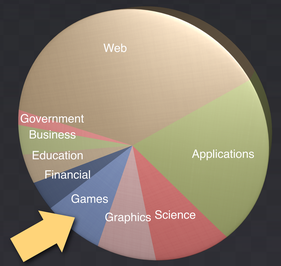
Thinking "Pythonically"

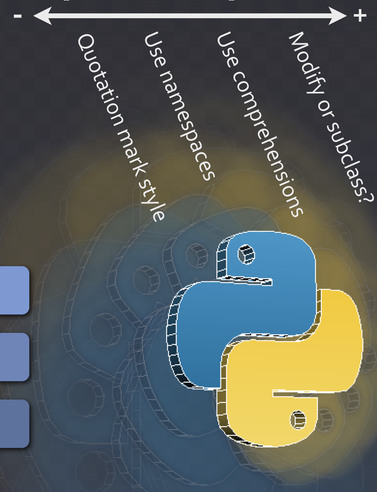
- Variety of users/cultures
  - No "one-size-fits-all"
- Readability is key
- "When in doubt, use your best judgment"
  - Learn about Python and its intricacies
  - Feel confident when you need to break rules
- How to balance idiomatic Python with Maya?
  - Forget MEL patterns
  - DO understand Maya's architecture
  - Where/when/how does your code execute?

Python Community*

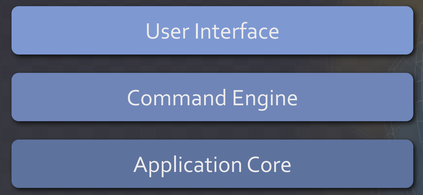


*not intended to be factual data

Spectrum of Importance



Maya Architecture



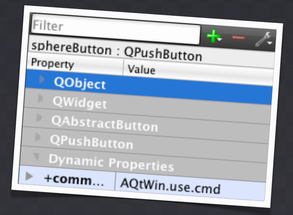
Surviving A Stringly-Typed World

```
# using command string is bad
class AWin(object):
  handle = 'a_window'
  def __init__(self):
    if cmds.window(AWin.handle, ex=True):
      cmds.deleteUI(AWin.handle)
    self.win = cmds.window(AWin.handle)
    cmds.columnLayout()
    # don't do this!
    cmds.button('A Button',
      c=('import maya.cmds;'+
      'maya.cmds.polySphere()'))
    cmds.showWindow(self.win)

# using a function pointer is better
class AWin(object):
  handle = 'a_window'
  def __init__(self):
    if cmds.window(AWin.handle, ex=True):
      cmds.deleteUI(AWin.handle)
    self.win = cmds.window(AWin.handle)
    cmds.columnLayout()
    # do this
    cmds.button('A Button', c=self.cmd)
    cmds.showWindow(self.win)
  def cmd(self, *args):
    cmds.polySphere()
```

- Minimize assumptions about what is in __main__
- Unfortunately, loadUI must link commands in __main__
- Class and instance attributes are not the same!
  - Class attribute can be used like a singleton

- The world of MEL is polluted with global variables
- Every time you glue Python code together with strings in __main__, a kitten dies
- Prefer function pointers to command strings




```
class AQtWin(object):
  # use like a singleton
  use = None
  handle = 'a_qt_window'
  def __init__(self, ui_file):
    AQtWin.use = self
    if cmds.window(AQtWin.handle,
        ex=True):
      cmds.deleteUI(AQtWin.handle)
    self.win = cmds.loadUI(
      uiFile=ui_file)
    cmds.showWindow(self.win)
  def cmd(self, *args):
    cmds.polySphere()
```

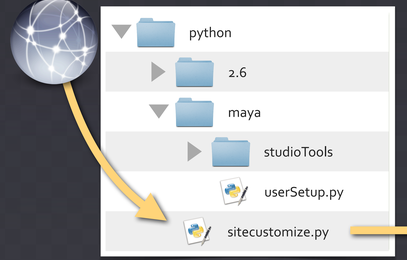
Plug-Ins and Deployment

- Devkit examples
  - Not organized via class attributes
  - Require lots of files in different places
- Single entry point?
  - Patch plug-in path
  - loadPlugin on self
  - Put classes in other modules
  - Can import or deploy in a package

1. Systemwide PYTHONPATH

```
setenv PYTHONPATH /Network/python
```

2. Studio sitecustomize & userSetup on Network



3. Add Needed Paths

```
folder = os.path.dirname(__file__)
# path for major point release lib
sys.path.append(
  os.path.join(
    folder, sys.version[:3]))
# custom locations if using Maya
if 'maya' in sys.executable:
  sys.path.append(
    os.path.join(folder, 'maya'))
```

```
kEnvVar = 'MAYA_PLUG_IN_PATH'
def appendToPluginPath():
  path = os.environ[kEnvVar]
  folder = os.path.dirname(__file__)
  if not folder in path:
    path = os.pathsep.join(
      [path, folder])
  os.environ[kEnvVar] = path
def pluginFileName():
  return '%s.py'%(
    os.path.splitext(
      os.path.basename(__file__))[0]
# import classes from modules
import classes.pluginOne
import classes.pluginTwo
def initializePlugin(mobject):
  # create fn, register plugins
  return
def uninitializePlugin(mobject):
  # create fn, deregister plugins
  return
# load plug-ins on module import
try:
    appendToPluginPath()
    cmds.loadPlugin(
        getPluginFileName())
# __file__ only exists when importing
except NameError: pass
```