Robinson Analytics

Using Python to Harness Windows

Tutorial Notes

O’Reilly Python Conference, Monterey, 21-24 August 1999

Andy Robinson, Robinson Analytics Ltd.

These notes closely follow the slides for the tutorial and include all code samples.

Table of Contents

Part 1: - Fundamentals

1 Background

1.1 What we will cover

- How Python works on Windows
- What's in Pythonwin
- Building applications with Python and COM
- Getting started on common tasks
    - Automating Office applications
    - Connecting to databases
    - Communications
    - GUI libraries

1.2 What we won't cover

- Low-level Windows internals
- Hardcore COM - how it works
- NT Services
- NT processes, events and threading models

2 What is Python good for on Windows?

2.1 An integration tool

- Works with files
- Works with DLLs and C programs
- Works with COM
- Works with Networks
- Works with Distributed Objects

2.2 "Low-threat" needs that Python fills in the corporate world

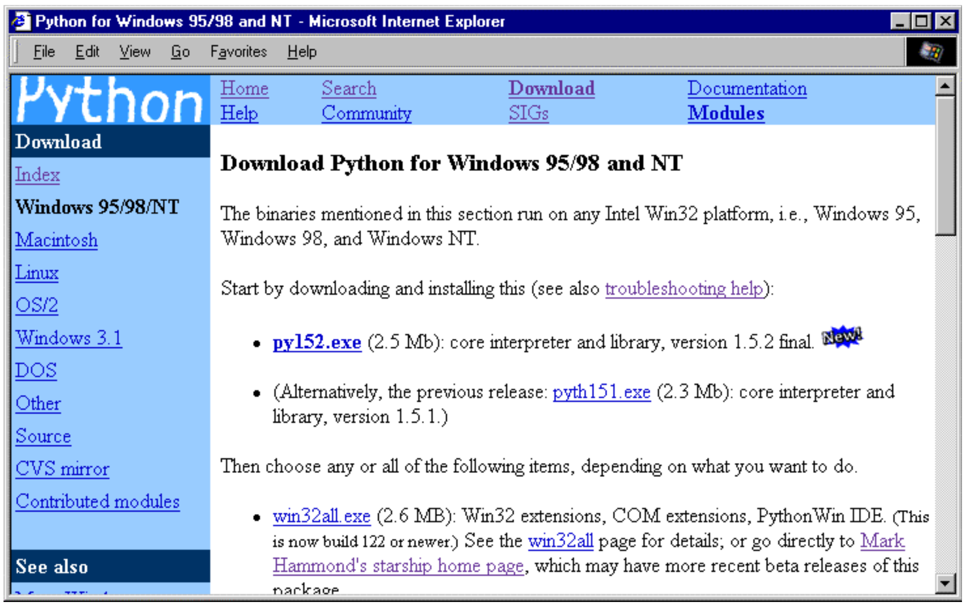
- Adding a macro language to applications
- Rapid Prototyping of object models and algorithms
- Building test harnesses for other systems
- Data Cleaning and Transformation
- Python as Glue

3 How Python works on Windows

3.1 Installation and setup

Two files to download from http://www.python.org/download/download_windows.html:

- py152.exe – Python itself
- win32all.exe – Windows extensions


Python for Windows 95/98 and NT - Microsoft Internet Explorer
File Edit View Go Favorites Help
Python
Home Search Download Documentation
Help Community SIGs Modules
Download
Index
Windows 95/98/NT
Macintosh
Linux
OS/2
Windows 3.1
DOS
Other
Source
CVS mirror
Contributed modules
See also
New!

Download Python for Windows 95/98 and NT

The binaries mentioned in this section run on any Intel Win32 platform, i.e., Windows 95, Windows 98, and Windows NT.

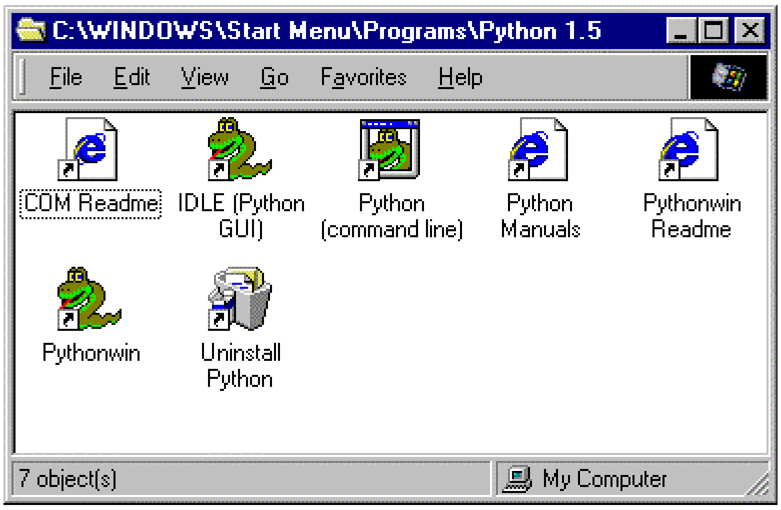
Start by downloading and installing this (see also troubleshooting help):

- **py152.exe** (2.5 Mb): core interpreter and library, version 1.5.2 final.
- (Alternatively, the previous release: pyth151.exe (2.3 Mb): core interpreter and library, version 1.5.1.)

Then choose any or all of the following items, depending on what you want to do.

- win32all.exe (2.6 MB): Win32 extensions, COM extensions, PythonWin IDE. (This is now build 122 or newer.) See the win32all page for details; or go directly to Mark Hammond's starship home page, which may have more recent beta releases of this package.

What you end up with:



3.2 The Python Core on Windows

python15.dll – 545kb, the language, exports almost everything

python.exe – 5kb console mode program

pythonw.exe – 6kb non-console mode program – avoids ugly black DOS boxes when you don't want standard input/outpu

Note: some people like to copy python.exe and pythonw.exe to their system directory, especially on Win95/98

Extensions and their meaning

.py	Python source.
.pyc	"Compiled" python source
.pyd	Extension module written in C – actually a DLL which has been renamed to .pyd
.pyw	(advanced) – a Python source file you wish to have run with pythonw.exe, not python.exe.

py, pyx and pyw all runnable with double-click (or right-click and choose Run).

Working with the command prompt on Win95/98

You need Python on your path, or a doskey macro!

```
C:\Scripts> doskey p="C:\Program Files\Python\Python.exe" $*
C:\Scripts>p hello.py
Hello from Python

C:\Scripts>doskey n=start notepad.exe $*
C:\Scripts>doskey pw=start pythonwin.exe $*
C:\Scripts>n hello.py
C:\Scripts>pw hello.py
```

Note also that you can drag filenames and directories from explorer into MSDOS window.

Working with the command prompt on NT

Much nicer! Readline-like recall with up and down arrows.

NT knows what a *py* file is, so you can type:

```
C:\Scripts>hello.py
Hello from Python

C:\Scripts>
```

You can go one further with the PATHEXT variable. To kmake it permanent, go to Control Panel | System | Environment:

```
C:\Scripts>echo %PATHEXT%
.exe;.bat;.cmd
C:\Scripts>set PATHEXT=%PATHEXT%;.py
C:\Scripts>echo %PATHEXT%
.exe;.bat;.cmd;.py
C:\Scripts>hello
Hello from Python

C:\Scripts>
```

..and of course you can use NT's other command line tools, like the scheduler to run Python jobs.

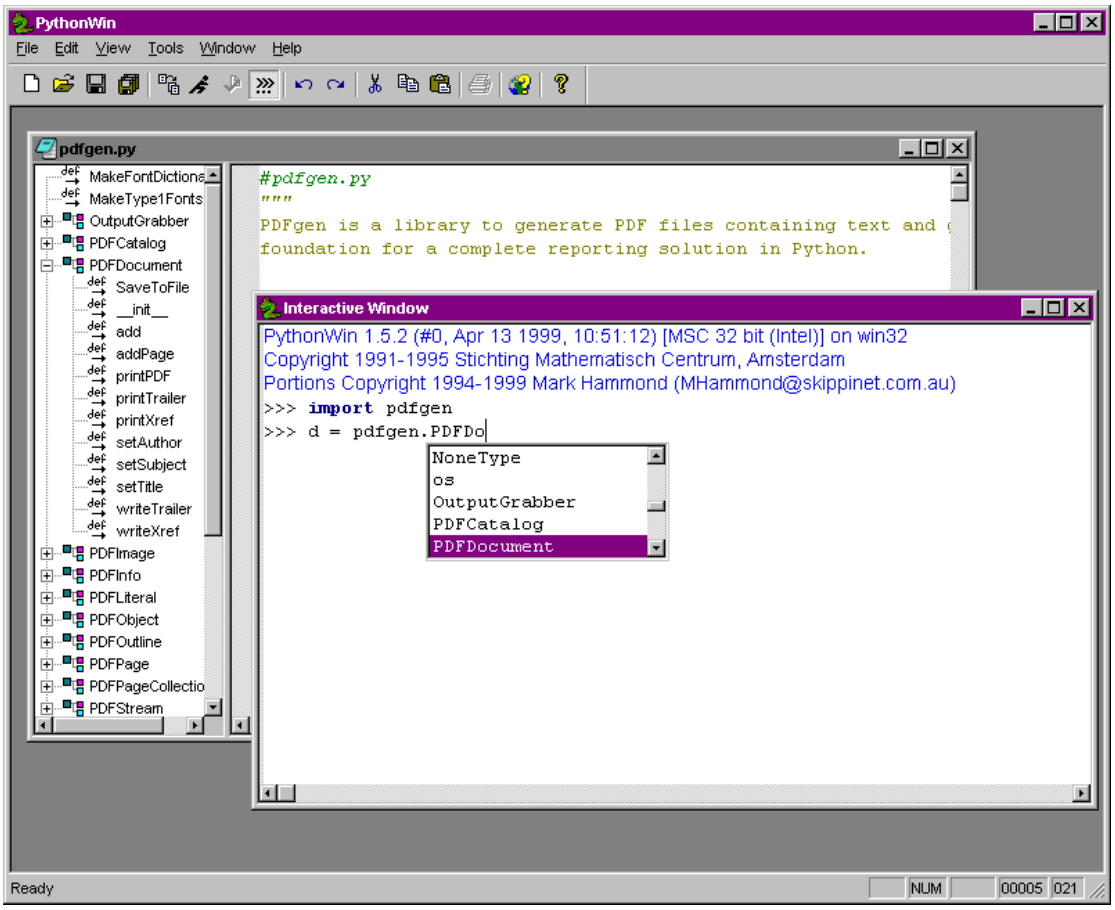
3.3 The Python for Windows Extensions

win32all includes:

- the win32 extensions
- the Pythonwin editor and MFC framework
- The PythonCOM framework
- Lots of help and examples

4 The Pythonwin IDE

Pythonwin 2.0:



```
#pdfgen.py
"""
PDFgen is a library to generate PDF files containing text and (
foundation for a complete reporting solution in Python.
```

```
PythonWin 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)] on win32
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
Portions Copyright 1994-1999 Mark Hammond (MHammond@skippinet.com.au)
>>> import pdfgen
>>> d = pdfgen.PDFDo
```

Key features:

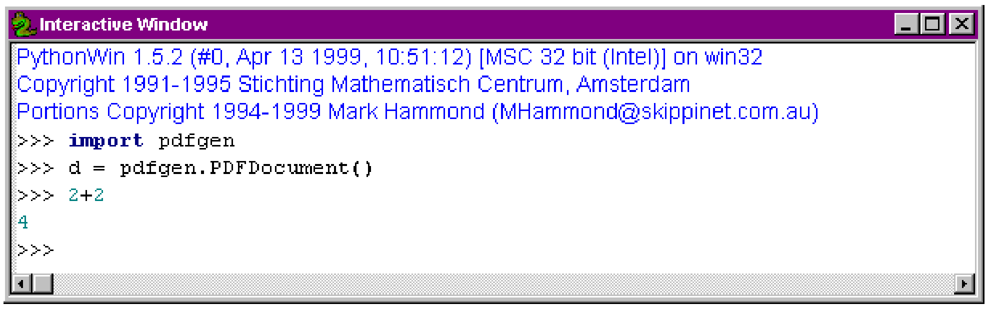
- C editor component
- Syntax coloring
- drop-down completion (as far as is possible in Python) and argument lists
- class and function browser which operates across modules

4.1 Modes

Pythonwin support a number of command line parameters:

Command Line	**Description**
/edit filename	Starts Pythonwin, and opens the named file for editing
/run filename	Starts Pythonwin, and runs the specified script.
/nodde	Must be the first parameter. Starts Pythonwin without DDE support, allowing for multiple Pythonwin instances. See Pythonwin and DDE later in this section
/app appmodule	Treats the named file as a Pythonwin application. This is for advanced users only, and is discussed in Chapter ?? - GUI Development.

4.2 Interactive window



```
PythonWin 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)] on win32
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
Portions Copyright 1994-1999 Mark Hammond (MHammond@skippinet.com.au)
>>> import pdfgen
>>> d = pdfgen.PDFDocument()
>>> 2+2
4
>>>
```

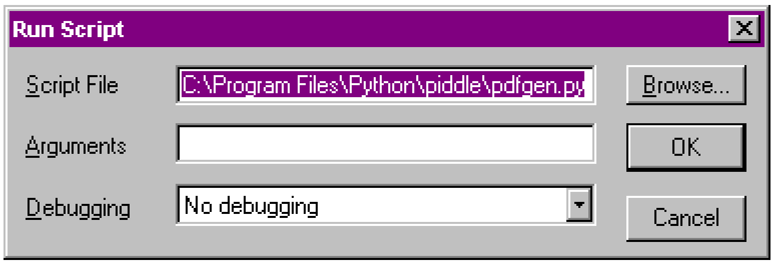
Recalls previous lines

Drop-down completion available

4.3 Import feature

Saves, and reloads all necessary files

4.4 Script dialog



Script File: C:\Program Files\Python\piddle\pdfgen.py
Arguments:
Debugging: No debugging

For scripts that work with files, know what directory you are in!

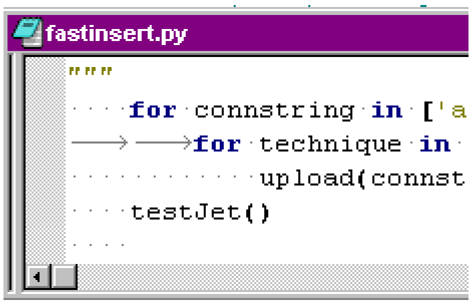
4.5 File | Locate

Searches path, checks in packages too

4.6 Source Code checking and tools

File | Check invokes TabNanny

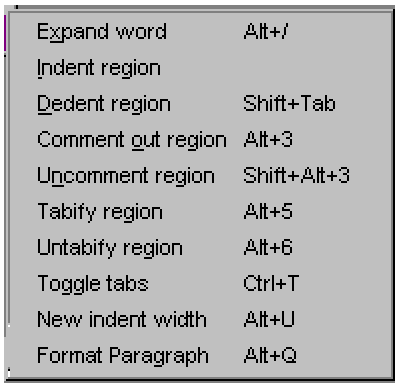
Right click and View Whitespace shows tabs/spaces:



fastinsert.py

```
"""
···for·connstring·in·['a
——→——→for·technique·in·
··········upload(connst
···testJet()
····
```

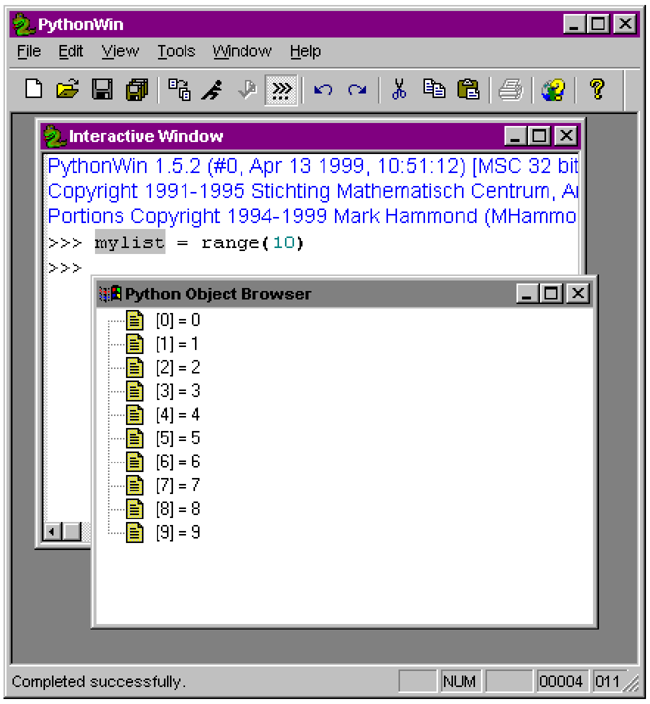
Some nice source tools, and no doubt more to come…from the context menu:



Expand word	Alt+/
Indent region	
Dedent region	Shift+Tab
Comment out region	Alt+3
Uncomment region	Shift+Alt+3
Tabify region	Alt+5
Untabify region	Alt+6
Toggle tabs	Ctrl+T
New indent width	Alt+U
Format Paragraph	Alt+Q

4.7 Old Object Browsers

Browse the entire top-level namespace, or a single object.



