при поддержке
{a}вижн

Лекция 11

Перевод и адаптация материалов Джона ДеНиро (John DeNero). Используется с разрешения автора.

Диаграммы типа Контейнер-и-Указатель

Свойство замыкания типов данных

Метод объединения значений данных удовлетворяет условию замыкания, если:

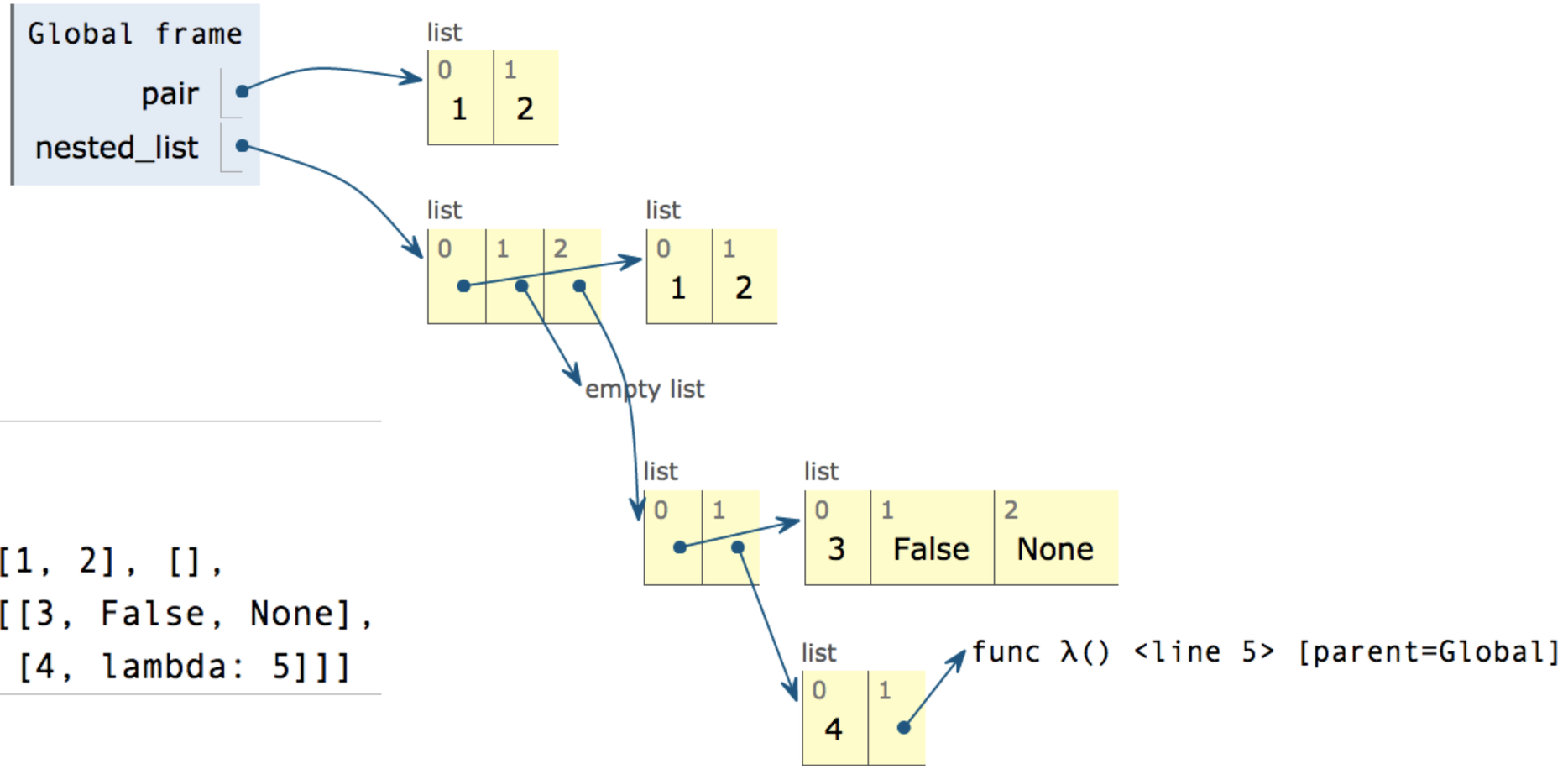
- Результат объединения может быть использован для объединения тем же методом.
- Замыкание позволяет эффективно объединять данные, поскольку позволяет создавать иерархические структуры.
- Иерархические структуры состоят из частей, которые в свою очередь состоят из частей и так далее.

