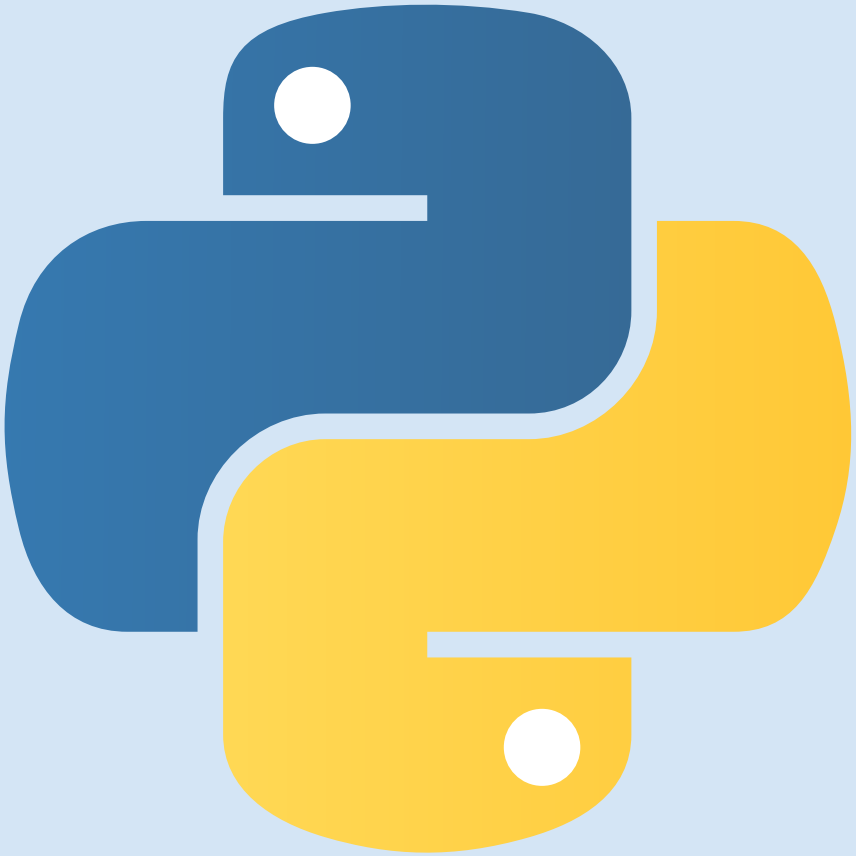


Python logo

Programiranje 2

tkinter

Vladimir Batagelj
Univerza v Ljubljani
FMF, matematika

različica: 25. februar 2014 / 03 : 37

Kazalo

Slikovni vmesniki

Sprva se je na računalnikih uporabljal *ukazni način dela*

```
imeIzvršljiveDatoteke parametri
```

pogojen s prejšnjo tehnologijo – kartice, teleprinterji, znakovni zasloni.

V drugi polovici osemdesetih let so se začeli uveljavljati *slikovni vmesniki* (GUI - graphic user interface), ki so jih omogočili slikovni zasloni in miška.

Gem, Windows

To je prineslo precejšne spremembe v delu z računalnikom

tipkanje	izbira z miško
spominjanje	prepoznavanje
vgrajeni znaki	pisave

kar je omogočilo nove, drugačne in lažje uporabe računalnika. Prepoznavanje je precej manj zahtevno od spominjanja.

Slikovni vmesniki

Stvari, ki so že bile enkrat vnešene v računalnik, ni (naj ne bi bilo) potrebno ponovno vnašati – manj napak.

Namizno založništvo.

Spletni pregledovalnik lahko uporablja skoraj vsak – ni potrebno biti računalniški strokovnjak.

Če želimo pognati nek program, dvokliknemo na pripadajočo mu ikono. V programu pripadajočem oknu izvemo kaj vse lahko počnemo. Na sodobnih računalnikih lahko sočasno teče več programov.

V nadaljevanju si bomo ogledali, kako uporabljamo slikovni vmesnik v Pythonu.

