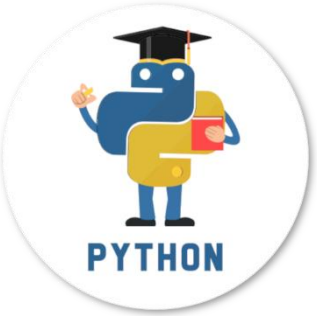
Quiz Application



PROJECT REPORT

INT 213 | Python Programming

K19UW

Group: 17

Developed by:

Arjun Rao Rachakonda	11903860	A33
Lalam Gangadhar	11912986	A12

Submitted To : Mrs. Neha Bagga

Introduction

- We have Developed an Quiz Application which is more interactive, bugs free and easy to use.

- Our Application is built using “Python’s tkinter” package.

- The tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit.

- Tk() provides a library of basic elements of GUI widgets for building a graphical user interface (GUI) in Our project.

- We have used many references inside widgets to make widgets look better, aligned and interactive with the user.

- We have applied many references to the “top level window” or “main window” in which widgets were placed.

Like,

1. We have titled main window as ‘PyQuiz’.
2. Setting the geometry of the main window as ‘700x600’.
3. We have set the background(bg) of the main window as white i.e; ‘#ffffff’ for good design when and logo or widget is placed on it.
4. We have set the state of the window to ‘zoomed’ to maintain proper alignment of the labels and widgets then the program executed.
5. We have set the resizable to (0,0) to put the main window always maximized.

- We have used many functions in program to execute a block of code whenever required using function call().

- We have used other python libraries like “random” to randomly pick any 8 questions among 10, after each time when executed**.**

Widgets

- We have used image widget “ImageTk.PhotoImage(Image.open('filename.type'))” and assigned it to a variable “logo” and then we have put the image in main window using another widget called

- “Label(main, image=logo, background = ‘#ffffff’)” and assigned it to a variable “labellogo” and then packed the labellogo widget with pady=(40,0) (where, pady=(40,0) → 40px space above the Image and 0px below the image) as “labellogo.pack(pady=(40,0))”.

NOTE:

1. To get the image widget we have used another library i.e; Python Image Library (PIL) and the image we use should be in same working directory other-wise we need to find the directory of the image.
2. The packer is Geometry manager used to specify the relative positioning of widgets within their container i.e; main window or a Frame.
3. The Widget “Label” has references like, (image = logo (variable of Image widget)), (background=’#ffffff’) i.e; white background.

- We have used Label widget to place the title of the application and put different references in it such as

  labeltext = Label(main,text='PyQuiz',font=('courgette',32,'bold'),bg='#ffffff')
  labeltext.pack(pady=(0,25))
  (where,pady=(0,25) → 0px space above the text and 25px below the text)

  then the widget is assigned to a variable “labeltext” then packed the variable.

- We have used Button widget assigned to "startbtn" variable and used varies references such as,

```
startbtn = Button(main,
        text='START',                → titled the button as 'START'
        font=('calibri',20,'bold'),  → setting title font = 'calibri' ,'bold' and font size = 20
        relief=FLAT,                 → setting relief is set to FLAT
        border=0,                    → setting border of Button to 0 (zero)
        bg='green',                  → Background set to "green"
        command=Clickstart)          → when Button is clicked it calls function Clickstart()
```

- Then the startbtn widget is packed with required pady i.e; "startbtn.pack(pady=(0,30))"

- We have used another Label widget to put some instructions in it with some references as per the requirement and assigned it to a variable "Instr" and packed as such:

-
```
Instr = Label(
        main,
        text='Instructions for the Quiz',
        width=152,
        font = ('Times',14,'bold'),
        bg='#000000',                → black(#000000)
        foreground='#FACA2F')        → yellow(#FACA2F)

    Instr.pack()
```

Functions

- In order to start quiz when a user clicks on the start button the widgets which we have created so far should be destroyed and question should be displayed so command is given to start button to call function Clickstart().

