Zigzag Matrix

```
class Mat:
    def __init__(self, n):
        self._m = []
        for i in range(n):
            self._m.append([0] * n)
    def get(self, r, c):
        return self._m[r][c]
    def __iter__(self):
        self._x, self._y, self._incr = (0, -1, 1)
        self._n = len(self._m)
        return self
    def next(self):
        self._y += self._incr
        if self._y >= self._n:
            self._x += 1
            self._y = self._n - 1
            self._incr = -1
        elif self._y < 0:
            self._x += 1
            self._y = 0
            self._incr = 1
        if self._x >= self._n:
            raise StopIteration
        return self._m[self._x][self._y]
```

chaincomms

```
from subprocess import *
def chaincomms(cmds):
    c = cmds[0].split(' ')
    p = Popen(c, stdout=PIPE, stdin=PIPE)
    i = p.stdin
    for k in cmds[1:]:
        c = k.split(' ')
        p = Popen(c, stdin=p.stdout, stdout=PIPE,
close_fds=True)
    return (i, p.stdout)
```

forks

```
class Forks:
    _l = Lock()
    def __init__(self, nforks):
        self._c = Condition(self._l)
        self._st = [True] * nforks
        self._n = nforks
    def getforks(self):
        self._l.acquire()
        t = self.testforks()
        while not t:
            self._c.wait()
            t = self.testforks()
        for i in t:
            self._st[i] = False
        self._l.release()
        return (t[0], t[1], t[2])
    def releaseforks(self, trip):
        self._l.acquire()
        for i in trip:
            self._st[i] = True
        self._c.notify()
        self._l.release()
```

cache

```
class CacheServ:
    _vars = {}
    def __init__(self, ipport):
        self._l = Lock()
        self.sock = socket(AF_INET, SOCK_STREAM)
        self.sock.bind(('', ipport))
        self.sock.listen(1)
    def set(self, var, value): # set var in dict to
value
        self._l.acquire()
        self._vars[var] = value
        self._l.release()
    def get(self, var): # get var from dict
        self._l.acquire()
        v = self._vars[var]
        self._l.release()
        return v
    def varl(self):
        self._l.acquire()
        v = ' '.join(self._vars.keys())
        self._l.release()
        return v
    def start(self): # start accepting connections
        c = self.sock.accept()
```

cache (cont)

```
        while c: # main loop, create a thread here
            a = Agent(self, c)
            a.start()
            c = self.sock.accept()
```

zigzag

```
def zigzag(a):
    (x, y, incr) = (0, -1, 1)
    n = len(a._m)
    while True:
        y += incr
        if y >= n:
            x += 1
            y = n - 1
            incr = -1
        elif y < 0:
            x += 1
            y = 0
            incr = 1
        if x >= n:
            break
        yield a.get(x, y)
```

regex

```
from re import *
s = '144::122::71::2::3'
print findall('[0-9]+', s)
s = '144::122::71::2'
print sub('([0-9]+)::([0-9]+)::([0-9]+)::([0-9]+)',
'\g<4>.\g<3>.\g<2>.\g<1>', s)
s = '/home/onur/498/2011/prj/project.py'
print sub('/[^/]*$', '', s).split('/')[1:]
print s.split('/')[1:-1]
print findall('[^/]+', sub('/[^/]*$', '', s))
print sub('^.*/([^/]+)$', '\g<1>', s)
s = '78.165.170.147 - - [19/Apr/2011:12:36:52 +0300]
"GET remindersheet.pdf HTTP/1.1" 304 181'
print sub('^([^ ]+) [^:]+:([0-9]+:[0-9]+:[0-9]+) .*GET
([^ ]+) .*$',
          'IP=\g<1> F=\g<3> T=\g<2>', s)
```

agent

```
class Agent(Thread):
    def __init__(self, serv, sock):
        Thread.__init__(self)
        self._s = serv
        self.sock = sock[0]
    def run(self):
        print self.sock
        l = self.sock.recv(1024)
        while l:
            c = l.strip(' \n').split(' ')
            if c[0] == 'get':
                self.sock.send(serv.get(c[1]) + '\n')
            elif c[0] == 'set':
                serv.set(c[1], c[2])
            elif c[0] == 'vars':
                self.sock.send(serv.varl() + '\n')
            l = self.sock.recv(1024)
```

proxy cache

```
class ProxyCache:
    def __init__(self, addr):
        self.sock = socket(AF_INET, SOCK_STREAM)
        self.sock.connect(addr)
        self._obs = []
    def get(self, var):
        self.sock.send('get ' + var + '\n')
        return self.sock.recv(1024).strip('\n')
    def set(self, var, val):
        self.sock.send(' '.join(['set', var, val]) +
'\n')
        self.notify()
    def varl(self):
        self.sock.send('vars\n')
        return
self.sock.recv(1024).strip('\n').split(' ')
    def notify(self):
        for obj in self._obs:
            obj.update(self)
    def register(self, obj):
        if not (obj in self._obs):
            self._obs.append(obj)
    def unregister(self, obj):
        self._obs.delete(obj)
```