Python in slikovni vmesniki

Python ima vgrajen preprost slikovni vmesnik v knjižnici `tkinter`, ki je pravzaprav sposojena iz sestava Tcl TK (tikl) – splošni skriptni jezik in slikovni vmesnik. Pred kratkim (2009) smo prešli na Python 3. V Pythonu 2 ima knjižnica ime `Tkinter`.

Obstajajo še druge, zmogljivejše knjižnice s slikovnimi vmesniki, ki so prirejene tudi za uporabo v Pythonu. Najbolj priljubnljeni sta PyQt in wxPython.

Mi bomo ostali pri knjižnici tkinter, ker ne zahteva nobenih dodatnih nameščanj. Poleg tega je pred kratkim izšla različica Tcl Tk 8.5, ki posodablja izgled sestavin, jih prilagaja OS in tudi razširja njihov nabor: Tcl Tk, 8.5.

V Python 3.2 so dostopne v podknjižnici `tkk`: Tk, tkinter, ttk, tix, py-ttk.

Razširimo ju z v Pythonu napisanima knjižnicama `Tix`: tix, pytix, demo in `Pmw` – Python megawidgets (nekoliko zastarela): pmw, Widgets.

Slikovni vmesniki

Pri delu s tipkovnico pošljemo običajno ukaz programu v računalniku s pritiskom na tipko ENTER.

Pri slikovnih vmesnikih prožimo dejavnosti z raznimi dogodki na zaslonu (pa tudi drugimi). To zahteva poseben način izvajanja/programiranja – *dogodkovno programiranje*.

Nadzorni programa neprestano preverja, ali se je pojavil nov dogodek. Ko se ta pojavi, preveri ali morda zahteva zaključek izvajanja; če ga, konča. Sicer opravi dogodku ustrezno opravilo in se vrne v čakanje na nov dogodek.

Poglejmo si prvi program (shranjen na datoteki `tk01.pyw`).

Pozor!!! Uporabljamo podaljšek `pyw`, ki zažene Python nekoliko drugače.

Prvi program

```
from tkinter import *
from tkinter import ttk
koren = Tk()
button = ttk.Button(koren, text="Dober dan").pack()
koren.mainloop()
```

Prvi dve vrstici programa povesta Pythonu, da program potrebuje dva modula (knjižnici). Modul `tkinter` vsebuje povezavo s knjižnico `Tk`; modul `ttk` pa vsebuje novejše *gradnike* (widgets), ki so bili vključeni šele v `Tk 8.5`.

Iz modula `tkinter` smo vnesli vse – njegove funkcije lahko kličemo brez začetne navedbe modula. Nasprotno pa bomo morali vse funkcije iz modula `ttk` začenjati s `ttk.` . Ker je več funkcij definiranih v obeh modulih in občasno potrebujemo eno ali drugo, nam ta način poimenovanja omogoča razmeroma enostavno uporabo obeh.

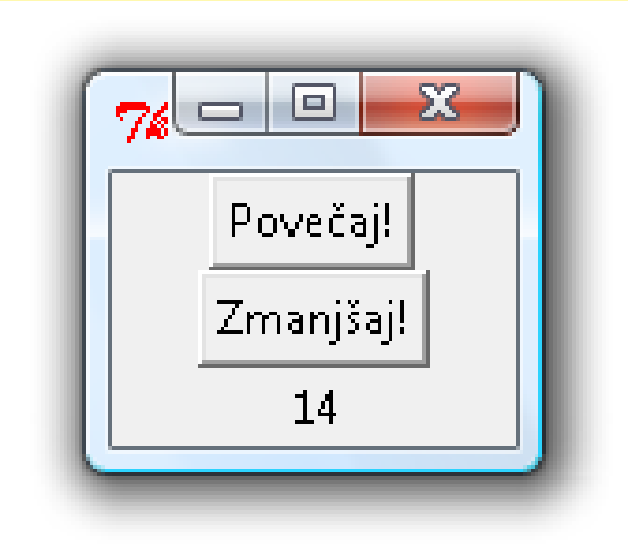
`koren` – predstavlja glavno okno programa. Čakalno zanko zapustimo, ko izberemo zaprtje okna.

