1. Primjer primjene objektno usmjerenog programiranja

Pristup programiranju kod kojeg radimo s objektima koji međusobno komuniciraju zovemo objektno usmjereno programiranje. Svuda oko nas nalaze se objekti: računalo, mobitel, vozilo, knjiga, neki geometrijski lik itd. Objekti imaju svojstva koja ih opisuju. Svojstva geometrijskih likova mogu biti: oblik, veličina, boja, debljina crte… Ta svojstva u objektnom programiranju zovemo atributima. Objektima je moguće dodati metode koje određuju ponašanje objekata, npr. metodu za pomicanje geometrijskog lika ili metodu kojom ćemo taj objekt obojiti nekom bojom.

Objekti koji se međusobno neznatno razlikuju mogu se svrstati u tzv. klase objekata. Klasu možemo zamisliti kao složenu strukturu koja sadrži popis atributa i definicije metoda. Svaki objekt klase preuzima sve atribute i na njega se mogu primijeniti sve metode te klase.

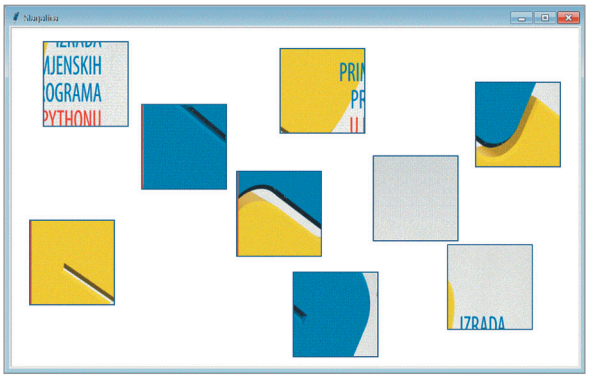
U Pythonu su svi tipovi podataka zapravo klase. Primjerice, tip podatka string je klasa čije je svojstvo pohranjeni tekst, a metode su npr. `capitalize()`, `lower()`, `strip()`.

Uobičajeno je naziv klase pisati velikim početnim slovom, a ako se naziv sastoji od više riječi, pišemo ih bez razmaka no svaka riječ počinje velikim slovom. Uporabom klasa mogu se stvoriti pregledniji algoritmi koji se mogu lakše dorađivati.

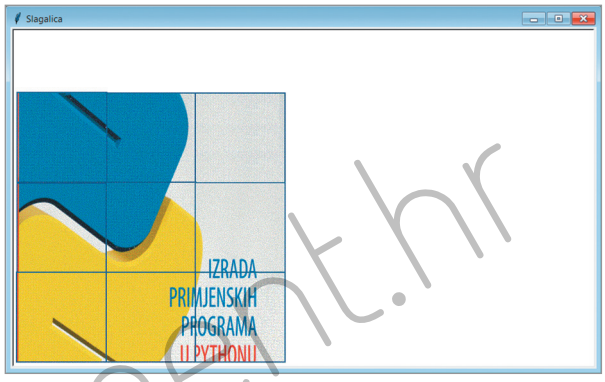
S obzirom na to da ćemo se u daljnjim poglavljima ove knjige koristiti isključivo objektno usmjerenim programiranjem, a programi će se temeljiti na postojećim klasama iz raznih modula, na jednom ćemo primjeru ponoviti osnovne principe objektno usmjerenog programiranja.

1.1. Jednostavna slagalica

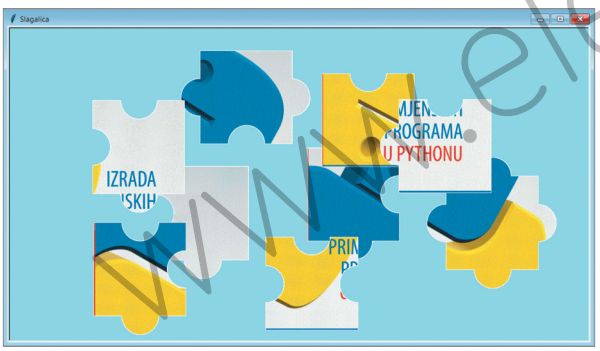
Napravimo slagalicu od 9 dijelova neke fotografije te napišimo program koji će nam omogućiti da s pomoću miša ponovno složimo dijelove slike u cjelinu. Na slikama 1.1.a i 1.1.b prikazana je slika podijeljena na dijelove s ravnim rubovima, dok su na slikama 1.2.a. i 1.2.b. prikazane slike podijeljene na klasične *puzzle* oblike. Dijelove slike preuzmite s *web*-stranice http://erz.element.hr/#python. Ako želite, možete uzeti i neku drugu sliku, uz poštovanje autorskih prava. Sliku podijelite na željeni broj dijelova npr. devet te dijelove slike spremite u *.gif* formatu. Postupak pripreme slika pogledajte u poglavlju 1.2.



Slika 1.1.a *Dijelovi slagalice s ravnim rubovima*



Slika 1.1.b *Složena slika*



Slika 1.2.a *Dijelovi slagalice s puzzle rubovima*

Slika 1.2.b *Složena slika*

Za izradu programa koristit ćemo se kornjačinom grafikom. Modul `turtle` je osmišljen tako da se može primijeniti u proceduralnom načinu i objektno usmjerenom načinu programiranja.

Kada rabimo modul u proceduralnom obliku (kao što smo u našim prethodnim knjigama radili) tada se sve funkcije modula importiraju na poznati način:

```
from turtle import *
```

Međutim, ako želimo objektno usmjereno programirati, onda trebamo uvesti dvije postojeće klase tog modula na sljedeći način:

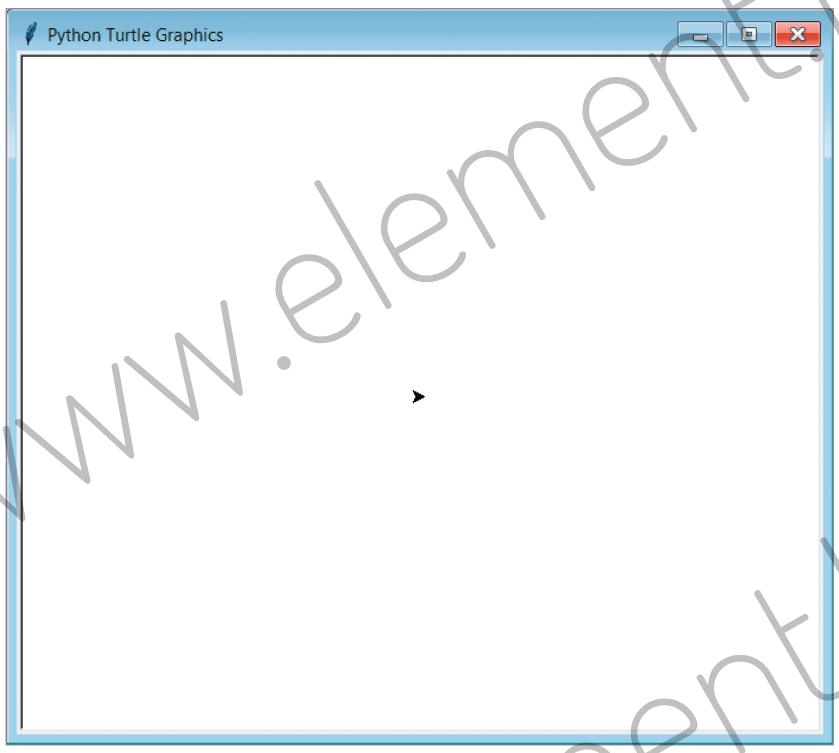
```
from turtle import Screen, Turtle
```

Sve funkcije koje smo rabili na proceduralni način sada postaju metode tih dviju klasa. U programu možemo definirati više objekata klase `Turtle`. Kod klase `Screen`, bez obzira na broj definiranih objekata, pojavit će se samo jedan objekt.

