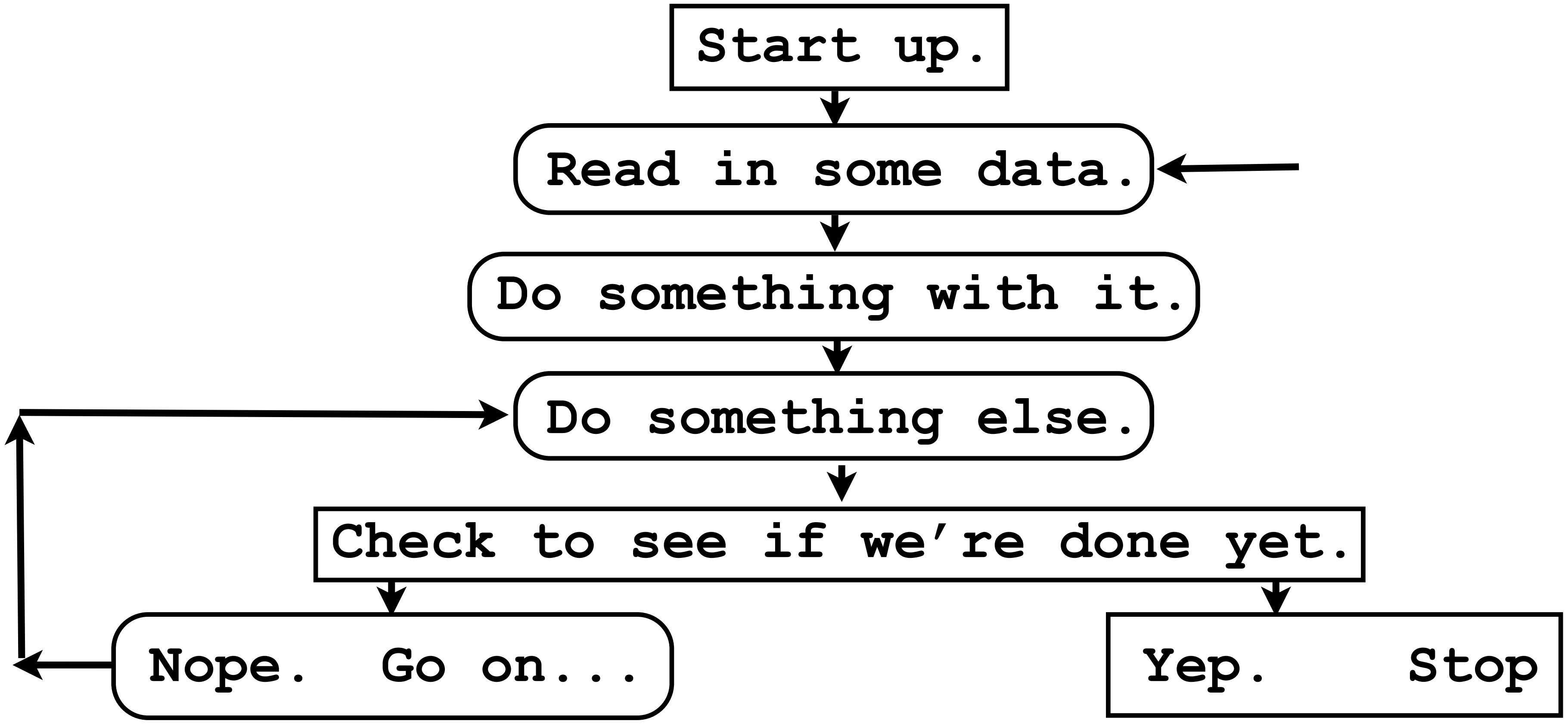
Allez ***OOP***!

a cartoon version of

Object-***O***riented ***P***rogramming

cartoon version of linear programming (*e.g.* FORTRAN)



Start up.

Read in some data.

Do something with it.

