PRATEEK

PYTHON OOPS TEST

QUESTION-1) TELECOM SMART RECOMMENDER

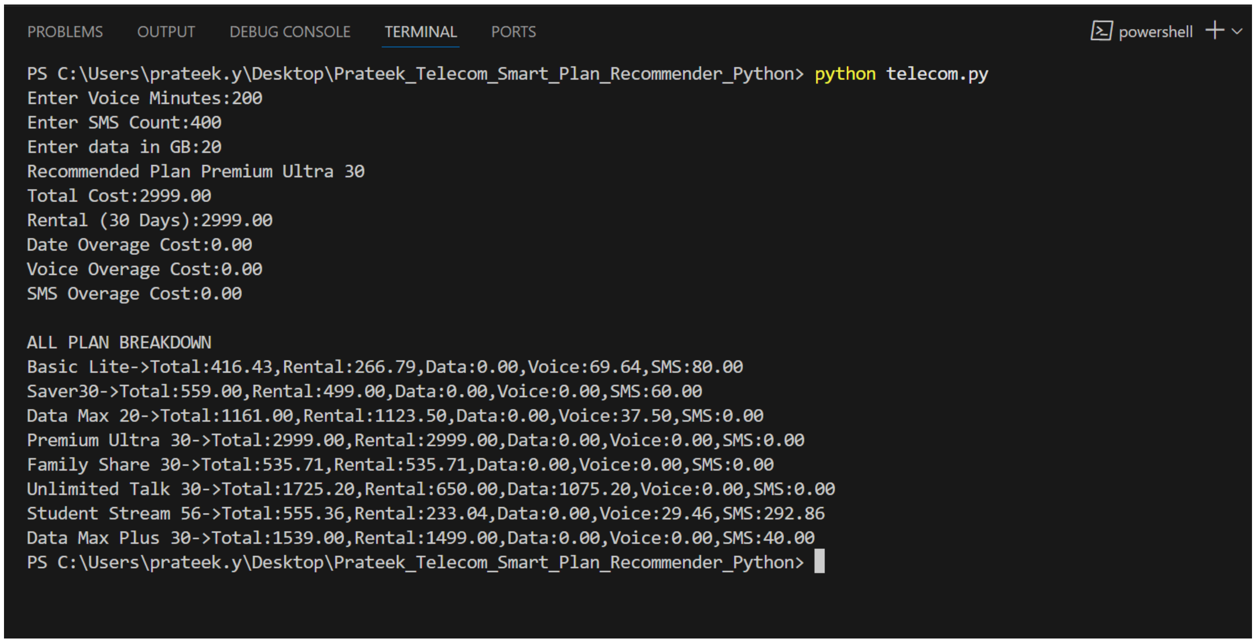
CODE:

```
#THE CORE BUSINESS LOGIC IS TO COMPUTE THE 30 DAY CHARGE FOR EACH PLAN

#WE'LL START WITH IMPLEMENTING THE CLASSES FIRST

from abc import ABC,abstractmethod
from dataclasses import dataclass
from typing import List,Optional
from enum import Enum

#We'll start by making the core classes as explicitly mentioned in the document

#Creating a list of ott services which expects an enum
class OTTService(Enum):
    NETFLIX="Netflix"
    PRIME="Amazon Prime"
    HOTSTAR="Hotstar"
    SPOTIFY="Spotify"

@dataclass #It automatically involves the constructors like __init__ etc
class Usage:
    voice_minutes:int
    sms_count:int
    data_mb:int

@dataclass
class OTTRequirement:#setting requirements for ott as asked
    netflix:bool=False
    prime:bool=False
    hotstar:bool=False
    spotify:bool=False

@dataclass
class PlanQuote:#to calculate for whole month the data
    plan_name:str
    rental_30d:float
    data_overage:float
    voice_overage:float
    sms_overage:float
    total_cost:float
    ott_service:List[str]

class Plan(ABC):
    def __init__(self,name:str,cost:float,validity_days:int):
        self.name=name
        self.cost=cost
        self.validity_days=validity_days
        self.ott_services=[]

    def get_normalized_rental(self)->float:#some plans are for 28 days or so to
get normalized values
        return (self.cost/self.validity_days)*30

    @abstractmethod
    def calculate_data_cost(self,usage_mb:int)->float:
        pass

    @abstractmethod
    def calculate_voice_cost(self,minutes:int)->float:
        pass

    #HERE TO CALCULATE OTT REQUIREMENTS WE ARE PASSING THE INITIAL OTT SERVICE
    def meets_ott_requirements(self,requirements:OTTRequirement)->bool:
        required_services=[]
        if requirements.netflix:
            required_services.append(OTTService.NETFLIX.value)
        if requirements.prime:
            required_services.append(OTTService.PRIME.value)
        if requirements.spotify:
            required_services.append(OTTService.SPOTIFY.value)
        if requirements.hotstar:
            required_services.append(OTTService.HOTSTAR.value)
        return all(service in self.ott_services for service in required_services)
        #it is generating whether the initilised bool value for corresponding ott
is there in the required_services

    @abstractmethod
    def calculate_sms_cost(self,sms_count:int)->bool:
```

```python
        pass

    def price(self,usage:Usage)->PlanQuote:#USING THE VALUES FROM USAGE TO
GENERATE QUOTE(LINE 20)
        rental=self.get_normalized_rental()
        data_cost=self.calculate_data_cost(usage.data_mb)
        voice_cost=self.calculate_voice_cost(usage.voice_minutes)
        sms_cost=self.calculate_sms_cost(usage.sms_count)

        #GOT THE QUOTE->REFER LINE 33
        return PlanQuote(