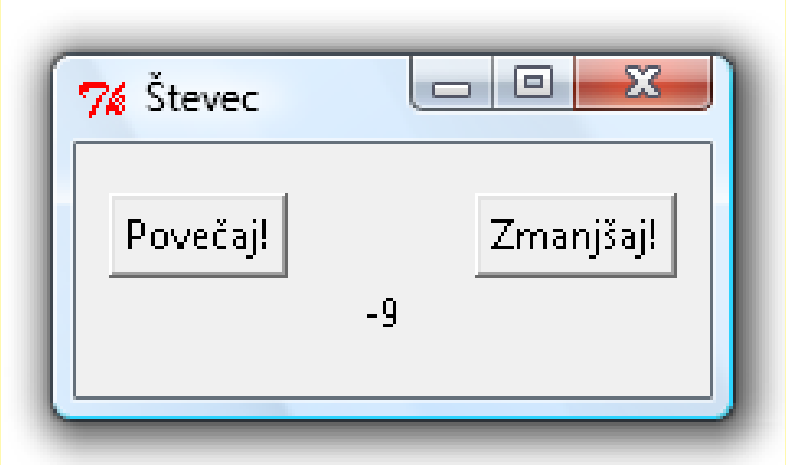
Korenu običajno damo okvir, na katerega zlagamo druge gradnike. Vsakemu dodanemu gradniku moramo določiti starša – nadrejeni gradnik. Tako dobimo drevo gradnikov.

Za vsak gradnik moramo določiti tudi razmestitev – kje bo v vmesniku postavljen. Prilagodljivost velikosti okna (spreminjanje velikosti); lep, urejen izgled; presledki, ...

`pack`, `grid` in `place` so metode za namestitev gradnika v prikazu. `pack` preprosto zlaga gradnike, kot mu jih dostavljamo; `grid` omogoča raporejanje po mreži; še natančnejše razmeščanje omogoča `place`.


Dober dan


Povečaj!
Zmanjšaj!
14


Števec
Povečaj!
Zmanjšaj!
-9

Števec: tk02.pyw

```
# -*- encoding: utf-8 -*-
# Program, ki odpre okno, ter vanj postavi dva gumba in števec

from tkinter import *

class Stevec():
    def __init__(self, glavnoOkno):
        '''V okno postavi dva gumba in števec.'''

        # Stanje števca je na začetku 0
        self.stevec = 0

        # najprej ustvarimo sestavino Frame,
        # v katero postavimo gumbe in napise
        okvir = Frame(glavnoOkno)   # ta okvir je vsebovan v glavnemOknu
        okvir.pack()                # to dejansko naredi vsebino vidno

        gumb_povecaj = Button(okvir, text="Povečaj!", command=self.povecaj)
        gumb_povecaj.pack()

        gumb_zmanjsaj = Button(okvir, text="Zmanjšaj!", command=self.zmanjsaj)
        gumb_zmanjsaj.pack()

        self.niz_stevec = StringVar(value=str(self.stevec))
        napis_stevec = Label(okvir, textvariable=self.niz_stevec)
        napis_stevec.pack()

    def povecaj(self):
        '''Povečaj števec za 1.'''
        self.stevec = self.stevec + 1
        self.niz_stevec.set(str(self.stevec))

    def zmanjsaj(self):
        '''Zmanjšaj števec za 1.'''
        self.stevec = self.stevec - 1
        self.niz_stevec.set(str(self.stevec))

koren = Tk()
stevec = Stevec(koren)
koren.mainloop()
```