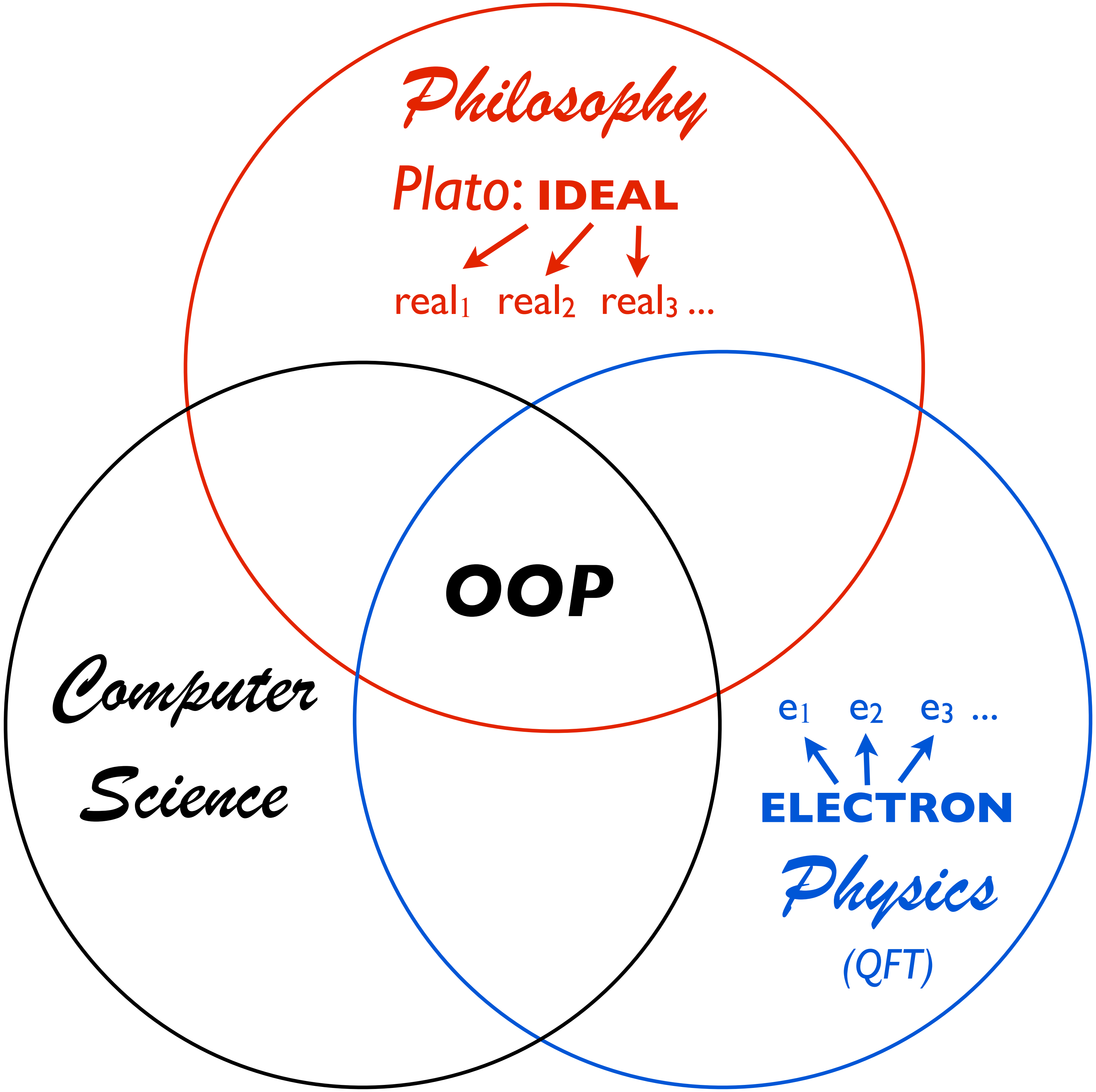
Do something else.

