Indian Institute of Information Technology, Nagpur

# Attendance Record and Management

ITW-1

Computer Science & Engineering

3rd Semester

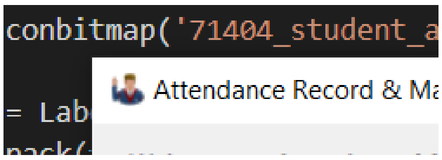
Prepared By-

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

# Index

# BUILDING THE GRAPHICAL INTERFACE

## Bitmapicon

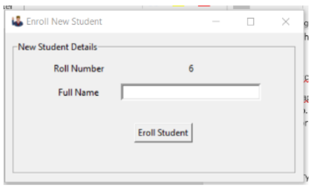


```
conbitmap('71404_student_a
= Lab
pack(
```

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.

```
root.iconbitmap('71404_student_attendance.ico')
```

## 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.

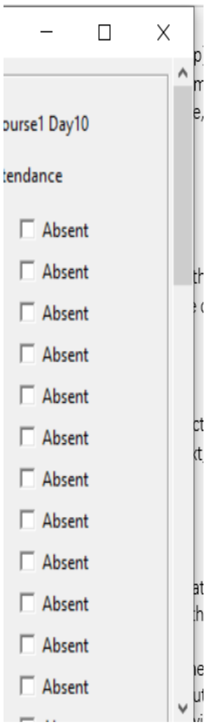
```
root.geometry("400x400")
```

## 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.

```
canvas = Canvas(record, borderwidth=0)
```

## 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.

```
vsb = Scrollbar(record, orient="vertical", command=canvas.yview)

canvas.configure(yscrollcommand=vsb.set, width=1200, height=80)

canvas.create_window((4,4), window=frame, anchor="nw", tags="frame")
```

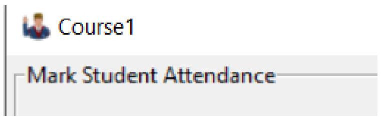
## Frame.Bind

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

```
    frame.bind("<Configure>", lambda event, canvas=canvas: OnFrameConfigure(
canvas))
```

## Frame Label

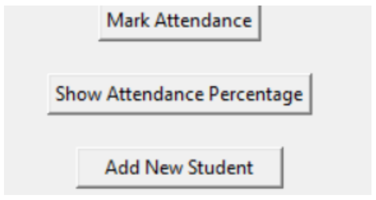
A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts. This widget has the features of a frame plus the ability to display a label.



```
frame = LabelFrame(canvas,text="Student Attendance Record",padx=50,pady=10)
```

## 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.

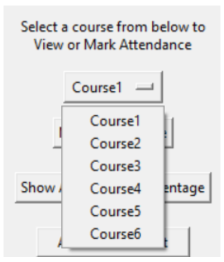


```
btn = Button(frame,text="Mark Attendance",command=open()).grid(row=2,colu
mn=2,pady=10)
```

## Dropdown

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

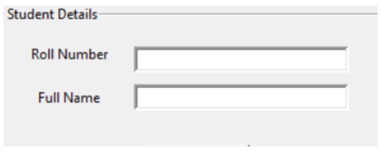
The value selected from the options provide in the dropdown menu can be access from the variable assigned to it by using the method, *variable.get()*.



```
drop = OptionMenu(frame, clicked, *options).grid(row=1,column=2,columnspan=2)
```

## Input Box

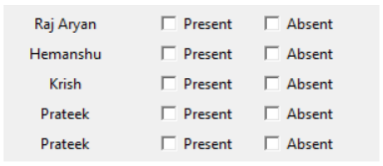
The entry widget is used to enter text strings. This widget allows the user to enter one line of text, in a single font. The value entered in the input box can be access from the variable assigned to it by using the method, *variable.get()*.



```
day = Entry(frame, width=30,borderwidth=3)
```

