Distinguishing Python

Kristine Middlemiss, *Senior Developer Consultant*
Autodesk Developer Network (ADN)

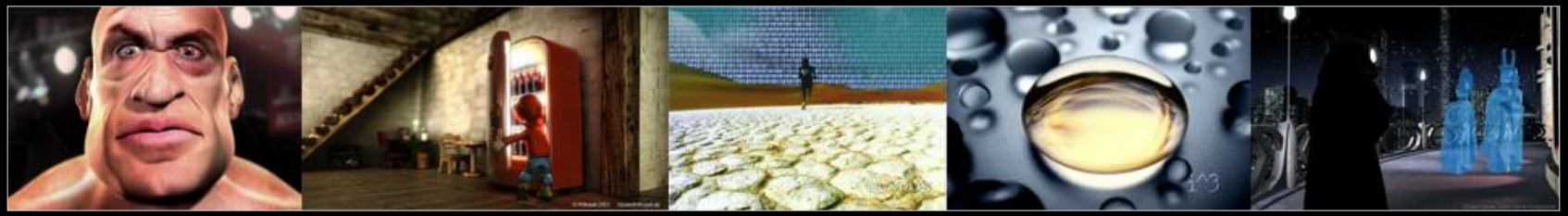
Autodesk

Session Agenda

- Working with Modules
- Utilities functions
- Maya Standalone
- Python and MEL Scripting

Working with Modules

Navigating the Maya Python namespace



Autodesk®

Maya Python Modules

maya	Top-level module
maya.cmds	Maya commands (e.g. sphere, getAttr) and plug-in commands
maya.OpenMaya*	OpenMaya API modules
maya.utils	Utilities not specific to API or Maya commands
maya.standalone	Initialization routine for standalone Python
maya.app	Python code used to implement Maya
maya.mel.eval()	Runs MEL commands as a string
maya.stringTable	This is the strings used to built the UI, for localization

Python Modules

- All Python functionality in contained in modules
- Modules may have hierarchy (e.g os.path)
- Import modules to access them

```
import maya.cmds
maya.cmds.sphere()

import maya.cmds as cmds
cmds.sphere()

from maya.cmds import *
sphere()
```

Importing Modules

There are three forms of import:

1. Import a module using full namespace
2. Import a module using full namespace, but choose a new name
3. Import an item from anther module into your global namespace

Using Modules

- To use a module in your code, use “import”
- “import” adds a module to the namespace of your code
- “import” searches sys.path
- Similar to “source” in MEL

Script Modules

- Module is used to define closely related functionality
  - Use Python "import" command to bring in modules

```
import myModule
myModule.myFunction()
# Result: 1 #
```

  - reload" command updates module:

```
reload( myModule )
```

  - Module must be available in path

Python Path

- PYTHONPATH environment variable
  - Read when Python initializes
  - Stored in sys.path
- sys.path
  - Can be modified after Python is initialized

Scripts Path

- **PYTHONPATH** defines search path for scripts
- Includes standard Maya script paths
- Update at startup via Maya.env file
- Update from Python:

```
import sys
sys.path.append( "C:/scripts" )
```

Maya Environment Variable

- Text file: Maya.env
- Recommended approach of setting environment variable
- Syntax:
  - PYTHONPATH = C:\Users\wengn\Documents\test
- Save it to
  - C:\Users\wengn\Documents\maya\2012\Maya.env
-

Config File

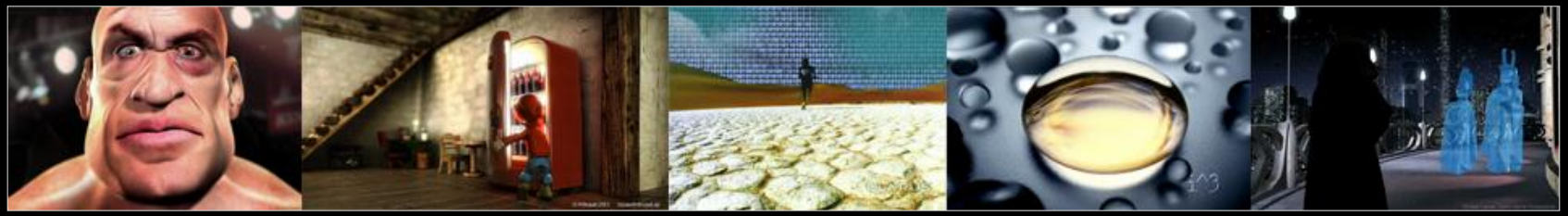
- Create a userConfig.py file and add it to somewhere on your path (PYTHONPATH)
- You can add imports and whatever
- Changes will show up in top level context

```
import maya.cmds as cmds
```

Other Useful Modules (externally)

os	**process management, environment variables, etc**
os.path	file path routines: join, split, etc.
sys	shell arguments (argv), python globals
re	regular expressions
urllib2	URL access
pickle	object archiving
getopt	command line argument parsing
math	assorted math routines and constants

Utility Functions



Autodesk®

Utility Functions

- maya.utils
  - Utilities that are not API or Command specific are located in maya.utils
  - Only two functions there so far
    1. maya.utils.executeInMainThreadWithResult()
    2. maya.utils.processIdleEvents()

Utility Functions

- maya.utils.executeInMainThreadWithResult
  - executeInMainThreadWithResult allows other threads to call Maya routines which are not thread safe
  - It only works from the GUI app as it uses idle events
  - Thread blocks until main thread completes execution
  - Any result or exception is passed to the calling thread

