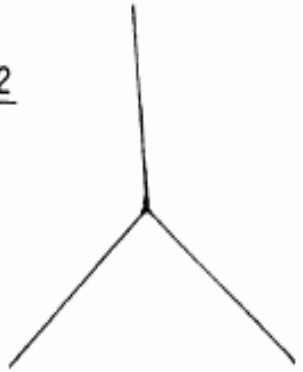

Extended Turtle Graphics
The last example script I wrote before releasing xturtle.py (V 0.9)
on the occasion of George W. Bush's visit to Vienna (22. 6. 2006)

Subsequently Daniel Ajoy (on Edu-sig) pointed out that a similar thing was already done approximately 35 years ago ....

The following commands will cause the turtle to draw Figure 2.

```
PENDOWN
FORWARD 100
RIGHT 60
FORWARD 100
BACK 100
LEFT 120
FORWARD 100
```

Figure 2



PEACE

We can now make a procedure using the old procedure PEACE as a sub-procedure:

```
TO SUPERPEACE
1  ARC 200 360
2  RIGHT 90
3  PEACE
END
```

SUPERPEACE

Figure 6

Critical remark: this resembles more the Mercedes star than the peace logo

by ...

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
A.I. LABORATORY

July 1971

Artificial Intelligence
Memo No. 249

LOGO
Memo No. 4

TEACHING CHILDREN TO BE MATHEMATICIANS VS. TEACHING ABOUT MATHEMATICS[1]

Seymour Papert

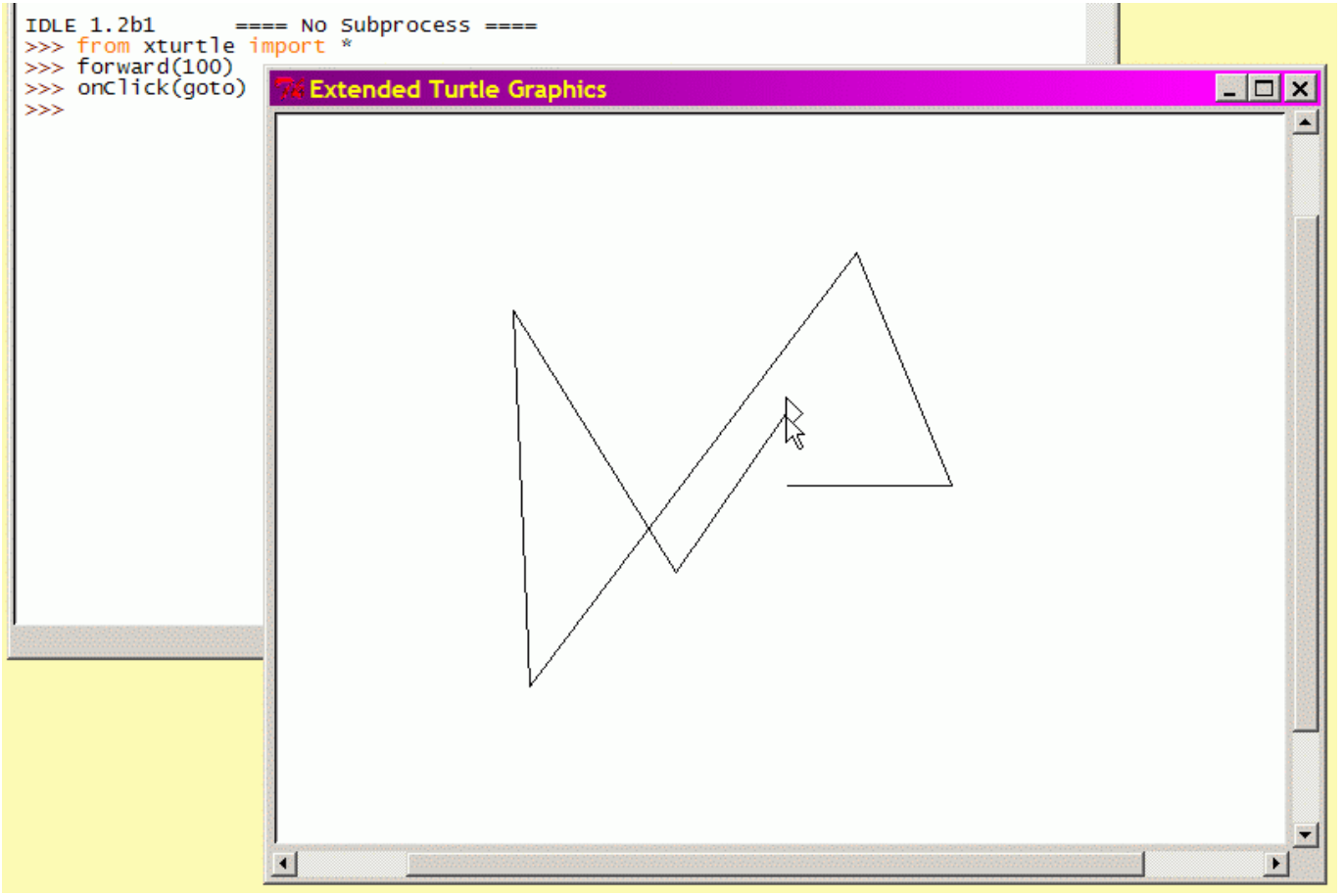
xturtle – a Tkinter based turtle module for Python

Author: Gregor Lingl

Profession: Teacher at a high school in Vienna/Austria

xturtle's aim: easy access to a versatile and fun to use graphics toolkit

```
From xturtle import *
>>> onClick(goto)
```



...clicking on the canvas makes the turtle move

Why do I propose a **new turtle module**?

