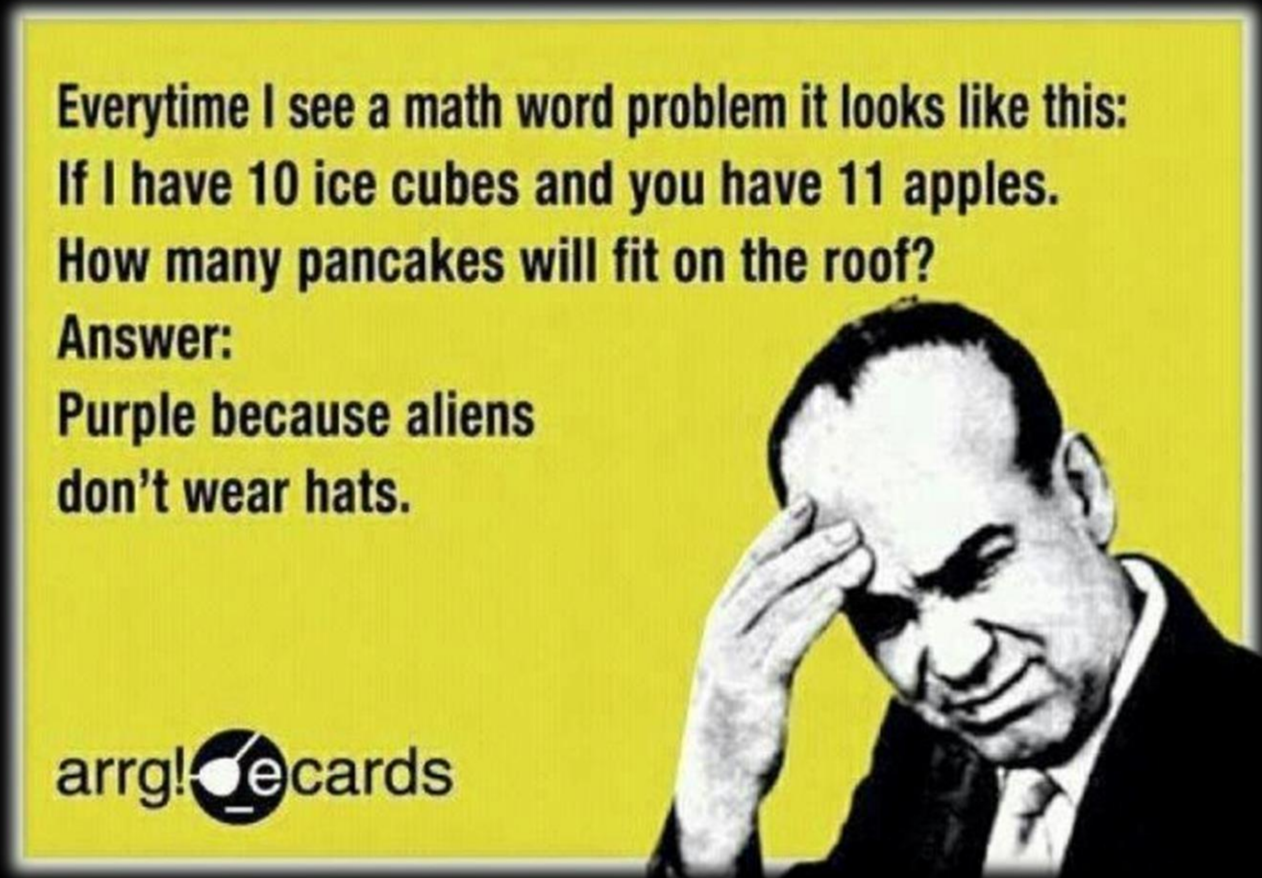

Everytime I see a math word problem it looks like this:
If I have 10 ice cubes and you have 11 apples.
How many pancakes will fit on the roof?
Answer:
Purple because aliens
don't wear hats.
arrg! ecards