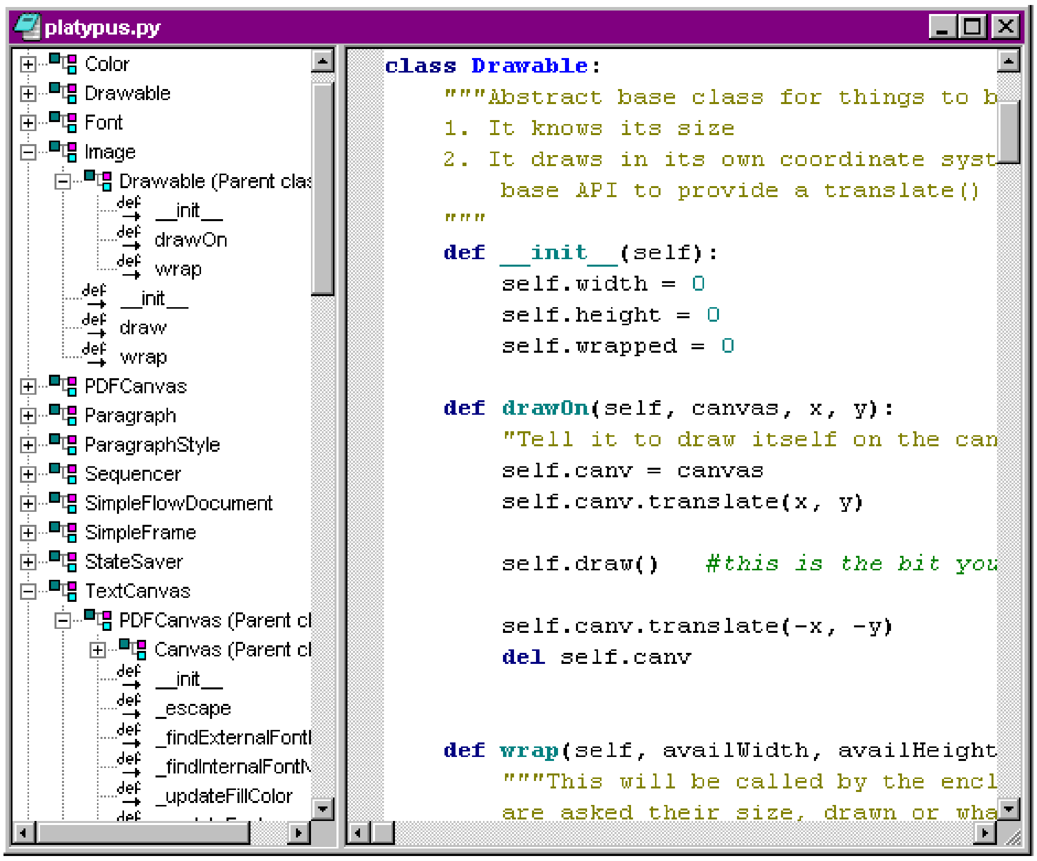
```
PythonWin 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit
Copyright 1991-1995 Stichting Mathematisch Centrum, A
Portions Copyright 1994-1999 Mark Hammond (MHammo
>>> mylist = range(10)
>>>
```

```
[0] = 0
[1] = 1
[2] = 2
[3] = 3
[4] = 4
[5] = 5
[6] = 6
[7] = 7
[8] = 8
[9] = 9
```

Completed successfully.

4.8 New Browser

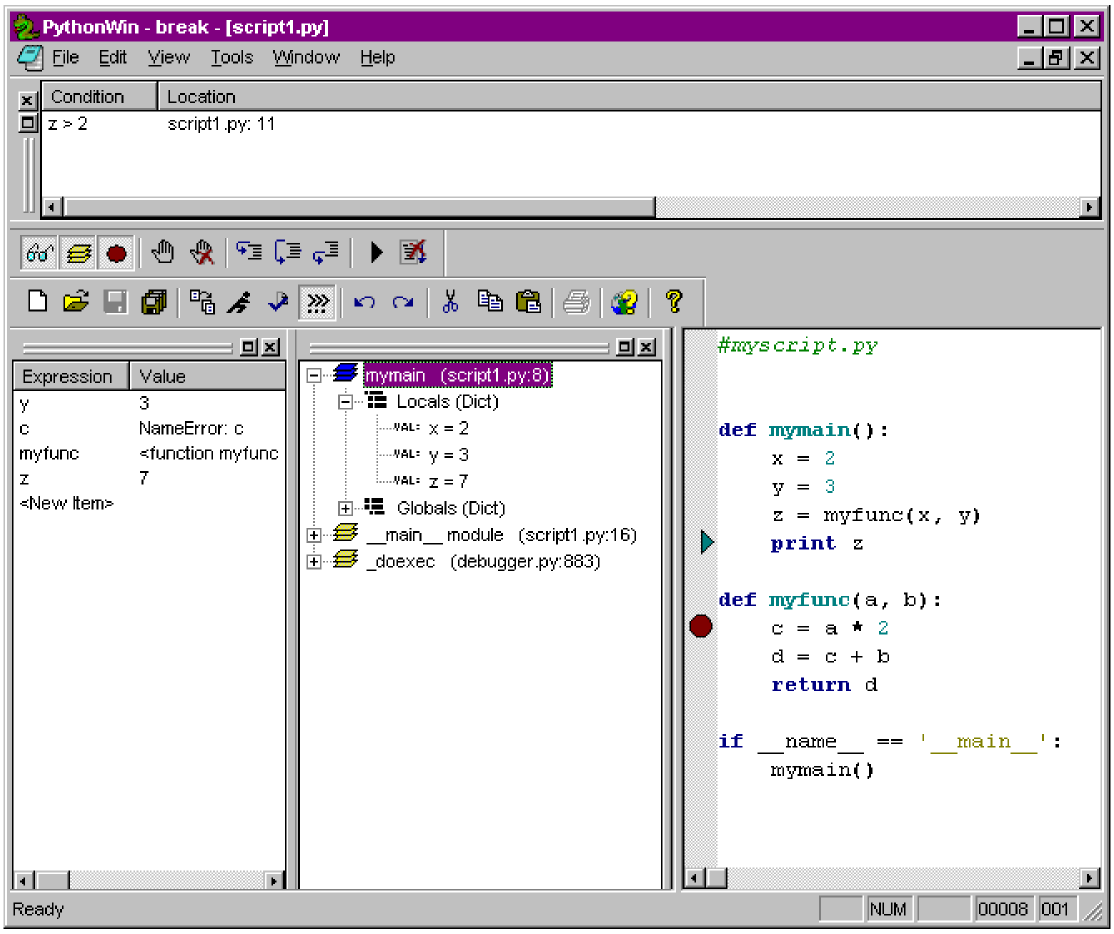
Left pane of any script window



```
class Drawable:
    """Abstract base class for things to b
    1. It knows its size
    2. It draws in its own coordinate syst
       base API to provide a translate()
    """
    def __init__(self):
        self.width = 0
        self.height = 0
        self.wrapped = 0

    def drawOn(self, canvas, x, y):
        "Tell it to draw itself on the can
        self.canv = canvas
        self.canv.translate(x, y)

        self.draw()    #this is the bit you

        self.canv.translate(-x, -y)
        del self.canv


    def wrap(self, availWidth, availHeight
        """This will be called by the encl
        are asked their size, drawn or wha
```

- Browses in-memory objects, must import first
- drill down to instance variables and base classes
- jumps to code definition, opening another script window if necessary

4.9 Debugging


PythonWin - break - [script1.py]
File Edit View Tools Window Help

Condition	Location
z > 2	script1.py: 11

Expression	Value
y	3
c	NameError: c
myfunc	<function myfunc
z	7
<New Item>	

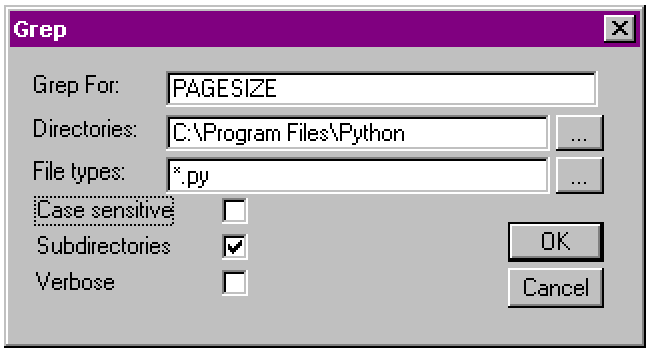
```
mymain   (script1.py:8)
    Locals (Dict)
        VAL: x = 2
        VAL: y = 3
        VAL: z = 7
    Globals (Dict)
__main__ module   (script1.py:16)
_doexec   (debugger.py:883)
```

```
#myscript.py


def mymain():
    x = 2
    y = 3
    z = myfunc(x, y)
    print z

def myfunc(a, b):
    c = a * 2
    d = c + b
    return d

if __name__ == '__main__':
    mymain()
```

Currently stepping through, at ‘print z’ line in right pane.

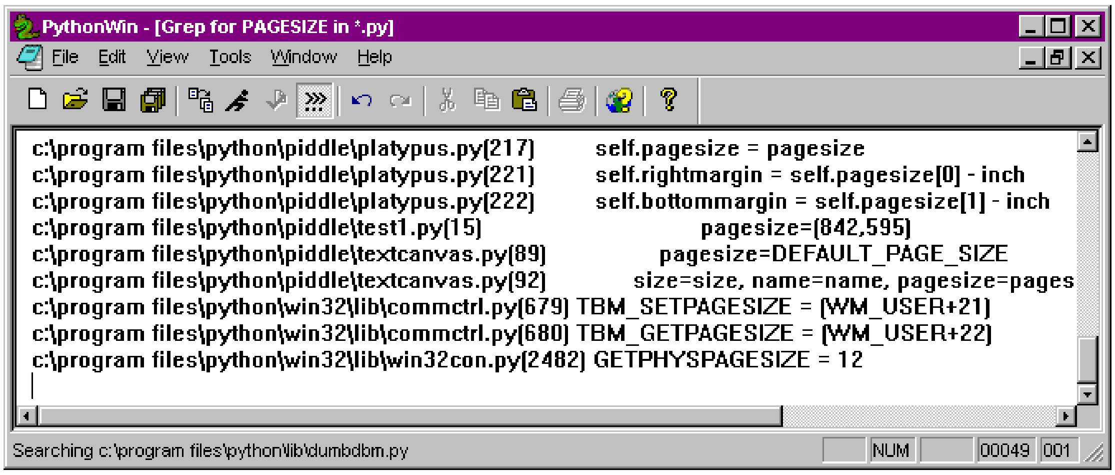
- Conditional breakpoints
- breakpoints
- watch list
- Stack
- Code Browser if you wish!

4.10 Grep



Grep For: PAGESIZE
Directories: C:\Program Files\Python
File types: *.py
Case sensitive [ ]
Subdirectories [✓]
Verbose [ ]

…leads to…



```
c:\program files\python\piddle\platypus.py(217)        self.pagesize = pagesize
c:\program files\python\piddle\platypus.py(221)        self.rightmargin = self.pagesize[0] - inch
c:\program files\python\piddle\platypus.py(222)        self.bottommargin = self.pagesize[1] - inch
c:\program files\python\piddle\test1.py(15)                    pagesize=(842,595)
c:\program files\python\piddle\textcanvas.py(89)            pagesize=DEFAULT_PAGE_SIZE
c:\program files\python\piddle\textcanvas.py(92)          size=size, name=name, pagesize=pages
c:\program files\python\win32\lib\commctrl.py(679) TBM_SETPAGESIZE = (WM_USER+21)
c:\program files\python\win32\lib\commctrl.py(680) TBM_GETPAGESIZE = (WM_USER+22)
c:\program files\python\win32\lib\win32con.py(2482) GETPHYSPAGESIZE = 12
```

Searching c:\program files\python\lib\dumbdbm.py

Click any line to go to source file.

4.11 Conclusion

- evolving fast,
- extensible
- not too shabby for a free editor!

Part 2: - COM

5 Introduction to Python and COM

5.1 What's COM about anyway?

COM

- Lets objects in different languages talk to each other
- Lets objects in different processes talk to each other
- Lets objects on different machines talk to each other
- Hides the details from the programmer
- No performance penalties compared to DLLs

Most big apps expose their functionality through COM servers. You can borrow their functionality for your own programs.

Programming for Windows is like being in a sea of objects all waiting to help you.

Discuss: Windows – the most open Operating System?

The Registry: where COM objects find out about each other. (Not just a big INI file!)

5.2 A Minimal COM Client

Connect to Excel and insert some data

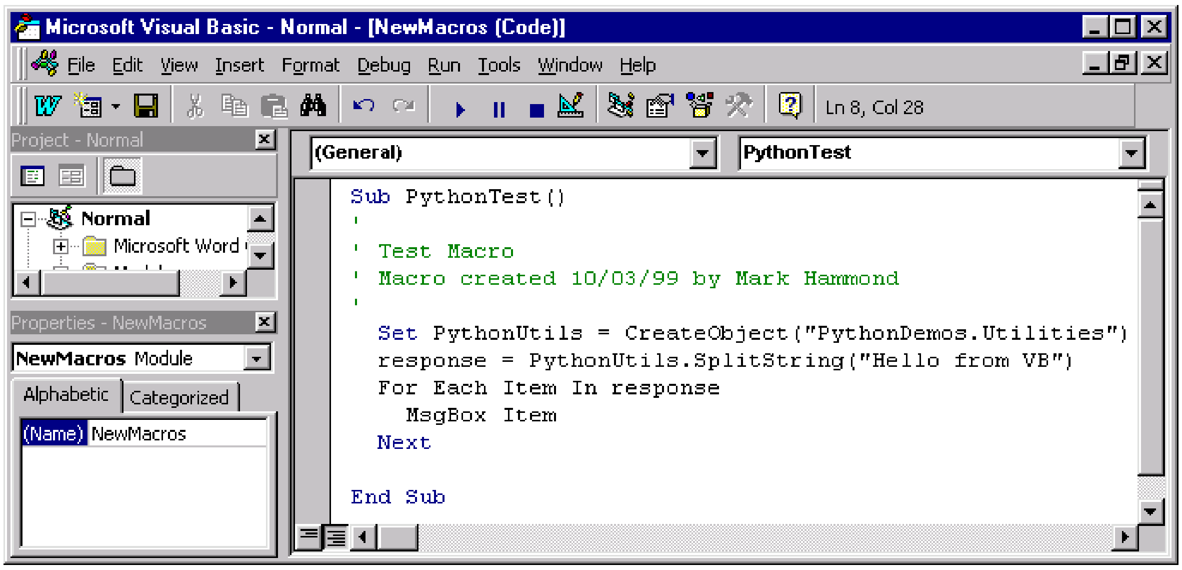
```
>>> from win32com.client import Dispatch
>>> xlApp = Dispatch("Excel.Application")
>>> xlApp.Visible = 1
>>> xlApp.Workbooks.Add()
<win32com.gen_py.Microsoft Excel 8.0 Object Library.Workbook>
>>> xlSheet.Cells(1,1).Value = 'What shall be the number of thy
counting?'
>>> xlSheet.Cells(2,1).Value = 3
```

Remember to install your Office Object Model Documentation!

5.3 A Minimal COM Server

```
# SimpleCOMServer.py - almost as small as they come!
class PythonUtilities:
    _public_methods_ = [ 'SplitString' ]
    _reg_progid_ = "PythonDemos.Utilities"
    # NEVER copy the following ID
    # Use "print pythoncom.CreateGuid()" to make a new one.
    _reg_clsid_ = "{41E24E95-D45A-11D2-852C-204C4F4F5020}"

    def SplitString(self, val, item=None):
        import string
        if item != None: item = str(item)
        return string.split(str(val), item)

# Add code so that when this script is run by Python.exe, it
self-registers.
if __name__=='__main__':
    print "Registering COM server..."
    import win32com.server.register
    win32com.server.register.UseCommandLine(PythonUtilities)
```

5.4 Using the minimal server from VB or VBA



```
Sub PythonTest()
'
' Test Macro
' Macro created 10/03/99 by Mark Hammond
'
  Set PythonUtils = CreateObject("PythonDemos.Utilities")
  response = PythonUtils.SplitString("Hello from VB")
  For Each Item In response
    MsgBox Item
  Next

End Sub
```

5.5 Why write Python COM Servers?

- Easiest way to expose Python functionality to your own or other applications
- Python is best at the business logic, other tools are best at other things (e.g. VB GUIs)

5.6 Doubletalk – Sample Application

Python Financial Modelling Toolkit. Models “Sets of Books” and “Transactions”

Good candidate for this architecture because

- Very wide general applicability – from local data input app to back-office server
- Every company needs to customize it a little!

How could we sell it?

- 100% Native Windows GUI
- Distributed, Dynamic Multi-tier Network Architecture
- Embedded Scripting Language – lets you customize the way it works!
- Extensible Plug-In Architecture
- Command Prompt for Power Users
- Integration with Word and Excel
- Open Database Connectivity
- Option to run critical tasks on Unix servers without changing a line of code!
- *Totally Buzzword Compliant!*

Now to discuss what the app is about:

5.7 Transactions

Crudely, a movement of money.

All accounts must sum to zero!

Simple two-line ("Double-Entry")

Date:	01/01/1998
Comment:	Start the company
Cash	+10 000
Share Capital	-10 000

Multi-line

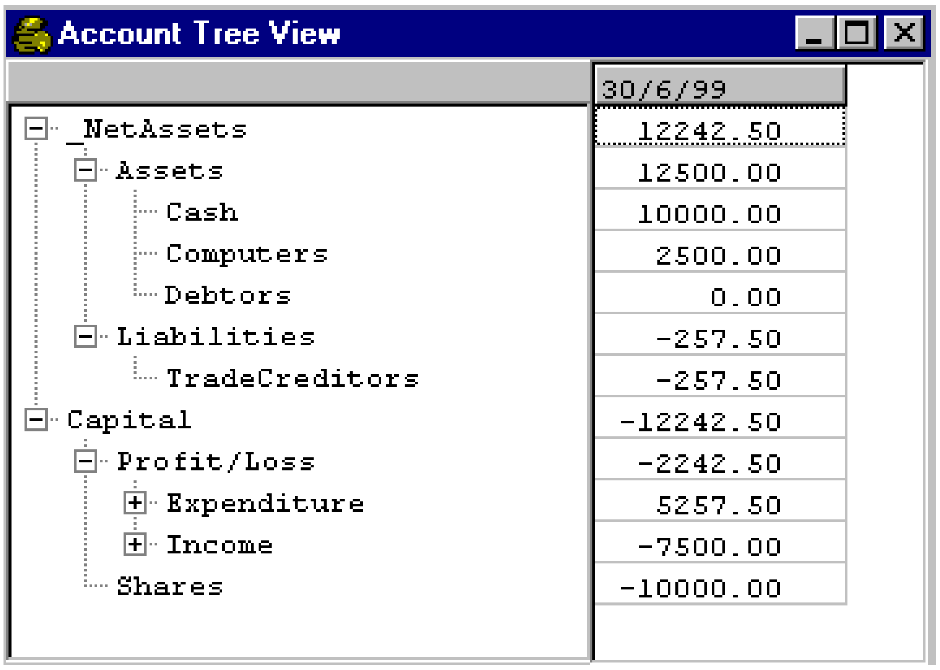
Date:	10/03/1999
Comment:	Sell Widgets
Cash	+117.50
Sales Category 1	-50.00
Sales Category 2	-30.00
Sales Category 3	-20.00
Sales tax on all three (owed to Customs & Excise)	-17.50

Functionality:

- Store
- Edit
- Add
- Validate
- effectOn(self, account)
- Extra keys/values
- add, multiply – an algebra for financial transactions!

5.8 Accounts

Accounts form a tree – this is the “Balance Sheet”…



```
                                    30/6/99
_NetAssets                          12242.50
    Assets                          12500.00
        Cash                        10000.00
        Computers                    2500.00
        Debtors                         0.00
    Liabilities                      -257.50
        TradeCreditors               -257.50
Capital                            -12242.50
    Profit/Loss                     -2242.50
        Expenditure                  5257.50
        Income                      -7500.00
    Shares                         -10000.00
```

- Represent tree as dotted string notation: “MyCo.Assets.Cash.PiggyBank”
- Assets, Cash and Expenditure are positive; Liabilities, Income and Profit are negative.

5.9 BookSets

A wrapper around a list of transactions.

