

THE GEORGE WASHINGTON UNIVERSITY  
School of Engineering and Applied Science  
Department of Computer Science

## User's Manual

For

Sython:  
A Security Addition to the Python Programming Language

PRELIMINARY RELEASE EDITION

# TABLE OF CONTENTS

LIST OF FIGURES.....	iii
LIST OF TABLES .....	iv
1. PRODUCT DESCRIPTION .....	5
1.1. MODULE DESCRIPTIONS.....	5
2. PRODUCT SPECIFICATIONS.....	7
3. INSTALLATION .....	8
4. TUTORIAL.....	10
5. SYTHON REFERENCE MANUAL.....	12
5.1. SYTHON SYNTAX.....	12
5.2. SYTHON BUILT-IN FUNCTIONS .....	13
5.3. SYTHON OBJECT METHODS .....	14
5.4. SYTHON DAEMON CONFIGURATION OPTIONS .....	15
5.5. SYTHON DAEMON FILES .....	16
6. MAINTAINANCE .....	19
7. TROUBLESHOOTING.....	20
7.1. INSTALLATION .....	20
7.2. PROGRAMMING PROBLEMS.....	20
7.3. CONFIGURATION PROBLEMS .....	21

## LIST OF FIGURES

Figure 1 sythond Starting Output.....	10
Figure 2 sython Starting Output.....	10

## LIST OF TABLES

Table 1 Sython Syntax Validity .....	12
Table 2 Sython Daemon Command Line Arguments .....	15
Table 3 Sython Daemon Log File .....	16
Table 4 Sython Daemon Database File .....	17
Table 5 Sython Daemon Input Text File .....	17
Table 6 Sython Daemon Output Text File.....	18

# 1. PRODUCT DESCRIPTION

Sython adds the ability to specify certain variables in Python as secure. The values of these secure variables are restricted and only acted upon in a black-box-like fashion. This allows software development to be outsourced, and testing to be performed upon live data while still protecting the data's sensitivity.

Sython accomplishes this data protection by separating the storage of secure variable into a different memory space. The memory space belongs to Sython's sister application, the Sython Daemon (sythond). Requests to modify secure variables are sent (transparently) to sythond. Sythond will not allow any request that could lead to the values of secure variables being known by the client operator.

Secure variables in Sython are denoted in a specific fashion that visually separates them from regular variables. This is in keeping with Python's philosophy of highly readable source code. In addition, Sython adds a number of new built-in functions to operate on the secure variables.

This manual assumes basic familiarity with regular Python. It details all changes to Python from the point of view of both the developer and the system administrator. For a well-written, high-level tutorial on Python, Wikipedia is an excellent resource; visit [http://en.wikipedia.org/wiki/Python\\_programming\\_language](http://en.wikipedia.org/wiki/Python_programming_language).

## 1.1. MODULE DESCRIPTIONS

The Sython distribution consists of two main applications: the Sython Interpreter, sython and the Sython Daemon, sythond. Within each application there are a number of modules. This section will discuss each module in detail.

The Sython interpreter consists of two modules: the Python module and the communicator module. The Python module is a modified version of the Python 2.3.4

distribution. It has been modified to support the dollar sign (\$) as the indicator of secure variables and adds several new built-in functions, as detailed in Chapter 4. The communicator module handles communication with the daemon application via a well-defined protocol. It is implemented as a Python module called `sython.comm`.

The Sython daemon consists of three modules. These modules are the main module, the handler module and the database module. The daemon main module initializes the other modules, based upon command-line arguments supplied when the daemon is launched. The handler module is the most significant module within the daemon. It consists of two sub-modules, the fetcher and the handler. The fetcher receives each request from the interpreter and performs basic validity checks. After passing those checks the request is forwarded to the handler for processing. The last module within the daemon is the database module. This module adds support for simple database functionality within Sython. The database module is implemented as an interface to a SQLite database.

## 2. PRODUCT SPECIFICATIONS

This version of Sython was developed for the Apple MacOS X platform. As such official testing has been conducted on Apple Macintosh computers. Sython officially supports Macs with G4 processors with 512 MB of RAM or more. In practice, however, any Macintosh that is able to run operating system versions 10.3 or above should be able to run Sython. Note: the Apple Developer Tools are required to install Sython on MacOS X.

