# Elegant String Searches

Martin Schweitzer

March 2, 2009

# The Problem

Given a set of letters and a dictionary, find all words that can be made from those letters.

| | | |
|---|---|---|
| H | C | O |
| C | A | L |
| E | T | O |

# The Problem

Given a set of letters and a dictionary, find all words that can be made from those letters.

| | | |
|---|---|---|
| H | C | O |
| C | A | L |
| E | T | O |

# The naiive approach

- Find all 9-letter permutations

# The naiive approach

- Find all 9-letter permutations
- Check each permutation against every word in dictionary

# The naiive approach

- Find all 9-letter permutations
- Check each permutation against every word in dictionary
- Find all 8-letter permutations

# The naiive approach

- Find all 9-letter permutations
- Check each permutation against every word in dictionary
- Find all 8-letter permutations
- Check each permutation against every word in dictionary

# The naiive approach

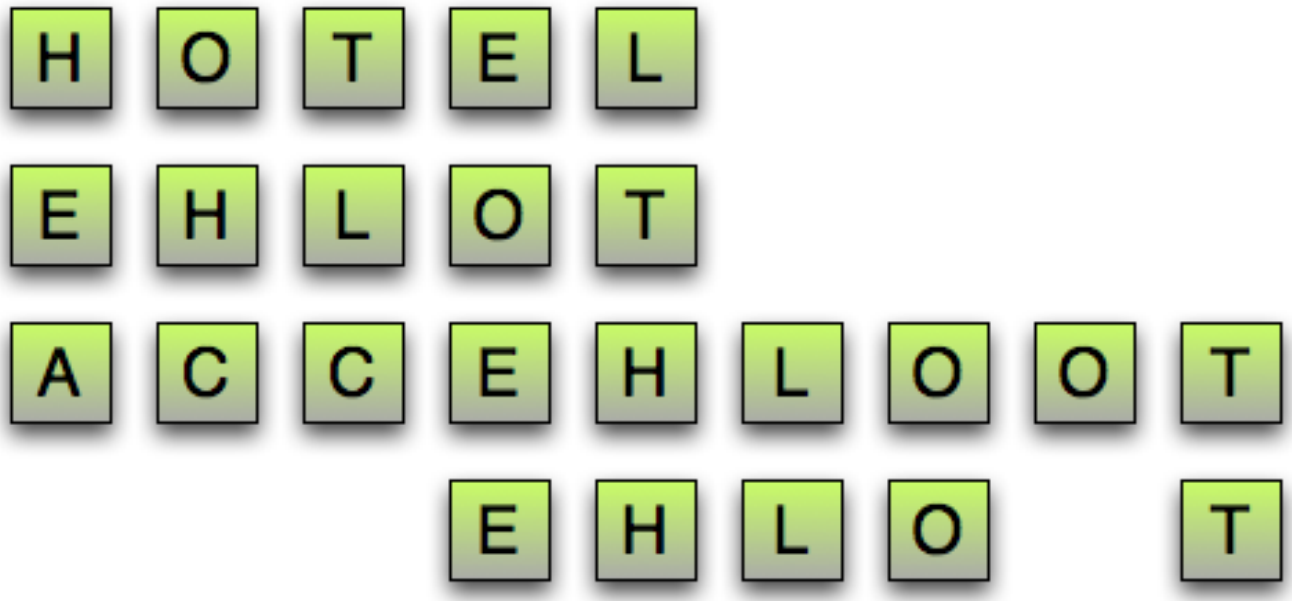
- Find all 9-letter permutations
- Check each permutation against every word in dictionary
- Find all 8-letter permutations
- Check each permutation against every word in dictionary
- Find all 7-letter permutations

# The naiive approach

- Find all 9-letter permutations
- Check each permutation against every word in dictionary
- Find all 8-letter permutations
- Check each permutation against every word in dictionary
- Find all 7-letter permutations
- ...

