Programming for Science
Possible answers for workshop 3

Here are possible answers to the exercises in the Workshop 3. Usually these are not the only way of solving the problem, but they are relatively straightforward and clean ways.

1. Here's a version of the birthday program using indexing.

```
# Find the product of the contents of a list using indexing

birthday = [ 13, 10, 1958]  # Paddington bear's birthday

prod = 1

for i in range(len(birthday)):
    prod = prod*birthday[i]

print 'Product of ', birthday, 'is', prod
```

By and large the indexing version is less useful than the version (last workshop) that iterates over the list directly.

2. Here's a version of the program to count the letters in the rhyme using indexing. Note that rather than counting the words individually, we can find it as the number of elements in the list.

```
rhyme = ['Mary', 'had', 'a', 'little', 'lamb', 'whose', 'fleece', 'was',
'white', 'as', 'snow']

Nwords = len(rhyme)
Nletters = 0
for j in range(Nwords):
    Nletters = Nletters + len(rhyme[j])


print Nwords, 'words and', Nletters, 'letters'
```

3. A version of the program to find the average length of the words in the file is:

```
# Find the average length of words in a file.

words = open('words.txt', 'r').readlines()

Nwords = len(words)
Nletters = 0
for i in range(Nwords):
    word = words[i].strip()             # Remove white space
    Nletters = Nletters + len(word)
```

```
print Nwords, 'words and', Nletters, 'letters'
print 'Average word length', Nletters/float(Nwords)
```

4. Here is a function to calculate the dot product between a pair of vectors and a couple of simple tests. Note the docstring for the function.

```
def dot(a, b):
    """dot(a, b) -- find the dot product between lists a and b

    Arguments:
    - `a`: List of numbers
    - `b`: List of numbers

    Returns:
        Dot product

    a and be must have the same length.  Returns None if not.
    """
    N = len(a)
    if len(b) != N:
        print 'Lists have unequal lengths!', len(a), len(b)
        return None

    S = 0.0
    for i in range(N):
        S = S + a[i]*b[i]

    return S


# Test dot()

a = [ 0.04904772,  1.38633599,  0.94084723, -0.96782833, -0.66058869]
b = [ 1.07227778, -0.75143678,  0.54550933,  0.20866252, -1.4619395 ]

print 'Dot product between'
print '  ', a, '\n  ', b
print 'is', dot(a, b)

# Dot product between two orthogonal vectors (at right angles) -- should b
u = [1.0, -2.0, 1.0]
v = [1.0, 1.0, 1.0]

print 'Dot product between'
print  '  ', u, '\n  ', v
print 'is', dot(u, v)
```

```
# Lists of unequal length
dot([0, 1], [1, 2, 3])
```

Here is the output:

```
Dot product between
   [0.04904772, 1.38633599, 0.94084723, -0.96782833, -0.66058869]
   [1.07227778, -0.75143678, 0.54550933, 0.20866252, -1.4619395]
is 0.287881070961
Dot product between
   [1.0, -2.0, 1.0]
   [1.0, 1.0, 1.0]
is 0.0
Lists have unequal lengths! 2 3
```

5. A function for zipping two lists is similar to `dot()` because it combines the corresponding elements from a pair of lists. Again it is easier to do using indexing rather than iterating over the lists directly.

```
def zipper(x, y):
    """zipper -- zip lists x and y together

    Arguments:
    - `x`: list
    - `y`: list

    Returns a list with corresponding elements of x and y as sublists
    """
    N = len(x)
    if len(y) != N:
        print 'Lists have unequal lengths!', len(x), len(y)
        return None                         # Signal error by returning None

    xy = []                                 # Start with empty list
    for n in range(N):
        xy.append([ x[n], y[n] ])
    return xy


# Test with a simple example
u = [3, 4, 5, 6]
v = ['a', 'b', 'c', 'd']
print 'zipping ', u, 'and', v
print zipper(u, v)
```

Here is the output:

```
python zipper.py
zipping  [3, 4, 5, 6] and ['a', 'b', 'c', 'd']
[[3, 'a'], [4, 'b'], [5, 'c'], [6, 'd']]
```

6. Here's a program to find palindromes. Note that the testing for whether a word is a palindrome has been isolated into the function `is_palindrome` that returns a Boolean depending on whether the word is a palindrome. Although this is very simple test (using slices to reverse the string), this sort of organisation makes the way the code operates clear.

```
# Find all the palindromes in words.txt

def is_palindrome(s):
    """
    Return True if `s` is a palindrome
    """
    return s == s[::-1]


lines = open('words.txt', 'r').readlines()

for line in lines:
    word = line.strip()
    if len(word) >= 3 and is_palindrome(word):
        print word,
print                                    # Finish the line
```

This code prints all the palindromes on a single long line and here they are:

```
python palindrome.py
aba acca adda affa aga aha ajaja aka ala alala alula ama amma
   ana anana anna apa ara arara atta ava awa bib bob boob bub
   civic dad deed deedeed degged did dod dud eke elle eme ere
   eve ewe eye gag gig gog hah hallah huh ihi imi immi kakkak
   kayak keek kelek lemel level maam madam mem mesem mim minim
   mum murdrum nan non noon nun oho otto pap peep pep pip poop
   pop pup radar redder refer repaper retter rever reviver
   rotator rotor siris sis sooloos tat tebbet teet tenet terret
   tit toot tot tst tut tyt ulu ululu umu utu waw wow yaray yoy
```

7. As the workshop sheet suggested, writing the Fibonacci program with indices is easier than without. Here is a version:

```
#  Generate the first few elements of the Fibonacci series from a list of

Fibonacci = [0, 1]        # Initialise with the first two elements of the

for n in range(18):       # Two elements already generated, so need anothe
    F = Fibonacci[-1] + Fibonacci[-2]
    Fibonacci.append(F)

for F in Fibonacci:
    print F
```

A third, not so elegant, version that indexes the list directly is

```
#  Generate the first few elements of the Fibonacci series from a list of

N = 20                # Number of elements
F = [0]*N             # List with N zeros
F[:2] = [0, 1]        # Initialise with the first two elements of the seri

for n in range(2, 20):    # n indexes the element being generated
    F[n] = F[n-1] + F[n-2]

for f in F:
    print f
```

8. Here is a program to calculate the first time the trader gets paid a bonus:

```
# Find the first time a trader's accumulated profit exceeds a bonus thresh

threshold = 200000   # He gets a bonus when accumulated profit exceeds thi

trades = open('trader.txt', 'r').readlines()

# Process all the trades to work out the profit to the company and the
# ratio of profitable trades to loss-making ones.
profit = 0                              # Trader profit as before
cprofit = 0                             # Company profit
Nprofitable = 0
day = 0

for trade in trades:                    # Number of days he's profitable
    trade = float(trade.strip())
    if trade > 0:
        Nprofitable = Nprofitable + 1

    profit = profit + trade
    if profit > threshold:
```

```
        print 'Bonus on day', day
        profit = 0                      # Start counting again
    else:
        cprofit = cprofit + trade       # Company keeps the profit

    day = day+1

days = len(trades)                      # Number of days traded
years = 7*float(days)/(5*(365-28))
salary = 100000
cprofit = cprofit - years*salary

print 'Fraction of profitable days', float(Nprofitable)/float(days)
print 'Average annual company profit', cprofit/years
```

The output is

```
python trader.py
Bonus on day 159
Bonus on day 345
Bonus on day 915
Bonus on day 971
Bonus on day 1300
Bonus on day 1642
Bonus on day 1967
Bonus on day 2290
Bonus on day 2434
Bonus on day 2562
Bonus on day 2807
Bonus on day 2992
Bonus on day 3036
Bonus on day 3187
Bonus on day 3360
Bonus on day 3498
Bonus on day 3740
Bonus on day 4160
Bonus on day 4232
Bonus on day 4344
Bonus on day 4453
Bonus on day 4718
...
Bonus on day 17548
Bonus on day 17768
Bonus on day 17860
Bonus on day 18031
Bonus on day 18289
Bonus on day 18444
Bonus on day 18660
```

```
Bonus on day 19038
Bonus on day 19638
Bonus on day 19758
Fraction of profitable days 0.544
Average annual company profit 116829.579816
```