CMPE 230 Homework #2

Halil Umut Özdemir 2016400168

In this project, problem is developing a Python program called bucourses.py that will crawl Bogazici University's OBIKAS registration pages and extract course offering information. To do that, I divide the problem into 3 parts:

1. Initialization and Taking Arguments

In this project, I use 3 Python libraries.

```
import re
import requests
import sys
```

In initiallization part, I create the department_short_names and department_link_names lists. I write contents of this 2 list by hand. The third list about departments is department_formal_names which stores the long names of department names. I create this list by changing some characters in strigns of department_link_names list. I replace '+' with ' ', '%26' with '&', '%3a' with ':' and '%2c' with ','.

```
department_short_names = ["ASIA","ASIA","ATA","AUTO","BM","BIS","CHE","CHEM","CE","COGS","CSE","CET","CMPE","INT","CEM"
                         ,"CCS","EQE","EC","EF","ED","CET","EE","ETM","ENV","ENVT","XMBA","FE","PA","FLED","GED","GPH","GUID","HIST"
                         ,"HUM","IE","INCT","MIR","MIR","INTT","INTT","LS","LING","AD","MIS","MATH","SCED","ME"
                         ,"MECA","BIO","PHIL","PE","PHYS","POLS","PRED","PSY","YADYOK","SCED","SPL","SOC","SWE","SWE","TRM","SCO","TRM"
                         ,"WTR","TR","TK","TKL","LL"]
department_link_names = ["ASIAN+STUDIES","ASIAN+STUDIES+WITH+THESIS","ATATURK+INSTITUTE+FOR+MODERN+TURKISH+HISTORY","AUTOMOTIVE+ENGINEERING","BIOMEDICAL+ENGINEERING","BUSINESS+INFORMATION+SYSTEMS"
                       ,"CHEMICAL+ENGINEERING","CHEMISTRY","CIVIL+ENGINEERING","COGNITIVE+SCIENCE","COMPUTATIONAL+SCIENCE+%26+ENGINEERING","COMPUTER+EDUCATION+%26+EDUCATIONAL+TECHNOLOGY"
                       ,"COMPUTER+ENGINEERING","CONFERENCE+INTERPRETING","CONSTRUCTION+ENGINEERING+AND+MANAGEMENT","CRITICAL+AND+CULTURAL+STUDIES","EARTHQUAKE+ENGINEERING","ECONOMICS"
                       ,"ECONOMICS+AND+FINANCE","EDUCATIONAL+SCIENCES","EDUCATIONAL+TECHNOLOGY","ELECTRICAL+%26+ELECTRONICS+ENGINEERING","ENGINEERING+AND+TECHNOLOGY+MANAGEMENT"
                       ,"ENVIRONMENTAL+SCIENCES","ENVIRONMENTAL+TECHNOLOGY","EXECUTIVE+MBA","FINANCIAL+ENGINEERING","FINE+ARTS","FOREIGN+LANGUAGE+EDUCATION","GEODESY","GEOPHYSICS"
                       ,"GUIDANCE+%26+PSYCHOLOGICAL+COUNSELING","HISTORY","HUMANITIES+COURSES+COORDINATOR","INDUSTRIAL+ENGINEERING","INTERNATIONAL+COMPETITION+AND+TRADE"
                       ,"INTERNATIONAL+RELATIONS%3aTURKEY%2cEUROPE+AND+THE+MIDDLE+EAST","INTERNATIONAL+RELATIONS%3aTURKEY%2cEUROPE+AND+THE+MIDDLE+EAST+WITH+THESIS","INTERNATIONAL+TRADE"
                       ,"INTERNATIONAL+TRADE+MANAGEMENT","LEARNING+SCIENCES","LINGUISTICS","MANAGEMENT","MANAGEMENT+INFORMATION+SYSTEMS","MATHEMATICS","MATHEMATICS+AND+SCIENCE+EDUCATION"
                       ,"MECHANICAL+ENGINEERING","MECHATRONICS+ENGINEERING","MOLECULAR+BIOLOGY+%26+GENETICS","PHILOSOPHY","PHYSICAL+EDUCATION","PHYSICS"
                       ,"POLITICAL+SCIENCE%26INTERNATIONAL+RELATIONS","PRIMARY+EDUCATION","PSYCHOLOGY","SCHOOL+OF+FOREIGN+LANGUAGES","SECONDARY+SCHOOL+SCIENCE+AND+MATHEMATICS+EDUCATION"
                       ,"SOCIAL+POLICY+WITH+THESIS","SOCIOLOGY","SOFTWARE+ENGINEERING","SOFTWARE+ENGINEERING+WITH+THESIS","SUSTAINABLE+TOURISM+MANAGEMENT","SYSTEMS+%26+CONTROL+ENGINEERING"
                       ,"TOURISM+ADMINISTRATION","TRANSLATION","TRANSLATION+AND+INTERPRETING+STUDIES","TURKISH+COURSES+COORDINATOR","TURKISH+LANGUAGE+%26+LITERATURE"
                       ,"WESTERN+LANGUAGES+%26+LITERATURES"]
# Replaces the + %26 %3a %2c with required characthers for all elements in department_link_names
department_formal_names = [names.replace("+"," ").replace("%26","&").replace("%3a",":").replace("%2c",",") for names in department_link_names ]
```

