Modular Programming

Arthur Molnar

Babes-Bolyai University

arthur@cs.ubbcluj.ro

October 21, 2015

Overview

Modules

Modular programming - a software design technique that increases the extent to which software is composed of separate, interchangeable "components", called **modules** by breaking down program functions into modules, each of which accomplishes one function and contains everything necessary to accomplish this.

What is a module - A module is a collection of functions and variables that implements a well defined functionality

Modules

A Python module - a file containing Python statements and definitions (executable statements).

- **name**: The file name is the module name with the suffix ".py" appended
- **docstring**: triple-quoted module doc string that defines the contents of the module file. Provide summary of the module and a description about the module's contents, purpose and usage.
- **executable statements**: function definitions, module variables,initialization code

The Eclipse IDE

Eclipse + PyDev IDE (Integrated Development Environment)

- Project
- Package
- Module
- Run, Debug, Navigate programs

The Eclipse IDE

Eclipse + PyDev IDE (Integrated Development Environment)

1. Install Java JDK **(make sure it's the JDK)**
2. Install Eclipse for Java from eclipse.org
3. Start Eclipse for the first time, see it works
4. Install PyDev plugin over Eclipse from http://www.pydev.org/download.html (section "as Eclipse plugin")

Why modules?

- Allows grouping related functionalities
- Allows easier delivery and deployment of related functionalities
- Helps with solving naming conflicts

Importing modules

In order to use a module wee need to import it. The import statement:

- Searches the global namespace for the module. If the module exists, it had already been imported nothing more needs to be done.
- Searches for the module. If the module name can't be found anywhere, an **ImportError** exception is raised.
- If the module file was found, execute the statements from the module.

Importing modules

The executable statements (including function definitions) from a module are executed only the *first* time the module is imported somewhere.

from doted.package[module] import {module, function}

```
from utils.numericlib import gcd

#invoke the gdc function from module utils.numericlib
print gdc(2,6)
```

```
from rational import *

#invoke the rational_add function from module rational
print rational_add(2,6,1,6)
```

```
import ui.console

#invoke the run method from the module ui.console
ui.console.run()
```

Importing modules

What if we don't want to execute statements at import?

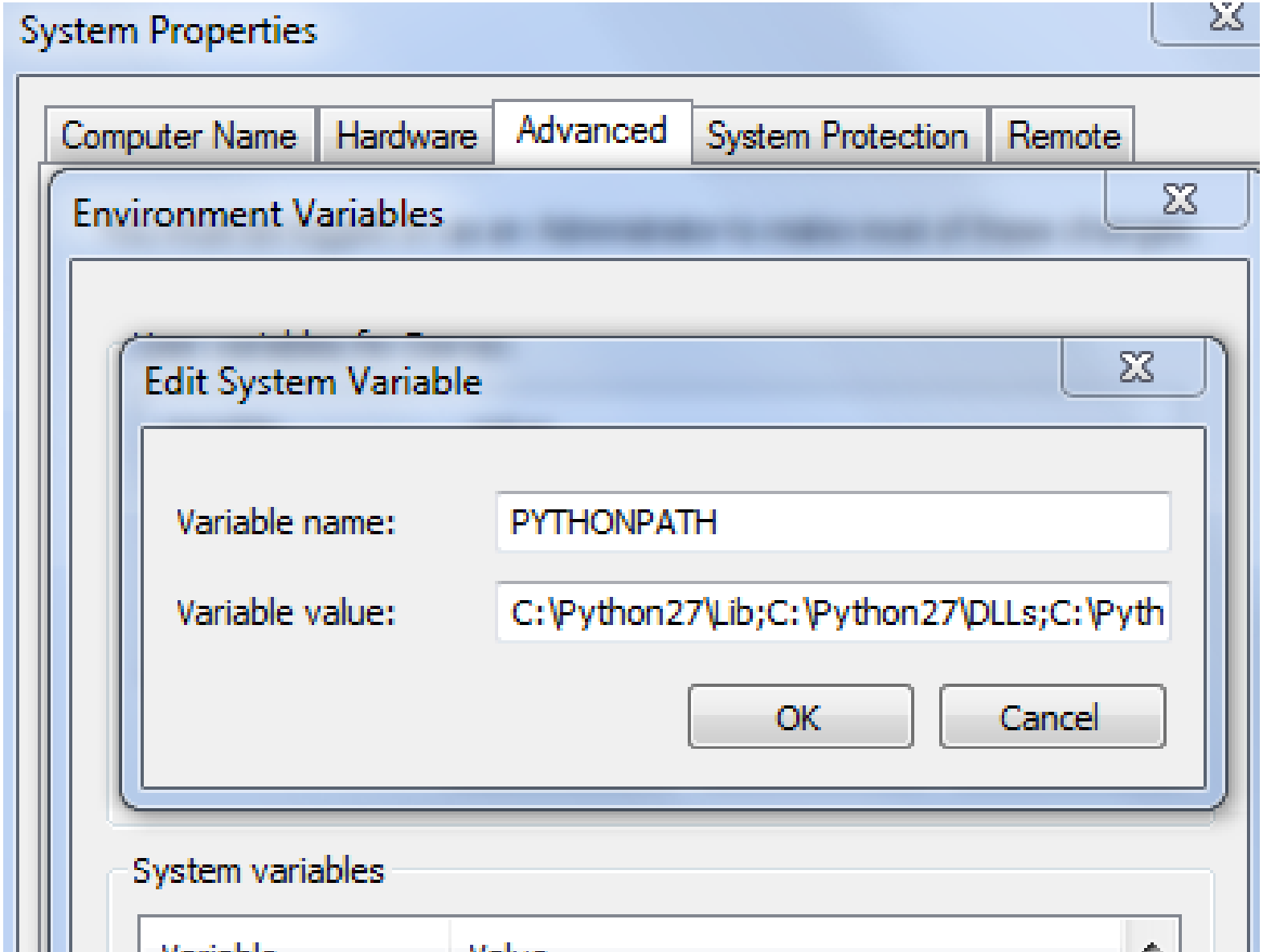
```
if __name__ == '__main__':
    # your code
```

Module search path

The import statement will search for a file called modulename.py:

- the directory containing the input script
- in the list of directories specified by the environment variable **PHYTONPATH**
- in the list of directories specified by the environment variable **PYTHONHOME**, an installation-dependent default path; on Unix, this is usually: /usr/local/lib/python.

Module search path - PYTHONPATH



Initialize the module

The module can contain any executable statements. When the module is first imported the statements are executed. We can put some statements (other than function definition) that will do any necessary initialization of the module.

Variable scope in a module

- On importing a module the variables and functions defined in the module will be inserted into a new symbol table (a new namespace). Only the module name will be added to the current symbol table
- You can use the built-in **dir() dir(module_name)** function to examine the symbol tables

Variable scope in a module


```
#only import the name ui.console into the current symbol table
import ui.console

#invoke run by providing the doted notation ui.console of the package
ui.console.run()
```

```
#import the function name gdc into the local symbol table
from utils.numericlib import gcd

#invoke the gdc function from module utils.numericlib
print gdc(2,6)
```

```
#import all the names (functions, variables) into the local symbol table
from rational import *

#invoke the rational_add function from module rational
print rational_add(2,6,1,6)
```

Example

Python modules example...

Packages

Packages are a way of structuring Python's module namespace by using "dotted module names"
The **__init__.py** files are required to make Python treat the directories as containing packages (it can also execute initialization code for the package)

How to organize your source code into modules

Create separate modules for:

- **User interface** - Functions related to the user interaction. Contains input, print operations. This is the only module where input/print operations are present.
- **Domain / Application** - Contains functions related to the domain of the problem.
- **Infrastructure** - Utility functions with high reuse potential.
- **Application coordinator** - Initialize and start the application.

How to document a module

- **help()** - Function that extracts the documentation for a module, function, or class.
- **pydoc** - A module that allows you to save extracted documentation to HTML format. Best used in command line at the operating system prompt.

Calculator - modular version