**Documentation for Scithon V0.1.12**

**Requirements**

* Windows 10
* Scilab V6.1.0
* Python V3.8.2 – *Additionally, it must be set as the default one, you can verify its default by typing python in cmd*
* Any and all modules you wish to use inside Scilab must also be installed externally

**Installation**

Extract the zip file and run the “loader.sce” file to load the toolbox.

**Functions**

***startPy*** – Starts the python interpreter, must be called before using any of the other functions offered by Scithon.

***quitPy*** – Stops the python interpreter, erasing all the information\*. Once called, startPy must again be called before using any other functions offered by Scithon.

***py(string code)*** – Executes any simple python code passed and returns the console output.

***pyExec(string filepath) –*** Executes the python file specified by the argument and returns the console output.

***pyGet(string name) –*** Retrieves a named variable previously defined through ***py()*** or ***pyExec()***.

***pyList(arg1, arg2…)*** – Creates and returns a python list containing the arguments passed. Arguments can be of type int, double, string, bool or python variable.

***pyDict(key1, val1, key2, val2…)*** – Creates and returns a python dictionary containing the key-value pairs passed. Arguments can be of type int, double, string, bool or python variable. *Python variables passed for keys must be hashable.*

***pyTuple(arg1, arg2…)*** – Creates and returns a python tuple containing the arguments passed. Arguments can be of type int, double, string, bool or python variable.

***pySet – 2 overloads***

***pySet(PyVar arg) –*** The python variable passed must be a python iterable like list/tuple/set. Creates and returns a python set containing the members of the python iterable passed.

***pySet(arg1, arg2…) –*** Creates and returns a python set containing the arguments passed. Arguments can be of type int, double, string, bool or python variable. *Python variables passed must be hashable.*

***pyEquals(arg1, arg2) –*** Compares the two arguments and returns the result; equivalent to arg1 == arg2. Arguments can be of type int, double, string, bool or python variable.

***pyImport(string module\_name)*** – Imports and returns the python module passed.

**Methods**

Any and all python variable’s methods can be accessed normally\*, as if calling directly from a python interpreter. The method will return what it normally would as a python variable.

**Sequence and Mapping based Data Types**

Members of sequence data types like list and tuple, and dictionary can be accessed\* similar to how one would normally access them on python, only change is that [ ] is replaced with ( ).

***X = pyList(1, 2, 3)***

***Y = pyDict(“hello”, “world”)***

***disp(X(0))***

***disp(Y(“hello”))***

The values of these members can be reassigned by passing the new value as the second argument. The new member will also be returned.

***X(0, 17)*** *//now X = [17, 2, 3]*

***Y(“hello”, 21)*** *//now Y = {“hello”: 21}*

***Y(“world”, 2.1)*** *//now Y = {“hello”: 21, “world”: 2.1}*

**Setting Values For Attributes**

The values of a python variable’s attribute can be reassigned by passing the name of the attribute as a string as the first argument and the new value as the second argument.

***math = pyImport(“math”)***

***math(“pi”, 3)*** *//now math.pi = 3*

**Things that you can do**

* Create all basic python data types and call any and all methods off of them

***X = pyList(1, 2, 3)***

***X.append(10)***

* Import and call any and all builtin python functions.

***Py = pyImport(“builtins”)***

***X = Py.complex(1, 10)***

* Import any module as long as it is installed externallyand use any and all of its functions.

***np = pyImport(“numpy”)***

***X = np.array(pyList(pyList(1, 2, 3), pyList(4, 5, 6)))***

***X = X.reshape(3, 2)***

**Things that you cannot do** (yet)

* Pass keyword arguments.
* Convert python variables into scilab variables.
* Slice python strings/lists/tuples.

