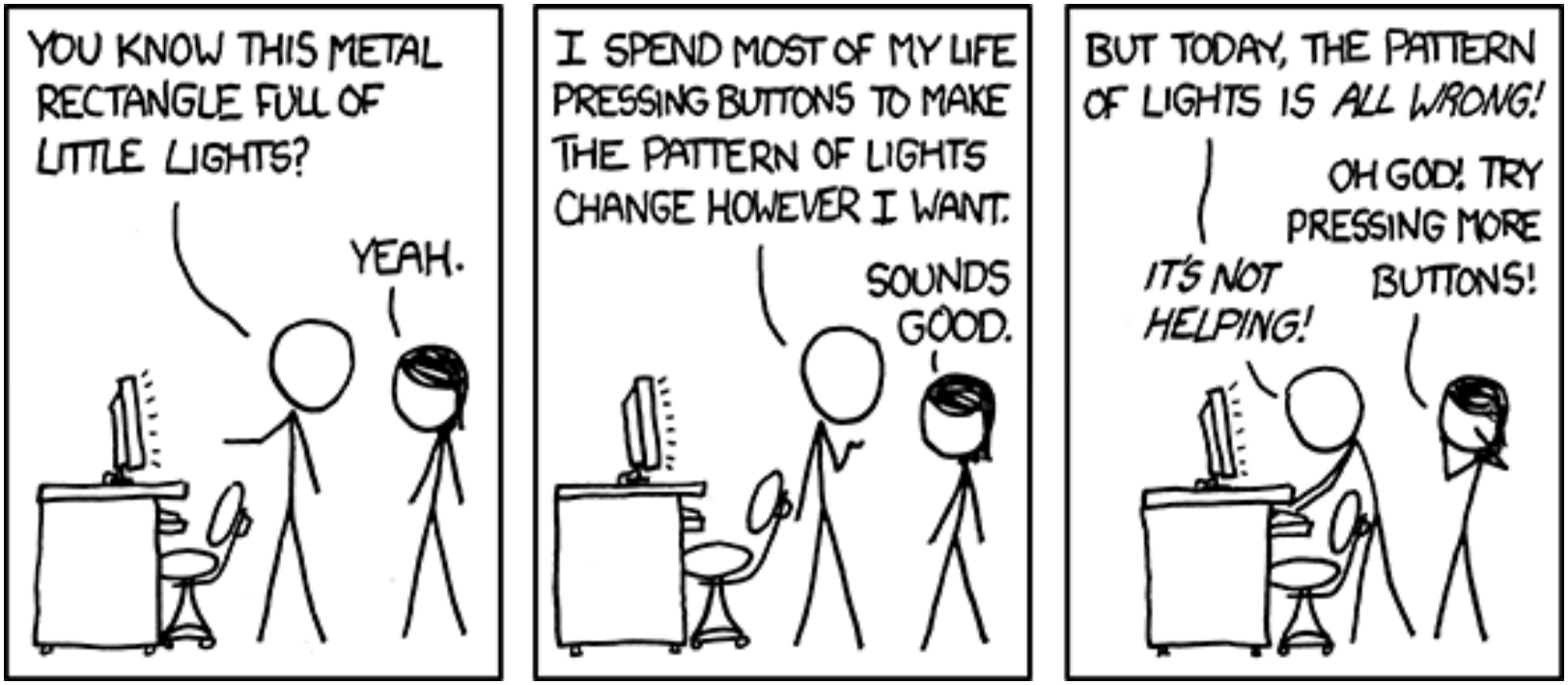
CS3110: Data Structures & Functional Programming

Lambda Calculus & Map-Reduce


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

Logistics

- First Quiz in class Friday
- No class Monday
- I'm behind on creating homeworks and the project. I will try to get the project out tomorrow to give you the weekend.

Homework 1 Solutions

- Hello World – have to reload in order to observe changes when using interactive python
- Everyone did fine.

Homework 2 Solutions

Little more variation on answers.

```
def FunSum(numbers):
    return int(bool(numbers)) and (numbers[0] +
                                   FunSum(numbers[1:]))
```

Note:

bool([]) == bool (()) == False

Style Guide

- Comments!!
- Whitespace!!
- Variable Use!!

