SAMMANFATTNING ÖVNING 3

DANIEL BOSK

DD13{10,11,14} Programmeringsteknik (prg), Kungliga Tekniska Högskolan*

1. Datatypen list

List är ytterligare en datatyp i Python, den kan användas för att lagra ett godtyckligt antal värden eller objekt i. T.ex. när man samlar data och har en datamängd som är större än ett (1) element.

En lista har ett antal metoder, en slags funktion definierad på objektet självt, nedan följer en sammanfattning av några av dessa. Om vi skapar en tom lista `l`, genom att skriva `l = list()` alternativt `l = []`, kan vi använda följande metoder:

Lägg till `e` som ett element sist i listan.
Lägg till listan `l2` sist i listan.
Sätt in `e` som ett element framför elementet på positionen `p`.
Returnera det lägsta möjliga index för elementet som matchar `e`.
Ta bort första förekomsten av ett element som matchar `e`.

Man kan också skapa en lista som redan innehåller några element genom koden `l = [1, 2, 3, "hej"]`.

Några exempel i Python-tolken:

```
Python 2.2.3 (#1, Jan  5 2005, 16:36:30)
[GCC 3.4.2] on sunos5
Type "help", "copyright", "credits" or "license" for more information.
>>> l = [1, 2, 3]
>>> print l
[1, 2, 3]
>>> print l[0]
1
>>> print l[2]
3
>>> l.append(5)
>>> print l
[1, 2, 3, 5]
>>> l.extend([2,3])
>>> print l
[1, 2, 3, 5, 2, 3]
>>> l.insert(0, 10)
>>> print l
[10, 1, 2, 3, 5, 2, 3]
>>> l.index(3)
3
>>> l.remove(3)
>>> l.index(3)
5
>>>
```

Date: 21 september 2010.

2. Datatypen dictionary

Ännu en datatyp hos Python, denna kan användas till att para ihop saker. T.ex. namn och telefonnummer, man kan söka på ett namn och få ut ett telefonnummer.

Om vi skapar ett dictionary `d`, genom koden `d = {}`, kan vi sedan använda följande metoder:

> Returnerar en lista med alla lagrade nycklar.
> Returnerar en lista med alla lagrade värden.
> Evalueras till sant (`True`) om nyckeln `k` finns i `d`.

För att skapa en dictionary innehållandes några par skriver man följande: `d = {"nyckel": "värde", "programmering": "python", 1 : -1}`.

Några exempel:

```
>>> d = {"hej" : "svejs", "kth" : "kungl tekn högskol"}
>>> d.keys()
['kth', 'hej']
>>> d.values()
['kungl tekn h\xf6gskol', 'svejs']
>>> "kth" in d
1
>>> "su" in d
0
>>>
```

3. Lite mer om strängar

Strängar har, likt listorna och dictionaries, också metoder. Några av dessa metoder beskrivs här:

> Delar upp strängen vid varje `c`, returnerar en lista med alla delarna.
> Returnerar en sträng där alla `c` tagits bort från början och slut av strängen.
> Samma som strip(), men behandlar enbart slutet av strängen – början lämnas orörd.

Några exempel på deras användning:

```
>>> "hej svejs i lingonskogen".split()
['hej', 'svejs', 'i', 'lingonskogen']
>>> s = "hej svejs i lingonskogen"
>>> print s
hej svejs i lingonskogen
>>> s.split()
['hej', 'svejs', 'i', 'lingonskogen']
>>> s.split('e')
['h', 'j sv', 'js i lingonskog', 'n']
>>> s = "   hej svejs i lingonskogen    "
>>> print s.strip()
hej svejs i lingonskogen
>>> print s.rstrip()
   hej svejs i lingonskogen
>>> s = "...hej svejs i lingonskogen...."
>>> print s.strip('.')
hej svejs i lingonskogen
>>> print s.rstrip('.')
...hej svejs i lingonskogen
>>>
```

4. Felhantering

Förutom den grundläggande felhantering man kan skriva själv i ett program finns det vissa fel som man inte lika enkelt kan skydda sig emot, där erbjuder Python något som kallas för exceptions. Ett *exception* är en slags signal som "sänds ut" när ett fel uppstår, detta exception kan man sedan fånga upp på ett speciellt ställe i koden och där behandla. Man hanterar exceptions enligt följande:

```
1 try:
2    # kod som kan ge upphov till exceptions
3 except <typ0>:
4    # kod som ska köras när ett exception av typ <typ0> fångas
5 except <typ1>:
6    # körs när man fångar ett exception av typ <typ1>
7 except:
8    # kod som körs för alla andra exceptions
```

Det är helt frivilligt hur många except man använder sig av, men man måste ha minst ett (1).