## 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.



| | | |
|---|---|---|
| Raj Aryan | ☐ Present | ☐ Absent |
| Hemanshu | ☐ Present | ☐ Absent |
| Krish | ☐ Present | ☐ Absent |
| Prateek | ☐ Present | ☐ Absent |
| Prateek | ☐ Present | ☐ Absent |

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()*.

```
c = Checkbutton(frame,text="Present",variable=Present,onvalue=1,
offvalue=0,command = (idx,x))
```

## Withdraw

Removes the window from the screen, without destroying it.

```
lambda:[classroom.withdraw(),root.deiconify()]
```

## Deiconify

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

```
lambda:[classroom.withdraw(),root.deiconify()]
```

# 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

First, we 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 dataframe.

## 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 six sheets present in the excel file represent six different courses. So, we need to maintain different dataframes which will intern represent 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 GUI

### 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 GUI.

```
if clicked.get()=='Course1':
    x=0
elif clicked.get()=='Course2':
    x=1 (Similarly for other courses)
```

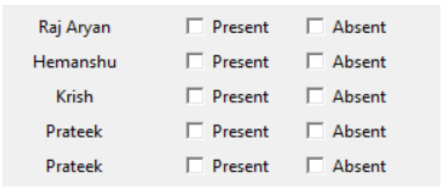
### Looking for Empty Columns to Take Input for That Day

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 in adjacent columns.



```
Raj Aryan    ☐ Present    ☐ Absent
Hemanshu     ☐ Present    ☐ Absent
Krish        ☐ Present    ☐ Absent
Prateek      ☐ Present    ☐ Absent
Prateek      ☐ Present    ☐ Absent
```

```
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)).grid(row=i+3,column=1)
  d = Checkbutton(frame,text="Absent",variable=Absent,onvalue=0,offvalue=1,
command = lambda idx = i: markabsent(idx,x)).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 in the dataframe.

```
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 ATTENDANCE PERCENTAGE

### Selecting the List Item Based on The Selection Of Course

According to the user's selection of Course from the dropdown menu, we can access which list item is to read to calculate percentage of the records by using the *variable.get()*.

```
if clicked.get()=='Course1':
    x=0
elif clicked.get()=='Course2':
    x=1 (Similarly for other courses)
```

### Reading Number of Presents in A Single Row

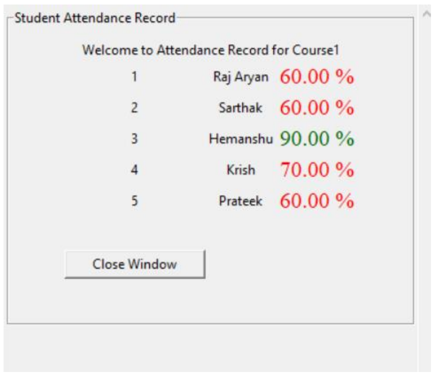
The data in the selected Dataframe is traversed row by row and in each row, we calculate the sum of the total number *"Present"* in that row and store it in variable *attended*.

```
attended = (dflist[x].iloc[i,2:]=="Present").sum()
```

### Reading Total Classes Attended

The data in the selected Dataframe is traversed row by row and in each row, we calculate the sum of the total number of *Null values (NAN)* in that row and subtract it from the total number of lectures (40) store it in variable *total*.

```
total = 40 - (dflist[x].iloc[i,2:].isnull()).sum()
```



```
Student Attendance Record
        Welcome to Attendance Record for Course1
        1        Raj Aryan   60.00 %
        2        Sarthak     60.00 %
        3        Hemanshu    90.00 %
        4        Krish       70.00 %
        5        Prateek     60.00 %

   Close Window
```

### Calculating Attendance Percentage

The attendance percentage is simply calculated by

```
((attended/total)*100)
```