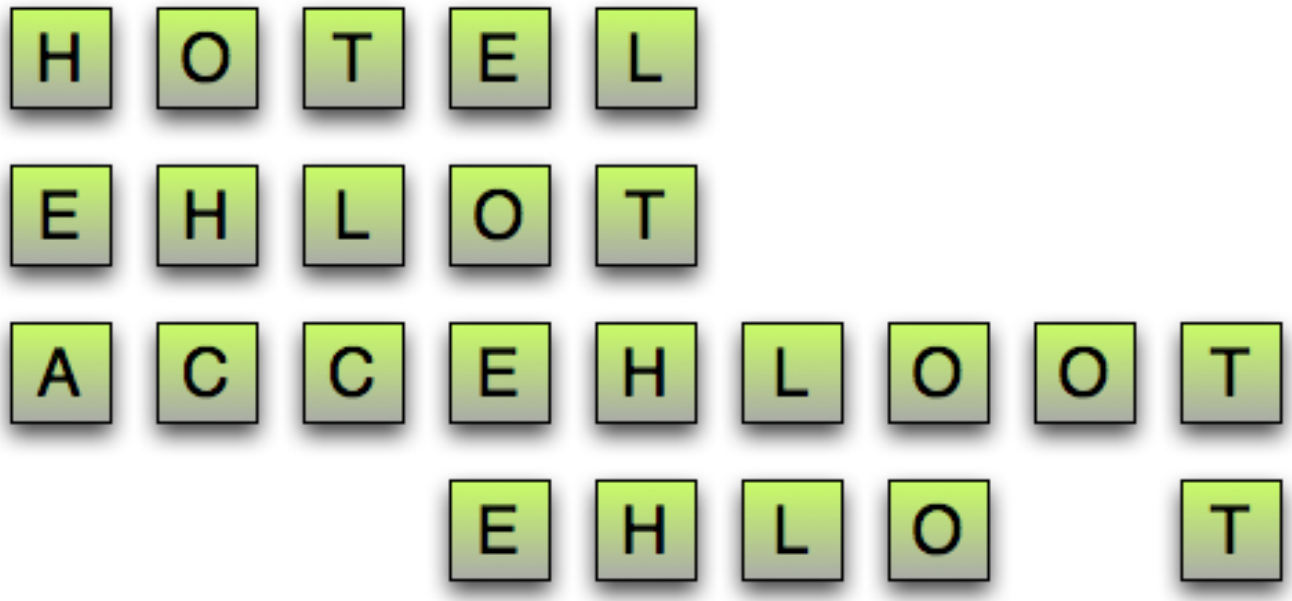
# The Experienced approach

- Sort the letters in the square



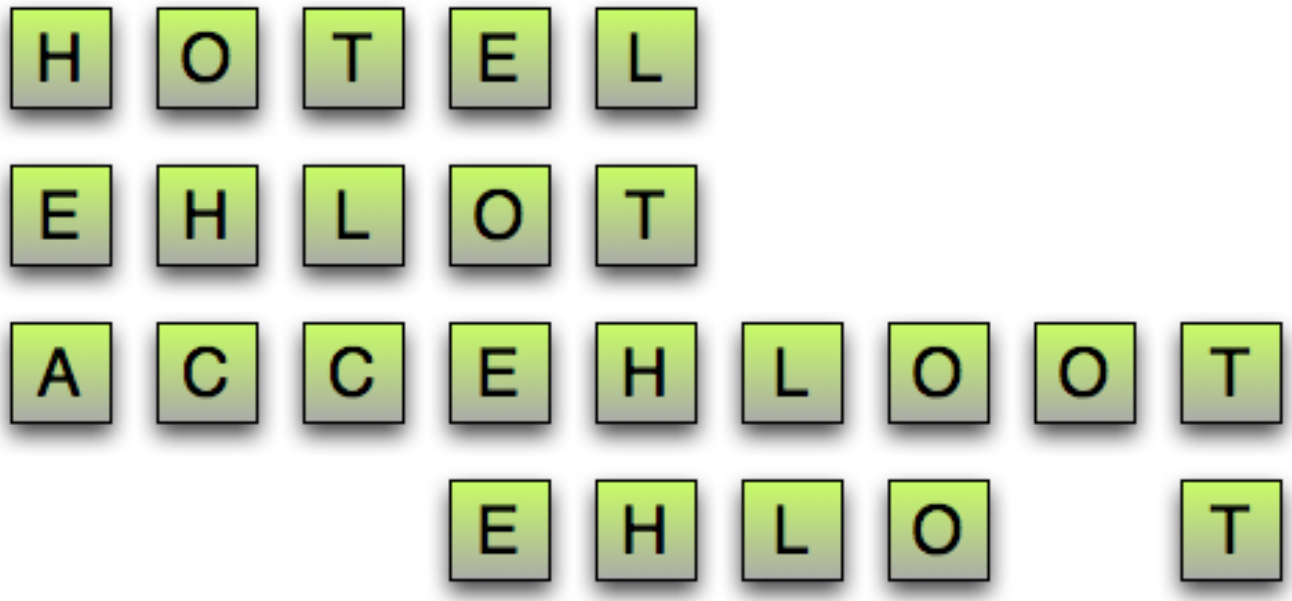
# The Experienced approach

- Sort the letters in the square
- For each word in the dictionary



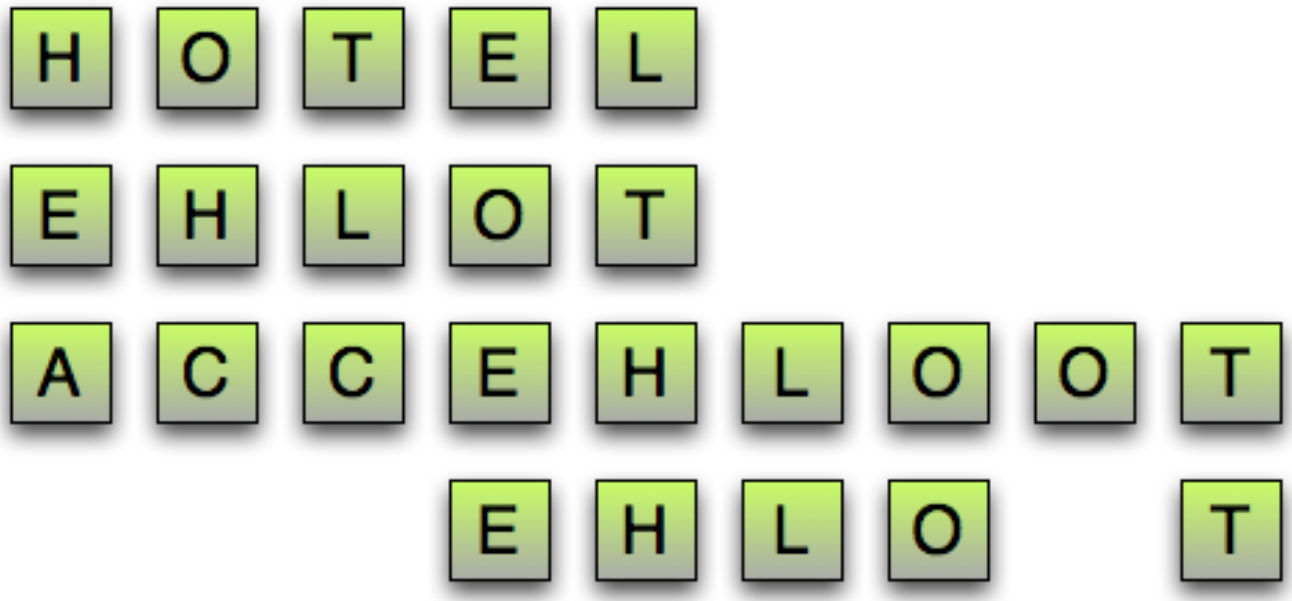
# The Experienced approach

- Sort the letters in the square
- For each word in the dictionary
  - sort the word



# The Experienced approach

- Sort the letters in the square
- For each word in the dictionary
  - sort the word
  - check if we can make that word



# The Experienced approach

- Only go through the dictionary once

# The Experienced approach

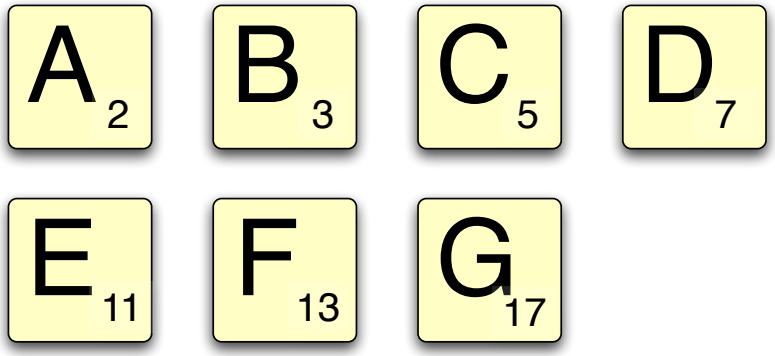
- Only go through the dictionary once
- Still have to sort each word in the dictionary

# The Fundamental Theorem of Arithmetic

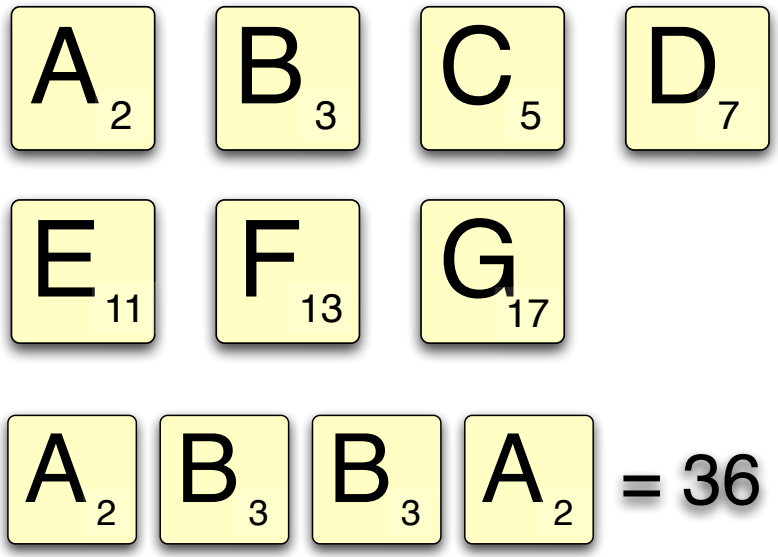
*Every natural number greater than 1 can be written as a unique product of prime numbers*

$6939 = 2^3 \times 3 \times 17^2$
$1200 = 2^4 \times 3 \times 5^2$
$36 = 2^2 \times 3^2$

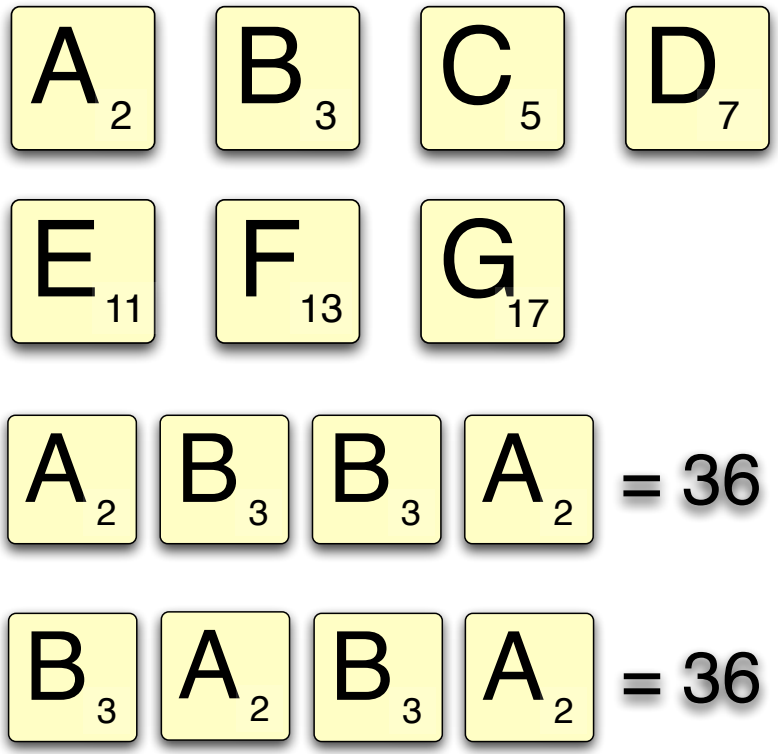
# Fundamental insight



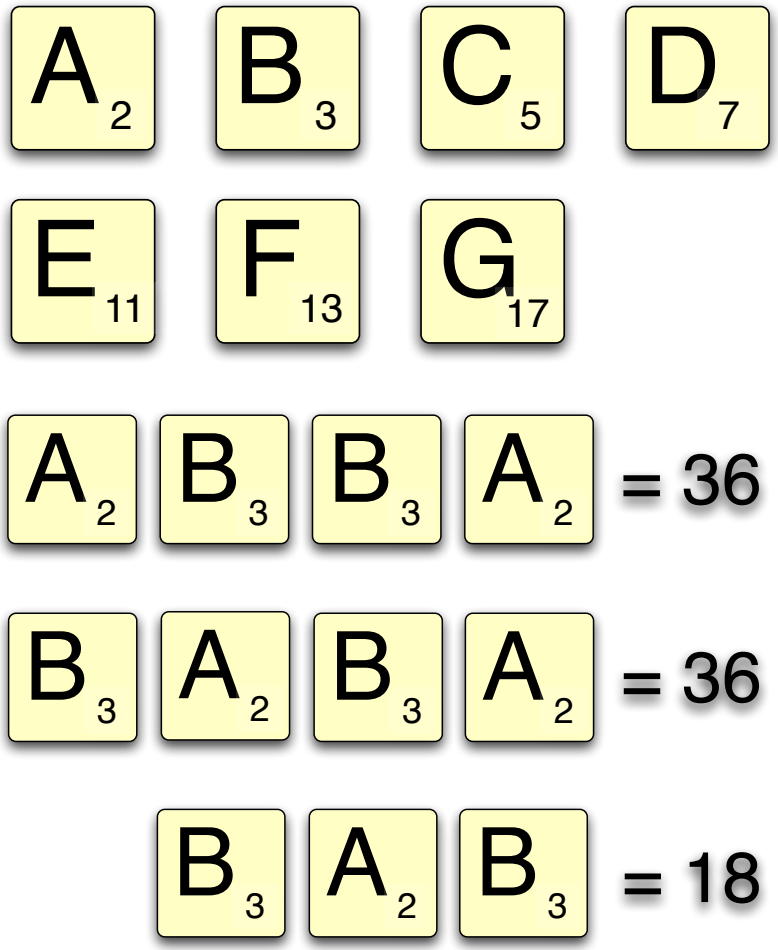
# Fundamental insight



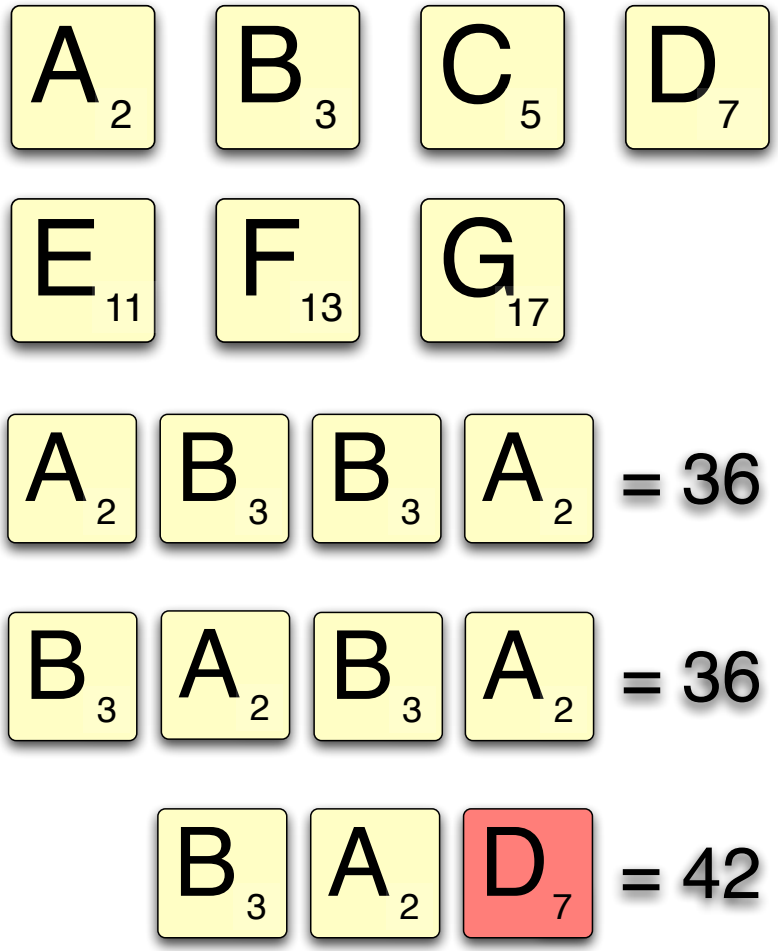
# Fundamental insight



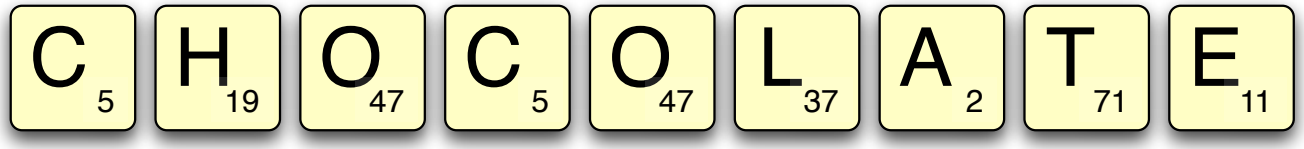
# Fundamental insight



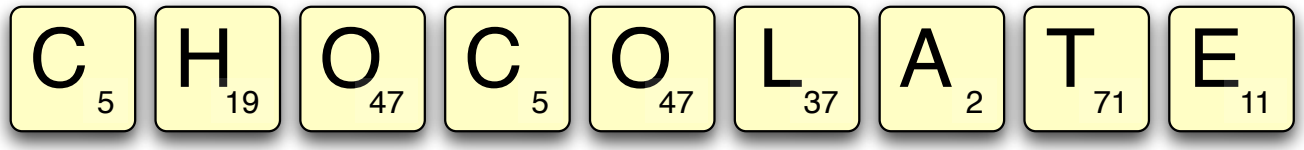
# Fundamental insight

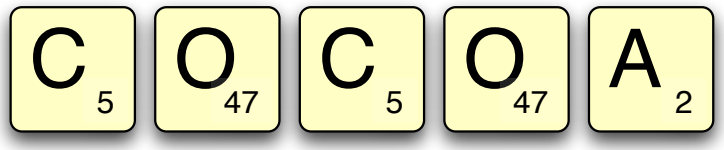


# Does chocolate contain cocoa?

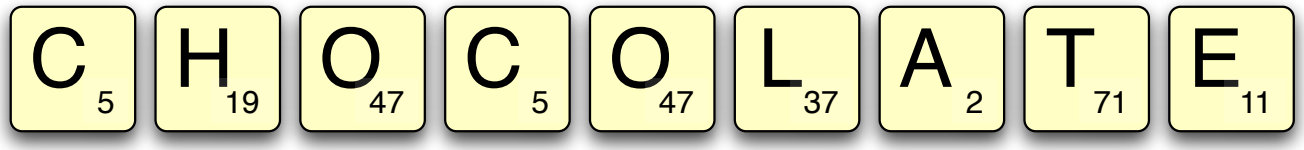


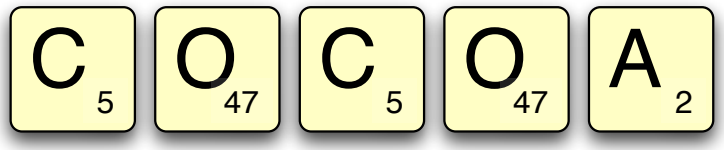
# Does chocolate contain cocoa?



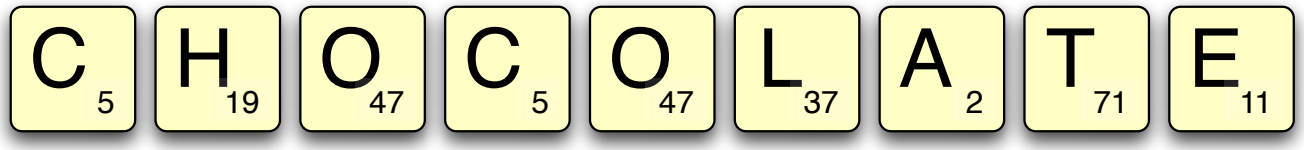
# Does chocolate contain cocoa?

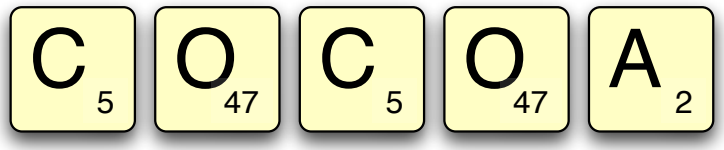



```
>>> print 5 * 19 * 47 * 5 * 47 * 37 * 2 * 71 * 11
```

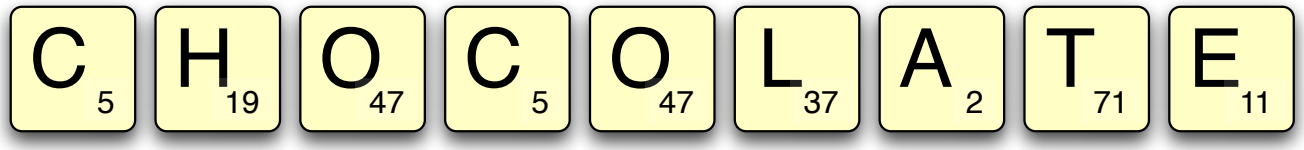
# Does chocolate contain cocoa?

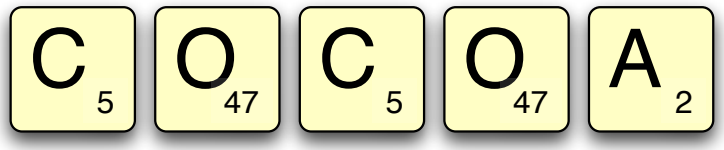



```
>>> print 5 * 19 * 47 * 5 * 47 * 37 * 2 * 71 * 11
60641799350
```

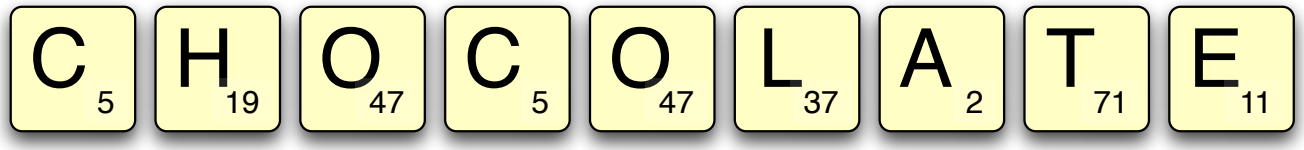
# Does chocolate contain cocoa?

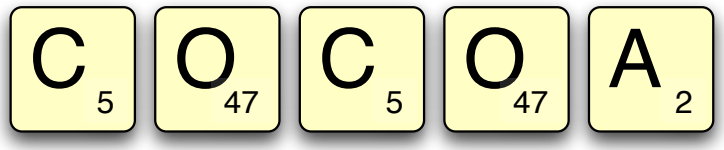



```
>>> print 5 * 19 * 47 * 5 * 47 * 37 * 2 * 71 * 11
60641799350
>>> print 5 * 47 * 5 * 47 * 2
```

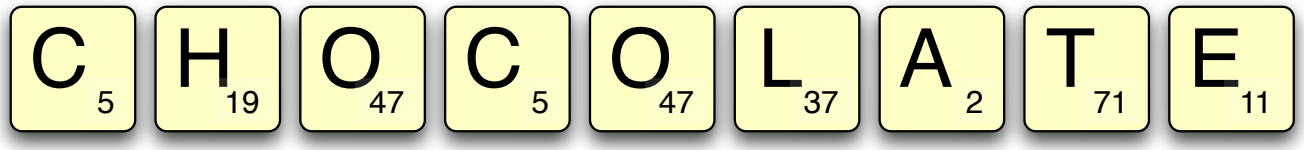
# Does chocolate contain cocoa?

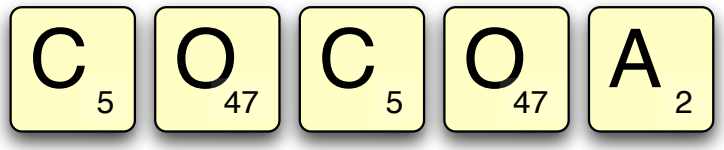



```
>>> print 5 * 19 * 47 * 5 * 47 * 37 * 2 * 71 * 11
60641799350
>>> print 5 * 47 * 5 * 47 * 2
110450
```

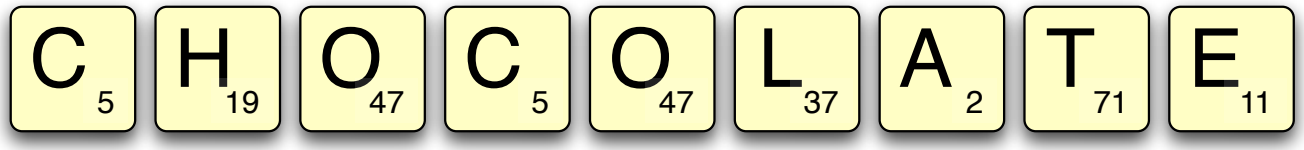
# Does chocolate contain cocoa?

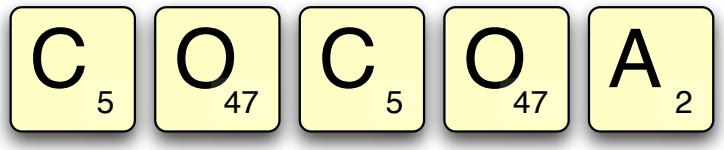



```
»> print 5 * 19 * 47 * 5 * 47 * 37 * 2 * 71 * 11
60641799350
»> print 5 * 47 * 5 * 47 * 2
110450
»> print 60641799350 % 110450
```

# Does chocolate contain cocoa?




```
>>> print 5 * 19 * 47 * 5 * 47 * 37 * 2 * 71 * 11
60641799350
>>> print 5 * 47 * 5 * 47 * 2
110450
>>> print 60641799350 % 110450
0
```

# Program

```
primes = [2, 3, 5, 7, 11, 17, ...

def find_number(word):
    tot = 1
    for ch in word:
        tot = tot * primes[ord(ch.lower()) - ord('a')]
    return tot
```

# Program 1

```
#!/usr/bin/python
dictionary = '/usr/share/dict/words'
secret = 'chocolate'
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,

def prime_val(ch):
    return primes[ord(ch.lower()) - ord('a')]

def get_val(word):
    total = 1
    for ch in word:
        total *= prime_val(ch)
    return total
```

# Program 1 (continued)

```
magic = get_val(secret)
for word in open(dictionary):
    word = word.rstrip('\n')
    if magic % get_val(word) == 0:
        print word
```

# Map/Reduce

The following lines:

```
def get_val(word):
    total = 1
    for ch in word:
        total *= prime_val(ch)
    return total
```

Can be replaced by:

```
def get_val(word):
    return reduce(lambda x,y: x*y, map(lambda x:
        primes[ord(x) - ord('a')], word.lower()))
```

# Program 2

```
dictionary = '/usr/share/dict/words'
secret = 'chocolate'
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,

def get_val(word):
    return reduce(lambda x,y: x*y, map(lambda x:
        primes[ord(x) - ord('a')], word.lower()))

magic = get_val(secret)
for word in open(dictionary):
    word = word.rstrip('\n')
    if magic % get_val(word) == 0:
        print word
```

# List comprehension

The following code:

```
def get_val(word):
    return reduce(lambda x,y: x*y, map(lambda x:
        primes[ord(x) - ord('a')], word.lower()))
```

Can be (and should be?) replaced with:

```
def get_val(word):
        return [j for j in [1] for i in word for j in
            [j * primes[ord(i.lower()) - ord('a')]]][-1]
```

# More reductions

And the following code:

```
for word in open(dictionary):
    word = word.rstrip('\n')
    if magic % get_val(word) == 0:
        print word
```

Can be replaced with:

```
[print3(word.strip()) for word in open(dictionary)
    if magic % get_val(word.rstrip('\n')) == 0]
```

# Program 3

```
dictionary = '/usr/share/dict/words'
secret = 'chocolate'
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,

def get_val(word):
    return [j for j in [1] for i in word for j in
        [j * primes[ord(i.lower()) - ord('a')]]][-1]

def print3(word): print word

magic = get_val(secret)
[print3(word.strip()) for word in open(dictionary)
    if magic % get_val(word.rstrip('\n')) == 0]
```

# Conclusions

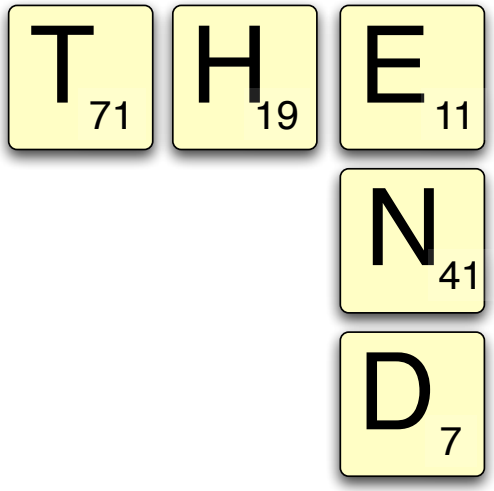
- The first solution is not necessarily the best.

# Conclusions

- The first solution is not necessarily the best.
- Python rocks.

# Conclusions

- The first solution is not necessarily the best.
- Python rocks.
- Sometimes something that they taught you in school is useful.


T 71
H 19
E 11
N 41
D 7