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.collections import EllipseCollection


def f (t):
   s1 = np.sin(2*np.pi*t); e1 = np.exp(-t)
   temp= np.absolute((s1*e1))+.05
   return temp
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.collections import EllipseColle

"""
f
Takes:
     t - a float
Returns:
    a float = |sin(2\pi t) * exp(t)| + 0.05
Helper function to Comparison
"""
def f(t):
   s1 = np.sin(2*np.pi*t)
   e1 = np.exp(-t)
   return np.absolute((s1*e1))+.05
```

```
def Comparison () :
  fig=plt.figure()

  t = np.arange(0.0, 5.0, 0.1);  s = f(t);
  nse = np.random.normal(0.0, 0.3, t.shape) * s


  plt.plot(s+nse, t, 'b^')
  plt.hlines(t, [0], s, lw=2)
  plt.xlabel('time (s)'); plt.xlim(xmin=0);
  plt.title('Comparison of model with data')

  plt.show()
```

```
“””
 Displays a Figure of Data Normally distributed arou
the function f(t)
“””
def Comparison():
    plt.figure()

    t = np.arange(0.0, 5.0, 0.1)
    s = f(t)
    nse = np.random.normal(0.0, 0.3, t.shape) * s


    plt.plot(s+nse, t, 'b^')
    plt.hlines(t, [0], s, lw=2)

    plt.xlabel('time (s)')
    plt.title('Comparison of model with data')

    plt.xlim(xmin=0)

    plt.show()
```

```
def Ellipses ( ):
  fig=plt.figure()
  x = np.arange(10);    y = np.arange(15)
  X, Y = np.meshgrid(x, y)
  XY = np.hstack((X.ravel()[:,np.newaxis], Y.ravel()[:,np.newaxis]))
  ww = X/10.0;  hh = Y/15.0;  aa = X*9
```

```
# Displays a Figure of Ellipses:
# whose shape is controlled by their (x,y) centers
def Ellipses():
    plt.figure()

    x = np.arange(10)
    y = np.arange(15)
    X, Y = np.meshgrid(x, y)
    XY = np.hstack((X.ravel()[:,np.newaxis], Y.ravel()[:,np.newaxis]))

    ww = X/10.0
    hh = Y/15.0
    aa = X*9
```

```
ax = plt.subplot(1,1,1)
ec = EllipseCollection( ww, hh, aa, units='x',
                                    offsets=XY, transOffset=ax.transData)
ec.set_array((X+Y).ravel())
ax.add_collection(ec)
ax.autoscale_view()
ax.set_xlabel('X');  ax.set_ylabel('y')
cbar = plt.colorbar(ec)
cbar.set_label('X+Y')
plt.show()
```

```
ec = EllipseCollection(  ww,
                         hh,
                         aa,
                         units='x',
                         offsets=XY,
                         transOffset=ax.transData)

ec.set_array((X+Y).ravel())

ax = plt.subplot(1,1,1)
ax.add_collection(ec)
ax.autoscale_view()

ax.set_xlabel('X')
ax.set_ylabel('y')

cbar = plt.colorbar(ec)
cbar.set_label('X+Y')

plt.show()
```

(Redo) Substitution Evaluation

- In Left to Right manner, evaluate and replace expressions with their bound value.
- Values are terms that need no further simplification to be understood.

Ex. Evaluating :

```
If cond1:
    fun1
else:
    fun2
```

Python

1. reduces (cond1) to true or false (also in the left -> right manner as we've seen in the boolean expressions)

2. Evaluates fun1 or fun2 accordingly, recursively in a left->right manner

Bound v. Unbound