### Displaying the Output with Name of Student

If the attendance percentage is less than the 75 then the student is ineligible for the examination and the percent is shown in Red else in the Dark Green Font along with the name and roll number of that student.

```
rollno = Label(frame,text=dflist[x]['RollNo'][i]).grid(row=i+1,column=0)
name = Label(frame,text=dflist[x]['Name'][i]).grid(row=i+1,column=1)
    if (attended/total)*100 < 75:
        percent = Label(frame,text = "{:.2f}".format((attended/total)*100) +" %", fg =
"red",font = "Times").grid(row=i+1,column=2)
    else:
        percent = Label(frame,text = "{:.2f}".format((attended/total)*100) + " %", fg =
"dark green",font = "Times").grid(row=i+1,column=2)
```

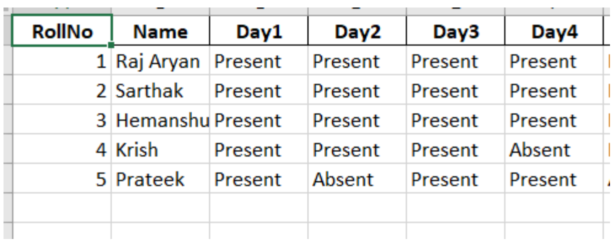
# UPDATING DATAFRAME LIST

## Taking Input from the Checkbox

### Checkbox Variable (Present & Absent List)

These lists are used to keep track of the presentees and absentees. They are linked to the checkboxes such that whenever a checkbox is checked/unchecked the corresponding *onvalue(1)/offvalue(0)* gets appended in the lists. These lists are then further accessed by *markPresent()* and *markAbsent()* functions through *idx* and then can be further used to update the data frame.

### Mark Present



| RollNo | Name | Day1 | Day2 | Day3 | Day4 |
|---|---|---|---|---|---|
| 1 | Raj Aryan | Present | Present | Present | Present |
| 2 | Sarthak | Present | Present | Present | Present |
| 3 | Hemanshu | Present | Present | Present | Present |
| 4 | Krish | Present | Present | Present | Absent |
| 5 | Prateek | Present | Absent | Present | Present |

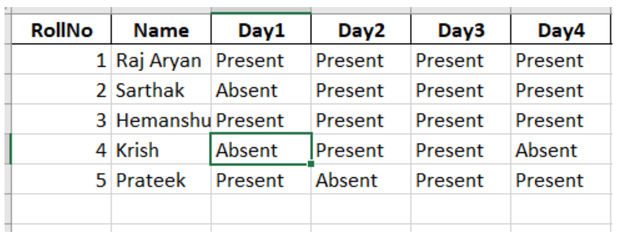
The function is attached to the *Present* checkbox button which is called automatically when the checkbox is clicked.

It takes two arguments *x* and *idx*. *x* determines the course (dataframe) of which the attendance is being taken while *idx* determines the person who is being marked present in the checkboxes.

*idx* is basically the row number and *DayN* is the column. With the help of these, a particular cell is selected in the corresponding dataframe, and *Present* value is written onto the cell.

```
def markpresent(idx,x):
    global DayN
    dflist[x]['Day'+str(int(DayN))].iloc[idx] = Present[idx].cget("text")
```

### Mark Absent



| RollNo | Name | Day1 | Day2 | Day3 | Day4 |
|---|---|---|---|---|---|
| 1 | Raj Aryan | Present | Present | Present | Present |
| 2 | Sarthak | Absent | Present | Present | Present |
| 3 | Hemanshu | Present | Present | Present | Present |
| 4 | Krish | Absent | Present | Present | Absent |
| 5 | Prateek | Present | Absent | Present | Present |

The function is attached to the *Absent* checkbox button which is called automatically when the checkbox is clicked.

It takes two arguments *x* and *idx*. *x* determines the course (dataframe) in which the attendance is being taken while *idx* determines the person who is being marked Absent in the checkboxes.