            plan_name=self.name,
            rental_30d=rental,
            data_overage=data_cost,
            voice_overage=voice_cost,
            sms_overage=sms_cost,
            total_cost=rental+data_cost+voice_cost+sms_cost,
            ott_service=self.ott_services.copy()
        )

#NOW WE WILL IMPLEMENT DATA PLANS

class BasicLitePlan(Plan):#For basic lite sms 0.20rs,0.75.min after 100min
    def __init__(self):
        super().__init__("Basic Lite",249,28)#REFER LINE 43
        self.daily_data_gb=1
        self.included_voice_mins=100
        self.ott_services=[]

    def calculate_data_cost(self, usage_mb:int)->float:
        daily_allowance_mb=self.daily_data_gb*1024#to convert into mb
        total_allowance_mb=daily_allowance_mb*30
        if usage_mb<=total_allowance_mb:
            return 0
        #calculating overusage
        overage_mb=usage_mb-total_allowance_mb
        overage_block=(overage_mb+9)//10
        return overage_block*0.7
    def calculate_voice_cost(self, minutes:int)->float:
        normalized_included=(self.included_voice_mins/self.validity_days)*30
        if(minutes<=normalized_included):
            return 0
        overage_minutes=minutes-normalized_included
        return overage_minutes*0.75

    def calculate_sms_cost(self, sms_count:int)->float:
        return sms_count*0.20

class Saver30Plan(Plan):
    def __init__(self):
        super().__init__("Saver30",499,30)
        self.daily_data_gb=1.5
        self.included_voice_mins=300
        self.included_sms=100
        self.ott_services=[OTTService.HOTSTAR.value]

    def calculate_data_cost(self,usage_mb:int)->float:
        daily_allowance_mb=self.daily_data_gb*1024
        total_allowance_mb=daily_allowance_mb*30

        if usage_mb<=total_allowance_mb:
            return 0
        overage_mb=usage_mb-total_allowance_mb
        overage_block=(overage_mb+9)//10
        return overage_block*0.70
    def calculate_voice_cost(self,minutes:int)->float:
        if minutes<=self.included_voice_mins:
            return 0
        return (minutes-self.included_voice_mins)*0.75
    def calculate_sms_cost(self,sms_count:int)->float:
        if sms_count<=self.included_sms:
            return 0
        return (sms_count-self.included_sms)*0.20

class UnlimitedTalk30Plan(Plan):
    def __init__(self):
        super().__init__("Unlimited Talk 30",650,30)
        self.total_data_gb=5
        self.ott_services=[OTTService.SPOTIFY.value]

    def calculate_data_cost(self,usage_mb:int)->float:
        total_allowance_mb=self.total_data_gb*1024

        if usage_mb<=total_allowance_mb:
            return 0
        overage_mb=usage_mb-total_allowance_mb
        overage_block=(overage_mb+9)//10
        return overage_block*0.70
    def calculate_voice_cost(self, minutes:int)->float:#unlimited
        return 0
    def calculate_sms_cost(self, sms_count:int)->float:#unlimited
        return 0

