7 — Klase i objekti

Python je programski jezik temeljen na objektima. Za razliku od nekih drugih, također objektno-orijentiranih, jezika, Python ne uvjetuje korištenje isključivo objektno-orijentirane paradigme. Python također podržava proceduralno (strukturalno) programiranje s modulima i funkcijama, tako da je moguće izabrati najprikladniju paradigmu programiranja za svaki dio programa. Općenito, objektno-orijentirana paradigma najbolja je kada se žele skupiti podaci i funkcije u praktične pakete. Isto tako može biti korisna kada se žele koristiti neki od Pythonovih objektno-orijentiranih mehanizama, kao npr. naslijeđivanje. Proceduralna paradigma, koja je bazirana na modulima i funkcijama, obično je jednostavnija i prikladnija u slučajima kada nikakve prednosti objektno-orijentiranog programiranja nisu potrebne. Novi stil objektnog modela postaje standard u novim verzijama Pythona, pa se preporučuje koristiti ga za programiranje.

7.1 Klase i instance

Klasa ili *razred* je Python objekt s nekoliko značajki:

- *Objekt klase* može se pozivati kao da je funkcija. Poziv stvara novi *objekt*, poznat kao *instanca klase*, za koju se zna kojoj klasi pripada.
- Klasa ima po volji *imenovane atribute* koji se mogu povezivati i referirati.
- Vrijednosti atributa klase mogu biti podatčani objekti ili funkcijski objekti.
- Atributi klase koji su povezani na funkcije poznati su kao *metode klase*.
- Metoda može imati posebno ime definirano od Pythona s dvije podcrte ('__') ispred i iza imena. Python upotrebljava takve posebne metode za neke operacije koje se izvršavaju na instancama klase.
- Klasa može podatke i metode *naslijediti* od drugih klasa, pa potraga nekog atributa koji nije unutar same klase prenosi se na druge objekte, roditelje klase.
- *Instanca* klase je Python objekt s po volji *imenovanim atributima* koji se mogu povezivati i referirati.
- Python klase su objekti, pa se s njima postupa kao i s drugim objektima. Tako se klasa može prenijeti kao argument u pozivu funkcij, a funkcija može vratiti klasu kao rezultat poziva. Klasa, kao i bilo koji drugi objekt, može biti povezana na varijablu (lokalnu lili globalnu), član niza ili atribut objekta. Klase također mogu biti i ključevi u riječniku.

7.2 Naredba 'class'

Naredba class je najčešći način stvaranja objekta klase, i to s ovakvom sintaksom:

```
class ime\_klase[(klase\_roditelji)]:
    naredba(e)
```

String ime_klase je identifikator. To je varijabla koja postaje povezana (ili re-povezana) s objektom klase nakon što naredba class završi s izvršavanjem. klase_roditelji je po volji zarezima odvojen niz izraza (obično stringovnih identifikatora) čije su vrijednosti objekti klasa. Ove su klase poznate pod različitim imenima u različitim jezicima - govori se o baznim klasama, superklasama ili roditeljima klase koja se stvara. Za klasu koja se stvara kaže se da 'nasljeđuje oblik', 'počinje od', 'produžava se' ili 'podklasira' od svoje bazne, roditeljske klase. Ova klasa također je poznata kao *izravna podklasa* ili *potomak* svojih baznih klasa. Podklasna veza između klasa je prijelazna: ako je C1 podklasa od C2, a C2 podklasa od C3, onda je C1 podklasa od C3. Ugrađena funkcija issubclass(C1,C2) prihvaća dva argumenta koji su objekti klase: vraća True ako je C1 podklasa od C2; u suprotnom vraća False. Svaka klasa je zapravo podklasa od same sebe, tako da issubclass(C, C) vraća True za bilo koju klasu C. Sintaksa naredbe class ima malu, nezgodnu razliku od sintakse naredbe def. U naredbidef, zagrade su obvezatne između imena funkcije i dvotočke. Da se definira funkcija bez formalnih parametara, mora se koristiti naredba kao:

```
def ime( ):
    naredba(e)
```