Prisjetimo se, prilikom pozivanja prve funkcije modula `turtle` otvara se novi grafički prozor s ishodištem u središtu tog prozora. Taj prozor je objekt klase *Screen*. Objekti klase *Turtle* imaju unaprijed definiran izgled: strelica je okrenuta udesno, crne boje, ostavlja trag kod pomicanja i

smještena je u središte koordinatnog sustava, slika 1.3. Provjerimo to u interaktivnom sučelju.

```
>>> from turtle import Screen, Turtle
>>> s = Screen()
>>> t = Turtle()
>>> reset()
```



Slika 1.3. *Grafički prozor s perom u obliku strelice koji se nalazi u sredini prozora*

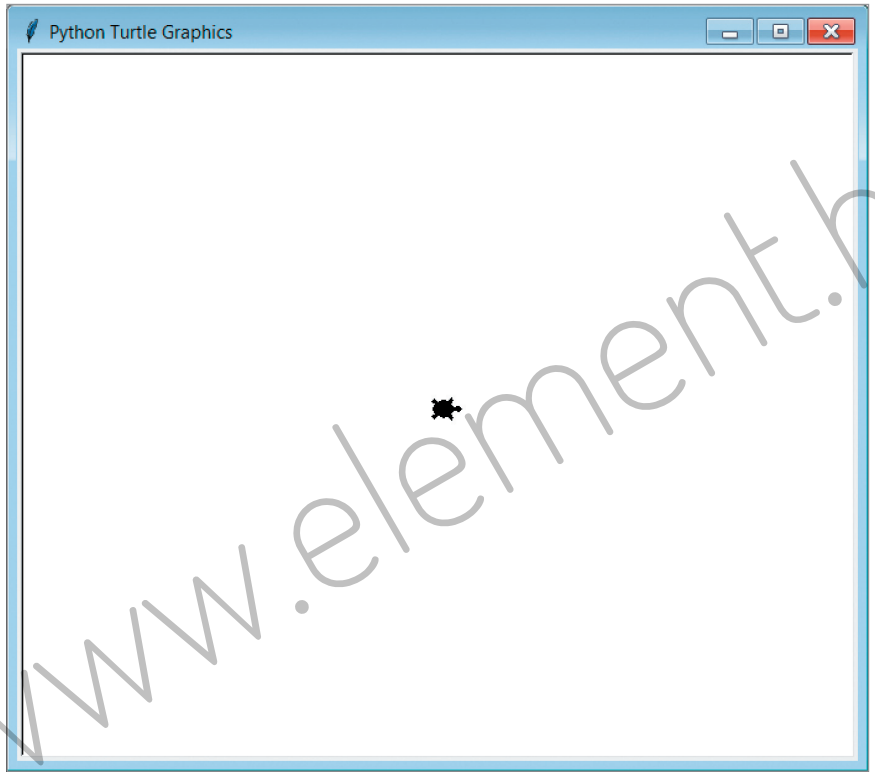
Umjesto strelice možemo odabrati neki od unaprijed definiranih izgleda: `'square'`, `'circle'`, `'turtle'`, `'triangle'`, `'classic'`. Za promjenu oblika pera koristit ćemo se naredbama iz modula `turtle`:

```
shape(name=None)
```

Promjenu izgleda pera možemo provjeriti u interaktivnom sučelju, slika 1.4., tako da napišemo naredbu:

```
>>> t.shape('turtle')
```

pri čemu je `t` objekt tipa `Turtle`.



Slika 1.4. *Grafički prozor s perom u obliku kornjače*

Ako umjesto unaprijed definiranih oblika želimo koristiti sliku, prvo moramo upotrijebiti metodu prozora `addshape()`. Ona dodaje odabranu sliku u popis oblika kojima se možemo koristiti za objekte.

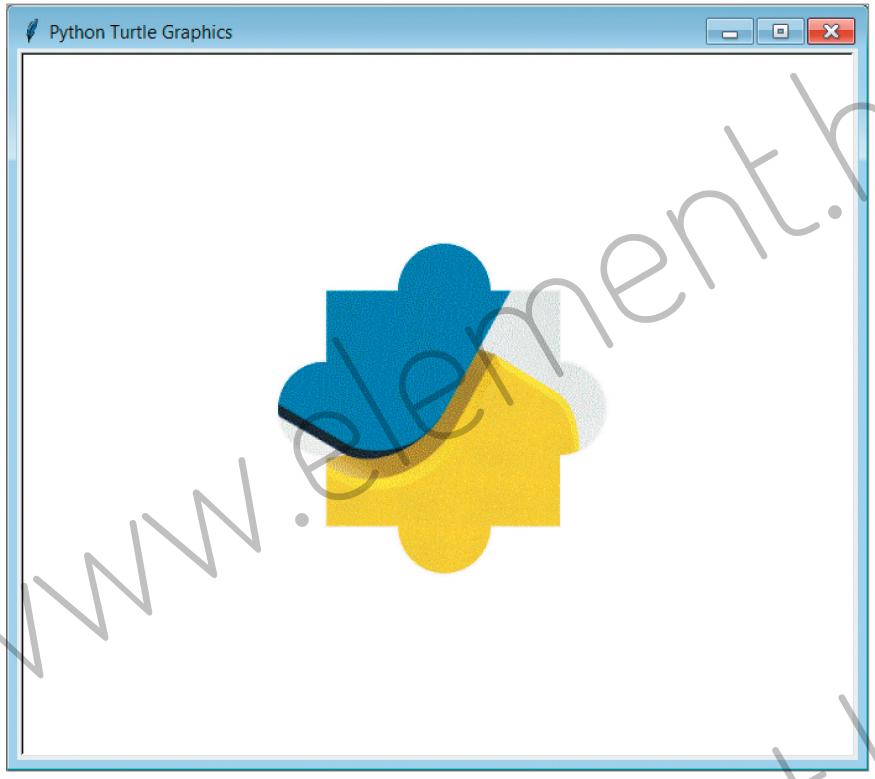
Napomena: slike trebaju biti u *.gif* formatu.

```
addshape(name, shape=None) ili register_shape(name, shape=None)
```

U našem će primjeru glavni prozor aplikacije biti objekt klase `Screen`, a dijelovi slagalice objekti klase `Turtle`. To možemo provjeriti tako da napišemo program u kojem ćemo upotrijebiti jedan dijelić naše slagalice umjesto standardnog izgleda pera, slika 1.5. Slika koju postavljamo za novi izgled pera treba biti u istoj mapi gdje smo pohranili program.

```
from turtle import Screen, Turtle
s = Screen()
t = Turtle()
s.addshape('slika5.gif')                                    #1
t.shape('slika5.gif')
```

Ako odabrana slika nije pohranjena u istoj mapi gdje se nalazi naš program, potrebno je u naredbi (#1) navesti putanju do željene slike.



Slika 1.5. *Grafički prozor s perom u obliku jednog dijela slagalice*

Za razliku od strelice koju je moguće usmjeriti u bilo kojem smjeru, kod pera u obliku slike ne vidi se u kojem je smjeru okrenut objekt.

