Week 4 Programming Assignment

Due on 2020-02-27, 23:59 IST

Write Python functions as specified below. Paste the text for all functions together into the submission window.

- You may define additional auxiliary functions as needed.
- In all cases you may assume that the value passed to the function is of the expected type, so your function does not have to check for malformed inputs.
- For each function, there are some public test cases and some (hidden) private test cases.
- "Compile and run" will evaluate your submission against the public test cases.
- "Submit" will evaluate your submission against the hidden private test cases and report a score on 100. There are 10 private testcases in all, each with equal weightage.
- Ignore warnings about "Presentation errors".

1. We represent scores of batsmen across a sequence of matches in a two level dictionary as follows:

```
{'match1':{'player1':57, 'player2':38}, 'match2':{'player3':9, 'player1':42}, 'match3':{'player2':41, 'player4':63, 'player3':91}
```

Each match is identified by a string, as is each player. The scores are all integers. The names associated with the matches are not fixed (here they are `'match1'`, `'match2'`, `'match3'`), nor are the names of the players. A player need not have a score recorded in all matches.

Define a Python function `orangecap(d)` that reads a dictionary `d` of this form and identifies the player with the highest total score. Your function should return a pair `(playername,topscore)` where `playername` is a

string, the name of the player with the highest score, and `topscore` is an integer, the total score of playername.

The input will be such that there are never any ties for highest total score.

For instance:

```
>>> orangecap({'match1':{'player1':57, 'player2':38}, 'match2':{'playe
r3':9, 'player1':42}, 'match3':{'player2':41, 'player4':63, 'player3':
91}})
('player3', 100)

>>> orangecap({'test1':{'Ashwin':84, 'Kohli':120}, 'test2':{'Ashwin':5
9, 'Pujara':42}})
('Ashwin', 143)
```

2. Let us consider polynomials in a single variable x with integer coefficients. For instance:

$3x^4 - 17x^2 - 3x + 5$

Each term of the polynomial can be represented as a pair of integers (coefficient,exponent). The polynomial itself is then a list of such pairs.

We have the following constraints to guarantee that each polynomial has a unique representation:

- Terms are sorted in descending order of exponent
- No term has a zero cofficient
- No two terms have the same exponent
- Exponents are always nonnegative

For example, the polynomial introduced earlier is represented as:

```
[(3,4),(-17,2),(-3,1),(5,0)]
```

The zero polynomial, 0, is represented as the empty list `[]`, since it has no terms with nonzero coefficients.

Write Python functions for the following operations:

```
addpoly(p1,p2)
multpoly(p1,p2)
```

that add and multiply two polynomials, respectively.

You may assume that the inputs to these functions follow the representation given above. Correspondingly, the outputs from these functions should also obey the same constraints.

You can write auxiliary functions to "clean up" polynomials – e.g., remove zero coefficient terms, combine like terms, sort by exponent etc. Build a library of functions that can be combined to achieve the desired format.

You may also want to convert the list representation to a dictionary representation and manipulate the dictionary representation, and then convert back.

Some examples:

```
>>> addpoly([(4,3),(3,0)],[(-4,3),(2,1)])
[(2, 1),(3, 0)]

Explanation: (4x^3 + 3) + (-4x^3 + 2x) = 2x + 3

>>> addpoly([(2,1)],[(-2,1)])
[]

Explanation: 2x + (-2x) = 0

>>> multpoly([(1,1),(-1,0)],[(1,2),(1,1),(1,0)])
[(1, 3),(-1, 0)]

Explanation: (x - 1) * (x^2 + x + 1) = x^3 - 1
```

Sample Test Cases

	Input	Output
Test Case 1	`orangecap({'match1':{'player1':57, 'player2':38}, 'match2':{'player3':9, 'player1':42}, 'match3': {'player2':41, 'player4':63, 'player3':91}, 'match4': {'player2':31, 'player4':73, 'player3':88}})`	`('player3', 188)`
Test Case 2	`orangecap({'match1':{'player1':38, 'player2':49}, 'match2':{'player3':99, 'player1':32}, 'match3': {'player2':56, 'player4':99, 'player3':89}, 'match4': {'player2':11, 'player4':123, 'player3':48}})`	`('player3', 236)`
Test Case 3	`orangecap({'test1':{'Ashwin':84, 'Kohli':120}, 'test2': {'Ashwin':59, 'Pujara':42},'test3':{'Rahul':48, 'Shreyas':120}, 'test4':{'Rahul':59, 'Pujara':42}})`	`('Ashwin', 143)`
Test Case 4	`orangecap({'test1':{'Ashwin':48, 'Kohli':20}, 'test2': {'Ashwin':39, 'Pujara':24},'test3':{'Rahul':84, 'Shreyas':95}, 'test4':{'Rahul':39, 'Pujara':94}})`	`('Rahul', 123)`
Test Case 5	`addpoly([(5,3),(3,1)],[(-4,3),(-2,1)])`	`[(1, 3), (1, 1)]`
Test Case 6	`addpoly([],[(1,1)])`	`[(1, 1)]`
Test Case 7	`addpoly([(5,4),(3,2)],[(-4,1),(-2,0)])`	`[(5, 4), (3, 2), (-4, 1), (-2, 0)]`

Mar 2021, 10:00-12:00 ()