- I use **turtle graphics** in my introductory programming lessons and in „Python für Kids“ **as a central tool to visualize computing concepts**.

- turtle.py is a suitable tool to do this

- but it has some deficiencies

Turtlegraphics (type 1): a mathematical tool to produce certain types of (very often recursive) pictures.

Turtlegraphics (type 2): an interactive computing environment.

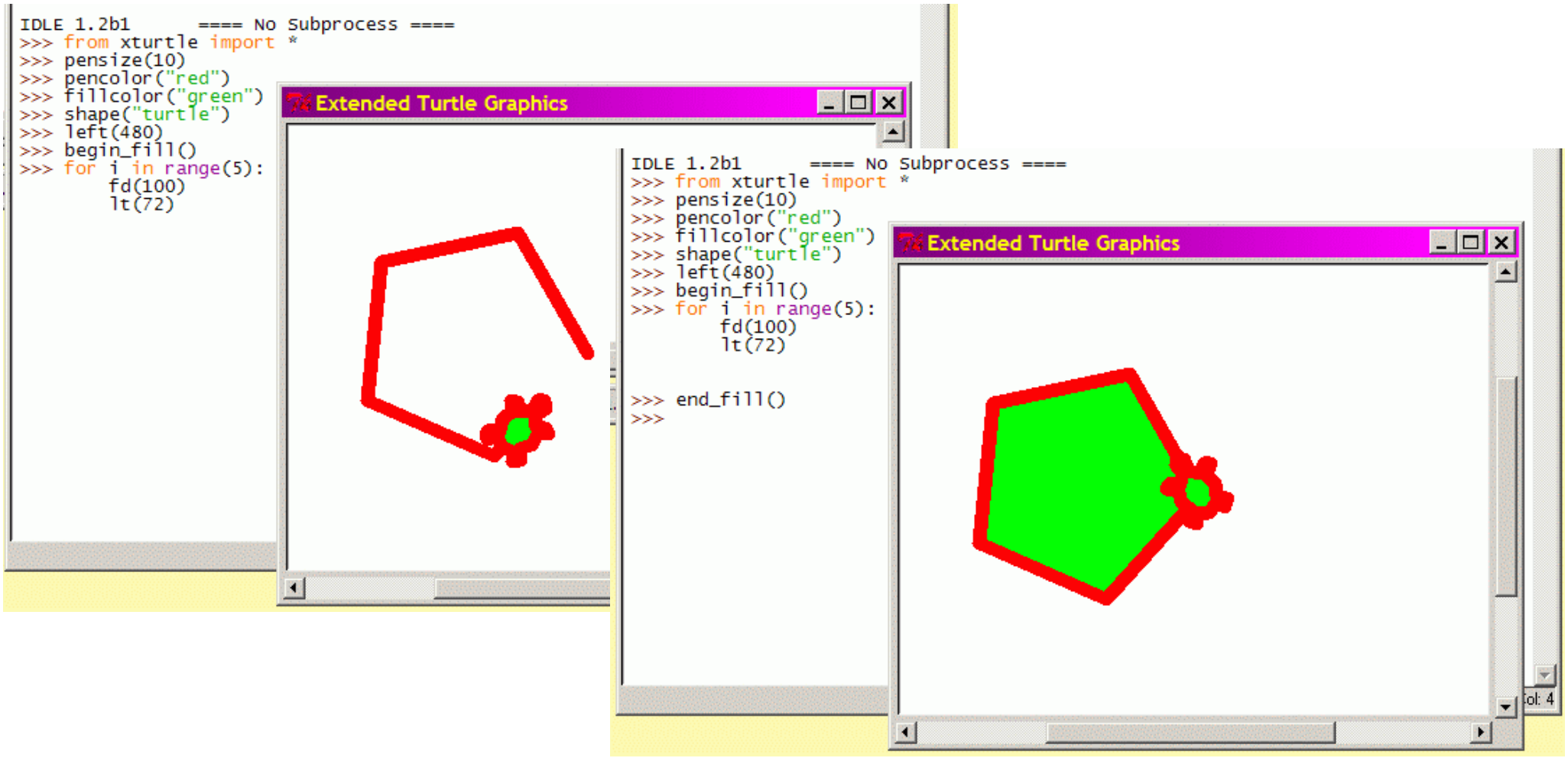
turtle.py (imho) was a compromise with some significant benefits:

- Turtles (RawPens) as objects
- All methods also available as functions
- interactive, but only partial animation

xturtle.py keeps the merits of turtle.py and at the same time enhances features to be used as type 2 - turtlegraphics...

The content of the xturtle.py - bundle:

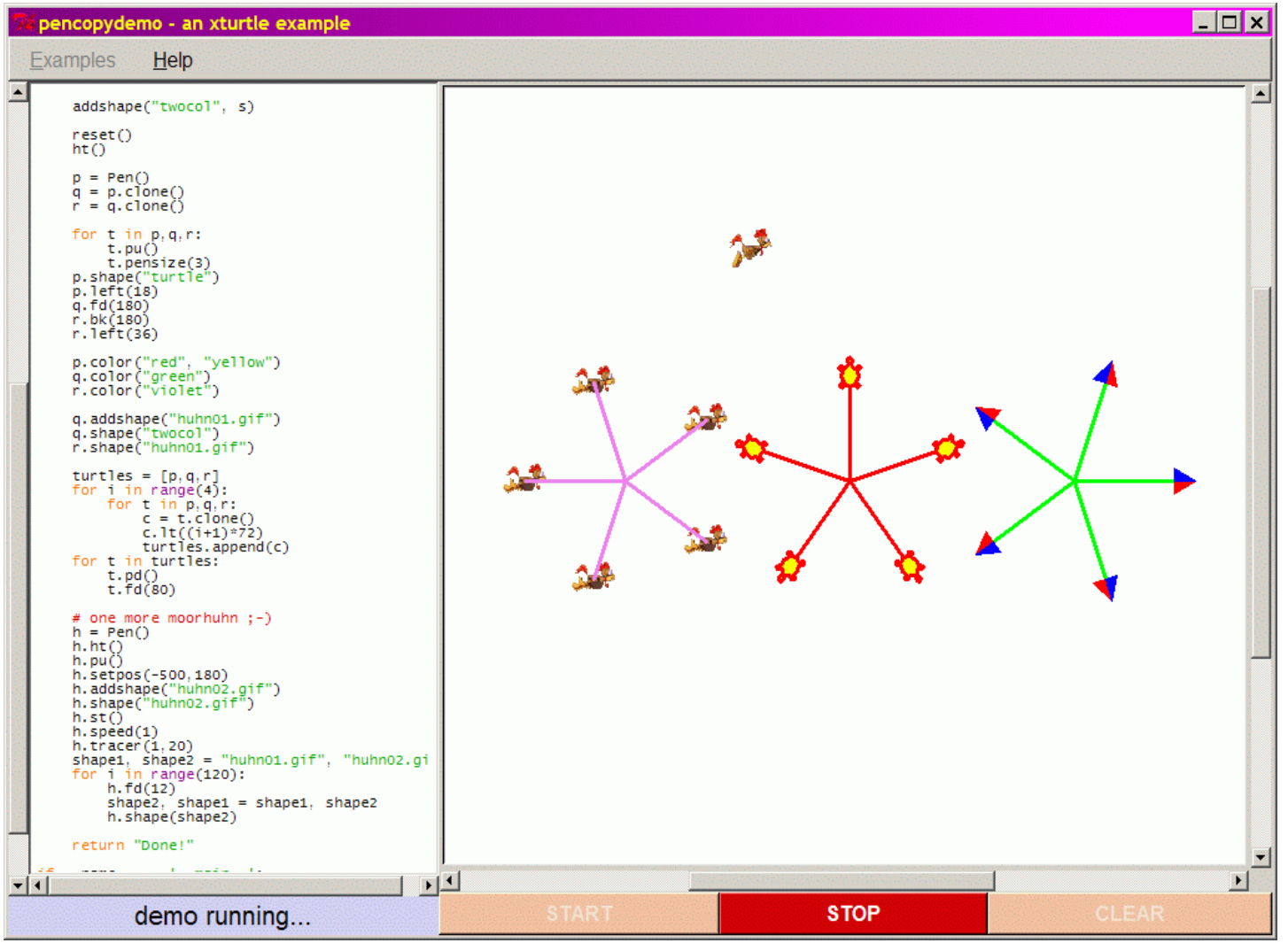
- xturtle.py – the module
- xturtleDemo.py – a DemoViewer
- an example – suite (you will see some of the examples)
- some Docs in *.txt format



