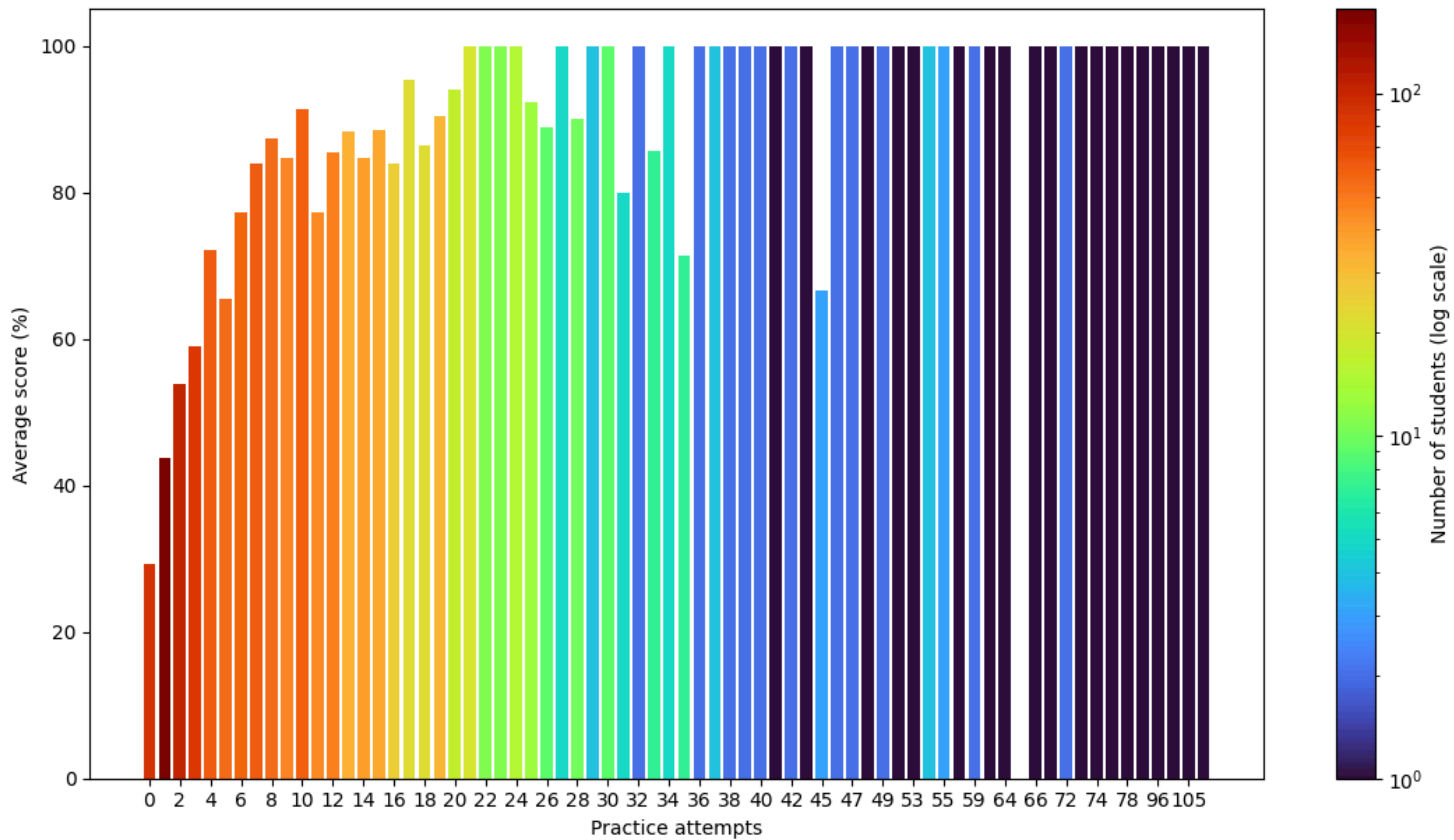


Midterm 2 Review

Announcements



Linked List Practice

Example: Above First

```
def above_first(s: LinkedList):
```

```
    """ Print all of the labels of s that are larger than first.
```

```
>>> above_first(Link(3, Link(0, Link(8, Link(5))))
```

```
8
```

```
5
```

```
"""
```

```
def process(t: LinkedList):
```

```
    _____:
```

```
        if _____ > _____:
```

```
            print(_____)
```

```
        _____
```

```
process(s)
```