class FamilyShare30Plan(Plan):
    def __init__(self):
        super().__init__("Family Share 30",500,28)
        self.total_data_gb=50
        self.included_voice_mins=1000
        self.included_sms=500
        self.ott_services=[OTTService.PRIME.value]

    def calculate_data_cost(self,usage_mb:int)->float:
        normalized_data_gb=(self.total_data_gb/self.validity_days)*30
        total_allowance_mb=normalized_data_gb*1024
        if usage_mb<=total_allowance_mb:
            return 0
        overage_mb=usage_mb-total_allowance_mb
        overage_block=(overage_mb+9)//10
        return overage_block*0.70

    def calculate_voice_cost(self, minutes:int)->float:
        normalized_included=(self.included_voice_mins/self.validity_days)*30
        if minutes<=normalized_included:
            return 0
        return (minutes-normalized_included)*0.60
    def calculate_sms_cost(self, sms_count:int)->float:
        normalized_included=(self.included_sms/self.validity_days)*30
        if(sms_count<=normalized_included):
            return 0
        return (sms_count-normalized_included)*0.20

#IMPLEMENTING UNLIMITED DATA PLANS

class DataMax20Plan(Plan):
    def __init__(self):
        super().__init__("Data Max 20",749,20)
        self.included_voice_mins=100
        self.ott_services=[OTTService.HOTSTAR.value]
    def calculate_data_cost(self, usage_mb:int)->float:
        return 0
    def calculate_voice_cost(self, minutes:int)->float:
        normalized_included=(self.included_voice_mins/self.validity_days)*30
        if minutes<=normalized_included:
            return 0
        return (minutes-normalized_included)*0.75
    def calculate_sms_cost(self, sms_count:int)->float:
        return 0

class PremiumUltra30Plan(Plan):
    def __init__(self):
        super().__init__("Premium Ultra 30",2999,30)
        self.included_voice_mins=100

self.ott_services=[OTTService.HOTSTAR.value,OTTService.NETFLIX.value,OTTService.S
POTIFY.value,OTTService.PRIME.value]

    def calculate_data_cost(self, usage_mb:int)->float:
        return 0
    def calculate_voice_cost(self, minutes:int)->float:
        return 0
    def calculate_sms_cost(self, sms_count:int)->float:
        return 0

class StudentStream56Plan(Plan):
    def __init__(self):
        super().__init__("Student Stream 56",435,56)
        self.daily_data_gb=2
        self.included_voice_mins=300
        self.included_sms=200
        self.ott_services=[OTTService.SPOTIFY.value]

    def calculate_data_cost(self, usage_mb:int)->float:
        daily_allowance_mb=self.daily_data_gb*1024
        total_allowance_mb=daily_allowance_mb*30
        if usage_mb<=total_allowance_mb:
            return 0
        overage_mb=usage_mb-total_allowance_mb
        overage_block=(overage_mb+9)//10
        return overage_block*0.70
    def calculate_sms_cost(self, sms_count:int)->float:
        normalized_included=(self.included_sms/self.validity_days)*30
