Övning 1 - Abstrakta datatyper

```
0 # coding: latin
```

Summering

Vi gick igenom betydelsen av **abstrakta datatyper/datastrukturer**. Detta exemplifierades genom att dels titta den abstrakta dataytyperna *stack* och *listan*. Vidare gick vi igenom användningen av abstrakta datayper genom att jämföra inläsning från fil och inläsning från URL. Vi skissade på en lösning till Lab 1 samt kolla på några egendefinerade abstrakta datatyper *fraction* and *temperatur*. Slutligen gick vi igenom functionen *iso2utf* vilken byter kodningen sträng från iso till utf.

Primitiva datatyper

Primitiva datatyper är sådana som programmeringsspråket tillhandahåller direkt. Det är enkla typer som man i princip inte kan klara sig utan, till exempel heltal, flyttal och boolska värden. De primitiva datatyperna kan i många språk kombineras till mer komplexa typer, som då inte längre är primitiva.

Abstrakta typer

En abstrakt datatyp är inom datorprogrammering en datatyp som förutom att definiera själva datats art (datatypen) även definierar de operationer som är tillåtna på detta data. Man säger att en abstrakt datatyp kapslar in såväl datat som operationer på detta data.

En abstrakt datatyp kan sägas vara en teoretisk beskrivning av en klass i ett objektorienterat programmeringsspråk. Implementation av abstrakta datatyper förutsätter dock inte att det programmeringsspråk man väljer har stöd för objektkonstruktioner - det går bra att skapa abstrakta datatyper i såväl funktionella som imperativa programmeringsspråk så länge dessa är strukturerade och ger stöd för funktionsanrop och godtyckliga definitioner av datastrukturer.

En väldefinerad abstrakt datatyp kännetecknas bland annat av att den har ett gränssnitt (ett API) som tillåter en användare att använda denna utan att behöva göra några antaganden om hur den underliggande implementationen är utförd. Ett exempel på en abstrakt datatyp kan till exempel vara en Lista. En lista består av element (dvs. listans rader), och möjligheten att lägga till element och att stryka bort dem. APIet för en abstrakt datatyp Lista kan således vara

- Create (Skapa)
- Add (Lägg till element)
- Get (Hämta ett visst element från listan)
- Remove (Ta bort/stryk över element)
- Destroy (Riv sönder/Kasta bort listan)

Själva datatypen är alltså Lista, och de definierade operationerna är Create, Add, Get, Remove och Destroy. (http://sv.wikipedia.org/wiki/Datatyp)

Ignorera detta:

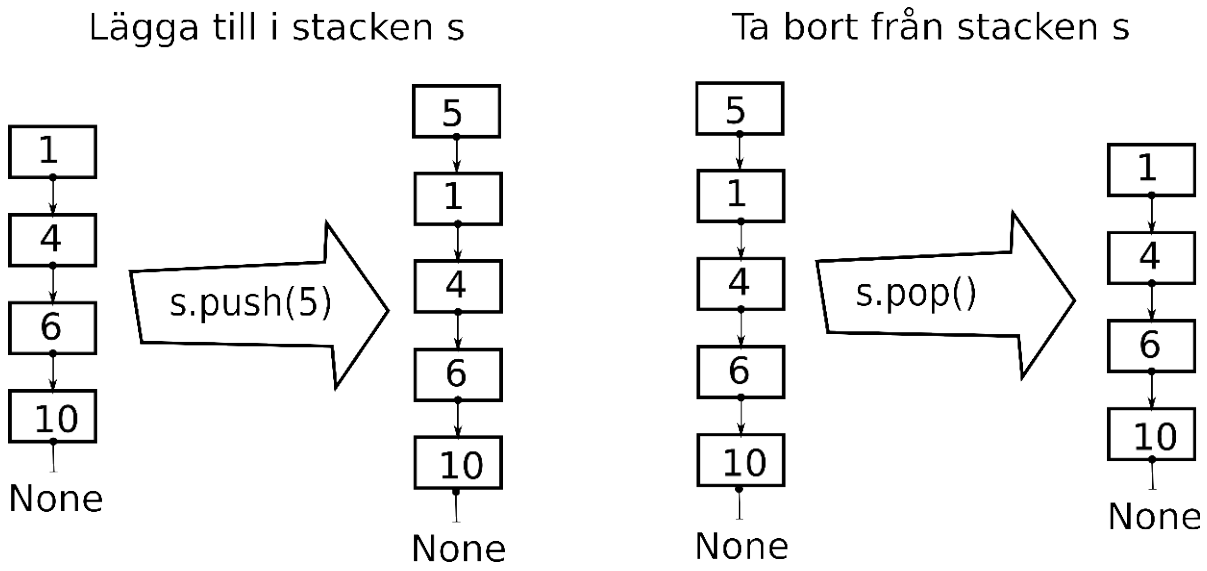
```
59 import sys, os
60 sys.path.append(os.getcwd())
61 from show_fig import show_fig
```

