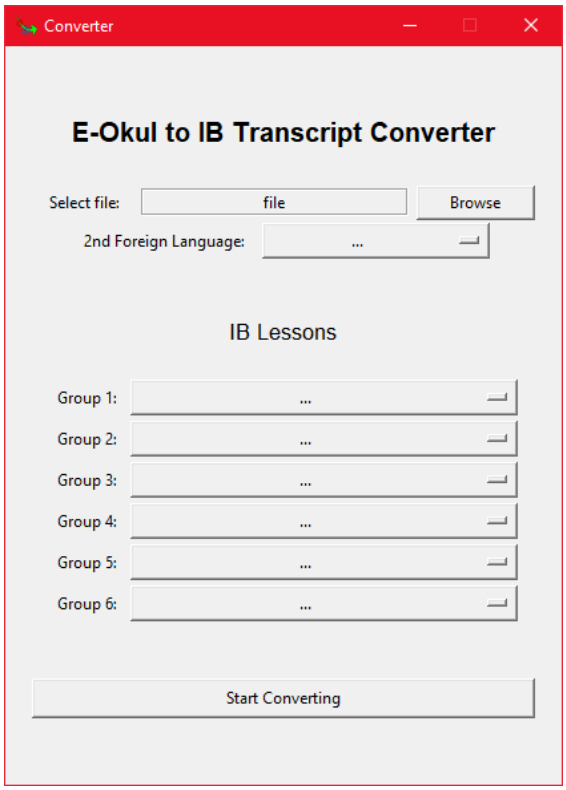
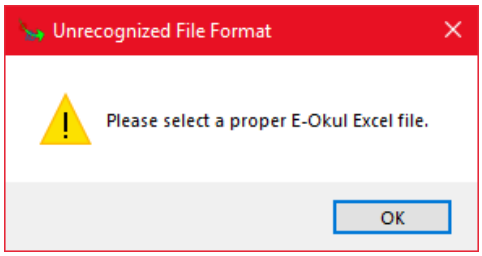
Criterion C: Development

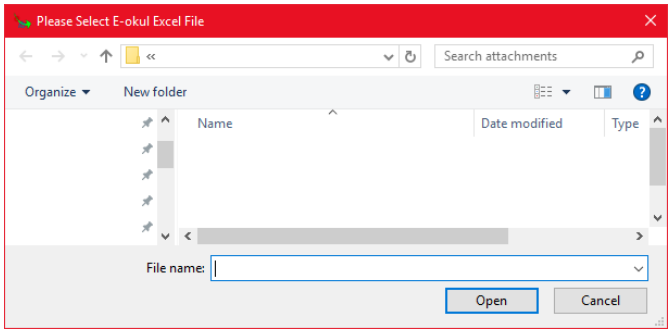
E-okul to IB Converter is a Python based program that helps students convert their e-okul formatted transcripts to the IB format. The program lets the user select a xls transcript file and their IB lessons to create a new IB formatted docx file. The user can then save the file to a location. In various stages of the execution if something isn't expected or an error has occurred, the program gives a helpful error.



Main Window

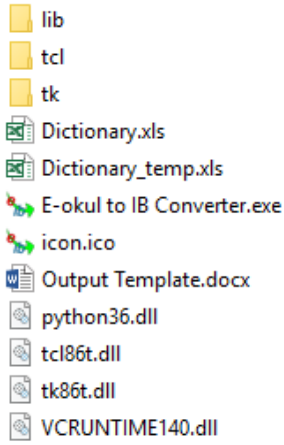


Error pop-up



File select dialog

Final Folder Structure:



The file structure needs to stay the same for the program to work correctly.

General Structure of the Python source code:

Importing Libraries

File names and other constants

File Set Up and Integrity Check related methods

Various helper methods for file handling and GUI

GUI Initialization

Set up basic output file and the dictionary for the main conversion operation

The main conversion operation

Importing Libraries:

for importing libraries

```
import datetime
import threading;

from tkinter.filedialog import *
from tkinter.messagebox import *

import xlrd;
from docx import Document
from xlrd import open_workbook
from xlutils.copy import copy
```

Note about all of the code snippets: The code snippets may not show complete or exact code. I shortened and edited some parts to make it easier to explain.