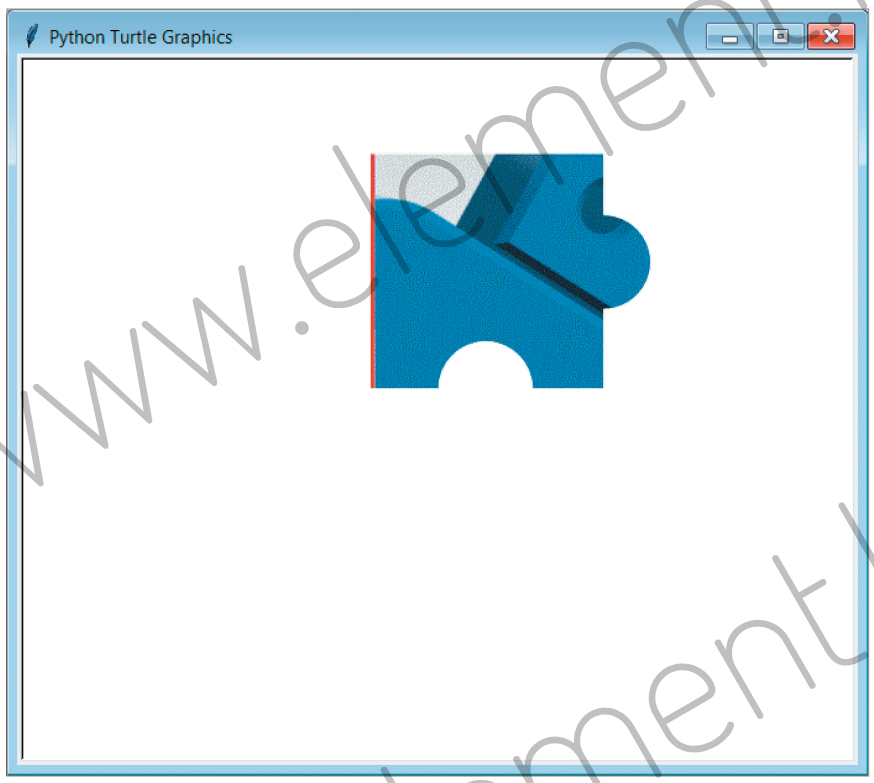
Pretpostavimo da smo pripremili 9 dijelova slagalice koje smo spremili pod nazivima od *slika1.gif* do *slika9.gif*. Napravimo objekt tipa `Turtle`. Odredimo svojstva objekta koja su nam važna:

- umjesto oblika strelice trebamo imati sliku dijela slagalice koji pomičemo
- objekt kod pomicanja ne smije ostavljati trag
- prije početka slaganja element slike nalazi se na nasumičnom mjestu u prozoru
- ne treba se vidjeti animacija što znači da brzina animacije treba biti 0.

Izvođenjem sljedećih naredbi dobit ćemo prikaz kao na slici 1.6.

```
from turtle import Screen, Turtle
from random import randint
dio = Turtle()                    # kreira jedan objekt - dio slagalice
```

```
s = Screen()                    # kreira prozor aplikacije
s.addshape('slika1.gif')        # dodaje sliku u popis oblika
dio.shape('slika1.gif')         # dodaje sliku dijelu slagalice
dio.pu()                        # podiže pero
dio.goto(randint(-300, 300), randint(-200, 200)) # nasumično mjesto
dio.speed(0)                    # animacija se neće vidjeti
```



Slika 1.6. *Grafički prozor s jednim elementom slagalice na nasumičnom mjestu*

Očito je da je potrebno devet takvih elemenata. Zato ćemo kreirati novu klasu koja će naslijediti sva svojstva postojeće klase, ali ćemo joj dodati i još neke atribute i metode.

Prisjetimo se, opći oblik definicije klase koja nasljeđuje neku klasu je:

```
class ime_klase(ime_roditeljske_klase):
    definicija klase
```

Isto tako znamo da je za postavljanje svojstava klase na neku vrijednost potrebno napisati posebnu metodu koju zovemo **konstruktor** klase. Ta se metoda izvodi prilikom kreiranja objekta

iz klase. Ime metode je predefinirano i oblika je `__init__(self, parametri)`. Kada želimo kreirati novu klasu iz postojeće, trebamo unutar konstruktora nove klase tj. unutar metode `__init__()` pozvati konstruktor roditeljske klase i to tako da ispred imena metode napišemo `super()` (#4). Umjesto `super()` možemo napisati ime postojeće roditeljske klase od koje nasljeđuje svojstva. U našem primjeru mogli bismo napisati i `Turtle.__init__(self)`.

Definicija klase podrazumijeva definiciju svih atributa i metoda. Metode definiramo jednako kao funkcije, s time da se u svim metodama mora pojaviti parametar `self` koji predstavlja objekt te klase. On se pojavljuje kao prvi parametar, a ponekad i kao jedini parametar.

Svakom atributu pristupamo tako da napišemo `self.ime_atributa` tj. tako da se napiše prvi parametar, najčešće `self`, iza njega točka te naziv metode.

Za potrebe našeg zadatka kreirat ćemo novu klasu `Slagalica` koja će naslijediti atribute i metode iz klase `Turtle`.

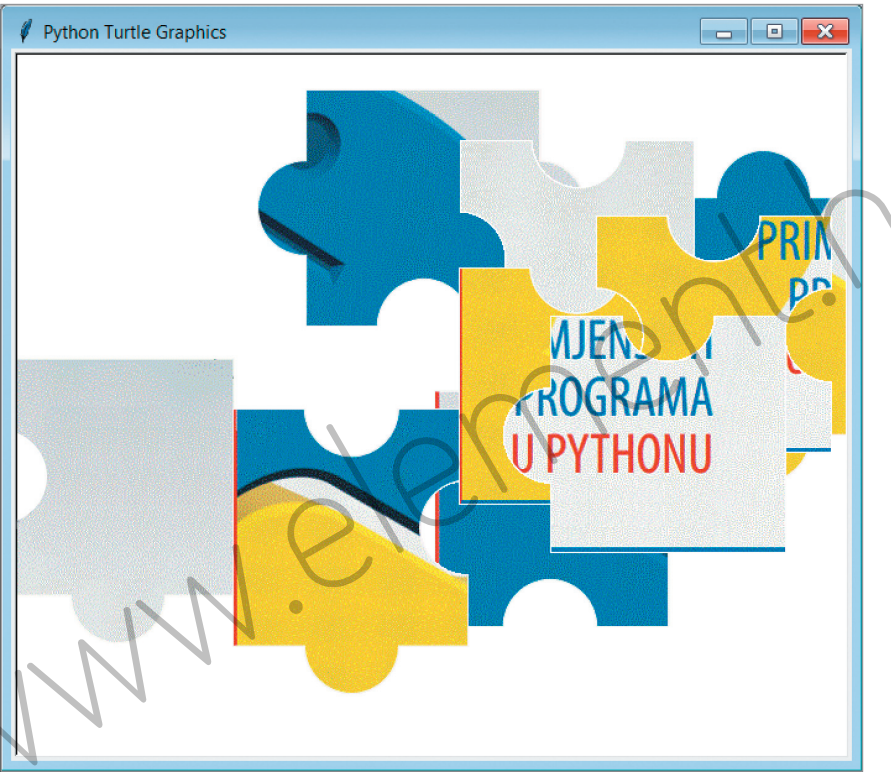
```
from turtle import Screen, Turtle
from random import randint

class Slagalica(Turtle):                                        #2
    def __init__(self, slika ):                                 #3
        super().__init__()                                      #4
        self.pu()                                               #5
        self.speed(0)
        self.shape(slika)
        self. goto(randint(-300, 300), randint(-200, 200))
```

Prilikom definiranja klase `Slagalica` unutar zagrada navodimo ime klase koja se nasljeđuje (#2). Zatim definiramo konstruktor nove klase `Slagalica` unutar kojeg navodimo i parametar naziv slike (#3) te unutar njega pozivamo konstruktor klase koja se nasljeđuje (#4). Umjesto unaprijed zadanog svojstva `pd()`, spuštenog pera koje ostavlja trag, dodajemo novo svojstvo `pu()` koje će dignuti pero tako da se trag ne vidi (#5).