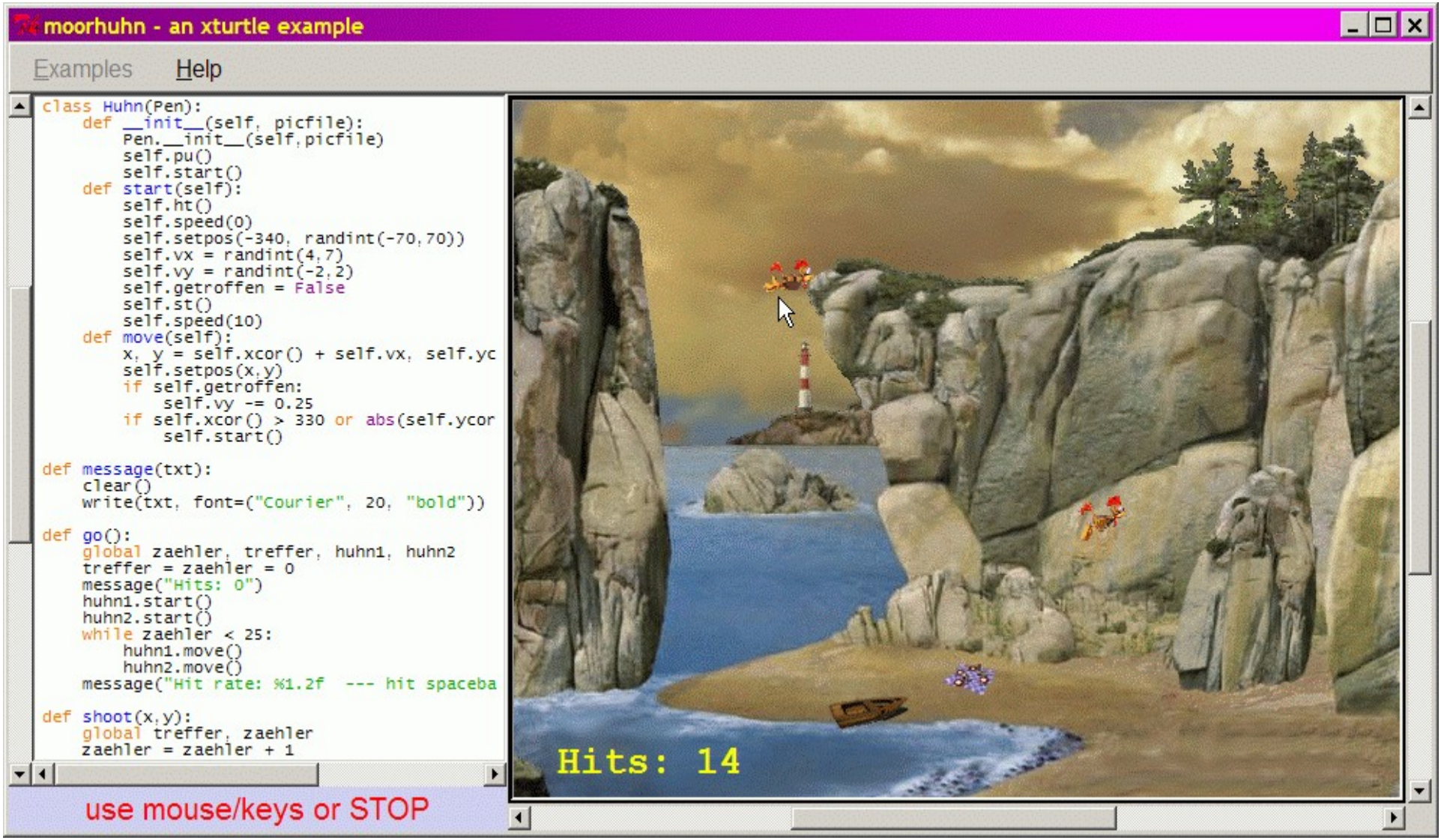
New features of xturtle.py

- Better animation of turtle movements, including representation of colors and pensize
- Different turtle shapes, gif-images as turtle shapes, user defined and user controllable turtle shapes
- Clonable turtles

This example running in the xturtle - DemoViewer



- Fine control over turtle movement and screen updates
- Aliases for the most commonly used commands, like fd for forward etc.
- Some simple commands/methods for creating event driven programs, e. g. games (mouse-, key-, timer-events)



- A scrollable Canvas class.
- Commands for controlling background color or background image
- The implementation of xturtle uses a 2-vector class _Vec, which can also (explicitly) be imported by the application programmer.

As an example of the use of `_Vec` a simulation of gravitational systems is included in the xturtle-package. The physics of this simulation is essentially contained in two methods of class `Star(Pen)`:

```
def acc(self):
    a = _Vec(0,0)
    for planet in self.gravSys.planets:
        if planet != self:
            r = planet.pos()-self.pos()
            a += (G*planet.m/abs(r)**3)*r
    return a

def step(self):
    self.setpos(self.pos() + self.dt*self.vel)
    if self != sun:
        self.setheading(self.towards(sun))
    self.vel = self.vel + self.dt*self.acc()
```