- **Tkinter:** This is a library for creating various GUI elements. I used text, input fields, buttons, drop down selections, and frames. More info on this in GUI Initialization section.
- **xlrd, xlutils, docx:** These three libraries are, in order, to read xls files, to write to xls files, and to read/write docx files. I use methods from these libraries in encapsulated methods in my main algorithm to read write to the dictionary file, student's transcript, and the resulting file.
- **Datetime:** I use this to convert a date from one format to another.
- **Threading:** The main conversion takes more than a few seconds, so threading was necessary to make the UI not freeze during this time.

File names and other constants:

```
# ------------------------------File Names
dictionary = "Dictionary.xls";
dictionaryTemp = "Dictionary_temp.xls"
outputTemplate = "Output Template.docx";
output = "Output.docx";


# ------------------------------Dictionary Constants
dictionary_normal_x = 3;
dictionary_normal_y = 1 + 8;

endOffTheLineValue = "__end_of_the_line__";
endOffTheColumnValue = "__switch_columns__";
```

File Set Up and Integrity Check related methods:

The program checks for wrong files and IO errors. If the code fails at any part of the process (antivirus block etc.) it calls the “FileReadorWriteError” method to handle the error. This method alerts the user of the problem and tries the operation again to not lose any data when the problem is solved by the user.

An overview of the IO methods:

```
def OpenBaseDocuments ():
def SetupInput ():
def SelectFile ():
def FixInput ():

def FileReadorWriteError (callback):
```

Explanations:

```
def OpenBaseDocuments ():
    global wb_dic, dic, wb_dic_write, dic_write, doc, tb_names, tb_IB, tb;
    try:
        wb_dic = open_workbook (dictionary);
        dic = wb_dic.sheets ()[0]  ## possible error throw
        wb_dic_write = copy (wb_dic);
        dic_write = wb_dic_write.get_sheet (0)  ## possible error throw

        doc = Document (outputTemplate);  ## possible error throw
        tb_names = doc.tables[0]
        tb_IB = doc.tables[1]
        tb = doc.tables[2]
    except:
        return FileReadorWriteError (OpenBaseDocuments);
```

This methods opens the main files and assigns various variables needed by the program at start.

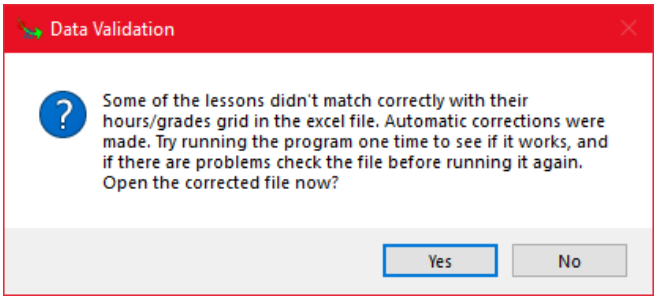
```
def SelectFile ():
    global input;
    global isInputProper;
    input = askopenfilename (initialdir = "myPath", title = "Please Select E-okul
Excel File");
    if(not(SetupInput ())):
        return;
    try:
        # if there isn't this specific text in this specific cell,
        # this is the wrong file. Show an error
        if (ws.cell_value (inputXlsXOffset, 0) != "ÖĞRENCİNİN"):
            showwarning (title = "Wrong E-Okul file", message = "Please select the
correct E-Okul Excel file. You probably got the 'data only' Excel file. You need
the other one.")

            inputDisp.set (input_def);
            isInputProper = False;
            return;
    except:  # if we can't read the file show an error
        showwarning (title = "Unrecognized File Format", message = "Please select a
proper E-Okul Excel file.");
        inputDisp.set (input_def);
        isInputProper = False;
        return;

    # if this slot has a number less than 10 then this slot is an "hours" slot.
    # This means that the lesson grades are shifted, which needs fixing
    if (float (ws.cell_value (inputXlsXGradeSearchBegin+2, 8)) < 10):
        FixInput ();

    # print(input);
    inputDisp.set (input[input.rindex ("/") + 1:]);

    print ("The input is proper!")
    isInputProper = True;
```

This is the method called when pressing the “browse” button to select an input transcript.

The actual dialog is handled by the OS. After the file is selected the program checks a specific cell to see if this is the correct file. Incorrect file results in an error message directing the user to get the correct file.

Even if this is the correct file, e-okul has a tendency to not line up the grades correctly. If this error is present the “FixInput” method tries to fix it and pop-up a window tells the user some shifting has occurred and should be checked. Normally there shouldn’t be any errors made by the program but e-okul system is unpredictable.

If everything went correctly we display the file’s name in the UI and set our input proper flag to true so that the further processes can run.



