|  |  |
| --- | --- |
|  | ''' |
|  | Q1] |
|  | tkinter is python's standerd GUI package. |
|  | The layout managers used to place components in GUI are also called Geometry Managers. These are used to place and align components. |
|  | There are 3 types of Geometry Managers in tkinter: |
|  | 1)pack: |
|  | This geometry manager organizes widgets in blocks before placing them in the parent widget. |
|  | Syntax: |
|  | widget.pack( pack\_options ) |
|  | expand − When set to true, widget expands to fill any space not otherwise used in widget's parent. |
|  | fill − Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal |
|  | dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally |
|  | and vertically). |
|  | side − Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT. |
|  | e.g.: |
|  | redbutton = Button(frame, text="Red", fg="red") |
|  | redbutton.pack( side = LEFT) |
|  |  |
|  | 2)Grid: |
|  | This geometry manager organizes widgets in a table-like structure in the parent widget. |
|  | Syntax: |
|  | widget.grid( grid\_options ) |
|  | column − The column to put widget in; default 0 (leftmost column). |
|  | columnspan − How many columns widgetoccupies; default 1. |
|  | ipadx, ipady − How many pixels to pad widget, horizontally and vertically, inside widget's borders. |
|  | padx, pady − How many pixels to pad widget, horizontally and vertically, outside v's borders. |
|  | row − The row to put widget in; default the first row that is still empty. |
|  | rowspan − How many rowswidget occupies; default 1. |
|  | sticky − What to do if the cell is larger than widget. By default, with sticky='', widget is centered in its cell. sticky may be the string concatenation of zero or more of N, E, S, W, NE, NW, SE, and SW, compass directions indicating the sides and corners of the cell to which widget sticks. |
|  | e.g.: |
|  | Tkinter.Label(root, text='R%s/C%s'%(r,c),borderwidth=1 ).grid(row=r,column=c) |
|  |  |
|  | 3)place: |
|  | This geometry manager organizes widgets by placing them in a specific position in the parent widget |
|  | Syntax: |
|  | widget.place( place\_options ) |
|  | anchor − The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget) |
|  | bordermode − INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise. |
|  | height, width − Height and width in pixels. |
|  | relheight, relwidth − Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget. |
|  | relx, rely − Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget. |
|  | x, y − Horizontal and vertical offset in pixels. |
|  | e.g.: |
|  | B = Tkinter.Button(top, text ="Hello", command = helloCallBack) |
|  | B.place(bordermode=OUTSIDE, height=100, width=100) |
|  |  |
|  | ''' |
|  | #Q2] |
|  | ''' |
|  | GUI app for convert foot to meter using python |
|  |  |
|  | ''' |
|  | from tkinter import \* |
|  | from tkinter import ttk |
|  |  |
|  | def calculate(\*args): |
|  | try: |
|  | value = float(feet.get()) |
|  | meters.set((0.3048 \* value \* 10000.0 + 0.5)/10000.0) |
|  | except ValueError: |
|  | pass |
|  |  |
|  | root = Tk() |
|  | root.title("Feet to Meters") |
|  |  |
|  | mainframe = ttk.Frame(root, padding="3 3 12 12") |
|  | mainframe.grid(column=0, row=0, sticky=(N, W, E, S)) |
|  | root.columnconfigure(0, weight=1) |
|  | root.rowconfigure(0, weight=1) |
|  |  |
|  | feet = StringVar() |
|  | meters = StringVar() |
|  | feet\_entry = ttk.Entry(mainframe, width=7, textvariable=feet) |
|  | feet\_entry.grid(column=2, row=1, sticky=(W, E)) |
|  | ttk.Label(mainframe, textvariable=meters).grid(column=2, row=2, sticky=(W, E)) |
|  | ttk.Button(mainframe, text="Calculate", command=calculate).grid(column=3, row=3, sticky=W) |
|  | ttk.Label(mainframe, text="feet").grid(column=3, row=1, sticky=W) |
|  | ttk.Label(mainframe, text="is equivalent to").grid(column=1, row=2, sticky=E) |
|  | ttk.Label(mainframe, text="meters").grid(column=3, row=2, sticky=W) |
|  | for child in mainframe.winfo\_children(): child.grid\_configure(padx=5, pady=5) |
|  | feet\_entry.focus() |
|  | root.bind('<Return>', calculate) |
|  | root.mainloop() |
|  |  |
|  |  |
|  |  |
|  | #Q3] |
|  | #a,b) |
|  | import MySQLdb |
|  | from numpy.random import randint |
|  |  |
|  | try: |
|  | conn=MySQLdb.connect('localhost','root','','event') |
|  | if conn: |
|  | print("Successfully Connected") |
|  | except: |
|  | print("Connection Failed") |
|  |  |
|  | c=conn.cursor() |
|  | sql = "CREATE TABLE IF NOT EXISTS FIFA (scores int)" |
|  | c.execute(sql) |
|  | for i in range(1,100000): |
|  | r=randint(low=10,high=30) |
|  | print(r) |
|  | c.execute('INSERT INTO FIFA VALUES(%s)',(r)) |
|  |  |
|  | c.execute("SELECT \* FROM FIFA") |
|  | print("After Inserting \n",c.fetchall()) |
|  |  |
|  | #--- 15.5646493434906 seconds --- to save 100000 recores in Database |
|  |  |
|  |  |
|  | #c) |
|  | from numpy.random import randint |
|  | f=open("events.txt","w+") |
|  | for i in range(1,100000): |
|  | r=str(randint(low=10,high=30)) |
|  | f.write(r) |
|  | cont=f.read() |
|  | print(cont) |
|  | f.close() |
|  | #--- 1.0710008144378662 seconds --- to save 100000 records in text file |
|  |  |
|  |  |
|  | Q4] |
|  | import socket |
|  | port = 60000 |
|  | s = socket.socket() |
|  | host = socket.gethostname() |
|  | s.bind((host, port)) |
|  | s.listen(5) |
|  | print('Server listening....') |
|  | while True: |
|  | conn, addr = s.accept() |
|  | print ('Got connection from', addr) |
|  | data = conn.recv(1024) |
|  | print('Server received', repr(data)) |
|  | filename='events.txt' |
|  | f = open(filename,'rb') |
|  | l = f.read(1024) |
|  | while (l): |
|  | conn.send(l) |
|  | print('Sent ',repr(l)) |
|  | l = f.read(1024) |
|  | f.close() |
|  | print('Done sending') |
|  | conn.send(b'Thank you for connecting') |
|  | conn.close() |
|  |  |
|  |  |
|  |  |
|  | #receiver.py |
|  | import socket |
|  | s = socket.socket() |
|  | host = socket.gethostname() |
|  | port = 60000 |
|  | s.connect((host, port)) |
|  | s.send(b'Hello server!') |
|  | with open('Received.txt', 'wb') as f: |
|  | print('file opened') |
|  | while True: |
|  | print('receiving data...') |
|  | data = s.recv(1024) |
|  | print((data)) |
|  | f.write(data) |
|  | if not data: |
|  | break |
|  | f.close() |
|  | print('Successfully get the file') |
|  | s.close() |
|  | print('connection closed') |