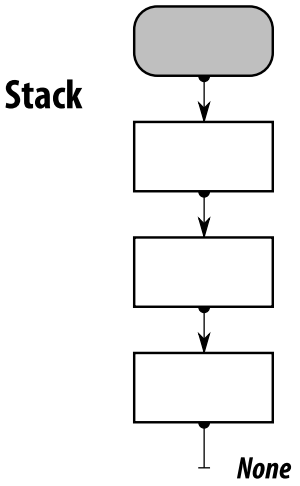
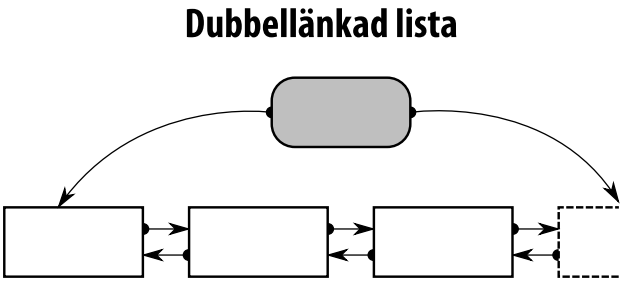
Per Sedholm
DD1320 (tilda12)
2012-08-29

Övning 1

Abstrakta datatyper

1. Stacken






```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
"""Classroom exercise 1, example 1."""

class Stack:
    """A representation of a last-in-first-out (LIFO) stack of
    objects."""

    def __init__(self):
        """Create an empty stack."""
        self.top = None

    def push(self,x):
        """Push an item onto the top of this stack."""
        ny = Node(x)
        ny.next = self.top
        self.top = ny

    def pop(self):
        """Remove the top object from this stack and return it."""
        x = self.top.value
        self.top = self.top.next
        return x

    def isempty(self):
        """Test if this stack is empty."""
        ## The naïve solution (below) is unnecessary:
        # if self.top == None:
        #     return True
        # else:
        #     return False
        return self.top == None

class Node:
    """An entry in a linked structure (e.g. Stack)."""
    def __init__(self, x):
        """Create a node linked to the given element."""
        self.value = x
        self.next = None
```

```
$ python3 -i ovn1-01.py
>>> help(Stack)
Help on class Stack in module __main__:

class Stack(builtins.object)
 |  A representation of a last-in-first-out (LIFO) stack of
 |  objects.
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      Create an empty stack.
 |
 |  isempty(self)
 |      Test if this stack is empty.
 |
 |  pop(self)
 |      Remove the top object from this stack and return it.
 |
 |  push(self, x)
 |      Push an item onto the top of this stack.

>>> help(Node)
Help on class Node in module __main__:

class Node(builtins.object)
 |  An entry in a linked structure (e.g. Stack).
 |
 |  Methods defined here:
 |
 |  __init__(self, x)
 |      Create a node linked to the given element.
```

Användning

```
data = "pining for the fjords"
s1 = Stack()
for word in data.split():
    print("pushing on s1:", word)
    s1.push(word)

s2 = Stack()
while not s1.isempty():
    word = s1.pop()
    print("pushing '" + word + "' on stack s2")
    s2.push(word)
```

```
$ python3 ovn1-01.py
pushing on s1: pining
pushing on s1: for
pushing on s1: the
pushing on s1: fjords
pushing 'fjords' on stack s2
pushing 'the' on stack s2
pushing 'for' on stack s2
pushing 'pining' on stack s2

$ python2 ovn1-01.py
('pushing on s1:', 'pining')
('pushing on s1:', 'for')
('pushing on s1:', 'the')
('pushing on s1:', 'fjords')
pushing 'fjords' on stack s2
pushing 'the' on stack s2
```

```
pushing 'for' on stack s2
pushing 'pining' on stack s2
```

