

# 1 RPN (Danny Yoo / Donovan Preston)

This code implements a reverse polish notation calculator. It's deliberately written in a slightly weird way to avoid being taken used as a homework answer.

```
1 """RPN calculator in Python.
3 This version of RPN is meant to be incredibly silly. It abuses
4 lambdas all around, uses global variables, and in general is just
5 really silly. No one in their right mind would do it this way.
6 *grin*
7 """
9 stack = []
11 def rpn(commands):
12     global stack
13     stack = []
14     for t in commands: t()
15     assert len(stack) == 1
16     return stack[0]
17
18 def makeOpCommand(op):
19     return lambda: stack.append(op(stack.pop(), stack.pop()))
21 def makeCommands(line):
22     ops = {'+' : makeOpCommand(lambda x, y: x + y),
23           '-' : makeOpCommand(lambda y, x: x - y),
24           '*' : makeOpCommand(lambda x, y: x * y),
25           '/' : makeOpCommand(lambda y, x: x / y) }
26     commands = []
27     for t in line.split():
28         # next line, using default parameters, is intentional.
29         commands.append(ops.get(t, lambda t=t: stack.append(float(t))))
30     return commands
31
32 if __name__ == '__main__':
33     print rpn(makeCommands('1 2.3 - 3 4 5 + * /'))
```

Donovan Preston later modified this code to be even more obscene. In his words,

This version uses no statements. The entire program is a single expression. It is very side-effecty, though.

```
1 globals().update(  
    dict(defs=lambda **args: globals().update(args))),  
3  
4 defs(  
5     sys=__import__('sys'),  
6     operator=__import__('operator')),  
7  
8 defs(  
9     swap=lambda op: lambda x, y: op(y, x),  
10    makeOp=lambda op: lambda stack: stack.append(op(stack.pop(),  
11 stack.pop()))),  
12  
13 defs(  
14    ops=dict([(char, makeOp(op)) for (char, op) in  
15              [( '+', operator.add), ('-', swap(operator.sub)),  
16               ('*', operator.mul), ('/', swap(operator.div)) ]]),  
17  
18 defs(  
19    popStack=lambda _, stack: stack.pop(0),  
20    rpn=lambda commands, stack=[]: popStack([t(stack) for t in  
21 commands], stack),  
22    makeCommands=lambda line: [  
23        ops.get(t, lambda stack, t=t: stack.append(float(t)))  
24        for t in line.split()],  
25  
26 operator.eq(__name__, '__main__') and sys.stdout.write(  
27    str(rpn(makeCommands('1 2.3 - 3 4 5 + * /')))+'\n')
```

## 2 XML Generator (Donovan Preston)

From Donovan Preston:

Here is my cool code submission. It's called the stan syntax, and it's a very easy way to generate xml from python. My web framework, Nevow, includes a more complete implementation of this idea.

```
1 """stan constructs xml from pure python
2 """
3
4
5 class Proto(str):
6     children = []
7     proto = property(lambda self: self)
8     def __call__(self, **kw):
9         return Tag(self)(**kw)
10
11     def __getitem__(self, *args):
12         return Tag(self)[args]
13
14
15 class Tag(object):
16     def __init__(self, proto):
17         self.proto = proto
18         self.attributes = {}
19         self.children = []
20
21     def __call__(self, **kw):
22         self.attributes.update(kw)
23         return self
24
25     def __getitem__(self, *args):
26         if not isinstance(args, (list, tuple)):
27             args = [args]
28         self.children.extend(args)
29         return self
30
31 def flatten(it, indent=0):
32     if isinstance(it, (Proto, Tag)):
33         attributes = ' '.join([
34             '='.join([key, '%s' % value]) for (key, value) in
35             it.attributes.items()])
36         if attributes:
37             attributes = ' ' + attributes
38         print ' ' * indent + "<%s%s>" % (it.proto, attributes)
39         [flatten(child, indent+1) for child in it.children]
40         print ' ' * indent + "</%s>" % it.proto
41     elif isinstance(it, (list, tuple)):
42         [flatten(child, indent) for child in it]
43     else:
44         print ' ' * indent, it
45
46
47 html, head, title, body, h1, div, a, img = (
```

```
49     Proto('html'), Proto('head'), Proto('title'), Proto('body'),
    Proto('h1'), Proto('div'), Proto('a'), Proto('img'))
51
53 document = html[
    head[
55         title["Hello"]],
    body[
57         h1["Goodbye"],
        div(style="color: red")["Red text!"],
59         a(href="http://google.com/")[
            img(
61                 title="Link to google",
                src="http://google.com/images/logo.gif"),
63                 "Google"]]]
65
66 if __name__ == '__main__':
67     flatten(document)
```