Class Customizations and Better Code

Santiago Paiva
@Stronnics

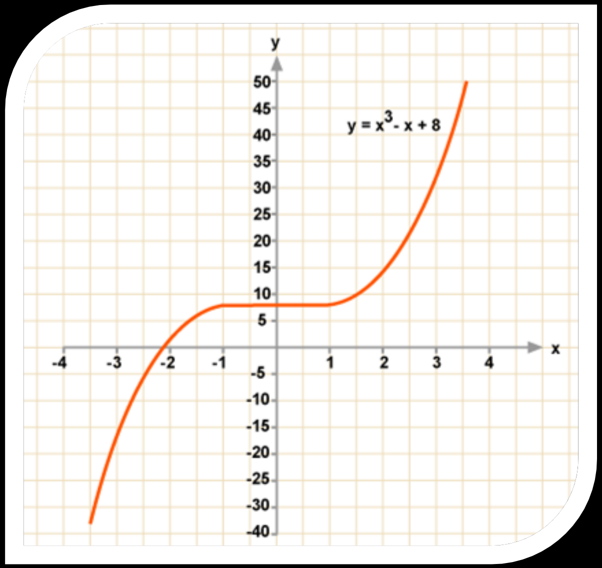
ConFoo.CA
WEB TECHNO CONFERENCE

Aims

- Python's Class customization
  - Better code
- Solving cubic equations
  - Difficult to design & test
- Calculation on demand

Introduction

- Example: Cubic class
  - Models cubic equations
  - getRoots(): x_1, x_2, x_3



$$ax^3 + bx^2 + cx + d$$

$$a, b, c, d \in \Re$$

$$a \neq 0$$

Finding Roots of a Cubic

$$f = \frac{1}{3}\left(\frac{3c}{a} - \frac{b^2}{a^2}\right) \qquad g = \frac{1}{27}\left(\frac{2b^3}{a^3} - \frac{9bc}{a^2} + \frac{27d}{a}\right)$$

$$h = \frac{g^2}{4} + \frac{f^3}{27} \qquad j = \sqrt{\frac{g^2}{4} - h}$$

$$k = \sqrt[3]{j} \qquad l = \arcsin\left(-\frac{g}{2j}\right)$$

$$m = -k \qquad n = \cos\left(\frac{l}{3}\right)$$

$$p = \sqrt{3}\sin\left(\frac{l}{3}\right) \qquad q = -\frac{b}{3a}$$

$$x_1 = 2k\cos\left(\frac{l}{3}\right) - \left(\frac{b}{3a}\right) \qquad x_2 = m(n+p)+q$$

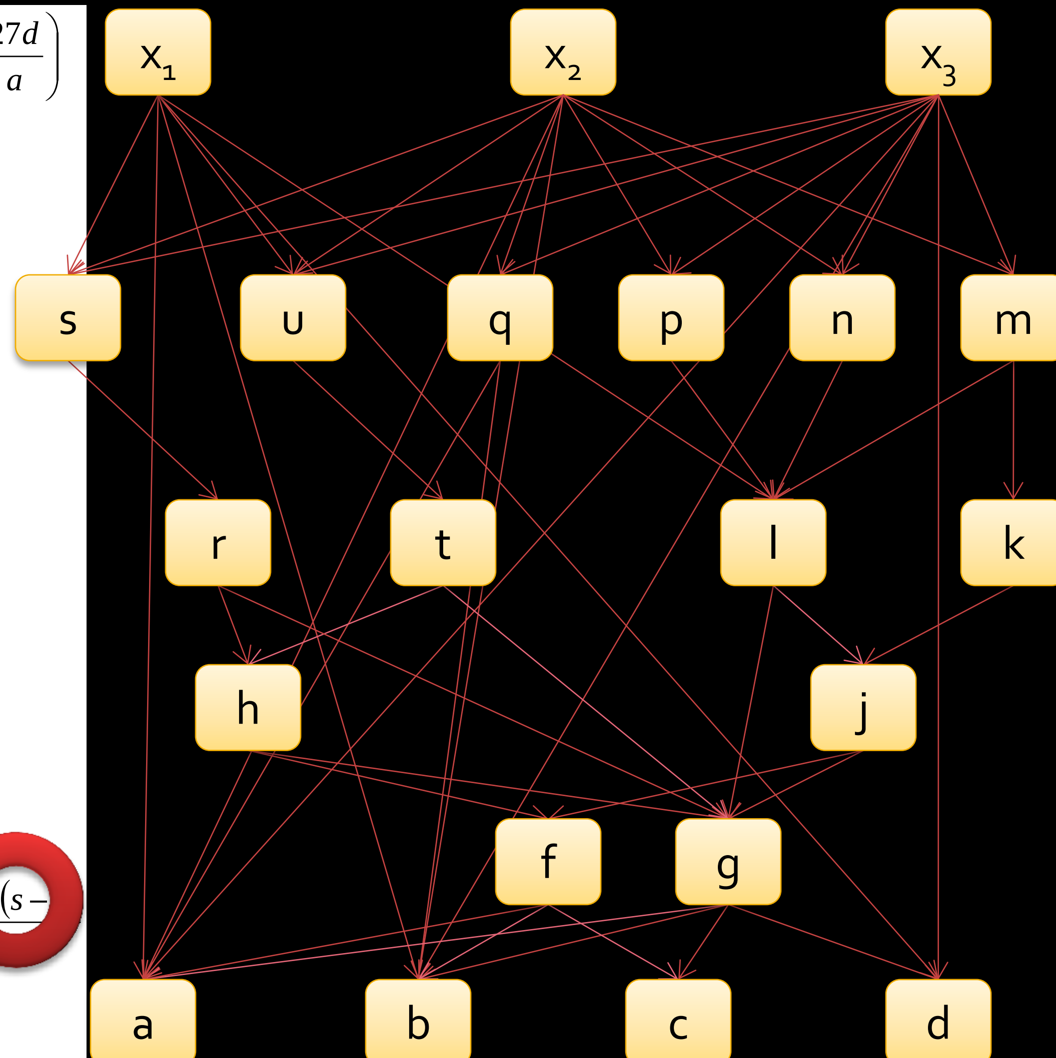
$$x_3 = m(n-p)+q \qquad r = -\frac{g}{2} + \sqrt{h}$$

