

Лекция 11

Диаграммы типа Контейнер-и-Указатель

Свойство замыкания типов данных

Свойство замыкания типов данных

Метод объединения значений данных удовлетворяет условию замыкания, если:

Свойство замыкания типов данных

Метод объединения значений данных удовлетворяет условию замыкания, если:

- Результат объединения может быть использован для объединения тем же методом.

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.

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.

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

Контейнер-и-Указатель на диаграммах окружения

Списки представляются строкой пронумерованных контейнеров, один на элемент.

Контейнер-и-Указатель на диаграммах окружения

Списки представляются строкой пронумерованных контейнеров, один на элемент.

Каждый контейнер содержит простое значение или указывает на составное значение.

Контейнер-и-Указатель на диаграммах окружения

Списки представляются строкой пронумерованных контейнеров, один на элемент.

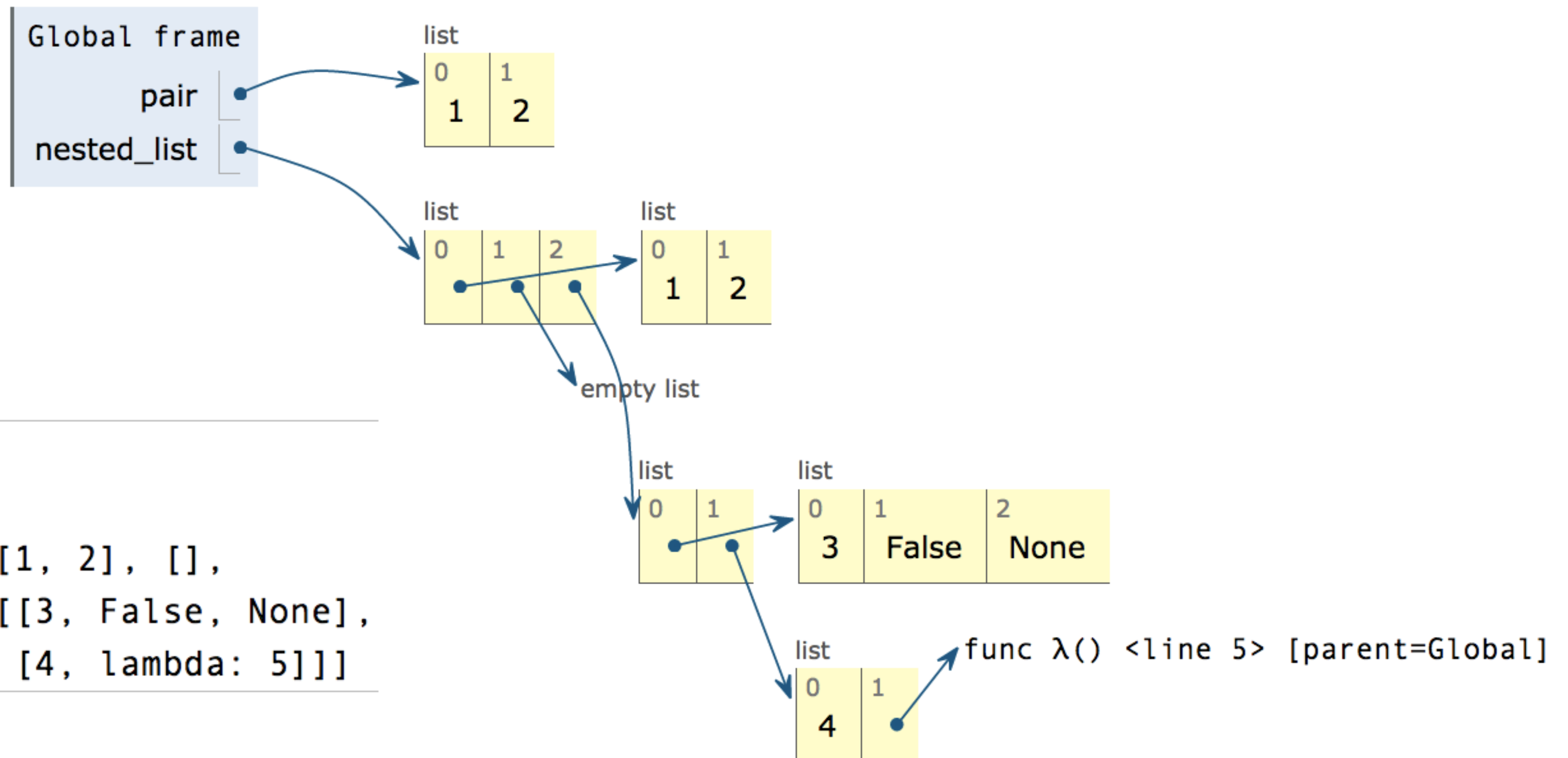
Каждый контейнер содержит простое значение или указывает на составное значение.

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

Контейнер-и-Указатель на диаграммах окружения

Списки представляются строкой пронумерованных контейнеров, один на элемент.

Каждый контейнер содержит простое значение или указывает на составное значение.



Операции с последовательностями

Принадлежность и срезы

В Python есть операторы принадлежности и среза.

Принадлежность и срезы

В Python есть операторы принадлежности и среза.

Принадлежность.

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
```

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.

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]
```

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]
```



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
```

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

Срез.

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

Возвращает новый объект

Принадлежность и срезы

В Python есть операторы принадлежности и среза.

Принадлежность.