After that I sort this three lists paralelly based on the order in first list (department_short_names).

```
# Sorts alphabetically department_short_names, department_link_names, department_formal_names lists paralel accordsng to department_short_names
department_short_names,department_link_names,department_formal_names = list(zip(*sorted(zip(department_short_names, department_link_names,department_formal_names))))
```

At the end of this part I take start term and end term as arguments and create list of terms in this interval. terms_link list stores terms like '2015/2016-01' and terms list stores '2015-Fall' . After that I initialize the dictionary that strores all of the information.(department_info)

```
start_term = sys.argv[1]
end_term = sys.argv[2]
terms_link,terms = find_interval(start_term,end_term)
# Dictionary stores all information
department_info = {}
```

The function find_interval creates the list of terms in given interval.

```
# Takes start and end term as input and
# calculates the the lists of printable format(e.g. 2016-Fall) and lik format(e.g 2016/2017-01)
def find_interval(start_term,end_term):
    if start_term[start_term.index("-")+1:] in "Fall":
        start_tnum = 1
        start_year = int(start_term[:start_term.index("-")])
    elif start_term[start_term.index("-")+1:] in "Spring":
        start_tnum = 2
        start_year = int(start_term[:start_term.index("-")]) - 1
    else:
        start_tnum = 3
        start_year = int(start_term[:start_term.index("-")]) - 1
    if end_term[end_term.index("-")+1:] in "Fall":
        end_tnum = 1
        end_year = int(end_term[:end_term.index("-")])
    elif end_term[end_term.index("-")+1:] in "Spring":
        end_tnum = 2
        end_year = int(end_term[:end_term.index("-")]) - 1
    else:
        end_tnum = 3
        end_year = int(end_term[:end_term.index("-")]) - 1
    list1 = []
    list2 = []
    for year in range(start_year,end_year+1):
        while start_tnum <= 3:
            list1.append("{}/{}-{}".format(year,year+1,start_tnum))
            if start_tnum is 1:
                list2.append("{}-{}".format(year,"Fall"))
            elif start_tnum is 2:
                list2.append("{}-{}".format(year+1,"Spring"))
            else:
                list2.append("{}-{}".format(year+1,"Summer"))
            if year == end_year and start_tnum == end_tnum:
                return list1,list2
            start_tnum += 1
        start_tnum = 1
```

2. Downloading and Creating the department_info Dictionary

In this part, fristly I create the link string and download the html information of this link. While downloading all links some of links give some connection time out errors, for this type of links I put the download line in try catch block. Also with this while loop I try at most 10 times to download a link. If program can not download at 5th time, it ignores that link (Takes its base_html as null string). Also I set the connection timeout limit to 5 seconds to obtain a faster program.

```
link = 'https://registration.boun.edu.tr/scripts/sch.asp?donem='+term_link+'&kisaadi='+department_short_name+'&bolum='+department_link_name
count = 0
# Downloading the link part
# In some download program given some errors about waiting long for requests, for this type errors program tries to download a link at most 5 times
# Also program makes new try for same link to download after 5 seconds passed (timeout is 5 seconds)
# if it can not download it ignores that term of department
while count < 10:
    try:
        html = requests.get(link,timeout=5)
        break
    except:
        count += 1
        continue
if count is 10:
    base_html = ""
else:
    base_html = html.text
```

After download part, I find the course codes, course names, and names of instructors with using regular expressions. Regular expression rules of this 3 is the following:

Course Codes:

r"<td>(.*?)\.[0-9]{2} </td>"

Course Names:

r"Desc\.</td>[\r\n]+\s*<td>(.*?) </td>"