        if sms_count<=normalized_included:
            return 0
        return (sms_count-normalized_included)
    def calculate_voice_cost(self, minutes:int)->float:
        normalized_included=(self.included_voice_mins/self.validity_days)*30
        if(minutes<=normalized_included):
            return 0
        return (minutes-normalized_included)*0.75

class DataMaxPlus30Plan(Plan):
    def __init__(self):
        super().__init__("Data Max Plus 30",1499,30)
        self.included_voice_mins=300
        self.include_sms=200
        self.ott_services=[OTTService.PRIME.value,OTTService.HOTSTAR.value]
    def calculate_data_cost(self, usage_mb:int)->float:
        return 0
    def calculate_voice_cost(self, minutes:int)->float:
        if minutes<=self.included_voice_mins:
            return 0
        return (minutes-self.included_voice_mins)*0.75
    def calculate_sms_cost(self, sms_count:int)->float:
        if sms_count<=self.include_sms:
            return 0
        return (sms_count-self.include_sms)*0.20
#NOW A CLASS FOR RECOMMENDATION

class PlanRecommender:
    def __init__(self):

self.plans=[BasicLitePlan(),Saver30Plan(),DataMax20Plan(),PremiumUltra30Plan(),Fa
milyShare30Plan(),UnlimitedTalk30Plan(),StudentStream56Plan(),DataMaxPlus30Plan()
]

    def get_eligible_plans(self,ott_requirements:OTTRequirement)->List[Plan]:
        return [plan for plan in self.plans if
plan.meets_ott_requirements(ott_requirements)]

    def recommend_plan(self,usage:Usage,ott_requirements:OTTRequirement):
        eligible_plans=self.get_eligible_plans(ott_requirements)

        if not eligible_plans:
            return None
```

```python
        quotes=[plan.price(usage) for plan in eligible_plans]
        best_quote=min(quotes,key=lambda q:(q.total_cost,-len(q.ott_service)))
        return best_quote
    def generate_all_quotes(self,usage:Usage)->List[PlanQuote]:
        return [plan.price(usage) for plan in self.plans]




def main():
    #instead we can also use while
    voice_minutes1=int(input("Enter Voice Minutes:"))
    sms_count1=int(input("Enter SMS Count:"))
    data_gb=int(input("Enter data in GB:"))
    data_gb=data_gb*1024


usage=Usage(voice_minutes=voice_minutes1,sms_count=sms_count1,data_mb=data_gb)
    ott_req=OTTRequirement(netflix=True)#HERE ADDING OTT REQUIREMENT
    recommmeder=PlanRecommender()

    recommended=recommmeder.recommend_plan(usage,ott_req)

    if recommended:
        print(f"Recommended Plan {recommended.plan_name}")
        print(f"Total Cost:{recommended.total_cost:.2f}")
        print(f"Rental (30 Days):{recommended.rental_30d:.2f}")
        print(f"Date Overage Cost:{recommended.data_overage:.2f}")
        print(f"Voice Overage Cost:{recommended.voice_overage:.2f}")
        print(f"SMS Overage Cost:{recommended.sms_overage:.2f}")
    else:
        print("No Plan Meet the Criteria")

    print("\nALL PLAN BREAKDOWN")

