

## nodebox/\_\_\_init\_\_\_py

```
__version__='1.9.29'
```

```
def get_version():  
    return __version__
```

## nodebox/console.py

```
import AppKit  
NSApplication = AppKit.NSApplication
```

```
try:
```

```
5     import nodebox  
except ImportError:  
    import sys, os  
    nodebox_dir = os.path.dirname(os.path.abspath(__file__))  
    sys.path.append(os.path.dirname(nodebox_dir))
```

```
10 import nodebox.graphics  
graphics = nodebox.graphics
```

```
import nodebox.util  
15 util = nodebox.util
```

```
#from nodebox import graphics  
#from nodebox import util
```

```
20 class NodeBoxRunner(object):
```

```
    def __init__(self):  
        # Force NSApp initialisation.  
        NSApplication.sharedApplication().activateIgnoringOtherApps_(0)  
25     self.namespace = {}  
        self.canvas = graphics.Canvas()  
        self.context = graphics.Context(self.canvas, self.namespace)  
        self.__doc__ = {}  
        self._pageNumber = 1  
30     self.frame = 1
```

```
    def _check_animation(self):  
        """Returns False if this is not an animation, True otherwise.  
        Throws an exception if the animation is not correct (missing a draw method)."""  
35     if self.canvas.speed is not None:  
         if not self.namespace.has_key('draw'):  
             raise graphics.NodeBoxError('Not a correct animation: No draw() method.')  
         return True  
    return False
```

```
40  
    def run(self, source_or_code):  
        self._initNamespace()  
        if isinstance(source_or_code, basestring):  
            source_or_code = compile(source_or_code + "\n\n", "<Untitled>", "exec")  
45     exec source_or_code in self.namespace  
    if self._check_animation():  
        if self.namespace.has_key('setup'):  
            self.namespace['setup']()  
            self.namespace['draw']()
```

```
50  
    def run_multiple(self, source_or_code, frames):  
        if isinstance(source_or_code, basestring):  
            source_or_code = compile(source_or_code + "\n\n", "<Untitled>", "exec")
```

```

55     # First frame is special:
        self.run(source_or_code)
        yield 1
        animation = self._check_animation()

60     for i in range(frames-1):
        self.canvas.clear()
        self.frame = i + 2
        self.namespace["PAGENUM"] = self.namespace["FRAME"] = self.frame
        if animation:
85             self.namespace['draw']()
        else:
            exec source_or_code in self.namespace
        yield self.frame

70     def _initNamespace(self, frame=1):
        self.canvas.clear()
        self.namespace.clear()
        # Add everything from the namespace
        for name in graphics.__all__:
75             self.namespace[name] = getattr(graphics, name)
        for name in util.__all__:
            self.namespace[name] = getattr(util, name)
        # Add everything from the context object
        self.namespace["_ctx"] = self.context
80     for attrName in dir(self.context):
        self.namespace[attrName] = getattr(self.context, attrName)
        # Add the document global
        self.namespace["__doc__"] = self.__doc__
        # Add the frame
85     self.frame = frame
        self.namespace["PAGENUM"] = self.namespace["FRAME"] = self.frame

def make_image(source_or_code, outputfile):

90     """"Given a source string or code object, executes the scripts and saves the result as
an image. Supported image extensions: pdf, tiff, png, jpg, gif""""

        runner = NodeBoxRunner()
        runner.run(source_or_code)
95     runner.canvas.save(outputfile)

def make_movie(source_or_code, outputfile, frames, fps=30):

100     """"Given a source string or code object, executes the scripts and saves the result as
a movie.

        You also have to specify the number of frames to render.
        Supported movie extension: mov"""

105     from nodebox.util import QTSupport
        runner = NodeBoxRunner()
        movie = QTSupport.Movie(outputfile, fps)
        for frame in runner.run_multiple(source_or_code, frames):
            movie.add(runner.canvas)
110     movie.save()

def usage(err=""):
    if len(err) > 0:
        err = '\n\nError: ' + str(err)
115     print """"NodeBox console runner
Usage: console.py sourcefile imagefile
or: console.py sourcefile moviefile number_of_frames [fps]
Supported image extensions: pdf, tiff, png, jpg, gif

```

```

Supported movie extension:  mov"" + err
120
def main():
    import sys, os
    if len(sys.argv) < 2:
        usage()
125     elif len(sys.argv) == 3: # Should be an image
        basename, ext = os.path.splitext(sys.argv[2])
        if ext not in ('.pdf', '.gif', '.jpg', '.jpeg', '.png', '.tiff'):
            return usage('This is not a supported image format.')
        make_image(open(sys.argv[1]).read(), sys.argv[2])
130     elif len(sys.argv) == 4 or len(sys.argv) == 5: # Should be a movie
        basename, ext = os.path.splitext(sys.argv[2])
        if ext != '.mov':
            return usage('This is not a supported movie format.')
        if len(sys.argv) == 5:
135             try:
                fps = int(sys.argv[4])
            except ValueError:
                return usage()
        else:
140             fps = 30
        make_movie(open(sys.argv[1]).read(), sys.argv[2], int(sys.argv[3]), fps)

def test():
    # Creating the NodeBoxRunner class directly:
145     runner = NodeBoxRunner()
    testscript = ('size(500,500)\n'
                  'for i in range(400):\n'
                  '    oval(random(WIDTH),random(HEIGHT),50,50, '
                  '    fill=(random(), 0,0,random()))')
150     runner.run(testscript)
    runner.canvas.save('console-test.pdf')
    runner.canvas.save('console-test.png')

    # Using the runner for animations:
155     runner = NodeBoxRunner()
    for frame in runner.run_multiple('size(300, 300)\ntext(FRAME, 100, 100)', 10):
        runner.canvas.save('console-test-frame%02i.png' % frame)

    # Using the shortcut functions:
160     make_image('size(200,200)\ntext(FRAME, 100, 100)', 'console-test.gif')
    make_movie('size(200,200)\ntext(FRAME, 100, 100)', 'console-test.mov', 10)

if __name__=='__main__':
    main()

```

## nodebox/PyFontify.py

"""Module to analyze Python source code; for syntax coloring tools.

Interface:

```

for tag, start, end, sublist in fontify(pytext, searchfrom, searchto):
    ...

```

The 'pytext' argument is a string containing Python source code.

The (optional) arguments 'searchfrom' and 'searchto' may contain a slice in pytext.

The returned value is a list of tuples, formatted like this:

```

10 [ ('keyword', 0, 6, None), ('keyword', 11, 17, None), ('comment', 23, 53, None), etc. ]

```

The tuple contents are always like this:

```

(tag, startindex, endindex, sublist)

```

tag is one of 'keyword', 'string', 'comment' or 'identifier'

sublist is not used, hence always None.

```

15 """

    # Based on FontText.py by Mitchell S. Chapman,
    # which was modified by Zachary Roadhouse,
    # then un-Tk'd by Just van Rossum.
20 # Many thanks for regular expression debugging & authoring are due to:
    # Tim (the-incredib-ly y'rs) Peters and Cristian Tismer
    # So, who owns the copyright? ;-) How about this:
    # Copyright 1996-2003:
    # Mitchell S. Chapman,
25 # Zachary Roadhouse,
    # Tim Peters,
    # Just van Rossum

    # from __future__ import generators
30
    __version__ = "0.5"

    import re
    import graphics
35 import util

    from keyword import kwlist as keywordsList
    keywordsList = keywordsList[:]
    keywordsList += ["None", "True", "False"]
40 keywordsList += graphics.__all__
    keywordsList += util.__all__
    keywordsList += dir(graphics.Context)

    # These keywords were not captured somehow
45 keywordsList += ["MOUSEX", "MOUSEY", "mousedown", "keydown", "key", "scrollwheel",
        "wheeldelta", "PAGENUM", "keycode", "FRAME", "canvas"]

    # Build up a regular expression which will match anything
    # interesting, including multi-line triple-quoted strings.
50 commentPat = r"#[^\n]*"

    pat = r"[uU]?[rR]?q[^\q\n]*(\\[\000-\377][^\q\n]*)*q?"
    quotePat = pat.replace("q", "'") + "|" + pat.replace('q', '"')

55 # Way to go, Tim!
    pat = r"""
        [uU]?[rR]?
        qq
        [^\q]*
60     (
        (   \\[\000-\377]
            |   q
              (   \\[\000-\377]
                  |   [^\q]
65                 |   q
                    (   \\[\000-\377]
                        |   [^\q]
                          )
                    )
              )
70         )
        [^\q]*
        )*
        (qq)?
    """
75 pat = "".join(pat.split()) # get rid of whitespace
    tripleQuotePat = pat.replace("q", "'") + "|" + pat.replace('q', '"')

    # Build up a regular expression which matches all and only

```

```

# Python keywords. This will let us skip the uninteresting
80 # identifier references.
keyPat = r"\b(" + "|".join(keywordsList) + r")\b"

matchPat = commentPat + "|" + keyPat + "|(" + tripleQuotePat + "|" + quotePat + ")"
matchRE = re.compile(matchPat)

85 idKeyPat = "[ \t]*([A-Za-z_][A-Za-z_0-9.]*)" # Ident w. leading whitespace.
idRE = re.compile(idKeyPat)
asRE = re.compile(r".*?\b(as)\b")

90 def fontify(pytext, searchfrom=0, searchto=None):
    if searchto is None:
        searchto = len(pytext)
    # Cache a few attributes for quicker reference.
    search = matchRE.search
    95 idMatch = idRE.match
    asMatch = asRE.match

    commentTag = 'comment'
    stringTag = 'string'
    100 keywordTag = 'keyword'
    identifierTag = 'identifier'

    start = 0
    end = searchfrom
    105 while 1:
        m = search(pytext, end)
        if m is None:
            break # EXIT LOOP
        if start >= searchto:
            break # EXIT LOOP
        110 keyword = m.group(1)
        if keyword is not None:
            # matched a keyword
            start, end = m.span(1)
            115 yield keywordTag, start, end, None
            if keyword in ["def", "class"]:
                # If this was a defining keyword, color the
                # following identifier.
                m = idMatch(pytext, end)
                120 if m is not None:
                    start, end = m.span(1)
                    yield identifierTag, start, end, None
            elif keyword == "import":
                # color all the "as" words on same line;
                # cheap approximation to the truth
                while 1:
                    m = asMatch(pytext, end)
                    if not m:
                        break
                    130 start, end = m.span(1)
                    yield keywordTag, start, end, None
            elif m.group(0)[0] == "#":
                start, end = m.span()
                yield commentTag, start, end, None
            135 else:
                start, end = m.span()
                yield stringTag, start, end, None

    def test(path):
        140 f = open(path)
        text = f.read()
        f.close()

```

```

    for tag, start, end, sublist in fontify(text):
        print tag, repr(text[start:end])
145
if __name__ == "__main__":
    import sys
    test(sys.argv[1])

```

### nodebox/geo/\_\_init\_\_.py

```

# Geometric functionality

import math

5 try:
    # Faster C versions.
    import cGeo
    isqrt = inverse_sqrt = cGeo.fast_inverse_sqrt
    angle = cGeo.angle
10    distance = cGeo.distance
    coordinates = cGeo.coordinates

except ImportError:
    def inverse_sqrt(x):
15        return 1.0 / math.sqrt(x)

    isqrt = inverse_sqrt

    def angle(x0, y0, x1, y1):
20        return math.degrees( math.atan2(y1-y0, x1-x0) )

    def distance(x0, y0, x1, y1):
        return math.sqrt(math.pow(x1-x0, 2) + math.pow(y1-y0, 2))

25    def coordinates(x0, y0, distance, angle):
        x1 = x0 + math.cos(math.radians(angle)) * distance
        y1 = y0 + math.sin(math.radians(angle)) * distance
        return x1, y1

30 def reflect(x0, y0, x1, y1, d=1.0, a=180):
    d *= distance(x0, y0, x1, y1)
    a += angle(x0, y0, x1, y1)
    x, y = coordinates(x0, y0, d, a)
    return x, y

```

### nodebox/geo/pathmatics.py

```

from math import sqrt, pow

# from nodebox.geo import distance

5 def linepoint(t, x0, y0, x1, y1):

    """Returns coordinates for point at t on the line.

    Calculates the coordinates of x and y for a point
10    at t on a straight line.

    The t parameter is a number between 0.0 and 1.0,
    x0 and y0 define the starting point of the line,
    x1 and y1 the ending point of the line,

15    """

```

```

    out_x = x0 + t * (x1-x0)
    out_y = y0 + t * (y1-y0)
20     return (out_x, out_y)

def linelength(x0, y0, x1, y1):

    """Returns the length of the line."""
25     #return distance(x0,y0, x1,y1)

    # fastest
    return math.sqrt((x1-x0)**2 + (y1-y0)**2)
    #a = pow(abs(x0 - x1), 2)
30    #b = pow(abs(y0 - y1), 2)
    #return sqrt(a+b)

def curvepoint(t, x0, y0, x1, y1, x2, y2, x3, y3, handles=False):

35    """Returns coordinates for point at t on the spline.

    Calculates the coordinates of x and y for a point
    at t on the cubic bezier spline, and its control points,
    based on the de Casteljau interpolation algorithm.
40

    The t parameter is a number between 0.0 and 1.0,
    x0 and y0 define the starting point of the spline,
    x1 and y1 its control point,
    x3 and y3 the ending point of the spline,
45    x2 and y2 its control point.

    If the handles parameter is set,
    returns not only the point at t,
    but the modified control points of p0 and p3
    should this point split the path as well.
50    """

    mint = 1 - t

55    x01 = x0 * mint + x1 * t
    y01 = y0 * mint + y1 * t
    x12 = x1 * mint + x2 * t
    y12 = y1 * mint + y2 * t
    x23 = x2 * mint + x3 * t
60    y23 = y2 * mint + y3 * t

    out_c1x = x01 * mint + x12 * t
    out_c1y = y01 * mint + y12 * t
    out_c2x = x12 * mint + x23 * t
65    out_c2y = y12 * mint + y23 * t
    out_x = out_c1x * mint + out_c2x * t
    out_y = out_c1y * mint + out_c2y * t

    if not handles:
70         return (out_x, out_y, out_c1x, out_c1y, out_c2x, out_c2y)
    else:
        return (out_x, out_y, out_c1x, out_c1y, out_c2x, out_c2y, x01, y01, x23, y23)

def curvelength(x0, y0, x1, y1, x2, y2, x3, y3, n=20):
75

    """Returns the length of the spline.

    Integrates the estimated length of the cubic bezier spline
    defined by x0, y0, ... x3, y3, by adding the lengths of
80    linear lines between points at t.

```

```

    The number of points is defined by n
    (n=10 would add the lengths of lines between 0.0 and 0.1,
    between 0.1 and 0.2, and so on).
85
    The default n=20 is fine for most cases, usually
    resulting in a deviation of less than 0.01.
    """

90    length = 0
    xi = x0
    yi = y0

    for i in range(n):
95        t = 1.0 * (i+1) / n
        pt_x, pt_y, pt_clx, pt_cly, pt_c2x, pt_c2y = curvepoint(t, x0, y0,
                                                                    x1, y1,
                                                                    x2, y2,
                                                                    x3, y3)

100        # TBD: replace distance calculation
        c = sqrt(pow(abs(xi-pt_x),2) + pow(abs(yi-pt_y),2))
        length += c
        xi = pt_x
        yi = pt_y

105    return length

```

## nodebox/graphics/\_\_init\_\_.py

```

# import pdb
import cocoa
graphics_impl = cocoa

5 import AppKit

# I really dont like it but cocoa.py has an __all__...
from cocoa import *

10 # from nodebox.util import _copy_attr, _copy_attrs
import nodebox.util
_copy_attr = nodebox.util._copy_attr
_copy_attrs = nodebox.util._copy_attrs

15 import nodebox.geo

# add graphics commands from cocoa
__all__ = list(graphics_impl.__all__)
__all__.extend(['Context'])

20
class Context(object):

    KEY_UP = graphics_impl.KEY_UP
    KEY_DOWN = graphics_impl.KEY_DOWN
25    KEY_LEFT = graphics_impl.KEY_LEFT
    KEY_RIGHT = graphics_impl.KEY_RIGHT
    KEY_BACKSPACE = graphics_impl.KEY_BACKSPACE
    KEY_TAB = graphics_impl.KEY_TAB
    KEY_ESC = graphics_impl.KEY_ESC

30
    NORMAL = graphics_impl.NORMAL
    FORTYFIVE = graphics_impl.FORTYFIVE

    def __init__(self, canvas=None, ns=None):

```



```

35         """Initializes the context.

        Note that we have to give the namespace of the executing script,
        which is a hack to keep the WIDTH and HEIGHT properties updated.
40         Python's getattr only looks up property values once: at assign time."""

        if canvas is None:
            canvas = Canvas()
        if ns is None:
45             ns = {}
        self.canvas = canvas
        self._ns = ns
        self._imagecache = {}
        self._vars = []
50         self._resetContext()

    def _resetContext(self):
        self._outputmode = RGB
        self._colormode = RGB
55         self._colorrange = 1.0
        self._fillcolor = self.Color()
        self._strokecolor = None
        self._strokewidth = 1.0
        self._capstyle = BUTT
60         self._joinstyle = MITER
        self.canvas.background = self.Color(1.0)
        self._path = None
        self._autoclosepath = True
        self._transform = Transform()
65         self._transformmode = CENTER
        self._transformstack = []
        self._fontname = "Helvetica"
        self._fontsize = 24
        self._lineheight = 1.2
70         self._align = LEFT
        self._noImagesHint = False
        self._oldvars = self._vars
        self._vars = []

75     def ximport(self, libName):

        lib = __import__(libName)
        self._ns[libName] = lib
        lib._ctx = self
80         return lib

    ### Setup methods ###

    def size(self, width, height):
85         if width == 0 and height == 0:
            # set to main screen size
            allsc = AppKit.NSScreen.screens()
            mainscreen = allsc[0]
            mainframe = mainscreen.frame()
90             width = mainframe.size.width
            height = mainframe.size.height

        self.canvas.width = width
        self.canvas.height = height
95         self._ns["WIDTH"] = width
        self._ns["HEIGHT"] = height

    def _get_width(self):

```

```

    return self.canvas.width
100 WIDTH = property(_get_width)

    def _get_height(self):
        return self.canvas.height
105 HEIGHT = property(_get_height)

    def speed(self, speed):
        self.canvas.speed = speed
110

    def background(self, *args):
        if len(args) > 0:
            if len(args) == 1 and args[0] is None:
                self.canvas.background = None
115             else:
                self.canvas.background = self.Color(args)
            return self.canvas.background

    def outputmode(self, mode=None):
120         if mode is not None:
            self._outputmode = mode
        return self._outputmode

    ### Variables ###
125

    def var(self, name, type,
            default=None, min=0, max=100, value=None,
            handler=None, menuitems=None):
        # pdb.set_trace()
130         v = Variable(name, type, default, min, max, value, handler, menuitems)
        self.addvar(v)
        return v

    def addvar(self, v):
135         oldvar = self.findvar(v.name)
        if oldvar is not None:
            if oldvar.compliesTo(v):
                v.value = oldvar.value
            self._vars.append(v)
140         self._ns[v.name] = v.value

    def findvar(self, name):
        for v in self._oldvars:
            if v.name == name:
145                 return v
        return None

    ### Objects ###

150    def _makeInstance(self, clazz, args, kwargs):
        """Creates an instance of a class defined in this document.
        This method sets the context of the object to the current context."""
        inst = clazz(self, *args, **kwargs)
        return inst
155

    def BezierPath(self, *args, **kwargs):
        return self._makeInstance(BezierPath, args, kwargs)

    def ClippingPath(self, *args, **kwargs):
160         return self._makeInstance(ClippingPath, args, kwargs)

    def Rect(self, *args, **kwargs):

```

```

        return self._makeInstance(Rect, args, kwargs)

165     def Oval(self, *args, **kwargs):
        return self._makeInstance(Oval, args, kwargs)

    def Color(self, *args, **kwargs):
        return self._makeInstance(Color, args, kwargs)
170
    def Image(self, *args, **kwargs):
        return self._makeInstance(Image, args, kwargs)

    def Text(self, *args, **kwargs):
175         return self._makeInstance(Text, args, kwargs)

    ### Primitives ###

    def rect(self, x, y, width, height, roundness=0.0, draw=True, **kwargs):
180         BezierPath.checkKwargs(kwargs)
        p = self.BezierPath(**kwargs)
        if roundness == 0:
            p.rect(x, y, width, height)
        else:
185             curve = min(width*roundness, height*roundness)
            p.moveto(x, y+curve)
            p.curveto(x, y, x, y, x+curve, y)
            p.lineto(x+width-curve, y)
            p.curveto(x+width, y, x+width, y, x+width, y+curve)
190             p.lineto(x+width, y+height-curve)
            p.curveto(x+width, y+height, x+width, y+height, x+width-curve, y+height)
            p.lineto(x+curve, y+height)
            p.curveto(x, y+height, x, y+height, x, y+height-curve)
            p.closepath()
195         p.inheritFromContext(kwargs.keys())

        if draw:
            p.draw()
        return p

200
    def oval(self, x, y, width, height, draw=True, **kwargs):
        BezierPath.checkKwargs(kwargs)
        path = self.BezierPath(**kwargs)
        path.oval(x, y, width, height)
205         path.inheritFromContext(kwargs.keys())

        if draw:
            path.draw()
        return path

210
    ellipse = oval

    def arc(self, x, y, r, startAngle, endAngle, draw=True, **kwargs):
        BezierPath.checkKwargs(kwargs)
215         path = self.BezierPath(**kwargs)
        path.arc(x, y, r, startAngle, endAngle)
        path.inheritFromContext(kwargs.keys())
        if draw:
            path.draw()
220         return path

    def line(self, x1, y1, x2, y2, draw=True, **kwargs):
        BezierPath.checkKwargs(kwargs)
        p = self.BezierPath(**kwargs)
225         p.line(x1, y1, x2, y2)
        p.inheritFromContext(kwargs.keys())

```

```

        if draw:
            p.draw()
        return p
230
def star(self, startx, starty, points=20, outer= 100, inner = 50, draw=True, **kwargs):
    BezierPath.checkKwargs(kwargs)
    from math import sin, cos, pi

235
    p = self.BezierPath(**kwargs)
    p.moveto(startx, starty + outer)

    for i in range(1, int(2 * points)):
        angle = i * pi / points
240
        x = sin(angle)
        y = cos(angle)
        if i % 2:
            radius = inner
        else:
245
            radius = outer
        x = startx + radius * x
        y = starty + radius * y
        p.lineto(x,y)

250
    p.closepath()
    p.inheritFromContext(kwargs.keys())
    if draw:
        p.draw()
    return p

255
def arrow(self, x, y, width=100, type=NORMAL, draw=True, **kwargs):

    """Draws an arrow.

260
    Draws an arrow at position x, y, with a default width of 100.
    There are two different types of arrows: NORMAL and trendy FORTYFIVE degrees arrows.
    When draw=False then the arrow's path is not ended, similar to endpath(draw=False)."""

    BezierPath.checkKwargs(kwargs)
265
    if type==NORMAL:
        return self._arrow(x, y, width, draw, **kwargs)
    elif type==FORTYFIVE:
        return self._arrow45(x, y, width, draw, **kwargs)
    else:
270
        raise NodeBoxError("arrow: available types for arrow() are NORMAL and FORTYFIVE\n")

def _arrow(self, x, y, width, draw, **kwargs):

275
    head = width * .4
    tail = width * .2

    p = self.BezierPath(**kwargs)
    p.moveto(x, y)
    p.lineto(x-head, y+head)
280
    p.lineto(x-head, y+tail)
    p.lineto(x-width, y+tail)
    p.lineto(x-width, y-tail)
    p.lineto(x-head, y-tail)
    p.lineto(x-head, y-head)
285
    p.lineto(x, y)
    p.closepath()
    p.inheritFromContext(kwargs.keys())
    if draw:
        p.draw()
290
    return p

```

```

def _arrow45(self, x, y, width, draw, **kwargs):
    head = .3
    tail = 1 + head

    p = self.BezierPath(**kwargs)
    p.moveto(x, y)
    p.lineto(x, y+width*(1-head))
    p.lineto(x-width*head, y+width)
    p.lineto(x-width*head, y+width*tail*.4)
    p.lineto(x-width*tail*.6, y+width)
    p.lineto(x-width, y+width*tail*.6)
    p.lineto(x-width*tail*.4, y+width*head)
    p.lineto(x-width, y+width*head)
    p.lineto(x-width*(1-head), y)
    p.lineto(x, y)
    p.inheritFromContext(kwargs.keys())
    if draw:
        p.draw()
    return p

### Path Commands ###

def beginpath(self, x=None, y=None):
    self._path = self.BezierPath()
    self._pathclosed = False
    if x != None and y != None:
        self._path.moveto(x,y)

def moveto(self, x, y):
    if self._path is None:
        raise NodeBoxError, "No current path. Use beginpath() first."
    self._path.moveto(x,y)

def lineto(self, x, y):
    if self._path is None:
        raise NodeBoxError, "No current path. Use beginpath() first."
    self._path.lineto(x, y)

def curveto(self, x1, y1, x2, y2, x3, y3):
    if self._path is None:
        raise NodeBoxError, "No current path. Use beginpath() first."
    self._path.curveto(x1, y1, x2, y2, x3, y3)

def closepath(self):
    if self._path is None:
        raise NodeBoxError, "No current path. Use beginpath() first."
    if not self._pathclosed:
        self._path.closepath()

def endpath(self, draw=True):
    if self._path is None:
        raise NodeBoxError, "No current path. Use beginpath() first."
    if self._autoclosepath:
        self.closepath()
    p = self._path
    p.inheritFromContext()
    if draw:
        p.draw()
    self._path = None
    self._pathclosed = False
    return p

```

```

355     def drawpath(self, path, **kwargs):
        BezierPath.checkKwargs(kwargs)
        if isinstance(path, (list, tuple)):
            path = self.BezierPath(path, **kwargs)
        else: # Set the values in the current bezier path with the kwargs
360             for arg_key, arg_val in kwargs.items():
                 setattr(path, arg_key, _copy_attr(arg_val))
            path.inheritFromContext(kwargs.keys())
            path.draw()

365     def autoclosepath(self, close=True):
        self._autoclosepath = close

    def findpath(self, points, curvature=1.0):
        import bezier
370         path = bezier.findpath(points, curvature=curvature)
        path._ctx = self
        path.inheritFromContext()
        return path

375     ### Clipping Commands ###

    def beginclip(self, path):
        cp = self.ClippingPath(path)
        self.canvas.push(cp)
380         return cp

    def endclip(self):
        self.canvas.pop()

385     ### Transformation Commands ###

    def push(self): #, all=False):
        top = (self._transform.matrix,)
        if False: # all:
390             top = (self._align, self._autoclosepath, self._capstyle, self._colormode,
                    self._fillcolor, self._fontname, self._fontsize, self._joinstyle,
                    self._lineheight, self._outputmode, self._strokecolor,
                    self._strokewidth, self._transformmode, self._transform.matrix)
            self._transformstack.append(top)

395     def pop(self):
        try:
            top = self._transformstack.pop()
        except IndexError, e:
400             raise NodeBoxError, "pop: too many pops!"
        if len(top) > 1:
            self._align, self._autoclosepath, self._capstyle, self._colormode, self._fillcolor, self._f
        else:
            self._transform.matrix = top[0]

405     def transform(self, mode=None):
        if mode is not None:
            self._transformmode = mode
        return self._transformmode

410     def translate(self, x, y):
        self._transform.translate(x, y)

    def reset(self):
415         self._transform = Transform()

    def rotate(self, degrees=0, radians=0):
        self._transform.rotate(-degrees, -radians)

```

```

420     def translate(self, x=0, y=0):
        self._transform.translate(x,y)

    def scale(self, x=1, y=None):
        self._transform.scale(x,y)
425
    def skew(self, x=0, y=0):
        self._transform.skew(x,y)

    ### Color Commands ###

430     color = Color

    def colormode(self, mode=None, range=None):
        if mode is not None:
            self._colormode = mode
435         if range is not None:
            self._colorrange = float(range)
        return self._colormode

440     def colorrange(self, range=None):
        if range is not None:
            self._colorrange = float(range)
        return self._colorrange

445     def nofill(self):
        self._fillcolor = None

    def fill(self, *args):
        if len(args) > 0:
450             self._fillcolor = self.Color(*args)
        return self._fillcolor

    def nostroke(self):
        self._strokecolor = None
455

    def stroke(self, *args):
        if len(args) > 0:
            self._strokecolor = self.Color(*args)
        return self._strokecolor

460     def strokewidth(self, width=None):
        if width is not None:
            self._strokewidth = max(width, 0.0001)
        return self._strokewidth
465

    def capstyle(self, style=None):
        if style is not None:
            if style not in (BUTT, ROUND, SQUARE):
                raise NodeBoxError, 'Line cap style should be BUTT, ROUND or SQUARE.'
470             self._capstyle = style
        return self._capstyle

    def joinstyle(self, style=None):
        if style is not None:
            if style not in (MITER, ROUND, BEVEL):
                raise NodeBoxError, 'Line join style should be MITER, ROUND or BEVEL.'
475             self._joinstyle = style
        return self._joinstyle

480     ### Font Commands ###

    def font(self, fontname=None, fontsize = None):

