**Exercise 1:**

**XML File:**

**import** **xml.etree.ElementTree** **as** **ET**

fname = input("Enter file name: ")

**try**:

tree = ET.parse(fname)

**except**:

print(f'File cannot be opened: **{fname}**')

exit()

root = tree.getroot()

print(f'Total Events: {len(root)}**\n**')

count = 0

print('Entry data Attributes:')

**for** event **in** root.findall("./viewentry/entrydata"): *#/\*[@name='EventName']/text"):*

**if** count == 61: **break**

print(event.attrib)

count += 1

print('**\n\n**')

count = 0

**for** event **in** root.findall(".viewentry"):

**if** count == 5: **break**

print(f'Event {count + 1}')

print('Name: '+ event.find(".//\*[@name='EventName']/text").text)

print('Area: ' + event.find(".//\*[@name='Area']/text").text)

print('Categories: ' + ', '.join(cat.text **for** cat **in** event.findall(".//\*[@name='CategoryList']//textlist//text")))

print('Date Begin Show: ' + event.find(".//\*[@name='DateBeginShow']/text").text)

print('Admission: ' + event.find(".//\*[@name='Admission']/text").text + '**\n**')

count += 1

**#JSON file:**

**import** **json**

**import** **re**

**def** dict\_keys(data,father, level):

result = []

**for** key **in** data.keys():

**if** type(data[key]) != dict **and** type(data[key]) != list:

result.append('**\t**'\*level + key + '**\n**')

**else**:

**if** type(data[key]) == dict:

result.append('**\t**'\*level + key + ':**\n**' + '**\t**'\*(level) + '{**\n**')

level += 1

result += dict\_keys(data[key], key, level)

level -= 1

**else**: *#keys with list*

**if** len(data[key]) == 0:

result.append('**\t**'\*level + key + '**\n**')

**else**:

result.append('**\t**'\*level + key + ':**\n**' + '**\t**'\*(level) + '{**\n**')

level += 1

**for** i **in** range(len(data[key])):

result += dict\_keys(data[key][i], key, level)

level -= 1

**if** level != 0:

result.append('**\t**'\*(level-1) + '}**\n**')

**return** result

fname = input('Enter file name: ')

**try**:

file\_data = open(fname, encoding='utf8')

**except**:

print(f'File cannot be opened: **{fname}**')

exit()

events = json.load(file\_data)

print(f'Total Events: {len(events)}**\n**')

print('Data structure:')

print (''.join(key **for** key **in** dict\_keys(events[0],'', 0)))

print('**\n**')

print('Last 5 events:')

**for** event **in** events[:len(events)-6:-1]:

print(f"Name: **{event['calEvent']['eventName']}**")

print(f"Location: **{event['calEvent']['locations'][0]['locationName']}**")

print(f"Categories: **{event['calEvent']['categoryString']}**")

print("Start date: ", re.findall('\d\d\d\d-\d\d-\d\d', event['calEvent']['startDate'])[0])

print(f"Free: **{event['calEvent']['freeEvent']}\n**")

**Exercise 2:**

# Referenced from Eduardo’s war card program

**import** **random**

**class** **card**:

**def** \_\_init\_\_(self, suit, value):

self.suit = suit

self.value = value

**class** **deckOfCards**:

deck = []

**def** \_\_init\_\_(self):

suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']

values = ['A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K']

**for** s **in** suits:

**for** v **in** values:

self.deck.append(card(s, v))

**def** shuffle(self):

self.deck = random.sample(self.deck, len(self.deck))

**def** deal(self):

p1 = self.deck.pop(0)

p2 = self.deck.pop(0)

**return** p1, p2

**def** print\_deck(doc):

**for** card **in** doc.deck: print(f'**{card.value}** **{card.suit}**')

num\_values = {'A': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7,\

'8': 8, '9': 9, '10': 10, 'J': 11, 'Q': 12, 'K': 13}

**while**(**True**):

input('Press any key to start...')

doc = deckOfCards()

doc.shuffle()

count\_player = 0

count\_opp = 0

**while**(len(doc.deck) > 0):

p1, p2 = doc.deal()

print(f'Your card: **{p1.value}** **{p1.suit}** **\t**Opponent**\'**s card: **{p2.value}** **{p2.suit}**')

