Prepare > Python

Python

34/115 challenges solved

Rank: 55038 | Points: 975

Write a function
Medium, Python (Basic), Max Score: 10, Success Rate: 90.33%
Solved

The Minion Game
Medium, Python (Basic), Max Score: 40, Success Rate: 86.80%
Solved

Merge the Tools!
Medium, Problem Solving (Basic), Max Score: 40, Success Rate: 93.76%
Solved

Time Delta
Medium, Python (Basic), Max Score: 30, Success Rate: 91.36%
Solved

STATUS
- [ ] Solved
- [ ] Unsolved

SKILLS
- [ ] Problem Solving (Basic)
- [ ] Python (Basic)
- [ ] Problem Solving (Advanced)
- [ ] Python (Intermediate)

DIFFICULTY
- [ ] Easy
- [x] Medium
- [ ] Hard

SUBDOMAINS

Find Angle MBC
Medium, Python (Basic), Max Score: 10, Success Rate: 89.15%
Solved

No Idea!
Medium, Python (Basic), Max Score: 50, Success Rate: 88.02%
Solved

Word Order
Medium, Python (Basic), Max Score: 50, Success Rate: 90.24%
Solved

Compress the String!
Medium, Python (Basic), Max Score: 20, Success Rate: 97.15%
Solved

Company Logo
Medium, Problem Solving (Basic), Max Score: 30, Success Rate: 89.84%
Solved

SUBDOMAINS
- [ ] Introduction
- [ ] Basic Data Types
- [ ] Strings
- [ ] Sets
- [ ] Math
- [ ] Itertools
- [ ] Collections
- [ ] Date and Time
- [ ] Errors and Exceptions
- [ ] Classes
- [ ] Built-Ins
- [ ] Python Functionals
- [ ] Regex and Parsing
- [ ] XML
- [ ] Closures and Decorators
- [ ] Numpy

Piling Up!
Medium, Python (Basic), Max Score: 50, Success Rate: 90.64%
Solved

Triangle Quest 2
Medium, Python (Basic), Max Score: 20, Success Rate: 95.38%
Solved

Iterables and Iterators
Medium, Python (Basic), Max Score: 40, Success Rate: 96.60%
Solved

Triangle Quest
Medium, Python (Basic), Max Score: 20, Success Rate: 93.84%
Solved

Classes: Dealing with Complex Numbers
Medium, Python (Basic), Max Score: 20, Success Rate: 90.92%
Solved

SUBDOMAINS
- [ ] Introduction
- [ ] Basic Data Types
- [ ] Strings
- [ ] Sets
- [ ] Math
- [ ] Itertools
- [ ] Collections
- [ ] Date and Time
- [ ] Errors and Exceptions
- [ ] Classes
- [ ] Built-Ins
- [ ] Python Functionals
- [ ] Regex and Parsing
- [ ] XML
- [ ] Closures and Decorators
- [ ] Numpy

Athlete Sort
Medium, Python (Basic), Max Score: 30, Success Rate: 95.53%
Solved

ginortS
Medium, Python (Basic), Max Score: 40, Success Rate: 97.63%
Solved

Validating Email Addresses With a Filter
Medium, Python (Basic), Max Score: 20, Success Rate: 90.83%
Solved

Reduce Function
Medium, Max Score: 30, Success Rate: 98.37%
Solved

Regex Substitution
Medium, Python (Basic), Max Score: 20, Success Rate: 94.11%
Solved

Medium
Hard

SUBDOMAINS
Introduction
Basic Data Types
Strings
Sets
Math
Itertools
Collections
Date and Time
Errors and Exceptions
Classes
Built-Ins
Python Functionals
Regex and Parsing
XML
Closures and Decorators
Numpy

Validating Credit Card Numbers
Medium, Python (Basic), Max Score: 40, Success Rate: 95.47%
Solved

Words Score
Medium, Max Score: 10, Success Rate: 94.94%
Solved

Default Arguments
Medium, Python (Intermediate), Max Score: 30, Success Rate: 78.83%
Solved

Math
Itertools
Collections
Date and Time
Errors and Exceptions
Classes
Built-Ins
Python Functionals
Regex and Parsing
XML
Closures and Decorators
Numpy
Debugging

Write a function

```
def is_leap(year):
    leap = False

    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                leap = True
        else:
            leap = True
```

```
    return leap
year = int(input())

print(is_leap(year))
```

The Minion Game

```
def minion_game(string):

    # your code goes here

    vowel = 'aeiou'.upper()

    strl = len(string)

    kevin = sum(strl-i for i in range(strl) if string[i] in vowel)

    stuart = strl*(strl + 1)/2 - kevin


    if kevin == stuart:

        print ('Draw')

    elif kevin > stuart:

        print ('Kevin %d' % kevin)

    else:

        print ('Stuart %d' % stuart)


if __name__ == '__main__':

    s = input()

    minion_game(s)
```