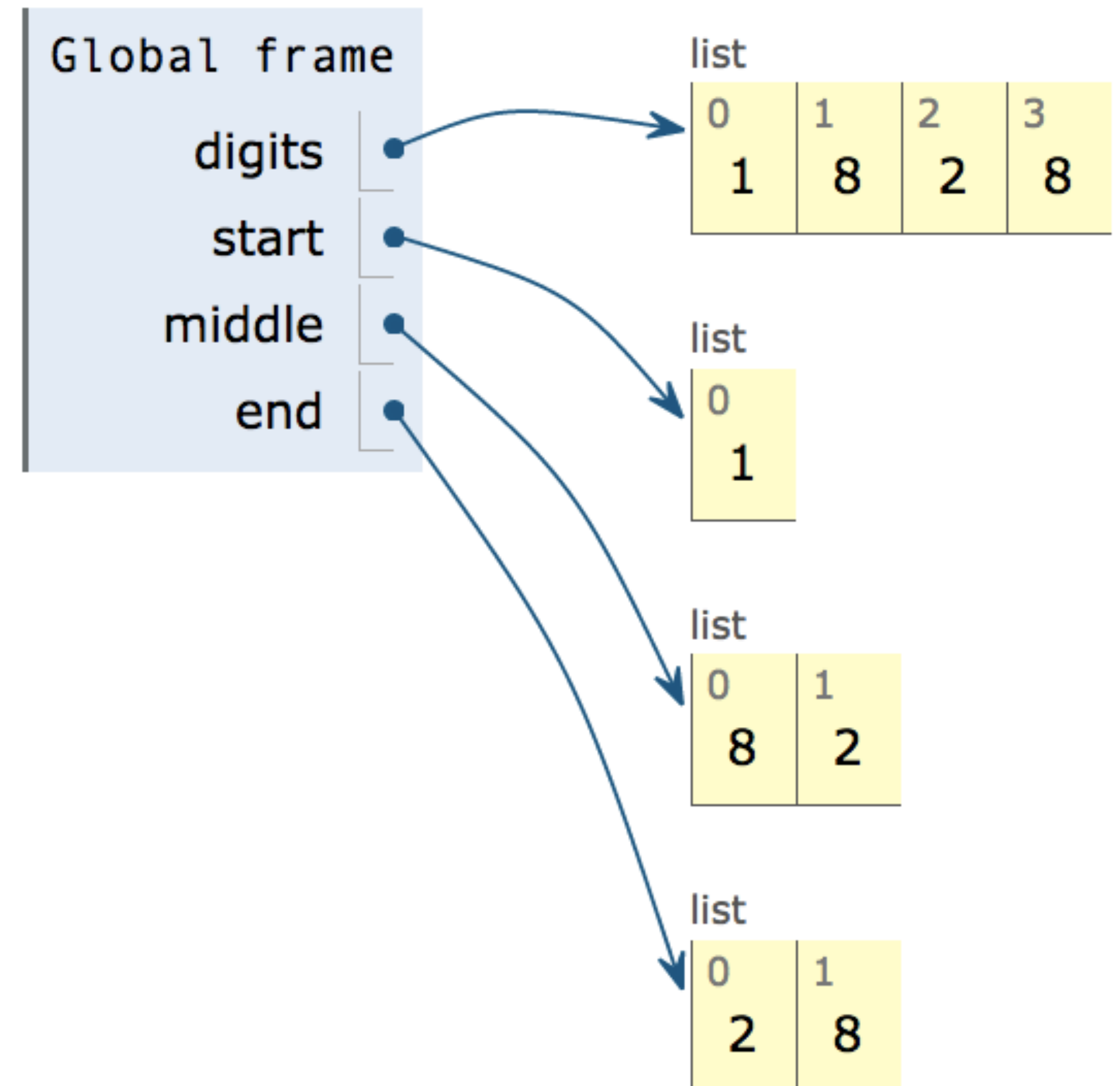
```
>>> digits = [1, 8, 2, 8]
>>> 2 in digits
True
>>> 1828 not in digits
True
```

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

Срез.