2. Inläsning och utskrift

```
#row = raw_input("Ge en rad tal: ")   ### python2
row = input("Ge en rad tal: ")

parted_row = row.split()
for num in parted_row:
    print num

# Ge en rad tal: 5 10 15 20 25 42
# 5 // 10 // 15 // 20 // 25 // 42
```

```
$ python2
>>> row = raw_input("Prompt> ")
Prompt> a single line of text
>>> expr = input("Prompt> ")
Prompt> 4 + 5

$ python3
>>> row = input("Prompt> ")
Prompt> a single line of text
>>> expr = eval(input("Prompt> "))
Prompt> 4 + 5
```

3. Läs från fil; sortera

```
input = open("tred.txt") # See ex.4 for URL

alist = []
for line in input:
    word = line.strip()
    alist.append(word)
alist.sort()

print(", ".join(alist))

# $ ln -s /afs/nada.kth.se/info/DD1320/www-csc/tilda12/ovn/tred.txt
# $ python3 ovn1-03.py
# Abies, Acer, Aesculus, Alnus, Betula, Carpinus, Corylus, Fagus,
# Fraxinus, Juniperus, Larix, Malus, Picea, Populus, Populus, Prunus,
# Quercus, Salix, Salix, Sambucus, Sorbus, Syringa, Taxus, Tilia,
# Ulmus
#
# $ fmt tred_sv.txt     ### egentligen radseparerad
# Banan Apelsin Äpple Plommon Öl Körsbär Åkerbär
# $ python3 ovn1-03.py   ### med tred_sv.txt
# Apelsin, Banan, Körsbär, Plommon, Äpple, Åkerbär, Öl
```

```
lst = ['Ärlig', 'Caesar', 'Adam', 'Bertil', 'Östen', 'Åke']
lst.sort()
print ", ".join(lst)
# Adam, Bertil, Caesar, Ärlig, Åke, Östen
#                       ^^^^^^^^^^

import locale
locale.setlocale(locale.LC_ALL, 'sv_SE.UTF-8') ## alt. LC_COLLATE
# returns 'sv_SE.UTF-8'
lst = ['Ärlig', 'Caesar', 'Adam', 'Bertil', 'Östen', 'Åke']
# lst.sort(cmp=locale.strcoll) ## Old-style sorting (Python <= 2.4)
lst.sort(key=locale.strxfrm)
print ", ".join(lst)
# Adam, Bertil, Caesar, Åke, Ärlig, Östen

import locale
```

```
locale.setlocale(locale.LC_ALL, 'en_US.UTF-8')
lst = ['Ärlig', 'Caesar', 'Adam', 'Bertil', 'Östen', 'Åke']
lst.sort(key=locale.strxfrm)
print ", ".join(lst)
# Adam, Åke, Ärlig, Bertil, Caesar, Östen
```

4. Mäta tid; läsa URL

```
# -*- coding: utf-8 -*-

import urllib.request
import time

def fetchSortedFile(filnamn):
    """Read file with the given name, return words as sorted list."""
    wfile = open(filnamn)
    alist = []
    for line in wfile:
        word = line.strip() ### remove whitespace
        alist.append(word)
    alist.sort()
    return alist

def lasURLSortera(url):
    """Fetch file from given URL, return words as sorted list."""
    wwwfile = urllib.request.urlopen(url)
    alist = []
    for line in wwwfile:
        word = line.strip()
        alist.append(word)
    alist.sort()
    return alist


before = time.time()
lst1 = fetchSortedFile("tred.txt")
after = time.time()
print("Tid för filläsning:", after - before)

before = time.time()
lst2 = lasURLSortera("http://www.csc.kth.se/utbildning/kth/kurser/DD"
                     "1320/tilda12/ovn/tred.txt") ### note linebreak
after = time.time()
print("Tid för webbläsning:", after - before)

# csc-datan$ python3 ovn1-04.py
# Tid för filläsning: 0.00157403945923
# Tid för webbläsning: 0.0176341533661
#
# csc-datan$ python3 ovn1-04.py
# Tid för filläsning: 0.00156807899475
# Tid för webbläsning: 0.0101079845428
#
# hemma-datan$ python ovn1-04.py
# Tid för filläsning: 6.139282982929929e-05
# Tid för webbläsning: 0.0269299299108
```