Några exempel:

```
Python 2.6.2 (r262:71600, Aug 12 2009, 11:11:06)
[GCC 3.3.5 (propolice)] on openbsd4
Type "help", "copyright", "credits" or "license" for more information.
>>> try:
...     int("hej")
... except:
...     print "exception"
...
exception
>>> int("hej")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hej'
>>> try:
...     int("hej")
... except ValueError:
...     print "valueerror"
... except:
...     print "exception"
...
valueerror
>>> def f(a,b):
...     return float(a)/float(b)
...
>>> f("5","3")
1.6666666666666667
>>> f("hej", "du")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
ValueError: invalid literal for float(): hej
>>> f("5","0")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
ZeroDivisionError: float division
>>> def f(a,b):
...     try:
...             return float(a)/float(b)
...     except ValueError:
...             print "valueerror"
...     except ZeroDivisionError:
```

```
...             print "zerodivisionerror"
...
>>> f("5","4")
1.25
>>> f("a","b")
valueerror
>>> f("5","0")
zerodivisionerror
>>>
```

5. Filhantering

Ibland kan man vilja "komma ihåg" saker mellan körningarna av programmet, när alla variabler försvinner när programmet avslutas, eller läsa in stora mängder data som är otympligt att mata in från tangentbordet. Där kommer filhantering lämpligt in i bilden. Vi kan skriva det vi vill komma ihåg till en fil, och sedan läsa in den nästa gång vi startar programmet.

För att öppna en fil i Python använder man sig av funktionen open(), denna funktion tar som första argument en sträng, som är sökvägen[1] till filen som ska öppnas. Det andra argumentet är sättet vi vill öppna filen på, för läsning eller för skrivning.

Open returnerar ett objekt som representerar filen. Objektet har några metoder för att kunna läsa och skriva data från respektive till filen. Om vi öppnar filen *testfil.txt* för läsning genom

```
1 fil = open("testfil.txt", "r")
```

kan vi använda oss av följande metoder för att utföra olika handlingar på filen:

Läs in hela filens innehåll till variabeln text.

Läs in hela filen, spara alla rader som separata element i listan rader.

Läs in nästa rad till variabeln rad.

När vi inte längre behöver använda filen stänger vi den med fil.close().

Om vi istället öppnar filen för skrivning genom

```
1 fil = open("testfil.txt", "w")
```

kan vi använda oss av följande metoder:

Skriv *hello file* till filen, följt av en radbrytning.

Skriv alla strängar i listan rader till filen, notera att de inte skrivs på varsin rad om de inte själva innehåller radbrytningstecken.

När vi inte längre behöver använda filen stänger vi den med fil.close().

Några exempel från terminalen och Python-tolken:

```
dbosk@my:/tmp\$ ls testfil.txt
testfil.txt: No such file or directory
dbosk@my:/tmp\$ python
Python 2.2.3 (#1, Jan  5 2005, 16:36:30)
[GCC 3.4.2] on sunos5
Type "help", "copyright", "credits" or "license" for more information.
>>> fil = open("testfil.txt", "w")
>>> fil.write("hello file\nen rad till")
>>> fil.close()
>>> \^D
dbosk@my:/tmp\$ ls testfil.txt
testfil.txt
dbosk@my:/tmp\$ python
```

[1] Sökvägen kan vara absolut eller relativ. Om den är relativ utgår man från den mapp som programfilen finns i.

```
Python 2.2.3 (#1, Jan  5 2005, 16:36:30)
[GCC 3.4.2] on sunos5
Type "help", "copyright", "credits" or "license" for more information.
>>> fil = open("testfil.txt", "r")
>>> print fil.read()
hello file
en rad till
>>> fil.close()
>>> fil = open("testfil.txt", "r")
>>> print fil.readline()
hello file

>>> print fil.readline()
en rad till
>>> print fil.readline()

>>> \^D
dbosk@my:/tmp\$
```

6. Klass och objekt

...

6.1. **Instantiering.** ...