Example: Above First

```
def above_first(s: LinkedList):
```

```
    """ Print all of the labels of s that are larger than first.
```

```
>>> above_first(Link(3, Link(0, Link(8, Link(5)))))
```

```
8
```

```
5
```

```
"""
```

```
def process(t: LinkedList):
```

```
    if isinstance(t, Link) :
```

```
        if t.first > s.first :
```

```
            print(t.first)
```

```
            process(t.rest)
```

```
process(s)
```

Example: Above First, Take 2

```
def above_first(s: LinkedList) -> LinkedList:
    """ Return a new list with only the items in s larger than first.

    >>> above_first(Link(3, Link(5, Link(1))))
    (5)
    >>> above_first(Link(3, Link(0, Link(8, Link(5)))))
    (8 5)
    """
```

Example: Above First, Take 2

```
def above_first(s: LinkedList) -> LinkedList:
    """ Return a new list with only the items in s larger than first.

>>> above_first(Link(3, Link(5, Link(1))))
(5)
>>> above_first(Link(3, Link(0, Link(8, Link(5)))))
(8 5)
"""

def process(t: LinkedList) -> LinkedList:
    if not isinstance(t, Link):
        return ()
    if t.first > s.first:
        return Link(t.first, process(t.rest))
    return process(t.rest)
return process(s)
```

List Practice

Example: Closest to Index

```
def closest_to_index(s):  
    """ Return the item in non-empty list s whose value has the  
        smallest absolute difference from its index.  
  
    >>> closest_to_index([8, 3, 10])  
    3  
    >>> closest_to_index([-3, 2, 100])  
    2  
    """  
  
    smallest = 0  
    for                                 :  
        if                                 :  
                                              
  
    return                                 
```

Example: Closest to Index

```
def closest_to_index(s):  
    """ Return the item in non-empty list s whose value has the  
        smallest absolute difference from its index.  
  
    >>> closest_to_index([8, 3, 10])  
    3  
  
    >>> closest_to_index([-3, 2, 100])  
    2  
    """  
  
    smallest = 0  
    for i in range(len(s)):  
        if abs(i - s[i]) < abs(smallest - s[smallest]):  
            smallest = i  
    return s[smallest]
```

Example: Closest to Index

```
def closest_to_index(s):  
    """ Return the item in non-empty list s whose value has the  
        smallest absolute difference from its index.  
  
    >>> closest_to_index([4, 2, 10])  
    2  
    >>> closest_to_index([-1, 100])  
    -1  
    """  
    return _____
```

Example: Closest to Index

```
def closest_to_index(s):  
    """ Return the item in non-empty list s whose value has the  
        smallest absolute difference from its index.  
  
    >>> closest_to_index([4, 2, 10])  
    2  
    >>> closest_to_index([-1, 100])  
    -1  
    """  
    return s[min(range(len(s)), key=lambda i: abs(i - s[i]))]
```

Recursion and Strings

Spring 2023 Midterm 2 Question 5(a) [modified a bit]

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.

