OJT - Python Exercise - Paper3

1. Write a program in which two strings are given and determine if they share a common substring. A substring may be as small as one character. The function returns either “YES” or “NO”.
2. Write a decorator function that will record the number of times a function is called. Your decorator function should be called record\_calls and call\_count attribute that keeps track of the number of times it was called.
3. Write a function called interleave which accepts two iterables of any type and returns a new iterable with each of the given items "interleaved" (item 0 from iterable 1, then item 0 from iterable 2, then item 1 from iterable 1, and so on). An assumption here that both iterables contain the same number of elements.
4. Write to\_celsius function that accepts a temperature in Fahrenheit as input and returns a temperature in Celsius.

# 4.Write to\_celsius function that accepts a temperature in Fahrenheit as

# input and returns a temperature in Celsius.

def to\_celsius(f):

    return (f-32)\*(5/9)

print(to\_celsius(565))

1. Write a function that accepts an [iterable](https://www.pythonmorsels.com/iterable/) and returns a new iterable with all items from the original iterable except for duplicates.

Ex. uniques\_only([1, 2, 2, 1, 1, 3, 2, 1])

[1, 2, 3]

# 5.Write a function that accepts an iterable and returns a new iterable with all items from the original iterable except for duplicates.

# Ex. uniques\_only([1, 2, 2, 1, 1, 3, 2, 1])

# [1, 2, 3]

l=[1, 2, 2, 1, 1, 3, 2, 1]

t=(1, 2, 2, 1, 1, 3, 2, 1)

s={1, 2, 2, 1, 1, 3, 2, 1}

def unique\_only(i):

    originaltype=type(i)

    i=set(i) # to remove duplicate

    i=originaltype(i)     # to convert it back to its original type

    return(i)

print(unique\_only(l))

print(unique\_only(t))

print(unique\_only(s))

1. Write a function to\_percent which accepts a number representing a ratio and returns a string representing the percentage representation of the number to one decimal place.
2. Write a function that accepts two strings and returns True if the two strings are [anagrams](https://en.wikipedia.org/wiki/Anagram) of each other.
3. Write Row class that accepts any keyword arguments given to it and stores these arguments as attributes.

Ex. >>> row = Row(a=1, b=2)

>>> row.a 1

>>> row.b 2

1. Create a function is\_leap\_year that accepts a year and returns True if (and only if) the given year is a leap year.
2. Write a function combine\_lists should take two lists and return a new list containing all elements from both lists.

# 10.Write a function combine\_lists should take two lists and return a new list

#  containing all elements from both lists.

def combinelist(l1,l2):

    l=l1+l2

    return l

l1=[1,2,2,4,3,5,6]

l2=[1,2,2,4,3,5,6]

print(combinelist(l1,l2))

1. Write a function, last\_lines, which returns lines in a given ASCII text file in reverse order.

For example, given the following file, my\_file.txt:

This is a file This is line 2 And this is line 3

The last\_lines function should work like this:

**>>>** for line in last\_lines('my\_file.txt'):

**...** print(line, end='')

# ...

And this is line 3 This is line 2 This is a file

def last\_lines(f):

    with open(f'{f}','r') as f:

        x=f.read().splitlines()

        x=x[::-1]

        return x

for line in last\_lines('3ojt11.txt'):

    print(line)

1. Write a function called parse\_ranges, which accepts a string containing ranges of numbers and returns an iterable of those numbers.

Ex: **>>>** parse\_ranges('1-2,4-4,8-13') [1, 2, 4, 8, 9, 10, 11, 12, 13]

# 12.Write a function called parse\_ranges, which accepts a string containing ranges of numbers

# and returns an iterable of those numbers.

# Ex: >>> parse\_ranges('1-2,4-4,8-13') [1, 2, 4, 8, 9, 10, 11, 12, 13]

def parse\_ranges(s):

    op=[]

    ranges=s.split(',')

    print(ranges)

    for r in ranges:

        digits=r.split('-')

        for dig in range(int(digits[0]),int(digits[1])+1):

            op.append(dig)

    print(op)

parse\_ranges('1-2,4-4,8-13')

1. Write a function that accepts a string containing lines of numbers and returns a list of lists of numbers.

Ex. matrix\_from\_string("3 4 5")

[[3.0, 4.0, 5.0]]

def matrix\_from\_string(s):

    op=[]

    for char in s:

        if char.isnumeric():

               op.append(float(char))

    return [op]

print(matrix\_from\_string("3 4 5"))

1. Write a command-line program which helps a traveler keep track of the restaurants they've visited in different cities and what they thought of each. The program will accept two CSV files of restaurants, a "primary list" CSV and a "sublist" one, and update the primary one with new restaurants from the trip one.
2. Write a function get\_hypotenuse that returns the hypotenuse of a right triangle given the other two sides.