Merge Tools

```
def merge_the_tools(string, k):

    # your code goes here

    temp = []

    len_temp = 0

    for item in string:

        len_temp += 1
```

```
        if item not in temp:
            temp.append(item)
        if len_temp == k:
            print (''.join(temp))
            temp = []
            len_temp = 0
if __name__ == '__main__':
    string, k = input(), int(input())
    merge_the_tools(string, k)


if __name__ == '__main__':
    string, k = input(), int(input())
    merge_the_tools(string, k)
```

Time Delta

```
#!/bin/python3

import math
import os
import random
import re
import sys
from datetime import datetime
# Complete the time_delta function below.
def time_delta(t1, t2):
    time_format = '%a %d %b %Y %H:%M:%S %z'
    t1 = datetime.strptime(t1, time_format)
    t2 = datetime.strptime(t2, time_format)
```

```
    return str(int(abs((t1-t2).total_seconds())))

if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    t = int(input())

    for t_itr in range(t):
        t1 = input()

        t2 = input()

        delta = time_delta(t1, t2)

        fptr.write(delta + '\n')

    fptr.close()
```

Find Angle MBC

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
import math

ab=int(input())
bc=int(input())

ca=math.hypot(ab,bc)
mc=ca/2
bca=math.asin(1*ab/ca)
bm=math.sqrt((bc**2+mc**2)-(2*bc*mc*math.cos(bca)))
mbc=math.asin(math.sin(bca)*mc/bm)
```

```
print(int(round(math.degrees(mbc),0)),'\u00B0',sep='')
```

No Idea!

```
if __name__ == "__main__":
    happiness = 0

    n, m = map(int, input().strip().split())
    arr = list(map(int, input().strip().split()))
    good = set(map(int, input().strip().split()))
    bad = set(map(int, input().strip().split()))
    happiness = sum(1 if x in good else -1 if x in bad else 0 for x in arr)

    print(happiness)
```

Word Order

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
n=int(input())
words=[input() for i in range(n)]
words_occurences={}
for word in words:
    words_occurences[word]=0
for word in words:
    words_occurences[word]+=1
print(len(words_occurences))
occurences=words_occurences.values()
for i in occurences:
    print(i,end=" ")
```

Compress the String

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
from itertools import groupby


if __name__ == "__main__":
   for k, c in groupby(input())

      print("(%d, %d)" % (len(list(c)), int(k)), end=' ')
```

Company Logo

```
#!/bin/python3


import math

import os

import random

import re

import sys

from collections import Counter


if __name__ == '__main__':

  s = sorted(input().strip()

  s_counter =Counter(s).most_common()

   s_counter = sorted(s_counter, key=lambda x: (x[1] * -1, x[0]))
   for i in range(0, 3):

      print(s_counter[i][0], s_counter[i][1])
```

Piling Up!

```
# Enter your code here. Read input from STDIN. Print output to STDOUT

from collections import *

def piling(d):

   while d:

      large = None
```

```
    if d[-1]>d[0]:
        large = d.pop()
    else:
        large = d.popleft()
    if len(d)==0:
        return "Yes"
    if d[-1]>large or d[0]>large:
        return "No"

for i in range(int(input())):
    n = int(input())
    d = deque(map(int,input().split()))
    print(piling(d))
```

Triangle Quest 2

```
for i in range(1,int(input())+1): #More than 2 lines will result in 0 score. Do not leave a blank line also
    print(((10**i)//9)**2)
```

Iterables and Iterators

```
from itertools import combinations, groupby

count, letters, to_select = int(input()), input().split(), int(input())

letters = sorted(letters)
combinations_of_letters = list(combinations(letters, to_select))
contain = len([c for c in combinations_of_letters if 'a' in c])
print(contain / len(combinations_of_letters))
```

Triangle Quest

```
for i in range(1,int(input())): #More than 2 lines will result in 0 score. Do not leave a blank line also
    print((10**(i)//9)*i)
```

Classes: Dealing with Complex Numbers

