Fakultet Elektrotehnike, Računarstva i Informacijskih Tehnologija Osijek
Kneza Trpimira 2b
31000 Osijek

www.ferit.unios.hr

Python – Klase i nasljeđivanje

Klase su mehanizam stvaranja novih tipova objekata. Predstavljaju shemu po kojoj će se stvarati ostali objekti. Klase su u Pythonu objekti prvog reda što znači da se s njima može manipulirati (modificirati, prosljeđivati i sl.). Skupina objekata koji su stvoreni pomoću klase i koje stoga imaju iste karakteristike se nazivaju *instance* te klase. Funkcije koje pripadaju nekoj instanci se nazivaju *metode*, a varijable neke instance se nazivaju *atributi*. Klase se stvaraju naredbom ***class***.

```
class ImeKlase(NadKlasa1, NadKlasa2, ...):          #zaglavlje
    atribut1 = 1234                                 #inicijalizacija atributa
    atribut2 = 'tekstualni atribut'
    def __init__(self, atrib1, atrib2):             #"konstruktor"
        self.atribut1 = atrib1                      #inicijalizacija konstruktorom
        self.atribut3 = open(atrib2)                #stvaranje novog atributa (datoteke)
    def __del__(self):                              #"destruktor"
        self.atribut3.close()
    def Metoda1(self):                              #metoda
        print('Ovo je metoda 1!')
    def Metoda2(self, a):
        return self.atribut1 / a
```

Zaglavlje klase definira naziv klase i u zagradi nabrojane klase koje nasljeđuje. Atributi klase se mogu definirati u tijelu klase, a svaka instanca te klase će preuzeti atribute s njihovim zadanim vrijednostima. Atributi instance se mogu stvarati dinamički, pa tako i prilikom inicijalizacije koju provodi „konstruktor" (inicijalizator) *__init__*. Inicijalizator služi za postavljanje atributa nove instance na neke željene početne vrijednosti ali on ne mora biti implementiran. Prilikom brisanja instance neke klase poziva se „destruktor" *__del__* koji također nije obavezan i uglavnom se ne koristi osim u iznimnim situacijama (kontrolirano zatvaranje datoteke, prekidanje komunikacije, i sl.). Prvi parametar svake metode je referenca na objekt koji poziva tu metodu. Po konvenciji se on naziva *self* ali može biti bilo koje drugo ime i ima istu svrhu kao objekt *this* u sistemskim jezicima (C++, C#). Iako Python definira prostor imena nad klasom, atributi i metode klase nisu izravno vidljivi unutar metoda iste klase već se njima mora pristupiti preko reference *self*.

Nasljeđivanje je mehanizam stvaranje nove klase koja dodatno specijalizira ili modificira ponašanje neke postojeće klase (omogućava iskorištavanje postojećeg koda). Originalna klasa se naziva osnovna klasa (engl. base class) ili nadklasa (engl. superclass), a nova klasa se naziva izvedena klasa (engl. derived class) ili podklasa (engl. subclass). Python ne podržava privatne varijable stoga izvedena klasa preuzima sve metode i atribute osnovne klase, ali im može promijeniti vrijednosti/ponašanje ili definirati nove atribute i metode. Prilikom pretrage

za traženim atributom ili metodom Python prvo traži u lokalnom prostoru imena trenutne izvedene klase (preko imena *self*) , a ako ne pronađe traži više u hijerarhiji nasljeđivanja, u prostorima imena nadklasa. U slučaju višestrukog nasljeđivanja stvara se stablo pretraživanja. Prilikom dizajniranja velike hijerarhije klasa koja traže interoperabilnost s ostalim klasama Python programskog jezika, na vrhu hijerarhije se tipično postavlja Pythonova ***object*** klasa.
Primjer hijerarhije klasa:

```
class A:
    a = 0
    def FunA(self):
    print self.a, 'FunA!'
class B(A):
    b = 0
    def __init__(self):
        print 'Konstruktor B'
class C(B):
    a = 4                       #mijenja atribut koji je naslijedio od nadklase
    def FunA(self):             #mijenja naslijeđenu metodu
        print 'FunA - verzija C'
        A.FunA(self)            #pokreće odgovarajuću metodu nadklase
```

Primjer korištenja hijerarhije klasa:

```
>>> o4 = Osoba.C()
Konstruktor B          #poziva prvi konstruktor koji je dostupan u hijerarhiji
>>> o4.FunA()
FunA - verzija C
4  FunA!
```

U slučaju implementacije atributa ili metode jednakog naziva kao i u nadklasi trajno se mijenja njegova vrijednost/funkcionalnost u izvedenoj klasi, ali se originalnoj vrijednosti/metodi može pristupiti putem naziva nadklase. Dohvaćanje funkcionalnosti nadklase se može postići i bez poznavanja njenog imena putem funkcije ugrađene klase ***super***.

