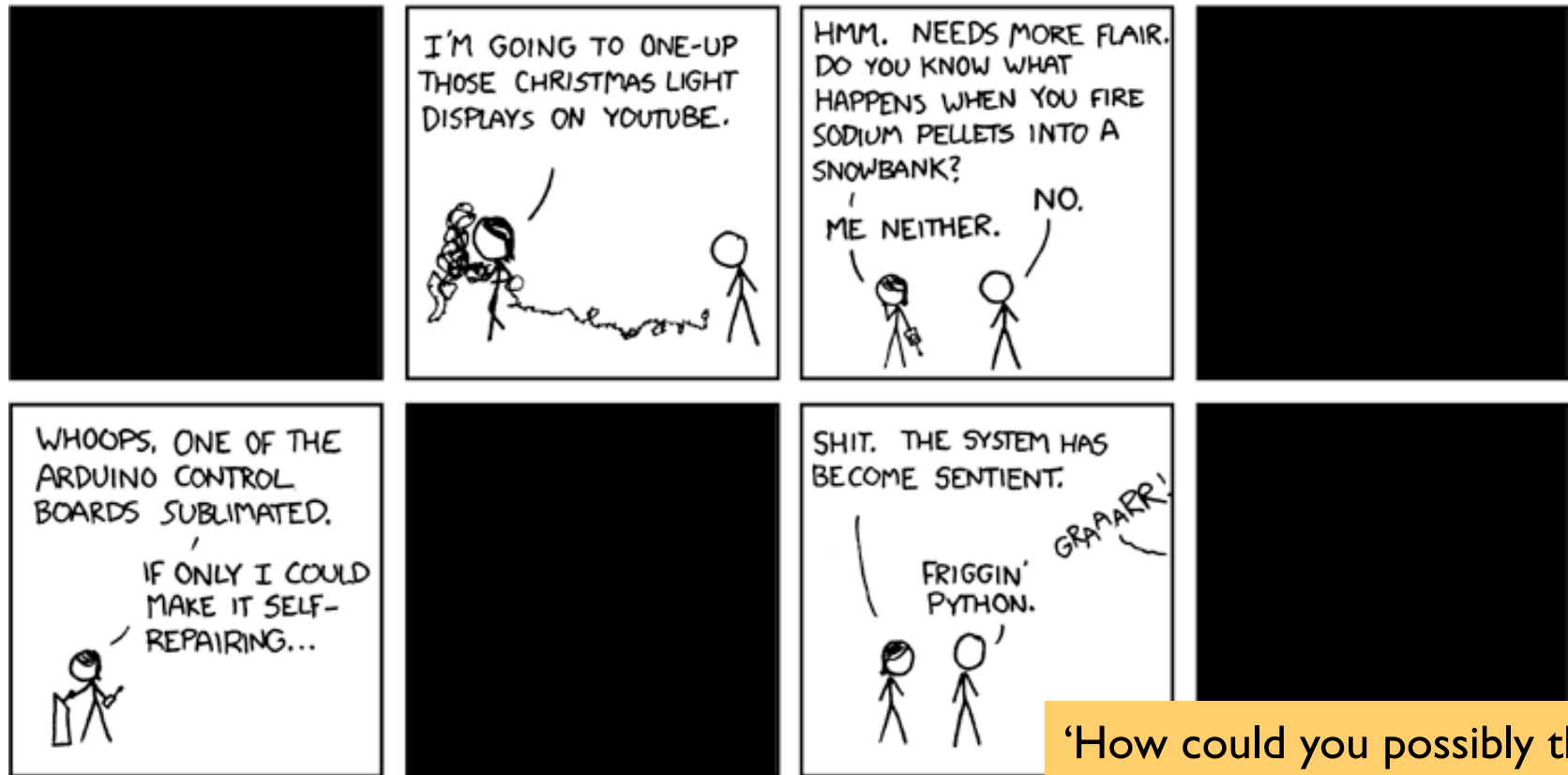


# Advanced Interactions

What if we could get the code to write itself?