Števec z grid: tk03.pyw

```
# -*- encoding: utf-8 -*-
# Program, ki odpre okno, ter vanj postavi dva guma in števec

from tkinter import *

class Stevec():
    def __init__(self, glavnoOkno):
        '''V okno postavi dva gumba in števec.'''

        # Stanje števca je na začetku 0
        self.stevec = 0

        okvir = Frame(glavnoOkno)
        okvir.grid(column=0, row=0, padx=10, pady=15)

        gumb_povecaj = Button(okvir, text="Povečaj!", command=self.povecaj)
        gumb_povecaj.grid(row=0, column=0)

        gumb_zmanjsaj = Button(okvir, text="Zmanjšaj!", command=self.zmanjsaj)
        gumb_zmanjsaj.grid(row=0, column=2)

        self.niz_stevec = StringVar(value=str(self.stevec))
        napis_stevec = Label(okvir, textvariable=self.niz_stevec)
        napis_stevec.grid(row=2, column=1, padx=20)

    def povecaj(self):
        '''Povečaj števec za 1.'''
        self.stevec = self.stevec + 1
        self.niz_stevec.set(str(self.stevec))

    def zmanjsaj(self):
        '''Zmanjšaj števec za 1.'''
        self.stevec = self.stevec - 1
        self.niz_stevec.set(str(self.stevec))

koren = Tk()
koren.title("Števec")
stevec = Stevec(koren)
koren.mainloop()
```

Osnove

Slikovni vmesnik je sestavljen iz gradnikov kot so: okna, gumbi, napisi, vnosna polja, drsniki, sličice, risalne ploskve, ... Na nekatere gradnike lahko postavimo druge, podrejene gradnike. Tako ustvarimo hierarhijo gradnikov. V njenem vrhu je *koren*ski gradnik. Vsi gradniki, razen korena, imajo prednika, ki ga ob ustvaritvi gradnika navedemo kot prvi argument.

```
koren = Tk()
okno = ttk.Frame(koren, ...)
gumb = ttk.Button(okno, ...)
```

Lastnosti gradnikov

Lastnosti gradnikov določajo njihov izgled in obnašanje.

Za dani gradnik g dobimo slovar vseh njegovih lastnosti z zahtevo `g.configure()` . Npr.

```
gumb.configure()
```

oziroma posamezno lastnost p z `g.configure('p')` .

Za izbrano lastnost (ključ) so sestavine slovarskega nabora (ime lastnosti, razred lastnosti, vgrajena vrednost, tekoča vrednost).

Vrednost lastnosti p pa z `g['p']`. Npr.

```
gumb['text']
```

Vrednost lastnosti spremenimo z `g['p']=v` , kjer je v nova vrednost. Spremembo dosežemo tudi z zahtevo `g.configure(p=v)`

vrednosti

Razmeščanje gradnikov

Za postavitev gradnika na ustreznem mestu v nadrejenem gradniku poskrbimo tako, da uporabimo nad gradnikom eno od namestitvenih metod. `Tk` jih pozna več: `pack`, `grid` in `place`.

Težave z nameščanjem gradnikov izhajajo iz:

- gradniki so različno veliki;
- pri spremembi velikosti nekega gradnika se morajo ostali ustrezno prilagoditi;
- poravnavanje in raztezanje/krčenje gradnikov.

Za razmeščanje podrejenih gradnikov v istem gradniku uporabljamo isto namestitveno metodo.

`pack` in `grid`

Packer

Dogodki

Delovanje slikovnega vmesnika je določeno z odzivi na *dogodke*, ki jih proži uporabnik ali programi: pritisk na gumb, pritisk na tipko na tipkovnici, premik miške, sprememba velikosti okno, ... Za zaznavanje dogodkov skrbi operacijski sistem.

Delovanje `Tk` temelji na dogodkovni zanki, ki poskrbi za ustrezno odzivanje na dogodke: če je za tekoči (dejavni) gradnik dogodek predviden, `Tk` sproži ustrezno dejavnost, ki je praviloma določena s *podporno* (callback) funkcijo, ki jo sestavi uporabnik – tekoči gradnik pokliče nazaj funkcijo z uporabniške ravni. To funkcijo običajno določimo z lastnostjo `command`.

