Testing admons

hpl

May 3, 2013

1 Introduction

First some ordinary text to compare font sizes in admonitions and the surrounding text.

1.1 Code

Need some code outside admons for color and font comparisons:

```
def some_code(x):
    return sin(x)*exp(1-x)
```

And some plain text verbatim:

```
x=1.0 y=0.9 z=0.4
x=1.1 y=0.3 z=0.1
```

1.2 Admonitions

Let us start with a plain warning environment.

Warning.

And here is a warning about something to pay attention to. We test how the heading behave and add quite some extra texts in comparison with the other admons.

- and a list
- with items

The next admonition features a title "Note, eventually!".

Note eventually!

Ah, we are soon close to the end. But first a bit of math:

$$p = q$$

Question.
So, how many admonition environments does Doconce support?

Question.
1. Once more, how many admonition environments does Doconce support?

Hint.
It is smart to read on and remember to
1. stay cool
2. read hints carefully

Also, remember

```
import urllib

def grab(url, filename):
    urllib.urlretrieve(url, filename=filename)
```

1.3 Going deeper environments

Here is a long notice environment with a custom title and much text, math and code.

Going deeper.
We have some equations that should be preceded by much text, so the task is to write and write. The number of words, and not the meaning, is what counts here. We need desperately to fill up the page in the hope that some admonitions will experience a page break, which the LaTeX environment should handle with ease.

Let us start with some equations:

$$\frac{Du}{dt} = 0$$
$$\frac{1}{2} = 1/2$$
$$\frac{1}{2}\boldsymbol{x} = \boldsymbol{n}$$

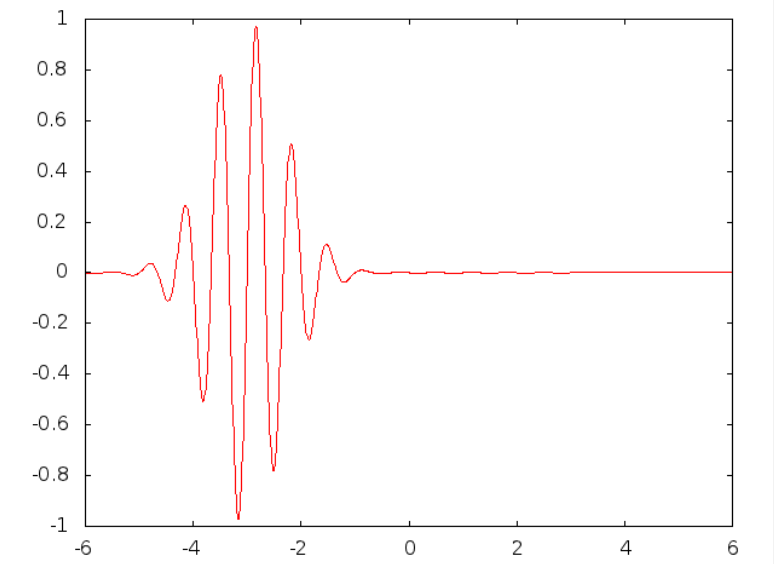
The implementation of such complicated equations in computer code is task that this "Going deeper" environment targets.

```
def Dudt(u):
    r = diff(u, t) + u*grad(u)
    return r

half = 0.5
x = 2*n
```

And some more text that can help going into the next page. Longer computer code requires vertical space:

```
    class Diff:
        def __init__(self, f, h=1E-5):
            self.f = f
            self.h = float(h)

    class Forward1(Diff):
        def __call__(self, x):
            f, h = self.f, self.h
            return (f(x+h) - f(x))/h

    class Backward1(Diff):
        def __call__(self, x):
            f, h = self.f, self.h
            return (f(x) - f(x-h))/h

    class Central2(Diff):
        def __call__(self, x):
            f, h = self.f, self.h
            return (f(x+h) - f(x-h))/(2*h)

    class Central4(Diff):
        def __call__(self, x):
            f, h = self.f, self.h
            return (4./3)*(f(x+h)   - f(x-h))  /(2*h) - \
                   (1./3)*(f(x+2*h) - f(x-2*h))/(4*h)

    class Central6(Diff):
        def __call__(self, x):
            f, h = self.f, self.h
            return (3./2) *(f(x+h)   - f(x-h))  /(2*h) - \
                   (3./5) *(f(x+2*h) - f(x-2*h))/(4*h) + \
                   (1./10)*(f(x+3*h) - f(x-3*h))/(6*h)

    class Forward3(Diff):
        def __call__(self, x):
            f, h = self.f, self.h
            return (-(1./6)*f(x+2*h) + f(x+h) - 0.5*f(x) - \
                    (1./3)*f(x-h))/h
```

And then we add a figure too.



1.4 The end

A bit of text before the summary, which we now call "Concluding remarks, for the novice", just because we can.

Concluding remarks, for the novice: We can summarize the most important things with admons: they have a different typesetting, and they may have a symbol. Titles should be optional.