Pop-up from the FixInput method

```
# this is a critical error, we try to do it again and exit if canceled
def FileReadorWriteError (callback):
    if (askretrycancel ("I/O Error", "There was an error while reading or
writing files. Please try closing any open files, checking your antivirus,
running with administrator rights, or reinstalling the program to fix this
issue.")):
        return callback ();
    else:
        root.destroy ();
        sys.exit ();
        return False;
```

Lots of things can go wrong with I/O. If there is an error the program alerts the user and shows possible fixes. If the user doesn't want to/can't fix it now we exit the program, else we try to run the method that had issues with read/write (callback method) again to hopefully execute correctly this time.

Various helper methods for file handling:

Most of the methods in this section are just here as clean wrappers to the three different API's. As such I won't go into detail about each method.

An overview of the methods:

```
def FindRowInXLS (item):
def FindRowInDOCX (item):
def WriteInDOCX (text, x, y):
def RemoveDOCXRow (table, row):
def WriteInDOCX_IB (text, x, y):
def WriteInDOCX_Names (text, x, y):
def WriteInDictionary (text, x, y):
```

Explanations:

```
def FindRowInXLS (item):  # finds the item in our input.xls
    rowX = inputXlsXGradeSearchBegin;
    isFound = False;

    try:
        while not (isFound):
            if (ws.cell_value (rowX, 0) == item):
                isFound = True;
            else:
                rowX += 1;
    except:
        return -1;

    return rowX;


# writes text into the given row and column without editing text formatting
def WriteInDOCX (text, x, y):
    try:
        paragraph = tb.cell (x, y).paragraphs[0]
        paragraph.runs[0].text = str (text);
    except:
        print ("-" * 50 + "ILLEGAL WRITE REQUEST --> " + str (text));
```

Rest of the methods are very similar to these two.

GUI Initialization:

For the UI of the application I used the Tkinter library. This library makes use of “frames” to give a layout to the various UI elements. After creating frames you can “pack” elements into it in various different options. I mostly used “grid packing” to make my UI. Because my UI was simple and I didn’t need many options I listed options in a simple list.

```
root = Tk ()
root.title ("Converter")
root.resizable (width = False, height = False)
root.iconbitmap ('icon.ico')


mainframe = Frame (root)
mainframe.config (width = 500);
mainframe.grid (column = 0, row = 0, sticky = (N, W, E, S))
mainframe.columnconfigure (0, weight = 1)
mainframe.rowconfigure (0, weight = 1)
mainframe.pack (pady = 50, padx = 20)

Label (mainframe, text = "E-Okul to IB Transcript Converter", font =
("TkDefaultFont", 15, "bold")).grid (row = 0, column = 0,sticky = (EW), ipadx =
20);

Label (mainframe).grid (row = 1, column = 0, sticky = (EW), ipady = 2); # spacer
```

Basic set up and some labels.

```
input = "input.xls";
inputDisp = StringVar (root);
inputDisp.set (input_def);

fileFrame = Frame (mainframe);
fileFrame.grid (row = 2, column = 0);
Label (fileFrame, text = "Select file: ").grid (row = 0, column = 0, ipadx = 10);
Label (fileFrame, textvariable = inputDisp, relief = GROOVE, width = 25).grid (row
= 0, column = 1, ipadx = 10);
Label (fileFrame, text = "").grid (row = 0, column = 2);  # spacer
Button (fileFrame, text = "Browse", command = SelectFile).grid (row = 0, column =
3, ipadx = 20);
```

You can use StringVars in Tkinter to have dynamic text in UI. For this case, the input file’s name is dynamically changed after the user selects a file.

```
firstSpace = False;  # first space designates a new lesson
endOfTheLine = False;  # second space designates end of the lessons
while not (endOfTheLine):  # repeat until there are no more lessons
    read_name = "";
    firstSpace = False;
    thisLessonOptions = []
    while not (firstSpace):
        in_x += 1;
        read_name = dic.cell_value (dictionary_normal_x + in_x,
                                    dictionary_normal_y + y_track + 1);
        if(read_name == ""):
            continue;
        if (read_name == split or read_name == dynamicFill):
            firstSpace = True;
        else:
            thisLessonOptions.append (read_name);

    if (len (thisLessonOptions) > 0):
        print (thisLessonOptions)
        ibLessons.append (
            InitializeSelectionWithSettings ("Group " +
                                             str (i + 1) + ":",
                                             ibFrame,
                                             thisLessonOptions,
                                             i,
                                             40));
        i += 1;

    if (dic.cell_value (
            dictionary_normal_x + in_x + 1,
            dictionary_normal_y + y_track)
            == endOffTheLineValue):
        endOfTheLine = True;
        print ("IB Lesson Options Read Complete")
        print ();
        break;
```

This section of code adds the possible IB lessons drop down menus.

