Lab 1

● Graded

Student

Sunny Sun

Total Points

133 / 133 pts

Autograder Score

133.0 / 133.0

Passed Tests

Public Tests

Autograder Results

Autograder Output

```
 _________      __        __
/  _____  \   __|  |__   __|  |__
|  /    \  |  |__    __|  |__    __|   ______    _  _____
| |      | |     |  |        |  |    |  ___  |  | |/ _____|
| |      | |     |  |        |  |    | |___| |  | |  /
| |      | |     |  |        |  |    | _______|  | |
|  \_____/ |     |  |_      |  |_   | |_____   | |
 \_________/      \__|      \__|   \_______|  |__|
                                    v3.1.4






------------------------------- GRADING SUMMARY ---------------------------------

No log found with which to verify student scores.


Total Score: 133.000 / 133.000 (100.000%)

            name  score  max_score
0   Public Tests    NaN        NaN
1             q0    0.0        0.0
2             q1   12.0       12.0
3             q2   15.0       15.0
4             q3   13.0       13.0
5             q4   13.0       13.0
6             q5   10.0       10.0
7             q6   11.0       11.0
8           q6.2   12.0       12.0
9             q7   14.0       14.0
10            q8   14.0       14.0
11            q9   19.0       19.0
```

Public Tests

```
q0 results: All test cases passed!

q1 results: All test cases passed!

q2 results: All test cases passed!

q3 results: All test cases passed!

q4 results: All test cases passed!

q5 results: All test cases passed!

q6 results: All test cases passed!

q6.2 results: All test cases passed!

q7 results: All test cases passed!

q8 results: All test cases passed!

q9 results: All test cases passed!
```

Submitted Files

▾ **lab.py** Download

```
# lab.py


from pathlib import Path
import io
import pandas as pd
import numpy as np
np.set_printoptions(legacy='1.21')


# ---------------------------------------------------------------------
# QUESTION 0
# ---------------------------------------------------------------------


def consecutive_ints(ints):
    if len(ints) == 0:
        return False

    for k in range(len(ints) - 1):
        diff = abs(ints[k] - ints[k+1])
        if diff == 1:
            return True

    return False


# ---------------------------------------------------------------------
# QUESTION 1
# ---------------------------------------------------------------------


def median_vs_mean(nums):
    nums = sorted(nums) # sort list

    count = len(nums)
    mean = sum(nums)/count # Calculate the mean

    mid = count // 2 # find middle index
    if count % 2 == 0: # If even, take the average of the middle two numbers
        median = (nums[mid] + nums[mid-1]) / 2
    else: # If odd, the median is the middle element
        median = nums[mid]
    return median <= mean # Return True if median <= mean, otherwise False



# ---------------------------------------------------------------------
# QUESTION 2
```

```python
# ---------------------------------------------------------------------


def n_prefixes(s, n):
    result = ''
    for i in range(1, n + 1):
        result = s[:i] + result
    return result


# ---------------------------------------------------------------------
# QUESTION 3
# ---------------------------------------------------------------------


def exploded_numbers(ints, n):
    lst = []
    if len(ints) == 0:
        return lst

    int_len = len(str(max(ints)+n))
    for num in ints:
        exploded = [f"{i:0{int_len}d}" for i in range(num-n, num+n+1)]
        lst.append(" ".join(exploded))

    return lst


# ---------------------------------------------------------------------
# QUESTION 4
# ---------------------------------------------------------------------


def last_chars(fh):
    result = ''
    for line in fh:
        line = line.rstrip('\n')
        if line:
            result += line[-1]
    return result


# ---------------------------------------------------------------------
# QUESTION 5
# ---------------------------------------------------------------------


def add_root(A):
    new_arr = A + np.sqrt(np.arange(len(A)))
    return new_arr

def where_square(A):
    sqrt_arr = np.sqrt(A)
    return sqrt_arr == np.floor(np.sqrt(A))


# ----------------------------------------------------------------------
# QUESTION 6
# ----------------------------------------------------------------------


def filter_cutoff_loop(matrix, cutoff):
    num_rows = len(matrix)
    num_cols = len(matrix[0])
    filtered = []

    for col in range(num_cols):
        col_vals = [matrix[row][col] for row in range(num_rows)]
        col_mean = sum(col_vals) / num_rows
        if col_mean > cutoff:
            filtered.append(col_vals)

    result = [[col[i] for col in filtered] for i in range(num_rows)]
    return np.array(result)


# ----------------------------------------------------------------------
# QUESTION 6
# ----------------------------------------------------------------------


def filter_cutoff_np(matrix, cutoff):
    col_means = np.mean(matrix, axis=0)
    keep = col_means > cutoff
    return matrix[:, keep]


# ----------------------------------------------------------------------
# QUESTION 7
# ----------------------------------------------------------------------


def growth_rates(A):
    return np.round((A[1:] - A[:-1]) / A[:-1], 2)

def with_leftover(A):
    day = -1
    can_buy = np.cumsum(20 % A) >= A
    indices = np.where(can_buy)[0]
    if len(indices)>0:
        day = int(indices[0])
    return day