```

```

    if fontname is not None:
        if not Text.font_exists(fontname):
485             raise NodeBoxError, 'Font "%s" not found.' % fontname
        else:
            self._fontname = fontname
    if fontsize is not None:
        self._fontsize = fontsize
490    return self._fontname

def fontsize(self, fontsize=None):
    if fontsize is not None:
        self._fontsize = fontsize
495    return self._fontsize

def lineheight(self, lineheight=None):
    if lineheight is not None:
        self._lineheight = max(lineheight, 0.01)
500    return self._lineheight

def align(self, align=None):
    if align is not None:
        self._align = align
505    return self._align

def textwidth(self, txt, width=None, **kwargs):
    """Calculates the width of a single-line string."""
    return self.textmetrics(txt, width, **kwargs)[0]
510

def textheight(self, txt, width=None, **kwargs):
    """Calculates the height of a (probably) multi-line string."""
    return self.textmetrics(txt, width, **kwargs)[1]

515 def text(self, txt, x, y, width=None, height=None, outline=False, draw=True, **kwargs):
    Text.checkKwargs(kwargs)
    txt = self.Text(txt, x, y, width, height, **kwargs)
    txt.inheritFromContext(kwargs.keys())
    if outline:
520         path = txt.path
        if draw:
            path.draw()
        return path
    else:
525         if draw:
            txt.draw()
        return txt

def textpath(self, txt, x, y, width=None, height=None, **kwargs):
530     Text.checkKwargs(kwargs)
    txt = self.Text(txt, x, y, width, height, **kwargs)
    txt.inheritFromContext(kwargs.keys())
    return txt.path

535 def textmetrics(self, txt, width=None, height=None, **kwargs):
    txt = self.Text(txt, 0, 0, width, height, **kwargs)
    txt.inheritFromContext(kwargs.keys())
    return txt.metrics

540 def alltextmetrics(self, txt, width=None, height=None, **kwargs):
    txt = self.Text(txt, 0, 0, width, height, **kwargs)
    txt.inheritFromContext(kwargs.keys())
    return txt.allmetrics

545 ### Image commands ###

```



```

def image(self, path, x, y, width=None, height=None, alpha=1.0, data=None, draw=True, **kwargs):
    img = self.Image(path, x, y, width, height, alpha, data=data, **kwargs)
    img.inheritFromContext(kwargs.keys())
550     if draw:
        img.draw()
    return img

def imagesize(self, path, data=None):
555     img = self.Image(path, data=data)
    return img.size

### Canvas proxy ###

560     def save(self, fname, format=None):
        self.canvas.save(fname, format)

    ## cGeo

565     def isqrt( self, v):
        return nodebox.geo.isqrt( v )

    def angle(self, x0, y0, x1, y1):
        return nodebox.geo.angle( x0, y0, x1, y1)
570

    def distance(self, x0, y0, x1, y1):
        return nodebox.geo.distance( x0, y0, x1, y1)

    def coordinates(self, x0, y0, distance, angle):
575     return nodebox.geo.coordinates(x0, y0, distance, angle)

    def reflect(self, x0, y0, x1, y1, d=1.0, a=180):
        return nodebox.geo.reflect(x0, y0, x1, y1, d, a)

```

## nodebox/graphics/bezier.py

```

# Bezier - last updated for NodeBox 1.8.3
# Author: Tom De Smedt <tomdesmedt@trapdoor.be>
# Manual: http://nodebox.net/code/index.php/Bezier
# Copyright (c) 2007 by Tom De Smedt.
5 # Refer to the "Use" section on http://nodebox.net/code
# Thanks to Dr. Florimond De Smedt at the Free University of Brussels for the math routines.

from nodebox.graphics import BezierPath, PathElement, NodeBoxError, Point
from nodebox.graphics import MOVETO, LINETO, CURVETO, CLOSE
10
try:
    import cPathmatics
    linepoint = cPathmatics.linepoint
    linelength = cPathmatics.linelength
    15     curvepoint = cPathmatics.curvepoint
    curvelength = cPathmatics.curvelength
except:
    import nodebox.geo.pathmatics
    linepoint = nodebox.geo.pathmatics.linepoint
    20     linelength = nodebox.geo.pathmatics.linelength
    curvepoint = nodebox.geo.pathmatics.curvepoint
    curvelength = nodebox.geo.pathmatics.curvelength

def segment_lengths(path, relative=False, n=20):
25     """Returns a list with the lengths of each segment in the path.

    >>> path = BezierPath(None)
    >>> segment_lengths(path)

```

```

30  []
    >>> path.moveto(0, 0)
    >>> segment_lengths(path)
    []
    >>> path.lineto(100, 0)
    >>> segment_lengths(path)
35  [100.0]
    >>> path.lineto(100, 300)
    >>> segment_lengths(path)
    [100.0, 300.0]
    >>> segment_lengths(path, relative=True)
40  [0.25, 0.75]
    >>> path = BezierPath(None)
    >>> path.moveto(1, 2)
    >>> path.curveto(3, 4, 5, 6, 7, 8)
    >>> segment_lengths(path)
45  [8.48528137423857]
    """

lengths = []
first = True

50  for el in path:
    if first == True:
        close_x, close_y = el.x, el.y
        first = False
55  elif el.cmd == MOVETO:
        close_x, close_y = el.x, el.y
        lengths.append(0.0)
    elif el.cmd == CLOSE:
        lengths.append(linelenlength(x0, y0, close_x, close_y))
60  elif el.cmd == LINETO:
        lengths.append(linelenlength(x0, y0, el.x, el.y))
    elif el.cmd == CURVETO:
        x3, y3, x1, y1, x2, y2 = (el.x, el.y, el.ctrl1.x, el.ctrl1.y,
                                   el.ctrl2.x, el.ctrl2.y)
65  lengths.append(curvelength(x0, y0, x1, y1, x2, y2, x3, y3, n))

    if el.cmd != CLOSE:
        x0 = el.x
        y0 = el.y
70

if relative:
    length = sum(lengths)
    try:
        return map(lambda l: l / length, lengths)
75  except ZeroDivisionError:
        # If the length is zero, just return zero for all segments
        return [0.0] * len(lengths)
    else:
        return lengths
80

def length(path, segmented=False, n=20):

    """Returns the length of the path.

85  Calculates the length of each spline in the path,
    using n as a number of points to measure.

    When segmented is True, returns a list
    containing the individual length of each spline
    as values between 0.0 and 1.0,
90  defining the relative length of each spline
    in relation to the total path length.

```

```

95     The length of an empty path is zero:
    >>> path = BezierPath(None)
    >>> length(path)
    0.0

    >>> path.moveto(0, 0)
100  >>> path.lineto(100, 0)
    >>> length(path)
    100.0

    >>> path.lineto(100, 100)
105  >>> length(path)
    200.0

    # Segmented returns a list of each segment
    >>> length(path, segmented=True)
110  [0.5, 0.5]
    """

    if not segmented:
        return sum(segment_lengths(path, n=n), 0.0)
115  else:
        return segment_lengths(path, relative=True, n=n)

def _locate(path, t, segments=None):
120  """Locates t on a specific segment in the path.

    Returns (index, t, PathElement)

    A path is a combination of lines and curves (segments).
125  The returned index indicates the start of the segment
    that contains point t.

    The returned t is the absolute time on that segment,
    in contrast to the relative t on the whole of the path.
130  The returned point is the last MOVETO,
    any subsequent CLOSETO after i closes to that point.

    When you supply the list of segment lengths yourself,
    as returned from length(path, segmented=True),
135  point() works about thirty times faster in a for-loop,
    since it doesn't need to recalculate the length
    during each iteration. Note that this has been deprecated:
    the BezierPath now caches the segment lengths the moment you use
    them.

140  >>> path = BezierPath(None)
    >>> _locate(path, 0.0)
    Traceback (most recent call last):
        ...
145  NodeBoxError: The given path is empty
    >>> path.moveto(0,0)
    >>> _locate(path, 0.0)
    Traceback (most recent call last):
        ...
150  NodeBoxError: The given path is empty
    >>> path.lineto(100, 100)
    >>> _locate(path, 0.0)
    (0, 0.0, Point(x=0.000, y=0.000))
    >>> _locate(path, 1.0)
155  (0, 1.0, Point(x=0.000, y=0.000))
    """

```

```

    if segments == None:
        segments = path.segmentlengths(relative=True)
160
    if len(segments) == 0:
        raise NodeBoxError, "The given path is empty"

    for i, el in enumerate(path):
165         if i == 0 or el.cmd == MOVETO:
            closeto = Point(el.x, el.y)
            if t <= segments[i] or i == len(segments)-1: break
            else: t -= segments[i]

170         try: t /= segments[i]
        except ZeroDivisionError: pass
        if i == len(segments)-1 and segments[i] == 0: i -= 1

        return (i, t, closeto)
175
def point(path, t, segments=None):

    """Returns coordinates for point at t on the path.

180     Gets the length of the path, based on the length
    of each curve and line in the path.
    Determines in what segment t falls.
    Gets the point on that segment.

185     When you supply the list of segment lengths yourself,
    as returned from length(path, segmented=True),
    point() works about thirty times faster in a for-loop,
    since it doesn't need to recalculate the length
    during each iteration. Note that this has been deprecated:
190     the BezierPath now caches the segment lengths the moment you use
    them.

    >>> path = BezierPath(None)
    >>> point(path, 0.0)
195     Traceback (most recent call last):
        ...
    NodeBoxError: The given path is empty
    >>> path.moveto(0, 0)
    >>> point(path, 0.0)
200     Traceback (most recent call last):
        ...
    NodeBoxError: The given path is empty
    >>> path.lineto(100, 0)
    >>> point(path, 0.0)
205     PathElement(LINETO, ((0.000, 0.000),))
    >>> point(path, 0.1)
    PathElement(LINETO, ((10.000, 0.000),))
    """

210     if len(path) == 0:
        raise NodeBoxError, "The given path is empty"

    i, t, closeto = _locate(path, t, segments=segments)

215     x0, y0 = path[i].x, path[i].y
    p1 = path[i+1]

    if p1.cmd == CLOSE:
        x, y = linepoint(t, x0, y0, closeto.x, closeto.y)
220         return PathElement(LINETO, ((x, y),))

```

```

elif p1.cmd == LINETO:
    x1, y1 = p1.x, p1.y
    x, y = linepoint(t, x0, y0, x1, y1)
    return PathElement(LINETO, ((x, y),))
225 elif p1.cmd == CURVETO:
    x3, y3, x1, y1, x2, y2 = (p1.x, p1.y,
                               p1.ctrl1.x, p1.ctrl1.y,
                               p1.ctrl2.x, p1.ctrl2.y)
    x, y, c1x, c1y, c2x, c2y = curvepoint(t, x0, y0, x1, y1, x2, y2, x3, y3)
230     return PathElement(CURVETO, ((c1x, c1y), (c2x, c2y), (x, y)))
else:
    raise NodeBoxError, "Unknown cmd for p1 %s" % p1

def points(path, amount=100):
235     """Returns an iterator with a list of calculated points for the path.
        This method calls the point method <amount> times, increasing t,
        distributing point spacing linearly.

    >>> path = BezierPath(None)
    >>> list(points(path))
    Traceback (most recent call last):
        ...
    NodeBoxError: The given path is empty
    >>> path.moveto(0, 0)
245     >>> list(points(path))
    Traceback (most recent call last):
        ...
    NodeBoxError: The given path is empty
    >>> path.lineto(100, 0)
250     >>> list(points(path, amount=4))
    [PathElement(LINETO, ((0.000, 0.000),)), PathElement(LINETO, ((33.333, 0.000),)), PathElement(LINETO, ((66.667, 0.000),)), PathElement(LINETO, ((100.000, 0.000),))]

    if len(path) == 0:
255         raise NodeBoxError, "The given path is empty"

    # The delta value is divided by amount - 1, because we also want the last point (t=1.0)
    # If I wouldn't use amount - 1, I fall one point short of the end.
    # E.g. if amount = 4, I want point at t 0.0, 0.33, 0.66 and 1.0,
    # if amount = 2, I want point at t 0.0 and t 1.0
260     try:
        delta = 1.0/(amount-1)
    except ZeroDivisionError:
        delta = 1.0
265

    for i in xrange(amount):
        yield point(path, delta*i)

def contours(path):
270     """Returns a list of contours in the path.

    A contour is a sequence of lines and curves
    separated from the next contour by a MOVETO.

275     For example, the glyph "o" has two contours:
    the inner circle and the outer circle.

    >>> path = BezierPath(None)
    >>> path.moveto(0, 0)
280     >>> path.lineto(100, 100)
    >>> len(contours(path))
    1

    A new contour is defined as something that starts with a moveto:

```

```

285     >>> path.moveto(50, 50)
    >>> path.curveto(150, 150, 50, 250, 80, 95)
    >>> len(contours(path))
    2

290     Empty moveto's don't do anything:
    >>> path.moveto(50, 50)
    >>> path.moveto(50, 50)
    >>> len(contours(path))
    2

295     It doesn't matter if the path is closed or open:
    >>> path.closepath()
    >>> len(contours(path))
    2
    """
300     contours = []
    current_contour = None
    empty = True
    for i, el in enumerate(path):
305         if el.cmd == MOVETO:
            if not empty:
                contours.append(current_contour)
                current_contour = BezierPath(path._ctx)
                current_contour.moveto(el.x, el.y)
            empty = True
310         elif el.cmd == LINETO:
            empty = False
            current_contour.lineto(el.x, el.y)
        elif el.cmd == CURVETO:
315             empty = False
            current_contour.curveto(el.ctrl1.x, el.ctrl1.y,
                                   el.ctrl2.x, el.ctrl2.y, el.x, el.y)
        elif el.cmd == CLOSE:
            current_contour.closepath()
320     if not empty:
        contours.append(current_contour)
    return contours

def findpath(points, curvature=1.0):
325     """Constructs a path between the given list of points.

    Interpolates the list of points and determines
    a smooth bezier path between them.

330     The curvature parameter offers some control on
    how separate segments are stitched together:
    from straight angles to smooth curves.
    Curvature is only useful if the path has more than three points.
335     """

    # The list of points consists of Point objects,
    # but it shouldn't crash on something straightforward
    # as someone supplying a list of (x,y)-tuples.

340     from types import TupleType
    for i, pt in enumerate(points):
        if type(pt) == TupleType:
            points[i] = Point(pt[0], pt[1])

345     if len(points) == 0: return None
    if len(points) == 1:
        path = BezierPath(None)

```

```

    path.moveto(points[0].x, points[0].y)
350     return path
    if len(points) == 2:
        path = BezierPath(None)
        path.moveto(points[0].x, points[0].y)
        path.lineto(points[1].x, points[1].y)
355     return path

    # Zero curvature means straight lines.

    curvature = max(0, min(1, curvature))
360     if curvature == 0:
        path = BezierPath(None)
        path.moveto(points[0].x, points[0].y)
        for i in range(len(points)):
            path.lineto(points[i].x, points[i].y)
365     return path

    curvature = 4 + (1.0-curvature)*40

    dx = {0: 0, len(points)-1: 0}
370     dy = {0: 0, len(points)-1: 0}
    bi = {1: -0.25}
    ax = {1: (points[2].x-points[0].x-dx[0]) / 4}
    ay = {1: (points[2].y-points[0].y-dy[0]) / 4}

375     for i in range(2, len(points)-1):
        bi[i] = -1 / (curvature + bi[i-1])
        ax[i] = -(points[i+1].x-points[i-1].x-ax[i-1]) * bi[i]
        ay[i] = -(points[i+1].y-points[i-1].y-ay[i-1]) * bi[i]

380     r = range(1, len(points)-1)
        r.reverse()
        for i in r:
            dx[i] = ax[i] + dx[i+1] * bi[i]
            dy[i] = ay[i] + dy[i+1] * bi[i]
385

    path = BezierPath(None)
    path.moveto(points[0].x, points[0].y)
    for i in range(len(points)-1):
        path.curveto(points[i].x + dx[i],
390                     points[i].y + dy[i],
                     points[i+1].x - dx[i+1],
                     points[i+1].y - dy[i+1],
                     points[i+1].x,
                     points[i+1].y)
395

    return path

def insert_point(path, t):

400     """Returns a path copy with an extra point at t.
    >>> path = BezierPath(None)
    >>> path.moveto(0, 0)
    >>> insert_point(path, 0.1)
    Traceback (most recent call last):
405         ...
    NodeBoxError: The given path is empty
    >>> path.moveto(0, 0)
    >>> insert_point(path, 0.2)
    Traceback (most recent call last):
410         ...
    NodeBoxError: The given path is empty
    >>> path.lineto(100, 50)

```

```

>>> len(path)
2
415 >>> path = insert_point(path, 0.5)
>>> len(path)
3
>>> path[1]
PathElement(LINETO, ((50.000, 25.000),))
420 >>> path = BezierPath(None)
>>> path.moveto(0, 100)
>>> path.curveto(0, 50, 100, 50, 100, 100)
>>> path = insert_point(path, 0.5)
>>> path[1]
425 PathElement(CURVETO, ((0.000, 75.000), (25.000, 62.5), (50.000, 62.500))
"""

i, t, closeto = _locate(path, t)

430 x0 = path[i].x
y0 = path[i].y
p1 = path[i+1]
plcmd, x3, y3, x1, y1, x2, y2 = (p1.cmd, p1.x, p1.y,
                                p1.ctrl1.x, p1.ctrl1.y,
435                                p1.ctrl2.x, p1.ctrl2.y)

if plcmd == CLOSE:
    pt_cmd = LINETO
    pt_x, pt_y = linepoint(t, x0, y0, closeto.x, closeto.y)
440 elif plcmd == LINETO:
    pt_cmd = LINETO
    pt_x, pt_y = linepoint(t, x0, y0, x3, y3)
elif plcmd == CURVETO:
    pt_cmd = CURVETO
445 s = curvepoint(t, x0, y0, x1, y1, x2, y2, x3, y3, True)
    pt_x, pt_y, pt_c1x, pt_c1y, pt_c2x, pt_c2y, pt_h1x, pt_h1y, pt_h2x, pt_h2y = s
else:
    raise NodeBoxError, "Locate should not return a MOVETO"

450 new_path = BezierPath(None)
new_path.moveto(path[0].x, path[0].y)
for j in range(1, len(path)):
    if j == i+1:
        if pt_cmd == CURVETO:
455             new_path.curveto(pt_h1x, pt_h1y,
                                pt_c1x, pt_c1y,
                                pt_x, pt_y)
            new_path.curveto(pt_c2x, pt_c2y,
                                pt_h2x, pt_h2y,
460                                path[j].x, path[j].y)
        elif pt_cmd == LINETO:
            new_path.lineto(pt_x, pt_y)
            if path[j].cmd != CLOSE:
                new_path.lineto(path[j].x, path[j].y)
465             else:
                new_path.closepath()
        else:
            raise NodeBoxError, "Didn't expect pt_cmd %s here" % pt_cmd

470 else:
    if path[j].cmd == MOVETO:
        new_path.moveto(path[j].x, path[j].y)
    if path[j].cmd == LINETO:
        new_path.lineto(path[j].x, path[j].y)
475     if path[j].cmd == CURVETO:
        new_path.curveto(path[j].ctrl1.x, path[j].ctrl1.y,

```



```

        path[j].ctrl2.x, path[j].ctrl2.y,
        path[j].x, path[j].y)
    if path[j].cmd == CLOSE:
480         new_path.closepath()
    return new_path

def _test():
    import doctest, bezier
485     return doctest.testmod(bezier)

if __name__ == '__main__':
    _test()

```

## nodebox/graphics/cocoa.py

```

import os
import warnings

import pdb
5
# from random import choice, shuffle
import random
choice = random.choice
shuffle = random.shuffle
10
import objc
super = objc.super

# from AppKit import *
15 import AppKit
NSBezierPath = AppKit.NSBezierPath
NSColor = AppKit.NSColor
NSGraphicsContext = AppKit.NSGraphicsContext

20 NSView = AppKit.NSView

NSDeviceCMYKColorSpace = AppKit.NSDeviceCMYKColorSpace
NSDeviceRGBColorSpace = AppKit.NSDeviceRGBColorSpace
NSAffineTransform = AppKit.NSAffineTransform
25 NSImage = AppKit.NSImage
NSImageCacheNever = AppKit.NSImageCacheNever
NSCompositeSourceOver = AppKit.NSCompositeSourceOver
NSLeftTextAlignment = AppKit.NSLeftTextAlignment
NSFont = AppKit.NSFont
30 NSMutableParagraphStyle = AppKit.NSMutableParagraphStyle
NSLineBreakByWordWrapping = AppKit.NSLineBreakByWordWrapping
NSParagraphStyleAttributeName = AppKit.NSParagraphStyleAttributeName
NSForegroundColorAttributeName = AppKit.NSForegroundColorAttributeName
NSFontAttributeName = AppKit.NSFontAttributeName
35 NSTextStorage = AppKit.NSTextStorage
NSLayoutManager = AppKit.NSLayoutManager
NSTextContainer = AppKit.NSTextContainer
NSRectFillUsingOperation = AppKit.NSRectFillUsingOperation
NSGIFFileType = AppKit.NSGIFFileType
40 NSJPEGFileType = AppKit.NSJPEGFileType
NSJPEGFileType = AppKit.NSJPEGFileType
NSPNGFileType = AppKit.NSPNGFileType
NSTIFFFileType = AppKit.NSTIFFFileType
NSBitmapImageRep = AppKit.NSBitmapImageRep
45 NSString = AppKit.NSString
NSData = AppKit.NSData
NSAffineTransformStruct = AppKit.NSAffineTransformStruct

```

```

from nodebox.util import _copy_attr, _copy_attrs, makeunicode
50
try:
    import cPolymagic
except ImportError, e:
    warnings.warn('Could not load cPolymagic: %s' % e)
55
__all__ = [
    "DEFAULT_WIDTH", "DEFAULT_HEIGHT",
    "inch", "cm", "mm",
    "RGB", "HSB", "CMYK",
60    "CENTER", "CORNER",
    "MOVETO", "LINETO", "CURVETO", "CLOSE",
    "MITER", "ROUND", "BEVEL", "BUTT", "SQUARE",
    "LEFT", "RIGHT", "CENTER", "JUSTIFY",
    "NORMAL", "FORTYFIVE",
65    "NUMBER", "TEXT", "BOOLEAN", "BUTTON", "MENU",
    "NodeBoxError",
    "Point", "Grob", "BezierPath", "PathElement", "ClippingPath", "Rect", "Oval",
    "Color", "Transform", "Image", "Text",
    "Variable", "Canvas",
70    ]

DEFAULT_WIDTH, DEFAULT_HEIGHT = 1000, 1000

# unused
75 inch = 72.0
    cm = inch / 2.54
    mm = cm * 10.0

    RGB = "rgb"
80 HSB = "hsb"
    CMYK = "cmyk"

    CENTER = "center"
    CORNER = "corner"
85
    MOVETO = AppKit.NSMoveToBezierPathElement
    LINETO = AppKit.NSLineToBezierPathElement
    CURVETO = AppKit.NSCurveToBezierPathElement
    CLOSE = AppKit.NSClosePathBezierPathElement
90
    MITER = AppKit.NSMiterLineJoinStyle
    ROUND = AppKit.NSRoundLineJoinStyle # Also used for NSRoundLineCapStyle, same value.
    BEVEL = AppKit.NSBevelLineJoinStyle
    BUTT = AppKit.NSButtLineCapStyle
95 SQUARE = AppKit.NSSquareLineCapStyle

    LEFT = AppKit.NSLeftTextAlignment
    RIGHT = AppKit.NSRightTextAlignment
    CENTER = AppKit.NSCenterTextAlignment
100 JUSTIFY = AppKit.NSJustifiedTextAlignment

    NORMAL=1
    FORTYFIVE=2

105 NUMBER = 1
    TEXT = 2
    BOOLEAN = 3
    BUTTON = 4
    MENU = 5
110
    KEY_UP = 126
    KEY_DOWN = 125

```

```

KEY_LEFT = 123
KEY_RIGHT = 124
115 KEY_BACKSPACE = 51
KEY_TAB = 48
KEY_ESC = 53

_STATE_NAMES = {
120     '_outputmode':    'outputmode',
        '_colorrange':  'colorrange',
        '_fillcolor':   'fill',
        '_strokecolor': 'stroke',
        '_strokewidth': 'strokewidth',
125     '_capstyle':     'capstyle',
        '_joinstyle':   'joinstyle',
        '_transform':   'transform',
        '_transformmode': 'transformmode',
        '_fontname':    'font',
130     '_fontsize':     'fontsize',
        '_align':       'align',
        '_lineheight':  'lineheight',
    }

135 def _save():
    NSGraphicsContext.currentContext().saveGraphicsState()

    def _restore():
        NSGraphicsContext.currentContext().restoreGraphicsState()

140 class NodeBoxError(Exception): pass

    class Point(object):

145         def __init__(self, *args):
            if len(args) == 2:
                self.x, self.y = args
            elif len(args) == 1:
                self.x, self.y = args[0]
150         elif len(args) == 0:
            self.x = self.y = 0.0
        else:
            raise NodeBoxError, "Wrong initializer for Point object"

155         def __repr__(self):
            return "Point(x=%.3f, y=%.3f)" % (self.x, self.y)

        def __eq__(self, other):
            if other is None: return False
160         return self.x == other.x and self.y == other.y

        def __ne__(self, other):
            return not self.__eq__(other)

165 class Grob(object):
    """A GGraphic Object is the base class for all DrawingPrimitives."""

    def __init__(self, ctx):
        """Initializes this object with the current context."""
170         self._ctx = ctx

    def draw(self):
        """Appends the grob to the canvas.
            This will result in a draw later on, when the scene graph is rendered."""
175         self._ctx.canvas.append(self)

```

```

def copy(self):
    """Returns a deep copy of this grob."""
    raise NotImplementedError, "Copy is not implemented on this Grob class."
180

def inheritFromContext(self, ignore=()):
    attrs_to_copy = list(self.__class__.stateAttributes)
    [attrs_to_copy.remove(k) for k, v in _STATE_NAMES.items() if v in ignore]
    _copy_attrs(self._ctx, self, attrs_to_copy)
185

def checkKwargs(self, kwargs):
    remaining = [arg for arg in kwargs.keys() if arg not in self.kwargs]
    if remaining:
        raise NodeBoxError, "Unknown argument(s) '%s'" % ", ".join(remaining)
190 checkKwargs = classmethod(checkKwargs)

class TransformMixin(object):

    """Mixin class for transformation support.
    Adds the _transform and _transformmode attributes to the class."""
195

    def __init__(self):
        self._reset()

200 def _reset(self):
    self._transform = Transform()
    self._transformmode = CENTER

    def _get_transform(self):
205         return self._transform
    def _set_transform(self, transform):
        self._transform = Transform(transform)
        transform = property(_get_transform, _set_transform)

210 def _get_transformmode(self):
    return self._transformmode
    def _set_transformmode(self, mode):
        self._transformmode = mode
        transformmode = property(_get_transformmode, _set_transformmode)
215

    def translate(self, x, y):
        self._transform.translate(x, y)

    def reset(self):
220         self._transform = Transform()

    def rotate(self, degrees=0, radians=0):
        self._transform.rotate(-degrees, -radians)

225 def translate(self, x=0, y=0):
    self._transform.translate(x,y)

    def scale(self, x=1, y=None):
        self._transform.scale(x,y)
230

    def skew(self, x=0, y=0):
        self._transform.skew(x,y)

class ColorMixin(object):
235

    """Mixin class for color support.
    Adds the _fillcolor, _strokecolor and _strokewidth attributes to the class."""

    def __init__(self, **kwargs):
240         try:

```

```

        self._fillcolor = Color(self._ctx, kwargs['fill'])
    except KeyError:
        self._fillcolor = Color(self._ctx)
    try:
245         self._strokecolor = Color(self._ctx, kwargs['stroke'])
    except KeyError:
        self._strokecolor = None
    self._strokewidth = kwargs.get('strokewidth', 1.0)

250     def _get_fill(self):
        return self._fillcolor
    def _set_fill(self, *args):
        self._fillcolor = Color(self._ctx, *args)
    fill = property(_get_fill, _set_fill)

255     def _get_stroke(self):
        return self._strokecolor
    def _set_stroke(self, *args):
        self._strokecolor = Color(self._ctx, *args)
260     stroke = property(_get_stroke, _set_stroke)

    def _get_strokewidth(self):
        return self._strokewidth
    def _set_strokewidth(self, strokewidth):
265         self._strokewidth = max(strokewidth, 0.0001)
    strokewidth = property(_get_strokewidth, _set_strokewidth)

class BezierPath(Grob, TransformMixin, ColorMixin):
    """A BezierPath provides a wrapper around NSBezierPath."""
270
    stateAttributes = ('_fillcolor', '_strokecolor', '_strokewidth', '_capstyle',
                       '_joinstyle', '_transform', '_transformmode')
    kwargs = ('fill', 'stroke', 'strokewidth', 'capstyle', 'joinstyle')

275     def __init__(self, ctx, path=None, **kwargs):
        super(BezierPath, self).__init__(ctx)
        TransformMixin.__init__(self)
        ColorMixin.__init__(self, **kwargs)
        self.capstyle = kwargs.get('capstyle', BUTT)
280         self.joinstyle = kwargs.get('joinstyle', MITER)
        self._segment_cache = None
        if path is None:
            self._nsBezierPath = NSBezierPath.bezierPath()
        elif isinstance(path, (list, tuple)):
285             self._nsBezierPath = NSBezierPath.bezierPath()
            self.extend(path)
        elif isinstance(path, BezierPath):
            self._nsBezierPath = path._nsBezierPath.copy()
            _copy_attrs(path, self, self.stateAttributes)
290         elif isinstance(path, NSBezierPath):
            self._nsBezierPath = path
        else:
            raise NodeBoxError, "Don't know what to do with %s." % path

295     def _get_path(self):
        s = "The 'path' attribute is deprecated. Please use _nsBezierPath instead."
        warnings.warn(s, DeprecationWarning, stacklevel=2)
        return self._nsBezierPath
    path = property(_get_path)

300     def copy(self):
        return self.__class__(self._ctx, self)

### Cap and Join style ###

