WEEK 1 Programming, Data Structures And Algorithms Using Python

Week 1 QUIZ

1. What does `h(27993)` return for the following function definition?

```
def h(x):
    (d,n) = (1,0)
    while d <= x:
        (d,n) = (d*3,n+1)
    return(n)
```

Answer. 10

2. What is `g(60) - g(48)`, given the definition of `g` below?

```
def g(n):
    s=0
    for i in range(2,n):
        if n%i == 0:
           s = s+1
    return(s)
```

Answer. 2

3. Consider the following function `f`.

```
def f(n):
    s=0
    for i in range(1,n+1):
        if n//i == i and n%i == 0:
           s = 1
    return(s%2 == 1)
```

The function `f(n)` given above returns `True` for a positive number `n` if and only if:

a. `n` is an odd number.
b. `n` is a prime number.
c. `n` is a perfect square.
d. `n` is a composite number.

Answer. `n` is a perfect square.

4. Consider the following function `foo`.

```
def foo(m):
    if m == 0:
      return(0)
    else:
      return(m+foo(m-1))
```

Which of the following is correct?

a. The function always terminates with foo(n) = factorial of n
b. The function always terminates with foo(n) = n(n+1)/2
c. The function terminates for nonnegative n with foo(n) = factorial of n
d. The function terminates for nonnegative n with foo(n) = n(n+1)/2

Answer. The function terminates for nonnegative n with foo(n) = n(n+1)/2

WEEK 2 Programming, Data Structures And Algorithms Using Python

Week 2 QUIZ

1. One of the following 10 statements generates an error. Which one? (Your answer should be a number between 1 and 10.)

```
x = [[3,5],"mimsy",2,"borogove",1]  # Statement 1
y = x[0:50]                          # Statement 2
z = y                                # Statement 3
w = x                                # Statement 4
x[1] = x[1][:5] + 'ery'              # Statement 5
y[1] = 4                             # Statement 6
w[1][:3] = 'fea'                     # Statement 7
z[4] = 42                            # Statement 8
x[0][0] = 5555                       # Statement 9
a = (x[3][1] == 1)                   # Statement 10
```

Answer. Statement 6

2. Consider the following lines of Python code.

```
b = [43,99,65,105,4]
a = b[2:]
d = b[1:]
c = b
d[1] = 95
b[2] = 47
c[3] = 73
```

Which of the following holds at the end of this code?

a. `a[0] == 47, b[3] == 73, c[3] == 73, d[1] == 47`
b. `a[0] == 65, b[3] == 105, c[3] == 73, d[1] == 95`
c. `a[0] == 65, b[3] == 73, c[3] == 73, d[1] == 95`
d. `a[0] == 95, b[3] == 73, c[3] == 73, d[1] == 95`

Answer. `a[0] == 65, b[3] == 73, c[3] == 73, d[1] == 95`

3. What is the value of `endmsg` after executing the following lines?

```
startmsg = "anaconda"
endmsg = ""
for i in range(1,1+len(startmsg)):
  endmsg = endmsg + startmsg[-i]
```

Answer. “adnocana”

4. What is the value of `mylist` after the following lines are executed?

```
def mystery(l):
  l = l[2:]
  return(l)

mylist = [7,11,13,17,19,21]
mystery(mylist)
```

Answer. [13, 17, 19, 21]

WEEK 3 Programming, Data Structures And Algorithms Using Python

Week 3 Programming Assignment

```
>>> contracting([9,2,7,3,1])
True
>>> contracting([-2,3,7,2,-1])
False
>>> contracting([10,7,4,1])
False
```

1. 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`.
   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. A square n×n matrix of integers can be written in Python as a list with n elements, where each element is in turn a list of n integers, representing a row of the matrix. For instance, the matrix

```
1  2  3
4  5  6
7  8  9
```

would be represented as `[[1,2,3], [4,5,6], [7,8,9]]`.

Write a function `leftrotate(m)` that takes a list representation m of a square matrix as input, and returns the matrix obtained by rotating the original matrix counterclockwize by 90 degrees. For instance, if we rotate the matrix above, we get

```
3  6  9

2  5  8

1  4  7
```

Your function should *not* modify the argument m provided to the function `rotate()`.

Here are some examples of how your function should work.

```
>>> leftrotate([[1,2],[3,4]])

[[2, 4], [1, 3]]

>>> leftrotate([[1,2,3],[4,5,6],[7,8,9]])

[[3, 6, 9], [2, 5, 8], [1, 4, 7]]

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

[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
```

Answers:

```
def contracting(l):
    if len(l) < 3:
        return(True)
    return((abs(l[1]-l[0]) > abs(l[2]-l[1])) and contracting(l[1:]))

def contracting_iterative(l):
    if len(l) < 3:
        return(True)
    for i in range(len(l)-2):
        diff = abs(l[i+1]-l[i])
        if diff <= abs(l[i+2]-l[i+1]):
            return(False)
    return(True)
```

```
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 leftrotate(m):
    size = len(m)
    rotated_m = []
    for i in range(size):
        rotated_m.append([])
    for c in range(size-1,-1,-1):
        for r in range(size):
            rotated_m[size-(c+1)].append(m[r][c])
    return(rotated_m)

import ast

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

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

if fname == "contracting":
  arg = parse(farg)
  print(contracting(arg))

if fname == "counthv":
  arg = parse(farg)
  print(counthv(arg))

if fname == "leftrotate":
  arg = parse(farg)
  savearg = arg
  ans = leftrotate(arg)
  if savearg == arg:
    print(ans)
  else:
    print("Side effect")
```

WEEK 4 Programming, Data Structures And Algorithms Using Python

Week 4 QUIZ

1. Consider the following Python function.

```
def mystery(l):
    if l == []:
        return(l)
    else:
        return(mystery(l[1:])+l[:1])
```

What does `mystery([22,14,19,65,82,55])` return?

