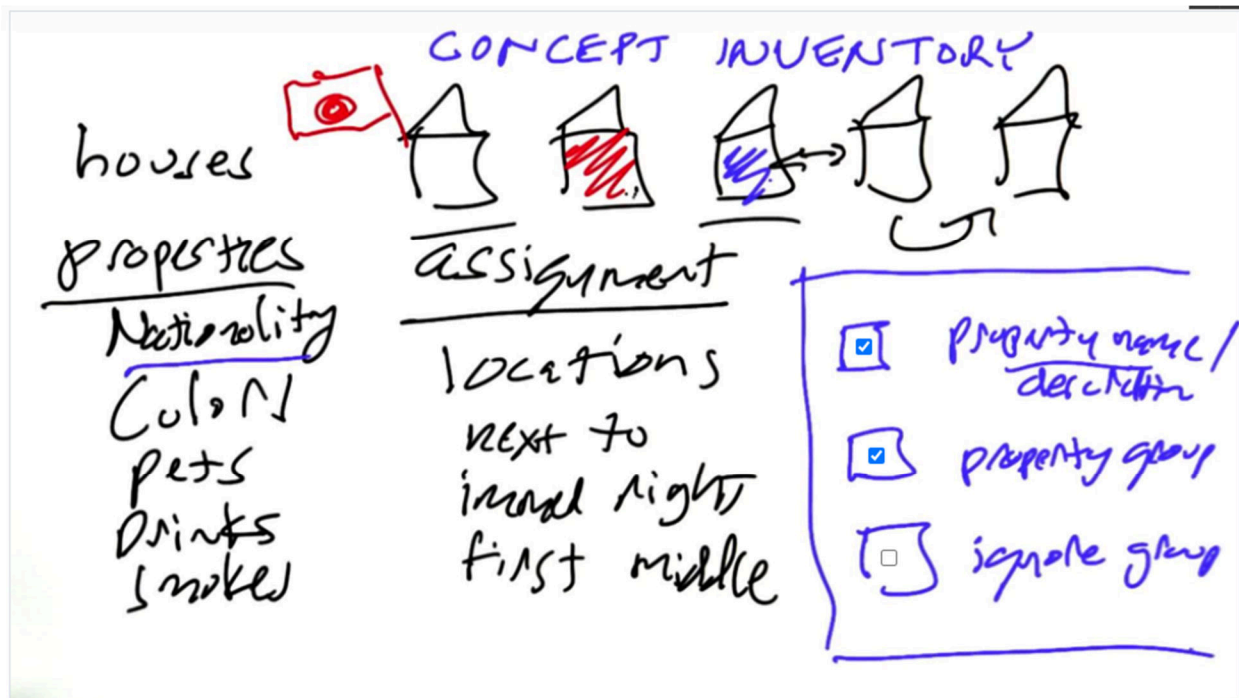
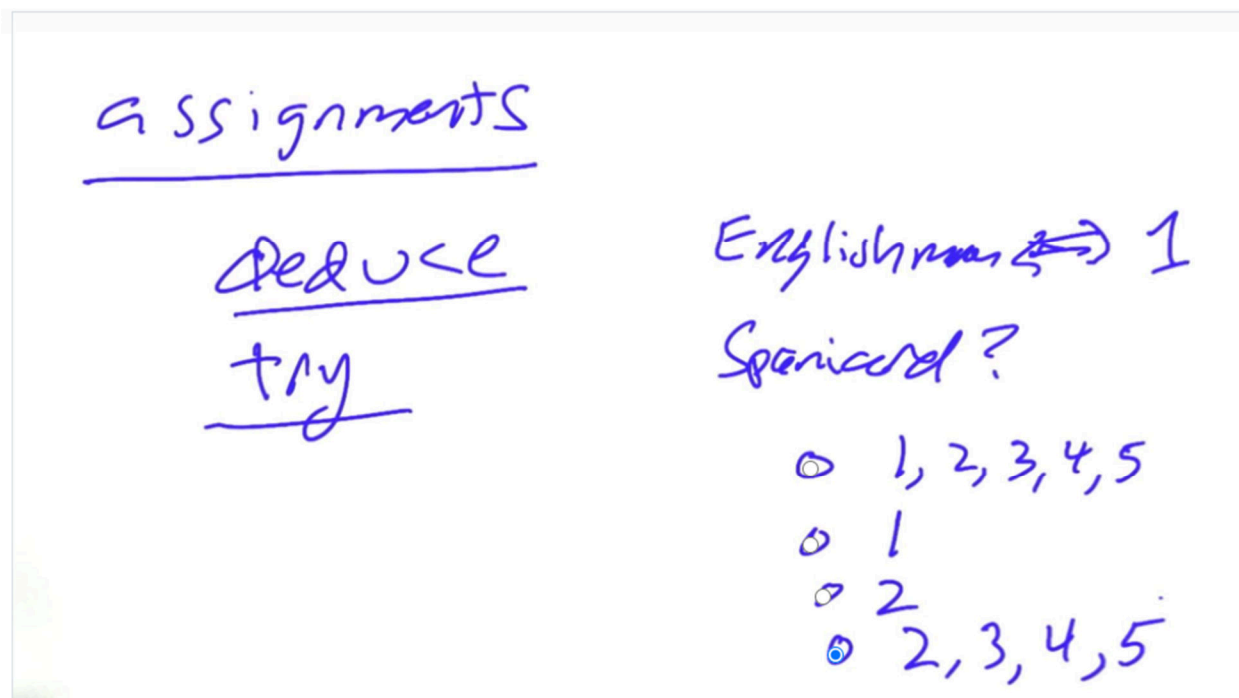


Zebra Puzzle

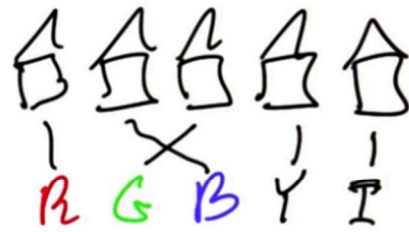


Where The Spaniard



Counting Assignments

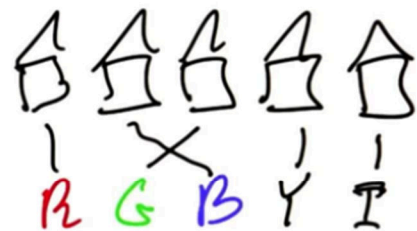
assignments



- 5
- $5^2 = 25$
- $2^5 = 32$
- $5! = 120$

Multiple Properties

assignments



all 5 properties

- $5 - 5!$
- $5!^2$
- $5!^5$
- $5!^1$

one property

- 5
- $5^2 = 25$
- $2^5 = 32$
- $5! = 120$

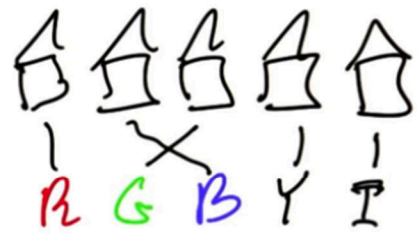
Back of the Envelope

$5!5 \approx$

- 1 million
- 20 billion
- 5 trillion

all 5 properties

- $5 \cdot 5!$
- $5!^2$
- $5!5$
- $5!^3$



one property

- 5
- $5^2 = 25$
- $2^5 = 32$
- $5! = 120$

Leaving Happy Valley

Assignment



- ☒ `house[1].add('red')`
- ☒ `house[1].color = 'red'`
- ☒ `red = 1`

`set(1
House 1)
~`

Ordering Houses

```
houses = [1, 2, 3, 4, 5]
orderings = F(houses)
for (red, green, ivory, yellow, blue) in orderings:
```

- permutations
- combinations
- factorial
- other

Length Of Orderings

```
houses = [1, 2, 3, 4, 5]
orderings = F(houses)
for (red, green, ivory, yellow, blue) in orderings:
```

- permutations
- combinations
- factorial
- other

itertools
`len(orderings)`

120

Estimating Runtime

```
1
2
3
4 import itertools
5
6 houses = [1, 2, 3, 4, 5]
7 orderings = list(itertools.permutations(houses))
8
9 for (red, green, ivory, yellow, blue) in orderings:
10     for (Englishman, Spaniard, Ukrainian, Japanese, Norwegian) in orderings:
11         for (dog, snails, fox, horse, ZEBRA) in orderings:
12             for (coffee, tea, milk, oj, WATER) in orderings:
13                 for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings:
14                     # constraints go here
15
16
17     # 1 sec
18     # 1 min
19     # 1 hour
20     # 1 day
```

Red Englishman

```
4 import itertools
5
6 houses = [1, 2, 3, 4, 5]
7 orderings = list(itertools.permutations(houses))
8
9 for (red, green, ivory, yellow, blue) in orderings:
10     for (Englishman, Spaniard, Ukrainian, Japanese, Norwegian) in orderings:
11         for (dog, snails, fox, horse, ZEBRA) in orderings:
12             for (coffee, tea, milk, oj, WATER) in orderings:
13                 for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings:
14                     if (Englishman == red): #2
15
16
17     # 1 sec
18     # 1 min
19     # 1 hour
20     # 1 day
```

Neighbors

```
7
8 import itertools
9
10 houses = [1, 2, 3, 4, 5]
11 orderings = list(itertools.permutations(houses))
12
13 def imright(h1, h2):
14     "House h1 is immediately right of h2 if h1-h2 == 1."
15     return h1-h2 == 1
16
17 def nextto(h1, h2):
18     "Two houses are next to each other if they differ by 1."
19     return abs(h1-h2) == 1
20
```

Generator Expressions

GENERATOR EXPRESSION
(term for-clause optional ~for-ifs...)

- ☐ confuse students
- ☒ less indentation
- ☒ stop early
- ☒ easier to edit

Eliminating Redundancy

```
3
4 def zebra_puzzle():
5     "Return a tuple (WATER, ZEBRA) indicating their house numbers."
6     houses = first, _, middle, _, _ = [1, 2, 3, 4, 5]
7     orderings = list(itertools.permutations(houses)) #1
8     return next((WATER, ZEBRA)
9                 for (red, green, ivory, yellow, blue) in orderings
10                  for (Englishman, Spaniard, Ukrainian, Japanese, Norwegian) in orderings
11                  if Englishman is red #2
12                  for (dog, snails, fox, horse, ZEBRA) in orderings
13                  for (coffee, tea, milk, oj, WATER) in orderings
14                  for (OldGold, Kools, Chesterfields, LuckyStrike, Parliaments) in orderings
15                  if Spaniard is dog #3
```


Good Science

TIMING — MEASUREMENT

mathematics |
experimental science

REPEAT 0.0

- ☒ reduce external event
- ☒ reduce randomness
- ☒ reduce errors



Timed Calls

```
6
7 def timedcalls(n, fn, *args):
8     """Call fn(*args) repeatedly: n times if n is an int, or up to
9     n seconds if n is a float; return the min, avg, and max time"""
10    if isinstance(n, int):
11        times = [timedcall(fn, *args)[0] for _ in range(n)]
12    else:
13        times = []
14        while sum(times) < n:
15            times.append(timedcall(fn, *args)[0])
16    return min(times), average(times), max(times)
17
```

Yeilding Results

GENERATOR FUNCTIONS

```
def ints(start, end=None):
    i = start
    while i <= end:
        yield i
        i = i + 1
```

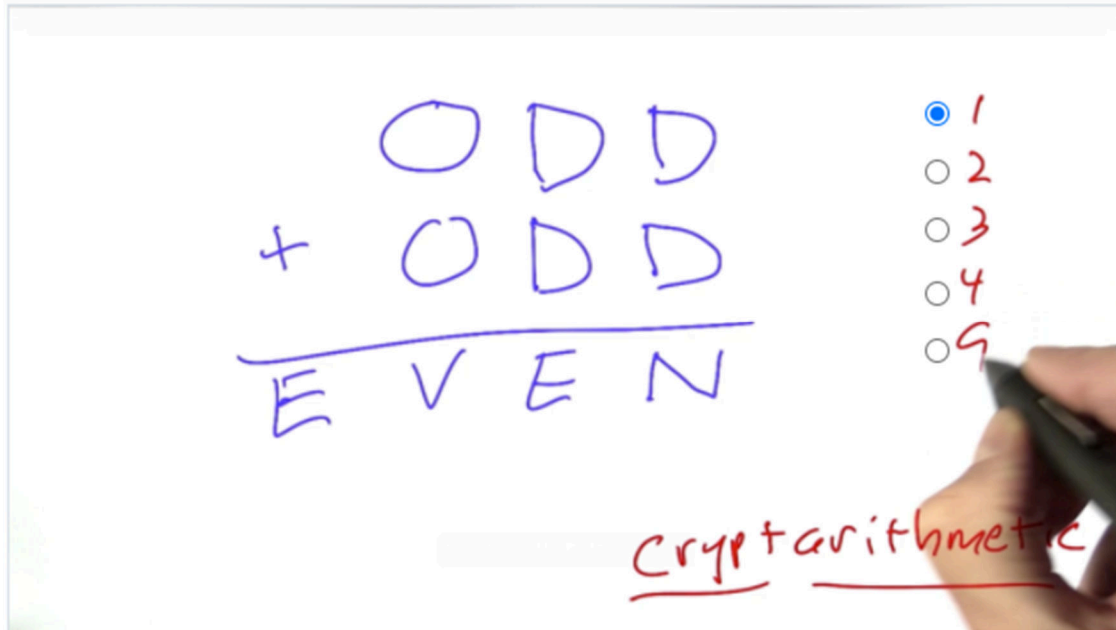
or end is None

ints(0)

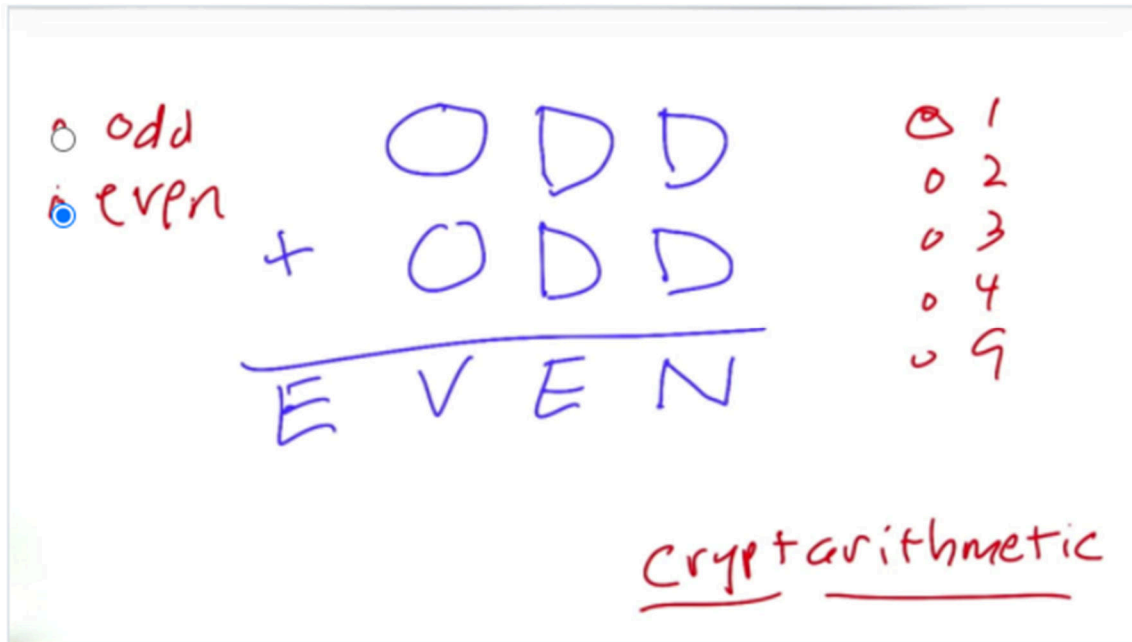
All Ints

```
14 ▾ def all_ints():  
15     "Generate integers in the order 0, +1, -1, +2, -2, +3, -3, ..."  
16     yield 0  
17 ▾     for i in ints(1):  
18         yield +i  
19         yield -i
```

Cryptarithmic



Odd or Even



Solving Cryptarithmic

```
1 def solve(formula):  
2     """Given a formula like 'ODD + ODD == EVEN', fill in digits to  
3     Input formula is a string; output is a digit-filled-in string  
4     for f in fill_in(formula):  
5         if valid(f):  
6             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

all values

- fill-in → list
- "return" to "print" in solve
- change valid
- add new function

Increasing Speed

QUIZ



A 90 B 10 100

Speedup B 10x

- ☐ 2x
- ☐ 5x
- ☐ 2x
- ☒ 1.1x

Rethinking Eval

eval (47/75) 63%

fewer calls
easier calls

easier: ☐ yes
☒ no

eval('short')
eval('EVEN')

divide and
conquer

Making Fewer Calls

eval (47/75) 63%

fewer calls
~~easier~~ calls

easier:
yes
no

- one big formula
- fill in one digit (zebra)
- eval formula once as a function with parameters

eval('100+011')
eval('EVEN')

divide and conquer

Compile Word

```
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 = []  
        for (i, d) in enumerate(word[::-1]):  
            terms.append('%s*%s' % (10**i, d))  
        return '(' + '+ '.join(terms) + ')'  
    else:  
        return word
```