Denne guide er oprindeligt
udgivet på Eksperten.dk

COMPUTERWORLD

I gang med Python

I denne artikel vil jeg forsøge at give et kort og hurtigt indblik i programmeringssproget Python, der desværre er alt for overset.

Skrevet den **03. Feb 2009** af **jensgram** I kategorien **Programmering / Andre** | ★★★☆☆

Hvad er Python? Og hvorfor bruge det?
Python er et programmeringssprog på niveau med C, C++, Java osv. Oftest sammenlignes det med Perl, Tcl og Java, men har dog væsentlige fordele på flere punkter:

1) Kvaliteten på produktet kan blive meget høj (dette er selvfølgelig afhængigt af programmøren), da Python pr. definition er let at læse og forstå. Python har helt særlige regler for blokke og indrykinger, som jeg vil komme ind på om et øjeblik.

2) Det er let at være produktiv med Python, da Python som regel kun har mellem 67 og 80% færre linier end tilsvarende C eller Java kode.

3) Langt de fleste Python programmer vil køre uændret på Windows, *nix og Mac.

4) Python dsitribueres med et enormt standardbibliotek indeholdende funktioner til streng-operationer, regulære udtryk osv. og kan udvides med tredjepartsmoduler som eksempelvis PIL (Python Imaging Library).

5) Python kan *meget* - GUI'er, interaktive websites, billedmanipulation og meget, meget mere. Og så er indlæringskurven behagelig! Og så er det gratis.

Som det fremgår af ovenstående er der altså ingen umiddelbar undskyldning for ikke at se nærmere på sproget. Eneste ulempe ved sproget er, at det ikke er lige så hurtigt som kompileret C eller C++, men i de fleste tilfælde vil Python være hurtigt nok. Nu tænker du måske "hvis Python ikke er et kompileret sprog, hvad er det så?" - Python er et fortolket sprog, der oversættes til byte code og køres af PVM på CPU'en:
Source (.py) -> Byte code (.pyc) -> Runtime (PVM)
Byte code'n er low-level kompileret Python kode, der er komplet planformsuafhængigt.
PVM står for Python Virtual Machine, der eksekverer byte code og er skrevet specifikt til platformen.

Installation og brug af Python under Windows
Sidste nye version af Python (p.t. 2.3.4) hentes fra http://www.python.org. Installationsfilen fylder ca. 10MB og er en almindelig WISE installation, der guider dig gennem forløbet. Python installeres som standard i c:\Python23.

Alternativt kan man benytte Jython, som er Python (oftest omtalt som CPython, når Jython også nævnes) skrevet i Java. Dette er dog noget langsommere, da Java selv kører på en virtuel maskine. Jython nævnes ikke mere i denne artikel, da jeg kun har beskæftiget mig med CPython.

Når Python er installeret på din computer kan du komme hurtigt i gang i Pythons shell IDLE, der tillader interaktiv kodning med syntaksfarvning. Dette er et udmærket miljø til at starte med Python.

Alle eksempler i denne artikel tager udgangspunkt i IDLE, hvor ">>>" indikerer vores statement linie.

Typer i Python
Operationer på typer behandles senere, men først en oversigt:
1) Tal:
- Integer: Positive og negative heltal (ex: 1, 0, -54)
- Float: Decimal tal (ex: 1.23, 3.14, -12.0)
- Long integer: Lange heltal, større end 32 bit (ex: 4294967296L)
- Octal: Oktaler (0-7) (ex: 0177 = 7 * 8^0 + 7 * 8^1 + 1 * 8^2 = 127 (decimal))
- Hex: Hexadecimal (0-F) (ex: 0x9F = 15 * 16^0 + 9 * 16^1 = 159 (decimal))
- Complex: Komplekse tal (ex: 1+1j, 3.14+1.2J, 9J)

2) Strenge: Indkapsles i qoutes (', ", ''' eller """)

3) Lister: Autoindekseret liste med alle typer indhold (ex: liste1 = [], liste2 = [24, 'test', 0xFF])

4) Dictionaries: Nøgle - værdi par (ex: dic1 = {}, dic2 = {'noegle': 'vaerdi', 'alder': 23, 23 = [1, 'er']})

