Tutorial: Python på 90 minuter

Human Centered Systems
Inst. för datavetenskap
Linköpings universitet

Läsöversikt

- LP Part I – Part IV, kap 18
- PL: kap 1, 2 (Sem 1), 3 (Sem 3), 5.1-5.3, 5.8 intro, 6.1-6.3, 7.1-7.3, 8.1-8.3, 9.1-9.2
- Wikipedia kan användas som stöd för PL-avsnitten.

Föreläsningar 2-6

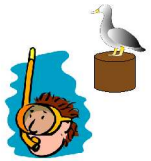
- **Föreläsning 2**: översikt över Python
- **Föreläsning 3**: algoritmer och imperativt tänkande
- **Föreläsning 4**: Datastrukturer, procedurell och dataabstraktion
- **Föreläsning 5**: Principer för programmeringsspråk
- **Föreläsning 6**: Inför tentamen

Genomförande av laborationer

- Arbetar parvis
  - Välj själva labbpartner
- Läs igenom laborationer i förväg
  - **börja läsa laboration 1 nu...**
- Installationslaborationen måste vi fixa före resten...
- Schema
  - Må-ti är intensivdagar med eget arbete i PC6-7
  - Handledningsmöte 4 ggr per vecka – gemensamt för TDP002 och TDP003 (må, 2 ggr ti, fre)

Översikt (LP kap 3, 4, 10, 15)

- Python genom en serie exempel ("snippets")
- Tutorial – snabbt igång med ny teknik/nytt språk
- Kopiera källkod – ett bra första steg
- Kom igång snabbt med labbar och projekt



OBS: man behöver inte fatta alla detaljer på en gång!

Konceptuell struktur (jmf. PL-boken)

LP Part II: Types and Operations

LP Part III: Statements and Syntax

LP Part IV: Functions

PL: kap 6 Data types

PL: kap 7 Expressions and Assignment Statements
PL: kap 8 Statement-Level Control Structures

PL: kap 9 Subprograms

Ex 0: Moduler: import/reload (Kap 3)

```
>>> import os
>>> help("os")
...man-sida
>>> reload(os)
```

- Python Standard Library
  - Utvidgar Python-språket med "bra-att-ha" kommandon/funktioner
  - http://docs.python.org/
  - Källkod: /usr/lib/python
- import laddar en modul
- help interaktiv hjälp (jmf man)
- reload laddar om en modul (bra vid utveckling)

Typer och operationer

Tal

```
>>> 20 + 22
42
>>> 1 + 2 + 3 + 5 + 7 + 11 + 13
42
>>> 3 * 4
12
>>> 3 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or
modulo by zero
>>> 44 - 13.3
30.699999999999999
>>> max(3,7)
7
>>> round(7.5)
8.0
>>> import math
>>> math.pi
3.1415926535897931
```

- Inbyggda (built-in)
- icke-muterbara (immutable)
- Matematiska beräkningar

Strängar

```
>>> str = 'hej'
>>> len(str)
3
>>> str[1]
'e'
>>> str[1:3]
'ej'
>>> str[-2]
'e'
>>> str + str
'hejhej'
>>> str * 5
'hejhejhejhejhej'
>>> str.split('e')
['h', 'j']
```

- Inbyggda (built-in)
- Sekvenstyp
- icke-muterbara (immutable)
- *metoder* med strängspecifika operationer

Listor

```
>>> mylist = ['a', 'b', 'c']
>>> mylist[1]
'b'
>>> mylist[1:3]
['b', 'c']
>>> mylist[0] = 'A'
>>> del mylist[1]
>>> mylist
['A', 'c']
>>> mylist.append('x')
>>> mylist
['A', 'c', 'x']
>>> mylist2 = [7,3,6]
>>> mylist2.sort()
>>> mylist2
[3, 6, 7]
```

- listor är sekvenser av objekt - "dynamiska arrayer"
- muterbara – både storlek och värden
- Elementen kan vara vilket typ av objekt som helst
- Listor kan blanda typer av objekt
- Metoder ger specifika operationer för listor

Dictionary

```
words = {'gul': 'yellow', 'vit':
'white' }
>>> words['gul']
'yellow'
>>> 'gul' in words
True
>>> words['gul'] = 'Yellow'
>>> words.values()
['white', 'Yellow']
```

- Hashtabell/map
  - Lagrar nyckel-värde associationer
- nycklarna är ickemuterbara objekt
- Metoder för typspecika operationer

Tupler

```
>>> ('a', 'b', 'c')
('a', 'b', 'c')
>>> tuple = ('a', ('b', 'c'))
>>> tuple[1]
('b', 'c')
>>> ('a', 'b', 'c') + ('d', 'e')
('a', 'b', 'c', 'd', 'e')
```

- Tupler är sekvenser av objekt
- de är icke-muterbara
  - Längden är fixerad
  - Elementvärdena är fixerade

Filer

```
>>> myfile = open("test.txt", "w")
>>> myfile.write("Rad 1 - test \n")
>>> myfile.write("Rad 2 - test igen \n")
>>> myfile.close()
```

Filen writetest.py:

```
#!/usr/bin/env python

myfile = open("test.txt", "w")
myfile.write("Rad 1 - test \n")
myfile.write("Rad 2 - test igen \n")
 myfile.close()
```

```
larde@~:more test.txt
Rad 1 - test
Rad 2 - test igen
```

- Datafiler är objekt i språket
- Metoder ger operationer
  - Plus open

Och kom ihåg att...

Program = fil = "modul"

Kommando = sats

Moduler har sideffekter

Satser och syntax

Pythons programstruktur

- 1. Program består av moduler
- 2. Moduler består av satser
- 3. **Satser** innehåller uttryck ← **Här**
- 4. Uttryck skapar och beräknar objekt

Tilldelningssatsen

```
sum = 1.0 + 2.0 + 3.0
prod = 4.0 * 5.0
div = sum * sum / prod
print div
sum = 'Aha en sträng'
print sum
print div
```

```
1.8
Aha en sträng
1.8
```

- Uttryck är det som "räknar ut något" i språket
  - Jmf. matematiska uttryck
  - Även struktur-uttryck t ex strängar. listor
- Variabler för att mellanspara värden
- = "tilldela uttryck till variabel"

Villkorssatsen

```
import random

comp_die = random.randint(1,6)
user_die = random.randint(1,6)

if comp_die > user_die:
    print "Datorn vann!"
elif comp_die < user_die:
    print "Du vann!"
else:
    print "Oavgjort!"
```

- if-satsen "om sant gör så"
- elif "annars om gör så"
- else "annars gör så"
- elif, else är optionella
- man kan ha flera elif
- **OBS** kolon och indentering nödvändig

for-loopen

```
number_list = range(7)
print number_list
print 'For-loop:'
for x in number_list:
    print x,
print "\n"
```

```
[0, 1, 2, 3, 4, 5, 6]
For-loop:
0 1 2 3 4 5 6
```

- **For** (each): för varje element gör något
- For-loopen är bra för förutbestämda loopar
  - Eng "Definite iterations"
- Loop = iteration
- **OBS** kolon och indentering nödvändig

while-loopen

```
print 'While-loop:'
i = 0
while number_list[i] ** 2 < 15:
    print number_list[i],
    i += 1
print '\n'
```

```
While-loop:
0 1 2 3
```

- **while** (condition): så länge villkoret är sant gör något
- Bra för obestämda loopar
  - Eng indefinite iterations
- **OBS** kolon och indentering nödvändig

Funktioner

def-satsen

```
def print_hello():
   print 'Hej',

def hello_twice():
   print_hello()
   print_hello()
```

- Funktioner är subprogram
  - "defineras" och "anropas"
- Paketering av källkod under visst namn
- Indentering avgör när defintionen slutar

OBS: inget händer när detta körs förutom att print_hello blir *definerad* som en funktion...

Anrop av funktioner

```
def print_hello():
   print 'Hej',

def hello_twice():
   print_hello()
   print_hello()


   print_hello()
   print
   hello_twice()
```

Hej
Hej Hej

- Anrop kan ske från modulnivå
- Funktioner kan anropa andra funktioner

Parametrar, returvärden och lokala variabler

```
def add_values(val1, val2):
   sum = val1 + val2
   return sum

mysum = add_values(7,9)
print mysum
```

16

- Funktioner kan ha indata i form av **parametrar**
- De värden som binds till parametrarna i ett anrop kallas **argument**
- Parametrar och variabler i en funktion är **lokala** för funktionen
  - Kan bara refereras inom funktionen

Räckvidd och skuggning

```
size = 45

def print_my_size(size):
    return size

def print_global_size():
    global size
    return size

print print_my_size(22)
print print_global_size()
```

```
22
45
```

- Variabler/parametrar har räckvidd (scope)
- Parametrar och lokala variabler "skuggar" globala variabler

Moduler

Egna moduler

file_handler.py:

```
_db_file = open('my-database.txt')
_db_list = _db_file.readlines()
_db_file.close()
def no_items():
    return len(_db_list)
def get_item(index):
    return _db_list[index]
```

present_results.py:

```
import file_handler
def main():
    i = 0
    while i < file_handler.no_items():
        print file_handler.get_item(i)
        i = i + 1
main()
```

- Delar upp större program i flera filer
- Konceptuell klarhet
- Återanvändning

Några större exempel (i mån av tid...)

Ex: Manipulera filer

```
import os

t = os.path.abspath('.')
print 'abspath:', t
print 'basename:', os.path.basename(t)
cwd = os.getcwd()
print 'cwd:' + cwd
print 'listdir:', os.listdir(cwd)
print 'is dir:', os.path.isdir('filedir.py')
```

```
abspath: /home/larde/svndoc/courses/TDP002/py-2
basename: py-2
cwd:/home/larde/svndoc/courses/TDP002/py-2
listdir: ('.svn', 'storedata_2.py', 'sumprod.py', 'config.py', ...)
stat: (33188, 5478488L, 2053L, 1...)
is dir: False
```

- os-modulen - interaktion med fil/kataloger/skal
- För automatisk hantering/ bearbetning av filer

Ex: Kommandorad och os.system

pdf_html.py: ett *skript* för att generera pdf och html-filer

```
import os
import sys

ps2pdf_call = 'ps2pdf %(name)s-4.ps %(name)s-4.pdf' % {"name" : sys.argv[1]}
os.system(ps2pdf_call)
tar_call = 'tar zcvf %(name)s-html.tgz %(name)s-html' % {"name" : sys.argv[1]}
os.system(tar_call)
scp_call = 'scp %s* remote.ida.liu.se:%s' % (sys.argv[1], sys.argv[2])
os.system(scp_call)
```

argument 1

argument 2

Anrop: pdf_html.py lecture-1 ~TDP002/www-pub/lectures

(Lite) större exempel: store_data

Initial pseudokod:

WHILE användaren vill:
läs in ny data
spara data på fil

Frågor:
1) vilken fil?
2) hur avsluta?

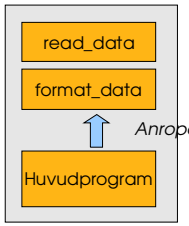
Förfining:

Öppna filen "data.txt"
WHILE användaren vill:
läs in ny data
IF NOT \q:
spara data på fil
Stäng filen

```
%: ./storedata_1.py
Enter data ('\q' to quit): Kom ihåg: X
Enter data ('\q' to quit): Händelse Y
Enter data ('\q' to quit): Ring Z
Enter data ('\q' to quit): \q
Added 3 lines to file data.txt
```


Programskiss

storedata_1.py:



Funktion som läser in indata

Funktion som väljer filformat

Anropar

storedata_1: huvudprogram

```
if __name__ == "__main__":
    data_file = open("data.txt", "a")
    more_data = True
    no_of_lines = 0
    while more_data:
        data = read_data()
        if data == "\Q":
            more_data = False
        else:
            data_file.write(format_data(data))
            no_of_lines = no_of_lines + 1
    print "Added %i of lines to file %s" % (
        (no_of_lines, 'data.txt'))
    data_file.close()
```

Testar om modul är toppmodul

Egna funktioner

storedata_2: tidsmärkning och filhantering

Förändringar (refactorings):
1) Jag vill spara i antingen egen fil eller default. (Framtid: konfigurerbart)
2) Jag vill tidsstämpla varje data

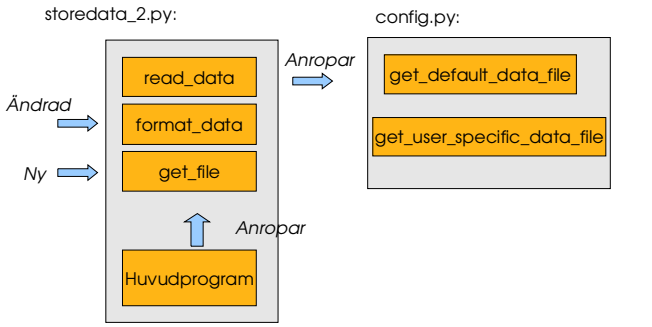
Ny pseudokod:

Välj fil (sep konfig-modul)
Öppna vald fil
WHILE användaren vill:
    läs in ny data
    IF NOT \q:
        formattera data
        spara data på fil
Stäng filen

Filen **~/.myapp/data.txt**:

```
Tue 08/28/07 (05:19:59 PM)::KOM IHÅG: X
Tue 08/28/07 (05:20:03 PM)::HÄNDELSE Y
Tue 08/28/07 (05:20:06 PM)::RING Z
```


Programskiss – två moduler



storedata_2: huvudprogram

```
if __name__ == "__main__":
    file_name = get_file()
    data_file = open(file_name, "a")
    more_data = True
    no_of_lines = 0
    while more_data:
        data = read_data()
        if data == "\Q":
            more_data = False
        else:
            data_file.write(format(data))
            no_of_lines = no_of_lines + 1
    print "Added %i lines to file %s" % (
        (no_of_lines, file_name))
    data_file.close()
```

Nya funktioner: minimalt ingrepp i gammal källkod

getFile anropar config

format anropar time

Separat config-modul: config.py

```
import os
import os.path

_metadata={"application_home": "/usr/local/myapp",
           "default_user_directory": "~/.myapp",
           "data_file": "data.txt"}

def get_default_data_file():
    return os.path.join(_metadata("application_home"),
                        _metadata("data_file"))

def get_user_specific_data_file():
    return os.path.join(
        os.path.expanduser(_metadata("default_user_directory")),
        _metadata("data_file"))
```


Summering

- Översikt över Learning Python Part II-IV, kap 18
  - Vi återkommer till detaljer i laborationer och teori på föreläsningar
- Glöm inte läsa Kap 2-3 och wikipedia från föreläsning 1!
- Gör egna övningar!