U naredbi class zagrade su obvezatne ako klasa ima jednu li više roditeljskih klasa, ali su zabranjene ako ih klasa nema. Tako se za definaciju klase bez roditeljskih klasa mora koristiti naredba ovakve sintakse:

```
class ime:
    naredba(e)
```

Niz naredbi koji nije prazan, a koji slijedi naredbu class poznat je kao *tijelo klase*. Tijelo klase izvršava se odmah, kao dio procesa izvršenja naredbe class. Dok se tijelo ne izvrši, novi objekt klase ne postoji i identifikator ime nije još povezan (ili re-povezan). Konačno, valja primijetiti da naredba class ne stvara nove instance klase, nego definira skup atributa koji se dijele između svih instanci koje se iz klase stvore.

7.3 Tijelo klase

Tijelo klase je mjesto gdje se specificiraju *atributi klase*; ovi atributi mogu biti podatčani ili funkcijski objekti.

7.3.1 Atributi klasnog objekta

Atribut objekta klase obično se definira povezivanjem neke vrijednosti (npr. literala) na identifikator unutar tijela klase. Na primjer:

```
class C1:
    x = 23
print C1.x                          # ispisuje se: 23
```

Objekt klase `C1` ima atribut s imenom `x`, povezan na vrijednost cjelobrojnog literala `23`, a `C1.x` se odnosi, referencira na taj atribut. Moguće je također povezati ili odvezati atribute klase izvan tijela klase. Na primjer:

```
class C2:
      pass

C2.x = 23
print C2.x                                  # ispisuje se: 23
```

Ipak, program je čitljiviji ako se povežu, tj. stvore, atributi klase s naredbama unutar tijela klase. Bilo koji atributi klase implicitno se dijele svim instancama klase kad se te instance stvore. Naredba `class` implicitno definira neke vlastite, posebne atribute klase. Atribut __name__ je ime klase, stringovni identifikator korišten u naredbi `class`. Atribut __bases__ je n-terac objekata klase koji su dani kao roditeljske klase u naredbi `class` (ili prazan n-terac, ako nema zadanih roditeljskih, baznih klasa). Na primjer, za već stvorenu klasu `C1` vrijedi:

```
print C1.__name__, C1.__bases__             # ispisuje se: C1, (  )
```

Klasa također ima atribut __dict__, što je rječnički objekt kojeg klasa koristi za potporu (popis) svih svojih ostalih atributa. Za svaki objekt klase `C`, bilo koji objekt `x` i za bilo kakav identifikator S (osim `__name__`, `__bases__` i `__dict__`) vrijedi:

`C.S=x` je ekvivalentan s `C.__dict__['S']=x`.

Na primjer, za poznatu klasu `C1`:

```
C1.y = 45
C1.__dict__['z'] = 67
print C1.x, C1.y, C1.z                      # ispisuje se: 23, 45, 67
```

Pritom nema razlike između atributa klase koji su stvoreni unutar tijela klase, izvan tijela klase zadavanjem atributa ili izvan tijela klase eksplicitnim povezivanjem sa `C.__dict__`.

U naredbama koje su izravno u tijelu klase, reference na atribute klase moraju koristiti puno, a ne jednostavno ime kvantifikatora. Na primjer:

```
class C3:
    x = 23
    def f1(self):
        print C3.x                      # mora se koristiti C3.x, a ne samo x
```

Treba primijetiti da reference atributa (npr. izraz kao `C.S`) imaju bogatiju semantiku nego jednostavno povezivanje atributa.

7.3.2 Definicija funkcije unutar tijela klase

Većina tijela klasa uključuju naredbu `def`, jer su funkcije (u ovom kontekstu se zovu *metode*) važni atributi za objekte klase. Uz to, metoda definirana unutar tijela klase uvijek ima obvezatan prvi parametar, koji se dogovorno imenuje kao `self`, a odnosi se na instancu u kojoj se metoda poziva. Parametar `self` igra posebnu ulogu u pozivima metode. Evo primjera klase koja uključuje definiciju metode:

```
class C4:
    def pozdrav(self):
        print "Zdravo"
```

Klasa može definirati također nekoliko posebnih metoda (metoda sa imenima koja imaju dvije podcrte ispred i iza imena), a koje se odnose na posebne operacije.