```
import math

class Complex(object):

  def __init__(self, real, imaginary):

    self.real = real

    self.imaginary = imaginary

  def __add__(self, no):

    return Complex((self.real+no.real), self.imaginary+no.imaginary)

  def __sub__(self, no):

    return Complex((self.real-no.real), (self.imaginary-no.imaginary))

  def __mul__(self, no):

    r = (self.real*no.real)-(self.imaginary*no.imaginary)

    i = (self.real*no.imaginary+no.real*self.imaginary)

    return Complex(r, i)

  def __truediv__(self, no):

    conjugate = Complex(no.real, (-no.imaginary))

    num = self*conjugate

    denom = no*conjugate

    try:

      return Complex((num.real/denom.real), (num.imaginary/denom.real))

    except Exception as e:

      print(e)

  def mod(self):

    m = math.sqrt(self.real**2+self.imaginary**2)

    return Complex(m, 0)

  def __str__(self):

    if self.imaginary == 0:
```

```
        result = "%.2f+0.00i" % (self.real)
    elif self.real == 0:
        if self.imaginary >= 0:
            result = "0.00+%.2fi" % (self.imaginary)
        else:
            result = "0.00-%.2fi" % (abs(self.imaginary))
    elif self.imaginary > 0:
        result = "%.2f+%.2fi" % (self.real, self.imaginary)
    else:
        result = "%.2f-%.2fi" % (self.real, abs(self.imaginary))
    return result

if __name__ == '__main__':
    c = map(float, input().split())
    d = map(float, input().split())
    x = Complex(*c)
    y = Complex(*d)
    print(*map(str, [x+y, x-y, x*y, x/y, x.mod(), y.mod()]), sep='\n')
```

Athelete Sort

```
#!/bin/python3
import math

import os

import random

import re

import sys


if __name__ == '__main__':

    nm = input().split()
```

```
    n = int(nm[0])

    m = int(nm[1])

    arr = []

    for _ in range(n):

        arr.append(list(map(int, input().rstrip().split())))

    k = int(input())

    arr.sort(key = lambda x : x[k])

    for i in arr:

        print(*i,sep=' ')
```

ginortS

```
# Enter your code here. Read input from STDIN. Print output to STDOUT

def key_function(character):

    return (character.isdigit() - character.islower(), character in "02468", character)

input_string = "Sorting1234"
print(*sorted(input_string, key=key_function), sep="")
```

Validating Email Addresses With a Filter

```
import re

def fun(s):

    a = re.match(r'[a-zA-Z0-9_-]+@[a-zA-Z0-9]+\.[a-zA-Z]{1,3}$', s)

    return(a)

    # return True if s is a valid email, else return False


def filter_mail(emails):

    return list(filter(fun, emails))
if __name__ == '__main__':

    n = int(input())

    emails = []

    for _ in range(n):
```

```
    emails.append(input())

filtered_emails = filter_mail(emails)
filtered_emails.sort()
print(filtered_emails)
```

Reduce Function

```
from fractions import Fraction
from functools import reduce
def product(fracs):
    t = Fraction(reduce(lambda x, y: x * y, fracs))
    return t.numerator, t.denominator

if __name__ == '__main__':
    fracs = []
    for _ in range(int(input())):
        fracs.append(Fraction(*map(int, input().split())))
    result = product(fracs)
    print(*result)
```

Regex substitution

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
import re

for i in range(int(input())):
    s = re.sub("(?<=\s)&&(?=\s)", "and", input())
    print(re.sub("(?<=\s)\|\|(?=\s)", "or", s))
```

Validating Credit Card Numbers

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
import re
```

```
n = int(input())

for i in range (n):

   number = input().split('-')

   if (len(number[0]) == 4 and len(number[1]) == 4 and len(number[2]) == 4 and len(number[3]) == 4) o
r len(number[0]) == 16:

      string = ''.join(number) + 'ok'

   else:

      string = ''.join(number) + 'nok'

   pattern = r'^[4-6]\d{15,15}+ok$'

   pattern_repeats = r'([\d])\1\1\1'

   if re.match(pattern, string) and not re.search(pattern_repeats, string):

     print('Valid')

   else:

      print('Invalid')
```

Words Score

```
def is_vowel(letter):

   return letter in ['a', 'e', 'i', 'o', 'u', 'y']


def score_words(words):

   score = 0

   for word in words:

      num_vowels = 0

      for letter in word:

         if is_vowel(letter):

            num_vowels += 1

      if num_vowels % 2 == 0:

         score += 2

      else:
```

```
        score +=1
    return score


n = int(input())
words = input().split()
print(score_words(words))
```

Default Arguments

```
class EvenStream(object):
    def __init__(self):
        self.current = 0

    def get_next(self):
        to_return = self.current
        self.current += 2
        return to_return

class OddStream(object):
    def __init__(self):
        self.current = 1

    def get_next(self):
        to_return = self.current
        self.current += 2
        return to_return

def print_from_stream(n, stream=EvenStream()):
    if stream is None:
```

```
        stream = EvenStream()
    for _ in range(n):
        print(stream.get_next())


queries = int(input())
for _ in range(queries):
    stream_name, n = input().split()
    n = int(n)
    if stream_name == "even":
        print_from_stream(n)
    else:
        print_from_stream(n, OddStream())
```