In the dictionary.xls file we have a column dedicated to holding all the IB lessons from group 1 to 6. They are placed as such there is a “dynamic fill” (__dynamic_fill_slot__) or a “split” (__) character between each group.

This bit of code reads this column from top to bottom and adds each non special or empty line to an array. When it finds a “dynamicFill” or a “split” character it breaks the small loop and adds a drop down menu option to the UI using “InitializSelectionWithSettings(array)” method.

When it reaches the “end of the line” (__end_of_the_line__) it finishes the loop.

IB
TURKISH A LITERATURE SL
TURKISH A LITERATURE HL
__dynamic_fill_slot__
__
ENGLISH A LANGUAGE AND LITERATURE SL
ENGLISH A LANGUAGE AND LITERATURE HL
__dynamic_fill_slot__
__
…
…
…
__end_of_the_line__

Related Dictionary Section

Set up basic output file and the dictionary for the main conversion operation:

In addition to converting lesson formats to one another, the application also needs to write the students name and other information to the output file. Also the dictionary file needs to be updated with the information filled in by the user.

```
def WriteStudentInfo ():
    WriteInDOCX_Names (ws.cell_value (inputXlsXOffset + 1, 3), 0, 1);
    WriteInDOCX_Names (ws.cell_value (inputXlsXOffset + 2, 3), 0, 3);
    WriteInDOCX_Names (ws.cell_value (inputXlsXOffset + 3, 3), 1, 1);
    WriteInDOCX_Names (ws.cell_value (inputXlsXOffset + 4, 3), 2, 1);
    WriteInDOCX_Names (ws.cell_value (inputXlsXOffset + 5, 3), 1, 5);
    WriteInDOCX_Names (ws.cell_value (inputXlsXOffset + 2, 10), 0, 5);
    WriteInDOCX_Names (ConvertDate (datetime.datetime
(*xlrd.xldate_as_tuple (ws.cell_value (inputXlsXOffset + 3, 10),
wb.datemode)).date ().isoformat ()), 1, 3);
    WriteInDOCX_Names ("Male" if ws.cell_value (inputXlsXOffset + 4, 10) ==
"Erkek" else "Female", 2, 3);
    gradevalue = "-MANUAL ENTRY-";
    myVal = ws.cell_value (inputXlsXOffset + 5, 10);
    if myVal == "AL - 11. Sınıf / A Şubesi":
        gradevalue = "11th Grade /A Branch"
    elif myVal == "AL - 11. Sınıf / B Şubesi":
        gradevalue = "11th Grade /B Branch"
    elif myVal == "AL - 12. Sınıf / A Şubesi":
        gradevalue = "12th Grade /A Branch"
    elif myVal == "AL - 12. Sınıf / B Şubesi":
        gradevalue = "12th Grade /B Branch"
    WriteInDOCX_Names (gradevalue, 2, 5);
```

Writing the student info.

```
def SetUpDictionary ():
    global dic;
    if (not isInputProper):
        showerror ("Error", "Please select a proper input file!")
        print ("input not set!")
        return;
    for stvar in ibLessons:
        if (stvar.get () == defaultOption):
            showerror ("Error", "Please select all the lessons!")
            print ("not all vars selected!")
            return;
    button.config (state = DISABLED, textvariable = buttonText,
disabledforeground = "GREY")  # configure our button to be a progress bar

    # change our dictionary file according to the settings choosen
    WriteInDictionary (foreignLang.get (), dictionary_normal_x,
dictionary_normal_y + 1);

    ##set every ib lesson according to users selection both in our
dictionary and word file
    in_x = -1;
    n = -1;
    for lesson in ibLessons:
        in_x += 1;
        n += 1;

        WriteInDOCX_IB (lesson.get (), n + 2, 0);

        while (True):
            if (dic.cell_value (dictionary_normal_x + in_x,
dictionary_normal_y + dictionary_columnChangeOffset + 1) == dynamicFill):

                WriteInDictionary (lesson.get (), dictionary_normal_x +
in_x, dictionary_normal_y + dictionary_columnChangeOffset + 1);
                print ("written value " + lesson.get ())


    try:
        wb_dic_write.save (dictionaryTemp);
        wb_dic = open_workbook (dictionaryTemp);
        dic = wb_dic.sheets ()[0]  ## possible error throw
    except:
        return FileReadorWriteError (SetUpDictionary);

    # setting up dictionary file complete
    print ();
    print ("-*-*-*-*-*-*-*- Dictionary Set-up Complete! -*-*-*-*-*-*-*-");
    print ();

    ConvertLessons ();
```

This section checks user input and fills out the dictionary file accordingly.