```
x = 1

def f(x):
    return x+2

y = 2*x
```

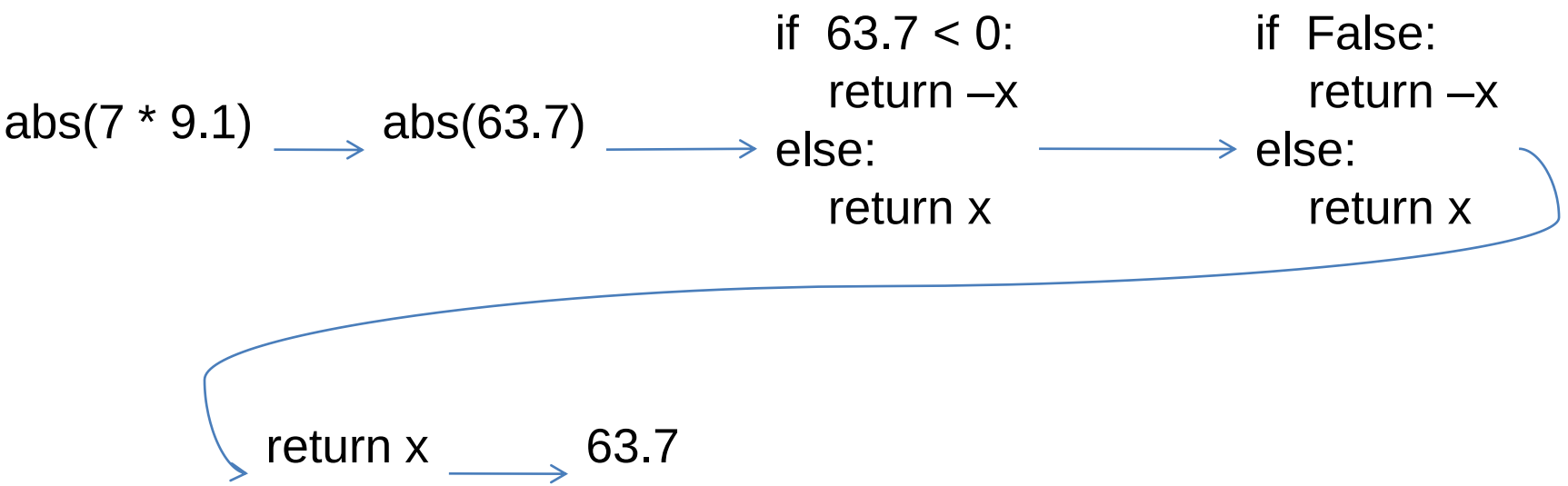
->

```
x = 1

def f(x):
    return x+2

y = 2*(1) = 2
```

Note: x is not Bound with f(x) and hence does not have its value substituted

Simple Substitution Ex.

