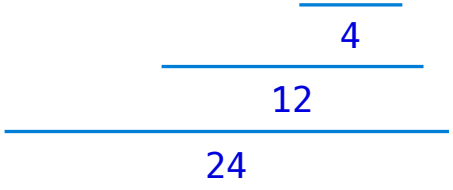
Composition

Announcements

Discussion 7

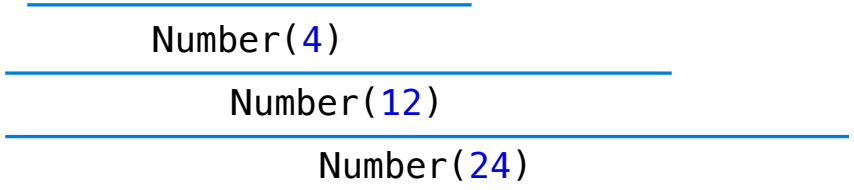
Order of Evaluation

```
def double(x):
    return 2 * x

def triple(y):
    return 3 * y

>>> double(triple(2 + 2))
```




```
24
```

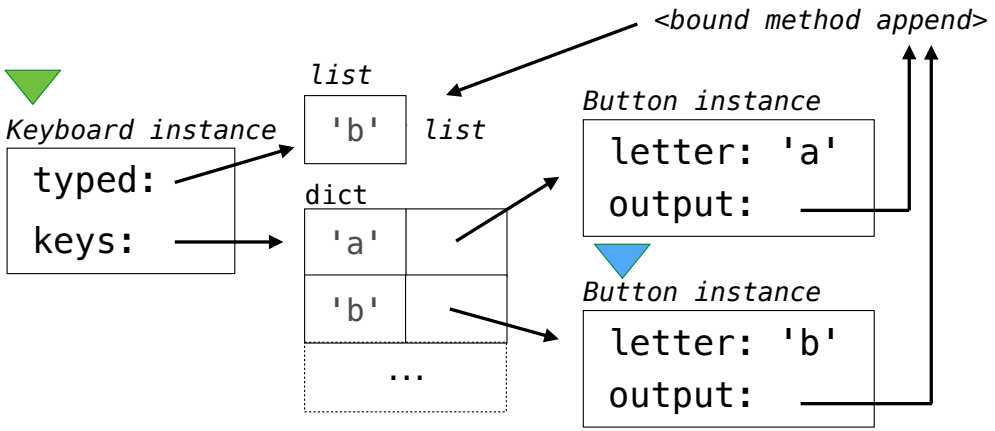
```
class Number:
    def __init__(self, z):
        self.z = z

    def __add__(self, other):
        return Number(self.z + other.z)

    def __repr__(self):
        return f'Number({self.z})'

    def duple(self):
        return Number(2 * self.z)

    def treble(self):
        return Number(3 * self.z)

>>> (Number(2) + Number(2)).treble().duple()
```




```
Number(24)
```

Keyboard

```
class Button:
    caps_lock = CapsLock()

    def __init__(self, letter, output):
        assert letter in LOWERCASE_LETTERS
        self.letter = letter
        self.output = output

    def press(self):
        if self.caps_lock.pressed % 2 == 1:
            self.output(self.letter.upper())
        else:
            self.output(self.letter)
        return self
```




```
class Keyboard:
    def __init__(self):
        self.typed = []
        self.keys = {c: Button(c, self.typed.append) for c in LOWERCASE_LETTERS}
        'beep'
    def type(self, word):
        start = len(self.typed)
        for w in word:
            self.keys[w].press()
        return self.typed[start:]
```

Bear

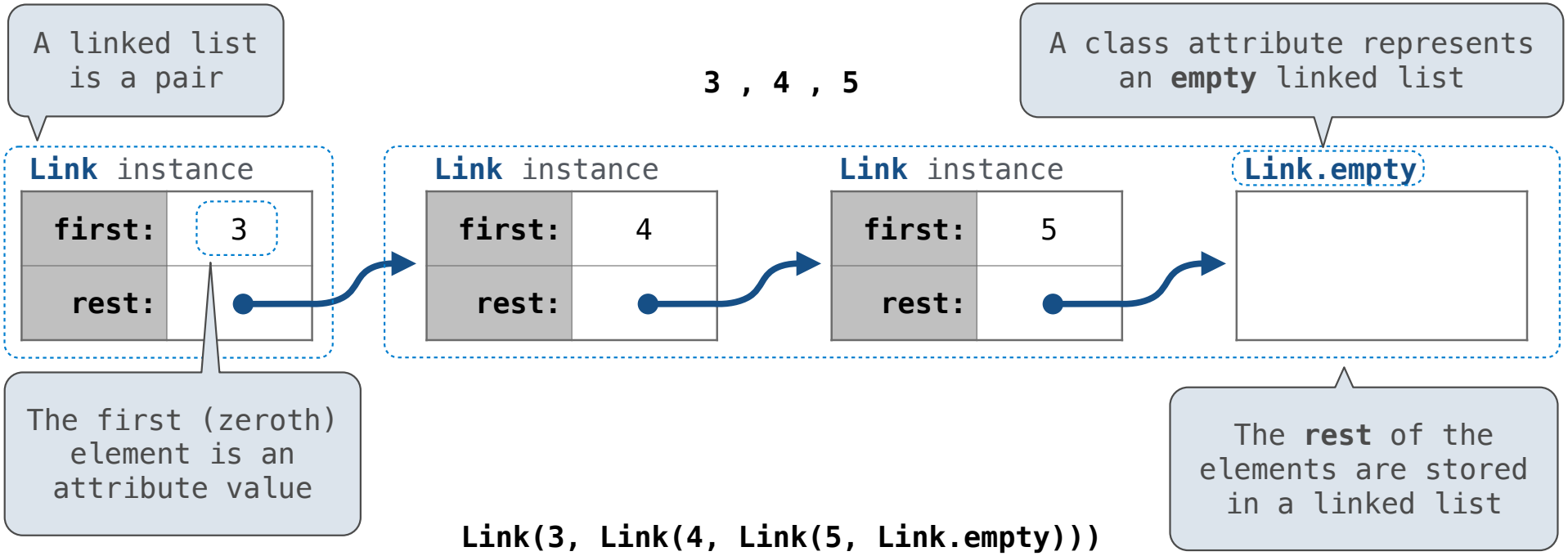
```
class Eye:
    def __init__(self, closed=False):
        self.closed = closed

    def draw_eye(self):
        if self.closed:
            return '–'
        else:
            return '•'

    def __str__(self):
        return self.draw_eye()

class Bear:  # ʕ •ᴥ•ʔ
    def __init__(self):
        self.nose_and_mouth = 'ᴥ'

    def eye(self):
        return Eye()

    def __str__(self):
        return 'ʕ ' + str(self.eye()) + self.nose_and_mouth + str(self.eye()) + 'ʔ'
```

```
class SleepyBear(Bear):   # ʕ –ᴥ–ʔ
    def eye(self):
        return Eye(True)

class WinkingBear(Bear):  # ʕ –ᴥ•ʔ
    def __init__(self):
        super().__init__()
        self.eye_calls = 0

    def eye(self):
        self.eye_calls += 1
        return Eye(self.eye_calls % 2)
```

