasásra nyitja meg bináris file-t.

Ha egy már létezo file-t írásra nyitunk meg akkor a file tartalma elvész. Olvasásra csak létezo file-t nyithatunk meg.

Példa:

```
>>> f = open("c:\\boot.ini")
>>> f.readline()
'[boot loader]\n'
>>> f.close()
```

Vegyük észre hogy az open() egy objektumot ad vissza (OOP-rol bovebben a következo részben), aminek különféle metódusai vannak, amiket az objektum nevétol ponttal elválasztva érhetünk el. Ilyen a close() ami a file bezárására szolgál. Ha nem akarunk több muveletet végezni a file-lal akkor ne felejtsük el mindig bezárni a file-t.
Olvasásra szolgál a readline() metódus ami egy sort olvas, és a read() ami az egész file-t. Van egy másik hasznos olvasásra használható függvény, a readlines(), ami szintén az egész file-t olvassa be, de a sorokat egy tömbbe helyezi. Írni pedig a write() és writelines() metódusokkal tudunk.

Most pedig jöjjön a program.

A felhasználó parancssoros paraméterként tudja megadni a bemeneti szövegfile nevét. A kimeneti file pedig a bemeneti + „.out” lesz. Ezután megnyitjuk a file-t, ha valami hiba lépne fel, akkor azt lekezeljük, majd soronként beolvassuk, a sorok szavait összekeverjük, úgyhogy az elso néhány karaktert a helyén hagyjuk, és kiírjuk a kimeneti file-ba.

Karakterek felcseréléséhez készítsünk egy kis függvényt, ami a megadott indexu karaktert felcseréli.

```
def sXchg(s, a, b) :
 ca = s[a]
 s = s[:a] + s[b] + s[a+1:]
 s = s[:b] + ca + s[b+1:]
 return s
```

Majd visszatér az új stringgel.

Két trim-elo rutinra is szükségünk lehet, ami levágja a szó végérol vagy elejérol a szóköz, illetve sorvége karaktereket.

```
def leftTrim(s) :
 L = len(s)
 i = 0
 while i < L and (s[i] == ' ' or s[i] == '\n'
or s[i] == '\r') : i+=1
 s = s[i:]
 return s
```

Most pedig jöjjön a teljes forrás.

import random, sys # a sys a paraméterek kezeléséhez kell, a random pedig a random függvényhez

```
def sXchg(s, a, b) :
 ca = s[a]
 s = s[:a] + s[b] + s[a+1:]
 s = s[:b] + ca + s[b+1:]
 return s

def leftTrim(s) :
 L = len(s)
 i = 0
 while i < L and (s[i] == ' ' or s[i] == '\n' or
s[i] == '\r') : i+=1
 s = s[i:]
 return s

def rightTrim(s) :
 L = len(s)
 i = L - 1
 while i > 0 and (s[i] == ' ' or s[i] == '\n' or
s[i] == '\r') : i-=1
 s = s[:i+1]
 return s

# Ez keveri össze a karaktereket. Ha kisebb a szó
hossza 3-nál akkor persze nem #csinál semmit.
def mixWord(s) :
 if len (s) <= 3 : return s
 m = s
 i = 0
# ha a szó végén pont, vesszo vagy más a szóhoz
tartozó karakter lenne, azt #békén hagyjuk.
 while (not s[i].isalnum()) and i<len(s) : i+=1
 bi = i + 1
 i = len(s) - 1
 ei = 1
 while (not s[i].isalnum()) and i>0 :
  ei+=1
  i -=1
 if len(s) - ei - bi < 2 : return s
#ha túl sok ilyen karakter van és a cserélheto
rész hossza túl kicsi akkor #visszatérünk

#Ha a cserélheto rész túl hosszú akkor több
karaktert hagyunk meg az elején és a #végén
 if len(s)-ei-bi > 11 :
  bi += 2
  ei += 2
 elif len(s)-ei-bi > 7 :
  bi += 2
  ei += 2
#Ez pedig a kavarás, ami össze cseréli a
karaktereket.
 for i in range(bi, len(s)-ei) :
  r = random.randint(bi, len(s)-(ei+1))
  while r == i : r = random.randint(bi, len(s)-
(ei+1))
  if m[r] == s [r] :
   s = sXchg(s, i, r)
 return s

#Innen kezdodik a foprogram
# a sys.argv egy tömb amiben a parancssoros
paraméterek vannak.
if len(sys.argv)<2 :
 print "Usage: wordmix filename"
else :
 fname = sys.argv[1] #elso paraméter a filename
 print 'Opening',fname
 try:
  fi = open(fname, 'r')
  fo = open(fname+'.out', 'w')
 except Exception :
 #kivételkezelés ami a file megnyitás közben
fellépo hibákat kezeli le, errol #bovebben késobb
  print 'File not found'
  exit
 print 'Reading',fname
 print 'Writing',fname+'.out'
 line = fi.readline()
# a file beolvasása most ciklussal történik,
lehetett volna itt is a readlines() #metódust
használni de nagy fileoknál talán így elonyösebb.
 while line :
  words = line.split(" ")
  for i in words :
   w = i
   ends = ' '
   if w.find('\n') != -1 : ends = '\n'
   w = leftTrim(w)
   w = rightTrim(w)
   fo.write(mixWord(w)+ends)
  line = fi.readline()
 fo.close()
 fi.close()
 print 'Done.'
```

