515_03_01_Debugging

November 14, 2024

1 Debugging

1.1 Syntax Errors

Given two lists, one of people's names and another of their scores, create a list of tuples such that for each person you have a tuple of their name and their score.

You might come up with a solution that looks like:

[1]:
```
names = ['a','b','c','d','e']
scores = [90,76,55,82,88]
```

[2]:
```
people_and_scores = []
for i in range(len(names)):
    people_and_scores.append((names[i],scores[i]))
people_and_scores
```

[2]:
```
[('a', 90), ('b', 76), ('c', 55), ('d', 82), ('e', 88)]
```

There's a better way of doing it: the zip command

Let's take a look at the documentation for the zip command: https://docs.python.org/3.5/library/functions.html#zip Hmmmmm. Not all that useful, so let's try it out:

[3]:
```
zip(names,scores)
```

[3]:
```
<zip at 0x70ff08176e80>
```

[4]:
```
for i in zip(names,scores):
    print(i)
```

```
('a', 90)
('b', 76)
('c', 55)
('d', 82)
('e', 88)
```

[5]:
```
people_and_scores2 = []
for i in zip(names,scores):
    people_and_scores2.append(i)
people_and_scores2
```

[5]:
```
[('a', 90), ('b', 76), ('c', 55), ('d', 82), ('e', 88)]
```

[6]:
```
people_and_scores3 = list(zip(names,scores))
people_and_scores3
```

[6]:
```
[('a', 90), ('b', 76), ('c', 55), ('d', 82), ('e', 88)]
```

Ok, but let's say you had a structure that looks like people_and_scores and you wanted to extract just the names. How would you do that?

[7]:
```
names = []
for i in people_and_scores:
    names.append(i[0])
names
```

[7]:
```
['a', 'b', 'c', 'd', 'e']
```

Back to our documentation: https://docs.python.org/3.5/library/functions.html#zip

There's a blurb there about > zip() in conjunction with the * operator can be used to unzip a list:

followed by a code example:

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> list(zipped)
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*zip(x, y))
>>> x == list(x2) and y == list(y2)
True
```

Google for "python zip explained", get https://stackoverflow.com/questions/19339/transpose-unzip-function-inverse-of-zip#19343

[8]:
```
list(list(zip(*people_and_scores))[0])
```

[8]:
```
['a', 'b', 'c', 'd', 'e']
```

Note also, however, that there's a link that talks about using generators:

https://stackoverflow.com/questions/30805000/how-to-unzip-an-iterator

[9]:
```
import itertools
```

[10]:
```
names,scores = itertools.tee(people_and_scores)
```

[11]:
```
names
```

[11]:
```
<itertools._tee at 0x70ff08121240>
```

[12]:
```
for n in names:
    print(n,type(n))
```

```
('a', 90) <class 'tuple'>
('b', 76) <class 'tuple'>
('c', 55) <class 'tuple'>
('d', 82) <class 'tuple'>
('e', 88) <class 'tuple'>
```

[13]:
```
names = (x[0] for x in names)
```

[14]:
```
for n in names:
    print(n,type(n))
```

[15]:
```
for i in names:
    print(i)
```

Ok, let's try this again.

1.1.1 Top-level goal: to create a list of (lat, lon) tuples where lat is between X and Y

We're going to read a file efficiently using a generator:

[16]:
```
filename = 'data/ride_final2.csv'
```

[17]:
```
def read_lat_and_lon_by_line(filename):
    with open(filename) as f:
        while True:
            line = f.readline()
            if not line:
                break
            data = line.split(',')
            yield (data[1],data[2])
```

[18]:
```
f = read_lat_and_lon_by_line(filename)
```

[19]:
```
f
```

[19]:
```
<generator object read_lat_and_lon_by_line at 0x70ff080c9740>
```

[20]:
```
count = 0
for i in read_lat_and_lon_by_line(filename):
```