```

```

305     def _get_capstyle(self):
        return self._capstyle
    def _set_capstyle(self, style):
        if style not in (BUTT, ROUND, SQUARE):
310             raise NodeBoxError, 'Line cap style should be BUTT, ROUND or SQUARE.'
        self._capstyle = style
    capstyle = property(_get_capstyle, _set_capstyle)

    def _get_joinstyle(self):
315         return self._joinstyle
    def _set_joinstyle(self, style):
        if style not in (MITER, ROUND, BEVEL):
            raise NodeBoxError, 'Line join style should be MITER, ROUND or BEVEL.'
        self._joinstyle = style
320    joinstyle = property(_get_joinstyle, _set_joinstyle)

    ### Path methods ###

    def moveto(self, x, y):
325         self._segment_cache = None
        self._nsBezierPath.moveToPoint_( (x, y) )

    def lineto(self, x, y):
        self._segment_cache = None
330        self._nsBezierPath.lineToPoint_( (x, y) )

    def curveto(self, x1, y1, x2, y2, x3, y3):
        self._segment_cache = None
        self._nsBezierPath.curveToPoint_controlPoint1_controlPoint2_(
335            (x3, y3), (x1, y1), (x2, y2) )

    # relativeMoveToPoint_( NSPoint )
    # relativeLineToPoint_( NSPoint )
    # relativeCurveToPoint:(NSPoint)aPoint controlPoint1:(NSPoint)controlPoint1 controlPoint2:(NSPoint)
340    # appendBezierPathWithOvalInRect_
    # appendBezierPathWithArcFromPoint_(NSPoint)fromPoint toPoint_(NSPoint)toPoint radius:(CGFloat)
    # appendBezierPathWithArcWithCenter:(NSPoint)center radius:(CGFloat)radius startAngle:(CGFloat)star
    # appendBezierPathWithArcWithCenter:(NSPoint)center radius:(CGFloat)radius startAngle:(CGFloat)star

345    def closepath(self):
        self._segment_cache = None
        self._nsBezierPath.closePath()

    def setlinewidth(self, width):
350        self.linewidth = width

    def _get_bounds(self):
        try:
            return self._nsBezierPath.bounds()
355        except:
            # Path is empty -- no bounds
            return (0,0) , (0,0)

    bounds = property(_get_bounds)

360    def contains(self, x, y):
        return self._nsBezierPath.containsPoint_((x,y))

    ### Basic shapes ###

365    def rect(self, x, y, width, height):
        self._segment_cache = None
        self._nsBezierPath.appendBezierPathWithRect_( ((x, y),

```

```

                                                                    (width, height)) )
370
def oval(self, x, y, width, height):
    self._segment_cache = None
    self._nsBezierPath.appendBezierPathWithOvalInRect_((x, y),
                                                         (width, height)) )
375
ellipse = oval

def arc(self, x, y, r, startAngle, endAngle):
    self._segment_cache = None
    self._nsBezierPath.appendBezierPathWithArcWithCenter_radius_startAngle_endAngle_(
380
        (x,y), r, startAngle, endAngle)

def line(self, x1, y1, x2, y2):
    self._segment_cache = None
    self._nsBezierPath.moveToPoint_( (x1, y1) )
385
    self._nsBezierPath.lineToPoint_( (x2, y2) )

### List methods ###

def __getitem__(self, index):
390
    cmd, el = self._nsBezierPath.elementAtIndex_associatedPoints_(index)
    return PathElement(cmd, el)

def __iter__(self):
    for i in range(len(self)):
395
        yield self[i]

def __len__(self):
    return self._nsBezierPath.elementCount()

400
def extend(self, pathElements):
    self._segment_cache = None
    for el in pathElements:
        if isinstance(el, (list, tuple)):
            x, y = el
            if len(self) == 0:
405
                cmd = MOVETO
            else:
                cmd = LINETO
            self.append(PathElement(cmd, ((x, y),)))
        elif isinstance(el, PathElement):
410
            self.append(el)
        else:
            raise NodeBoxError, "Don't know how to handle %s" % el

415
def append(self, el):
    self._segment_cache = None
    if el.cmd == MOVETO:
        self.moveto(el.x, el.y)
    elif el.cmd == LINETO:
420
        self.lineto(el.x, el.y)
    elif el.cmd == CURVETO:
        self.curveto(el.ctrl1.x, el.ctrl1.y, el.ctrl2.x, el.ctrl2.y, el.x, el.y)
    elif el.cmd == CLOSE:
        self.closepath()
425

def _get_contours(self):
    from nodebox.graphics import bezier
    return bezier.contours(self)
    contours = property(_get_contours)
430

### Drawing methods ###

```

```

def _get_transform(self):
    trans = self._transform.copy()
435     if (self._transformmode == CENTER):
        (x, y), (w, h) = self.bounds
        deltax = x + w / 2
        deltay = y + h / 2
        t = Transform()
440         t.translate(-deltax, -deltay)
        trans.prepend(t)
        t = Transform()
        t.translate(deltax, deltay)
        trans.append(t)
445     return trans
transform = property(_get_transform)

def _draw(self):
    _save()
450     self.transform.concat()
    if (self._fillcolor):
        self._fillcolor.set()
        self._nsBezierPath.fill()
    if (self._strokecolor):
455         self._strokecolor.set()
        self._nsBezierPath.setLineWidth_(self._strokewidth)
        self._nsBezierPath.setLineCapStyle_(self._capstyle)
        self._nsBezierPath.setLineJoinStyle_(self._joinstyle)
        self._nsBezierPath.stroke()
460     _restore()

### Geometry ###

def fit(self, x=None, y=None, width=None, height=None, stretch=False):
465     """Fits this path to the specified bounds.

All parameters are optional; if no parameters are specified,
nothing will happen. Specifying a parameter will constrain its value:
470     - x: The path will be positioned at the specified x value
    - y: The path will be positioned at the specified y value
    - width: The path will be of the specified width
    - height: The path will be of the specified height
475     - stretch: If both width and height are defined, either stretch the path or
        keep the aspect ratio.
    """

    (px, py), (pw, ph) = self.bounds
    t = Transform()
    if x is not None and y is None:
        t.translate(x, py)
    elif x is None and y is not None:
        t.translate(px, y)
485     elif x is not None and y is not None:
        t.translate(x, y)
    else:
        t.translate(px, py)
    if width is not None and height is None:
490         t.scale(width / pw)
    elif width is None and height is not None:
        t.scale(height / ph)
    elif width is not None and height is not None:
        if stretch:
495             t.scale(width / pw, height / ph)
        else:

```



```

        t.scale(min(width / pw, height / ph))
        t.translate(-px, -py)
        self._nsBezierPath = t.transformBezierPath(self)._nsBezierPath
500
    ### Mathematics ###

    def segmentlengths(self, relative=False, n=10):
        import bezier
505        if relative: # Use the opportunity to store the segment cache.
            if self._segment_cache is None:
                self._segment_cache = bezier.segment_lengths(self,
                                                                relative=True, n=n)

            return self._segment_cache
510        else:
            return bezier.segment_lengths(self, relative=False, n=n)

    def _get_length(self, segmented=False, n=10):
        import bezier
515        return bezier.length(self, segmented=segmented, n=n)
    length = property(_get_length)

    def point(self, t):
        import bezier
520        return bezier.point(self, t)

    def points(self, amount=100):
        import bezier
        if len(self) == 0:
525            raise NodeBoxError, "The given path is empty"

        # The delta value is divided by amount - 1, because we also want the
        # last point (t=1.0)
        # If I wouldn't use amount - 1, I fall one point short of the end.
530        # E.g. if amount = 4, I want point at t 0.0, 0.33, 0.66 and 1.0,
        # if amount = 2, I want point at t 0.0 and t 1.0
        try:
            delta = 1.0/(amount-1)
        except ZeroDivisionError:
535            delta = 1.0

        for i in xrange(amount):
            yield self.point(delta*i)

540    def addpoint(self, t):
        import bezier
        self._nsBezierPath = bezier.insert_point(self, t)._nsBezierPath
        self._segment_cache = None

545    ### Clipping operations ###

    def intersects(self, other):
        return cPolymagic.intersects(self._nsBezierPath, other._nsBezierPath)

550    def union(self, other, flatness=0.6):
        return BezierPath(self._ctx, cPolymagic.union(self._nsBezierPath,
                                                         other._nsBezierPath, flatness))

    def intersect(self, other, flatness=0.6):
555        return BezierPath(self._ctx, cPolymagic.intersect(self._nsBezierPath,
                                                             other._nsBezierPath, flatness))

    def difference(self, other, flatness=0.6):
        return BezierPath(self._ctx, cPolymagic.difference(self._nsBezierPath,
                                                             other._nsBezierPath, flatness))
560

```

```

    def xor(self, other, flatness=0.6):
        return BezierPath(self._ctx, cPolymagic.xor(self._nsBezierPath,
                                                    other._nsBezierPath, flatness))

565
class PathElement(object):

    def __init__(self, cmd=None, pts=None):
        self.cmd = cmd
570     if cmd == MOVETO:
            assert len(pts) == 1
            self.x, self.y = pts[0]
            self.ctrl1 = Point(pts[0])
            self.ctrl2 = Point(pts[0])
575     elif cmd == LINETO:
            assert len(pts) == 1
            self.x, self.y = pts[0]
            self.ctrl1 = Point(pts[0])
            self.ctrl2 = Point(pts[0])
580     elif cmd == CURVETO:
            assert len(pts) == 3
            self.ctrl1 = Point(pts[0])
            self.ctrl2 = Point(pts[1])
            self.x, self.y = pts[2]
585     elif cmd == CLOSE:
            assert pts is None or len(pts) == 0
            self.x = self.y = 0.0
            self.ctrl1 = Point(0.0, 0.0)
            self.ctrl2 = Point(0.0, 0.0)
590     else:
            self.x = self.y = 0.0
            self.ctrl1 = Point()
            self.ctrl2 = Point()

595     def __repr__(self):
        if self.cmd == MOVETO:
            return "PathElement(MOVETO, ((%.3f, %.3f),))" % (self.x, self.y)
        elif self.cmd == LINETO:
            return "PathElement(LINETO, ((%.3f, %.3f),))" % (self.x, self.y)
600     elif self.cmd == CURVETO:
        s = "PathElement(CURVETO, ((%.3f, %.3f), (%.3f, %.3f), (%.3f, %.3f)))"
            return s % (self.ctrl1.x, self.ctrl1.y,
                        self.ctrl2.x, self.ctrl2.y,
                        self.x, self.y)
605     elif self.cmd == CLOSE:
            return "PathElement(CLOSE)"

    def __eq__(self, other):
        if other is None: return False
610     if self.cmd != other.cmd: return False
        return self.x == other.x and self.y == other.y \
            and self.ctrl1 == other.ctrl1 and self.ctrl2 == other.ctrl2

    def __ne__(self, other):
615     return not self.__eq__(other)

class ClippingPath(Grob):

    def __init__(self, ctx, path):
620     self._ctx = ctx
        self.path = path
        self._grobs = []

    def append(self, grob):

```

```

625         self._grobs.append(grob)

    def _draw(self):
        _save()
        cp = self.path.transform.transformBezierPath(self.path)
630        cp._nsBezierPath.addClip()
        for grob in self._grobs:
            grob._draw()
        _restore()

635 class Rect(BezierPath):

    def __init__(self, ctx, x, y, width, height, **kwargs):
        warnings.warn("Rect is deprecated. Use BezierPath's rect method.",
                      DeprecationWarning, stacklevel=2)
640        r = (x,y), (width,height)
        super(Rect, self).__init__(ctx, NSBezierPath.bezierPathWithRect_(r),
                                   **kwargs)

    def copy(self):
645        raise NotImplementedError, "Please don't use Rect anymore"

class Oval(BezierPath):

    def __init__(self, ctx, x, y, width, height, **kwargs):
650        warnings.warn("Oval is deprecated. Use BezierPath's oval method.",
                      DeprecationWarning, stacklevel=2)
        r = (x,y), (width,height)
        super(Oval, self).__init__(ctx, NSBezierPath.bezierPathWithOvalInRect_(r),
                                   **kwargs)
655

    def copy(self):
        raise NotImplementedError, "Please don't use Oval anymore"

class Color(object):
660

    def __init__(self, ctx, *args):
        self._ctx = ctx
        params = len(args)

665        # Decompose the arguments into tuples.
        if params == 1 and isinstance(args[0], tuple):
            args = args[0]
            params = len(args)

670        if params == 1 and args[0] is None:
            clr = NSColor.colorWithDeviceWhite_alpha_(0.0, 0.0)
        elif params == 1 and isinstance(args[0], Color):
            if self._ctx._outputmode == RGB:
                clr = args[0]._rgb
675            else:
                clr = args[0]._cmyk
        elif params == 1 and isinstance(args[0], NSColor):
            clr = args[0]
        elif (
680             params == 1
            and isinstance(args[0], (str,unicode))
            and len(args[0]) in (3,4,5,6,7,8,9)):
            # hex param
            try:
                a = args[0]
                # kill hash char
                if a[0] == '#':
                    a = a[1:]
                alpha = 1.0

```

```

n = len(a)
690 if n in (3,4):
    div = 15.0
    if n == 3:
        r, g, b = a[:]
    else:
695         r, g, b, alpha = a[:]
    else:
        div = 255.0
        if n == 6:
            r, g, b = a[:2], a[2:4], a[4:6]
700         else:
            r, g, b, alpha = a[:2], a[2:4], a[4:6], a[6:8]
        r = int(r, 16) / div
        g = int(g, 16) / div
        b = int(b, 16) / div
705 if n in (4,8):
    alpha = int(alpha, 16) / div
    clr = NSColor.colorWithDeviceRed_green_blue_alpha_(r, g, b, alpha)
except Exception, err:
    print "Color parsing error:", err
710 clr = NSColor.colorWithDeviceWhite_alpha_(0, 1)

elif params == 1: # Gray, no alpha
    args = self._normalizeList(args)
    g, = args
715 clr = NSColor.colorWithDeviceWhite_alpha_(g, 1)
elif params == 2: # Gray and alpha
    args = self._normalizeList(args)
    g, a = args
    clr = NSColor.colorWithDeviceWhite_alpha_(g, a)
720 elif params == 3 and self._ctx._colormode == RGB: # RGB, no alpha
    args = self._normalizeList(args)
    r,g,b = args
    clr = NSColor.colorWithDeviceRed_green_blue_alpha_(r, g, b, 1)
elif params == 3 and self._ctx._colormode == HSB: # HSB, no alpha
725 args = self._normalizeList(args)
    h, s, b = args
    clr = NSColor.colorWithDeviceHue_saturation_brightness_alpha_(h, s, b, 1)
elif params == 4 and self._ctx._colormode == RGB: # RGB and alpha
    args = self._normalizeList(args)
730 r,g,b, a = args
    clr = NSColor.colorWithDeviceRed_green_blue_alpha_(r, g, b, a)
elif params == 4 and self._ctx._colormode == HSB: # HSB and alpha
    args = self._normalizeList(args)
    h, s, b, a = args
735 clr = NSColor.colorWithDeviceHue_saturation_brightness_alpha_(h, s, b, a)
elif params == 4 and self._ctx._colormode == CMYK: # CMYK, no alpha
    args = self._normalizeList(args)
    c, m, y, k = args
    clr = NSColor.colorWithDeviceCyan_magenta_yellow_black_alpha_(c, m, y, k, 1)
740 elif params == 5 and self._ctx._colormode == CMYK: # CMYK and alpha
    args = self._normalizeList(args)
    c, m, y, k, a = args
    clr = NSColor.colorWithDeviceCyan_magenta_yellow_black_alpha_(c, m, y, k, a)
else:
745 clr = NSColor.colorWithDeviceWhite_alpha_(0, 1)

self._cmk = clr.colorUsingColorSpaceName_(NSDeviceCMYKColorSpace)
self._rgb = clr.colorUsingColorSpaceName_(NSDeviceRGBColorSpace)

750 def __repr__(self):
    return "%s(%s, %s, %s, %s)" % (self.__class__.__name__, self.red,
        self.green, self.blue, self.alpha)

```

```

755     def set(self):
        self.nsColor.set()

    def _get_nsColor(self):
        if self._ctx._outputmode == RGB:
            return self._rgb
760         else:
            return self._cmyk
    nsColor = property(_get_nsColor)

    def copy(self):
765         new = self.__class__(self._ctx)
        new._rgb = self._rgb.copy()
        new._updateCmyk()
        return new

770     def _updateCmyk(self):
        self._cmyk = self._rgb.colorUsingColorSpaceName_(NSDeviceCMYKColorSpace)

    def _updateRgb(self):
        self._rgb = self._cmyk.colorUsingColorSpaceName_(NSDeviceRGBColorSpace)
775

    def _get_hue(self):
        return self._rgb.hueComponent()

    def _set_hue(self, val):
780         val = self._normalize(val)
        h, s, b, a = self._rgb.getHue_saturation_brightness_alpha_(None, None, None, None)
        self._rgb = NSColor.colorWithDeviceHue_saturation_brightness_alpha_(val, s, b, a)
        self._updateCmyk()
    h = hue = property(_get_hue, _set_hue, doc="the hue of the color")
785

    def _get_saturation(self):
        return self._rgb.saturationComponent()
    def _set_saturation(self, val):
        val = self._normalize(val)
790         h, s, b, a = self._rgb.getHue_saturation_brightness_alpha_(None, None, None, None)
        self._rgb = NSColor.colorWithDeviceHue_saturation_brightness_alpha_(h, val, b, a)
        self._updateCmyk()
    s = saturation = property(_get_saturation,
                               _set_saturation,
795                               doc="the saturation of the color")

    def _get_brightness(self):
        return self._rgb.brightnessComponent()

800     def _set_brightness(self, val):
        val = self._normalize(val)
        h, s, b, a = self._rgb.getHue_saturation_brightness_alpha_(None, None, None, None)
        self._rgb = NSColor.colorWithDeviceHue_saturation_brightness_alpha_(h, s, val, a)
        self._updateCmyk()
805     v = brightness = property(_get_brightness,
                               _set_brightness,
                               doc="the brightness of the color")

    def _get_hsba(self):
810         return self._rgb.getHue_saturation_brightness_alpha_(None, None, None, None)

    def _set_hsba(self, values):
        val = self._normalize(val)
        h, s, b, a = values
815         self._rgb = NSColor.colorWithDeviceHue_saturation_brightness_alpha_(h, s, b, a)
        self._updateCmyk()

```

```

hsba = property(_get_hsba,
                _set_hsba,
                doc="the hue, saturation, brightness and alpha of the color")
820
def _get_red(self):
    return self._rgb.redComponent()

def _set_red(self, val):
825     val = self._normalize(val)
        r, g, b, a = self._rgb.getRed_green_blue_alpha_(None, None, None, None)
        self._rgb = NSColor.colorWithDeviceRed_green_blue_alpha_(val, g, b, a)
        self._updateCmyk()
    r = red = property(_get_red, _set_red, doc="the red component of the color")
830

def _get_green(self):
    return self._rgb.greenComponent()

def _set_green(self, val):
835     val = self._normalize(val)
        r, g, b, a = self._rgb.getRed_green_blue_alpha_(None, None, None, None)
        self._rgb = NSColor.colorWithDeviceRed_green_blue_alpha_(r, val, b, a)
        self._updateCmyk()
    g = green = property(_get_green, _set_green, doc="the green component of the color")
840

def _get_blue(self):
    return self._rgb.blueComponent()
def _set_blue(self, val):
    val = self._normalize(val)
845     r, g, b, a = self._rgb.getRed_green_blue_alpha_(None, None, None, None)
        self._rgb = NSColor.colorWithDeviceRed_green_blue_alpha_(r, g, val, a)
        self._updateCmyk()
    b = blue = property(_get_blue, _set_blue, doc="the blue component of the color")

850 def _get_alpha(self):
    return self._rgb.alphaComponent()
def _set_alpha(self, val):
    val = self._normalize(val)
        r, g, b, a = self._rgb.getRed_green_blue_alpha_(None, None, None, None)
855     self._rgb = NSColor.colorWithDeviceRed_green_blue_alpha_(r, g, b, val)
        self._updateCmyk()
    a = alpha = property(_get_alpha, _set_alpha, doc="the alpha component of the color")

def _get_rgba(self):
860     return self._rgb.getRed_green_blue_alpha_(None, None, None, None)

def _set_rgba(self, val):
    val = self._normalizeList(val)
        r, g, b, a = val
865     self._rgb = NSColor.colorWithDeviceRed_green_blue_alpha_(r, g, b, a)
        self._updateCmyk()
    rgba = property(_get_rgba,
                    _set_rgba,
                    doc="the red, green, blue and alpha values of the color")
870

def _get_cyan(self):
    return self._cmyk.cyanComponent()

def _set_cyan(self, val):
875     val = self._normalize(val)
        c, m, y, k, a = self.cmyka
        self._cmyk = NSColor.colorWithDeviceCyan_magenta_yellow_black_alpha_(val, m, y, k, a)
        self._updateRgb()
    c = cyan = property(_get_cyan, _set_cyan, doc="the cyan component of the color")
880

```

```

def _get_magenta(self):
    return self._cmyk.magentaComponent()

def _set_magenta(self, val):
885     val = self._normalize(val)
        c, m, y, k, a = self.cmyka
        self._cmyk = NSColor.colorWithDeviceCyan_magenta_yellow_black_alpha_(c, val, y, k, a)
        self._updateRgb()
m = magenta = property(_get_magenta,
890                     _set_magenta,
                        doc="the magenta component of the color")

def _get_yellow(self):
    return self._cmyk.yellowComponent()
895

def _set_yellow(self, val):
    val = self._normalize(val)
    c, m, y, k, a = self.cmyka
    self._cmyk = NSColor.colorWithDeviceCyan_magenta_yellow_black_alpha_(
900                                     c, m, val, k, a)
        self._updateRgb()
y = yellow = property(_get_yellow,
                      _set_yellow,
                      doc="the yellow component of the color")
905

def _get_black(self):
    return self._cmyk.blackComponent()

def _set_black(self, val):
910     val = self._normalize(val)
        c, m, y, k, a = self.cmyka
        self._cmyk = NSColor.colorWithDeviceCyan_magenta_yellow_black_alpha_(
                                     c, m, y, val, a)
        self._updateRgb()
915     k = black = property(_get_black,
                          _set_black,
                          doc="the black component of the color")

def _get_cmyka(self):
920     return (self._cmyk.cyanComponent(),
            self._cmyk.magentaComponent(),
            self._cmyk.yellowComponent(),
            self._cmyk.blackComponent(),
            self._cmyk.alphaComponent())
925     cmyka = property(_get_cmyka, doc="a tuple containing the CMYKA values for this color")

def blend(self, otherColor, factor):
    """Blend the color with otherColor with a factor; return the new color. Factor
    is a float between 0.0 and 1.0.
930     """
    if hasattr(otherColor, "color"):
        otherColor = otherColor._rgb
    return self.__class__(color=self._rgb.blendedColorWithFraction_ofColor_(
        factor, otherColor))
935

def _normalize(self, v):
    """Bring the color into the 0-1 scale for the current colorrange"""
    if self._ctx._colorrange == 1.0:
        return v
940     return v / self._ctx._colorrange

def _normalizeList(self, lst):
    """Bring the color into the 0-1 scale for the current colorrange"""
    r = self._ctx._colorrange

```

```

945         if r == 1.0:
            return lst
        return [v / r for v in lst]

color = Color
950
class Transform(object):

    def __init__(self, transform=None):
        if transform is None:
955             transform = NSAffineTransform.transform()
        elif isinstance(transform, Transform):
            matrix = transform._nsAffineTransform.transformStruct()
            transform = NSAffineTransform.transform()
            transform.setTransformStruct_(matrix)
960        elif isinstance(transform, (list, tuple, NSAffineTransformStruct)):
            matrix = tuple(transform)
            transform = NSAffineTransform.transform()
            transform.setTransformStruct_(matrix)
        elif isinstance(transform, NSAffineTransform):
965             pass
        else:
            raise NodeBoxError, "Don't know how to handle transform %s." % transform
            self._nsAffineTransform = transform

970    def _get_transform(self):
        s = ("The 'transform' attribute is deprecated. "
            "Please use _nsAffineTransform instead.")
        warnings.warn(s, DeprecationWarning, stacklevel=2)
        return self._nsAffineTransform
975    transform = property(_get_transform)

    def set(self):
        self._nsAffineTransform.set()

980    def concat(self):
        self._nsAffineTransform.concat()

    def copy(self):
        return self.__class__(self._nsAffineTransform.copy())
985
    def __repr__(self):
        return "<%s [%s %s %s %s %s %s]>" % ((self.__class__.__name__,)
            + tuple(self))

990    def __iter__(self):
        for value in self._nsAffineTransform.transformStruct():
            yield value

    def _get_matrix(self):
995        return self._nsAffineTransform.transformStruct()

    def _set_matrix(self, value):
        self._nsAffineTransform.setTransformStruct_(value)
        matrix = property(_get_matrix, _set_matrix)
1000
    def rotate(self, degrees=0, radians=0):
        if degrees:
            self._nsAffineTransform.rotateByDegrees_(degrees)
        else:
1005            self._nsAffineTransform.rotateByRadians_(radians)

    def translate(self, x=0, y=0):
        self._nsAffineTransform.translateXBy_yBy_(x, y)

```



```

1010     def scale(self, x=1, y=None):
        if y is None:
            y = x
        self._nsAffineTransform.scaleXBy_yBy_(x, y)

1015     def skew(self, x=0, y=0):
        import math
        x = math.pi * x / 180
        y = math.pi * y / 180
        t = Transform()
1020     t.matrix = 1, math.tan(y), -math.tan(x), 1, 0, 0
        self.prepend(t)

    def invert(self):
        self._nsAffineTransform.invert()

1025     def append(self, other):
        if isinstance(other, Transform):
            other = other._nsAffineTransform
        self._nsAffineTransform.appendTransform_(other)

1030     def prepend(self, other):
        if isinstance(other, Transform):
            other = other._nsAffineTransform
        self._nsAffineTransform.prependTransform_(other)

1035     def transformPoint(self, point):
        return self._nsAffineTransform.transformPoint_(point)

    def transformBezierPath(self, path):
1040     if isinstance(path, BezierPath):
        path = BezierPath(path._ctx, path)
    else:
        raise NodeBoxError, "Can only transform BezierPaths"
    path._nsBezierPath = self._nsAffineTransform.transformBezierPath_(path._nsBezierPath)
1045     return path

class Image(Grob, TransformMixin):

    stateAttributes = ('_transform', '_transformmode')
1050     kwargs = ()

    def __init__(self, ctx, path=None, x=0, y=0,
                  width=None, height=None, alpha=1.0, image=None, data=None):
        """
1055     Parameters:
        - path: A path to a certain image on the local filesystem.
        - x: Horizontal position.
        - y: Vertical position.
        - width: Maximum width. Images get scaled according to this factor.
1060     - height: Maximum height. Images get scaled according to this factor.
            If a width and height are both given, the smallest
            of the two is chosen.
        - alpha: transparency factor
        - image: optionally, an Image or NSImage object.
1065     - data: a stream of bytes of image data.
        """
        super(Image, self).__init__(ctx)
        TransformMixin.__init__(self)
        if data is not None:
1070             if not isinstance(data, NSData):
                data = NSData.dataWithBytes_length_(data, len(data))
            self._nsImage = NSImage.alloc().initWithData_(data)

```

```

    if self._nsImage is None:
        raise NodeBoxError, "can't read image %r" % path
1075     self._nsImage.setFlipped_(True)
        self._nsImage.setCacheMode_(NSImageCacheNever)
    elif image is not None:
        if isinstance(image, NSImage):
            self._nsImage = image
1080         self._nsImage.setFlipped_(True)
        else:
            raise NodeBoxError, "Don't know what to do with %s." % image
    elif path is not None:
        if not os.path.exists(path):
1085             raise NodeBoxError, 'Image "%s" not found.' % path
        curtime = os.path.getmtime(path)
        try:
            image, lasttime = self._ctx._imagecache[path]
            if lasttime != curtime:
1090                 image = None
        except KeyError:
            pass
        if image is None:
            image = NSImage.alloc().initWithContentsOfFile_(path)
1095             if image is None:
                raise NodeBoxError, "Can't read image %r" % path
            image.setFlipped_(True)
            image.setCacheMode_(NSImageCacheNever)
            self._ctx._imagecache[path] = (image, curtime)
1100         self._nsImage = image
        self.x = x
        self.y = y
        self.width = width
        self.height = height
1105         self.alpha = alpha
        self.debugImage = False

    def _get_image(self):
        w = "The 'image' attribute is deprecated. Please use _nsImage instead."
1110         warnings.warn(w, DeprecationWarning, stacklevel=2)
        return self._nsImage
    image = property(_get_image)

    def copy(self):
1115         new = self.__class__(self._ctx)
        _copy_attrs(self, new, ('image', 'x', 'y', 'width', 'height',
                                '_transform', '_transformmode', 'alpha', 'debugImage'))
        return new

    def getSize(self):
1120         return self._nsImage.size()

    size = property(getSize)

    def _draw(self):
1125         """Draw an image on the given coordinates."""

        srcW, srcH = self._nsImage.size()
        srcRect = ((0, 0), (srcW, srcH))
1130
        # Width or height given
        if self.width is not None or self.height is not None:
            if self.width is not None and self.height is not None:
                factor = min(self.width / srcW, self.height / srcH)
1135             elif self.width is not None:
                factor = self.width / srcW

```

```

elif self.height is not None:
    factor = self.height / srcH
    _save()
1140
    # Center-mode transforms: translate to image center
    if self._transformmode == CENTER:
        # This is the hardest case: center-mode transformations with given
        # width or height.
        # Order is very important in this code.

        # Set the position first, before any of the scaling or transformations
        # are done.
        # Context transformations might change the translation, and we don't
        # want that.
1150
        t = Transform()
        t.translate(self.x, self.y)
        t.concat()

        # Set new width and height factors. Note that no scaling is done yet:
        # they're just here to set the new center of the image according to
        # the scaling factors.
        srcW = srcW * factor
        srcH = srcH * factor

1160
        # Move image to newly calculated center.
        dX = srcW / 2
        dY = srcH / 2
        t = Transform()
        t.translate(dX, dY)
1165
        t.concat()

        # Do current transformation.
        self._transform.concat()

1170
        # Move back to the previous position.
        t = Transform()
        t.translate(-dX, -dY)
        t.concat()

1175
        # Finally, scale the image according to the factors.
        t = Transform()
        t.scale(factor)
        t.concat()

1180
    else:
        # Do current transformation
        self._transform.concat()
        # Scale according to width or height factor
        t = Transform()
        t.translate(self.x, self.y) # Here we add the positioning of the image.
1185
        t.scale(factor)
        t.concat()

        # A debugImage draws a black rectangle instead of an image.
1190
        if self.debugImage:
            Color(self._ctx).set()
            pt = BezierPath()
            pt.rect(0, 0, srcW / factor, srcH / factor)
            pt.fill()

1195
        else:
            self._nsImage.drawAtPoint_fromRect_operation_fraction_((0, 0),
                                                                    srcRect, NSCompositeSourceOver, self.alpha)
            _restore()
        # No width or height given
1200
    else:

```

```

        _save()
        x,y = self.x, self.y
        # Center-mode transforms: translate to image center
        if self._transformmode == CENTER:
1205             deltaX = srcW / 2
                deltaY = srcH / 2
                t = Transform()
                t.translate(x+deltaX, y+deltaY)
                t.concat()
1210             x = -deltaX
                y = -deltaY
        # Do current transformation
        self._transform.concat()
        # A debugImage draws a black rectangle instead of an image.
1215         if self.debugImage:
            Color(self._ctx).set()
            pt = BezierPath()
            pt.rect(x, y, srcW, srcH)
            pt.fill()
1220         else:
            # The following code avoids a nasty bug in Cocoa/PyObjC.
            # Apparently, EPS files are put on a different position when drawn
            # with a certain position.
            # However, this only happens when the alpha value is set to 1.0: set
1225             # it to something lower and the positioning is the same as a bitmap
            # file.
            # I could of course make every EPS image have an alpha value of
            # 0.9999, but this solution is better: always use zero coordinates for
            # drawAtPoint and use a transform to set the final position.
1230             t = Transform()
                t.translate(x,y)
                t.concat()
                self._nsImage.drawAtPoint_fromRect_operation_fraction_(
                    (0,0), srcRect, NSCompositeSourceOver, self.alpha)
1235         _restore()

class Text(Grob, TransformMixin, ColorMixin):

    stateAttributes = ('_transform', '_transformmode', '_fillcolor', '_fontname',
1240                       '_fontsize', '_align', '_lineheight')
    kwargs = ('fill', 'font', 'fontsize', 'align', 'lineheight')

    __dummy_color = NSColor.blackColor()

1245     def __init__(self, ctx, text, x=0, y=0, width=None, height=None, **kwargs):
        super(Text, self).__init__(ctx)
        TransformMixin.__init__(self)
        ColorMixin.__init__(self, **kwargs)
        self.text = makeunicode(text)
1250         self.x = x
        self.y = y
        self.width = width
        self.height = height
        self._fontname = kwargs.get('font', "Helvetica")
1255         self._fontsize = kwargs.get('fontsize', 24)
        self._lineheight = max(kwargs.get('lineheight', 1.2), 0.01)
        self._align = kwargs.get('align', NSLeftTextAlignment)

    def copy(self):
1260         new = self.__class__(self._ctx, self.text)
        _copy_attrs(self, new,
            ('x', 'y', 'width', 'height', '_transform', '_transformmode',
             '_fillcolor', '_fontname', '_fontsize', '_align', '_lineheight'))
        return new

```

```

1265     def font_exists(cls, fontname):
        # Check if the font exists.
        f = NSFont.fontWithName_size_(fontname, 12)
        return f is not None
1270 font_exists = classmethod(font_exists)

    def _get_font(self):
        return NSFont.fontWithName_size_(self._fontname, self._fontsize)
    font = property(_get_font)

1275     def _getLayoutManagerTextContainerTextStorage(self, clr=__dummy_color):
        paraStyle = NSMutableParagraphStyle.alloc().init()
        paraStyle.setAlignment_(self._align)
        paraStyle.setLineBreakMode_(NSLineBreakByWordWrapping)
1280         paraStyle.setLineHeightMultiple_(self._lineheight)

        d = {
            NSParagraphStyleAttributeName: paraStyle,
            NSForegroundColorAttributeName: clr,
1285             NSFontAttributeName: self.font
        }

        t = unicode(self.text)
        textStorage = NSTextStorage.alloc().initWithString_attributes_(t, d)
1290         try:
            textStorage.setFont_(self.font)
        except ValueError:
            raise NodeBoxError("Text.draw(): font '%s' not available.\n" % self._fontname)
            return

1295         layoutManager = NSLayoutManager.alloc().init()
        textContainer = NSTextContainer.alloc().init()
        if self.width != None:
            textContainer.setContainerSize_((self.width, 1000000))
1300             textContainer.setWidthTracksTextView_(False)
            textContainer.setHeightTracksTextView_(False)
        layoutManager.addTextContainer_(textContainer)
        textStorage.addLayoutManager_(layoutManager)
        return layoutManager, textContainer, textStorage

1305     def _draw(self):
        if self._fillcolor is None:
            return

1310         s = self._getLayoutManagerTextContainerTextStorage(self._fillcolor.nsColor)
        layoutManager, textContainer, textStorage = s

        x,y = self.x, self.y
        glyphRange = layoutManager.glyphRangeForTextContainer_(textContainer)
1315         s = layoutManager.boundingBoxRectForGlyphRange_inTextContainer_(glyphRange,
                                                                           textContainer)

        (dx, dy), (w, h) = s
        preferredWidth, preferredHeight = textContainer.containerSize()
        if self.width is not None:
1320             if self._align == RIGHT:
                x += preferredWidth - w
            elif self._align == CENTER:
                x += preferredWidth/2 - w/2

1325         _save()
        # Center-mode transforms: translate to image center
        if self._transformmode == CENTER:
            deltaX = w / 2

```

```

1330     deltaY = h / 2
        t = Transform()
        t.translate(x+deltaX, y-self.font.defaultLineHeightForFont()+deltaY)
        t.concat()
        self._transform.concat()
        layoutManager.drawGlyphsForGlyphRange_atPoint_(glyphRange, (-deltaX-dx,-deltaY-dy))
1335     else:
        self._transform.concat()
        layoutManager.drawGlyphsForGlyphRange_atPoint_(glyphRange,
                                                         (x-dx, y-dy-self.font.defaultLineHeightForFont()))
        _restore()
1340     return (w, h)

    def _get_allmetrics(self):
        items = self._getLayoutManagerTextContainerTextStorage()
        layoutManager, textContainer, textStorage = items
1345        glyphRange = layoutManager.glyphRangeForTextContainer_(textContainer)
        (dx, dy), (w, h) = layoutManager.boundingBoxRectForGlyphRange_inTextContainer_(
                                                         glyphRange, textContainer)

        # print "metrics (dx,dy):", (dx,dy)
        return dx,dy,w,h
1350    allmetrics = property(_get_allmetrics)

    def _get_metrics(self):
        dx,dy,w,h = self._get_allmetrics()
        return w,h
1355    metrics = property(_get_metrics)

    def _get_path(self):
        items = self._getLayoutManagerTextContainerTextStorage()
        layoutManager, textContainer, textStorage = items
1360        x, y = self.x, self.y
        glyphRange = layoutManager.glyphRangeForTextContainer_(textContainer)
        (dx, dy), (w, h) = layoutManager.boundingBoxRectForGlyphRange_inTextContainer_(
                                                         glyphRange, textContainer)

        preferredWidth, preferredHeight = textContainer.containerSize()
1365    if self.width is not None:
        if self._align == RIGHT:
            x += preferredWidth - w
        elif self._align == CENTER:
            x += preferredWidth/2 - w/2
1370    length = layoutManager.numberOfGlyphs()
    path = NSBezierPath.bezierPath()
    for glyphIndex in range(length):
        lineFragmentRect = layoutManager.lineFragmentRectForGlyphAtIndex_effectiveRange_(
                                                         glyphIndex, None)
1375        # HACK: PyObjc 2.0 and 2.2 are subtly different:
        # - 2.0 (bundled with OS X 10.5) returns one argument: the rectangle.
        # - 2.2 (bundled with OS X 10.6) returns two arguments: the rectangle and the range.
        # So we check if we got one or two arguments back (in a tuple) and unpack them.
        if isinstance(lineFragmentRect, tuple):
1380            lineFragmentRect = lineFragmentRect[0]
            layoutPoint = layoutManager.locationForGlyphAtIndex_(glyphIndex)

            # Here layoutLocation is the location (in container coordinates)
            # where the glyph was laid out.
1385            finalPoint = [lineFragmentRect[0][0],lineFragmentRect[0][1]]
            finalPoint[0] += layoutPoint[0] - dx
            finalPoint[1] += layoutPoint[1] - dy
            g = layoutManager.glyphAtIndex_(glyphIndex)
            if g == 0:
1390                continue
            path.moveToPoint_((finalPoint[0], -finalPoint[1]))
            path.appendBezierPathWithGlyph_inFont_(g, self.font)

```

```

        path.closePath()
    path = BezierPath(self._ctx, path)
1395    trans = Transform()
    trans.translate(x,y-self.font.defaultLineHeightForFont())
    trans.scale(1.0,-1.0)
    path = trans.transformBezierPath(path)
    path.inheritFromContext()
1400    return path
    path = property(_get_path)

class Variable(object):
    def __init__(self, name, typ,
1405        default=None, minV=0, maxV=100, value=None,
        handler=None, menuitems=None):
        self.name = makeunicode(name)
        self.type = typ or NUMBER
        self.default = default
1410    self.min = minV
        self.max = maxV

        self.handler = None
        if handler is not None:
1415            self.handler = handler

        self.menuitems = None
        if menuitems is not None:
            if type(menuitems) in (list, tuple):
1420                self.menuitems = [makeunicode(i) for i in menuitems]

        if self.type == NUMBER:
            if default is None:
                self.default = 50
1425            self.min = minV
                self.max = maxV

        elif self.type == TEXT:
            if default is None:
1430                self.default = makeunicode("hello")
            else:
                self.default = makeunicode(default)

        elif self.type == BOOLEAN:
1435            if default is None:
                self.default = True
            else:
                self.default = bool(default)

        elif self.type == BUTTON:
1440            self.default = makeunicode(self.name)

        elif self.type == MENU:
            # value is list of menuitems
            # default is name of function to call with selected menu item name

            # old interface
            if type(value) in (list, tuple): # and type(default) in (function,):
                # print "type(default)", type(default)
1450                if default is not None:
                    self.handler = default
                    self.menuitems = [makeunicode(i) for i in value]
                    default = None
                    value = ""
1455                if default is None:

```

```

        if self.menuitems is not None:
            if len(self.menuitems) > 0:
                default = self.menuitems[0]
1460         else:
                default = u""
            self.default = default
        self.value = value or self.default

1465     def sanitize(self, val):
        """Given a Variable and a value, cleans it out"""
        if self.type == NUMBER:
            try:
                return float(val)
1470             except ValueError:
                return 0.0
            elif self.type == TEXT:
                return unicode(str(val), "utf_8", "replace")
            try:
1475                 return unicode(str(val), "utf_8", "replace")
            except:
                return ""
            elif self.type == BOOLEAN:
                if unicode(val).lower() in ("true", "1", "yes"):
1480                     return True
                else:
                    return False

    def compliesTo(self, v):
1485        """Return whether I am compatible with the given var:
        - Type should be the same
        - My value should be inside the given vars' min/max range.
        """
        if self.type == v.type:
            if self.type == NUMBER:
                if self.value < self.min or self.value > self.max:
                    return False
                return True
            return False
1495
    def __repr__(self):
        s = ("Variable(name=%s, typ=%s, default=%s, min=%s, max=%s, value=%s, "
            "handler=%s, menuitems=%s)")
        return s % (self.name, self.type, self.default, self.min, self.max, self.value,
1500                    repr(self.handler), repr(self.menuitems))

class _PDFRenderView(NSView):

    # This view was created to provide PDF data.
    # Strangely enough, the only way to get PDF data from Cocoa is by asking
    # dataWithPDFInsideRect_ from a NSView. So, we create one just to get to
    # the PDF data.

    def initWithCanvas_(self, canvas):
1510
        # for some unknown reason the following line stopped working
        # Solution: use objc.super -- see import
        super(_PDFRenderView, self).initWithFrame_(((0, 0), (canvas.width, canvas.height)))
        # for some unknown reason this is the solution for the preceding problem
1515        # self.initWithFrame_(((0, 0), (canvas.width, canvas.height)))
        # it is the only super in this file, having a NS* superclass

        self.canvas = canvas
        return self
1520

```



```

    def drawRect_(self, rect):
        self.canvas.draw()

    def isOpaque(self):
1525         return False

    def isFlipped(self):
        return True

1530 class Canvas(Grob):

    def __init__(self, width=DEFAULT_WIDTH, height=DEFAULT_HEIGHT):
        self.width = width
        self.height = height
1535         self.speed = None
        self.mousedown = False
        self.clear()

    def clear(self):
1540         self._grobs = self._container = []
        self._grobstack = [self._grobs]

    def _get_size(self):
        return self.width, self.height
1545 size = property(_get_size)

    def append(self, el):
        self._container.append(el)

1550 def __iter__(self):
    for grob in self._grobs:
        yield grob

    def __len__(self):
1555         return len(self._grobs)

    def __getitem__(self, index):
        return self._grobs[index]

1560 def push(self, containerGrob):
    self._grobstack.insert(0, containerGrob)
    self._container.append(containerGrob)
    self._container = containerGrob

1565 def pop(self):
    try:
        del self._grobstack[0]
        self._container = self._grobstack[0]
    except IndexError, e:
1570         raise NodeBoxError, "pop: too many canvas pops!"

    def draw(self):
        if self.background is not None:
            self.background.set()
1575             NSRectFillUsingOperation(((0,0), (self.width, self.height)),
                                     NSCompositeSourceOver)

        for grob in self._grobs:
            grob._draw()

1580 def _get_nsImage(self):
    img = NSImage.alloc().initWithSize_((self.width, self.height))
    img.setFlipped_(True)
    img.lockFocus()
    self.draw()

```

```

1585         img.unlockFocus()
            return img
        _nsImage = property(_get_nsImage)

    def _getImageData(self, format):
1590         if format == 'pdf':
            view = _PDFRenderView.alloc().initWithCanvas_(self)
            return view.dataWithPDFInsideRect_(view.bounds())
        elif format == 'eps':
            view = _PDFRenderView.alloc().initWithCanvas_(self)
1595         return view.dataWithEPSInsideRect_(view.bounds())
        else:
            imgTypes = {"gif": NSGIFFileType,
                        "jpg": NSJPEGFileType,
                        "jpeg": NSJPEGFileType,
1600         "png": NSPNGFileType,
                        "tiff": NSTIFFFileType}
            if format not in imgTypes:
                e = "Filename should end in .pdf, .eps, .tiff, .gif, .jpg or .png"
                raise NodeBoxError, e
1605         data = self._nsImage.TIFFRepresentation()
            if format != 'tiff':
                imgType = imgTypes[format]
                rep = NSBitmapImageRep.imageRepWithData_(data)
                return rep.representationUsingType_properties_(imgType, None)
1610         else:
            return data

    def save(self, fname, format=None):
        if format is None:
1615         basename, ext = os.path.splitext(fname)
            format = ext[1:].lower() # Skip the dot
            data = self._getImageData(format)
            fname = NSString.stringByExpandingTildeInPath(fname)
            data.writeToFile_atomically_(fname, False)
1620
    def _test():
        import doctest, cocoa
        return doctest.testmod(cocoa)

1625 if __name__ == '__main__':
    _test()

```

## nodebox/gui/\_\_init\_\_.py

## nodebox/gui/mac/\_\_init\_\_.py

```

import sys
import os
import traceback, linecache
import re
5 import objc
import time
import random
import signal
import atexit
10
import pprint
pp = pprint.pprint

import pdb

```

```

15     # set to true to have stdio on the terminal for pdb
    debugging = False

    # if true print out some debug info on stdout
20 kwlog = False

    import Foundation
    import AppKit
    NSObject = AppKit.NSObject
25 NSColor = AppKit.NSColor
    NSScriptCommand = AppKit.NSScriptCommand

    NSDocument = AppKit.NSDocument
    NSDocumentController = AppKit.NSDocumentController
30    NSNotificationCenter = AppKit.NSNotificationCenter

    NSFontAttributeName = AppKit.NSFontAttributeName
    NSScreen = AppKit.NSScreen
35 NSMenu = AppKit.NSMenu
    NSCursor = AppKit.NSCursor
    NSTimer = AppKit.NSTimer
    NSForegroundColorAttributeName = AppKit.NSForegroundColorAttributeName

40 NSPasteboard = AppKit.NSPasteboard
    NSPDFPboardType = AppKit.NSPDFPboardType
    NSPostScriptPboardType = AppKit.NSPostScriptPboardType
    NSTIFFPboardType = AppKit.NSTIFFPboardType

45 NSBundle = AppKit.NSBundle
    NSSavePanel = AppKit.NSSavePanel
    NSLog = AppKit.NSLog
    NSApp = AppKit.NSApp
    NSPrintOperation = AppKit.NSPrintOperation
50 NSWindow = AppKit.NSWindow
    NSBorderlessWindowMask = AppKit.NSBorderlessWindowMask
    NSBackingStoreBuffered = AppKit.NSBackingStoreBuffered
    NSView = AppKit.NSView
    NSGraphicsContext = AppKit.NSGraphicsContext
55 NSRectFill = AppKit.NSRectFill
    NSAffineTransform = AppKit.NSAffineTransform
    NSFocusRingTypeExterior = AppKit.NSFocusRingTypeExterior
    NSResponder = AppKit.NSResponder

60 NSURL = AppKit.NSURL
    NSWorkspace = AppKit.NSWorkspace
    NSBezierPath = AppKit.NSBezierPath

    import threading
65 Thread = threading.Thread

    import ValueLadder
    MAGICVAR = ValueLadder.MAGICVAR

70 import PyDETextView

    import preferences
    NodeBoxPreferencesController = preferences.NodeBoxPreferencesController
    LibraryFolder = preferences.LibraryFolder
75    import util
    errorAlert = util.errorAlert

```

```

# from nodebox import util
80 import nodebox.util
    util = nodebox.util
    makeunicode = nodebox.util.makeunicode

    import nodebox.util.ottobot
85 genProgram = nodebox.util.ottobot.genProgram

    import nodebox.util.QTSupport
    QTSupport = nodebox.util.QTSupport

90 # from nodebox import graphics
    import nodebox.graphics
    graphics = nodebox.graphics

    # AppleScript enumerator codes for PDF and Quicktime export
95 PDF = 0x70646678 # 'pdfx'
    QUICKTIME = 0x71747878 # 'qt '

    black = NSColor.blackColor()
    VERY_LIGHT_GRAY = black.blendedColorWithFraction_ofColor_(0.95,
100                                     NSColor.whiteColor())
    DARKER_GRAY = black.blendedColorWithFraction_ofColor_(0.8,
                                                             NSColor.whiteColor())

    # from nodebox.gui.mac.dashboard import *
105 # from nodebox.gui.mac.progressbar import ProgressBarController
    import dashboard
    DashboardController = dashboard.DashboardController

    import progressbar
110 ProgressBarController = progressbar.ProgressBarController

    class ExportCommand(NSScriptCommand):
        pass

115 class OutputFile(object):

    def __init__(self, data, isErr=False):
        self.data = data
        self.isErr = isErr

120
    def write(self, data):
        if isinstance(data, str):
            try:
                data = unicode(data, "utf_8", "replace")
125            except UnicodeDecodeError:
                data = "XXX " + repr(data)
            self.data.append((self.isErr, data))

    # class defined in NodeBoxDocument.xib
130 class NodeBoxDocument(NSDocument):

    graphicsView = objc.IBOutlet()
    outputView = objc.IBOutlet()
    textView = objc.IBOutlet()
135 window = objc.IBOutlet()
    variablesController = objc.IBOutlet()
    dashboardController = objc.IBOutlet()
    animationSpinner = objc.IBOutlet()

140 # The ExportImageAccessory adds:
    exportImageAccessory = objc.IBOutlet()
    exportImageFormat = objc.IBOutlet()

```

```

exportImagePageCount = objc.IBOutlet()

145  # The ExportMovieAccessory adds:
exportMovieAccessory = objc.IBOutlet()
exportMovieFrames = objc.IBOutlet()
exportMovieFps = objc.IBOutlet()

150  # When the PageCount accessory is loaded, we also add:
pageCount = objc.IBOutlet()
pageCountAccessory = objc.IBOutlet()

# When the ExportSheet is loaded, we also add:
155  exportSheet = objc.IBOutlet()
exportSheetIndicator = objc.IBOutlet()

path = None
exportDir = None
160  magicvar = None # Used for value ladders.
_code = None
vars = []
movie = None

165  def windowNibName(self):
    return "NodeBoxDocument"

def init(self):
    self = super(NodeBoxDocument, self).init()
170  nc = NSNotificationCenter defaultCenter()
    nc.addObserver_selector_name_object_(self,
                                           "textFontChanged:",
                                           "PyDETextFontChanged",
                                           None)

175  self.namespace = {}
self.canvas = graphics.Canvas()
self.context = graphics.Context(self.canvas, self.namespace)
self.animationTimer = None
self.__doc__ = {}
180  self._pageNumber = 1
self._frame = 150
self.fullScreen = None
self._seed = time.time()

185  # this is None
self.currentView = self.graphicsView
return self

def autosavesInPlace(self):
190  return True

def close(self):
    self.stopScript()
    super(NodeBoxDocument, self).close()

195  def __del__(self):
    nc = NSNotificationCenter defaultCenter()
    nc.removeObserver_name_object_(self, "PyDETextFontChanged", None)
    # text view has a couple of circular refs, it can let go of them now
200  self.textView._cleanup()

def textFontChanged_(self, notification):
    font = PyDETextView.getBasicTextAttributes()[NSFontAttributeName]
    self.outputView.setFont_(font)

205  def readFromFile_ofType_(self, path, tp):

```

```

    if self.textView is None:
        # we're not yet fully loaded
        self.path = path
210     else:
        # "revert"
        self.readFromUTF8_(path)
    return True

215     def writeToFile_ofType_(self, path, tp):
        f = file(path, "w")
        text = self.textView.string()
        f.write(text.encode("utf8"))
        f.close()
220     return True

    def windowControllerDidLoadNib_(self, controller):
        if self.path:
            self.readFromUTF8_(self.path)
225         font = PyDETextView.getBasicTextAttributes()[NSFontAttributeName]
        self.outputView.setFont_(font)
        self.textView.window().makeFirstResponder_(self.textView)
        self.windowControllers()[0].setWindowFrameAutosaveName_("NodeBoxDocumentWindow")

230         # switch off automatic substitutions
        try:
            self.textView.setAutomaticQuoteSubstitutionEnabled_( False )
            self.textView.setAutomaticDashSubstitutionEnabled_( False )

235             # This does not work well with syntax coloring
            #self.textView.setAutomaticLinkDetectionEnabled_( True )
            #self.textView.setDisplaysLinkToolTips_( True )

            self.outputView.setAutomaticQuoteSubstitutionEnabled_( False )
240             self.outputView.setAutomaticDashSubstitutionEnabled_( False )
            #self.outputView.setAutomaticLinkDetectionEnabled_( True )
            #self.outputView.setDisplaysLinkToolTips_( True )
        except Exception, err:
            pass

245     def readFromUTF8_(self, path):
        f = file(path)
        text = unicode(f.read(), "utf_8")
        f.close()
250         self.textView.setString_(text)
        self.textView.usesTabs = "\t" in text

    def cleanRun_newSeed_buildInterface_(self, fn, newSeed, buildInterface):
        self.animationSpinner.startAnimation_(None)
255         # Prepare everything for running the script
        self.prepareRun()

        # Run the actual script
260         success = self.fastRun_newSeed_(fn, newSeed)
        self.animationSpinner.stopAnimation_(None)

        if success and buildInterface:

265             # Build the interface
            self.vars = self.namespace["_ctx"]._vars
            if len(self.vars) > 0:
                self.buildInterface_(None)

270     return success

```

```

def prepareRun(self):

    # Compile the script
275    success, output = self.boxedRun_args_(self._compileScript, [])
    self.flushOutput_(output)
    if not success:
        return False

280    # Initialize the namespace
    self._initNamespace()

    # Reset the pagenum
    self._pageNum = 1

285    # Reset the frame
    self._frame = 1

    self.speed = self.canvas.speed = None

290    def fastRun_newSeed_(self, fn, newSeed = False):
        """This is the old signature. Dispatching to the new with args"""
        return self.fastRun_newSeed_args_(fn, newSeed, [])

295    def fastRun_newSeed_args_(self, fn, newSeed = False, args=[]):
        # Check if there is code to run
        if self._code is None:
            return False

300    # Clear the canvas
    self.canvas.clear()

    # Generate a new seed, if needed
    if newSeed:
305        self._seed = time.time()
        random.seed(self._seed)

    # Set the mouse position

310    # kw fix
    if not self.currentView:
        self.currentView = self.graphicsView

    window = self.currentView.window()
    pt = window.mouseLocationOutsideOfEventStream()
    mx, my = window.contentView().convertPoint_toView_(pt, self.currentView)
    # Hack: mouse coordinates are flipped vertically in FullscreenView.
    # This flips them back.
    if isinstance(self.currentView, FullscreenView):
320        my = self.currentView.bounds()[1][1] - my
    if self.fullScreen is None:
        mx /= self.currentView.zoom
        my /= self.currentView.zoom
    self.namespace["MOUSEX"], self.namespace["MOUSEY"] = mx, my
325    self.namespace["mousedown"] = self.currentView.mousedown
    self.namespace["keydown"] = self.currentView.keydown
    self.namespace["key"] = self.currentView.key
    self.namespace["keycode"] = self.currentView.keycode
    self.namespace["scrollwheel"] = self.currentView.scrollwheel
330    self.namespace["wheeldelta"] = self.currentView.wheeldelta

    # Reset the context
    self.context._resetContext()

```

```

335     # Initialize the magicvar
        self.namespace[MAGICVAR] = self.magicvar

        # Set the pagenum
        self.namespace['PAGENUM'] = self._pageNumber

340     # Set the frame
        self.namespace['FRAME'] = self._frame

        # Run the script
345     success, output = self.boxedRun_args_(fn, args)
        self.flushOutput_(output)
        if not success:
            return False

350     # Display the output of the script
        self.currentView.setCanvas_(self.canvas)

        return True

355 @objc.IBAction
def runFullscreen_(self, sender):
    if self.fullScreen is not None: return
    self.stopScript()
    self.currentView = FullscreenView.alloc().init()
360     self.currentView.canvas = None
    fullRect = NSScreen.mainScreen().frame()
    self.fullScreen = FullscreenWindow.alloc().initWithRect_(fullRect)
    self.fullScreen.setContentView_(self.currentView)
    self.fullScreen.makeKeyAndOrderFront_(self)
365     self.fullScreen.makeFirstResponder_(self.currentView)
    NSMenu.setMenuBarVisible_(False)
    NSCursor.hide()
    self._runScript()

370 @objc.IBAction
def runScript_(self, sender):
    self.runScript()

def runScript(self, compile=True, newSeed=True):
375     if self.fullScreen is not None: return
    self.currentView = self.graphicsView
    self._runScript(compile, newSeed)

def _runScript(self, compile=True, newSeed=True):
380     if not self.cleanRun_newSeed_buildInterface_(self._execScript, True, True):
        pass

    # Check whether we are dealing with animation
    if self.canvas.speed is not None:
385         if not self.namespace.has_key("draw"):
            showAlert("Not a proper NodeBox animation",
                    "NodeBox animations should have at least a draw() method.")
            return

390     # Check if animationTimer is already running
    if self.animationTimer is not None:
        self.stopScript()

    self.speed = self.canvas.speed

395     # Run setup routine
    if self.namespace.has_key("setup"):
        self.fastRun_newSeed_(self.namespace["setup"], False)

```



```

window = self.currentView.window()
400 window.makeFirstResponder_(self.currentView)

    # Start the timer
    timer = NSTimer.scheduledTimerWithTimeInterval_target_selector_userInfo_repeats_
    self.animationTimer = timer(1.0 / self.speed,
405         self,
        objc.selector(self.doFrame, signature="v:@"),
        None,
        True)

410     # Start the spinner
    self.animationSpinner.startAnimation_(None)

def runScriptFast(self):
    if self.animationTimer is None:
415         self.fastRun_newSeed_(self._execScript, False)
    else:
        # XXX: This can be sped up. We just run _execScript to get the
        # method with __MAGICVAR__ into the namespace, and execute
        # that, so it should only be called once for animations.
420         self.fastRun_newSeed_(self._execScript, False)
        self.fastRun_newSeed_(self.namespace["draw"], False)

def doFrame(self):
    self.fastRun_newSeed_(self.namespace["draw"], True)
425     self._frame += 1

def source(self):
    return self.textView.string()

430 def setSource_(self, source):
    self.textView.setString_(source)

@objc.IBAction
def stopScript_(self, sender=None):
435     self.stopScript()

def stopScript(self):
    if self.namespace.has_key("stop"):
        success, output = self.boxedRun_args_(self.namespace["stop"], [])
440         self.flushOutput_(output)
    self.animationSpinner.stopAnimation_(None)
    if self.animationTimer is not None:
        self.animationTimer.invalidate()
        self.animationTimer = None
445     if self.fullScreen is not None:
        self.currentView = self.graphicsView
        self.fullScreen = None
        NSMenu.setMenuBarVisible_(True)
    NSCursor.unhide()
450     self.textView.hideValueLadder()
    window = self.textView.window()
    window.makeFirstResponder_(self.textView)

def _compileScript(self, source=None):
455     if source is None:
        source = self.textView.string()

    # if this is activated, all unicode carrying scripts NEED a "encoding" line
    # OTOH if this is on, NB accepts scripts with an encoding line.
460     # currently an error
    # source = source.encode("utf-8")
    self._code = None