$$s = \qquad t = -\frac{g}{2} - \sqrt{h}$$

$$u = \sqrt[3]{t} \qquad = s + u - \frac{b}{3a}$$

$$x_2 = -\frac{s+u}{2} - \frac{b}{3a} + \frac{\sqrt{3}i(s-}{2} \qquad x_3 = \quad \frac{s+u}{2} \quad \frac{}{3a} \quad 3i(s-$$

$$x_1 = x_2 = x_3 = -\sqrt[3]{\frac{d}{a}}$$



```python
import math

class Cubic:
  def __init__(self, a, b, c, d):
    self.a = a
    self.b = b
    self.c = c
    self.d = d

  def getRoots(self):
    f = (3*self.c/self.a - self.b**2/self.a**2)/3

    g = (2*self.b**3/self.a**3 -    \
         9*self.b*self.c/self.a**2 \
         + 27*self.d/self.a)/27

    h = g**2/4 + f**3/27

    if f == 0 and g == 0 and h == 0:
      x1, x2, x3 = [-(self.d/self.a)**(1/3)]*3

    elif h > 0:
      r = -(g/2) + h**0.5
      if r < 0:
        s = -abs(r)**(1/3)
      else:
        s = r**(1/3)
      t = -(g/2) - h**0.5
      if t < 0:
        u = -abs(t)**(1/3)
      else:
        u = t**(1/3)
      x1 = (s+u)-(self.b/(3*self.a))
      x2 = complex(-(s+u)/2 - (self.b/(3*self.a))

      x3 = complex(-(s+u)/2 - (self.b/(3*self.a)

    else:
      i = ((g**2/4)-h)**0.5
      j = i**(1/3)
      k = math.acos(-(g/(2*i)))
      m = math.cos(k/3)
      n = math.sqrt(3) * math.sin(k/3)
      p = -(self.b/(3*self.a))
      x1 = 2*j*math.cos(k/3)-(self.b/(3*self.a))
      x2 = -j*(m+n)+p
      x3 = -j*(m-n)+p

    return x1, x2, x3

def printRoots(cubic):
  for index, root in enumerate(cubic.getRoots()):
    print "x{0}: {1}".format(index+1, root))
printRoots(Cubic(2,  -4, -22, 24))
printRoots(Cubic(3, -10,  14, 27))
printRoots(Cubic(1,   6,  12,  8))
```