    for quote in recommmeder.generate_all_quotes(usage):
        print(f"{quote.plan_name}-
>Total:{quote.total_cost:.2f},Rental:{quote.rental_30d:.2f},Data:{quote.data_over
age:.2f},Voice:{quote.voice_overage:.2f},SMS:{quote.sms_overage:.2f}")
if __name__=="__main__":
    main()
```

OUTPUT:



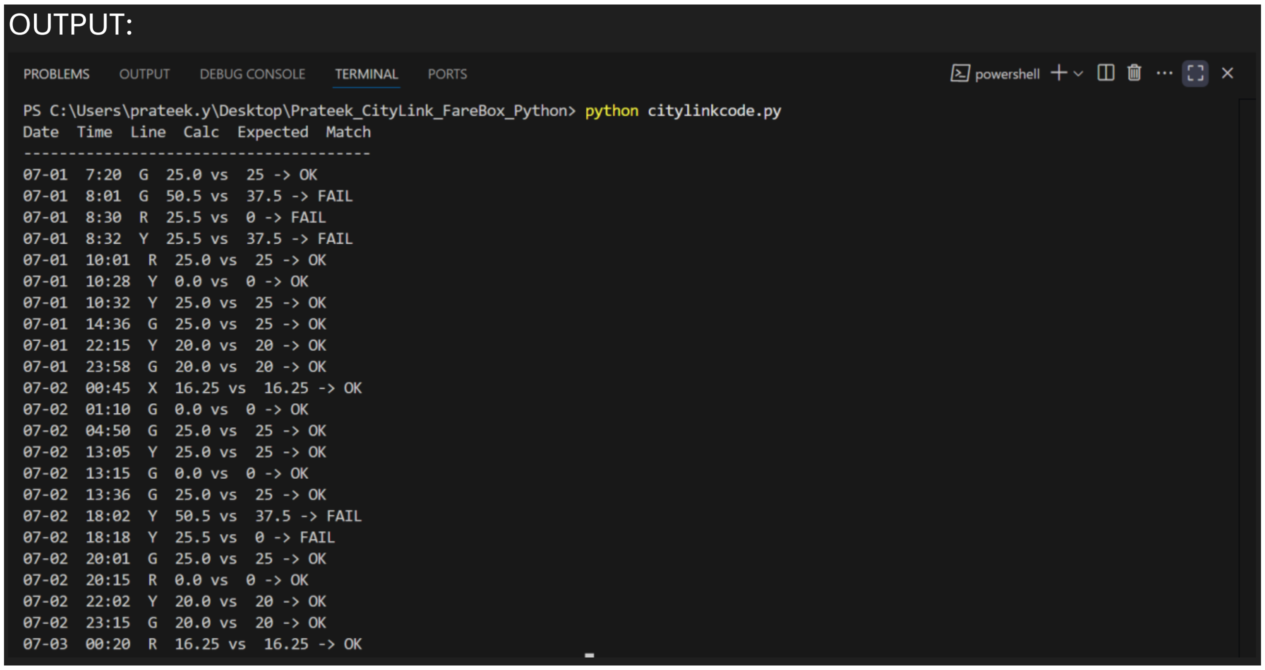
```
PS C:\Users\prateek.y\Desktop\Prateek_Telecom_Smart_Plan_Recommender_Python> python telecom.py
Enter Voice Minutes:200
Enter SMS Count:400
Enter data in GB:20
Recommended Plan Premium Ultra 30
Total Cost:2999.00
Rental (30 Days):2999.00
Date Overage Cost:0.00
Voice Overage Cost:0.00
SMS Overage Cost:0.00

ALL PLAN BREAKDOWN
Basic Lite->Total:416.43,Rental:266.79,Data:0.00,Voice:69.64,SMS:80.00
Saver30->Total:559.00,Rental:499.00,Data:0.00,Voice:0.00,SMS:60.00
Data Max 20->Total:1161.00,Rental:1123.50,Data:0.00,Voice:37.50,SMS:0.00
Premium Ultra 30->Total:2999.00,Rental:2999.00,Data:0.00,Voice:0.00,SMS:0.00
Family Share 30->Total:535.71,Rental:535.71,Data:0.00,Voice:0.00,SMS:0.00
Unlimited Talk 30->Total:1725.20,Rental:650.00,Data:1075.20,Voice:0.00,SMS:0.00
Student Stream 56->Total:555.36,Rental:233.04,Data:0.00,Voice:29.46,SMS:292.86
Data Max Plus 30->Total:1539.00,Rental:1499.00,Data:0.00,Voice:0.00,SMS:40.00
PS C:\Users\prateek.y\Desktop\Prateek_Telecom_Smart_Plan_Recommender_Python>
```

QUESTION-2) CITY FAREBOX CALCULATOR

CODE:

```python
#THE BASIC CONCEPT IS TO IMPLEMENT A FAREBOX SYSTEM BASED ON THE REQUIREMENTS AND
THEN TEST IT ON A DATA

#WE HAVE TO INCLUDE BASE FARES-25 AND PEAK SURCHARGE,NIGHT DISCOUNTS AND STUFF



from abc import ABC,abstractmethod
from datetime import datetime,timedelta
from dataclasses import dataclass
from typing import List,Optional

@dataclass#USED TO GENERATE SPECAIL METHODS LIKE __INIT__ ON IT'S OWN
class Tap:
    datetime:datetime
    line:str #Metro/Rail Line
    station:str  #Station Code
    @classmethod #CLS REPRESENTS the class itself
    def from_string(cls,date_str:str,time_str:str,line:str,station:str):
        dt_str=f"2025-{date_str} {time_str}"
        dt=datetime.strptime(dt_str,"%Y-%m-%d %H:%M")
        return cls(dt,line,station)

class FareCalculationContext:
    def __init__(self):
        self.trip_history:List[Tap]=[]#to keep a record of the trips
        self.last_paid_tap:Optional[Tap]=None #Last tap
    def add_trip(self,tap:Tap):
        self.trip_history.append(tap)
    def update_last_paid(self,tap:Tap):
        self.last_paid_tap=tap

class FareRule(ABC):
    def __init__(self,enabled:bool=True):
        self.enabled=enabled
    @abstractmethod
    def calculate_fare(self,tap:Tap,context:FareCalculationContext)-
>float:#abstract method we will change it
        pass
    @abstractmethod
    def applies_to(self,tap:Tap,context:FareCalculationContext)->bool:
        pass
class BaseFareRule(FareRule):#USING INHERITANCE TO INHERI FARERULE
    #base fare of 25 irrespective of the line
    BASE_FARE=25.0
