Operator Overloading
- the class DoubleDouble
- defining +, −, *, and /
- expression evaluation

Inheritance
- visualizing points in the plane
- representing lines in the plane
- defining the class Parabola
- extending the Parabola class for visualization
1. Operator Overloading
   - the class DoubleDouble
   - defining +, −, *, and /
   - expression evaluation

2. Inheritance
   - visualizing points in the plane
   - representing lines in the plane
   - defining the class Parabola
   - extending the Parabola class for visualization
Recall what we did in the second half of lecture 25:

1. We looked at a C program to use the double double arithmetic of the QD library to apply Newton’s method to approximate $\sqrt{2}$.

2. We defined the module `doubleDouble` that exports the basic functionality of the C interface of the QD library to perform double double arithmetic in Python.

In Python, a double double was seen as a tuple of two doubles.

The arithmetic was performed by methods that took 4 arguments: the high and low parts of both operators.
operator overloading

To add two double doubles \( x \) and \( y \), instead of
\[
z = \text{doubleDouble.add}(x[0], x[1], y[0], y[1])
\]
we would like to write \( z = x + y \).

We will define the class \texttt{DoubleDouble()} and export the method \texttt{__add__}.

Defining the method \texttt{__add__} allows to use the + operator on any two elements of the class \texttt{DoubleDouble}.

The + is a binary operator:

1. the \texttt{self} is the first operand of +,
2. \texttt{other} is the second operand of +.

The definition starts with \texttt{def __add__(self, other):}.

Subtraction, multiplication, and division are defined respectively by \texttt{__sub__}, \texttt{__mul__}, and \texttt{__div__}. 
data attributes of the class DoubleDouble

Importing the module `doubleDouble` that wraps some of the basic functionality of the QD library:

```python
import doubleDouble

class DoubleDouble():
    
    # Wraps some of the functionality of the QD library to perform double double arithmetic.
    
    def __init__(self, hi=0.0, lo=0.0):
        
        # A double double consists of a high and a low part.
        
        self.high = hi
        self.low = lo
```
def __str__(self):
    """
    Returns the string representation of a double double.
    """
    return doubleDouble.str(self.high,self.low)

def __repr__(self):
    """
    Returns the representation of a double double.
    """
    return self.__str__()

def copy(self):
    """
    Returns a copy of the double double.
    """
    return DoubleDouble(self.high,self.low)
Operator Overloading
- the class DoubleDouble
- defining +, −, ∗, and /
- expression evaluation

Inheritance
- visualizing points in the plane
- representing lines in the plane
- defining the class Parabola
- extending the Parabola class for visualization
def __add__(self, other):
    """
    Returns the sum of two double doubles.
    """
    z = doubleDouble.add(self.high, self.low,
                         other.high, other.low)
    return DoubleDouble(z[0], z[1])

def __sub__(self, other):
    """
    Returns the difference of self and the other.
    """
    z = doubleDouble.sub(self.high, self.low,
                         other.high, other.low)
    return DoubleDouble(z[0], z[1])
multiplication and division

def __mul__(self, other):
    """
    Returns the product of two double doubles.
    """
    z = doubleDouble.mul(self.high, self.low,
                         other.high, other.low)
    return DoubleDouble(z[0], z[1])

def __div__(self, other):
    """
    Returns the division of self by the other.
    """
    z = doubleDouble.div(self.high, self.low,
                          other.high, other.low)
    return DoubleDouble(z[0], z[1])
computing powers

To compute \( x^{**5} \), we overload the \(__pow__\) method:

```python
def __pow__(self, n):
    """
    Returns self to the power n.
    """
    z = self.copy()
    for i in range(1, n):
        z = z * self
    return z
```

Note: it is more efficient to wrap \( \text{c_dd_npwr()} \).
def test():
    """
    Basic test on arithmetical operations.
    """

    hi = 1.4142135623730951
    lo = -9.6672933134529147e-17
    x = DoubleDouble(hi,lo)  # defines the sqrt(2)
    print 'x = sqrt(2) =', str(x)
    y = x**2
    print 'x*x =', y
    z = y/x
    print 'x*x/x =', z
    u = x+x
    print 'x+x =', u
    two = DoubleDouble(2.0)
    print '2*x =', two*x
    v = u-x
    print 'x+x-x =', v
running the test