Test

Read

Debug

```
import math

class Cubic:
  def __init__(self, a, b, c, d):
    self.a = a
    self.b = b
    self.c = c
    self.d = d

  def getRoots(self):
    f = (3*self.c/self.a - self.b**2/self.a**2)/3

    g = (2*self.b**3/self.a**3 -    \
         9*self.b*self.c/self.a**2 \
         + 27*self.d/self.a)/27

    h = g**2/4 + f**3/27

    if f == 0 and g == 0 and h == 0:
      x1, x2, x3 = [-(self.d/self.a)**(1/3)]*3

    elif h > 0:
      r = -(g/2) + h**0.5
      if r < 0:
        s = -abs(r)**(1/3)
      else:
        s = r**(1/3)
      t = -(g/2) - h**0.5
      if t < 0:
        u = -abs(t)**(1/3)
      else:
        u = t**(1/3)
      x1 = (s+u)-(self.b/(3*self.a))
      x2 = complex(-(s+u)/2 - (self.b/(3*self.a)),
                   ((s-u)*3**0.5)/2)
      x3 = complex(-(s+u)/2 - (self.b/(3*self.a)),
                   -((s-u)*3**0.5)/2)

    else:
      i = ((g**2/4)-h)**0.5
      j = i**(1/3)
      k = math.acos(-(g/(2*i)))
      m = math.cos(k/3)
      n = math.sqrt(3) * math.sin(k/3)
      p = -(self.b/(3*self.a))
      x1 = 2*j*math.cos(k/3)-(self.b/(3*self.a))
      x2 = -j*(m+n)+p
      x3 = -j*(m-n)+p

    return x1, x2, x3

def printRoots(cubic):
  for index, root in enumerate(cubic.getRoots()):
    print "x{0}: {1}".format(index+1, root))
printRoots(Cubic(2,  -4, -22, 24))
printRoots(Cubic(3, -10,  14, 27))
printRoots(Cubic(1,   6,  12,  8))
```

Can we do better?
(of course)

```python
import math

class Cubic:
  def __init__(self, a, b, c, d):
    self.a = a
    self.b = b
    self.c = c
    self.d = d

  def getRoots(self):
    f = (3*self.c/self.a - self.b**2/self.a**2)/3

    g = (2*self.b**3/self.a**3 -    \
         9*self.b*self.c/self.a**2 \
         + 27*self.d/self.a)/27
```

self.calc_h()

```python
    if f == 0 and g == 0 and h == 0:
      x1, x2, x3 = [-(self.d/self.a)**(1/3)]*3

    elif h > 0:
      r = -(g/2) + h**0.5
      if r < 0:
        s = -abs(r)**(1/3)
      else:
        s = r**(1/3)
      t = -(g/2) - h**0.5
      if t < 0:
        u = -abs(t)**(1/3)
      else:
        u = t**(1/3)
      x1 = (s+u)-(self.b/(3*self.a))
      x2 = complex(-(s+u)/2 - (self.b/(3*self.a)),
                   ((s-u)*3**0.5)/2)
      x3 = complex(-(s+u)/2 - (self.b/(3*self.a)),
                   -((s-u)*3**0.5)/2)
    else:
      i = ((g**2/4)-h)**0.5
      j = i**(1/3)
      k = math.acos(-(g/(2*i)))
      m = math.cos(k/3)
      n = math.sqrt(3) * math.sin(k/3)
      p = -(self.b/(3*self.a))
      x1  = 2*j*math.cos(k/3)-(self.b/(3*self.a))
      x2  = -j*(m+n)+p
      x3  = -j*(m-n)+p

    return x1, x2, x3


def printRoots(cubic):
  for index, root in enumerate(cubic.getRoots()):
    print "x{0}: {1}".format(index+1, root))
printRoots(Cubic(2,  -4, -22, 24))
printRo ... 10,  14, 27))
pri ...
```

```python
def calc_h(self):
  self.h = g**2/4 + f**3/27
```

Calculation Methods

Calculation Methods

```
import math

class Cubic:
  def __init__(self, a, b, c, d):
    self.a = a
    self.b = b
    self.c = c
    self.d = d

  def _f(self):
    self.f = (3*self.c/self.a - self.b**2/self.a**2)/3

  def _g(self):
    self.g = (2*self.b**3/self.a**3 - 9*self.b*self.c/self.a**2 + 27*self.d/self.a)/27

  def _h(self):
    self.h = self.g**2/4 + self.f**3/27

  def _r(self):
    self.r = -(self.g/2) + self.h**0.5

  def _s(self):
    self.s = -abs(self.r)**(1/3) if self.r < 0 else self.r**(1/3)

  def _t(self):
    self.t = -(self.g/2) - self.h**0.5

  def _u(self):
    self.u = -abs(self.t)**(1/3) if self.t < 0 else self.t**(1/3)

  def _i(self):
    self.i = ((self.g**2/4)-self.h)**0.5

  def _j(self):
    self.j = self.i**(1/3)

  (ect...)
```

