Bourton-on-the-water



Bourton-on-the-water



town of 2000 people

Bourton-on-the-water's *1/9 model*



has a level-2 model...



has a level-2 model...



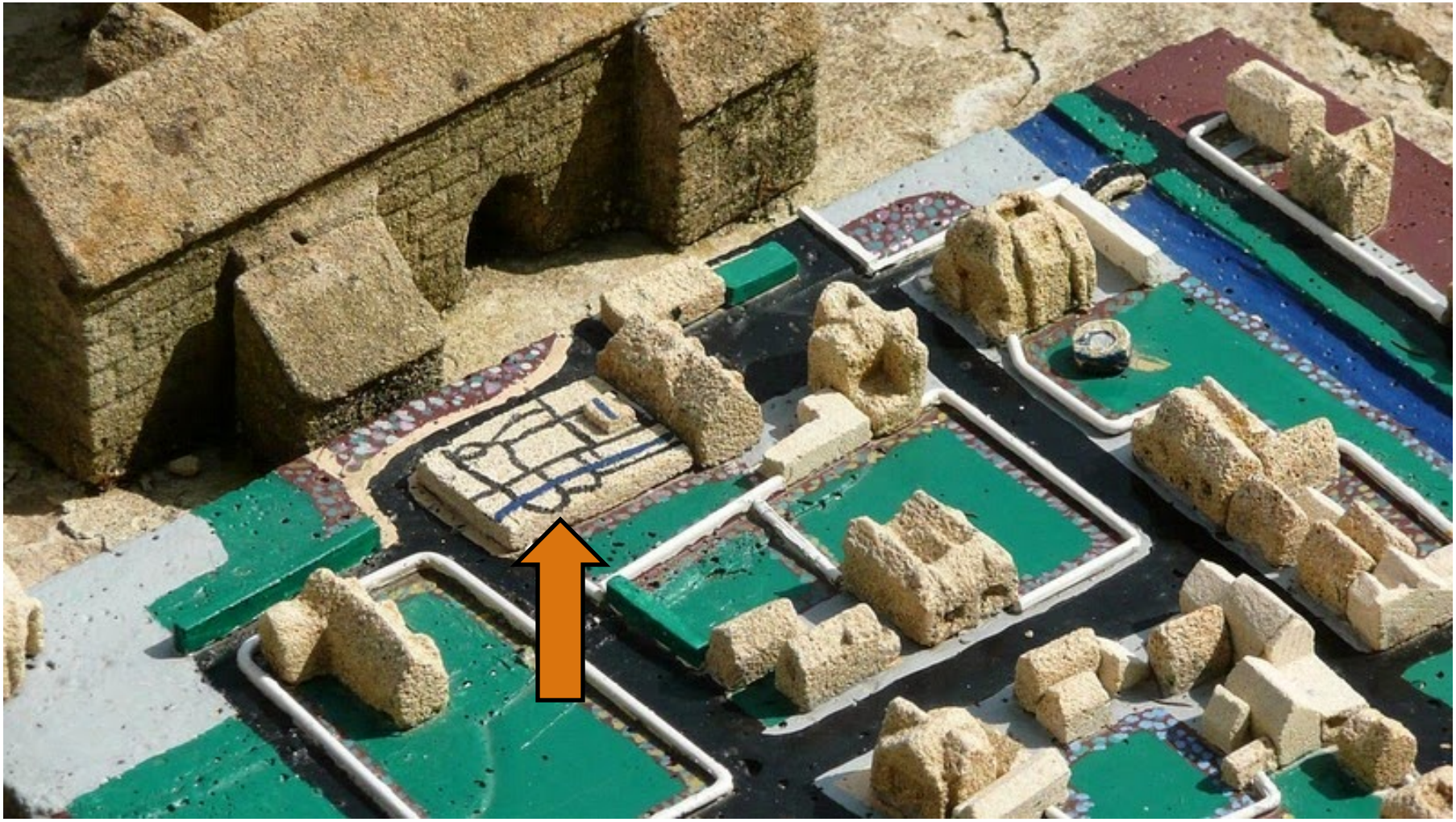
and a level-3 model…



and a level-3 model…



and even a (very small!) level-4 model



Functional Programming

```
>>> 'fun' in 'functional'
True
```

- Representation via list structures ***(data)***
- Leverage self-similarity ***(recursion)***
- Create small building blocks ***(functions)***

