Övning 5 - Programmeringsteknik 2012

```
0 # coding:latin
```

Sammanfattning

Fortsatte med **klasser** och kollade på **sortering** (**.sort()**). Extra fokus på **inkapsling**, **arv** och **metoderna __lt__(self, other)**, **__str__(self)** och **__init_(self, ...)__**. Vi skrev ett program som kunde hantera produkter och kunder. Vi gick igenom arv där **Produkt** och **Kund** klasserna ärvde från klassen **Affar**. Vi avslutade med att skriva en funktion som räknade ut fibonaccis summa iterativt och ett som gjorde det rekursivt.

About editing code

From http://google-styleguide.googlecode.com/svn/trunk/pyguide.html If you're editing code, take a few minutes to look at the code around you and determine its style. If they use spaces around all their arithmetic operators, you should too. If their comments have little boxes of hash marks around them, make your comments have little boxes of hash marks around them too.

The point of having style guidelines is to have a common vocabulary of coding so people can concentrate on what you're saying rather than on how you're saying it. We present global style rules here so people know the vocabulary, but local style is also important. If code you add to a file looks drastically different from the existing code around it, it throws readers out of their rhythm when they go to read it. Avoid this

Arv

- Ibland är de klasser man skriver besläktade med varandra, då kan man lägga det gemensamma i en basklass och det specifika för varje klass i subklasser, t ex kan superklassen ”geometrisk figur” sägas ha subklasserna ”cirkel”, ”rektangel” etc.

Iteration & rekursion

- Iteration
  - Iteration betyder upprepning.
  - Det innebär att utföra något med hjälp av en slinga.
  - Exempel: 5! = 1*2*3*4*5
- Rekursion
  - Att lösa ett problem genom att först lösa en mindre variant av samma problem.
  - Exempel: 5! = 5*4!
  - Man får en metod som anropar sig själv.
  - Metoden måste veta när den ska sluta, dvs det måste finnas ett basfall, t ex 0! = 1

Rich comparison

These are the so-called "rich comparison" methods. Used to implement comparison operation on objects. object.__ lt __(self, other) $\Rightarrow x < y$
object.__ le __(self, other) $\Rightarrow x <= y$
object.__ eq __(self, other) $\Rightarrow x == y$
object.__ ne __(self, other) $\Rightarrow x! = y$
object.__ gt __(self, other) $\Rightarrow x > y$
object.__ ge __(self, other) $\Rightarrow x >= y$

Problem

En ICA vill hålla reda på sina produkter och kundköp med hjälp av ett program. De vill kunna:

1. Sortera och lista produkter
2. Skriva ut kundinformation
3. Skriv ut tom fem mest lönsamma kunder

De vill också ha bra kontroll på hur produkt och kundinformation kan ändras. Slutligen vill det att både produkt och kund ska tillhöra en butik. I det här fallet kommer kommer produkterna och kunderna från ICA nortälje.

Produkt och kunddata finns lagrat på text filer. Produktfilen innehåller produktbeskrivnng, produktnummer och pris för varje produkt.

```
Milda Vaniljvisp 12 % fett/0/36
Italiensk lufttorkad skinka gris/1/75
...
Chorizo korv stekt/2047/61
```

Kundfilen innehåller lista med kundnamn följt and kundens köp under året separerade med backslash.

```
Alice/Jacobson/1713/1831/169/1120/1835/669/1541/1037/...
Maja/Alexanderson/1852/1985/1175835/669/1541/1037/428/...
...
Ella/Loveson/1730/1789/1785/2040/1691/804/1302/128/1701 ...
```

För att lösa detta skriver vi tre klasser **Affar**, **Produkt** och **Kund**, där Produkt och Kundklasserna kommer ärva från klassen Affar.. För säkerheten bestämer vi att instansvariabler ska vara privata (inkapsling) och därigenom kunda styra vilka variabler användaren kan ändra.

```
125
126 class Affar:
127     """
128     Affär(affar)
129     Klass som definierar en affär
130
131     Attributes:
132         affar    - namn på affär
133
134     Example:
135         >> a1 = Affar('ICA Nortälje')
136     """
137     def __init__(self, affar):
138         self.affar = affar
```

```
145 class Produkt(Affar):
146     """
147     Produkt(affar, beskrivning, pris)
148     Klass som definierar en produkt med beskrivning och pris
149 
150     Attributes:
151         affar           - namn på affär
152         beskrivning     - produktbeskrivning
153         nr              - produktnummer
154         pris            - pris på produkten
155 
156     Example:
157         >> p1 = Produkt('En kaffekokare',199.5)
158     """
159     def __init__(self, affar, beskrivning, nr, pris):
160         Affar.__init__(self, affar)
161         self.__beskrivning = beskrivning
162         self.__nr   = nr
163         self.__pris = float(pris)
164 
165     def __lt__(self, other):
166         # Compare less than
167         return self.__beskrivning < other.__beskrivning
168 
169     def __str__(self):
170         return ('Beskrivning: ' + self.__beskrivning + '\n'
171                 'Produktnummer: ' + str(self.__nr) + '\n'
172                 'Pris: ' + str(self.__pris)+'\n'
173                 'Affar: '+ str(self.affar)+'\n')
174 
175     def beskrivning(self):
176         return self.__beskrivning
177 
178     def nr(self):
179         return self.__nr
180 
181     def pris(self):
182         return self.__pris
```

```
201 class Kund(Affar):
202     """
203     Kund(fnamn, enamn)
204     Klass som definierar en kund
205 
206     Attributes:
207         belopp        - total belopp kund köpt för
208         fnamn         - förnamn
209         enamn         - efternamn
210         produkter     - lista med produkter kund köpt
211 
212     Example:
213         >> k1 = Kund(Mikael, Lindahl)
214     """
215     def __init__(self, fnamn, enamn, affar):
216 
217         Affar.__init__(self, affar)
218 
219         self.__belopp = 0.
220         self.__fnamn = fnamn
221         self.__enamn = enamn
222         self.__produkter = list()
223 
224     def __lt__(self, other):
225         return self.__belopp < other.__belopp
226 
227     def __str__(self):
228         return ('Förnamn: ' + self.__fnamn +'\n'
229                 'Efternamn: ' + str(self.__enamn) + '\n'
230                 'Handlat för:' + str(self.__belopp) + '\n'
231                 'Affär:' + str(self.affar) + '\n')
232     def addera(self, produkt):
233         self.__produkter.append(produkt)
234         self.__belopp +=produkt.pris()
235 
236     #def radera(self, produkt):
237     #    self.__produkter
238 
239     def fnamn(self):
240         return self.__fnamn
241 
242     def enamn(self):
243         return self.__enamn
244 
245     def produkter(self):
246         return self.__produkter
```

