Indian Institute of Information Technology, Nagpur



Attendance Record and Management

ITW-1

Computer Science & Engineering

3rd Semester

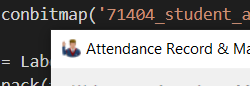
|  |  |
| --- | --- |
| Prepared By- | |
| RAJ ARYAN | BT19CSE043 |
| PRATEEK SHENDE | BT19CSE051 |
| SARTHAK GUPTA | BT19CSE054 |
| HEMANSHU CHAUDHARI | BT19CSE056 |
| KRISH RUSTAGI | BT19CSE089 |

**ACKNOWLEDGEMENT**

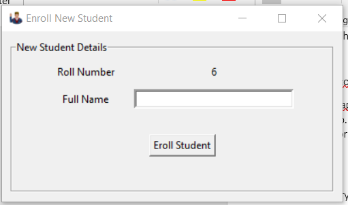
**Index**

|  |  |  |
| --- | --- | --- |
| Serial No. | Description | Page No. |
| 1 | Building the Graphical User Interface (GUI) |  |
| 2 | Creating Dataframe by Reading Excel |  |
| 3 | Display the Dataframe Record on GUI |  |
| 4 | Calculating the Percentage from Dataframe |  |
| 5 | Updating the Dataframe |  |
| 6 | Saving onto the Excel File |  |
| 7 | Modifying Records |  |
| 8 | References |  |

**BUILDING THE GRAPHICAL INTERFACE**

**Bitmapicon**

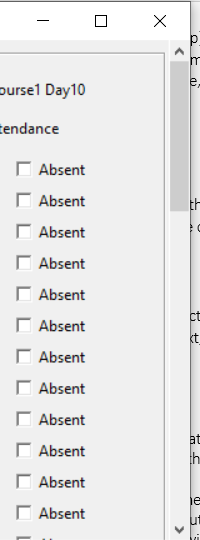
Iconbitmap(bitmap) sets the icon of the window/frame widget to bitmap. The bitmap must be an ico type, but not png or jpg type, otherwise, the image will not display as the icon.

**Geometry**

The geometry method is a fundamental one which decides the size, position and some other attributes of the screen layout we are going to create.

**Canvas**

The Canvas is a rectangular area intended for drawing pictures or other complex layouts. You can place graphics, text, widgets or frames on a Canvas.

**Scrollbar**

Canvas is not updated automatically when its content is modified, so we need to define it and update it manually using the scrollregion argument:

The Scrollbar frame works differently than other Tkniter widgets.

The values (0,0) tell the canvas on which position to draw the window. The argument anchor= ”nw” tells the canvas to place the frame's top left corner on position (0,0.

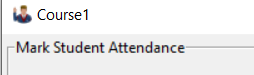
Finally, we have to configure the canvas so that when its y-position changes, the scrollbar moves:

That means that we'll be able to set the Canvas size to whatever we want, and then when we scroll, we'll be moving along the window inside it.

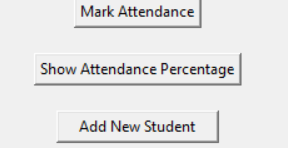
In order to do so, we must also specify what the scrollable area will be. Often, the scrollable area matches the contents of the inner window—but it can also be different if we wish.

**Frame.Bind**

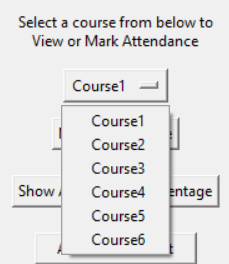
This is about binding events to functions and methods so that when the event occurs that specific function is executed.

**Frame Label**

A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts. This widget has the features of a frame plus the ability to display a label.

**Buttons**

The Button widget is used to add buttons in a Python application. These buttons can display text that convey the purpose of the buttons. You can attach a function or a method to a button which is called automatically when you click the button.

**Dropdown**

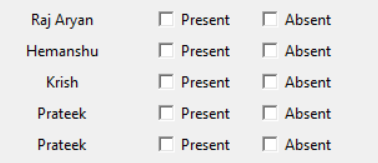
Dropdowns are toggleable, contextual overlays for displaying lists of links and more.

The value selected from the options provide in the dropdown menu can be access from the variable assigned to it by using the method, variable.get().

**Input Box**

The entry widget is used to enter text strings. This widget allows the user to enter one line of text, in a single font. The value entered in the input box can be access from the variable assigned to it by using the method, variable.get().