```
def abs(x):
    if x < 0:
        return –x
    else:
         return x
```



Normal v Application – order evaluation

- So far we have been dealing with the Python model of evaluation. Python *eagerly* evaluates, as soon as it can reduce an expression it does. This left-right *eager* evaluation is also known as *applicative-order evaluation.*
- *Normal-order evaluation*: (delayed as long as possible, Haskell, Scala)

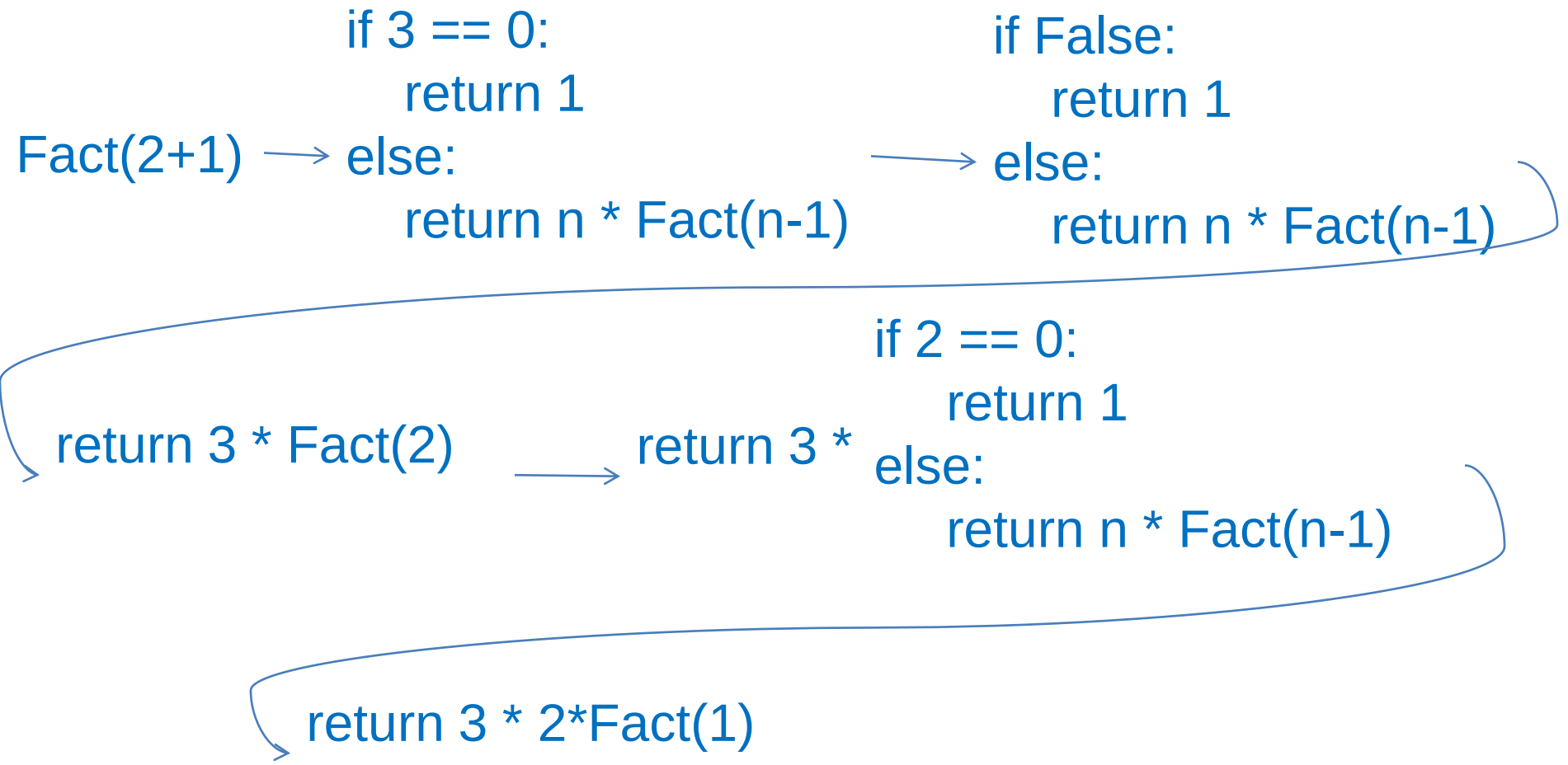
(2+3)^4 = (2+3) * (2+ 3) *(2+3) * (2+ 3)

= (5) * (2+ 3) *(2+3) * (2+ 3)

= (5) *(5) *(2+3) * (2+ 3)

Recursive Substitution (*Application-order evaluation*)

