Python Exercises - Part III [solutions]

Python and R for Data Science

Data Science and Management

LUISS

Exercise 1: Shortest Words

Write a function `shortest_words` that:

- takes a list of words
- returns a list containing the shortest words in the list received as argument. The list will containt more than one word when there are multiple words with the same length.

Examples:

- `shortest_words([])` returns `[]`
- `shortest_words(['sheldon', 'cooper'])` returns `['cooper']`
- `shortest_words(['sheldon', 'cooper', 'howard'])` returns `['cooper', 'howard']`

NOTE: do not use any buit-in function from Python to solve the exercise

In [711]:

```python
# Solution goes here
```

Solution

```
In [712]: def shortest_words(lista_stringhe):
              lunghezza_minima = float('inf')
              for parola in lista_stringhe:
                  lunghezza_parola = len(parola)
                  if lunghezza_parola < lunghezza_minima:
                      lunghezza_minima = lunghezza_parola

              parole_corte = []
              for parola in lista_stringhe:
                  if len(parola) == lunghezza_minima:
                      parole_corte.append(parola)

              return parole_corte
```

Test your code

Run this code to test your solution:

```
In [713]:
try: assert shortest_words([]) == [] and not print("Test #1 passed")
except: print('Test #1 failed')

try: assert sorted(shortest_words(['sheldon', 'cooper'])) == ['cooper'] and not p
except: print('Test #2 failed')

try: assert sorted(shortest_words(['sheldon', 'cooper', 'howard'])) == ['cooper',
except: print('Test #3 failed')
```

```
Test #1 passed
Test #2 passed
Test #3 passed
```

Exercise 2: Multiply Tuples

Write a function called `mul_tuple` that:

- Takes two tuples containing integers as arguments
- Returns a new tuple containing the products of the corresponding elements (at the same position) of the two tuples. If the two tuples have different lengths, the function should return `None`.

In [714]:
```
# Solution goes here
```

Solution

```
In [715]: def mul_tuple(tuple1, tuple2):
              if len(tuple1) != len(tuple2):
                  return None
              res = tuple()
              for i in range(len(tuple1)):
                  res = res + (tuple1[i] * tuple2[i], )
              return res
```

Test your code

Run this code to test your solution:

In [716]:
```
try: assert mul_tuple((1, 2, 3), (4, 5, 6, 7)) == None and not print("Test #1 pas
except: print('Test #1 failed')

try: assert mul_tuple((1, 2), (4, 5)) == (4, 10) and not print("Test #2 passed")
except: print('Test #2 failed')
```

```
Test #1 passed
Test #2 passed
```

Exercise 3: Max Point Distance

Write a function `max_dist_point` that:

- Takes as arguments:
    - A point in the Cartesian plane represented as a tuple with its coordinates (x, y), where x and y are integers.
    - A list of points in the Cartesian plane.
- Returns:
    - If the list received as an argument is empty: `None`.
    - Otherwise: a tuple of two values, consisting of:
        1. The maximum distance (integer) between the given point and all the points in the list. To calculate the distance between a pair of points ((x1, y1)) and ((x2, y2)), use the Euclidean distance formula:

$$distance = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

        2. The point from the list that produced the maximum distance. Round the distance down using `int(distance)`.

NOTE: The square root can be calculated using `math.sqrt()` from the math library.