U glavnom programu kreirat ćemo praznu listu kojoj ćemo kroz petlju dodati devet objekata klase `Slagalica`. Nakon toga u popis oblika treba dodati devet dijelova slagalice s pomoću metode nad prozorom `addshape()`. Svaki od njih trebamo dodati pojedinom elementu liste.

```
s = Screen()
dio = []
for i in range(9):
    sl = 'slika{}.gif'.format(i + 1)
    s.addshape(sl)          # dodavanje svake od 9 slika u popis oblika
    dio.append(Slagalica(sl)) # kreiranje liste od 9 dijelova slike
```

Nakon svega dobit ćemo prozor kao na slici 1.7.



Slika 1.7. *Grafički prozor s dijelovima slagalice na nasumičnim mjestima*

Da bi program funkcionirao kako treba, svaki objekt moramo moći mišem pomaknuti na neko mjesto na ekranu i za to ćemo se koristiti posebnom metodom `ondrag()` koja se pokreće nakon određenog **događaja** na ekranu.

```
ondrag(funkcija)
```

Metoda `ondrag()` definirana je na peru. Funkciji koju metoda `ondrag()` poziva proslijeđuju se koordinate točke na kojoj se u tom trenutku nalazi pero. Metoda se ponavlja onoliko puta koliko se događaja dogodi.

Ostaje nam još da na svakom elementu slagalice tj. *nad svakim objektom tipa Slagalica* primijenimo metodu `ondrag()` za koju smo u klasu morali dodati metodu `pomak()`. Metodu ćemo definirati kao funkciju s tim da će se kao prvi pojaviti parametar `self` koji predstavlja objekt te klase.

```
def pomak(self, x, y):
    self.goto(x, y)
```

Sada možemo nad svakim elementom slagalice primijeniti metodu `ondrag()`.

Pogledajmo mogući oblik programa:

```
from turtle import Screen, Turtle
from random import randint

class Slagalica(Turtle):
    def __init__(self, slika):
        super().__init__()
        self.pu()
        self.speed(0)
        self.shape(slika)
        self. goto(randint(-300, 300), randint(-200, 200))
        self.ondrag(self.pomak)

    def pomak(self, x, y):
        self.goto(x, y)

s = Screen()
dio = []

def glavni():
    s.title('Slagalica')
    s.setup(1200,650,0,0)       # postavlja veličinu i poziciju prozora
    s.bgcolor('lightblue')      # zadaje boju pozadine
    for i in range(9):
        sl = 'slika{}.gif'.format(i + 1)
        s.addshape(sl)
        dio.append(Slagalica(sl))
if __name__ == '__main__':
    glavni()
```

1.2. Priprema slike za slagalicu

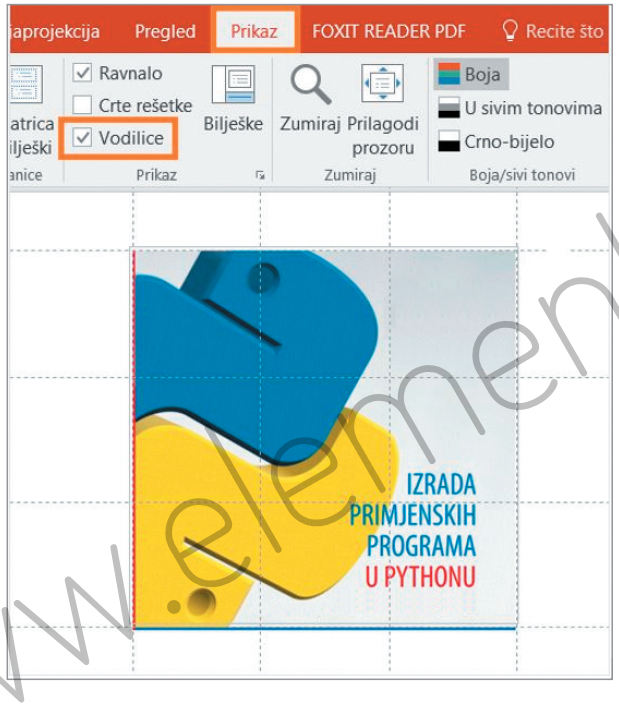
1.2.1. Podjela slike na pravokutne oblike

U ovom ćemo dijelu objasniti kako možemo prirediti sliku za potrebe programa slagalica. Jedan od jednostavnih načina je podjelu slike napraviti uporabom MS PowerPointa.

Nakon što na prazan slajd umetnemo željenu sliku, uključit ćemo vodilice koje se nalaze na kartici *Prikaz (View)*. Kako nama trebaju i dodatne vodilice da bismo sliku podijelili na 9 dijelova, njih možemo dodati ako pritisnemo desnu tipku miša te odaberemo *Dodaj okomitu (vodoravnu) vodilicu*, slika 1.8.

Slika 1.8. *Dodavanje vodilica u PowerPointu*

S pomoću vodilica sliku podijelimo na željeni broj dijelova, slika 1.9. Nakon podjele slike, pomoću *Alata za izrezivanje* (engl. *Snipping tool*) možemo izrezati svaki od npr. devet dijelova te ih spremiti u *.gif* formatu.



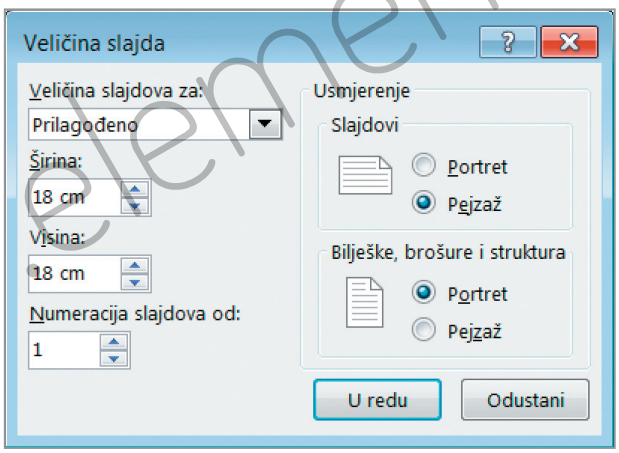
Slika 1.9. *Vodilice dijele sliku na devet dijelova*

Prisjetimo se, slike je važno spremiti u istu mapu gdje će se nalaziti i program u *Pythonu*.