DUE TO THE SLOWING ECONOMY, WE COULD ONLY AFFORD TO PRODUCE THE PRIME-NUMBERED PANELS. YOU SHOULD BE ABLE TO INFER THE MISSING PARTS OF THE STORY EASILY ENOUGH. WE APOLOGIZE FOR THE INCONVENIENCE.



'How could you possibly think typing 'import skynet' was a good idea?'

That's So Meta!

# Lambda Functions

(anonymous functions)

*from Lisp & functional programming*

Allow functions to be defined without an (explicit) identifier:

```
>>> tmp = lambda x: x**2
>>> type(tmp)
<type 'function'>
>>> tmp(2)
4

>>> (lambda x,y: x**2+y)(2,4.5) # forget about creating a new function name...just do it!
8.5

>>> ## create a list of lambda functions
>>> lamfun = [lambda x: x**2, lambda x: x**3, \
              lambda y: math.sqrt(y) if y >= 0 else "Really? I mean really? %f" % y]
>>> for i in lamfun: print i(-1.3)
1.69
-2.197
Really? I mean really? -1.300000
```

Practically speaking...

lambda functions are meant to be short one liners. If you need more complex functions, it's probably better just to name them.

# Sorting Lists

```
# airline, number, heading to, gate, time (decimal hours) ~
flights = [("Southwest",145,"DCA",1,6.00),("United",31,"IAD",1,7.1),("United",302,"LHR",5,6.5),\
          ("Aeroflot",34,"SVO",5,9.00),("Southwest",146,"CDA",1,9.60), ("United",46,"LAX",5,6.5),\
          ("Southwest",23,"SBA",6,12.5),("United",2,"LAX",10,12.5),("Southwest",59,"LAX",11,14.5),\
          ("American", 1,"JFK",12,11.3),("USAirways", 8,"MIA",20,13.1),("United",2032,"MIA",21,15.1),\
          ("SpamAir",1,"AUM",42,14.4)]~
```

```
>>> help(flights.sort)
      L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;
      cmp(x, y) -> -1, 0, 1
```

```
>>> #sort flights by time:
>>> flights.sort(key=lambda x: x[4])
>>> flights
```

```
[('Southwest', 145, 'DCA', 1, 6.0),
 ('United', 302, 'LHR', 5, 6.5),
 ('United', 46, 'LAX', 5, 6.5),
 ('United', 31, 'IAD', 1, 7.1),
 ('Aeroflot', 34, 'SVO', 5, 9.0),
 ('Southwest', 146, 'CDA', 1, 9.6),
 ('American', 1, 'JFK', 12, 11.3),
 ('Southwest', 23, 'SBA', 6, 12.5),
 ('United', 2, 'LAX', 10, 12.5),
 ('USAirways', 8, 'MIA', 20, 13.1),
 ('SpamAir', 1, 'AUM', 42, 14.4),
 ('Southwest', 59, 'LAX', 11, 14.5),
 ('United', 2032, 'MIA', 21, 15.1)]
```

# Multiple column sorting

`operator.itemgetter(item[, args...])`¶

Return a callable object that fetches *item* from its operand using the operand's `__getitem__()` method. If multiple items are specified, returns a tuple of lookup values.

<http://docs.python.org/library/operator.html#module-operator>

```
>>> #sort flights by time, then number, then
airline:
>>> import operator
>>> flights.sort(key=operator.itemgetter(4,1,0))
>>> flights
[('Southwest', 145, 'DCA', 1, 6.0),
 ('United', 46, 'LAX', 5, 6.5),
 ('United', 302, 'LHR', 5, 6.5),
 ('United', 31, 'IAD', 1, 7.1),
 ('Aeroflot', 34, 'SVO', 5, 9.0),
 ('Southwest', 146, 'CDA', 1, 9.6),
 ('American', 1, 'JFK', 12, 11.3),
 ('United', 2, 'LAX', 10, 12.5),
 ('Southwest', 23, 'SBA', 6, 12.5),
 ('USAirways', 8, 'MIA', 20, 13.1),
 ('SpamAir', 1, 'AUM', 42, 14.4),
 ('Southwest', 59, 'LAX', 11, 14.5),
 ('United', 2032, 'MIA', 21, 15.1)]
```