In [717]:

```
# Solution goes here
```

Solution

In [718]:
```
import math

def max_dist_point(punto, lista_punti):
    if not lista_punti: return None
    distanza_massima = -1
    punto_massimo = None

    for punto_lista in lista_punti:
        distanza = int(math.sqrt((punto_lista[0] - punto[0])**2 + (punto_lista[1]
        if distanza > distanza_massima:
            distanza_massima = distanza
            punto_massimo = punto_lista

    return (distanza_massima, punto_massimo)
```

Test your code

Run this code to test your solution:

In [719]:
```
try: assert max_dist_point((0, 0), []) == None and not print("Test #1 passed")
except: print('Test #1 failed')

try: assert max_dist_point((0, 0), [(1, 1), (2, 2), (3, 3)]) == (4, (3, 3)) and n
except: print('Test #2 failed')

try: assert max_dist_point((10, 12), [(1, 3), (4, 23), (-100, 0), (1, 1)]) and no
except: print('Test #3 failed')
```

```
Test #1 passed
Test #2 passed
Test #3 passed
```

Exercise 4: Character Position Tracker

Write a function `track_char_positions` that:

- Takes a string as input.
- Returns a dictionary where:
    - The keys are the unique characters in the string.
    - The values are lists of positions (indices) where each character appears in the string.

NOTE: The function should track both uppercase and lowercase characters as distinct. NOTE: Spaces and punctuation should also be tracked as characters.

Example

```
text = "hello"
result = track_char_positions(text)
```

The `result` should be:

```
{
  'h': [0],
  'e': [1],
  'l': [2, 3],
  'o': [4]
}
```

In [720]:
```
# Solution goes here
```

Solution

In [721]:
```python
def track_char_positions(text):
    # Create an empty dictionary to store the character positions
    char_positions = {}

    # Loop through the string and keep track of the index of each character
    for index, char in enumerate(text):
        # If the character is not already a key in the dictionary, add it
        if char not in char_positions:
            char_positions[char] = []
        # Append the index of the character to the list
        char_positions[char].append(index)

    return char_positions
```

Test your code

Run this code to test your solution:

```
In [722]:
text = "hello"
expected_result = {'h': [0], 'e': [1], 'l': [2, 3], 'o': [4]}
try: assert track_char_positions(text) == expected_result and not print("Test #1
except: print('Test #1 failed')

text = "banana"
expected_result = {'b': [0], 'a': [1, 3, 5], 'n': [2, 4]}
try: assert track_char_positions(text) == expected_result and not print("Test #2
except: print('Test #2 failed')

text = "Hi, there !"
expected_result = {
    'H': [0], 'i': [1], ',': [2], ' ': [3, 9], 't': [4], 'h': [5], 'e': [6, 8], '
}
try: assert track_char_positions(text) == expected_result and not print("Test #3
except: print('Test #3 failed')
```

```
Test #1 passed
Test #2 passed
Test #3 passed
```

Exercise 5: Anagram Grouping

Write a function `group_anagrams` that:

- Takes a list of strings as input.
- Returns a dictionary where:
    - The keys are the strings received as input.
    - The values are alphabetically sorted lists of words from the input list that are anagrams of each other.

NOTE: The words should be grouped based on their sorted letter order.
NOTE: If no anagram pairs are found, each word should still appear in its own list.

Example

An anagram is a word formed by rearranging the letters of another word, using all the original letters exactly once. For instance:

```python
words = ["listen", "silent", "enlist", "hello", "world", "drown", "word"]
result = group_anagrams(words)
```

The `result` should be:

```
{
  'eilnst': ['enlist', 'listen', 'silent'],
  'ehllo': ['hello'],
  'dlorw': ['world'],
  'dnorw': ['drown', 'word']
}
```

In [723]:

```
# Solution goes here
```

Solution

```
In [724]:
def group_anagrams(words):
    anagram_dict = {}

    for word in words:
        # Sort the word to create a key
        sorted_word = ''.join(sorted(word))

        # Add the word to the list corresponding to the sorted key
        if sorted_word not in anagram_dict:
            anagram_dict[sorted_word] = []
        anagram_dict[sorted_word].append(word)

    # Sort the list of anagrams for each key
    for each in anagram_dict:
        anagram_dict[each] = sorted(anagram_dict[each])

    return anagram_dict
```

Test your code

Run this code to test your solution:

In [725]:
```
words = ["listen", "silent", "enlist", "hello"]
expected_result = {
    'eilnst': ['enlist', 'listen', 'silent'],
    'ehllo': ['hello']
}
try: assert group_anagrams(words) == expected_result and not print("Test #1 passe
except: print('Test #1 failed')

words = ["apple", "banana", "orange"]
expected_result = {
    'aelpp': ['apple'], 'aaabnn': ['banana'], 'aegnor': ['orange']
}
try: assert group_anagrams(words) == expected_result and not print("Test #2 passe
except: print('Test #2 failed')

words = ["Listen", "Silent", "enlist"]
expected_result ={'Leinst': ['Listen'], 'Seilnt': ['Silent'], 'eilnst': ['enlist'
try: assert group_anagrams(words) == expected_result and not print("Test #3 passe
except: print('Test #3 failed')
```

```
Test #1 passed
Test #2 passed
Test #3 passed
```

Exercise 6: ISBN Validator

Write a function `validate_isbn` that:

- Takes a string `isbn` as input, representing a 10-digit ISBN number.
- Returns a dictionary containing:
    - `valid` (key): as value, a boolean indicating whether the ISBN is valid.
    - `digits` (key): as value, a list of the individual digits in the ISBN.

An ISBN is considered valid if it meets the following criteria:

1. It consists of exactly 10 characters (excluding hyphens or spaces, which are ignored) where the first 9 are digits (0-9), and the last character can be a digit or an 'X' (which represents the number 10).
2. The ISBN is valid if the weighted sum of the digits (where the weight decreases from 10 to 1) is divisible by 11. For example, for ISBN `0-306-40615-2`, the calculation would be:

$$\begin{aligned}(0\times 10)+(3\times 9)+(0\times 8)+(6\times 7)+(4\times 6)+(0\times 5)+(6\times 4)+(1\times 3)\\ +(5\times 2)+(2\times 1)=0+27+0+42+24+0+24+3+10+2=132\end{aligned}$$

Since $(132 \mod 11 = 0)$, it is valid.

Example

```
isbn = "0306406152"
result = validate_isbn(isbn)
```

The `result` should be:

```
{
    'valid': True,
    'digits': ['0', '3', '0', '6', '4', '0', '6', '1', '5', '2']
}
```

NOTE: If the input is not a valid ISBN (e.g., it contains non-digit characters or is of the wrong length), return `{'valid': False, 'digits': []}`.
NOTE: Ensure to treat 'X' as a digit representing 10.

In [726]:

```
# Solution goes here
```

Solution

```
In [727]: def validate_isbn(isbn):
              # Remove any hyphens or spaces from the input
              isbn = isbn.replace("-", "").replace(" ", "")

              # Check if the length is 10
              if len(isbn) != 10: return {'valid': False, 'digits': []}

              # Initialize variables
              total = 0
              digits = []

              for i, char in enumerate(isbn):
                  if char.isdigit():
                      digit = int(char)
                      total += digit * (10 - i)  # Weighted sum
                      digits.append(char)
                  elif char == 'X' and i == 9:
                      total += 10  # Last character can be 'X'
                      digits.append('X')
                  else:
                      return {'valid': False, 'digits': []}  # Invalid character

              # Check divisibility by 11
              valid = (total % 11 == 0)

              return {'valid': valid, 'digits': digits}
```

Test your code

Run this code to test your solution:

In [728]:
```python
isbn = "0-306-40615-2"
expected_result = {'valid': True, 'digits': ['0', '3', '0', '6', '4', '0', '6', '
try: assert validate_isbn(isbn) == expected_result and not print("Test #1 passed"
except: print('Test #1 failed')

isbn = "123456789X"
expected_result = {'valid': True, 'digits': ['1', '2', '3', '4', '5', '6', '7', '
try: assert validate_isbn(isbn) == expected_result and not print("Test #2 passed"
except: print('Test #2 failed')

isbn = "12345678"
expected_result = {'valid': False, 'digits': []}
try: assert validate_isbn(isbn) == expected_result and not print("Test #3 passed"
except: print('Test #3 failed')
```

```
Test #1 passed
Test #2 passed
Test #3 passed
```

Exercise 7: Acronym Generator

Write a function `generate_acronym` that:

- Takes a string as input, representing a multi-word phrase (e.g., "As Soon As Possible").
- Returns a dictionary where:
    - The key is the acronym formed from the first letter of each word in the phrase (case insensitive).
    - The value is the original phrase with each word capitalized.