Dogodke, za katere niso določene podporne funkcije v gradnikih, lahko prestrežemo z vezavo `bind`.

bind

`Tk`-jeve spremenljivke

V `Tk`-ju ne moremo uporabljati naravnost Pythonskih spremenljivk. V gradnikih so dovoljene le posebne `Tk`-jeve spremenljivke, ki so lahko naslednjih zvrsti `StringVar`, `IntVar`, `BooleanVar` in `DoubleVar`. Z gradnikom jih povežemo z lastnostmi `variable`, `textvariable`, `onvalue`, `offvalue` in `value`. Sprememba vrednosti spremenljivki se takoj odrazi na ustrezni spremembi na gradniku in obratno.

Vrednost spremenljivke dobimo z metodo `get`, postavimo pa jo z metodo `set`.

Samostojni gradniki

`Tk` pozna tudi nekaj samostojnih gradnikov, ki omogočajo izpis raznih sporočil in vnos podatkov.

```
from tkinter import messagebox
from tkinter import simpledialog
m1=messagebox.askyesno("naslov","si zadovoljen?")
m2=messagebox.askokcancel("naslov","si zadovoljen?")
m3=messagebox.askquestion("naslov","si zadovoljen?")
m4=messagebox.askretrycancel("naslov","si zadovoljen?")
m5=messagebox.showerror("naslov","pazi na velikost")
m6=messagebox.showinfo("naslov","pazi na velikost")
m7=messagebox.showwarning("naslov","pazi na velikost")
af=simpledialog.askfloat("naslov","velikost")
ai=simpledialog.askinteger("naslov","velikost")
at=simpledialog.askstring("naslov","ime")
```