- Load/Save with cPickle (one of Python’s killer features!)
- Import/Export ASCII text, list/dictionary/tuple structures etc.

Fundamental change operations

- Add/Edit/Delete transactions
- Rename Account

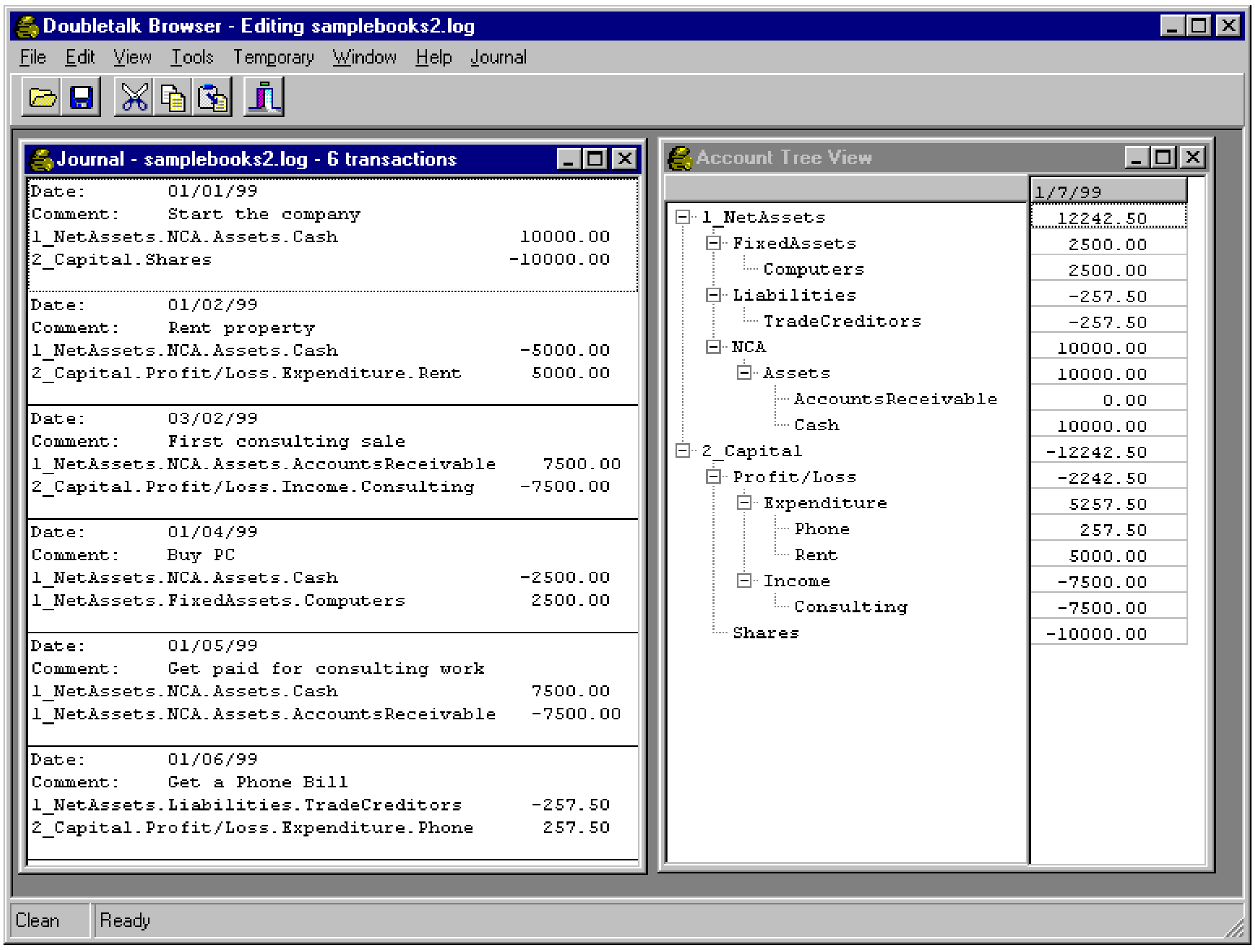
Querying

- get history of an account
- get the ‘tree of accounts’
- get all balances on date -> Balance Sheet report
- get all changes between two dates -> Profit & Loss reports

Advanced

- map from one accounts structure to another
- analyse and trace cash flows
- Multidimensional analysis

5.10 What we'd like…


Doubletalk Browser - Editing samplebooks2.log
File Edit View Tools Temporary Window Help Journal
Journal - samplebooks2.log - 6 transactions
Account Tree View
Clean
Ready

```
Date:         01/01/99
Comment:      Start the company
1_NetAssets.NCA.Assets.Cash                     10000.00
2_Capital.Shares                               -10000.00

Date:         01/02/99
Comment:      Rent property
1_NetAssets.NCA.Assets.Cash                     -5000.00
2_Capital.Profit/Loss.Expenditure.Rent           5000.00

Date:         03/02/99
Comment:      First consulting sale
1_NetAssets.NCA.Assets.AccountsReceivable        7500.00
2_Capital.Profit/Loss.Income.Consulting         -7500.00

Date:         01/04/99
Comment:      Buy PC
1_NetAssets.NCA.Assets.Cash                     -2500.00
1_NetAssets.FixedAssets.Computers                2500.00

Date:         01/05/99
Comment:      Get paid for consulting work
1_NetAssets.NCA.Assets.Cash                      7500.00
1_NetAssets.NCA.Assets.AccountsReceivable       -7500.00

Date:         01/06/99
Comment:      Get a Phone Bill
1_NetAssets.Liabilities.TradeCreditors           -257.50
2_Capital.Profit/Loss.Expenditure.Phone           257.50
```

	1/7/99
1_NetAssets	12242.50
FixedAssets	2500.00
Computers	2500.00
Liabilities	-257.50
TradeCreditors	-257.50
NCA	10000.00
Assets	10000.00
AccountsReceivable	0.00
Cash	10000.00
2_Capital	-12242.50
Profit/Loss	-2242.50
Expenditure	5257.50
Phone	257.50
Rent	5000.00
Income	-7500.00
Consulting	-7500.00
Shares	-10000.00

5.11 Design Patterns for the COM Server

COM servers and Python apps handle some arg types differently…

- Unicode String Handling – Gotcha Number One! (hopefully goes in 1.6)

```
    # our ordinary save method for use from Python
    def save(self, filename):
        f = open(filename,'wb')
        cPickle.dump(self.__journal,f)
        f.close()

    # what we would need for use from COM
    def save(self, unicode_filename):
        # convert it to a python string:
        python_filename = str(unicode_filename)

        f = open(python_filename,'wb')
        cPickle.dump(self.__journal,f)
        f.close()
```

- Wrap/Unwrap subobjects

…so a single class not the best design for real apps. Others options:

- COM Base Class, Python Server
- Pure Python Base Class, COM Subclass
- COM interface, Python Delegate

We go for option 3: Delegate. Keeps our Python package pure and portable.

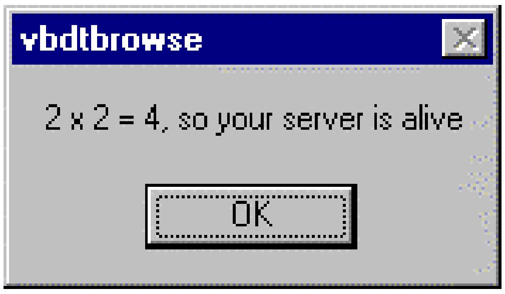
Startup Code:

```
# comservers.py – to be expanded
class COMBookSet:
    _reg_clsid_ = '{38CB8241-D698-11D2-B806-0060974AB8A9}'
    _reg_progid_ = 'Doubletalk.BookServer'
    _public_methods_ = ['double']

    def __init__(self):
        self.__BookSet = doubletalk.bookset.BookSet()

    def double(self, arg):
        # trivial test function to check it is alive
        return arg * 2

if __name__ == '__main__':
    win32com.server.register.UseCommandLine(COMBookSet)
```

5.12 Visual Basic GUI Startup Code

```
Public BookServer As Object

Private Sub MDIForm_Load()
    InitCOMServer
    frmJournal.Show
End Sub
Private Sub MDIForm_Unload(Cancel As Integer)
    CloseCOMServer
End Sub

Sub InitCOMServer()
    'called when the program starts
    On Error GoTo InitCOMServer_error
    Set BookServer = CreateObject("Doubletalk.BookServer")
    Exit Sub

InitCOMServer_error:
    Dim msg As String
    msg = "There was an error trying to initialize the
BookServer." + _
            "Please check that it is properly registered and try
the Python " + _
            "test functions first.  The program will now abort."
    MsgBox msg
    End
End Sub

Sub CloseCOMServer()
    Set BookServer = Nothing
End Sub

Sub TestCOMServer()
    'just to check it is alive
    Dim hopefully_four As Integer
    hopefully_four = BookServer.Double(2)
    MsgBox "2 x 2 = " & hopefully_four & ", so your server is
alive"
End Sub

Private Sub mnuToolsTestServer_Click()
    'this helps establish if the COM server is alive
    'using a minimal diagnostic function in the modMain module
    TestCOMServer
End Sub
```

With a little luck…


vbdtbrowse

2 x 2 = 4, so your server is alive

5.13 Our first view – The Journal

Goal:

Date-Ordered List of Transactions

Python Code Needed:

```
# more methods for COMBookSet - must be named in
_public_methods_
    def load(self, filename):
        self.__BookSet.load(str(filename))

    def count(self):
        # return number of transactions
        return len(self.__BookSet)


    def getTransactionString(self, index):
        return self.__BookSet[index].asString()
```

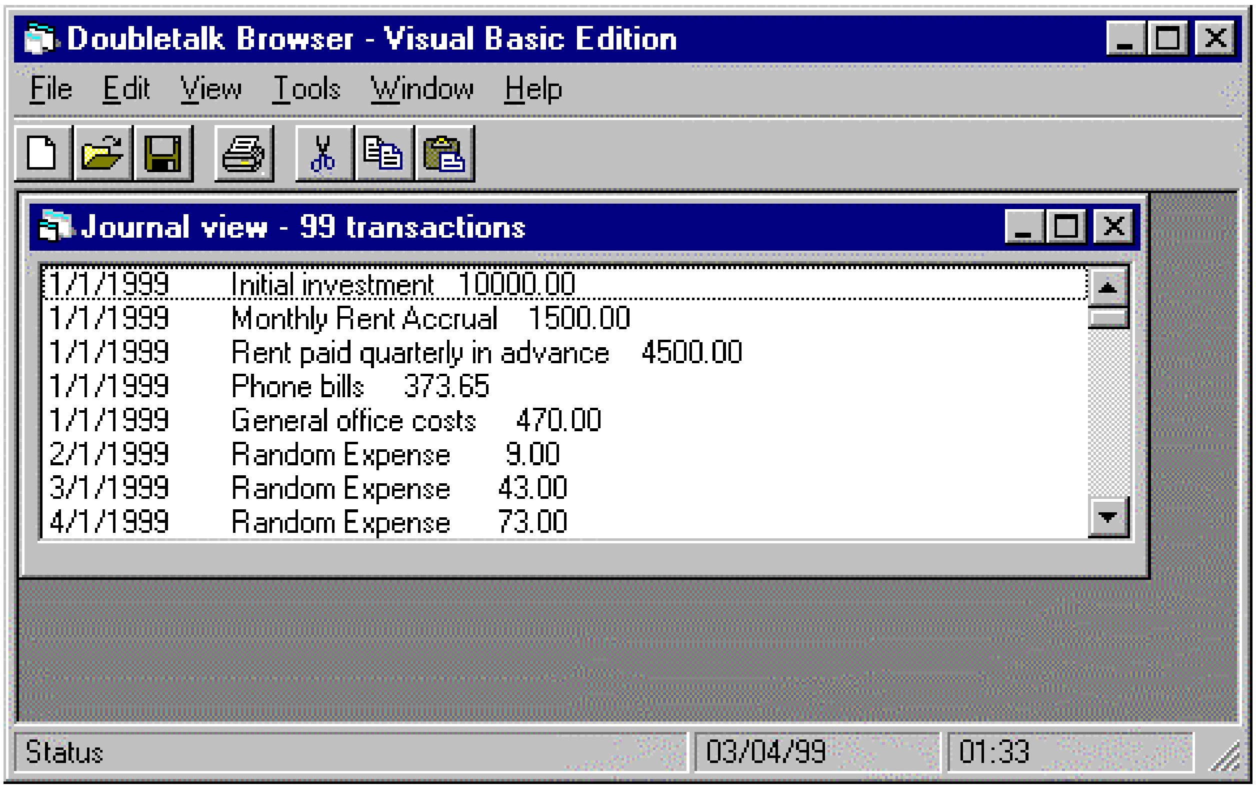
Visual Basic Code – File / Open handler

```
Private Sub mnuFileOpen_Click()
    Dim sFile As String
    With dlgCommonDialog
        .DialogTitle = "Open"
        .CancelError = False
        'ToDo: set the flags and attributes of the common dialog
control
        .Filter = "Doubletalk Journal Files (*.dtj)|*.dtj"
        .ShowOpen
        If Len(.FileName) = 0 Then
            Exit Sub
        End If
        sFile = .FileName
    End With
    BookServer.Load sFile

    'display something helpful in the Journal caption
    frmJournal.Caption = sFile & ", " & BookServer.count & "
Transactions"
End Sub
```

Visual Basic – The Journal View

```
Public Sub UpdateView()
    'make a list with a string describing each transaction

    Dim count, i As Integer
    Dim trantext As String
    Dim tran As Object

    Screen.MousePointer = vbHourglass
    lstJournal.Clear

    For i = 0 To frmMain.BookServer.count - 1
        trantext = frmMain.BookServer.getOneLineDescription(i)
        lstJournal.AddItem trantext
    Next i

    Screen.MousePointer = vbDefault
    Caption = "Journal view - " & lstJournal.ListCount & "
transactions"
End Sub
```

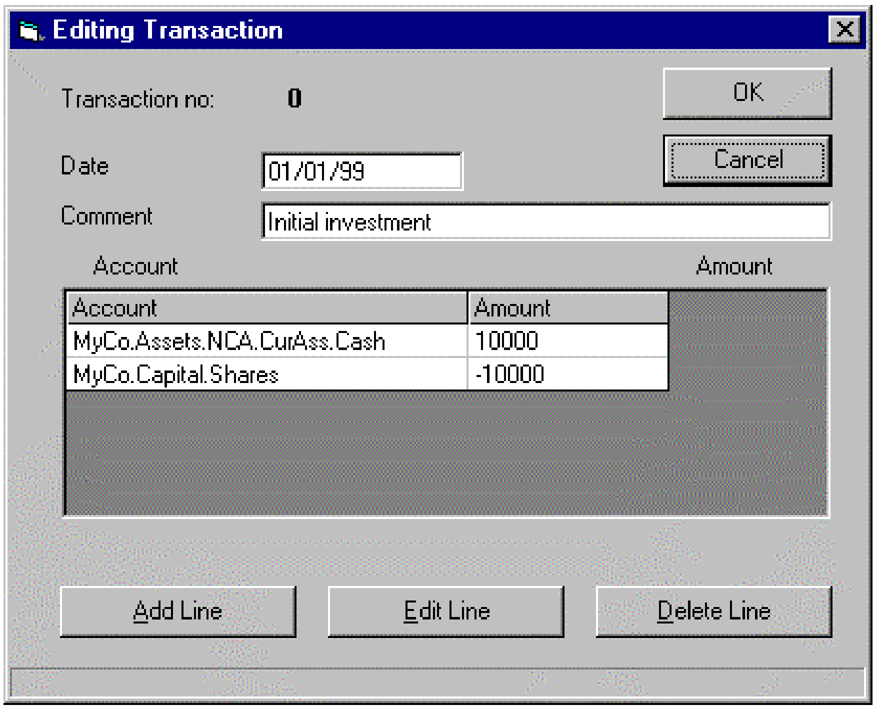
The Result


Doubletalk Browser - Visual Basic Edition
File
Edit
View
Tools
Window
Help
Journal view - 99 transactions

```
1/1/1999    Initial investment   10000.00
1/1/1999    Monthly Rent Accrual    1500.00
1/1/1999    Rent paid quarterly in advance    4500.00
1/1/1999    Phone bills    373.65
1/1/1999    General office costs    470.00
2/1/1999    Random Expense    9.00
3/1/1999    Random Expense    43.00
4/1/1999    Random Expense    73.00
```

5.14 Transaction Editing

Our target: add and edit transactions in the GUI:



Editing Transaction

Transaction no: 0

Date: 01/01/99

Comment: Initial investment

Account	Amount
MyCo.Assets.NCA.CurAss.Cash	10000
MyCo.Capital.Shares	-10000

So far, only BookSet is a COM object. How to deal with Transactions?

5.15 Design Pattern for Transactions

So far we only passed back and forth integers and strings. We need to pass in and out transactions for editing, and make a choice on their design pattern. Here’s how we’d edit one from VB.

Creatable from Registry

```
Dim newtran As Object
Set newtran = CreateObject("Doubletalk.Transaction")
newtran.setDateString "31/12/99"
newtran.setComment "Python on Windows Royalty Cheque"
newtran.addLine "MyCo.Assets.NCA.CurAss.Cash", 5000
newtran.addLastLine "MyCo.Capital.PL.Income.Writing"
BookServer.Add newtran
```

Created by BookSet

```
Dim newtran As Object
Set newtran = BookServer.CreateTransaction
newtran.setDateString "31/3/2000"
newtran.setComment "Even more royalties"
newtran.addLine "MyCo.Assets.NCA.CurAss.Cash", 5000
newtran.addLastLine "MyCo.Capital.PL.Income.Writing"
BookServer.Add newtran
```

The latter means less Python code, less in the registry, and less choice / more consistency for users!

5.16 Wrapping and Unwrapping sub-objects

If you pass a Python object as an argument across a COM boundary, need to ‘wrap’ and ‘unwrap’ it: VB gets, and gives, Idispatch wrappers around Python objects.

```
# more methods of COMBookSet class

    def createTransaction(self):
        comTran = COMTransaction()
        idTran = win32com.server.util.wrap(comTran)
        return idTran

    def add(self, idTran):
        comTran = win32com.server.util.unwrap(idTran)
        pyTran = comTran._tran
        self.__BookSet.add(pyTran)
```

- pyTran = the “pure python” class, nothing to do with COM
- comTran = our wrapper class with *_public_methods_* etc.
- idTran = the IDispatch object created and managed by Python COM framework – what VB gets and gives back.