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

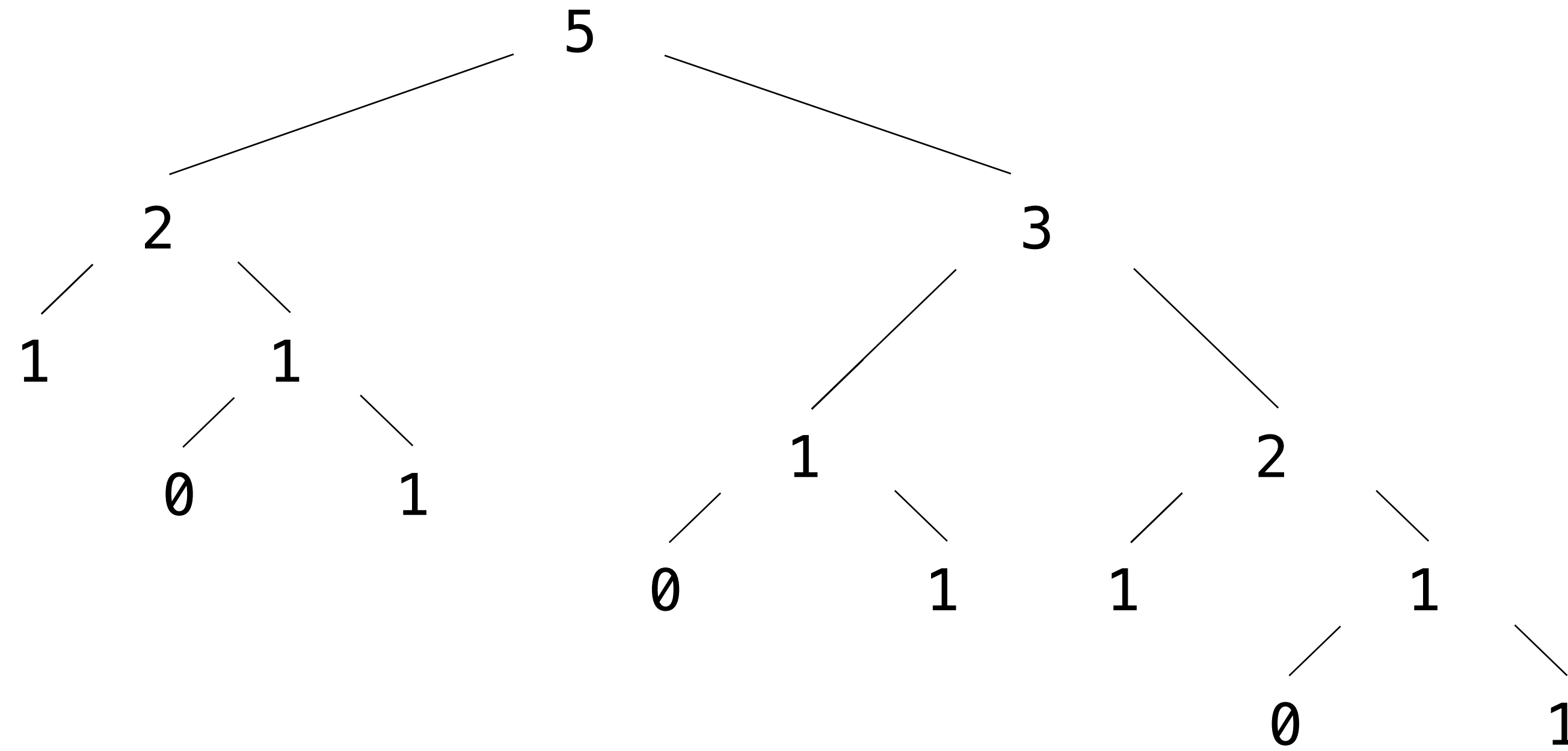
Возвращает новый объект



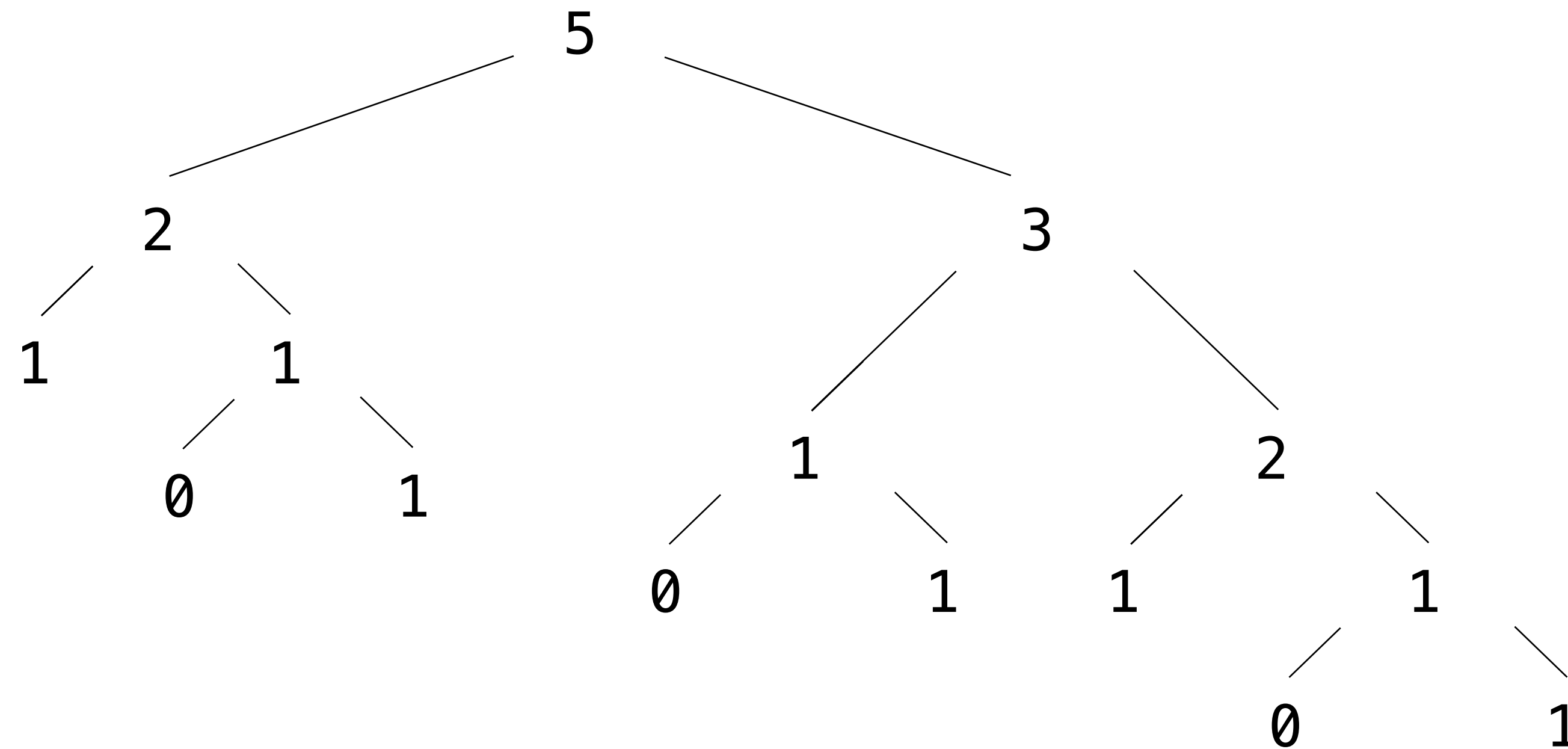
Деревья

Абстракция дерева