NOTE: Ignore any non-alphabetic characters when forming the acronym.
NOTE: The acronym should be in uppercase.
NOTE: If the input string is empty, return `{'acronym': '', 'phrase': ''}`.

Example

```
phrase = "as soon as possible"
result = generate_acronym(phrase)
```

The `result` should be:

```
{
    'ASAP': 'As Soon As Possible'
}
```

In [729]:
```
# Solution goes here
```

Solution

```
In [730]: def generate_acronym(phrase):
              # Remove any leading/trailing spaces and split the phrase into words
              words = phrase.strip().split()

              # Check for empty input
              if not words:
                  return {'acronym': '', 'phrase': ''}

              # Create acronym and capitalized phrase
              acronym = ''.join(word[0].upper() for word in words if word[0].isalpha())
              capitalized_phrase = ' '.join(word.capitalize() for word in words)

              return {'acronym': acronym, 'phrase': capitalized_phrase}
```

Test your code

Run this code to test your solution:

```
In [731]: phrase = "as soon as possible"
          expected_result = {'acronym': 'ASAP', 'phrase': 'As Soon As Possible'}
          try: assert generate_acronym(phrase) == expected_result and not print("Test #1 pa
          except: print('Test #1 failed')

          phrase = "   keep it simple stupid  "
          expected_result = {'acronym': 'KISS', 'phrase': 'Keep It Simple Stupid'}
          try: assert generate_acronym(phrase) == expected_result and not print("Test #2 pa
          except: print('Test #2 failed')

          phrase = "for your information."
          expected_result = {'acronym': 'FYI', 'phrase': 'For Your Information.'}
          try: assert generate_acronym(phrase) == expected_result and not print("Test #3 pa
          except: print('Test #3 failed')
```

```
Test #1 passed
Test #2 passed
Test #3 passed
```

Exercise 8: Movie Rating Organizer

Write a function `organize_movie_ratings` that:

- Takes a list of tuples as input, where each tuple contains two elements:
    - A string representing the name of a movie.
    - An integer representing the rating of that movie (from 1 to 10).
- Returns a dictionary where:
    - The keys are the unique movie titles.
    - The values are lists of ratings for each movie.

NOTE: If a movie appears multiple times in the input list, all ratings should be included in the list for that movie.
NOTE: The order of the ratings in the lists should reflect the order they appear in the input list.

Example

```python
ratings = [
    ("Inception", 9),
    ("The Matrix", 8),
    ("Inception", 10),
    ("The Godfather", 9),
    ("The Matrix", 9)
]
result = organize_movie_ratings(ratings)
```

The `result` should be:

```python
{
    'Inception': [9, 10],
    'The Matrix': [8, 9],
    'The Godfather': [9]
}
```

In [732]:
```
# Solution goes here
```

Solution

```
In [733]: def organize_movie_ratings(ratings):
              movie_dict = {}

              for movie, rating in ratings:
                  if movie not in movie_dict:
                      movie_dict[movie] = []  # Initialize the list for new movies
                  movie_dict[movie].append(rating)  # Append the rating to the list

              return movie_dict
```

Test your code

Run this code to test your solution:

In [734]:
```
ratings = [
    ("Inception", 9),
    ("The Matrix", 8),
    ("Inception", 10),
    ("The Godfather", 9),
    ("The Matrix", 9)
]
expected_result = {
    'Inception': [9, 10],
    'The Matrix': [8, 9],
    'The Godfather': [9]
}
try: organize_movie_ratings(ratings) == expected_result and not print("Test #1 pa
except: print('Test #1 failed')
```

```
Test #1 passed
```