**if** num\_values[p1.value] == num\_values[p2.value]:

print('WAR')

**elif** num\_values[p1.value] > num\_values[p2.value]:

print('You win the hand!')

count\_player += 1

**else**:

print('You lose the hand')

count\_opp += 1

**if**(count\_player > count\_opp):

print(f'**\n**YOU WIN: **{count\_player}** x **{count\_opp}**')

**elif**(count\_player < count\_opp):

print(f'**\n**YOU LOSE: **{count\_player}** x **{count\_opp}**')

**else**:

print(f'**\n**WAR: **{count\_player}**')

inValue = input('**\n**Press any key to continue or **\'**q**\'** to quit')

**if** inValue == 'q': **break**

**Exercise 4:**

**import** **sqlite3**

**try**:

conn = sqlite3.connect('testdata.db')

conn.execute("PRAGMA foreign\_keys = ON")

**except** Error **as** e:

print(e)

exit()

cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS COURSE)

cur.execute('CREATE TABLE COURSE(**\**

course\_id integer PRIMARY KEY,**\**

course\_name text NOT NULL)')

cur.execute('DROP TABLE IF EXISTS STUDENT)

cur.execute('CREATE TABLE STUDENT(**\**

student\_id integer PRIMARY KEY,**\**

Student\_name text NOT NULL,**\**

course1 integer,**\**

course2 integer,**\**

course3 integer,**\**

course4 integer,**\**

course5 integer,**\**

course6 integer,**\**

FOREIGN KEY (course1)**\**

REFERENCES course(course\_id) ON DELETE SET NULL ON UPDATE CASCADE**\**

FOREIGN KEY (course2)**\**

REFERENCES course(course\_id) ON DELETE SET NULL ON UPDATE CASCADE**\**

FOREIGN KEY (course3)**\**

REFERENCES course(course\_id) ON DELETE SET NULL ON UPDATE CASCADE**\**

FOREIGN KEY (course4)**\**

REFERENCES course(course\_id) ON DELETE SET NULL ON UPDATE CASCADE**\**

FOREIGN KEY (course5)**\**

REFERENCES courses(course\_id) ON DELETE SET NULL ON UPDATE CASCADE**\**

FOREIGN KEY (course6)**\**

REFERENCES course(course\_id) ON DELETE SET NULL ON UPDATE CASCADE)')

cur.execute('INSERT INTO courses (course\_id, course\_name) VALUES(?, ?)', (1, 'Data Programming'))

cur.execute('INSERT INTO courses (course\_id, course\_name) VALUES(?, ?)', (2, 'Data Manipulation Techniques'))

cur.execute('INSERT INTO courses (course\_id, course\_name) VALUES(?, ?)', (3, 'Data Systems Architecture'))

cur.execute('INSERT INTO courses (course\_id, course\_name) VALUES(?, ?)', (4, 'Business Process'))

cur.execute('INSERT INTO courses (course\_id, course\_nam) VALUES(?, ?)', (5, 'Math for Data Analytics'))

cur.execute('INSERT INTO courses (course\_id, course\_name) VALUES(?, ?)', (6, 'Information Encoding Standards'))

cur.execute('INSERT INTO student (student\_id, student\_name, course1, course2)**\**

VALUES (?, ?, ?, ?)', (1, 'Nuthan', 1, 2))

cur.execute('INSERT INTO student (student\_id, student\_name, course\_id, course2)**\**

VALUES (?, ?, ?, ?)', (2, 'Reyhan', 3, 4))

cur.execute('INSERT INTO student (student\_id, student\_name, course\_id)**\**

VALUES (?, ?, ?)', (3, 'Camilo', 5))

conn.commit()

cur.execute('SELECT course.name, student.student\_id, student.student\_name**\**

FROM student**\**

INNER JOIN course on course.course\_id = student.course1 OR**\**

course.course\_id = student.course2 OR**\**

course.course\_id = student.course3 OR**\**

course.course\_id = student.course4 OR**\**

course.course\_id = student.course5 OR**\**

course.course\_id = student.course6**\**

ORDER BY course.course\_name')

rows = cur.fetchall()

cur.close()

**for** row **in** rows:

print(row)