1.2.2. Podjela slike na *puzzle* oblike

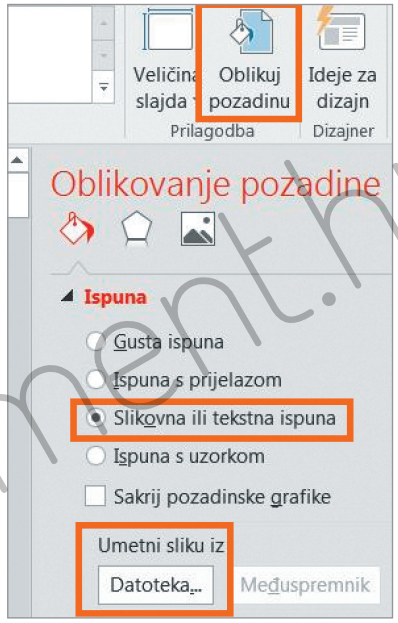
Nešto složeniji postupak je ako umjesto pravokutnika želimo napraviti *puzzle*. Možemo se koristiti različitim *online* programima, ali to možemo napraviti i u PowerPointu.

Na kartici *Dizajn* odabrat ćemo *Veličina slajda* te iz padajućeg izbornika odabrati *Prilagođena veličina slajda,* slika 1.10. Poželjno je veličinu slajda prilagoditi veličini slike.



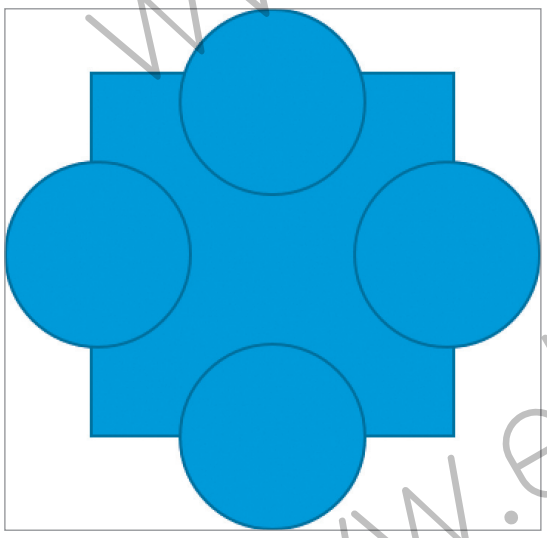
Slika 1.10. *Odabir veličine slajda*

Sad postavimo sliku kao pozadinu tako da na kartici *Dizajn*, u grupi naredbi *Prilagodba* odaberemo *Oblikuj pozadinu*, slika 1.11. U oknu *Oblikovanje pozadine* odabrat ćemo *Slikovna ili tekstna ispuna* te *Umetni sliku iz*. Nakon klika na *Datoteka* izabrat ćemo sliku.



Slika 1.11. *Postavljanje slike kao pozadine*

Sad ćemo pripremiti izgled rešetke naše slagalice. Na prazan slajd, s uključenim vodilicama, umetnut ćemo kvadrat 5×5 i poravnati ga sa sredinom okomito i vodoravno. Zatim ćemo umetnuti krug i kopirati ga još tri puta te krugove postaviti kako je prikazano na slici 1.12. Za precizno ćemo poravnanje na kartici *Polazno* odabrati *Razmještaj* te iz padajućeg izbornika kliknuti na *Poravnaj* i odabrati potrebno poravnanje. Uklonit ćemo obrube tako da na kartici *Alati za crtanje – Oblikovanje* odaberemo *Kontura oblika* pa *Bez konture* te ćemo dobiti izgled kao na slici 1.13.

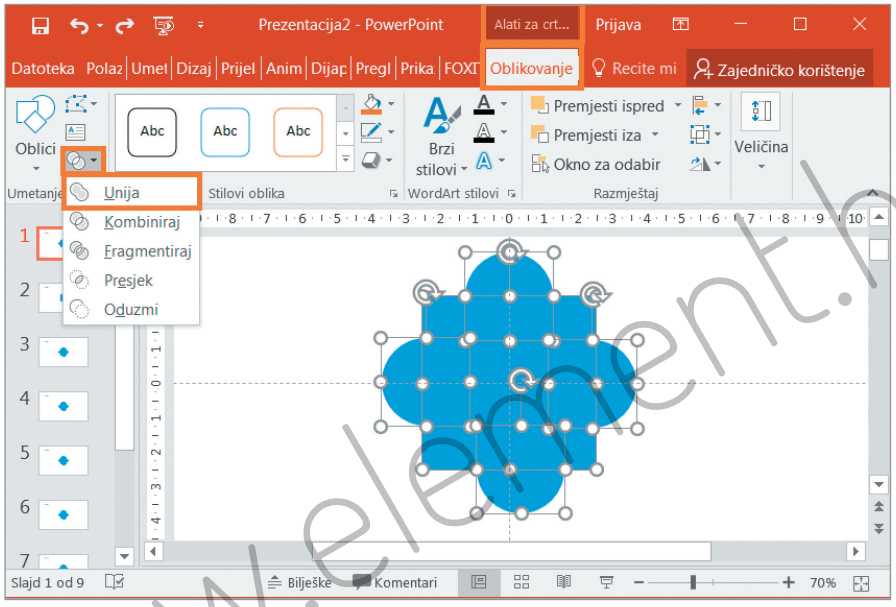


Slika 1.12. *Kvadrat s četiri kruga*

Slika 1.13. *Kvadrat s četiri kruga bez obruba*

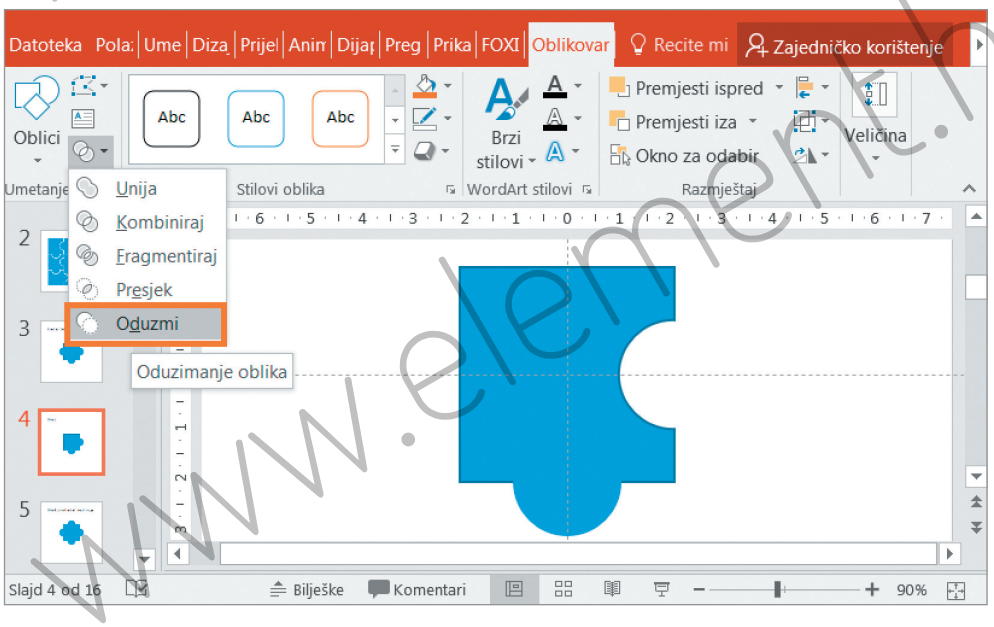
Ovo će nam biti podloga za izradu svih devet potrebnih dijelova te odmah možemo trenutačni slajd kopirati još osam puta tj. potreban broj puta.