_ + h = _h

```python
def getRoots(self):
  f = (3*self.c/self.a - self.b**2/self.a**2)/3

  g = (2*self.b**3/self.a**3 -    \
       9*self.b*self.c/self.a**2 \
       + 27*self.d/self.a)/27

  h = g**2/4 + f**3/27

  if f == 0 and g == 0 and h == 0:
    x1, x2, x3 = [-(self.d/self.a)**(1/3)]*3

  elif h > 0:
    r = -(g/2) + h**0.5
    if r < 0:
      s = -abs(r)**(1/3)
    else:
      s = r**(1/3)
    t = -(g/2) - h**0.5
    if t < 0:
      u = -abs(t)**(1/3)
    else:
      u = t**(1/3)
    x1 = (s+u)-(self.b/(3*self.a))
    x2 = complex(-(s+u)/2 - (self.b/(3*self.a)),
                 ((s-u)*3**0.5)/2)
    x3 = complex(-(s+u)/2 - (self.b/(3*self.a)),
                 -((s-u)*3**0.5)/2)

  else:
    i = ((g**2/4)-h)**0.5
    j = i**(1/3)
    k = math.acos(-(g/(2*i)))
    m = math.cos(k/3)
    n = math.sqrt(3) * math.sin(k/3)
    p = -(self.b/(3*self.a))
    x1 = 2*j*math.cos(k/3)-(self.b/(3*self.a))
    x2 = -j*(m+n)+p
    x3 = -j*(m-n)+p

  return x1, x2, x3
```

Remove Calculations

CALL CALCULATION METHODS

```python
def getRoots(self):
  self._f()
  self._g()
  self._h()

  if self.f == 0 and self.g == 0 and sel
    x1, x2, x3 = [-(self.d/self.a)**(1/3)]*3

  elif self.h > 0:
    self._r()
    self._s()
    self._t()
    self._u()
    x1 = (self.s+self.u)-(self.b/(3*self.a))
    x2 = complex(-(self.s+self.u)/2 - (self.b/(3*self.a)),
                 ((self.s-self.u)*3**0.5)/2)
    x3 = complex(-(self.s+self.u)/2 - (self.b/(3*self.a)),
                 -((self.s-self.u)*3**0.5)/2)
  else:
    self._i()
    self._j()
    self._k()
    self._m()
    self._n()
    self._p()
    x1 = 2*self.j*math.cos(self.k/3)-(self.b/(3*self.a))
    x2 = -self.j*(self.m+self.n)+self.p
    x3 = -self.j*(self.m-self.n)+self.p

  return x1, x2, x3
```