A továbbiakban egy-két gyakran használt algoritmust fogok bemutatni.

Gyakran használt algoritmusok:

Ilyen például a rendezés vagy a keresés. Habár a legtöbb nyelvnél van arra is lehetoség hogy beépített függvényeket használjunk erre, de azért illik ismerni ezeket az eljárásokat.

Keresés:

Ha egy tömbben vagy más lineáris adatstruktúrában meg szeretnénk keresni egy elemet, akkor egyszeruen végig kell gyalogolnunk az adatfolyamon, és összehasonlítani az aktuális elemet a keresettel. Ez az ún. lineáris keresés, ami nagy tömböknél elég lassú. Ha rendezett a tömbünk, vagyis növekvo, vagy csökkeno számokat tartalmaz, akkor használhatunk bináris keresést is. Ezzel a tömb rendezettségét kihasználva, gyorsabb eredményre jutunk mint a lineáris kereséssel. Ilyenkor megfelezzük a tömböt, és eldöntjük hogy a keresett elem, a tömb melyik felében található (ha egyáltalán benne van). Majd azt a felét megint megfelezzük addig amig már csak egy elem maradt.

Ennek python megvalósítása:

```
t = [5, 8, 10 ,20 ,36 ,48 ,157 ,651]
def binfind(array, n) :
 height = len(array)-1
 low   = 0
 done = 0
 while not done and low<=height:
  mid = (height + low) / 2
  if n < array[mid] :
   height = mid -1
  elif n > array[mid] :
   low = mid + 1
  else : done = 1
 if array[mid] == n :
  return mid
 else : return -1

print binfind(t, 157)
raw_input("ok")
```

A kimenete:
6
ok

Nem szabad elfelejtenünk hogy ez a keresés csak rendezett tömbökre használható, és az sem mindegy hogy növekvo vagy csökkeno a tömb.

Rendezések:

Ilyen algoritmusokból is létezik párfajta, vannak lassabbak és gyorsabbak. Az egyik legegyszerubb, bár elég lassú rendezés, az úgynevezett buborék rendezés. Ennél a rendezésnél végiggyalogolunk a tömbön, és ha az aktuális elem kisebb (vagy nagyobb, attól függ hogyan rendezzük), az akuális + 1-edik elemnél (tehát a szomszédos elemeket hasonlítgatjuk) akkor megcséréljük a kettot. Az egészet addig folytatjuk amig egyetlen egy elemet sem kell megcserélni.

