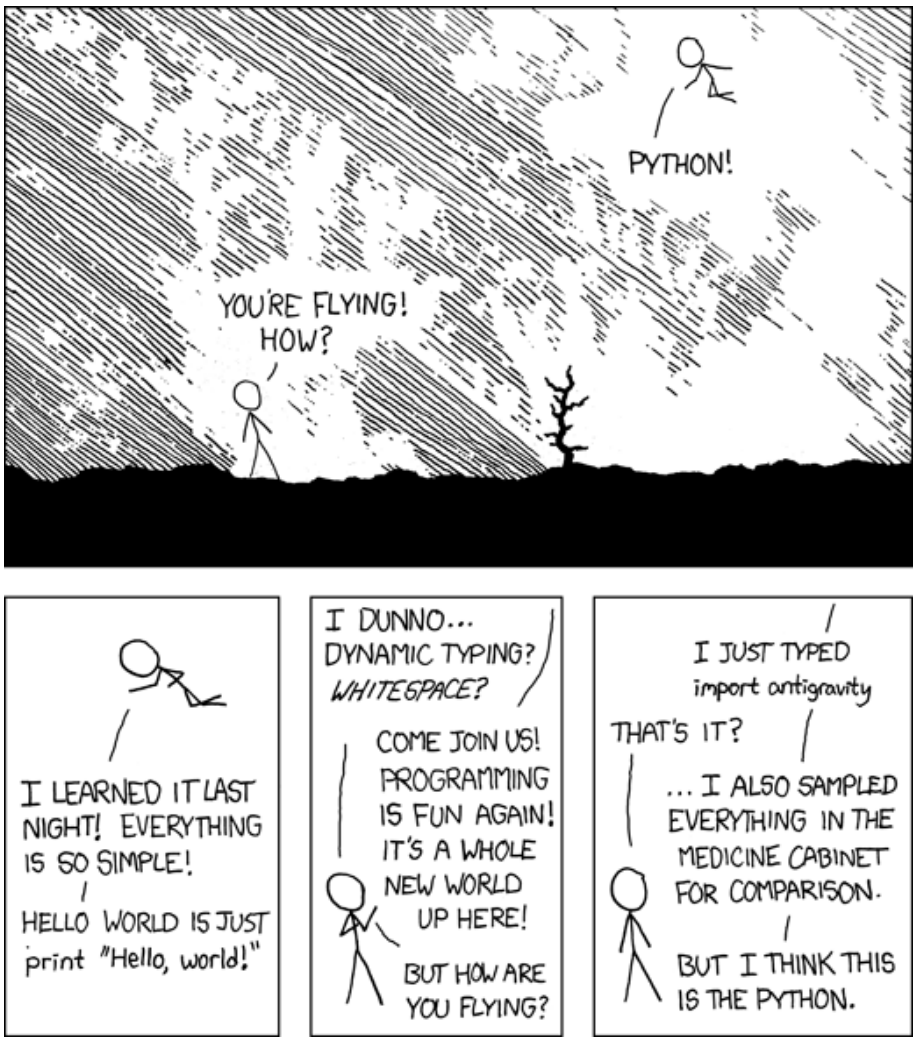
Det luktar Python

En introduktion till programmeringsspråket Python på 30 minuter

Per Erik Strandberg

1 december 2008



Innehåll

1 Om detta dokument

Detta är en introduktion till Python riktad till programmerare med liten eller ingen tidigare kunskap om Python. Författaren, Per Erik Strandberg, skrev den inför en intern presentation på Addiva Consulting. Denna text kan komma att uppdateras - i så fall finns en version med nyare datum än 1 december 2008 på författarens hemsida.

Alla bilder med streckgubbar kommer från från xkcd (med vissa förbehållna rättigheter (CC-BY-NC 2.5)) och är ritade av Randall Munroe.

2 Om Python

Python är ett objektorienterat högnivå språk som kan ses både som ett programmeringsspråk och som ett skriptspråk. Python kan vara interaktivt. Ofta sägs att Python är designat med maximering av utvecklarens produktivitet och med en mycket läsbar syntax som mål, men Pythons författare Guido van Rossum menar att målet var att få ett andra språk till C och C++-utvecklare. En tidig slogan var *bridge the gap between the shell and C*. Python är ett interpreterat språk, men vissa optimeringar kan göras genom att bytekompilera koden.

Vidare är Python minimalistisk (10 tusen rader kod i Python kan lätt motsvara 50 tusen rader Java, eller 100 tusen rader C), har ett stort standardbibliotek och är lätt att utöka med egna moduler. Guido van Rossum menar att Python är utomordentligt bra som klister mellan olika moduler skrivna i C.

Något som är ganska ovanligt i populära språk är att Python använder whitespace för att dela in koden i block (istället för till exempel måsvingar eller `begin ... end`).

Python (CPython) är skrivet i C och C++ och kan som vi ska se enkelt utökas med moduler i samma språk. Även implementeringar i Java (Jython), .NET (Ironpython) och Python (PyPy) finns.

Guido van Rossum behövde ett skriptspråk till ett operativsystem han jobbade med i slutet på 1980-talet. När han under jullovet satt och kodade (precis som alla andra nördar drömmer om att hinna med) tittade han på gamla avsnitt Monty Python. Namnet på språket blev Python och tack vare det är många av läroböckerna och exemplen i pythonvärlden inspirerade av Monty Python. Det är ganska trevligt att ha variabler som `bacon`, `egg` och `spam` istället för `foo`, `bar` och `baz`.

Idag är problemet Guido ville lösa glömt och borta men Python lever kvar och kommer i dagarna i sin 3'e stora verion: Python 3.0 (som vi presenteras i 4.5). Exempel i denna text är anpassade för Python 2..

Några stora projekt som använder Python är Zope application server, YouTube, den ursprungliga BitTorrent klienten och några stora organisationer som använder Python är Google, Yahoo!, CERN och NASA. Värt att notera är även att eShop, i princip den första ehandelsfirman gjorde en prototyp i Python. eShop, som så många andra geniala startupföretag köptes sedan av Microsoft som porterade all kod till C++ och sedan har ingen hört av den igen.

Python har framgångsrikt bäddats in i olika mjukvaruprodukter som scriptspråk. Titlar värda att nämna är Houdini, Maya, Softimage XSI, TrueSpace, Poser, Modo, Nuke och Blender. Men även GIMP, Krita, Inkscape, Scribus and

Paint Shop Pro. Python används även som kontrollspråk för moddning och event hantering i spel som Civilization IV och Mount&Blade. Eve Online, ett MMORPG, är byggt med Python Python.

Nokias valde Python som skriptspråk i sin mobiltelefonserie S60.

I alla operativsystem utom MS Windows är Python en standardkomponent (språket kommer med de flesta GNU/Linux distributionerna, med NetBSD, med OpenBSD och med Mac OS X). Red Hat Linux och Fedora använder båda den pythonisk Anaconda installern. Gentoo Linux använder Python i sin pakethanterare Portage. Srivbordsmiljön GNOME har som policy att endast tillåta C, C# och Python på sina skrivbordsapplikationer.

2.1 Hello world

2.1.1 En första hello world

En minimal hello world för Python 2.:

```
print 'hell o world'
```

2.1.2 Som modul

En komplett, dokumenterad och unix-anpassad modul på omkring 10 rader kod får vi med följande exempel.

```
#!/usr/bin/python

"Small package for illustrating hello world in Python."

def hello():
    "Say hello."
    print "hell o world"

if __name__ == "__main__":
    hello()
```

Detta skript kan man köra direkt från konsoll (på UNIX och Cygwin med `./hi.py`), köra med `python hi.py` och dessutom importera till andra script (mer om det senare).

2.2 Interaktiv pythonsession

Python har även stöd för att köra kommandon eller iterativa sessioner direkt i konsollen. Man behöver med andra ord inte nödvändigtvis tillgång till filer för att kunna använda Python - tänk en användare utan skrivrättigheter på ett system.

```
/home/per/Dropbox/python-tutorial
>python -c "print 'none shall pass'"
none shall pass

/home/per/Dropbox/python-tutorial
>python
Python 2.5.2 (r252:60911, Jul 31 2008, 17:28:52)
[GCC 4.2.3 (Ubuntu 4.2.3-2ubuntu7)] on linux2
```

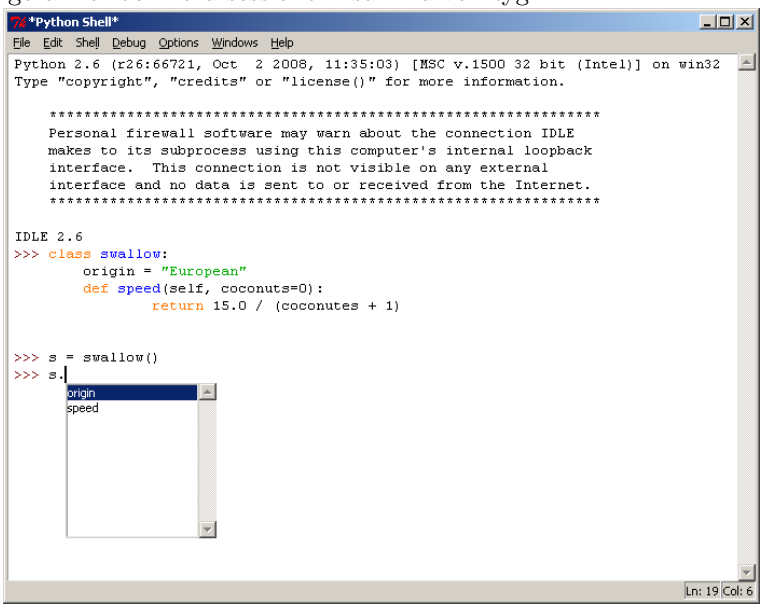
```
Type "help", "copyright", "credits" or "license" for more information.
>>> (1 + 0x17) * (1 + 0x18)
600
```

Avsluta en session med `exit()`, `exit`, `Ctrl+Z`, `Ctrl+D` eller `Ctrl+D+Retur` beroende på version och operativsystem.

2.3 Hjälp under utveckling

2.3.1 Utvecklingsverkyget IDLE

Eric Idle är en av medlemmarna i Monthy Python. Så ett verktyg som kommer med Python fick samma namn. IDLE (Interactive DeveLopment Environment) har stöd för klassvyer, syntax completion och syntax coloring. Man kan både redigera filer och köra sessioner i samma verktyg.



Mången text-editor med en monospacead font duger till att skapa Python, å andra sidan får man inte mycket stöd om man jobbar i till exempel MS Notepad.

2.3.2 `import`, `help` och `dir`

Syntaxen för att importera paket, lista innehåll i dem och få hjälp är metoderna `import`, `help` och `dir`. Att importera den hello world vi precis gjorde kan gå till enligt följande:

```
>>> import hi
>>> dir(hi)
['__builtins__', '__doc__', '__file__', '__name__', 'hello']
>>> help(hi)
Help on module hi:

NAME
    hi - Small package for illustrating hello world in Python.
```

```
FILE
    /home/per/Dropbox/python-tutorial/hi.py

FUNCTIONS
    hello()
        Say hello.
```

Funkar även på bland annat funktioner `help(hi.hello)`. Testa även funktionerna `locals()` och `globals()` för en lista på aktiva varibler.

Det finns även en ren hjälpmiljö inuti den interaktiva sessionen:

```
>>> help()

Welcome to Python 2.5!  This is the online help utility.

# [snip]

help> modules

Please wait a moment while I gather a list of all available modules...

BaseHTTPServer      bdb                 imghdr              signal
Bastion             binascii            imp                 site

# [snip]

atexit              idlelib             sha                 zlib
audiodev            ihooks              shelve
audioop             imageop             shlex
base64              imaplib             shutil

Enter any module name to get more help.  Or, type "modules spam" to search
for modules whose descriptions contain the word "spam".
```

2.3.3 Python debugger

Python kommer med en debugger, i modulen `pdb`. För detaljer se debuggerns dokumentation.

```
>>> import pdb
>>> import hi
>>> pdb.run(hi.hello())
> [snip]Dropbox/python-tutorial/hi.py(7)hello()
-> print "hell o world"
(Pdb) n
hell o world
--Return--
> [snip]Dropbox/python-tutorial/hi.py(7)hello()->None
-> print "hell o world"
(Pdb)
```

2.4 Grundläggande syntax

2.4.1 Variabler och typer

Man behöver aldrig tvinga en variabel att tillhöra en viss typ. På samma sätt har funktioner aldrig retur-typ deklarerat. Detta ger ett extremt flexibelt system - men kan för oinitierade kännas som att gå på en kombinerad SM och gay/bi-klubb för kristdemokrater och kommunister med ögonbindel och öronproppar: det är kanske lätt att få napp - men man vet aldrig riktigt vad man får (se duck typing 4.1).

Bland standardklasserna hittar vi numeriska typer för heltal (både `int` som är 32-bitar och `long` som är godtyckligt stor), flyttal, komplexa tal och decimals. `str`, `unicode`, `list` och `tuple` är typer för sekvensiell data. Även standard klasser för `set`, `dict` och `file` finns. Se 4.5 för hur dessa typer kommer ändras i Python 3.0.

```
>>> n = 42 ** 1337
>>> import math
>>> math.log10(n)
2170.2843012619928
>>> n = "str"
>>> n
'str'
>>> n = """another "str"."""
>>> n
'another "str".'
>>> n = 'another "str", yet again'
>>> n
'another "str", yet again'
>>> n = u"Önikåd"
>>> n
u'\xd6nik\xe5d'
>>> l = [0, 1, 2, 3, 4, 5]
>>> len(l)
6
>>> l[0]
0
>>> l[4]
4
>>> l[-2]
4
```

2.4.2 Syntax: indentering och funktioner

Python är känt för att ha signifikant whitespace:

```
>>>  "notera det initiala mellanslaget"
  File "<stdin>", line 1
    "notera det initiala mellanslaget"
    ^
IndentationError: unexpected indent
```

Whitespace bestämmer vart loopar, if-satser, klasser, metoder och så vidare börjar och slutar. Ett block måste ha samma whitespace (kör man med switchen

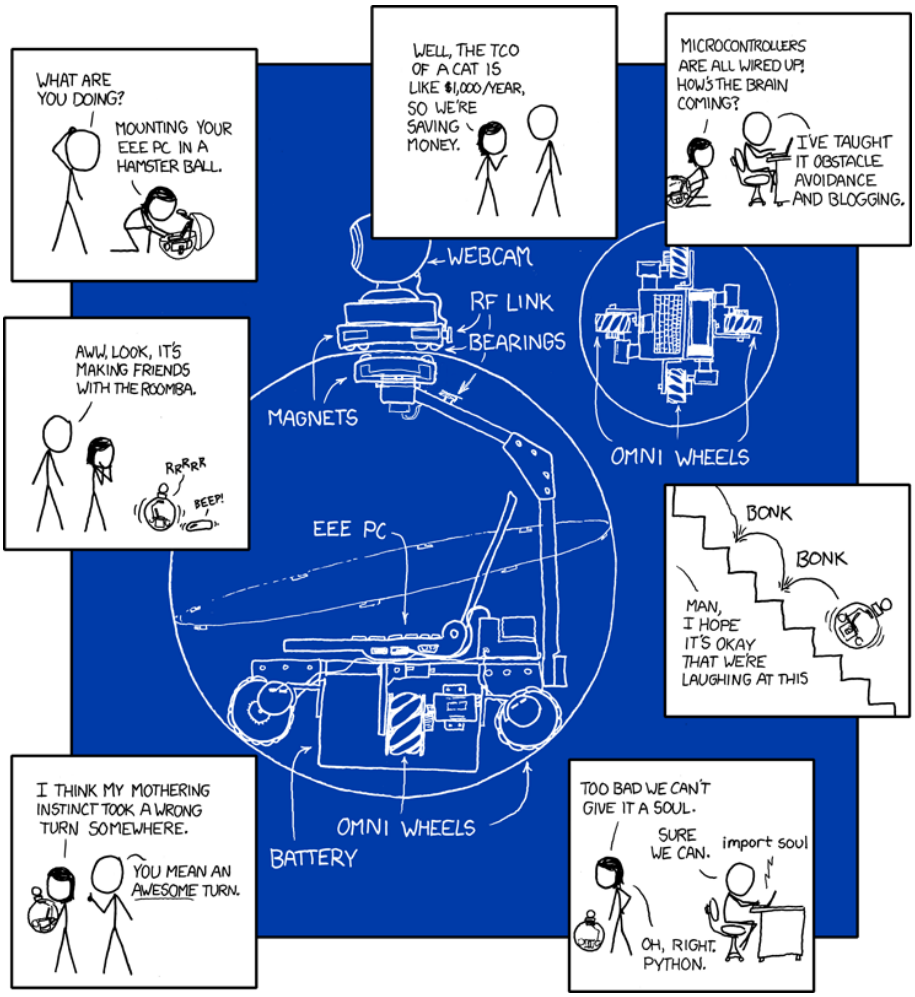
t får man en varning, förekommer switchen två gånger får man istället error, om man har en fil som blandar tabbar och space). Felaktig blandning av space och tab ger ett Exception.

```
>>> class foo:
...     "A simple class"
...     def bar(self, a, b, c = 0, d = None):
...         if d:
...             return a+b+c+d+c+b+a
...         else:
...             return a+b+c
...
>>> f = foo()
>>> f.bar(1,2)
3
>>> f.bar(1,2,d=100)
106
>>> f.bar("det", "luktar", "inte", "python")
'detluktarintepythoninteluktardet'
```

En funktion som sväljer alla argument man matar in i den följer denna syntax:

```
>>> def foo(*args, **kwargs):
...     for arg in args:
...         print arg
...     for key in kwargs.keys():
...         print key,
...         print kwargs[key]
...
>>> foo(12, 34, a="b", b=45)
12
34
a b
b 45
```

3 Om bibliotek



3.1 Standardbiblioteket

Som tidigare nämnts ingår en hel del redan i standardbiblioteket, till exempel filhantering och stränghantering:

```
i = 0
f = open("python-tutorial.tex", "r")
for line in f:
    i += 1
    if line.find("png") != -1:
        print str(i), line.strip()
```

Outputen blir i stil med detta:

```
21 \includegraphics[width=12cm]{python-antigravity.png}
177 \includegraphics[width=10cm]{idle.png}
374 \includegraphics[width=12cm]{python-import-soul.png}
```

```
387 if line.find("png") != -1:
[Recursion snipped]
465 \includegraphics[width=10cm]{gui-tk.png}
505 \includegraphics[width=10cm]{matte-plot.png}
797 \includegraphics[width=12cm]{python-warning.png}
985 \includegraphics[width=12cm]{python-next.png}
```

3.2 Att använda och installera moduler

Ni kanske minns vår lite finare `hell o world` tidigare - vi importerar den igen och låter det illustrera hur man gör för att importera moduler.

```
>>> import hi
>>> hi.hello()
hell o world
```

Moduler kan även skrivas helt i till exempel C eller C++. För att hitta moduler letar Python i ordning igenom nuvarande mapp, mappar listade i miljövariablerna `PYTHONPATH` och `PATH`.

För att installera moduler behöver man i till exempel Ubuntu bara köra `sudo apt-get install python-numpy` för att allt ska hamna på rätt plats. På andra operativsystem löser troligen större och stabila projekt detta åt användaren, annars får man själv lösa det bäst man kan.

Se även python eggs för ett system som hanterar beroenden mellan moduler på ett automatiserat sätt. Python eggs används mycket i stora projekt som Zope.

3.3 Grafiska användarinterface

Det finns flera moduler för programmering med GUI. Den som är defaktostandard heter Tkinter. Ska man göra något mer seriöst skulle mitt heta tips vara pythonbinbings till Gtk, Qt och OpenGL. Man kan även ganska enkelt rita lite mer avancerade interface i Glade som man sedan importerar till en klass - lite om Windows Presentation Foundation verkar fungera. I min introduktion till PyGtk visar jag just detta.

Detta lilla exempel gör ett gui med två knappar, trycker man på den ena knappen stänger man av applikationen, den andra skriver ett meddelande på konsollen. Notera användandet av funktionspekare.

```
from Tkinter import *

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()

        self.button = Button(frame, text="QUIT", fg="red", command=frame.quit)
        self.button.pack(side=LEFT)

        self.hi_there = Button(frame, text="Hello", command=self.say_hi)
        self.hi_there.pack(side=LEFT)
```

```
    def say_hi(self):
        print "hi there, everyone!"

if __name__ == "__main__":
    root = Tk()
    app = App(root)
    root.mainloop()
```

I Ubuntu kan resultatet se ut på detta sätt (notera utskriften på terminalen i bakgrunden).

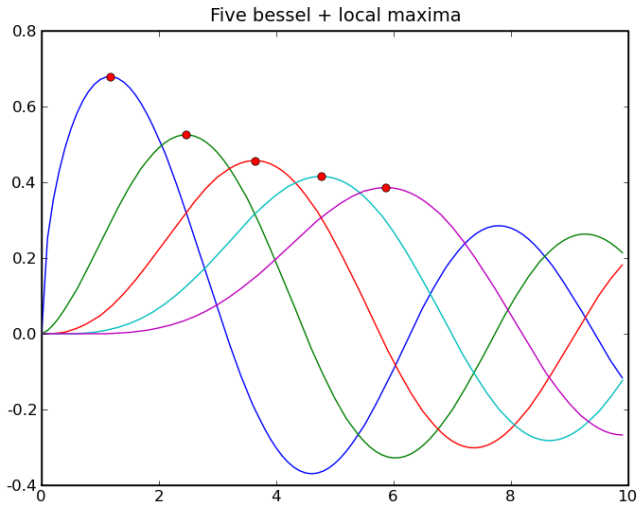
3.4 Några matematikbibliotek

På `comp.lang.python` finns en och annan tråd som jämför Python med Matlab. Klart är att det finns krafter som vill att Python med några moduler ska kunna göra i princip det Matlab gör, fast i Python. Och bättre.

Detta exempel ritar upp några kurvor och hittar lokala max-punkter. Lägg även märke till att exemplet använder en lambdafunktion och funktionspekare.

```
# On Ubuntu install python-scipy python-numpy python-matplotlib python-tk

from scipy import *
from numpy import *
from matplotlib import pyplot

x = arange( 0, 10, 0.10)

for k in arange(0.5, 5.5):
    y = special.jv(k, x)
    pyplot.plot(x, y)
    f = lambda x: -special.jv(k, x)
    x_max = optimize.fminbound(f, 0, 6)
    pyplot.plot([x_max], [special.jv(k, x_max)], 'ro')

pyplot.title('Five bessel + local maxima')
pyplot.show()
```

Resultatet blir något i stil med detta:



3.5 Databaser

Den enklaste formen av kombination av databas och Python går troligen via sqlite. Detta exempel skapar en tabell i en databas och lägger in ett värde, stänger kopplingen till databasen, öppnar en ny uppkoppling och skriver ut alla värden den hittar.

```
import sqlite3

db1 = sqlite3.connect('temp.db')
db1.execute("CREATE TABLE IF NOT EXISTS test(id int, data text)")
db1.execute("INSERT INTO test(id, data) VALUES(1, 'hello')")
db1.commit()
db1.close()

db2 = sqlite3.connect('temp.db')
print db2.execute("SELECT * FROM test").fetchall()
db2.close()
```

3.6 Paket för webb

Med Python kommer modulen `urllib2` som enligt sin dokumentation är till för att öppna url'er med olika protokoll. I kombination med tredjehandsmodulen `BeautifulSoup` - som är mycket bra på att hantera html - får vi en extremt kraftfull kombination.

Detta exempel öppnar och läser innehållet på en välkänd hemsida och plockar ut innehållet i attributet `href` i alla `a`-tagger - det vill säga alla utlänkar - och skriver ut dem.

```
import urllib2
from BeautifulSoup import BeautifulSoup

url = urllib2.urlopen("http://www.addiva.se")
```

```
soup = BeautifulSoup(url)

for link in soup('a'):
    print link['href']
```

Innehållet när jag körde skriptet gav denna output:

```
index.php
http://www.addiva.se/index.php?option=com_frontpage&Itemid=1
http://www.addiva.se/index.php?option=com_content&task=blogcategory&id=23&Itemid=33
http://www.addiva.se/index.php?option=com_content&task=blogcategory&id=24&Itemid=34
http://www.addiva.se/index.php?option=com_content&task=blogcategory&id=10&Itemid=32
http://www.addiva.se/index.php?option=com_contact&catid=5&Itemid=7
http://www.addiva.se/index.php?option=com_registration&task=lostPassword
http://www.addiva.se/index.php?option=com_content&task=view&id=41&Itemid=1
http://www.addiva.se/index.php?option=com_content&task=view&id=38&Itemid=1
http://www.addiva.se/index.php?option=com_content&task=view&id=36&Itemid=1
http://www.addiva.se/index.php?option=com_content&task=view&id=34&Itemid=1
http://www.addiva.se/index.php?option=com_content&task=view&id=30&Itemid=1
http://www.addiva.se/index.php?option=com_content&task=view&id=34&Itemid=1
http://www.addiva.se/files/marketing/reklamskylt-med-gubbar-2b-liten.jpg
```

3.7 Unit Testing i Python

Stöd för testdriven utveckling finns i modulen `unittest`. Behöver man köra lite startkod eller avslutande kod (för att stänga filer och så vidare) kombinerar man sina test med `setUp` och `tearDown`. Värt att notera är att testerna måste följa det halvmagiska namnmönstret `test*`, annars tolkas de inte som tester och körs inte.

```
import unittest

class SimpleTests(unittest.TestCase):

    nice = [(True, True), (False, False)]
    bad  = [(True, False), (False, True)]

    def testNice(self):
        for pair in self.nice:
            self.assertEqual(pair[0], pair[1])

    def testBad(self):
        for pair in self.bad:
            self.assertNotEqual(pair[0], pair[1])

    def testFail(self):
        self.assertEqual(True, False)

if __name__ == "__main__":
    unittest.main()
```

En trevlig och nästan grafisk display visas när man kör testerna och en punkt kommer om testet går bra, annars ett F.

```
.F.
======================================================================
FAIL: testFail (__main__.SimpleTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unit-test.py", line 17, in testFail
    self.assertEqual(True, False)
AssertionError: True != False

----------------------------------------------------------------------
Ran 3 tests in 0.015s

FAILED (failures=1)
```

Se även Python Unit Test i min blogg där även konceptet Unit Test förklaras lite mer.

3.8 Dokumentationsdriven utveckling med `doctest`

En utvecklingsmetodik som är sjukt produktiv och kod-effektiv är dokumentationsdriven utveckling som med modulen `doctest` har stöd i Python. Testmotorn fungerar så att den parsar dokumentationen och letar efter något som liknar efter en interaktiv session och utför det som finns där. På detta sätt kan man skriva hur man vill att sin kod ska fungera rätt in i dokumentationen, och vips så har man både dokumentation och test på samma gång!

```
"""
This is the "unique" module, supplying one function: solo(a, b, c=0), that
returns the unique input or the lowest value if they were all unique.

Two examples:
>>> solo(102, 102, 103)
103

>>> solo(101, 102, 103)
101
"""

def solo(a, b, c=0):

    """Return the unique value, or the lowest if only unique ones are used.

    Only values that pass the integer constructor can be used:

    >>> solo("6", 7.0, 0.8E1)
    6

    >>> solo("6.1", 7, 8)
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
      File "unique.py", line 22, in solo
        raise ValueError("a, b and c must be integeresque.")
    ValueError: a, b and c must be integeresque.
```

```
    Beware of the default parameter c that is zero if not specified:

    >>> solo(1, 2)
    0
    """

    from math import floor

    try:
        a = int(a)
        b = int(b)
        c = int(c)
    except:
        raise ValueError("a, b and c must be integeresque.")

    if (a == b) and (a != c):
        return c
    elif (a == c) and (a != b):
        return b
    elif (b == c) and (a != c):
        return a
    else:
        return min(a, b, c)

def _test():
    import doctest
    doctest.testmod()

if __name__ == "__main__":
    _test()
```

Som ni ser körs `_test` om man kör skriptet direkt utan `import`. Hittas något fel i dokumentationen får man ett meddelande liknande detta:

```
[snip]Dropbox\python-tutorial> python unique.py
**********************************************************************
File "unique.py", line 9, in __main__
Failed example:
    solo(101, 102, 103)
Expected:
    104
Got:
    101
**********************************************************************
1 items had failures:
   1 of   2 in __main__
***Test Failed*** 1 failures.
```

3.9 Använda C-dll'er

Att importera och använda kod från C och C++ är löjligt enkelt som detta exempel ska visa. Börja med att kompilera följande lilla c-fil med `cl /LD /Zi combo-hello.c` till `combo-hello.dll`, eller om du kör till exempel Ubuntu: `gcc -shared -o combo-hello.so combo-hello.c`

```
#include <stdio.h>

// __declspec(dllexport) void __stdcall hello() /* windows */
void hello(void) /* ubuntu */
{
  printf("C  >> Blue. No red!\n");
}
```

Med tre rader python kan vi nu importera och köra C-koden:

```
import ctypes

# mydll = ctypes.windll.LoadLibrary('combo-hello.dll')
mydll = ctypes.cdll.LoadLibrary('./combo-hello-ubuntu-gcc.so')

mydll.hello()
```

Outputen blir inte helt oväntat:

```
/home/per/Dropbox/python-tutorial$ python combo-hello-ubuntu.py
C  >> Blue. No red!
```

I en mer utförlig introduktion, som eventuellt är något föråldrad, behandlar jag bland annat in- och retur-värden, arrayer och funktionspekare. Även strukturer ska gå utan problem.

3.10 Fler moduler

Ett sätt att hitta fler moduler är att titta i PyPI (Python Package Index - tidigare Python Cheeseshop). Där finns alla stora och många små moduler listade och karakteriserade.

I till exempel Ubuntu kan man spana efter många vanliga moduler som inte ingår i standard-python genom att leta efter deb-paket som heter `python-*`, till exempel `python-crypto`.

Python kan även distribuera paket i så kallade Python Eggs, ett sätt att hantera beroenden och förenkla installationen av moduler.

Det går även bra att tanka hem py-filer och stoppa in i sin `PYTHONPATH` utan att behöva tänka så mycket på vad man gör.

4 Se upp! En liten varning

Precis som alla språk har sina egenheter finns det ett par hos Python. Några är avsiktliga men för en traditionell programmerare lite svårsmälta.

4.1 Duck typing

Python tillämpar duck typing - det vill säga att man inte gör typkoller vid funktionsanrop. Om man förväntar sig en anka och det man får *talks like a duck and walks like a duck* så är man nöjd med sin anka, oavsett vad det är. I följande lilla exempel har vi två klasser där den ena är en anka och inte den andra. Trots det passerar de utan exception igenom en funktion som förväntar sig en anka.

```
class duck:
    def talk(self, n):
        return "quack " * n

    def walk(self):
        return True

    def sink(self):
        return False

class witch:
    def talk(self, n):
        return "please don't kill me - I am not a witch..."

    def walk(self):
        return True

    def sink(self):
        return True

def spanishTest(victim):
    burn = victim.walk() or victim.sink()
    return (victim.talk(4), burn)

if __name__ == "__main__":
    d = duck()
    spanishTest(d)

    w = witch()
    spanishTest(w)
```

4.2 Monkey patch ($\pi = 3$)

Det jag nu kommer lära er får ni inte komma ihåg om ni inte lovar att använda med förstånd eller inte alls. För med en monkey patch kan man förstöra otroligt mycket. Följande exempel illustrerar baksidan med tekniken:

```
>>> import math
>>> math.pi = 3
>>> area = math.pi * 4 ** 2
>>> area
48
```

Namnet *monkey patch* är en gullig och trevlig omskrivning av namnet *gorilla patch*. En *gorilla patch* i sin tur är en gullig omskrivning av *guerilla patch*. Att det först kallades *guerilla patch* är för att man kan se det som bitar med kod som krigar med varandra om att dominera.

```
class parrot:
    def dead(self):
        print "The parrot is dead!"
        return False

def sleep(self):
    print "It's only sleeping!"
    return False

if __name__ == "__main__":
    p = parrot()
    p.dead()

    parrot.dead = sleep
    p.dead()
```

Resultatet blir:

```
The parrot is dead!
It's only sleeping!
```

Begreppet *monkey patch* har ytterligare en synonym värt att nämna: *duck punching*, som är mer i linje med *duck typing*, och som inte har en lika gullig klang.

4.3 Listor är mutabla och valfria parametrar skapas

Vi har tidigare sett att man kan skapa metoder med valfria parametrar, till exempel `def add1(l = [])`. Värt att notera är att dessa valfria parametrar skapas och ligger någonstans. Skickas ett heltal in ligger det altså ett heltal någonstans och används varje gång det behövs. Tack vare att metoder för vissa typer använder sig av anrop med referens och ibland anrop med värde är risken att man skapar en global variabel om man inte tänker sig för.

```
>>> def add1(l = []):
...     l.append(1)
...     return l
```

Test year	2000	2007
Standard Output	4.54	2.20
Hashtable	2.06	0.64
I/O	1.19	0.38
List	0.31	0.42
Interpreter Init	6.25	12.0
Object Allocation	0.11	0.034
Interpreter Speed	0.18	0.86

Tabell 1: Hastighetsratio Java/Python år 2000 och 2007. Värden större än 1 innebär att Python är snabbare.

```
...
>>> l1 = add1()
>>> l1
[1]
>>> l1 = add1()
>>> l1
[1, 1]
```

4.4 Hastighet

Ofta jämförs Python med Java. Ofta hävdas det att Python är slött. Men Python är inte Java eller något annat språk. Ska man lösa en uppgift ska man välja språk efter uppgift. I allmänhet är Java snabbare än Python och kommer troligen förbli det. Å andra sidan är Python lättare att utveckla med, kodmängen blir omkring en femtedel än motsvarande i Java, utvecklingstiden blir betydligt kortare och underhållsarbetet mycket mindre.

Med detta sagt är det dags för en inkomplett och helt ovetenskaplig jämförelse: I blogosfären finns många mer eller mindre seriösa jämförelser mellan till exempel Python och Java. I det exemplet jag hittat som gjordes 2000 och igen 2007, är Java för det mesta snabbare och ibland mycket snabbare, för vissa tester var Python snabbare. Se fler värden i tabellen.

4.5 Python 3.0

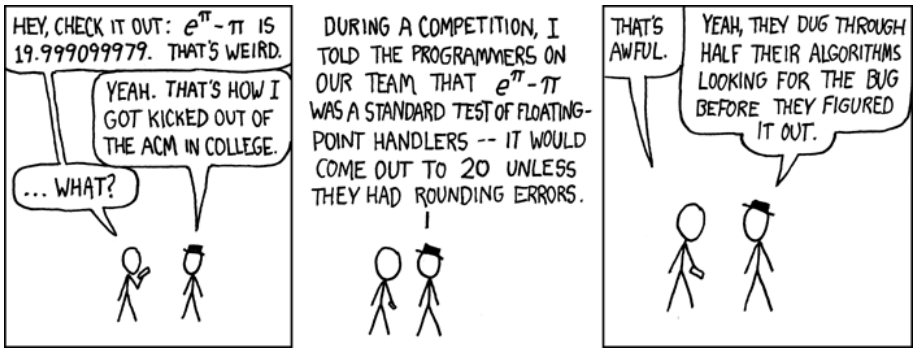
Python 3.0 (eller Python 3000, eller Py3k) är ny och fräsch (21 November 2008 kom release candidate 3). Bland de viktigare förändringarna hittar vi:

- `print` och `exception` är nu funktioner
  `print(a, b, c, sep=" ", end="\n", file=sys.stdout)`
  `except("None shall pass!")`
- Alla strängar är Unicode, för gammal ohederlig ASCII finns istället typen byte som i arrayer blir ganska lik str-typen.
- Skillnaden mellan heltalstyperna försvinner.
- Heltalsdivision returnerar flyttal som standard.

Antagligen kommer det dröja riktigt länge innan ens hälften av de viktigaste tredjehandsmodulerna blir porterade till Python, så man ska nog ligga ganska lågt med att implementera något tills Python 3.1 kommit ut. Å andra sidan är portering ganska smärtfritt: är koden körbar i Python 2.6 finns switchen -3 som ger varningar vid inkompatibilitet. När koden är utan varningar finns verktyg för att portera kod.

Se listan nyheter i release 3.0 på python.org för mer detaljer.

5 Hur fortsätter jag?



5.1 Övningsuppgifter

Några uppgifter du borde kunna lösa med hjälp av den här introduktionen är:

1. Gör en webbkrälare som hittar alla sidor på en domän givet en startsida. Internt ska alla sidor lagras i en datastruktur som givet en sida har koll på alla utlänkar från den sidan. Eventuella döda länkar ska särbehandlas.
2. Använd `doctest`-modulen för att, givet resultatet av föregående uppgift, lagrar hur många länkar varje sida på en webbplats har i en databas.
3. Givet databasen i föregående uppgift, rita ett histogram över hur distributionen.
4. Gör ett grafiskt användarinterface som när man trycker på en knapp visar innehållet i databasen ovan.
5. Gör en C-dll med en funktion som multiplicerar ett tal och till produkten adderar ett tredje. Summan ska returneras. Skriv en modul som använder dll'en och skriv vettiga unittests.
6. Installera en bildbehandlingsmodul, till exempel PIL, och skriv en metod som given en bild beräknar medelfärgen, byter alla svarta pixlar till rött, roterar och förminskar bilden.
7. Lär dig pythons debugger och skriv en tutorial för den.

5.2 Lär dig mer Python

- Dive into Python är en bok publicerad med GNU FDL och kan tankas hem gratis. Även fysiska böcker finns att köpas. Dive into Python är ett snabbt och kodfokuserat sätt att lära sig Python. Varje avsnitt börjar med kod, därefter förklaras vad man ska göra och hur man bygger upp resultatet.
- Python Challange: lite knep och knåp som är kul att lösa med Python.
- Extremt mycket annat finns om Python på internet - ett bra ställe att börja på är, förutom Pythons hemsida, Engelska Wikipedias sida om Python.