

## Les 10: Voorbereiding Miniproject: GUI

### Voorbereiding

Les 10 kent geen Final Assignment omdat de stof van deze les niet getoetst zal worden op het tentamen. Voor het miniproject is het echter wel de bedoeling dat je de gebruiker een Grafische User Interface (GUI) kunt aanbieden. Deze les bestaat onder andere uit een tutorial waarbij je basiskennis oproeft met betrekking tot het maken van een eenvoudige GUI in Python. Het kan handig zijn om daarvoor Perkovic, paragraaf 9.1 en 9.2 door te nemen, maar dat is niet noodzakelijk!

### Tutorial 1: Tkinter

Bestudeer de verschillende stappen hieronder, en probeer de code uit op je eigen laptop. Verander ook zo nu en dan de code om te kijken wat het effect is en of je bijvoorbeeld het uiterlijk van je programma kunt beïnvloeden.

#### Stap 1: Een GUI openen

Om in Python een GUI te maken, gebruiken we module **tkinter**. Deze is standaard in Python aanwezig:

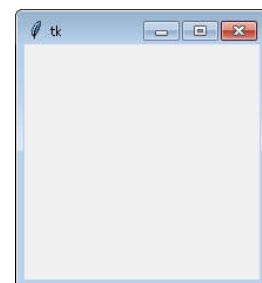
```
from tkinter import *
root = Tk()                      # Creeer het hoofdscherm
root.mainloop()                   # Toon het hoofdscherm
```

De eerste regel valt op, want hier importeren we op een andere manier dan gebruikelijk. Je zegt daarmee als het ware: importeer alles (\*) van module tkinter. Het voordeel is dan dat je niet steeds voor alles wat je wilt gebruiken de modulenaam hoeft te zetten. Zie het verschil hieronder:

<code>from tkinter import *</code>	<code>import tkinter</code>
<code>root = Tk()</code>	<code>root = <u>tkinter</u>.Tk()</code>
<code>root.mainloop()</code>	<code>root.mainloop()</code>

Het uitvoeren van deze code levert overigens een leeg scherm op. Dit is de basis waarop je allerlei GUI-onderdelen kunt gaan plaatsen. Een onderdeel kan bijvoorbeeld een Label of Button zijn. In tkinter heten die onderdelen **widgets**.

We gaan nu achtereenvolgens de widgets Label, Button en Entry toevoegen aan het hoofdscherm! Probeer deze voorbeelden ook steeds uit op je eigen laptop!



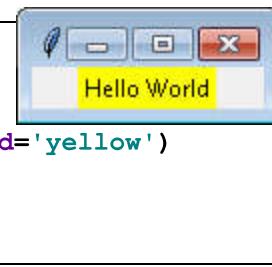
#### Stap 2: Widgets toevoegen

We beginnen met het toevoegen van een Label:

```
from tkinter import *
root = Tk()

label = Label(master=root, text='Hello World', background='yellow')
label.pack()

root.mainloop()
```



Aan een Label geven we een aantal parameters mee:

<code>master=root</code>	= plaatsen op hoofdscherm ( <b>root</b> )
<code>text='Hello World'</code>	= de tekst dit getoond moet worden
<code>background='yellow'</code>	= de achtergrondkleur van het label

Methode **pack()** plaatst het label zodanig in de hoofdvenster (**root**), dat het label het window volledig opvult. Daarbij wordt het item door deze methode automatisch tegen de bovenkant (TOP) van het scherm geplakt (tenzij je anders aangeeft, dat zien we zo direct).

Er zijn nog veel meer parameters voor een Label mogelijk, zie bijvoorbeeld Perkovic, Table 9.1 (blz 293). De code hieronder toont bijvoorbeeld hoe je het lettertype, de letterkleur en de hoogte + breedte (in aantal tekst-units) van een Label kunt aanpassen:

```
from tkinter import *
root = Tk()

label = Label(master=root,
              text='Hello World',
              background='yellow',
              foreground='blue',
              font=('Helvetica', 16, 'bold italic'),
              width=14,
              height=3)
label.pack()

root.mainloop()
```



Omdat hier veel parameters worden ingesteld, zijn ze voor de leesbaarheid onder elkaar geplaatst! De tekst van een Label wordt automatisch in het midden uitgelijnd! We voegen nu ook een Button toe aan het hoofdscherm (`master=root`). Dat gaat op dezelfde wijze:

```
from tkinter import *
root = Tk()

label = Label(master=root, text='Hello World', height=2)
label.pack()

button = Button(master=root, text='Druk hier')
button.pack(pady=10)

root.mainloop()
```



De button **doet** nu nog helemaal niets, daar kijken we pas naar bij stap 4. Merk verder op dat er enige ruimte is tussen de button en de rand van het window! Dat is gedaan door aan de functie pack een **padding** mee te geven (`pady`, dus op de y-as) van 10 pixels. Dit zorgt voor een lege rand van 10 pixels hoog, onder en boven de button.

