Γραφικά User Interfaces με Κλάσεις

Dictionaries

Μία πολύ χρήσιμη δομή για συλλογή πληροφοριών και αναζήτηση μέσω κλειδιού είναι το Dictionary. Τα dictionaries είναι mutable, αταξινόμητα και indexed. Για τη δήλωση τους χρησιμοποιούνται αγκύλες, μέσα στις οποίες βρίσκονται ζευγάρια από κλειδιά και τιμές, χωρισμένα με κόμμα. Τα dictionaries χρησιμοποιούν hash function για την αναζήτηση.

```
carDict = {
    'brand': 'Fiat',
    'model': 'Tipo',
    'cc':     1400,
    'year':   1992
}
```

Πρόσβαση στην πληροφορία έχουμε μέσω του κλειδιού με δύο τρόπους: χρησιμοποιώντας το κλειδί σαν δείκτη ή μέσω getter.

```
>>> print(carDict)
{'brand': 'Fiat', 'model': 'Tipo', 'cc': 1400, 'year': 1992}
>>> type(carDict)
<class 'dict'>
>>> carDict['cc']
1400
>>> carDict.get('cc')
1400
>>>
```

Μπορούμε να ζητήσουμε τα κλειδιά, τις τιμές ή όλα τα items με τις μεθόδους keys(), values() και items().

```
>>> carDict.keys()
dict_keys(['brand', 'model', 'cc', 'year'])
>>> carDict.values()
dict_values(['Fiat', 'Tipo', 1400, 1992])
>>> carDict.items()
dict_items([('brand', 'Fiat'), ('model', 'Tipo'), ('cc', 1400),
('year', 1992)])
>>>
```

Τα κλειδιά, οι τιμές και τα ζευγάρια ανήκουν στις τρεις κλάσεις:

```
>>> type(carDict.keys())
<class 'dict_keys'>
>>> type(carDict.values())
<class 'dict_values'>
>>> type(carDict.items())
<class 'dict_items'>
>>>
```

Τα κλειδιά, οι τιμές και τα ζευγάρια μπορούν εύκολα να μετατραπούν σε λίστες καλώντας την list():

```
>>> k=carDict.keys()
>>> v=carDict.values()
>>> i=carDict.items()
>>> list(k)
['brand', 'model', 'cc', 'year']
>>> list(v)
['Fiat', 'Tipo', 1400, 1992]
>>> list(i)
[('brand', 'Fiat'), ('model', 'Tipo'), ('cc', 1400), ('year', 1992)]
>>>
```

Οι τιμές μπορούν να ενημερωθούν μέσω του κλειδιού και ζευγάρια μπορούν εύκολα να καταργηθούν μέσω της pop(). Η pop επιστρέφει την τιμή που διαγράφει:

```
>>> carDict['cc']=1600
>>> carDict['year']=2019
>>> carDict
{'brand': 'Fiat', 'model': 'Tipo', 'cc': 1600, 'year': 2019}
>>> carDict.pop('year')
2019
>>> carDict
{'brand': 'Fiat', 'model': 'Tipo', 'cc': 1600}
>>>
```

Ενημέρωση μπορεί να γίνει και με την μέθοδο update, η οποία μπορεί να χρησιμοποιηθεί και για να προστεθούν ζεύγη κλειδιών-τιμών:

```
>>> carDict.update(year = 2019)
>>> carDict
{'brand': 'Fiat', 'model': 'Tipo', 'cc': 1600, 'year': 2019}
>>> carDict.update(year = 2018, color = 'blue')
>>> carDict
{'brand': 'Fiat', 'model': 'Tipo', 'cc': 1600, 'year': 2018, 'color':
'blue'}
>>>
```

GUIs

Ένα ελάχιστο GUI πρόγραμμα (από Wikipedia):

```
1 #!/usr/bin/env python3
2 from tkinter import *
3 root = Tk()                                # Create the root (base) window
4 w = Label(root, text="Hello, world!")      # Create a label with words
5 w.pack()                                   # Put the label into the window
6 root.mainloop()                           # Start the event loop
```

Η υλοποίηση γραφικών προγραμμάτων γίνεται σε 4 φάσεις:

- Δημιουργία μέσα σε ένα frame
- Τροποποιήσεις των attributes του widget
- Τοποθέτηση για να γίνει ορατό
- Σύνδεση με λειτουργία ή event

Το ελάχιστο πρόγραμμα μπορεί να γραφτεί και έτσι:

```
import tkinter as tk

root = tk.Tk()
w = tk.Label(root, text="Hello, world!")
w.pack()
root.mainloop()
```

Ένα πολύ χρήσιμο widget για γραφικά user interfaces είναι το Label, το οποίο μας δίνει την δυνατότητα να παρουσιάζουμε κείμενο ή εικόνες. Το Label το χρησιμοποιούμε μόνο για παρουσίαση πληροφοριών.

Κάνουμε import το tkinter και κατασκευάζουμε ένα root widget. Το root widget είναι ένα παράθυρο με τα ντεκόρ του. Το κατασκευάζουμε στην αρχή.

```
import tkinter as tk

root = tk.Tk()
logo = tk.PhotoImage(file="python.gif")

w1 = tk.Label(root, image=logo).pack(side="right")

explanation = """At present, only GIF and PPM/PGM
formats are supported, but an interface
exists to allow additional image file
formats to be added easily."""

w2 = tk.Label(root,
              justify=tk.LEFT,
              padx = 10,
              text=explanation).pack(side="left")
root.mainloop()
```

Ένα δεύτερο παράδειγμα από Wikipedia:

```
 1 #!/usr/bin/env python3
 2 import tkinter as tk
 3
 4 class Application(tk.Frame):
 5
 6     def __init__(self, master=None):
 7         tk.Frame.__init__(self, master)
 8         self.grid()
 9         self.createWidgets()
10
11     def createWidgets(self):
12         self.mondialLabel = tk.Label(self, text='Hello World')
13         self.mondialLabel.config(bg="#00ffff")
14         self.mondialLabel.grid()
15         self.quitButton = tk.Button(self, text='Quit',
command=self.quit)
16         self.quitButton.grid()
17
18 app = Application()
19 app.master.title('Sample application')
20 app.mainloop()
```

- line 1: Hashbang directive to the program launcher, allowing the selection of an appropriate interpreter executable, when self-executing.
- line 2: This line imports the tkinter module into your program's namespace, but renames it as tk.
- line 4: The application class inherits from Tkinter's Frame class.
- line 6: Defines the function that sets up the Frame
- line 7: Calls the constructor for the parent class, Frame.
- line 11: Defining the widgets
- line 12: Creates a label, named MondialLabel with the text "Hello World"
- line 13: Sets the MondialLabel background colour to cyan
- line 14: Places the label on the application so it is visible using the grid geometry manager method
- line 15: Creates a button labeled “Quit”.
- line 16: Places the button on the application. Grid, place and pack are all methods of making the widget visible
- line 18: The main program starts here by instantiating the Application class.
- line 19: This method call sets the title of the window to “Sample application”.
- line 20: Starts the application's main loop, waiting for mouse and keyboard events.

```
#!/usr/bin/env python3
import tkinter as tk

class Application(tk.Frame):

    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self.grid()
        self.createWidgets()
    def createWidgets(self):
        self.mondialLabel = tk.Label(self, text='Hello World')
        self.mondialLabel.config(bg="#00ffff")
        self.mondialLabel.grid()
        self.quitButton = tk.Button(self, text='Quit',
                                                command=self.quit)
        self.quitButton.grid()

app = Application()
app.master.title('Sample application')
app.mainloop()
```

Το tkinter χρησιμοποιεί 4 **τύπους μεταβλητών**

`StringVar`: Για χαρακτήρες, όπως το Python string.
`IntVar`: Για ακέραιους.
`DoubleVar`: Double, αριθμοί με δεκαδικά.
`BooleanVar`: Τιμές Boolean.

Το tkinter διαθέτει τρεις Διαχειριστές Διάταξης

Pack, Grid και Place.

Pack: Προσαρμόζει το μέγεθος του παραθύρου έτσι ώστε να χωράει τα περιεχόμενα. Αν προσθέσουμε widgets το μέγεθος του παράθυρου μεγαλώνει.