Explicit Calls

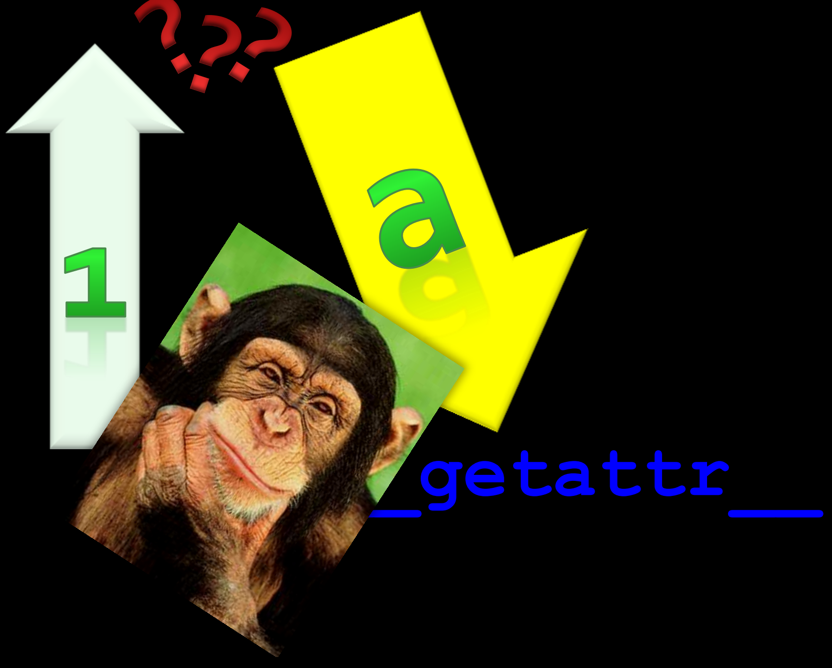
Correct Ordering

CALCULATION ON DEMAND

Class Customization: __getattr__

- Special method
  - __getattr__
- Called when variable not found
- Passed unfound variable name
- Returns an appropriate value
- Raises AttributeError

```
def myMethod(self):
    return self.a
```



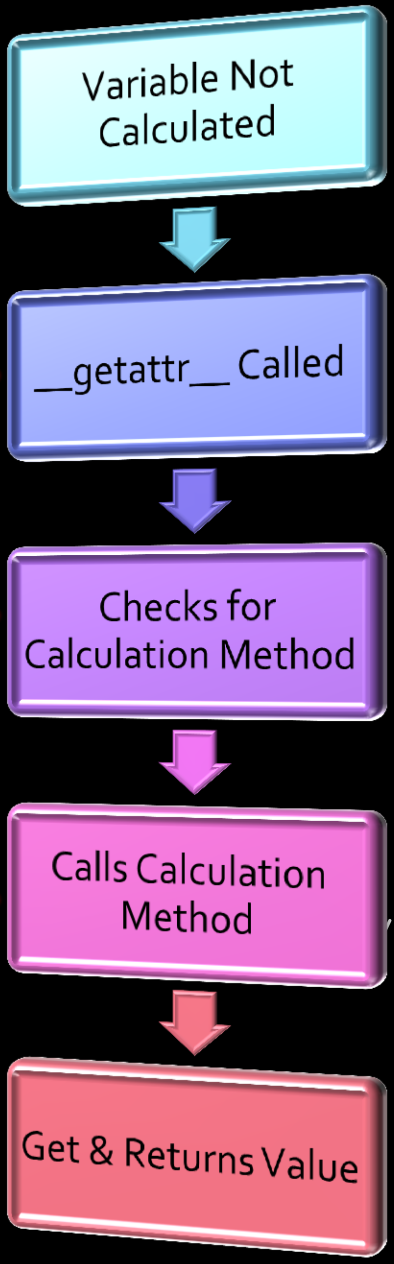
```python
def getRoots(self):
  self._f()
  self._g()
  self._h()

  if self.f == 0 and self.g == 0 and self.h == 0:
    x1, x2, x3 = [-(self.d/self.a)**(1/3)]*3

  elif self.h > 0:
    self._r()
    self._s()
    self._t()
    self._u()
    x1 = (self.s+self.u)-(self.b/(3*self.a))
    x2 = complex(-(self.s+self.u)/2 - (self.b/(3*self.a)),
                 ((self.s-self.u)*3**0.5)/2)
    x3 = complex(-(self.s+self.u)/2 - (self.b/(3*self.a)),
                 -((self.s-self.u)*3**0.5)/2)
  else:
    self._i()
    self._j()
    self._k()
    self._m()
    self._n()
    self._p()
    x1 = 2*self.j*math.cos(self.k/3)-(self.b/(3*self.a))
    x2 = -self.j*(self.m+self.n)+self.p
    x3 = -self.j*(self.m-self.n)+self.p

  return x1, x2, x3
```

```python
def getRoots(self):
  if self.f == 0 and self.g == 0 and self.h == 0:
    x1, x2, x3 = [-(self.d/self.a)**(1/3)]*3

  elif self.h > 0:
    x1 = (self.s+self.u)-(self.b/(3*self.a))
    x2 = complex(-(self.s+self.u)/2 - (self.b/(3*self.a)),
                 ((self.s-self.u)*3**0.5)/2)
    x3 = complex(-(self.s+self.u)/2 - (self.b/(3*self.a)),
                 -((self.s-self.u)*3**0.5)/2)

  else:
    x1 = 2*self.j*math.cos(self.k/3)-(self.b/(3*self.a))
    x2 = -self.j*(self.m+self.n)+self.p
    x3 = -self.j*(self.m-self.n)+self.p

  return x1, x2, x3
```