Абстракция дерева

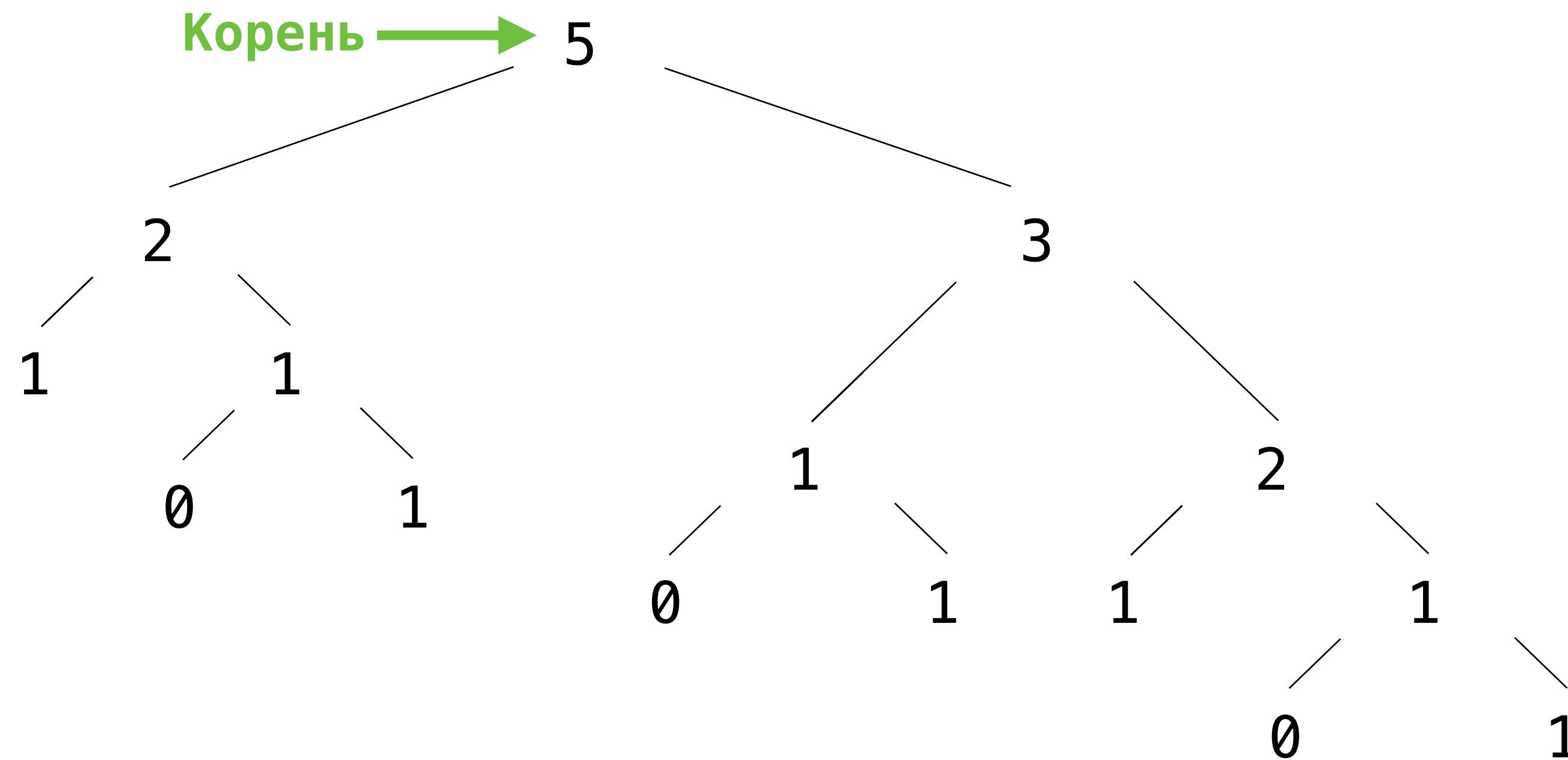


Абстракция дерева



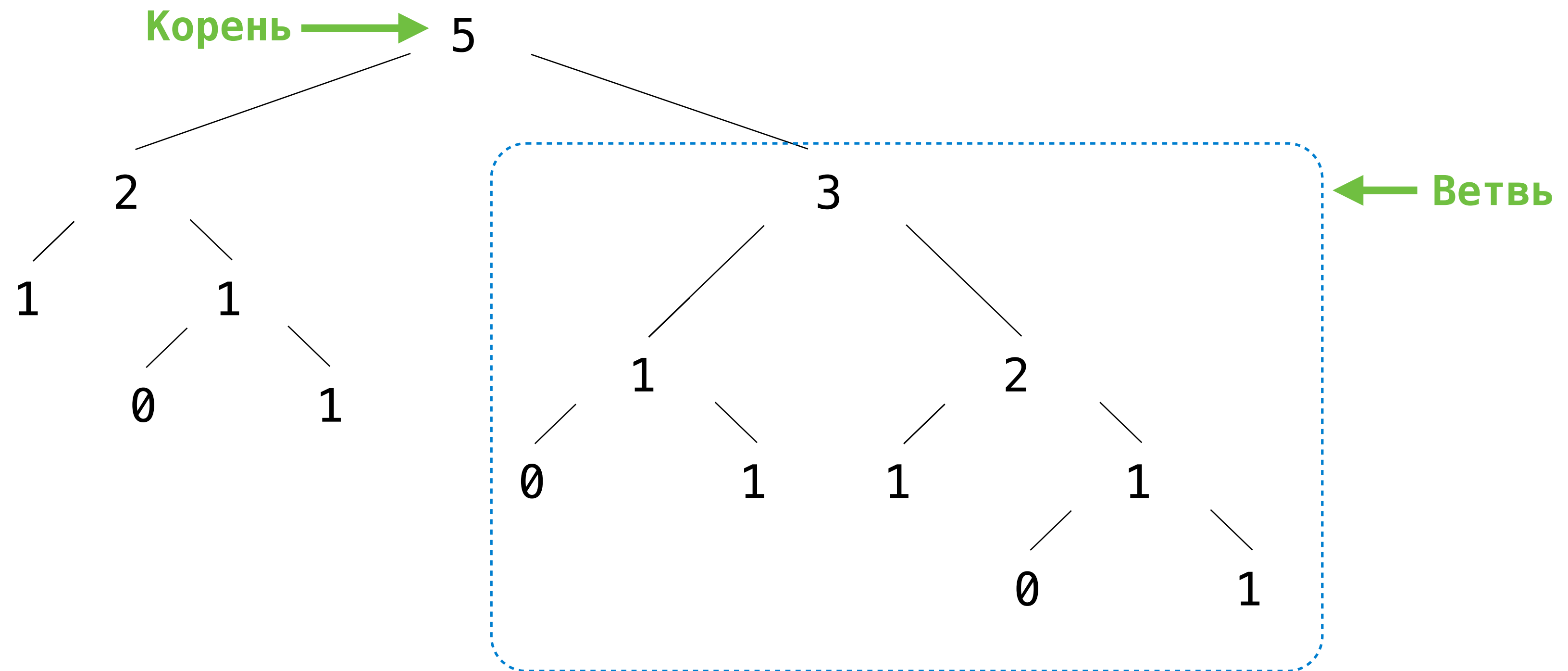
Дерево содержит корневое значение и набор ветвей; каждая ветвь – тоже дерево.