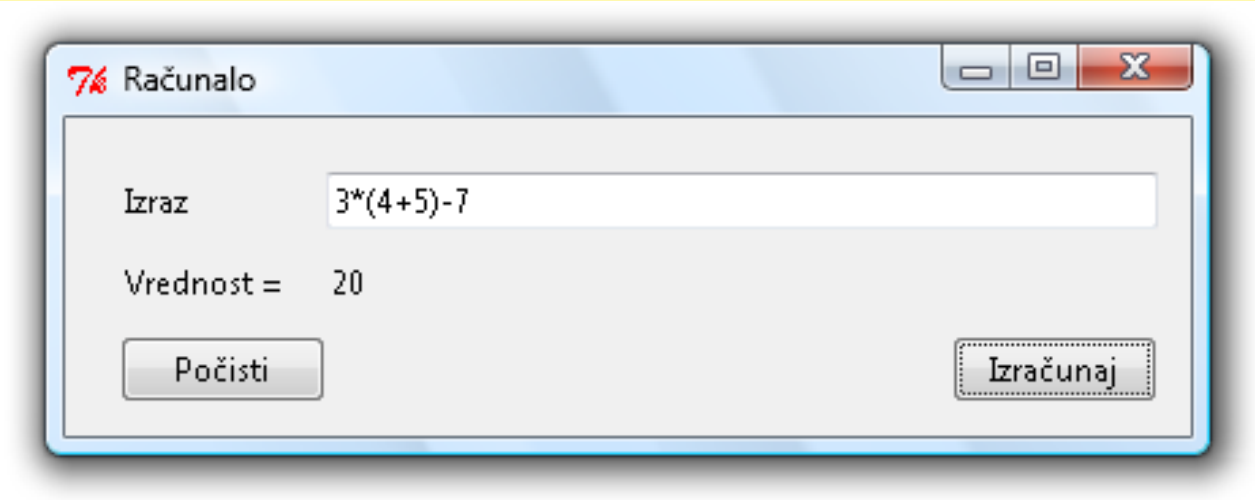
pogovorna okna

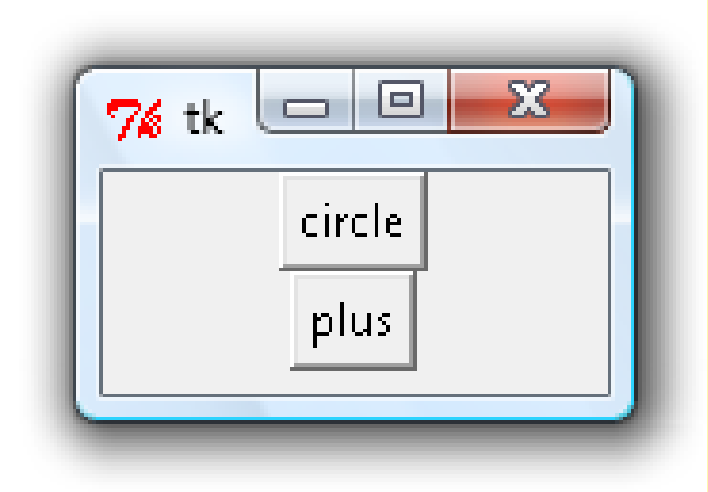
Samostojni gradniki

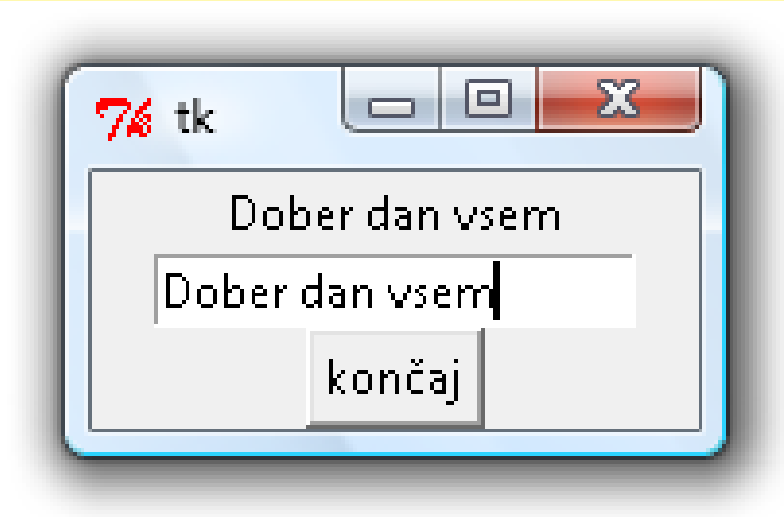
```
from tkinter import filedialog
from tkinter import colorchooser
pics=filedialog.askdirectory(title="Slike",initialdir='/')
print(pics)
slike = [
  ('Windows Bitmap','*.bmp'),
  ('Portable Network Graphics','*.png'),
  ('JPEG / JFIF','*.jpg'),
  ('CompuServer GIF','*.gif'),
]
dat=filedialog.askopenfilename(title="Slike",initialdir=pics,
      filetypes=slike)
print(dat)
shrani=filedialog.asksaveasfilename(title="Shrani",initialdir=pics)
print(shrani)
# askopenfile    mode="rb"
# askopenfilenames
barva=colorchooser.askcolor(initialcolor='#ff0000')
print(barva)
```

Računalo: tk04.pyw

```
# -*- encoding: utf-8 -*-
# računalo
# izračuna in prikaže vrednost aritmetičnega izraza, ki ga vnesemo

from tkinter import *
from tkinter import ttk
from math import *

def izracunaj(*args):
    try:
        value = eval(izraz.get())
        vrednost.set(value)
    except Exception:
        messagebox.showerror("Napaka", "Napačen izraz")
        pocisti()

def pocisti():
    izraz.set("")
    vrednost.set("")
    izraz_entry.focus()

koren = Tk()
koren.title("Računalo")

okno = ttk.Frame(koren, padding="20 20 12 12")
okno.grid(column=0, row=0, sticky=(N, W, E, S))

izraz = StringVar()
vrednost = StringVar()

ttk.Label(okno, text="Izraz").grid(column=0, row=0, sticky=W)
izraz_entry = ttk.Entry(okno, width=50, textvariable=izraz)
izraz_entry.grid(column=1, row=0, sticky=(E,W))

ttk.Label(okno, text="Vrednost = ").grid(column=0, row=1, pady=10, sticky=W)
ttk.Label(okno, textvariable=vrednost).grid(column=1, row=1, sticky=(E,W))

ttk.Button(okno, text="Počisti", command=pocisti).grid(column=0, row=3, sticky=W)
ttk.Button(okno, text="Izračunaj", command=izracunaj).grid(column=1, row=3, sticky=E)

izraz_entry.focus()
koren.bind('<Return>', izracunaj)

koren.mainloop()
```