- The function Clickstart() destroys the all the widgets and further maximizes the screen such that
  1. no option to exit out of the quiz in between and
  2. no option to minimize the window with an attribute applied to a container.
  → main.attributes('-fullscreen',True)
  3. Further, call another functions startQuiz() and generate().

```
def Clickstart():
    labellogo.destroy()
    labeltext.destroy()
    Instr.destroy()
    startbtn.destroy()
    generate()
    main.attributes('-fullscreen',True)
    startQuiz()
```

- The function generate() is used to generate random indices for choosing random questions for the user.
  For, the above requirement we have created an empty list named indices as such “indices = [ ]”.
  Then declared indices variable as global because any change made to this List must be reflected through out the program in any other function.

- In generate(),

```
indices = []
def generate():
    global indices
    while(len(indices) < 8):
        x = random.randint(0,9)
        if x in indices:
            continue
        else:
            indices.append(x)
```

- while loop executes till the length of the “indices” list is less than (<) 8 since, we have to insert 8 indices to the list.

- To get and random index from (0-9) we have used x = random.randint(0,9) for this functionality we have “import random“ as discussed in introduction, during each iteration when visited “x” we get the random value for x.

- We need to append the randomly picked “x” to “indices” list during each iteration.

- But, there arises a problem that the same index many and get append to the list “indices” occur during iteration so, we have used a if statement i.e; if x in indices then the while loop get executed again with the **continue** keyword then we get the new value for ‘x’.

- If the value for ‘x’ is unique then else part is executed i.e;

```
        if x in indices:
            continue
        else:
            indices.append(x)
```

- Now, we have an randomly picked indices in "indices" list.

- Next, task is to create a list of 10 Questions as such:
  Questions = ['Q1','Q2', 'Q3', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10'] with index ranging from (0-9).

- Then, we have created another list of option corresponding to the given question as such:
  answer_option = [['1','2','3','4'],['1','2','3','4'] ,['1','2','3','4'] ,['1','2','3','4'] ,['1','2','3','4'] ,['1','2','3','4'] ,['1','2','3','4'] ,['1','2','3','4'] ,['1','2','3','4'] ,['1','2','3','4']]

- Now, Inside function startQuiz() (which was called previous called in Clickstart() function) we have to create a "Label" widget and assigned it to a variable "labelques" to display the question.
  As we know previously inside Clickstart() function all the widgets previously created was destroyed and then called another function startQuiz(). Now this function further created all the widgets required.

```
labelques = Label(main,                          → Inside main container.
        text =Questions[indices[0]],             → Initially when starts it must show
                                                   the Question from "Questions" list
                                                   at index (which we get from from
                                                   "indices" list at 0 index of indices
                                                   list).
        font=('calibri',18),                     → setting font='calibri' and font size=18
        width=500,                               → width of the label is 500px.
        justify='left',                          → justify = 'left' means text aligned left.
        bg='#ffffff',                            → background of the widget is white.
        wraplength=600,                          → length of widget is 600px.
    )
    labelques.pack(pady=(100,30))  → 100px space above the widget and 30px below.
```

- Now, we have to create RadioButtons widgets under the question before that we have to a use **Intvar()** (When a radiobutton is turned on by the user, its control variable is set to its current value option. If the control variable is an IntVar, given to each radiobutton in the group a different value options) then the Intvar() is assigned to variable "Rdvar".

