Simple usage of the PyDSTool VCL image

Open PyScripter application from desktop icon. **PyScripter is the editor and debugger.** Edit and save a python script named mytest.py to the Documents folder (should be already selected as the default location). Simply add the text

```
print "hello"
```

in place of 'pass' on the line after 'def main():'. The indentation is crucial, but fortunately the editor takes care of it almost every time!

Open the IPython application from desktop icon. **IPython is the interactive command prompt for python.** Type

```
cd
```

to make IPython change to your Users home directory, then

```
cd Documents
```

to navigate to the Documents folder (TAB will complete after the first three letters have been typed) where PyScripter saves files by default, then

```
%run mytest
```

(again, TAB will complete after the first two or three letters). The % means a special IPython command follows. (The other useful one is %edit, which will open a temporary notepad file to run a set of commands.) You should see 'hello' appear in IPython.

CTRL-d will exit IPython (just press RETURN to confirm exit). All the things changed in python's memory are lost at exit, so scripts must be re-run when you re-open IPython. Fortunately, during your VCL session you will be able to use the UP arrow on your keyboard to access previously entered commands (from this or previous IPython sessions, but not previous VCL sessions) that you wish to repeat. This can save you a lot of time.

Using PyScripter for debugging

Add the following lines to your main() function, keeping the same indent as the print statement you added earlier:

```
a = 0
print 'Value is', 1.5/a
```

This comma separation is a simple way to print a mixture of strings and values to the screen. In addition, add the lines

```
from __future__ import division
```

```
b = 'nothing'
```

with *no* indent on a line above the definition of 'main'. The first line is a trick to ensure that the version of Python we are using correctly divides integers to yield floating point values. (You should have this at the top of every script you create, just to be safe.) The second line creates a global variable b with value equal to a character string, 'nothing'.

Now try to run the program within PyScripter using the little green 'play' icon (a little less efficient for computations, but handy to check out a script while you're developing it). An error will be raised. It's pretty clear why this happened, but let's pretend we don't understand what happened and would like to investigate more closely.

At the side of the 'print 1.5/a' statement there will be a little bullet point in the margin. Click it once to add a little red dot with a tick. This sets a BREAKPOINT in your code. Now re-run your program, this time using the icon to the right of the play button: it's a smaller play button with a bug behind it.

PyScripter should have stopped at the breakpoint before executing the problem line, which should appear in blue. If it's not already showing in the panel below, select the Python Interpreter tab below. It should show

```
[Dbg]>>>
hello
```

Put the cursor focus in there and press RETURN to give you a new >>> prompt. You can now explore the values of variables in the function that is currently running. For instance, type

```
b
```

to see that this global variable is visible inside the function and has the string value 'nothing'. Now type

```
a
```

to see that it has value 0. This is not good for the division on the next line! Change the value live inside the function call by entering

```
a = 3
```

at the [Dbg]>>> prompt. Then resume the debugger using the same button (or F9). You should now see

```
Value is 0.5
```

successfully appear in the interpreter window. Now that you've seen how to fix this error during debugging, and the program has stopped, you can permanently change the value of a in the main function to be non-zero.

An alternative way to see some of this information during debugging is to switch to the

Variables tab, where a will be shown to be zero. The Call Stack tab also tells you that you are inside the main function of the mytest module. ("Module" is the python name for any distinct python file.)

Now, create a function in mytest called 'reciprocal':

```
def reciprocal(x):
    """Return the reciprocal of any real number x"""
    return 1/x
```

The triple-quoted string (a doc string) is a convention to allow us to remember what the function does. The function returns a value that it computes. It does not 'print' any value to the screen.

Change the last line of main() to be (with appropriate indent)

```
print 'Reciprocal of %f is %f' % (a, reciprocal(a))
```

The %f character is a string formatting command (the same as in C and Matlab) that tells python to take a value from the arguments following the print command and render them as a floating point value when printing to the screen.

Now run this script from IPython. Once it has finished, try calling `reciprocal` from the prompt for different numbers. What exactly happens when you call it with the value 0?

To prevent this being a problem, edit the reciprocal function in PyScripter to look like this:

```
def reciprocal(x):
    """Return the reciprocal of any real number x"""
    try:
        return 1/x
    except ZeroDivisionError:
        import numpy
        return numpy.Inf
```

Save the file. This has *only* changed the file on disc, *not* what is live and currently running within IPython. Now, instead of re-running the script in IPython (which we could have done here because it's computationally cheap to re-run it) we can replace the definition of this function live in the IPython session. Type

```
%edit
```

then copy-and-paste just this function definition into the notepad window that pops up. Then save and exit it. Some text will show up on the screen to confirm what was executed. Now try calling the function with zero for the argument. What is the screen output?

Finally

From Explorer, copy your mytest.py file to permanent storage before logging out, so that you have a record of it for a future session.