Welcome to Data C88C!

Lecture 10: Containers
Tuesday, July 8th, 2025
Week 3
Summer 2025
Instructor: Eric Kim (ekim555@berkeley.edu)

Announcements

- Project 01 ("Maps") released! [link]
  - Checkpoint: due Sunday July 13th
  - All: Due Thursday July 24th
  - Group size 2, but you can also work lone
- Practice online midterm (SU24) released on Gradescope
  - "(Optional) Practice Online Midterm (SU24)"
  - Get used to Gradescope exam format, as well as timed exam (120 mins)

Midterm

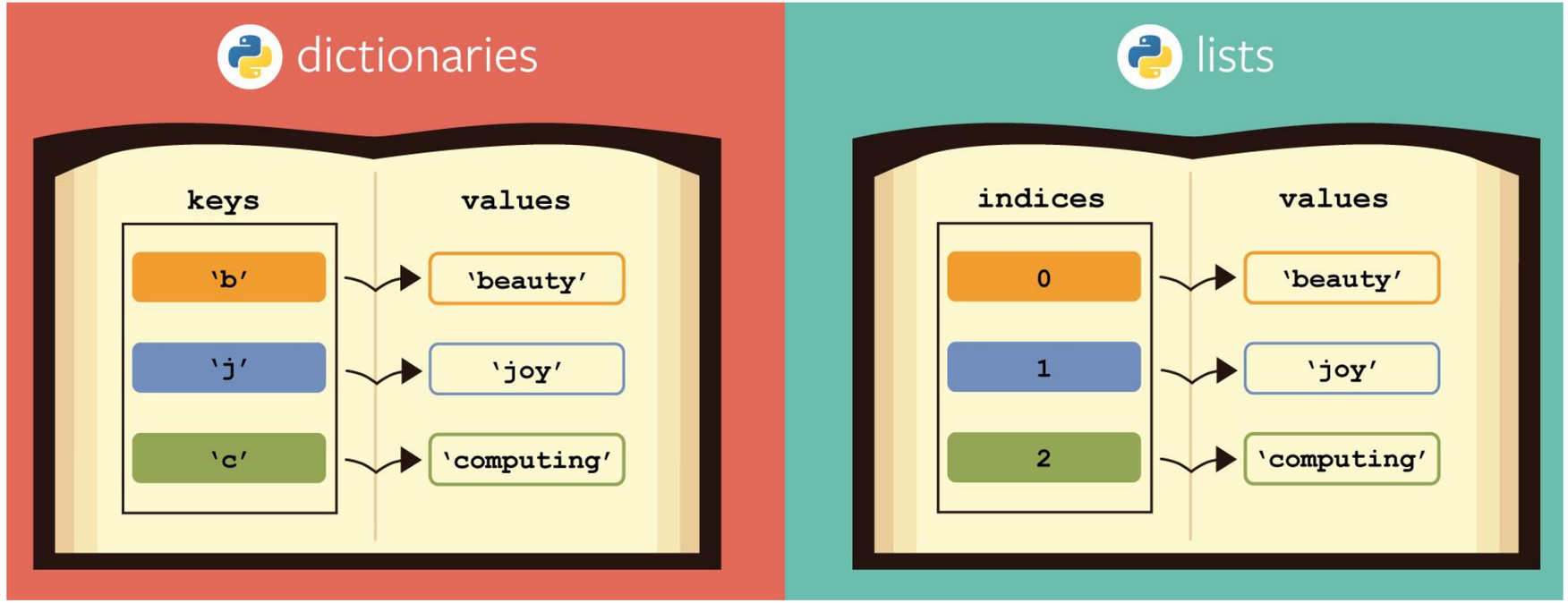
- Midterm scheduling
  - Midterm "main" time: Tuesday July 15th, 3pm-5pm PST
  - There will be a few alternate exam times
  - If you can't make ANY exam time: we will extrapolate your midterm score from your final exam score [link]
- DSP students: 150%, 200% time accommodations
- Midterm will cover everything up to and including Lecture 12 (Object Oriented Programming)
  - Includes: Lab07, HW07, Disc07
- More info will be shared out in the next 1-2 days!