    def applies_to(self, tap:Tap, context:FareCalculationContext):
        return self.enabled
    def calculate_fare(self, tap:Tap, context:FareCalculationContext)->float:
        return self.BASE_FARE if self.enabled else 0.0
class PeakHourSurchargeRule(FareRule):
    SURCHARGE_RATE=0.5
    def applies_to(self, tap:Tap, context:FareCalculationContext)->bool:
        if not self.enabled:
            return False
        hour=tap.datetime.hour
        return (8<=hour<10) or (18<=hour<20) #these are the peak hours so
surcharge will be applied
    def calculate_fare(self, tap:Tap, context:FareCalculationContext)->float:
        if self.applies_to(tap,context):
            return BaseFareRule.BASE_FARE+self.SURCHARGE_RATE
        return 0.0
class TranferWindowRule(FareRule):
    TRANSFER_WINDOW_MINUTES=30

    def applies_to(self, tap:Tap, context:FareCalculationContext)->bool:
        if not self.enabled or not context.last_paid_tap:
            return False
        time_diff=tap.datetime-context.last_paid_tap.datetime
        return time_diff<=timedelta(minutes=self.TRANSFER_WINDOW_MINUTES)
    def calculate_fare(self, tap:Tap, context:FareCalculationContext)->float:
        if self.applies_to(tap,context):
            return -BaseFareRule.BASE_FARE
        return 0.0 #FREE RIDE WITHIN 30 MINUTES OF LAST PAID THAT IS WHY WE ARE
APPLYING THIS LOGIC
class NightDiscountRule(FareRule):
    DISCOUNT_RATE=0.2

    def applies_to(self, tap:Tap, context:FareCalculationContext)->bool:
        if not self.enabled:
            return False
        return 22<=tap.datetime.hour<=23
    def calculate_fare(self, tap:Tap, context:FareCalculationContext)->float:
        return -BaseFareRule.BASE_FARE*self.DISCOUNT_RATE if
self.applies_to(tap,context) else 0

    #FROM 10 PM AND MIDNIGHT WE HAVE 20% DISCOUNT SO THAT IS WHY WE ARE USING
THIS LOGIC HERE

class PostMidnightDiscountRule(FareRule):
    DISCOUNT_RATE=0.35

    def applies_to(self, tap:Tap, context:FareCalculationContext)->bool:
        if not self.enabled:
            return False
        return 0<=tap.datetime.hour<4

    def calculate_fare(self, tap:Tap, context:FareCalculationContext)->float:
        return -BaseFareRule.BASE_FARE*self.DISCOUNT_RATE if
self.applies_to(tap,context) else 0.0

    #BECASUE 35% DISCOUNT IF TAP BETWEEN MIDNIGHT AND 4



class TariffEngine:

    def __init__(self):

self.rules:List[FareRule]=[BaseFareRule(),PeakHourSurchargeRule(),TranferWindowRu
le(),

NightDiscountRule(),PostMidnightDiscountRule()]
        self.context=FareCalculationContext()
    def toggle_rule(self,rule_type:type,enabled:bool):
```

```python
        for rule in self.rules:
            if(isinstance(rule,rule_type)):
                rule.enabled=enabled
                break
    def calculate_fare(self,tap:Tap)->float:
        total_fare=0.0
        for rule in self.rules:
            total_fare+=rule.calculate_fare(tap,self.context)