- Then the Rdvar is set to '-1' since, Radio Button should not be marked before clicked and declared the Rdvar as global.

```
global Rdvar
Rdvar = IntVar()
Rdvar.set(-1)
R1 = Radiobutton(
    main,                                  → Container = main
    text=answer_option[indices[0]][0],     → Initially, it must show option for the corresponding Question at
                                             index (present at indices[0]) of list "indices" as
                                             discussed above for "labelques" widget from the "answer_option"
                                             list.so, answer_option[indices[0][0]] chooses the 1st option i.e; '1'
                                             of 1st question option list ~ ['1','2','3','4]. Similarly, in 2nd
                                             RadioButton R2 chooses 2nd option i.e;'2'of 1st question option list.
                                             For, that text = answer_option[indices[0]][1].

    font=('calibri',12),
    value=1,                               → value for 1st option is 1 and value for 2nd option is 2 and so on…
    variable=Rdvar,                        → As we discussed IntVar() i.e; Rdvar same control variable is given to
                                             all the Radio buttons options. When a particular radiobutton is
                                             turned click by the user, its control variable is set to its current
                                             value option. For e.g; If user click on 1nd  option then Rdvar = 1.
    bg='#ffffff',
    justify='center',
    command=choose,                        → when R1 – Radio button is click it calls another function choose()
)
R1.pack(pady=(0,10))                       → 0px space above R1 -- Radio button and 10px space below.
```

- Similarly, steps to be followed to create remaining RadioButtons – R2, R3 and R4 with their corresponding value = 2, 3 and 4 respectively.

- Now, When any Radio button is selected then we must store user responds in list of "User_answer" to compare it with the actual "answers" list and to display

the result then it must proceed to the next Question among the ”Questions” list which was chosen according to randomly shuffled “indices” list.

```
global Rdvar
Rdvar = IntVar()
Rdvar.set(-1)
R1 = Radiobutton(
    main,
    text=answer_option[indices[0]][0],
    font=('calibri',12),
    value=1,
    variable=Rdvar,
    bg='#ffffff',
    justify='center',
    command=choose,
)
R1.pack(pady=(0,10))

R2 = Radiobutton(
    main,
    text=answer_option[indices[0]][1],
    font=('calibri',12),
    value=2,
    variable=Rdvar,
    bg='#ffffff',
    justify='center',
    command=choose,
)
R2.pack(pady=(0,10))

R3 = Radiobutton(
    main,
    text=answer_option[indices[0]][2],
    font=('calibri',12),
    value=3,
    variable=Rdvar,
    bg='#ffffff',
    justify='center',
    command=choose,
)
R3.pack(pady=(0,10))

R4 = Radiobutton(
    main,
    text=answer_option[indices[0]][3],
    font=('calibri',12),
    value=4,
    variable=Rdvar,
    bg='#ffffff',
    justify='center',
    command=choose,
)
R4.pack()
```

- To proceed to the next question the index must be changed that means we need to get another index for another question so, we get next index from “indices" list as the index of “indices” list get incremented.

- All the tasks must be done when the user click on the any option So, command is given to radio buttons R1,R2,R3 and R4 as choose() performs all the tasks.

```
User_answer = []
```
→ As discussed we have created an empty list and which stores the responds of the user.

```
ques = 1
```
→ As we go inside the function we see that “ques” is an index given to indices list we need to assign ques = 1 since, we were done with the 1st question i.e; the index 0. we have to get the next question so, we have to get the next index from indices list so, we have started with ques=1 .

```
def choose():
    global Rdvar,User_answer
    global labelques, R1, R2, R3, R4
    global ques
```
→ we have set scope of all the variable global so, changes in the variables will be reflected throughout the program in any other function.

```
    x = Rdvar.get()
```
→ we need to get the user response with Rdvar.get() we get the user selected option and then given to ”x”.

```
    Rdvar.set(-1)
    User_answer.append(x)
```
→ Then x is append to the list “User_answer”

```
    if  ques < 8:
```
→ This proper should be continued till the 8th question so,(0-7) indices

```
        labelques.config(text = Questions[indices[ques]])
        R1['text'] = answer_option[indices[ques]][0]
        R2['text'] = answer_option[indices[ques]][1]
        R3['text'] = answer_option[indices[ques]][2]
        R4['text'] = answer_option[indices[ques]][3]
        ques += 1

    else:
        calculate()
```

➔ config is used to access an object's attributes after its initialization.