Exempel på att använda klasserna

```
220 p1 = Produkt('ICA Karlaplan','Omega-3',0, 99)
221 p2 = Produkt('ICA Karlaplan','Mellanmjölk',1, 5)
222 p3 = Produkt('ICA Karlaplan','Banan',2, 10)
223 produkterna = [p1, p2, p3]
224 
225 produkterna.sort()
226 for p in produkterna:
227     print(p)
228 
229 k1 = Kund('Sten', 'Andersson', 'ICA Karlaplan')
230 k2 = Kund('Ada', 'Adamsson', 'ICA Sveavägen')
231 k3 = Kund('Beda', 'Bengtsson', 'ICA Karlaplan')
232 k1.addera(p1)
233 k2.addera(p2)
234 k3.addera(p3)
235 kunderna = [k1, k2, k3]
236 
237 kunderna.sort()
238 for k in kunderna:
239    print(k)
```

ÖVERKURS: Går det att sortera efter anda attribut? Ja

```
242 kunderna.sort(key=lambda x: x.enamn())
243 for k in kunderna:
244     print(k)
```

Eller

```
247 kunderna.sort(key=lambda x: x.fnamn())
248 for k in kunderna:
249     print(k)
```

Lösa affärens uppgifter

```
264 def lasIn(fileName):
265     infil = open(fileName,'r')
266     rad = infil.readline()
267     data = list()
268     while rad != '':
269         rad = rad.rstrip('\n')
270         delar = rad.split('/')
271         data.append(delar)
272         rad = infil.readline()
273     return data
274 
275 def lasInProdukter():
276     data=lasIn('ovn5_produktlista.txt')
277     produktDict=dict()
278     for d in data:
279         tmp=Produkt('ICA Nortälje', d[0],d[1], d[2] )
280         produktDict.update({str(tmp.nr()): tmp})
281     return produktDict
282 
283 def lasInKundKop(produktDict):
284     data=lasIn('ovn5_arsinkop.txt')
285     kundLista=list()
286     for dd in data:
287         tmp=Kund(dd[0],dd[1], 'ICA Nortälje')
288         kundLista.append(tmp)
289         for d in dd[2:]:
290             kundLista[-1].addera(produktDict[str(d)])
291     return kundLista
```

0 Hämta produkter och kunder

```
286 produktDict=lasInProdukter()
287 produktLista=list(produktDict.values())
288 produktLista.sort()
289
290 kundLista=lasInKundKop(produktDict)
291
292
293
294 val=0
295 while val!=3:
296     val=int(input(('Välj:\n'
297                    ' 0 - Skriv ut produkter\n'
298                    ' 1 - Skriv ut kundinfo\n'
299                    ' 2 - Skriv ut top 5 kunder\n'
300                    ' 3 - Avsluta\n'
301                    ': ')))
302
303     if val==0:
304         iFran=int(input(('Välj produkter 0-' +
305                          str(len(produktLista)) +
306                          ' (sorterade efter beskrivning) \n'
307                          'Från: ')))
308         iTill=int(input('Till: '))
309
310         for p in produktLista[iFran:iTill+1]:
311             print(p)
312
313     elif val==1:
314         iKund=int(input(('Välj kund 0-' +
315                          str(len(kundLista)) +
316                          ': ')))
317         print(kundLista[iKund])
318
319     elif val==2:
320         #! 3 Kundranking
321         kundLista.sort(reverse=True)
322         print('Top fem kunder \n')
323         for k in kundLista[0:5]:
324             print(k)
325     elif val != 3:
326         print('Ogiltigt val!\n')
```

Iteration vs rekursion

1. A simple base case (or cases), and
2. A set of rules which reduce all other cases toward the base case.

The Fibonacci numbers are the integer sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, ...,

Iterativt

```
340 n=10
341 def fib_iter(n):
342     n1=0
343     n2=1
344     for i in range(n-1):
345         n3=n1+n2
346         n1=n2
347         n2=n3
348     return n2
```

Rekursivt

```
357
358 def fib_rec(n):
359     if n == 0:
360         return 0
361     elif n == 1:
362         return 1
363     else:
364         return fib_rec(n-1) + fib_rec(n-2)
365 print('Fibonacci iterativt:')
366 print(fib_iter(n))
367 print('Fibonacci rekursivt:')
368 print(fib_rec(n))
```