Importing modules

The concept of importing modules may lead to some confusion, especially if you expect a behaviour similar to other programming languages.

To see clearer I did some simple tests.

mymodule.py is the module to be imported and contains this:

```
print "HAHA"

def sayhi():
    print "Hi"

class Hello:
    def __init__(self):
        print "HELLO"
```

The main program for the test that imports mymodule is located in the same directory.

I tried different importing mechanisms:

1. **Import only the name of mymodule to the namespace:**
To access functions or objects, their name must be preceeded by the name of the imported module

```
import mymodule
h = mymodule.Hello()
mymodule.sayhi()
```

2. **Import the whole namespace of mymodule:**

(with the disadvantage of eventual name confusion, if the same name exists in both)

```
from importtest import *
h=Hello()
sayhi()
```

3. **Import only selected functions / objects**

```
from mymodule import sayhi, Hello
h=Hello()
sayhi()
```

The three methods had the expected result:

HAHA
HELLO
Hi

So:

1. All code residing in the imported module is executed, even if only selected functions or objects are imported explicitely.
2. The 3 different methods of importing give the same result, as expected.
3. If the imported module is found in the same folder as the main program, there is no problem.

Modules and global variables

1. Using a variable defined in a module that is imported

Module mymodule is the module to be imported and contains only two statements:

```
x = 1
print „hello“
```

The first sets a variable, the second tells us that the module has well been imported.

Now we try to import the module and use the variable x

First try for the test program:

```
import mymodule
print x
```

This does not work!
The ‚hello‘ ist printed, that means the module is imported. But we get an error when we try to print the variable x.
I must confess that, at first, this was an unexpected behavior.
As every statement of an imported module is executed, my conclusion was that importing a module would be roughly the same as writing all the code into one bigger file.
And there I was wrong!
Why?
Variables are only global within the module in which they are defined! So they can't be accessed from outside, even if the module is imported.

Second try:

```
from mymodule import x
```

```
print x
```

Now we have explicitely imported the name x from the namespace of mymodule.
So it is known to the main program. And so this works correctly!

Third try:

```
from mymodule import *
print x
```

This works also as we import the whole namespace to the main program.

Conclusion:
If we want to use variables defined in an imported module, we must import the whole namespace (from mymodule import *) or seletively import the needed variables (from mymodule import x).

2. Using a variable defined in the main program in an imported module

First try:

The main program contains:

```
y=1
import mymodule
```

and mymodule:

```
print 'hello'
print y
```

This does not work, as main program and mymodule have different namespaces.
Even a statement `global y` in one or both files does not help.

The solution:

Use get and set functions and a globaql variable in the imported module:

mymodule:

```
global y
y=-1        # default value

def set_y(yvalue):
    global y
    y=yvalue
```

```
def get_y():
    return y


def print_y():
    print y
```

Now we can work with the variable y in the main program:

```
from mymodule import *

print_y()            # print default value

set_y(5)             # set new value to 5
print_y()            # and print it

set_y(7)             # set new value
x= get_y()           # get it from main module level
print x
```

Note:
In mymodule, it is important to use the global statement in the set_y function, because here the value is changed. When the value is only read, as in print_y or get_y, the global statement is not needed.

Conclusion:

- **At module level (for a module that shall be imported), it is good to put the whole code into functions that can be called from the outside.**
- **Default values can be defined at the beginning of the code.**
- **Variables declared with 'global' are global at module level, but not outside of it!**

A still more pythonic way would be to use only objects in the imported module.