# Multiple column sorting

`operator.itemgetter(item[, args...])`

Return a callable object that fetches *item* from its operand using the operand's `__getitem__()` method. If multiple items are specified, returns a tuple of lookup values.

<http://docs.python.org/library/operator.html#module-operator>

```
>>> #sort flights by time, then number, then  
airline:
```

```
>>> import operator
```

```
>>> flights.sort(key=operator.itemgetter(4,1,0))
```

```
>>> flights
```

```
[('Southwest', 145, 'DCA', 1, 6.0),  
 ('United', 46, 'LAX', 5, 6.5),  
 ('United', 302, 'LHR', 5, 6.5),  
 ('United', 31, 'IAD', 1, 7.1),  
 ('Aeroflot', 34, 'SVO', 5, 9.0),  
 ('Southwest', 146, 'CDA', 1, 9.6),  
 ('American', 1, 'JFK', 12, 11.3),  
 ('United', 2, 'LAX', 10, 12.5),  
 ('Southwest', 23, 'SBA', 6, 12.5),  
 ('USAirways', 8, 'MIA', 20, 13.1),  
 ('SpamAir', 1, 'AUM', 42, 14.4),  
 ('Southwest', 59, 'LAX', 11, 14.5),  
 ('United', 2032, 'MIA', 21, 15.1)]
```

```
>>> #sort flights by time:
```

```
>>> flights.sort(key=lambda x: x[4])
```

```
>>> flights
```

```
[('Southwest', 145, 'DCA', 1, 6.0),  
 ('United', 302, 'LHR', 5, 6.5),  
 ('United', 46, 'LAX', 5, 6.5),  
 ('United', 31, 'IAD', 1, 7.1),  
 ('Aeroflot', 34, 'SVO', 5, 9.0),  
 ('Southwest', 146, 'CDA', 1, 9.6),  
 ('American', 1, 'JFK', 12, 11.3),  
 ('Southwest', 23, 'SBA', 6, 12.5),  
 ('United', 2, 'LAX', 10, 12.5),  
 ('USAirways', 8, 'MIA', 20, 13.1),  
 ('SpamAir', 1, 'AUM', 42, 14.4),  
 ('Southwest', 59, 'LAX', 11, 14.5),  
 ('United', 2032, 'MIA', 21, 15.1)]
```

# filter, map, reduce, zip



## Filter is a certain way to do list comprehension

`filter(function, sequence)` returns a sequence consisting of those items from the sequence for which `function(item)` is true.

```
>>>#Create a list for which numbers between 0 and 100 are even and divisible by 11

>>> mylist=[num for num in xrange(101) if (num % 2 == 0.0) and (num % 11 == 0.0)]
>>> print mylist
[0, 22, 44, 66, 88]

>>> def f(num): return (num % 2 == 0.0) and (num % 11 == 0.0)
>>> mylist = filter(f,xrange(101))
[0, 22, 44, 66, 88]
```

if the input is a string, so is the output...

```
>>> # also works on strings...try it with lambdas!
>>> a="Charlie Brown said \"!@!@$@!\"" ; a
'Charlie Brown said "!@!@$@!"'

>>> # Get just the alphabetical characters:
>>> import string
>>> filter(lambda c: c in string.ascii_letters,a)
'CharlieBrownsaid'
```

# Iterable xrange()

`xrange()` is an iterable version of `range()`:

- `range(10)` creates a 10-element list,
- `xrange(10)` creates an iterable object which returns 0 the first time it's called, 1 the next time, etc.