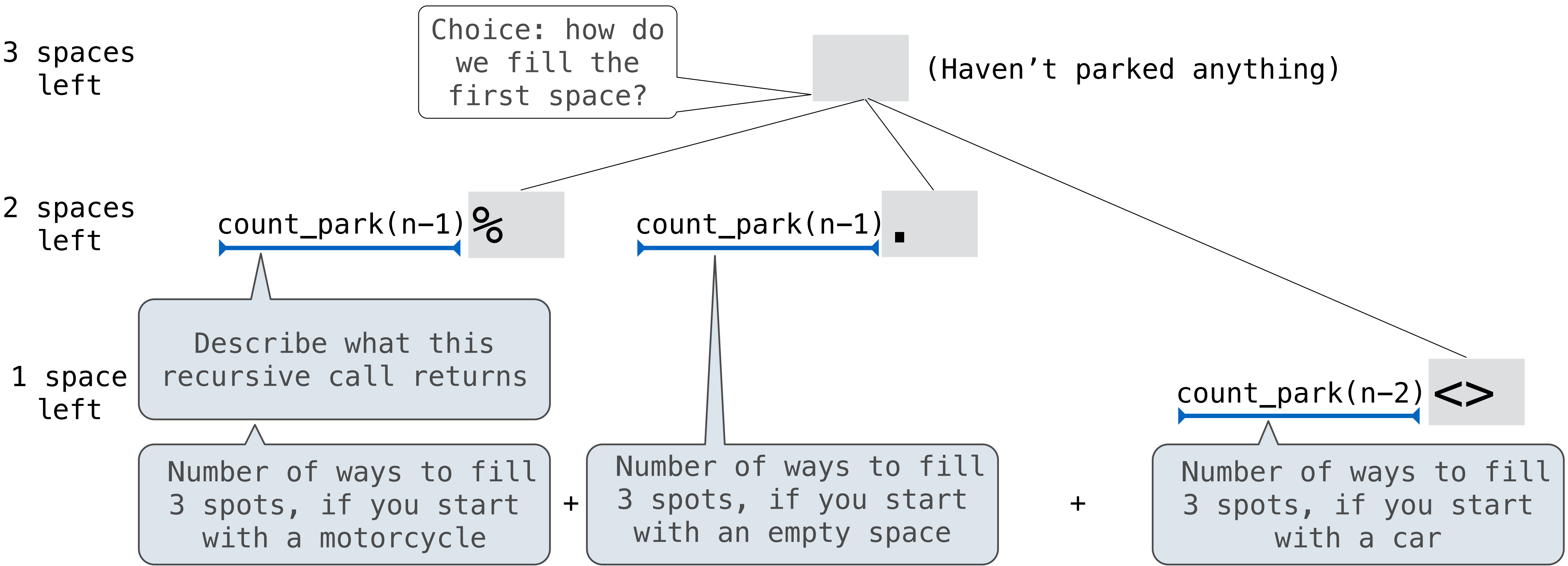
For example: `'.%%.<><>'` (Thanks to the Berkeley Math Circle for introducing this question.)

