ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ
ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΠΟΛΙΤΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΔΟΜΟΣΤΑΤΙΚΗΣ

Ακαδ. Έτος: 2020-2021

Πέμπτη, 20/5/2021

Μάθημα: Προγραμματισμός Η/Υ

Διδάσκοντες: Ν.Δ. Λαγαρός (Αν. Καθηγητής), Α. Στάμος (ΕΔΙΠ), Χ. Φραγκουδάκης (ΕΔΙΠ)
Αμβ. Σαββίδης (ΥΔ)

Θεωρία και παραδείγματα για την 9η παράδοση – GUI

1. Υπολογισμός παροχής ανοιχτού αγωγού

Παρακάτω δίνεται ο κώδικας υπολογισμού παροχής ανοιχτού με τις ακόλουθες προσθήκες:
α. Το παράθυρο με τα widgets είναι τώρα Toplevel() αντί για Tk(), έτσι ώστε να μπορούμε να δημιουργούμε αυθαίρετο πλήθος παραθύρων (ταυτόχρονα).
β. Δημιουργία καθολικής μεταβλητής openedwins που περιέχει τα στιγμιότυπα (instances) της κλάσης που έχουν δημιουργηθεί (μαζί με τα αντίστοιχα παράθυρα Toplevel). Όταν δημιουργείται νέο στιγμιότυπο (και το παράθυρό του) το προσθέτουμε στην openedwins, και όταν κλείνει ένα παράθυρο διαγράφουμε το αντίστοιχο στιγμιότυπο από την openedwins.
γ. Μέθοδος fileNew() που δημιουργεί νέο στιγμιότυπο και νέο παράθυρο, και την αντίστοιχη εντολή στο μενού.
δ. Δημιουργία καθολικής μεταβλητής maiwin που περιέχει το κύριο παράθυρο Tk(), το οποίο είναι απαραίτητο από το tkinter. Το κύριο παράθυρο δεν κάνει τίποτα, και το κάνουμε αόρατο μέσω της μεθόδου withdraw().
ε. Μέθοδο exitall() και την αντίστοιχη εντολή στο μενού, που κλείνει όλα τα ανοιγμένα παράθυρα (με έλεγχο αν υπάρχουν μη αποθηκευμένες αλλαγές).
στ. Η μέθοδος telos() τώρα κλείνει το κύριο παράθυρο mainwin, όταν ο χρήστης κλείσει όλα τα παράθυρα Toplevel.