## Computational advantage?

*Time how long it takes with the ipython magic `%timeit`:*

```
>>> def f(num): return (num % 2 == 0.0) and (num % 11 == 0.0)
>>> %timeit len(filter(f,range(1L)))
100000 loops, best of 3: 5.63 us per loop
>>> %timeit len(filter(f,xrange(1L)))
100000 loops, best of 3: 4.94 us per loop

>>> # try more
>>> %timeit len(filter(f,range(10000000L)))
1 loops, best of 3: 5.2 s per loop
>>> %timeit len(filter(f,xrange(10000000L)))
1 loops, best of 3: 4.33 s per loop
```

**Note:** `xrange` (like `range`) can be reversed

```
>>> for i in reversed(xrange(1,10,2)): print i,
9 7 5 3 1
```

# filter, map, reduce, zip



## Map is just another way to do list comprehension

`map(function, sequence)` calls `function(item)` for each of the sequence's items and returns a list of the return values

```
>>> def cube_it(x): return x**3
>>> map(cube_it, xrange(1,10))
[1, 8, 27, 64, 125, 216, 343, 512, 729]

>>> map(lambda x: x**3, xrange(1,10))
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

## Reduce returns one value

`reduce(function, sequence)` returns a single value constructed by calling the binary function `function` on the first two items of the sequence, then on the result and the next item, and so on:

```
>>> reduce(lambda x,y: x + y, xrange(1,11))    # sum from 1 to 10
55
>>> %timeit reduce(lambda x,y: x + y, xrange(1,11))
100000 loops, best of 3: 2.18 us per loop
>>> %timeit sum(xrange(1,11))                  # sum() is a built in function...it's bound to be faster
1000000 loops, best of 3: 655 ns per loop
```



# filter, map, reduce, zip



## zip()

built in function to pairwise concatenate items in iterables into a list of tuples

```
>>> zip(["I","you","them"],["=spam","=eggs","=dark knights"])
[('I', '=spam'), ('you', '=eggs'), ('them', '=dark knights')]
>>> zip(["I","you","them"],["=spam","=eggs","=dark knights"],["!","?","#"])
[('I', '=spam', '!'), ('you', '=eggs', '?'), ('them', '=dark knights', '#')]
>>> zip(["I","you","them"],["=spam","=eggs","=dark knights"],["!","?"])
[('I', '=spam', '!'), ('you', '=eggs', '?')]
```

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print 'What is your %s? It is %s.' % (q, a)
What is your name? It is lancelot.
What is your quest? It is the holy grail.
What is your favorite color? It is blue.
```

*not to be confused with [zipfile](#) module which exposes file compression*

# Try/Except/Finally

Billy: Let's keep going with "Airplanes", for \$200.

Bobby Wheat: "Airplanes" for \$200: "And what is the Deal With the Black Box?" [ Tommy buzzes in ] Tommy!

Tommy: It's the only thing that survives the crash - why don't they build the whole plane out of the Black Box!



<http://snltranscripts.jt.org/91/91rstandup.phtml>

# Wrap volatile code in try/except/finally

```
>>> tmp = raw_input("Enter a number and I'll square it: ")
>>> print float(tmp)**2
Enter a number and I'll square it: monty
ValueError: invalid literal for float(): monty
```

Python error  
generates an  
“exception”

instead....

**try:**

```
tmp = raw_input("Enter a number " + \
                and I'll square it: ")
print float(tmp)**2
```

**except:**

```
print "dude. I asked you for a number and " + \
"%s is not a number." % tmp
```

**finally:**

```
print "thanks for playing!"
```

volatile stuff!