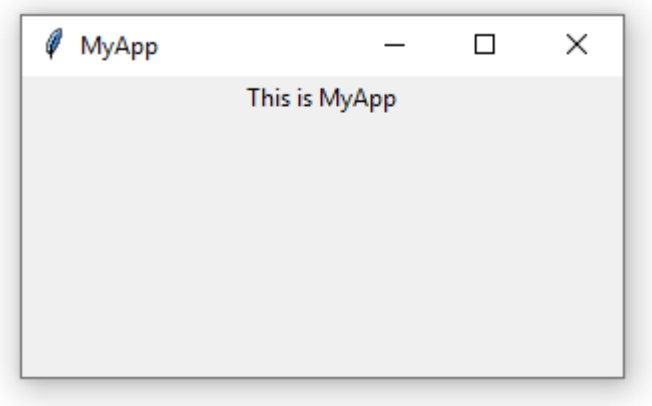
Attributes:
side (LEFT, RIGHT, TOP, BOTTOM)
fill (NONE, BOTH, X και Y)
expand (YES, NO)

anchor (N, S, E, W, NW, SE, NE, SE και CENTER)
padx, pady, ipadx και ipady

Παράδειγμα από το βιβλίο "Μαθαίνετε εύκολα Python" του Δημήτρη Καρολίδη, εκδόσεις Άβακας:

```
from tkinter import *

class MyApp:
    def __init__(self):
        self.root = Tk()
        self.root.title('MyApp')
        self.root.geometry('300x150')
        self.myFrame = Frame(self.root)
        self.myFrame.pack()
        self.myLabel = Label(self.myFrame)
        self.myLabel['text']='This is MyApp'
        self.myLabel.pack()
        self.root.mainloop()

if __name__ == '__main__':
    app=MyApp()
```

Το κείμενο του Label θα μπορούσε να έχει περαστεί στον κατασκευαστή. Εδώ βλέπουμε ότι υπάρχει η εναλλακτική λύση, να χρησιμοποιήσουμε dictionary.



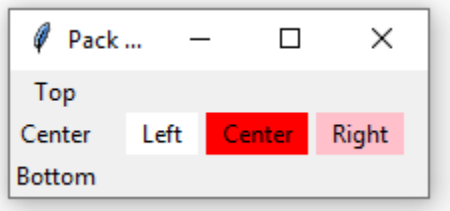
Το μέγεθος του παράθυρου αλλάζει αυτόματα εαν καταργήσουμε τις διαστάσεις και προσθέσουμε άλλο ένα label:

Label(self.myFrame, text='Onother Label').pack()

Αυτό το πρόγραμμα χρησιμοποιεί διάφορα χαρακτηριστικά της διάταξης pack:

```
from tkinter import *

class MyApp:
    def __init__(self):
        self.root = Tk()
        self.root.title('Pack Example')
        self.fm1 = Frame(self.root)
        self.l1 = Label(self.fm1, text='Top')
        self.l1.pack(side=TOP, anchor=W, fill=X, expand=YES)
        self.l2 = Label(self.fm1, text='Center')
        self.l2.pack(side=TOP, anchor=W, fill=X, expand=YES)
        self.l3 = Label(self.fm1, text='Bottom')
        self.l3.pack(side=TOP, anchor=W, fill=X, expand=YES)
        self.fm1.pack(side=LEFT, fill=BOTH, expand=YES)
        self.fm2 = Frame(self.root)
        self.l4 = Label(self.fm2, text='Left', bg='white')
        self.l4.pack(side=LEFT, padx=2, ipadx=5)
        self.l5 = Label(self.fm2, text='Center', bg='red')
        self.l5.pack(side=LEFT, padx=2, ipadx=5)
        self.l6 = Label(self.fm2, text='Right', bg='pink')
        self.l6.pack(side=LEFT, padx=2, ipadx=5)
        self.fm2.pack(side=LEFT, padx=10)
        self.root.mainloop()

if __name__ == '__main__':
    app=MyApp()
```

Το αποτέλεσμα:



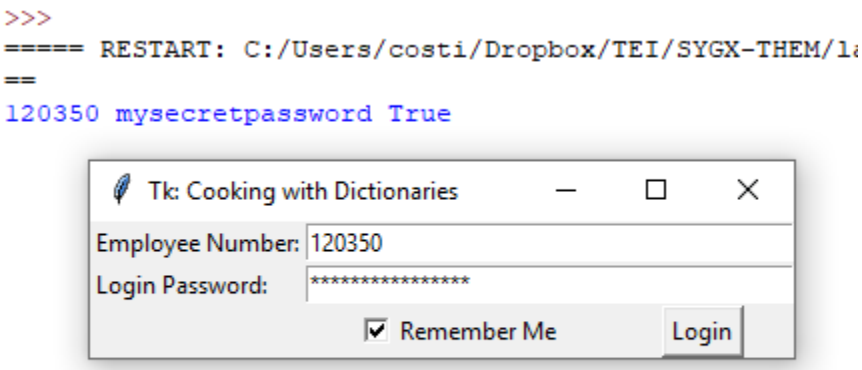
Μία τεχνική που βοηθάει να δουλέψουμε με το πέλαγος των ιδιοτήτων των widgets είναι η χρήση dictionaries. Εδώ ένα παράδειγμα από το DaniWeb[1]:

```
from tkinter import *


    # Main #
def main():
    tk_TkGUI = Tk()
    tk_TkGUI.title("Tk: Cooking with Dictionaries")
                # Tk Variables
    v_EmployeeNum = IntVar() ## Init the built-in IntVar().
    v_EmployeeNum.set("120350")
    v_Password = StringVar() ## Init th ebuilt-in StringVar()
    v_Password.set("mysecretpassword")
    v_Remember = BooleanVar()
    v_Remember.set(True)
                # Tk Config Dicts.
    tkl_ConfigLabel1 = {"text": "Employee Number:"}
    tkl_ConfigLabel2 = {"text": "Login Password:"}
    tke_ConfigEntry1 = {"width": 40, "textvariable": v_EmployeeNum}
    tke_ConfigEntry2 = {"width": 40, "show": "*", "textvariable":
v_Password}
    tkb_ConfigButton1 = {"text": "Login", "command": (lambda:
f_ShowInfo(v_EmployeeNum, v_Password, v_Remember))}
    tkcb_ConfigCheckButton1 = {"text": "Remember Me", "variable":
v_Remember}
                # Tk Grid Dicts.
    tke_GridEntry2 = {'column': 1, 'columnspan': 2, 'row': 1}
    tke_GridEntry1 = {'column': 1, 'columnspan': 2, 'row': 0}
    tkl_GridLabel1 = {'column': 0, 'columnspan': 1, 'row': 0}
    tkl_GridLabel2 = {'column': 0, 'columnspan': 1, 'row': 1,
'sticky': "w"}
    tkcb_GridCheckButton1 = {'column': 1, 'columnspan': 1, 'row': 2}
    tkb_GridButton1 = {'column': 2, 'columnspan': 1, 'row': 2}
                # Tk Widgets
    tkl_Label1 = Label(tk_TkGUI,
tkl_ConfigLabel1).grid(tkl_GridLabel1)
    tke_Entry1 = Entry(tk_TkGUI,
tke_ConfigEntry1).grid(tke_GridEntry1) ## Display an Entry widget.
    tkl_Label2 = Label(tk_TkGUI,
tkl_ConfigLabel2).grid(tkl_GridLabel2) ## Display a Label.
    tke_Entry2 = Entry(tk_TkGUI,
tke_ConfigEntry2).grid(tke_GridEntry2)
```

1 https://www.daniweb.com/programming/software-development/code/498607/python-using-dictionaries-to-config-and-display-tkinter

```
    tkcb_CheckButton1 = Checkbutton(tk_TkGUI,
tkcb_ConfigCheckButton1).grid(tkcb_GridCheckButton1)
    tkb_Button1 = Button(tk_TkGUI,
tkb_ConfigButton1).grid(tkb_GridButton1) ## Display our button with a
"lambda" call to a func with passed parameters.



    tk_TkGUI.mainloop()
    # Main Loop #

def f_ShowInfo(num, passwd, remember):
        print(num.get(), passwd.get(), remember.get())


if __name__ == '__main__':
    main()
```