Test your code (cont'd)

In [735]:
```
ratings = [
    ("Titanic", 7),
    ("Titanic", 7),
    ("Titanic", 7)
]
expected_result = {
    'Titanic': [7, 7, 7]
}
try: organize_movie_ratings(ratings) == expected_result and not print("Test #2 pa
except: print('Test #2 failed')

ratings = [
    ("Avatar", 8),
    ("Avatar", 9),
    ("Avatar", 10)
]
expected_result = {
    'Avatar': [8, 9, 10]
}
try: organize_movie_ratings(ratings) == expected_result and not print("Test #3 pa
except: print('Test #3 failed')
```

```
Test #2 passed
Test #3 passed
```

Exercise 9: Contact Book

Write a function `create_contact_book` that:

- Takes a list of tuples as input, where each tuple contains two elements:
    - A string representing the name of a contact.
    - A string representing the contact's phone number.
- Returns a dictionary where:
    - The keys are the unique names of the contacts (case insensitive).
    - The values are the corresponding phone numbers.

NOTE: If a contact appears multiple times in the input list, the last occurrence should be kept in the dictionary.
NOTE: The names in the dictionary should be in lowercase to maintain case insensitivity.

Example

```
contacts = [
    ("Alice", "123-456-7890"),
    ("Bob", "987-654-3210"),
    ("alice", "555-555-5555"),
    ("Charlie", "111-222-3333")
]
result = create_contact_book(contacts)
```

The `result` should be:

```
{
    'alice': '555-555-5555',
    'bob': '987-654-3210',
    'charlie': '111-222-3333'
}
```

In [736]:
```
# Solution goes here
```

Solution

In [737]:
```python
def create_contact_book(contacts):
    contact_dict = {}

    for name, phone_number in contacts:
        # Use lowercase for names to ensure case insensitivity
        contact_dict[name.lower()] = phone_number

    return contact_dict
```

Test your code

```
In [738]: contacts = [
              ("Alice", "123-456-7890"),
              ("Bob", "987-654-3210"),
              ("alice", "555-555-5555"),
              ("Charlie", "111-222-3333")
          ]
          expected_result = {
              'alice': '555-555-5555',
              'bob': '987-654-3210',
              'charlie': '111-222-3333'
          }
          try: assert create_contact_book(contacts) == expected_result and not print("Test
          except: print('Test #1 failed')
```

```
Test #1 passed
```

Test your code (cont'd)

In [739]:
```
contacts = [
    ("John", "555-123-4567"),
    ("john", "555-765-4321"),
    ("Doe", "555-987-6543")
]
expected_result = {
    'john': '555-765-4321',
    'doe': '555-987-6543'
}
try: assert create_contact_book(contacts) == expected_result and not print("Test
except: print('Test #2 failed')

# Test Case 3: Only one contact
contacts = [
    ("Alice", "123-456-7890")
]
expected_result = {
    'alice': '123-456-7890'
}
try: assert create_contact_book(contacts) == expected_result and not print("Test
except: print('Test #3 failed')
```

```
Test #2 passed
Test #3 passed
```

Exercise 10: Library Management System

Write a function `manage_library` that:

- Takes a list of tuples as input, where each tuple contains:
    - A string representing the title of the book.
    - An integer representing the number (quantity) of copies to be added to or removed from the library.
      NOTE: If the quantity is negative, it means that books are being removed from the library.
- Returns a dictionary representing the current inventory of the library where:
    - The keys are unique book titles (case insensitive).
    - The values are dictionaries containing:
        - `total_copies` (key): as value, the total number of copies of the book available in the library (should not go below zero).
        - `available_copies` (key): as value, the number of copies currently available for borrowing (initially equal to `total_copies`).