5) Tupler: Som lister, men "immutable" - kan altså ikke ændres (ex: tuple1 = (), tuple2 = ('hej', 23, [1, 23]))

Operatorer
1) Logisk sammenligning (True / False): x < y, x <= y, x == y, x >= y, x > y, x is y, x is not y, x in y, x not in y
2) Addition og subtraktion: x + y, x - y, x + (-y) osv.
3) Multiplikation, division, potens og modulo: x * y, x / y, x ** y, x % y
4) Logisk OR: x or y
5) Logisk AND: x and y
6) Negation: not x

Kommentarer
Kommentarer i Python kode startes med et hash (#) og fortsætter linien ud (se eksempel 1).

Variabler og assignments
Python skelner mellem store og små bogstaver overalt. Variabler kan bestå af tegnene a-z, A-Z, 0-9 og _ (underscore) og skal være minimum ét tegn lange. Desuden må de ikke starte med 0-9.

Det er desuden værd at bemærke, at Python arbejder med referencer (modsat værdier), hvis ikke andet angives.

Variabler kan tildeles som følger:

Eksempel 1:

```
>>> min_var = 'noget tekst' # Simpel form
>>> var1 = var2 = var3 = 'indhold' # Multiple assignment (reference til
_samme_ streng
>>> [pi, e] = [3.14, 2.72] # Liste assignment
>>> (pi, e) = (3.14, 2.72) # Tuple assigment
>>> pi, e = 3.14, 2.72 # Som ovenstående
>>> x += 1 # Augmented assignment (x tælles én op)
```

Augmented assignment gælder også -, /, % osv.

IDLE udskriver som standard, men i .py filer skal man benytte print. Følgende eksempel vil udskrive den samme streng to gange i IDLE, men i en .py fil vil var kun udskrives på tredje linie:

Eksempel 2:

```
var = 'noget tekst her'
var
print var
```

If, for, while og indrykninger

Indentering (indrykning) er meget vigtigt i Python. I Java (og mange andre sprog) afgrænses blokke af "tuborg"-klammer {}. Det kan man ikke i Python og er faktisk en hovedårsag til, at Python er så letlæst, da man tvinges til at have orden på sine indrykninger.

Vi springer straks til et eksempel for at illustrere - her et if-statement:

Eksempel 3:

```
if <condition>: # Generel struktur
    <statements>
[elif <conditionN>:
    <statements>]*
[else:
    <statements>]?

>>> if 2 >= 5:
    print 'Næppe'
else:
    print 'Ja, nemlig ja'
# Vil udskrive: Ja, nemlig ja
```

Som antydet med [], * og ? kan elif og else udelades - elif kan desuden optræde et vilkårligt antal gange.

Python er udstyret med to løkke-strukturer - en generel konstruktion samt en konstruktion til gennemløb af sekvenser (liste, tupler m.v.). While-løkken er den generelle, der kører så længe et givent udsagn er sandt. Derfor er en uendelig løkke let at lave. Strukturen for while er som følger:

Eksempel 4:

```
while <condition>:
    <statements>
[else
    <statements>]?

>>> i = 1
>>> while i < 10:
```

```
    i += 1
    if i == 5:
        continue
    elif i == 9:
        break
    print i, # Ingen newline
else:
    print 'Løkken kørte færdig'
# Vil udskrive: 2 3 4 6 7 8
```

Her er fyldt lidt ekstra på, men det er stadig relativt simpelt. Så længe i er mindre end 10 tælles den én op. Er i lig 5 startes løkken forfra (de næste statements udføres ikke) og er i lig 9 afbrydes løkken. Således vil else-delen ikke udføres (dette sker, hvis løkken bliver færdig (<condition> == False) *uden* brug af break.

For-løkken laver gennemløb af lister m.v. Derfor vil jeg lige præsentere den indbyggede funktion range(), der tit kan være nyttig i denne sammenhæng. Følgende skulle gerne give et indblik i dens argumenter og opførsel:

Eksempel 5:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 5)
[1, 2, 3, 4]
>>> range(10, 20, 2)
[10, 12, 14, 16, 18]
```

For-løkken ser ud som følger:

Eksempel 6:

```
for <item> in <sequence>:
    <statements>
[else:
    <statements>]?

>>> for i in range(5):
    print i,
else:
    print '- og nu er jeg færdig'
# Vil udskrive: 0 1 2 3 4 - og nu er jeg færdig
```

Ligesom i while-løkken kan der benyttes break og continue med samme effekt. Desuden kan de indbyggede metoder keys() og values() være gode at kende, hvis man kunne tænke sig at iterere over et dictionary. Benyt da mit_dictionary.keys() som <sequence>.

Indexing og slicing
Indexing og slicing kan udføres på strenge, lister og tupler. Man benytter hårde parenteser [] og kan - som navnene antyder - hente et indeks eller slice (stykke). Skrives et tal mellem parenteserne vil der hentes et indeks. Slicing laves med et kolon (:). Eksempelvis vil [0:2] hente det nulte og første indeks osv.:

Eksempel 7:

```
>>> l = [112, 911, 'ild']
>>> l[1]
911
>>> l[0:2]
[112, 911]
>>> l[:2]
[112, 911] # Som ovenstående
>>> l[-2]
911
>>> l[1:]
[911, 'ild']
>>> l[:]
[112, 911, 'ild'] # NB: Dette er en kopi af l
```

Værd at vide: Strenge
Formatering af strenge kan foregå ligesom C's sprintf funktion, hvor % benyttes til at indsætte variabler. De mest normale er %s (streng), %d (decimal) og %i (heltal). Efter strengen angives variablen / variablerne i en liste eller tuple adskilt fra strengen med et procenttegn (%).

Desuden er der en hel metoder, der knytter sig til strenge. En par eksempler er replace(), upper() og split().

Funktionen len() - der også kan bruges på lister, tupler og dictionaries - returnerer antallet af tegn i en streng.

Repetition af strenge kan foregå med * operatoren. Konkatinering klares med +. Desuden kan operatoren in bruges til at undersøge, hvorvidt en streng findes i en anden streng.

Eksempel 8:

```
>>> s = '%s is %i years old' % ('Jens', 20)
>>> print s
Jens is 20 years old
>>> s2 = s.replace('Jens is', 'I\'m')
>>> s2
"I'm 20 years old"
>>> s.split(' ')
['Jens', 'is', '20', 'years', 'old']
>>> len(s2)
16
>>> '-' * 20 + '|-'
'--------------------|-'
```

Værd at vide: Lister
Lister kan - som strenge - gentages og konkatineres med * og +.

Nyttige metoder er: append(), extend(), sort() og reverse(). Bemærk, at disse udføres direkte på listen:

Eksempel 9:

```
>>> l = range(3)
>>> l + [3, 4]
[0, 1, 2, 3, 4]
>>> l.append(5)
>>> l
[0, 1, 2, 5]
>>> l.extend([6, 7, 8])
>>> l
[0, 1, 2, 5, 6, 7, 8]
>>> l.reverse()
>>> l
[8, 7, 6, 5, 2, 1, 0]
```

Afslutning
Dette var en kort introduktion til Python. Jeg håber du fik noget ud af at læse den, og jeg kan på det kraftigste anbefale, at du kigger nærmere på Python. Man kan rigtig, rigtig meget og det er meget let at komme i gang med. Desuden bliver man glad for, at indrykning afgrænser blokke, når man først har vænnet sig til det.

- Jens Gram, http://www.jensgram.dk/

Relaterede artikler og ressourcer
- Pythons officielle site (http://www.python.org/)
- Python 2.2 Quick Reference (http://rgruet.free.fr/PQR2.2.html)

Change log:

```
2004/10/24: Første version publiceret
```

Kommentar af simonvalter d. 23. Dec 2004 | 1

god introduktion

Kommentar af izemate d. 25. Oct 2004 | 2

God og beskrivende artikel med eksempler

Kommentar af bernie d. 25. Oct 2004 | 3

Meget sjovt lige at læse :)

Kommentar af optical d. 02. Nov 2007 | 4

God kort intro

Kommentar af morteeart d. 24. Oct 2004 | 5

Top nice. Gode danske guides er altid godt. Skal klart til at lege med python nu :)