Online Programming Test 2, 09 Mar 2021, 20:00-22:00 ()

Test Case 8	`multpoly([(3,1),(-2,0)],[(4,2),(7,1),(11,0)])`	`[(12, 3), (13, 2), (19, 1), (-22, 0)]`
Test Case 9	`multpoly([(1,1),(1,0)],[(1,1),(-1,0)])`	`[(1, 2), (-1, 0)]`
Test Case 10	`multpoly([(3,1),(-2,0)],[])`	`[]`
Test Case 11	`orangecap({'match1':{'player1':57, 'player2':38}, 'match2':{'player3':9, 'player1':42}, 'match3': {'player2':41, 'player4':63, 'player3':91}})`	`('player3', 100)`
Test Case 12	`orangecap({'test1':{'Ashwin':84, 'Kohli':120}, 'test2': {'Ashwin':59, 'Pujara':42}})`	`('Ashwin', 143)`
Test Case 13	`addpoly([(4,3),(3,0)],[(-4,3),(2,1)])`	`[(2, 1), (3, 0)]`
Test Case 14	`addpoly([(2,1)],[(-2,1)])`	`[]`
Test Case 15	`multpoly([(1,1),(-1,0)],[(1,2),(1,1),(1,0)])`	`[(1, 3), (-1, 0)]`

The due date for submitting this assignment has passed.
As per our records you have not submitted this assignment.
Sample solutions (Provided by instructor)

```
def orangecap(d):
  total = {}
  for k in d.keys():
    for n in d[k].keys():
      if n in total.keys():
        total[n] = total[n] + d[k][n]
      else:
        total[n] = d[k][n]

  maxtotal = -1
  for n in total.keys():
    if total[n] > maxtotal:
      maxname = n
      maxtotal = total[n]

  return(maxname,maxtotal)

# Dictionary is better than list: use exponent as key so only one term per ex

# listtodict and dicttolist convert back and forth between representations

def listtodict(poly):
  dpoly = {}
  for term in poly:
    coeff = term[0]
    exp = term[1]
    dpoly[exp] = coeff
  return(dpoly)

```

```
def dicttolist(dpoly):
  lpoly = []
  for exp in sorted(dpoly.keys()):
    lpoly.append((dpoly[exp],exp))
  lpoly.reverse()
  return(lpoly)

# dpolyadd: initialize sum to dpoly1 and either update term or add a new term

def dpolyadd (dpoly1,dpoly2):
  sumpoly = {}
  for exp in dpoly1.keys():
    sumpoly[exp] = dpoly1[exp]

  for exp in dpoly2.keys():
    if exp in sumpoly.keys():
      sumpoly[exp] = sumpoly[exp] + dpoly2[exp]
    else:
      sumpoly[exp] = dpoly2[exp]

  return(sumpoly)

# dpolymult: compute each cross term and update result multpoly

def dpolymult (dpoly1,dpoly2):
  multpoly = {}
  for exp1 in dpoly1.keys():
    for exp2 in dpoly2.keys():
      newexp = exp1 + exp2
      newcoeff = dpoly1[exp1] * dpoly2[exp2]
      if newexp in multpoly.keys():
        multpoly[newexp] = multpoly[newexp] + newcoeff
      else:
        multpoly[newexp] = newcoeff
  return(multpoly)

# Remove 0 coefficient terms

def cleanup(dpoly):
  dpolyclean = {}
  for exp in dpoly.keys():
    if dpoly[exp] != 0:
      dpolyclean[exp] = dpoly[exp]
  return(dpolyclean)

# Convert to dictionary, apply operations on dictionaries, convert back

def addpoly(p1,p2):
  d1 = listtodict(p1)
  d2 = listtodict(p2)
  res = dpolyadd(d1,d2)
  return(dicttolist(cleanup(res)))

def multpoly(p1,p2):
  d1 = listtodict(p1)
  d2 = listtodict(p2)
  res = dpolymult(d1,d2)
  return(dicttolist(cleanup(res)))


# Hidden code below

import ast

def todict(inp):
    inp = ast.literal_eval(inp)
    return (inp)

def topairoflists(inp):
    inp = "["+inp+"]"
    inp = ast.literal_eval(inp)
    return (inp[0],inp[1])

def tostring(s):
  lquote = s.find('"')
  rquote = s.rfind('"')
  return(s[lquote+1:rquote])
```

```
107
108 def tolist(s):
109   lbrack = s.find('[')
110   rbrack = s.rfind(']')
111   slist = s[lbrack+1:rbrack].split(',')
112   if slist == ['']:
113     slist = []
114   else:
115     for i in range(0,len(slist)):
116       slist[i] = int(slist[i])
117   return(slist)
118
119 fncall = input()
120 lparen = fncall.find("(")
121 rparen = fncall.rfind(")")
122 fname = fncall[:lparen]
123 farg = fncall[lparen+1:rparen]
124
125 if fname == "orangecap":
126    arg = todict(farg)
127    print(orangecap(arg),end="")
128 elif fname == "addpoly":
129    (arg1,arg2) = topairoflists(farg)
130    print(addpoly(arg1,arg2),end="")
131 elif fname == "multpoly":
132    (arg1,arg2) = topairoflists(farg)
133    print(multpoly(arg1,arg2),end="")
134 else:
135    print("Function", fname, "unknown")
136
137
```