```

```

        self._code = compile(source + "\n\n",
                               self.scriptName.encode('ascii', 'ignore'),
                               "exec")
465

def _initNamespace(self):

    self.namespace.clear()
470    # Add everything from the namespace
    for name in graphics.__all__:
        self.namespace[name] = getattr(graphics, name)
    for name in util.__all__:
        self.namespace[name] = getattr(util, name)
475

    # debug print all collected keywords
    if kwlog:
        #print "util.__all__:"
        #pp(util.__all__)
480        #print "graphics.__all__:"
        #pp(graphics.__all__)
        print "namespace.keys():"
        pp(namespace.keys())

485    # Add everything from the context object
    self.namespace["_ctx"] = self.context
    for attrName in dir(self.context):
        self.namespace[attrName] = getattr(self.context, attrName)
    # Add the document global
490    self.namespace["__doc__"] = self.__doc__
    # Add the page number
    self.namespace["PAGENUM"] = self._pageNumber
    # Add the frame number
    self.namespace["FRAME"] = self._frame
495    # Add the magic var
    self.namespace[MAGICVAR] = self.magicvar
    # XXX: will be empty after reset.
    #for var in self.vars:
    #    self.namespace[var.name] = var.value
500

def _execScript(self):
    exec self._code in self.namespace
    self.__doc__ = self.namespace.get("__doc__", self.__doc__)

505 def boxedRun_args_(self, method, args):
    """
    Runs the given method in a boxed environment.
    Boxed environments:
    - Have their current directory set to the directory of the file
    - Have their argument set to the filename
    - Have their outputs redirect to an output stream.
    Returns:
    A tuple containing:
    - A boolean indicating whether the run was successful
    - The OutputFile
515    """

    self.scriptName = self.fileName()
    libpath = LibraryFolder()
520    libDir = libpath.libDir

    if not self.scriptName:
        curDir = os.getenv("HOME")
        self.scriptName = "<untitled>"
525    else:
        curDir = os.path.dirname(self.scriptName)

```

```

save = sys.stdout, sys.stderr
saveDir = os.getcwd()
530 saveArgv = sys.argv
sys.argv = [self.scriptName]
if os.path.exists(libDir):
    sys.path.insert(0, libDir)
os.chdir(curDir)
535 sys.path.insert(0, curDir)
output = []

# for pdb debugging in terminal this needs to be switched off
if not debugging:
540     sys.stdout = OutputFile(output, False)
    sys.stderr = OutputFile(output, True)
self._scriptDone = False
try:
    if self.animationTimer is None:
545         pass
        # Creating a thread is a heavy operation,
        # don't install it when animating, where speed is crucial
        #t = Thread(target=self._userCancelledMonitor,
        #           name="UserCancelledMonitor")
550         #t.start()
    try:
        method(*args)
    except KeyboardInterrupt:
        self.stopScript()
555     except:
        etype, value, tb = sys.exc_info()
        if tb.tb_next is not None:
            tb = tb.tb_next # skip the frame doing the exec
            traceback.print_exception(etype, value, tb)
560         etype = value = tb = None
        return False, output
finally:
    self._scriptDone = True
    sys.stdout, sys.stderr = save
565    os.chdir(saveDir)
    sys.path.remove(curDir)
    try:
        sys.path.remove(libDir)
    except ValueError:
570         pass
    sys.argv = saveArgv
    #self.flushOutput_()
    return True, output

575 # UNUSED - Referenced in commented out Thread section of boxedRun_args_
# Should be removed since Carbon is not available anymore

# from Mac/Tools/IDE/PyEdit.py
def _userCancelledMonitor(self):
580     from Carbon import Evt
    while not self._scriptDone:
        if Evt.CheckEventQueueForUserCancel():
            # Send a SIGINT signal to ourselves.
            # This gets delivered to the main thread,
585             # cancelling the running script.
            os.kill(os.getpid(), signal.SIGINT)
            break
        time.sleep(0.25)

590 def flushOutput_(self, output):

```

```

        outAttrs = PyDETextView.getBasicTextAttributes()
        errAttrs = outAttrs.copy()
        # XXX err color from user defaults...
        errAttrs[NSForegroundColorAttributeName] = NSColor.redColor()

595
        outputView = self.outputView
        outputView.setSelectedRange_((outputView.textStorage().length(), 0))
        lastErr = None
        for isErr, data in output:
            if isErr != lastErr:
                attrs = [outAttrs, errAttrs][isErr]
                outputView.setTypingAttributes_(attrs)
                lastErr = isErr
                outputView.insertText_(data)
605
        # del self.output

@objc.IBAction
def copyImageAsPDF_(self, sender):
    pboard = NSPasteboard.generalPasteboard()
    # graphicsView implements the pboard delegate method to provide the data
    pboard.declareTypes_owner_( [NSPDFPboardType,
                                NSPostScriptPboardType,
                                NSTIFFPboardType],
                                self.graphicsView)

615

@objc.IBAction
def exportAsImage_(self, sender):
    exportPanel = NSSavePanel.savePanel()
    exportPanel.setRequiredFileType_("pdf")
    exportPanel.setNameFieldLabel_("Export To:")
    exportPanel.setPrompt_("Export")
    exportPanel.setCanSelectHiddenExtension_(True)
    if not NSBundle.loadNibNamed_owner_("ExportImageAccessory", self):
        NSLog("Error -- could not load ExportImageAccessory.")
    self.exportImagePageCount.setIntValue_(1)
    exportPanel.setAccessoryView_(self.exportImageAccessory)
    path = self.fileName()
    if path:
        dirName, fileName = os.path.split(path)
        630
        fileName, ext = os.path.splitext(fileName)
        fileName += ".pdf"
    else:
        dirName, fileName = None, "Untitled.pdf"
    # If a file was already exported, use that folder as the default.
    635
    if self.exportDir is not None:
        dirName = self.exportDir
    exportPanel.beginSheetForDirectory_file_modalForWindow_modalDelegate_didEndSelector_contextInfo_
        dirName,
        fileName,
    640
        NSApp().mainWindow(),
        self,
        "exportPanelDidEnd:returnCode:contextInfo:", 0)

def exportPanelDidEnd_returnCode_contextInfo_(self, panel, returnCode, context):
    645
    if returnCode:
        fname = panel.filename()
        self.exportDir = os.path.split(fname)[0] # Save the directory we exported to.
        pages = self.exportImagePageCount.intValue()
        format = panel.requiredFileType()
    650
        panel.close()
        self.doExportAsImage_fmt_pages_(fname, format, pages)
    exportPanelDidEnd_returnCode_contextInfo_ = objc.selector(
        exportPanelDidEnd_returnCode_contextInfo_,
        signature="v:@:ii")

```

```

655 @objc.IBAction
def exportImageFormatChanged_(self, sender):
    image_formats = ('pdf', 'eps', 'png', 'tiff', 'jpg', 'gif')
    panel = sender.window()
660 panel.setRequiredFileType_(image_formats[sender.indexOfSelectedItem()])

def doExportAsImage_fmt_pages_(self, fname, format, pages):
    basename, ext = os.path.splitext(fname)
    # When saving one page (the default), just save the current graphics
    # context. When generating multiple pages, we run the script again
    # (so we don't use the current displayed view) for the first page,
    # and then for every next page.
    if pages == 1:
        if self.graphicsView.canvas is None:
670 self.runScript()
        self.canvas.save(fname, format)
    elif pages > 1:
        pb = ProgressBarController.alloc().init()
        pb.begin_maxval_("Generating %s images..." % pages, pages)
675 try:
            if not self.cleanRun_newSeed_buildInterface_(self._execScript,
                                                         True, True):

                return
            self._pageNumber = 1
680 self._frame = 1

            # If the speed is set, we are dealing with animation
            if self.canvas.speed is None:
                for i in range(pages):
685 if i > 0: # Run has already happened first time
                    self.fastRun_newSeed_(self._execScript, True)
                    counterAsString = "-%5d" % self._pageNumber
                    counterAsString = counterAsString.replace(' ', '0')
                    exportName = basename + counterAsString + ext

                    self.canvas.save(exportName, format)
                    self.graphicsView.setNeedsDisplay_(True)
                    self._pageNumber += 1
                    self._frame += 1
690 pb.inc()
            else:
                if self.namespace.has_key("setup"):
                    self.fastRun_newSeed_(self.namespace["setup"], False)
                for i in range(pages):
700 self.fastRun_newSeed_(self.namespace["draw"], True)
                    # 1-based
                    counterAsString = "-%5d" % self._pageNumber
                    # 0-based
                    # counterAsString = "-%5d" % i
705 counterAsString = counterAsString.replace(' ', '0')
                    exportName = basename + counterAsString + ext
                    self.canvas.save(exportName, format)
                    self.graphicsView.setNeedsDisplay_(True)
                    self._pageNumber += 1
710 self._frame += 1
                    pb.inc()
                if self.namespace.has_key("stop"):
                    success, output = self.boxedRun_args_(self.namespace["stop"],
                                                         [])
715 self.flushOutput_(output)
except KeyboardInterrupt:
    pass
pb.end()

```

```

    del pb
720     self._pageNumber = 1
        self._frame = 1

@objc.IBAction
def exportAsMovie_(self, sender):
725     exportPanel = NSSavePanel.savePanel()
        exportPanel.setRequiredFileType_("pdf")
        exportPanel.setNameFieldLabel_("Export To:")
        exportPanel.setPrompt_("Export")
        exportPanel.setCanSelectHiddenExtension_(True)
730     exportPanel.setAllowedFileTypes_(["mov"])
        if not NSBundle.loadNibNamed_owner_("ExportMovieAccessory", self):
            NSLog("Error -- could not load ExportMovieAccessory.")
        self.exportMovieFrames.setIntValue_(150)
        self.exportMovieFps.setIntValue_(30)
735     exportPanel.setAccessoryView_(self.exportMovieAccessory)
        path = self.fileName()
        if path:
            dirName, fileName = os.path.split(path)
            fileName, ext = os.path.splitext(fileName)
740             fileName += ".mov"
        else:
            dirName, fileName = None, "Untitled.mov"
        # If a file was already exported, use that folder as the default.
        if self.exportDir is not None:
745             dirName = self.exportDir
        exportPanel.beginSheetForDirectory_file_modalForWindow_modalDelegate_didEndSelector_contextInfo_(
            dirName,
            fileName,
            NSApp().mainWindow(),
750             self,
            "qtPanelDidEnd:returnCode:contextInfo:", 0)

def qtPanelDidEnd_returnCode_contextInfo_(self, panel, returnCode, context):
    if returnCode:
755         fname = panel.filename()
        self.exportDir = os.path.split(fname)[0] # Save the directory we exported to.
        frames = self.exportMovieFrames.intValue()
        fps = self.exportMovieFps.floatValue()
        panel.close()
760
        if frames <= 0 or fps <= 0: return
        self.doExportAsMovie_frames_fps_(fname, frames, fps)

qtPanelDidEnd_returnCode_contextInfo_ = objc.selector(qtPanelDidEnd_returnCode_contextInfo_,
765             signature="v:@:ii")

def doExportAsMovie_frames_fps_(self, fname, frames, fps):
    # Only load QTSupport when necessary.
    # QTSupport loads QTKit, which wants to establish a connection to the window
770     # server.
    # If we load QTSupport before something is on screen, the connection to the
    # window server cannot be established.

    try:
775         os.unlink(fname)
    except:
        pass
    try:
        fp = open(fname, 'w')
780         fp.close()
    except:
        errorAlert("File Error", ("Could not create file '%s'. "

```

```

                                "Perhaps it is locked or busy.") % fname)
    return

785     movie = None

    pb = ProgressBarController.alloc().init()
    pb.begin_maxval_("Generating %s frames..." % frames, frames)
790     try:
        if not self.cleanRun_newSeed_buildInterface_(self._execScript, True, True):
            return
        self._pageNumber = 1
        self._frame = 1

795         movie = QTSupport.Movie(fname, fps)
        # If the speed is set, we are dealing with animation
        if self.canvas.speed is None:
            for i in range(frames):
                if i > 0: # Run has already happened first time
                    self.fastRun_newSeed_(self._execScript, True)
                    movie.add(self.canvas)
                    self.graphicsView.setNeedsDisplay_(True)
                    pb.inc()
                self._pageNumber += 1
                self._frame += 1

            else:
                if self.namespace.has_key("setup"):
                    self.fastRun_newSeed_(self.namespace["setup"], False)
810                 for i in range(frames):
                    self.fastRun_newSeed_(self.namespace["draw"], True)
                    movie.add(self.canvas)
                    self.graphicsView.setNeedsDisplay_(True)
                    pb.inc()
                    self._pageNumber += 1
                    self._frame += 1
                if self.namespace.has_key("stop"):
                    success, output = self.boxedRun_args_(self.namespace["stop"], [])
                    self.flushOutput_(output)

820     except KeyboardInterrupt:
        pass
    pb.end()
    del pb
    movie.save()
825     self._pageNumber = 1
    self._frame = 1

    @objc.IBAction
    def printDocument_(self, sender):
830         op = NSPrintOperation.printOperationWithView_printInfo_(self.graphicsView,
                                                                    self.printInfo())
        op.runOperationModalForWindow_delegate_didRunSelector_contextInfo_(
            NSApp().mainWindow(), self, "printOperationDidRun:success:contextInfo:",
            0)

835     def printOperationDidRun_success_contextInfo_(self, op, success, info):
        if success:
            self.setPrintInfo_(op.printInfo())

840     printOperationDidRun_success_contextInfo_ = objc.selector(
        printOperationDidRun_success_contextInfo_,
        signature="v@:@ci")

    @objc.IBAction
845     def buildInterface_(self, sender):
        self.dashboardController.buildInterface_(self.vars)

```

```

def validateMenuItem_(self, menuItem):
    if menuItem.action() in ("exportAsImage:", "exportAsMovie:"):
850         return self.canvas is not None
    return True

# Zoom commands, forwarding to the graphics view.
@objc.IBAction
855 def zoomIn_(self, sender):
    if self.fullScreen is not None: return
    self.graphicsView.zoomIn_(sender)

@objc.IBAction
860 def zoomOut_(self, sender):
    if self.fullScreen is not None: return
    self.graphicsView.zoomOut_(sender)

@objc.IBAction
865 def zoomToTag_(self, sender):
    if self.fullScreen is not None: return
    self.graphicsView.zoomTo_(sender.tag() / 100.0)

@objc.IBAction
870 def zoomToFit_(self, sender):
    if self.fullScreen is not None: return
    self.graphicsView.zoomToFit_(sender)

class FullscreenWindow(NSWindow):
875     def initWithRect_(self, fullRect):
        objc.super(FullscreenWindow,
                    self).initWithContentRect_styleMask_backing_defer_(
                                fullRect,
                                NSBorderlessWindowMask,
880                                NSBackingStoreBuffered,
                                True)

        return self

    def canBecomeKeyWindow(self):
885         return True

class FullscreenView(NSView):

    def init(self):
890         super(FullscreenView, self).init()
        self.mousedown = False
        self.keydown = False
        self.key = None
        self.keycode = None
895         self.scrollwheel = False
        self.wheeldelta = 0.0
        return self

    def setCanvas_(self, canvas):
900         self.canvas = canvas
        self.setNeedsDisplay_(True)
        if not hasattr(self, "screenRect"):
            self.screenRect = NSScreen.mainScreen().frame()
            cw, ch = self.canvas.size
905             sw, sh = self.screenRect[1]
            self.scalingFactor = calc_scaling_factor(cw, ch, sw, sh)
            nw, nh = cw * self.scalingFactor, ch * self.scalingFactor
            self.scaledSize = nw, nh
            self.dx = (sw - nw) / 2.0
910             self.dy = (sh - nh) / 2.0

```



```

def drawRect_(self, rect):
    NSGraphicsContext.currentContext().saveGraphicsState()
    NSColor.blackColor().set()
915    NSRectFill(rect)
    if self.canvas is not None:
        t = NSAffineTransform.transform()
        t.translateXBy_yBy_(self.dx, self.dy)
        t.scaleBy_(self.scalingFactor)
920        t.concat()
        clip = NSBezierPath.bezierPathWithRect_(
            ((0, 0), (self.canvas.width, self.canvas.height)) )
        clip.addClip()
        self.canvas.draw()
925    NSGraphicsContext.currentContext().restoreGraphicsState()

def isFlipped(self):
    return True

930 def mouseDown_(self, event):
    self.mousedown = True

def mouseUp_(self, event):
    self.mousedown = False
935

def keyDown_(self, event):
    self.keydown = True
    self.key = event.characters()
    self.keycode = event.keyCode()
940

def keyUp_(self, event):
    self.keydown = False
    self.key = event.characters()
    self.keycode = event.keyCode()
945

def scrollWheel_(self, event):
    self.scrollwheel = True
    self.wheeldelta = event.deltaY()

950 def canBecomeKeyView(self):
    return True

def acceptsFirstResponder(self):
    return True
955

def calc_scaling_factor(width, height, maxwidth, maxheight):
    return min(float(maxwidth) / width, float(maxheight) / height)

class ZoomPanel(NSView):
960     pass

# class defined in NodeBoxGraphicsView.xib
class NodeBoxGraphicsView(NSView):
    document = objc.IBOutlet()
965    zoomLevel = objc.IBOutlet()
    zoomField = objc.IBOutlet()
    zoomSlider = objc.IBOutlet()

# The zoom levels are 10%, 25%, 50%, 75%, 100%, 200% and so on up to 2000%.
970    zoomLevels = [0.1, 0.25, 0.5, 0.75]
    zoom = 1.0
    while zoom <= 20.0:
        zoomLevels.append(zoom)
        zoom += 1.0

```

```

975     def awakeFromNib(self):
        self.canvas = None
        self._dirty = False
        self.mousedown = False
980     self.keydown = False
        self.key = None
        self.keycode = None
        self.scrollwheel = False
        self.wheeldelta = 0.0
985     self._zoom = 1.0
        self.setFrameSize_( (graphics.DEFAULT_WIDTH, graphics.DEFAULT_HEIGHT) )
        self.setFocusRingType_(NSFocusRingTypeExterior)
        if self.superview() is not None:
            self.superview().setBackgroundColor_(VERY_LIGHT_GRAY)
990
        def setCanvas_(self, canvas):
            self.canvas = canvas
            if canvas is not None:
                w, h = self.canvas.size
995                self.setFrameSize_([w*self._zoom, h*self._zoom])
            self.markDirty()

        def getZoom(self):
            return self._zoom
1000    def setZoom_(self, zoom):
        self._zoom = zoom
        self.zoomLevel.setTitle_("%i%" % (self._zoom * 100.0))
        self.zoomSlider.setFloatValue_(self._zoom * 100.0)
        self.setCanvas_(self.canvas)
1005    zoom = property(getZoom, setZoom_)

    @objc.IBAction
    def dragZoom_(self, sender):
        self.zoom = self.zoomSlider.floatValue() / 100.0
1010    self.setCanvas_(self.canvas)

    def findNearestZoomIndex_(self, zoom):
        """Returns the nearest zoom level, and whether we found a direct, exact
        match or a fuzzy match."""
1015    try: # Search for a direct hit first.
        idx = self.zoomLevels.index(zoom)
        return idx, True
    except ValueError: # Can't find the zoom level, try looking at the indexes.
        idx = 0
1020    try:
        while self.zoomLevels[idx] < zoom:
            idx += 1
        except KeyError: # End of the list
            idx = len(self.zoomLevels) - 1 # Just return the last index.
1025    return idx, False

    @objc.IBAction
    def zoomIn_(self, sender):
        idx, direct = self.findNearestZoomIndex_(self.zoom)
1030    # Direct hits are perfect, but indirect hits require a bit of help.
        # Because of the way indirect hits are calculated, they are already
        # rounded up to the upper zoom level; this means we don't need to add 1.
        if direct:
            idx += 1
1035    idx = max(min(idx, len(self.zoomLevels)-1), 0)
        self.zoom = self.zoomLevels[idx]

    @objc.IBAction

```

```

1040     def zoomOut_(self, sender):
        idx, direct = self.findNearestZoomIndex_(self.zoom)
        idx -= 1
        idx = max(min(idx, len(self.zoomLevels)-1), 0)
        self.zoom = self.zoomLevels[idx]

1045     @objc.IBAction
    def resetZoom_(self, sender):
        self.zoom = 1.0

    def zoomTo_(self, zoom):
1050         self.zoom = zoom

    @objc.IBAction
    def zoomToFit_(self, sender):
        w, h = self.canvas.size
1055         fw, fh = self.superview().frame()[1]
        factor = min(fw / w, fh / h)
        self.zoom = factor

    def markDirty(self, redraw=True):
1060         self._dirty = True
        if redraw:
            self.setNeedsDisplay_(True)

    def setFrameSize_(self, size):
1065         self._image = None
        NSView.setFrameSize_(self, size)

    def isOpaque(self):
        return False

1070    def isFlipped(self):
        return True

    def drawRect_(self, rect):
1075         if self.canvas is not None:
            NSGraphicsContext.currentContext().saveGraphicsState()
            try:
                if self.zoom != 1.0:
1080                     t = NSAffineTransform.transform()
                     t.scaleBy_(self.zoom)
                     t.concat()
                     clip = NSBezierPath.bezierPathWithRect_( ( (0, 0),
                                                                    (self.canvas.width,
                                                                    self.canvas.height)) )
1085                     clip.addClip()
                     self.canvas.draw()
            except:
                # A lot of code just to display the error in the output view.
                etype, value, tb = sys.exc_info()
1090                 if tb.tb_next is not None:
                     tb = tb.tb_next # skip the frame doing the exec
                     traceback.print_exception(etype, value, tb)
                     data = "".join(traceback.format_exception(etype, value, tb))
                     attrs = PyDETextView.getBasicTextAttributes()
1095                     attrs[NSForegroundColorAttributeName] = NSColor.redColor()
                     outputView = self.document.outputView
                     outputView.setSelectedRange_((outputView.textStorage().length(), 0))
                     outputView.setTypingAttributes_(attrs)
                     outputView.insertText_(data)
1100                 NSGraphicsContext.currentContext().restoreGraphicsState()

    def _updateImage(self):

```

```

        if self._dirty:
            self._image = self.canvas._nsImage
1105         self._dirty = False

# pasteboard delegate method
def pasteboard_provideDataForType_(self, pboard, type):
    if NSPDFPboardType:
1110         pboard.setData_forType_(self.pdfData, NSPDFPboardType)
    elif NSPostScriptPboardType:
        pboard.setData_forType_(self.epsData, NSPostScriptPboardType)
    elif NSTIFFPboardType:
        pboard.setData_forType_(self.tiffData, NSTIFFPboardType)
1115

def _get_pdfData(self):
    if self.canvas:
        return self.canvas._getImageData('pdf')
pdfData = property(_get_pdfData)
1120

def _get_epsData(self):
    if self.canvas:
        return self.canvas._getImageData('eps')
epsData = property(_get_epsData)
1125

def _get_tiffData(self):
    return self.canvas._getImageData('tiff')
tiffData = property(_get_tiffData)

1130 def _get_pngData(self):
    return self.canvas._getImageData('png')
pngData = property(_get_pngData)

def _get_gifData(self):
1135     return self.canvas._getImageData('gif')
gifData = property(_get_gifData)

def _get_jpegData(self):
    return self.canvas._getImageData('jpeg')
1140 jpegData = property(_get_jpegData)

def mouseDown_(self, event):
    self.mousedown = True

1145 def mouseUp_(self, event):
    self.mousedown = False

def keyDown_(self, event):
    self.keydown = True
1150     self.key = event.characters()
    self.keycode = event.keyCode()

def keyUp_(self, event):
    self.keydown = False
1155     self.key = event.characters()
    self.keycode = event.keyCode()

def scrollWheel_(self, event):
    NSResponder.scrollWheel_(self, event)
1160     self.scrollwheel = True
    self.wheelDelta = event.deltaY()

def canBecomeKeyView(self):
    return True
1165

def acceptsFirstResponder(self):

```

```

        return True

class NodeBoxAppDelegate(NSObject):
1170     def awakeFromNib(self):
        self._prefsController = None
        libpath = LibraryFolder()

1175     @objc.IBAction
    def showPreferencesPanel_(self, sender):
        if self._prefsController is None:
            self._prefsController = NodeBoxPreferencesController.alloc().init()
            self._prefsController.showWindow_(sender)

1180     @objc.IBAction
    def generateCode_(self, sender):
        """Generate a piece of NodeBox code using OttoBot"""
        # from nodebox.util.ottobot import genProgram
1185        controller = NSDocumentController.sharedDocumentController()
        doc = controller.newDocument_(sender)
        doc = controller.currentDocument()
        doc.textView.setString_(genProgram())
        doc.runScript()

1190     @objc.IBAction
    def showHelp_(self, sender):
        url = NSURL.URLWithString_("http://nodebox.net/code/index.php/Reference")
        NSWorkspace.sharedWorkspace().openURL_(url)

1195     @objc.IBAction
    def showSite_(self, sender):
        url = NSURL.URLWithString_("http://nodebox.net/")
        NSWorkspace.sharedWorkspace().openURL_(url)

1200     @objc.IBAction
    def showLibrary_(self, sender):
        libpath = LibraryFolder()
        url = NSURL.fileURLWithPath_( makeunicode(libpath.libDir) )
1205        NSWorkspace.sharedWorkspace().openURL_(url)

    def applicationWillTerminate_(self, note):
        # import atexit
        atexit._run_exitfuncs()

```

## nodebox/gui/mac/AskString.py

```

__all__ = ["AskString"]

import objc

5 import Foundation

import AppKit
NSApp = AppKit.NSApp
# class defined in AskString.xib
10 class AskStringWindowController(AppKit.NSWindowController):
    questionLabel = objc.IBOutlet()
    textField = objc.IBOutlet()

    def __new__(cls, question, resultCallback, default="", parentWindow=None):
15        self = cls.alloc().initWithWindowNibName_("AskString")
        self.question = question
        self.resultCallback = resultCallback

```

```

        self.default = default
        self.parentWindow = parentWindow
20     if self.parentWindow is None:
            self.window().setFrameUsingName_("AskStringPanel")
            self.setWindowFrameAutosaveName_("AskStringPanel")
            self.showWindow_(self)
        else:
25         NSApp().beginSheet_modalForWindow_modalDelegate_didEndSelector_contextInfo_(
            self.window(), self.parentWindow, None, None, 0)
        self.retain()
        return self

30     def windowWillClose_(self, notification):
        self.autorelease()

        def awakeFromNib(self):
            self.questionLabel.setStringValue_(self.question)
35         self.textField.setStringValue_(self.default)

        def done(self):
            if self.parentWindow is None:
                self.close()
40         else:
            sheet = self.window()
            NSApp().endSheet_(sheet)
            sheet.orderOut_(self)

45     def ok_(self, sender):
        value = self.textField.stringValue()
        self.done()
        self.resultCallback(value)

50     def cancel_(self, sender):
        self.done()
        self.resultCallback(None)

    def AskString(question, resultCallback, default="", parentWindow=None):
55     AskStringWindowController(question, resultCallback, default, parentWindow)

```

## nodebox/gui/mac/dashboard.py

```

import pdb
import AppKit

NSObject = AppKit.NSObject
5 NSFont = AppKit.NSFont
NSMiniControlSize = AppKit.NSMiniControlSize
NSOnState = AppKit.NSOnState
NSOffState = AppKit.NSOffState
NSTextField = AppKit.NSTextField
10 NSRightTextAlignment = AppKit.NSRightTextAlignment
NSSlider = AppKit.NSSlider
NSMiniControlSize = AppKit.NSMiniControlSize
NSGraphiteControlTint = AppKit.NSGraphiteControlTint
NSButton = AppKit.NSButton
15 NSSwitchButton = AppKit.NSSwitchButton
NSSmallControlSize = AppKit.NSSmallControlSize
NSPopUpButton = AppKit.NSPopUpButton

import objc
20
from nodebox import graphics

```

```

# just to make the next lines print better
smfontsize = NSFont.smallSystemFontSize()
25 smctrlsize = NSFont.systemFontSizeForControlSize_(NSMiniControlSize)

SMALL_FONT = NSFont.systemFontOfSize_(smfontsize)
MINI_FONT = NSFont.systemFontOfSize_(smctrlsize)

30 # class defined in NodeBoxDocument.xib
class DashboardController(NSObject):
    document = objc.IBOutlet()
    documentWindow = objc.IBOutlet()
    panel = objc.IBOutlet()

35
    def clearInterface(self):
        for s in list(self.panel.contentView().subviews()):
            s.removeFromSuperview()

40
    def numberChanged_(self, sender):
        var = self.document.vars[sender.tag()]
        var.value = sender.floatValue()
        if var.handler is not None:
            args = [var.value, var.name]
45             if var.handler.func_code.co_argcount < 2:
                args = [var.value]
            self.document.fastRun_newSeed_args_(var.handler, False, args)
        else:
            self.document.runScript(compile=False, newSeed=False)

50
    def textChanged_(self, sender):
        var = self.document.vars[sender.tag()]
        var.value = sender.stringValue()
        if var.handler is not None:
55             args = [var.value, var.name]
            if var.handler.func_code.co_argcount < 2:
                args = [var.value]
            self.document.fastRun_newSeed_args_(var.handler, False, args)
        else:
60             self.document.runScript(compile=False, newSeed=False)

    def booleanChanged_(self, sender):
        var = self.document.vars[sender.tag()]
        if sender.state() == NSOnState:
65             var.value = True
        else:
            var.value = False
        if var.handler is not None:
            args = [var.value, var.name]
70             if var.handler.func_code.co_argcount < 2:
                args = [var.value]
            self.document.fastRun_newSeed_args_(var.handler, False, args)
        else:
75             self.document.runScript(compile=False, newSeed=False)

    def buttonClicked_(self, sender):
        var = self.document.vars[sender.tag()]
        # self.document.fastRun_newSeed_(self.document.namespace[var.name], True)
        #self.document.runFunction_(var.name)
80         if var.handler is not None:
            args = ["", var.name]
            if var.handler.func_code.co_argcount < 2:
                args = [var.value]
            self.document.fastRun_newSeed_args_(var.handler, False, args)
85         else:
            self.document.runScript(compile=False, newSeed=False)