Абстракция дерева



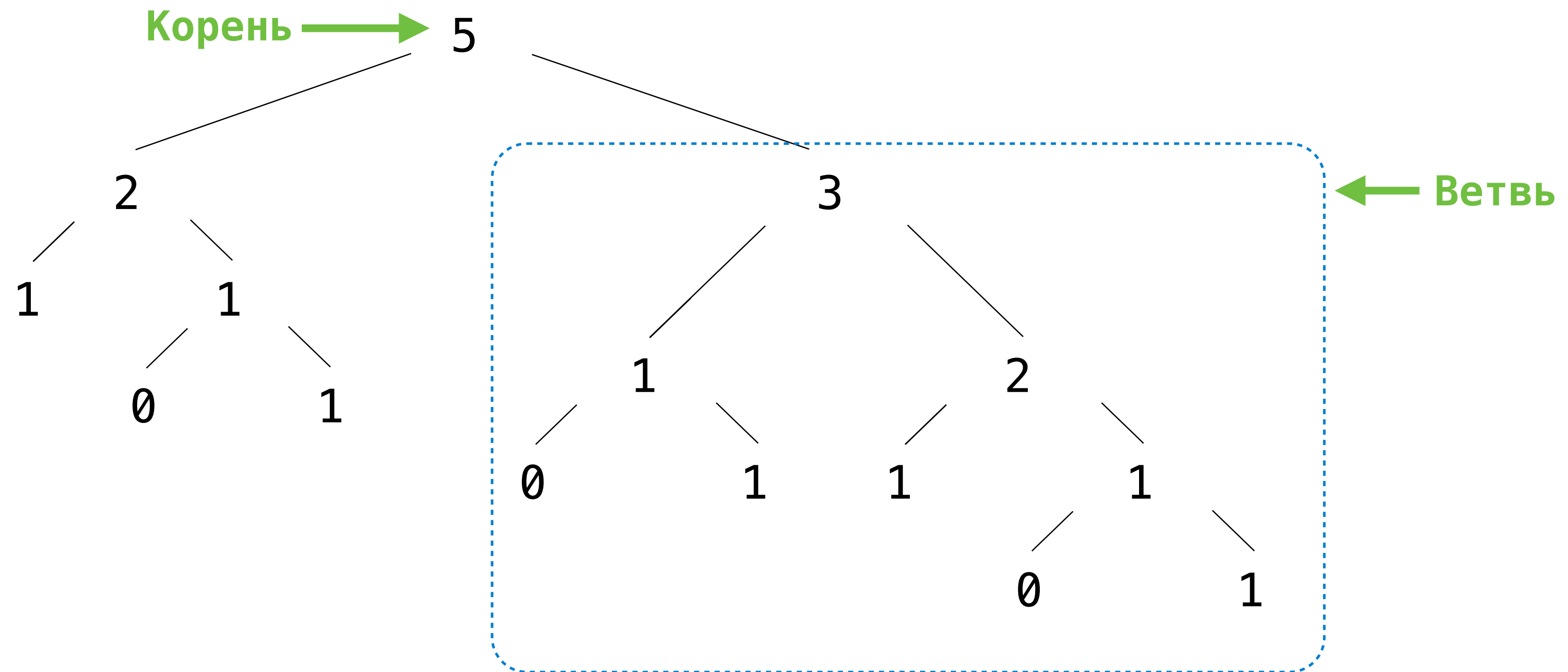
Дерево содержит корневое значение и набор ветвей; каждая ветвь – тоже дерево.

Абстракция дерева



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

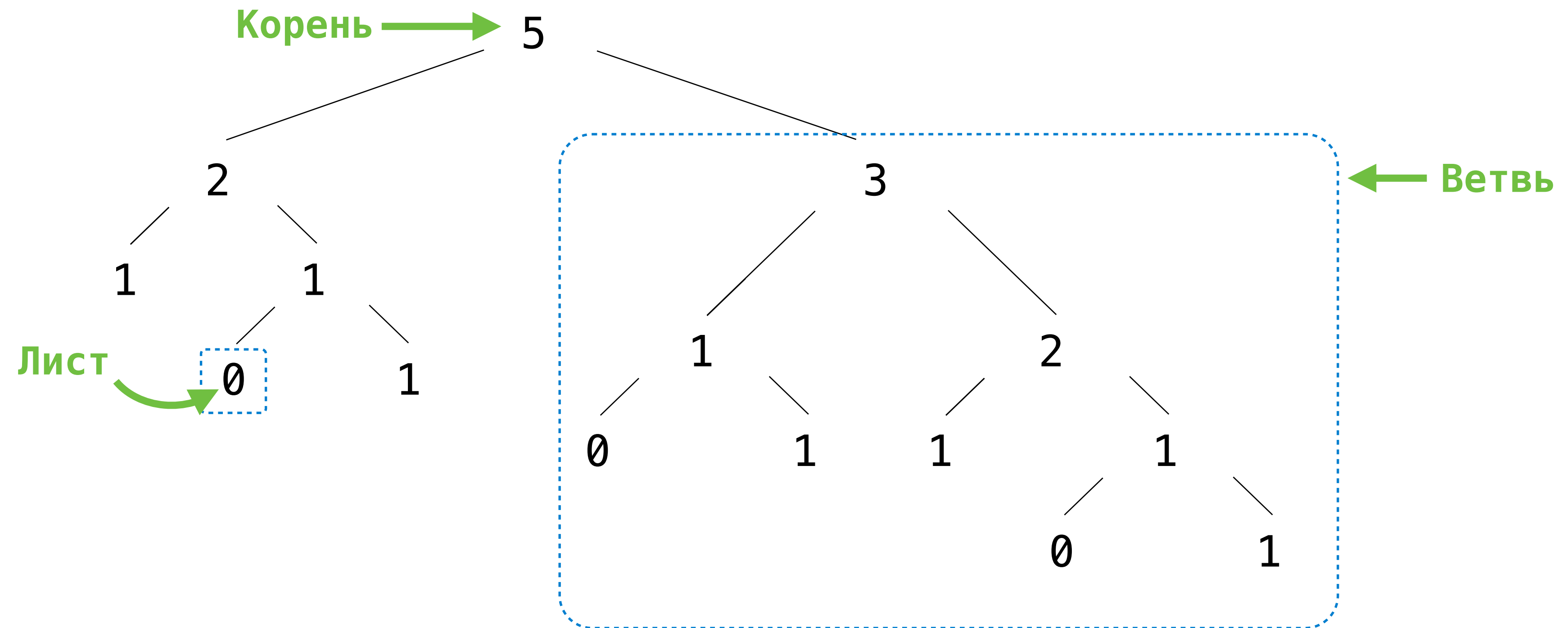
Абстракция дерева



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

Дерево без ветвей называют листом.