Megvalósítása pythonban:

Fontos hogy ne tuple tömböt használjunk hanem listát, hogy meg lehessen változtatni a tömbelemket. Régebbi python verziók még nem támogatták a boolean típust, ezért használok inkább int-et logikai típusként.

```
def bublesort(array) :
 done = 0
 end = 1
 while not done :
  done = 1
  for i in range(len(array)-end) :
   if array[i]>array[i+1] :
    done = 0
    array[i],array[i+1] = array[i+1], array[i]
  end += 1
 return array
t = [24 ,45, 5 ,11 ,4 , 7, 13,]
bublesort(t)
print t
raw_input("ok")
```

Kimenete:
[4, 5, 7, 11, 13, 24, 45]
ok

Az end változó azért van hogy feleslegesen ne mennyünk végig az egész tömbön ha a vége már úgyis rendezett, ugyanis minden ciklusban a legnagyobb elemet vegigcipeljük magunkkal a tömbvégére.

Nézzünk egy más elven muködo algoritmust, a beszúrós rendezést. Ennél a rendezésnél, a kiválasztott elemet, a tömb megfelelo helyére szúrjuk be, közben a tömböt mindig “elscrollozzuk” eggyel.

Python implementációja:

```
t = [24 ,45, 5 ,11 ,4 , 7, 13,]
def insertsort(array):
 for rIndex in range(1, len(array)):
  rValue = array[rIndex]
  iIndex = rIndex
  while iIndex > 0 and array[iIndex - 1] >
rValue:
   array[iIndex] = array[iIndex - 1]
   iIndex = iIndex - 1
  array[iIndex] = rValue

insertsort(t)
print str(t)
raw_input("done")

Eredmény:[4, 5, 7, 11, 13, 24, 45]
done
```

Most pedig jöjjön az egyik leggyorsabb a quicksort, vagyis a találóan elnevezett gyorsrendezés.

Ez egy úgynevezett rekurzív rendezés, a rekurzió tulajdonképpen azt jelenti hogy a függvény saját magát hívja meg.

Pl.

```
def fakt(n) :
   if n>1 : return n * fakt(n-1)
   return n
```

Ez egy faktoriális számoló rutin.

A rekurziónak van némi veszélye, de ez védett módú operációs rendszeren nemigen jön elo, ugyanis a rekurziónál használt vermet az oprendszer képes futásidoben megnövelni. Ennek ellenére a nagyon mély rekúrzió használata kerülendo. A valós módú operációsrendszerek idejében egy végtelen, vagy túl mély rekúrzió sokkal komolyabb gondokat tudott okozni mint egy sima végtelen ciklus. Ezért most is érdemes jobban odafigyelni az ilyen rutinokra.

Egy másik rekurzív rutin, a fibonacci sorozat eloállítására:

```
def fib(n):
  if n < 2:
    return n
  else:
    return fib(n - 2) + fib(n - 1)
print fib(20)
```

Gyorsrendezésnél kiválasztunk egy elemet a tömbbol, és minden egyes maradék elemere megnézzük hogy a kiválasztottnál kisebb vagy nagyobb. A nagyobbakat a kiválasztott elem egyik oldalára, a kisebbeket a másik oldalára helyezzük, majd mindkét oldalon végrehajtjuk ezt a rendezést rekurzívan amig rendezett tömböt nem kapunk. Ez pedig a gyorsrendezés pythonban:

```
def partition(array, start, end):
 while start < end:
  while start < end:
   if array[start] > array[end]:
    (array[start], array[end]) = (array[end],
array[start])
    break
   end = end - 1
  while start < end:
   if array[start] > array[end]:
    (array[start], array[end]) = (array[end],
array[start])
    break
   start = start + 1
 return start

def quicksort(array, start=None, end=None):
 if start is None: start = 0
 if end is None: end = len(array)
 if start < end:
  pivot = partition(array, start, end-1)
  quicksort(array, start, pivot)
  quicksort(array, pivot+1, end)


t = [24 ,45, 5, 54 ,11 ,4 ,32, 7, 13,]
quicksort(t)
print t
input()
```

Eredmény: [4, 5, 7, 11, 13, 24, 32, 45, 54]

A következo részben az objektum orientált programozással fogunk foglalkozni.

Magyar Attila zeroflag@elte.hu