Python Coding Assessment

# Instructions

* Answer all questions to the best of your ability.
* Do not use the internet or external assistance to complete the assessment. This includes the use of LLMs such as ChatGPT.
* Do NOT spend more than 2 hours on the assessment.
* Give pseudocode where appropriate.
* Explain your reasoning for your answers where appropriate.

# Section 1: Knowledge Questions

## Python Fundamentals

1. What is the difference between a list, tuple, and set in Python? Give examples.

List : Ordered, Allow duplicates, Muttable (declare with square brackets, access by index value. Index value start with zero)

A = [1,2,3,4]

Tuple : Ordered, Allow duplicates, Immutable (declare with rouded brackets, access by index value. Index value start with zero)

A = (1,2,3)

Set : Unordered, No duplicates, Immutable (declare with curly brackets. One dimensional)

A = {1,2,3}

Dict : Unordered, No duplicates, Muttable (declare with curly brackets, two or multidimensional, access by key. Key must be unique)

A = {‘name’:’surendar’}

1. Explain how Python’s memory management works.

Python memory management works on heap and garbage collection mechanism

Heap -> it stores all the default data structure and default functions in memory like (list, set, dict, min, max, sum)

Garbage collection -> it works on reverse count algorithm. If the reverse count is zero the memory will be free

For ex:

A = 1 # count is 1

B = A # count is 2

Del B # count is 1

Del A # count is 0. Allocated memory automatically removed

1. What is a Python decorator? Give a use-case and example.

Decorator is like a wrapup mechanisum. Its will change the actual behaivour of the functions without changing the function code. Declare with @ key before the function.

Example :

def dec\_call(func):

def wrapper():

print(“start”)

func()

print(“stop”)

return wrapper()

@dec\_call

def main():

Print(“do something”)

main()

## OOP in Python

1. Explain the difference between class attributes and instance attributes.

Class is a collection of data members and member function. Data members are arguments or values that are going to use inside a function. Member function are behaiviour of the particular function.

Instance -> Instance are nothing but an object. Instance of the class is object. Each class can have a multiple objects.

class Person:

def \_\_init\_\_(self, name):

self.name = name # data member

def display(self ): #member function

return “Hi” + self.name

obj = Person(‘surendar’) #instance 1

obj1 = Person(‘raj) #instance 1

obj.display()

obj1.display()

1. What are @staticmethod, @classmethod, and instance methods?

Staticmethod – static method is a normal function that is declared inside the class without passing self args. Static method are used to do some small validations are write business logics inside the class. These function cannot be access outside of the class.

To declare @staticmethod above the function.

Class Method – class method Is used to access the class level values. For that we have to pass cls argument and return the values with in the cls fuction. To declare @classmethod above the function.

1. Provide a Python example using inheritance and polymorphism.

Inheritance:

One class acquires the values of another class. Get the base class value by declaring it in \_\_Init\_\_ function using super() key. There several types of inheritance

Single Inheritance -> base class to derived class

Multiple Inheritance -> more than one base class to one derived class

Multilevel Inheritance -> one class acquires the property of another class which already a derived class like (grandfather -> father -> son)

Hierarchical inheritance -> More than one derived class acquires the properties of single base class

Example;

Class Vehicle:

def \_\_init\_\_(self, color):

self. color = color

def vehicle\_color(self):

return self.color

class Car(Vehicle):

def \_\_init\_\_(self, engine):

super().\_\_init\_\_(color)

self.engine = engine

def display(self):

return f”{self.engine} {self.color}”

a = Car(‘s1’, ‘red’)

a.display()

Polymorphisum:

One that can take many form. For example class name is difference inside the function name is same perform different behaviours

class Car:

def display():

return ‘car’

class Bike:

def display():

return ‘Bike’

a = [Car(), Bike()]

for i in a:

print(i.display())

## Error Handling

1. How is exception handling done in Python?

There is two ways to handle the exceptions.

We are exception some error and want to raise a exception and stop the program we can use raise. Value error and stop the program.

We are not exception some error and don’t want to stop the program and continue the next process means we have to use try except

try:

// do something

except Exception as Err:

print(Err)

1. What is the purpose of the finally block?

Finally block it will run the statement. Whatever it may be the result either exception or the statement running successfully does’t matter its simply run the statement after running try and except.

