Week 2 Programming Assignment

Due on 2020-02-15, 23:59 IST

Write three Python functions as specified below. Paste the text for all three 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 12 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".

1. A positive integer m can be expresseed as the sum of three squares if it is of the form p + q + r where p, q, r ≥ 0, and p, q, r are all perfect squares. For instance, 2 can be written as 0+1+1 but 7 cannot be expressed as the sum of three squares. The first numbers that cannot be expressed as the sum of three squares are 7, 15, 23, 28, 31, 39, 47, 55, 60, 63, 71, … (see Legendre's three-square theorem (https://en.wikipedia.org/wiki/Legendre%27s_three-square_theorem)).

   Write a Python function `threesquares(m)` that takes an integer `m` as input and returns `True` if `m` can be expressed as the sum of three sq and `False` otherwise. (If `m` is not positive, your function should return `False`.)

(/noc20_cs26/progassignment?name=92)

Week 3: Lists, inductive function definitions, sorting ()

Week 3 Programming Assignment ()

Week 4: Sorting, Tuples, Dictionaries, Passing Functions, List Comprehension ()

Week 4 Quiz ()

Week 4 Programming Assignment ()

Week 5: Exception handling, input/output, file handling, string processing ()

Week 5 Programming Assignment ()

Week 6: Backtracking, scope, data structures; stacks, queues and heaps ()

[False.]

Here are some examples of how your function should work.

```
>>> threesquares(6)
True

>>> threesquares(188)
False

>>> threesquares(1000)
True
```

2. Write a function `repfree(s)` that takes as input a string `s` and checks whether any character appears more than once. The function should return `True` if there are no repetitions and `False` otherwise.

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

```
>>> repfree("zb%78")
True

>>> repfree("(7)(a")
False

>>> repfree("a)*(?")
True

>>> repfree("abracadabra")
False
```

3. A list of numbers is said to be a hill if it consists of an ascending sequence followed by a descending sequence, where each of the sequences is of length at least two. Similarly, a list of numbers is said to be a valley hill if it consists of an descending sequence followed by an ascending sequence. You can assume that consecutive numbers in the input sequence are always different from each other.

   Write a Python function `hillvalley(l)` that takes a list `l` of integers and returns `True` if it is a hill or a valley, and `False` otherwise.

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

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

>>> hillvalley([1,2,3,4,5])
False

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

```
True

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

Sample Test Cases

	Input	Output
Test Case 1	`threesquares(8)`	`True`
Test Case 2	`threesquares(191)`	`False`
Test Case 3	`threesquares(1001)`	`True`
Test Case 4	`threesquares(199)`	`False`
Test Case 5	`repfree("(x+6)[y-5]")`	`True`
Test Case 6	`repfree("expis1")`	`True`
Test Case 7	`repfree("pingpong")`	`False`
Test Case 8	`repfree("95tumblers")`	`True`
Test Case 9	`hillvalley([5,3,2,1,2,3,5,4,3,2,1])`	`False`
Test Case 10	`hillvalley([1,2])`	`False`
Test Case 11	`hillvalley([])`	`False`
Test Case 12	`hillvalley([5,4,3,2,1,0,-3,-2,-1])`	`True`
Test Case 13	`threesquares(6)`	`True`
Test Case 14	`threesquares(143)`	`False`
Test Case 15	`threesquares(1024)`	`True`
Test Case 16	`repfree("qwerty!@#2")`	`True`
Test Case 17	`repfree("(x+6)(y-5)")`	`False`
Test Case 18	`repfree("94templars")`	`True`
Test Case 19	`repfree("siruseri")`	`Fa`

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

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

Test Case 20	`hillvalley([1,2,3,5,4,3,2,1])`	`True`
Test Case 21	`hillvalley([1,2,3,4,5,3,2,4,5,3,2,1])`	`False`
Test Case 22	`hillvalley([9,5,4,-1,-2,3,7])`	`True`
Test Case 23	`hillvalley([5,4,3,2,1,0,-1,-2,-3])`	`False`

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 factors(n):
    if n == 0:
        return([0])
    factorlist = []
    for i in range(1,n+1):
        if n%i == 0:
            factorlist.append(i)
    return(factorlist)

def square(n):
    return(len(factors(n))%2 == 1)

def threesquares(n):
    for i in range(0,n+1):
        for j in range(i,n+1):
            if square(i) and square(j) and square(n-(i+j)):
                return(True)
    return(False)

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


def repfree(s):
    for i in range(len(s)):
        for j in range(i+1,len(s)):
            if s[i] == s[j]:
                return(False)
    return(True)


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

def ascending(l):
    if len(l) <= 1:
        return(True)
    else:
        return(l[0] < l[1] and ascending(l[1:]))

def descending(l):
    if len(l) <= 1:
        return(True)
    else:
        return(l[0] > l[1] and descending(l[1:]))

def hill(l):
    for i in range(1,len(l)-1):
        if ascending(l[:i+1]) and descending(l[i:]):
            return(True)
    return(False)

def valley(l):
    for i in range(1,len(l)-1):
        if descending(l[:i+1]) and ascending(l[i:]):
            return(True)
    return(False)

def hillvalley(l):
    return(hill(l) or valley(l))
```

```
59
60 ##################################################
61
62 import ast
63
64 def tolist(inp):
65   inp = "["+inp+"]"
66   inp = ast.literal_eval(inp)
67   return (inp[0],inp[1])
68
69 def parse(inp):
70   inp = ast.literal_eval(inp)
71   return (inp)
72
73 fncall = input()
74 lparen = fncall.find("(")
75 rparen = fncall.rfind(")")
76 fname = fncall[:lparen]
77 farg = fncall[lparen+1:rparen]
78
79 if fname == "threesquares":
80    arg = parse(farg)
81    print(threesquares(arg))
82 elif fname == "repfree":
83    arg = parse(farg)
84    print(repfree(arg))
85 elif fname == "hillvalley":
86    arg = parse(farg)
87    print(hillvalley(arg))
88 else:
89    print("Function", fname, "unknown")
90
```