Names of Instructors:

r"<td>[0-9\.]* </td>[\r\n]+\t*<td>[0-9\.]* </td>[\r\n]+\t*<td>(.*?) </td>[\r\n]+\t*<td>[(TBA)MTW(Th)F(St)S]* "

I use findall function of regular expressions library. At the end, because of the Info columns which exists in recent years' tables I apply an additional rule to all instructor names to remove that part if it exists.

```
# Finds codes of courses in html file (e.g. CMPE230)
rule_codes = r"<td><font style='font-size:12px'>(.*?)\.[0-9]{2}</font> </td>"
course_codes = re.findall(rule_codes, base_html, re.M | re.I | re.S)
# Finds names of courses in html file (e.g. SYSTEMS PROGRAMMING)
rule_course_names = r"Desc\.</a></td>[\r\n]+\s*<td>(.*?) </td>"
course_names = re.findall(rule_course_names, base_html, re.M | re.I | re.S)
# Finds names of instructors in html file
rule_instructors = r"<td>[0-9\.]* </td>[\r\n]+\t*<td>[0-9\.]* </td>[\r\n]+\t*<td>(.*?) </td>[\r\n]+\t*<td>[(TBA)MTW(Th)F(St)S]* "
instructors = re.findall(rule_instructors, base_html, re.M | re.I | re.S)
# In tables of some years there are extra Info column so if there is this extra column this part removes it
info_link_rule = r"(<a.*>Info</a> </td>[(\r\n)\n]*\t*<td>)?(.*)"
instructors = list(map(lambda instructor: re.search(info_link_rule, instructor).group(2).strip(), instructors))
```

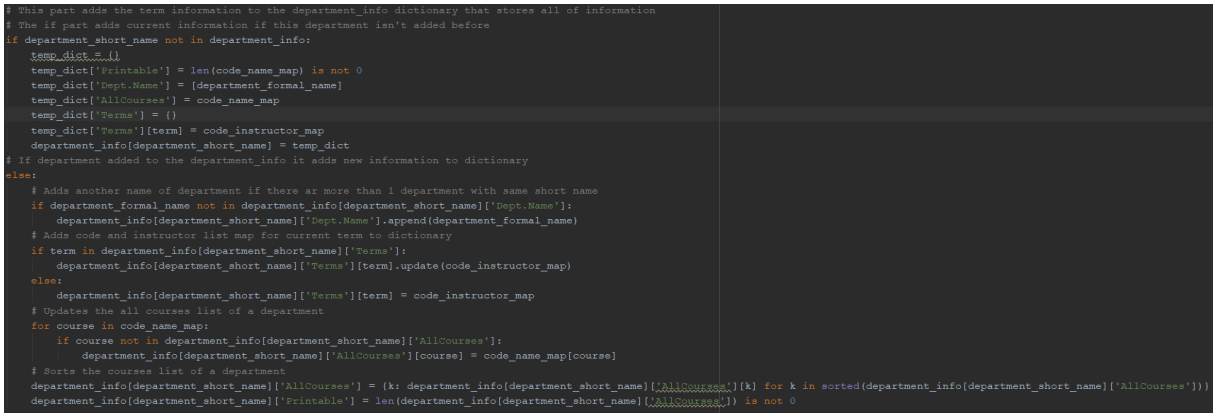
After the regular expression part, I cretate 2 maps. First is code_instructor_map, this map stores the list of instructors that gives this course at given term.

```
# This is a map of course codes and the list of instructors that gives that course of current term
code_instructor_map = {}
for [code,instructor] in [[code,instructor] for code,instructor in zip(course_codes,instructors)]:
    if code not in code_instructor_map:
        code_instructor_map[code] = [instructor]
    else:
        if(instructor not in code_instructor_map[code]):
            code_instructor_map[code].append(instructor)
# This is a map of course codes and course names that gives that course of current term
```

Second map is code_name_map, this map stores the course codes and name of that courses that are given in current term.

```
code_name_map = {}
for [code, name] in [[code, name] for code, name in zip(course_codes, course_names)]:
    if code not in code_name_map:
        code_name_map[code] = name
```