regardless of whether you hit an  
error, execute everything inside the  
finally block

upon error, jump  
here inside  
except and  
execute that  
code

# Wrap volatile code in try/except/finally

```
>>> tmp = raw_input("Enter a number and I'll square it: ")
>>> print float(tmp)**2
Enter a number and I'll square it: monty
ValueError: invalid literal for float(): monty
```

instead....

```
>>> def f():
    try:
        tmp = raw_input("Enter a number and I'll square it: ")
        print float(tmp)**2
    except:
        print "dude. I asked you for a number and %s is not a number." % tmp
    finally:
        print "thanks for playing!"

>>> f()
Enter a number and I'll square it: 3
9.0
thanks for playing!

>>> f()
Enter a number and I'll square it: monty
dude. I asked you for a number and monty is not a number.
thanks for playing!
```

# exec & eval

**exec** executes strings as if they were Python code

```
>>> a = "print 'checkit' "  
>>> exec a  
Checkit
```

```
>>> a = "x = 4.56"  
>>> exec a  
>>> print x  
4.56
```

```
>>> exec "del x"  
>>> print x
```

```
-----  
Traceback (most recent call last):  File "<ipython console>", line 1, in  
<module>NameError: name 'x' is not defined
```

- dynamically create Python code (!)
- execute that code w/ implication for current namespace

# exec & eval

```
>>> import math
>>> while True:
    bi = raw_input("what built in function would you like me to coopt? ")
    nn = raw_input("what new name would you like to give it? ")
    exec "%s = %s" % (nn,bi)
what built in function would you like me to coopt? math.sin
what new name would you like to give it? monty_sin
what built in function would you like me to coopt? Range
what new name would you like to give it? python_range

>>> monty_sin (math.pi/2)
1.0

>>> python_range(3)
[0, 1, 2]
```

# exec & eval

`eval` evaluates strings as Python expressions

```
>>> x = eval('5') ; x
5
>>> x = eval('%d + 6' % x) ; x
11
>>> x = eval('abs(%d)' % -100) ; x
100
>>> x = eval('print 5')
# INVALID: print is a statement, not an expression (in Python 2.x).
-----
File "<string>", line 1
    print 5
    ^SyntaxError: invalid syntax
>>> exec "print 5"
5

>>> x = eval('if 1: x = 4')
# INVALID: if is a statement, not an expression.
-----
File "<string>", line 1
    if 1: x = 4
    ^SyntaxError: invalid syntax

>>> exec 'if True: x=4' ; x
4
```

# Breakout

Write a code which generates python code that approximates the function  $x^2 + x$  in the range  $[-3, 3]$  using the given vocabulary.

## hints:

Randomly generate lambda functions using a restricted vocabulary:

```
vocab = [ "x" , "x" , " " , "+" , "-" , "*" , "/" , "1" , "2" , "3" ]
```

Evaluate these lambda functions at a fixed number of x values and save the difference between those answers and  $x^2 + x$ .