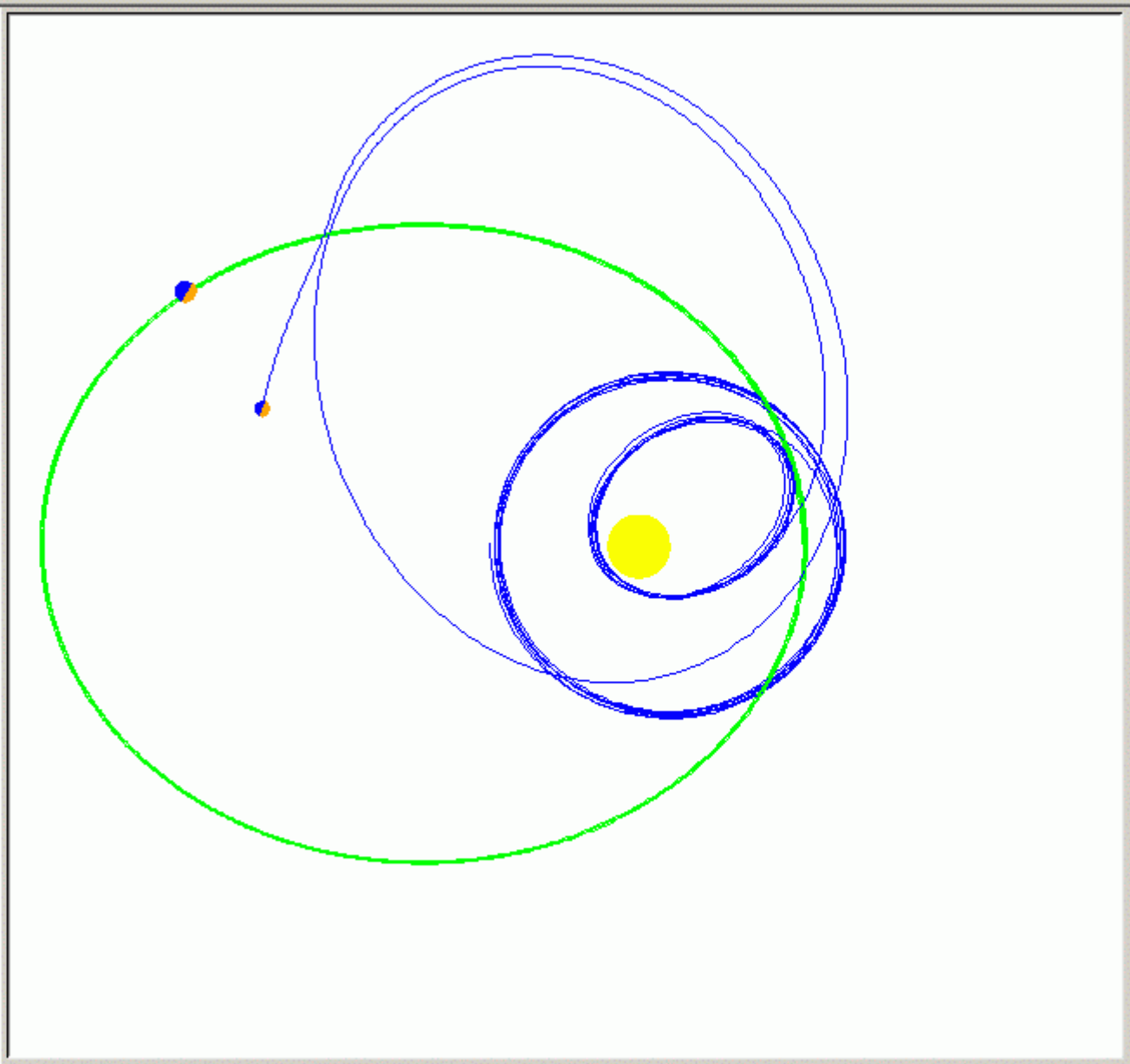
Method `pos()` returns a vector

Makes bright side of planets head towards the 'sun'

planets - an xturtle example

Examples Help

```
pressing the left mouse button with mouse
on the scrollbar of the canvas.

"""
from xturtle import _Vec, Shape, Pen, mainloo
from time import sleep

G = 8    # gravitational constant

class GravSys(object):
    def __init__(self):
        self.planets = []
        self.dt = 0.01
    def init(self):
        for planet in self.planets:
            planet.init()
    def start(self):
        for i in range(20000):
            for planet in self.planets:
                planet.step()

class Star(Pen):
    def __init__(self, m, x, v, gravSys, shap
        Pen.__init__(self, shape)
        self.resizemode("user")
        gravSys.planets.append(self)
        self.gravSys = gravSys
        self.dt = self.gravSys.dt
        self.penup()
        self.m = m
        self.setpos(x)
        self.vel = v
        self.pendown()
    def init(self):
        self.vel = self.vel + 0.5*self.dt*sel
    def acc(self):
        a = _Vec(0,0)
        for planet in self.gravSys.planets:
            if planet != self:
                r = planet.pos()-self.pos()
                a += (G*planet.m/abs(r)**3)*r
        return a
    def step(self):
        self.setpos(self.pos() + self.dt*self
        if self != sun:
            self.setheading(self.towards(sun)
        self.vel = self.vel + self.dt*self.ac

## create compound yellow/blue turtleshape fo
## yellow semicircle will always point toward
def createPlanetShape():
    s = Pen()
    s.tracer(0,0)
    s.ht()
    s.pu()
```



demo running...

START STOP CLEAR

The module xturtle has two main classes: `RawPen` and `TurtleScreen`. The structure is like this:

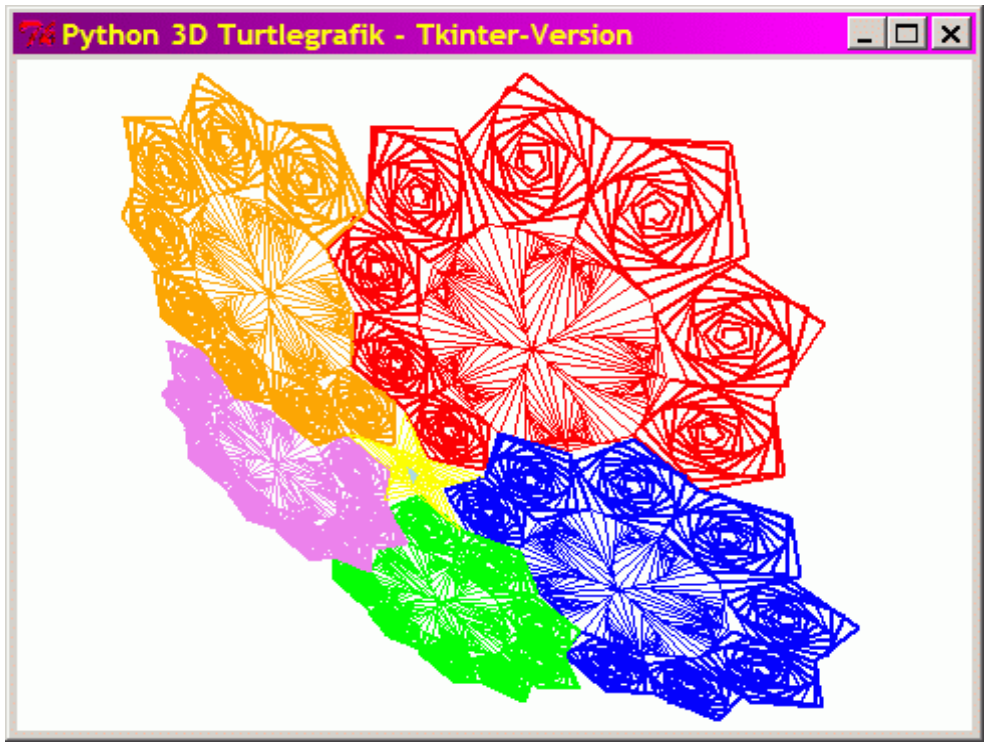
```
class Pen(RawPen)
 |  Method resolution order:
 |      Pen
 |      RawPen
 |      TPen
 |      TNavigator
 |      __builtin__.object

################################

class RawPen(TPen, TNavigator):
    """Animation part of the RawPen.
    Put RawPen upon a TurtleScreen and
    provide tools for it's animation.
    """
```

