Buffered versus unbuffered output for Python in Windows

I use NppExec/Notepad++ to run Python scripts, and flush my output buffers continuously during runtime to update my console window with print statements (The default buffered output only shows *all* print statements after the script finishes execution).

[This](http://superuser.com/questions/381942/stop-nppexec-from-trapping-console-output-until-program-finishes) link shows that all you need to do is use the command python -u to get unbuffered output. Is there a downside to using this execution mode for **all** my Python scripts, *regardless of the editor used*? I'm not clear about the difference between a buffered an unbuffered output.

**EDIT**: I included this small Python timer script for an example:

#!usr/bin/env python

import time

import threading

import sys

class Timer(threading.Thread):

def \_\_init\_\_(self, seconds):

self.runTime = seconds

threading.Thread.\_\_init\_\_(self)

def run(self):

counter = self.runTime

for sec in range(self.runTime):

print counter

time.sleep(1.0)

counter -= 1

print "Done."

if \_\_name\_\_ == '\_\_main\_\_':

t = Timer(10)

t.start()

How much difference does a buffered and unbuffered output make in terms of efficiency in this case?

Buffered output means that the computer spools the output somewhere in memory until a certain amount has accumulated. Then it writes the entire block at once. This is more efficient than using unbuffered output which writes the output as soon as you request for it to be written.

The downside is that your programs will run a little (or a lot) slower depending on how much output you're writing. If they're short programs which don't do too much output, you're unlikely to notice a difference...

**EDIT**

buffered vs. unbuffered output isn't only a python question. The same concepts (and terms) apply to other languages as well. In lower level languages it becomes even more important in some ways -- If I write messages in a C program using buffered output, and then my program dies because of a programming error, any of the data that was spooled before the error, but not written to disk is lost. That's not such a problem because it is reasonably difficult to get the python interpreter to abort on an error -- even if your code is bad, the interpreter still cleans up alright in the end...(unless you send a kill signal to it or something)...