ASSIGNMENT 5 22BCA136

Q1. Write a Python function that takes two numbers and returns their sum.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:04:24 2024

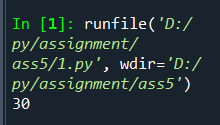
@author: ADMIN

"""

def sum\_numbers(a, b):

return a + b

print(sum\_numbers(10, 20))



Q2. Build a function that accepts a string and returns the number of vowels in it.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:05:08 2024

@author: ADMIN

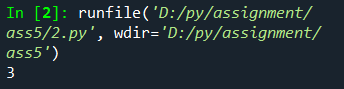
"""

def count\_vowels(s):

vowels = "aeiouAEIOU"

return sum(1 for char in s if char in vowels)

print(count\_vowels("Hello World"))



Q3. Create a function that checks if a given string is a palindrome.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:05:20 2024

@author: ADMIN

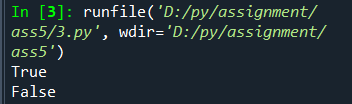
"""

def is\_palindrome(s):

return s == s[::-1]

print(is\_palindrome("madam"))

print(is\_palindrome("hello"))



Q4. Create a function that accepts a list of numbers and returns their average.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:05:29 2024

@author: ADMIN

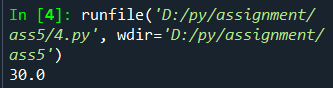
"""

def average(lst):

return sum(lst) / len(lst) if lst else 0

numbers = [10, 20, 30, 40, 50]

print(average(numbers))



Q5. Write a function that counts the frequency of a particular element in a list.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:05:33 2024

@author: ADMIN

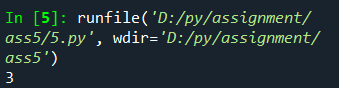
"""

def count\_frequency(lst, element):

return lst.count(element)

lst = [1, 2, 3, 1, 4, 1]

print(count\_frequency(lst, 1))



Q6. Create a function that accepts a list of strings and returns the longest string.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:05:43 2024

@author: ADMIN

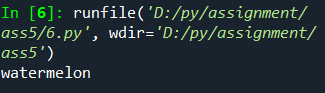
"""

def longest\_string(lst):

return max(lst, key=len) if lst else None

strings = ["apple", "banana", "cherry", "watermelon"]

print(longest\_string(strings))



Q7. Build a function that removes duplicates from a given list.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:05:55 2024

@author: ADMIN

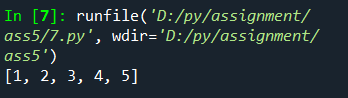
"""

def remove\_duplicates(lst):

return list(set(lst))

lst = [1, 2, 2, 3, 4, 4, 5]

print(remove\_duplicates(lst))



Q8. Write a function that accepts a strings and returns a dictionary.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:06:06 2024

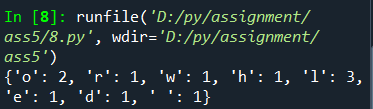
@author: ADMIN

"""

def char\_frequency(s):

return {char: s.count(char) for char in set(s)}

print(char\_frequency("hello world"))



Q9. Write a function that generates a multiplication table for a given number.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:06:16 2024

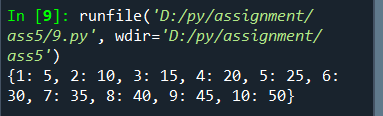
@author: ADMIN

"""

def multiplication\_table(n, upto=10):

return {i: n \* i for i in range(1, upto + 1)}

print(multiplication\_table(5))



Q10. Write a function that returns the intersection of two lists (common elements).