At the end, I use this 2 maps to update department_info dictionary. This dictionary has 4 keys for each department. These keys are ‘Printable’, ‘Dept.Name’, ‘AllCourses’ and ‘Terms’. ‘Printable’ key stores a boolean that represents this department is printable or not. If there isn’t any course of a department in given interval this department is not printable. ‘Dept.Name’ stores the list of long department names with same short name. ‘AllCourses’ stores the map of all course codes and their names in given interval. This is a dictionary, and contains all course code only 1 times. This dictionary is sorted after each update during the for loop to print courses in correct order. ‘Terms’ key stores a dictionary for each term in the given interval. For each term, dictionaries stores the code_instructor_map that was created before.



```
# This part adds the term information to the department_info dictionary that stores all of information
# The if part adds current information if this department isn't added before
if department_short_name not in department_info:
    temp_dict = {}
    temp_dict['Printable'] = len(code_name_map) is not 0
    temp_dict['Dept.Name'] = [department_formal_name]
    temp_dict['AllCourses'] = code_name_map
    temp_dict['Terms'] = {}
    temp_dict['Terms'][term] = code_instructor_map
    department_info[department_short_name] = temp_dict
# If department added to the department_info it adds new information to dictionary
else:
    # Adds another name of department if there ar more than 1 department with same short name
    if department_formal_name not in department_info[department_short_name]['Dept.Name']:
        department_info[department_short_name]['Dept.Name'].append(department_formal_name)
    # Adds code and instructor list map for current term to dictionary
    if term in department_info[department_short_name]['Terms']:
        department_info[department_short_name]['Terms'][term].update(code_instructor_map)
    else:
        department_info[department_short_name]['Terms'][term] = code_instructor_map
    # Updates the all courses list of a department
    for course in code_name_map:
        if course not in department_info[department_short_name]['AllCourses']:
            department_info[department_short_name]['AllCourses'][course] = code_name_map[course]
    # Sorts the courses list of a department
    department_info[department_short_name]['AllCourses'] = {k: department_info[department_short_name]['AllCourses'][k] for k in sorted(department_info[department_short_name]['AllCourses'])}
    department_info[department_short_name]['Printable'] = len(department_info[department_short_name]['AllCourses']) is not 0
```

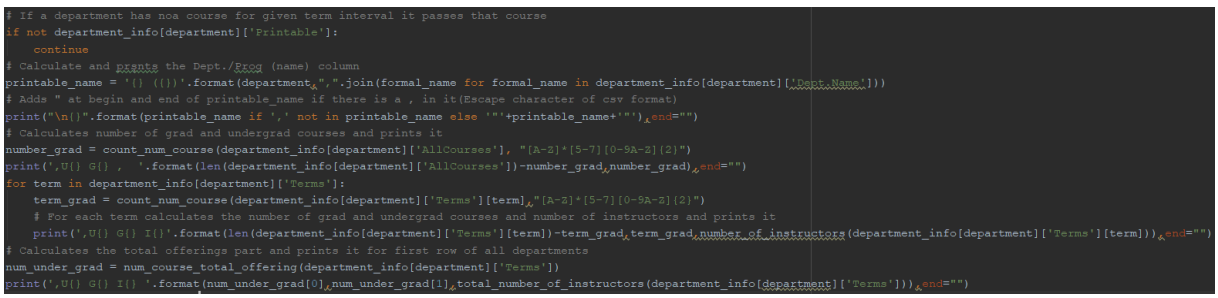
While I make this calculations, I assume that all courses with same course code have same course name.

3. Printing Output

After downloading part, this part prints the csv format with using the department_info dictionary. At the begining, I print the column names of csv table.

```
# Output Part
# Prints the columns of csv to stdout
print('Dept./Prog (name),Course Code,Course Name ',end="")
for t in terms:
    print(', {} '.format(t),end="")
print(',Total Offerings',end="")
```

After that for each department program calculates the numbers in the first row and prints it, if this department is printable. (Calculations will be explained later.)



```
# If a department has noa course for given term interval it passes that course
if not department_info[department]['Printable']:
    continue
# Calculate and prints the Dept./Prog (name) column
printable_name = '{} ({})'.format(department,",".join(formal_name for formal_name in department_info[department]['Dept.Name']))
# Adds " at begin and end of printable_name if there is a , in it(Escape character of csv format)
print("\n{}".format(printable_name if ',' not in printable_name else '"'+printable_name+'"'),end="")
# Calculates number of grad and undergrad courses and prints it
number_grad = count_num_course(department_info[department]['AllCourses'], "[A-Z]*[5-7][0-9A-Z]{2}")
print(',U{} G{} ,  '.format(len(department_info[department]['AllCourses'])-number_grad,number_grad),end="")
for term in department_info[department]['Terms']:
    term_grad = count_num_course(department_info[department]['Terms'][term],"[A-Z]*[5-7][0-9A-Z]{2}")
    # For each term calculates the number of grad and undergrad courses and number of instructors and prints it
    print(',U{} G{} I{}'.format(len(department_info[department]['Terms'][term])-term_grad,term_grad,number_of_instructors(department_info[department]['Terms'][term])),end="")
# Calculates the total offerings part and prints it for first row of all departments
num_under_grad = num_course_total_offering(department_info[department]['Terms'])
print(',U{} G{} I{} '.format(num_under_grad[0],num_under_grad[1],total_number_of_instructors(department_info[department]['Terms'])),end="")
```