### 3 Python Metric Calculator (Reg Charney)

From Reg. Charney:

I should have a program for producing Python metrics. These measurements will include McCabe's Cyclomatic Complexity, Fan In/Fan Out, Lines of Comments, Lines of Code, number of doc strings, etc. The unique thing for this type of program will be the output format.

## 4 Cookbook Recipes (Jimmy Retzlaff)

```
1 class attrdict(dict):
2     """A dict whose items can also be accessed as member variables.
3
4     >>> d = attrdict(a=1, b=2)
5     >>> d['c'] = 3
6     >>> print d.a, d.b, d.c
7     1 2 3
8     >>> d.b = 10
9     >>> print d['b']
10    10
11
12    # but be careful, it's easy to hide methods
13    >>> print d.get('c')
14    3
15    >>> d['get'] = 4
16    >>> print d.get('a')
17    Traceback (most recent call last):
18    TypeError: 'int' object is not callable
19    """
20    def __init__(self, *args, **kwargs):
21        dict.__init__(self, *args, **kwargs)
22        self.__dict__ = self
```

```
1 import BaseHTTPServer
2 import webbrowser
3
4 def LoadInDefaultBrowser(html):
5     """Display html in the default web browser without creating a temp file.
6
7     Instantiates a trivial http server and calls webbrowser.open with a URL
8     to retrieve html from that server.
9     """
10
11    class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
12        def do_GET(self):
13            bufferSize = 1024*1024
14            for i in xrange(0, len(html), bufferSize):
15                self.wfile.write(html[i:i+bufferSize])
16
17    server = BaseHTTPServer.HTTPServer(('127.0.0.1', 0), RequestHandler)
18    webbrowser.open('http://127.0.0.1:%s' % server.server_port)
19    server.handle_request()
20 if __name__ == '__main__':
21     LoadInDefaultBrowser('<b>Hello World</b>')
```

```
1 def iterQueue(queue, sentinel):
2     """Iterate over the values in queue until sentinel is reached."""
3     while True:
4         value = queue.get()
5         if value != sentinel:
6             yield value
7         else:
8             return
```

## 5 Pipe-like Syntax (Maxim Krikun)

Maxim Krikun's Pipe-like syntax

```
1 from itertools import izip, imap, count, ifilter
  import re
3
4 def cat(fname):
5     return file(fname).xreadlines()
6
7 class grep:
8     """keep only lines that match the regexp"""
9     def __init__(self,pat,flags=0):
10        self.fun = re.compile(pat,flags).match
11    def __ror__(self,input):
12        return ifilter(self.fun,input)
13
14 class tr:
15    """apply arbitrary transform to each sequence element"""
16    def __init__(self,transform):
17        self.tr=transform
18    def __ror__(self,input):
19        return imap(self.tr,input)
20
21 class printlines_class:
22    """print sequence elements one per line"""
23    def __ror__(self,input):
24        for l in input:
25            print l
26
27 printlines=printlines_class()
28
29 class terminator:
30    """to be used at the end of a pipe-sequence"""
31    def __init__(self,method):
32        self.process=method
33    def __ror__(self,input):
34        return self.process(input)
35
36 # those objects transform generator to list, tuple or dict
37 aslist = terminator(list)
38 asdict = terminator(dict)
39 astuple = terminator(tuple)
40
41 # this object transforms seq to tuple sequence
42 enum = terminator( lambda input: izip(count(),input) )
43
44 #####
45 # example 1: equivalent to shell grep ".*bin/bash" /etc/passwd
46 cat('/etc/passwd') | tr(str.rstrip) | grep('.*bin/bash') | printlines
47
48 # example 2: get a list of int's methods beginning with '__r'
49 dir(int) | grep('__r') | aslist
50
51 # example 3: useless; returns a dict {0:'l',1:'a',2:'m',3:'b',4:'d',5:'a'}
52 'lambda' | enum | asdict
```