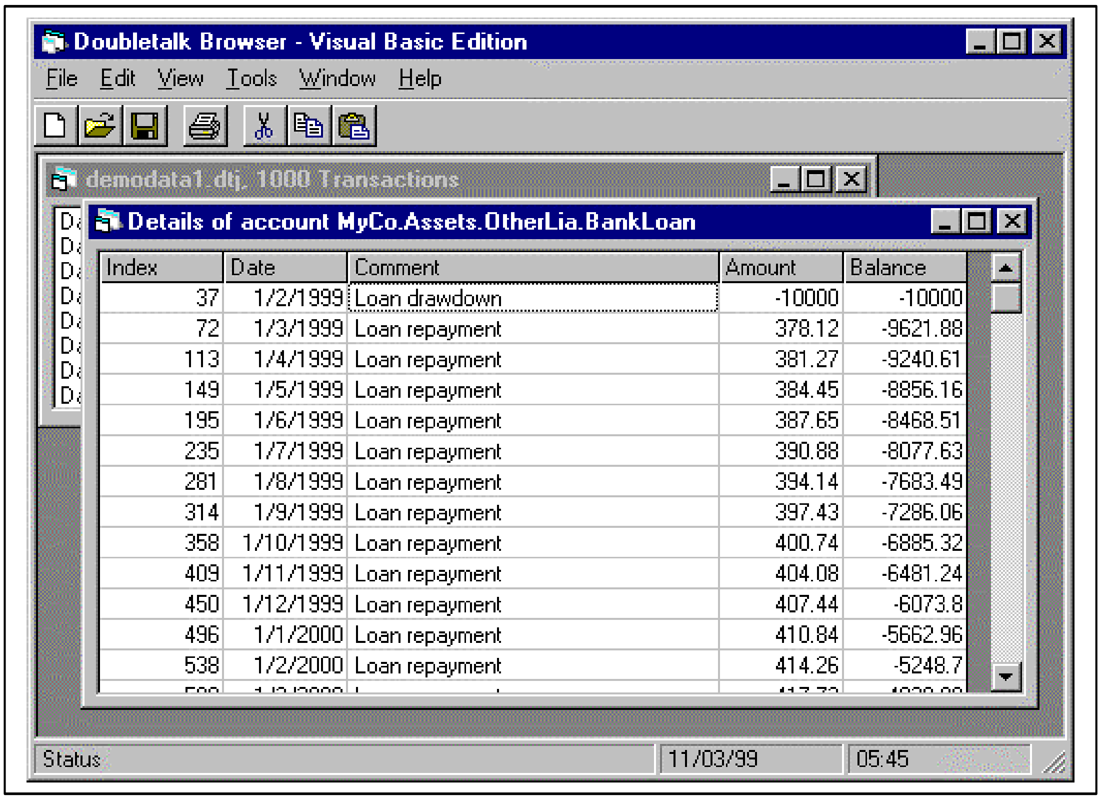
5.17 Passing Arrays

Move whole lists or tables at once – FAST!

Python lists/tuples <-> COM Safe Arrays

Makes possible tabular views.

```
Public Sub UpdateView()
    Dim table As Variant
    Dim rows As Integer, cols As Integer
    Dim row As Integer, col As Integer

    table = frmMain.BookServer.getAccountDetails(AccountName)

    rows = UBound(table, 1) -  LBound(table, 1) + 1
    cols = UBound(table, 2) - LBound(table, 2) + 1

    grdTable.rows = rows + 1  'leave room for titles
    For row = 0 To rows - 1
        For col = 0 To cols - 1
            grdTable.TextMatrix(row+1, col) = table(row,col)
        Next col
    Next row
End Sub
```


Doubletalk Browser - Visual Basic Edition
File
Edit
View
Tools
Window
Help
demodata1.dtj, 1000 Transactions
Details of account MyCo.Assets.OtherLia.BankLoan
Status
11/03/99
05:45

Index	Date	Comment	Amount	Balance
37	1/2/1999	Loan drawdown	-10000	-10000
72	1/3/1999	Loan repayment	378.12	-9621.88
113	1/4/1999	Loan repayment	381.27	-9240.61
149	1/5/1999	Loan repayment	384.45	-8856.16
195	1/6/1999	Loan repayment	387.65	-8468.51
235	1/7/1999	Loan repayment	390.88	-8077.63
281	1/8/1999	Loan repayment	394.14	-7683.49
314	1/9/1999	Loan repayment	397.43	-7286.06
358	1/10/1999	Loan repayment	400.74	-6885.32
409	1/11/1999	Loan repayment	404.08	-6481.24
450	1/12/1999	Loan repayment	407.44	-6073.8
496	1/1/2000	Loan repayment	410.84	-5662.96
538	1/2/2000	Loan repayment	414.26	-5248.7

5.18 Callbacks – Python controlling VB

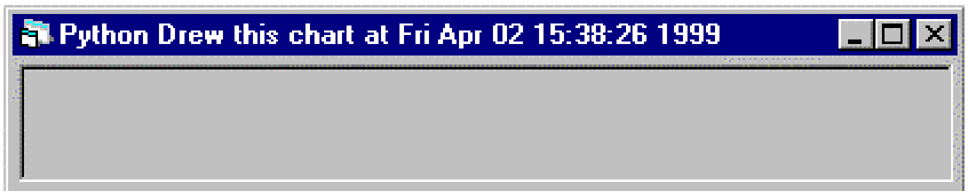
Make a chart view which asks Python to draw on it…

```
'Method of frmAccountChart
Public Sub UpdateView()
    'ask Python to scribble on me
    frmMain.BookServer.drawAccountChart Me
End Sub
```

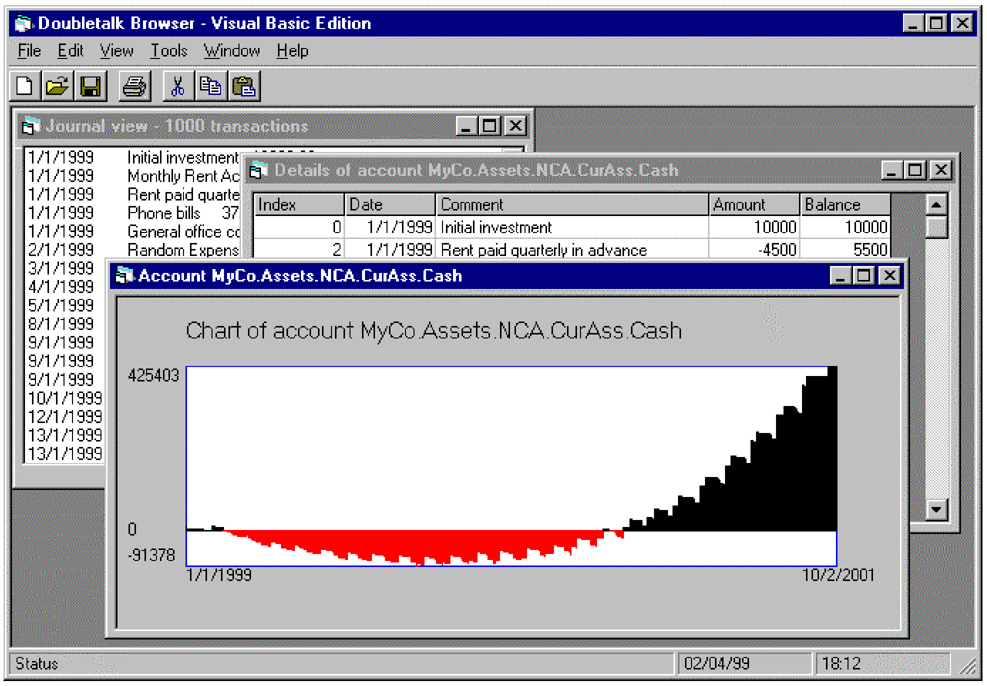
Dummy draw method in Python

```
def drawAccountChart(self, vbForm):
    # Make a Dispatch wrapper around the VB Form object so we
can call
    # any of its methods.
    idForm = win32com.client.dynamic.Dispatch(vbForm)
    # access a property of the VB form
    idForm.Caption = "Python Drew this chart at " + \
                     time.ctime(time.time())
```

Not a very impressive chart, yet!



Here's one we put in the oven earlier!


Doubletalk Browser - Visual Basic Edition
File Edit View Tools Window Help
Journal view - 1000 transactions
Details of account MyCo.Assets.NCA.CurAss.Cash
Account MyCo.Assets.NCA.CurAss.Cash
Chart of account MyCo.Assets.NCA.CurAss.Cash
425403
0
-91378
1/1/1999
10/2/2001
Status
02/04/99
18:12

```
1/1/1999    Initial investment
1/1/1999    Monthly Rent Ac
1/1/1999    Rent paid quarte
1/1/1999    Phone bills    37
1/1/1999    General office cc
2/1/1999    Random Expens
3/1/1999
4/1/1999
5/1/1999
8/1/1999
9/1/1999
9/1/1999
9/1/1999
10/1/1999
12/1/1999
13/1/1999
13/1/1999
```

Index	Date	Comment	Amount	Balance
0	1/1/1999	Initial investment	10000	10000
2	1/1/1999	Rent paid quarterly in advance	-4500	5500

5.19 Summary of argument type passing

Type	**The Rules**
Integers	No problem
Floating Point	No problem
Strings	Call str() on incoming unicode strings
One-dimensional arrays and lists	Python lists <-> Safe Arrays ("Variant Arrays" in Visual Basic)
Multi-Dimensional Arrays	As above, watch out for transposal
Odd-shaped lists: ['one','['two','three'],[[4,[5,6]]]]	At your own risk!
Dates	Python does not have a single native date type. Suggest conversion methods e.g. set/getDateString, set/getCOMDate etc. Pythoncom provides needed types.
Python Objects	Wrap and unwrap

5.20 Top tips for preserving sanity

- Design and test your Python engine in Python. Write thorough test scripts.
- Use a Python client to test your Python COM Server exhaustively.
- Use the Trace Collector Debugging Tool
- Use a Reload button
- VB can and will crash, so:
    - back up often
    - do a full compile, and test from the compiled app with VB minimized.

5.21 Conclusions

What have we achieved?

- An industry-standard, 100% Windows User Interface
  of the kind users expect
- An embedded and portable Python engine
- Techniques to add Python into an existing application
- Use Python where Python does best, VB where VB does best

Notes on other languages

Works with most of them. Tried and tested with Delphi, PowerBuilder.

Callbacks take extra work in Delphi; we got away with it here as "every VB object is a COM object" (supports IDispatch). But they are bad design anyway.

6 Adding a Macro Language

Goal: Make our application extensible.

Let users:

- Write scripts
- Handle events (adding, editing, deleting transactions)
- Create Validation Rules
- Create User-Defined Views
- Work at a command prompt

6.1 Dynamic Evaluation Background

In Python, the interpreter is always available!

`eval(`*`expression, [globals[, locals]]`*`)` evaluates a string,

`exec(`*`expression, [globals[, locals]]`*`)` executes one.

```
>>> exec("print 'this expression was compiled on the fly' ")
this expression was compiled on the fly
>>> exec("x = 3.14")
>>> eval("x + 1")
4.14
```

Namespaces can be accessed too

```
>>> # where is the 'x' kept?
>>> for item in globals().items():
...     print item
...
('__doc__', None)
('pywin', <module 'pywin'>)
('x', 3.14)
('__name__', '__main__')
('__builtins__', <module '__builtin__'>)
>>>
```

Variables at the command prompt go in a namespace accessible through `globals()`

A namespace is just a dictionary.

6.2 …so make our own namespace and expose it

```
# COMBookSet methods again
   def __init__(self):
        self.__BookSet = doubletalk.bookset.BookSet()

        # create a custom namespace for the user to work with
        # with one variable name already defined
        self.userNameSpace = {'TheBookServer', self.__BookSet}

    def interpretString(self, exp):
        """Makes it easier to build consoles.
        """
        if type(exp) not in [type(''), UnicodeType]:
            raise Exception(desc="Must be a string", \
                        scode=winerror.DISP_E_TYPEMISMATCH)
        try:
            # first, we assume it is an expression
            result_object = eval(str(exp), self.userNameSpace)
            if result_object == None:
                return ''
            else:
                return str(result_object)
        except:
            #failing that, try to execute it
            exec str(exp) in self.userNameSpace
            return ''
```

6.3 *Grabbing Python's output*

Goal : see console output in a VB window.

Python lets you redirect its own standard output to any object with a write() method. Example:

```
>>> import sys
>>> mylogfile = open('c:\\temp\\mylog.txt', 'w')
>>> sys.stdout = mylogfile
>>> print 'hello'
>>> # no output on console, it's in the file!
```

Our plan

- give the COMBookSet a write() method
- ways to start trapping output, end trapping it, and retrieve any available output.
- Keep it in a list of strings

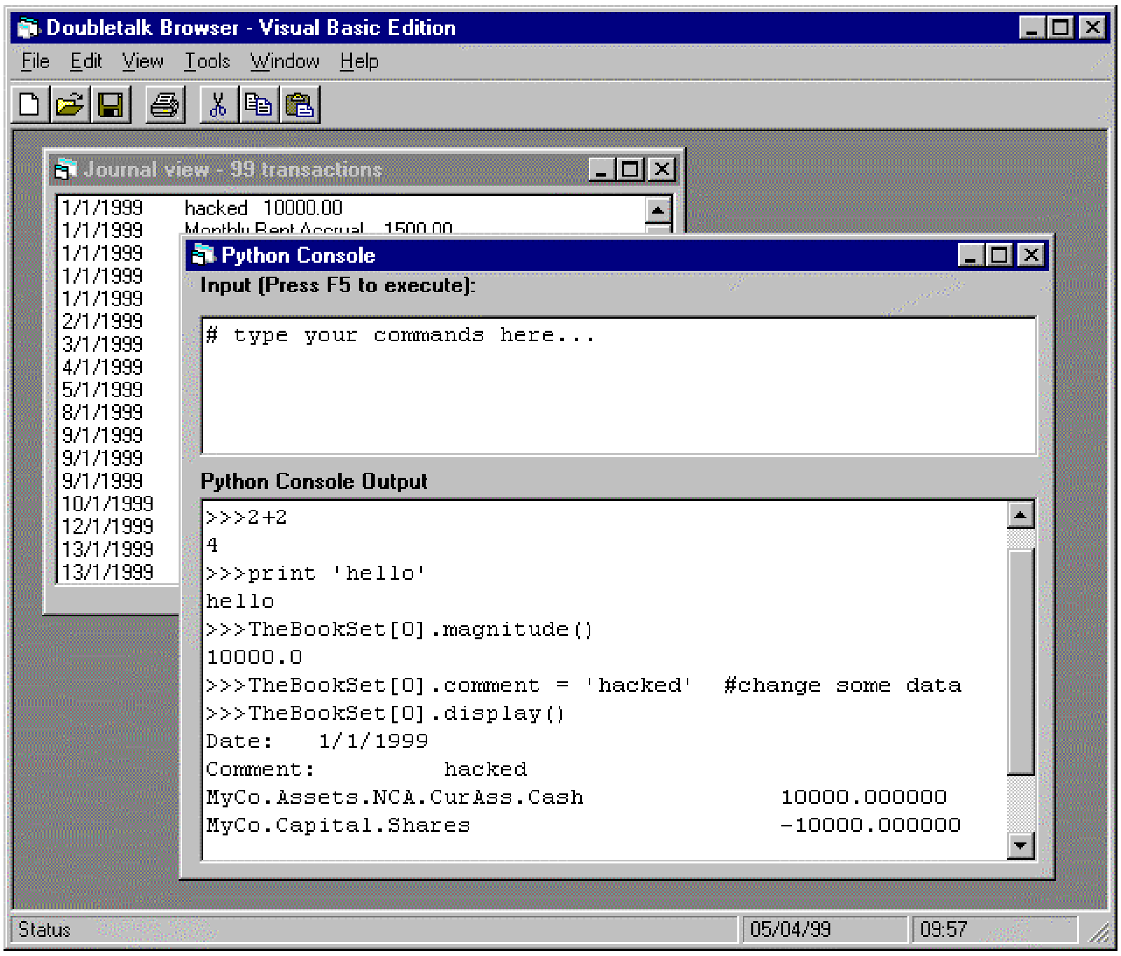
Implementation

```
   def beginTrappingOutput(self):   # exposed as public method
        self.outputBuffer = []
        self.old_output = sys.stdout
        sys.stdout = self

    def write(self, expr):          # private
        """ this is an internal utility used to trap the output.
        add it to a list of strings - this is more efficient
        than adding to a possibly very long string."""
        self.outputBuffer.append(str(expr))

    def getStandardOutput(self): # exposed as public method
        "Hand over output so far, and empty the buffer"
        text = string.join(self.outputBuffer, '')
        self.outputBuffer = []
        return text

    def endTrappingOutput(self): # exposed as public method
        sys.stdout = self.old_output
        # return any more output
        return self.getStandardOutput()
```

Warning: Conflict with Trace Collector

There is a Python utility called the Trace Collector debugging tool which can also be set up to capture the output, so you can use 'print' debugging with COM servers. This also tries to capture the standard output, and will win if running!

6.4 Now we can build a console for our app…


Doubletalk Browser - Visual Basic Edition
File Edit View Tools Window Help
Journal view - 99 transactions
Python Console
Input (Press F5 to execute):
Python Console Output
Status
05/04/99
09:57

```
1/1/1999      hacked  10000.00
1/1/1999      Monthly Rent Accrual    1500.00
1/1/1999
1/1/1999
1/1/1999
2/1/1999
3/1/1999
4/1/1999
5/1/1999
8/1/1999
9/1/1999
9/1/1999
9/1/1999
10/1/1999
12/1/1999
13/1/1999
13/1/1999
```

```
# type your commands here...
```

```
>>>2+2
4
>>>print 'hello'
hello
>>>TheBookSet[0].magnitude()
10000.0
>>>TheBookSet[0].comment = 'hacked'  #change some data
>>>TheBookSet[0].display()
Date:    1/1/1999
Comment:          hacked
MyCo.Assets.NCA.CurAss.Cash              10000.000000
MyCo.Capital.Shares                      -10000.000000
```

6.5 More macro-related features

Executing Scripts:

- Menu option to run a script
- `exec(`*`expression, [globals[, locals]]`*`)` does the work in one line.