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  
"""
```

We haven't parked anything yet. What's a first choice we can make?

Spring 2023 Midterm 2 Question 5(a) [modified a bit]



Spring 2023 Midterm 2 Question 5(a) [modified a bit]

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.)

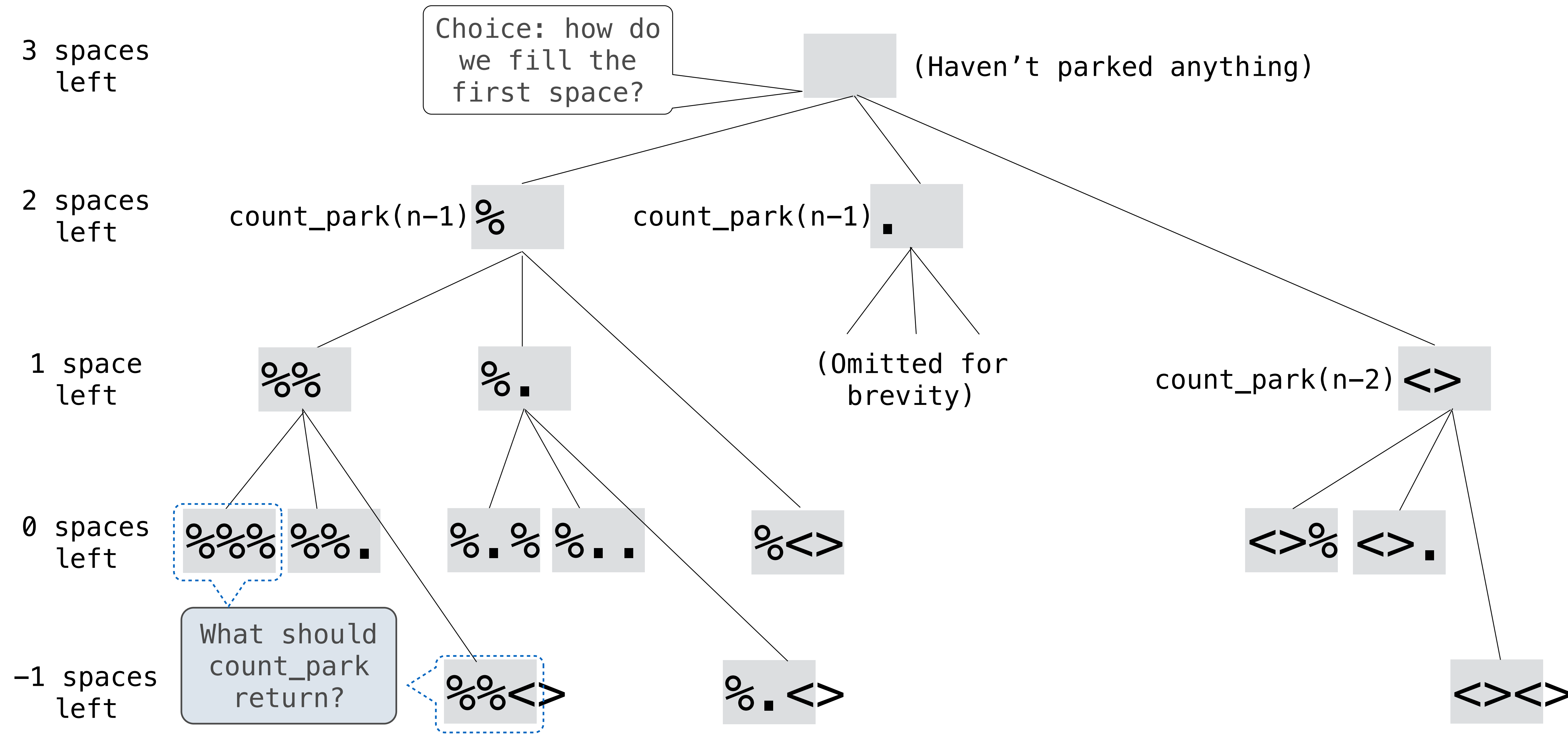
Implement **count_park**, which returns the number of ways that vehicles can be parked in n adjacent parking spots for positive integer n . Some or all spots can be empty.

```
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 _____  
    elif n == 0:  
        return _____  
    else:  
        return count_park(n-1) + count_park(n-1) + count_park(n-2)
```

One way to think about these base cases:
Number of ways to fill 3 spots, if you start with a motorcycle lead to these cases, their values be?

