X

Practice Assignment 2

Due on 2021-12-31, 23:59 IST

Write five Python functions as specified below. Copy and paste the text for all five functions together into the submission window. Your function will be called automatically with various inputs and should return values as specified. Do not write commands to read any input or print any output.

- 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 normally 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. There are 20 private test cases, with equal weightage. You will get feedback about which private test cases pass or fail, though you cannot see the actual test cases.
- Ignore warnings about "Presentation errors".
- You can submit as many times as you like. Your final submission will be used for scoring.

1. Function: `counthv(l)`

   In a list of integers `l`, the neighbours of `l[i]` are `l[i-1]` and `l[i+1]`. `l[i]` is a *hill* if it is strictly greater than its neighbours and a *valley* if it is strictly less than its neighbours.

   Write a function `counthv(l)` that takes as input a list of integers `l` and returns a list `[hc,vc]` where `hc` is the number of hills in `l` and `vc` is the number of valleys in `l`.

due Fri 12 Nov 2021

Quiz 2, Mon 8 Nov 2021

PDSP Assignment 3, due Wed 24 Nov 2021

PDSP Assignment 4, due Fri 17 Dec 2021

Quiz 3, Thu 16 Dec 2021

PDSP Quiz 4, Thu 23 Dec 2021

PDSP Assignment 5, due Fri 31 Dec 2021

Here are some examples to show how your function should work.

```
>>> counthv([1,2,1,2,3,2,1])
[2, 1]

>>> counthv([1,2,3,1])
[1, 0]

>>> counthv([3,1,2,3])
[0, 1]
```

2. Function: `rotatelist(l,n)`

   A list rotation consists of taking the first element and moving it to the end. For instance, if we rotate the list `[1,2,3,4,5]`, we get `[2,3,4,5,1]`. If we rotate it again, we get `[3,4,5,1,2]`.

   Write a Python function `rotatelist(l,k)` that takes a list `l` and a positive integer `k` and returns the list `l` after `k` rotations. If `k` is not positive, your function should return `l` unchanged. Note that your function should not change `l` itself, and should return the rotated list.

   Here are some examples to show how your function should work.

```
>>> rotatelist([1,2,3,4,5],1)
[2, 3, 4, 5, 1]

>>> rotatelist([1,2,3,4,5],3)
[4, 5, 1, 2, 3]

>>> rotatelist([1,2,3,4,5],12)
[3, 4, 5, 1, 2]
```

3. Function: `transpose(m)`

   A two dimensional matrix can be represented in Python row-wise, as a list of lists: each inner list represents one row of the matrix. For instance, the matrix

```
1  2  3  4
5  6  7  8
```

   would be represented as `[[1, 2, 3, 4], [5, 6, 7, 8]]`.

   The transpose of a matrix converts each row into a column. The transpose of the matrix above is:

```
1  5
2  6
3  7
4  8
```

which would be represented as `[[1, 5], [2, 6], [3, 7], [4, 8]]`.

Write a Python function `transpose(m)` that takes as input a two dimensional matrix `m` and returns the transpose of `m`. The argument `m` should remain undisturbed by the function.

Here are some examples to show how your function should work. You may assume that the input to the function is always a non-empty matrix.

```
>>> transpose([[1,2,3],[4,5,6]])
[[1, 4], [2, 5], [3, 6]]

>>> transpose([[1],[2],[3]])
[[1, 2, 3]]

>>> transpose([[3]])
[[3]]
```

4. Function: `addpoly(p1,p2), multpoly(p1,p2)`

   Let us consider polynomials in a single variable `x` with integer coefficients: for instance, $3x^4 - 17x^2 - 3x + 5$. We can represent a polynomial as a dictionary in which each element represents one term: the key is the exponent and the value is the coefficient. For instance, the polynomial above would be represented as `{4:3, 2:-17, 1:-3, 0:5}`.

   To ensure that each polynomial has a unique representation, we insist that no term has a zero coefficient (so every key in the dictionary should have a non-zero value) and exponents are always nonnegative (so no key in the dictionary is negative). The zero polynomial, `0`, is represented as the empty dictionary `{}`, 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.