Importing Scripts

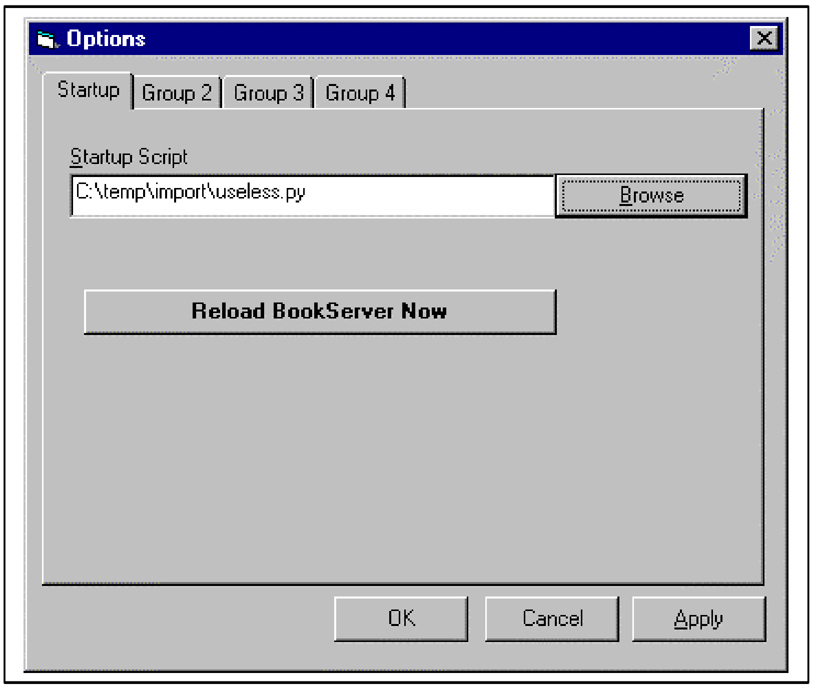
- Python imp module exposes import mechanism; need to grab the module object and add it to our namespace.
- Add an importFile() method to COMBookSet

Startup Script in Options

- Highly useful for users

Reload option

- Useful for users, and for us! Closes and reloads BookServer.



Startup Script: C:\temp\import\useless.py

Define user and system code directories

- Get added to Python path at startup.

6.6 *Making the app extensible*

Changing the delegate class

Feasible, as COMBookSet creates BookSet at startup. An options dialog could specify the module and class. But puts a big onus on the user – must implement every method.

Delegation Framework: Views and Validators

- a *Validator* is an object which a BookSet notifies before changing data, asking for permission to proceed.
- a *View* is an object which the BookSet notifies after changes have been made. It also has a method to return a two-dimensional array of data on demand, which could contain whatever the user wished.
- class UserBookSet maintains a list of Views and Validators, and notifies them of changes

Users now only need to write a new view, not a whole new Bookset.

What do Views return?

Our convention: a 2-d array of data to go in a grid. Your app may differ.

Base class for User-Defined Views

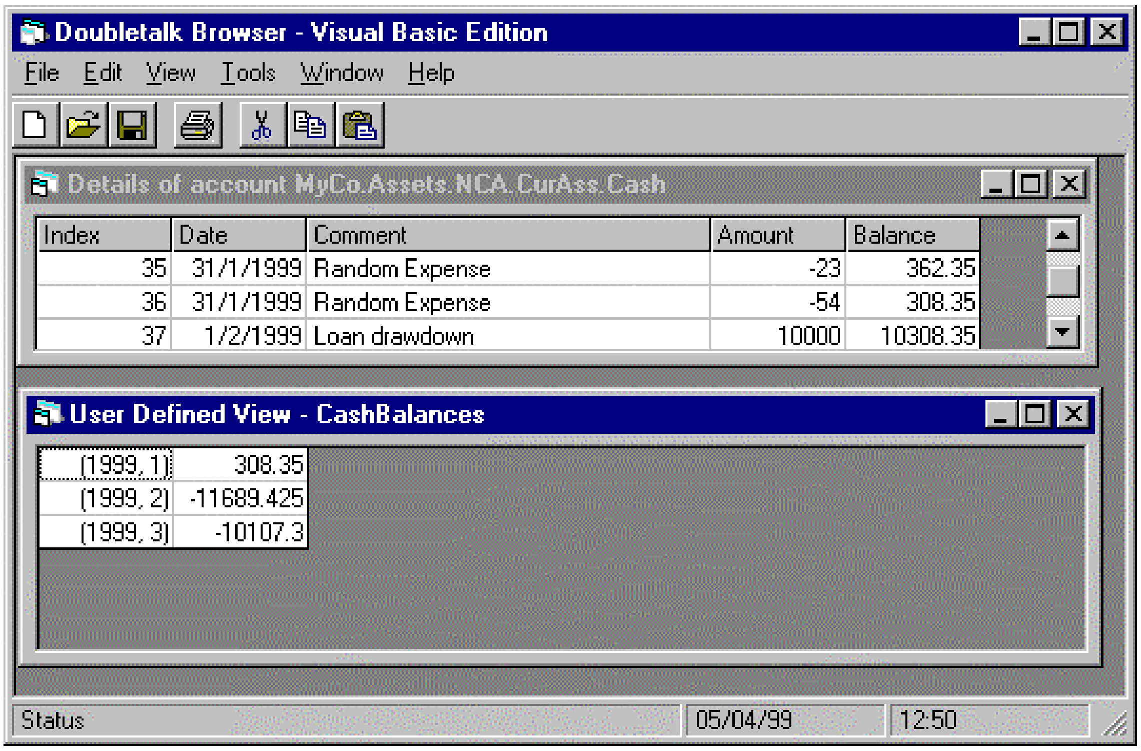
```
class View:
    """This delegate is informed of all changes after they
occur,
    and returns a 2d array of data when asked."""
    def setBookSet(self, aBookSet):
        self.BookSet = aBookSet
        self.recalc()
    def getDescription(self):
        return 'abstract base class for Views'
    # hooks for notification after the event
    def didAdd(self, aTransaction):
        pass
    def didEdit(self, index, newTransaction):
        pass
    def didRemove(self, index):
        pass
    def didRenameAccount(self, oldname, newname):
        pass
    def didChangeDrastically(self):
        #can be used to notify of major changes such as
file/open
        self.recalc()
    def recalc(self):
        #override this to work out the data
        pass
    def getData(self):
        return [()]  # simple 2-d array for display
```

What can you do with Views and Validators?

- Add an audit trail (logs all add/edit/delete/rename operations)
- Security
  - Only the Finance Director can modify last quarter’s data.
  - Read-only for the relevant people
  - Editing “time window” to stop them entering 1989 data by mistake!
- New back ends – fetch from and store to a relational database on demand
- Update other programs when certain changes happen
- Cache to improve performance – a View to hold pre-computed month-end balances for all accounts and months.

Front End

Maintain a list of modules and class names. Short chunk of Python passed to interpretString() to instantiate them.


Doubletalk Browser - Visual Basic Edition
File Edit View Tools Window Help
Details of account MyCo.Assets.NCA.CurAss.Cash

Index	Date	Comment	Amount	Balance
35	31/1/1999	Random Expense	-23	362.35
36	31/1/1999	Random Expense	-54	308.35
37	1/2/1999	Loan drawdown	10000	10308.35

User Defined View - CashBalances

(1999, 1)	308.35
(1999, 2)	-11689.425
(1999, 3)	-10107.3

6.7 Macro Languages Conclusion

We've built a powerful cross-platform engine in a pure Windows GUI. Now we've just added a macro language so users can customize the system for their own needs.

This goes beyond normal Windows development and into an area which is one of Python's greatest strengths – extensibility.

This kind of extensible app would be prohibitively expensive and difficult without Python. Macro languages are normally only available to Microsoft, Visio et al. With Python it is straightforward.

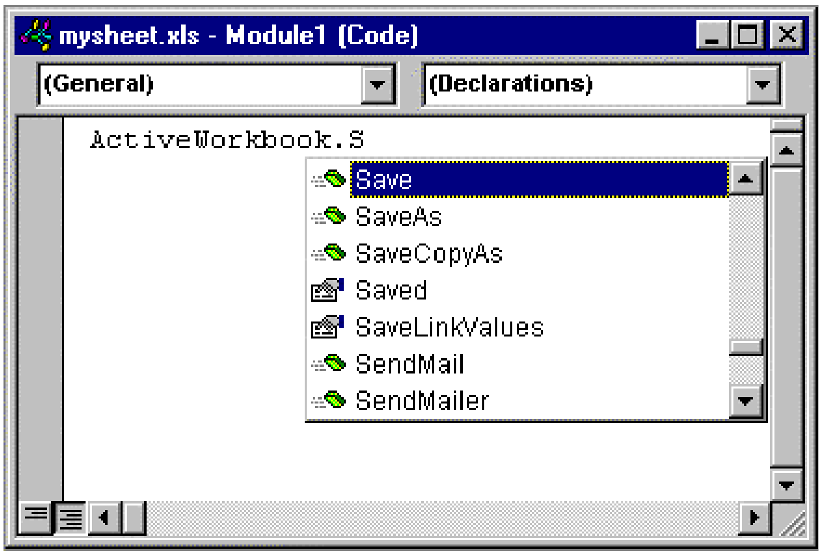
7 Client Side COM and Excel

Why Excel?

- Very widely used in financial and scientific circles.
- Key source and destination for numeric data.

Learning the Excel Object Model

- There's not much to be learned about client side COM. But there's a lot to be learned about the object model of each target application.
- Excel has many objects; the Range alone has 84 properties and 72 methods
- Don't learn it from Python, do it from VBA!



```
ActiveWorkbook.S
Save
SaveAs
SaveCopyAs
Saved
SaveLinkValues
SendMail
SendMailer
```

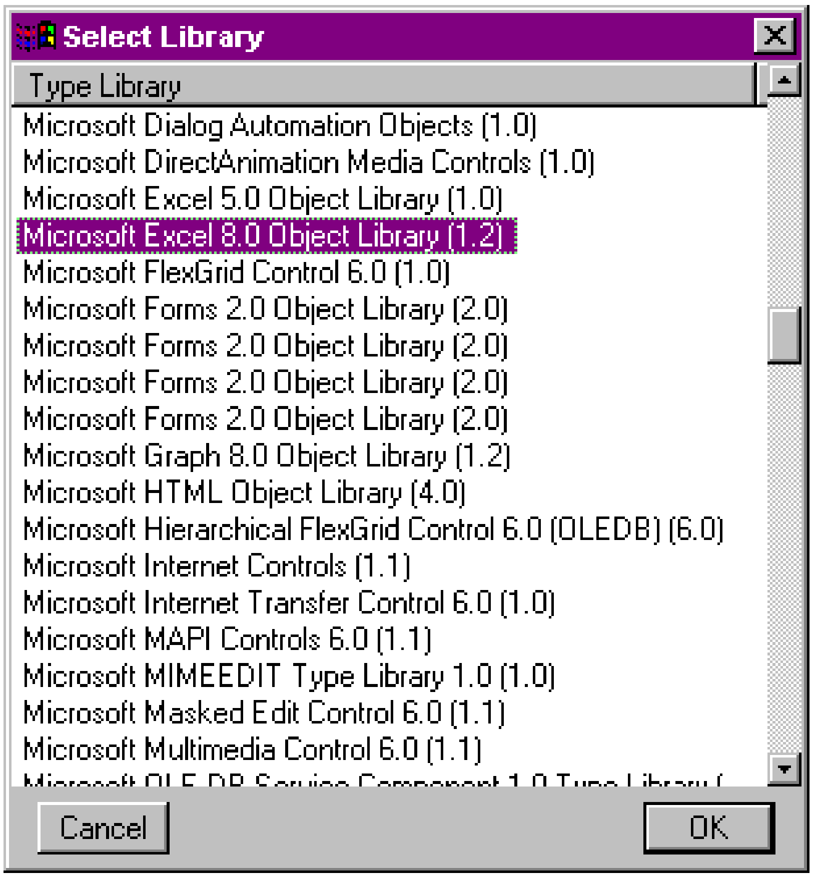
7.1 Connecting to the client

Starting up Excel

```
>>> from win32com.client import Dispatch
>>> xlApp = Dispatch("Excel.Application")
>>> xlApp.Visible = 1
>>> xlApp.Workbooks.Add()
<win32com.gen_py.Microsoft Excel 8.0 Object Library.Workbook>
>>>
```

If you can, run MakePy first…

- MakePy runs a bit faster
- MakePy provides type info
- MakePy provides all the constants!



Microsoft Dialog Automation Objects (1.0)
Microsoft DirectAnimation Media Controls (1.0)
Microsoft Excel 5.0 Object Library (1.0)
Microsoft Excel 8.0 Object Library (1.2)
Microsoft FlexGrid Control 6.0 (1.0)
Microsoft Forms 2.0 Object Library (2.0)
Microsoft Forms 2.0 Object Library (2.0)
Microsoft Forms 2.0 Object Library (2.0)
Microsoft Forms 2.0 Object Library (2.0)
Microsoft Graph 8.0 Object Library (1.2)
Microsoft HTML Object Library (4.0)
Microsoft Hierarchical FlexGrid Control 6.0 (OLEDB) (6.0)
Microsoft Internet Controls (1.1)
Microsoft Internet Transfer Control 6.0 (1.0)
Microsoft MAPI Controls 6.0 (1.1)
Microsoft MIMEEDIT Type Library 1.0 (1.0)
Microsoft Masked Edit Control 6.0 (1.1)
Microsoft Multimedia Control 6.0 (1.1)
Microsoft OLE DB Service Component 1.0 Type Library (

7.2 Object model basics and warnings

Navigating through collections: How to modify Cell at top left

There's more than one way to do it!

```
xlApp.ActiveSheet.Cells(1,1).Value = 'Python Rules!'
>>> xlApp.ActiveWorkbook.ActiveSheet.Cells(1,1).Value = 'Python
Rules!'
>>> xlApp.Workbooks("Book1").Sheets("Sheet1").Cells(1,1).Value =
"Python Rules!"
>>> xlApp.Workbooks(1).Sheets(1).Cells(1,1).Value = "Python
Rules!"
>>> xlApp.Workbooks(1).Sheets(1).Cells(1,1).Value = "Python
Rules!"
>>>
>>> xlBook = xlApp.Workbooks(1)
>>> xlSheet = xlApp.Sheets(1)
>>> xlSheet.Cells(1,1).Value = "Python Rules!"
>>>
```

Recommendation: Get hold of the sheet in a variable, and use that.

Round and Square Brackets,

```
>>> xlBook.Sheets(1)
<win32com.gen_py.Microsoft Excel 8.0 Object Library._Worksheet>
>>> xlBook.Sheets[1]
<win32com.gen_py.Microsoft Excel 8.0 Object Library._Worksheet>
>>> xlBook.Sheets["Sheet1"]
(some error details omitted)
TypeError: Only integer indexes are supported for enumerators
>>>
```

String arguments only work with round brackets.

One-and Zero-based collections

```
>>> xlBook.Sheets(1).Name
'Sheet1'
>>> xlBook.Sheets[1].Name
'Sheet2'
>>>
```

Square brackets always count from 0.

Round brackets count the way the author intended. Most office apps count from 1

Recommendation: use round brackets, and read your object model's documentation to find out the base. Most office apps count from 1.

Keyword Arguments

Excel likes these a lot:

```
expression.SaveAs(Filename, FileFormat, Password,
                  WriteResPassword, ReadOnlyRecommended,
                  CreateBackup, AddToMru, TextCodePage,
                  TextVisualLayout)
```

Supply what you need:

```
>>> xlBook.SaveAs(Filename='C:\\temp\\mysheet.xls')
>>>
```

Watch the capitalisation! Microsoft are not always 100% consistent.

7.3 *Passing data in and out*

Use the *Value* property:

```
>>> xlSheet.Cells(1,1).Value = 'What shall be the number of thy
counting?'
>>> xlSheet.Cells(2,1).Value = 3
>>> xlSheet.Cells(1,1).Value
'What shall be the number of thy counting?'
>>> xlSheet.Cells(2,1).Value
3.0
>>>
```

Converting times and dates – MS apps and Python have different standard

```
>>> import time
>>> now = time.time()
>>> now    # how many seconds since 1970?
923611182.35
>>> import pythoncom
>>> time_object = pythoncom.MakeTime(now)
>>> int(time_object)    # can get the value back...
923611182
>>> xlSheet.Cells(3,1).Value = time_object # ...or send it
>>> xlSheet.Cells(3,1).Value
<time object at 188c080>
```

Modify formulae with the Formula (!) property

```
>>> xlSheet.Cells(4,1).Formula = '=A2*2'
>>> xlSheet.Cells(4,1).Value
6.0
>>> xlSheet.Cells(4,1).Formula
'=A2*2'
>>>
```

Empty Cells accept/return Python Value “None”

```
>>> xlSheet.Cells(1,1).Value = None  # clear a cell
>>> xlSheet.Cells(1,1).Value  # returns None
>>>
```

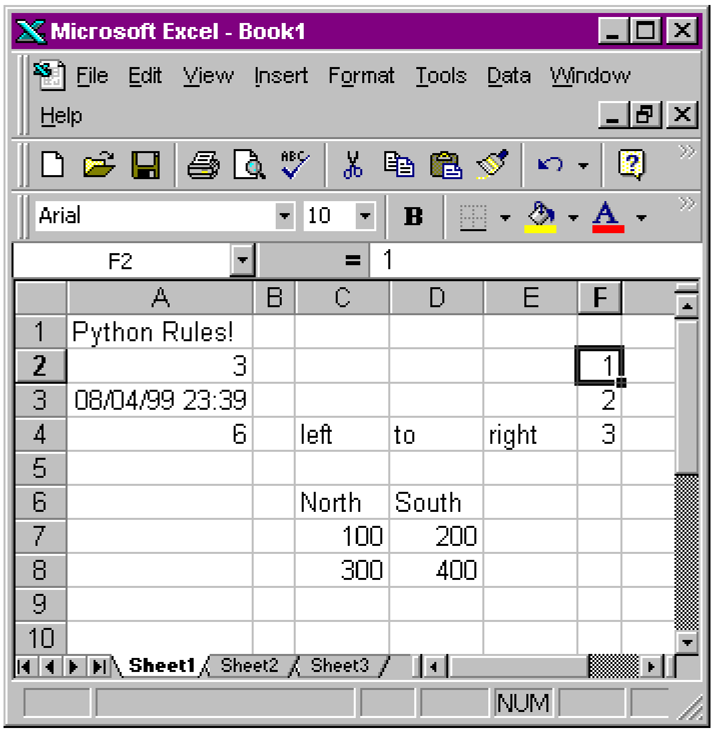
Use Range objects to work with more than one cell

```
>>> myRange1 = xlSheet.Cells(4,1)        # one-cell range
>>> myRange2 = xlSheet.Range("B5:C10")   # excel notation
>>> myRange3 = xlSheet.Range(xlSheet.Cells(2,2),
xlSheet.Cells(3,8))
>>>
```

Ranges have about 80 methods for formatting and modifying their data.

7.4 *You're going to love working with Arrays*

In our VB app, we fetched a 2d array, then wrote code to loop over it putting it into the GUI. Excel works directly with COM arrays.



	A	B	C	D	E	F
1	Python Rules!					
2	3					1
3	08/04/99 23:39					2
4	6		left	to	right	3
5						
6			North	South		
7			100	200		
8			300	400		
9						
10						

Horizontal range:

```
>>> xlSheet.Range('C3:E3').Value
((L'left', L'to', L'right'),)
>>>
```

Matrix with several rows and columns

```
>>> xlSheet.Range('C5:D7').Value
((L'North', L'South'), (100.0, 200.0), (300.0, 400.0))
>>>
```

Note: this is just how we'd represent it in Python!