➔ So, labelques.config(text = Questions[indices[ques]]) it changes the text of the labelques widget and radio buttons R1, R2, R3 and R4.
R1[‘text’] acts as R1.config(text=)

- We need to get the next question and its corresponding options as soon as the user clicked on this desired option and the responses is saved.

- We uses config method to change the question, to change that

```
Ques = 1
if  ques < 8:

    labelques.config(text = Questions[indices[ques]])
    R1['text'] = answer_option[indices[ques]][0]
    R2['text'] = answer_option[indices[ques]][1]
    R3['text'] = answer_option[indices[ques]][2]
    R4['text'] = answer_option[indices[ques]][3]
    ques += 1
```

➔ 1st Click on any Radio button:

For, suppose the randomly shuffled indices list be: indices = [1,3,0,2,4,5,7,6]

At first when ques = 1 the indices[ques] going to get 3 so, indices[1] = 3

labelques.config(text = Questions[indices[1]]) → labelques.config(text = Questions[3])

similarly, for Radio buttons R1,R2,R3,R4 i.e;

R1['text'] = answer_option[indices[1]][0] → R1['text'] = answer_option[3][0]

R2['text'] = answer_option[indices[1]][1] → R2['text'] = answer_option[3][1]

R3['text'] = answer_option[indices[1]][2] → R3['text'] = answer_option[3][2]

R4['text'] = answer_option[indices[1]][3] → R4['text'] = answer_option[3][3]

- Through, this procedure we change the question with there corresponding options.
  That means we have changed the text of labelques widget and R1, R2, R3 and R4 radio button when the user click on the his desired radio button which invoke the function choose() and it change the above requirements.

  ques is incremented by 1 → ques = ques + 1

➔ 2nd Click on any Radio button:

Now, ques = 2 then indices[ques] going to get 0 so, indices[2] = 0

labelques.config(text = Questions[indices[2]]) → labelques.config(text = Questions[0])

similarly, for Radio buttons R1,R2,R3,R4 i.e;

R1['text'] = answer_option[indices[2]][0] → R1['text'] = answer_option[0][0]

R2['text'] = answer_option[indices[2]][1] → R2['text'] = answer_option[0][1]

R3['text'] = answer_option[indices[2]][2] → R3['text'] = answer_option[0][2]

R4['text'] = answer_option[indices[2]][3] → R4['text'] = answer_option[0][3]

➔ This procedure continues till the ques = 7 since, we need only 8 questions then after 8th Click the value of ques is incremented by 1 now, ques is 8. So, else statement is executed.

- After completion of 8 questions quiz should terminate and display the score of the user. So, else block calls the function calculate().

- This function calculate the score of the user by comparing the "User_answer" list with "answers" list index by index.

  Note : "answers" list consists of correct options of corresponding questions
  Which are present in "Questions" list.

```
def calculate():
    global indices,User_answer,answers
    y = 0
    score = 0
    for i in indices:                    → we have uses for loop to iterate through the "indices" list
        if User_answer[y] == answers[i]:  → Then used a if condition to check whether the user response is
                                            same as that of correct answer in "answers" list.
                                            Of course the user response is for that same index present in
                                            indices list.
            score = score+5              → If the condition is true the score is incremented by 5 since,
                                            each question carries 5 marks.
    y += 1                               → This should continue till 8 questions that means from index
                                            (0-7) and which must compare all 8 response with their
                                            respective answer.
    result(score)                        → After calculating the score we need to display the score.
                                            So, result(score) function does this required function to
                                            Display the score of the user.
```

- So, result(score) function with score as an attributes which destroys widgets labelques and radio buttons R1,R2,R3 and R4 and creates another label which displays the score, and a pie chart which shows the performance and comment according to the score and a quit button to exit from quiz main window.

```
def result(score):
    labelques.destroy()
    R1.destroy()
    R2.destroy()
    R3.destroy()
    R4.destroy()
```

→ Since, we didn't declared score scope as global so we given it as an attribute for result() function.