- ➔ Je kan er ook nog voor kiezen om de button over de gehele breedte van het venster uit te rekken door de parameter `fill=X` mee te geven aan de pack functie. Probeer dit eens uit en bekijk het resultaat! Combineer dit met de parameter `padx`. Wat is het effect daarvan?

De volgende stap is het toevoegen van een invoerveld. We gebruiken wederom dezelfde werkwijze:

```
from tkinter import *
root = Tk()

label = Label(master=root, text='Hello World', height=2)
label.pack()

button = Button(master=root, text='Druk hier')
button.pack(pady=10)

entry = Entry(master=root)
entry.pack(padx=10, pady=10)

root.mainloop()
```



Naast deze belangrijke widgets zijn er nog 14 andere widgets die standaard in tkinter voorkomen. Deze zullen we niet allemaal bekijken, maar je kunt het overzicht vinden in de [documentatie](#) van tkinter. Je moet er wel rekening mee houden dat je niet bij elke widget dezelfde parameters kunt gebruiken! Lees daarom de documentatie of gebruik de **help()** functie om informatie over een widget op te vragen.

Er zijn echter nog een paar problemen: er gebeurt nog helemaal niets als we de knop indrukken, en wat als we veel meer widgets op verschillende posities aan het scherm willen toevoegen? We kijken daarom in stap 3 eerst hoe je ervoor kunt zorgen dat een knop ook werkt, en in stap 4 nemen we de layoutmanagers onder de loep.

### Stap 3: Event-based widgets

We gebruiken het laatste programma als uitgangspunt, en we gaan ervoor zorgen dat als je op de knop klikt, de tekst in het invoerveld in het Label geplaatst wordt. We doen dat als volgt:

1. We schrijven een functie **clicked()** die deze acties uitvoert.
2. We koppelen deze functie als **event handler** aan de knop.

Deze code is redelijk eenvoudig:

```
from tkinter import *

def clicked():
    label['text'] = entry.get()

root = Tk()

label = Label(master=root, text='Hello World', height=2)
label.pack()

button = Button(master=root, text='Druk hier', command=clicked)
button.pack(pady=10)

entry = Entry(master=root)
entry.pack(padx=10, pady=10)

root.mainloop()
```



Doordat aan de Button nu de parameter **command=clicked** is toegevoegd, zal bij het klikken op de knop de functie **clicked** worden aangeroepen. Het enige dat de functie doet, is het veranderen van de eigenschap **text** van het Label. De nieuwe waarde wordt datgene dat de user heeft ingevoerd in de **Entry**. Deze invoer vraagt de functie op met **entry.get()**.

**Let op:** de functie **clicked** MOET een functie **ZONDER parameters** zijn! Er zijn ook andere soorten event handler-functies waarbij wel een parameter meegegeven moet worden, maar die behandelen we niet in deze tutorial. Zie daarvoor Perkovic, blz 306-308!

**Let op:** de functie **clicked** wordt pas aangeroepen als iemand op de knop klikt! Daarom staat er ook **command=clicked** en geen **command=clicked()**. Op deze manier geef je de functie **clicked** mee aan de **Button** als parameter, zonder de functie al uit te voeren!

Vanzelfsprekend kan je ook andere code laten uitvoeren. In de onderstaande code zal bijvoorbeeld de ingevoerde waarde in de Entry gekwadrateerd worden. De uitkomst wordt weer in het Label geplaatst:

```
from tkinter import *

def clicked():
    grondtal = int(entry.get())
    kwadraat = grondtal ** 2
    tekst = "kwadraat: van {} = {}"
    label["text"] = tekst.format(grondtal, kwadraat)

root = Tk()

label = Label(master=root, text='Hello World', height=2)
label.pack()

button = Button(master=root, text='Druk hier', command=clicked)
button.pack(pady=10)

entry = Entry(master=root)
entry.pack(padx=10, pady=10)

root.mainloop()
```



Deze korte stukjes code geven een aardig beeld van de mogelijkheden en hiermee ben je in staat om een eenvoudige GUI op te bouwen en aan te passen.