```
def Fact(n):
    if n == 0:
        return 1
    else:
        return n * Fact(n-1)
```



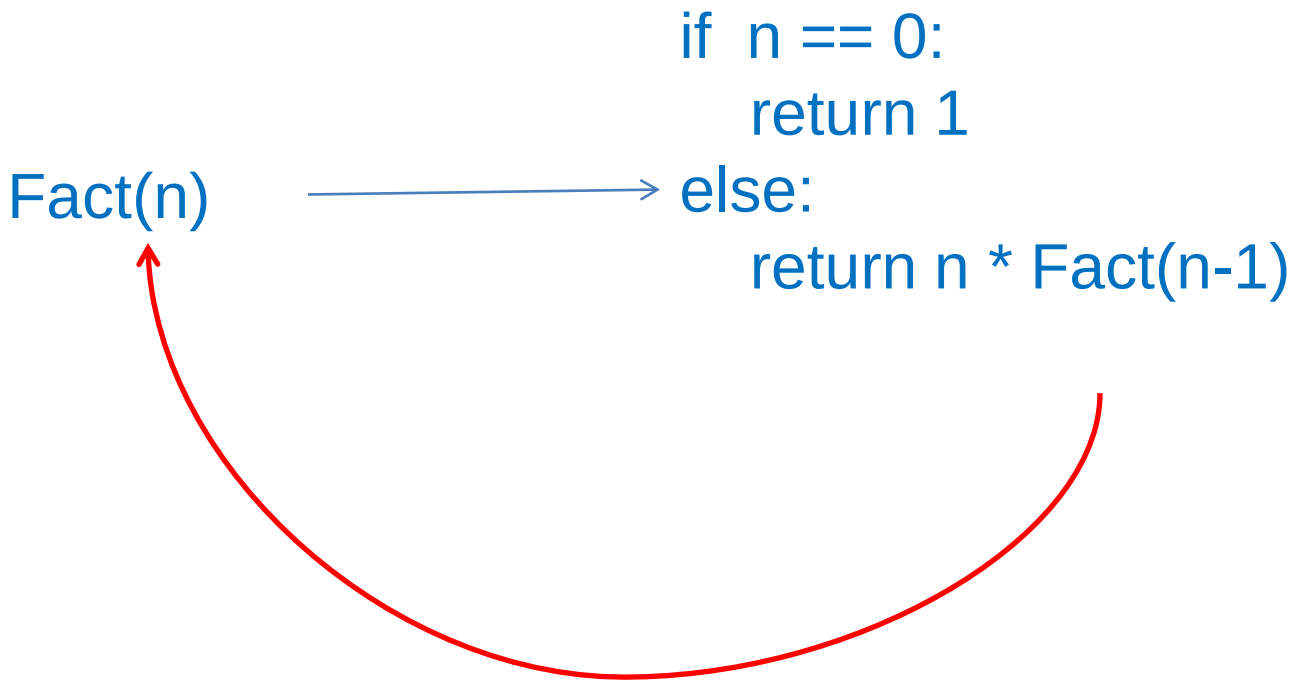
Recursive Substitution (*Normal-order evaluation*)

Thinking in the same way quickly gets messy.

```
Fact(2+1) ⟶ if 2+1 == 0:
                return 1
            else:
                return 2+1 * Fact(2+1-1)
```