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:06:26 2024

@author: ADMIN

"""

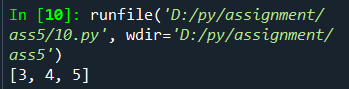
def list\_intersection(lst1, lst2):

return list(set(lst1) & set(lst2))

lst1 = [1, 2, 3, 4, 5]

lst2 = [3, 4, 5, 6, 7]

print(list\_intersection(lst1, lst2))



Q11. Write a function that accepts a dictionary and returns the key with the highest value.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:06:34 2024

@author: ADMIN

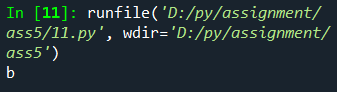
"""

def key\_with\_highest\_value(d):

return max(d, key=d.get)

d = {'a': 100, 'b': 300, 'c': 200}

print(key\_with\_highest\_value(d))



Q12. Create a function that takes a dictionary and returns a list of key-value tuples sorted by the values.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:06:43 2024

@author: ADMIN

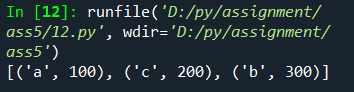
"""

def sort\_dict\_by\_values(d):

return sorted(d.items(), key=lambda item: item[1])

d = {'a': 100, 'b': 300, 'c': 200}

print(sort\_dict\_by\_values(d))



Q13. Write a function that takes two dictionaries and finds the keys that are common to both.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:06:51 2024

@author: ADMIN

"""

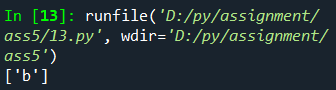
def common\_keys(d1, d2):

return list(set(d1.keys()) & set(d2.keys()))

d1 = {'a': 100, 'b': 200}

d2 = {'b': 300, 'c': 400}

print(common\_keys(d1, d2))



Q14. Write a program with python function to perform banking operations like current balance, withdraw, deposit with dictionary and manage the transaction history of each account holder. e.g. {0902:{‘name’:’suresh’,’current\_balance’:5000}, 0904:{‘name’:’jayesg’,’current\_balance’:15000} } transaction\_dict={0902:[[3000,‘d’,8000,11000], [5000,‘w’,2000,3000]],}

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:07:00 2024

@author: ADMIN

"""

def banking\_operations(accounts, transaction\_dict, acc\_id, operation, amount):

if operation == 'deposit':

accounts[acc\_id]['current\_balance'] += amount

transaction\_dict[acc\_id].append([amount, 'd', accounts[acc\_id]['current\_balance']])

elif operation == 'withdraw':

if accounts[acc\_id]['current\_balance'] >= amount:

accounts[acc\_id]['current\_balance'] -= amount

transaction\_dict[acc\_id].append([amount, 'w', accounts[acc\_id]['current\_balance']])

return accounts, transaction\_dict

accounts = {902: {'name': 'Suresh', 'current\_balance': 5000}, 904: {'name': 'Jayesh', 'current\_balance': 15000}}

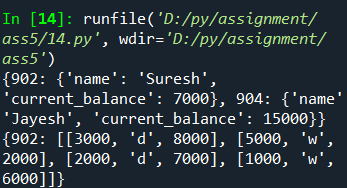
transaction\_dict = {902: [[3000, 'd', 8000], [5000, 'w', 2000]]}

accounts, transaction\_dict = banking\_operations(accounts, transaction\_dict, 902, 'deposit', 2000)

print(accounts)

accounts, transaction\_dict = banking\_operations(accounts, transaction\_dict, 902, 'withdraw', 1000)

print(transaction\_dict)



Q15. Write a program with python function to add products to cart, proceed to buy and print the invoice. Products={‘pid1’:{‘Name’:’Watch’,’Price’:400’}, ‘pid2’:{‘Name’:’Wallet’,’Price’:300’} } • Retrieve each product and print details • While retrieving, ask user to add product in cart • Ask quantity if user is adding product into the cart • Count the total item prices with product price \* quantity • Final cart dictionary structure would be

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:07:11 2024

@author: ADMIN

"""

def add\_to\_cart(cart, products, prod\_id, quantity):

if prod\_id in products:

product = products[prod\_id]

cart[prod\_id] = {

'Name': product['Name'],

'Price': product['Price'],

'Quantity': quantity,

'ItemAmount': product['Price'] \* quantity

}

return cart

def calculate\_total(cart):

return sum(item['ItemAmount'] for item in cart.values())

products = {'pid1': {'Name': 'Watch', 'Price': 400}, 'pid2': {'Name': 'Wallet', 'Price': 300}}