Элементами списка могут быть списки (помимо всего прочего)

Контейнер-и-Указатель на диаграммах окружения

Списки представляются строкой пронумерованых контейнеров, один на элемент.

Каждый контейнер содержит простое значение или указывает на составное значение.




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

Интерактивная диаграмма

Операции с последовательностями

Принадлежность и срезы

В Python есть операторы принадлежности и среза.

Принадлежность.

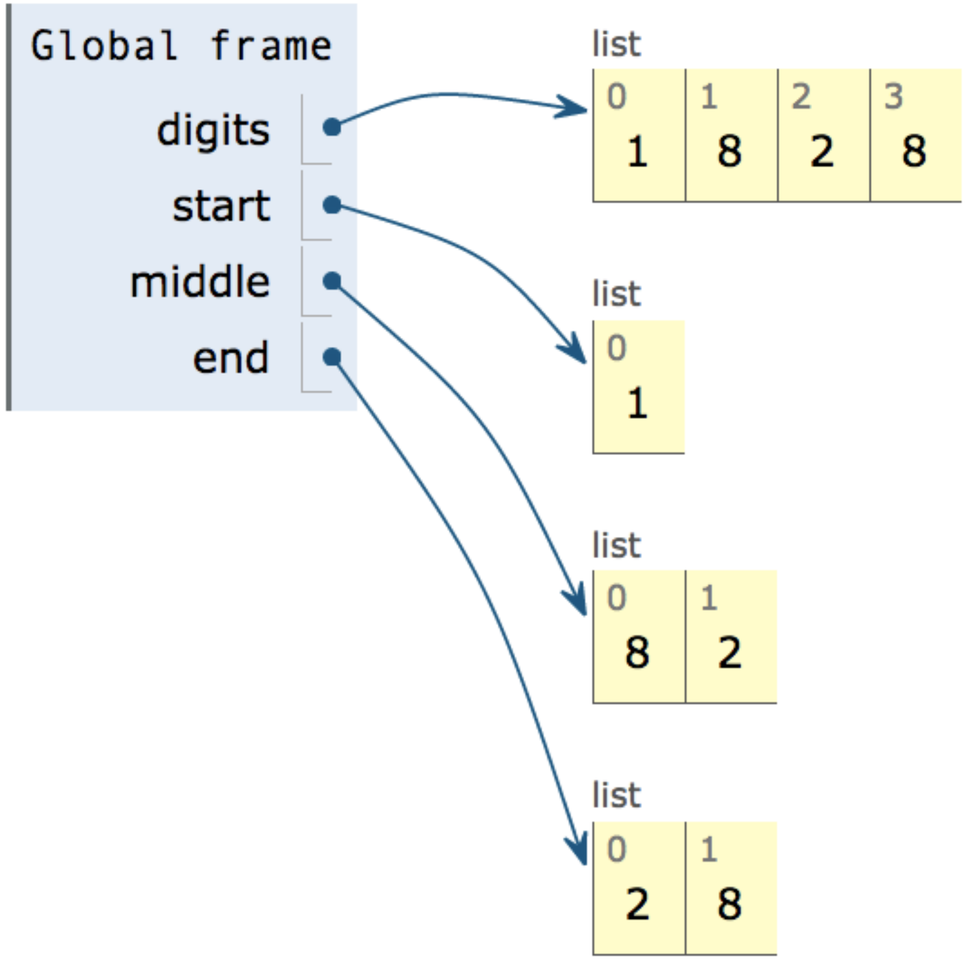
```
>>> digits = [1, 8, 2, 8]
>>> 2 in digits
True
>>> 1828 not in digits
True
```

