CSE 643: Assignment 3

Abhimanyu Gupta - 2019226

Brief about working:

The python program takes inputs as user interest and the courses they have already taken with the grade scored. Based on this input information the program predicts possible careers for the user.

Steps to run the program:

1. Run the program using the python interpreter.
2. Supply the interest areas and courses taken as input.
3. The program would exit with possible careers if any, based on the forward chaining.

Working Examples:

1)

```
D:\Codes\CSE643_AI\A3>python code.py
Welcome to Career Advisory System
Follow instructions to proceed
Available Interests for selection: Data Development Security Maths Cloud
Available Courses for selection: AI ML IP AP FCS NS DMG BDA DBMS CldC DSCD AC IBC
Enter grade as Numeric Value from 1 - 10
Enter interests separated by space:
Development Security
Enter courses each per line as "Course Grade" (ignore qoutes). To finish entering, press enter:
IP 8
AP 8
FCS 10
NS 6
AI 7

Good Career choices for you would be Cybersecurity Analyst, Software Developer
```

2)

```
D:\Codes\CSE643_AI\A3>python code.py
Welcome to Career Advisory System
Follow instructions to proceed
Available Interests for selection: Data Development Security Maths Cloud
Available Courses for selection: AI ML IP AP FCS NS DMG BDA DBMS CldC DSCD AC IBC
Enter grade as Numeric Value from 1 - 10
Enter interests separated by space:
Data
Enter courses each per line as "Course Grade" (ignore qoutes). To finish entering, press enter:
IBC 9
AC 10

Sorry, we can't recomment any career path for you with this knoweldge
```