Kako bismo spojili oblike u novi geometrijski oblik, označit ćemo ih te na kartici *Alati za crtanje – Oblikovanje* kliknuti na opciju *Unija*, slika 1.14.



Sllika 1.14. *Spajanje oblika u novi geometrijski oblik*

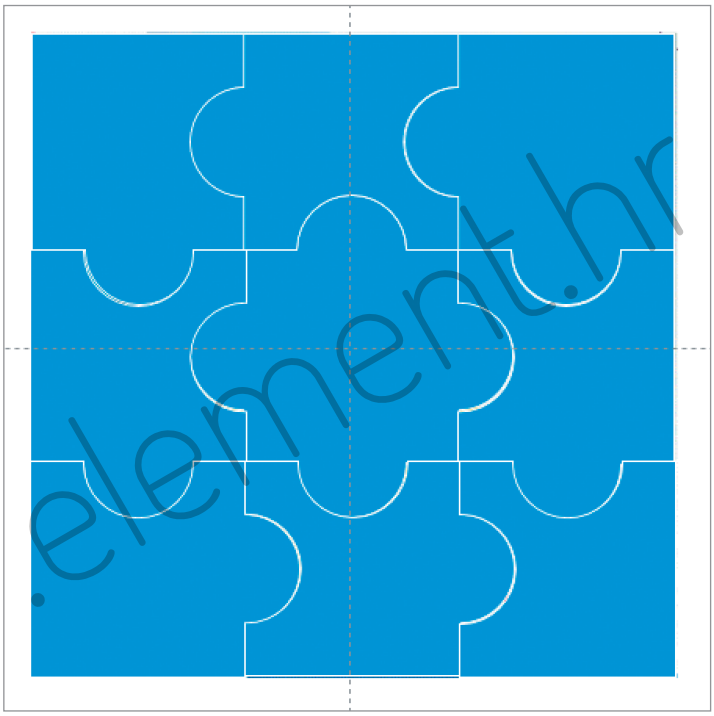
Na sličan način mogu se napraviti i ostali oblici, s tim da želimo li izrezati dio iz slike, označimo sliku i dio koji treba izrezati te u istom izborniku odaberemo *Oduzmi*, slika 1.15.



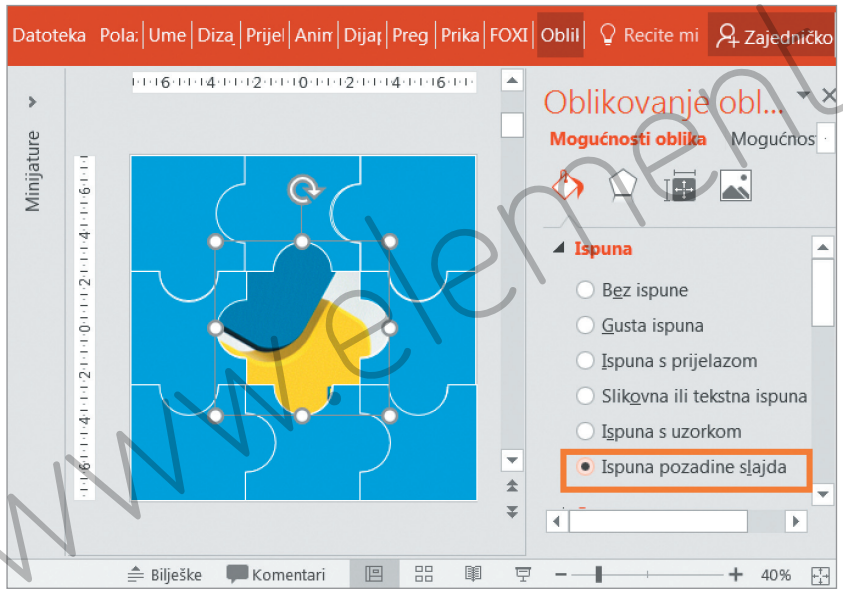
Slika 1.15. *Izrezivanje oblika*

Sada ćemo od svih dijelova složiti cjelinu tako da ih sve smjestimo na jedan slajd, slika 1.16. Na sličan način na koji smo uklonili obrube, sada možemo sve dijelove obrubiti bijelom bojom.

Označit ćemo sve oblike, grupirati ih te ih postaviti preko slajda na kojem se u pozadini nalazi slika. Ako se veličine ne poklapaju, grupirani oblik možemo podesiti tako da točno poklapa sliku. Nakon toga ćemo razgrupirati oblik te za ispunu odabrati *Ispuna pozadine slajda*, slika 1.17.

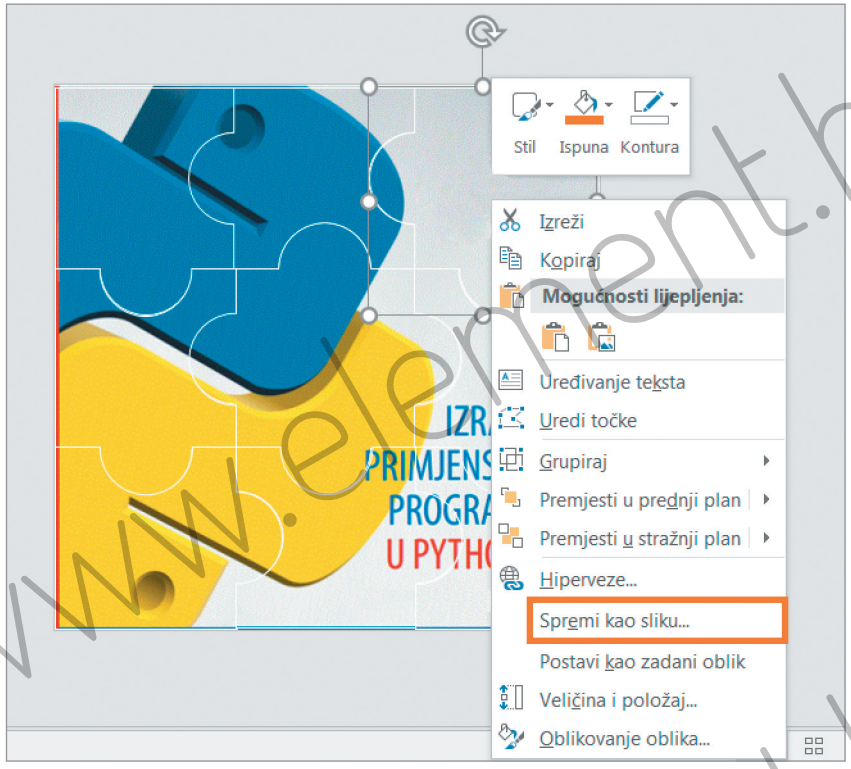


Slika 1.16. *Složena mreža*



Slika 1.17. *Ispuna pozadine slajda*

Zadnji korak je svaki dio spremiti u *.gif* formatu na način da na njega kliknemo desnim klikom te u padajućem izborniku odaberemo *Spremi kao sliku*, slika 1.18.