6.2. **Instansvariabler och instansmetoder.** ...

6.3. **Konstruktor.** ...

6.4. **self.** ...

6.5. **Inkapsling.** ...

7. Programexempel

Två programexempel följer. Det första, o3.py, behandlar felhantering, listor och dictionaries. När man skriver in kommandon i terminalen, t.ex. ls eller python, kollar terminalprogrammet (kallat en shell) upp kommandot i en dictionary-liknande struktur för att veta hela sökvägen till programmet som skall köras. Detta enkla lilla program gör samma sak. Det låter användaren skriva in ett kommando, sedan kollar det upp sökvägen i ett dictionary och skriver ut den.

```
1  # en lista att spara kommandohistoriken i.
2  history = []
3  # vår dictionary som håller koll på var programmet
4  # finns för att köra ett visst kommando.
5  cmds = {"ls" : "/usr/bin/ls",
6      "cp" : "/usr/bin/cp",
7      "mv" : "/usr/bin/mv",
8      "res" : "/usr/local/bin/res",
9      "course" : "/usr/local/bin/course",
10     "python" : "/opt/sfw/bin/python",
11     "emacs" : "/usr/local/bin/emacs",
12     "vim" : "/opt/sfw/bin/vim",
13     "g++" : "/usr/sfw/bin/g++",
14     "cc" : "/usr/local/bin/cc",
15     "gdb" : "/opt/sfw/bin/gdb"}
16
17 cmd = raw_input("Enter command: ")
18 while cmd != "":
19     try:
20         print cmds[cmd]
21         # koden nedan kommer aldrig inträffa om det blir
```

```
22          # en exception i cmds[cmd], då hoppar den direkt
23          # till except nedan.
24          history.append(cmd)
25      except KeyError:
26          print "Sorry, that command is not in the dictionary."
27      except:
28          print "There is something seriously wrong here."
29      cmd = raw_input("Enter command: ")
30
31  for cmd in history:
32      print cmd,
```

Det andra programmet, O3−sten−version3.py, behandlar klasser samt in- och utmatning till fil. Det är ett bankprogram som håller ordning på en massa bankkonton, tillåter att man sätter in pengar och kollar saldot.

```
1  #
2
3  class Konto:
4
5      def __init__(self, nr, namn, saldo):
6          self.nr = nr
7          self.namn = namn
8          self.saldo = saldo
9
10     def __str__(self):
11         return str(self.nr) + '\n' + self.namn + '\n' + str(self.saldo
               ) + ' kr\n'
12
13     def deposit(self, amount):
14         self.saldo += amount
15
16     def withdraw(self, amount):
17         self.saldo -= amount
18
19
20 def lasIn():
21     infil = open('infil.txt','r')
22     rad = infil.readline()
23     kontona = list()
24     while rad != '':
25         rad = rad.rstrip('\n')
26         delar = rad.split('/')
27         nr = int(delar[0])
28         namn = delar[1]
29         saldo = float(delar[2])
30         tmp = Konto(nr,namn,saldo)
31         kontona.append(tmp)
32         rad = infil.readline()
33     infil.close()
34     return kontona
35
36 def sparaPaFil(kontona):
37     utfil = open('utfil.txt','w')
38     for i in range(len(kontona)):
39         utfil.write(str(kontona[i].nr) + '/' + kontona[i].namn +'/' +
               str(kontona[i].saldo)+ '\n')
40     utfil.close()
41
42
43 def skrivUt(kontona):
44     for i in range(len(kontona)):
45         print kontona[i]
```

```
46
47 def degPaBanken(kontona):
48     sum = 0
49     for i in range(len(kontona)):
50         sum += kontona[i].saldo
51     return sum
52
53 # Huvudprogram
54 kontona = lasIn()
55
56 val = ''
57 while val != '0':
58     print '0 - avsluta'
59     print '1 - skriv ut alla konton'
60     print '2 - skriv ut summan av insatta pengar'
61     print '3 - sätt in 50kr åt alla!'
62     val = raw_input('Ditt val: ')
63     if val == '1':
64         skrivUt(kontona)
65     elif val == '2':
66         print 'Banken har totalt ', degPaBanken(kontona), ' kr'
67     elif val == '3':
68         for k in kontona:
69             k.deposit(50)
70
71
72 sparaPaFil(kontona)
```