- ➔ Probeer eens wat het effect is als je toch **command=clicked()** (dus met haakjes) gebruikt!
- ➔ Probeer het programma aan te passen zodat er twee buttons zijn. De ene button kwadrateert het getal in de Entry, en de andere button berekent de wortel van het ingevoerde getal. Schrijf voor de tweede button een aparte functie!

### Stap 4\_1: Layoutmanagers: pack()

We hebben nu steeds gebruik gemaakt van de functie **pack()**. Deze functie heeft echter als nadeel dat je beperkte mogelijkheden hebt. Zo kan je wel de voorkeurspositie van een widget opgeven, maar bepaalt de layoutmanager van tkinter waar dat onderdeel precies komt te staan. We zullen eerst eens kijken hoe dat werkt, en daarna zullen we ook twee alternatieven bespreken. Je kunt dan zelf bepalen welke je wilt gebruiken.

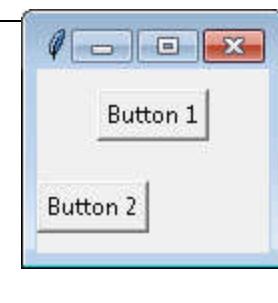
Stel je wilt twee buttons, dan zullen die met de functie **pack()** onder elkaar geplaatst worden. De functie probeert namelijk normaal gesproken de widget bovenaan het venster te plaatsen. Als daar al een widget staat, komt de volgende eronder. Dit gedrag kan je beïnvloeden met de parameter **side**:

```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.pack(pady=10)

button2 = Button(master=root, text='Button 2')
button2.pack(side=LEFT, pady=10)

root.mainloop()
```



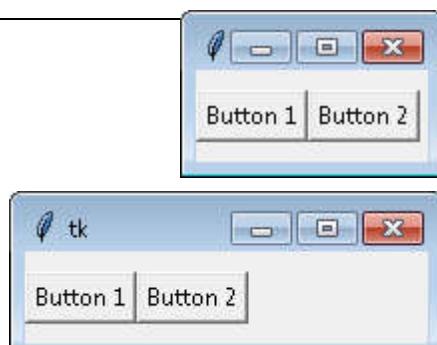
Er zijn 4 verschillende waarden mogelijk voor parameter **side**: TOP (default), LEFT, RIGHT en BOTTOM. Stel dat we button1 ook aan de linkerkant zouden plaatsen, dan krijg je een, misschien, onverwacht effect. Het bovenste plaatje is wat tkinter ervan maakt:

```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.pack(side=LEFT, pady=10)

button2 = Button(master=root, text='Button 2')
button2.pack(side=LEFT, pady=10)

root.mainloop()
```



Omdat button1 als eerste geplaatst is, staat deze helemaal links te de vensterrand. Button2 staat in feite ook links, maar dan naast het item dat er al stond. Dit is pas goed te zien als je het venster breder maakt, wat in het onderste plaatje is weergegeven. We bekijken nog een voorbeeld:

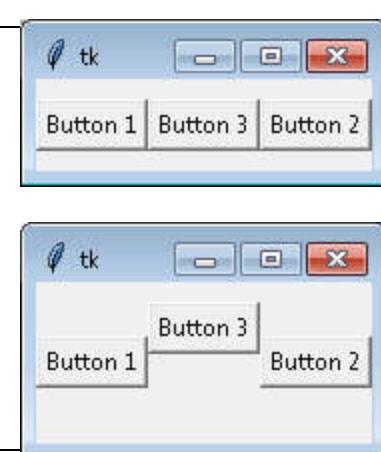
```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.pack(side=LEFT, pady=10)

button2 = Button(master=root, text='Button 2')
button2.pack(side=RIGHT, pady=10)

button3 = Button(master=root, text='Button 3')
button3.pack(side=TOP, pady=10)

root.mainloop()
```



Omdat er met de functie pack() slechts 4 verschillende posities mogelijk zijn, kan het soms wat lastig kan worden als je heel veel widgets wilt plaatsen. Stel dat je bijvoorbeeld twee buttons aan de linkerkant van het scherm wilt plaatsen, maar je wilt ze onder elkaar hebben. Dan kan je niet zeggen dat beide buttons links moeten komen te staan, want dan zullen ze automatisch naast elkaar geplaatst worden. Een oplossing kan dan zijn om een **Frame** te gebruiken!