```
    count = count+1
    if count > 5:
        break
    print(i)
```

```
('"Latitude"', '"Longitude"')
('"504719750"', '"-998493490"')
('"504717676"', '"-998501870"')
('"504716354"', '"-998506792"')
('"504714055"', '"-998515244"')
```

Let's get rid of the first line (the header line):

[21]:
```
def read_lat_and_lon_by_line(filename):
    with open(filename) as f:
        first = True
        while True:
            line = f.readline()
            if first
                line = f.readline()
                first = False
            if not line:
                break
            data = line.split(',')
            yield (data[1],data[2])
```

```
  Cell In[21], line 6
    if first
            ^
SyntaxError: invalid syntax
```

[22]:
```
def read_lat_and_lon_by_line(filename):
    with open(filename) as f:
        first = True
        while True:
            line = f.readline()
            if first:
                line = f.readline()
                first = False
            if not line:
                break
            data = line.split(',')
            yield (data[1],data[2])
```

[23]:
```
count = 0
for i in read_lat_and_lon_by_line(filename):
```

```
    count = count+1
    if count > 5:
        break
    print(i)
```

```
('"504719750"', '"-998493490"')
('"504717676"', '"-998501870"')
('"504716354"', '"-998506792"')
('"504714055"', '"-998515244"')
('"504711900"', '"-998523278"')
```

[24]:
```
((lat,lon) for (lat,lon) in read_lat_and_lon_by_line(filename) if lon <␣
 ↪-998493490 )
```

[24]:
```
<generator object <genexpr> at 0x70ff080c9c10>
```

[25]:
```
import csv
def read_lat_and_lon_with_reader(filename):
    with open(filename, 'r') as csvfile:
        csvreader = csv.DictReader(csvfile)
        for row in csvreader:
            yield (int(row['Latitude']),int(row['Longitude']))
```

[26]:
```
g = ((lat,lon) for (lat,lon) in read_lat_and_lon_with_reader(filename) if lon <␣
 ↪-998493490 )
```

[27]:
```
for r in g:
    print(r)
```