   Here are some examples to show how your functions should behave:

```
>>> addpoly({3:4,0:3},{3:-4,1:2})
{1:2,0:3}
```

```
{1:2,0:3}

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

>>> addpoly({1:2},{1:-2})
{}

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

>>> multpoly({1:2,0:-1},{1:2,0:1})
{2:4,0:-1}

Explanation: (2x + 1) * (2x - 1) = 4x^2 - 1

>>> multpoly({1:1,0:-1},{2:1,1:1,0:1})
{3:1,0:-1}

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

Sample Test Cases

	Input	Output
Test Case 1	counthv([23,44,22,1,26,10])	[2, 1]
Test Case 2	counthv([23,44,22,1,5,1])	[2, 1]
Test Case 3	counthv([1,10,2,11,3,12,4,13,5,14,6])	[5, 4]
Test Case 4	counthv([1,10,2,11,3,12,4,13,5,14,23])	[4, 4]
Test Case 5	counthv([12,55,22,88,40])	[2, 1]
Test Case 6	rotatelist([1,2,3,4,5,6,7,8],9)	[2, 3, 4, 5, 6, 7, 8, 1]
Test Case 7	rotatelist([1,2,3,4,5,6,7,8],2)	[3, 4, 5, 6, 7, 8, 1, 2]
Test Case 8	rotatelist([1,2,3,4,5,6,7,8],19)	[4, 5, 6, 7, 8, 1, 2, 3]
Test Case 9	rotatelist([1,2,3,4,5,6,7,8],300)	[5, 6, 7, 8, 1, 2, 3, 4]
Test Case 10	rotatelist([1,2,3,4,5,6,7,8],24)	[1, 2, 3, 4, 5, 6, 7, 8]
Test Case 11	transpose([[1,2,3],[4,5,6],[7,8,9]])	[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
Test Case 12	transpose([[1,2,3,4]])	[[1], [2], [3], [4]]
Test Case 13	transpose([[1],[2],[3],[4]])	[[1, 2, 3, 4]]

Test Case 14	`transpose([[1,0,0],[0,1,0],[0,0,1]])`	`[[1, 0, 0], [0, 1, 0], [0, 0, 1]]`
Test Case 15	`addpoly({3:5,1:3},{3:-4,1:-2})`	`{3: 1, 1: 1}`
Test Case 16	`addpoly({},{1:1})`	`{1: 1}`
Test Case 17	`addpoly({4:5,2:3},{1:-4,0:-2})`	`{4: 5, 2: 3, 1: -4, 0: -2}`
Test Case 18	`multpoly({1:3,0:-2},{2:4,1:7,0:11})`	`{3: 12, 2: 13, 1: 19, 0: -22}`
Test Case 19	`multpoly({1:1,0:1},{1:1,0:-1})`	`{2: 1, 0: -1}`
Test Case 20	`multpoly({1:3,0:-2},{})`	`{}`
Test Case 21	`counthv([1,2,1,2,3,2,1])`	`[2, 1]`
Test Case 22	`counthv([1,2,3,1])`	`[1, 0]`
Test Case 23	`counthv([3,1,2,3])`	`[0, 1]`
Test Case 24	`rotatelist([1,2,3,4,5],1)`	`[2, 3, 4, 5, 1]`
Test Case 25	`rotatelist([1,2,3,4,5],3)`	`[4, 5, 1, 2, 3]`
Test Case 26	`rotatelist([1,2,3,4,5],12)`	`[3, 4, 5, 1, 2]`
Test Case 27	`transpose([[1,2,3],[4,5,6]])`	`[[1, 4], [2, 5], [3, 6]]`
Test Case 28	`transpose([[1],[2],[3]])`	`[[1, 2, 3]]`
Test Case 29	`transpose([[3]])`	`[[3]]`
Test Case 30	`addpoly({3:4,0:3},{3:-4,1:2})`	`{1: 2, 0: 3}`
Test Case 31	`addpoly({1:2},{1:-2})`	`{}`
Test Case 32	`multpoly({1:2,0:-1},{1:2,0:1})`	`{2: 4, 0: -1}`
Test Case 33	`multpoly({1:1,0:-1},{2:1,1:1,0:1})`	`{3: 1, 0: -1}`

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 counthv(l):
    hills = 0
```

```
    valleys = 0
    for i in range(1,len(l)-1):
        if l[i] > l[i-1] and l[i] > l[i+1]:
            hills = hills + 1
        if l[i] < l[i-1] and l[i] < l[i+1]:
            valleys = valleys + 1
    return([hills,valleys])

##########

def rotatelist(l,k):
  retlist = l[:]

  if k <= 0:
    return(retlist)

  for i in range(1,k+1):
    retlist = retlist[1:] + [retlist[0]]
  return(retlist)

##########

def transpose(l):
  outl = []
  for row in l[:1]:
    for i in range(len(row)):
      outl.append([])
  for row in l:
    for i in range(len(row)):
      outl[i].append(row[i])
  return(outl)

##########

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


def addpoly(p1,p2):
    presult = p1
    for exp in p2.keys():
        try:
            presult[exp] = presult[exp] + p2[exp]
        except KeyError:
            presult[exp] = p2[exp]
    return(cleanup(presult))

def multpoly(p1,p2):
    presult = {}
    for exp1 in p1.keys():
        for exp2 in p2.keys():
            exp = exp1+exp2
            try:
                presult[exp] = presult[exp] + p1[exp1]*p2[exp2]
            except KeyError:
                presult[exp] = p1[exp1]*p2[exp2]
    return(cleanup(presult))

##########


def printpoly(p):
    newp = {}
    for exp in sorted(p.keys(), reverse=True):
        newp[exp] = p[exp]
    print(newp)
    return()

import ast

def tolist(inp):
  inp = "["+inp+"]"
  inp = ast.literal_eval(inp)
```

```
  return (inp[0],inp[1])

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

fncall = input().strip()
lparen = fncall.find("(")
rparen = fncall.rfind(")")
fname = fncall[:lparen]
farg = fncall[lparen+1:rparen]

if fname == "counthv":
  arg = parse(farg)
  print(counthv(arg))
elif fname == "rotatelist":
  (l,k) = parse(farg)
  savel = l[:]
  rotl = rotatelist(l,k)
  if l == savel:
    print(rotl)
  else:
    print("Side effect")
elif fname == "transpose":
  m = parse(farg)
  savem = m[:]
  transm = transpose(m)
  if m == savem:
    print(transm)
  else:
    print("Side effect")
elif fname == "addpoly":
  (d1,d2) = parse(farg)
  printpoly(addpoly(d1,d2))
elif fname == "multpoly":
  (d1,d2) = parse(farg)
  printpoly(multpoly(d1,d2))
else:
  print("Function", fname, "unknown")

```