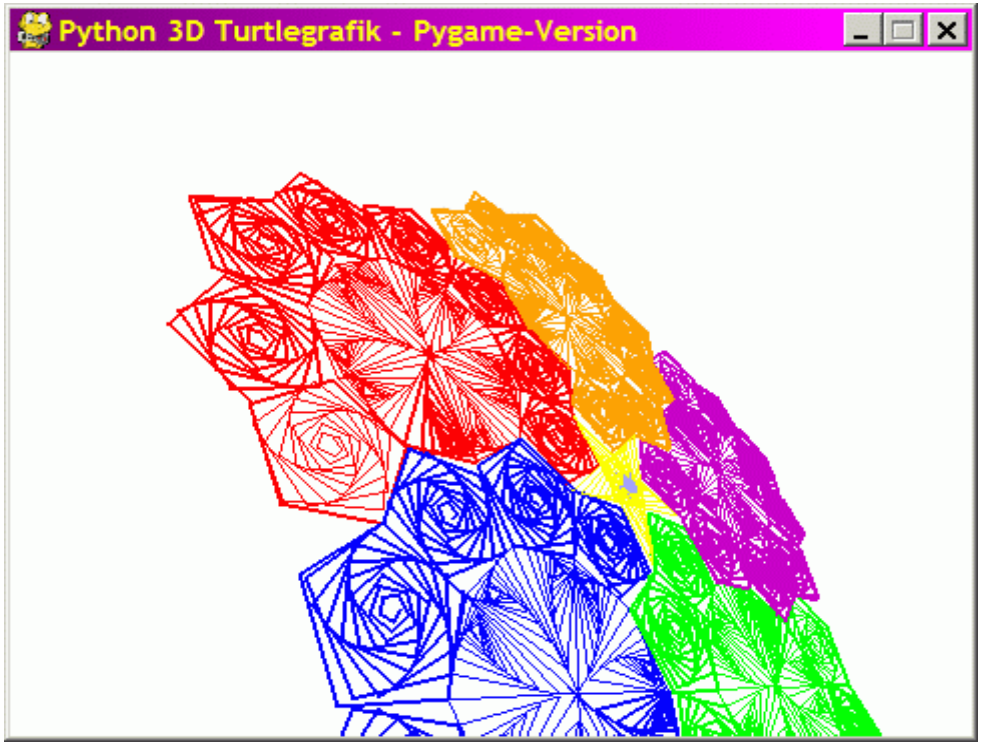
`RawPen` inherits from `TPen`, which contains the drawing stuff, and from `TNavigator`, which contains the stuff for the turtle movement.

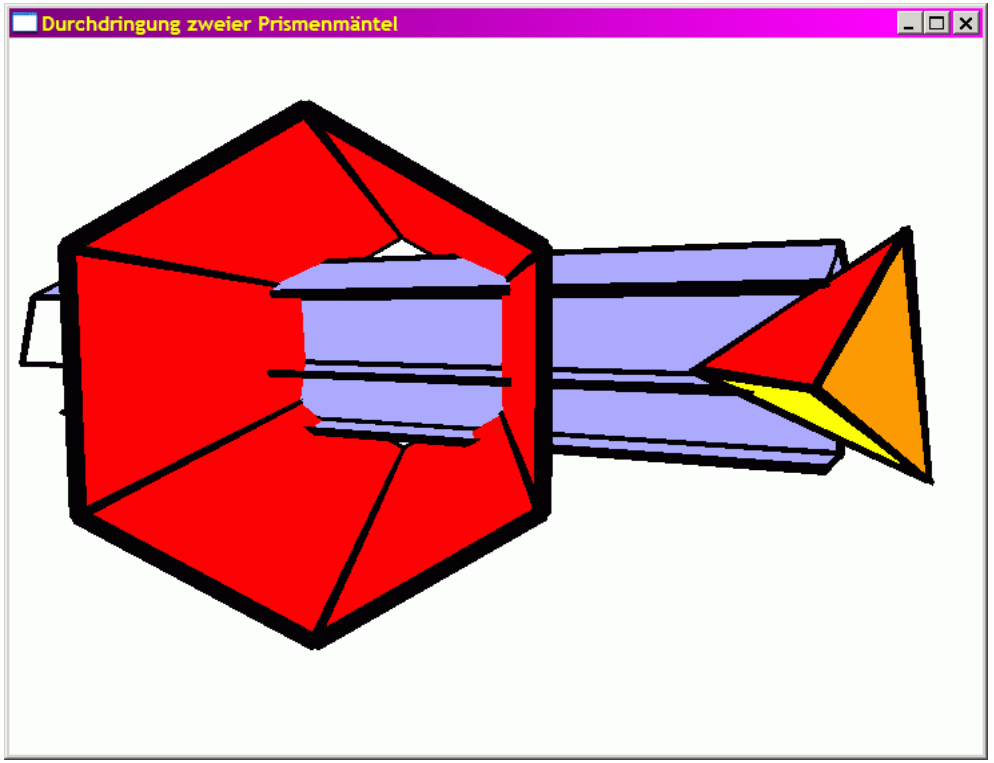
```
class TurtleScreen(TurtleScreenBase)
 |  Provides screen oriented methods like setbg etc.
 |
 |  All those methods are also transferred to the RawPen
 |  class, so they can be called as RawPen methods or
 |  - consequently - functions.


class TurtleScreenBase(object)
 |  Provides the basic graphics functionality.
 |  Interface between Tkinter and xturtle.py.
 |
 |  To port xturtle.py to some other graphics toolkit,
 |  a corresponding TurtleScreenBase class has to be
 |  implemented.
```