**>>>** get\_hypotenuse(0, 0)

0.0

**>>>** get\_hypotenuse(3, 4)

5.0

# 15.Write a function get\_hypotenuse that returns the hypotenuse of a right triangle given

# the other two sides.

# >>> get\_hypotenuse(0, 0)

# 0.0

import math

def get\_hypotenuse(x,y):

    return math.sqrt((x\*\*2)+(y\*\*2))

    # OR>> return ((x\*\*2)+(y\*\*2))\*\*0.5

print(get\_hypotenuse(3,4))

1. Write a function split\_in\_half that splits a list in half and returns both halves.

**>>>** split\_in\_half([1, 2, 3, 4])

([1, 2], [3, 4])

def split\_in\_half(l):

    mid=len(l)//2

    return (l[:mid],l[mid:])

print(split\_in\_half([1, 2, 3, 4,]))

1. Write a function that takes a sequence (like a list, string, or tuple) and a number n and returns the last n elements from the given sequence, as a list. For example:

**>>>** tail([1, 2, 3, 4, 5], 3)

[3, 4, 5]

1. Create your own exception.
2. Write a function that takes two strings representing dates and returns the string that represents the earliest point in time ? Ex. get\_earliest("01/27/1832", "01/27/1756") return '01/27/1756'.

def get\_earliest(s1,s2):

    l1=s1.split('/')[::-1]

    l2=s2.split('/')[::-1]

    for i in range(len(l1)):

        if int(l1[i])>int(l2[i]):

            return s2

        elif int(l1[i])<int(l2[i]):

            return s1

        else:

            continue

    # return "Both are same date"

print(get\_earliest("01/27/1832", "01/27/1756"))

print(get\_earliest("01/17/1756", "01/28/1756"))

print(get\_earliest("05/28/1756", "01/28/1756"))

print(get\_earliest("01/28/1756", "01/28/1756"))

1. Create a function that determines which day of the month the San Diego Python meetup should be. It should accept year and month arguments and should return a datetime.date object representing the day of the month for the meetup.

**>>>** meetup\_date(2012, 3)

datetime.date(2012, 3, 22)

1. Write a callable called float\_range that acts sort of like the built-in range callable but it should allow for floating point numbers to be specified as start, stop, and step values.

**>>>** r = float\_range(0.5, 2.5, 0.5)

# >>> r

float\_range(0.5, 2.5, 0.5)

**>>>** list(r)

[0.5, 1.0, 1.5, 2.0]

**>>>** len(r) 4

**>>>** for n in r:

**...** print(n)

# ...

0.5

1.0

1.5

2.0

1. Write a function is\_iterator so that it accepts an [iterable](https://www.pythonmorsels.com/iterable/) and returns True if the given iterable is an [iterator](https://www.pythonmorsels.com/what-is-an-iterator/).

is\_iterator(iter([])) True

**>>>** is\_iterator([1, 2]) False

1. Create a context manager. Context managers use a with block to bookend a block of code with automatic setup and tear down steps.Your context manager, suppress, should suppress exceptions of a given type:

**>>>** with suppress(NameError):

**...** print("Hi!")

**...** print("It's nice to meet you,", name)

**...** print("Goodbye!")

Hi!

But exceptions of *other* types shouldn't be suppressed (we're suppressing a TypeError and a NameError is raised):

**>>>** with suppress(TypeError):

**...** print("Hi!")

**...** print("It's nice to meet you,", name)

**...** print("Goodbye!")

# ...

Hi!

Traceback (most recent call last): File "<stdin>", line 3, in <module>

NameError: name 'name' is not defined

1. Write a class that represents a circle.The circle should have a radius, a diameter, and an area. It should also have a nice string representation.
2. Write a program to convert integers to Roman numbers.
3. Write a function so that it accepts an iterable and returns True if the given iterable is an iterator.
4. Write a class that represents a bank account, do bank operations.
5. Standardize mobile numbers when given N mobile numbers. Sort them in ascending order. Print them in the standard format.
6. Write a function called interleave which accepts two iterables of any type and returns a new iterable with each of the given items "interleaved" (item 0 from iterable 1, then item 0 from iterable 2, then item 1 from iterable 1, and so on).
7. Convert each list element to a key-value pair. ex:

Input : test\_list = [2323, 82, 129388, 95]

Output : {23: 23, 8: 2, 129: 388, 9: 5}