Tree Recursion

https://pythonsandbox.com/code/pythonsandbox_u109615_nFrjtntu0wBBkr6FdfLipvZ1_v3.py

Announcements

Recursion Review

How to Know That a Recursive Implementation is Correct

Tracing: Diagram the whole computational process (only feasible for very small examples)

Induction: Check f(0), then check that f(n) is correct as long as f(n-1) ... f(0) are.

Abstraction: Assume f is correct (on simpler examples), then use it to implement f.

Streak (from Spring 2024's Midterm 1, A+ Question)

```
def streak(n):
    """Return True if all the digits in positive integer n are the same.

    >>> streak(22222)
    True
    >>> streak(4)
    True
    >>> streak(2222322)  # 2 and 3 are different digits.
    False
    """
    return (n >= 0 and n <= 9) or (n > 9 and _n % 10 == n // 10 % 10_ and streak(_n // 10_))
```

Idea: In a streak, all *pairs* of adjacent digits are equal

Hint: floor division //
(divides, discards the remainder)

```
>>> 1234 // 10
123
```

Hint: modulo %
(divides, returns the remainder)

```
>>> 1234 % 10
4
```

Mutual Recursion

Mutually Recursive Functions

Two functions f and g are mutually recursive if f calls g and g calls f.

```
def unique_prime_factors(n):
    """Return the number of unique prime factors of n.

    >>> unique_prime_factors(51)    # 3 * 17
    2
    >>> unique_prime_factors(27)    # 3 * 3 * 3
    1
    >>> unique_prime_factors(120)    # 2 * 2 * 2 * 3 * 5
    3
    """
    k = smallest_factor(n)
    def no_k(n):
        "Return the number of unique prime factors of n other than k."
        if n == 1:
            return 0
        elif n % k != 0:
            return unique_prime_factors(n)
        else:
            return no_k(n // k)
    return 1 + no_k(n)
```

```
def smallest_factor(n):
    "The smallest divisor of n above 1."
```

```
 27         120
 /\          /\
3 9         2  60
   /\           /\
  3  3         2  30
                  /\
                 2  15
                     /\
                    3  5
```

https://pythontutor.com/
render.html#code=def%20smallest_factor%28n%29%3A%0A%20%20%20%20if%20%28n%252%20%3D%3D%200%29%3A%0A%20%20%20%20%20%20%20%20return%202%0A%20%20%20%20k%20%3D%203%0A%20%20%20%20while%20%28k%20%3C%20n%29%3A%0A%20%20%20%20%20%20%20%20if%20%28n%25k%20%3D%3D%200%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20k%0A%20%20%20%20%20%20%20%20k%20%2B%3D%201%0A%20%20%20%20return%20n%0A%0A%0Adef%20unique_prime_factors%28n%29%3A%0A%20%20%20%20%22%22%22Return%20the%20number%20of%20unique%20prime%2
0factors%20of%20n.
%0A%0A%20%20%20%20%3E%3E%3E%20unique_prime_factors%2851%29%20%20%20%23%203%20*%2017%0A%20%20%20%202%0A%20%20%20%20%3E%3E%3E%20unique_prime_factors%2827%29%20%20%20%23%203%20*%203%20*%203%0A%20%20%20%201%0A%20%20%20%20%3E%3E%3E%20unique_prime_factors%2860%29%20%23%202%20*%202%20*%202%20*%203%0A%20%20%20%202%0A%20%20%20%20%22%22%22%0A%20%20%20%20k%20%3D%20smallest_factor%28n%29%0A%20%20%20%20def%20no_k%28n%29%3A%0A%20%20%20%20%20%20%20%20%22Return%20the%20number%20of%20unique%20prime%20fac
tors%20of%20n%20other%20than%20k.%22%0A%20%20%20%20%20%20%20%20if%20n%20%3D%3D%201%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%200%0A%20%20%20%20%20%20%20%20elif%20n%20%25%20k%20!%3D%200%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20unique_prime_factors%28n%29%0A%20%20%20%20%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20no_k%28n//
k%29%0A%20%20%20%20return%201%2Bno_k%28n%29%0A%0Aunique_prime_factors%2860%29&cumulative=false&curInstr=0&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false