**Known issues**

* S̶o̶m̶e̶t̶i̶m̶e̶s̶,̶ ̶u̶n̶s̶u̶c̶c̶e̶s̶s̶f̶u̶l̶l̶y̶ ̶i̶m̶p̶o̶r̶t̶i̶n̶g̶ ̶a̶ ̶m̶o̶d̶u̶l̶e̶ ̶d̶o̶e̶s̶n̶’̶t̶ ̶g̶i̶v̶e̶ ̶a̶n̶y̶ ̶e̶r̶r̶o̶r̶s̶ ̶a̶n̶d̶ ̶r̶e̶s̶u̶l̶t̶s̶ ̶i̶n̶ ̶a̶ ̶c̶r̶a̶s̶h̶. Not able to recreate this issue as of 0.1.11
* quitPy doesn’t actually clear the data, they can be accessed again after calling startPy. The cause of this behavior is the inconsistent reference counting, will get it fixed towards the end of development.
* ̶P̶a̶s̶s̶i̶n̶g̶ ̶s̶c̶i̶l̶a̶b̶ ̶a̶r̶r̶a̶y̶ ̶w̶i̶l̶l̶ ̶n̶o̶t̶ ̶t̶h̶r̶o̶w̶ ̶a̶ ̶t̶y̶p̶e̶ ̶e̶r̶r̶o̶r̶ ̶a̶s̶ ̶i̶n̶t̶e̶n̶d̶e̶d̶,̶ ̶a̶n̶d̶ ̶w̶o̶u̶l̶d̶ ̶i̶n̶s̶t̶e̶a̶d̶ ̶t̶a̶k̶e̶ ̶t̶h̶e̶ ̶f̶i̶r̶s̶t̶ ̶m̶e̶m̶b̶e̶r̶ ̶o̶f̶ ̶t̶h̶e̶ ̶a̶r̶r̶a̶y̶.̶ ̶T̶h̶i̶s̶ ̶i̶s̶ ̶d̶u̶e̶ ̶t̶o̶ ̶s̶c̶i̶l̶a̶b̶ ̶a̶s̶s̶o̶c̶i̶a̶t̶i̶n̶g̶ ̶t̶y̶p̶e̶s̶ ̶t̶o̶ ̶t̶h̶e̶ ̶a̶r̶r̶a̶y̶s̶ ̶t̶h̶e̶m̶s̶e̶l̶v̶e̶s̶,̶ ̶s̶h̶o̶u̶l̶d̶ ̶b̶e̶ ̶a̶n̶ ̶e̶a̶s̶y̶ ̶f̶i̶x̶. Fixed as of 0.1.11
* ̶O̶n̶c̶e̶ ̶a̶n̶ ̶e̶r̶r̶o̶r̶ ̶h̶a̶s̶ ̶b̶e̶e̶n̶ ̶e̶n̶c̶o̶u̶n̶t̶e̶r̶e̶d̶ ̶b̶y̶ ̶p̶a̶s̶s̶i̶n̶g̶ ̶a̶n̶ ̶u̶n̶h̶a̶s̶h̶a̶b̶l̶e̶ ̶v̶a̶l̶u̶e̶ ̶f̶o̶r̶ ̶a̶ ̶d̶i̶c̶t̶i̶o̶n̶a̶r̶y̶ ̶k̶e̶y̶ ̶o̶r̶ ̶a̶ ̶s̶e̶t̶,̶ ̶t̶h̶e̶ ̶i̶n̶t̶e̶r̶p̶r̶e̶t̶e̶r̶ ̶t̶h̶r̶o̶w̶s̶ ̶e̶r̶r̶o̶r̶s̶ ̶f̶o̶r̶ ̶a̶l̶l̶ ̶m̶e̶t̶h̶o̶d̶s̶ ̶c̶a̶l̶l̶e̶d̶.̶ ̶R̶e̶s̶e̶t̶t̶i̶n̶g̶ ̶t̶h̶e̶ ̶i̶n̶t̶e̶r̶p̶r̶e̶t̶e̶r̶ ̶b̶y̶ ̶c̶a̶l̶l̶i̶n̶g̶ ̶q̶u̶i̶t̶P̶y̶ ̶f̶o̶l̶l̶o̶w̶e̶d̶ ̶b̶y̶ ̶s̶t̶a̶r̶t̶P̶y̶ ̶c̶a̶n̶ ̶b̶e̶ ̶u̶s̶e̶d̶ ̶a̶s̶ ̶a̶ ̶t̶e̶m̶p̶o̶r̶a̶r̶y̶ ̶f̶i̶x̶. Fixed as of 1.0.0