*idx* is basically the row number and *DayN* is the column. With the help of these, a particular cell is selected in the corresponding dataframe, and *Absent* value is written onto the cell.

```
def markabsent(idx,x):
    global DayN
    dflist[x]['Day'+str(int(DayN))].iloc[idx] = Absent[idx].cget("text")
```

## 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.

```
def save():
    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 six times so that it writes all the six dataframes (representing six different courses) onto the excel file.

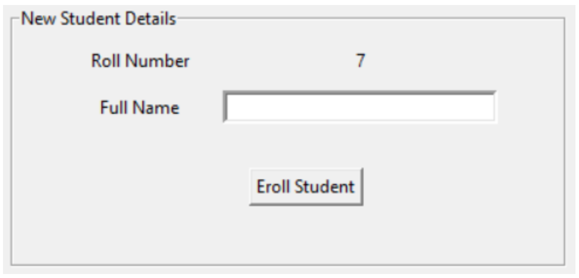
### Why All List Object Being Written to The File?

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, on clicked, updates the dataframes 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 number is given to the student in last of our student records.

```
RollNo = Label(frame,text = dflist[0].iloc[-1]['RollNo'] + 1).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

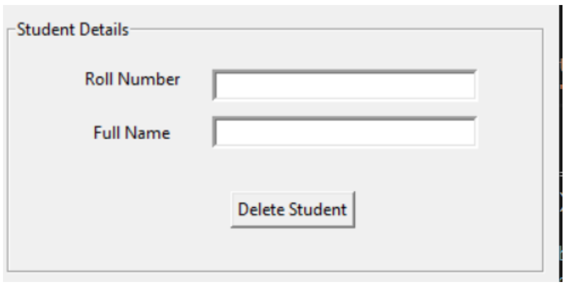
```
def updateAddData():
    for i in range(6):
        df2 = {'RollNo':dflist[0].iloc[-1]['RollNo'] + 1,'Name': f_name.get()}
        dflist[i]=dflist[i].append(df2,ignore_index = True)
    save()
```

In above code, the loop is executing six times so that it adds the new student record in the dataframe of all the six 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, on clicked, updates the dataframes and opens main window again.



### Taking Input Roll Number

Taking the roll number of student by user in entry widget.

```
R_no = Entry(frame, width=30,borderwidth=3)
```

### Taking Name Input

Taking the name of student by user in entry widget.

```
f_name = Entry(frame, width=30,borderwidth=3)
```

### Updating the Record and Deleting the Row

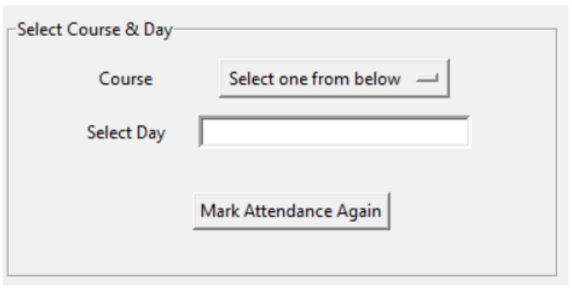
To delete the record of student, the existing roll number and name of the student whose record is to be deleted is taken from the user and we will recreate the dataframe in which we will write all the records of the earlier dataframe except the record to be deleted.

```
def updateDelData():
    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)]
    save()
```

In above code, the loop is executing six times so that it adds the new student record in the dataframe of all the six 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 Number 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 a particular student's record leaving the rest unmodified.

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

## CONCLUSION

The Attendance Record and Management System is developed by using Tkinter and Pandas. Tkinter is excellent for small, quick GUI applications, and since it runs on more platforms, it is a good choice where portability could be a prime concern. Through this project we got comfortable with tkinter and pandas also creating a basic yet useful utility for keeping track of attendence. Also creating a light GUI helped us understand how input/output works alongside a GUI. The system has reached a steady state where all the bugs have been eliminated. The system is operated at a high level of efficiency and has a very vast scope in future. With a proper software of database Space Manager, it can run the entire work in a much better, accurate and error free manner.

## REFERENCES

- **TutorialsPoint**
- **Geeksforgeeks**
- **Stackoverflow**
- **Effbot**
- **Open Menu Widget**
- **Pandas' Documentation**