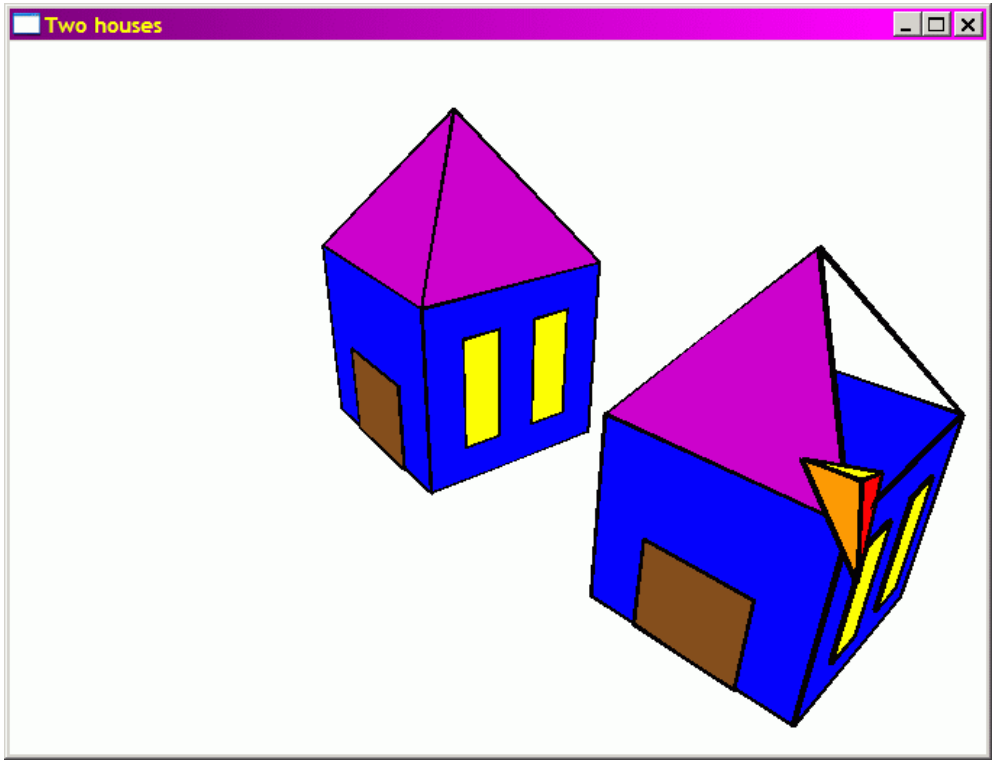
This design makes it easy to extend xturtle.py in several directions, e.g. towards using a 3D-turtle.

Presently such extensions exist only for a prototype of xturtle:


Python 3D Turtlegrafik - Tkinter-Version


Python 3D Turtlegrafik - Pygame-Version


Durchdringung zweier Prismenmäntel


Two houses

Current state of the project:

- It is submitted as a “patch“ to turtle.py, which means it is proposed as a replacement.

- This will possibly take place in Python 2.6 but depends on successful reviewing.

- In the meantime it would be fine to build a user community, to discuss and possibly enhance xturtle's features and especially the API. This is already emerging.

Next steps (e. g.):

- Discuss and, if desired, modify API
- Revise example scripts
- Translate Docs into other languages -
  e. g. German ;)
- Create supplementary learning/teaching materials
- Port to other graphics toolkits
- Check pros and cons of 3D extension

Finally:

- Python is less widespread in the educational area than it deserves (and than kids deserve)

- Having a battery included that gives immediate access to graphics programming with a great variety of possible applications could help to better the situation.

- It could especially help to convince more teachers to use Python as a first programming language.

Infos and contact:

Website:

http://ada.rg16.asn-wien.ac.at/~python/xturtle
http://python4kids.net

Email:

glingl@aon.at

Discussion:

Edu-sig -- Python in education:
http://mail.python.org/mailman/listinfo/edu-sig
edu-sig@python.org

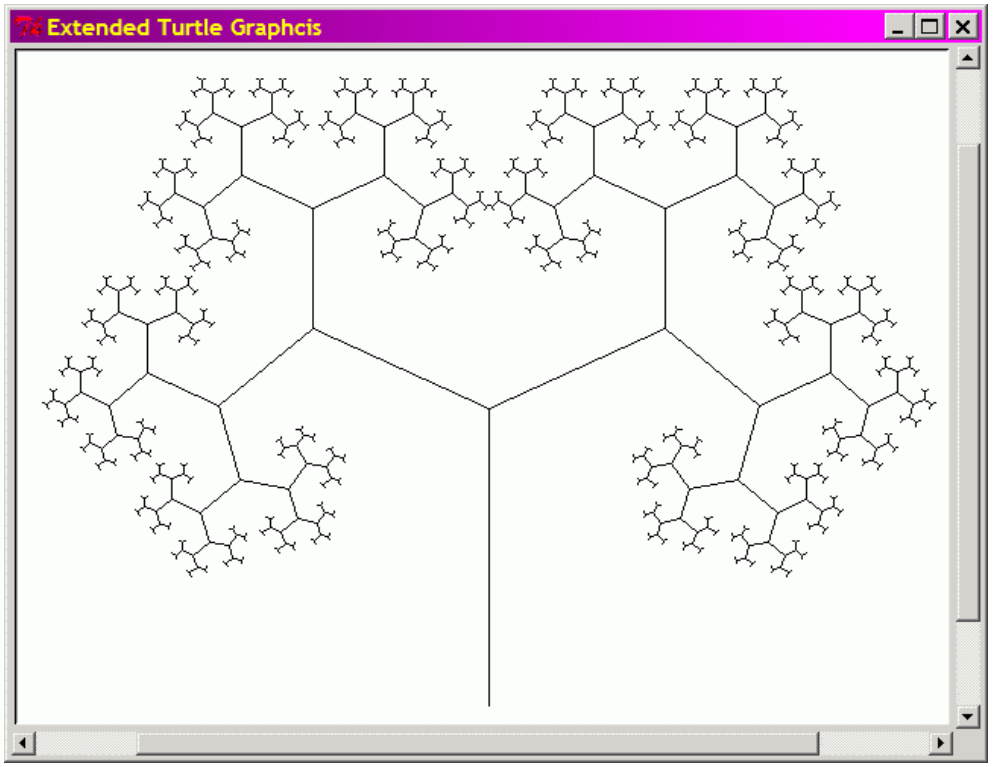
Set up especially for discussion of turtle-graphics related topics:
edupython@googlegroups.com