Example

```python
library_updates = [
    ("The Great Gatsby", 5),
    ("1984", 10),
    ("the great gatsby", 2),
    ("1984", -3),
    ("To Kill a Mockingbird", 7),
    ("1984", -8),
    ("Moby Dick", -2)
]
result = manage_library(library_updates)
```

Example (cont'd)

The `result` should be:

```
{
    'the great gatsby': {
        'total_copies': 7,
        'available_copies': 7
    },
    '1984': {
        'total_copies': 2,
        'available_copies': 2
    },
    'to kill a mockingbird': {
        'total_copies': 7,
        'available_copies': 7
    },
    'moby dick': {
        'total_copies': 0,
        'available_copies': 0
    }
}
```

NOTE:

- If the quantity for a book goes below zero, it should not be removed from the inventory; instead, it should be set to zero for both `total_copies` and `available_copies`.
- The function should maintain case insensitivity for book titles (e.g., "The Great Gatsby" and "the great gatsby" should be treated as the same book).

In [740]:

```
# Solution goes here
```

Solution

```
In [741]: def manage_library(library_updates):
              library_inventory = {}

              for book_title, quantity in library_updates:
                  # Normalize book title to lowercase for case insensitivity
                  normalized_title = book_title.lower()

                  if normalized_title not in library_inventory:
                      library_inventory[normalized_title] = {
                          'total_copies': 0,
                          'available_copies': 0
                      }

                  # Update the total copies
                  library_inventory[normalized_title]['total_copies'] += quantity

                  # Ensure total copies do not go below zero
                  if library_inventory[normalized_title]['total_copies'] < 0:
                      library_inventory[normalized_title]['total_copies'] = 0

                  # Update available copies
                  library_inventory[normalized_title]['available_copies'] = library_invento

              return library_inventory
```

Test your code

```
In [742]:
library_updates = [
    ("The Great Gatsby", 5),
    ("1984", 10),
    ("the great gatsby", 2),
    ("1984", -3),
    ("To Kill a Mockingbird", 7),
    ("1984", -8),
    ("Moby Dick", -2)
]
expected_result = {
    'the great gatsby': { 'total_copies': 7, 'available_copies': 7 },
    '1984': { 'total_copies': 0, 'available_copies': 0 },
    'to kill a mockingbird': { 'total_copies': 7, 'available_copies': 7 },
    'moby dick': {'total_copies': 0, 'available_copies': 0 }
}
try: assert manage_library(library_updates) == expected_result and not print("Tes
except: print('Test #1 failed')
```

```
Test #1 passed
```

Test your code (cont'd)

```
In [743]: library_updates = [
              ("The Catcher in the Rye", 5),
              ("The Catcher in the Rye", -5),
              ("Brave New World", 10),
              ("Brave New World", -10)
          ]
          expected_result = {
              'the catcher in the rye': {
                  'total_copies': 0,
                  'available_copies': 0
              },
              'brave new world': {
                  'total_copies': 0,
                  'available_copies': 0
              }
          }
          try: assert manage_library(library_updates) == expected_result and not print("Tes
          except: print('Test #2 failed')

          library_updates = []
          expected_result = {}
          try: assert manage_library(library_updates) == expected_result and not print("Tes
          except: print('Test #3 failed')
```

```
Test #2 passed
Test #3 passed
```

Exercise 11: Social Media Connections

Write a function `manage_connections` that:

- Takes a list of tuples as input, where each tuple contains:
    - A string representing the username.
    - A set of strings representing the usernames of friends that the user is connected to.
- The function should return a dictionary representing each user and their unique connections (friends) where:
    - The keys are unique usernames (case insensitive).
    - The values are sets of unique friends for that user.

NOTE:

- If a user has multiple connections with the same friend, those should only be counted once.
- The function should maintain case insensitivity for usernames (e.g., "Alice" and "alice" should be treated as the same user).
- If a user has no friends, their value in the dictionary should be an empty set.