At the end of the loop for each course of the current department, program prints the row of this course. To determine whether a column is marked or not, it searches the course code in the dictionaries stored in 'Term' key of department_info dictionaries. If it can find then marks this column. During this loop, count intefer counts the total offering for each course.

```
for course in department_info[department]['AllCourses']:
    temp_name = department_info[department]['AllCourses'][course]
    # Prints course codes and course name for each row
    print('\n ,{},{}'.format(course, temp_name if ',' not in temp_name else '"'+temp_name+'"'),end="")
    count = 0
    # Prints x if this course exists in current term if not prints only a space character
    # Also counts how many times a course is opened for total offerings column
    for term in terms:
        if course in department_info[department]['Terms'][term]:
            print(',x',end="")
            count += 1
        else:
            print(', ',end="")
    # Calculates and prints total offering colmun's information
    print(',{}/{}'.format(count,num_instructors_of_a_course(department_info[department]["Terms"],course)),end="")
```

Functions Using To Make Calculations

- num_course_total_offering(dictionary):
  This fucntion is calculating the number of graduate and undergraduate courses that are given in one department for all terms in the given interval. This is for the numbers in the Total Offerings column. Takes the dictionary stored 'Terms' key of department_info dictionary as input.

```
# Takes a dictionary of course codes and the instructor list which gives that course for all terms
# and counts total number of graduate and undergraduate courses given totally
def num_course_total_offering(dictionary):
    grad = 0
    undergrad = 0
    for term in dictionary:
        temp_grad = count_num_course(dictionary[term], "[A-Z]*[5-7][0-9A-Z]{2}")
        temp_under = len(dictionary[term])-temp_grad
        grad += temp_grad
        undergrad += temp_under
    return undergrad, grad
```

- num_instructors_of_a_course(dictionary,course):
  This function is calculating the number of distinct instructors that gives a given course. Takes the dictionary stored 'Terms' key of department_info dictionary and the course code as input.

```
# Takes a dictionary of course codes and the instructor list which gives that course for all terms
# and calculates the number of distinct instructor that gives given course
def num_instructors_of_a_course(dictionary,course):
    list_ = []
    for term in dictionary:
        if course not in dictionary[term]:
            continue
        for ins in dictionary[term][course]:
            if ins not in list_:
                list_.append(ins)
    return(len(list_))
```

- total_number_of_instructors(dictionary):
  This function calculates the total number of distinct instructors for a department in given term interval. Takes the dictionary stored ‘Terms’ key of department_info dictionary as input.

```
# Takes a dictionary of course codes and the instructor list which gives that course for all terms
# and calculates the total number of instructors
def total_number_of_instructors(dictionary):
    list_ = []
    for term_list in dictionary.values():
        for ins_list in term_list.values():
            for ins in ins_list:
                if ins not in list_:
                    list_.append(ins)
    return(len(list_))
```

- number_of_instructors(dictionary):
  This function calculates the number of distinct instructors of a department in one term. Takes the dictionary of a term stored ‘Terms’ key of department_info dictionary as input. Counts for only the term that the dictiononary of which is given as input.

```
# Takes a dictionary of course codes and the instructor list which gives that course for one term
# and calculates the total number of instructors of that term
def number_of_instructors(dictionary):
    list_ = []
    for ins_list in dictionary.values():
        for ins in ins_list:
            if ins not in list_:
                list_.append(ins)
    return(len(list_))
```

- count_num_course(dictionary,rule):
  This function counts the number of courses in a dictionary that obey the given regular expression rule. Takes a dictionary stored in ‘AllCourses’ key of department_info or a dictionary of a term stored ‘Terms’ key of department_info dictionary and the regular expreesion rule as a string as input. I use this funtion for counting the number of graduate courses in given interval or a term in given interval.

```
# Takes a dictionary of course codes and the instructor list which gives that course for one term
# and calculates number of courses that satisfies given regex rule
def count_num_course(dictionary,rule):
    count = 0;
    for code in dictionary:
        if(re.search(rule,code) is not None):
            count += 1
    return count
```