Срез.

```
>>> digits[0:2]
[1, 8]
>>> digits[1:]
[8, 2, 8]
```

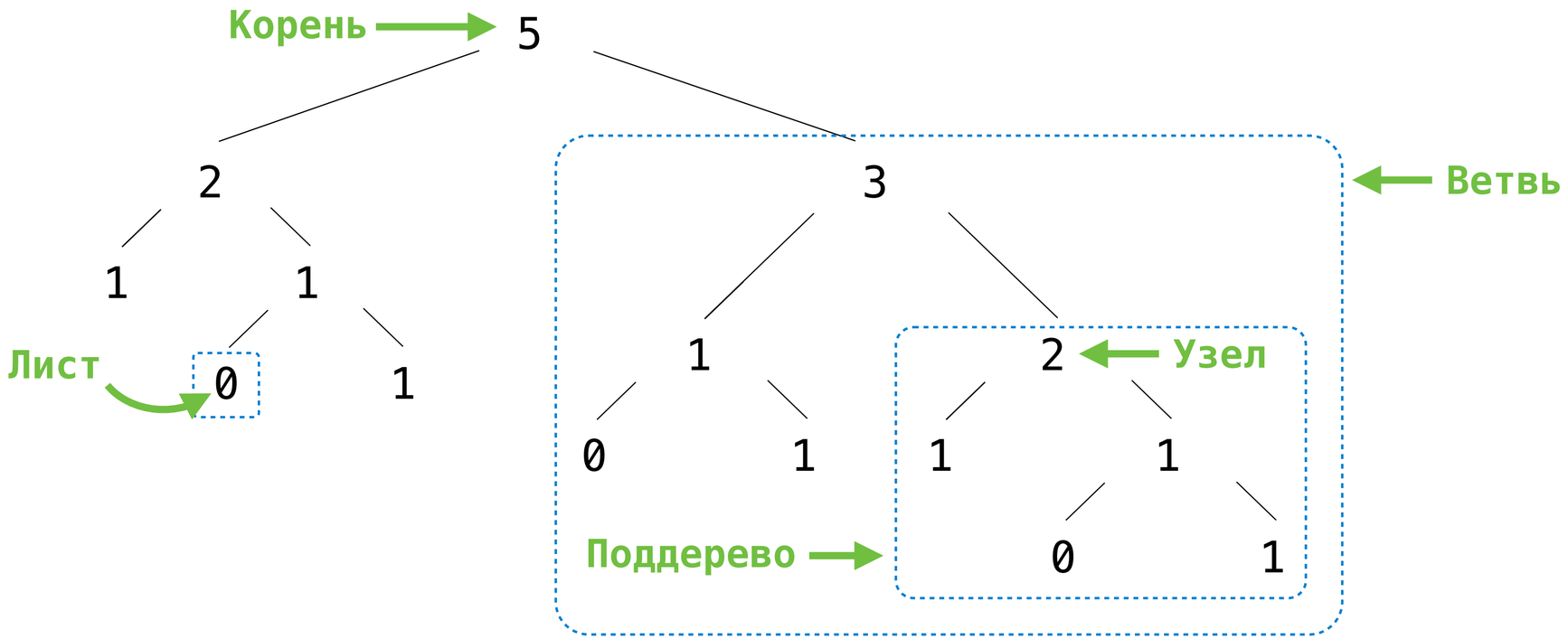
Возвращает новый объект

```
1  digits = [1, 8, 2, 8]
2  start =  digits[:1]
3  middle = digits[1:3]
4  end =    digits[2:]
```



Деревья

Абстракция дерева



Дерево содержит корневое значение и набор ветвей; каждая ветвь — тоже дерево.