Vertical Range

```
>>> xlSheet.Range('F2:F4').Value
((1.0,), (2.0,), (3.0,))
>>>
```

Passing arrays is FAST – 60 columns, 100 rows in an eyeblink!

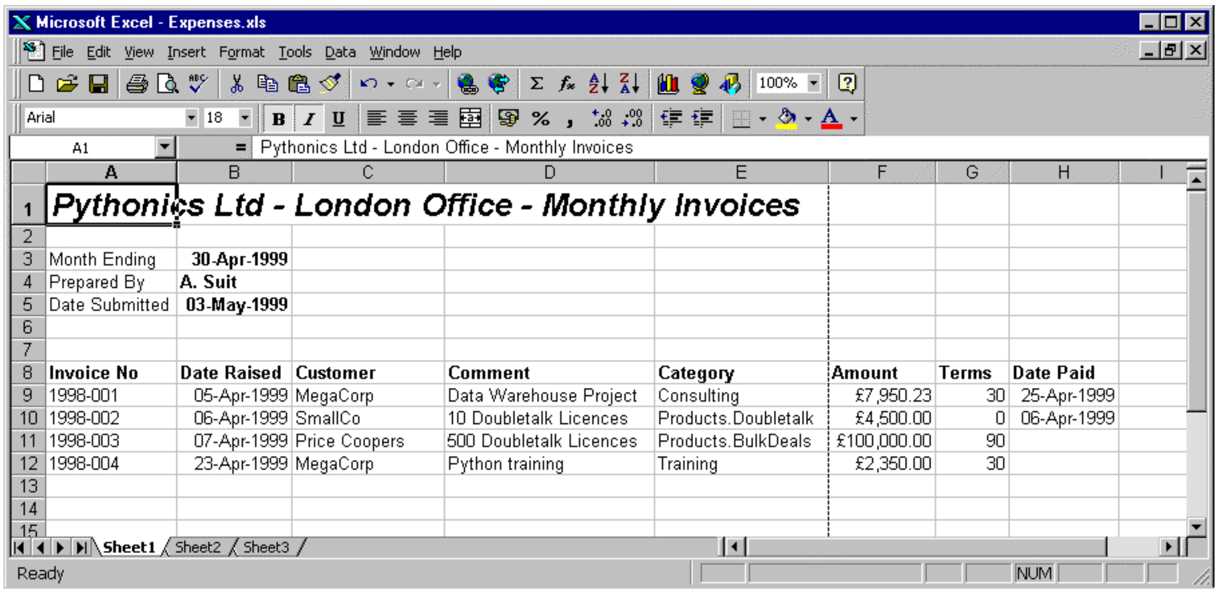
7.5 Make your own wrapper

For example (showing headers only):

```
class easyExcel:
    """A utility to make it easier to get at Excel.  Remembering
    to save the data is your problem, as is  error handling.
    Operates on one workbook at a time."""
      def __init__(self, filename=None):
    def save(self, newfilename=None):
    def close(self):
    def getCell(self, sheet, row, col):
    def setCell(self, sheet, row, col, value):
    def getRange(self, sheet, row1, col1, row2, col2):
    def setRange(self, sheet, leftCol, topRow, data):
    def getContiguousRange(self, sheet, startrow, startcol):
# and rather importantly, for cleaning up,
    def fixStringsAndDates(self, aMatrix):
```

Let your wrapper grow with time. Share it with me!

7.6 Exercise – Financial Data Import


Microsoft Excel - Expenses.xls
File Edit View Insert Format Tools Data Window Help
100%
Arial
18
A1
Sheet1
Sheet2
Sheet3
Ready
NUM

= Pythonics Ltd - London Office - Monthly Invoices

	A	B	C	D	E	F	G	H
1	Pythonics Ltd - London Office - Monthly Invoices							
2								
3	Month Ending	30-Apr-1999						
4	Prepared By	A. Suit						
5	Date Submitted	03-May-1999						
6								
7								
8	Invoice No	Date Raised	Customer	Comment	Category	Amount	Terms	Date Paid
9	1998-001	05-Apr-1999	MegaCorp	Data Warehouse Project	Consulting	£7,950.23	30	25-Apr-1999
10	1998-002	06-Apr-1999	SmallCo	10 Doubletalk Licences	Products.Doubletalk	£4,500.00	0	06-Apr-1999
11	1998-003	07-Apr-1999	Price Coopers	500 Doubletalk Licences	Products.BulkDeals	£100,000.00	90	
12	1998-004	23-Apr-1999	MegaCorp	Python training	Training	£2,350.00	30	
13								
14								
15								

How would you turn these into transactions in a BookSet?

8 Automating Word

When to use Word

- when a Word document would be intrinsically useful (e.g. correspondence database)
- obviously, to acquire Word data – thankfully a rare occurrence!

When not to use Word

- when you just want a dead tree edition
- when you don't want the output tampered with
- target readers on other platforms

Shameless Plug - Use PDFgen instead!

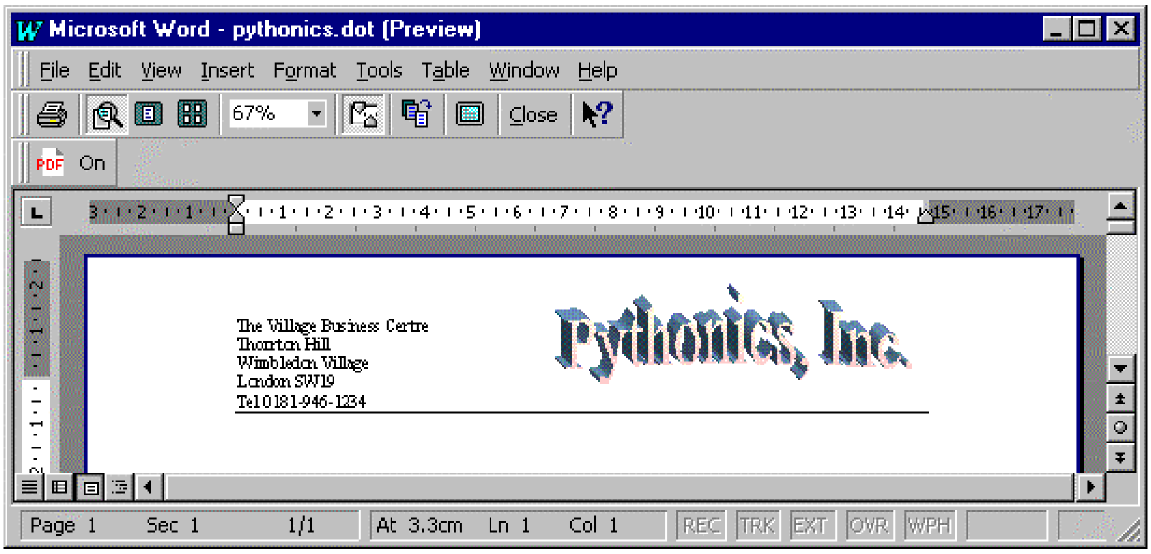
Our target: Management accounts

- Text with given paragraph styles
- Charts and other Images
- Tables of nicely formatted data

8.1 Hello, Word

```
from win32com.client import Dispatch
MYDIR = 'c:\\data\\project\\oreilly\\examples\\ch12_print'

def simple():
    myWord = Dispatch('Word.Application')
    myWord.Visible = 1  # comment out for production

    myDoc = myWord.Documents.Add()
    myRange = myDoc.Range(0,0)
    myRange.InsertBefore('Hello from Python!')

    # uncomment these for a full script
    #myDoc.SaveAs(MYDIR + '\\python01.doc')
    #myDoc.PrintOut()
    #myDoc.Close()
```

8.2 Why work harder? Use a template…


Microsoft Word - pythonics.dot (Preview)
File Edit View Insert Format Tools Table Window Help
67%
Close
On
Page 1 Sec 1 1/1 At 3.3cm Ln 1 Col 1 REC TRK EXT OVR WPH

The Village Business Centre
Thornton Hill
Wimbledon Village
London SW19
Tel 0181-946-1234

Pythonics, Inc.

Construct it supplying the template as an optional argument.

```
>>>myDoc = myWord.Documents.Add(template=MYDIR +
'\\pythonics.dot')
```

Design all the graphic elements in Word. Bookmark your insertion points. Only insert the text you need to.

8.3 Word Wrapping

As before, we will wrap Word up in a class which makes it easy to generate documents.

Creating, printing and saving:

```
class WordWrap:
    """Wrapper aroud Word 8 documents to make them easy to
build.
    Has variables for the Applications, Document and Selection;
    most methods add things at the end of the document"""
    def __init__(self, templatefile=None):
        self.wordApp = Dispatch('Word.Application')
        if templatefile == None:
            self.wordDoc = self.wordApp.Documents.Add()
        else:
            self.wordDoc = self.wordApp.Documents.Add(
Template=templatefile)
   #set up the selection
        self.wordDoc.Range(0,0).Select()
        self.wordSel = self.wordApp.Selection
        #fetch the styles in the document - see below
        self.getStyleDictionary()

    def show(self):
        # convenience when developing
        self.wordApp.Visible = 1
```

Save and Print

```
    def saveAs(self, filename):
        self.wordDoc.SaveAs(filename)

    def printout(self):
        self.wordDoc.PrintOut()
```

Adding text

Move the selection to the end, and insert into selection

```
def selectEnd(self):
    # ensures insertion point is at the end of the document
    self.wordSel.Collapse(0)
    # 0 is the constant wdCollapseEnd; don't want to depend
    # on makepy support.

def addText(self, text):
    self.wordSel.InsertAfter(text)
    self.selectEnd()
```

Insert a paragraph in a named style

Fast, versatile, and lets graphic designers set the styles in Word.

```
>>> from win32com.client import constants
>>> mySelection.Style = constants.wdStyleHeading1
>>>
```

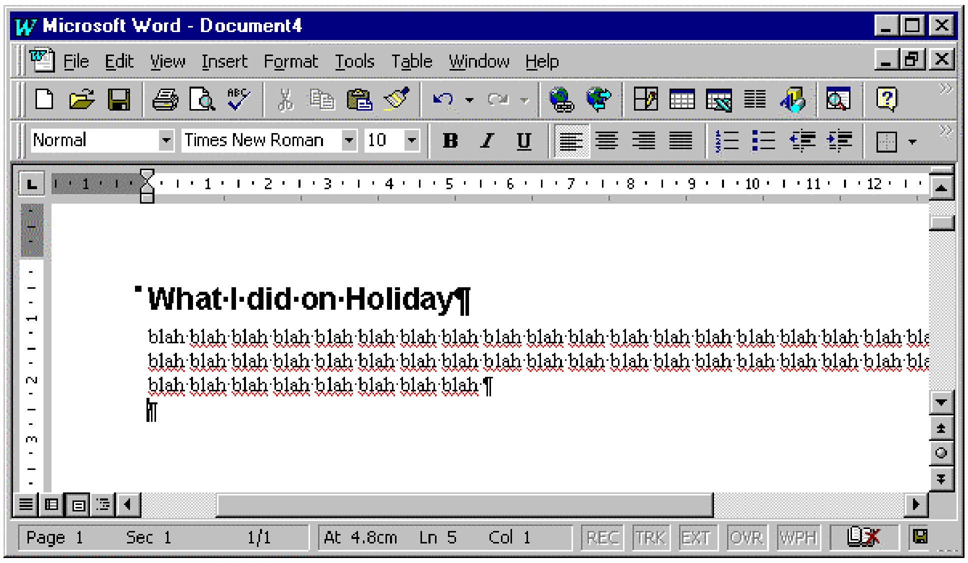
However, (a) this is limited to the built-in styles, and (b) won't work without MakePy.

Our solution: get a style list back at run time, and use it to add them by name:

```
      def getStyleList(self):
        # returns a dictionary of the styles in a document
        self.styles = []
        stylecount = self.wordDoc.Styles.Count
        for i in range(1, stylecount + 1):
            styleObject = self.wordDoc.Styles(i)
            self.styles.append(styleObject.NameLocal)

      def addStyledPara(self, text, stylename):
        if text[-1] <> '\n':
            text = text + '\n'
        self.wordSel.InsertAfter(text)
        self.wordSel.Style = stylename
        self.selectEnd()
```

Time for a test:

```
>>> import easyword
>>> w = easyword.WordWrap()
>>> w.show()
>>> w.addStyledPara('What I did on Holiday', 'Heading 1')
>>> w.addStyledPara('blah ' * 50, 'Normal')
>>>
```


Microsoft Word - Document4
File Edit View Insert Format Tools Table Window Help
Normal
Times New Roman
10

What·I·did·on·Holiday¶

blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·bla
blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·blah·bla
blah·blah·blah·blah·blah·blah·blah·blah·¶
¶

Page 1 Sec 1 1/1 At 4.8cm Ln 5 Col 1 REC TRK EXT OVR WPH

Styles and templates together give a very easy way to build up a sophisticated document.

Adding Tables

Let's be lazy and use AutoFormat – saves a lot of code: Fortunately, most arguments are optional!

Table.AutoFormat(*Format, ApplyBorders, ApplyShading, ApplyFont, ApplyColor, ApplyHeadingRows, ApplyLastRow, ApplyFirstColumn, ApplyLastColumn, AutoFit*)

Create a block of tab-delimited text, then ask Word to turn it into a table:

```
def addTable(self, table, styleid=None):
    # Takes a 'list of lists' of data.
    # first we format the text.  You might want to preformat
    # numbers with the right decimal places etc. first.
    textlines = []
    for row in table:
        textrow = map(str, row)    #convert to strings
        textline = string.join(textrow, '\t')
        textlines.append(textline)
    text = string.join(textlines, '\n')

    # add the text, which remains selected
    self.wordSel.InsertAfter(text)

    #convert to a table
    wordTable = self.wordSel.ConvertToTable(Separator='\t')
    #and format
    if styleid:
        wordTable.AutoFormat(Format=styleid)
```

Adding Charts and Images

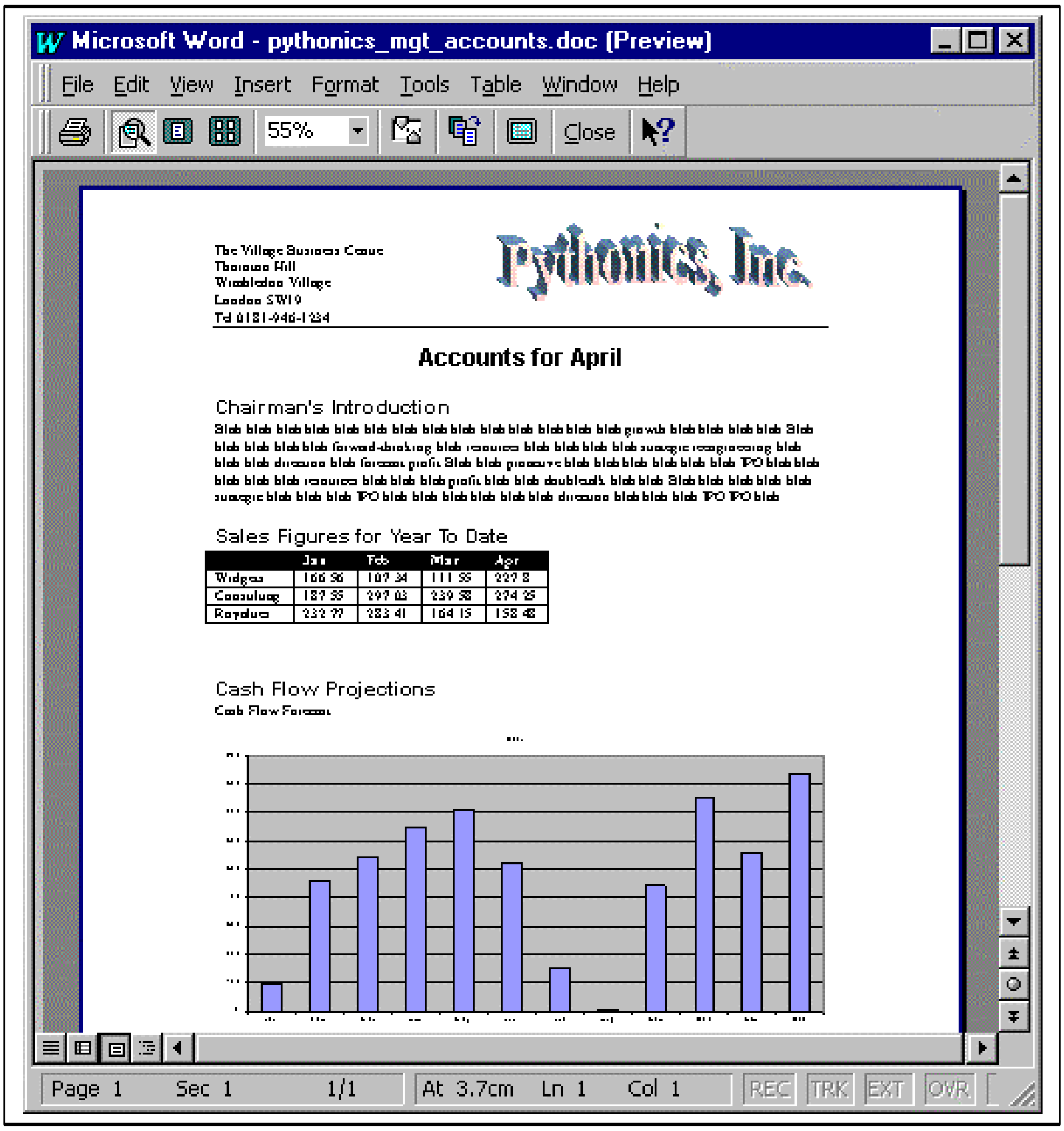
Let’s assume we’ve set up and saved an Excel sheet with the right chart in it already.

```
    def  addInlineExcelChart(self, filename, caption='',
                     height=216, width=432):
        # adds a chart inline within the text, caption below.
        # add an InlineShape to the InlineShapes collection
        #- could appear anywhere
        shape = self.wordDoc.InlineShapes.AddOLEObject(
            ClassType='Excel.Chart',
            FileName=filename
            )
        # set height and width in points
        shape.Height = height
        shape.Width = width

        # put it where we want
        shape.Range.Cut()
        self.wordSel.InsertAfter('chart will replace this')
        self.wordSel.Range.Paste()  # goes in selection
        self.addStyledPara(caption, 'Normal')
```

All the more obvious approaches failed, but it works!

Tip: Avoid shapes on the drawing layer – they are really hard to control from a programming language.

TaDa!


Microsoft Word - pythonics_mgt_accounts.doc (Preview)
File Edit View Insert Format Tools Table Window Help
55%
Close
Page 1 Sec 1 1/1 At 3.7cm Ln 1 Col 1 REC TRK EXT OVR

The Village Business Centre
Thornton Hill
Wimbledon Village
London SW19
Tel 0181-946-1234

Pythonics, Inc.

Accounts for April

Chairman's Introduction

Blah blah blah blah blah blah blah blah blah blah blah blah blah blah growth blah blah blah blah Blah blah blah blah blah forward-thinking blah resources blah blah blah blah strategic reengineering blah blah blah discussion blah forecast profit. Blah blah proactive blah blah blah blah blah blah TPO blah blah blah blah blah resources blah blah blah profit blah blah downloads blah blah Blah blah blah blah blah strategic blah blah blah TPO blah blah blah blah blah blah discussion blah blah blah TPO TPO blah

Sales Figures for Year To Date

	Jan	Feb	Mar	Apr
Widgets	166 96	107 34	111 55	227 8
Consulting	187 55	297 03	239 58	274 25
Royalties	232 77	283 41	164 15	158 48

Cash Flow Projections

Cash Flow Forecast

Inserting HTML: A Really Powerful Shortcut

Word can import and export HTML. The following line inserts an entire file into the current document.

```
>>> wordSelection.InsertFile(MYDIR + '\\tutorial.html')
>>>
```

You don’t even need a full HTML document - just a fragment saved with the extension HTML.

8.4 The Last Word

Word is hard to work with, so.

- Prototype your code in VBA to get it right
- Expect Word to fall over if you feed it bad arguments in development
- Consider ‘pull technology’ – Word template with VBA macro connects to Python COM server to get the data.

9 Distributing our Application with DCOM

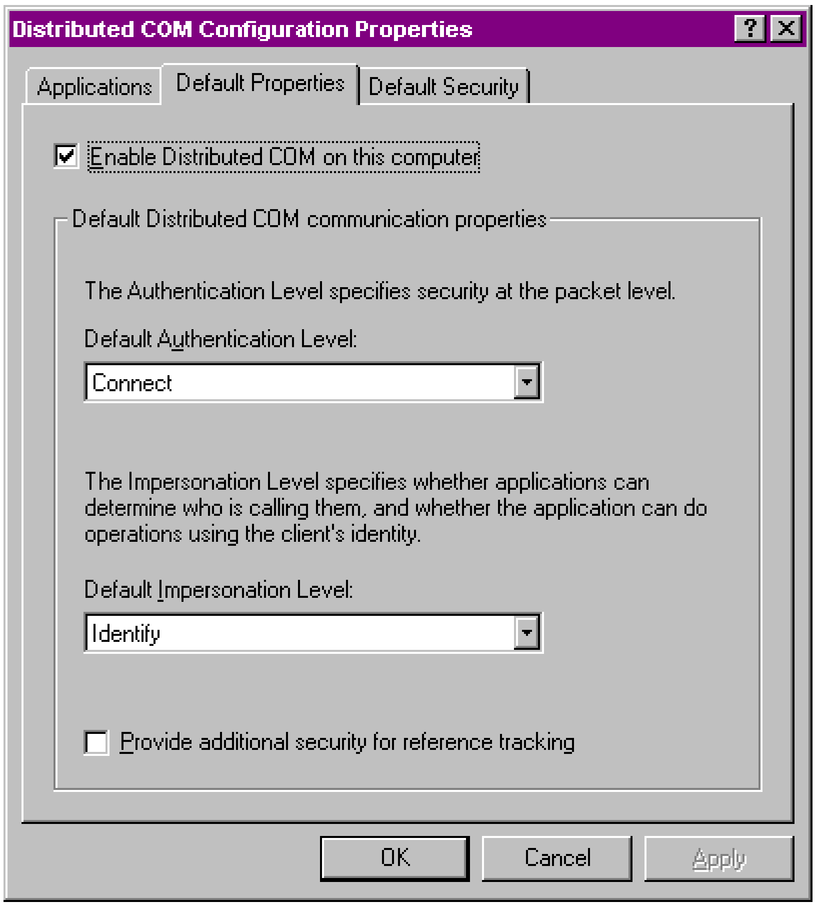
DCOM does all the work for you.

Goal:

Our BookServer running in the back office, VB clients elsewhere

Configuration

DCOM needs to be enabled on the remote server. Choose **Start -> Run** and enter **DCOMCNFG.EXE**. A configuration dialog starts up. Select the **Default Properties** tab and check the box for *Enable Distributed COM on this computer*.



[✓] Enable Distributed COM on this computer

Default Distributed COM communication properties

The Authentication Level specifies security at the packet level.

Default Authentication Level: Connect

The Impersonation Level specifies whether applications can determine who is calling them, and whether the application can do operations using the client's identity.

Default Impersonation Level: Identify

[ ] Provide additional security for reference tracking

Implementation

Visual Basic startup code needs one extra argument

```
Set BookServer = CreateObject("Doubletalk.BookServer",
RemoteMachine)
```

That's all!

…apart from Singletons, threads etc. – need to ensure multiple clients reach the same server instance, simultaneous clients don't create a problem etc. Read the book!

Part 3: - Cookbook

10 Database Access

Why do database work with Python?

- Connect to anything
- Lists and Dictionaries make data handling tasks trivial

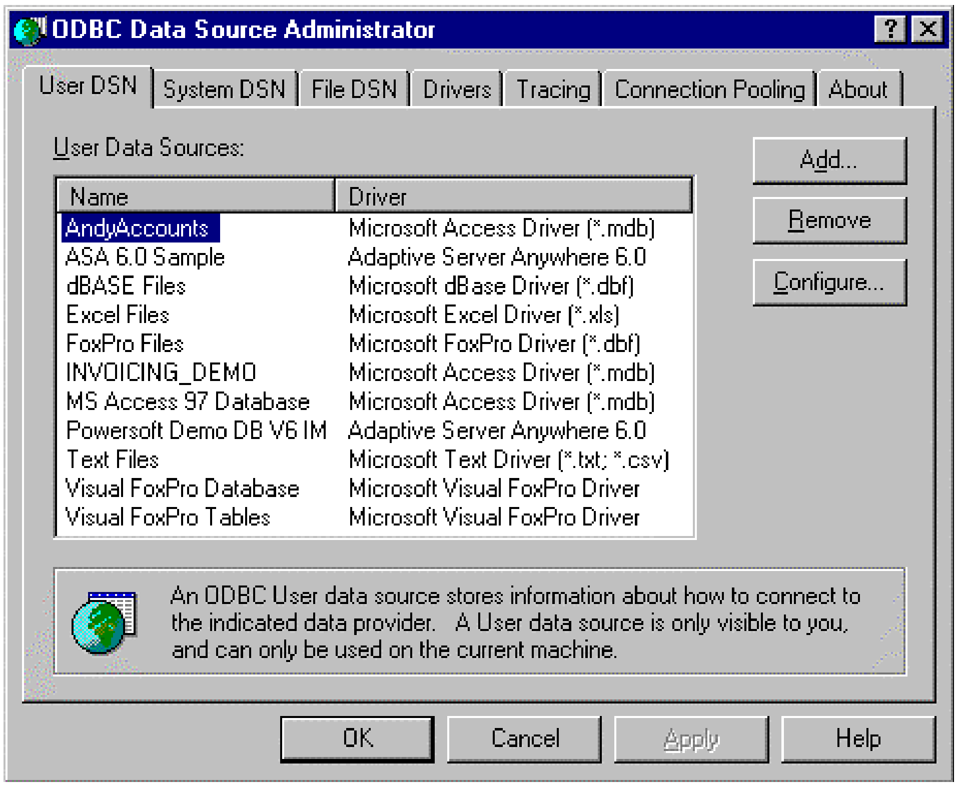
10.1 DAO, ADO, ODBC, OLEDB and other GBFLAs[1]

ODBC	Open Database Connectivity – industry standard C API for connecting to client-server SQL databases. Mature, stable and well supported.
DAO & Jet	COM object model for manipulating Access databases. ODBC support grafted on later, but not designed for client-server use. Used by the infamous Data Control in VB.
RDO	Remote Data Objects – Another COM hierarchy on top of ODBC, presents a DAO-like API, but much better remote server performance than DAO. Behind the slightly better Remote Data Control in VB.
OLEDB	Low-level COM interfaces suited to C programmers. Extends ODBC concept to query arbitrary data providers like hierarchical file systems, mail stores etc. Most existing OLEDB drivers use ODBC for connecting to relational databases.
ADO	ActiveX Data Objects – easy-to-use DAO-like COM object model sitting on top of ODBC. Good performance on remote databases; bells and whistles like client-side datasets and synchronization.

We will look at ODBC for client-server work, and DAO/ADO for getting at Access databases.

[1] Great Big Five Letter Acronyms

10.2 ODBC Data Source Administration



User Data Sources:

Name	Driver
AndyAccounts	Microsoft Access Driver (*.mdb)
ASA 6.0 Sample	Adaptive Server Anywhere 6.0
dBASE Files	Microsoft dBase Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
FoxPro Files	Microsoft FoxPro Driver (*.dbf)
INVOICING_DEMO	Microsoft Access Driver (*.mdb)
MS Access 97 Database	Microsoft Access Driver (*.mdb)
Powersoft Demo DB V6 IM	Adaptive Server Anywhere 6.0
Text Files	Microsoft Text Driver (*.txt; *.csv)
Visual FoxPro Database	Microsoft Visual FoxPro Driver
Visual FoxPro Tables	Microsoft Visual FoxPro Driver

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

10.3 Python's own database API and libraries

Currently on version 2.0. Most database extensions comply with it

Your options

- ODBC module in Pythonwin (version 1.0 only)
- mxODBC and mxDateTime (recommended for serious work)
- Various server-specific modules.
- Sam Rushing's calldll-based ODBC library

Recommendations:

- ODBC module in Pythonwin for occasional use
- mxODBC for regular database work,
- Sam Rushing's library if you love the raw ODBC API

10.4 Connecting with mxODBC

```
>>> import ODBC.Windows
>>> conn = ODBC.Windows.Connect('PYDBDEMOS')
>>> cursor = conn.cursor()
>>> cursor.execute('SELECT InvoiceID, ClientID, InvoiceDate FROM
Invoices')
>>> from pprint import pprint
>>> pprint(cursor.description)
(('InvoiceID', 4, None, None, 10, 0, 1),
 ('ClientID', 12, None, None, 10, 0, 1),
 ('InvoiceDate', 11, None, None, 19, 0, 1))
>>> data = cursor.fetchall()
>>> pprint(data)
[(199904001, 'MEGAWAD', 1999-04-15 00:00:00.00),
 (199904002, 'MEGAWAD', 1999-04-14 00:00:00.00),
 (199904003, 'MEGAWAD', 1999-04-21 00:00:00.00),
 (199904004, 'NOSHCO', 1999-04-22 00:00:00.00)]
```

Note those handy mxDateTime objects!

10.5 mxDateTime

```
>>> import DateTime
>>> DateTime.DateTimeFromCOMDate(0)  # the Microsoft system
1899-12-30 00:00:00.00
>>> aDateTime.COMDate()    # convert to Microsoft COM/Excel dates
36265.0
>>>
>>> DateTime.now() - aDateTime    # RelativeDateTime object
<DateTimeDelta object for '16:23:40:16.18' at 1106530>
>>> aDateTime + DateTime.RelativeDateTime(months=+3)
1999-07-15 00:00:00.00
>>> # first of next month...
>>> aDateTime + DateTime.RelativeDateTime(months=+1,day=1)
1999-05-01 00:00:00.00
>>> DateTime.now()
1999-05-01 23:42:20.15
>>> DateTime.Feasts.EasterSunday(2001)
2001-04-15 00:00:00.00
>>> DateTime.Parser.DateFromString('Friday 19th October 1987')
1987-10-19 00:00:00.00
>>>
```

Databases are full of dates. This is really useful.

10.6 Support for Prepared Statements

Database engines do two things: parse the SQL, then execute the query.

If doing many similarly-structured statements (e.g. 1000 imports), it only need be parsed once.

This saves memory on the server as well as time.

Cursors keep a handle to the last statement and reuse it.

SLOW: Cursor.execute(operation)

```
cursor.execute("""INSERT INTO analysis
     (tranid, trandate, account, amount) VALUES
     (1, {d '1999-12-31 23:59:59'}, 'Cash',100.00)""")
```

FASTER: Cursor.execute(operation[,parameters])

```
cursor.execute("""INSERT INTO analysis
     (tranid, trandate, account, amount)
     VALUES (?,?,?,?)""", (2, aDate, 'Cash', 117.50))
```

FASTEST: Cursor.executemany(operation,seq_of_parameters)

```
cursor.execute("""INSERT INTO analysis
     (tranid, trandate, account, amount)
     VALUES (?,?,?,?)""", [
(2, aDate, 'Cash', 117.50)
(3, aDate, 'Shares', -117.50)
# and so on
])
```

More importantly, parsing SQL takes memory on the server. Apps using prepared ones can support many more users while maintaining performance.

Benchmarking

```
accessdemo/slow:  141.643058/second
accessdemo/faster:  117.619382/second
accessdemo/fastest:  148.148148/second
asademo/slow:  292.825772/second
asademo/faster:  426.621164/second
asademo/fastest:  528.262016/second
Jet raw SQL inserts:  113.352981/second
Jet AddNew/Delete:  186.985792/second
```

Conclusions:

(1) Access doesn't know about prepared statements and is best accessed through DAO

(2) SQL databases are faster

(3) SQL databases go 80% faster with prepared statements

10.7 Connecting with Data Access Objects

Connecting

```
>>> import win32com.client
>>> daoEngine = win32com.client.Dispatch('DAO.DBEngine')
>>> daoDB = daoEngine.OpenDatabase('C:\\MYDIR\\pydbdemos.mdb')
>>> daoRS = daoDB.OpenRecordset('SELECT ClientID, InvoiceDate, \
                    Consultant, Hours FROM Invoices')
```

Iterating through Recordset

```
>>> daoRS.MoveLast()
>>> daoRS.Fields('ClientID').Value   # reference fields by name
'NOSHCO'
>>> daoRS.Fields(3).Value   # or by position
18.0
>>> for i in range(daoRS.Fields.Count):
...     daoField = daoRS.Fields[i]
...     print '%s = %s' % (daoField.Name, daoField.Value)
...
ClientID = NOSHCO
InvoiceDate = <time object at 1191860>
Consultant = Tim Trainee
Hours = 18.0
>>>
```

Grabbing Bulk Data with GetRows()

```
>>> daoRS.MoveFirst()
>>> data = daoRS.GetRows(4)
>>> pprint(data)
((L'MEGAWAD', L'MEGAWAD', L'MEGAWAD', L'NOSHCO'),
 (<time object at 11921f0>,
  <time object at 11921d0>,
  <time object at 11921b0>,
  <time object at 1192190>),
 (L'Joe Hacker', L'Arthur Suit', L'Joe Hacker', L'Tim Trainee'),
 (42.0, 24.0, 57.0, 18.0))
```

Note

1. We get columns, not rows!
2. We get COM dates and Unicode strings , just as from Excel

Regrettably there is no PutRows – need mxODBC for that!

Editing and Adding Data – Edit(), AddNew()

```
>>> daoRS2 = daoDB.OpenRecordset('SELECT * FROM Clients')
>>> daoRS2.AddNew()   # or Edit() for existing ones
>>> daoRS2.Fields('ClientID').Value = 'WOMBLES'
>>> daoRS2.Fields('CompanyName').Value = 'Wimbledon Disposal
Ltd.'
>>> daoRS2.Fields('ContactName').Value = 'Uncle Bulgaria'
>>> daoRS2.Update()    # save the record
>>> daoRS2.Close()
```

10.8 Connecting with ADO

Connecting is similar to DAO…

```
>>> import win32com.client
>>> adoConn = win32com.client.Dispatch('ADODB.Connection')
>>> adoConn.Open('PYDBDEMOS')  # use our ODBC alias again
>>> (adoRS, success) = adoConn.Execute('SELECT * FROM Clients')
>>> adoRS.MoveFirst()
>>> adoRS.Fields("CompanyName").Value
'MegaWad Investments'
>>>
```

The rest works exactly like DAO!

10.9 Gadfly – Pure Python Relational Database

Gadfly is intended to give Python programs relational database capabilities without relying on any external database engines. It offers the following features:

- Compliance with the Python Database API
- Transaction handling
- Total portability between platforms
- A transaction log and recovery procedure
- a built-in TCPIP server mode, allowing it to serve clients on remote machines
- Security policies to prevent accidental deletion of data

It is NOT intended as a multi-user production system, and some features are missing at present - notably Null values and Date/Time variables.

It also offers a TCPIP Network Client in just 15k of code!

The usual proof that it works…

```
>>> from gadfly import gadfly
>>> connection = gadfly("test", "c:\\mydir\\gadfly\\dbtest")
>>> cursor = connection.cursor()
>>> cursor.execute('SELECT * FROM Frequents')
>>> from pprint import pprint
>>> cursor.description     # only does fieldnames at present
(('PERWEEK', None, None, None, None, None, None),
('BAR', None, None, None, None, None, None),
('DRINKER', None, None, None, None, None, None))
>>> print cursor.pp()    # it can format its own output
PERWEEK | BAR      | DRINKER
============================
1       | lolas    | adam
3       | cheers   | woody
5       | cheers   | sam
3       | cheers   | norm
2       | joes     | wilt
1       | joes     | norm
6       | lolas    | lola
2       | lolas    | norm
3       | lolas    | woody
0       | frankies | pierre
1       | pans     | peter
```

Note nice console output!

11 Communications

- Serial I/O
- Remote Access
- Sockets

11.1 Serial I/O

Roger Burnham's Serial module – wraps MarshallSoft's WSC libraries.

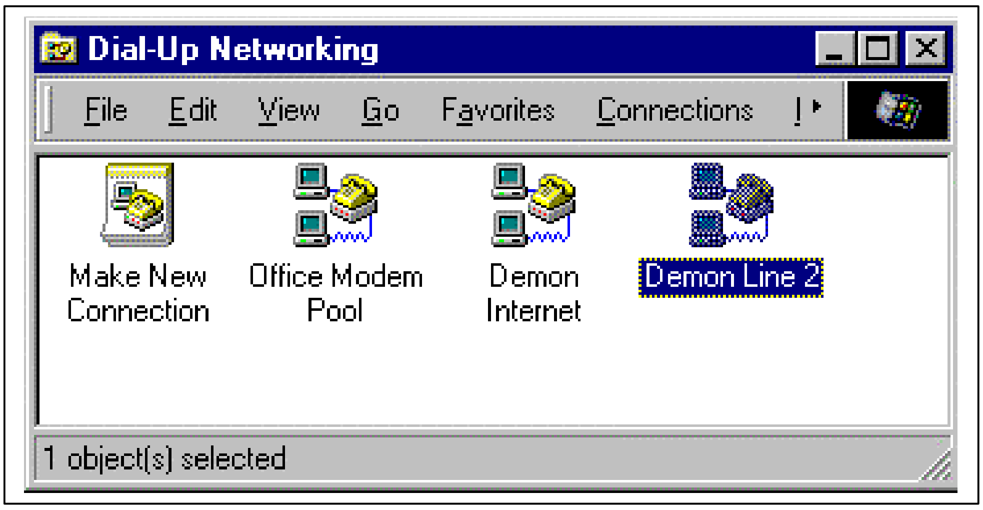
```
from Serial import Serial
#fill a special dictionary with the settings we want
cfg = Serial.PortDict()
cfg['port'] = port
cfg['baud'] = Serial.Baud9600
cfg['parity'] = Serial.NoParity
cfg['stopBits'] = Serial.OneStopBit
cfg['timeOutMs'] = 10000    # ten seconds

# create a Port object based on these settings
prt = Serial.Port(cfg)
prt.open()

#read some data
header = prt.read(22,timed=1)
prt.close()
```

High level wrappers – read() allows time interval.