7.3.3 Varijable specifične za klasu

Kada naredba u tijelu klase (ili u metodi unutar tijela) koristi identifikator koji počinje s dvije podcrte (ali ne završava podcrtama) - `__ident`, Python compiler implicitno mijenja identifikator u `_classname__ident`, gdje je `classname` ime klase. Ovo dopušta klasi korištenje privatnih imena za atribute, metode, globalne varijable i druge svrhe, bez rizika njihovog slučajnog dupliciranja negdje drugdje. Dogovorno, svi identifikatori koji počinju jednom podcrtom također se namijenjeni kao privatni, za doseg koji ih povezuje, bez obzira je li doseg unutar klase ili nije. Python compiler ne prisiljava dogovore oko privatnosti varijabli i metoda, to je ostavljeno programerima u Pythonu.

7.3.4 Dokumentacijski string

Ako je prva naredba u tijelu klase literal niza znakova (string), onda compiler povezuje taj niz znakova kao atribut dokumentacijskog stringa za tu klasu. Ovaj se atribut zove `__doc__` te je poznat kao dokumentiranje klase. Ima isto svojstvo kao već opisan početni komentar ili *pomoć* (engl. *help*) u definiciji funkcije.

7.4 Instanca klase

Neka je zadana klasa `ZiroRacun` ovako:

```
class ZiroRacun:
     "Jednostavna klasa"
     tip_racuna = "Ziro"
     def _ _init_ _(self,ime,racun):
         "Inicijalizacije nove klase"
         self.osoba = name
         self.novac = racun
     def polog(self,iznos):
         "Dodaje iznosu na računu"
         self.novac = self.novac + iznos
     def povuci(self,iznos):
         "Oduzima od iznosa na računu"
         self.novac = self.novac - iznos
     def provjeri(self):
         "Vraca iznos na racunu"
         return self.novac
```

Instance klase stvaraju se pozivom `class` objekta. Tada se istodobno stvara nova instanca i poziva `__init__()` metoda te klase (ako je u njoj definirana). Na primjer:

```
# Neka se klasa zove ZiroRacun
a = ZiroRacun("Mario", 1000.00)
# Stvara instancu 'a' i poziva ZiroRacun.__init__(a,"Mario", 1000.00)
b = ZiroRacun("Bill", 100000000000L)
# Stvara instancu 'b' i poziva ZiroRacun.__init__(b,"Bill", 100000000000L)
```

Jednom načinjeni, atributi i metode novo nastale instance dohvatljivi su korištenjem '*točka*' (.) operatora:

```
a.polog(100.00)          # Poziva ZiroRacun.deposit(a,100.00)
b.povuci(sys.maxint)     # Poziva ZiroRacun.povuci(b,sys.maxint)
```

```
i = a.ime                # Dohvaca a.ime
print a.tip_racuna       # Ispisuje tip_racuna
```

Interno, svaka instanca se stvara koristeći rječnik koji je dohvatljiv kao instanca `__dict__` atributa. Ovaj rječnik sadrži informaciju koja je jedinstvena za svaku instancu. Na primjer:

```
>>> print a.__dict__
{'novac': 1000.0, 'osoba': 'Mario'}
>>> print b.__dict__
{'novac': 100000000000L, 'osoba': 'Bill'}
```

Kad god se atributi instance promijene, ove promjene nastaju u njihovom lokalnom rječniku instance. Unutar metoda definiranih u klasi, atributi se mijenjaju kroz pridružbu `self` varijabli. Međutim, novi atributi mogu se dodati nekoj instanci u bilo kojem trenutku, kao na primjer:

```
a.jmbg = 123456    # dodaje atribut 'jmbg' u a.__dict__
```

