Generators

Announcements

Iterator Practice

Discussion Question: Iterators

`iter(iterable)`: Return an iterator over the elements of an iterable value

`next(iterator)`: Return the next element in an iterator

```
>>> t = iter([3, 4, 5, 6])
>>> next(t)
3
>>> next(t)
4
```

```
def get_n(t, n):
    """ Return a list of the next n items from iterator t.

    >>> t = iter(range(1000000000))
    >>> get_n(t, 10)
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    """
    return ____________________________
```

Discussion Question: Iterators

iter(iterable): Return an iterator over the elements of an iterable value

next(iterator): Return the next element in an iterator

```
>>> t = iter([3, 4, 5, 6])
>>> next(t)
3
>>> next(t)
4
```

```
def get_n(t, n):
    """ Return a list of the next n items from iterator t.

    >>> t = iter(range(1000000000))
    >>> get_n(t, 10)
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    """
    return [next(t) for _ in range(n)]
```

Generators

Generators and Generator Functions

```
>>> def plus_minus(x):
...     yield x
...     yield -x

>>> t = plus_minus(3)
>>> next(t)
3
>>> next(t)
-3
>>> t
<generator object plus_minus ...>
```

A *generator function* is a function that **yield**s values instead of **return**ing them

A normal function **return**s once; a *generator function* can **yield** multiple times

A *generator* is an iterator created automatically by calling a *generator function*

When a *generator function* is called, it returns a *generator* that iterates over its yields

(Demo: cycle)

Fall 2026 Practice Midterm 2, Question 3 (modified)

```
def print_cycle(n, s: list):
    """ Prints the first n items of the cycle sequence that repeats the
    tems of s, but only the odd items.

    >>> pi = [3, 1, -4, -1, 5]
    >>> print_cycle(7, pi)  # [3] [1] -4 [-1] [5] [3] [1]
    3
    1
    -1
    5
    3
    1
    """
    _____________
    for _ in range(n):
        _____________
        if _____________:
            _________
```

Use cycle!

```
def cycle(s):
    """Return an iterator that iterates over the elements of
    s repeatedly.

    >>> t = cycle([3, 4, 5])
    >>> get_n(t, 10)
    [3, 4, 5, 3, 4, 5, 3, 4, 5, 3]
    """
```

Fall 2026 Practice Midterm 2, Question 3 (modified)

```
def print_cycle(n, s: list):
    """ Prints the first n items of the cycle sequence that repeats the
    tems of s, but only the odd items.

    >>> pi = [3, 1, -4, -1, 5]
    >>> print_cycle(7, pi)  # [3] [1] -4 [-1] [5] [3] [1]
    3
    1
    -1
    5
    3
    1
    """
    t = cycle(s)
    for _ in range(n):
        x = next(t)
        if x % 2 == 1:
            print(x)
```

Use cycle!

```
def cycle(s):
    """Return an iterator that iterates over the elements of
    s repeatedly.

    >>> t = cycle([3, 4, 5])
    >>> get_n(t, 10)
    [3, 4, 5, 3, 4, 5, 3, 4, 5, 3]
    """
```

Discussion Question: Ints

```
def yield_ints(start):
    """ Yield all integers, expanding from start.

    >>> t = yield_ints(100)
    >>> [next(t), next(t), next(t), next(t), next(t)]
    [100, 101, 99, 102, 98]
    """
    ____________
    offset = 1
    while _________________:
        ____________________
        ____________________
        offset += 1
```

Discussion Question: Ints

```
def yield_ints(start):
    """ Yield all integers, expanding from start.

    >>> t = yield_ints(100)
    >>> [next(t), next(t), next(t), next(t), next(t)]
    [100, 101, 99, 102, 98]
    """
    yield start
    offset = 1
    while True:
        yield start + offset
        yield start - offset
        offset += 1
```

Spring 2023 Midterm 2 Question 5(b) Revisited

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.