```

```

def menuSelected_(self, sender):
    var = self.document.vars[sender.tag()]
90    sel = sender.titleOfSelectedItem()
    var.value = sel
    fn = var.handler
    if var.handler:
        args = [sel,var.name]
95        if var.handler.func_code.co_argcount < 2:
            args = [sel]
        self.document.fastRun_newSeed_args_(fn, False, args)
    #self.document.runFunction_(var.name)

100 def buildInterface_(self, variables):
    self.vars = variables
    self.clearInterface()
    if len(self.vars) > 0:
        self.panel.orderFront_(None)
105 else:
        self.panel.orderOut_(None)
    return

    # Set the title of the parameter panel to the title of the window
110 self.panel.setTitle_(self.documentWindow.title())

    (px,py),(pw,ph) = self.panel.frame()
    # Height of the window. Each element has a height of 21.
    # The extra "fluff" is 38 pixels.
115 ph = len(self.vars) * 21 + 54
    # Start of first element
    # First element is the height minus the fluff.
    y = ph - 49
    cnt = 0
120 for v in self.vars:
    if v.type == graphics.NUMBER:
        self.addLabel_y_c_(v, y, cnt)
        self.addSlider_y_c_(v, y, cnt)

125 elif v.type == graphics.TEXT:
        self.addLabel_y_c_(v, y, cnt)
        self.addTextField_y_c_(v, y, cnt)

    elif v.type == graphics.BOOLEAN:
130 self.addSwitch_y_c_(v, y, cnt)

    elif v.type == graphics.BUTTON:
        self.addButton_y_c_(v, y, cnt)

135 elif v.type == graphics.MENU:
        self.addLabel_y_c_(v, y, cnt)
        self.addMenu_y_c_(v, y, cnt)
        y -= 21
        cnt += 1
140 self.panel.setFrame_display_animate_((px,py),(pw,ph)), True, True )

def addLabel_y_c_(self, v, y, cnt):
    control = NSTextField.alloc().init()
    control.setFrame_(((0,y),(100,13)))
145 control.setStringValue_(v.name + ":")
    control.setAlignment_(NSRightTextAlignment)
    control.setEditable_(False)
    control.setBorder_(False)
    control.setDrawsBackground_(False)
150 control.setFont_(SMALL_FONT)

```



```

        self.panel.contentView().addSubview_(control)

    def addSlider_y_c_(self, v, y, cnt):
        control = NSSlider.alloc().init()
155     control.setMaxValue_(v.max)
        control.setMinValue_(v.min)
        control.setFloatValue_(v.value)
        control.setFrame_(((108,y-1),(172,13)))
        control.cell().setControlSize_(NSMiniControlSize)
160     control.cell().setControlTint_(NSGraphiteControlTint)
        control.setContinuous_(True)
        control.setTarget_(self)
        control.setTag_(cnt)
        control.setAction_(objc.selector(self.numberChanged_, signature="v:@@"))
165     self.panel.contentView().addSubview_(control)

    def addTextField_y_c_(self, v, y, cnt):
        control = NSTextField.alloc().init()
        control.setStringValue_(v.value)
170     control.setFrame_(((108,y-2),(172,15)))
        control.cell().setControlSize_(NSMiniControlSize)
        control.cell().setControlTint_(NSGraphiteControlTint)
        control.setFont_(MINI_FONT)
        control.setTarget_(self)
175     control.setTag_(cnt)
        control.setAction_(objc.selector(self.textChanged_, signature="v:@@"))
        self.panel.contentView().addSubview_(control)

    def addSwitch_y_c_(self, v, y, cnt):
180     control = NSButton.alloc().init()
        control.setButtonType_(NSSwitchButton)
        if v.value:
            control.setState_(NSOnState)
        else:
185     control.setState_(NSOffState)
        control.setFrame_(((108,y-2),(172,16)))
        control.setTitle_(v.name)
        control.setFont_(SMALL_FONT)
        control.cell().setControlSize_(NSSmallControlSize)
190     control.cell().setControlTint_(NSGraphiteControlTint)
        control.setTarget_(self)
        control.setTag_(cnt)
        control.setAction_(objc.selector(self.booleanChanged_, signature="v:@@"))
        self.panel.contentView().addSubview_(control)
195

    def addButton_y_c_(self, v, y, cnt):
        control = NSButton.alloc().init()
        control.setFrame_(((108, y-2),(172,16)))
        control.setTitle_(v.name)
200     control.setBezelStyle_(1)
        control.setFont_(SMALL_FONT)
        control.cell().setControlSize_(NSMiniControlSize)
        control.cell().setControlTint_(NSGraphiteControlTint)
        control.setTarget_(self)
205     control.setTag_(cnt)
        control.setAction_(objc.selector(self.buttonClicked_, signature="v:@@"))
        self.panel.contentView().addSubview_(control)

    def addMenu_y_c_(self, v, y, cnt):
210     control = NSPopUpButton.alloc().init()
        control.setFrame_(((108, y-2),(172,16)))
        control.setPullsDown_(False)
        control.removeAllItems()
        if v.menuitems is not None:

```

```

215         for title in v.menuitems:
            control.addItemWithTitle_( title )
            control.setTitle_(v.value)
            control.synchronizeTitleAndSelectedItem()
            control.setBezelStyle_(1)
220         control.setFont_(SMALL_FONT)
            control.cell().setControlSize_(NSMiniControlSize)
            control.cell().setControlTint_(NSGraphiteControlTint)
            control.setTarget_(self)
            control.setTag_(cnt)
225         control.setAction_(objc.selector(self.menuSelected_, signature="v:@@"))
            self.panel.contentView().addSubview_(control)

```

## nodebox/gui/mac/preferences.py

```

import sys
import os
# import pdb

5 import objc

import AppKit
NSWindowController = AppKit.NSWindowController
NSForegroundColorAttributeName = AppKit.NSForegroundColorAttributeName
10 NSNotificationCenter = AppKit.NSNotificationCenter
NSFontManager = AppKit.NSFontManager
NSFontAttributeName = AppKit.NSFontAttributeName
NSUserDefaults = AppKit.NSUserDefaults
NSOpenGLPanel = AppKit.NSOpenGLPanel
15

from PyDETextView import getBasicTextAttributes, getSyntaxTextAttributes
from PyDETextView import setTextFont, setBasicTextAttributes, setSyntaxTextAttributes

class LibraryFolder(object):
20     def __init__(self):
        prefpath = ""
        try:
            prefpath = NSUserDefaults.standardUserDefaults().objectForKey_("libraryPath")
        except Exception, err:
25             print "LibraryFolder: prefpath:", repr(prefpath)
            prefpath = ""
        stdpath = os.path.join(os.getenv("HOME"), "Library", "Application Support",
                               "NodeBox")

30         if prefpath and os.path.exists( prefpath ):
            self.libDir = prefpath
            NSUserDefaults.standardUserDefaults().setObject_forKey_( self.libDir,
                                                                       "libraryPath")
        else:
35             self.libDir = stdpath
            try:
                if not os.path.exists(self.libDir):
                    os.mkdir(libDir)
            except OSError:
40                 pass
            except IOError:
                pass

        # class defined in NodeBoxPreferences.xib
45 class NodeBoxPreferencesController(NSWindowController):
    commentsColorWell = objc.IBOutlet()
    fontPreview = objc.IBOutlet()
    libraryPath = objc.IBOutlet()

```

```

funcClassColorWell = objc.IBOutlet()
50 keywordsColorWell = objc.IBOutlet()
stringsColorWell = objc.IBOutlet()

def init(self):
    self = self.initWithWindowNibName_("NodeBoxPreferences")
55 self.setWindowFrameAutosaveName_("NodeBoxPreferencesPanel")
    self.timer = None
    return self

def awakeFromNib(self):
60 self.textFontChanged_(None)
    syntaxAttrs = syntaxAttrs = getSyntaxTextAttributes()
    self.stringsColorWell.setColor_(syntaxAttrs["string"][NSForegroundColorAttributeName])
    self.keywordsColorWell.setColor_(syntaxAttrs["keyword"][NSForegroundColorAttributeName])
    self.funcClassColorWell.setColor_(syntaxAttrs["identifier"][NSForegroundColorAttributeName])
65 self.commentsColorWell.setColor_(syntaxAttrs["comment"][NSForegroundColorAttributeName])
    libpath = LibraryFolder()
    self.libraryPath.setStringValue_( libpath.libDir )

    nc = NSNotificationCenter.defaultCenter()
70 nc.addObserver_selector_name_object_(self, "textFontChanged:", "PyDETextFontChanged", None)

def windowWillClose_(self, notification):
    fm = NSFontManager.sharedFontManager()
    fp = fm.fontPanel_(False)
75 if fp is not None:
    fp.setDelegate_(None)
    fp.close()

@objc.IBAction
80 def updateColors_(self, sender):
    if self.timer is not None:
        self.timer.invalidate()
    self.timer = NSTimer.scheduledTimerWithTimeInterval_target_selector_userInfo_repeats_(
        1.0, self, "timeToUpdateTheColors:", None, False)
85

def timeToUpdateTheColors_(self, sender):
    syntaxAttrs = getSyntaxTextAttributes()
    syntaxAttrs["string"][NSForegroundColorAttributeName] = self.stringsColorWell.color()
    syntaxAttrs["keyword"][NSForegroundColorAttributeName] = self.keywordsColorWell.color()
90 syntaxAttrs["identifier"][NSForegroundColorAttributeName] = self.funcClassColorWell.color()
    syntaxAttrs["comment"][NSForegroundColorAttributeName] = self.commentsColorWell.color()
    setSyntaxTextAttributes(syntaxAttrs)

@objc.IBAction
95 def chooseFont_(self, sender):
    fm = NSFontManager.sharedFontManager()
    basicAttrs = getBasicTextAttributes()
    fm.setSelectedFont_isMultiple_(basicAttrs[NSFontAttributeName], False)
    fm.orderFrontFontPanel_(sender)
100 fp = fm.fontPanel_(False)
    fp.setDelegate_(self)

@objc.IBAction
def chooseLibrary_(self, sender):
105 panel = NSOpenPanel.openPanel()
    panel.setCanChooseFiles_(False)
    panel.setCanChooseDirectories_(True)
    panel.setAllowsMultipleSelection_(False)
    rval = panel.runModalForTypes_([])
110 if rval:
    s = [t for t in panel.fileNames()]
    s = s[0]

```

```

        NSUserDefaults.standardUserDefaults().setObject_forKey_( s,
                                                                    "libraryPath")
115     libpath = LibraryFolder()
        self.libraryPath.setStringValue_( libpath.libDir )

    @objc.IBAction
    def changeFont_(self, sender):
120     oldFont = getBasicTextAttributes()[NSFontAttributeName]
        newFont = sender.convertFont_(oldFont)
        if oldFont != newFont:
            setTextFont(newFont)

125     def textFontChanged_(self, notification):
        basicAttrs = getBasicTextAttributes()
        font = basicAttrs[NSFontAttributeName]
        self.fontPreview.setFont_(font)
        size = font.pointSize()
130     if size == int(size):
        size = int(size)
        s = u"%s %s" % (font.displayName(), size)
        self.fontPreview.setStringValue_(s)

```

### nodebox/gui/mac/progressbar.py

```

import objc
import AppKit
NSDefaultRunLoopMode = AppKit.NSDefaultRunLoopMode

5 class ProgressBarController(AppKit.NSWindowController):
    messageField = objc.IBOutlet()
    progressBar = objc.IBOutlet()

    def init(self):
10     AppKit.NSBundle.loadNibNamed_owner_("ProgressBarSheet", self)
        return self

    def begin_maxval_(self, message, maxval):
        self.value = 0
15     self.message = message
        self.maxval = maxval
        self.progressBar.setMaxValue_(self.maxval)
        self.messageField.cell().setTitle_(self.message)
        parentWindow = AppKit.NSApp().keyWindow()
20     AppKit.NSApp().beginSheet_modalForWindow_modalDelegate_didEndSelector_contextInfo_(self.window(),
        parentWindow, self, None, None, None)

    def inc(self):
        self.value += 1
        self.progressBar.setDoubleValue_(self.value)
25     date = AppKit.NSDate.dateWithTimeIntervalSinceNow_(0.01)
        AppKit.NSRunLoop.currentRunLoop().acceptInputForMode_beforeDate_(NSDefaultRunLoopMode, date)

    def end(self):
        AppKit.NSApp().endSheet_(self.window())
30     self.window().orderOut_(self)

```

### nodebox/gui/mac/PyDETextView.py

```

from bisect import bisect
import re
import objc
super = objc.super
5

```

```

import AppKit

NSBackgroundColorAttributeName = AppKit.NSBackgroundColorAttributeName
NSBeep = AppKit.NSBeep
10 NSColor = AppKit.NSColor
NSCommandKeyMask = AppKit.NSCommandKeyMask
NSDictionary = AppKit.NSDictionary
NSEvent = AppKit.NSEvent
NSFont = AppKit.NSFont
15 NSFontAttributeName = AppKit.NSFontAttributeName
NSForegroundColorAttributeName = AppKit.NSForegroundColorAttributeName
NSLigatureAttributeName = AppKit.NSLigatureAttributeName
NSLiteralSearch = AppKit.NSLiteralSearch
NSNotificationCenter = AppKit.NSNotificationCenter
20 NSObject = AppKit.NSObject
NSStringPboardType = AppKit.NSStringPboardType
NSTextStorage = AppKit.NSTextStorage
NSTextStorageEditedCharacters = AppKit.NSTextStorageEditedCharacters
NSTextView = AppKit.NSTextView
25 NSURL = AppKit.NSURL
NSURLPboardType = AppKit.NSURLPboardType
NSViewWidthSizable = AppKit.NSViewWidthSizable

NSCalibratedRGBColorSpace = AppKit.NSCalibratedRGBColorSpace
30 NSUserDefaults = AppKit.NSUserDefaults

import nodebox.PyFontify
fontify = nodebox.PyFontify.fontify

35 from nodebox.gui.mac.ValueLadder import ValueLadder

from nodebox.util import _copy_attr, _copy_attrs, makeunicode

whiteRE = re.compile(r"[ \t]+")
40 commentRE = re.compile(r"[ \t]*(#)")

def findWhitespace(s, pos=0):
    m = whiteRE.match(s, pos)
    if m is None:
45         return pos
    return m.end()

stringPat = r"q[^\q\n]*(\\[\000-\377][^\q\n]*)*q"
stringOrCommentPat = stringPat.replace("q", "'") + "|" + stringPat.replace('q', "'") + "|#.*"
50 stringOrCommentRE = re.compile(stringOrCommentPat)

def removeStringsAndComments(s):
    items = []
    while 1:
55         m = stringOrCommentRE.search(s)
        if m:
            start = m.start()
            end = m.end()
            items.append(s[:start])
60             if s[start] != "#":
                items.append("X" * (end - start)) # X-out strings
                s = s[end:]
            else:
                items.append(s)
65             break
    return "".join(items)

class PyDETextView(NSTextView):

```

```

70 document = objc.IBOutlet()

def awakeFromNib(self):
    # Can't use a subclass of NSTextView as an NSTextView in IB,
    # so we need to set some attributes programmatically
75 scrollView = self.superview().superview()
    self setFrame_(((0, 0), scrollView.contentSize()))
    self.setAutoresizingMask_(NSViewWidthSizable)
    self.textContainer().setWidthTracksTextView_(True)
    self.setAllowsUndo_(True)
80 self.setRichText_(False)
    self.setTypingAttributes_(getBasicTextAttributes())
    self.setUsesFindPanel_(True)
    self.usesTabs = 0
    self.indentSize = 4
85 self._string = self.textStorage().mutableString().nsstring()
    self._storageDelegate = PyDETextStorageDelegate(self.textStorage())

    # FDB: no wrapping
    # Thanks to http://cocoa.mamasam.com/COCOADEV/2003/12/2/80304.php
90 scrollView = self.enclosingScrollView()
    scrollView.setHasHorizontalScroller_(True)
    self.setHorizontallyResizable_(True)
    layoutSize = self.maxSize()
    layoutSize = (layoutSize[1], layoutSize[1])
95 self.setMaxSize_(layoutSize)
    self.textContainer().setWidthTracksTextView_(False)
    self.textContainer().setContainerSize_(layoutSize)

    # FDB: value ladder
100 self.valueLadder = None

    nc = NSNotificationCenter.defaultCenter()
    nc.addObserver_selector_name_object_(self, "textFontChanged:",
                                         "PyDETextFontChanged", None)
105

def drawRect_(self, rect):
    NSTextView.drawRect_(self, rect)
    if self.valueLadder is not None and self.valueLadder.visible:
        self.valueLadder.draw()
110

def hideValueLadder(self):
    if self.valueLadder is not None:
        self.valueLadder.hide()
        if self.valueLadder.dirty:
115 self.document.updateChangeCount_(True)
    self.valueLadder = None

def mouseUp_(self, event):
    self.hideValueLadder()
120 NSTextView.mouseUp_(self, event)

def mouseDragged_(self, event):
    if self.valueLadder is not None:
        self.valueLadder.mouseDragged_(event)
125 else:
        NSTextView.mouseDragged_(self, event)

def mouseDown_(self, event):
    if event.modifierFlags() & NSCommandKeyMask:
130 screenPoint = NSEvent.mouseLocation()
        viewPoint = self.superview().convertPoint_fromView_(event.locationInWindow(),
                                                             self.window().contentView())

```

```

135         c = self.characterIndexForPoint_(screenPoint)

        txt = self.string()
        # XXX move code into ValueLadder
        try:
            if txt[c] in "1234567890.":
140                 # Find full number
                 begin = c
                 end = c
                 try:
145                     while txt[begin-1] in "1234567890.":
                         begin-=1
                     except IndexError:
                         pass
                 try:
150                     while txt[end+1] in "1234567890.":
                         end+=1
                     except IndexError:
                         pass
                 end+=1
                 self.valueLadder = ValueLadder(self,
155                                         eval(txt[begin:end]),
                                         (begin,end),
                                         screenPoint, viewPoint)

            except IndexError:
                pass
160        else:
            NSTextView.mouseDown_(self,event)

    def acceptableDragTypes(self):
        return list(super(PyDETextView, self).acceptableDragTypes()) + [NSURLPboardType]
165

    def draggingEntered_(self, dragInfo):
        pboard = dragInfo.draggingPasteboard()
        types = pboard.types()
        if NSURLPboardType in pboard.types():
170            # Convert URL to string, replace pboard entry, let NSTextView
            # handle the drop as if it were a plain text drop.
            url = NSURL.URLFromPasteboard_(pboard)
            if url.isFileURL():
                s = url.path()
175            else:
                s = url.absoluteString()
                s = 'u"%s"' % s.replace('"', '\\\\"')
                pboard.declareTypes_owner_([NSStringPboardType], self)
                pboard.setString_forType_(s, NSStringPboardType)
180        return super(PyDETextView, self).draggingEntered_(dragInfo)

    def _cleanup(self):
        # delete two circular references
        del self._string
185        del self._storageDelegate

    def __del__(self):
        nc = NSNotificationCenter defaultCenter()
        nc.removeObserver_name_object_(self, "PyDETextFontChanged", None)
190

@objc.IBAction
def jumpToLine_(self, sender):
    from nodebox.gui.mac.AskString import AskString
    AskString("Jump to line number:", self.jumpToLineCallback_,
195                parentWindow=self.window())

def jumpToLineCallback_(self, value):

```

```

        if value is None:
            return # user cancelled
200    try:
        lineNo = int(value.strip())
    except ValueError:
        NSBeep()
    else:
205        self.jumpToLineNr_(lineNo)

def jumpToLineNr_(self, lineNo):
    lines = self.textStorage().string().splitlines()
    lineNo = min(max(0, lineNo - 1), len(lines))
210    length_of_prevs = sum([len(line)+1 for line in lines[:lineNo]])
    curlen = len(lines[lineNo])
    rng = (length_of_prevs, curlen)
    self.setSelectedRange_(rng)
    self.scrollRangeToVisible_(rng)
215    self.setNeedsDisplay_(True)

def textFontChanged_(self, notification):
    basicAttrs = getBasicTextAttributes()
    self.setTypingAttributes_(basicAttrs)
220    # Somehow the next line is needed, we crash otherwise :(
    self.layoutManager().invalidateDisplayForCharacterRange_(
                                                (0, self._string.length()))
    self._storageDelegate.textFontChanged_(notification)

225    def setTextStorage_str_tabs_(self, storage, string, usesTabs):
        storage.addLayoutManager_(self.layoutManager())
        self._string = string
        self.usesTabs = usesTabs

230    @objc.IBAction
    def changeFont_(self, sender):
        # Change the font through the user prefs API, we'll get notified
        # through textFontChanged_
        font = getBasicTextAttributes()[NSFontAttributeName]
235        font = sender.convertFont_(font)
        setTextFont(font)

    def getLinesForRange_(self, rng):
        rng = self._string.lineRangeForRange_(rng)
240        return self._string.substringWithRange_(rng), rng

    def getIndent(self):
        if self.usesTabs:
            return "\t"
245        else:
            return self.indentSize * " "

    def drawInsertionPointInRect_color_turnedOn_(self, pt, color, on):
        self.insertionPoint = pt
250        super(PyDETextView, self).drawInsertionPointInRect_color_turnedOn_(pt, color, on)

    def keyDown_(self, event):
        super(PyDETextView, self).keyDown_(event)
        char = event.characters()[0]
255        if char in ")}":
            selRng = self.selectedRange()
            line, lineRng, pos = self.findMatchingIndex_paren_(selRng[0] - 1, char)
            if pos is not None:
                self.balanceParens_(lineRng[0] + pos)
260
    def balanceParens_(self, index):

```



```

    rng = (index, 1)
    oldAttrs, effRng = self.textStorage().attributesAtIndex_effectiveRange_(index,
                                                                                   None)
265     balancingAttrs = {
        NSBackgroundColorAttributeName: NSColor.selectedTextBackgroundColor()
    }
    # Must use temp attrs otherwise the attrs get reset right away due to colorizing.
    self.layoutManager().setTemporaryAttributes_forCharacterRange_(balancingAttrs,
                                                                    rng)
270     self.performSelector_withObject_afterDelay_("resetBalanceParens:",
        (oldAttrs, effRng), 0.2)

def resetBalanceParens_(self, (attrs, rng)):
275     self.layoutManager().setTemporaryAttributes_forCharacterRange_(attrs, rng)

def iterLinesBackwards_maxChars_(self, end, maxChars):
    begin = max(0, end - maxChars)
    if end > 0:
280         prevChar = self._string.characterAtIndex_(end - 1)
        if prevChar == "\n":
            end += 1
    lines, linesRng = self.getLinesForRange_((begin, end - begin))
    lines = lines[:end - linesRng[0]]
285     linesRng = (linesRng[0], len(lines))
    lines = lines.splitlines(True)
    lines.reverse()
    for line in lines:
        nChars = len(line)
290         yield line, (end - nChars, nChars)
        end -= nChars
    assert end == linesRng[0]

def findMatchingIndex_paren_(self, index, paren):
295     openToCloseMap = {"(": ")", "[": "]", "{": "}"}
    if paren:
        stack = [paren]
    else:
        stack = []
300     line, lineRng, pos = None, None, None
    for line, lineRng in self.iterLinesBackwards_maxChars_(index, 8192):
        line = removeStringsAndComments(line)
        pos = None
        for i in range(len(line)-1, -1, -1):
305             c = line[i]
            if c in ")]}":
                stack.append(c)
            elif c in "[{":
                if not stack:
310                     if not paren:
                        pos = i
                        break
                elif stack[-1] != openToCloseMap[c]:
                    # mismatch
                    stack = []
                    break
                else:
                    stack.pop()
                    if paren and not stack:
320                         pos = i
                        break
            if not stack:
                break
    return line, lineRng, pos
325

```

```

def insertNewline_(self, sender):
    selRng = self.selectedRange()
    super(PyDETextView, self).insertNewline_(sender)
    line, lineRng, pos = self.findMatchingIndex_paren_(selRng[0], None)
330    if line is None:
        return
    leadingSpace = ""
    if pos is None:
        m = whiteRE.match(line)
335        if m is not None:
            leadingSpace = m.group()
    else:
        leadingSpace = re.sub(r"^\t", " ", line[:pos + 1])
    line, lineRng = self.getLinesForRange_((selRng[0], 0))
340    line = removeStringsAndComments(line).strip()
    if line and line[-1] == ":":
        leadingSpace += self.getIndent()

    if leadingSpace:
345        self.insertText_(leadingSpace)

def insertTab_(self, sender):
    if self.usesTabs:
        return super(PyDETextView, self).insertTab_(sender)
350    self.insertText_("")
    selRng = self.selectedRange()
    assert selRng[1] == 0
    lines, linesRng = self.getLinesForRange_(selRng)
    sel = selRng[0] - linesRng[0]
355    whiteEnd = findWhitespace(lines, sel)
    nSpaces = self.indentSize - (whiteEnd % self.indentSize)
    self.insertText_(nSpaces * " ")
    sel += nSpaces
    whiteEnd += nSpaces
360    sel = min(whiteEnd, sel + (sel % self.indentSize))
    self.setSelectedRange_((sel + linesRng[0], 0))

def deleteBackward_(self, sender):
    self.delete_fwd_superf_(sender, False, super(PyDETextView, self).deleteBackward_)
365

def deleteForward_(self, sender):
    self.delete_fwd_superf_(sender, True, super(PyDETextView, self).deleteForward_)

def delete_fwd_superf_(self, sender, isForward, superFunc):
370    selRng = self.selectedRange()
    if self.usesTabs or selRng[1]:
        return superFunc(sender)
    lines, linesRng = self.getLinesForRange_(selRng)
    sel = selRng[0] - linesRng[0]
375    whiteEnd = findWhitespace(lines, sel)
    whiteBegin = sel
    while whiteBegin and lines[whiteBegin-1] == " ":
        whiteBegin -= 1
    if not isForward:
380        white = whiteBegin
    else:
        white = whiteEnd
    if white == sel or (whiteEnd - whiteBegin) <= 1:
        return superFunc(sender)
385    nSpaces = (whiteEnd % self.indentSize)
    if nSpaces == 0:
        nSpaces = self.indentSize
    offset = sel % self.indentSize
    if not isForward and offset == 0:

```

```

390         offset = nSpaces
        delBegin = sel - offset
        delEnd = delBegin + nSpaces
        delBegin = max(delBegin, whiteBegin)
        delEnd = min(delEnd, whiteEnd)
395     self.setSelectedRange_((linesRng[0] + delBegin, delEnd - delBegin))
        self.insertText_("")

@objc.IBAction
def indent_(self, sender):
400     def indentFilter(lines):
        indent = self.getIndent()
        indentedLines = []
        for line in lines:
            if line.strip():
405                 indentedLines.append(indent + line)
            else:
                indentedLines.append(line)
        [indent + line for line in lines[:-1]]
        return indentedLines
410     self.filterLines_(indentFilter)

@objc.IBAction
def dedent_(self, sender):
    def dedentFilter(lines):
415         indent = self.getIndent()
        dedentedLines = []
        indentSize = len(indent)
        for line in lines:
            if line.startswith(indent):
420                 line = line[indentSize:]
            dedentedLines.append(line)
        return dedentedLines
    self.filterLines_(dedentFilter)

425 @objc.IBAction
def comment_(self, sender):
    def commentFilter(lines):
        commentedLines = []
        indent = self.getIndent()
430         pos = 100
        for line in lines:
            if not line.strip():
                continue
            pos = min(pos, findWhitespace(line))
435         for line in lines:
            if line.strip():
                commentedLines.append(line[:pos] + "#" + line[pos:])
            else:
                commentedLines.append(line)
440         return commentedLines
    self.filterLines_(commentFilter)

@objc.IBAction
def uncomment_(self, sender):
445     def uncommentFilter(lines):
        commentedLines = []
        commentMatch = commentRE.match
        for line in lines:
            m = commentMatch(line)
450             if m is not None:
                pos = m.start(1)
                line = line[:pos] + line[pos+1:]
            commentedLines.append(line)

```

```

        return commentedLines
455     self.filterLines_(uncommentFilter)

    def filterLines_(self, filterFunc):
        selRng = self.selectedRange()
        lines, linesRng = self.getLinesForRange_(selRng)
460
        filteredLines = filterFunc(lines.splitlines(True))

        filteredLines = "".join(filteredLines)
        if lines == filteredLines:
465             return
        self.setSelectedRange_(linesRng)
        self.insertText_(filteredLines)
        newSelRng = linesRng[0], len(filteredLines)
        self.setSelectedRange_(newSelRng)
470
    class PyDETextStorageDelegate(NSObject):

        def __new__(cls, *args, **kwargs):
            return cls.alloc().init()
475

        def __init__(self, textStorage=None):
            self.__syntaxColors = getSyntaxTextAttributes()
            self.__haveScheduledColorize = False
            self.__source = None # XXX
480            self.__dirty = []
            if textStorage is None:
                textStorage = NSTextStorage.alloc().init()
            self.__storage = textStorage
            self.__storage.setAttributes_range_(getBasicTextAttributes(),
485                (0, textStorage.length()))
            self.__string = self.__storage.mutableString().nsstring()
            self.__lineTracker = LineTracker(self.__string)
            self.__storage.setDelegate_(self)

490        def textFontChanged_(self, notification):
            self.__storage.setAttributes_range_(getBasicTextAttributes(),
                (0, self.__storage.length()))
            self.__syntaxColors = getSyntaxTextAttributes()
            self.__dirty = [0]
495            self.scheduleColorize()

        def textStorage(self):
            return self.__storage

500        def string(self):
            return self.__string

        def lineIndexFromCharIndex_(self, charIndex):
            return self.__lineTracker.lineIndexFromCharIndex_(charIndex)
505

        def charIndexFromLineIndex_(self, lineIndex):
            return self.__lineTracker.charIndexFromLineIndex_(lineIndex)

        def numberOfLines(self):
510            return self.__lineTracker.numberOfLines()

        def getSource(self):
            if self.__source is None:
                self.__source = unicode(self.__string)
515            return self.__source

        def textStorageWillProcessEditing_(self, notification):