Linked Lists

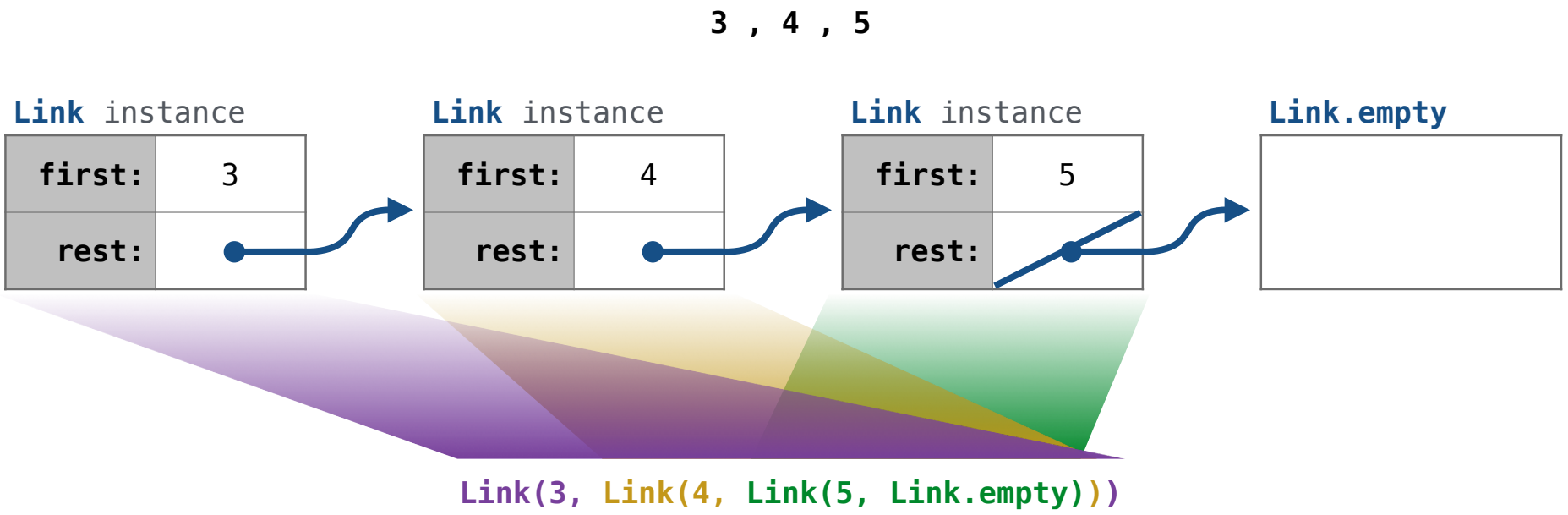
Linked List Structure

A linked list is either empty **or** a first value and the rest of the linked list



Linked List Structure

A linked list is either empty **or** a first value and the rest of the linked list



Linked List Class

Linked list class: attributes are passed to `__init__`

```
class Link:
    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
```

Some zero-length sequence

Returns whether rest is a Link

`help(isinstance)`: Return whether an object is an instance of a class or of a subclass thereof.

```
Link(3, Link(4, Link(5            )))
```

(Demo)

Linked List Practice

Spring 2023 Midterm 2 Question 3(b)

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

Implement tens, which takes a non-empty linked list of numbers s represented as a Link instance. It prints all of the prefix sums of s that are multiples of 10 in increasing order of the length of the prefix.

```
def tens(s):
    """Print all prefix sums of Link s that are multiples of ten.
    >>> tens(Link(3, Link(9, Link(8, Link(10, Link(0, Link(14, Link(6)))))))
    20
    30
    30
    50
    """
    def f(suffix, total):
        if total % 10 == 0:
            print(total)

        if ____suffix is not Link.empty____:
            ____f(suffix.rest, total + suffix.first)____

    ____f(s.rest, s.first)____
```