Python version 2.3.4, from which Sython is derived, runs on a number of platforms including Windows and Linux. Care has been taken to use platform-independent coding practices when constructing Sython; as such Sython should be portable to any platform that Python supports. Changes may need to be made to the installation scripts.

For database functionality Sython requires SQLite version 3.1 or above. SQLite can be downloaded from <http://www.sqlite.org/>. In addition, Python SQLite bindings are required. A package called PySQLite handles this task. It can be found at <http://initd.org/tracker/pysqlite>. Versions supporting SQLite 3 are required. Note: Sython will run without database support. If database functionality is not required, SQLite and PySQLite are not required.

### 3. INSTALLATION

This chapter will detail the installation procedure for Sython. If database functionality is required, SQLite and PySQLite will need to be installed before Sython can be installed. To install these packages, visit the sites listed in Chapter 2. Please refer to SQLite and PySQLite documentation for installation directions for those packages.

Installation of Sython is very similar to the installation of Python. To install, copy the Sython distribution found in the `install/sython` directory of the distribution CD into a working directory (such as the Desktop) then unpack it using the `tar -zxvf Sython.tgz` command. This will create a new folder containing the Sython distribution. The following paragraphs detail the installation procedure for both the Sython Interpreter and the Sython Daemon.

From here there are two methods of installation. Sython supports the traditional Python Unix-style installation, but it also supports the installation of a Darwin/MacOS X Framework. The Framework allows Sython to be embedded in MacOS X applications. It is also less likely to be effected by operating system upgrades than the traditional Unix-style installation. The Unix-style installation will be covered first.

In Terminal.app or another terminal application, change into the unpacked sython directory. Once there, three simple commands will install the Sython Interpreter. Type `./configure` and then hit Return. This will create a MakeFile tailored to your environment. Once the configure command has succeeded type `make` into the terminal and hit Return. This will actually build Sython. Once make has completed, the one final command is needed to actually install Sython. Type into the terminal `sudo make install`, hit Return and type in your password if prompted. Because Sython will install itself in the System's `/usr/local` directory, administrator privileges are required. Once this command has completed, Sython has been successfully installed. If during any of these steps, you encountered an error, please consult the troubleshooting chapter at the end of this manual.



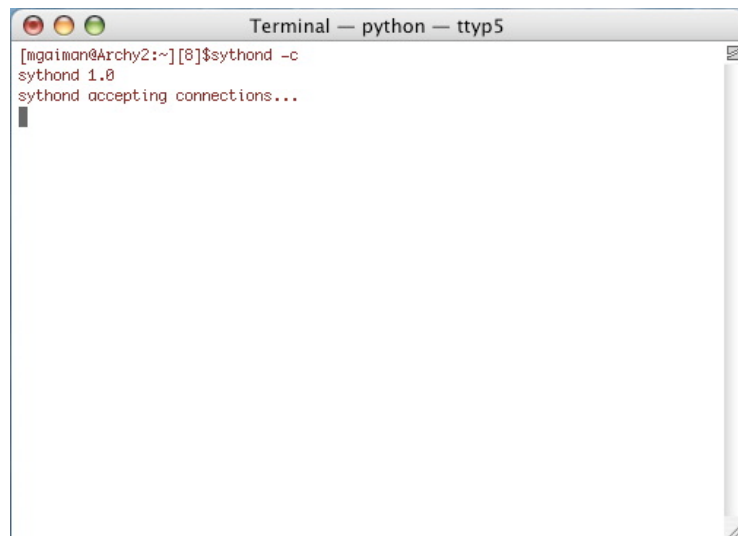
The Framework installation of Sython is very similar to the Unix-style installation. In Terminal.app or another terminal application, change into the unpacked sython directory. Once there, follow the steps detailed in the Unix-style installation, but change the commands to the following: `./configure` becomes `./configure --enable-framework` and `sudo make install` becomes `sudo make frameworkinstall`. The Sython framework will be installed in the `/Library/Frameworks` directory. To install in a different directory, change the configure command to `./configure --enable-framework=<installation directory>`.