5. Abstrakt datastruktur för temperatur

Observera att uppgiften går att lösa utan objektorientering:

`templib.py`

```
def CtoF(c):
    return c*5/9 + 32

def FtoC(f):
    return (f - 32)*5/9
```

```
import templib

c = 37.0
f = templib.CtoF(c)
print(c, "°C =", f, "°F =", templib.FtoC(f), "°C")

from templib import CtoF, FtoC
c = 37.0
f = CtoF(c)
print(c, "°C =", f, "°F =", FtoC(f), "°C")

# 37.0 °C = 52.5555555556 °F = 11.4197530864 °C
```

I detta fall skulle vi dock använda en abstrakt datatyp.

`temp.py`

```
"""Temp - an abstract datatype for temperature

Temp is used to represent temperature in multiple scales.
"""

zeroC = 273.15      # 0°C == 273.15 K
zeroF = 459.67*5/9  # 0°F == 255.37222... K

# Conversion between Kelvin, degrees Celsius, degrees Fahrenheit
# [°C] = [K] + 273.15               [K] = [°C] - 273.15
# [°C] = ([°F] - 32)*5/9            [°F] = [°C]*9/5 + 32
# [K]  = ([°F] + 459.67)*5/9        [°F] = [K]*9/5 - 459.67

class Temp:
    def __init__(self):
        self.K = 0   # Temperature in Kelvin

    def setK(self,K):
        self.K = K

    def setC(self,C):
        self.K = zeroC + C

    def setF(self,F):
        self.K = zeroF + 5*F/9

    def getK(self):
        return self.K

    def getC(self):
        return self.K - zeroC

    def getF(self):
        return (self.K - zeroF)*9/5
```

```
from temp import Temp

t = Temp()
c = eval(input("Vad är temperaturen i Celsius? "))
t.setC(c)
print "Amerikaner kallar det", str(t.getF()) + "°F"

# Vad är temperaturen i Celsius? 100
# Amerikaner kallar det 212.0°F
```

Vi fick dock väldigt många decimaler för 37°C – Representationsfel. *Använd formaterad utskrift!*

Representationsfel kan ge många decimaler:

```
>>> 0.1+0.2
0.30000000000000004
```

I bas 2 (binärt) så kan värdet 0.1 inte representeras exakt. (Jfr. $\frac{1}{3}$ decimalt.)

$$0.1_{10} = 0.0001100110011001100110011001100110011001100110011001100110011010\ldots_2$$

Fraction-klassen

```
class Fraction:
    def __init__(self, top, bottom):
        self.num = top
        self.den = bottom

    def __str__(self):
        return str(self.num) + "/" + str(self.den)

    def show(self):
        print(self.num, "/", self.den, end=" ")

    def __add__(self,otherfraction):
        newnum = self.num*otherfraction.den + \
                 self.den*otherfraction.num
        newden = self.den * otherfraction.den
        return Fraction(newnum,newden)

    def __lt__(self,otherfraction):
        ## Cross-multiply by bd: a/b < c/d --> ad < bc
        num1 = self.num*otherfraction.den
        num2 = self.den*otherfraction.num
        return num1 < num2

    def __eq__(self,otherfraction):
        ## See __lt__
        num1 = self.num*otherfraction.den
        num2 = self.den*otherfraction.num
        return num1 == num2
```

```
f = Fraction(6, 8)
g = Fraction(10, 16)
h = f + g

f.show()
# print "+",   ### python2: trailing space supresses newline
print("+", end=" ")
g.show()
print("=", end=" ")
h.show()
print()

if f < g:
    print(str(f), "<", str(g))
else:
    print(str(f), ">=", str(g))

# 6 / 8 + 10 / 16 = 176 / 128
# 6/8 < 10/16
```