Iako se pridružba atributima uvjek provodi na lokalnom rječniku neke instance, pristup atributu ponekad je vrlo složen. Kad god se pristupa atributu, Python interpreter (compiler) prvo pretražuje rječnik instance. Ako ne nađe traženo, onda se pretražuje rječnik objekta klase koja se koristila za stvaranje instance. Ako ni to nije urodilo plodom, onda se provodi pretraživanje roditeljskih klasa (ako ih klasa ima). Ako i to propadne, konačni pokušaj traženja atributa je pozivom `__getattr__()` metode te klase (ako je definirana). Na koncu, ako ni to ne uspije, onda se podiže `AttributeError` iznimka.

Kako je pokazano, da bi se stvorila instanca klase, treba se (poput poziva funkcije) pozvati objekt klase. Svaki poziv vraća novi objekt instance te klase: `jedna_instanca = C5()`. Ugrađena funkcija `isinstance(Obj,C)` s objektom klase kao argumentom `C` vraća `True` ako je objekt `Obj` instanca klase `C` ili bilo koje podklase od `C`. U protivnom `isinstance` vraća `False`.

7.4.1 __init__

Kada klasa ima ili nasljeđuje metodu `__init__`, poziv objektu klase implicitno izvršava `__init__` na novoj instanci kako bi se obavila inicijalizacija koja je potrebna instanci. Argumenti koji se predaju putem poziva moraju odgovarati formalnim parametrima `__init__` metode. Na primjer, u sljedećoj klasi:

```
class C6:
    def __init__(self,n):
        self.x = n
```

Evo kako stvoriti jednu instancu klase `C6`:

```
jedna_instanca = C6(42)
```

Kako je pokazano u `C6` klasi, metoda `__init__` obično sadrži naredbe koje povezuju atribute instance. Metoda `__init__` ne smije vraćati nikakvu vrijednost osim vrijednosti `None`, jer svaka druga vraćena vrijednost uzrokuje `TypeError` iznimku. Glavna svrha `__init__` je povezivanje, tj. stvaranje atributa novostvorene instance. Atributi instance također se mogu povezivati ili re-povezivati i izvan `__init__` metode. Programski kod bit će čitljiviji ako se svi atributi klasne instance inicijalno povežu naredbama u metodi `__init__`.

7.4.2 Atributi objekta instance

Jednom kada se stvori instanca, može se pristupiti njenim atributima (podacima i metodama) koristeći '*točka*' (.) operator. Na primjer:

```
i4=C4();
i4.pozdrav(  )              # ispisuje se: Zdravo
print jedna_instanca.x      # ispisuje se: 42
```

Objektu instance može se dodati neki atribut povezivanjem neke vrijednosti na referencu atributa. Na primjer:

```
class C7: pass
z = C7( )
z.x = 23
print z.x                   # ispisuje se: 23
```

Objekt instance `z` sada ima atribut koji se zove `x`, povezan na vrijednost 23, i `z.x` se odnosi na taj atribut. To povezivanje atributa omogućuje također posebna metoda `__setattr__`. Stvaranje instance implicitno definira dva atributa instance. Za svaku instancu `z`, `z.__class__` je klasa objekta kojemu `z` pripada, a `z.__dict__` je rječnik koji `z` koristi za čuvanje svih svojih ostalih atributa. Na primjer, za instancu `z` koja je upravo stvorena:

```
print z.__class__.__name__, z.__dict__     # ispisuje se: C7, {'x':23}
```

Oba ova atributa mogu se re-povezati (ali ne odvezati), iako je ovo rijetko nužno. Za bilo koji objekt instance `z`, bilo koji objekt `x`, i bilo koji identifikator `S` (osim `__class__` i `__dict__`), `z.S=x` je ekvivalentan s `z.__dict__['S']=x` (osim ako `__setattr__` posebna metoda presretne pokušaj povezivanja). Na primjer, za poznatu stvarenu instancu `z`:

```
z.y = 45
z.__dict__['z'] = 67
print z.x, z.y, z.z                         # ispisuje: 23, 45, 67
```

Nema razlike u atributima instance stvorenima s funkcijom `__init__`, zadavanjem preko atributa ili po eksplicitnom povezivanju sa `z.__dict__`.