Correct Answer is [55,82,65,19,14,22]

2. What is the value of `pairs` after the following assignment?

```
pairs = [ (x,y) for x in range(4,1,-1) for y in range(5,1,-1) if (x+y)%3 == 0 ]
```

Correct Answer is

3. Consider the following dictionary.

```
wickets = {"Tests":{"Bumrah":[3,5,2,3],"Shami":[4,4,1,0],"Ashwin":[2,1,7,4]},"ODI":{"Bumrah":[2,0],"Shami":[1,2]}}
```

Which of the following statements does not generate an error?

1. `wickets["ODI"]["Ashwin"][0:] = [4,4]`
2. `wickets["ODI"]["Ashwin"].extend([4,4])`
3. `wickets["ODI"]["Ashwin"] = [4,4]`
4. `wickets["ODI"]["Ashwin"] = wickets["ODI"]["Ashwin"] + [4,4]`

Correct Answer is

`wickets["ODI"]["Ashwin"] = [4,4]`

5. Assume that hundreds has been initialized as an empty dictionary:

```
hundreds = {}
```

Which of the following generates an error?

1. `hundreds["Tendulkar, international"] = 100`
2. `hundreds["Tendulkar"] = {"international":100}`
3. `hundreds[("Tendulkar","international")] = 100`
4. `hundreds[["Tendulkar","international"]] = 100`

Correct Answer is

`hundreds[["Tendulkar","international"]] = 100`

Week 4 Programming Assignment

1. Write a Python function `frequency(l)` that takes as input a list of integers and returns a pair of the form `(minfreqlist,maxfreqlist)` where `minfreqlist` is a list of numbers with minimum frequency in `l`, sorted in ascending order `maxfreqlist` is a list of numbers with maximum frequency in `l`, sorted in ascending order.

   Here are some examples of how your function should work.

```
>>> frequency([13,12,11,13,14,13,7,11,13,14,12])
([7], [13])
>>> frequency([13,12,11,13,14,13,7,11,13,14,12,14,14])
([7], [13, 14])
>>> frequency([13,12,11,13,14,13,7,11,13,14,12,14,14,7])
([7, 11, 12], [13, 14])
```

2. An airline has assigned each city that it serves a unique numeric code. It has collected information about all the direct flights it operates, represented as a list of pairs of the form `(i,j)`, where `i` is the code of the starting city and `j` is the code of the destination.

   It now wants to compute all pairs of cities connected by one intermediate hope — city `i` is connected to city `j` by one intermediate hop if there are direct flights of the form `(i,k)` and `(k,j)` for some other city `k`. The airline is only interested in one hop flights between different cities — pairs of the form `(i,i)` are not useful.

   Write a Python function `onehop(l)` that takes as input a list of pairs representing direct flights, as described above, and returns a list of all pairs `(i,j)`, where `i != j`, such that `i` and `j` are connected by one hop. Note that it may already be the case that there is a direct flight from `i` to `j`. So long as there is an intermediate `k` with a flight from `i` to `k` and from `k` to `j`, the list returned by the function should include `(i,j)`. The input list may be in any order. The pairs in the output list should be in lexicographic (dictionary) order. Each pair should be listed exactly once.

   Here are some examples of how your function should work.

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

Answers:

```
def frequency(l):
    count = {}
    for n in l:
        if n in count.keys():
            count[n] = count[n]+1
        else:
            count[n] = 1
    minlist = findmin(count)
    maxlist = findmax(count)
    return((minlist,maxlist))

def findmin(d):
    upperbound = 0
    for n in d.keys():
        if d[n] > upperbound:
            upperbound = d[n]
    minlist = []
    mincount = upperbound

    for n in d.keys():
        if d[n] < mincount:
            minlist = [n]
            mincount = d[n]
        elif d[n] == mincount:
            minlist.append(n)
    return(sorted(minlist))


def findmax(d):
    maxlist = []
    maxcount = 0

    for n in d.keys():
        if d[n] > maxcount:
            maxlist = [n]
            maxcount = d[n]
```

```
        elif d[n] == maxcount:
            maxlist.append(n)
    return(sorted(maxlist))

####################

def onehop(l):
    direct = {}
    for (i,j) in l:
        if i in direct.keys():
            direct[i].append(j)
        else:
            direct[i] = [j]

    hopping = []

    for src in direct.keys():
        for dest in direct[src]:
            if dest in direct.keys():
                for remote in direct[dest]:
                    if src != remote:
                        hopping.append((src,remote))

    return(remdup(sorted(hopping)))

def remdup(l):
    if len(l) < 2:
        return(l)

    if l[0] != l[1]:
        return(l[0:1]+remdup(l[1:]))
    else:
        return(remdup(l[1:]))

####################

import ast

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

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

if fname == "frequency":
  arg = parse(farg)
  print(frequency(arg))

if fname == "onehop":
  arg = parse(farg)
  print(onehop(arg))
```