Adding to the end of `double_double.py` the line

```python
if __name__=='__main__': test()
```

so we can run

```
$ python double_double.py
x = sqrt(2) = 1.4142135623730950488016887242096
x*x = 2.0000000000000000000000000000000
x*x/x = 1.4142135623730950488016887242096
x+x = 2.8284271247461900976033774484193
2*x = 2.8284271247461900976033774484193
x+x-x = 1.4142135623730950488016887242096
```
Newton’s method for $\sqrt{2}$

def newton4sqrt2():
    """
    Applies Newton’s method to approximate $\sqrt{2}$.
    """
    x = DoubleDouble(2.0)
    print 'step 0 :', x
    for i in range(1,9):
        z = x**2
        z = z + DoubleDouble(2.0)
        z = z/x
        z = DoubleDouble(0.5)*z
        x = z.copy()
    print 'step', i, ':', x
operator overloading and inheritance

1. Operator Overloading
   - the class DoubleDouble
   - defining +, −, *, and /
   - expression evaluation

2. Inheritance
   - visualizing points in the plane
   - representing lines in the plane
   - defining the class Parabola
   - extending the Parabola class for visualization
Problem: Evaluate $f(x, y) =$

$$(333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

at $(77617, 33096)$.


Let us evaluate this expression with double doubles...
print 'checking the motivating interval arithmetic example'
f = lambda x,y: (DoubleDouble(333.75) - x**2)*y**6 
    + x**2*(DoubleDouble(11)*x**2*y**2 
    - DoubleDouble(121)*y**4 - DoubleDouble(2)) 
    + DoubleDouble(5.5)*y**8 + x/(DoubleDouble(2)*y);

a = 77617; b = 33096
z = f(DoubleDouble(a),DoubleDouble(b))

    # to check the answer with SymPy :
    import sympy as sp
    x,y = sp.var('x,y')
    g = (sp.Rational(33375)/100 - x**2)*y**6 
    + x**2*(11*x**2*y**2 - 121*y**4 - 2) 
    + sp.Rational(55)/10*y**8 
    + sp.Rational(1)*x/(2*y);
    print 'evaluating', g, 'at', (a,b)
    e = sp.Subs(g,(x,y),(a,b)).doit()
e15 = e.evalf(15)
print 'numerical value :', z
print 'exact value :', e, '~', e15
print 'error :', abs(e15 - z.high)

We run the script as follows:

$ python double_double_eval.py
checking the motivating interval arithmetic example
evaluating x**2*(11*x**2*y**2 - 121*y**4 - 2) + x/(2*y) + 1
numerical value : 1.17260394005317863185883490452018e+00
exact value : -54767/66192 ~ -0.827396059946821
error : 2.00000000000000
$
Object-oriented programming is good at defining

- models of objects, e.g.: points, lines, parabolas;
- hierarchies between objects, via `isinstance`.

A parabola is defined by

- a point: its focus; and
- a line: its directrix.

We will define a parabola, inheriting from the line.

This definition is advantageous for visualization.
Operator Overloading
- the class DoubleDouble
- defining +, -, *, and /
- expression evaluation

Inheritance
- visualizing points in the plane
- representing lines in the plane
- defining the class Parabola
- extending the Parabola class for visualization
class Point:
    
    Stores a point in the plane.
    
    def __init__(self, x=0, y=0):
        
        Defines the coordinates.
        
        self.x = x
        self.y = y
For a point \( p \), if we do `print p` then we want to see \((p.x, p.y)\):

```python
def __str__(self):
    """
    Returns the string representation.
    """
    return '(' + str(self.x) + ',' + str(self.y) + ')'
```
a test program

In the file with the class definition:

def main():
    """
    Instantiates two points.
    """
    p = Point()
    print 'p =', p
    q = Point(3,4)
    print 'q =', q

if __name__=="__main__": main()
drawing points

Suppose we want to draw a point on canvas.

As before, with Tkinter, we could create a GUI class with a method to draw a point on canvas.

Instead, extend the class `Point` with

- a data attribute: a canvas; and
- a method: to draw the point on canvas.

Benefit: focus on the object that matters: the point.
Inheriting from `Point`

```python
from Tkinter import *
from classpoint import *

class ShowPoint(Point):
    ""
    Extends the Point class
    with a draw method.
    ""
    def __init__(self,cv,x=0,y=0):
        ""
        Defines the coordinates
        and stores the canvas.
        ""
        Point.__init__(self,x,y)
        self.canvas = cv
```
def draw(self):
    ""
    Draws the point on canvas.
    ""
    a = self.x; b = self.y
    self.canvas.create_oval(a-3, \n        b-3,a+3,b+3,fill='SkyBlue2')
def main():
    """
    Shows 10 random points on canvas.
    """

top = Tk()
d = 400
c = Canvas(top, width=d, height=d)
c.pack()
from random import randint
L = []
for i in xrange(10):
    a = randint(6, d-6)
    b = randint(6, d-6)
    L.append(ShowPoint(c, a, b))
for p in L: p.draw()
top.mainloop()
ten random points
Operator Overloading
- the class DoubleDouble
- defining +, -, *, and /
- expression evaluation

Inheritance
- visualizing points in the plane
- representing lines in the plane
- defining the class Parabola
- extending the Parabola class for visualization
What is a line?