Check to see if we're done yet.

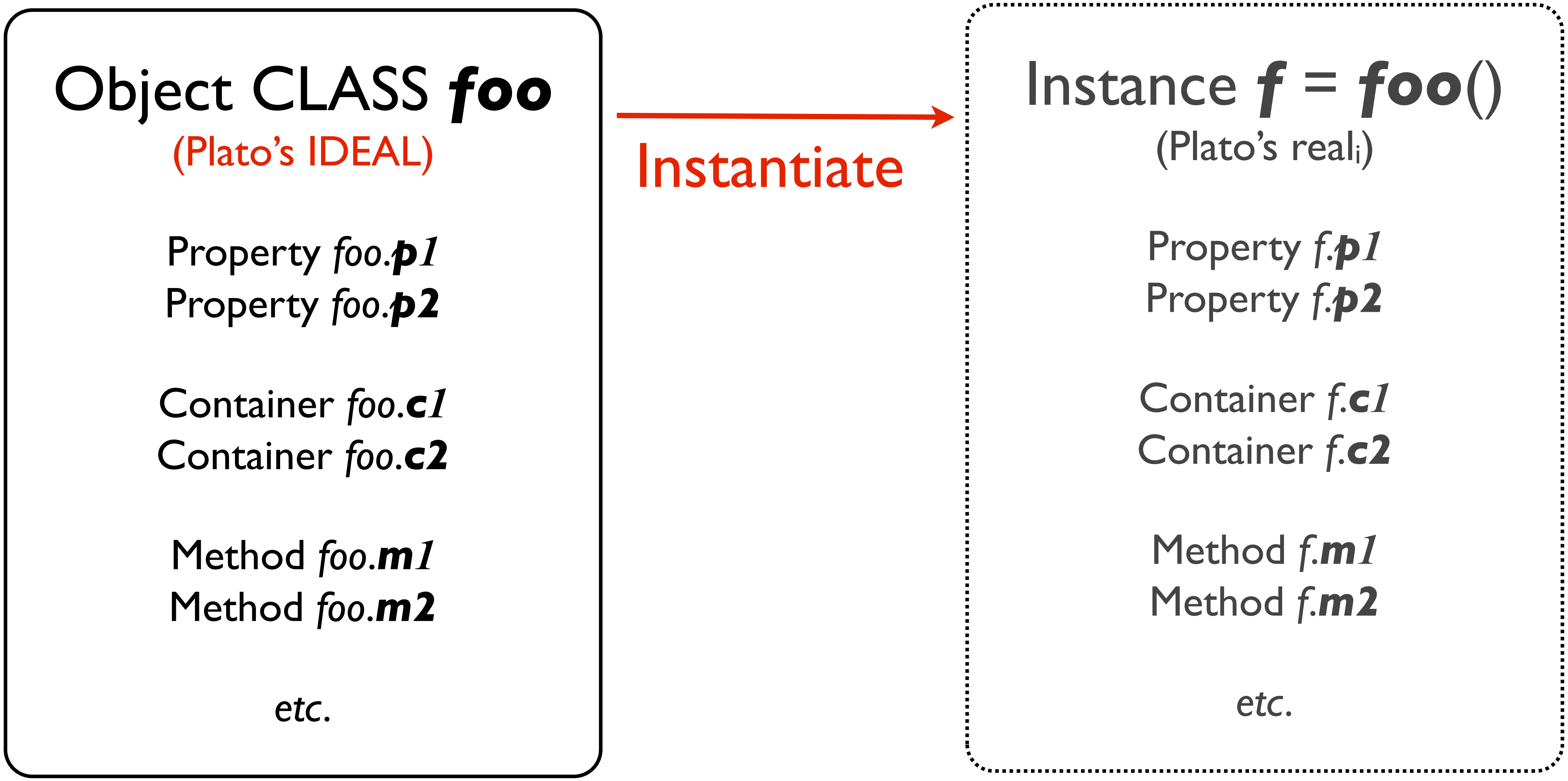
Nope. Go on...

