Övning 3 - Programmeringsteknik 2012

```
0  # coding: latin
```

Sammanfattning

Gick igenom **lists**, **dictionaries**, **felhantering**, **strängjämförelser**, **split()**, **rstrip()**, och **filer** (läsning och skrivning till fil)

1. Listor

- Dataypen list är en numrerad lista med värden. Exempel l=[1,3,'hej'] eller l=[[1,3],4,5,[7,8]]
  - Varje värde kallas för ett element
  - Värdets position kallas för ett index.
  - En lista med N element har index från 0 upp till N-1.
- Skapa tom genom att skriva l=list() eller l=[]

List-manipulering

append(value)

```
30 l=['hej']
31 l.append('då')
32 print('Tillagt "d": ', l)     # l=['hej', 'då']
```

eller

```
34 l.append(['ros', 4])
35 print('Tillagt ["ros",4]: ', l)     #l=['hej','då', ['ros',4]] (lista i lista
       )
```

extend(value)

```
37 l.extend(['ros',4])
38 print('Utökat med ["ros",4]: ',l)     # l=['hej','då', ['ros',4], 'ros', 4]
```

insert(position, value)

```
40 l.insert(1, 1)
41 print('Satt in 1 på plats 1: ', l)     # l=['hej', 1, 'då', ['ros',4], 'ros',
       4]
```

index(value)

```
44 l.index('då')  # writes 2, om det inte finns ges ett felmeddelande
```

remove(value)

```
45 l.remove('då')
46 print('"Då" borta: ', l)      # l=['hej', 1, ['ros',4], 'ros', 4]
47
48                             # felmeddelande om 'hej' inte finns.
```

- Använd len för att få fram längden

Iterera igenom lista med **for**

```
52 l=[ [1, 3], 4, 5, [7, 8] ]
53 for val in l: print('Värde: ', val)
```

Eller på detta sättet;

```
55 for i in range(len(l)): print('Värde: ',l[i], 'index: ',i)
```

Matris

- Matris, lista i lista, matris[i][j] i=rad, j=kolumn
- Dubbla for-slingor krävs vid t ex utskrift

```
matris=[[1,2],
        [4,3]]
print(matris[0])    # [1,2]

print(matris[0][0]) # 1

print(matris[0][1]) # 2
```

Dubbla for-slingor för utskrift

```
print('Matrisutskrift')
for i in range(len(matris)):         # iterera över antal rader

    for j in range(len(matris[0])): # iterera över antal kolumner

        print(matris[i][j])
```

2. Dictionary

- En oordnad samling värden ordnade par

```
d=dict(rod=2,gul=1, bla=4)
```

eller

```
d={'röd':2,'gul':1, 'blå':4}
print('Värde för nyckel "röd": ',d['röd'])
```

- Består av en nyckel och ett värde
- Nyckel kan vara alla "ej icke muterbara" variabler (objekt som inte går att ändra efter att de skapas) främst int, str, bool

```
d={10:2,'gul':1, True:4}
print('Värde för nyckel 1:', d[1])
print('Värde för nyckel True:', d[True])
```

- Skapa tom genom att skriva d=dict() eller d={}
- Använd len för att antal ordnade par

Dict-manipuleringar

lägg till 1

```
d['h']=2
print('"h":2 tillagt: ',d)
```

lägg till 2

```
d.update({'h':2, 'f':5})
print(" 'k':5 och 'f':5 tillagt: " ,d)
```

del, ta bort värden

```
del d[True]
print('Nyckel True borttaget: ', d)     # d={'gul':1, True:4}
```

iterera över värden

```
for val in d.values():
    print('Val: ', val)
```

iterera över nycklar

```
for key in d.keys():
    print('Key',key)
```

iterara över värden och nycklar

```
for val, key in d.items():
    print('Val key: ', val, key)
```