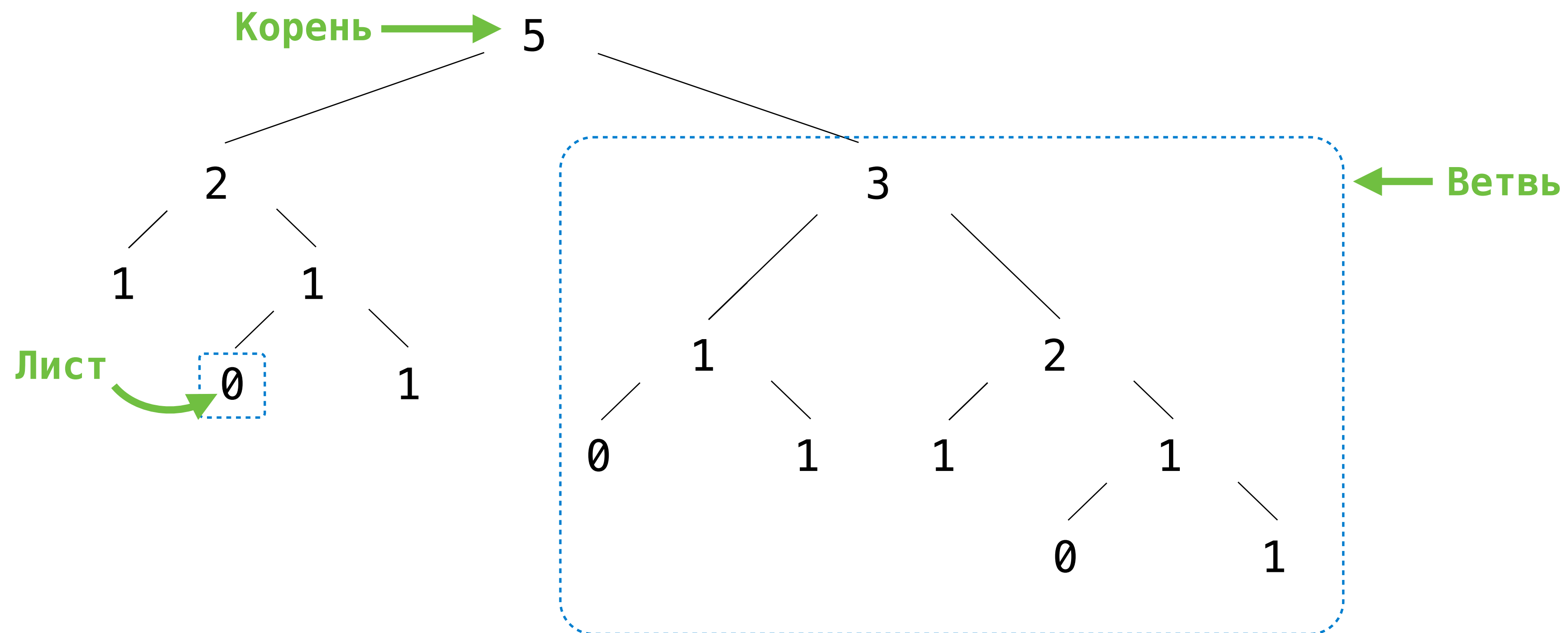
Абстракция дерева



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

Дерево без ветвей называют листом.

