Flik 4: In- och utmatning

Filer - öppna och stänga

- Öppna en fil med *open()*

```
>>> fp = open('/etc/passwd')       # öppna för läsning
>>> fp = open('test', 'w')         # för skrivning
>>> fp = open('data', 'r+')        # läsning och skrivning
>>> fp = open(r'C:\Windows\nytt\resultat.bmp', 'rb') # binärfil
```

  - Misslyckas open genereras ett undantag: IOError
- Stänga en fil med *close()*:

```
>>> fp.close()
```

- Öppna en fil med *with/open*:

```
with open('myfile.txt') as fp:
    for line in fp:
        print(line)
```

- Från 3.1.2, öppna med flera *open* i en *with*:

```
with open('A.txt') as infile, open('B', 'w') as outfile:
   for line in infile:
        if '<critical>' in line:
           outfile.write(line)
```

Exempel

- Vem vann?

```
maxp = -1
with open("score.txt") as infil:
    for rad in infil:
        namn, poäng = rad.split()
        if int(poäng) > maxp:
            maxp = int(poäng)
            vinnare = namn
print(vinnare, "vann med", maxp, "poäng")

$ python vinnare.py score.txt
Linus vann med 33 poäng
$
```

Score.txt:

Bill 12
Steve 17
Linus 33
Ken 27

Filer - exempel

- Konvertera till stor bokstav

```
infil=open("fil.txt", "r", encoding="utf-8")
utfil=open("utfil.txt", "w")
for rad in infil:
    s=rad.rstrip().upper()
    print(s, file=utfil)

$ cat fil.txt
Ett två
och tre.
$ python3 konv.py
$ cat utfil.txt
ETT TVÅ
OCH TRE.
```

Filer - läsning

- *read()*

```
>>> fp.read()      # Läser in hela filen
'This is the entire file.\n'
>>> fp.read(size) # Läser in size bytes
```

- *readline()*

```
>>> fp.readline()      # Läser en rad
'This is the first line of the file.\n'
>>> fp.readline()      # Läser in nästa rad
'This is the second line of the file.\n'
```

- *readlines()*

```
>>> fp.readlines()      # Läser in hela filen
['This is the first line of the file.\n', 'Second line of the file\
  n']
```

- loopa över filobjektet - mer minneseffektivt

```
>>> for line in fp:
...     print(line, end='')
This is the first line of the file.
Second line of the file
```

Filer - skrivning, sökning

- *write()*

```
>>> fp.write('This is a test\n')
```

- *writelines()*

```
>>> s = ['First line\n', 'Second line\n']
>>> fp.writelines(s)      # Skriver listan till fil
```

- *seek()* flyttar filpekaren, *tell()* talar om var filpekaren står
- Se vidare *kapitel 7 - File Handling* för övriga filobjekt-metoder

Exempel

- Konvertera till stor bokstav i andra raden

```
import sys

with open(sys.argv[1], "r+") as fp:
    fp.readline()
    pos=fp.tell()
    rad2=fp.readline()
    fp.seek(pos)
    fp.write(rad2.upper())
```

```
$ python andraraden.py fil.txt
$ cat fil.txt
Ett två
OCH TRE.
$
```

Filer – fel teckenkodning

- Om filen är i en annan teckenkodning än den man anger:

```
infil=open("fil.txt", "r", encoding="latin-1")
utfil=open("utfil.txt", "w")
for rad in infil:
    s=rad.rstrip().upper()
    print(s, file=utfil)

$ cat fil.txt
Ett två
och tre.
$ python3 konv.py
$ cat utfil.txt
ETT TVÃ¥
OCH TRE.
```

Unicode

- Från version 3.0 är strängar Unicode!
- Representation Unicode vs UTF-8
  - å, ä, ö motsvaras t.ex. i unicode av \u00e5, \u00e4, \u00f6 (sk. code point)

```
>>> 'åäö'.encode('utf-8')
b'\xc3\xa5\xc3\xa4\xc3\xb6'
>>> x = b'\xc3\xa5\xc3\xa4\xc3\xb6'
>>> x.decode('utf-8')
'åäö'
>>> 'åäö'.encode("latin1")
b'\xe5\xe4\xf6'
>>> print("G\u00f6ran")
Göran
```

...