1 Stacken

- Rita bilder som visar hur det ser ut när man lägger in ett nytt element i stacken.
- Rita bilder som visar hur det ser ut när man tar bort ett element från stacken.
- Skriv kommentarer till koden nedan!
- Skriv ett program som använder två stackar för att lösa något problem.

In och uttag från stacken

```
74
75 show_fig('stack_example.png')
```



```
77
78 # -*- coding: utf-8 -*-
79 """Classroom exercise 1, example 1."""
80
81 class Stack:
82     """Representation av en Stack som lagrar objekt enligt principen
83        sist in först ut."""
84     def __init__(self):
85         """Skapa en tom stack"""
86         self.top = None
87
88     def push(self,x):
89         """Lägga till ett element"""
90         ny = Node(x)
91         ny.next = self.top
92         self.top = ny
93
94     def pop(self):
95         """Ta bort ett element och returnera det."""
96         x = self.top.value
97         self.top = self.top.next
98         return x
99     def isempty(self):
100        """Testa om stacken är tom"""
101        ## The naïve solution (below) is unnecessary:
102        # if self.top == None:
103        #    return True
104        # else:
105        #    return False
106        return self.top == None
107
108 class Node:
109     """En nod i en länkad structur (t.ex. en Stack)."""
110     def __init__(self, x):
111         """Skapa en nod och ge den ett värde."""
112         self.value = x
113         self.next = None
```

Med help kan vi nu skriva ut en kortfattad beskrivning av programmet

```
116 print(help(Stack))
```

```
Help on class Stack in module pyreport.main:

class Stack
 |  Representation av en Stack som lagrar objekt enligt principen
 |  sist in först ut.
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      Skapa en tom stack
 |
 |  isempty(self)
 |      Testa om stacken är tom
 |
 |  pop(self)
 |      Ta bort ett element och returnera det.
 |
 |  push(self, x)
 |      Lägga till ett element
```

```
None
```

Exempel användning av stacken

Problem: Lagra listan i en stack så att första elemented i listan ligger högst up på stacken.

```
123 data = [1,2,3,4]
124 s1 = Stack()
125 for number in data:
126     print "pushing on s1:", number
127     s1.push(number)
```

```
pushing on s1: 1
pushing on s1: 2
pushing on s1: 3
pushing on s1: 4
```

```
129
130 s2 = Stack()
131 while not s1.isempty():
132     number = s1.pop()
133     print "pushing", number, " on stack s2"
134     s2.push(number)
```

```
pushing 4  on stack s2
pushing 3  on stack s2
pushing 2  on stack s2
pushing 1  on stack s2
```

```
135
136 print "s1 isempty?", s1.isempty()
```

```
s1 isempty? True
```

```
136 print "s2 isempty?", s1.isempty()
```

```
s2 isempty? True
```