Pythons klass *Fraction* (modulen *fractions*) har funktionen *gcd* för att beräkna största gemensamma nämnare (*greatest common divisor*, GCD). Det använder Euklides algoritm, som i Python kan skrivas:

```
def gcd(a, b):
    while b:
        a, b = b, a%b
    return a
```

7. Teckenkodning

Det är stor skillnad mellan Python2 och Python3. I Python2 finns datatyperna *str, unicode*, som bägge kan vara *raw* eller inte. I Python3 finns typerna text (klassen *str*) och (binär) data (*bytes*).

Se http://docs.python.org/release/3.0.1/whatsnew/3.0.html#text-vs-data-instead-of-unicode-vs-8-bit

```
named_unicode = "\N{section sign}\N{digit four}\u0032 §42\\"
raw_string    = r"\N{section sign}\N{digit four}\u0032 §42\\"
print(str(type(named_unicode)) + ": " + named_unicode)
print(str(type(raw_string)) + ": " + raw_string)

# <class 'str'>: §42 §42\
# <class 'str'>: \N{section sign}\N{digit four}\u0032 §42\\

ustr = "Å ·"
print(ustr == "\u00C5 \u2022")
# True

#bstr = b"Å ·" # SyntaxError: bytes can only contain ASCII literal characters.
bstr = b"\xc3\x85 \xe2\x80\xa2"

print("ustr:", str(type(ustr)))
for c in ustr:
    print("unicode character", c, "code point", ord(c), "hex", hex(ord(c)))

#ustr: <class 'str'>
# unicode character Å code point 197 hex 0xc5
# unicode character   code point 32 hex 0x20
# unicode character · code point 8226 hex 0x2022

print("bstr:", str(type(bstr)))
for b in bstr:
    print("byte", b, "hex", hex(b))

# bstr: <class 'bytes'>
# byte 195 hex 0xc3
# byte 165 hex 0xa5
# byte 32 hex 0x20
# byte 226 hex 0xe2
# byte 128 hex 0x80
# byte 162 hex 0xa2

print(bstr.decode() == ustr)
# True
print(ustr.encode() == bstr)
# True
```

```
def iso2utf(strang):
    """
    Konverterar från iso8859 till utf-8
    """
    ustrang = strang.decode('iso-8859-1')
    return ustrang.encode('utf-8')

isostr = b'R\xe4ksm\xf6rg\xe5s';
#isostr.decode()
# UnicodeDecodeError: 'utf8' codec can't decode bytes in
#      position 1-3: invalid data

#iso2utf(isostr)
# b'R\xc3\xa4ksm\xc3\xb6rg\xc3\xa5s'
utfstr = iso2utf(isostr).decode();
print(isostr, "->", utfstr)
```

8. (extra) Kopiera referenser

```
a = [1, 7, 10, 13]
ac = a[:]
ac[2] = -1;

print "-" * 40
print "a  = ", a
print "ac = ", ac

# ----------------------------------------
# a  =  [1, 7, 10, 13]
# ac =  [1, 7, -1, 13]

class Person:
    def __init__(self, name):
        self.__name = name
    def change_name(self, newn):
        self.__name = newn
    def __repr__(self):
        return "<" + self.__class__.__name__ + ":" + self.__name + ">"

a = [ Person("Alice"), Person("Bob"), Person("Charlie") ]
ac = a[:]
ac[1] = Person("Duncan")
ac[2].change_name("Eric")
print "-" * 40
print "a  = ", a
print "ac = ", ac

# ----------------------------------------
# a  =  [<Person:Alice>, <Person:Bob>, <Person:Eric>]
# ac =  [<Person:Alice>, <Person:Duncan>, <Person:Eric>]
```