Preporučeno je grupiranje korisničkih funkcija i klasa u prostore imena koje se potom mogu uključiti kao moduli u ostale programe (iskorištavanje postojećeg koda). Ako se korisničke funkcije, klase i ostala imena napišu i pohrane u datoteku npr. *MojLib.py* tada se u drugom programu ili konzoli mogu učitati kao modul preko *import MojLib* naredbe. Preko operatora '.' se tada mogu dohvatiti funkcije/klase iz prostora imena npr. *MojLib.MojaFunkcija()*.
Datoteke se naravno mogu pokrenuti i kao programi (skripte) putem funkcije ***runpy.run_path*** ili ***runpy.run_module*** u konzoli ili izvan u command prompt-u (*python MojLib.py*). Kada se neki kod napisan u .py datoteci pokreće na takav način (preko command prompt-a) tada Python postavlja privatnu varijablu *__name__* (koja predstavlja naziv lokalnog prostora imena) na vrijednost *'__main__'* što sugerira da se datoteka koristi kao program/skripta, a ne kao modul. Ako se neka datoteka učita u neki program preko *import* naredbe tada objekt tog modula ima naziv kao i modul (ime datoteke bez ekstenzije) npr. *MojLib.__name__* → *'MojLib'*.
Takav način imenovanja otvara mogućnost da se neka datoteka s kodom po potrebi pokrene kao program ili učita kao modul. Kada se učita neki modul preko datoteke, tada se izvrše sve naredbe napisane u nekoj datoteci, a ako se želi da se neke naredbe izvrše samo ako se

datoteka izvršava kao program, a ne kao modul, tada sve što je potrebno jest u datoteku dodati dio koda koji provjerava koje je trenutno „ime“, npr.

```
if __name__ == '__main__':  #Ako je naše lokalno „ime“ '__main__' znači da je datoteka
    root = Tk.Tk()          #pokrenuta kao program (runfile), a ne kao modul (import)
    app = Application(root) #Jedino u tom slučaju pokrećemo našu aplikaciju
    root.mainloop()
    root.destroy()
```

Uključivanjem naredbi u ***if*** blok koda koji provjerava trenutni naziv prostora imena, omogućava nam da funkcije i klase napisane kao programska podrška aplikaciji pohranjenoj u datoteci koristimo i u ostalim programima bez da narušimo integritet novih programa koji će učitavati postojeći kod.

Python – Tkinter

Pythonova ugrađena podrška za programiranje grafičkih sučelja se nalazi u sklopu *tkinter* prostora imena. Tkinter je u biti Python omotač (engl. wrapper) za Tk biblioteku razvijenu za skriptni jezik Tcl. Zasniva se na prevođenju Python naredbi u Tcl i pokretanju Tcl interpretera, time je omogućeno kombiniranje Python i Tcl koda u isto vrijeme. Nudi prirodni izgled aplikacije koja se pokreće na pojedinoj platformi koristeći stilove i teme. Tada se može koristiti i dodatni *ttk* prostor imena. Izgradnja sučelja se svodi na stvaranje hijerarhije kontrola (engl. widget), a na vrhu hijerarhije se nalazi glavni prozor aplikacije koju stvara *tkinter.**Tk*** klasa.
Tkinter aplikacija se može stvarati izvršavanjem naredbu po naredbu kojim se slaže sučelje, ali se tipično sučelja programa ugrađuju u korisničke klase radi bolje preglednosti i objektno orijentiranom pristupu razvoja aplikacije.
Primjer osnovnog korištenja preko korisničke klase:

```
from tkinter import *
class Application(Frame):               #nasljeđujemo tkinter.Frame (može biti i tkinter.Tk)
    def Pozdrav(self):
        print "Pozdrav svima!!"
    def CreateWidgets(self):
        self.lab = Label(self, text = 'Aplikacija!') #stvaramo labelu
        self.lab.pack()                 #postavljamo labelu na određeno mjesto sučelja
        self.tipka = Button(self, text = 'Pozdrav', command = self.Pozdrav) #stvaramo tipku
        self.tipka.pack()               #postavljamo tipku na određeno mjesto sučelja
    def __init__(self, master = None):
        Frame.__init__(self, master)    #inicijaliziramo originalni Frame
        self.pack()                     #postavljamo Frame kontrolu na glavni prozor
        self.CreateWidgets()            #postavljamo naše ostale kontrole

root = Tk()
app = Application(root)                 #stvaramo našu klasu
app.mainloop()                          #pokrećemo grafičko sučelje
```