Lecture Overview

- Containers
  - dicts

Python `dict`

- **Lists** let us index a value by a number, or position.
- **Dictionaries** let us index data by other kinds of data.
  - KEY -> VALUE mapping. Aka "**lookup table**"
- Extremely useful data type used in many languages/systems
-



Credit: https://berkeley-cs61as.github.io/textbook/dictionaries-and-memoization.html

dict basic examples

```
>>> item_to_price = {}
>>> item_to_price["apple"] = 1
>>> item_to_price["soda"] = 2
>>> item_to_price
{'apple': 1, 'soda': 2}
```

Create the dict ("constructor").
Another way to create this dict:
`>>> item_to_price = {"apple": 1, "soda": 2}`

```
>>> item_to_price["soda"]
2
```

Retrieve value for a key ("Getter")

```
>>> item_to_price["latte"] = 5
>>> item_to_price
{'apple': 1, 'soda': 2, 'latte': 5}
```

Add new key+value mapping ("Setter")

```
>>> item_to_price["burger"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'burger'
```

Note: asking for a key that's not in the dict will result in an error!

(reference) common dict operations

Operation	Result

`dict((('key1', 'value1', 'key2', 'value2')))`
`dict(key='value1', key2='value2')`
`{'key1': 'value1', 'key2': 'value2'}`
	Creates a dictionary with 'key1'->'value1, 'key2'->'value2' mappings. Constructors.

`d['key1']`	Returns the value for 'key1'. Throws a "KeyError" if 'key1' is not in the dict.
`'key1' in d`	Returns True if 'key1' is in the dict, False otherwise.
`d['key1'] = 42`	Sets a 'key1'->42 mapping. If 'key1' already exists in the dict, this overwrites the existing value.
`{key: value for (key, value) in [("key1", "value"), ("key2", "value2")]}`	Dictionary comprehension. Creates a new dict.
`d.pop('key1')`	Removes the key 'key1' (and its associated value) from the dict. Returns the value of 'key1'. Throws a "KeyError" if 'key1' is not in the dict.
`len(d)`	Returns the number of key+value pairs in dict.
`d.keys()`	Returns an iterator over the keys of the dict.
`d.values()`	Returns an iterator over the values of the dict
`d.items()`	Returns an iterator over the key+value pairs of the dict