- Tips för att använda Unicode
  - Använd endast *encode* och *decode* vid läsning/skrivning från/till fil, db etc
  - Källkod antas vara kodad i UTF-8 om inget annat sägs (=använd editor som kan förstå UTF-8)
  - Annan kodning, t.ex. latin-1: `# -*- coding: latin-1 -*-` på rad 2
- Se vidare *kapitel 2 - Fundamental Data Types: Character Encodings*

bytes (och bytearray)

- Både *bytes* och *bytearray* är mycket lika strängar. Hanterar 8-bit unsigned integers
- Typen *bytes* är immutable och består av tecken med värdena 0-255
  - Kan skrivas som `b'123'`
- Typen *bytearray* är mutable
- Exempel:

```
>>> b = b'\x00123\xff'
>>> b[4]
255
>>> list(b)
[0, 49, 50, 51, 255]
>>> ba = bytearray(b'\x00123\xff')
>>> ba[4] = 200
>>> ba
bytearray(b'\x00123\xc8')

>>> bytes('\u00e5','latin-1')
b'\xe5'
>>> bytes('\u00e5','utf-8') # -> sträng.encode('utf-8')
b'\xc3\xa5'
```

Läsa och skriva viss teckenkodning

- Exempel 1 – det krångliga sättet

```
#!/usr/bin/env python3
hello_out = 'Hallå\n'
bytes_out = hello_out.encode('utf-8')
f = open('unicode-utf8.txt', 'wb')
f.write(bytes_out)
f.close()
f = open('unicode-utf8.txt', 'rb')
bytes_in = f.read()
f.close()
hello_in = bytes_in.decode('utf-8')
```

- Exempel 2 – det lätta sättet (låt python göra jobbet)

```
#!/usr/bin/env python3
hello_out = 'Hallå\n'
f = open('unicode-utf8.txt', 'w', encoding='utf-8')
f.write(hello_out)
f.close()
...
```

Socket

- Kommunikation över socket sker med bytes, inte strängar:

```
import socket

cmd = ['ticket fetch\r\n', 'PING ester\r\n',
       'PING åsa\r\n']
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect( ("lab04.bredbandskollen.se", 80) )
for msg in cmd:
    s.sendall(msg.encode())
    reply = b""
    while reply[-2:] != b"\r\n":
        r = s.recv(2000)
        if not r:
            raise SystemExit  # connection closed
        reply += r
    print("Reply:", reply.decode())
s.close()
```

Övning

- **ovn0401/summera.py** Filen `tal.txt` innehåller ett antal heltal, ett per rad. Skriv ett program som beräknar och skriver ut summan av alla talen i filen.
- **ovn0402/palindrom.py** Filen ord.txt innehåller ett antal ord, ett per rad. Skriv ett program som letar reda på och skriver ut alla palindrom i filen, dvs ord som är samma framlänges och baklänges (t.ex. radar, ror, Anna). Ta inte hänsyn till stora/små bokstäver.
  Tips: om `s` är strängen `'Göran'`, så är `s[::-1]` strängen `'naröG'`.
- **ovn0403/tounicode.py*** Skriv ett program som läser in filen `fil2.txt` (som är kodad i latin-1) och skriver ut samma text till filen `fil3.txt`, fast kodad i utf-8. (Se till att det görs close på den skrivna filen.)
- **ovn0404/webbsida.py**** Öppna en connection till lab04.bredbandskollen.se port 80. Skicka kommandot

  ```
  GET /bigfile.bin?len=400 HTTP/1.0\r\n\r\n
  ```

  Läs svaret. (Hur vet man när man läst hela svaret?) Programmet ska skriva ut antalet byte ”payload” i svaret (svaret består av en ”header” följt av en blankrad följt av det egentliga innehållet, alltså ”payload”).
  Ledning: `header, payload = reply.split('\r\n\r\n')`

Filer - kommandoradsparametrar

- *sys.argv*

```
#! /usr/bin/env python3
#
import sys
print('you entered', len(sys.argv) - 1,
      'arguments...')
print('they were: ', sys.argv[1:])

$ chmod a+x args.py
$ ./args.py arg1 arg2 arg3
you entered 3 arguments...
they were:  ['arg1', 'arg2', 'arg3']
```

Filer - stdin, stdout, stderr

- Läs/skriv från filerna *sys.stdin, sys.stdout, sys.stderr*
  - Behöver inte öppnas eller stängas

```
#! /usr/bin/env python3
#
import sys
linecnt = 0
for line in sys.stdin:
    linecnt += 1

print(linecnt)

$ chmod a+x wc.py
$ cat wc.py | ./wc.py
8
$ wc wc.py
       8      17     105 wc.py
```

Filer - stdin, stdout, stderr

- Skriva till stderr, med *print()*

```
import sys
print('Felmeddelande', file=sys.stderr)
```

- Skriva till stderr med *write()*

```
import sys
sys.stderr.write('Felmeddelande\n')
```

- Omdirigering

```
import sys
out = open('out.txt','w')
sys.stdout = out
print('This goes to out.txt')
sys.stdout = sys.__stdout__   # Återställ stdout
out.close()
```

Filer - filsystem

- Modulerna *os, os.path*

```
os.getcwd()                 # nuvarande katalog
os.chdir(path)              # byt katalog
os.listdir(path)            # lista filer i katalog
os.mkdir(path)              # skapa katalog
os.mkdirs(path)             # skapa flera nivåer av katalog
os.rmdir(path)              # ta bort katalog
os.removedirs(path)         # ta bort flera nivåer av katalog
os.rename(old, new)         # byt namn
os.renames(old, new)        # flytta till helt ny sökväg
os.remove(path)             # ta bort fil
os.path.join(path, file)    # sökväg till file i path
os.path.getsize(filename)   # filstorlek
os.stat(file)               # filstorlek, accesstid m.m.
```

Filer - filsystem

- Exempel

```
>>> import os
>>> os.path.isdir(r"C:\TEMP")
False
>>> os.path.isdir("/tmp")
True
>>> os.getcwd()
'/home/goran/GIT/pythonkurs/ovn0401'
>>> kat = os.getcwd()
>>> os.listdir(kat)
['summera.py', 'tal.txt', 'verify.py']
>>> os.path.join(kat, "verify.py")
'/home/goran/GIT/pythonkurs/ovn0401/verify.py'
>>>
```

Några konstanter

- Modulen *os* (*import os*) - Cross-platform

```
os.name    # posix, nt, dos m.fl
os.linesep # \n för Unix, \r\n för Windows, \r för Mac
os.curdir  # . för Unix+Windows, : för Mac
os.pardir  # .. för Unix+Windows, :: för Mac
os.sep     # / för Unix, \ för Windows, : för Mac
os.pathsep # : för Unix, ; för Windows
```

Filer - filsystem

- Modulen *shutil* erbjuder filoperationer på högre nivå, t.ex.:
  - shutil.copy()
  - shutil.copytree()
  - shutil.rmtree()
  - shutil.move()

- Se vidare *kapitel 5 - Modules: File, Directory and Process Handling*

Persistens/Datalagring - pickle

- Python har stöd för att konvertera ett objekt till en dataström för lagring i t.ex. fil, sträng, databas eller överförd över nätverk samt givetvis återskapande av samma objekt.
- Modulen *pickle* stödjer t.ex. lagring av tal, strängar, listor, tupler, dictionaries, klassinstanser mm. Lagring av enstaka objekt:

```
# Spara undan
import pickle
favorite_color = { "lion": "yellow", "kitty": "red" }
with open('data.3pickle', ’wb') as f:
   pickle.dump( favorite_color, f)

# Ladda tillbaka
import pickle
with open('data.3pickle', 'rb') as f:
    favorite_color = pickle.load(f)
print(favorite_color)
```

Persistens/Datalagring - shelve

- Modulen shelve läser och lagrar objekt i en ”databas” i form av ett dictionary
- Nycklarna kan endast vara strängar. Använder pickle för att lagra värden

```
import shelve

db = shelve.open('shelf.obj', 'c')
db['primtal'] = [2, 3, 5, 7]
db['udda'] = [1, 3, 5, 7, 9]
db['jämna'] = [2, 4, 6, 8, 10]

db.close()
```

Relationsdatabaser

- Filbaserad databas: SQLite

```
import sqlite3

conn = sqlite3.connect('competition.db')
c = conn.cursor()
c.execute('SELECT id, name FROM dudes')
for row in c:
    print(row)

c.execute('SELECT name FROM dudes WHERE id=3')
print(c.fetchone())
```

- Utskrift t.ex.

```
(1, 'Bill')
(2, 'Steve')
(3, 'Linus')
(4, 'Ken')
('Linus',)
```

Skapa tabell, infoga data

```
import sqlite3

conn = sqlite3.connect('competition.db')
c = conn.cursor()

c.execute('''
    CREATE TABLE dudes (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT)
''')

sql = 'INSERT INTO dudes (name) VALUES (?)'
for dude in "Bill Steve Linus Ken".split():
    c.execute(sql, (dude,))

c.execute('SELECT last_insert_rowid()')
print(c.fetchone())     # (4, )   Dvs. Ken fick id 4

conn.commit()
```

Relationsdatabaser: Foreign key

```
import sqlite3

 conn = sqlite3.connect('competition.db')
 c = conn.cursor()

 c.execute('''
     CREATE TABLE scores (
         dude INTEGER,
         score INT,
         FOREIGN KEY(dude) REFERENCES dudes(id))
 ''')

 sql = 'INSERT INTO scores (dude, score) VALUES (?, ?)'
 c.execute(sql, (1, 12))
 c.execute(sql, (2, 19))
 c.execute(sql, (3, 33))
 c.execute(sql, (4, 27))

 conn.commit()
```

Relationsdatabaser: sökningar

```
import sqlite3

conn = sqlite3.connect('competition.db')
c = conn.cursor()

c.execute('''SELECT score, name FROM scores, dudes
             WHERE dude=id''')
print(c.fetchall())

sql = '''SELECT s.score, d.name FROM scores s
         LEFT JOIN dudes d on s.dude=d.id
         WHERE score>20'''
c.execute(sql)
print("\nDe bästa:", c.fetchall())
```

- Utskrift:

```
[(12, 'Bill'), (19, 'Steve'), (33, 'Linus'), (27, 'Ken')]

De bästa: [(33, 'Linus'), (27, 'Ken')]
```

Serialisering med JSON

- Spara och läsa data i JSON-format

```
import json
lista = [ 2, 3, 5, 7, 11, 13, 17, 19, 23]

with open("lista.json", "w") as utfil:
    print(json.dumps(lista), file=utfil)

# ...

with open("lista.json") as infil:
    nylista = json.loads(infil.read())

print(nylista[5])
# 13
```

Övning

- **ovn0405/sparajson.py**
  - Skapa funktionen `sparajson`. Den ska ta två parametrar: en lista (eller annan datastruktur) och ett filnamn. Funktionen ska konvertera listan till JSON-format och spara i filen.
  - Skapa funktionen `laddajson`. Den ska ta ett filnamn som parameter. Filens innehåll ska läsas in som JSON och returneras som ett Python-värde.
- **ovn0406/sparapickle.py*** Som ovan, men använd pickle i stället för JSON. Funktionerna ska heta `sparapickle` och `laddapickle`.

Övning

- **ovn0407/databas.py****
  - Skapa funktionen `create` som tar en sträng (ett filnamn) som parameter. Om filen finns, så ska den raderas. Sedan ska en SQLite-databas skapas med tabellerna `dudes` och `scores` (enligt tidigare exempel) i den filen.
  - Skapa funktionen `add_dude` som tar tre parametrar:
    - filnamn till SQLite-databasen
    - en sträng som ger ett namn på en person (t.ex. ”Steve”)
    - ett heltal som ger en poäng (t.ex. 19)

    Funktionen ska lägga in namnet i tabellen `dudes` och poängen (med referens till namnet i `dudes`) i tabellen `scores`.
  - Skapa funktionen `get_score` som tar två strängar som parametrar. Den första parametern ska vara ett filnamnet till SQLite-databasen och den andra ska vara ett namn, t.ex. ”Steve”. Funktionen ska returnera poängen som personen med det namnet fick. Om personen inte fanns i databasen ska `None` returneras.