Абстракция дерева

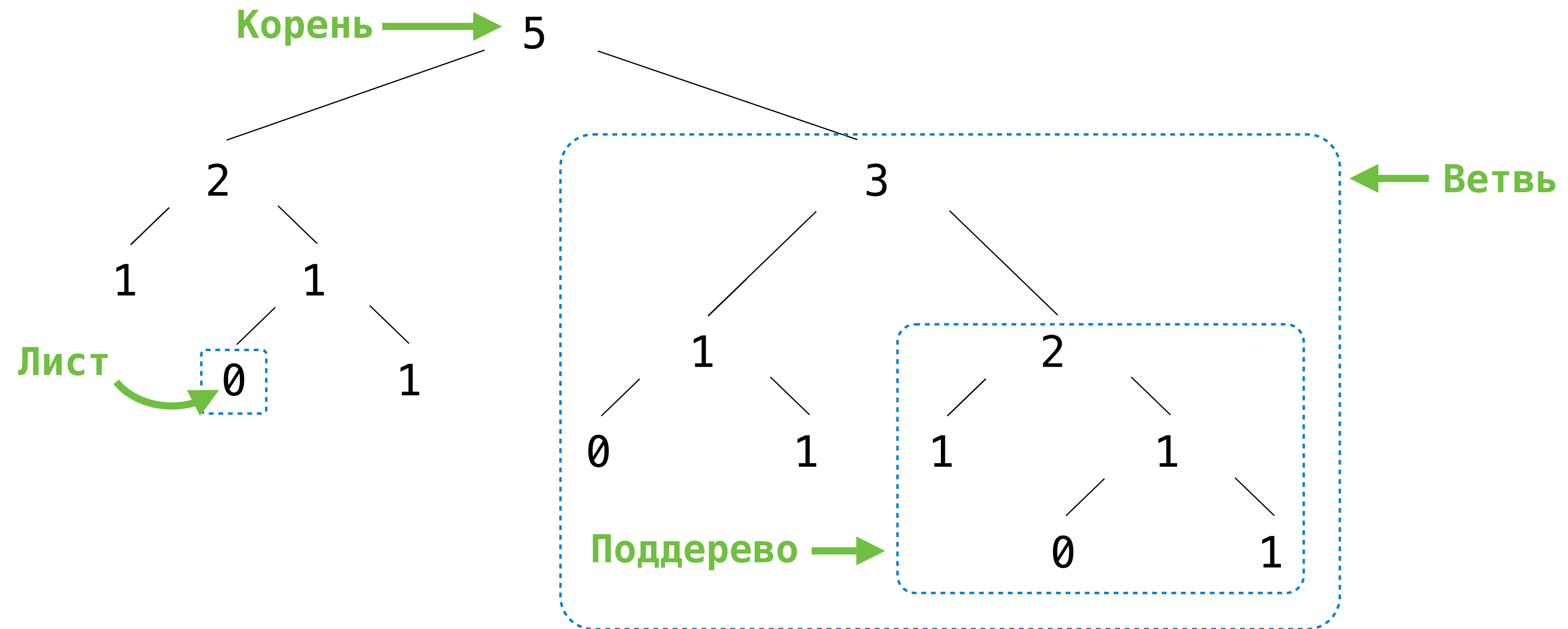


Дерево содержит корневое значение и набор ветвей; каждая ветвь – тоже дерево.

Дерево без ветвей называют листом.

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

Абстракция дерева

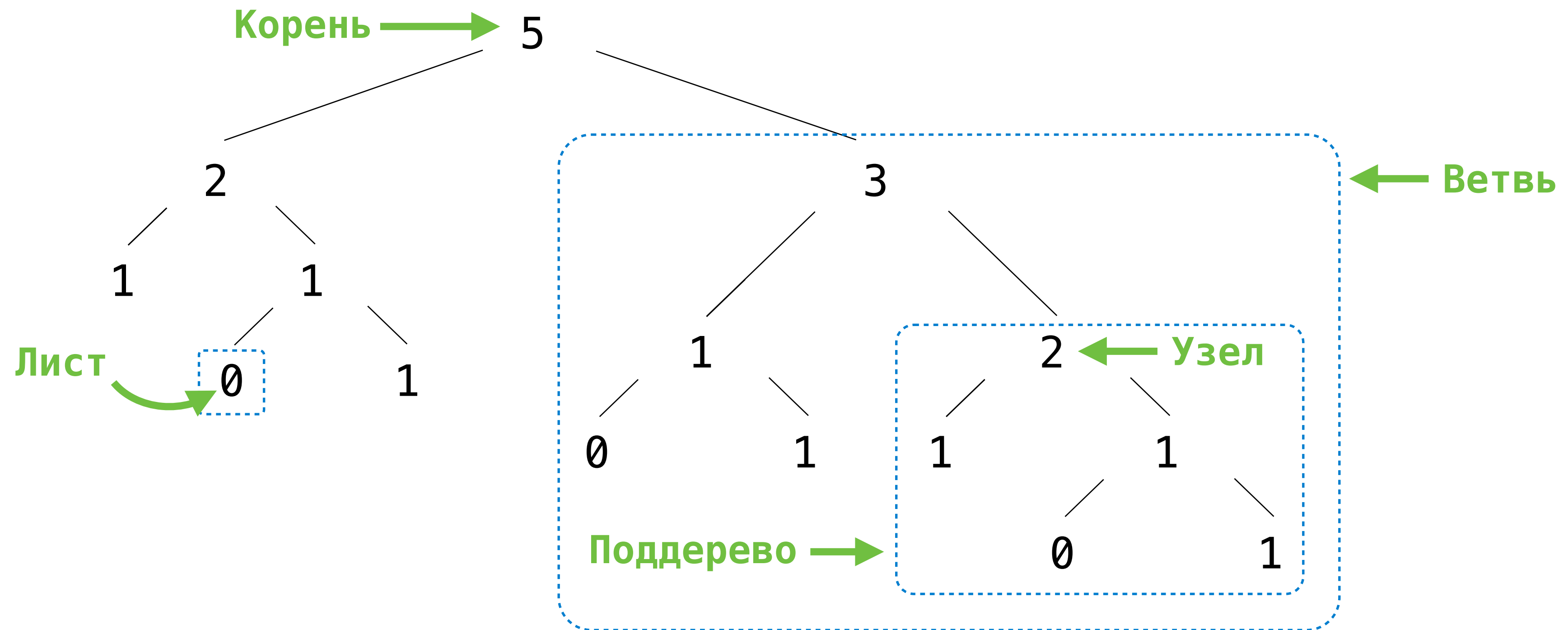


Дерево содержит корневое значение и набор ветвей; каждая ветвь – тоже дерево.

Дерево без ветвей называют листом.

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

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.