Een Frame is als het ware een **container** waarin je weer andere widgets kunt plaatsen. Die container kan je dan als één element op het hoofdscherm plaatsen. Je kunt dan aangeven of de container LEFT, RIGHT, TOP of BOTTOM in het scherm moet komen. Binnen de container kan je ook weer een onderverdeling in LEFT, RIGHT, TOP of BOTTOM maken.

In onderstaande code is bijvoorbeeld een linkerframe gemaakt, waarin twee buttons zijn opgenomen. De buttons zijn zonder **side** parameter, en dat betekent dat ze bovenin de container geplaatst worden. Merk op dat de **master** van button1 en button2 dus **linkerframe** is. Het frame heeft zelf weer als master **root**, en wordt daarin aan de linkerkant geplaatst. Tot slot is er nog een button3, die rechts in het hoofdscherm is opgenomen:

```
from tkinter import *
root = Tk()

linkerframe = Frame(master=root)
linkerframe.pack(side=LEFT)

button1 = Button(master=linkerframe, text='Button 1')
button1.pack(pady=4)

button2 = Button(master=linkerframe, text='Button 2')
button2.pack(pady=4)

button3 = Button(master=root, text='Button 3')
button3.pack(side=RIGHT, pady=4)

root.mainloop()
```



Met een beetje nadenken kan je met de functie pack() een heel eind komen, en hier heb je dan ook waarschijnlijk genoeg aan voor een relatief eenvoudige GUI. Mocht dat niet het geval zijn, dan zijn er nog enkele alternatieven, die we kort in stap 4\_2 en 4\_3 bespreken.

### Stap 4\_2: Layoutmanagers: grid()

Als je niet voldoende hebt aan pack(), dan biedt tkinter ook nog de **grid()** functie. Hiermee kan je het hele scherm in kolommen en rijen verdelen, waardoor een soort tabel met cellen ontstaat. Voor elke widget geef je dan aan in welke cel deze moet komen te staan. Een voorbeeld:

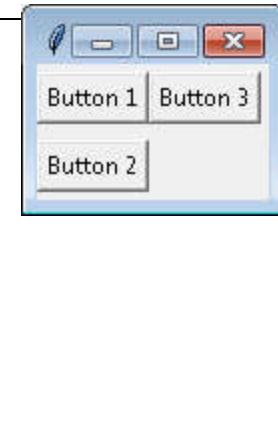
```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.grid(row=0, column=0, pady=4)

button2 = Button(master=root, text='Button 2')
button2.grid(row=1, column=0, pady=4)

button3 = Button(master=root, text='Button 3')
button3.grid(row=0, column=1, pady=4)

root.mainloop()
```



Door het gebruik van de functie **grid()** rekent tkinter zelf uit hoeveel rijen en kolommen er zijn. Dat hoef je verder nergens op te geven. We bespreken deze functie hier verder niet, maar Perkovic behandelt de **grid()** functie in §9.2! Het is belangrijk om te beseffen dat je **pack()** OF **grid()** moet gebruiken, niet beiden!!

### Stap 4\_3: Layoutmanagers: place()

Als je dit allemaal maar ingewikkeld vindt, dan kan je eventueel nog terugvallen op de **place()** functie. Deze komt niet aan de orde in Perkovic, en is ook niet erg efficiënt, maar kan in sommige gevallen uitkomst bieden. Hierbij kan je de exacte coördinaten (x,y) van een widget opgeven. Hierbij loopt de x-as van linksboven naar rechtsboven, en de y-as van linksboven naar linksonder:

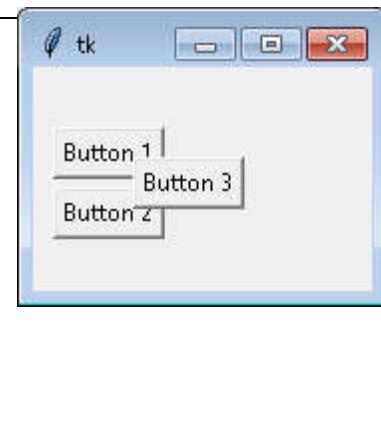
```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.place(x=10, y=30)

button2 = Button(master=root, text='Button 2')
button2.place(x=10, y=60)

button3 = Button(master=root, text='Button 3')
button3.place(x=50, y=45)

root.mainloop()
```



Zoals je kunt zien, kan je hierbij dus widgets (per ongeluk) over elkaar heen plaatsen. Daarnaast is deze oplossing niet erg schaalbaar, want als het scherm breder of hoger zou worden, blijven de widgets exact op dezelfde positie staan. Hier is binnen de functie **place()** wel iets aan te doen, maar ook dat valt buiten de scope van deze tutorial. Deze methode is echter wel toegestaan bij het miniproject!