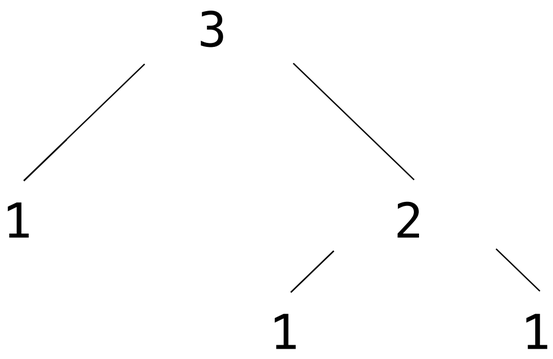
Дерево без ветвей называют листом.

Корневые значения поддеревьев корневого дерева часто называют узловыми значениями или узлами.

Создание абстракции корневого дерева

```
def tree(label, branches=[]):
    return [label] + branches

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]
```

Дерево содержит корневое значение и набор ветвей; каждая ветвь — тоже дерево.




```
>>> tree(3, [tree(1),
...          tree(2, [tree(1),
...                   tree(1)])])
[3, [1], [2, [1], [1]]]
```

Создание абстракции корневого дерева

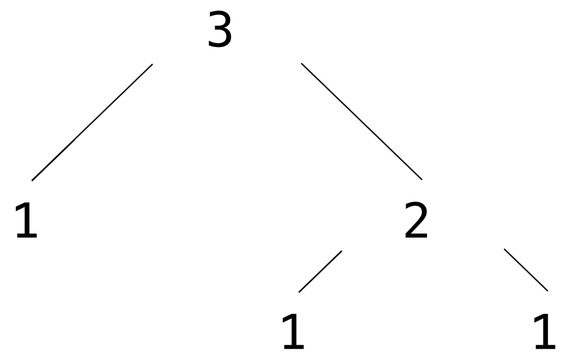
```
def tree(label, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [label] + list(branches)

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]

def is_tree(tree):
    if type(tree) != list or len(tree) < 1:
        return False
    for branch in branches(tree):
        if not is_tree(branch):
            return False
    return True
```

Ветви — это деревья!

Список из последовательности ветвей

Проверяет, что дерево сделано на списках

Дерево содержит корневое значение и набор ветвей; каждая ветвь — тоже дерево.




```
>>> tree(3, [tree(1),
...          tree(2, [tree(1),
...                   tree(1)])])
[3, [1], [2, [1], [1]]]
```

```
def is_leaf(tree):
    return not branches(tree)
```

(Пример)

Рекурсивная обработка деревьев

Обработка листа зачастую является базовым случаем функции обработки дерева.

Рекурсивная часть обычно содержит рекурсивный вызов для каждой ветви и затем агрегирует результаты.

```
def count_leaves(tree):
    """Считает количество листьев в дереве."""
    if is_leaf(tree):
        return 1
    else:
        branch_counts = [count_leaves(b) for b in branches(tree)]
        return sum(branch_counts)
```

(Пример)

Вопрос

Дополни функцию `leaves`, которая принимает дерево и возвращает список его листьев.

Подсказка: Если объединять `(sum)` список списков, то получится общий список из элементов этих списков.

```
>>> sum([[1], [2, 3], [4]], [])
[1, 2, 3, 4]
>>> sum([[1]], [])
[1]
>>> sum([[[1]], [2]], [])
[[1], 2]
```

```
def leaves(tree):
    """Возвращает список листьев дерева.

    >>> leaves(fib_tree(5))
    [1, 0, 1, 0, 1, 1, 0, 1]
    """
    if is_leaf(tree):
        return [label(tree)]
    else:
        return sum([leaves(b) for b in branches(tree)], [])
```