3. Felhandering

Varför behövs felhantering?

```

listLangd=int(raw_input('Längd på listan: '))
print(list(range(listLangd)))
 #! Om användare skriver in 7.5, eller 'tio' kommer vi få ett
 #! exekveringsfel och programmet avbryts. Om vi istället använder
 #! oss av **try** and **except** kan vi
 #!
try:
    listLangd=int(input('Längd på listan: '))
    print(list(range(listLangd)))
except ValueError:
    print('Längden på listan måste vara ett heltal!')
except:
    print('Annat fel')   # Fångar alla andra fel
```

- Ett exekveringsfel i python kallas för ett exception.
- **Det finns många fördefinierade exception t ex**
  - **ValueError**, t e x om en sträng inte kan tolkas som ett tal vid konvertering
  - **ZeroDivisionError**, uppstår t ex vid division med 0
  - **TypeError**, Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch.
  - **SyntaxError**, raised when the parser encounters a syntax error. This may occur in an import statement, in an exec statement, in a call to the built-in function eval() or input(), or when reading the initial script or standard input (also interactively). När ett exception uppstår kan man välja mellan att "kasta det" eller att hantera det
- **Kasta**
  - Som standard kastas felet till anropande metod om inget skrivs.
  - Om man har flera anropsnivåer sker detta hela vägen upp till huvudprogrammet, om man inte hanterar felet där heller kraschar programmet.
- **Hantera**
  - Med try: inkapslar man den "känsliga" koden.
  - Med except: fångar man det exception som skett samt skriver eventuellt ett felmeddelande, p s s slipper man exekveringsfel.

```
    # >>> int('a')
    # Traceback (most recent call last):
    #  File "<console>", line 1, in <module>
    # ValueError: invalid literal for int() with base 10: 'a'
    # >>> range('s')
    #  Traceback (most recent call last):
    #    File "<stdin>", line 1, in <module>
```

```
#  TypeError: 'str' object cannot be interpreted as an integer
#
# >>> print('hello world)
#   File "<console>", line 1
#     print('hello world)
#                        ^
#  SyntaxError: EOL while scanning string literal
```

Variabler: kol, rad

```
import sys
def rektangel(rad, kol):
    for i in range(rad):
        s = ''
        for j in range(kol):
            s = s +'*'    # Adderar strängar

        print( s )

finnished=False
while not finnished:
    try:
        rad = int(input('Hur många rader? '))
        kol = int(input('Hur många kolumner? '))
        rektangel(rad,kol)
        finnished=True
    except:
        print "Error:", sys.exc_info()[0]
```

Exempel

```
# Hur många rader? 4.5
# Hur många kolumner? 'r'
# Error: <type 'exceptions.ValueError'>
```

4. Strängjämförelser

```
print('Ada' < 'Beda')    # True

print('ada' < 'Beda')    # False

print('Ada'.upper())     # 'ADA'
```

5. Funktionen split()

str.split(sep), Return a list of the words in the string, using sep as the delimiter string.

```
string='fdsf'
l=string.split('f')
print('"fdsf".split()=', l)
```

6. Funktionen rstrip()

str.rstrip(chars), return a copy of the string with chars removed.

```
'hej då\n hejsan\n'.rstrip('\n') # blir 'hej då hejsan'
```

eller

```
'hej då\n hejsan\n'.rstrip('ej') # blir 'hej då\n hejsan\n'

'hej da\n hejsan\n'.rstrip('ejsan\n') # blir 'hej da\n h'
```

7. Filer

- Vill man spara data mellan programkörningar måste man lagra data på fil. Läsning från fil har många likheter med att läsa från tangentbordet och skrivning till fil har många likheter med att skriva till skärmen.
- Man öppnar en fil för läsning / skrivning med funktionen open().
- När man arbetar klart med filen stänger man den för vidare inläsning / utskrift med close()
- **readline()**, läser in en rad varje gång den används
- **readlines()**, läser in all rader och lägger dem i en lista.