```

```python
# ------------------------------------------------------------------------
# QUESTION 8
# ------------------------------------------------------------------------

def salary_stats(salary):
    # number of players
    num_players = len(salary)

    # number of teams
    num_teams = salary['Team'].nunique()

    # total salary amount for all players
    total_salary = salary['Salary'].sum()

    # name of the player with the highest salary
    highest_salary = salary.loc[salary['Salary'].idxmax(), 'Player']

    # average salary of the 'Los Angeles Lakers', rounded to two decimal places
    avg_los = round(salary[salary['Team'] == 'Los Angeles Lakers']['Salary'].mean(), 2)

    # name and team of the player who has the fifth lowest salary, e.g 'Andy J, Lakers'
    fifth_lowest_row = salary.sort_values(by='Salary').iloc[4]
    fifth_lowest = f"{fifth_lowest_row['Player']}, {fifth_lowest_row['Team']}"

    # boolean if any duplicate last name, ignore suffix
    def clean_name(name):
        parts = name.split()
        if parts[-1] in ['Jr.', 'Sr.', 'II', 'III', 'IV']:
            return parts[-2]
        else:
            return parts[-1]
    last_names = salary['Player'].apply(clean_name)
    duplicates = last_names.duplicated().any()

    # total salary of the team that has highest_salary
    highest_paid_team = salary.loc[salary['Salary'].idxmax(), 'Team']
    total_highest = salary[salary['Team'] == highest_paid_team]['Salary'].sum()

    #return as a Series
    return pd.Series({
        'num_players': num_players,
        'num_teams': num_teams,
        'total_salary': total_salary,
        'highest_salary': highest_salary,
        'avg_los': avg_los,
        'fifth_lowest': fifth_lowest,
        'duplicates': duplicates,
        'total_highest': total_highest
    })


# ------------------------------------------------------------------------
# QUESTION 9
# ------------------------------------------------------------------


def parse_malformed(fp):
    rows = []
    with open(fp) as f:
        next(f)  # skip header

        for line in f:
            line = line.strip().replace('"', '')  # remove all quotes
            if not line:
                continue
            # split line into parts(a list) using comma
            parts = [p.strip() for p in line.split(',') if p.strip()]

            # expect 6 parts: first, last, weight, height, geo(lat, lon)
            if len(parts) < 6:
                continue  # not enough data

            try:
                first = parts[0]
                last = parts[1]
                weight = float(parts[2])
                height = float(parts[3])
                geo = parts[4] + ',' + parts[5]  # combine lat + lon

                rows.append({
                    'first': first,
                    'last': last,
                    'weight': weight,
                    'height': height,
                    'geo': geo
                })
            except:
                continue  # skip any row that can't be parsed

    return pd.DataFrame(rows)
```