```

```

    if not self._storage.editedMask() & NSTextStorageEditedCharacters:
        return
520   rng = self._storage.editedRange()
       # make darn sure we don't get infected with return chars
       s = self._string
       s.replaceOccurrencesOfString_withString_options_range_("\r", "\n",
                                                                NSLiteralSearch , rng)
525
def textStorageDidProcessEditing_(self, notification):
    if not self._storage.editedMask() & NSTextStorageEditedCharacters:
        return
    self._source = None
530   rng = self._storage.editedRange()
       try:
           self._lineTracker._update(rng, self._storage.changeInLength())
       except:
           import traceback
           traceback.print_exc()
535   start = rng[0]
       rng = (0, 0)
       count = 0
       while start > 0:
540       # find the last colored token and start from there.
           start -= 1
           attrs, rng = self._storage.attributesAtIndex_effectiveRange_(start, None)
           value = attrs.objectForKey_(NSForegroundColorAttributeName)
           if value != None:
545               count += 1
                   if count > 1:
                       break
           # uncolorized section, track back
           start = rng[0] - 1
550   rng = self._string.lineRangeForRange_((rng[0], 0))
       self._dirty.append(rng[0])
       self.scheduleColorize()

def scheduleColorize(self):
555   if not self._haveScheduledColorize:
       self.performSelector_withObject_afterDelay_("colorize", None, 0.0)
       self._haveScheduledColorize = True

def colorize(self):
560   self._haveScheduledColorize = False
       self._storage.beginEditing()
       try:
           try:
565               self._colorize()
           except:
               import traceback
               traceback.print_exc()
       finally:
           self._storage.endEditing()
570
def _colorize(self):
    if not self._dirty:
        return
    storage = self._storage
    source = self.getSource()
    sourceLen = len(source)
    dirtyStart = self._dirty.pop()

    getColor = self._syntaxColors.get
    580   setAttrs = storage.setAttributes_range_
       getAttrs = storage.attributesAtIndex_effectiveRange_

```

```

basicAttrs = getBasicTextAttributes()

lastEnd = end = dirtyStart
count = 0
sameCount = 0
585 for tag, start, end, sublist in fontify(source, dirtyStart):
    end = min(end, sourceLen)
    rng = (start, end - start)
    590 attrs = getColor(tag)
    oldAttrs, oldRng = getAttrs(rng[0], None)
    if attrs is not None:
        clearRng = (lastEnd, start - lastEnd)
        if clearRng[1]:
            595 setAttrs(basicAttrs, clearRng)
            setAttrs(attrs, rng)
            if rng == oldRng and attrs == oldAttrs:
                sameCount += 1
                if sameCount > 4:
                    600 # due to backtracking we have to account for a few more
                        # tokens, but if we've seen a few tokens that were already
                            # colored the way we want, we're done
                                return
                else:
                    605 sameCount = 0
            else:
                rng = (lastEnd, end - lastEnd)
                if rng[1]:
                    setAttrs(basicAttrs, rng)
                    610 count += 1
                    if count > 200:
                        # enough for now, schedule a new chunk
                        self._dirty.append(end)
                        self.scheduleColorize()
                    615 break
                lastEnd = end
        else:
            # reset coloring at the end
            end = min(sourceLen, end)
            620 rng = (end, sourceLen - end)
            if rng[1]:
                setAttrs(basicAttrs, rng)

class LineTracker(object):
625     def __init__(self, string):
        self.string = string
        self.lines, self.lineStarts, self.lineLengths = self._makeLines()

    630     def _makeLines(self, start=0, end=None):
        lines = []
        lineStarts = []
        lineLengths = []
        string = self.string
        635 if end is None:
            end = string.length()
        else:
            end = min(end, string.length())
        rng = string.lineRangeForRange_((start, end - start))
        640 pos = rng[0]
        end = pos + rng[1]
        while pos < end:
            lineRng = string.lineRangeForRange_((pos, 0))
            line = makeunicode(string.substringWithRange_(lineRng))
            645 assert len(line) == lineRng[1]

```

```

        lines.append(line)
        lineStarts.append(lineRng[0])
        lineLengths.append(lineRng[1])
        if not lineRng[1]:
650             break
            pos += lineRng[1]
        return lines, lineStarts, lineLengths

def _update(self, editedRange, changeInLength):
655     oldRange = editedRange[0], editedRange[1] - changeInLength
    start = self.lineIndexFromCharIndex_(oldRange[0])
    if oldRange[1]:
        end = self.lineIndexFromCharIndex_(oldRange[0] + oldRange[1])
    else:
660         end = start

    lines, lineStarts, lineLengths = self._makeLines(
        editedRange[0], editedRange[0] + editedRange[1] + 1)
    self.lines[start:end + 1] = lines
665     self.lineStarts[start:] = lineStarts # drop invalid tail
    self.lineLengths[start:end + 1] = lineLengths
    # XXX: This assertion doesn't actually assert
    # assert "".join(self.lines) == unicode(self.string)

670     def lineIndexFromCharIndex_(self, charIndex):
        lineIndex = bisect(self.lineStarts, charIndex)
        if lineIndex == 0:
            return 0
        nLines = len(self.lines)
675         nLineStarts = len(self.lineStarts)
        if lineIndex == nLineStarts and nLineStarts != nLines:
            # update line starts
            i = nLineStarts - 1
            assert i >= 0
680             pos = self.lineStarts[i]
            while pos <= charIndex and i < nLines:
                pos = pos + self.lineLengths[i]
                self.lineStarts.append(pos)
                i += 1
685             lineIndex = i

        lineIndex -= 1
        start = self.lineStarts[lineIndex]
        line = self.lines[lineIndex]
690         if (    line[-1:] == "\n"
            and not (start <= charIndex < start + self.lineLengths[lineIndex])):
            lineIndex += 1
        return lineIndex

695     def charIndexFromLineIndex_(self, lineIndex):
        if not self.lines:
            return 0
        if lineIndex == len(self.lines):
            return self.lineStarts[-1] + self.lineLengths[-1]
700         try:
            return self.lineStarts[lineIndex]
        except IndexError:
            # update lineStarts
            for i in range(min(len(self.lines), lineIndex + 1) - len(self.lineStarts)):
705                 self.lineStarts.append(self.lineStarts[-1] + self.lineLengths[-1])
            # XXX: Assertion doesn't actually assert.
            #assert len(self.lineStarts) == len(self.lineLengths) == len(self.lines)
            if lineIndex == len(self.lineStarts):
                return self.lineStarts[-1] + self.lineLengths[-1]

```

```

710         return self.lineStarts[lineIndex]

    def numberOfLines(self):
        return len(self.lines)

715 _basicFont = NSFont.userFixedPitchFontOfSize_(11)

    _BASICATTRS = {NSFontAttributeName: _basicFont,
                   NSLigatureAttributeName: 0}
    _SYNTAXCOLORS = {
720     "keyword": {NSForegroundColorAttributeName: NSColor.blueColor()},
        "identifier": {
            NSForegroundColorAttributeName: NSColor.redColor().shadowWithLevel_(0.2)},
        "string": {NSForegroundColorAttributeName: NSColor.magentaColor()},
        "comment": {NSForegroundColorAttributeName: NSColor.grayColor()},
725 }
    for key, value in _SYNTAXCOLORS.items():
        newVal = _BASICATTRS.copy()
        newVal.update(value)
        _SYNTAXCOLORS[key] = NSDictionary.dictionaryWithDictionary_(newVal)
730 _BASICATTRS = NSDictionary.dictionaryWithDictionary_( _BASICATTRS)

    def unpackAttrs(d):
        unpacked = {}
        for key, value in d.items():
735             if key == NSFontAttributeName:
                name = value["name"]
                size = value["size"]
                value = NSFont.fontWithName_size_(name, size)
            elif key in (NSForegroundColorAttributeName, NSBackgroundColorAttributeName):
740                 r, g, b, a = map(float, value.split())
                value = NSColor.colorWithCalibratedRed_green_blue_alpha_(r, g, b, a)
            elif isinstance(value, (dict, NSDictionary)):
                value = unpackAttrs(value)
            unpacked[key] = value
745         return unpacked

    def packAttrs(d):
        packed = {}
        for key, value in d.items():
750             if key == NSFontAttributeName:
                value = {"name": value.fontName(), "size": value.pointSize()}
            elif key in (NSForegroundColorAttributeName, NSBackgroundColorAttributeName):
                col = value.colorUsingColorSpaceName_(NSCalibratedRGBColorSpace)
                channels = col.getRed_green_blue_alpha_(None, None, None, None)
755                 value = " ".join(map(str, channels))
            elif isinstance(value, (dict, NSDictionary)):
                value = packAttrs(value)
            packed[key] = value
        return packed

760
    def getBasicTextAttributes():
        attrs = NSUserDefaults.standardUserDefaults().objectForKey_(
            "PyDEDefaultTextAttributes")
        return unpackAttrs(attrs)
765
    def getSyntaxTextAttributes():
        attrs = NSUserDefaults.standardUserDefaults().objectForKey_(
            "PyDESyntaxTextAttributes")
        return unpackAttrs(attrs)
770
    def setBasicTextAttributes(basicAttrs):
        if basicAttrs != getBasicTextAttributes():
            NSUserDefaults.standardUserDefaults().setObject_forKey_(

```



```

        packAttrs(basicAttrs), "PyDEDefaultTextAttributes")
775     nc = NSNotificationCenter defaultCenter()
        nc.postNotificationName_object_("PyDETextFontChanged", None)

    def setSyntaxTextAttributes(syntaxAttrs):
        if syntaxAttrs != getSyntaxTextAttributes():
780             NSUserDefaults.standardUserDefaults().setObject_forKey_(
                packAttrs(syntaxAttrs), "PyDESyntaxTextAttributes")
            nc = NSNotificationCenter defaultCenter()
            nc.postNotificationName_object_("PyDETextFontChanged", None)

785 def setTextFont(font):
    basicAttrs = getBasicTextAttributes()
    syntaxAttrs = getSyntaxTextAttributes()
    basicAttrs[NSFontAttributeName] = font
    for v in syntaxAttrs.values():
790         v[NSFontAttributeName] = font
    setBasicTextAttributes(basicAttrs)
    setSyntaxTextAttributes(syntaxAttrs)

_defaultUserDefaults = {
795     "PyDEDefaultTextAttributes": packAttrs(_BASICATTRS),
    "PyDESyntaxTextAttributes": packAttrs(_SYNTAXCOLORS),
}

NSUserDefaults.standardUserDefaults().registerDefaults(_defaultUserDefaults)

```

## nodebox/gui/mac/util.py

```

import AppKit

def errorAlert(msgText, infoText):
    # Force NSApp initialisation.
5     AppKit.NSApplication.sharedApplication().activateIgnoringOtherApps_(0)
    alert = AppKit.NSAlert.alloc().init()
    alert.setMessageText_(msgText)
    alert.setInformativeText_(infoText)
    alert.setAlertStyle_(AppKit.NSCriticalAlertStyle)
10    btn = alert.addButtonWithTitle_("OK")
    return alert.runModal()

```

## nodebox/gui/mac/ValueLadder.py

```

#from Foundation import *
#from AppKit import *

import compiler
5 parse = compiler.parse

import compiler.ast
Sub = compiler.ast.Sub
UnarySub = compiler.ast.UnarySub
10 Add = compiler.ast.Add

import Foundation
import AppKit

15 NSObject = AppKit.NSObject
NSColor = AppKit.NSColor
NSMutableParagraphStyle = AppKit.NSMutableParagraphStyle
NSCenterTextAlignment = AppKit.NSCenterTextAlignment
NSFont = AppKit.NSFont

```

```

20 NSForegroundColorAttributeName = AppKit.NSForegroundColorAttributeName
    NSCursor = AppKit.NSCursor
    NSGraphicsContext = AppKit.NSGraphicsContext
    NSBezierPath = AppKit.NSBezierPath
    NSString = AppKit.NSString
25 NSEvent = AppKit.NSEvent
    NSAlternateKeyMask = AppKit.NSAlternateKeyMask
    NSShiftKeyMask = AppKit.NSShiftKeyMask
    NSParagraphStyleAttributeName = AppKit.NSParagraphStyleAttributeName
    NSFontAttributeName = AppKit.NSFontAttributeName
30
    MAGICVAR = "__magic_var__"

    class ValueLadder:

35         view = None
        visible = False
        value = None
        origValue = None
        dirty = False
40         type = None
        negative = False
        unary = False
        add = False

45         def __init__(self, textView, value, clickPos, screenPoint, viewPoint):
            self.textView = textView
            self.value = value
            self.origValue = value
            self.type = type(value)
50             self.clickPos = clickPos
            self.origX, self.origY = screenPoint
            self.x, self.y = screenPoint
            self.viewPoint = viewPoint
            (x,y),(self.width,self.height) = self.textView.bounds()
55             self.originalString = self.textView.string()
            self.backgroundColor = NSColor.colorWithCalibratedRed_green_blue_alpha_(
                                                                    0.4,0.4,0.4, 1.0)
            self.strokeColor = NSColor.colorWithCalibratedRed_green_blue_alpha_(
                                                                    0.1,0.1,0.1, 1.0)
60             self.textColor = NSColor.colorWithCalibratedRed_green_blue_alpha_(
                                                                    1.0,1.0,1.0, 1.0)

            paraStyle = NSMutableParagraphStyle.alloc().init()
            paraStyle.setAlignment_(NSCenterTextAlignment)
            font = NSFont.fontWithName_size_("Monaco", 10)
65             self.textAttributes = {
                NSForegroundColorAttributeName: self.textColor,
                NSParagraphStyleAttributeName: paraStyle,NSFontAttributeName:font}

            # To speed things up, the code is compiled only once.
70             # The number is replaced with a magic variable, that is set in the
            # namespace when executing the code.
            begin,end = self.clickPos
            self.patchedSource = (self.originalString[:begin]
                                + MAGICVAR
75                                + self.originalString[end:])

            #ast = parse(self.patchedSource + "\n\n")
            #self._checkSigns(ast)
            success, output = self.textView.document.boxedRun_args_(self._parseAndCompile, [])
80             if success:
                self.show()
            else:
                self.textView.document._flushOutput(output)

```

```

85  def _parseAndCompile(self):
    ast = parse(self.patchedSource.encode('ascii', 'replace') + "\n\n")
    self._checkSigns(ast)
    self.textView.document._compileScript(self.patchedSource)

90  def _checkSigns(self, node):
    """Recursively check for special sign cases.

    The following cases are special:
    - Substraction. When you select the last part of a substraction
    95   (e.g. the 5 of "10-5"), it might happen that you drag the number to
      a positive value. In that case, the result should be "10+5".
    - Unary substraction. Values like "-5" should have their sign removed
      when you drag them to a positive value.
    - Addition. When you select the last part of an addition
    100   (e.g. the 5 of "10+5"), and drag the number to a negative value,
      the result should be "10-5".

    This algorithm checks for these cases. It tries to find the magic var,
    and then checks the parent node to see if it is one of these cases,
    105   then sets the appropriate state variables in the object.

    This algorithm is recursive. Because we have to differ between a
    "direct hit" (meaning the current child was the right one) and a
    "problem resolved" (meaning the algorithm found the node, did its
    110   work and now needs to bail out), we have three return codes:
    - -1: nothing was found in this node and its child nodes.
    - 1: direct hit. The child you just searched contains the magicvar.
      check the current node to see if it is one of the special cases.
    - 0: bail out. Somewhere, a child contained the magicvar, and we
    115   acted upon it. Now leave this algorithm as soon as possible.
    """

    # Check whether I am the correct node
    try:
    120     if node.name == MAGICVAR:
        return 1 # If i am, return the "direct hit" code.
    except AttributeError:
        pass

    125   # We keep an index to see what child we are checking. This
    # is important for binary operations, were we are only interested
    # in the second part. ("a-10" has to change to "a+10",
    # but "10-a" shouldn't change to "+10-a")
    index = 0

    130   # Recursively check my children
    for child in node.getChildNodes():
        retVal = self._checkSigns(child)
        # Direct hit. The child I just searched contains the magicvar.
        # Check whether this node is one of the special cases.
    135     if retVal == 1:
        # Unary substitution.
        if isinstance(node, UnarySub):
            self.negative = True
            self.unary = True
        140     # Binary substitution. Only the second child is of importance.
        elif isinstance(node, Sub) and index == 1:
            self.negative = True
        # Binary addition. Only the second child is of importance.
        elif isinstance(node, Add) and index == 1:
    145         self.add = True
        # Return the "bail out" code, whether we found some
        # special case or not. There can only be one magicvar in the

```

```

        # code, so once that is found we can stop looking.
        return 0
150     # If the child returns a bail out code, we leave this routine
        # without checking the other children, passing along the
        # bail out code.
        elif retVal == 0:
            return 0 # Nothing more needs to be done.

155     # Next child.
        index += 1

    # We searched all children, but couldn't find any magicvars.
160     return -1

def show(self):
    self.visible = True
    self.textView.setNeedsDisplay_(True)
165     NSCursor.hide()

def hide(self):
    """Hide the ValueLadder and update the code.

170     Updating the code means we have to replace the current value with
    the new value, and account for any special cases."""

    self.visible = False
    begin,end = self.clickPos

175     # Potentially change the sign on the number.
    # The following cases are valid:
    # - A subtraction where the value turned positive "random(5-8)" --> "random(5+8)"
    # - A unary subtraction where the value turned positive "random(-5)" --> "random(5)"
180     # Note that the sign dissapears here.
    # - An addition where the second part turns negative "random(5+8)" --> "random(5-8)"
    # Note that the code replaces the sign on the place where it was, leaving the code intact.

    # Case 1: Negative numbers where the new value is negative as well.
185     # This means the numbers turn positive.
    if self.negative and self.value < 0:
        # Find the minus sign.
        i = begin - 1
        notFound = True
190         while True:
            if self.originalString[i] == '-':
                if self.unary: # Unary subtractions will have the sign removed.
                    # Re-create the string: the spaces between the value and the '-' + the value
                    value = self.originalString[i+1:begin] + str(abs(self.value))
195                 else: # Binary subtractions get a '+'
                    value = '+' + self.originalString[i+1:begin] + str(abs(self.value))
                range = (i,end-i)
                break
            i -= 1

200     # Case 2: Additions (only additions where we are the second part
    # interests us, this is checked already on startup)
    elif self.add and self.value < 0:
        # Find the plus sign.
        i = begin - 1
        notFound = True
205         while True:
            if self.originalString[i] == '+':
                # Re-create the string:
                # - a '+' (instead of the minus)
                # - the spaces between the '-' and the constant
210                # - the constant itself

```

```

        value = '-' + self.originalString[i+1:begin] + str(abs(self.value))
        range = (i,end-i)
        break
215         i -= 1
        # Otherwise, it's a normal case. Note that here also, positive numbers
        # can turn negative, but no existing signs have to be changed.
        else:
            value = str(self.value)
220         range = (begin, end-begin)

        # The following textView methods make sure that an undo operation
        # is registered, so users can undo their drag.
        self.textView.shouldChangeTextInRange_replacementString_(range, value)
225 self.textView.textStorage().replaceCharactersInRange_withString_(range, value)
        self.textView.didChangeText()
        self.textView.setNeedsDisplay_(True)
        self.textView.document.currentView.direct = False
        NSCursor.unhide()

230 def draw(self):
        mx,my=self.viewPoint

        x = mx-20
235 w = 80
        h = 20
        h2 = h*2

        context = NSGraphicsContext.currentContext()
240 aa = context.shouldAntialias()
        context.setShouldAntialias_(False)
        r = ((mx-w/2,my+12),(w,h))
        NSBezierPath.setDefaultLineWidth_(0)
        self.backgroundColor.set()
245 NSBezierPath.fillRect_(r)
        self.strokeColor.set()
        NSBezierPath.strokeRect_(r)

        # A standard value just displays the value that you have been dragging.
250 if not self.negative:
            v = str(self.value)
            # When the value is negative, we don't display a double negative,
            # but a positive.
            elif self.value < 0:
255 v = str(abs(self.value))
            # When the value is positive, we have to add a minus sign.
            else:
                v = "-" + str(self.value)

260 NSString.drawInRect_withAttributes_(v, ((mx-w/2,my+14),(w,h2)), self.textAttributes)
        context.setShouldAntialias_(aa)

def mouseDragged_(self, event):
        mod = event.modifierFlags()
265 newX, newY = NSEvent.mouseLocation()
        deltaX = newX-self.x
        delta = deltaX
        if self.negative:
            delta = -delta
270 if mod & NSAlternateKeyMask:
            delta /= 100.0
        elif mod & NSShiftKeyMask:
            delta *= 10.0
        self.value = self.type(self.value + delta)
275 self.x, self.y = newX, newY

```

```

        self.dirty = True
        self.textView.setNeedsDisplay_(True)
        self.textView.document.magicvar = self.value
        self.textView.document.currentView.direct = True
280     self.textView.document.runScriptFast()

```

## nodebox/util/\_\_init\_\_.py

```

import os
import time
import datetime
import glob
5
import random as librandom
choice = librandom.choice

import unicodedata
10 import objc

import Foundation
import AppKit
import PyObjCTools.Conversion
15
import kgp

__all__ = ('grid', 'random', 'choice', 'files', 'autotext', '_copy_attr', '_copy_attrs',
          'datestring', 'makeunicode', 'filelist', 'imagefiles',
20          'fontnames', 'fontfamilies', 'voices', 'voiceattributes', 'anySpeakers', 'say')

### Utilities ###

g_voicetrash = []
25
def makeunicode(s, srcencoding="utf-8", normalizer="NFC"):
    typ = type(s)
    # convert to str first; for number types etc.
    if typ not in (str, unicode, Foundation.NSMutableAttributedString,
30         objc.pyobjc_unicode, Foundation.NSMutableStringProxyForMutableAttributedString,
        Foundation.NSString):
        # print "makeunicode() convert:", typ
        s = str(s)
    if typ not in (unicode, Foundation.NSMutableAttributedString, objc.pyobjc_unicode,
35         Foundation.NSMutableStringProxyForMutableAttributedString):
        try:
            s = unicode(s, srcencoding)
        except TypeError, err:
            print
            print "makeunicode():", err
            print repr(s)
            print type(s)
            print
40
    if typ in (unicode,):
45         s = unicodedata.normalize(normalizer, s)
    return s

def datestring(dt = None, dateonly=False, nospaces=True, nocolons=True):
    """Make an ISO datestring. The defaults are good for using the result of
50     'datestring()' in a filename.
    """
    if not dt:
        now = str(datetime.datetime.now())
    else:
55         now = str(dt)

```

```

    if not dateonly:
        now = now[:19]
    else:
        now = now[:10]
60  if nospaces:
        now = now.replace(" ", "_")
    if nocolons:
        now = now.replace(":", "")
    return now
65
def grid(cols, rows, colSize=1, rowSize=1, shuffled=False):
    """Returns an iterator that contains coordinate tuples.

    The grid can be used to quickly create grid-like structures.
    A common way to use them is:
        for x, y in grid(10,10,12,12):
            rect(x,y, 10,10)
    """

    # Prefer using generators.
75  rowRange = xrange(int(rows))
    colRange = xrange(int(cols))
    # Shuffled needs a real list, though.
    if (shuffled):
        rowRange = list(rowRange)
80     colRange = list(colRange)
        shuffle(rowRange)
        shuffle(colRange)
    for y in rowRange:
        for x in colRange:
85         yield (x*colSize,y*rowSize)

def random(v1=None, v2=None):
    """Returns a random value.

    This function does a lot of things depending on the parameters:
    - If one or more floats is given, the random value will be a float.
    - If all values are ints, the random value will be an integer.

    - If one value is given, random returns a value from 0 to the given value.
    This value is not inclusive.
95  - If two values are given, random returns a value between the two; if two
    integers are given, the two boundaries are inclusive.
    """

    if v1 != None and v2 == None: # One value means 0 -> v1
100     if isinstance(v1, float):
            return librandom.random() * v1
        else:
            return int(librandom.random() * v1)
    elif v1 != None and v2 != None: # v1 -> v2
105     if isinstance(v1, float) or isinstance(v2, float):
            start = min(v1, v2)
            end = max(v1, v2)
            return start + librandom.random() * (end-start)
        else:
110         start = min(v1, v2)
            end = max(v1, v2) + 1
            return int(start + librandom.random() * (end-start))
    else: # No values means 0.0 -> 1.0
        return librandom.random()
115
def autotext(sourceFile):
    k = kgp.KantGenerator(sourceFile)
    return k.output()

```

```

120 def files(path="*"):
    """Returns a list of files.

    You can use wildcards to specify which files to pick, e.g.
    f = files('*.gif')
    """
    f = glob.glob(path)
    f = [makeunicode(t) for t in f]
    return f

130 def filelist( folderpathorlist, pathonly=True ):
    """Walk a folder or a list of folders and return
    paths or ((filepath, size, lastmodified, mode) tuples..
    """

135     folders = folderpathorlist
    if type(folderpathorlist) in (str, unicode):
        folders = [folderpathorlist]
    result = []
    for folder in folders:
140         folder = os.path.expanduser( folder )
        folder = os.path.abspath( folder )
        for root, dirs, files in os.walk( folder ):
            root = makeunicode( root )

145             # skip if dir starts with '.'
            _, parentfolder = os.path.split(root)
            if parentfolder[0] == u".":
                continue

150             for thefile in files:
                thefile = makeunicode( thefile )
                basename, ext = os.path.splitext(thefile)

                # exclude dotfiles
155                 if thefile.startswith('.'):
                    continue

                # exclude the specials
                for item in (u'\r', u'\n', u'\t'):
160                     if item in thefile:
                        continue

                filepath = os.path.join( root, thefile )

165                 record = filepath
                if not pathonly:
                    islink = os.path.islink( filepath )
                    if islink:
                        info = os.lstat( filepath )
170                     else:
                        info = os.stat( filepath )
                        lastmodified = datetime.datetime.fromtimestamp( info.st_mtime )
                        record = (filepath, info.st_size, lastmodified,
                                oct(info.st_mode), islink )

175                 yield record

def imagefiles( folderpathorlist, pathonly=True ):
    """Use filelist to extract all imagefiles"""
    result = []
180     filetuples = filelist( folderpathorlist, pathonly=pathonly )

    # 2017-06-23 - kw .eps dismissed
    extensions = tuple(".pdf .tif .tiff .gif .jpg .jpeg .png".split())

```



```

    for filetuple in filetuples:
185         path = filetuple
            if not pathonly:
                path = filetuple[0]
            _, ext = os.path.splitext( path )
            if ext.lower() not in extensions:
190                 continue
            if pathonly:
                yield path
            else:
                yield filetuple
195
def fontnames():
    fm = AppKit.NSFontManager.sharedFontManager()
    l = fm.availableFonts()
    result = []
200     for i in l:
        # filter out the weird fontnames
        if i.startswith(u'.'):
            continue
        result.append( makeunicode(i) )
205     return result

class FontRecord:
    def __init__(self, psname, familyname, style, weight, traits, traitnames):
        self.psname = psname
210         self.familyname = familyname
        self.style = style
        self.weight = weight
        self.traits = traits
        self.traitnames = traitnames
215     def __repr__(self):
        return (u'FontRecord( psname="%s", familyname="%s", style="%s", '
            u'weight=%.2f, traits="%s", traitnames=%s)') % (
                self.psname, self.familyname, self.style,
                self.weight, self.traits, self.traitnames)
220
def fontfamilies(flat=False):
    fm = AppKit.NSFontManager.sharedFontManager()
    l = fm.availableFontFamilies()

225     def makeTraitsList( traits ):
        appleTraits = {
            0x00000001: u"italic",
            0x00000002: u"bold",
            0x00000004: u"unbold",
230            0x00000008: u"nonstandardcharacter",
            0x00000010: u"narrow",
            0x00000020: u"expanded",
            0x00000040: u"condensed",
            0x00000080: u"smallcaps",
235            0x00000100: u"poster",
            0x00000200: u"compressed",
            0x00000400: u"fixedpitch",
            0x01000000: u"unitalic"}
        result = []
        keys = appleTraits.keys()
240         for key in keys:
            if traits & key == key:
                result.append( appleTraits[key])
        return result
245
    def makeFontRecord(fnt):
        psname, styl, weight, traits = fnt

```

```

        psname = makeunicode(psname)
        styl = makeunicode(styl)
250     weight = float( weight )
        traits = int(traits)
        traitNames = makeTraitsList( traits )
        return FontRecord(psname, familyName, styl, weight, traits, traitNames)

255     if flat:
        result = []
    else:
        result = {}
    for fn in l:
260         familyName = makeunicode( fn )
        if not flat:
            result[familyName] = famfonts = {}

        subs = fm.availableMembersOfFontFamily_( familyName )
265         for fnt in subs:
            fontRec = makeFontRecord( fnt )
            if not flat:
                result[familyName][fontRec.style] = fontRec
            else:
270                 result.append( fontRec )
    return result

def voices():
    """Return a list of voice names."""
275     vcs = AppKit.NSSpeechSynthesizer.availableVoices()
    vcs = [makeunicode(t) for t in vcs]
    vcs = [x.replace(u"com.apple.speech.synthesis.voice.", u"") for x in vcs]
    return vcs

280 def voiceattributes(voice):
    """Return a dict with attributes for voice.

    voice is passed without the 'com.apple.speech.synthesis.voice.' prefix, e.g.
    'Albert' or 'petra.premium'.
285     """
    result = {}
    if voice and voice in voices():
        voice = u"com.apple.speech.synthesis.voice.%s" % (voice,)
        attrs = AppKit.NSSpeechSynthesizer.attributesForVoice_( voice )
290         result = PyObjCTools.Conversion.pythonCollectionFromPropertyList(attrs)
        keys = result.keys()
        for key in keys:
            result[key] = makeunicode(result[key])
    return result

295 def anySpeakers():
    """Return if ANY application is currently speaking."""
    global g_voicetrash

300     b = bool(AppKit.NSSpeechSynthesizer.isAnyApplicationSpeaking())
    if b == False:
        # empty accumulated voices
        while len(g_voicetrash) > 0:
            f = g_voicetrash.pop()
305         del f
    return b

def say(txt, voice=None, outfile=None, wait=True):
    """Say txt with a voice."""
310     global g_voicetrash
    if voice and voice in voices():

