Math 56: MATLAB vs python code example

Alex Barnett, May 3, 2013

MATLAB and python are quite similar (both high-level interpreted languages that can be interactive, or use script or function files). Going back and forth is like going from Spanish to Italian. Main differences: MATLAB has trailing semicolons to prevent output, has `end` to close conditionals or loops; python uses *indentation* (leading spaces) to indicate a conditional or looped block (no `end`), colon to start the block. Array indexing is also different. To help you, here's the same code in both. Please study them, and the simple algorithm itself (a variant will come up on Tuesday 5/7/13).

MATLAB version

```
function printbinary(n)
% printbinary(n) prints integer n in binary

if n<0, error('n cannot be negative'), end
if n==0, fprintf('0\n'); return, end

% t is largest power of 2 not exceeding n
t = 2^floor(log(n)/log(2));

while t>=1
  if n>=t, fprintf('1'), n = n-t;
  else, fprintf('0')
  end
  t = t/2;
end
fprintf('\n')    % terminate the string
```

The above must be a file called `printbinary.m`. Here's its test script (a separate file, but you could put it in the same file if you made it a function too):

```
% test the printbinary function
disp('the following two should match')
n = 153637;
printbinary(n)
dec2bin(n)
for n=0:10, printbinary(n); end    % a loop
try, printbinary(-4)   % should fail nicely
catch, 'n<0 error correctly detected', end
```

Note the use of `try` and `catch` to test error reporting without halting the program.
Can you spot a place in the code for improvement in reliability at large `n` ?

python version

```
from math import *      # needed for logarithm

def printbinary(n):
    """printbinary(n), print integer in binary
    """
    if n<0:
        print 'n cannot be negative'; return
    if n==0:
        print '0'; return
    # t is largest power of 2 not exceeding n
    t = 2**floor(log(n)/log(2))
    s = ''            # output string
    while t>=1:
        if n>=t: s = s+'1'; n = n-t  # appends
        else: s = s+'0'
        t = t/2
    print s

# main script, run if imported/reloaded, etc
print 'the following two should match'
n = 153637; printbinary(n); print bin(n)
for n in range(11):  # loops from 0 to 11-1=10
    printbinary(n)
printbinary(-4)      # should fail gracefully
```

Say this file is named `pb.py`. You can run in two ways: 1) from the UNIX/Mac terminal via `python pb.py`, or 2) from the python interactive environment via `execfile("pb.py")`. The latter is closest to running a Matlab script, since the variables are still accessible, useful for debugging. You can also 3) `import pb` which loads it as a *module*, then run `pb.printbinary(...)`. Modules are the way to build bigger packages. For changes to take effect you must `reload(pb)`.
Notice use of a "docstring" for documentation; once imported, typing `help pb.printbinary` shows it.