Example

```
connections = [
    ("Alice", {"Bob", "Charlie"}),
    ("Bob", {"Alice", "David"}),
    ("alice", {"Eve"}),
    ("Charlie", {"Bob"}),
    ("david", {"Alice", "Eve"}),
    ("Eve", set())
]
result = manage_connections(connections)
```

The `result` should be:

```
{
    'alice': {"bob", "charlie", "eve"},
    'bob': {"alice", "david"},
    'charlie': {"bob"},
    'david': {"alice", "eve"},
    'eve': set()
}
```

In [744]:
```
# Solution goes here
```

Solution

In [745]:
```python
def manage_connections(connections):
    user_connections = {}

    for user, friends in connections:
        # Normalize the username to lowercase for case insensitivity
        normalized_user = user.lower()

        # If the user is not in the dictionary, initialize their set
        if normalized_user not in user_connections:
            user_connections[normalized_user] = set()

        # Update the user's friends set with the provided friends
        user_connections[normalized_user].update(friends)

    # Normalize friends to be case insensitive as well
    for user in user_connections:
        user_connections[user] = {friend.lower() for friend in user_connections[u

    return user_connections
```

Test your code

Run this code to test your solution:

```
In [746]:
connections = [
    ("Alice", {"Bob", "Charlie"}),
    ("Bob", {"Alice", "David"}),
    ("alice", {"Eve"}),
    ("Charlie", {"Bob"}),
    ("david", {"Alice", "Eve"}),
    ("Eve", set())
]
expected_result = {
    'alice': {"bob", "charlie", "eve"},
    'bob': {"alice", "david"},
    'charlie': {"bob"},
    'david': {"alice", "eve"},
    'eve': set()
}
try: assert manage_connections(connections) == expected_result and not print("Tes
except: print('Test #1 failed')
```

```
Test #1 passed
```

Test your code (cont'd)

```
In [747]: connections = [
    ("John", set()),
    ("Doe", set())
]
expected_result = {
    'john': set(),
    'doe': set()
}
try: assert manage_connections(connections) == expected_result and not print("Tes
except: print('Test #2 failed')

connections = [
    ("Alice", {"Bob", "Charlie"}),
    ("Alice", {"Bob", "Eve"}),
    ("bob", {"Alice"}),
    ("charlie", {"Alice"}),
]
expected_result = {
    'alice': {"bob", "charlie", "eve"},
    'bob': {"alice"},
    'charlie': {"alice"},
}
try: assert manage_connections(connections) == expected_result and not print("Tes
except: print('Test #3 failed')
```

```
Test #2 passed
Test #3 passed
```

Exercise 12: Company Employee Records

Write a function `manage_employees` that:

- Takes a list of tuples as input, where each tuple contains:
    - A string representing the name of the department (e.g., "HR", "Engineering").
    - A string representing the name of the employee.
    - An integer representing the employee's salary (can be negative to indicate salary reductions).
- The function should return a dictionary representing each department's employees where:
    - The keys are unique department names (case insensitive).
    - The values are dictionaries containing:
        - `employees`: a dictionary of employee names (case insensitive) and their current salaries.
        - `average_salary`: the average salary of employees in that department, rounded to two decimal places.

NOTE:

- If an employee appears multiple times in the updates for the same department, update its salary condering the value as an increment or a reduction.
- If a salary goes below zero after an update, set it to zero.
- The function should maintain case insensitivity for department names and employee names by putting everything lowercase.
- If the input list is empty, return an empty dictionary.