Yep. Stop

Object-**O**riented ***P***rogramming:
another Instance of the Platonic Ideal?



cartoon version of **OO programming** (e.g. python)



Object CLASS ***foo***

(Plato's IDEAL)

Property *foo*.***p1***
Property *foo*.***p2***

Container *foo*.***c1***
Container *foo*.***c2***

Method *foo*.***m1***
Method *foo*.***m2***

etc.

Instantiate

Instance ***f*** = ***foo***()

(Plato's real$_i$)

Property *f*.***p1***
Property *f*.***p2***

Container *f*.***c1***
Container *f*.***c2***

Method *f*.***m1***
Method *f*.***m2***

etc.

+ Communication between Instances, . . .

There are many ***O***bject-***O***riented ***P***rogramming languages.

Some examples are:

Java (the Queen of ***OOP***)

PHP (surprise!)

C++ (?)

Python (this week's lesson)

. . .

One thing to be keenly aware of:

You can *build your own* Classes, but usually you are ***instantiating*** extremely sophisticated Classes developed by others!

When you steal from one author, it's plagiarism; if you steal from many, it's research. - Wilson Mizner

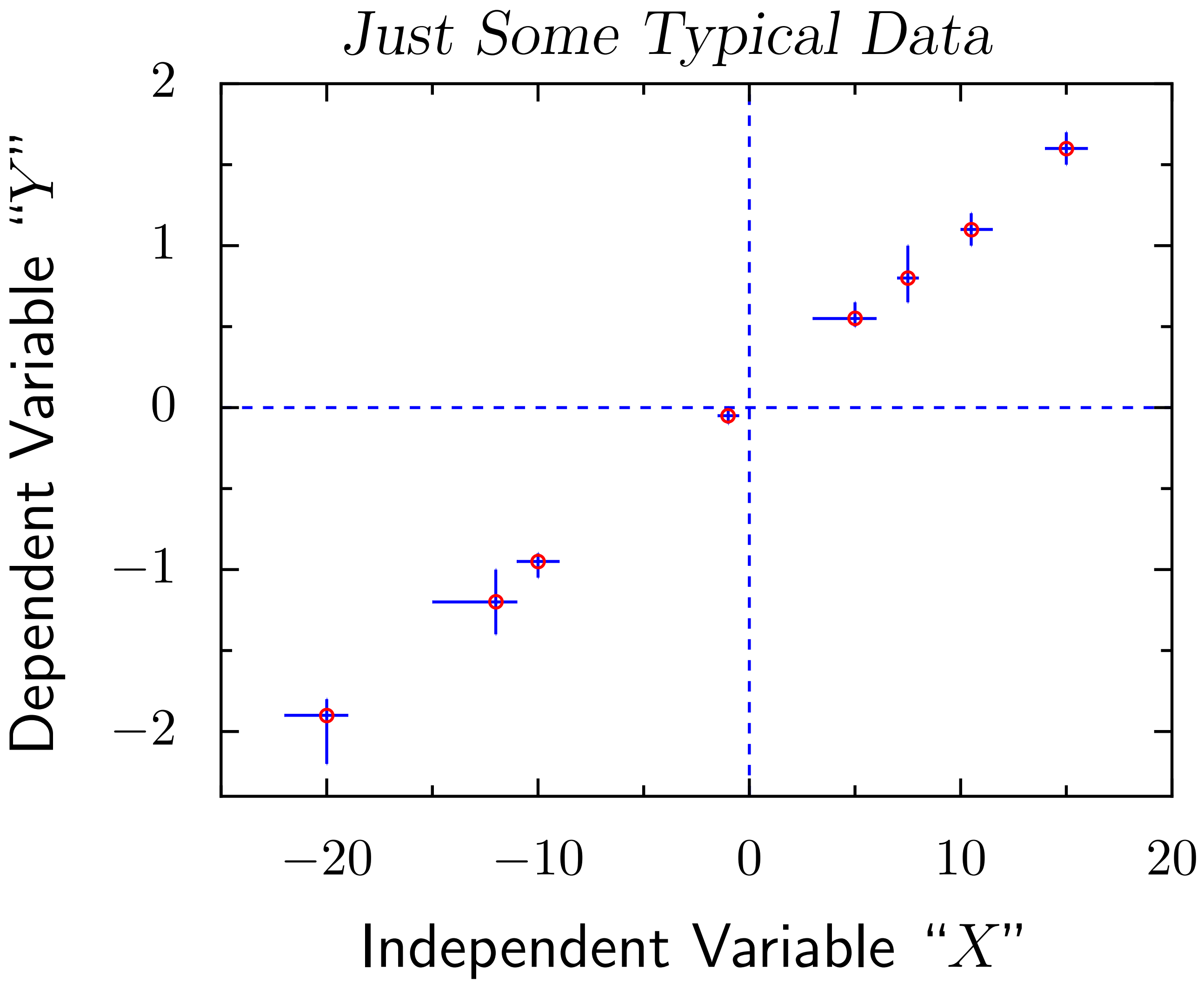
```
#! /usr/bin/env  python

from pyx import *
import sys

# The command argument is the data file name:
print sys.argv[1]

# Use LaTeX to make title & axis labels:
text.set( mode="latex" )

# Instantiate graphxy object with size, limits and labels:
h = 0.8*6     # (height of plot box [in inches?])
w = 0.8*8    # (width of plot box [in inches?])
xmin = -25.0 # Note: without decimal points, these are treated as integers!
xmax = 20.0
ymin = -2.4
ymax = 2.0
g = graph.graphxy( width=w, height=h,
            x=graph.axis.linear( min=xmin, max=xmax,
                          title="{\large\sf Independent Variable ``$X$"}" ),
            y=graph.axis.linear( min=ymin, max=ymax,
                           title="{\large\sf Dependent Variable ``$Y$"}" )
            )

. . .
```