Good excuse to play with threads!

11.2 Remote Access

win32ras extension exposes Remote Access management. Dial out under script control.



Also provides callbacks you can set to be triggered on connection.

```
>>> import win32ras
>>> myParams = ('Demon Internet','0845-3535666','',  \
                            'username','password','')
>>> (hras, success) = win32ras.Dial(None, None, myParams, None)
>>> # do your stuff on the network now
>>> win32ras.HangUp(hras)
>>>
```

11.3 Sockets, TCPIP and Internet Protocols

The Python library includes

- TCP/IP Sockets
- Web servers
- FTP libraries
- Email and News libraries

Connect to anything internet-based.

12 System Administration

User Names:

```
>>> import win32api
>>> userName=win32api.GetUserName()
```

User Info:

Windows API structs exposed as editable dictionaries (if you have permission)

```
>>> import win32net
>>> info=win32net.NetUserGetInfo(None, userName, 1)
>>> print info['name'] # print just the user name
skip
>>> dump(info)
priv = 2
home_dir = c:\winnt\profiles\skip\personal
password = None
script_path =
name = skip
flags = 66049
password_age = 23792806
comment =
>>>
```

This is level 1 – others levels offer around 30 pieces of information.

Adding Users

```
>>> d={}
>>> d['name'] = "PythonTestUser"
>>> d['password'] = "Top Secret"
>>> d['comment'] = "A user created by some Python demo code"
>>> d['flags'] = win32netcon.UF_NORMAL_ACCOUNT |
win32netcon.UF_SCRIPT
>>> d['priv'] = win32netcon.USER_PRIV_USER
>>> win32net.NetUserAdd(None, 1, d)
>>>
```

We said the dictionaries were editable…

What else?

Similar functions for working with

- groups
- servers
- network shares (e.g. create users and their shares in bulk)
- rebooting the system

13 Active Scripting

Lets you embed any scripting language using COM.

Use Python in

- Internet Information Server
- Internet Explorer
- Windows Scripting Host

13.1 Internet Explorer

Insert this in an hTML page

```
<SCRIPT Language=Python>
alert("Hello there")
</SCRIPT>
```

But how many people's browsers can you count on to have Python set up?

13.2 Internet Information Server <!--

```
ServerSample.asp - an example of Python
and Active Scripting
-->

<%@ Language=Python %>

<%
# Save the URL into a local variable
url = Request.ServerVariables("PATH_INFO")
%>

<H2>Current Document</H2>
The URL to this file is <pre><%= url %></pre><p>
The local path to this URL is <pre><%= Server.mappath(url)
%></pre>

<H2>Client Request Headers</H2>
<%
for key in Request.ServerVariables:
     Response.Write("%s=%s<br>" % (key,
Request.ServerVariables(key)))
%>
```

13.3 Windows Scripting Host

Easy-to-use COM objects for manipulating drives, users etc.

```
# wsh.pys
# A Windows Scripting Host file that uses Python.

WScript.Echo("The script name is", WScript.ScriptName)
if len(WScript.Arguments):
     WScript.Echo("The first argument is",
WScript.Arguments(0))

net = WScript.CreateObject("Wscript.Network")

netInfo = net.EnumNetworkDrives()
WScript.Echo(netInfo[0], "=", netInfo[1])
```

You can also access these objects from ordinary Python scripts.

14 GUI Development

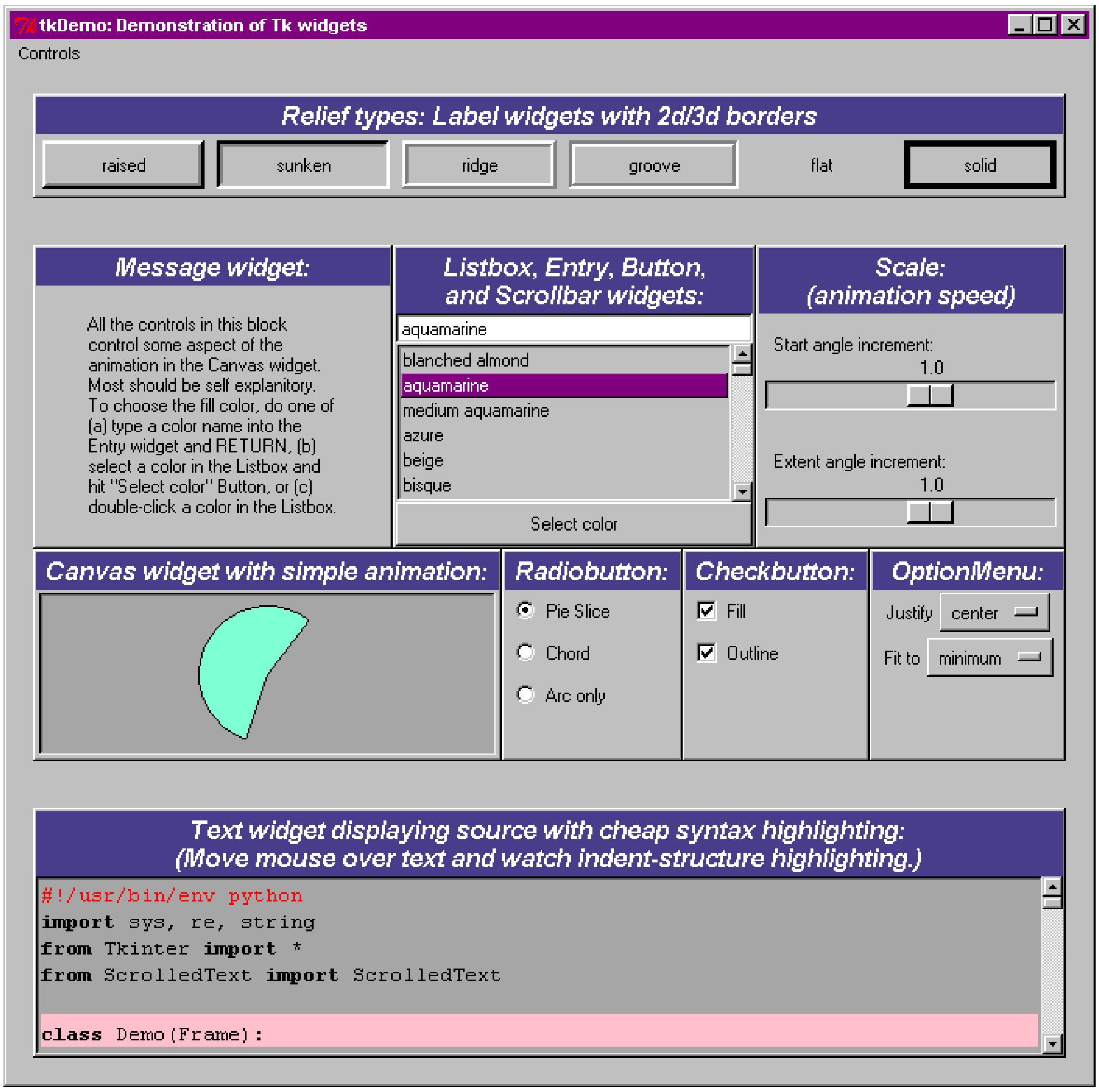
Options:

- Tkinter
- WxWindows
- PythonWin
- Embed via COM
- Embed at C level (C++, Delphi)

14.1 Pythonwin GUI

For MFC lovers. Extend Pythonwin, or write a whole app. You've seen enough screens.

14.2 Tkinter - Python's own platform-independent GUI.


tkDemo: Demonstration of Tk widgets
Controls
Relief types: Label widgets with 2d/3d borders
raised
sunken
ridge
groove
flat
solid
Message widget:
Listbox, Entry, Button, and Scrollbar widgets:
Scale: (animation speed)
Select color
Canvas widget with simple animation:
Radiobutton:
Pie Slice
Chord
Arc only
Checkbutton:
Fill
Outline
OptionMenu:
Text widget displaying source with cheap syntax highlighting:
(Move mouse over text and watch indent-structure highlighting.)

All the controls in this block control some aspect of the animation in the Canvas widget. Most should be self explanitory. To choose the fill color, do one of (a) type a color name into the Entry widget and RETURN, (b) select a color in the Listbox and hit "Select color" Button, or (c) double-click a color in the Listbox.

aquamarine

- blanched almond
- aquamarine
- medium aquamarine
- azure
- beige
- bisque

Start angle increment: 1.0

Extent angle increment: 1.0

● Pie Slice ○ Chord ○ Arc only

☑ Fill ☑ Outline

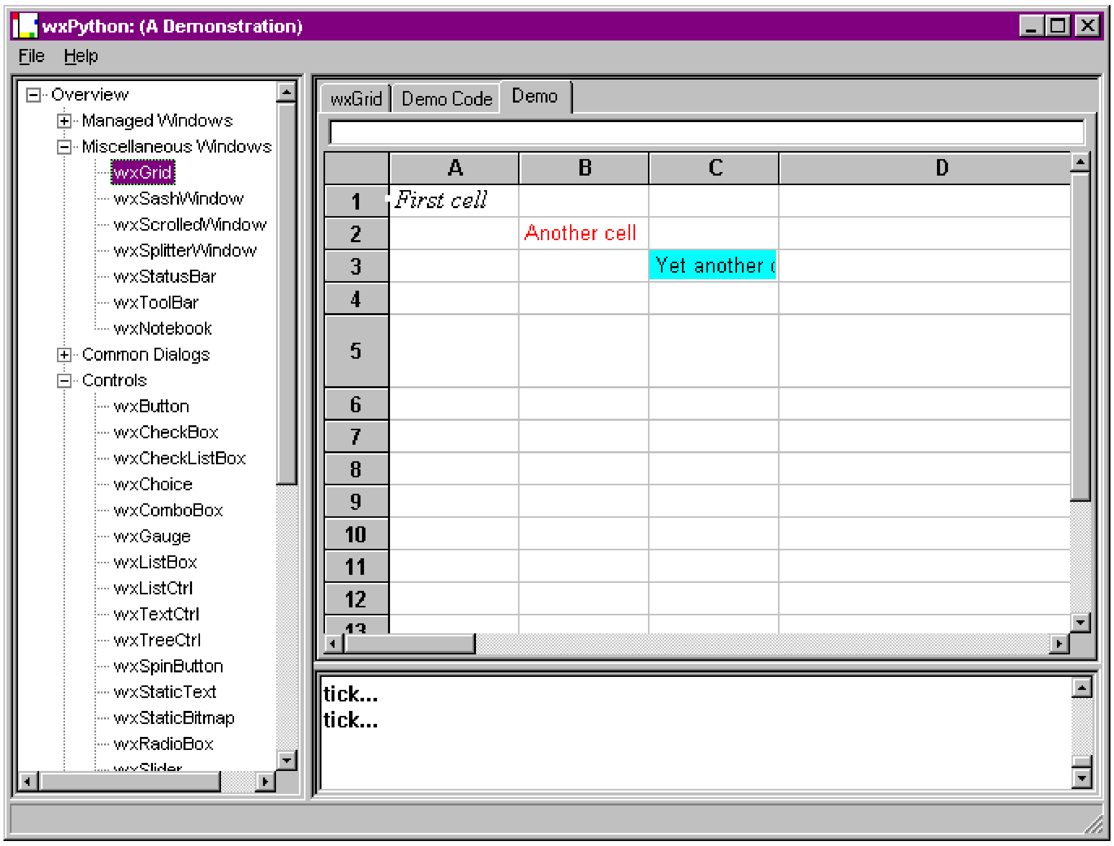
Justify: center

Fit to: minimum

```
#!/usr/bin/env python
import sys, re, string
from Tkinter import *
from ScrolledText import ScrolledText

class Demo(Frame):
```

14.3 wxWindows

Cross-platform, but based on heavily on Windows.



	A	B	C	D
1	*First cell*			
2		Another cell		
3			Yet another c	
4				
5				
6				
7				
8				
9				
10				
11				
12				

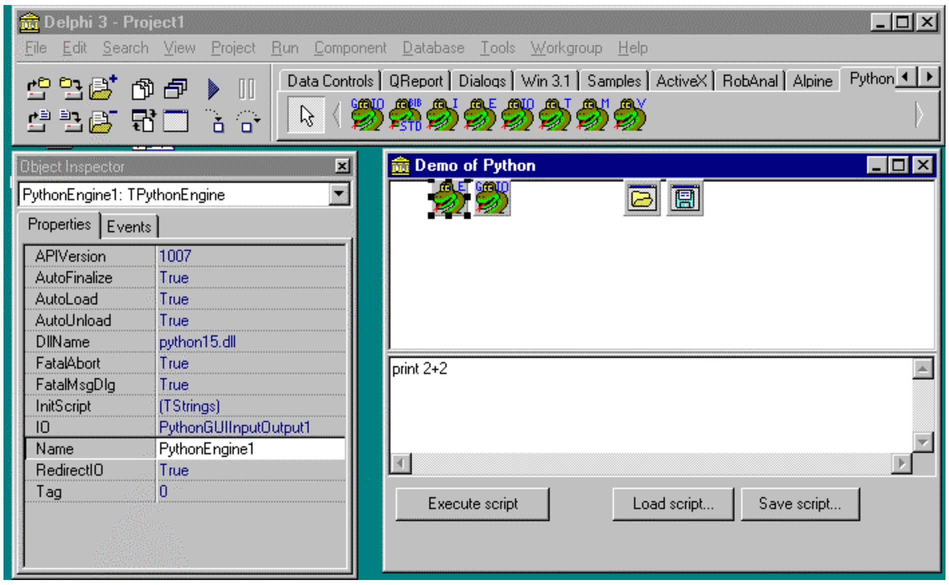
```
tick...
tick...
```

15 Delphi

C-Level API.

Python wrapped up as Pascal module

Python VCL components make it much easier than embedding in a C++ app.


Delphi 3 - Project1
File Edit Search View Project Run Component Database Tools Workgroup Help
Data Controls QReport Dialogs Win 3.1 Samples ActiveX RobAnal Alpine Python
Object Inspector
Properties Events
Demo of Python
Execute script
Load script...
Save script...

PythonEngine1: TPythonEngine

APIVersion	1007
AutoFinalize	True
AutoLoad	True
AutoUnload	True
DllName	python15.dll
FatalAbort	True
FatalMsgDlg	True
InitScript	(TStrings)
IO	PythonGUIInputOutput1
Name	PythonEngine1
RedirectIO	True
Tag	0

```
print 2+2
```

16 C and C++ Level Integration

Python is written in ANSI C, and is highly modular

- You can write new extension modules
- You can include Python in your existing C/C++ programs
- The win32 extensions are examples of this.
- Simplified Wrapper and Interface Generator (SWIG) helps automate the process of building extensions
- CallDLL/WinDLL dynamically calls DLLs without a C compiler

16.1 Win32 extensions

Module	**Description**
mmapfile	Interfaces to Windows memory-mapped files, a mechanism that allows data to be shared among multiple processes.
odbc	An interface to the Open DataBase Connectivity API, a portable API for connecting to multiple databases.
win32api	Access to many of the common and simple Windows APIs. A very general-purpose module with a cross-section of API support.
win32event	Access to the Windows event and signaling API. Allows you to manipulate and wait for Windows Events, Semaphores, Mutexes, and so forth.
win32evtlog win32evtlogutil	An interface to the windows NT Event Log. The `win32evtlog` module provides a raw interface to the Windows NT API, while the `win32evtlogutil` module provides utilities to make working with the module much simpler. This is discussed in Chapter 18, *Windows NT Services.*
win32pdh	An interface to the Windows NT Performance Monitor. This module uses a helper DLL provided by Microsoft knows as the “Performance Data Helper”, or PDH, hence the name.
win32pipe	Access to the pipe related Win32 functions. These include functions for creating and using pipes, included named pipes. We discuss pipes briefly in Chapter 17, *Files and Processes*, and use a pipe from the `win32pipe` module in Chapter 18, *Windows NT Services*.
win32file	Access to the file-related Win32 functions. This exposes a low-level, raw interface to files on Windows, and is only used when the standard Python file object is not suitable. Python files, and the `win32file` module are discussed in Chapter 17, *Files and Processes*.
win32lz	An interface to the Windows LZ compression library. Note that Python also ships with support for the gzip compression format, but `win32lz` is handy as it does not require any external DLLs (eg, *zlibp.dll*) to be installed.
win32net win32wnet	Interface to the Windows networking API.
win32print	Interface to the printer-related Windows APIs.
win32process	Interface to the process related Windows APIs. This is discussed in detail in Chapter 17, *Files and Processes*.
win32ras	Interface to the Windows Remote Access Service (RAS). Used for

	establishing remote connections to Windows NT Servers, typically using a modem.
win32security	Access to the Windows NT security related functions.
win32service win32serviceutil	Access to the Windows NT Services-related API. This is discussed in detail in Chapter 18, *Windows NT Services*.
win32trace win32traceutil	Debugging related modules. These modules allow you to collect the output of a Python process in a separate process. This is most useful when debugging server-style applications, where Python error and other messages are not available.

16.2 CallDLL/WinDLL

www.nightmare.com

Tiny (14k) Python extension giving you pointers, memory buffers, LoadLibrary() and GetProcAddress().

Windll wraps it up nicely: call any DLL!

```
>>> from dynwin.windll import *
>>> mod1 = module('c:\\temp\\simple')  # loads the DLL
>>> mod1.handle   # it can report its location in memory
22806528
>>> mod1.Min(27, 28) # loads and executes Min function
27
>>> mod1.Min       # we now have a 'callable function' object...
<callable function "Min">
>>> mod1.Min.address    #...which knows its address too
22836704

>>> inBuf = cstring('spam')  # make a buffer holding a c string
>>> outBuf = cstring('',50)  # make another big enough for
output
>>> mod1.StringRepeat(inBuf, outBuf, 10)  # returns the length
of out string
40
>>> outBuf
'spamspamspamspamspamspamspamspamspamspam'
```

Of course, this also gives you the ability to crash, unlike a well-made Python extension.

The End