Modules and Packages

Objectives

To write and call our own user-written modules, and to continue practising Python.

Reference Material

Chapter 10 Modules and Packages.

Questions

1. In this exercise we will take two functions you wrote earlier and turn them into a module.

   The previous chapter included a question where you were asked to write two timing functions, **`start_timer()`** and **`end_timer()`**. If you did not complete that exercise don't worry, a sample solution is provided in **Ex10.1.py**. Use that file as a basis of this exercise, or your own solution if your wish.

   Create a module called **mytimer**, which contains these functions (and any other supporting variables). Test the module by importing it and calling the functions before and after a lengthy operation, as before.

   Note: there is a module called **timeit** in the Python Standard Library. If you look in the documentation you will find it is rather more complex then ours. On Windows there is also a module bundled with Python called **timer**. So do not use either of those module names.

2. Now test your module's docstring using IDLE. You did document your module, didn't you? If you did not, now is a good time.

   To test under IDLE, first **`import mytimer`**. Did that work? If IDLE did not find your module then maybe you should tell it where it is (hint: **`sys.path`**)? The easiest way to grab the path is to copy it from the Address bar in Windows Explorer and paste it into IDLE (use a "raw" string).

   Once you have managed to import the module, type:

   ```
   >>> help(mytimer)
   ```

3. Our module is not complete without some tests. Add a simple test to the docstring: call `start_timer()` immediately followed by `end_timer()`, so that the result is predictable. Do not forget to add the expected output. Then add the test for__ **`main`**__, with the call to **`doctest.testmod()`**.

   Test by running **`timer.py -v`** from the Windows command-line (cmd.exe).

If time allows…

Create a sub-directory called **mymodules**, and copy your timer.py module into it, but rename the file to **timer2.py**.

Add an empty **`__init__.py`** file to the sub-directory.

What modifications are required to your test code to use this package?

4. Write a module `printf.py` which provides functions similar to the C library routines sprintf, fprintf, and printf, using the 'old style' format syntax. See the slides after the summary of the "04 String Handling" chapter.

   Functions should be as follows:

   `sprintf(`*fmt*, **args*`)`
   Where *fmt* is a format string
   args is the argument list

   Returns a formatted string

   `fprintf(`*file*, *fmt*, **args*`)`
   Where *file* is a file object opened for write
   fmt is a format string
   args is the argument list

   Writes the formatted string to *file*

   `printf(`*fmt*, **args*`)`
   Where fmt is a format string
   args is the argument list

   Writes the formatted string to `sys.stdout`

Write **doctest** tests for your `printf` and `sprintf` functions. Note: omit "\n" from the format strings in your tests because doctest sees them as end-of-test.

Solutions
Questions 1-3

The test script looks like this:

```
import mytimer
mytimer.start_timer()
lines = 0
for row in open("words"):
    lines += 1
mytimer.end_timer()
print "Number of lines:",lines
```

Here is our final module:

```
"""
This user written module contains a simple mechanism for
timing operations from Python.  It contains two
functions, start_timer(), which must be called first to
initialise the present time, and end_timer() which
calculates the elapsed CPU time and displays it.

>>> start_timer()
>>> end_timer()
End time    : 0.000 seconds
"""
import os
start_time = 0;

############################################################
# TIMER FUNCTIONS
def start_timer():
    """
    The start_timer() function marks the start of a timed
    interval, to be completed by end_timer().
    This function requires no parameters.
    """
    global start_time
    (utime,stime) = os.times()[0:2]
    start_time = utime+stime

def end_timer(txt='End time'):
    """
    The end_timer() function completes a timed interval
    started by start_timer.  It prints an optional text
    message (default 'End time') followed by the CPU time
    used in seconds.
    This function has one optional parameter, the text to
    be displayed.
    """
    (utime,stime) = os.times()[0:2]
    end_time = utime+stime
    print "%-12s: %05.3f seconds" % \
          (txt,end_time-start_time)

if __name__ == "__main__":
    import doctest
```

```
    doctest.testmod()
```

If time allows:

The test script can be modified as follows:

```
import mymodules.mytimer2 as mytimer
```

That way we do not need to change the function call code.

Question 4

```
"""
    This module supplies functions sprintf, fprintf,
    and printf.

    >>> printf("%s","hello")
    hello
    >>> printf("%x",42)
    2a
    >>> printf("|%06.2f %-12s|",3.1426,"hello")
    |003.14 hello       |
    >>> var = sprintf("%X",3735928559)
    >>> print(var)
    DEADBEEF
"""
import sys

def sprintf(fmt, *args):
    rstr = fmt % args
    return rstr

def fprintf(file, fmt, *args):
    file.write(sprintf(fmt, *args))

def printf(fmt, *args):
    fprintf(sys.stdout,fmt,*args)

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```