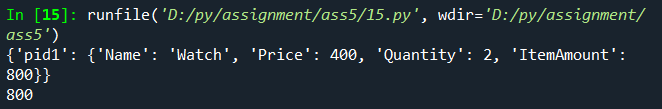
cart = {}

cart = add\_to\_cart(cart, products, 'pid1', 2)

print(cart)

total = calculate\_total(cart)

print(total)



Q16. Write a program with python function of dictionary with employee salary details and calculate the increment amount from number of years’ experience and update the latest salary. • employee\_dict={ ‘emp1’: {‘name’:’ramesh’,’salary’:15000,’expe’:2}, { ‘emp2’: {‘name’:’suresh’,’salary’:22000,’expe’:3} } • Retrieve the number of years of experience with the salary • Count the increment amount, if experience is less than equals 2 years then 10% increment or give 15% increment • Update the current salary with the after increment in the dict • print final employee salary

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:07:22 2024

@author: ADMIN

"""

def update\_salary(employee\_dict):

for emp\_id, details in employee\_dict.items():

increment = 0.10 if details['expe'] <= 2 else 0.15

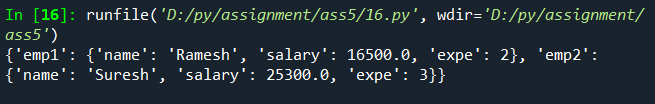
employee\_dict[emp\_id]['salary'] += employee\_dict[emp\_id]['salary'] \* increment

return employee\_dict

employee\_dict = {'emp1': {'name': 'Ramesh', 'salary': 15000, 'expe': 2}, 'emp2': {'name': 'Suresh', 'salary': 22000, 'expe': 3}}

employee\_dict = update\_salary(employee\_dict)

print(employee\_dict)



Q17. Write a program with python function to ensure that voter can give vote for only once and count the number of voters of the candidates. • Candtinate\_vote\_count={‘party1’:0,’party2’:0,’party2’:0} • We have empty voter\_set object to store the voter to has given the vote • So first, ask user to enter their name • Check whether that name in available in voter\_set. if available, means voter has already gave the vote so we don't allow to vote again • if not available, will ask enter to name of candidate, • will check, whether candidate name is correct or available in candidate\_vote\_count dictionary • update the vote count • Display each candidate vote count and display winner of the election

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:07:29 2024

@author: ADMIN

"""

def voting\_system(candidate\_votes, voter\_set, voter\_name, candidate\_name):

if voter\_name in voter\_set:

return "Already voted"

if candidate\_name in candidate\_votes:

candidate\_votes[candidate\_name] += 1

voter\_set.add(voter\_name)

return candidate\_votes, voter\_set

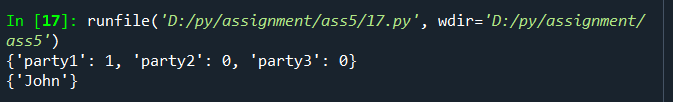
candidate\_votes = {'party1': 0, 'party2': 0, 'party3': 0}

voter\_set = set()

candidate\_votes, voter\_set = voting\_system(candidate\_votes, voter\_set, 'John', 'party1')

print(candidate\_votes)

print(voter\_set)



Q18. Write a program with python function to order a food from restaurant menu, user will select food items from the menu if available. User can also able to add or order to few more food items in order\_book. Count the total bill of user. e.g menu={‘burger’:{‘price’:50,’available’:True}, ‘pizza:{‘price’:150,’available’:False}, ‘Pasta:{‘price’:100,’available’:True} } Order\_book={tableid:[{‘fooditem’:’burget’,’quanlity’:2}, {‘fooditem’:pasta,’quanlity’:2}]}