Composed together —to solve/investigate problems

Functional programming

conceptually concise **functional** vs. easiest *for the computer*… **procedural or sequential**

Data

```
[13, 14, 15]
```

```
[3, 4, 5, 6, 7, 8, 9]
```

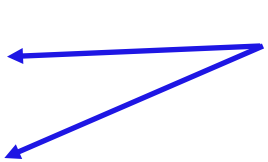
Functions

```
sum(…)
```

… and their compositions

sum range

```
def sum(L):
    """Argument: L, a list of #s
       Result: L's sum
    """
    if len(L) == 0:
        return 0
    else:
        return L[0] + sum(L[1:])
```



Base Case

Recursive Case

sum

list range

list(range(low,hi))



what's cookin' here?



step?

```
def range(low, high):
    """Arguments:  ints low and high
       Result: int list from low to high
    """
    if low >= high:
        return []
    else:
        return
```

excluding **high**

sum

list range

list(range(low,hi))



what's cookin' here?



step?

```
def range(low, high):
    """Arguments:  ints low and high
       Result: int list from low to high
    """
    if low >= high:
        return []
    else:
        return [low] + range(low+1, high)
```

excluding **high**

Data

```
[8, 9, 10]
```

Functions

```
sq()
```



```
[64, 81, 100]
```

```
[sq(x) for x in [8, 9, 10]]
```

...together

List Comprehensions

```
[2*x for x in [0, 1, 2, 3, 4, 5]]
```

List Comprehension

```
[0, 2, 4, 6, 8, 10]
```

result

What's the syntax saying here?