→ All the widgets Label and RadioButton were destroyed.

```
    labelresult = Label(
        main,
        text=f'Your Score is {score}',
        font=('calibri',20),
        justify='center',
        bg='#ffffff',
    )
    labelresult.pack(pady=(100,0))
```

→ Now, we have created another label widget to display the score of the user and assigned it to a variable labelresult.

→ text of the widget is given using string formatting such as f'Your Score is {score}' it can also be done as 'your score is{ }'.format(score) with this we can get the score.

→ we have set the justify = 'center'.

→ background = 'white' i.e; #ffffff.

```
    labels = 'Correct', 'Wrong'
    sizes = [score,40-score]

    colors = ['green', 'red']
    explode = (0,0)
```

→ labels of the graphs are 'correct' and 'Wrong'

→ size of the labels is should be arranged according to score so, max score = 40 i.e; if score is 40 the correct must be 100% , if score is 0 wrong is 100%.

```
    fig = matplotlib.figure.Figure(figsize=(7,7))

    ax = fig.add_subplot(111)
```

→ To adjust the figure of graph we have imported figures from matplotlib and imported pyplot to plot the graph perfectly.

```
    ax.pie(sizes, explode=explode, labels=labels, colors=colors,
           autopct='%1.1f%%', shadow=False)
```

→ The figure we need is a circle so we use pie() with labels,sizes,colors and explode as an attributes and autopct – auto precision gives us the label percentage and shadow = False → eliminates the shadows if any.

```
    circle=matplotlib.patches.Circle( (0,0), 0.7, color='white')
    ax.add_artist(circle)
```

→ To make a circle into a donut we need another library patches So, we imported patches. We need to adjust diameter of the circle such that forms a donut so inner diameter is 0.7 and color is white which match with background so, it looks like a donut.

```
    insert = FigureCanvasTkAgg(fig,main)
    insert.get_tk_widget().pack()
```

→ Now, comes the actual task to put the pie chart into the main window. So, we import "from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg"

```
    result_text = Label(main,
                        font=('calibri',20),
                        bg='#ffffff')
    result_text.pack()
```

→ Now, to display the comments after the score display according the score we use another Label widget with the following references assigned to "result_text" variable.

```
    if score>=35:

        result_text.config(text='Your Excellent !!\nKeep it up')
```

→ As we need comments according to the score if score >=35 result_text widget should show the following text:

NOTE: As we discussed earlier config is used to access an object's attributes after its initialization.

```
    elif(score > 25 and score < 30):
        result_text.config(text='You can Perform better\nBetter Luck Next time')
```

→ if the (25 < score < 30) the another comment will be displayed.

```
else:                                                    → if score < 25 then following comment will be displayed.
  result_text.config(text='Your performance is poor need to improve\n')

Button(                                                  → To exit out from the quiz we have used another button with some references and with
                                                           command which call function exit(). The function exit destroys the main window with
                                                           main.destroy().
  main,
  text='Quit',
  bg='red',
  fg='#000000',
  relief=FLAT,
  border=0,
  font=('calibri',14,'bold'),
  command=exit).pack(pady=(110,0),anchor=NE)             → Then packed the widget with pady=(110,0) i.e; 110px space above the widget
                                                           result_text and 0px below.
```

TECHNOLOGIES AND FRAMEWORK USED IN DEVELOPING THE APPLICATION

- The python code was written using jupyter notebook So, the file type is .ipynb.
- Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows us to launch applications and easily manage conda packages, environments, and channels without using command-line commands.
- Jupyter notebook application is present in the Navigator by default.
- Jupyter Notebook helped us to combine the code, descriptive text, output, images, and interactive interfaces into a single notebook file that is edited, viewed, during working and testing phase.

Github link for the source code:

https://github.com/Arjun144/Arjun-Rao-Rachakonda.git