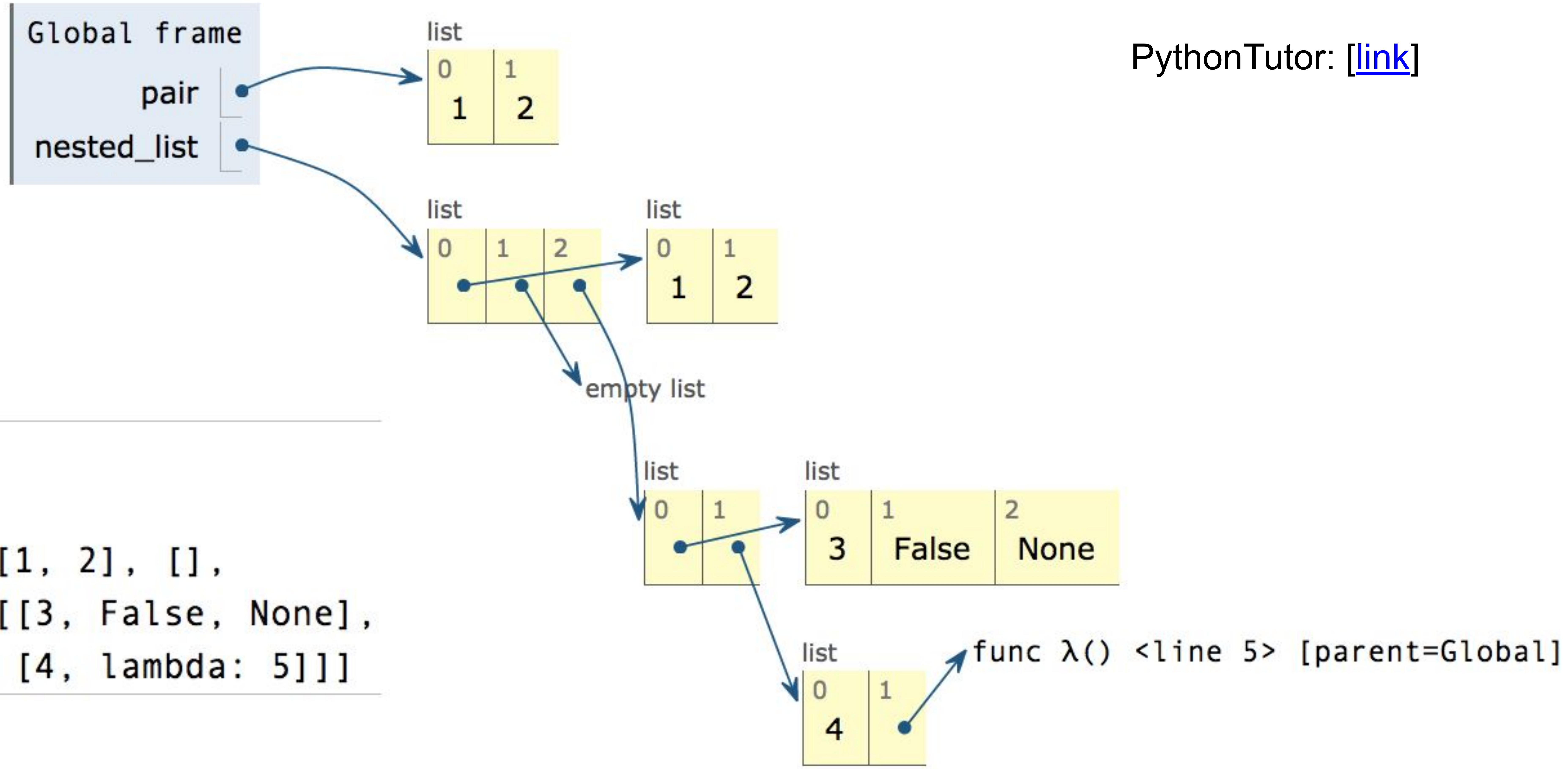
Box-and-Pointer Notation

Box-and-Pointer Notation in Environment Diagrams

Lists are represented as a row of index-labeled adjacent boxes, one per element

Each box either contains a **primitive value** (eg int) or points to a **compound value** (another arrow)

PythonTutor: [link]




```
pair = [1, 2]

nested_list = [[1, 2], [],
               [[3, False, None],
                [4, lambda: 5]]]
```


pythontutor.com/composingprograms.html#code=pair%20%3D%20[1,%202]%0A%0Anested_list%20%3D%20[[1,%202],%20[],%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20[[3,%20False,%20None],%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20[4,%20lambda%3A%205]]]&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=4

Discussion Question

Question: What's the environment diagram? What gets printed?

```
def f(s):
    x = s[0]
    return [x]

t = [3, [2+2, 5]]
u = [f(t[1]), t]
print(u)
```

Answer: `[[4], [3, [4, 5]]]`

PythonTutor: [link]

Double-Eights with a List

Implement double_eights, which takes a list s and returns whether two consecutive items are both 8.

using positions (indices)...

```
def double_eights(s):
    """Return whether two consecutive items
    of list s are 8.

    >>> double_eights([1, 2, 8, 8])
    True
    >>> double_eights([8, 8, 0])
    True
    >>> double_eights([5, 3, 8, 8, 3, 5])
    True
    >>> double_eights([2, 8, 4, 6, 8, 2])
    False
    """
    for ____________________________:
        if ____________________________:
            return True
    return False
```

i in range(len(s)-1)

s[i] == 8 and s[i+1] == 8

using slices...

```
def double_eights(s):
    """Return whether two consecutive items
    of list s are 8.

    >>> double_eights([1, 2, 8, 8])
    True
    >>> double_eights([8, 8, 0])
    True
    >>> double_eights([5, 3, 8, 8, 3, 5])
    True
    >>> double_eights([2, 8, 4, 6, 8, 2])
    False
    """
    if ____________ == [ _____ ]:
        return True
    elif len(s) < 2:
        return False
    else:
        return ________________________
```

s[:2]

8, 8

double_eights(s[1:])

https://pythontutor.com/cp/composingprograms.html#code=def%20double_eights%28s%29%3A%0A%20%20%20%20if%20s%5B%3A2%5D%20%3D%3D%20%5B8,%208%5D%3A%0A%20%20%20%20%20%20%20%20return%20True%0A%20%20%20%20elif%20len%28s%29%20%3C%202%3A%0A%20%20%20%20%20%20%20%20return%20False%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20return%20double_eights%28s%5B1%3A%5D%29%0A%20%20%20%20%20%20%20%20%0Adouble_eights%28%5B5,%203,%208,%208,%203,%205%5D%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Double-Eights with a List

Implement double_eights, which takes a list s and returns whether two consecutive items are both 8.