```
if 2+1 == 0:
    return 1
else:
    return 2+1 * if 2+1-1 == 0:
                     return 1
                 else:
                     return 2+1-1 * Fact(2+1-1-1)
```

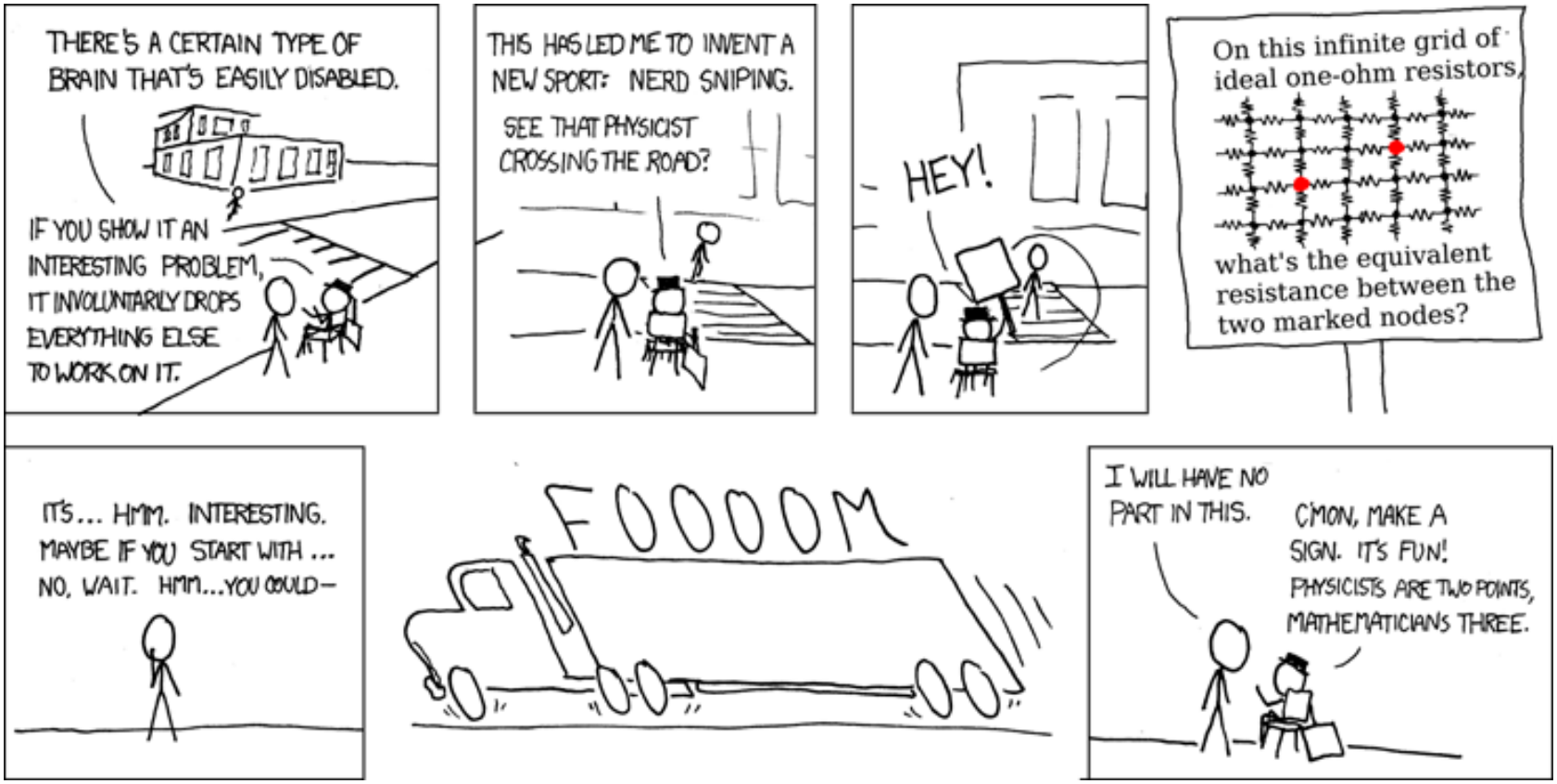
Better way of short cutting recursive substitution modeling in Normal-order evaluations. (what's cute here is the idea.)



Where this line represents the recursive substitution.

In programs with no side effects Normal-order and Application-order evaluation models produce the same output. However, as the order of evaluation is different, programs with side effects may produce different outputs.

- Now that we understand evaluation models, let's look at some functional programming staples.



Lambda Function Syntax

From p.78:

- Use a *lambda* function, wherever you would a normal function
- *Lambda* automatically returns an expression
- It's form is:

    lambda *paramenters*: *expression*

Ex. Lambda

```
a_list = range(9)
low = 3
high = 7
filtered_list = filter(lambda x, bottom=low, top=high:
                                    top>x>bottom,
                                    a_list)
```

filter returns a list of the elements of a list that satisfy the passed function. In this case, lambda.

Note: filter has no side effects.

- Pros:

1. cute (better organized code)
2. handy to replace once-called functions (happens many times with calling functions that require another function, filter, sort, etc)

- Cons:

1. in Python can only use 1-line lambdas
2. sometimes harder to read

Clever Use of lambda

- Use lambda to have a function return a function!

```
def SetParameters(para_1, para_2, para_3):

    part_1 = f(para_1, para_2, para_3)  # something involved

    return lambda x: x * part_1 + x * g(para_1)
```

- Remember the substitution model?

What does Python do for us here?

Map Function Syntax

From p.164:

- Use a *map* function, wherever you want to iterate a function over an iterable
- No side effects
- Returns a list of the function iterated over the list
- It's form is:

  map(*function*, *list)*

Map or for?

- If you can use map, do. It guarantees no side effects.
- First introduction to thinking about parallel execution. (*board single v. multi-core execution*)
- Because map has no side effects, guarantees each operation can be done independently.
- Google reports using map-reduce for 130,000 machine-months of processing time.

Map Function Syntax

```
capped = map(lambda x: min(0, x), a_list)
```

Returns a list and does not modify a_list.