Three after-talk slides:

After the talk a group of interested attendees watched some additional demos on my laptop. Two of them follow:

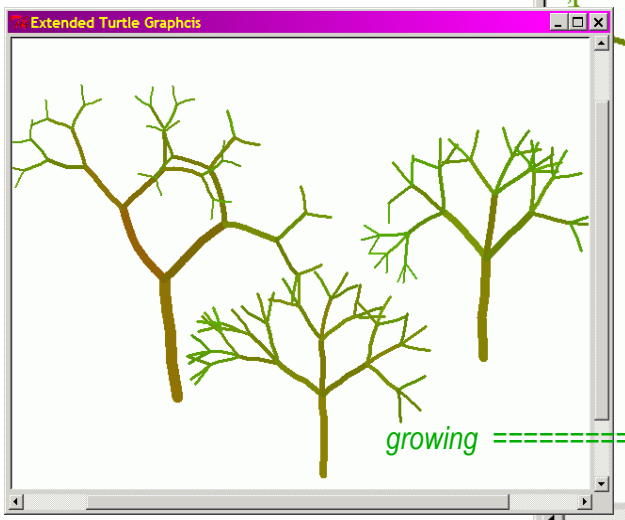
„Breadth first“ growing trees using Python's generators recursively:

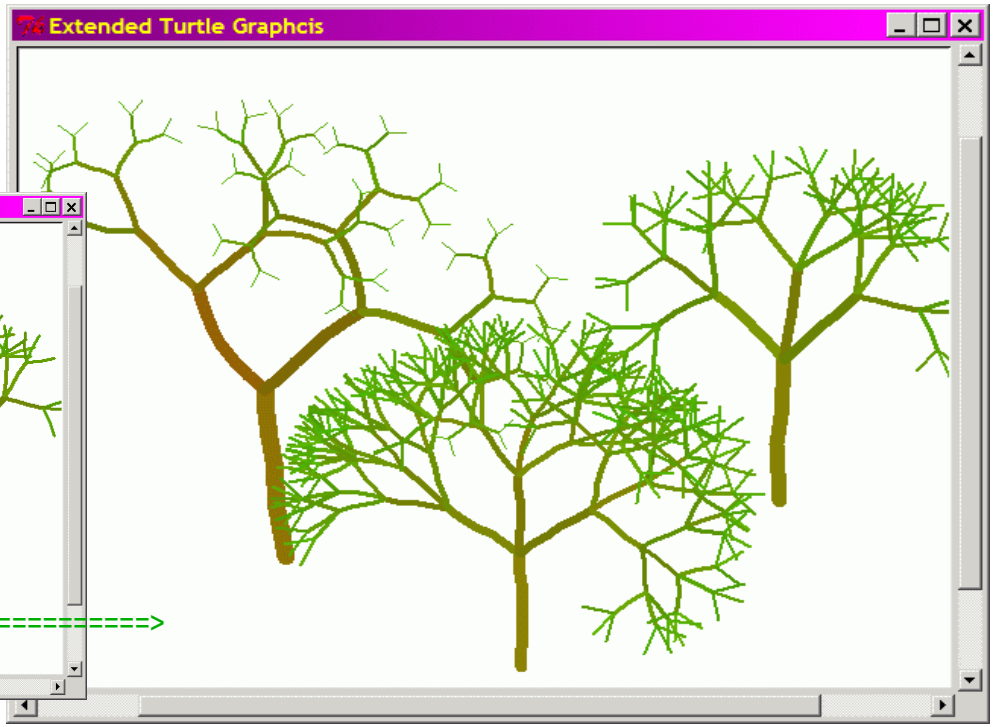
```
def tree(plist, l, a, f):
    """ plist is list of pens
    l is length of branch
    a is half of the angle between 2 branches
    f is factor by which branch is shortened
    from level to level."""
    if l > 3:
        lst = []
        for p in plist:
            p.forward(l)
            q = p.clone()
            p.left(a)
            q.right(a)
            lst.append(p)
            lst.append(q)
        for x in tree(lst, l*f, a, f):
            yield None
```



This code clones the turtles at each branching point, so it ends up with 512 (invisible) turtles.

One can do the same also a bit more fanciful, with more trees:





A tiny reminiscence of Alan Kay's keynote: