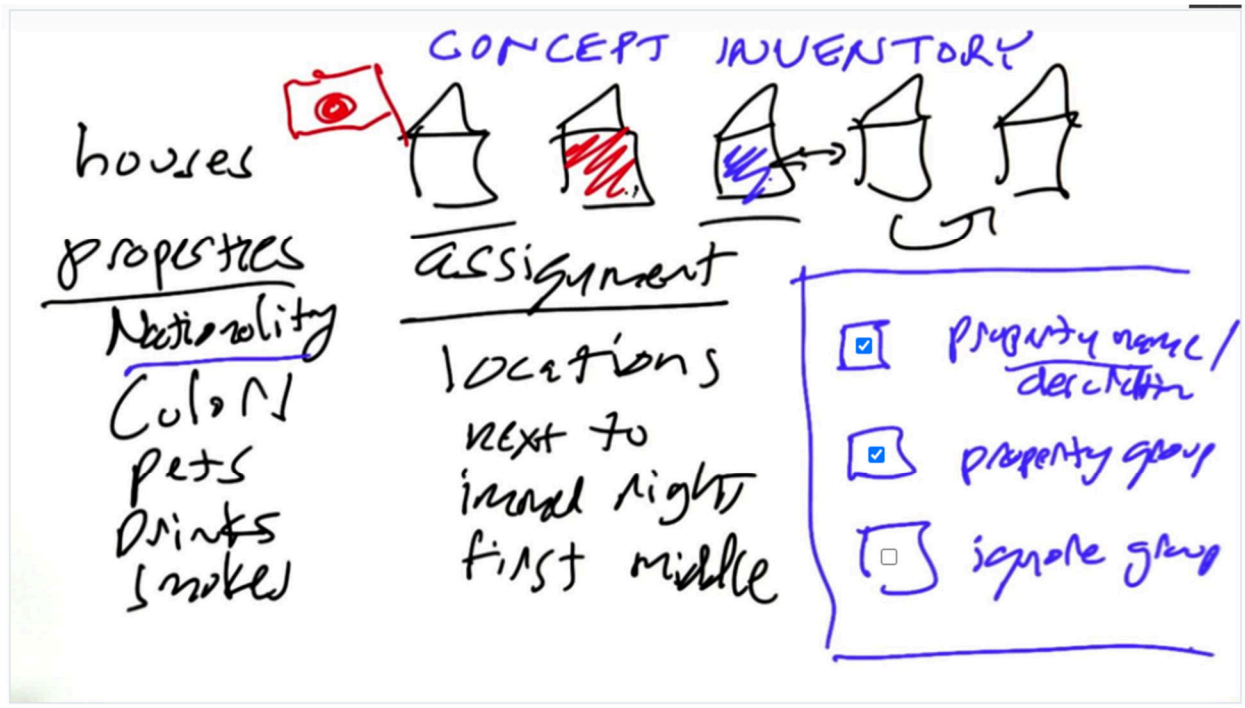
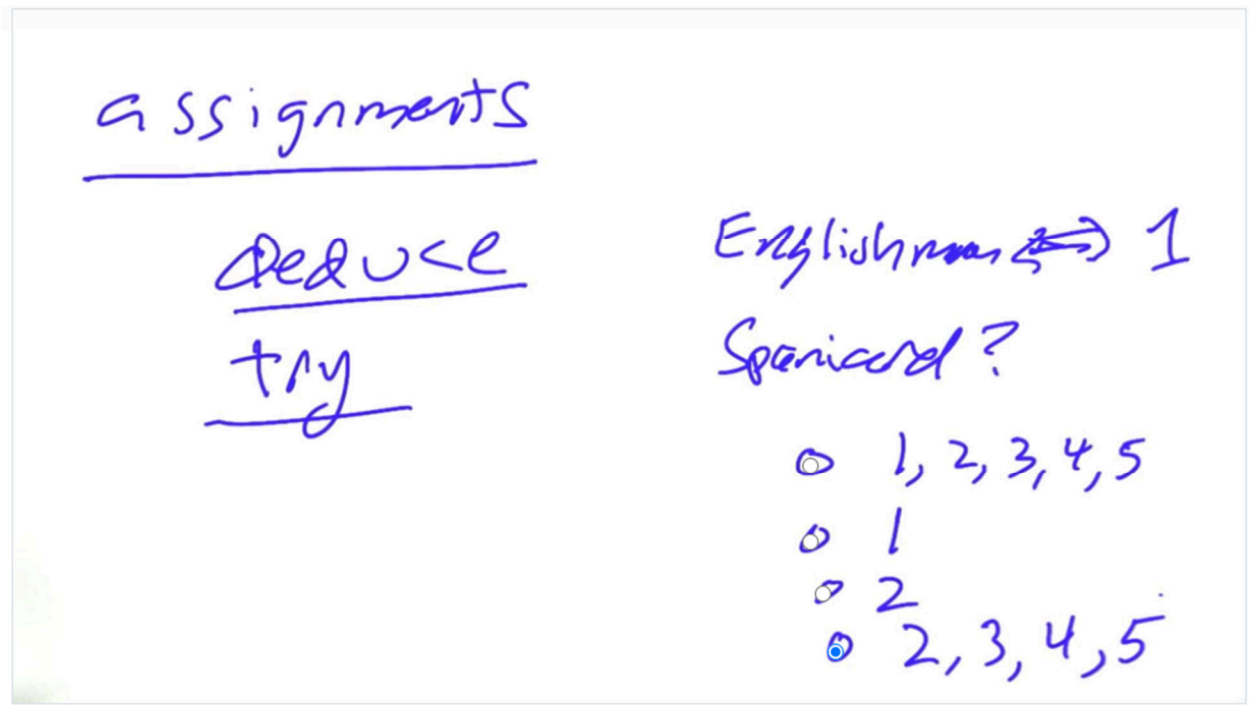
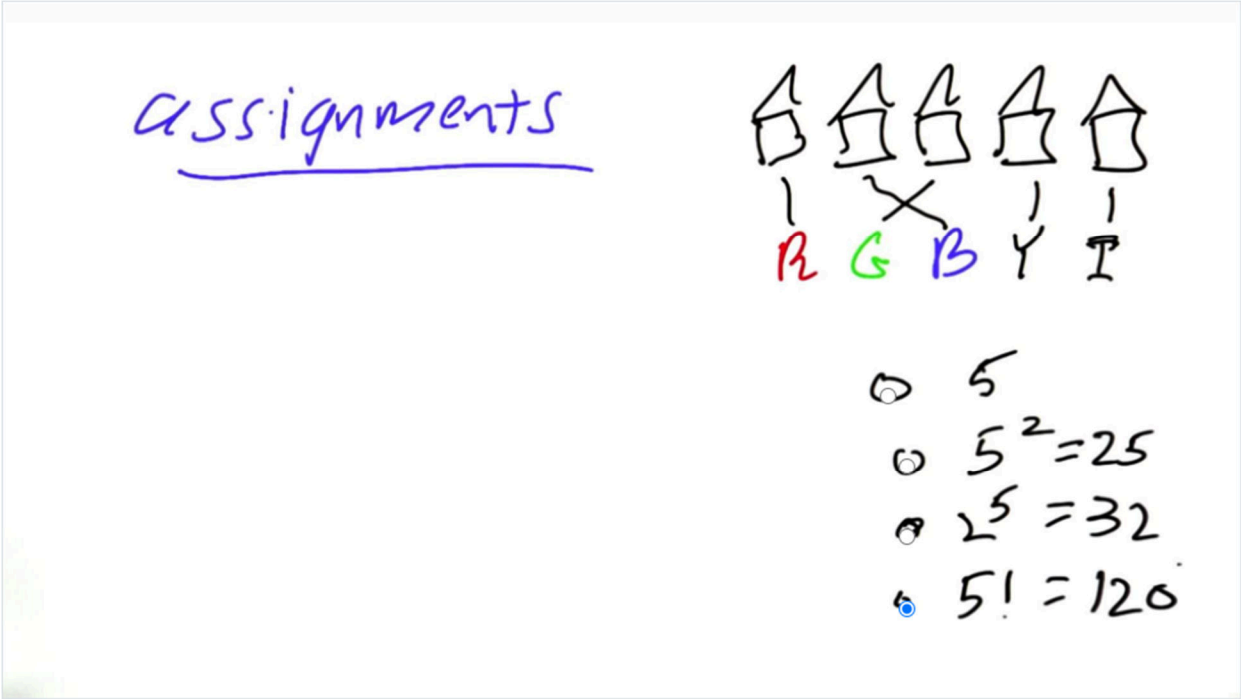
Zebra Puzzle



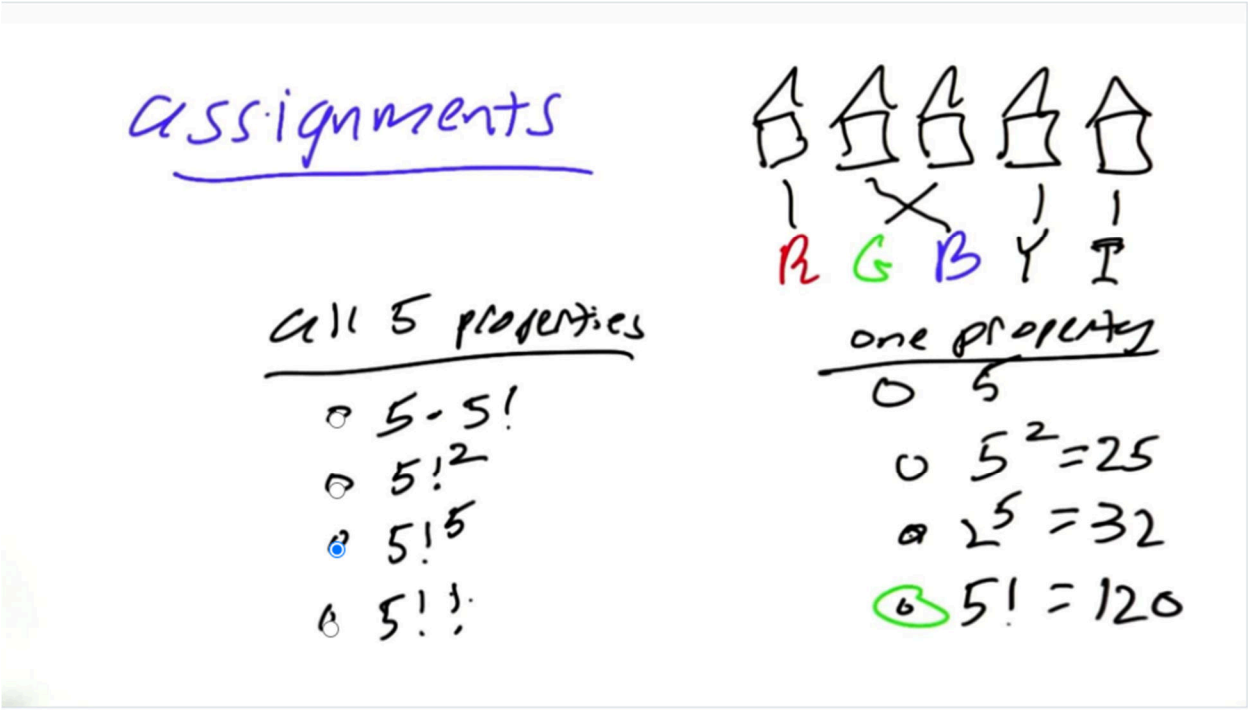
Where The Spaniard



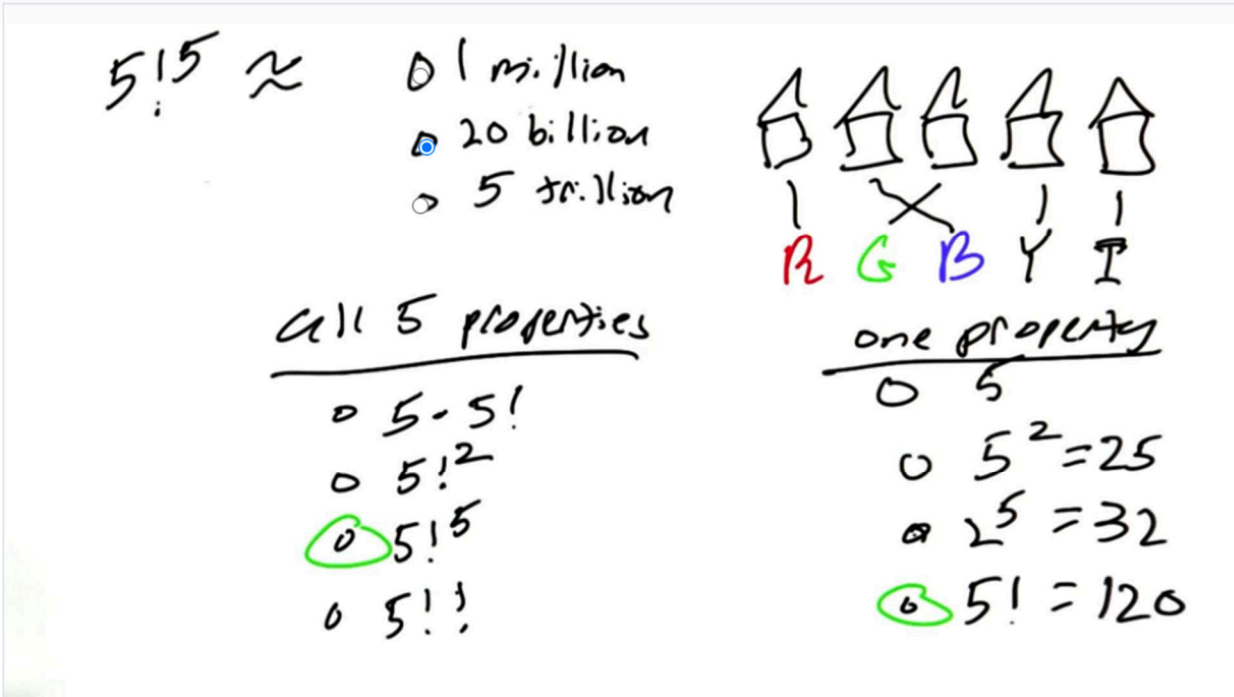
Counting Assignments



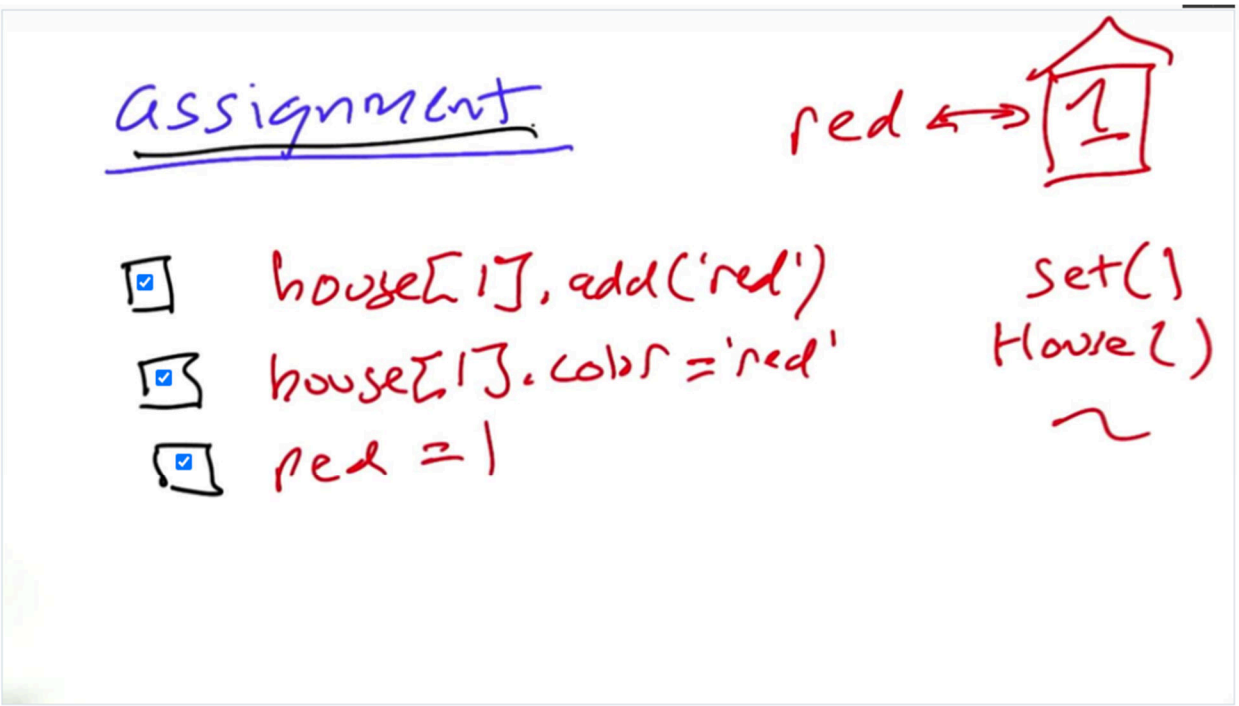
Multiple Properties



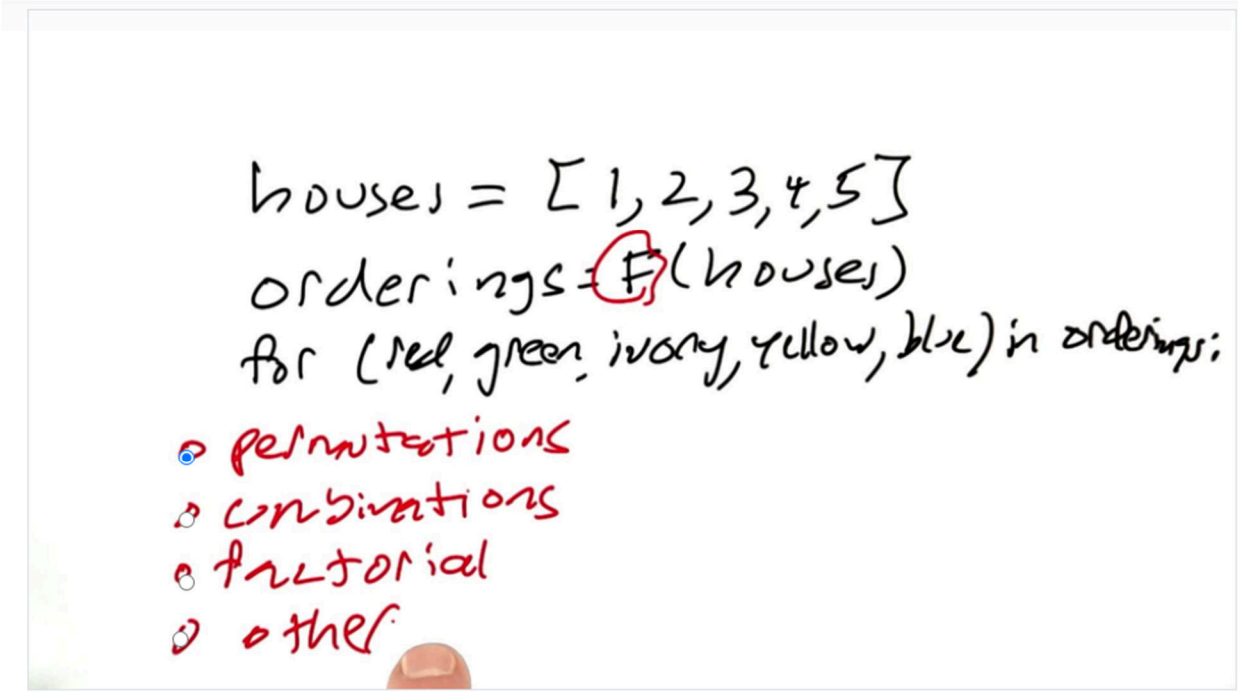
Back of the Envelope



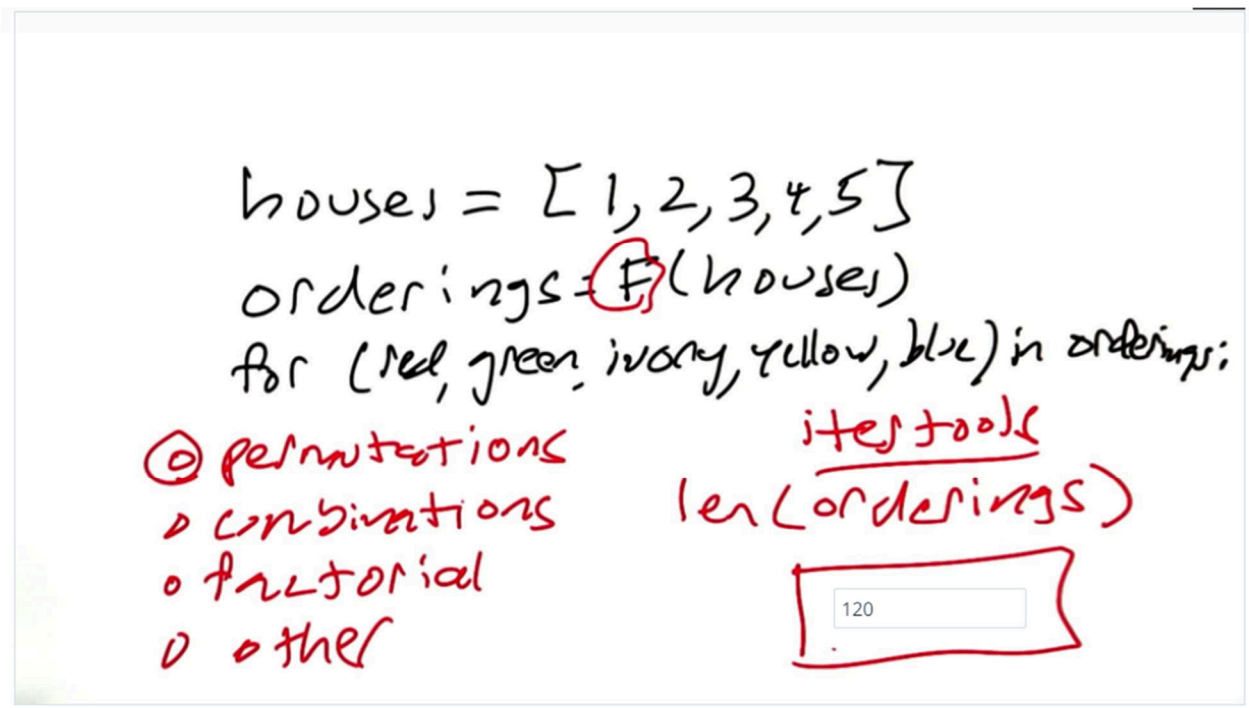
Leaving Happy Valley



Ordering Houses



Length Of Orderings



Estimating Runtime

```python


import itertools

houses = [1, 2, 3, 4, 5]
orderings = list(itertools.permutations(houses))

for (red, green, ivory, yellow, blue) in orderings:
  for (Englishman, Spaniard, Ukranian, Japanese, Norwegian) in orderings:
    for (dog, snails, fox, horse, ZEBRA) in orderings:
      for (coffee, tea, milk, oj, WATER) in orderings:
        for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings:
          # constraints go here


          # 1 sec
          # 1 min
          # 1 hour
          # 1 day
```

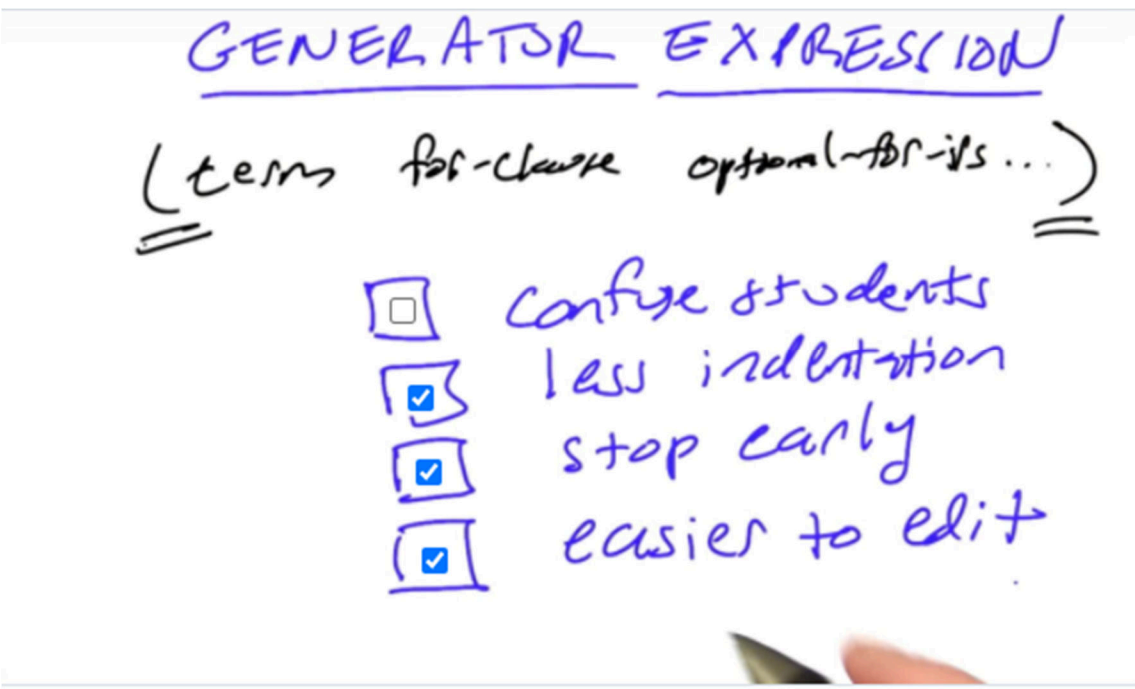
Red Englishman

```python
import itertools

houses = [1, 2, 3, 4, 5]
orderings = list(itertools.permutations(houses))

for (red, green, ivory, yellow, blue) in orderings:
  for (Englishman, Spaniard, Ukranian, Japanese, Norwegian) in orderings:
    for (dog, snails, fox, horse, ZEBRA) in orderings:
      for (coffee, tea, milk, oj, WATER) in orderings:
        for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings:
          if ( Englishman == red ): #2


          # 1 sec
          # 1 min
          # 1 hour
          # 1 day
```

Neighbors

```python

import itertools

houses = [1, 2, 3, 4, 5]
orderings = list(itertools.permutations(houses))

def imright(h1, h2):
    "House h1 is immediately right of h2 if h1-h2 == 1."
    return h1-h2 == 1

def nextto(h1, h2):
    "Two houses are next to each other if they differ by 1."
    return abs(h1-h2) == 1

```

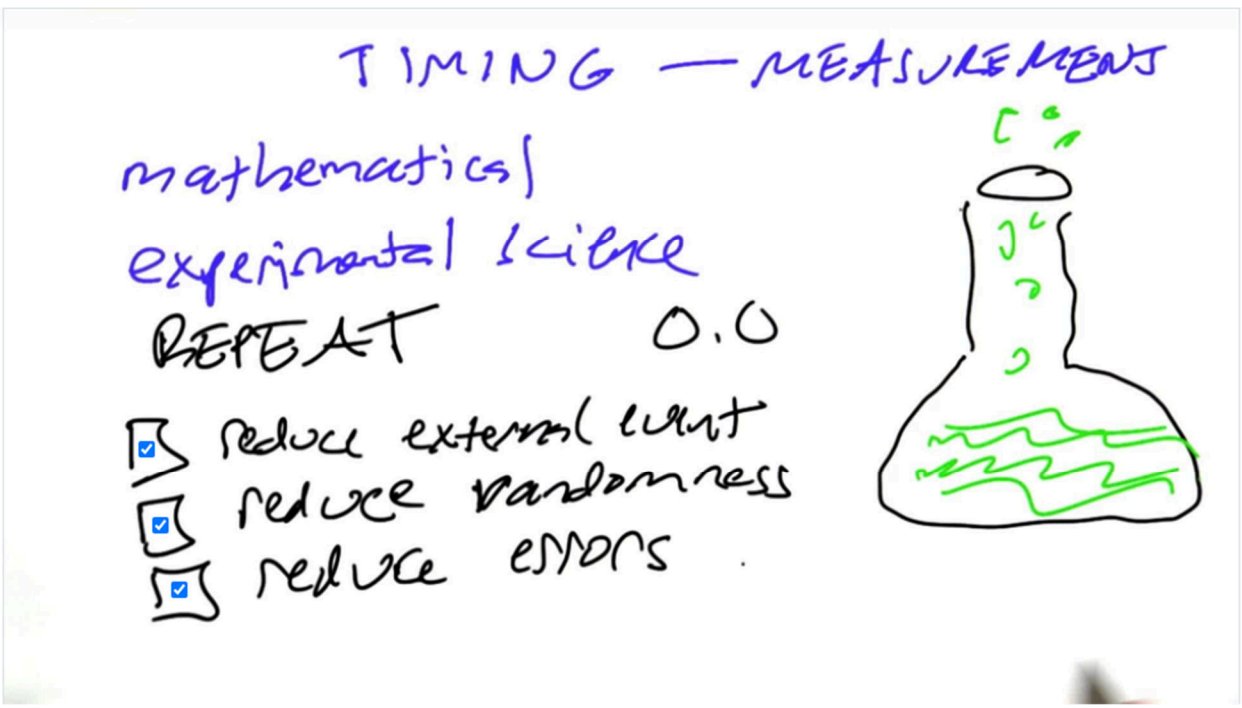
Generator Expressions



Eliminating Redundancy

```
def zebra_puzzle():
    "Return a tuple (WATER, ZEBRA) indicating their house numbers."
    houses = first, _, middle, _, _ = [1, 2, 3, 4, 5]
    orderings = list(itertools.permutations(houses))  #1
    return next((WATER, ZEBRA)
            for (red, green, ivory, yellow, blue) in orderings

            for (Englishman, Spaniard, Ukranian, Japanese, Norwegian) in orderings

            if Englishman is red            #2
            for (dog, snails, fox, horse, ZEBRA) in orderings

            for (coffee, tea, milk, oj, WATER) in orderings

            for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings
            if Spaniard is dog              #3
```

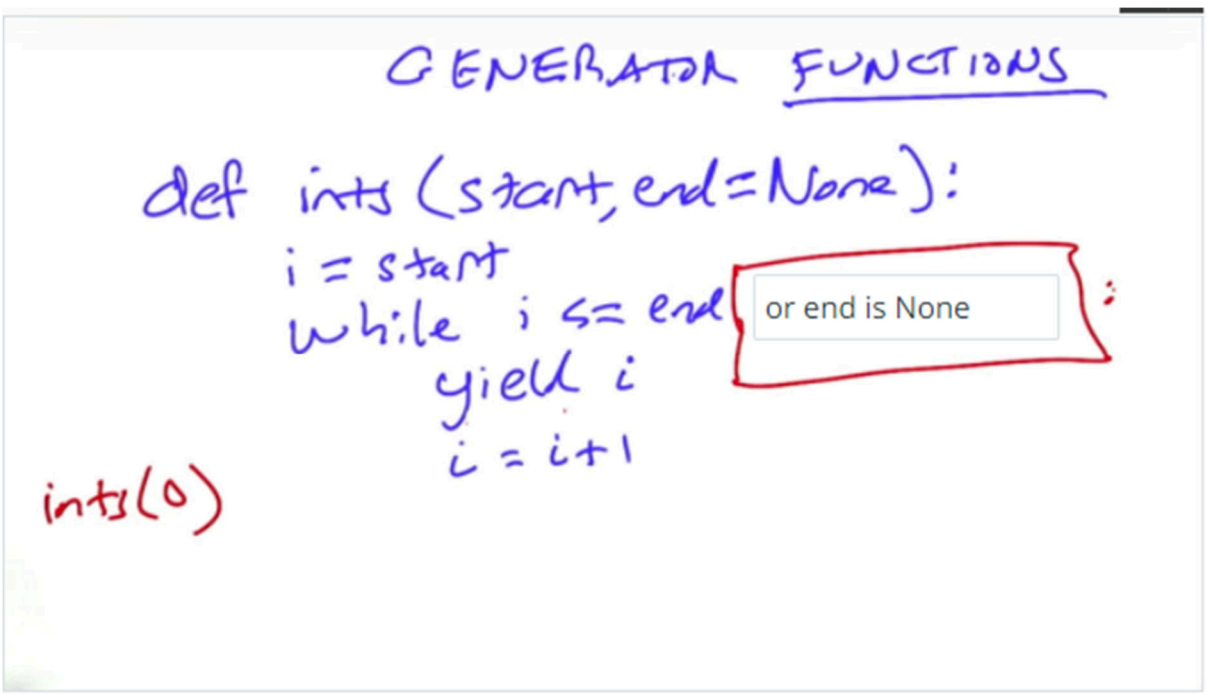
Good Science



Timed Calls

```python

def timedcalls(n, fn, *args):
    """Call fn(*args) repeatedly: n times if n is an int, or up to
    n seconds if n is a float; return the min, avg, and max time"""
    if isinstance(n, int):
        times = [timedcall(fn, *args)[0] for _ in range(n)]
    else:
        times = []
        while sum(times) < n:
            times.append(timedcall(fn, *args)[0])
    return min(times), average(times), max(times)

```

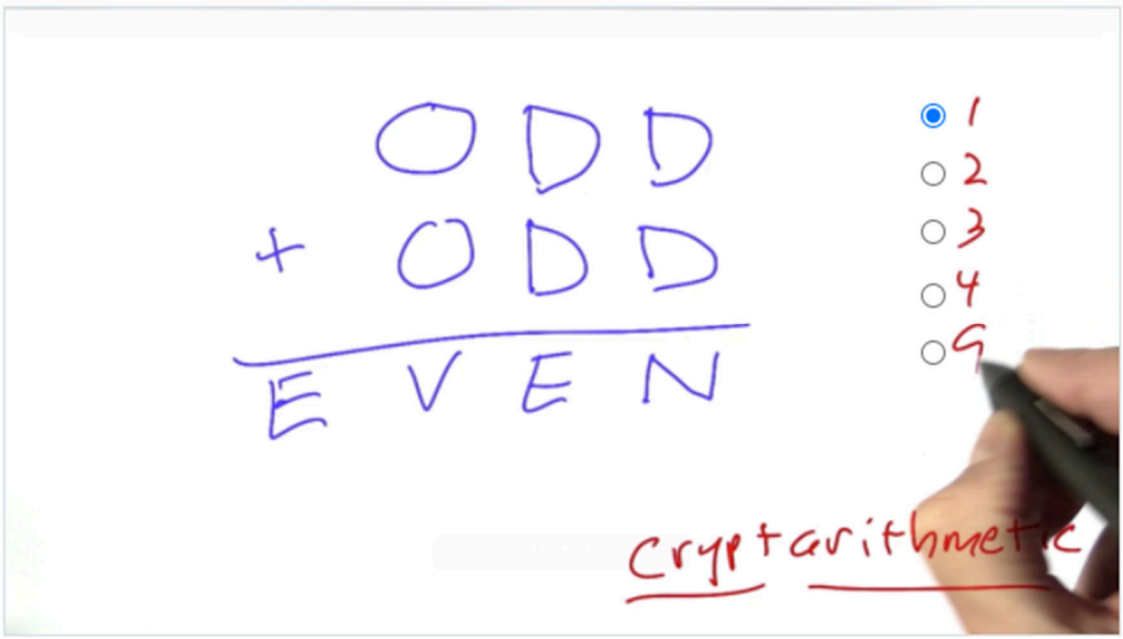
Yeilding Results



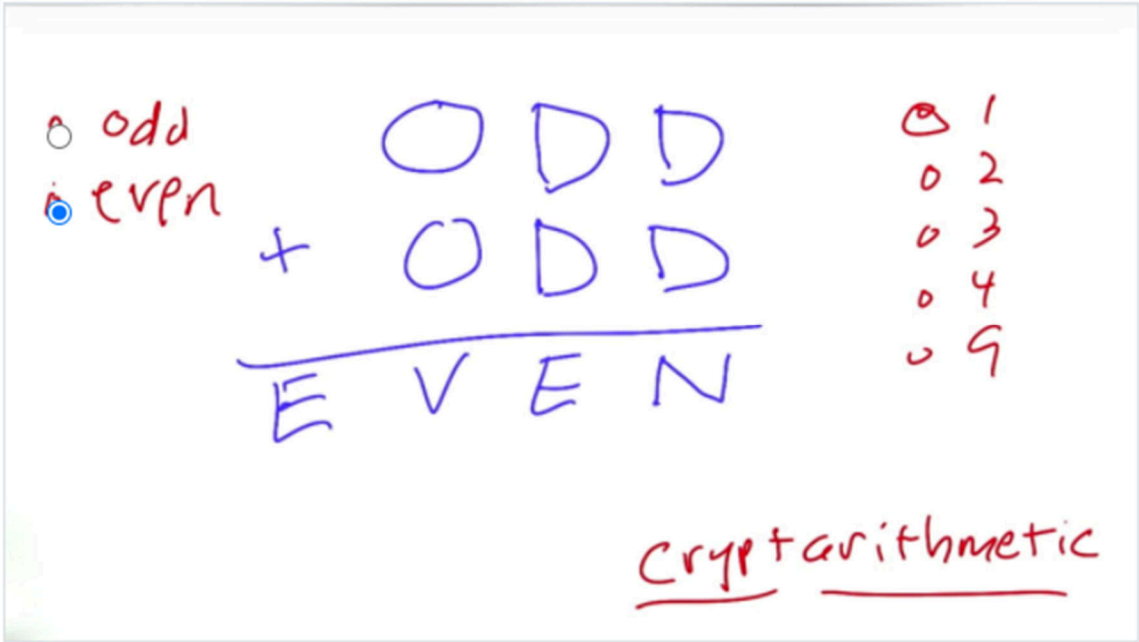
All Ints

```python
def all_ints():
    "Generate integers in the order 0, +1, -1, +2, -2, +3, -3, ..."
    yield 0
    for i in ints(1):
        yield +i
        yield -i
```

Cryptarithmetic



Odd or Even



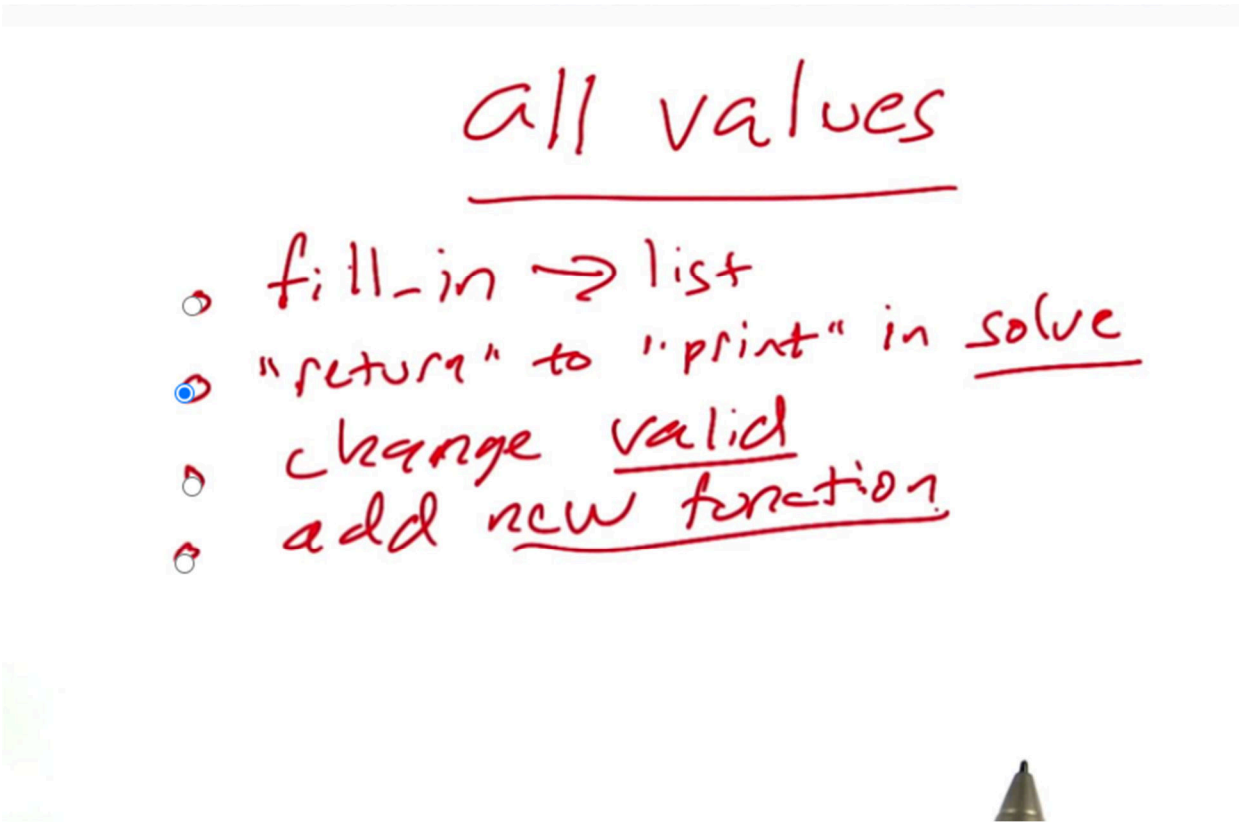
Solving Cryptarithmetic

```
def solve(formula):
    """Given a formula like 'ODD + ODD == EVEN', fill in digits to
    Input formula is a string; output is a digit-filled-in string
    for f in fill_in(formula):
        if valid(f):
            return f
```

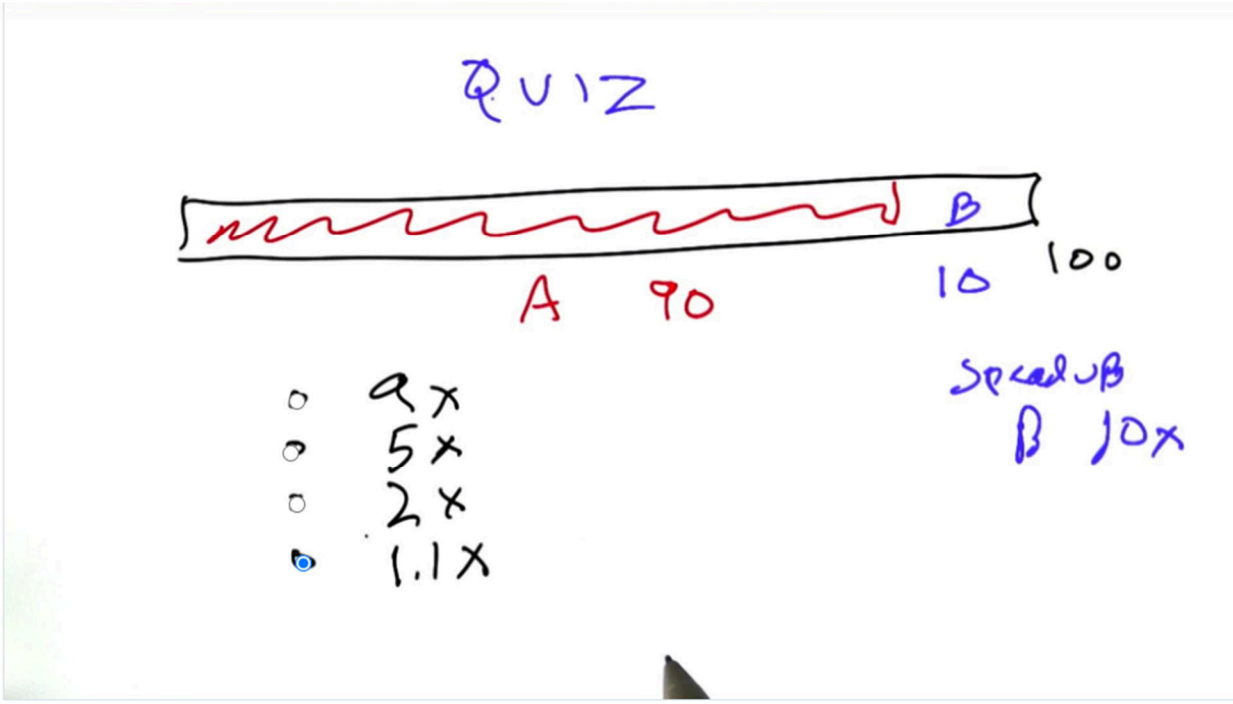
Filling In Fill In

```
def fill_in(formula):
    "Generate all possible fillings-in of letters in formula with digits."
    letters = ''.join(set(re.findall('[A-Z]', formula))) #should be a string
    for digits in itertools.permutations('1234567890', len(letters)):
        table = string.maketrans(letters, ''.join(digits))
        yield formula.translate(table)
```

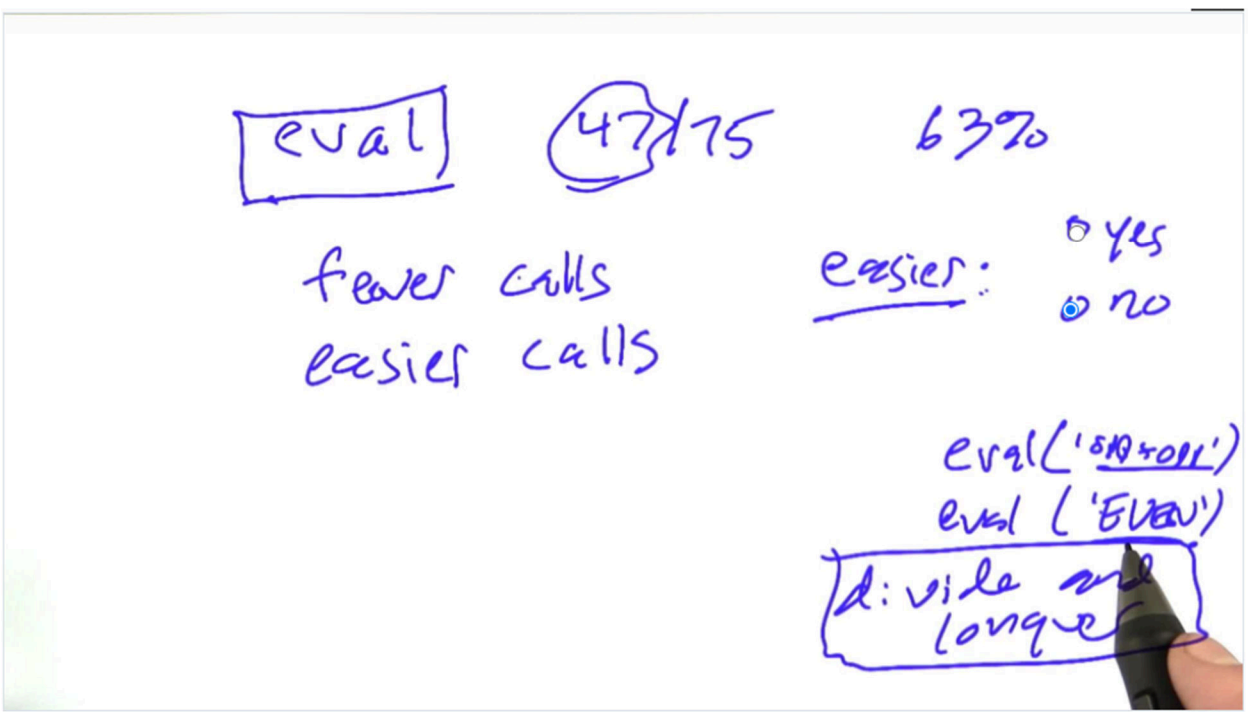
Find All Values



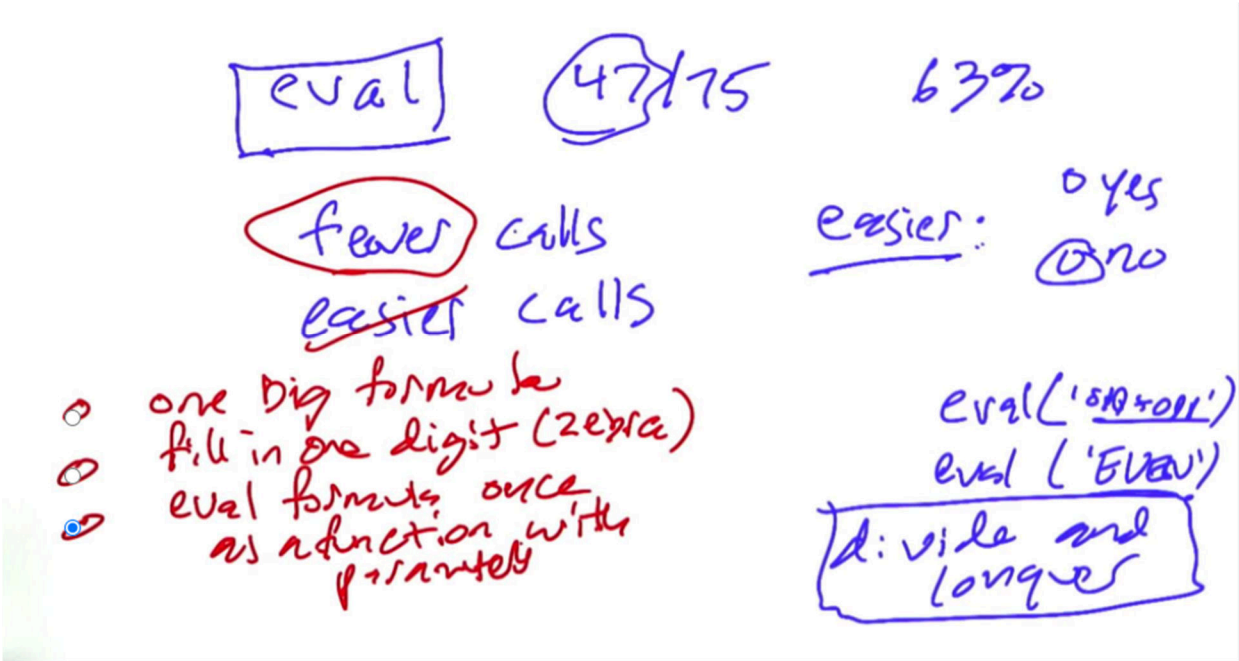
Increasing Speed



Rethinking Eval



Making Fewer Calls



Compile World

```
def compile_word(word):
    """Compile a word of uppercase letters as numeric digits.
    E.g., compile_word('YOU') => '(1*U+10*O+100*Y)'
    Non-uppercase words unchanged: compile_word('+') => '+'"""
    if word.isupper():
        terms = [('%s*%s' % (10**i, d))
                 for (i, d) in enumerate(word[::-1])]
        return '('+ '+' .join(terms) + ')'
    else:
        return word
```