Parametri pojedinih kontrola se mogu postavljati na tri načina:

- Prilikom stvaranja instance kontrole: *lab = Label(root, text = 'Aplikacija!', bg = 'red')*
- Referenciranjem parametra putem ključa: *lab['text'] = 'Aplikacija 2'; lab['bg'] = 'red'*
- Korištenjem metode *config*: *lab.config(text = 'Aplikacija!', bg = 'red')*

Mnoge kontrole imaju slične parametre:

- *bg, fg* ili *background, foreground* – Boja pozadine ili slova. Boje su definirane hex zapisom '#rrggbb' ili nazivima poznatih boja: ‘white’, ‘black’, ‘red’, ‘green’, ‘blue’, ‘cyan’, ‘yellow’ i ‘magenta’.
- *width, height* – željena širina i visina kontrole
- *text* – tekst koji se prikazuje na kontroli
- *font* – font, veličina slova i ostali parametri kao n-torka npr. ('Arial', 10)
- *command* – funkcija koja će se pokrenuti kada se klikne na kontrolu
- *variable* – ime varijable s kojom će se povezati kontrola (radi mogućnosti provjere vrijednosti bilo gdje u kodu). Mogu biti samo one koje nasljeđuju klasu *Tkinter.**Variable*** kao što su ugrađene klase ***StringVar***, ***IntVar***, ***DoubleVar*** i ***BooleanVar*** ili neka korisnička klasa.

Razmještaj kontrola na prozoru zavisi o metodi koju pozovu prilikom pridruživanja nadzornoj kontroli:

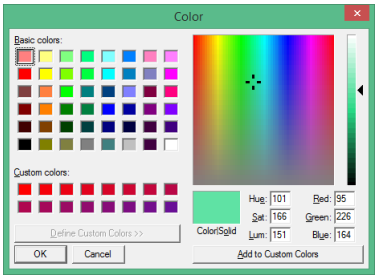
- pack() – kontrole se razmještaju kao blokovi
- grid() – kontrole se razmještaju kao u ćelijama tablice
- place() – kontrole se postavljaju na mjesto određeno koordinatama u pikselima

Pored izvršavanja neke korisničke funkcije pridruženoj nekoj kontroli preko '*command*' parametra, moguće je detektirati i događaje kao što su npr. klik mišem na neku kontrolu ili pomak mišem nad nekom kontrolom koristeći funkciju *bind* koju ima svaka instanca neke kontrole. Primjer detektiranja pritiska lijeve tipke miša za kontrolu tipa ***Canvas***:

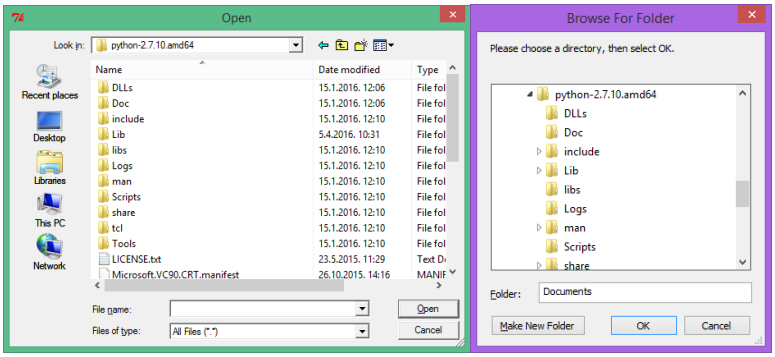
```
import tkinter as Tk
def CanvasClick(event):        #Event objekt generira Python (sadrži podatke o događaju)
    print event.x, event.y     #Ispisuje koordinate gdje je pritisnuta lijeva tipka miša
root = Tk.Tk()
C = Tk.Canvas(root, bg="white", height=600, width=800)
C.bind("<Button-1>", CanvasClick) #Izvršiti će se korisnička funkcija za svaki lijevi klik
C.pack()
```

Pored kontrola u *tkinter* prostoru imena, postoje još dodatne tipične kontrole kojima se pristupa preko zasebnih prostora imena (moraju se zasebno dodati):