Code Snippets:

```
from durable.lang import *
import json


available_interests = ['Data', 'Development', 'Security', 'Maths', 'Cloud']
available_courses = ['AI', 'ML', 'IP', 'AP', 'FCS', 'NS', 'DMG', 'BDA', 'DBMS', 'CldC', 'DSCD', 'AC', 'IBC']
recommended_careers = set()


def convert_to_dict(data) :
    return json.loads(str(data).replace("'", "\""))

def get_average_gpa(data_dict) :
    return sum([course['grade'] for course in data_dict]) / len(data_dict)

with ruleset('recommend') :
    @when_all(m.interests.anyItem(item == 'Data') & m.courses.anyItem((item.course == 'AI') | (item.course == 'ML') | (item.course == 'DMG') | (item.course == 'BDA')))
    def rule1(c):
        m = convert_to_dict(c.m)
        c1 = []
        c2 = []
        for course in m['courses'] :
            if course['course'] in ['AI', 'ML', 'DMG'] :
                c1.append({'course' : course['course'], 'grade' : course['grade']})
            elif course['course'] in ['DMG', 'BDA'] :
                c2.append({'course' : course['course'], 'grade' : course['grade']})
        c.assert_fact('stronglyadvise', {'interest' : 'Machine Learning Engineer', 'courses' : c1})
        c.assert_fact('advise', {'interest' : 'Machine Learning Engineer', 'average_gpa' : get_average_gpa(c1)})
        c.assert_fact('stronglyadvise', {'interest' : 'Data Scientist', 'courses' : c2})
        c.assert_fact('advise', {'interest' : 'Data Scientist', 'average_gpa' : get_average_gpa(c2)})
```

```
    @when_all(m.interests.anyItem(item == 'Development') & m.courses.anyItem((item.course == 'IP') | (item.course == 'IP')))
    def rule2(c):
        m = convert_to_dict(c.m)
        cc = []
        for course in m['courses'] :
            if course['course'] in ['IP', 'AP'] :
                cc.append({'course' : course['course'], 'grade' : course['grade']})
        c.assert_fact('stronglyadvise', {'interest' : 'Software Developer', 'courses' : cc})
        c.assert_fact('advise', {'interest' : 'Software Developer', 'average_gpa' : get_average_gpa(cc)})

    @when_all(m.interests.anyItem(item == 'Security') & m.courses.anyItem((item.course == 'FCS') | (item.course == 'NS') | (item.course == 'iBC') | (item.course == 'AC')))
    def rule3(c):
        m = convert_to_dict(c.m)
        c1 = []
        c2 = []
        for course in m['courses'] :
            if course['course'] in ['FCS', 'NS'] :
                c1.append({'course' : course['course'], 'grade' : course['grade']})
            elif course['course'] in ['IBC', 'AC'] :
                c2.append({'course' : course['course'], 'grade' : course['grade']})
        c.assert_fact('stronglyadvise', {'interest' : 'Cybersecurity Analyst', 'courses' : c1})
        c.assert_fact('advise', {'interest' : 'Cybersecurity Analyst', 'average_gpa' : get_average_gpa(c1)})
        c.assert_fact('stronglyadvise', {'interest' : 'BlockChain Engineer', 'courses' : c2})
        c.assert_fact('advise', {'interest' : 'BlockChain Engineer', 'average_gpa' : get_average_gpa(c2)})
```

```
    @when_all(m.interests.anyItem(item == 'Maths') & m.courses.anyItem((item.course == 'AI') | (item.course == 'ML') | (item.course == 'IBC') | (item.course == 'AC')))
    def rule4(c):
        m = convert_to_dict(c.m)
        c1 = []
        c2 = []
        for course in m['courses'] :
            if course['course'] in ['AI', 'ML'] :
                c1.append({'course' : course['course'], 'grade' : course['grade']})
            elif course['course'] in ['IBC', 'AC'] :
                c2.append({'course' : course['course'], 'grade' : course['grade']})
        c.assert_fact('stronglyadvise', {'interest' : 'Machine Learning Engineer', 'courses' : c1})
        c.assert_fact('advise', {'interest' : 'Machine Learning Engineer', 'average_gpa' : get_average_gpa(c1)})
        c.assert_fact('stronglyadvise', {'interest' : 'BlockChain Engineer', 'courses' : c2})
        c.assert_fact('advise', {'interest' : 'BlockChain Engineer', 'average_gpa' : get_average_gpa(c2)})

    @when_all(m.interests.anyItem(item == 'Cloud') & m.courses.anyItem((item.course == 'CldC') | (item.course == 'DBMS') | (item.course == 'DSCD')))
    def rule5(c):
        m = convert_to_dict(c.m)
        cc = []
        for course in m['courses'] :
            if course['course'] in ['CldC', 'DBMS', 'DSCD'] :
                cc.append({'course' : course['course'], 'grade' : course['grade']})
        c.assert_fact('stronglyadvise', {'interest' : 'DevOps Engineer', 'courses' : cc})
        c.assert_fact('advise', {'interest' : 'DevOps Engineer', 'average_gpa' : get_average_gpa(cc)})

    @when_all(+m.interests)
    def nothing(c):
        pass
```

```
with ruleset('stronglyadvise') :
    @when_all((m.interest == 'Machine Learning Engineer') & m.courses.allItems(item.grade >= 7))
    def ml_engineer(c) :
        global recommended_careers
        recommended_careers.add('Machine Learning Engineer')

    @when_all((m.interest == 'Data Scientist') & m.courses.allItems(item.grade >= 7))
    def data_scientist(c) :
        global recommended_careers
        recommended_careers.add('Data Scientist')

    @when_all((m.interest == 'Software Developer') & m.courses.allItems(item.grade >= 7))
    def sde(c) :
        global recommended_careers
        recommended_careers.add('Software Developer')

    @when_all((m.interest == 'Cybersecurity Analyst') & m.courses.allItems(item.grade >= 7))
    def cyberexpert(c) :
        global recommended_careers
        recommended_careers.add('Cybersecurity Analyst')

    @when_all((m.interest == 'BlockChain Engineer') & m.courses.allItems(item.grade >= 7))
    def blockchain(c) :
        global recommended_careers
        recommended_careers.add('BlockChain Engineer')

    @when_all((m.interest == 'DevOps Engineer') & m.courses.allItems(item.grade >= 7))
    def devops(c) :
        global recommended_careers
        recommended_careers.add('DevOps Engineer')
```

```
116
117         @when_all(m.interest == 'Machine Learning Engineer')
118         def not_ml_engineer(c) :
119             pass
120
121         @when_all(m.interest == 'Data Scientist')
122         def not_data_scientist(c) :
123             pass
124
125         @when_all(m.interest == 'Software Developer')
126         def not_sde(c) :
127             pass
128
129         @when_all(m.interest == 'Cybersecurity Analyst')
130         def not_cyberexpert(c) :
131             pass
132
133         @when_all(m.interest == 'BlockChain Engineer')
134         def not_blockchain(c) :
135             pass
136
137         @when_all(m.interest == 'DevOps Engineer')
138         def not_devops(c) :
139             pass
140
141     with ruleset('advise') :
142         @when_all((+m.interest) & (m.average_gpa >= 7.5))
143         def advise_career(c):
144             global recommended_careers
145             recommended_careers.add(c.m.interest)
146
147         @when_all(+m.interest)
148         def check(c) :
149             pass
```

```
def show_instructions():
    print("Welcome to Career Advisory System")
    print("Follow instructions to proceed")
    global available_interests, available_courses
    print("Available Interests for selection:", *available_interests)
    print("Available Courses for selection:", *available_courses)
    print("Enter grade as Numeric Value from 1 - 10")

def read_user_input() :
    global available_interests, available_courses

    interests = input("Enter interests separated by space:\n").split()
    for interest in interests :
        if interest not in available_interests :
            raise Exception("Invalid Interest")

    courses_list = []
    print("Enter courses each per line as \"Course Grade\" (ignore qoutes). To finish entering, press enter:")
    while True :
        course = input().split()
        if len(course) == 0 :
            break
        elif len(course) != 2 :
            raise Exception("Invalid Course Input Format")
        course_name = course[0]
        if course_name not in available_courses :
            raise Exception("Invalid Course")
        try :
            course_grade = int(course[1])
            if not(course_grade >= 1 and course_grade <= 10) :
                raise Exception("Invalid Course Grade")
        except ValueError:
            raise Exception("Invalid Course Grade")
        courses_list.append({'course' : course_name, 'grade' : course_grade})

    return interests, courses_list
```

```
if __name__ == '__main__' :
    show_instructions()
    interests, courses = read_user_input()
    # print(interests, courses)

    assert_fact('recommend', {'interests': interests, 'courses' : courses})
    # print(recommended_careers)

    if len(recommended_careers) == 0 :
        print("Sorry, we can't recomment any career path for you with this knoweldge")
    else :
        print("Good Career choices for you would be", ', '.join(list(dict.fromkeys(recommended_careers))))
```