List Comprehensions

Expression to evaluate for each list element

Name for each list element

The **list** - or **string** to *run*

```
[2*x for x in [0, 1, 2, 3, 4, 5]]
```

List Comprehension

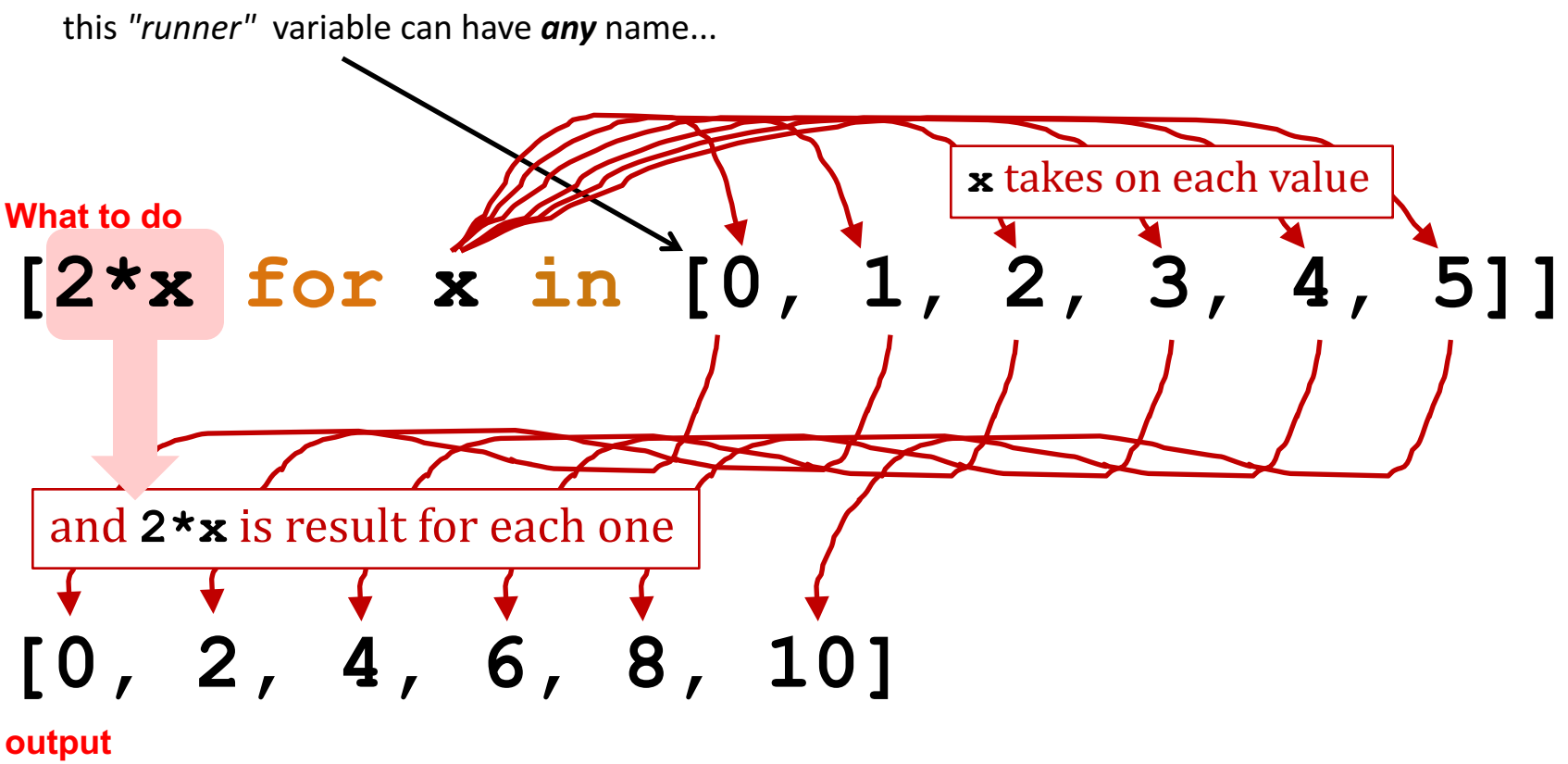
```
[0, 2, 4, 6, 8, 10]
```

result

What's the syntax saying here?



List Comprehensions



List Comprehensions

```
[10*x for x in [0, 1, 2, 3, 4, 5] if x%2 == 0]
```

result

```
[y*21 for y in list(range(0, 3))]
```

result

```
[s[1] for s in [”Hi", "Vassar!"]]
```

result

List Comprehensions

```
[10*x for x in [0, 1, 2, 3, 4, 5] if x%2 == 0]
```

```
[0, 20, 40]
```

result

```
[y*21 for y in range(0, 3)]
```

```
[0, 21, 42]
```

result

```
[s[1] for s in ["Hi", "Vassar!"]]
```

```
['i', 'a']
```

result

Quiz!

A **range** of list comprehensions...
Write Python's result for each L.C.:

```
[n**2 for n in range(0, 5)]
```
list

```
[42 for z in [0, 1, 2]]
```

```
[z for z in [0, 1, 2]]
```

```
[s[1::2] for s in ['aces', '54124!']]
```

```
[-7*b for b in range(-6, 6) if abs(b) > 4]
```
list

```
[a*(a-1) for a in range(8) if a % 2 == 1]
```
list

Got it. But what about that *name*?



Quiz!

A **range** of list comprehensions...
Write Python's result for each L.C.:

```
[n**2 for n in range(0, 5)]
```

[0, 1, 4, 9, 16]

```
[42 for z in [0, 1, 2]]
```

[42, 42, 42]

```
[z for z in [0, 1, 2]]
```

[0, 1, 2]

```
[s[1::2] for s in ['aces', '54124!']]
```

['cs', '42!]

[-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]

```
[-7*b for b in range(-6, 6) if abs(b) > 4]
```

[42, 35, -35]

[-6, -5, 5]

[0, 1, 2, 3, 4, 5, 6, 7]

```
[a*(a-1) for a in range(8) if a % 2 == 1]
```

[0, 6, 20, 42]

[1, 3, 5, 7]

Designing with LCs

write `[_____ for x in range(4)]`

list `[0, 1, 2, 3]`

result `[0, 14, 28, 42]`

write `[_____ for c in 'igetthis']`

result `[True, False, False, False, False, False, True, False]`

`[ 1, 0, 0, 0, 0, 0, 1, 0 ]`

And if we wanted the ints (in red)…?

Designing with LCs

write `[ x*14 for x in range(4)]`

list `[0, 1, 2, 3]`

result `[0, 14, 28, 42]`

write `[ c == 'i' for c in 'igetthis']`

result `[True, False, False, False, False, False, True, False]`

`[ 1, 0, 0, 0, 0, 0, 1, 0 ]`

And if we wanted the ints (in red)...? `int(c == 'i')`

Using LCs

Short and sweet!



[7, 8, 9]

```
def fun1(L):
    LC = [1 for x in L]
    return sum(LC)
```

```
def letScore(c):
    lots of ifs…
    return score
```

'twelve'

```
def fun2(S):
    LC = [letScore(c) for c in S]
    return sum(LC)
```

Using LCs

[7, 8, 9]

```
def len(L):
    LC = [1 for x in L]
    return sum(LC)
```

But ***one-liners*** are my specialty...



```
def letScore(c):
    lots of ifs...
    return score
```

'twelve'

```
def scrabbleScore(S):
    LC = [letScore(c) for c in S]
    return sum(LC)
```

Write each of these functions using list comprehensions...

Go!

Argument: **L**, any list of numbers
Result: the count of odd numbers in **L**
Example: nodds([3, 4, 5, 7, 42]) == 3

```
def nodds(L):
    LC = [                  for x in L                  ]
    return sum(LC)
```

Y are your #s **W** are the winning #s
Arguments: **Y** and **W**, two lists of "lottery" numbers (ints)
Result: the number of matches between **Y** & **W**
Example: lotto([5, 7, 42, 47], [3, 5, 7, 44, 47]) == 3

```
def lotto(Y, W):
    LC = [
    return sum(LC)
```

Argument: **N**, an int >= 2
Result: the number of positive divisors of **N**
Example: numdivs(12) == 6 (1, 2, 3, 4, 6, 12)

```
def ndivs(N):
    LC = [
    return sum(LC)
```

Write each of these functions using list comprehensions…

Go!

```
def nodds(L):
    LC = [1 for x in L if x%2 == 1]
    return sum(LC)
```

too terse?

[x % 2 for x in L]

```
def lotto(Y, W):
    LC = [                                ]
    return sum(LC)
```

```
def ndivs(N):
    LC = [                                ]
    return sum(LC)
```

Write each of these functions using list comprehensions…

Go!

```
def nodds(L):
    LC = [1 for x in L if x%2 == 1]
    return sum(LC)
```

too terse?

[x % 2 for x in L]

```
def lotto(Y, W):
    LC = [1 for y in Y if y in W]
    return sum(LC)
```

too terse?

[y in W for y in Y]

```
def ndivs(N):
    LC = [                          ]
    return sum(LC)
```

Write each of these functions using list comprehensions…

Go!

```
def nodds(L):
    LC = [1 for x in L if x%2 == 1]
    return sum(LC)
```

too terse?

[x % 2 for x in L]

```
def lotto(Y, W):
    LC = [1 for y in Y if y in W]
    return sum(LC)
```

too terse?

[y in W for y in Y]

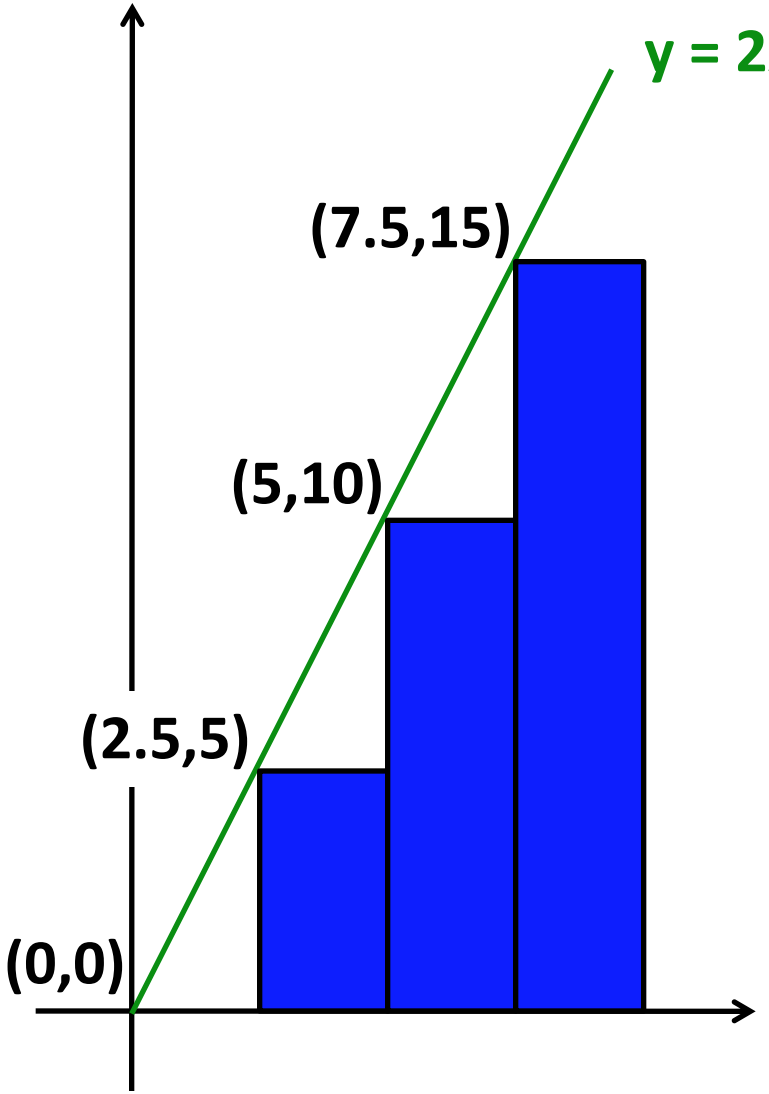
```
def ndivs(N):
    LC = [1 for x in range(1, N + 1) if N % x == 0]
    return sum(LC)
```

list

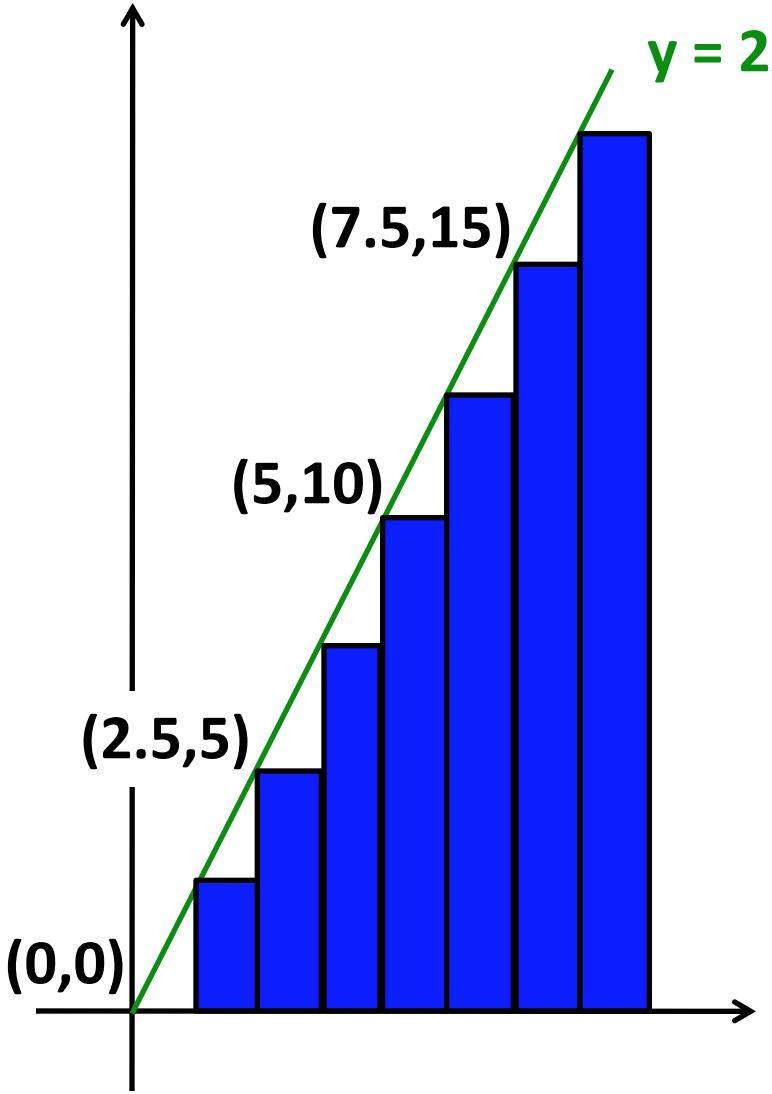
too terse?

[N % x == 0 for x in range(1, N + 1)]

exercise: *areas from rectangles*

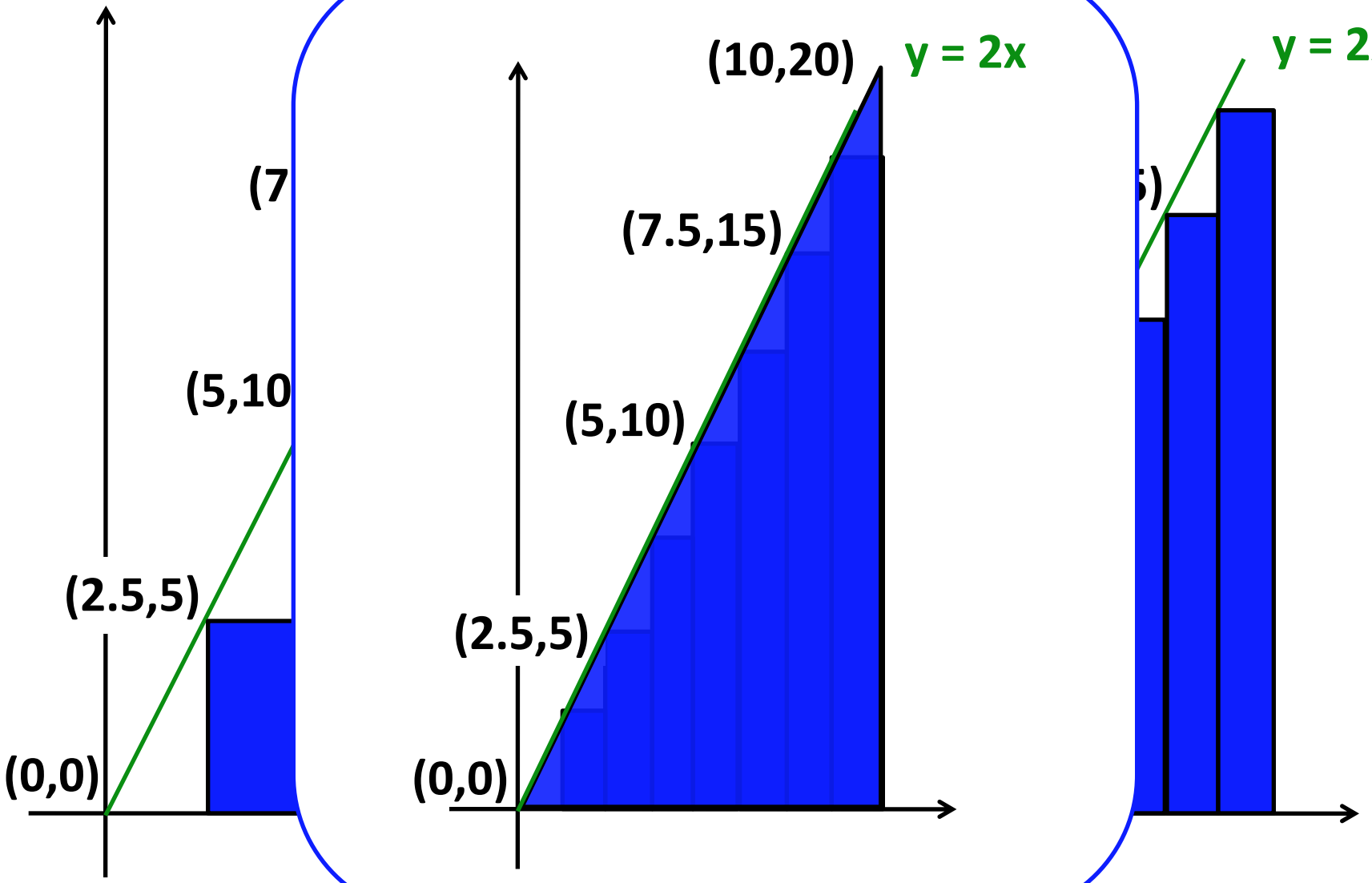


Areas of 4 rectangles



Areas of 8 rectangles

exercise: *areas from rectangles*



Area of N rectangles *in the limit*

Building blocks == CS!

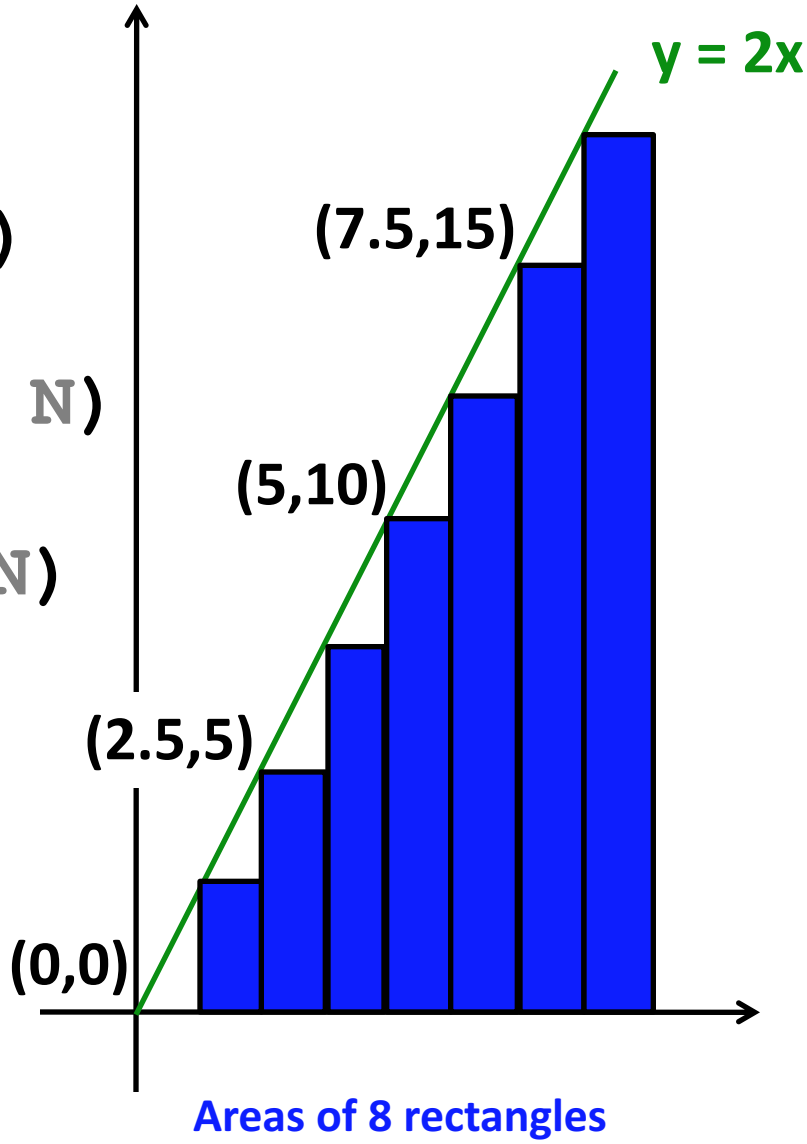
```
scaledfracs(low, hi, N)

f_of_fracs(f, low, hi, N)

integrate(f, low, hi, N)
```

only a few lines...

Where are the LCs?



Maya Lin, *Artist and Computer Scientist…*





One building block, deliberately applied, *over 50,000 times*…

Maya Lin, *Artist and Computer Scientist…*





"two-by-four landscape"

Maya Lin, *Architect…*





http://www.cc.com/video-clips/at64ut/drunk-history-maya-lin-designs-the-vietnam-veterans-memorial