**Checkbox**

****The Checkbutton widget is used to display a number of options to a user as toggle buttons. The user can then select one or more options by clicking the button corresponding to each option.

The value selected from the options provide in the dropdown menu can be access from the variable assigned to it by using the method, variable.get().

**Withdraw**

Removes the window from the screen, without destroying it.

**Deiconify**

Displays the window, after using either the iconify or the withdraw methods.

**READING FROM AN EXCEL RECORD**

**What is DataFrame**

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:

* Dict of 1D ndarrays, lists, dicts, or Series
* 2-D numpy.ndarray
* Structured or record ndarray
* A Series
* Another DataFrame

In the project we are working on dict of Series objects which is present in an excel file named ‘Output.xlsx’.

**Loading Excel File as Dataframe**

We first need to import Pandas and load excel file, and then parse excel file sheets as a Pandas dataframe. To read the excel file named ‘output.xlsx’, Pandas’ ExcelFile can be used. ExcelFile is equivalent to read\_excel(Excel\_File). pd.Excelfile returns DataFrame or dict of DataFrames.

Sheet\_names\_list = df.sheet\_names

After loading the excel file into df, we are storing the sheet names into a list for further usage.

Sheet\_names\_list = df.sheet\_names

**Dataframe.Parse**

Dataframe .parse is used to parse the specified sheets in the dataframe. Since all the sheets names are present in the Sheet\_names\_list, we can make use of that individually parse through each sheet and store the sheet’s data into dataframes.

**List of Dataframes**

Dataframes are generic data objects which are used to store the tabular data. They are two-dimensional, heterogeneous data structures. A list, however, comprises of elements, vectors, data frames, variables, or lists that may belong to different data types.

We are using a list of dataframes so as to read all the different sheets of excel file separately, as each of the 6 sheets present in the excel file represent 6 different courses. So, we need to maintain 6 different dataframes which will intern represent 6 different courses. These six different dataframes can be represented as six different list objects.

dflist = []

for sheet in Sheet\_names\_list :

    dflist.append(df.parse(sheet\_name=sheet,index\_col=None))

**Significance of Index\_Col = None**

Column(s) to use as the row labels of the DataFrame, either given as string name or column index. If a sequence of int / str is given, a MultiIndex is used. index\_col=None/False can be used to force pandas to not use the first column as the index, e.g. when you have a malformed file with delimiters at the end of each line. Since we want the first column of each excel sheet to be empty, we are using such labels.

**DISPLAYING THE DATA FROM DATAFRAME TO UI**

* **Dynamically Selecting Which List Item to Show**

According to the user’s selection of Course from the dropdown menu, we can access which course is selected by using the variable.get() and based on that, we can decide which dataframe from the list is to be displayed onto the UI.

if clicked.get()=='Course1':

        x=0

elif clicked.get()=='Course2':

        x=1

* **Looking for Empty Columns to Take Input for That Day**

After the selection of course and list item, we will look for a column in that dataframe which is completely empty, i.e., the day for which the attendance has not yet been taken.

    for i in range(1,41):

        if dflist[x]['Day'+str(i)].isnull().sum()==len(dflist[x].index):

            DayN =i

            break

* **Creating Checkboxes**

Now, according to the number of students in the dataframe, we will be creating two checkboxes (Present & Absent) corresponding to each student.

    for i in range(len(dflist[x].index)):

        name = Label(frame,text=dflist[x]['Name'][i]).grid(row=i+3,column=0)

        c = Checkbutton(frame,text="Present",variable=Present,onvalue=1,offvalue=0,command = lambda idx = i: markpresent(idx,x))

        d = Checkbutton(frame,text="Absent",variable=Absent,onvalue=0,offvalue=1,command = lambda idx = i: markabsent(idx,x))

        c.grid(row=i+3,column=1)

        d.grid(row=i+3,column=2)

* **Different Rows**

As there are multiple students, different students along with their corresponding Present and Absent button are to be displayed in different rows. So the griding will be based on the rows.

name = Label(frame,text=dflist[x]['Name'][i]).grid(row=i+3,column=0)

c.grid(row=i+3,column=1)        d.grid(row=i+3,column=2)

Calculating %

-selecting the list item based on the selection of course

-reading number of presents in a single row

-reading total classes attended

-calculating

- displaying the output with name of student

-

Updating dataframe list

-taking input from the checkbox

-present variable

-absent variable

-markpresent()

-markabsent()

**Saving the record**

The *to\_excel()* method is used to export the DataFrame to the excel file.  To write a single object to the excel file, we have to specify the target file name. To save the changes onto the excel file, Excelwriter can be used.