Slika 1.18. *Spremanje dijelova slagalice kao slike u .gif formatu*

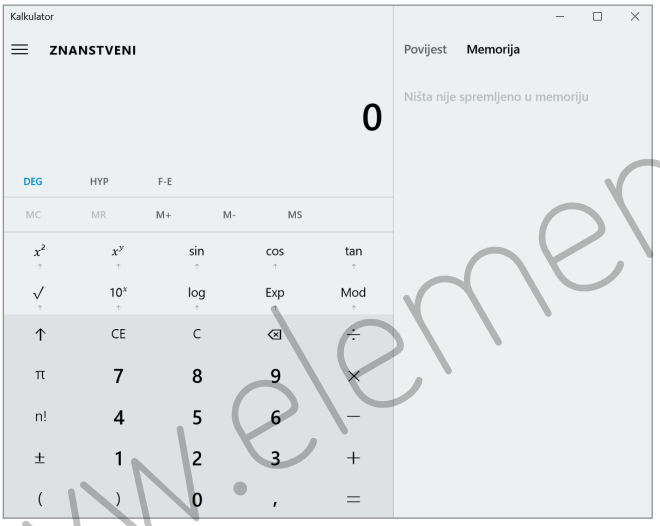
Koristeći principe objektno-usmjerenog programiranja mogu se napraviti i druge igre i aplikacije.

2. Programi s grafičkim korisničkim sučeljima

gumb **događaj** *Grid* Listbox tkinter trake za pomicanje

2.1. O grafičkom korisničkom sučelju

Programi koje smo dosad radili, unosili su podatke iz komandne linije ili datoteke te su svoje rezultate ispisivali na komandnu liniju, u datoteku ili su ih prikazivali na grafičkom zaslonu (crtali). Takvi su programi vrlo praktični kad trebamo nešto napraviti za vlastitu upotrebu, na brzinu nešto isprogramirati... Međutim, želimo li da korisnici prihvate našu aplikaciju, neće biti dovoljna samo njezina osnovna funkcionalnost, već će biti važan i oblik sučelja s korisnikom. Danas većina programa za komercijalnu upotrebu ima vrlo moderna grafička korisnička sučelja.



Slika 2.1. *Primjer programa s grafičkim korisničkim sučeljem*

Python omogućava kreiranje takvih programskih sučelja te ćemo se time baviti u nastavku. Za kreiranje takvih programa koristit ćemo modul `tkinter`[1]. Osnovni element svakog programa s grafičkim korisničkim sučeljem bit će prozor. Na njega ćemo dodavati (ponekad ćemo reći i

1 Modul `tkinter` je sastavni dio osnovnog *Pythonova* paketa.

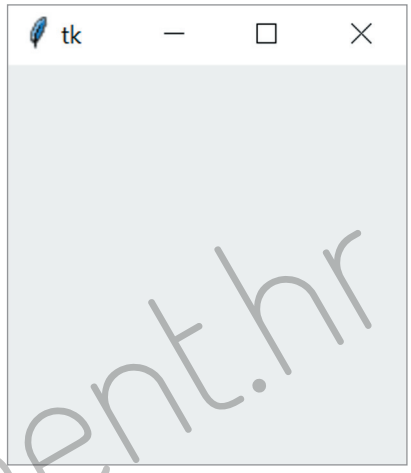
lijepiti) različite elemente kao što su: okviri za unos teksta, različite vrste gumba, izbornici itd. Za svaki od navedenih elemenata, pa i za sam prozor, kreirana je posebna klasa unutar modula `tkinter`, a dodavanje pojedinog elementa na sučelje programa svodit će se na kreiranje instancije (jedinke) pripadne klase, postavljanje parametara te smještanje objekta na sučelje.

Kao što znamo, neka radnja unutar takvog korisničkog sučelja rezultirat će određenim odgovorom (reakcijom). Primjerice, kliknemo li na gumb 4 u programu kalkulator (slika 2.1), na zaslonu programa pojavit će se broj 4. Općenito ćemo reći da nad pojedinim elementima sučelja možemo pratiti određene vrste **događaja**. Naše programiranje će se u osnovi svoditi na dizajniranje grafičkog korisničkog sučelja te pisanja odgovora na događaje. Prvo trebamo odrediti događaje koje ćemo pratiti i što se treba dogoditi kao odgovor na pojedini događaj.

2.2. Osnovni prozor programa – klasa `Tk`

Kao što smo rekli, prozor, ali i svi ostali elementi grafičkog korisničkog sučelja su klase unutar modula `tkinter`. Klasa koja kreira osnovni prozor programa je `Tk()`. Kreiranjem instancije ove klase na ekranu će se odmah pojaviti prozor kao na slici 2.2. Na samom početku uvest ćemo modul `tkinter` što ćemo načiniti naredbom: `from tkinter import *`. Isprobajmo to u *Pythonovu* interaktivnom sučelju:

```
>>> from tkinter import *
>>> t = Tk()
>>> mainloop()
```



Slika 2.2. *Osnovni prozor programa*

Primijetimo da smo nakon kreiranja instancije klase `Tk()` pozvali funkciju `mainloop()`. Prisjetimo se, ovom smo se funkcijom koristili kod kornjačine grafike kad smo htjeli da grafički prozor ostane aktivan sve dok ga ne zatvorimo klikom na gumb *Close* (). U nekim situacijama prozor se neće pojaviti ako izostavimo funkciju `mainloop()` te ćemo je iz toga razloga mi u svim primjerima ipak zapisivati.

Kao što možemo primijetiti, ovaj prozor u naslovnoj traci ima tekst *tk* te ima neku veličinu. Naslov, veličinu, ali i još niz drugih parametara prozora moguće je programski promijeniti. Za promjenu naslova koristit ćemo se metodom:

```
title(naslov)
```

Dok ćemo veličinu, ali i niz drugih parametara prozora, promijeniti metodom:

```
config(p_1 = v_1, p_2 = v_2,...)
```

pri čemu su `p_1`, `p_2`, . . . nazivi parametara, a neki od naziva parametara za objekt klase `Tk()` dani su u tablici 2.1.

Naziv parametra	Opis
`background, bg`	pozadinska boja
`borderwidth, bd`	debljina ruba prozora
`cursor`	oblik kursora dok je nad prozorom, neki od oblika kursora su: *arrow, man, mouse, pencil, plus, clock, cross, sizing, draft_large, draft_small, exchange, hand1, hand2, heart, umbrella...*
`height`	visina prozora
`padx, pady`	udaljenost sadržaja od rubova prozora
`width`	širina prozora

Tablica 2.1. *Neki od parametara klase* `Tk()`

Boju je općenito moguće zadati njenim engleskim nazivom ili RGB kôdom. U oba slučaja naziv boje je string te ga pišemo unutar navodnika.

Isto tako moguće je odrediti hoćemo li dozvoliti da se prozoru mijenja veličina ili ćemo postaviti da je veličina prozora fiksna i nepromjenjiva. To ćemo definirati metodom:

```
resizable(width, height)
```

Parametri `width` i `height` odnose se na mogućnost promjene veličine prozora po širini, odnosno visini. Želimo li da se veličina po nekoj komponenti može mijenjati, postavit ćemo vrijednost pripadnog parametra na `True`, a inače ćemo postaviti vrijednost parametra na `False`.

Prozor ćemo zatvoriti metodom:

```
destroy()
```

Primjer 2.1.