Utility Functions

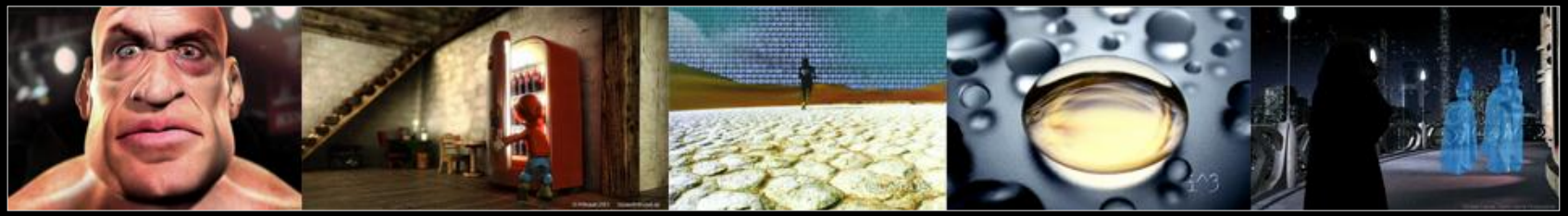
- maya.utils.executeInMainThreadWithResult
  - executeInMainThreadWithResult can accept either a string to evaluate or a Python function

```
def myProc():
        return maya.cmds.ls()
res = maya.utils.executeInMainThread( myProc )
```

Utility Functions

- maya.utils.processIdleEvents
  - maya.utils.processIdleEvents() is mainly used for testing
  - It tells the idle queue to process existing idle events
  - Returns True if all items on the idle queue were processed

Maya Standalone



Autodesk®

Importing Maya into external interpreter

- Maya modules can be used within an external Python interpreter as long as:
  - version numbers match well enough
  - environment is correct
- Maya ships with a “python” command in the bin directory of the distribution
- Included “python” command sets up correct environment and runs Maya’s version of Python

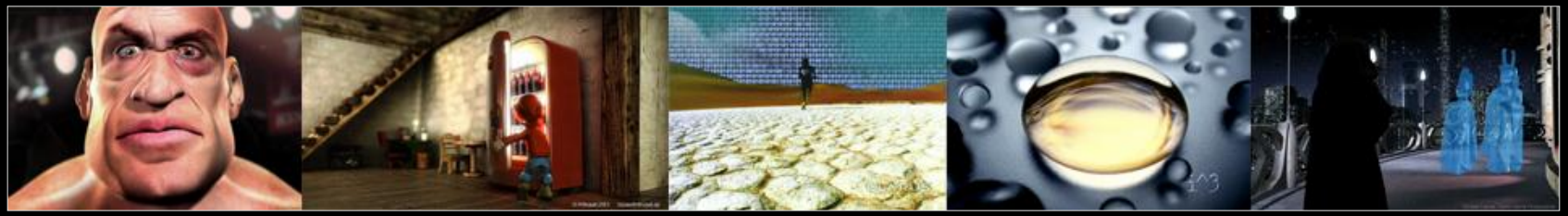
Initialization

- Using Maya in a standalone Python interpreter is similar to writing a library mode app using the API
- Maya must be initialized before Maya's Python APIs are used
- Initialization is done as follows

```
import maya.standalone
maya.standalone.initialize( name='python' )
```

- Initialization takes a while to run

Python & MEL Scripting



Autodesk®

Maya Python Scripts

- All "Maya Commands" (previously called "MEL Commands") now accessible from Python

- Exposed through the maya.cmds module:

MEL:

```
select pCube1;
ls –sl;
```

Python:

```
import maya.cmds
maya.cmds.select( “pCube1“ )
maya.cmds.ls( sl=True )
```


Autodesk®

Python vs. MEL

- Use a good text editor and don’t mix tabs and spaces
  - Tabs are 8 spaces by default
- You must explicitly import all external code
  - No auto finding of global functions
  - Global scope has a different definition in both languages
- All object names must be quoted
- Booleans in Python are “True” and “False” capitalized
- Importing a module again does not reload it
  - Use reload command instead (i.e. reload(maya.test.UtilsTest))
- Use PYTHONPATH instead of MAYA_SCRIPT_PATH

Interoperability between scripting languages

- Python can call MEL and MEL can call Python
- New Python scripts must hook into existing MEL-based UI

MEL to Python

- MEL has a new “python” command that can be used to call into Python
- Takes a string containing Python code
- Returns the result of the computation
  - Basic Python types are converted into matching MEL type (e.g.strings, ints, lists of ints, etc.)
  - All other types are converted into their string representation

```
import myModule
$result = python( “myModule.myMelProc()” )
```

Python to MEL

- Python has maya.mel() function, which is possibly
- moving to maya.cmds.mel()
- Takes a string containing MEL code
- Returns the result of the computation
- All MEL return values are converted into a logical Python value

```
result = maya.mel( “myMelProc()” )
```

Python/MEL communication

- Invoke MEL from Python:

```
import maya.mel
maya.mel.eval( "ls -sl" )
# Result: ['pSphere1'] #
```

- Invoke Python from MEL:

“python” command

```
string $foo[] = python( "['a','b','c']" );
size($foo);
// Result: 3 //
```

Python/MEL communication

- Accessing Python information from MEL

  Python:

```
class MyFile:
        _f = "mytestfile.txt"

mf = MyFile()
```

  MEL:

```
string $f = python("mf._f");
print $f;
```

Autodesk