Threading is not a model

Joe Gregorio
Developer Relations, Google Wave
My Opinion
I want to annoy you.
A short story, a book, design patterns, and Djikstra
"The Short Happy Life of the Brown Oxford"

Philip K. Dick
The Principle of Sufficient Irritation in action

Determining the radioactive irritant is left as an exercise for the reader.
A short story, a book, design patterns, and Djikstra
The Design of Everyday Things

The book
The path

A short story, a book, design patterns, and Djikstra
Let's talk about Design Patterns

I did not say that patterns are bad.
I did say that using them may be a sign of weakness in a language.
Python isn't Java without the compile

Design Patterns in Dynamic Programming – Peter Norvig
Beyond Java – Bruce Tate
Not talking just about Python
Aren't patterns good?

Yes, but also a sign of weakness
There is a lack of patterns in Python

1. Define 'lack of patterns'
2. Demonstrate that lack
3. Explain why
Hard numbers

comp.lang.python

100,000+ messages
“factory method pattern” - 0
“abstract-factory pattern” - 0
“flyweight pattern” - 3
“state pattern” - 10
“strategy pattern” - 25
“flyweight” - 36
“visitor pattern” - 60
For your comparison

“dark matter” - 2
For your comparison

“dark matter” - 2

“the pope” - 16
For your comparison

“dark matter” - 2
“the pope” - 16
“sausage” - 66

Presuming there is no overlap among these messages
There is a lack of patterns in Python

1. Define 'lack of patterns'

2. Demonstrate that lack

3. Explain why
The patterns are built in.

No one talks about the 'structured programming' pattern or the 'object-oriented' pattern any more.
class Bisection (FindMinima):
    def algorithm(self,line):
        return (5.5,6.6)

class ConjugateGradient (FindMinima):
    def algorithm(self,line):
        return (3.3,4.4)

class MinimaSolver: # context class
    strategy=''
    def __init__(self,strategy):
        self.strategy=strategy
    def minima(self,line):
        return self.strategy.algorithm(line)
    def changeAlgorithm(self,newAlgorithm):
        self.strategy = newAlgorithm

solver=MinimaSolver(ConjugateGradient())
print solver.minima((5.5,5.5))
solver.changeAlgorithm(Bisection())
print solver.minima((5.5,5.5))
“When most of your code does nothing in a pompous way that is a sure sign that you are heading in the wrong direction. Here's a translation into python”

- Peter Otten
def bisection(line):
    return 5.5, 6.6

def conjugate_gradient(line):
    return 3.3, 4.4

solver = conjugate_gradient
print solver((5.5, 5.5))
solver = bisection
print solver((5.5, 5.5))
“This pattern is invisible in languages with first-class functions.”


What other language features are there, and what patterns do they make invisible?
Catalog of Language Features

- First-class functions
- Meta-programming
- Iterators
- Closures
In object-oriented programming, the Iterator pattern is a design pattern in which iterators are used to access the elements of an aggregate object sequentially without exposing its underlying representation.

http://en.wikipedia.org/wiki/Iterator_pattern

The definition of low-hanging fruit.
Iterators

for element in [1, 2, 3]:
    print element

definition of example

for element in (1, 2, 3):
    print element

definition of example

for key in {'one': 1, 'two': 2}:
    print key

definition of example

for char in "123":
    print char

definition of example

for line in open("myfile.txt"):
    print line

definition of example
There is a lack of patterns in Python

1. Define 'lack of patterns'
2. Demonstrate that lack
3. Explain why
A short story, a book, design patterns, and Djikstra
"Go to statement considered harmful"

Edsger W. Dijkstra, 1968

Letter to the editor, Communications of the ACM, Volume 11, Issue 3 (March 1968)
We are talking about Routines!
(or procedures, or functions, or methods) being controversial.

Along with 'if', 'while', and 'switch' statements
The controversy went on for a while

"GOTO Considered Harmful"
Considered Harmful

Frank Rubin, 1987

def hyp(x, y):
    return math.sqrt(x**2 + y**2)

>> hyp(3, 4)
5
What if Structured Programming wasn't built in?