7.4.3 Tvornička funkcija

Postoji potreba stvaranja instance različitih klasa ovisno o nekim uvjetima ili izbjegavanje stvaranja nove instance ako je postojeća još uvijek dostupna. Te bi se potrebe mogle riješiti tako da `__init__` vrati specifičan objekt, ali to nije moguće jer Python podiže iznimku kad `__init__` vrati bilo koju vrijednost osim `None`. Najbolji način implementacije fleksibilnog stvaranja objekata je koristeći običnu funkciju, a ne pozivanjem objekta klase izravno. Funkcija koja se koristi u ovu svrhu zove se *tvornička funkcija*.

Pozivanje tvorničke funkcije je fleksibilno rješenje, jer takva funkcija može vraćati postojeću instancu koja se može ponovno koristiti ili stvoriti novu instancu pozivanjem odgovarajuće klase. Recimo da postoje dvije slične klase (`Posebna` i `obicna`) i da se treba generirati jedna od njih, ovisno o argumentu. Sljedeća tvornička funkcija `Prikladna` to omogućuje:

```
class Posebna:
    def metoda(self): print "posebna"
class Obicna:
```

```
    def metoda(self): print "obicna"
def Prikladna(obicnaje=1):
    if obicnaje: return Obicna(  )
    else: return Posebna(  )

i1 = Prikladna(obicnaje=0)
i1.metoda(  )                  # ispisuje se: "posebna", kako je traženo
```

7.4.4 Brojanje referenci i brisanje instance

Sve instance imaju brojilo referenci. Ako brojilo referenci postane jednako nuli, instanca se briše. Iako postoji više načina kako izbrisati referencu, program najčešće koristi `del` naredbu za brisanje reference na objekt. Ako brisanje neke refernce učini da se brojilo referenci objekta smanji na nulu, program će pozvati `__del__()` metodu. Python-ov 'skupljač smeća' briše iz memorije sve instance koje su ostale bez referenci i na taj način oslobađa memorijski prostor drugim objektima.

7.5 Modul

U modul je moguće uključiti bilo koju valjanu datoteku s Python kôdom, te pozvati ga s naredbom `import`. Na primjer, neka je sljedeći kôd spremljen u datoteku '`primjer.py`':

```
# datoteka: primjer.py
a = 37                      # Varijabla
def A:                      # Funkcija
    print "Ja sam u A"
class K:                    # Klasa
    def B(self):
        print "Ja sam u K.B"
b = K()                     # Tvorba instance
```

Pozvati u memoriju ovakav kôd kao modul, postiže se naredbom `import primjer`. Prvi put kad se `import` koristi da se modul učita u memoriju, događaju se tri stvari:

1. Modul stvara novi prostor imena koji služi svim objektima definiranim u pripadnoj izvornoj datoteci. Ovaj prostor imena dohvaća se ako funkcije i metode definirane unutar modula koriste naredbu `global`.
2. Modul izvršava kôd koji je sadržan unutar novonastalog prostora imena.
3. Modul izvršava ime unutar pozivatelja, koje se odnosi na prostor imena modula. Ovo ime podudara se s imenom modula.

```
import primjer          # Učitava u memoriju i izvodi modul 'primjer'
print primjer.a         # Dohvaća izvodi neki član (varijablu) modula 'primjer'
primjer.A()             # Dohvaća i izvodi funkciju iz modula
c = primjer.K()         # Dohvaća i izvodi klasu iz modula, tvorba objekta
...
```

Učitavanje, uvoz (eng. *import*) višestrukih modula izvodi se tako da se iza naredbe `import` dopišu imena modula, odijeljena zarezom, na primjer ovako:

```
import socket, os, regex
```