Före

```
# Infil
# Rad 1: hej hopp
# Rad 2: slask smurf mupp
#
# Utfil: tom
```

Manipulera

```
infil = open('infil.txt','r')
utfil = open('utfil.txt','w')
print('Första raden i infil.tex: ', infil.readline().rstrip('\n'))
print('Andra raden i infil.tex: ',  infil.readline().rstrip('\n'))
infil.close()
namn = raw_input('Vad heter du? ')
utfil.write(namn+'\n')
namn = raw_input('Favorit färg? ')
utfil.write(namn+'\n')
utfil.close()
utfil = open('utfil.txt','r')
for rad in utfil.readlines():
    print('Utfil: ', rad.rstrip('\n'))
```

Efter

```
# Infil
# Rad 1: hej hopp
# Rad 2: slask smurf mupp
#
# Utfil:
# Rad1: Mikael
# Rad2: lila
```

8. Programexempel

läsning, skriving, while-loop, for-loop, och lista. Skapa ett program som;

1. Läser in ord från en fil, flera ord på den rad
2. Skriver orden på en fil, nu ett ord per rad
3. Skriver ut totala antalet ord

```
# Infil
# Rad 1: hej hopp
# Rad 2: slask smurf mupp
allaOrden = list()
infil = open('infil.txt','r')    # inläsing

rad = infil.readline().rstrip('\n')
```

```
278 while rad != '':
279     ordenPaDennaRad = rad.split(' ')
280     for ord in ordenPaDennaRad:
281         allaOrden.append(ord)
282         print(ord)
283     rad = infil.readline().rstrip('\n')
284 print('Antal ord i filen = ', len(allaOrden))
285 infil.close()
286 
287 utfil = open('utfil.txt','w')
288 for ord in allaOrden:
289     utfil.write(ord+'\n')
290 utfil.close()
291 
292 # Infil
293 # Rad 1: hej
294 # Rad 2: hopp
295 # Rad 3: slask
296 # Rad 4: smurf
297 # Rad 5: mupp
```

Extra uppgift (om vi hinner)

- Skriv ett program som läser in från en fil med glospar och lägger dem i ett uppslasverk där svenska ordet blir nyckel och det engelska blir värdet.
- Skriv sen ut en lista med ordparen svenka : engelska

  Filen: äpple/apple (nyrad) hej/hello (nyrad) bil/car (nyrad) etc.

9. Klasser

Begrepp relaterade till klasser

- **Instantiering**, skapa ett objet från en klass. Får en variabel som håller reda på objektet. (d1=Dice(), d1=d2 skapar inte ett nytt object)
- **Instansvariabler** och **instansmetoder**
- **Konstruktor**
- **self**, reference till det aktuella objecktet i instansmetoder
- **Inkapsling**, bergränsa åtkomst av instansvariabler för användaren

Varför skriva klasser?

- För att vi är lata, vi vill kunna återanvända kod som vi skrivigt.
- För att gruppera funktioner och variabler på ett ställe där de kan interagera.

Från en klass kan man skapa objekt med vissa egenskaper (instansvariabler) och funktionalitet (instansmetoder). Objecten går sedan att ändras genom sina metoder. Med en klass slipper vi hålla reda på både en massa variabler i huvudprogrammet och hur de ska användas i relation till funktioner. Klassen löser det åt os.

Exempel

Vi vill hantera komplexa tal. Skapa en funktion som kan lägga ihop två komplexa tal.

```
345 
346 def add_complex(r1,i1,r2,i2):
347     return r1+r2, i1+i2
348 
```

```
# Tal ett
r1=1
i1=1

# Tal 2
r2=2
i2=-1

r3,i3=add_complex(r1,i1,r2,i2)
print(r3,i3)
# Utskrift: (3, 0)
```

Nu lös det istället med en klass

```
class Complex:
    def __init__(self, realpart, imagpart):
         self.r = realpart
         self.i = imagpart

    def __add__(self, other):
        return Complex(self.r+other.r, self.i+ other.i)

    def __str__(self):
        return str(self.r)+' i'+str(self.i)

    def scale(self, factor):
        self.r=self.r*factor
        self.i=self.i*factor

c1=Complex(1,1)
c2=Complex(2,-1)
c3=c1+c2
print (c3)
# Utskrift: 3 i0

c3.scale(3)
print(c3)
# Utskrift 9 i0
```

Andra special metoder som kan användas * __sub__ for subtraction * __mul__ for multiplication (*) * __div__ for division