## Async and Multithreading

1. What is the difference between asyncio, threading, and multiprocessing?

Asyncio -> asyncio it’s a package. It will create the task in the background and run it in the background. It wont wait for the results. It will simply run the statement and move on to the next statement

Threading -> Do simuntaneously task and doest wait for the results

Multiprocessing -> Do simultaneously run the program and wait for the results and append it to the result

1. Provide an example where asyncio is better than multithreading.

Asyncio is simple to declare and run while multithreading we have to declare a separate functions to call it

# Section 2: Code Bug Fixing and Optimization

## Question 1: Python Function Bug Fixing

The following function fetches data from an API but has multiple issues. Identify and fix the bugs and suggest improvements.

import requests  
def get\_user\_data(user\_id):  
 url = f"https://api.example.com/user/{user\_id}"  
 response = requests.get(url, headers = {})  
 if response.status\_code == 200:  
 return response.json()  
 else:  
 print("Failed")

## Question 2: Python Class Bug Fixing

The following Python class is meant to track and summarize user purchases but has bugs and lacks optimization. Identify and fix the issues.  
  
class User:  
 def \_\_init\_\_(self, id, name, purchase = []):  
 self.id = id  
 self.name = name  
 self.purchases = purchase  
  
 def add\_purchase(self, item, amount):  
 self.purchases.append({item: amount})

return self.purchases

def total\_spent(self):   
 return sum(list(p.values()))

# Section 3: Code Functions – With Answers

## Question 1: File Automation

Write a Python function that monitors a folder for new `.csv` files, reads them, summarizes the number of rows and columns, and logs the result.  
- The function should:  
 - Use async or concurrent processing for file reading.  
 - Handle large files gracefully.  
 - Log meaningful errors and results.  
 - Include a retry mechanism on file read failure.

For reading the csv file we can use pandas library. It’s way faster than the predefined python packages and also it will give us clean and releavent data which is very important in data science. Also it will give us the structured data frames format

Import pandas as pd

df = pd.read\_csv(‘data.csv’)

Print(df)

## Question 2: Data Aggregation

You are given the following list of dictionaries representing test scores:  
  
data = [  
 {'student': 'Alice', 'subject': 'Math', 'score': 90},  
 {'student': 'Bob', 'subject': 'Math', 'score': 95},  
 {'student': 'Alice', 'subject': 'Science', 'score': 85},  
 {'student': 'Bob', 'subject': 'Science', 'score': 91},  
 {'student': 'Charlie', 'subject': 'Math', 'score': 95}  
]  
  
1. Return a list of students who didn’t appear for any exams (assume a master list of students).

Master\_list = [‘surendar’, ‘rahul’, ‘Alice’, ‘Bob’ ‘Charlie’]

val = list(set([i['student'] for i in data]))

not\_attend = [i for i in Master\_list if i not in val]

2. Identify the top scorer(s) in each subject. If two students share the top score, return both.

subject = list(set([i['subject'] for i in data]))

op = []

for i in subject:

    values= []

    for j in data:

        if j['subject'] == i:

            values.append(j)

    for idx, k in enumerate(values):

        if idx+1 < len(values) and k['score'] <= values[idx+1]['score']:

            op.append(values[idx+1])

print(op)

# Section 4: System Design – Email Automation System

**Design a Python-Based Automation System for Processing Incoming Support Emails**

Build a high-level design for a system that automatically processes support emails and performs the following:

* Classifies emails based on subject or content (e.g., bug report, feature request, account issue).
* Extracts key metadata (sender, priority, ticket ID).
* Pushes structured data into a ticketing system via an API.
* Sends automated responses using templates.
* Uses a queue for asynchronous processing.
* Scales to handle thousands of emails daily.

**Your design should include:**

* A high-level architecture overview (describe in text or provide a simple ASCII diagram).
* Key components (e.g., parser, classifier, queue, API client).
* Technologies or frameworks you’d use (e.g., FastAPI, Celery, Redis, scikit-learn).
* Approach to testing and logging.
* Optional: basic schema or data structure definitions for email data.

from fastapi import FastAPI

import email, re, requests, logging

app = FastAPI()

data = ['bug report', 'feature request', 'account issue']

@app.repeat\_every(5)

def call():

    # call external email

    email\_data = email('username', 'password')

    for i in email\_data:

        try:

            if i['subject'] in data:

                sender = re.findall('sender', i['content'])

                priority = re.findall('priority', i['content'])

                ticket = re.findall('ticket ID', i['content'])

                payload = {

                    'sender': sender,

                    'priority': priority,

                    'ticket': ticket

                }

                url = "http://www.report.com/"

                res = requests.post(url, headers={}, json=payload)

                if res.status\_code == 200:

                    logging.info(f'ticket raised successfully for {sender}: {ticket}')

                else:

                    logging.error(f"failed {sender}: {ticket} {res.text}")

        except Exception as Err:

            logging.error(f"failed : {Err}")