- $y = ax + b$ is a linear function. But what with vertical lines?
- $ax + by + c = 0$ is more general. But the algebraic expression is not suited for geometry.

In a geometric representation, every line has

- a basis point, given by $(x, y)$; and
- a direction, given by an angle $\theta$.

Then any point on the line is $(x + t \cos(\theta), y + t \sin(\theta))$. 
the class Line

from classpoint import *
from math import cos, sin

class Line(Point):
    """
    A line is a base point and
    a direction angle.
    """
    def __init__(self,x=0,y=0,a=0):
        """
        Defines base point and angle.
        """
        Point.__init__(self,x,y)
        self.angle = a
implications of inheritance

>>> from classpoint import *
>>> from classline import *
>>> L = Line(3,4,1.23)
>>> isinstance(L,Line)
True
>>> isinstance(L,Point)
True

Because of the inheritance, the line L is also an instance of the class Point.
We view a line as an extension of a point.

Different design: Point is an attribute in Line.
We override the string representation for Line, using the string representation defined in Point:

```python
def __str__(self):
    '''
    Returns the string representation.
    '''
    p = Point.__str__(self)
a = ', angle = ' + str(self.angle)
return p + a
```
line as function

def __call__(self,t):
    
    Returns a new point on the line.
    
    xt = self.x + t*cos(self.angle)
yt = self.y + t*sin(self.angle)
return Point(xt,yt)

Example: line through (2,1) with angle $\pi/4$:

```python
>>> from classline import Line
>>> from math import pi
>>> L = Line(2,1,pi/4)
>>> p = L(1)
>>> p
(2.70710678119,1.70710678119)
```
def main():
    """
    Instantiates two lines.
    """
    L = Line()
    print 'L =', L
    q = Line(3, 4, 10)
    p = q(4)
    print p

if __name__=='__main__': main()
eleven lines
extending the Line class

from Tkinter import *
from classpoint import *
from classline import *

class ShowLine(Line):
    """
    Extends the Line class with a draw method.
    """
    def __init__(self,c,d,x=0,y=0,a=0):
        """
        Defines the line and stores the canvas c of dimension d.
        """
        Line.__init__(self,x,y,a)
        self.canvas = c
        self.dimension = d
computing the range of the line

The extension of the class stores the canvas and the dimension of the canvas so we may compute the part of the line that fits on canvas.

The base point is \((x, y)\) and we want to compute the point \(x + t \cos(\theta) = d\), where \(d\) is the right edge of the canvas.

\[ t = \frac{d - x}{\cos(\theta)} \]

One special case: a vertical line.
def draw(self):
    "Draws the line on canvas."
    a = self.x; b = self.y
    self.canvas.create_oval(a-3, b-3, a+3, b+3, fill='SkyBlue2')
da = self.dimension - a
cs = cos(self.angle)
if cs + 1.0 != 1.0:
    t = da/cs
    p = Line.__call__(self,t)
    self.canvas.create_line(a,b,p.x,p.y)
t = a/cs
    q = Line.__call__(self,-t)
    self.canvas.create_line(q.x,q.y,a,b)
else: # vertical line
    self.canvas.create_line(a,0,a,self.dimension)
the test program

def main():
    
    Shows 11 lines on canvas.
    
    top = Tk()
    d = 400
c = Canvas(top,width=d,height=d)
c.pack()
from random import uniform, randint
from math import pi
L = [ShowLine(c,d,100,300,pi/2)]
for i in xrange(10):
    a = randint(6,d-6)
    b = randint(6,d-6)
    r = uniform(0,2*pi)
    L.append(ShowLine(c,d,a,b,r))
for e in L: e.draw()
top.mainloop()
Operator Overloading
- the class DoubleDouble
- defining +, -, *, and /
- expression evaluation

Inheritance
- visualizing points in the plane
- representing lines in the plane
- defining the class Parabola
- extending the Parabola class for visualization
The expression $y = ax^2 + bx + c$ is a quadratic which defines points $(x, y)$ on a parabola.

Problem: $y^2 = x$ is also a parabola.

Geometric definition: a parabola is the set of points
- at equal distance from a point: the focus
- at equal distance from a line: the directrix.

Thus we define a parabola by a line and a point.
focus and directrix
from classpoint import *
from classline import *

class Parabola(Line):
    
    """
    A parabola is defined by a line, its directrix and a point, its focus.
    """
    
def __init__(self,x,y,a,b,c):
        
        """
        Defines the line at (x,y) and angle a and focus (b,c).
        """
        Line.__init__(self,x,y,a)
        self.focus = Point(b,c)
is instance versus has a

```python
class Parabola(Line):
    def __init__(self, x, y, a, b, c):
        Line.__init__(self, x, y, a)
        self.focus = Point(b, c)
```

An instance of a Parabola

- is an instance of a Line,
- has an instance of a Point as data attribute.
def __str__(self):
    
    Returns the string representation.
    