The last installation task is to install the Sython Daemon. In the install directory of the distribution CD there is a subdirectory labeled sythond. To install the Sython Daemon copy that directory into a more permanent place such as `/usr`, then from within the terminal application, change into the moved sythond directory, type `./install_syd` and hit Return. You may be prompted for the administrator password. This script will write a script that launches the Sython Daemon in `/usr/local/bin`, allowing the user to invoke the daemon from any location. In addition to the Daemon application itself, the sythond folder contains sample input, output and database files.

## 4. TUTORIAL

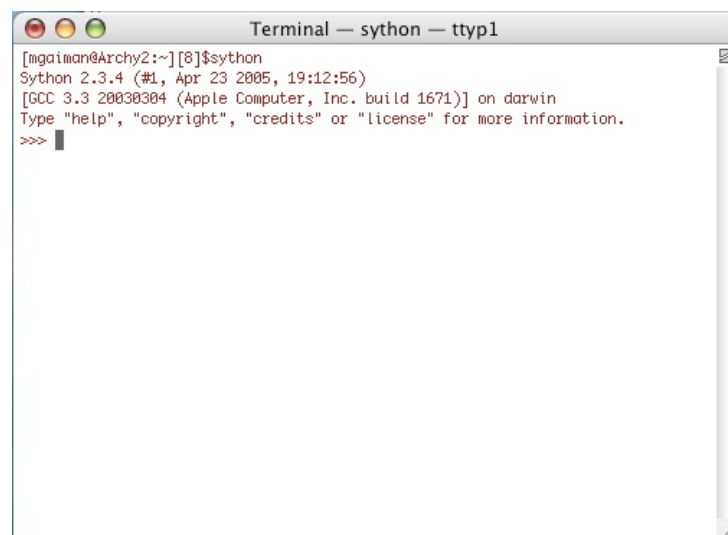
This chapter offers a brief run though using `sython` and `sythond`. It does not seek to be exhaustive, but rather touches upon the most used features of Sython.

To start the daemon, type `sythond -c` from the terminal window. Typing `sythond -h` will display all of the command line parameters. The `-c` parameter starts it in an interactive console mode. The output of `sythond` when first started, looks like Figure 1. Now, open a new terminal window and type `sython`. The output of `sython` when first started, looks like Figure 2.

A terminal window titled "Terminal — python — tty5" showing the output of the command `sythond -c`. The output is:

```
[mgaiman@Archy2:~][8]$sythond -c
sythond 1.0
sythond accepting connections...
█
```

**Figure 1 sythond Starting Output**

A terminal window titled "Terminal — sython — tty1" showing the output of the command `sython`. The output is:

```
[mgaiman@Archy2:~][8]$sython
Sython 2.3.4 (#1, Apr 23 2005, 19:12:56)
[GCC 3.3 20030304 (Apple Computer, Inc. build 1671)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

**Figure 2 sython Starting Output**

We will begin by allocating a new sython variable. This is done via the line `$x=syalloc('i')` This line allocates a new variable `x`. The variable is a handle for an integer that is stored on the Sython Daemon. The value was randomly assigned. The function `syalloc()` is just one of a number of new built-in functions. Consult the reference manual in the next chapter for a complete list. Try to determine the value of `x` by typing `print x` and hitting Return. Only the memory location of the `syobj` handle is revealed. The actual value isn't even stored in Sython—it is in sythond's protected memory. Next we will allocate another secure variable. Type in `$y=syalloc('i', (10,20))` This will allocate a new variable `y` which will be a handle to another integer in the Sython Daemon. This integer will be randomly chosen from the range ten to twenty. Just like normal Python variables, secure variables support mathematic operations, type `$z=x+y` The variable `z` now contains the sum of `x` and `y`. A useful debugging tool in Sython is the new built-in function `syval()`. The function returns a string representation of a secure variable. The string will stay the same between calls as long as the secure variable itself does not change. If the variable changes, the return value may or may not change, too. To see `syval()` in action, type `syval(z)`.

This has been a brief tutorial on using the new features of Sython. It has been far from exhaustive. Users are urged to read the reference manual in the next chapter for a more complete description of all new Sython features. Also, the Python built-in help is a valuable resource. Simply type `help("syalloc")` into the Sython interpreter prompt to see help about `syalloc()` (or any other new Sython function).

## 5. SYTHON REFERENCE MANUAL

This chapter is the main reference for Sython. It details all of the changes to the standard Python Interpreter as well as the various configuration options for sythond.

### 5.1. SYTHON SYNTAX

Sython adds one new syntactical construct to Python. This construct is the use of the dollar sign (\$) to visually distinguish secure variables. The dollar sign goes in front of the variable name. It is required when declaring a new variable (binding a new name), but after that the dollar sign is optional while that variable is still secure. See Table 1 for the complete specification.

**Table 1 Sython Syntax Validity**

Syntax	Validity
<code>\$x=syalloc('i')</code>	Valid syntax. If <code>x</code> is unbound, <code>x</code> will become a secure variable. If <code>x</code> is bound, it will be reassigned as a secure variable.
<code>x=syalloc('i')</code>	Ambiguous syntax. If <code>x</code> is unbound, this will raise a <code>SySecurityException</code> . If <code>x</code> is bound as a secure variable, <code>x</code> will continue to be a secure variable (now containing the results of <code>syalloc('i')</code> ). If <code>x</code> is bound as an insecure variable, a <code>SySecurityException</code> will be raised.
<code>\$x='abc'</code>	Invalid syntax. The dollar sign denotes a secure variable, but the string <code>'abc'</code> is not a secure variable. A <code>SySecurityException</code> will be raised.
<code>\$x=syalloc('s')</code> <code>x='abc'</code>	Valid syntax. <code>x</code> is bound as a secure variable, initially. It is then rebound as an insecure variable containing the string <code>'abc'</code> .

## 5.2. SYTHON BUILT-IN FUNCTIONS

To support development using secure variables, Sython adds a number of new built-in functions. This section will detail each new function.

```
syalloc(type [,range/len, constraint_sets]) → syobj
```

Allocates a new syobj. Variables to hold syobj must be denoted with a dollar sign (\$) when assigning a new syobj.

type is the type of sython variable to allocate, either 'i' for int or 's' for string.

range is a two element tuple containing the valid range of values from which to choose an initial value. range is used with ints and must be a distance of ten or more units apart.

len is the initial length that a str-based syobj should be.

constraint\_sets is a tuple containing strings of constraints.

Valid constraints are:

- 'ALL' for no constraints
- 'EVEN' for only even numbers
- 'ODD' for only odd numbers
- 'POSITIVE' for numbers greater than zero
- 'NEGATIVE' for numbers less than one
- 'UPPER' for uppercase letters
- 'LOWER' for lowercase letters
- 'PUNCT' for punctuation letters
- 'SPACE' for whitespace
- 'DIGITS' for numerical digits

The tuple can combine constraints where they make sense for example:

```
('EVEN','POSITIVE') or ('UPPER','LOWER')
```

Please note: integer-based constraints are restrictive whereas string-based constraints are additive.

Examples:

```
$x=syalloc('i',(0,100),('EVEN',)) #for an even integer between 0 and 100
```

```
$y=syalloc('s',10,('UPPER','SPACE')) #for a string containing uppercase letters and spaces that is 10 letters long.
```

```
$z=syalloc('s',None,('ODD','NEGATIVE')) #for an odd, negative integer.
```

`syval(syobj) → str`

Returns a string representing a sython variable. The string will not change between `syval` calls if the `syobj` value does not change. If the `syobj` value does change, the string returned by `syval` may or may not change.

`syinput() → syobj`

Returns a `syobj` containing a line from the daemon's input file. See Section 5.5 for more information about the input file.

`syoutput(syobj) → bool`

Writes the current value of `syobj` to a new line at the end of the daemon's output file. See Section 5.5 for more information about the output file.

True is returned if writing the output was successful, False otherwise.

`syquery(sql[,syobj, ...]) → [(syobj, ...), (syobj, ...), ...]`

Performs a database query. The `sql` statement should reference `syobj` by either `%i` for `syints` or `%s` for `systrs`.

Examples:

`syquery('select * from users')` might return `[(syobj,syobj,syobj)]` if the table `users` was made up of three columns and contained one row.

`syquery('select * from users where users.id=%i',x)` <where `x` is a secure variable> might return `[]` if the `id` column in the `users` table contained no entry that equaled the value of `x`.

## 5.3. SYTHON OBJECT METHODS

Internally, Sython represents secure variables as Objects of the type `syobj`. These objects have two methods in addition to support for various arithmetic operations. Note: Sython Objects can be handles for both integers and strings in the Sython Daemon.

`syobj.isstr() → bool`

Returns True if this `syobj` represents a string in the Sython Daemon.

`syobj.isint() → bool`

Returns True if this `syobj` represents an integer in the Sython Daemon.

Sython Objects support the following arithmetic operations: addition, subtraction, multiplication, division, modulus, negation and absolute value. These operations are only supported with the Sython Object represents an integer. Operations can be mixed between two Sython Objects (representing integers) or one Sython Object (representing an integer) and one integer (or long). All arithmetic operations that support Sython Objects produce additional Sython Objects. Please note: unlike in division between two integers when dividing by zero, no `ZeroDivisionError` is raised. Instead  $x/0 \rightarrow 0$  (for all values  $x$ ). This is done to protect the actual values of sythond variables. When a string is represented, Sython Objects support the concatenation and length operations.

## 5.4. SYTHON DAEMON CONFIGURATION OPTIONS

The Sython Daemon supports a number of configuration options. These options are specified as command line arguments when launching the daemon. Table 2 details each command line argument. Note: there is no specific command used to exit the Sython Daemon. This unix command `kill` can be used when sythond is launched in non interactive mode.

**Table 2 Sython Daemon Command Line Arguments**

Command Line Argument	Argument Description
<code>-b &lt;file&gt;</code>	Specify an alternative SQLite database filename. See section 5.5 for more information about the database file.
<code>-c</code>	Launch sythond in interactive console mode. To exit sythond when launched in this mode, type <code>control-c</code> .
<code>-d</code>	Enable debugging mode logging. The log file (and console if in console mode) will display additional information about each sythond request.

Command Line Argument	Argument Description
-g	Enable global client support. Without this argument, sythond will only allow communications between clients on the localhost.
-i <file>	Specify alternative input filename. The default input filename is sydinput.txt
-p <port>	Specify alternative port number. The default port number is 60123.
-o <file>	Specify alternative output filename. The default output filename is sydoutput.txt

## 5.5. SYTHON DAEMON FILES

The Sython Daemon interacts with four external files. This section details the use and format of each file. Section 5.4 details how to specify alternative files. By default each file is searched for within the same directory as the `sythond` application. System administrators should take precautions to limit access to all of these files as they can contain the actual values of secure variables.

**Table 3 Sython Daemon Log File**

File:	Sython Daemon Log File
Default filename:	<code>syd.log</code> Note: This filename cannot be changed at <code>sythond</code> runtime.
Purpose:	This file keeps a record of events in the Sython Daemon. By default only new client sessions are logged, but if debugging is turned on, each client request will be logged.
File format:	<DATE> <TIME> <INFORMATION LEVEL> <MESSAGE>



**Table 4 Sython Daemon Database File**

File:	Sython Daemon Database File
Default filename:	syd.db
Purpose:	This is a SQLite database file. It contains the database that is queried in the <code>syquery()</code> command.
File format:	See the SQLite reference manual for more information about the file format.

**Table 5 Sython Daemon Input Text File**

File:	Sython Daemon Input Text File
Default filename:	sydinput.txt
Purpose:	This file is used in coordination with the function <code>syinput()</code> . It is used to securely input fixed data into Sython secure variables. The <code>syinput()</code> function together with this file may be used to simulate user input during testing.
File format:	<code>syinput()</code> reads one line of this file with each call. # preceded comments are supported and will not be included in the <code>syinput()</code> results. Example: #File start 1234 hello this is a string#that was a string, this is a comment

**Table 6 Sython Daemon Output Text File**

File:	Sython Daemon Output Text File
Default filename:	<code>sydoutput.txt</code>
Purpose:	This file is used in coordination with the function <code>syoutput()</code> . It is used to output the values of secure variables. The <code>syoutput()</code> function together with this file may be used to simulate screen output during testing.
File format:	The file is formatted into two line groups in this structure: <code>#&lt;Date&gt; &lt;Time&gt;</code> <code>&lt;variable value&gt;</code>

## 6. MAINTAINANCE

Because Sython is derived from Python 2.3.4, it needs to be maintained similarly to regular Python installations. The big difference is that Python patches should not be directly applied to the Sython source code as they may inadvertently disable Sython functionality. Instead, Sython specific updates which have been tested explicitly with Sython should be used.

If the Sython database functionality has been enabled, care must also be taken when maintaining SQLite. Upgrades to SQLite can be found at <http://www.sql.org>, but should be tested for compatibility with PySQLite.

## 7. TROUBLESHOOTING

This chapter offers solutions to many common problems when working with Sython. Please read through these solutions before contacting the manufacturer.

### 7.1. INSTALLATION

If you experience difficulties during the installation of Sython, please follow this procedure: if you're experiencing difficulties installing SQLite, please visit their website at <http://www.sqlite.org>. The website has answers to various frequently asked questions. There is also a special section dealing with installing SQLite on various architectures including MacOS X at <http://www.sqlite.org/cvstrac/wiki?p=HowToCompile>. Reading the README file that comes with the distribution can also provide installation hints.

If you're experiencing difficulties with installing PySQLite, please visit their website at <http://initd.org/tracker/pysqlite>. In addition the README file in the distribution is useful and points to a more complete usage manual in the doc/rest subdirectory.

If you're experiencing installation difficulties with the Sython distribution, the Python README in the main distribution directory has helpful installation tips. The Python website at <http://www.python.org> also features helpful installation documentation. If you're experiencing difficulties installing the Sython Daemon, you may need to carefully reread the installation directions. Ensure that before running the `install_syd` script your current working directory is the same as that of `sythond`. Additionally administrator privileges are required for installation.

### 7.2. PROGRAMMING PROBLEMS

This section details common difficulties when programming with certain parts of Sython.

One of the main difficulties with the current Sython implementation is the limited support for secure variables as function or method arguments. The obvious syntax of `def f( $x, $y, z )` is not currently supported. Secure variables may still be passed as function or method arguments, but automatic enforcement of secure status is not currently available. Developers are encouraged to use type testing as a temporary workaround.

Developers may notice that division operations never raise a `ZeroDivisionError`. This is an intentional security decision. A malicious user could write a script that would use the `ZeroDivisionError` to discover values of secure integers. Instead `ZeroDivisionErrors` are transparently mapped to the value zero. See Section 5.3 for more information.

For the most part, the Sython Daemon transparently supports multiple simultaneous clients. There are some caveats to this, however. The input and output files are shared between clients and there is only a single file descriptor that is shared. This means that, for example, if an input file contains three lines and client A executes `syinput()`, then client B executes `syinput()` and then finally client A executes `syinput()` again, client A will have variables containing the first and third lines while client B has a variable containing the second line. The other main problem with threading is an internal Python issue. There is only one interpreter thread within each Python process. This generally isn't a problem, but users may notice some slow down when the Sython Daemon is servicing many clients simultaneously.

### 7.3. CONFIGURATION PROBLEMS

By default Sython is configured to communicate with the Sython Daemon running on the same machine over TCP port 60123. This configuration can be changed. If an alternative port is desired, two steps are required. First, when launching `sythond`, the command line argument `-p` and then the alternative port number is required. So to launch `sythond` on port 54321, type in `sythond -p 54321` in the command line and hit Return. Next, the Sython Interpreter needs to be configured to communicate on the alternative port. This is done programmatically. Simply

type `sython.comm.dPort=54321` to change the port number to 54321. This should be done before creating any secure variables. The change is for the current session only.

The steps are similar to use Sython on different machine than the Sython Daemon. On machine A, launch `sythond` using the command line argument `-g`. This enables non-local (or “global”) clients. Then, in the Sython Interpreter on machine B use the command `sython.comm.dHost=machineA.local` where `machineA.local` is the hostname of the machine running the Sython Daemon instance. In addition to host names, IP addresses may be used. Please note: the protocol Sython uses to communicate with the Daemon is not encrypted. As such it may be intercepted by third parties when used over a network. To protect against this attack, users are encouraged to use SSH port tunneling. Please see the SSH man page for more information on port tunneling.

Sometimes when launching the Sython Daemon, an error message will be displayed saying `socket.error (48, 'Address already in use')`. This can happen in two cases. The first, and most obvious, is when there is already a program bound to the port `sythond` is attempting to launch using. To solve this problem exit the other program. This error message can also happen when relaunching `sythond` immediately after exiting the program. In this case, simply wait a few minutes before relaunching `sythond`.