CLOSER TO MATH

CLEARER

```python
import math

class Cubic:
  def __init__(self, a, b, c, d):
    self.a = a
    self.b = b
    self.c = c
    self.d = d

  def __getattr__(self, name):
    calcName = "_" + name

    if hasattr(self, calcName):
      getattr(self, calcName)()
      return getattr(self, name)
    else:
      raise AttributeError

  def _f(self):
    self.f = (3*self.c/self.a - self.b**2/self.a**2)/3

  def _g(self):
    self.g = (2*self.b**3/self.a**3 - 9*self.b*self.c/self.a**2 /27

  def _h(self):
    self.h = self.g**2/4 + self.f**3/27

  def _r(self):
    self.r = -(self.g/2) + self.h**0.5

  def _s(self):
    self.s = -abs(self.r)**(1/3) if self.r < 0 else self.r**(1/3)
```



Output

```
__getattr__: f
__getattr__: h
__getattr__: g
__getattr__: j
__getattr__: i
__getattr__: k
__getattr__: m
__getattr__: n
__getattr__: p
x1: 4.0
x2: -3.0
x3: 1.0

__getattr__: f
__getattr__: h
__getattr__: g
__getattr__: s
__getattr__: r
__getattr__: u
__getattr__: t
x1: -1.0
x2: (2.16666666667+2.07498326633j)
x3: (2.16666666667-2.07498326633j)

__getattr__: f
__getattr__: g
__getattr__: h
x1: -2.0
x2: -2.0
x3: -2.0
```

How about 4^{th} degree polynomials?

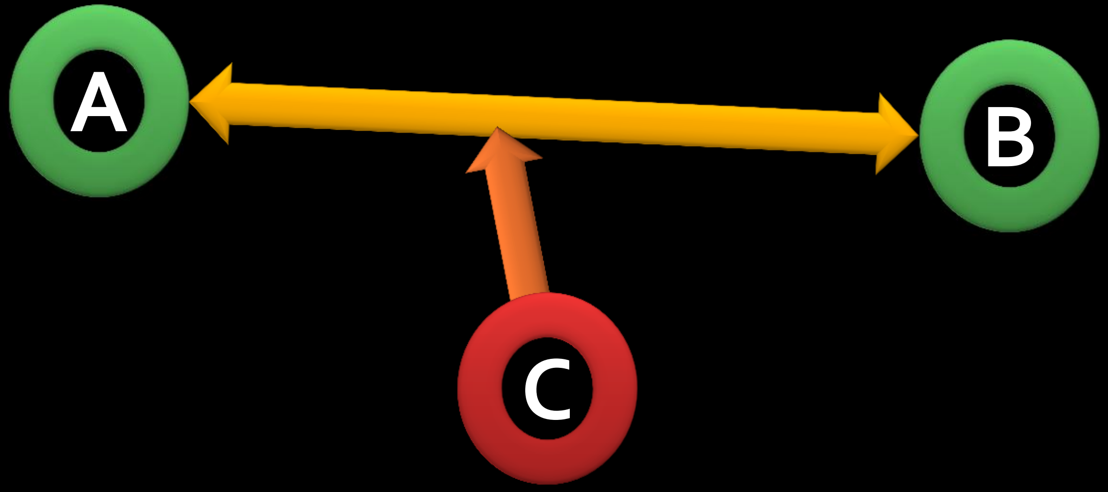
YES!

How about 5^{th} degree polynomials?

No!
Galois Theory

What's next?

- Elliptic Curves Cryptosystems
  - Encrypt and Decrypt messages through a secure channel



Summary

- Discrete calculation methods
  - Improved testability
  - Clearer code
- Class customization
- Implemented calculation on demand
- Do not worry evaluation order
  - Faster, more reliable design & implementation
- Easier code modification
  - Calculation order automatically changes

Credit, where credit is due

- Sutton, Peter. *"Advanced Python, Better Code"*. The University of Manchester, 2009.

Questions?

- Slides: https://speakerdeck.com/paiva
- Code: https://github.com/paiva/cubic
- YouTube: https://www.youtube.com/watch?v=4EcrtSRrYF8
- Say 'Hi' on Twitter: @Stronnics
- I want your feedback!