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:07:37 2024

@author: ADMIN

"""

def add\_order(order\_book, menu, table\_id, food\_item, quantity):

if food\_item in menu and menu[food\_item]['available']:

if table\_id not in order\_book:

order\_book[table\_id] = []

order\_book[table\_id].append({'fooditem': food\_item, 'quantity': quantity})

return order\_book

def calculate\_total\_bill(order\_book, menu):

total\_bill = 0

for orders in order\_book.values():

for order in orders:

total\_bill += menu[order['fooditem']]['price'] \* order['quantity']

return total\_bill

menu = {'burger': {'price': 50, 'available': True}, 'pizza': {'price': 150, 'available': False}, 'pasta': {'price': 100, 'available': True}}

order\_book = {}

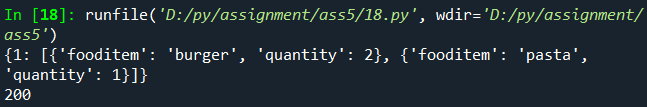
order\_book = add\_order(order\_book, menu, 1, 'burger', 2)

order\_book = add\_order(order\_book, menu, 1, 'pasta', 1)

print(order\_book)

total\_bill = calculate\_total\_bill(order\_book, menu)

print(total\_bill)



Q19. Write a program with python function to track fitness activities. Track users by their IDs, names, and activities (e.g., running, cycling) along with the duration. e.g fitness\_tracker = { 1: {'name': 'Eve', 'activities': {'running': 30, 'cycling': 60}}, 2: {'name': 'Frank', 'activities': {'swimming': 45}} } • Add new user into dictionary with their selected activities. • Add activities to user • Modify the activities duration

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:07:47 2024

@author: ADMIN

"""

def update\_fitness(fitness\_tracker, user\_id, name, activity, duration):

if user\_id not in fitness\_tracker:

fitness\_tracker[user\_id] = {'name': name, 'activities': {}}

if activity in fitness\_tracker[user\_id]['activities']:

fitness\_tracker[user\_id]['activities'][activity] += duration

else:

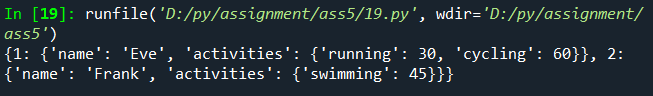
fitness\_tracker[user\_id]['activities'][activity] = duration

return fitness\_tracker

fitness\_tracker = {1: {'name': 'Eve', 'activities': {'running': 30, 'cycling': 60}}}

fitness\_tracker = update\_fitness(fitness\_tracker, 2, 'Frank', 'swimming', 45)

print(fitness\_tracker)



Q20. Write a program with python function to manage vehicle rentals. Track vehicles by their IDs, types, availability status, and rental rates. e.g. vehicles = { 101: {'type': 'Car', 'available': True, 'rate': 50}, 102: {'type': 'Bike', 'available': False, 'rate': 15} } • Add New vehicle for rental • User can search vehicles and show all available list of vehicles. • User can take vehicles on rent based on hours and will update status in vehicles and user dict. • User will return the vehicles and will update status in vehicles and user dict. • Manage user dictionary of to take and return the vehicles User={userid:[{vehicleid: 101, ‘Status’:’return’, ‘hour’:10}], userid:[{vehicleid: 102, ‘Status’:’Rent’,’hour’:5}] } • Count user bill

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 23 08:07:56 2024

@author: ADMIN

"""

def rent\_vehicle(vehicles, user, user\_id, vehicle\_id, hours):

if vehicles[vehicle\_id]['available']:

vehicles[vehicle\_id]['available'] = False

if user\_id not in user:

user[user\_id] = []

user[user\_id].append({'vehicleid': vehicle\_id, 'Status': 'Rent', 'hour': hours})

return vehicles, user

def return\_vehicle(vehicles, user, user\_id, vehicle\_id):

for record in user[user\_id]:

if record['vehicleid'] == vehicle\_id and record['Status'] == 'Rent':

vehicles[vehicle\_id]['available'] = True

record['Status'] = 'Return'

return vehicles, user

def calculate\_bill(user, user\_id, vehicles):

return sum(record['hour'] \* vehicles[record['vehicleid']]['rate'] for record in user[user\_id])

vehicles = {101: {'type': 'Car', 'available': True, 'rate': 50}, 102: {'type': 'Bike', 'available': False, 'rate': 15}}

user = {}

vehicles, user = rent\_vehicle(vehicles, user, 1, 101, 5)

print(user)

vehicles, user = return\_vehicle(vehicles, user, 1, 101)

print(vehicles[101]['available'])

bill = calculate\_bill(user, 1, vehicles)

print(bill)