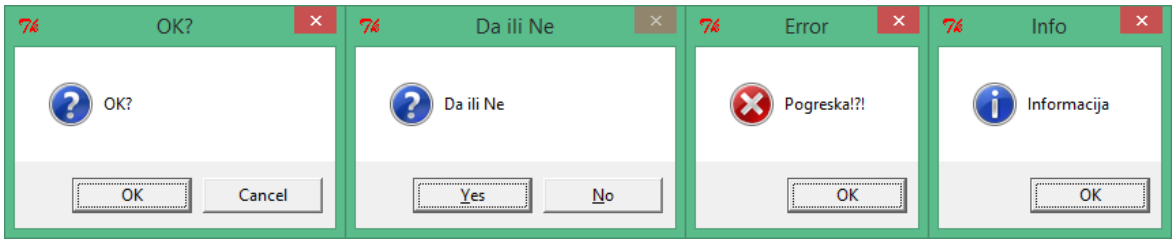
- colorchooser – Otvara prozor za odabir boja



- filedialog – Otvara prozor za odabir datoteka ili direktorija

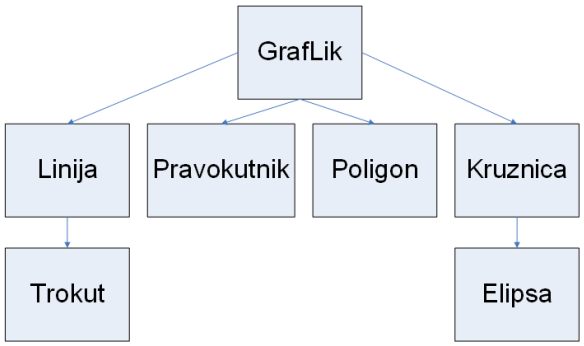


- messagebox – Otvara prozor za informiranje korisnika ili interakciju s korisnikom



Zadatak

Potrebno je napraviti Python aplikaciju s *tkinter* grafičkim sučeljem koja će na radnoj površini glavnog prozora u ***Canvas*** kontrolu nacrtati sliku učitanu iz tekstualne datoteke. U učitanoj tekstualnoj datoteci je u svakoj liniji opisan neki od geometrijskih likova: linija, trokut, pravokutnik, mnogokut, kružnica ili elipsa. Pored naziva geometrijskog lika napisani su još podaci bitni za crtanje pojedinog geometrijskog lika, te boja. Učitavanje i crtanje geometrijskih likova realizirati korištenjem objektno orijentirane paradigme pri čemu grafički objekti moraju biti organizirani u slijedeću hijerarhiju klasa:



Klasa *GrafLik* pored svojeg inicijalizatora mora još imati funkcije: *SetColor*, *GetColor* i *Draw*. Također mora imati i dva atributa odnosno varijable: *boja* (pogledati prethodni opis kako se koriste boje u Tkinter-u) i *tocka*. Ostale klase trebaju nasljeđivati ovu osnovnu klasu prema hijerarhiji na slici. Sve ostale klase trebaju imati svoje odgovarajuće inicijalizatore (koji će uvijek pozivati inicijalizatore nadklase) te dodatne atribute koje opisuju grafički objekt u kojem se nalaze kao što je to npr. druga točka za liniju (prva se već nalazi u *GrafLik* klasi), polumjer za kružnicu itd. Metoda *Draw* u svojoj osnovnoj implementaciji u *GrafLik* klasi ne treba ništa raditi pošto će svaka klasa implementirati svoju verziju *Draw* metode. Metoda *Draw* pored reference na objekt kojemu pripada (*self*) treba primiti i referencu na

objekt tipa ***Canvas*** na kojem će crtati geometrijske likove. Za realiziranje tih *Draw* metoda koristiti slijedeće već implementirane metode unutar ***Canvas*** klase:

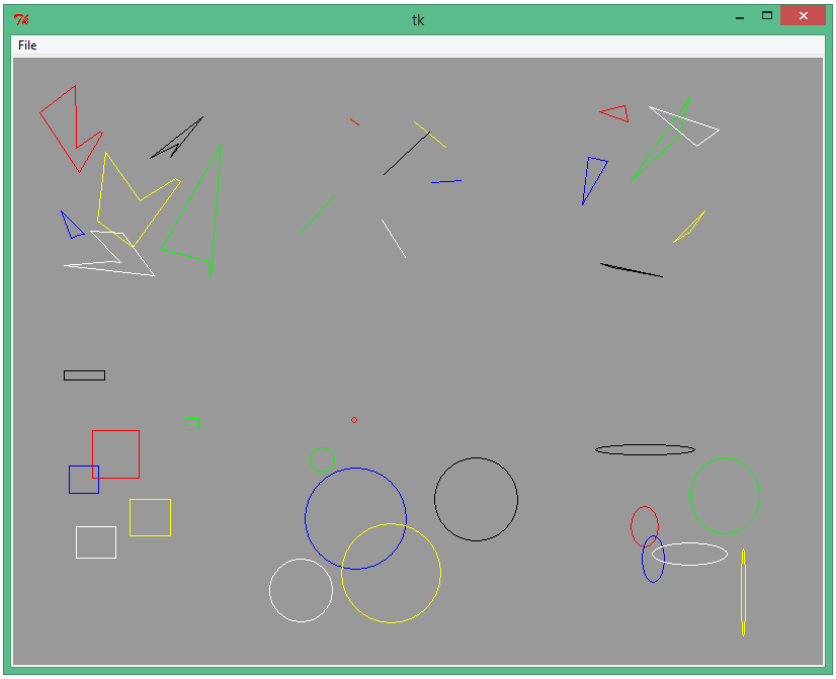
- ***create_line*** – crta liniju između dvije točke, a kao parametar uzima koordinate točaka (mogu biti i u n-torki), te boju koju postavlja u parametar *'fill'*.
- ***create_polygon*** – crta poligon pri čemu kao parametre uzima popis koordinata točaka poligona u n-torki, te u parametar 'outline' upisuje boju, a u parametar 'fill' prazan string ('). Koristi se za trokute i mnogokute.
- **create_rectangle** – crta pravokutnik, kao parametre uzima koordinate gornjeg lijevog vrha pravokutnika i donjeg desnog vrha pravokutnika, te u parametar 'outline' upisuje boju, a u parametar 'fill' prazan string (').
- ***create_oval*** – crta elipsu, kao parametre uzima koordinate gornjeg lijevog vrha pravokutnika i donjeg desnog vrha pravokutnika **u koji je elipsa upisana**, te u parametar 'outline' upisuje boju, a u parametar 'fill' prazan string ('). Koristi se za kružnice i elipse.

Sučelje aplikacije treba sadržavati meni 'File' s dvije opcije 'Open' i 'Exit'. 'Open' otvara željenu datoteku sa slikom (koristiti ***filedialog***), a 'Exit' zatvara cijelu aplikaciju (*Frame → self.quit*). Širina *Canvas* kontrole mora biti najmanje 800 točaka, a visina 600 točaka te imati sivu podlogu npr. '#999999'.

Ulazna datoteka se zove 'test.lik' i u svakoj liniji je opisan jedan geometrijski lik:

- Linija datoteke koja opisuje geometrijsku liniju sadrži riječ 'Line' kao prvi podatak, nakon čega je zapisana boja u obliku stringa i koordinate dvije točke (4 broja, x1, y1, x2, y2), npr.
  Line red 60.0 58.0 52.0 61.3
- Linija datoteke koja opisuje trokut sadrži riječ 'Triangle' kao prvi podatak, nakon čega je zapisana boja u obliku stringa i koordinate tri točke (6 brojeva, x1, y1, x2, y2, x3, y3), npr.
  Triangle green 420.0 75.0 465.1 46.1 385.1 108.7
- Linija datoteke koja opisuje pravokutnik sadrži riječ 'Rectangle' kao prvi podatak, nakon čega je zapisana boja u obliku stringa i koordinate gornje lijeve točke pravokutnika (2 broja, x, y), a nakon toga visina i širina pravokutnika, npr.
  Rectangle blue 613.0 62.0 25.8 37.8
- Linija datoteke koja opisuje mnogokut sadrži riječ 'Polygon' kao prvi podatak, nakon čega je zapisana boja u obliku stringa i koordinate N točka (N * 2 brojeva, x1, y1, x2, y2, x3, y3, …), npr. (poligon sa 6 točaka)
  Polygon yellow 197 392 147.2 411.3 196.0 412.9 243.4 406.0 226.6 432.4 246.2 435.2
- Linija datoteke koja opisuje kružnicu sadrži riječ 'Circle' kao prvi podatak, nakon čega je zapisana boja u obliku stringa i koordinate središta kružnice (2 broja, x, y), a nakon toga radijus, npr.
  Circle white 392.0 411.0 38.7
- Linija datoteke koja opisuje elipsu sadrži riječ 'Ellipse' kao prvi podatak, nakon čega je zapisana boja u obliku stringa i koordinate središta elipse (2 broja, x, y), a nakon toga dva radijusa (x-poluos i y-poluos elipse), npr.
  Ellipse black 587.0 379.0 48.6 25.2

Na sljedećoj slici može se vidjeti konačni izgled aplikacije i izgled učitane slike.


tk
File