Κώδικας GUI:

```
import tkinter as tk
import tkinter.font, tkinter.messagebox, tkinter.filedialog
import hydr

openedwins = set()
mainwin = None

class YdrWin:
    "A tkinter window for hydraulics."

    def __init__(self):
        "Make the window."
        self.fn = ""    #Previous filename
        self.root = tk.Toplevel(mainwin)
        openedwins.add(self)    #add current window to to set of opened windows
        self.setFonts()
        self.setMenus()
        self.makeWidgets()
        self.root.columnconfigure(1, weight=1)
        self.valsprev = 1.0, 0.5, 0.7, 0.015    #Previous values
        self.setValues(self.valsprev)
```

```
        self.root.protocol("WM_DELETE_WINDOW", self.telos)


    def makeWidgets(self):
        "Create the widgets."
              lab  =  tk.Label(self.root,  text="School  of  civil  engineering",
foreground="green")
        lab.grid(row=0, column=0, columnspan=2)
        self.entWidth   = self.labEntry(1, "Width (m):")
        self.entDepth   = self.labEntry(2, "Depth (m):")
        self.entSlope   = self.labEntry(3, "Slope (%):")
        self.entManning = self.labEntry(4, "Manning:")
        self.entVel     = self.labEntry(5, "Velocity (m/s):")
        self.entDis     = self.labEntry(6, "Discharge (m3/s):")
        self.entVel.config(state="readonly", readonlybackground="blue")
        self.entDis.config(state="readonly", readonlybackground="blue")


    def getValues(self, showerror=True):
        "Get the values of the widgets and check if they are valid."
        ents  = self.entWidth, self.entDepth, self.entSlope, self.entManning
        descs = "width", "depth", "slope", "manning"
        vals = []
        for ent, desc in zip(ents, descs):
            try: v = float(ent.get())
            except: v = None
            if v is None or v <= 0:
                if showerror:
                       tk.messagebox.showinfo("Error", desc+" is not a positive
number",
                        parent=self.root, icon=tk.messagebox.ERROR)
                    ent.focus_set()
                return None
            vals.append(v)
        return tuple(vals)


    def setValues(self, vals):
        "Set the values of the widgets."
        ents  = self.entWidth, self.entDepth, self.entSlope, self.entManning
        for ent, v in zip(ents, vals):
            ent.delete(0, tk.END)
            ent.insert(0, str(v))


    def setResults(self, vals):
        "Set the values of the result widgets."
        ents  = self.entVel, self.entDis
        for ent, v in zip(ents, vals):
            ent.config(state=tk.NORMAL)     #unset readonly
            ent.delete(0, tk.END)
            ent.insert(0, str(v))
            ent.config(state="readonly")    #set readonly again


    def labEntry(self, ir, desc):
        "Make a label and an entry widget."
        lab = tk.Label(self.root, text=desc)
        lab.grid(row=ir, column=0, sticky="e")
        ent = tk.Entry(self.root, width=10)
        ent.grid(row=ir, column=1, sticky="we")
        return ent
```

```
    def setFonts(self):
        "Set the fonts a little bigger."
        for t in "TkDefaultFont", "TkFixedFont", "TkMenuFont", "TkTextFont":
            font1 = tk.font.Font(name=t, exists=True)
            font1.config(size=20) # Negative size means size in pixels
        font1.config(family="Arial")


    def setMenus(self):
        "Create menu system."
        menuBar = tk.Menu(self.root, activebackground="green")
        self.root.config(menu=menuBar)

        menu2 = tk.Menu(menuBar, activebackground="green", tearoff=False)
        menuBar.add_cascade(label="File", menu=menu2, foreground="blue")
        menu2.add_command(label="New", command=self.fileNew, foreground="blue")
                        menu2.add_command(label="Open",   command=self.fileOpen,
foreground="blue")
                        menu2.add_command(label="Save",   command=self.fileSave,
foreground="blue")
                    menu2.add_command(label="Saveas",   command=self.fileSaveas,
foreground="blue")
        menu2.add_separator()
        menu2.add_command(labe="Close", command=self.telos, foreground="blue")
        menu2.add_separator()
        menu2.add_command(labe="Exit", command=self.exitall, foreground="blue")

        menu2 = tk.Menu(menuBar, activebackground="green", tearoff=False)
        menuBar.add_cascade(label="Compute", menu=menu2, foreground="blue")
             menu2.add_command(label="Compute  Discharge",  command=self.ypolPar,
foreground="blue")


    def ypolPar(self):
        "Compute discharge."
        vals = self.getValues()
        if vals is None: return
        print("values=", vals)
        b, h, s, n = vals
        v, q = hydr.calcDisRect(b, h, s, n)
        self.setResults((v, q))


    def fileNew(self):
        "Make a new window."
        win = YdrWin()

    def fileOpen(self):
        "Open and read a file."
        #if not self.ok2quit(): return
        while True:
                            fr   =   tk.filedialog.askopenfile(parent=self.root,
defaultextension=".man",
                title="Open file")
            if fr is None: return   #User cancelled save
            fntemp = fr.name
            try:
                vals = hydr.readData(fr)
                break
            except IOError as e:
                tk.messagebox.showinfo("Error reading file "+fr.name, str(e),
                    parent=self.root, icon=tk.messagebox.ERROR)
            finally:
                fr.close()
```

```
        win = YdrWin()
        win.setValues(vals)
        win.ypolPar()
        win.fn = fntemp
          win.valsprev = vals      #So that if user exits the program will not
complain


    def fileSave(self):
        "Save data to previous file."
        vals = self.getValues()
        if vals is None: return    #Errors were found
        b, h, s, n = vals

        if self.fn == "": return self.fileSaveas()   #No previous file
        er = False
        try:             fw = open(self.fn, "w")
        except IOError:  er = True    #Could not open previous file
        if er: return self.fileSaveas()

        try:                 hydr.writeData(fw, b, h, s, n)
        except IOError as e: er = True
        finally:             fw.close()
        if er: return self.fileSaveas()
          self.valsprev = vals      #So that if user exits the program will not
complain


    def fileSaveas(self):
        "Save data to a file."
        vals = self.getValues()
        if vals is None: return    #Errors were found
        b, h, s, n = vals
        while True:
                            fw  =  tk.filedialog.asksaveasfile(parent=self.root,
defaultextension=".man",
                title="Open file for save")
            if fw is None: return   #User cancelled save
            fntemp = fw.name
            try:
                hydr.writeData(fw, b, h, s, n)
                break
            except IOError as e:
                tk.messagebox.showinfo("Error writting file "+fw.name, str(e),
                    parent=self.root, icon=tk.messagebox.ERROR)
            finally:
                fw.close()
        self.fn = fntemp
          self.valsprev = vals      #So that if user exits the program will not
complain


    def telos(self):
        "Break circular references and terminate GUI."
        if not self.ok2quit(): return  False  #User cancelled exit
            del  self.entWidth,  self.entDepth,  self.entSlope,  self.entManning,
self.entVel, self.entDis
        openedwins.remove(self)     #remove currnet windows from opened windows
        self.root.destroy()
        del self.root
        if len(openedwins) == 0:    #if no opened windows, terminate application
            winmain.destroy()
        return True
```

```
    def exitall(self):
        "Ask all windows to close themselves."
        while len(openedwins) > 0:
            for win in openedwins: break   #Select one of the windows
            if not win.telos(): return   #User cancelled exit


    def ok2quit(self):
        "Ask the user if they want to abandon unsaved changes."
        vals = self.getValues(showerror=False)
        if vals == self.valsprev: return True
           ok = tk.messagebox.askokcancel("Values modified", "You have unsaved
changes.\n\nOk to abandon?",
            default="cancel", icon=tk.messagebox.WARNING)
        return ok


def pyMain():
    "Start here."
    global winmain
    winmain = tk.Tk()
    win = YdrWin()
    winmain.withdraw()
    winmain.mainloop()


pyMain()
```