```
(504717676, -998501870)
(504716354, -998506792)
(504714055, -998515244)
(504711900, -998523278)
(504709729, -998531192)
(504707299, -998540018)
(504705967, -998544934)
(504703695, -998553170)
(504701326, -998561924)
(504700547, -998564668)
(504698641, -998571568)
(504696909, -998577942)
(504695977, -998581247)
(504695051, -998584562)
(504692926, -998592970)
(504692111, -998597619)
```

(504691299, -998606407)
(504691173, -998612263)
(504690996, -998620769)
(504690902, -998629455)
(504690919, -998630192)
(504690924, -998630680)
(504691253, -998633926)
(504691172, -998633950)
(504684780, -998633605)
(504677722, -998633704)
(504675542, -998633688)
(504673200, -998633697)
(504666467, -998633752)
(504665599, -998633751)
(504661165, -998633676)
(504656517, -998633739)
(504653635, -998633704)
(504646885, -998633610)
(504645873, -998633596)
(504638364, -998633928)
(504636279, -998634532)
(504635317, -998635011)
(504631201, -998637323)
(504630451, -998638943)
(504630652, -998641677)
(504630408, -998643369)
(504630203, -998644131)
(504629659, -998645557)
(504627046, -998648458)
(504626088, -998648999)
(504625585, -998649172)
(504624528, -998649402)
(504623425, -998649372)
(504618911, -998648141)
(504617469, -998648256)
(504616089, -998648760)
(504614850, -998649629)
(504613462, -998651542)
(504612723, -998653576)
(504612491, -998654944)
(504611674, -998658568)
(504610818, -998659191)
(504609895, -998659600)
(504607616, -998660706)
(504607388, -998663713)
(504606997, -998666603)
(504605105, -998672534)
(504601240, -998678138)

(504595830, -998683008)
(504595132, -998683746)
(504590399, -998690509)
(504587816, -998696975)
(504587080, -998706208)
(504587179, -998715339)
(504587265, -998719519)
(504587183, -998720352)
(504586969, -998723936)
(504586735, -998727648)
(504586387, -998729178)
(504585370, -998729769)
(504582781, -998730391)
(504580448, -998730710)
(504576978, -998731285)
(504574238, -998731748)
(504573319, -998731886)
(504566977, -998732447)
(504565203, -998731969)
(504563563, -998730988)
(504560644, -998728699)
(504555981, -998724473)
(504554045, -998723629)
(504553554, -998723542)
(504551574, -998723663)
(504548714, -998724973)
(504548108, -998725456)
(504546852, -998726669)
(504545724, -998728332)
(504545232, -998729335)
(504544566, -998731564)
(504544377, -998732752)
(504544217, -998735213)
(504544356, -998742165)
(504544234, -998751359)
(504544309, -998759543)
(504544101, -998768597)
(504544041, -998771592)
(504544025, -998776724)
(504544333, -998784923)
(504544341, -998794192)
(504544427, -998799444)
(504544983, -998802253)
(504544879, -998805355)
(504543998, -998810071)
(504543872, -998812354)
(504544678, -998815462)
(504542426, -998816004)

(504541043, -998816334)
(504539288, -998816793)
(504537290, -998817243)
(504536270, -998817589)
(504535254, -998818083)
(504534198, -998818683)
(504531988, -998819812)
(504527193, -998820953)
(504524817, -998821169)
(504520598, -998820403)
(504518779, -998819650)
(504515846, -998818587)
(504514719, -998818352)
(504513882, -998818747)
(504513013, -998819685)
(504510857, -998822649)
(504509650, -998824375)
(504509157, -998825402)
(504508779, -998826550)
(504508485, -998827806)
(504508183, -998829191)
(504507882, -998830663)
(504507635, -998832227)
(504507385, -998835385)
(504507323, -998836862)
(504507179, -998846416)
(504507131, -998847659)
(504507059, -998856665)
(504506827, -998864665)
(504506509, -998871147)
(504506442, -998874137)
(504506533, -998878213)
(504506401, -998882060)
(504506099, -998890767)
(504505826, -998899659)
(504505923, -998905530)
(504505836, -998913680)
(504506079, -998916193)
(504506133, -998916968)
(504506220, -998919172)
(504506229, -998919896)
(504506070, -998921937)
(504505898, -998923284)
(504505768, -998925991)
(504505667, -998929485)
(504505405, -998936042)
(504505258, -998936773)
(504503680, -998942456)

(504502704, -998946170)
(504501817, -998951141)
(504501335, -998954849)
(504500727, -998961059)
(504499739, -998965321)
(504499599, -998968172)
(504499297, -998969496)
(504498760, -998970527)
(504497999, -998971490)
(504496138, -998976217)
(504493350, -998983027)
(504491622, -998986958)
(504490695, -998988979)
(504487074, -998996383)
(504485781, -998998855)
(504483710, -999001310)
(504480227, -999004536)
(504478789, -999005562)
(504477325, -999006359)
(504474132, -999007661)
(504471957, -999008250)
(504469812, -999008505)
(504468125, -999008574)
(504466550, -999008347)
(504465209, -999008140)
(504464054, -999008083)
(504462371, -999007968)
(504460519, -999007568)
(504458080, -999007078)
(504457520, -999007079)
(504455689, -999006455)
(504449175, -999005828)
(504447179, -999005722)
(504444954, -999005667)
(504443214, -999006014)
(504441051, -999005821)
(504439082, -999005422)
(504437717, -999005994)
(504437277, -999007182)
(504437248, -999009066)
(504437192, -999011100)
(504436922, -999017878)
(504433602, -999024353)
(504427084, -999024293)
(504425267, -999024438)
(504424849, -999025357)
(504424585, -999034494)
(504424584, -999038416)

```
(504424374, -999047484)
(504424487, -999051230)
(504425416, -999051046)
(504425984, -999050165)
(504426837, -999050457)
```

Next: debugging with print() statements

For example, let's say there's some bad data in the data file.

Understanding error stacks

Passing reference to pdb, set_trace

PixieDebugger?

1.2 Copy-and-paste errors

From https://datascienceplus.com/how-to-achieve-parallel-processing-in-python-programming/

Copy

```
import multiprocessing as multip
print("Total number of processors on your machine is: ", multip.cpu_count())
```

What's wrong?

[ ]: