`datahora`: aritmética de datas e horas com Python

Nelson Luís Dias

4 de junho de 2008

1 – Por que Python

Em primeiro lugar, porque de vez em quando é bom mudar da mesmice de C e FORTRAN. Em segundo, terceiro, quarto, etc., lugares:

2. Portabilidade: Python roda em Windows, Linux e OS X, com uma mesma interface (*Idle*).
3. Python é muito simples e fácil de usar, inclusive interativamente, como uma "calculadora".
4. Python é grátis.

Este artigo não é destinado a ensinar Python, nem a entrar em detalhes sobre a linguagem. Uma excelente fonte de informações em Português é `http://www.pythonbrasil.com.br`. Python e sua interface, *Idle*, podem ser obtidos gratuitamente de : `http://www.python.org`.

Na sequência, vamos supor que *Idle* foi iniciada no mesmo diretório onde estão os arquivos-fonte Python (em particular, onde está `datahora.py`), em modo *shell*. Uma seção muito simples de *Idle*, por exemplo, é

```
Python 2.5.1 (r251:54863, Mar  7 2008, 04:10:12)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "copyright", "credits" or "license()" for more information.

    **********************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    **********************************************************************

IDLE 1.2.1
>>> print 'Bom dia'
Bom dia
>>>
```

Na sequência, o início de uma seção *Idle* será indicado apenas por `IDLE 1.2.1` (que é a versão que estou usando neste momento) para economizar espaço.

2 – O padrão ISO 8601

O padrão ISO 8601 define as formas universais de reportar datas e instantes. De acordo com o padrão, existem várias formas possíveis; a *única* forma que vamos usar neste artigo é a seguinte: o dia 04 de janeiro de 2008, às 18 horas, quatorze minutos e 47.71 segundos é escrito na forma

`2008-01-04T18:14:47.71`.

Assim, o ano tem *sempre* 4 dígitos, o mês 2, e o dia 2; a hora sempre varia de 00 a 23, os minutos de 00 a 59, e os segundos de 00 a 59.999999. Devido a uma restrição inerente às bibliotecas-padrão de Python, a resolução máxima temporal que poderemos ter será de 1 μs (hum microssegundo). Embora o padrão ISO admita 24 horas, 60 minutos, etc., isto introduz ambigüidade, e não será utilizado.

O padrão é pensado de tal forma que a ordem das datas/instantes é também a ordem alfabética das *strings* que os representam. Por exemplo, 03 de janeiro de 2007 vem antes de 03 de março de 2008, e, de fato:

```
IDLE 1.2.1
>>> print '2008-01-03' < '2008-03-03'
True
>>>
```

Ou seja: para Python, a *string* `'2008-01-03'` é de fato "menor" (lexicograficamente) do que a *string* `'2008-01-03'`.

Em climatologia, nós freqüentemente nos deparamos com problemas do tipo

calcular o instante `2008-06-03T11:57:05` "mais" `00:10:17.05`,

ou seja: somar 10 minutos, 17 segundos e 5 centésimos ao instante `2008-06-03T11:57:05`.

A soma é relativamente complicada, por causa da mistura das representações decimal (representando frações de segundo) e sexagesimal (representando segundos, minutos e horas):

```
 11:57:05.00
+00:10:17.05
 -----------
 12:07:22.05
```

O módulo `datahora`, e sua (até agora única) função `incdth`, implementam esta operação:

```
IDLE 1.2.1
>>> from datahora import incdth
>>> incdth('2008-06-03T11:57:05','00:10:17.05')
'2008-06-03T12:07:22.050000'
>>>
```

3 – Uso detalhado de `incdth`

A função `incdth` é uma implementação simples, baseada essencialmente em duas classes padrão existentes em Python: `datetime` e `timedelta`. O módulo `datahora` define duas funções locais: `_datetimeparse`, e `_timeparse`.

`_datetimeparse` transforma a *string* no formato ISO do primeiro argumento (no exemplo acima: `'2008-06-03T12:07:22.050000'`), em um objeto da classe `datetime`; `_timeparse` transforma a *string* do segundo argumento (no exemplo acima: `'00:10:17.05'`) em um objeto da classe `timedelta`.

Este segundo argumento *não* é uma *string* ISO8601: ele é uma variação da forma $\pm h{:}m{:}s.f$, onde $\pm$ é um sinal opcional de mais ou de menos, e, como antes, $0 \leq m \leq 59$, $0 \leq s \leq 59$, e a parte fracionária dos segundos é restrita a no máximo 999999 microsegundos: $0 \leq f \leq 999999$. A diferença é que agora: (i) m e s podem conter apenas um dígito, e (ii) h pode ter um valor arbitrário, ou seja: nós podemos adicionar ou subtrair *mais* de 24 horas. Os exemplos a seguir ilustram estes fatos:

```
IDLE 1.2.1
>>> from datahora import incdth
>>> incdth('2008-06-03T00:00:00','24:00:00')
'2008-06-04T00:00:00'
>>> incdth('2008-06-03T00:00:00','-72:00:00')
'2008-05-31T00:00:00'
>>> incdth('2008-06-03T19:49:00','5:7:3.755555')
'2008-06-04T00:56:03.755555'
>>> incdth('2006-06-03T19:49:00','-4:0:0')
'2006-06-03T15:49:00'
>>> incdth('2006-06-03T19:51:00','-4:38:17.777')
'2006-06-03T15:12:42.223000'
>>>
```

Para tornar `datahora` um módulo disponível para qualquer *shell* de *Idle* rodando em qualquer diretório, é conveniente tornar o diretório onde ele reside "globalmente" visível para Python, via a variável de ambiente `PYTHONPATH`; se você usa Linux/bash, faça algo do tipo

```
export PYTHONPATH=/home/nldias/work/microbas/
```

em algum arquivo de inicialização de sua conta/sistema, como por exemplo em `.bashrc`. Por exemplo, em meu computador a localização de `datahora` é `/home/nldias/work/microbas/datahora.py`.

Com `datahora` agora globalmente visível, é possível também escrever um *script* bem pequeno de Python com o mesmo nome da rotina (`incdth`), e colocá-lo em algum diretório apontado pela variável de ambiente `PATH`. No meu caso, um diretório conveniente é `/home/nldias/bin`; lá, reside o pequeno *script* `incdth` (o mesmo nome da função, sem extensão `.py` embora ele *seja* um arquivo-fonte Python):

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
from datahora import incdth
from sys import argv
assert len(argv) == 3
print incdth(argv[1],argv[2])
```

Isto nos permite agora "invocar" `incdth` diretamente da linha de comando; um exemplo, novamente em bash, é

```
> incdth 2008-06-03T20:14:14.1414 -4:4:4.444444
2008-06-03T16:10:09.696957
```

4 – Implementação

A listagem a seguir contém a implementação integral de `datahora`, usando apenas elementos padrão de Python.

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
# ------------------------------------------------------------------------------
# --< datahora.py:
#
# módulo que define, até agora, uma única rotina 'externa' (incdth), para fazer
# aritmética com datas e horários
#
# Nelson Luís Dias
# 2008-06-03T11:51:03
# 2008-06-03T11:51:05
# ------------------------------------------------------------------------------
from datetime import *
from math import modf
# ------------------------------------------------------------------------------
# --> _datetimeparse
# ------------------------------------------------------------------------------
def _datetimeparse(a) :
   '''Data/hora -> obj. datetime.
```

```
      _datetimeparse:
      Transforma uma string do tipo YYYY-MM--DDTHH:MM:SS.XXXXXXX em um objeto datetime.

   '''
# --------------------------------------------------------------------------------
# usa uma string separada para a parte fracionária dos segundos
# --------------------------------------------------------------------------------
   if len(a) > 19 :
      mcs = a[19:]
# --------------------------------------------------------------------------------
# agora calcula o número de microssegundos
# --------------------------------------------------------------------------------
      dcs = int(float(mcs) * 1000000)
   else :
      dcs = 0
# --------------------------------------------------------------------------------
# agora constrói um objeto datetime: data e hora, até segundos
# --------------------------------------------------------------------------------
   dts = a[0:19]
   tas = datetime.strptime(dts,"%Y-%m-%dT%H:%M:%S")
   tbs = timedelta(days=0,seconds=0,microseconds=dcs)
   tas = tas + tbs
   return tas
# --------------------------------------------------------------------------------
# --> _timeparse
# --------------------------------------------------------------------------------
def _timeparse(b) :
   '''Hora -> obj. timedelta.

      _timeparse:
      Transforma uma string do tipo [+-]HH:MM:SS.XXXXXXX em um objeto timedelta.

   '''
# --------------------------------------------------------------------------------
# há um sinal na string?
# --------------------------------------------------------------------------------
   if b[0] == "+" :
      a = b[1:]
      sig = 1.0
   elif b[0] == "-":
      sig = -1.0
      a = b[1:]
   else:
      sig = 1.0
      a = b
# --------------------------------------------------------------------------------
# separa, talvez de forma mais eficiente, os campos numéricos da string a
# --------------------------------------------------------------------------------
   c = a.split(':')
   assert len(c) == 3
   hh = int(c[0])
   mm = int(c[1])
   fs = float(c[2])
# --------------------------------------------------------------------------------
# parte inteira e fracionária dos segundos
# --------------------------------------------------------------------------------
   sf = modf(fs)
   mu = int(sf[0]*1000000)
   ss = int(sf[1])
# --------------------------------------------------------------------------------
# quantos dias?
# --------------------------------------------------------------------------------
   dd = hh / 24
   hh = hh % 24
# --------------------------------------------------------------------------------
# se havia um sinal de menos, troca o sinal de *todos* os campos
# --------------------------------------------------------------------------------
   if sig < 0.0 :
      dd=-dd
      hh=-hh
      mm=-mm
      ss=-ss
```

```
      mu=-mu
   tbs = timedelta(days=dd, hours=hh, minutes=mm, seconds=ss, microseconds=mu)
   return tbs
# --------------------------------------------------------------------------
# --> incdth
# --------------------------------------------------------------------------
def incdth(zas,zin) :
   '''
Soma uma data a um certo número de horas, minutos e segundos.

------------------------------------------------------------------------------
incdth:
          incdth('2008-06-02T10:59:54.437','36:28:16.577')
          incdth('2008-06-02T16:12:37.999','-40:28:15.75')
          incdth('2008-06-02T16:12:37','4:55:20')

Note que o ponto decimal é facultativo em cada um dos dois argumentos.

------------------------------------------------------------------------------
   '''
# --------------------------------------------------------------------------
# a data/hora
# --------------------------------------------------------------------------
   tas = _datetimeparse(zas)
# --------------------------------------------------------------------------
# o incremento temporal
# --------------------------------------------------------------------------
   tin = _timeparse(zin)
   return (tas+tin).isoformat()
```

5 – Conclusões

Este artigo apresentou uma pequena aplicação (`datahora/incdth`) que utiliza os pontos fortes de Python (simplicidade de expressão de algoritmos; facilidade de implantação de espaços de nomes globalmente visíveis; quase-universalidade de plataformas; e bibliotecas-padrão amplas e de "largo espectro") para implementar uma tarefa banal, mas "chata": a aritmética com datas e horas. Isto foi feito de forma totalmente compatível com o padrão ISO8601 para a representação de datas e horas e que tem uma equivalência perfeita na classe-padrão `timedate` de Python, implementando-se ao mesmo tempo um formato razoavelmente flexível de *strings* para representar a classe padrão `timedelta`. O resultado é uma rotina capaz de adicionar/subtrair a uma data/instante um número arbitrário de horas, minutos e segundos (e frações), tanto dentro de um programa Python como diretamente na linha de comando.