• • •

```
# Plot the data directly from the input file:
g.plot( graph.data.file( sys.argv[1],
                         x=2, dxmax=3, dxmin=4, y=5, dymax=6, dymin=7 ),
       styles=[graph.style.errorbar( size=0, errorbarattrs=[color.rgb.blue] ),
               graph.style.symbol( graph.style.symbol.circle,size=0.075,
                                   symbolattrs=[color.rgb.red] )] )

# Include the graph title, aesthetically located:
g.text( g.width/8, g.height+0.2, "{\large\sl Just Some Typical Data}" )

# Plot the zero-axes as "strokes" in screen coordinates:
xrng = xmax-xmin
x0pos = g.width*abs(xmin)/xrng
yrng = ymax-ymin
y0pos = g.height*abs(ymin/yrng)
g.stroke( path.line( x0pos, 0, x0pos, g.height ),
         [style.linestyle.dashed, color.rgb.blue] )
g.stroke( path.line( 0, y0pos, g.width, y0pos ),
         [style.linestyle.dashed, color.rgb.blue] )
```

• • •

```
#Write an eps file:
g.writeEPSfile("data-pyx.eps")
#Write a pdf file:
g.writePDFfile("data-pyx.pdf")

print( " " )
print( "------------------------------------------------------------" )
print( "NOTE: PyX is designed for writing eps or pdf files!" )
print( "You will not see the plot on your screen unless" )
print( "you # gv data-pyx.eps or # xpdf data-pyx.pdf later." )
print( "------------------------------------------------------------" )
print( " " )
```


Just Some Typical Data
Dependent Variable "Y"
Independent Variable "X"