```

```

        voice = u"com.apple.speech.synthesis.voice.%s" % (voice,)
else:
        voice = AppKit.NSSpeechSynthesizer.defaultVoice()
315
        # outfile is a path to an AIFF file to be exported to
        # if the containing folder does not exist, abort
        path = url = None
        if outfile:
320
            path = os.path.abspath( makeunicode(outfile) )
            folder, filename = os.path.split( path )
            if not os.path.exists( folder ):
                path = None

325
        if path:
            url = Foundation.NSURL.fileURLWithPath_isDirectory_( path, False )
            speaker = AppKit.NSSpeechSynthesizer.alloc().initWithVoice_(voice)

        if speaker and url:
330
            g_voicetrash.append( speaker )
            speaker.startSpeakingString_toURL_(txt, url)
            return speaker

        if speaker:
335
            if wait:
                while anySpeakers():
                    time.sleep(0.1)
                # it is important that speaker gets added AFTER anySpeakers()
                # it does garbage collection
340
                g_voicetrash.append( speaker )
                speaker.startSpeakingString_(txt)
                return speaker

    def _copy_attr(v):
345
        if v is None:
            return None
        elif hasattr(v, "copy"):
            return v.copy()
        elif isinstance(v, list):
350
            return list(v)
        elif isinstance(v, tuple):
            return tuple(v)
        elif isinstance(v, (int, str, unicode, float, bool, long)):
            return v
355
        else:
            raise NodeBoxError, "Don't know how to copy '%s'." % v

    def _copy_attrs(source, target, attrs):
        for attr in attrs:
360
            setattr(target, attr, _copy_attr(getattr(source, attr)))

```

## nodebox/util/kgp/\_\_\_init\_\_\_.py

```
#!/usr/bin/env python2
```

```
"""Kant Generator for Python
```

```
Generates mock philosophy based on a context-free grammar
```

5

```
Usage: python kgp.py [options] [source]
```

```
Options:
```

```

    -g ..., --grammar=...    use specified grammar file or URL
10  -h, --help                show this help
    -d                        show debugging information while parsing

```

Examples:

```
kgp.py          generates several paragraphs of Kantian philosophy
15 kgp.py -g husserl.xml  generates several paragraphs of Husserl
kgp.py "<xref id='paragraph'/>"  generates a paragraph of Kant
kgp.py template.xml    reads from template.xml to decide what to generate
```

This program is part of "Dive Into Python", a free Python book for  
20 experienced programmers. Visit <http://diveintopython.org/> for the  
latest version.

```
"""
__author__ = "Mark Pilgrim (f8dy@diveintopython.org)"
25 __version__ = "$Revision: 1.3 $"
__date__ = "$Date: 2002/05/28 17:05:23 $"
__copyright__ = "Copyright (c) 2001 Mark Pilgrim"
__license__ = "Python"

30 from xml.dom import minidom
    import random
    import sys
    import getopt

35 _debug = 0

def openAnything(source):
    """URI, filename, or string --> stream

40    This function lets you define parsers that take any input source
    (URL, pathname to local or network file, or actual data as a string)
    and deal with it in a uniform manner. Returned object is guaranteed
    to have all the basic stdio read methods (read, readline, readlines).
    Just .close() the object when you're done with it.

45    Examples:
    >>> from xml.dom import minidom
    >>> sock = openAnything("http://localhost/kant.xml")
    >>> doc = minidom.parse(sock)
50 >>> sock.close()
    >>> sock = openAnything("c:\\inetpub\\wwwroot\\kant.xml")
    >>> doc = minidom.parse(sock)
    >>> sock.close()
    >>> sock = openAnything("<ref id='conjunction'><text>and</text><text>or</text></ref>")
55 >>> doc = minidom.parse(sock)
    >>> sock.close()
    """

    if hasattr(source, "read"):
60         return source

    if source == "-":
        import sys
        return sys.stdin

65    # try to open with urllib (if source is http, ftp, or file URL)
    import urllib
    try:
        return urllib.urlopen(source)
70    except (IOError, OSError):
        pass

    # try to open with native open function (if source is pathname)
    try:
75         return open(source)
```

```

    except (IOError, OSError):
        pass

    # treat source as string
80    import StringIO
    return StringIO.StringIO(str(source))

class NoSourceError(Exception): pass

85 class KantGenerator:
    """generates mock philosophy based on a context-free grammar"""

    def __init__(self, grammar, source=None):
        self.loadGrammar(grammar)
90    self.loadSource(source and source or self.getDefaultSource())
        self.refresh()

    def _load(self, source):
        """load XML input source, return parsed XML document
95
        - a URL of a remote XML file ("http://diveintopython.org/kant.xml")
        - a filename of a local XML file ("~/diveintopython/common/py/kant.xml")
        - standard input ("-")
        - the actual XML document, as a string
100    """
        sock = openAnything(source)
        xmldoc = minidom.parse(sock).documentElement
        sock.close()
        return xmldoc

105    def loadGrammar(self, grammar):
        """load context-free grammar"""
        self.grammar = self._load(grammar)
        self.refs = {}
110    for ref in self.grammar.getElementsByTagName("ref"):
        self.refs[ref.attributes["id"].value] = ref

    def loadSource(self, source):
        """load source"""
115    self.source = self._load(source)

    def getDefaultSource(self):
        """guess default source of the current grammar
120
        The default source will be one of the <ref>s that is not
        cross-referenced. This sounds complicated but it's not.
        Example: The default source for kant.xml is
        "<xref id='section'/>", because 'section' is the one <ref>
        that is not <xref>'d anywhere in the grammar.
125    In most grammars, the default source will produce the
        longest (and most interesting) output.
        """
        xrefs = {}
        for xref in self.grammar.getElementsByTagName("xref"):
130            xrefs[xref.attributes["id"].value] = 1
        xrefs = xrefs.keys()
        standaloneXrefs = [e for e in self.refs.keys() if e not in xrefs]
        if not standaloneXrefs:
            raise NoSourceError, "can't guess source, and no source specified"
135    return '<xref id="%s"/>' % random.choice(standaloneXrefs)

    def reset(self):
        """reset parser"""
        self.pieces = []

```

```

140         self.capitalizeNextWord = 0

def refresh(self):
    """reset output buffer, re-parse entire source file, and return output

145    Since parsing involves a good deal of randomness, this is an
    easy way to get new output without having to reload a grammar file
    each time.
    """
    self.reset()
    self.parse(self.source)
150    return self.output()

def output(self):
    """output generated text"""
155    return "".join(self.pieces)

def randomChildElement(self, node):
    """choose a random child element of a node

160    This is a utility method used by do_xref and do_choice.
    """
    choices = [e for e in node.childNodes
                if e.nodeType == e.ELEMENT_NODE]
    chosen = random.choice(choices)
165    if _debug:
        sys.stderr.write('%s available choices: %s\n' % \
                          (len(choices), [e.toxml() for e in choices]))
        sys.stderr.write('Chosen: %s\n' % chosen.toxml())
    return chosen

170
def parse(self, node):
    """parse a single XML node

    A parsed XML document (from minidom.parse) is a tree of nodes
    of various types. Each node is represented by an instance of the
175    corresponding Python class (Element for a tag, Text for
    text data, Document for the top-level document). The following
    statement constructs the name of a class method based on the type
    of node we're parsing ("parse_Element" for an Element node,
    "parse_Text" for a Text node, etc.) and then calls the method.
180    """
    parseMethod = getattr(self, "parse_%s" % node.__class__.__name__)
    parseMethod(node)

185    def parse_Document(self, node):
        """parse the document node

        The document node by itself isn't interesting (to us), but
        its only child, node.documentElement, is: it's the root node
190        of the grammar.
        """
        self.parse(node.documentElement)

    def parse_Text(self, node):
195        """parse a text node

        The text of a text node is usually added to the output buffer
        verbatim. The one exception is that <p class='sentence'> sets
        a flag to capitalize the first letter of the next word. If
        that flag is set, we capitalize the text and reset the flag.
200        """
        text = node.data
        if self.capitalizeNextWord:

```

```

        self.pieces.append(text[0].upper())
205         self.pieces.append(text[1:])
        self.capitalizeNextWord = 0
    else:
        self.pieces.append(text)

210 def parse_Element(self, node):
    """parse an element

    An XML element corresponds to an actual tag in the source:
    <xref id='...'>, <p chance='...'>, <choice>, etc.
215     Each element type is handled in its own method. Like we did in
    parse(), we construct a method name based on the name of the
    element ("do_xref" for an <xref> tag, etc.) and
    call the method.
    """

220     handlerMethod = getattr(self, "do_%s" % node.tagName)
    handlerMethod(node)

    def parse_Comment(self, node):
        """parse a comment

225         The grammar can contain XML comments, but we ignore them
        """

        pass

230 def do_xref(self, node):
    """handle <xref id='...'> tag

    An <xref id='...'> tag is a cross-reference to a <ref id='...'>
    tag. <xref id='sentence' /> evaluates to a randomly chosen child of
235     <ref id='sentence'>.
    """

    id = node.attributes["id"].value
    self.parse(self.randomChildElement(self.refs[id]))

240 def do_p(self, node):
    """handle <p> tag

    The <p> tag is the core of the grammar. It can contain almost
    anything: freeform text, <choice> tags, <xref> tags, even other
245     <p> tags. If a "class='sentence'" attribute is found, a flag
    is set and the next word will be capitalized. If a "chance='X'"
    attribute is found, there is an X% chance that the tag will be
    evaluated (and therefore a (100-X)% chance that it will be
    completely ignored)
250     """

    keys = node.attributes.keys()
    if "class" in keys:
        if node.attributes["class"].value == "sentence":
            self.capitalizeNextWord = 1
255     if "chance" in keys:
        chance = int(node.attributes["chance"].value)
        doit = (chance > random.randrange(100))
    else:
        doit = 1
260     if doit:
        for child in node.childNodes: self.parse(child)

    def do_choice(self, node):
        """handle <choice> tag

265         A <choice> tag contains one or more <p> tags. One <p> tag
        is chosen at random and evaluated; the rest are ignored.

```

```

        """
        self.parse(self.randomChildElement(node))
270
    def usage():
        print __doc__

    def main(argv):
275        grammar = "kant.xml"
        try:
            opts, args = getopt.getopt(argv, "hg:d", ["help", "grammar="])
        except getopt.GetoptError:
            usage()
            sys.exit(2)
280        for opt, arg in opts:
            if opt in ("-h", "--help"):
                usage()
                sys.exit()
            elif opt == '-d':
                global _debug
                _debug = 1
            elif opt in ("-g", "--grammar"):
                grammar = arg
285

        source = "".join(args)
        k = KantGenerator(grammar, source)
        print k.output()

290

if __name__ == "__main__":
    main(sys.argv[1:])

```

### nodebox/util/ottobot/\_\_init\_\_.py

```

from AppKit import NSFontManager

from nodebox.util import random, choice

5 COMP_WIDTH = 500
  COMP_HEIGHT = 500

  XCOORD = 1
  YCOORD = 2
10 XSIZE = 3
  YSIZE = 4
  ROTATION = 5
  SCALE = 6
  CONTROLPOINT = 7
15 COLOR = 8
  STROKEWIDTH = 9
  LOOP = 10
  GRIDDELTA = 12
  GRIDCOUNT = 13
20 GRIDWIDTH = 14
  GRIDHEIGHT = 15
  SKEW = 16
  STARPOINTS = 17

25 class Context:
    def __init__(self):
        self._indent = 0
        self._grid = False

30    def indent(self):
        self._indent += 1

```



```

def dedent(self):
    self._indent -= 1
35
def spaces(self):
    return " " * self._indent

def inGrid(self):
40    return self._grid

def nrReally(ctx, numberclass):
    if numberclass == XCOORD:
        if ctx.inGrid():
45            #return "x"
            return "x + %s" % nr(ctx, GRIDDELTA)
        else:
            return random(-COMP_WIDTH/2, COMP_WIDTH/2)
    elif numberclass == YCOORD:
50        if ctx.inGrid():
            #return "y"
            return "y + %s" % nr(ctx, GRIDDELTA)
        else:
            return random(-COMP_HEIGHT/2, COMP_HEIGHT/2)
55    elif numberclass == XSIZE:
        return random(0, COMP_WIDTH)
    elif numberclass == YSIZE:
        return random(0, COMP_HEIGHT)
    elif numberclass == ROTATION:
60        return random(0, 360)
    elif numberclass == SCALE:
        return random(0.5, 1.5)
    elif numberclass == CONTROLPOINT:
        return random(-100, 100)
65    elif numberclass == COLOR:
        return random()
    elif numberclass == STROKEWIDTH:
        return random(1, 20)
    elif numberclass == LOOP:
70        return random(2, 20)
    elif numberclass == GRIDDELTA:
        return random(-100, 100)
    elif numberclass == GRIDCOUNT:
        return random(2, 10)
75    elif numberclass == GRIDWIDTH:
        return 20
        return random(1, 100)
    elif numberclass == GRIDHEIGHT:
        return 20
80        return random(1, 100)
    elif numberclass == SKEW:
        return random(1, 80)
    elif numberclass == STARPOINTS:
        return random(2, 100)
85

def nr(ctx, numberclass):
    if not ctx.inGrid() and random() > 0.5:
        return "random(%s)" % nrReally(ctx, numberclass)
    else:
90        return "%s" % nrReally(ctx, numberclass)

### DRAWING COMMANDS ###

def genDraw(ctx):
95    fn = choice((genRect, genOval, genArrow, genStar, genPath))

```

```

    return fn(ctx)

def genRect(ctx):
    return ctx.spaces() + """rect(%s,%s,%s,%s)\n""" % (
100     nr(ctx,XCOORD),nr(ctx,YCOORD),nr(ctx,XSIZE),nr(ctx,YSIZE))

def genOval(ctx):
    return ctx.spaces() + """oval(%s,%s,%s,%s)\n""" % (
105     nr(ctx,XCOORD),nr(ctx,YCOORD),nr(ctx,XSIZE),nr(ctx,YSIZE))

def genArrow(ctx):
    return ctx.spaces() + """arrow(%s,%s,%s)\n""" % (
        nr(ctx,XCOORD),nr(ctx,YCOORD),nr(ctx,XSIZE))

110 def genStar(ctx):
    return ctx.spaces() + """star(%s,%s,%s,%s,%s)\n""" % (
        nr(ctx,XCOORD),nr(ctx,YCOORD),nr(ctx,STARPOINTS),nr(ctx,XSIZE),nr(ctx,YSIZE))

def genPath(ctx):
115     s = ctx.spaces() + """beginpath(%s,%s)\n""" % (
        nr(ctx,XCOORD),nr(ctx,YCOORD))
    for i in range(random(1,10)):
        s += genPathDraw(ctx)
    s += ctx.spaces() + """endpath()\n"""
120     return s

def genPathDraw(ctx):
    fn = choice((genLineto, genCurveto))
    return fn(ctx)

125 def genLineto(ctx):
    return ctx.spaces() + """lineto(%s,%s)\n""" % (nr(ctx,XCOORD),nr(ctx,YCOORD))

def genCurveto(ctx):
130     return ctx.spaces() + """curveto(%s,%s,%s,%s,%s,%s)\n""" % (
        nr(ctx,XCOORD),nr(ctx,YCOORD),nr(ctx,CONTROLPOINT),nr(ctx,CONTROLPOINT),nr(ctx,CONTROLPOINT),nr(ctx,CONTROLPOINT))

### TRANSFORM ###

135 def genTransform(ctx):
    fn = choice((genRotate, genTranslate, genScale, genSkew, genReset))
    return fn(ctx)

def genRotate(ctx):
140     return ctx.spaces() + """rotate(%s)\n""" % nr(ctx,ROTATION)

def genTranslate(ctx):
    return ctx.spaces() + """translate(%s,%s)\n""" % (nr(ctx,XCOORD), nr(ctx,YCOORD))

145 def genScale(ctx):
    return ctx.spaces() + """scale(%s)\n""" % (nr(ctx,SCALE))

def genSkew(ctx):
    return ctx.spaces() + """skew(%s)\n""" % (nr(ctx,SKEW))

150 def genReset(ctx):
    return ctx.spaces() + """reset()\n"""

### COLOR ###

155 def genColor(ctx):
    fn = choice((genFill,genFill,genFill,genFill,genFill,genFill,genStroke,genStroke,genStroke,genStroke,genStroke,genStroke))
    return fn(ctx)

```

```

160 def genFill(ctx):
    return ctx.spaces() + """"fill(%s,%s,%s,%s)\n"""" % (nr(ctx,COLOR),nr(ctx,COLOR), nr(ctx,COLOR), nr(c

    def genStroke(ctx):
    return ctx.spaces() + """"stroke(%s,%s,%s,%s)\n"""" % (nr(ctx,COLOR), nr(ctx,COLOR), nr(ctx,COLOR), n

165 def genNofill(ctx):
    return ctx.spaces() + """"nofill()\n""""

    def genNostroke(ctx):
170     return ctx.spaces() + """"nostroke()\n""""

    def genStrokewidth(ctx):
    return ctx.spaces() + """"strokewidth(%s)\n"""" % nr(ctx,STROKEWIDTH)

175 ### LOOP ###
    def genLoop(ctx):
        fn = choice((genFor, genGrid))
        return fn(ctx)

180 def genFor(ctx):
    if ctx._indent >= 2: return ""
    s = ctx.spaces() + """"for i in range(%s):\n"""" % nr(ctx,LOOP)
    ctx.indent()
    for i in range(random(5)):
185         s += genStatement(ctx)
    s += genVisual(ctx)
    ctx.dedent()
    return s

190 def genGrid(ctx):
    if ctx.inGrid(): return ""
    s = ctx.spaces() + """"for x, y in grid(%s,%s,%s,%s):\n"""" % (nr(ctx,GRIDCOUNT), nr(ctx,GRIDCOUNT),
    ctx.indent()
    ctx._grid = True
195     for i in range(random(5)):
        s += genStatement(ctx)
    s += genVisual(ctx)
    ctx.dedent()
    ctx._grid = False
200     return s

    ### MAIN ###

    def genVisual(ctx):
205         fn = choice((genDraw,))
        return fn(ctx)

    def genStatement(ctx):
        fn = choice((genVisual,genLoop,genColor,genTransform))
210         return fn(ctx)

    def genProgram():
        s = """"# This code is generated with OTTOBOT,
        # the automatic NodeBox code generator.
215 size(%s, %s)
        translate(%s, %s)
        colormode(HSB)
        """" % (COMP_WIDTH, COMP_HEIGHT, COMP_WIDTH/2, COMP_HEIGHT/2)
        ctx = Context()
220         for i in range(random(10,20)):
            s += genStatement(ctx)
        return s

```

```

    if __name__ == '__main__':
225     print genProgram()

```

## nodebox/util/QTSupport/\_\_\_init\_\_\_py

```

import os
import tempfile
import Foundation
NSNumber = Foundation.NSNumber
5
import AppKit
NSImage = AppKit.NSImage
NSApplication = AppKit.NSApplication
NSColor = AppKit.NSColor
10 NSData = AppKit.NSData
NSBitmapImageRep = AppKit.NSBitmapImageRep
NSJPEGFileType = AppKit.NSJPEGFileType

import QTKit
15 QTMovie = QTKit.QTMovie
QTDataReference = QTKit.QTDataReference
QTMovieFileNameAttribute = QTKit.QTMovieFileNameAttribute
QTMakeTimeRange = QTKit.QTMakeTimeRange
QTMakeTime = QTKit.QTMakeTime
20 QTMovieEditableAttribute = QTKit.QTMovieEditableAttribute
QTAddImageCodecType = QTKit.QTAddImageCodecType
QTMovieFlatten = QTKit.QTMovieFlatten

class Movie(object):
25
    def __init__(self, fname, fps=30):
        if os.path.exists(fname):
            os.remove(fname)
        self.frame = 1
30     self.fname = fname
        self.tmpfname = None
        self.firstFrame = True
        self.movie = None
        self.fps = fps
35     self._time = QTMakeTime(int(600/self.fps), 600)

    def add(self, canvas_or_context):
        if self.movie is None:
            # The first frame will be written to a temporary png file,
            # then opened as a movie file, then saved again as a movie.
40         handle, self.tmpfname = tempfile.mkstemp('.tiff')
            canvas_or_context.save(self.tmpfname)
            try:
                movie, err = QTMovie.movieWithFile_error_(self.tmpfname, None)
45                 movie.setAttribute_forKey_(NSNumber.numberWithBool_(True), QTMovieEditableAttribute)
                range = QTMakeTimeRange(QTMakeTime(0,600), movie.duration())
                movie.scaleSegment_newDuration_(range, self._time)
                if err is not None:
                    raise str(err)
50                 movie.writeToFile_withAttributes_(self.fname, {QTMovieFlatten:True})
                self.movie, err = QTMovie.movieWithFile_error_(self.fname, None)
                self.movie.setAttribute_forKey_(NSNumber.numberWithBool_(True), QTMovieEditableAttribute)
                if err is not None:
                    raise str(err)
55                 self.imageTrack = self.movie.tracks()[0]
            finally:
                os.remove(self.tmpfname)
        else:

```

```

        try:
80             canvas_or_context.save(self.tmpfname)
                img = NSImage.alloc().initWithReferencingFile_(self.tmpfname)
                self.imageTrack.addImage_forDuration_withAttributes_(img, self._time, {QTAddImageCodecT
        finally:
            try:
85                 os.remove(self.tmpfname)
            except OSError, err:
                print err
                # pass
            self.frame += 1

70     def save(self):
        self.movie.updateMovieFile()

    def test():
80         import sys
        sys.path.insert(0, '../..')
        sys.path.insert(0, '../../..')
        from nodebox.graphics import Canvas, Context
        from math import sin

80         NSApplication.sharedApplication().activateIgnoringOtherApps_(0)
        w, h = 500, 300
        m = Movie("xx3.mov")
        for i in range(200):
85             print "Frame", i
            ctx = Context()
            ctx.size(w, h)
            ctx.rect(100.0+sin(i/10.0)*100.0,i/2.0,100,100)
            ctx.text(str(i), i*2, 200)
90             m.add(ctx)
        m.save()

    if __name__=='__main__':
        test()

```

## nodebox/util/vdiff.py

```

import os
import PIL.Image as Image

HTML_HEADER = r'''
5 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>Vdiff Test Results</title>
<style type="text/css" media="all">
10 body { margin: 20px 0 20px 150px; }
    body, td, th { font: 11px/1.5em "Lucida Grande", sans-serif; }
    h1 { font-size: 160%; padding: 0; margin: 0em 0 -2em 0; }
    h2 { font-size: 130%; padding: 0; margin: 4em 0 0.2em 0; clear:both; }
    img { float: left; border: 1px solid #000; margin: 2px; }
15 .different table { background: red; }
    table.statistics { margin:2px; width:16em; border:1px solid #666; }
    table.statistics td { font-weight: bold; text-align: right; padding: 2px 5px; }
    table.statistics td + td { font-weight: normal; text-align: left; }
    tr.even { background: #eee; }
20 tr.odd { background: #ddd; }
</style>
</head>
<body>
<h1>vdiff tests</h1>

```

```

25 '''
    HTML_FOOTER = r'''
    </body>
    </html>
30 '''

    def format_stats(stats):
        if stats.number_of_differences > 0:
            clz = " different"
35         else:
            clz = ""

        html = """<h2>%s</h2>\n""" % stats.name
        html += """<div class="stats%s">""" % clz
40         html += """<a href="%s" target="_blank"></a>\n""" % (stats.f
        html += """<a href="%s" target="_blank"></a>\n""" % (stats.f
        if stats.comparison_image_fname is not None:
            html += """<a href="%s" target="_blank">
        html += """<table class="statistics" height="152">\n"""
45         html += """<tr class="odd"><td>Differences:</td><td>%i</td></tr>\n""" % len(stats.differences)
        html += """<tr class="even"><td>Total delta:</td><td>%i</td></tr>\n""" % stats.total_delta
        html += """<tr class="odd"><td>Max delta:</td><td>%i</td></tr>\n""" % stats.max_delta
        html += """<tr class="even"><td>Mean:</td><td>%.4f</td></tr>\n""" % stats.mean
        html += """<tr class="odd"><td>Stdev:</td><td>%.4f</td></tr>\n""" % stats.stdev
50         html += """</table>\n"""
        html += """</div>"""
        return html

    def format_stats_list(stats_list):
55         html = HTML_HEADER
        for stats in stats_list:
            html += format_stats(stats)
        html += HTML_FOOTER
        return html

60     def compare_pixel(px1, px2):
        if px1 == px2:
            return 0
        r1, g1, b1, a1 = px1
65         r2, g2, b2, a2 = px2
        return abs(r1-r2) + abs(g1-g2) + abs(b1-b2) + abs(a1-a2)

    def visual_diff(img1, img2, threshold=0, stop_on_diff=False):
        if isinstance(img1, str) or isinstance(img1, unicode):
70             img1 = Image.open(img1)
            img1 = img1.convert("RGBA")
        if isinstance(img2, str) or isinstance(img2, unicode):
            img2 = Image.open(img2)
            img2 = img2.convert("RGBA")
75         assert img1.size == img2.size
        w, h = img1.size
        data1 = img1.getdata()
        data2 = img2.getdata()
        size = len(data1)
80         differences = []
        for i in xrange(size):
            delta = compare_pixel(data1[i], data2[i])
            if delta > threshold:
                x = i % w
85                 y = i / w
                differences.append( ( (x, y), data1[i], data2[i], delta ) )
                if stop_on_diff:
                    # print data1[i], data2[i]

```

```

        break
90     return differences

    def make_comparison_image(size, differences):
        img = Image.new("L", size, color=255)
        for pos, d1, d2, delta in differences:
95             img.putpixel(pos, 255-delta)
        return img

    def isEqual(fname1, fname2, threshold=0):
        diff = visual_diff(fname1, fname2, threshold, stop_on_diff=True)
100        if len(diff) == 0:
            return True
        return False

    class Statistics(object):
105        def __init__(self, fname1, fname2, differences=None, name=""):
            self.fname1 = fname1
            self.fname2 = fname2
            if differences is None:
                differences = visual_diff(fname1, fname2)
110            self.differences = differences
            self.name = name

            img1 = Image.open(fname1)
            self.width, self.height = img1.size

115            self._comparison_image = None
            self.comparison_image_fname = None
            self.calculate()

120        def calculate(self):
            diff = self.differences

            total_delta = 0
            max_delta = 0
125            for pos, d1, d2, delta in diff:
                total_delta += delta
                max_delta = max(max_delta, delta)
            self.total_delta = total_delta
            self.max_delta = max_delta
130            self.mean = mean = total_delta / float(self.width * self.height)

            stdev = 0
            for pos, d1, d2, delta in diff:
                stdev += pow(delta-mean, 2)
135            stdev /= float(self.width * self.height)
            self.stdev = stdev

        def _get_size(self):
            return self.width, self.height
140        size = property(_get_size)

        def _get_number_of_differences(self):
            return len(self.differences)
        number_of_differences = property(_get_number_of_differences)

145        def _get_comparison_image(self):
            if self._comparison_image is None:
                self._comparison_image = make_comparison_image(self.size, self.differences)
            return self._comparison_image
150        comparison_image = property(_get_comparison_image)

        def save_comparison_image(self, fname):

```

```

        self.comparison_image.save(fname)
        self.comparison_image_fname = fname
155
    def __str__(self):
        return "<Statistics diff:%s total_delta:%s max_delta:%s mean:%.4f stdev:%.4f>" % (
            len(self.differences), self.total_delta, self.max_delta, self.mean, self.stdev)

160 def statistics(fname1, fname2, threshold=0):
    diff = visual_diff(fname1, fname2)
    stats = Statistics(fname1, fname2, diff)

    print "Differences:", len(stats.differences)
165    print "Total delta:", stats.total_delta
    print "Max delta:", stats.max_delta
    print "Mean:", stats.mean
    print "Stdev:", stats.stdev

170    stats.comparison_image.save('cmp.png')

    def test_vdiff(self):
        #fname1 = 'vdiff-tests/001-added-square/original.png'
        #fname2 = 'vdiff-tests/001-added-square/bluesquare.png'
175
        #fname1 = 'vdiff-tests/002-antialiased-text/preview.png'
        #fname2 = 'vdiff-tests/002-antialiased-text/photoshop.png'

        #fname1 = 'vdiff-tests/003-movement/original.png'
180    #fname2 = 'vdiff-tests/003-movement/moved.png'

        #fname1 = 'vdiff-tests/004-color/original.png'
        #fname2 = 'vdiff-tests/004-color/darker.png'

185    #fname1 = 'vdiff-tests/005-antialiased-text/none.png'
        #fname2 = 'vdiff-tests/005-antialiased-text/smooth.png'

        #fname1 = 'vdiff-tests/006-totally-different/ant.png'
        #fname2 = 'vdiff-tests/006-totally-different/people.png'
190

        fname1 = 'vdiff-tests/007-black-white/black.png'
        fname2 = 'vdiff-tests/007-black-white/white.png'

        statistics(fname1, fname2)

195
    def usage():
        print """vdiff -- visually compare images
Usage: vdiff <image1> <image2> [threshold]"""

200 if __name__=='__main__':
    import sys
    if len(sys.argv) < 3:
        usage()
    else:
205        fname1 = sys.argv[1]
        fname2 = sys.argv[2]
        try:
            threshold = int(sys.argv[3])
        except:
210            threshold = 0
        statistics(fname1, fname2, threshold)

```