        total_fare=max(0.0,total_fare)
        self.context.add_trip(tap)
        if(total_fare>0):
            self.context.update_last_paid(tap)
        return total_fare
    def process_tap_long(self,tap_data:List[tuple])->List[tuple]:
        results=[]
        for date,time,line,station in tap_data:
            tap=Tap.from_string(date,time,line,station)
            fare=self.calculate_fare(tap)
            results.append((date,time,line,fare,station))
        return results
def test_citylink_system():
    test_data=[
        ("07-01","7:20","G","BD"),
        ("07-01","8:01","G","NC"),
        ("07-01","8:30","R","YH"),
        ("07-01","8:32","Y","YH"),
        ("07-01","10:01","R","KL"),
        ("07-01","10:28","Y","NC"),
        ("07-01","10:32","Y","JT"),
        ("07-01","14:36","G","NC"),
        ("07-01","22:15","Y","BD"),
        ("07-01","23:58","G","NC"),
        ("07-02","00:45","X","NC"),
        ("07-02","01:10","G","BD"),
        ("07-02","04:50","G","BD"),
        ("07-02","13:05","Y","JT"),
        ("07-02","13:15","G","KL"),
        ("07-02","13:36","G","JT"),
        ("07-02","18:02","Y","BD"),
        ("07-02","18:18","Y","NC"),
        ("07-02","20:01","G","KL"),
        ("07-02","20:15","R","YT"),
        ("07-02","22:02","Y","KL"),
        ("07-02","23:15","G","BD"),
        ("07-03","00:20","R","NC")
    ]


expected_fares=[25,37.5,0,37.5,25,0,25,25,20,20,16.25,0,25,25,0,25,37.5,0,25,0,20
,20,16.25]
    engine=TariffEngine()
    results=engine.process_tap_long(test_data)

    print("Date  Time  Line  Calc  Expected  Match")
    print("---------------------------------------")



    for i,(date,time,line,calc,station) in enumerate(results):
        print(f"{date}  {time}  {line}  {calc} vs  {expected_fares[i]} -> {'OK'
if abs(calc-expected_fares[i])<0.01 else 'FAIL'}")

if __name__=="__main__":
    test_citylink_system()
```



OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\prateek.y\Desktop\Prateek_CityLink_FareBox_Python> python citylinkcode.py
Date  Time  Line  Calc  Expected  Match
---------------------------------------
07-01  7:20  G  25.0 vs  25 -> OK
07-01  8:01  G  50.5 vs  37.5 -> FAIL
07-01  8:30  R  25.5 vs  0 -> FAIL
07-01  8:32  Y  25.5 vs  37.5 -> FAIL
07-01  10:01  R  25.0 vs  25 -> OK
07-01  10:28  Y  0.0 vs  0 -> OK
07-01  10:32  Y  25.0 vs  25 -> OK
07-01  14:36  G  25.0 vs  25 -> OK
07-01  22:15  Y  20.0 vs  20 -> OK
07-01  23:58  G  20.0 vs  20 -> OK
07-02  00:45  X  16.25 vs  16.25 -> OK
07-02  01:10  G  0.0 vs  0 -> OK
07-02  04:50  G  25.0 vs  25 -> OK
07-02  13:05  Y  25.0 vs  25 -> OK
07-02  13:15  G  0.0 vs  0 -> OK
07-02  13:36  G  25.0 vs  25 -> OK
07-02  18:02  Y  50.5 vs  37.5 -> FAIL
07-02  18:18  Y  25.5 vs  0 -> FAIL
07-02  20:01  G  25.0 vs  25 -> OK
07-02  20:15  R  0.0 vs  0 -> OK
07-02  22:02  Y  20.0 vs  20 -> OK
07-02  23:15  G  20.0 vs  20 -> OK
07-03  00:20  R  16.25 vs  16.25 -> OK
```