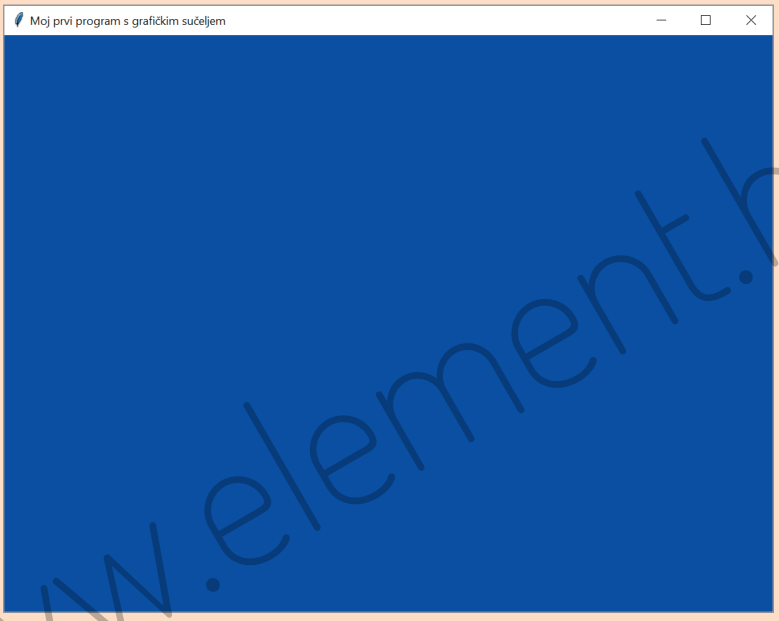
Kreirajmo prozor u čijoj će naslovnoj traci pisati: Moj prvi program s grafičkim sučeljem, dok će dimenzije prozora biti 800×600, a pozadinska boja neka bude plava.

Rješenje:

Budući da se radi o vrlo jednostavnom zadatku, rješenje možemo napisati u interaktivnom sučelju:

```
>>> from tkinter import *
>>> t = Tk()
>>> t.config(width = 800, height = 600, bg = 'blue')
>>> t.title('Moj prvi program s grafičkim sučeljem')
''
>>> mainloop()
```

Nakon što su izvedene sve naredbe, na zaslonu će se pojaviti prozor kao na slici 2.3.



Slika 2.3. *Prozor dimenzija 800 × 600 piksela*

Vrijednosti pojedinog parametra moguće je dohvatiti metodom:

```
cget(p)
```

pri čemu je `p` naziv parametra i piše se unutar navodnika.

Iskoristimo prozor kreiran u prethodnom primjeru kako bismo ilustrirali upotrebu ove metode:

```
>>> from tkinter import *
>>> t = Tk()
>>> t.config(width = 800, height = 600, bg = 'blue')
>>> t.cget('width')
800
>>> t.cget('bg')
'blue'
>>> mainloop()
```

Osim s pomoću `config()`, neki parametar možemo promijeniti i na način da nad imenom objekta unutar uglatih zagrada pod navodnicima napišemo naziv parametra te mu pridružimo željenu vrijednost. Primjetimo da smo na isti način pristupali i elementima riječnika. Dakle, objekti GUI-a se na neki način ponašaju i kao rječnici, pri čemu je ključ rječnika naziv odgovarajućeg parametra. Ilustrirajmo to na prethodnom primjeru:

```
>>> from tkinter import *
```

```
>>> t = Tk()
>>> t['width'] = 800
>>> t['height'] = 600
>>> t['bg'] = 'blue'
>>> t['width']
800
>>> t['bg']
'blue'
>>> mainloop()
```

2.3. Okviri

Nakon što smo kreirali prozor dodat ćemo mu okvir (engl. *frame*). Okvir neće biti vidljiv no poslužit će nam za lakše organiziranje elemenata koje ćemo dodavati u naš prozor.

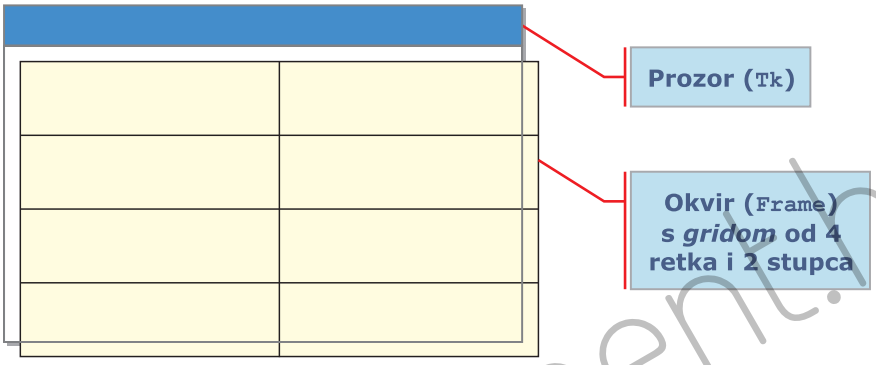
Za postavljanje okvira na prozor koristit ćemo se klasom `Frame`. Parametar konstruktora ove klase je objekt tipa `Tk`. Osim objekta tipa `Tk` u konstruktoru je moguće definirati i vrijednost za neki parametar. Parametri su ekvivalentni parametrima kod klase `Tk`. Nakon kreiranja okvira koji će kao parametar imati glavni prozor aplikacije, veličina glavnog prozora aplikacije će se inicijalno prilagoditi veličini okvira. Nakon kreiranja objekta tipa `Frame`, trebamo ga još pripremiti kako bismo mogli dodavati željene elemente. Tri su načina na koje je moguće postavljati elemente sučelja u okvir, a zovemo ih upraviteljima geometrije prozora (engl. *geometry management*). Njihovi nazivi su:

- ***Grid*** – okvir se podijeli u retke i stupce te se elementi sučelja stavljaju u odgovarajuće ćelije
- ***Pack*** – za svaki element moguće je odrediti na koje mjesto okvira ćemo ga staviti (*TOP*, *BOTTOM*, *LEFT*, *RIGHT*), ukoliko više elemenata stavljamo na isto mjesto oni će se postaviti jedan pokraj drugoga
- ***Place*** – svaki element stavljamo na točno određeno mjesto (koordinate) na prozoru.

Za svaki od načina postavljanja elemenata kreirana je metoda unutar klase `Frame` i tom metodom naglašavamo koji ćemo upravitelj koristiti. Metode su redom:

```
grid()
pack()
place()
```

Mi ćemo se uglavnom koristiti prvim načinom smještanja elementa na prozor – stavljat ćemo elemente u tablicu (engl. *grid*), dakle koristit ćemo se metodom `grid()`, slika 2.4.



Slika 2.4. *Shematski prikaz prozora s okvirom i gridom*

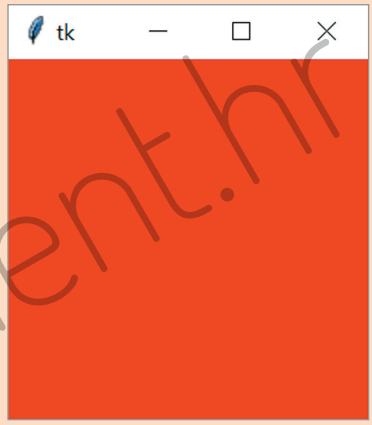
Primjer **2.2.**

Kreirajmo prozor veličine 800×600 s plavom bojom pozadine te na njemu okvir veličine 400×300 s crvenom bojom pozadine i gridom.

Rješenje:

```
from tkinter import *
t = Tk()
t.config(width = 800, height = 600, bg = 'blue')
f = Frame(t, width = 400, height = 300, bg = 'red')
f.grid()
mainloop()
```

Pokretanjem ovog programa dobit ćemo prozor kao na slici 2.5. Kao što možemo primijetiti veličina prozora se prilagodila veličini okvira.



Slika 2.5. *Rezultat izvođenja programa iz primjera 2.2.*

U nastavku ćemo se upoznati s najčešćim elementima sučelja te kako ih koristiti.