**ExcelWriter**

Excel writer is a python pandas’ module for writing DataFrame objects into excel sheets. It can be used to write text, numbers, and formulas to multiple worksheets.

Writing to excel:

**with** ExcelWriter('path\_to\_file.xlsx') **as** writer:

df.to\_excel(writer)

Multiple sheets may be written to by specifying unique *sheet\_name*. With all data written to the file it is necessary to save the changes.

**with** ExcelWriter('path\_to\_file.xlsx') **as** writer:

df1.to\_excel(writer, sheet\_name='Sheet1')

df2.to\_excel(writer, sheet\_name='Sheet2')

**Save()**

The updated data need to be retained in database for further use.  It is necessary to save the changes for all the data written to the file.

with pd.ExcelWriter('output.xlsx') as writer:

        for i in range(6):

            dflist[i].to\_excel(writer,sheet\_name="Sheet\_name\_" + str(i+1),index=False)

In above code, the loop is executing 6 times so that it appends the new record in the dataframe of all the 6 courses.

**Why All List Object Being Written to The File?**

Creating an *ExcelWriter* object with a file name that already exists will result in the contents of the existing file being erased. So we need to write all the dataframes on the file once again, otherwise if we only write a single sheet, all other sheets will get erased.

**MODIFYING THE RECORD**

* **Add Student**

To add a student in the record a specific button is given in the main window. On clicking the button, a new frame is opened which asks user to give full name(f\_name) of the student and a button to update is given which updates the data and opens main window again.

**Taking Input from The Input Box**

Taking the name of student by user in entry widget and adding the student to the last of our database i.e. a new roll no is given to the student in last of our student records.

RollNo = Label(frame,text = len(dflist[0].index)+2).grid(row=0,column=1)

f\_name = Entry(frame, width=30,borderwidth=3)

The name of the student entered into the input box can be accessed through the variable assigned to the Entry Widget.

f\_name.get()

**Updating Dataframe**

To update the data in all the 6 sheets or 6 courses, add the student’s name and roll no in all 6 sheets by get attribute of entry widget and the specific roll no given

for i in range(6):

df2 = {'RollNo':len(dflist[0].index)+2,'Name': f\_name.get()}

dflist[i]=dflist[i].append(df2,ignore\_index = True)

In above code, the loop is executing 6 times so that it appends the new record in the dataframe of all the 6 courses.

The save function is called to updated the excel file according to the modified dataframes.

* **Delete Student**

To delete a student in the record a specific button is given in the main window. On clicking the button, a new frame is opened which asks user to give full name(f\_name) and roll no of the student and a button to update is given which updates the data and opens main window again.

**Taking Input Roll Number**

Taking the roll no of student by user in entry widget.

R\_no = Entry(frame, width=30,borderwidth=3)

**Taking Name Input**

Taking the roll no of student by user in entry widget.

f\_name = Entry(frame, width=30,borderwidth=3)

**Updating the Record and Deleting the Row**

To delete the record of student, the existing roll number and name of the student whose record is to be deleted is taken from the user and we will recreate the dataframe in which we will write all the records of the earlier dataframe except the record to be deleted.

name = f\_name.get()

rno  = R\_no.get()

for i in range(6):

dflist[i] = dflist[i][(dflist[i].Name != name) & (dflist[i].RollNo != rno)]

In above code, the loop is executing 6 times so that it recreates the dataframe of all the 6 courses.

The save function is called to updated the excel file according to the modified dataframes.

* **Edit Record**

To edit the record of a particular day a button is given in the main window, on clicking the button a new frame is opened which asks user to select course from drop down list and to give the day number of which the record is to be edited in form of input text (entry).

**Selecting the Course**

A drop-down menu is used for selecting course. The course selected can be accessed through the variable assigned to the dropdown Widget.

f\_name.get()

**Taking Day Number as Input**

The day No of the entered into the input box can be accessed through the variable assigned to the Entry Widget.

day.get()

After selection of course and day, clicking the button **MARK ATTENDANCE AGAIN,** it will open a new window where the user can either mark the attendance of all the student of that particular day again or can only make changes to particular student’s record leaving the rest unmodified.

The save function is called to updated the excel file according to the modified dataframes.

References