The main conversion operation:

The main conversion happen in three stages:

- Read and write grades loop
    - Read grade from the input file
    - Write converted grade to the output file
- Cleanup empty lessons
- Calculate average grades

Read and write grades loop:

```
while not (firstSpace):  # ------READ GRADES FROM XLS FILE
    in_x += 1;
    try:
        read_name = dic.cell_value (dictionary_normal_x + in_x,
                                    dictionary_normal_y + y_track);
    except:
        print ("read failure" + str (dictionary_normal_x + in_x) +
               " - " + str (dictionary_normal_y + y_track));

    if (read_name == ""):
        continue;
    if (read_name == split):
        firstSpace = True;
    else:
        print (read_name + " *");
        read_rowX = FindRowInXLS (read_name);
        # check if the row actually exists
        if (read_rowX != -1):
            # check if there is an actual value for this lesson -
            # if not pass this lesson.
            try:
                t_hours += int (float (
                    ws.cell_value (read_rowX, read_y)));
                t_grade += int (float (
                    ws.cell_value (read_rowX, read_y))) * float (
                    ws.cell_value (read_rowX, read_y + 1));
            except Exception as e:
                print ("Error: " + str (
                    ws.cell_value (read_rowX, read_y)) + " - " +
                       str (ws.cell_value (read_rowX, read_y + 1)));
                print (e)
                pass;
```

This section of the code reads values from the input.xls file. Because each lesson can have more than one name, the dictionary file has multiple lines for some lessons. The code finds all the names in the dictionary file in the input.xls file and adds the corresponding hours and the grades into variables. This is done for each alternative name until the code finds a “split” character, then it writes all the hours and grades added into the output file in the next bit of code.

```
write_name = dic.cell_value (
    dictionary_normal_x + in_x - 1, dictionary_normal_y + 1);

if (t_hours > 0):  # no need to write lessons that have no class time

    write_rowX = FindRowInDOCX (write_name);
    WriteInDOCX (t_hours, write_rowX, i * 2 + 1);
    WriteInDOCX ('%.2f' % (t_grade / t_hours), write_rowX, i * 2 + 2);
```

The code finds the corresponding lesson write name in the output file and writes the total values there.

Cleanup empty lessons:

```
while (tb.cell (x, 0).text != "TOTAL NUMBER OF WEEKLY CLASS HOURS"):
    skipRow = False;
    x += 1;

    for y in range (1, 8):
        myText = tb.cell (x, y).text
        if (myText != "0" and myText != ""):
            skipRow = True;
            continue;

    if(not skipRow):
        RemoveDOCXRow (tb, tb.rows[x]);
        x -= 1;
```

The code has to remove empty lessons because the template docx file includes all possible lessons, but each student take only some of the lessons. The lessons the student didn't take have 0 hours for each slot. So if row has only zero and empty cells then the code removes that row.

Calculate Averages:

```
# add up grades and hours of all the lesson until avg slot
while (tb.cell (x, g * 2 + 1).text != averageGrade):
    if (tb.cell (x, g * 2 + 2).text != ""):
        t_hours += int (tb.cell (x, g * 2 + 1).text);
        t_grade += int (tb.cell (x, g * 2 + 1).text) * float (tb.cell (x, g
* 2 + 2).text);
    x += 1;

if (t_hours > 0):  # write found values to the file
    WriteInDOCX (t_hours, x, g * 2 + 1);
    WriteInDOCX ('%.2f' % (t_grade / t_hours), x + 1, g * 2 + 2);
else:
    print ("ERROR - there isnt any grades for this grade: " + str (9 + g));
```

This just adds up the values in a column and writes it down. There can be "errors" in this part but this error is for debugging only, the users will never see this error.

```
if (TrySave ()):
    showinfo (title = "Complete", message = "Conversion Complete. Please
check the file, especially the total class hours.")

    to2 = threading.Thread (target = OpenFinalFile)
    to2.start ()

else:
    showwarning (title = "Error", message = "Conversion Failed")
```

When the conversion is completed the file is saved then immediately opened so the user can check it.

A full source code is in the product folder.

Word count: 1143