Catch errors!



```
import random
import numpy

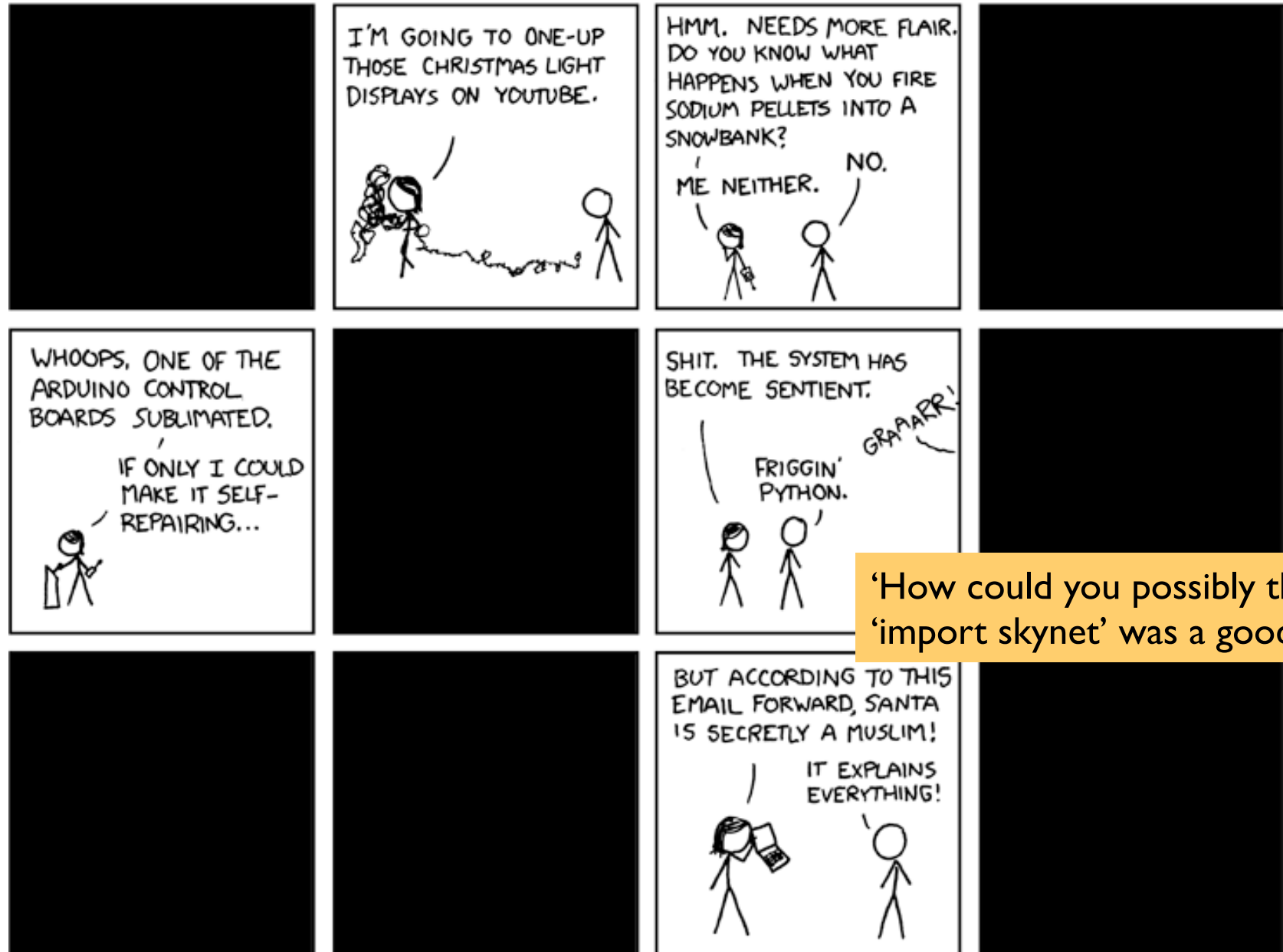
vocab = ["x", "x", "", "+", "-", "*", "/", "1", "2", "3"]

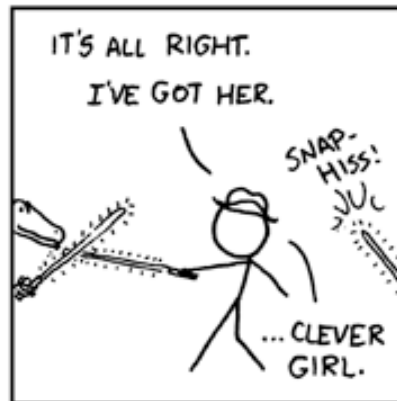
max_try = 1000000
max_chars = 10 #max number of characters to generate
x_array = numpy.arange(-3, 3, 0.4)
real_function = x_array**2 + x_array

tries = []
for loop...
```

# THE 2008 XKCD CHRISTMAS SPECIAL

DUE TO THE SLOWING ECONOMY, WE COULD ONLY AFFORD TO PRODUCE THE PRIME-NUMBERED PANELS. YOU SHOULD BE ABLE TO INFER THE MISSING PARTS OF THE STORY EASILY ENOUGH. WE APOLOGIZE FOR THE INCONVENIENCE.







skynet