### Stap 5: Pop-up vensters (optioneel)

Mocht je de gebruiker een melding willen geven, dan kan je daarvoor het beste een Label gebruiken. In sommige gevallen is het echter mogelijk dat dit niet genoeg opvalt. Je kunt dan eventueel een popup-venster gebruiken:

```
from tkinter import *
from tkinter.messagebox import showinfo

def clicked():
    bericht = 'Dit een bericht voor de gebruiker!'
    showinfo(title='popup', message=bericht)

root = Tk()

button = Button(master=root, text='Druk hier', command=clicked)
button.pack(pady=10)

root.mainloop()
```



### Stap 6: Sub-vensters (optioneel)

Zeer waarschijnlijk heb je bij het miniproject ruimschoots voldoende aan de voorgaande voorbeelden. Maar het kan natuurlijk zijn dat je in de GUI van jullie miniproject ook een tweede venster wilt openen. Je kunt dan gebruikmaken van de **Toplevel** widget:

```
from tkinter import *

def toonVenster():
    def close():
        subwindow.withdraw()

    subwindow = Toplevel(master=root)
    button2 = Button(master=subwindow, text='Sluit mij', command=close)
    button2.pack(padx=10, pady=10)

root = Tk()

button1 = Button(master=root, text="Druk hier", command=toonVenster)
button1.pack()

root.mainloop()
```

Je ziet hier een gewone *root*, met daarop een knop. Als je op die knop klikt, wordt de functie **toonVenster()** uitgevoerd. Deze functie beschrijft eerst een nieuwe functie, en creëert daarna een Toplevel widget (een nieuw scherm). Aan deze Toplevel is 1 knop toegevoegd, (gekoppeld met **master=subwindow**). Als iemand op die knop klikt, sluit het subvenster met behulp van de opdracht **subwindow.withdraw()**.

### Stap 7: Python scripts aanroepen

Het is verstandig om scripts waarin je een GUI opbouwt, niet te mengen met de scripts waarin je berekeningen en andere logica hebt geprogrammeerd. In Les 3 heb je bijvoorbeeld een functie moeten programmeren waarin de standaardprijs van een treinrit werd uitgerekend. Stel dat je die hebt opgeslagen in het bestand '**Treinfuncties.py**', dat er als volgt uitziet:

```
def standaardprijs(afstandKM):
    if afstandKM < 0:
        afstandKM = 0

    if afstandKM < 50:
        prijs = afstandKM * 0.80
    else:
        prijs = 15 + afstandKM * 0.60

    return prijs
```

We gaan ervan uit dat dit bestand in dezelfde directory staat als jouw GUI-script, en dat beide bestanden in de hoofddirectory van het project staan. Dan kan je deze functies importeren, en gebruiken in het GUI-script:

```
from tkinter import *
from Treinfuncties import *

def berekenTarief():
    afstand = afstandEntry.get()
    prijs = int(standaardprijs(afstand))
    label["text"] = "De ritprijs is: {}".format(prijs)

root = Tk()

afstandEntry = Entry(master=root)
afstandEntry.pack()

button = Button(master=root, text="Druk hier", command=berekenTarief)
button.pack()

label = Label(master=root)
label.pack()

root.mainloop()
```

**Let op:** als de bestanden in een subdirectory van je project staan, dan moet je op de tweede regel van het bovenstaande script dus het volledige path naar het betreffende Python-script opnemen, zodat Python dat script kan vinden als je jouw programma uitvoert.

### Stap 8: Aan de slag (optioneel)

Maak een GUI waarin je de gebruiker een stationsnaam laat invoeren (Entry). Nadat de gebruiker op OK klikt (Button), worden de eerstvolgende vijf vertrektijden vanaf dat station getoond (5x Label). Neem hiervoor de code die je in les 9, tutorial 1 hebt gemaakt, als uitgangspunt. Als je deze opdracht succesvol weet af te ronden, ben je meer dan voorbereid op het miniproject! Het is ook niet erg als dat niet helemaal lukt, want tijdens het miniproject kan je jouw teamleden om hulp vragen!