Listan - exempel på en datastruktur i Python

Data: ett antal tal, tecken eller objekt

Metoder och operationer:

- append(x) Lägger till x sist i listan.
- insert(i,x) Lägger till x på plats i.
- pop() Plockar bort sista elementet från listan.
- sort() Sorterar listan.
- reverse() Vänder på listan.
- index(x) Returnerar index för första förekomsten av x.
- count(x) Räknar antalet x i listan.
- remove(x) Tar bort första förekomsten av x.
- [i] Returnerar i:te elementet i listan.
- [:] Plockar ut en dellista.
- Med operatorn + lägger man ihop två listor
- Med operatorn * flerdubblar man en listan.
- in Kollar om ett element finns med i listan.
- len Returnerar listans längd.

2. Inläsning och utskrift

```
162
163 raden = raw_input("Ge en rad tal: ")
```

```
Ge en rad tal:
```

```
163 raden_i_delar = raden.split()
164 for tal in raden_i_delar:
165     print tal
```

```
1
2
```

3. Läs från fil och sortera

```
170 infil = open("bilar.txt")
171
172 lista = []
173 for rad in infil:
174     ordet = rad.strip()
175     lista.append(ordet)
176 lista.sort()
177
178 print lista
```

```
['Bmw', 'Fiat', 'Porsche', 'Saab', 'Volvo']
```

4. Jämföra inläsning från text mot inläsning från web

```
182 import urllib
183 import time
184
185 def lasFilSortera(filnamn):
186     infil = open(filnamn)
187     lista = []
188     for rad in infil:
189         ordet = rad.strip()
190         lista.append(ordet)
191     lista.sort()
192     return lista
193
194 def lasURLSortera(url):
195     infil = urllib.urlopen(url)
196     lista = []
197     for rad in infil:
198         ordet = rad.strip()
199         lista.append(ordet)
200     lista.sort()
201     return lista
202
203
204 innan = time.time()
205 lista1 = lasFilSortera("tred.txt")
206 efter = time.time()
207 print "Tid för filläsning:", efter - innan
```

```
Tid för filläsning: 0.000530958175659
```

```
209
```

```
210 innan = time.time()
211 lista2 = lasURLSortera("http://www.csc.kth.se/utbildning/kth/kurser/DD1320/tild\
212 all/ovn/tred.txt"                    )
213 efter = time.time()
214 print "Tid för webbläsning:", efter - innan
```

```
Tid för webbläsning: 0.0113170146942
```

5. Egna datastrukturer

I kursen ska ni själva implementera datastrukturer, oftast genom att skriva egna klasser i Python.

Lab 1

1. Skriv en egen klass som representerar ett gympapass. Klassen ska ha attributen lokal, tid, passtyp, rum, ledare och platser. Klassen ska ha minst fem metoder, bland dem metoden__str__
2. Skriv en funktion som läser in data från filen fsdata.txt, skapar gympapass-objekt, och lagrar objekten i en lista. (lista = []).
3. Skriv ett huvudprogram där man kan söka efter önskat pass.

Skissa på hur klassen skulle kunna se ut. Vilka metoder och vilka attribut skulle ni ha?

Abstrakt datatyp för temperatur

- Temperatur kan anges i olika skalor. En abstrakt datatyp minskar risken för missförstånd. Definiera klassen temp med metoderna setK, setC, setF och getK, getC, getF.
- Använd sedan klassen i ett program som läser in utomhustemperaturen (Celsius) och skriver ut temperaturen så att en amerikan förstår (Fahrenheit).