```
count_park(3):  
    %%%  
    %%.  
    %.%  
    %..  
    %<>  
    ---  
    .%%  
    .%.  
    ..%  
    ...  
    .<>  
    ---  
    <>%  
    <>.
```

Spring 2023 Midterm 2 Question 5(a) [modified a bit]



Spring 2023 Midterm 2 Question 5(a) [modified a bit]

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.)

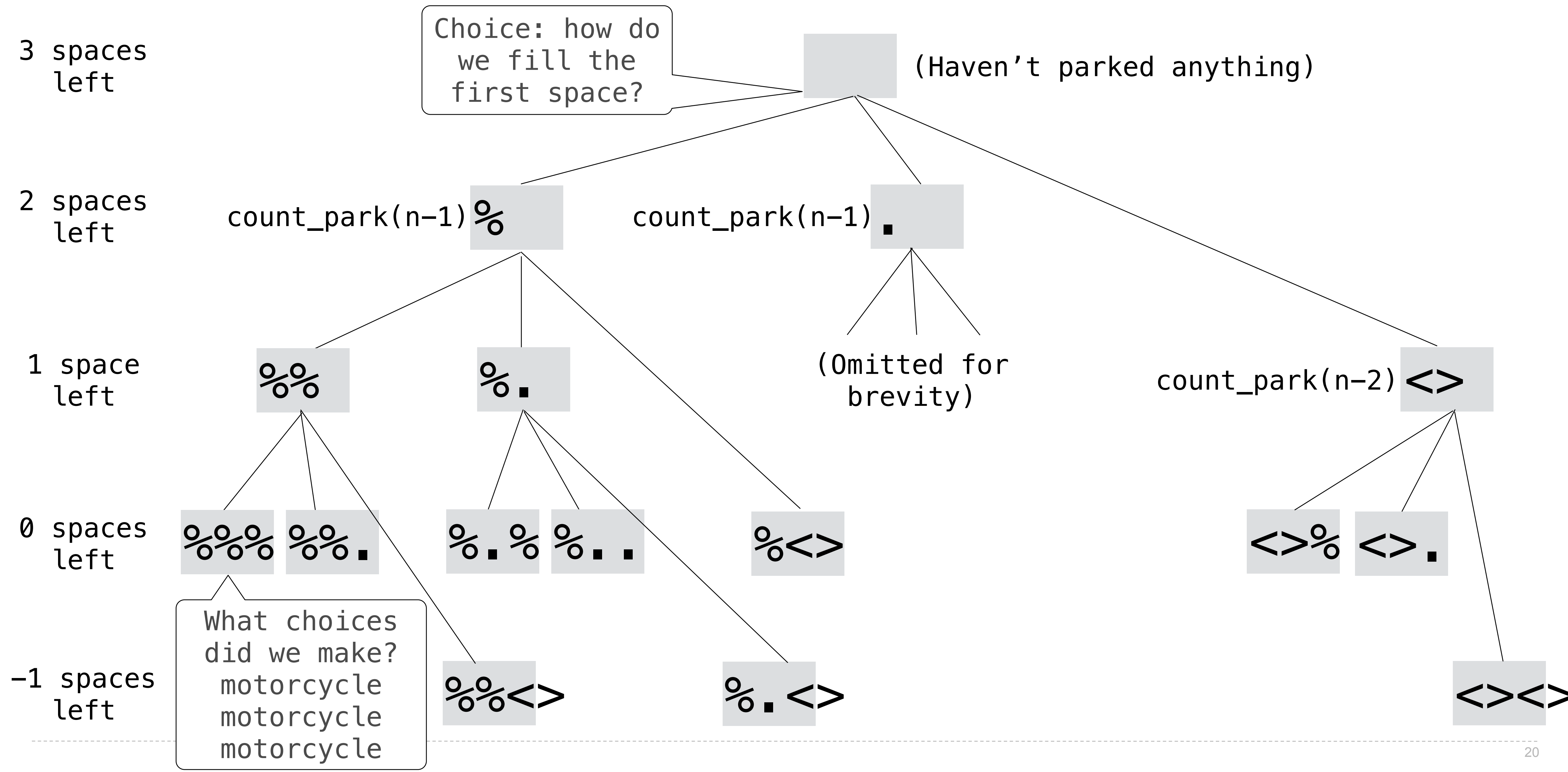
Implement **count_park**, which returns the number of ways that vehicles can be parked in n adjacent parking spots for positive integer n . Some or all spots can be empty.