You can do Structure Programming with our built in stack and 'call' primitives!
Patterns and Primitives

Pattern

Language Feature

Primitives

Model
Some Concurrency Patterns listed on Wikipedia

Lock
Monitor Object
Reactor
Thread pool
Thread-specific storage

These you will see on comp.lang.python
Some Concurrency Patterns listed on Wikipedia

Lock
Monitor Object
Reactor
Thread pool
Thread-specific storage

These you will see on comp.lang.python
Patterns and Primitives

Threadpool (Pattern)  Threads + queue + lock (Primitives)

Language Feature  Concurrency (Model)
“Just” use threads

Threading is not a model

Threading is a primitive, along with locks, transactional memory, etc.
What are the concurrency models?

1. Communicating Sequential Processes (CSP)

2. Actors

The difference is only in 'what' is concurrent
• Based on CSP by C.A.R. Hoare.
• An actual model for processes
• All code is written single threaded
• Communication via channels.
Sieve of Eratosthenes

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
</tr>
<tr>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
</tr>
<tr>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
</tr>
<tr>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
</tr>
<tr>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
<td>106</td>
<td>107</td>
<td>108</td>
<td>109</td>
</tr>
<tr>
<td>111</td>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
<td>118</td>
<td>119</td>
</tr>
</tbody>
</table>

Prime numbers
Sieve of Eratosthenes

N 2 3 5 7 11 13
import stackless

def generate(ch):
    for i in range(2, 1000):
        ch.send(i)

def pfilter(chin, chout, p):
    for i in chin:
        if i % p != 0:
            chout.send(i)

def primes(chin):
    while 1:
        prime = chin.receive()
        print prime
        chout = stackless.channel()
        stackless.tasklet(pfilter)(chin, chout, prime)
        chin = chout

c = stackless.channel()
stackless.tasklet(generate)(c)
stackless.tasklet(primes)(c)
stackless.run()
import stackless

def generate(ch):
    for i in range(2, 1000):
        ch.send(i)

def pfilter(chin, chout, p):
    for i in chin:
        if i % p != 0:
            chout.send(i)

def primes(chin):
    while 1:
        prime = chin.receive()
        print prime
        chout = stackless.channel()
        stackless.tasklet(pfilter)(chin, chout, prime)
        chin = chout

c = stackless.channel()
stackless.tasklet(generate)(c)
stackless.tasklet(primes)(c)
stackless.run()
func generate(ch chan int) {
    for i := 2; ; i++ { ch <- i } // Send 'i' to channel 'ch'.
}

func filter(in, out chan int, prime int) {
    for {
        i := <-in  // Receive 'i' from 'in'.
        if i % prime != 0 { out <- i } // Send 'i' to 'out'.
    }
}

func main() {
    ch := make(chan int)  // Create a new channel.
    go generate(ch)  // Start generate() as a goroutine.
    for {
        prime := <-ch
        fmt.Println(prime)
        ch1 := make(chan int)
        go filter(ch, ch1, prime)
        ch = ch1
    }
}
An implementation could use:

- Threads
- Locks
- Transactional Memory
Objects are concurrent
Objects send, and respond to messages
All code is written single threaded

Note that the 'channels' are implicit
Filter := Object clone
Filter init := method(p,
  self prime := p
  self next := nil
  self
)

Filter number := method(n,
  r := n % prime;
  if (r != 0,
    if (self next == nil,
      n println;
      next = self clone init(n)
    )
    next @number(n); yield
  )
)

Filter init(2)
for (i, 2, 1000,
  Filter number(i); yield
)
The path

A short story, a book, design patterns, and Djikstra
Further Reading

http://golang.org
http://www.iolanguage.com/
http://www.stackless.com/

Things not mentioned

- Futures
- Deterministic vs Non-Deterministic
- REST, MapReduce and other share-nothing architectures
My Goal

Every time you use a concurrency pattern you remember the lack of affordances, and it proves sufficiently irritating.

The short story, the book, and design patterns.