Filen temp.py:

```
249
250 """Temp - an abstract datatype for temperature
251 Temp is used to represent temperature in multiple scales.
252 """
253
254 ZERO_C = 273.15  # 0 C == 273.15 K
255
256 ZERO_F = 459.67*5/9  #0 F == 255.37222... K
257
258
259 # Conversion between Kelvin, degrees Celsius, degrees Fahrenheit
260 # [C] = [K] + 273.15            [K] = [C] - 273.15
261 # [C] = ([F] - 32)*5/9          [F] = [C]*9/5 + 32
262 # [K] = ([F] + 459.67)*5/9      [F] = [K]*9/5 - 459.67
263
264 class Temp:
265
266     def __init__(self):
267         self.K = 0                      #Temperatur i Kelvin
268
269
270     def setK(self,K):
271         self.K = K
272
273     def setC(self,C):
274         self.K = ZERO_C+C
275
276     def setF(self,F):
277         self.K = ZERO_F+5*F/9
278
279     def getK(self):
280         return self.K
281
282     def getC(self):
283         return self.K-ZERO_C
284
285     def getF(self):
286         return (self.K-ZERO_F)*9/5
```

Import temp

```
282 from temp import Temp
283 t = Temp()
284 c = input("Vad är temperaturen i Celsius? ")
```

```
Vad är temperaturen i Celsius?
```

```
285 t.setC(c)
286 print "Amerikaner kallar det", str(t.getF()) + "F"
```

```
Amerikaner kallar det 69.8F
```

```
286 # Vad är temperaturen i Celsius? 1
```

6. Fraction-klassen: Ett exempel från Miller & Ranums bok, s 35:

```
292 # encoding: Latin1 (tillåt å, ä och ö)
293 # Fraction-klassen
294
295 class Fraction:
296
297     def __init__(self, top, bottom):
298         self.num = top
299         self.den = bottom
300
301     def __str__(self):
302         return str(self.num)+"/"+str(self.den)
303
304     def show(self):
305         print self.num,"/",self.den
306
307     def __add__(self,otherfraction):
308         newnum = self.num*otherfraction.den + self.den*otherfraction.num
309         newden = self.den*otherfraction.den
310         return Fraction(newnum,newden)
311
312     def __cmp__(self,otherfraction):
313         num1 = self.num*otherfraction.den
314         num2 = self.den*otherfraction.num
315         if num1 < num2:
316             return -1
317         else:
318             if num1 == num2:
319                 return 0
320             else:
321                 return 1
```

Svara på följande frågor:

1. Vad är relationen mellan en klass och ett objekt?
2. Hur skriver man för att skapa ett Fraction-objekt?
3. Vad gör __init__?
4. Vilka av metoderna ovan har returvärden?
5. Vad är self?
6. När anropas metoden __str__?
7. När anropas metoden __add__?
8. När anropas metoden __cmp__?
9. Hur används parametern otherfraction i __add__ och __cmp__?

Svar:

1. objekt är instantieringar av klasser, engelskan “instance”
2. fr=Fraction(2,3)
3. anropas när ett object skapas, sätter attributen top och bottom
4. Det har: __str__, __add__ and __cmp__
5. representation av objektet inne i klassen
6. vis str(fr) och print(fr)
7. när vi summerar två object fr+fr
8. när vi jämför två object fr1<fr2
9. reference till fr2

7. Text konvertering

Följande funktion konverterar en sträng från teckenkodningen “iso8859-1” till “utf-8”. Gör funktionen generellare så att den kan konvertera från en valfri given teckenkodning till en annan.

```
379
380 def iso2utf(strang):
381     """
382     Konverterar från iso8859 till utf-8
383     """
384     ustrang = strang.decode('iso8859-1')
385     return ustrang.encode('utf-8')
```

Konvertering av valfri teckenkodning

```
393 def convert_caracter_set(strang, current_set, target_set):
394     """
395      Convert from one caracter set to another
396
397      Input:
398          string - string to convert
399          current_set - current caracter set as a string, e.g. iso8859
400          target_set - target caracter set as a string, e.g. utf-8
401
402     """
403     ustrang = strang.decode(current_set)
404     return ustrang.encode(target_set)
```