Example

```
employee_updates = [
    ("HR", "Alice", 50000),
    ("Engineering", "Bob", 70000),
    ("HR", "Alice", -5000), # Salary reduction for Alice
    ("Engineering", "Charlie", 60000),
    ("HR", "Dave", 55000),
    ("Engineering", "Charlie", -10000), # Salary reduction for Charlie
    ("HR", "Eve", 45000)
]
result = manage_employees(employee_updates)
```

The `result` should be:

```
{
    'hr': {
        'employees': {
            'alice': 45000,
            'dave': 55000,
            'eve': 45000
        },
        'average_salary': 48333.33
    },
    'engineering': {
        'employees': {
            'bob': 70000,
            'charlie': 50000
        },
        'average_salary': 60000.0
    }
}
```

In [748]:
```
# Solution goes here
```

Solution

In [749]:
```python
def manage_employees(employee_updates):
    department_records = {}

    for department, employee_name, salary in employee_updates:
        # Normalize department and employee names to lowercase for case insensiti
        normalized_department = department.lower()
        normalized_employee = employee_name.lower()

        # If the department is not in the records, initialize it
        if normalized_department not in department_records:
            department_records[normalized_department] = {
                'employees': {},
                'average_salary': 0.0
            }

        # If the employee is not in the department, initialize their salary
        if normalized_employee not in department_records[normalized_department]['
            department_records[normalized_department]['employees'][normalized_emp

        # Update the employee's salary
        updated_salary = department_records[normalized_department]['employees'][n
        # Ensure the salary doesn't go below zero
        department_records[normalized_department]['employees'][normalized_employe

    # Calculate average salary for each department
    for department, records in department_records.items():
        employees = records['employees']
        if employees:
            total_salary = sum(employees.values())
            average_salary = total_salary / len(employees)
            records['average_salary'] = round(average_salary, 2)

    return department_records
```

Test your code

```
In [750]: employee_updates = [
              ("HR", "Alice", 50000),
              ("Engineering", "Bob", 70000),
              ("HR", "Alice", -5000),  # Salary reduction for Alice
              ("Engineering", "Charlie", 60000),
              ("HR", "Dave", 55000),
              ("Engineering", "Charlie", -10000), # Salary reduction for Charlie
              ("HR", "Eve", 45000)
          ]
          expected_result = {
              'hr': {
                  'employees': { 'alice': 45000, 'dave': 55000, 'eve': 45000 },
                  'average_salary': 48333.33
              },
              'engineering': {
                  'employees': { 'bob': 70000, 'charlie': 50000 },
                  'average_salary': 60000.0
              }
          }
          try: assert manage_employees(employee_updates) == expected_result and not print("
          except: print('Test #1 failed')
```

```
Test #1 passed
```

Test your code (cont'd)

```
In [751]: employee_updates = [
              ("HR", "Alice", 30000),
              ("HR", "Alice", -15000),  # Reduction
              ("HR", "Bob", 20000),
              ("Engineering", "Charlie", 100000),
              ("Engineering", "Charlie", -20000),
              ("HR", "Alice", -20000),  # Reduction to zero
          ]
          expected_result = {
              'hr': {
                  'employees': { 'alice': 0, 'bob': 20000 },
                  'average_salary': 10000.0
              },
              'engineering': {
                  'employees': { 'charlie': 80000 },
                  'average_salary': 80000.0
              }
          }
          try: assert manage_employees(employee_updates) == expected_result and not print("
          except: print('Test #2 failed')
```

```
Test #2 passed
```

Test your code (cont'd)

```
In [752]: employee_updates = [
              ("HR", "Alice", 30000),
              ("HR", "Alice", 25000),
              ("Engineering", "Bob", 50000),
              ("Engineering", "Bob", 10000),
              ("Engineering", "Bob", -20000),
          ]
          expected_result = {
              'hr': {
                  'employees': {
                      'alice': 55000
                  },
                  'average_salary': 55000
              },
              'engineering': {
                  'employees': {
                      'bob': 40000
                  },
                  'average_salary': 40000.0
              }
          }
          try: assert manage_employees(employee_updates) == expected_result and not print("
          except: print('Test #3 failed')
```

```
Test #3 passed
```