Mutually Recursive Functions

Two functions f and g are mutually recursive if f calls g and g calls f.

```
def unique_prime_factors(n):
    """Return the number of unique prime factors of n.

    >>> unique_prime_factors(51)   # 3 * 17
    2
    >>> unique_prime_factors(27)   # 3 * 3 * 3
    1
    >>> unique_prime_factors(120)   # 2 * 2 * 2 * 3 * 5
    3
    """
    k = smallest_factor(n)
    def no_k(n):
        "Return the number of unique prime factors of n other than k."
        if n == 1:
            return 0
        elif n % k != 0:
            return unique_prime_factors(n)
        else:
            return no_k(n // k)
    return 1 + no_k(n)
```

```
def smallest_factor(n):
    "The smallest divisor of n above 1."
```

https://pythontutor.com/
render.html#code=def%20smallest_factor%28n%29%3A%0A%20%20%20%20if%20%28n%252%20%3D%3D%200%29%3A%0A%20%20%20%20%20%20%20%20return%202%0A%20%20%20%20k%20%3D%203%0A%20%20%20%20while%20%28k%20%3C%20n%29%3A%0A%20%20%20%20%20%20%20%20if%20%28n%25k%20%3D%3D%200%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20k%0A%20%20%20%20%20%20%20%20k%20%2B%3D%201%0A%20%20%20%20return%20n%0A%0A%0Adef%20unique_prime_factors%28n%29%3A%0A%20%20%20%20%22%22%22Return%20the%20number%20of%20unique%20prime%2
0factors%20of%20n.
%0A%0A%20%20%20%20%3E%3E%3E%20unique_prime_factors%2851%29%20%20%23%203%20*%2017%0A%20%20%20%202%0A%20%20%20%20%3E%3E%3E%20unique_prime_factors%2827%29%20%20%23%203%20*%203%20*%203%0A%20%20%20%201%0A%20%20%20%20%3E%3E%3E%20unique_prime_factors%2860%29%20%23%202%20*%202%20*%202%20*%203%0A%20%20%20%202%0A%20%20%20%20%22%22%22%0A%20%20%20%20k%20%3D%20smallest_factor%28n%29%0A%20%20%20%20def%20no_k%28n%29%3A%0A%20%20%20%20%20%20%20%20%22Return%20the%20number%20of%20unique%20prime%20fac
tors%20of%20n%20other%20than%20k.%22%0A%20%20%20%20%20%20%20%20if%20n%20%3D%3D%201%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%200%0A%20%20%20%20%20%20%20%20elif%20n%20%25%20k%20!%3D%200%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20unique_prime_factors%28n%29%0A%20%20%20%20%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20no_k%28n//
k%29%0A%20%20%20%20return%201%2Bno_k%28n%29%0A%0Aunique_prime_factors%2860%29&cumulative=false&curInstr=0&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false

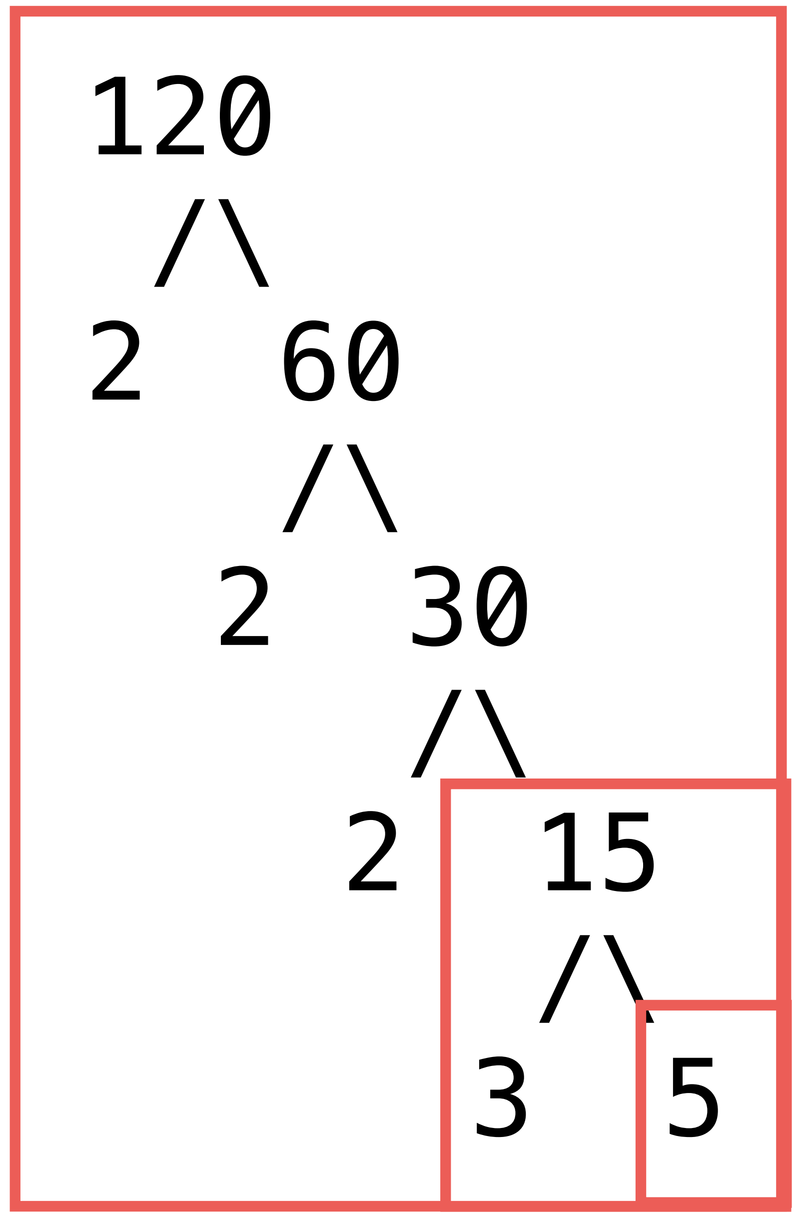
Mutually Recursive Functions

Two functions f and g are mutually recursive if f calls g and g calls f.

```
def unique_prime_factors(n):
    """Return the number of unique prime factors of n.

    >>> unique_prime_factors(51)    # 3 * 17
    2
    >>> unique_prime_factors(27)    # 3 * 3 * 3
    1
    >>> unique_prime_factors(120)    # 2 * 2 * 2 * 3 * 5
    3
    """
    k = smallest_factor(n)
    def no_k(n):
        "Return the number of unique prime factors of n other than k."
        if n == 1:
            return 0
        elif n % k != 0:
            return unique_prime_factors(n)
        else:
            return no_k(n // k)
    return 1 + no_k(n)

def smallest_factor(n):
    "The smallest divisor of n above 1."
```



https://pythontutor.com/
render.html#code=def%20smallest_factor%28n%29%3A%0A%20%20%20%20if%20%28n%252%20%3D%3D%200%29%3A%0A%20%20%20%20%20%20%20%20return%202%0A%20%20%20%20k%20%3D%203%0A%20%20%20%20while%20%28k%20%3C%20n%29%3A%0A%20%20%20%20%20%20%20%20if%20%28n%25k%20%3D%3D%200%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20k%0A%20%20%20%20%20%20%20%20k%20%2B%3D%201%0A%20%20%20%20return%20n%0A%0A%0Adef%20unique_prime_factors%28n%29%3A%0A%20%20%20%20%22%22%22Return%20the%20number%20of%20unique%20prime%2
0factors%20of%20n.
%0A%0A%20%20%20%20%3E%3E%3E%20unique_prime_factors%2851%29%20%20%23%203%20*%2017%0A%20%20%20%202%0A%20%20%20%20%3E%3E%3E%20unique_prime_factors%2827%29%20%20%20%23%203%20*%203%20*%203%0A%20%20%20%201%0A%20%20%20%20%3E%3E%3E%20unique_prime_factors%28120%29%20%23%202%20*%202%20*%202%20*%203%20*%205%0A%20%20%20%203%0A%20%20%20%20%22%22%22%0A%20%20%20%20k%20%3D%20smallest_factor%28n%29%0A%20%20%20%20def%20no_k%28n%29%3A%0A%20%20%20%20%20%20%20%20%22Return%20the%20number%20of%20unique%20pr
ime%20factors%20of%20n%20other%20than%20k.%22%0A%20%20%20%20%20%20%20%20if%20n%20%3D%3D%201%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%200%0A%20%20%20%20%20%20%20%20elif%20n%20%25%20k%20!%3D%200%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20unique_prime_factors%28n%29%0A%20%20%20%20%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20no_k%28n//
k%29%0A%20%20%20%20return%201%2Bno_k%28n%29%0A%0Aunique_prime_factors%2860%29&cumulative=false&curInstr=0&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false

Tree Recursion

Counting Partitions

The number of *partitions* of a positive integer ***n***, using parts up to size ***m***, is the number of ways in which n can be expressed as the sum of positive integer parts up to m.

```
count_partitions(6, 4)
```

2 + 4 = 6

1 + 1 + 4 = 6

3 + 3 = 6

1 + 2 + 3 = 6

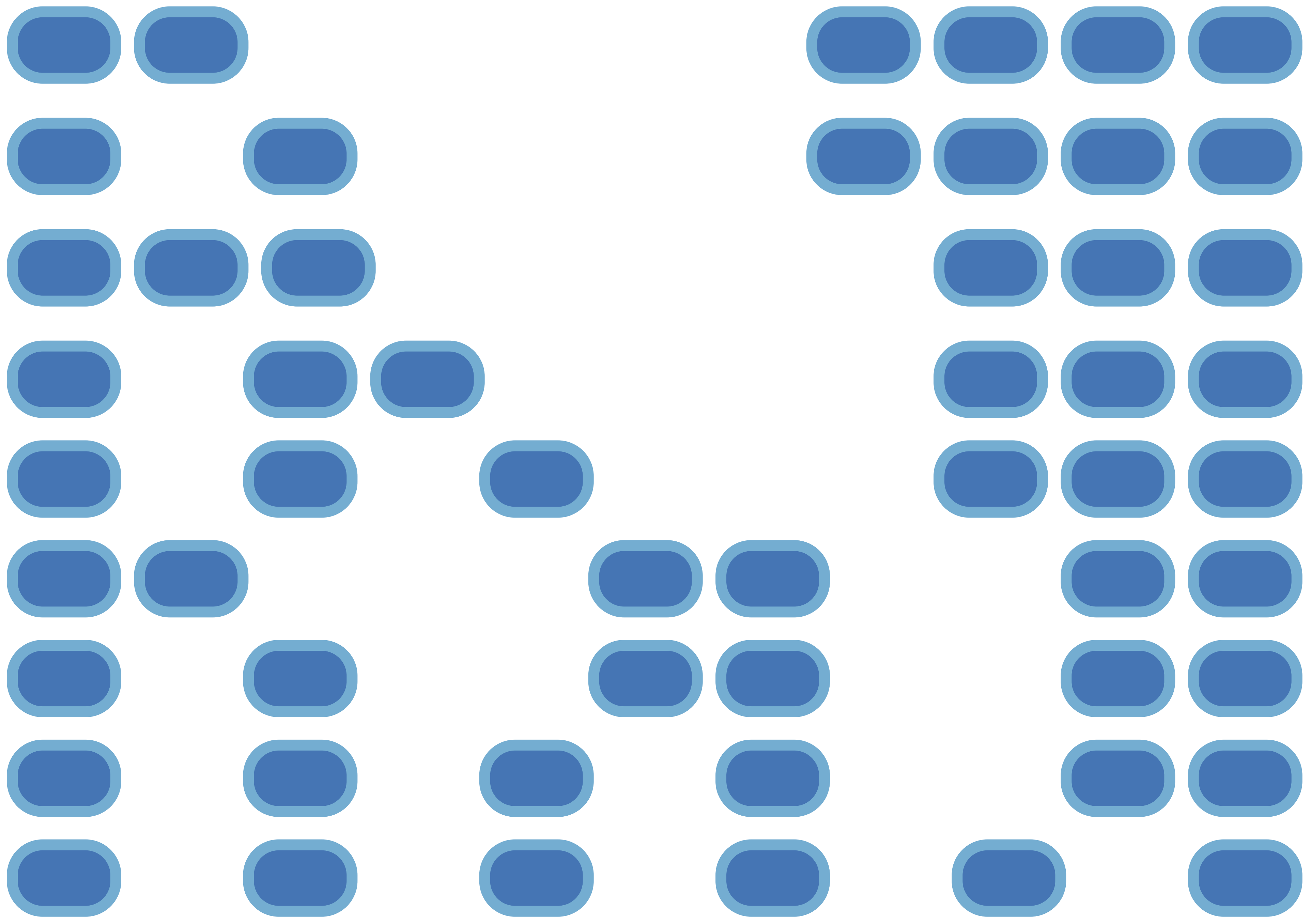
1 + 1 + 1 + 3 = 6

2 + 2 + 2 = 6

1 + 1 + 2 + 2 = 6

1 + 1 + 1 + 1 + 2 = 6

1 + 1 + 1 + 1 + 1 + 1 = 6

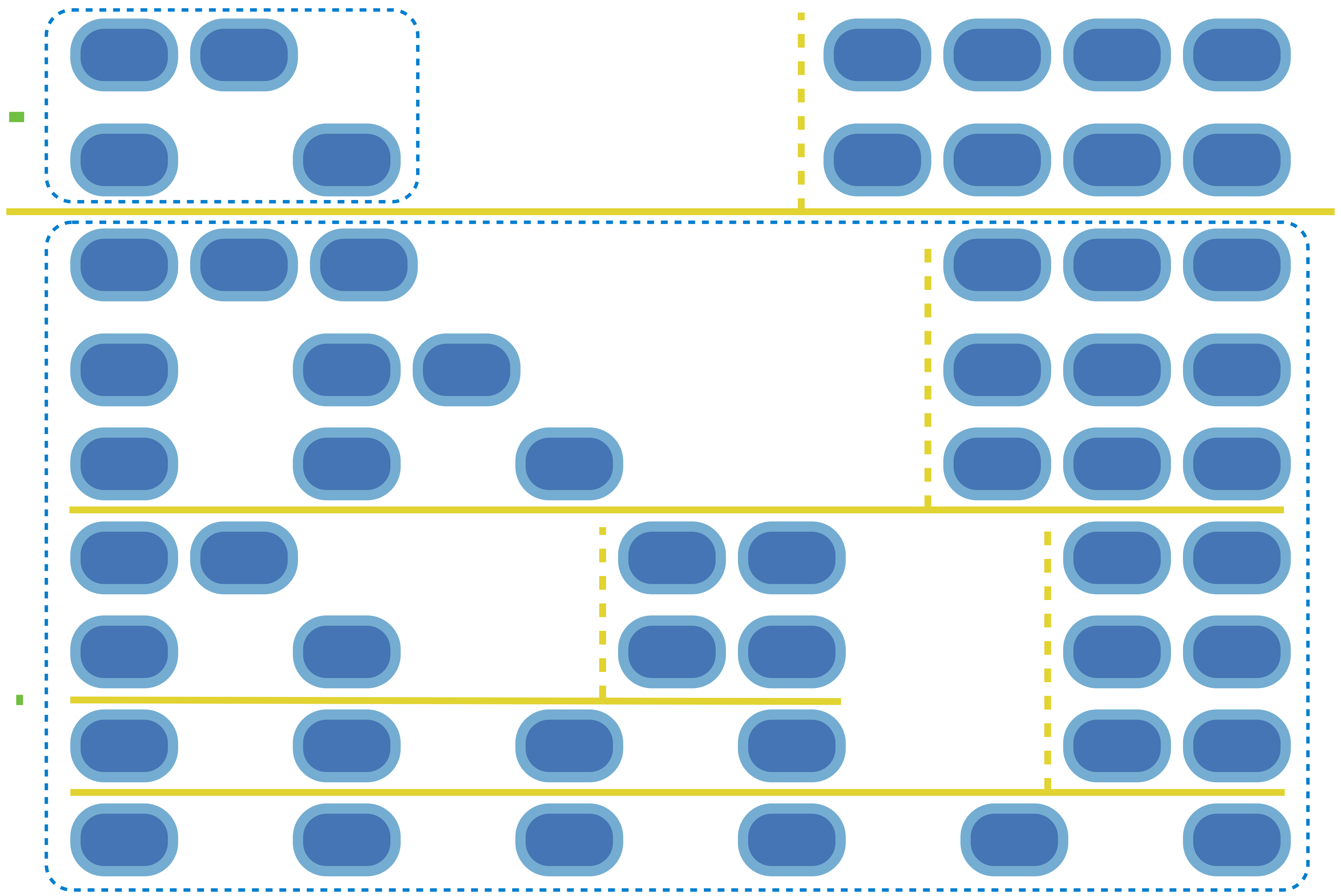


Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m.

count_partitions(6, 4)

- Recursive decomposition: finding simpler instances of the problem.
- Explore two possibilities:
  - Use at least one 4
  - Don't use any 4
- Solve two simpler problems:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.



Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m.

- Recursive decomposition: finding simpler instances of the problem.
- Explore two possibilities:
  - Use at least one 4
  - Don't use any 4
- Solve two simpler problems:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.

```
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0

    else:
        with_m = count_partitions(n-m, m)
        without_m = count_partitions(n, m-1)
        return with_m + without_m
```

(Demo)

pythontutor.com/composingprograms.html#code=def%20count_partitions%28n,%20m%29%3A%0A%20%20%20%20if%20n%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20return%201%0A%20%20%20%20elif%20n%20<%200%3A%0A%20%20%20%20%20%20%20%20return%200%0A%20%20%20%20elif%20m%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20return%200%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20with_m%20%3D%20count_partitions%28n-m,
%20m%29%20%0A%20%20%20%20%20%20%20%20without_m%20%3D%20count_partitions%28n,
%20m-1%29%0A%20%20%20%20%20%20%20%20return%20with_m%20%2B%20without_m%0A%20%20%20%20%20%20%20%20%0Aresult%20%3D%20count_partitions%285,%203%29%0A%0A#%201%20%2B%201%20%2B%201%20%2B%201%20%2B%201%20%3D%205%0A#%201%20%2B%201%20%2B%201%20%2B%202%20%2B%20%20%20%3D%205%0A#%201%20%2B%202%20%2B%202%20%2B%20%20%20%20%20%20%20%3D%205%0A#%201%20%2B%201%20%2B%203%20%2B%20%20%20%20%20%20%20%3D%205%0A
20%2B%203%20%2B%20%20%20%20%20%20%20%20%20%20%20%20%3D%205&mode=display&origin=composingprograms.js&cumulative=false&py=3&rawInputLstJSON=[]&curInstr=0

Spring 2023 Midterm 2 Question 5

Definition. When parking vehicles in a row, a motorcycle takes up 1 parking spot and a car takes up 2 adjacent parking spots. A string of length n can represent n adjacent parking spots using % for a motorcycle, <> for a car, and . for an empty spot.

For example: '.%%.<><>' (Thanks to the Berkeley Math Circle for introducing this question.)

Implement **count_park**, which returns the number of ways that vehicles can be parked in n adjacent parking spots for positive integer n. Some or all spots can be empty.

```
def count_park(n):
    """Count the ways to park cars and motorcycles in n adjacent spots.
    >>> count_park(1)  # '.' or '%'
    2
    >>> count_park(2)  # '..', '.%', '%.', '%%', or '<>'
    5
    >>> count_park(4)  # some examples: '<><>', '.%%.', '%<>%', '%.<>'
    29
    """
    if n < 0:
        return ____0_____
    elif n == 0:
        return ____1_____
    else:
        return count_park(n-2) + count_park(n-1) + count_park(n-1)
```