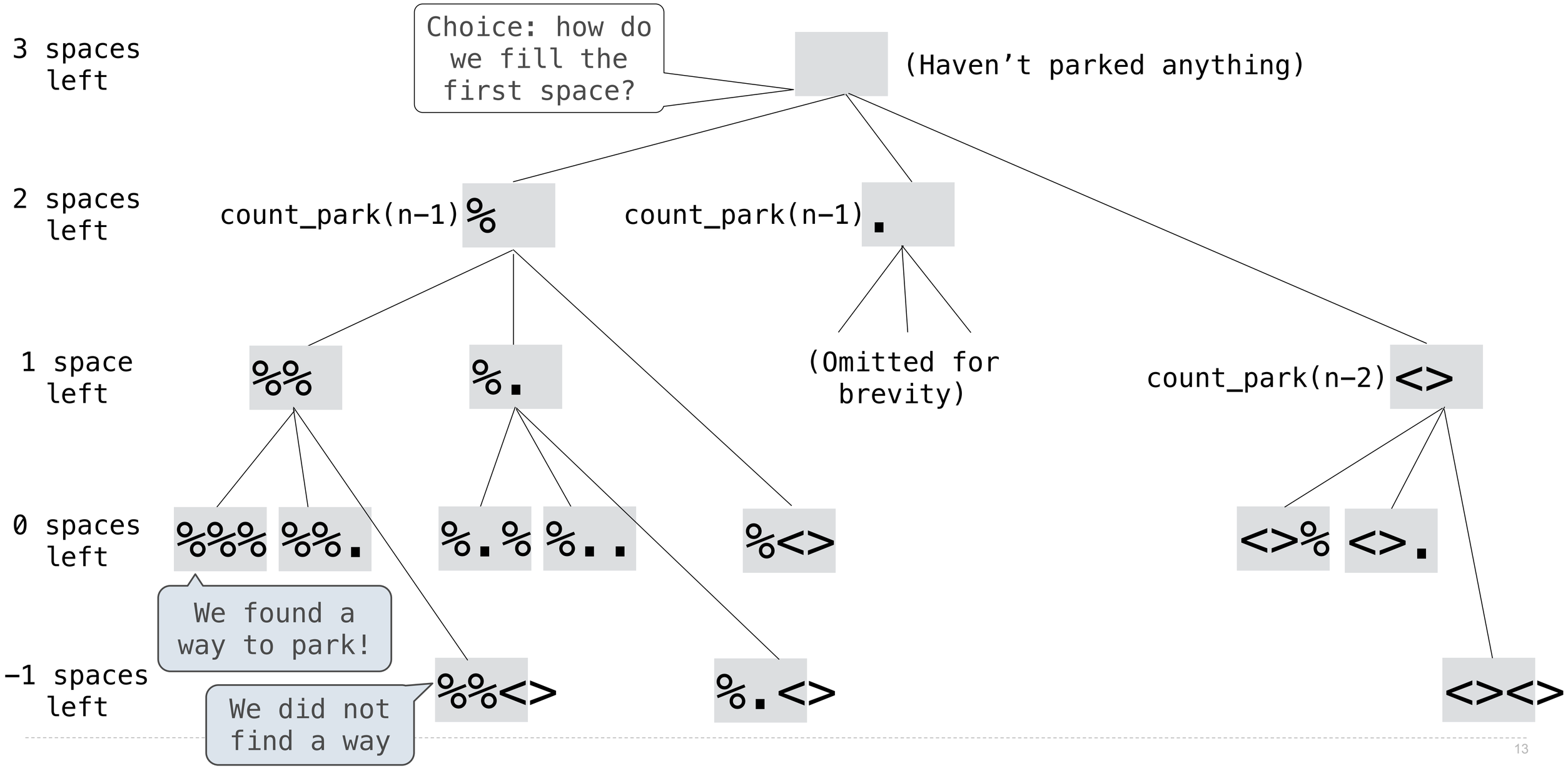
For example: '.%%.<><>' (Thanks to the Berkeley Math Circle for introducing this question.)

Implement **park**, a **generator function** that yields all the ways, represented as strings, that vehicles can be parked in n adjacent parking spots for positive integer n.

```
def park(n):
    """Yield the ways to park cars and motorcycles in n adjacent spots.

    >>> sorted(park(1))
    ['%', '.']
    >>> sorted(park(2))
    ['%%', '%.', '.%', '..', '<>']
    >>> len(list(park(4)))  # some examples: '<><>', '.%%.', '%<>%', '%.<>'
    29
    """
```

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



List Practice

Spring 2023 Midterm 2 Question 1

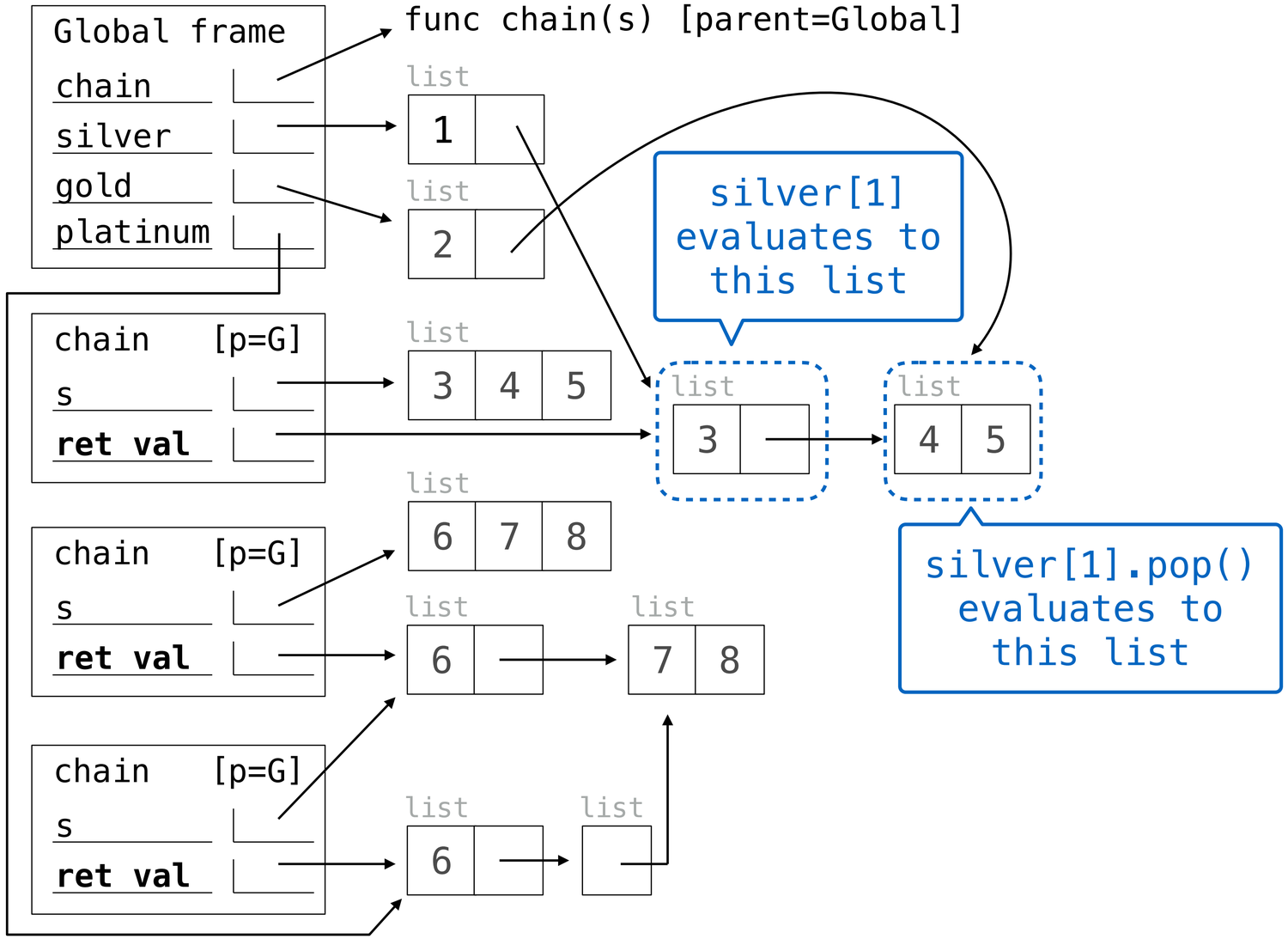
```
def chain(s):
    return [s[0], s[1:]]
silver = [2, chain([3, 4, 5])]
gold = [silver[0], silver[1].pop()]
silver[0] = 1
platinum = chain(chain([6, 7, 8]))
```

Reminder: s.pop() removes and returns the last item in list s.

```
>>> silver
[1, [3]]

>>> gold
[2, [4, 5]]

>>> platinum
[6, [[7, 8]]]
```



https://pythontutor.com/cp/composingprograms.html#code=def%20chain%28s%29%3A%0A%20%20%20%20return%20%5Bs%5B0%5D,%20s%5B1%3A%5D%5D%0Asilver%20%3D%20%5B2,%20chain%28%5B3,%204,%205%5D%29%5D%0Agold%20%3D%20%5Bsilver%5B0%5D,%20silver%5B1%5D.pop%28%29%5D%0Asilver%5B0%5D%20%3D%201%0Aplatinum%20%3D%20chain%28chain%28%5B6,%207,%208%5D%29%29%0A&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D