Računalo
Izraz
3*(4+5)-7
Vrednost =
20
Počisti
Izračunaj


tk
circle
plus


tk
Dober dan vsem
Dober dan vsem
končaj

Gumba: tk05.pyw

```
from tkinter import *
top = Tk()
B1 = Button(top, text ="circle", relief=RAISED,cursor="circle")
B2 = Button(top, text ="plus", relief=RAISED,cursor="plus")
B1.pack()
B2.pack()
top.mainloop()
```

Sestavine: tk06.pyw

```
from tkinter import *
koren = Tk()
s = StringVar()
Label(textvariable=s).pack()
Entry(textvariable=s).pack()
s.set('Pozdravljeni!')
Button(text="končaj",command=koren.quit).pack()
mainloop()
print("s = ",s.get())
try: koren.destroy()
except: pass
```

Barva ozadja: tk07.pyw

```
from tkinter import *
from colorchooser import askcolor

def setBgColor():
    (rgb, hexstr) = askcolor()
    if hexstr:
        print(hexstr)
        klik.config(bg=hexstr)

koren = Tk()
klik = Button(koren, text='Izberi barvo ozadja', command=setBgColor)
klik.config(height=3, font=('times', 20, 'bold'))
klik.pack(expand=YES, fill=BOTH)
koren.mainloop()
```

Vezave dogodkov

Povezavo dogodka in ustrezne podporne funkcije lahko opravimo na različnih ravneh:

- programska: povezava velja za vse dejavne gradnike v programu;
- razredna: povezava velja za vse primerke gradnika (do preklica);
- korenska: povezava velja za Toplevel ali korenski gradnik;
- posamična: povezava velja za izbrani gradnik.

Metode `bind`, `bind_all`, `bind_class`. Z metodami `unbind`, `unbind_all`, `unbind_class` lahko povezave tudi razdremo.

Dogodek s tipkovnice ali miši s tekočega gradnika potuje proti korenu dokler ne naleti na gradnik, ki ga prestreže (opravi ustrezno opravilo ali odvrže).

Ura: `tk09.pyw`

Časovno nadzorovano ponavljanje nam omogoča metoda g`.after(`t`,`a`)`, ki nastavi "štoparico"na gradniku g in po preteku časa t sproži podporno funkcijo a.

```
from tkinter import *
import time

sedaj = ''
ura = Label()
ura.pack()

def tik():
  global sedaj
  novi = time.strftime('%H:%M:%S')
  if novi != sedaj:
    sedaj = novi
    ura.config(text=sedaj)
  ura.after(200,tik)
tik()
ura.mainloop()
```

Igre

Igra je določena s *pravili*. S pravili je določen tudi *cilj* igre.

Število igralcev:

- en igralec (solitaire): Tetris, prestavljanke, sudoku, ...
- dva igralca: šah, križci in krožci, štiri v vrsto, potapljanje ladjic, hex, mora, ...
- več igralcev: človek ne jezi se, tarok, ...

Igra poteka tako, da igralci v okvirih pravil spreminjajo stanja. Te spremembe so lahko zvezne ali diskretne.

Diskretnim spremembam rečemo poteze. Pri več igralcih so lahko igralci izbirajo poteze vzporedno (sočasno) ali zaporedno. Pri zaporednem izbiranju je s pravili določen tudi vrstni red igralcev.

Računalniška izvedba igre lahko poteka na enem ali več računalnikih.

Sestavine igre

Sestavine računalniške izvedbe igre:

- izpis navodil
- izbira lastnosti igre / igralca
- podatki o programu (avtorji, različica, (c),...)
- prekinitev/nadaljevanje
- program glede na pravila igre preverja igralčeve izbire (poteze)
- beleži zgodovino (shrani, preklic nekaj zadnjih potez, ...)
- točkovanje (Hall of fame)
- pomočnik (svetovalec)
- avtomatični igralci
- zvočna oprema