    F = str(self.focus)
    L = Line.__str__(self)
    s = 'focus : ' + F
    s = s + ', directrix : ' + L
    return s
computing points on a parabola

We compute points on the parabola, running along the directrix
\((c_x, c_y) = (x + t \cos(\theta), y + t \sin(\theta))\).

Let the focus have coordinates \((f_x, f_y)\).

The point \((\alpha, \beta)\) on the parabola at \(t\) satisfies
\[
(\alpha - c_x)^2 + (\beta - c_y)^2 = (\alpha - f_x)^2 + (\beta - f_y)^2
\]
and the line spanned by \((\alpha, \beta)\) and \((c_x, c_y)\) is orthogonal
to the line spanned by \((x, y)\) and \((c_x, c_y)\).

This leads to a linear system in \(\alpha\) and \(\beta\).
making a parabola callable

def __call__(self, t):
    
    Returns a point on the parabola as far from the focus and the point obtained by evaluating the directrix.
    
    c = Line.__call__(self, t)
    fx = self.focus.x
    fy = self.focus.y
    r1 = fx**2 + fy**2 - c.x**2 - c.y**2
    d = 2*(fx - c.x)*(c.y - self.y) - 2*(fy - c.y)*(c.x - self.x)
if d + 1.0 != 1.0:
    r2 = c.x*(c.x - self.x) + c.y*(c.y - self.y)
    dx = (r1*(c.y - self.y) \n          - 2*r2*(fy - c.y))/float(d)
    dy = (2*(fx - c.x)*r2 \n          - (c.x - self.x)*r1)/float(d)
else:
    c10 = Line.__call__(self,10)
    d = 2*(fx - c.x)*(c10.y - self.y) \n        - 2*(fy - c.y)*(c10.x - self.x)
    r2 = c10.x*(c10.x - self.x) \n         + c10.y*(c10.y - self.y)
    dx = (r1*(c10.y - self.y) \n          - 2*r2*(fy - c.y))/float(d)
    dy = (2*(fx - c.x)*r2 \n          - (c10.x - self.x)*r1)/float(d)
return Point(dx,dy)
the test function

def main():
    ""
    Instantiates a parabola.
    ""
    p = Parabola(3,4,-1.23,10,0)
    print p
    q = p(4)
    print q

    if __name__=='__main__': main()
Operator Overloading
- the class DoubleDouble
- defining +, −, *, and /
- expression evaluation

Inheritance
- visualizing points in the plane
- representing lines in the plane
- defining the class Parabola
- extending the Parabola class for visualization
from Tkinter import *
from classparabola import *

class ShowParabola(Parabola):
    
    Extends the Parabola class
    with a draw method.
    
    def __init__(self,cv,d,x,y,a,b,c):
        
        Defines the parabola and stores
        the canvas cv of dimension d.
        
        Parabola.__init__(self,x,y,a,b,c)
        self.canvas = cv
        self.dimension = d
def draw(self):
    """
    Draws the parabola on canvas.
    """
    self.draw_directrix()
    self.draw_focus()
    for t in xrange(-1000,1000):
        p = Parabola.__call__(self,t)
        self.canvas.create_oval(p.x-1,  
                                p.y-1,p.x+1,p.y+1,fill='SkyBlue2')
the main program

def main():
    top = Tk()
    d = 400
    c = Canvas(top,width=d,height=d)
    c.pack()
    from math import pi
    from random import randint
    from random import uniform
    L = []
    for i in xrange(1):
        x = randint(6,d-6)
        y = randint(6,d-6)
        r = uniform(0,2*pi)
        a = randint(6,d-6);
        b = randint(6,d-6);
        L.append(ShowParabola(c,d,x,y,r,a,b))
    for e in L: e.draw()
    top.mainloop()
Summary + Exercises

Operator overloading makes double double arithmetic natural. By inheritance we defined points, lines, and parabolas. See Chapter 9 in the textbook for UML diagrams.

1. Wrap the basic operations for quad double arithmetic of the QD library and make the class QuadDouble. Use the class to evaluate the expression of the motivating example for interval arithmetic.

2. Bind mouse events to the canvas so the user can move the focus of the parabola. The parabola is redrawn automatically after each move of the focus.

3. Using inheritance extend the class DoubleDouble with an extra field count to count the number of arithmetical operations performed to compute a double double. For example, if two constants have counts $m$ and $n$, their sum has count $m + n$.

Project Two due Monday 29 October at 10AM.