```
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-1) + count_park(n-1) + count_park(n-2)
```

One way to think about these base cases:
which recursive calls lead to these cases,
and what should their values be?

```
count_park(3):  
    %%%  
    %%.  
    %.%  
    %..  
    %<>  
    ---  
    .%%  
    .%.  
    ..%  
    ...  
    .<>  
    ---  
    <>%  
    <>.
```

Spring 2023 Midterm 2 Question 5(a) [modified a bit]



Recursion so far

double_eights(s: list[int]) -> bool:

Strategy: Check if the first two elements are both 8s
Call double_eights on everything except the first element

streak(n: int) -> bool:

Return whether n is a dice integer in which all digits the same

Strategy: Check if last digit is a dice integer, and matches the previous
Call streak on everything except the last digit

Deal with one item or digit; recurse for the rest

reverse(s: list) -> list:

Strategy: Get the first element into place
Call reverse on the rest

count_partitions(n: int, m: int) -> int:

Return how many ways we can count to n, using pieces of up to size m

Strategy: Use a piece of size m; recurse for the rest
Don't use any pieces of size m; recurse for the rest

Tree recursion:
Make a SMALL choice; recurse

park(n: int) -> int: Return the ways to park in n adjacent spots

Strategy: Use a motorcycle; recurse for the rest
Use nothing; recurse for the rest
Use a car; recurse for the rest

Spring 2023 Midterm 2 Question 5(b) [modified a lot]

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 **park**, which returns a list of all the ways, represented as strings, that vehicles can be parked in n adjacent parking spots for positive integer n . Spots can be empty.

```
def park(n):  
    """Return the ways to park cars and motorcycles in n adjacent spots.  
    >>> park(1)  
    ['%', '.']  
    >>> park(2)  
    ['%%', '%.', '.%', '..', '<>']  
    >>> len(park(4)) # some examples: '<><>', '.%%.', '%<>%', '%.<>'  
    29  
    """  
    if n < 0:  
        return []  
    elif n == 0:  
        return ['']  
    else:  
        return
```

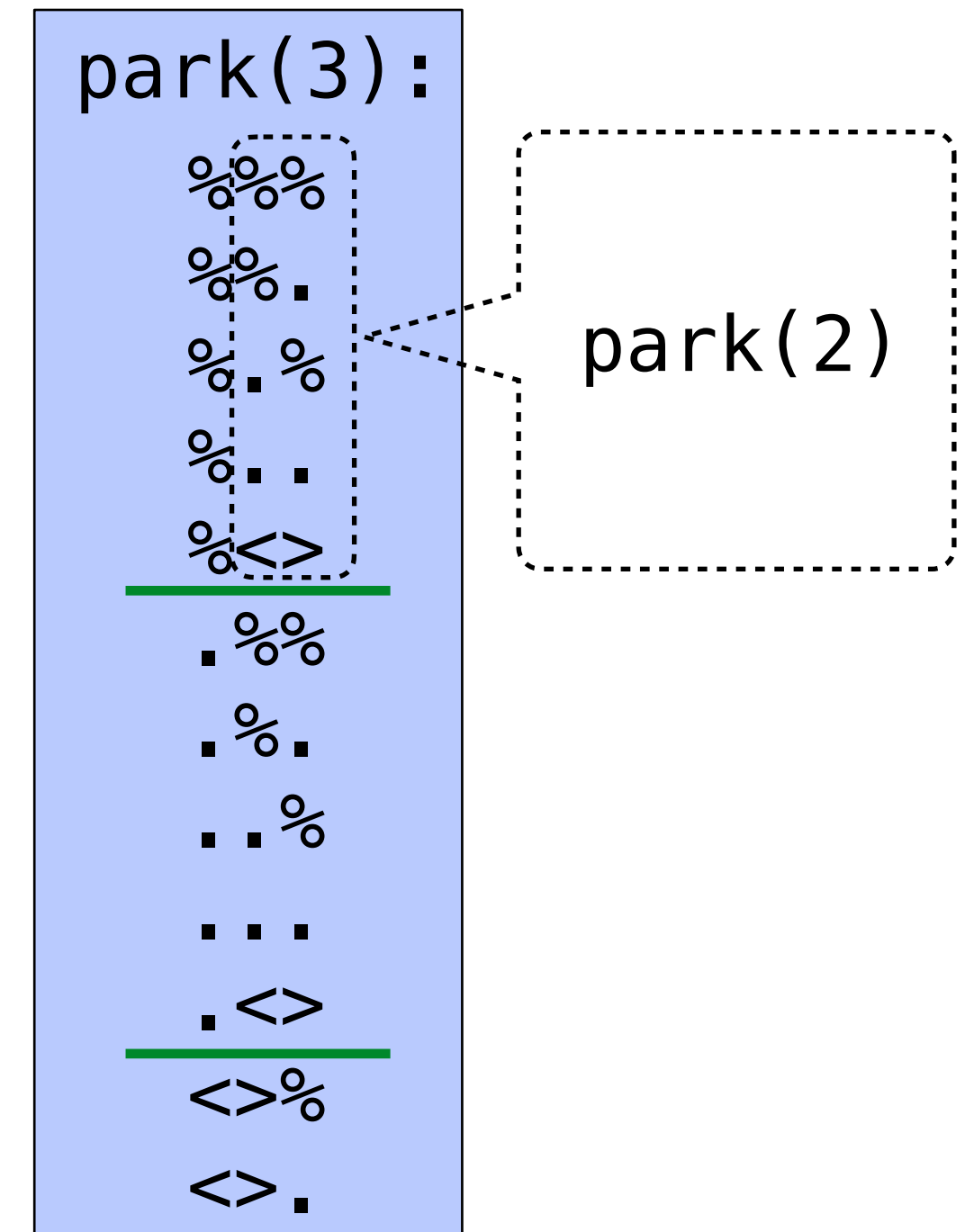
motorcycle first

+

nothing first

+

car first



Spring 2023 Midterm 2 Question 5(b) [modified a lot]

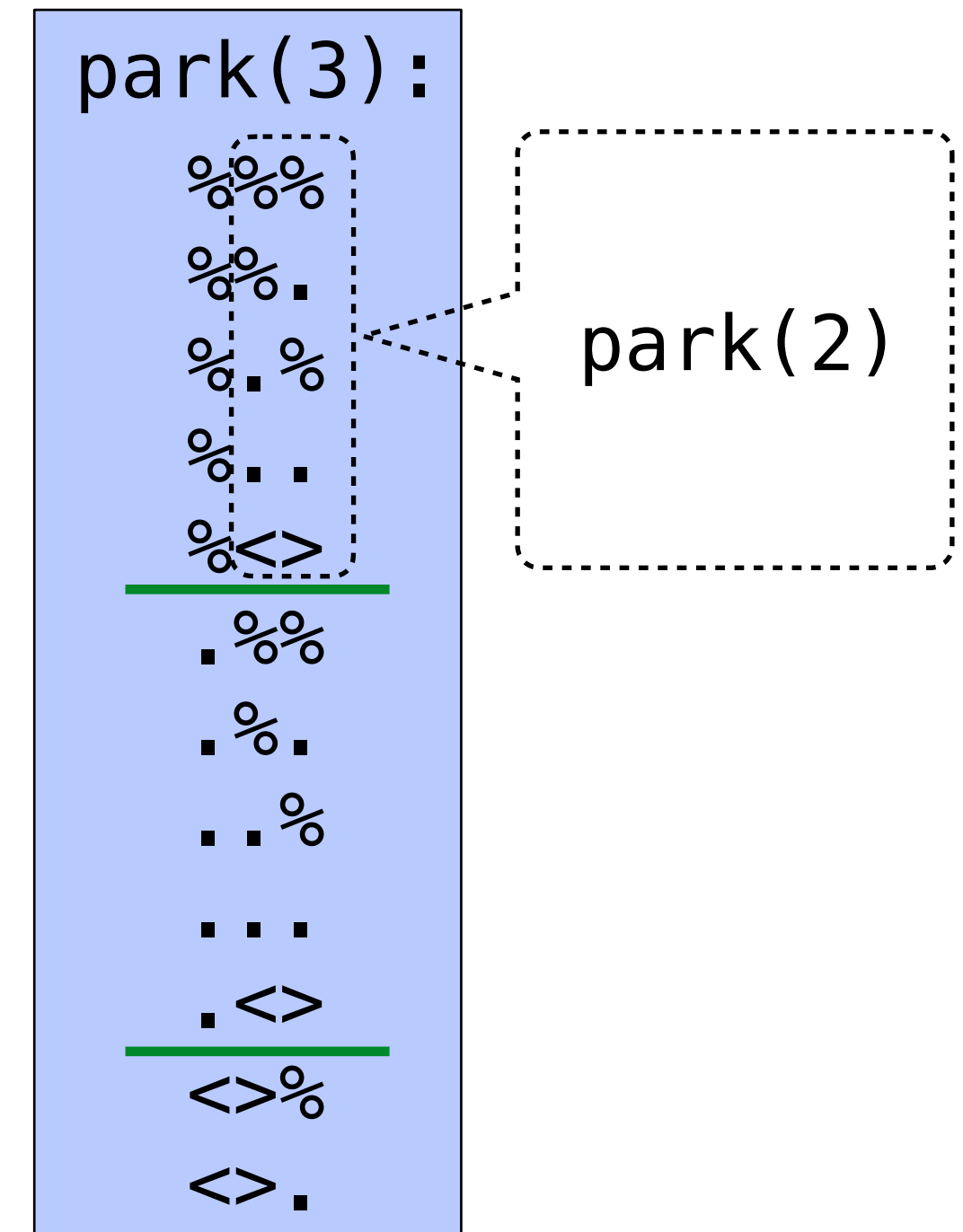
Definition. When parking vehicles in a row, a motorcycle takes up 1 parking spot and a car takes up 2 adjacent parking spots. A string of length n can represent n adjacent parking spots using % for a motorcycle, <> for a car, and . for an empty spot.

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

Implement **park**, which returns a list of all the ways, represented as strings, that vehicles can be parked in n adjacent parking spots for positive integer n . Spots can be empty.

```
def park(n):  
    """Return the ways to park cars and motorcycles in n adjacent spots.  
    >>> park(1)  
    ['%', '.']  
    >>> park(2)  
    ['%%', '%.', '.%', '..', '<>']  
    >>> len(park(4)) # some examples: '<><>', '%.%%.', '%<>%', '%.<>'  
    29  
    """  
    if n < 0:  
        return []  
    elif n == 0:  
        return ['']  
    else:  
        return ['%' + s for s in park(n-1)] + [ '.' + s for s in park(n-1)] + ['<>' + s for s in park(n-2)]
```

motorcycle first + nothing first + car first



Quick Review: Adding Lists & Strings

```
>>> x = 'cal'
>>> y = 'bears'
>>> u = [x]
>>> v = [y]
```

```
>>> x + y
'calbears'
```

```
>>> u + v
['cal', 'bears']
```

```
>>> ['go ' + z for z in [x, y]]
['go cal', 'go bears']
```

```
>>> ['cal' + x for x in s]
```

What `s` will result in `['cal']`?

What `s` will result in `[]`?

```
>>> s = []
>>> ['cal' + x for x in s]
[]
```

```
>>> s = ['']
>>> ['cal' + x for x in s]
['cal']
```

pollev.com/cs61a