Το αποτέλεσμα φαίνεται στην εικόνα που ακολουθεί. Στο background το πρόγραμμα τυπώνει στο shell τις τιμές αφού πατηθεί το button:



Στο παράδειγμα αυτό γίνεται χρήση lambda function. Τα lambdas είναι μικρές ανώνυμες συναρτήσεις που βρίσκονται συνήθως μέσα σε άλλες συναρτήσεις.

```
x = lambda a : a + 10
print(x(5))
```

Επιστρέφει 15

```
x = lambda a, b : a * b
print(x(5, 6))
```

Επιστρέφει 30

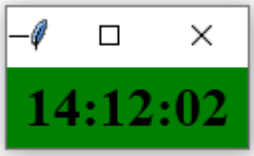
Δέχονται όποιον αριθμό από arguments χρειάζεται. Σε αυτό το παράδειγμα δέχεται ένα argument:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

Ένα ακόμη παράδειγμα από το DaniWeb, το οποίο μάλιστα είναι κατασκευασμένο έτσι, ώστε να τρέχει και με Python 2. Επειδή στην Python 2 η βιβλιοθήκη tkinter ονομάζεται Tkinter, γίνεται χρήση του try-except για να βρει το πρόγραμμα την σωστή βιβλιοθήκη.

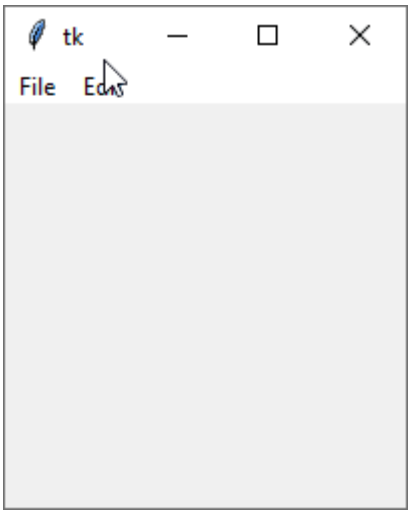
```
''' tk_clock101.py
    use Tkinter to show a digital clock
    tested with Python27 and Python33
'''
import time
try:
    # Python2
    import Tkinter as tk
except ImportError:
    # Python3
    import tkinter as tk

def tick(time1=''):
    # get the current local time from the PC
    time2 = time.strftime('%H:%M:%S')
    # if time string has changed, update it
    if time2 != time1:
        time1 = time2
        clock.config(text=time2)
    # calls itself every 200 milliseconds
    # to update the time display as needed
    clock.after(200, tick)

root = tk.Tk()
clock = tk.Label(root, font=('times', 20, 'bold'), bg='green')
clock.pack(fill='both', expand=1)

tick()
root.mainloop()
```

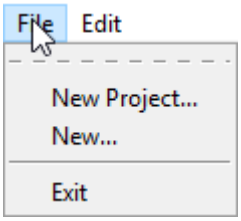
Το πρόγραμμα κατασκευάζει ένα ρολόι που τρέχει μόνιμα μέχρι να το σταματήσει ο χρήστης.



Η κατασκευή menu είναι πολύ απλή. Εδώ βλέπουμε πως δηλώνουμε τα widgets για ένα απλό menu με δύο drop down menus:

```
from tkinter import *

def doNothing():
    print("ok ok I won't...")

root = Tk()

menu = Menu(root)
root.config(menu=menu)

subMenu=Menu(menu)
menu.add_cascade(label='File', menu=subMenu)
subMenu.add_command(label='New Project...', command=doNothing)
subMenu.add_command(label='New...', command=doNothing)
subMenu.add_separator()
subMenu.add_command(label='Exit', command=doNothing)

editMenu=Menu(menu)
menu.add_cascade(label='Edit', menu=editMenu)
editMenu.add_command(label='Redo', command=doNothing)

root.mainloop()
```


tk
File
Edit


File
Edit
New Project...
New...
Exit