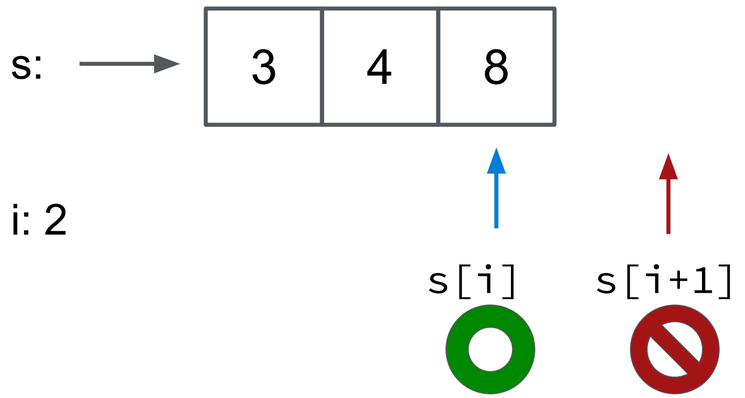
```
def double_eights(s):
    """Return whether two consecutive items
    of list s are 8.

    >>> double_eights([1, 2, 8, 8])
    True
    >>> double_eights([2, 8, 4, 6, 8, 2])
    False
    """
    for ____________________________:
        if ____________________________:
            return True
    return False
```

i in range(len(s)-1)

s[i] == 8 and s[i+1] == 8

Question: why is it important for the range to be `range(len(s)-1)`? What if I instead used `range(len(s))`?

Answer: for certain inputs (ex: s=[3, 4, 8]), the code will crash with an "IndexError: list index out of range".
This is because, at the final iteration, s[i+1] will attempt to index outside of the list, and throw an `IndexError`.



Demo: PythonTutor: [link]

Processing Container Values

Aggregation

Several built-in functions take iterable arguments and aggregate them into a value

- **sum**(iterable[, start]) -> value

  Return the sum of an iterable (not of strings) plus the value of parameter 'start' (which defaults to 0). When the iterable is empty, return start.

- **max**(iterable[, key=func]) -> value
  max(a, b, c, ...[, key=func]) -> value

  With a single iterable argument, return its largest item.
  With two or more arguments, return the largest argument.

- **all**(iterable) -> bool

  Return True if bool(x) is True for **all** values x in the iterable.
  If the iterable is empty, return True.

- **any**(iterable) -> bool: return True if bool(x) is True for **any** value in iterable. If iterable is empty, return False.

```
>>> sum([1, 2, 3])
6
>>> max([1, 2, 3])
3
>>> max(1, 2, 3)
3
>>> all([True, 1 == 1, True])
True
>>> any([True, False, False])
True
```

Spring 2023 Midterm 2 Question

Definition. A *prefix sum* of a sequence of numbers is the sum of the first `n` elements for some positive length `n`.

(a) (4.0 points)

Implement `prefix`, which takes a list of numbers `s` and returns a list of the prefix sums of `s` in increasing order of the length of the prefix.

```
def prefix(s):
    """Return a list of all prefix sums of list s.

    >>> prefix([1, 2, 3, 0, 4, 5])
    [1, 3, 6, 6, 10, 15]
    >>> prefix([2, 2, 2, 0, -5, 5])
    [2, 4, 6, 6, 1, 6]
    """
    return [_________ for k in _________]
             (a)              (b)
```

sum(s[:k+1])

range(len(s))

ii. (1.0 pt) Fill in blank (b).

- ◯ `s`
- ◯ `[s]`
- ◯ `s[1:]`
- ◯ `range(s)`
- ◯ `range(len(s))`

Tree Recursion (with Strings)

(again) 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)
```

Three choices:
(a) Place a car down (n-2)
(b) Place a motorcycle down (n-1)
(c) Leave an empty space (n-1)

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 _____________________________________________________________
```

[]

['']

(b) (c) (a)

Three choices:
(a) Place a car "<>" down (n-2)
(b) Place a motorcycle "%" down (n-1)
(c) Leave an empty space "." (n-1)

['%'+s for s in park(n-1)] + ['.'+s for s in park(n-1)] + ['<>'+s for s in park(n-2)]

park(3):

%%%
%%.
%.%
%..
%<>
.%%
.%.
..%
...
.<>
<>%
<>.