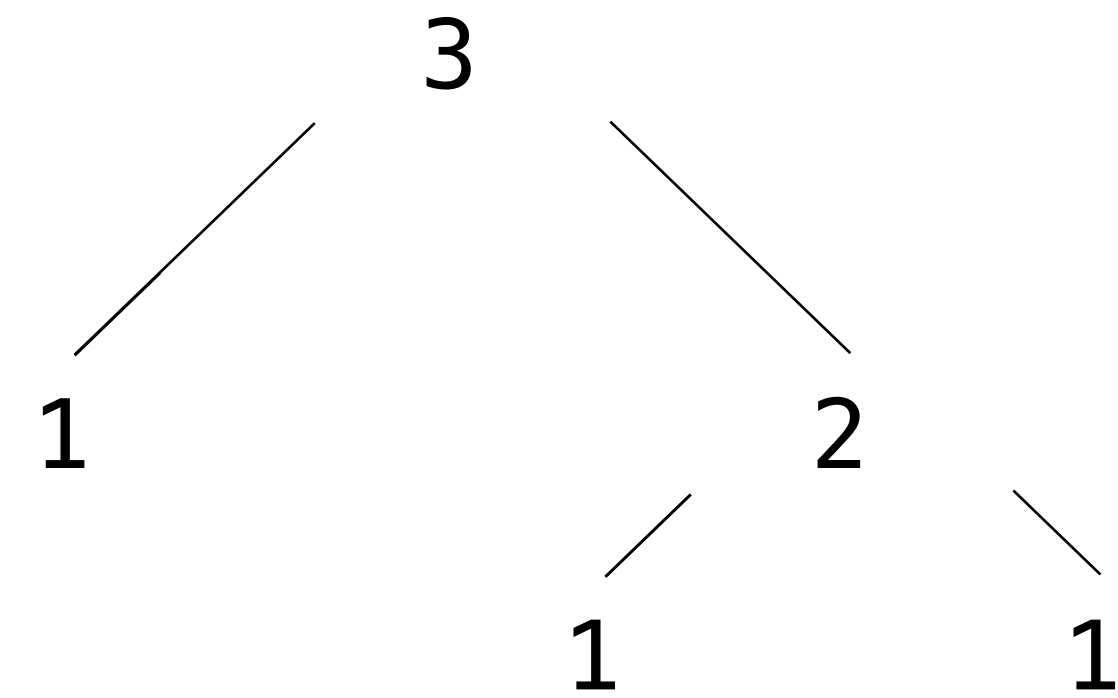
Создание абстракции корневого дерева

Создание абстракции корневого дерева

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

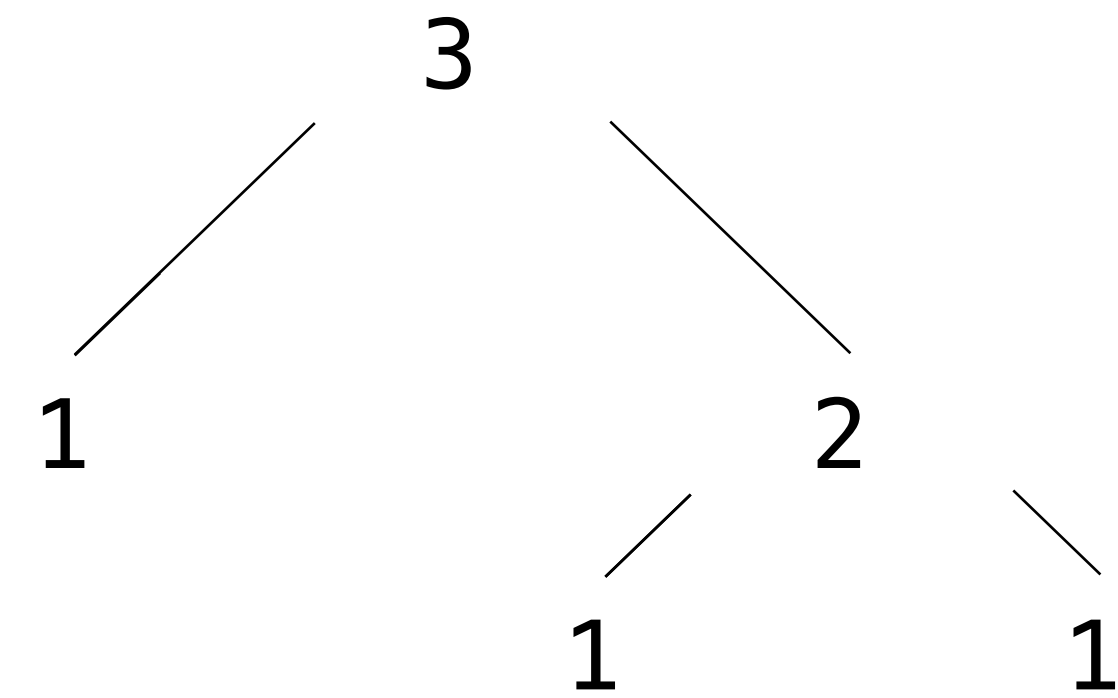
Создание абстракции корневого дерева

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



Создание абстракции корневого дерева

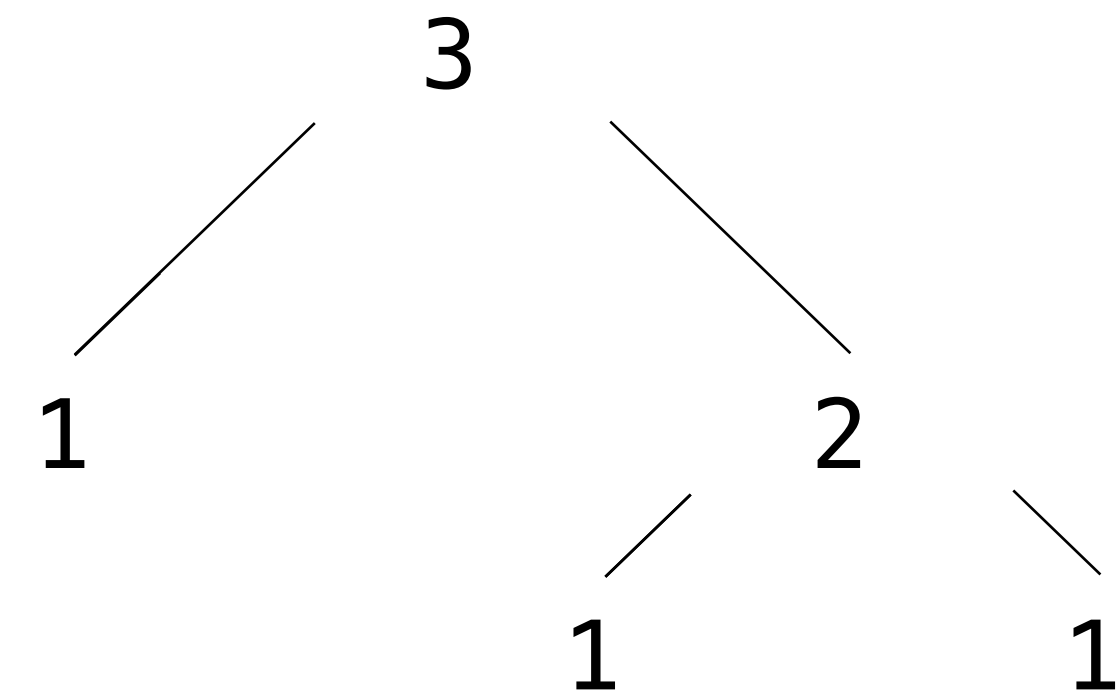
Дерево содержит корневое значение и набор ветвей; каждая ветвь – тоже дерево.



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

Создание абстракции корневого дерева

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

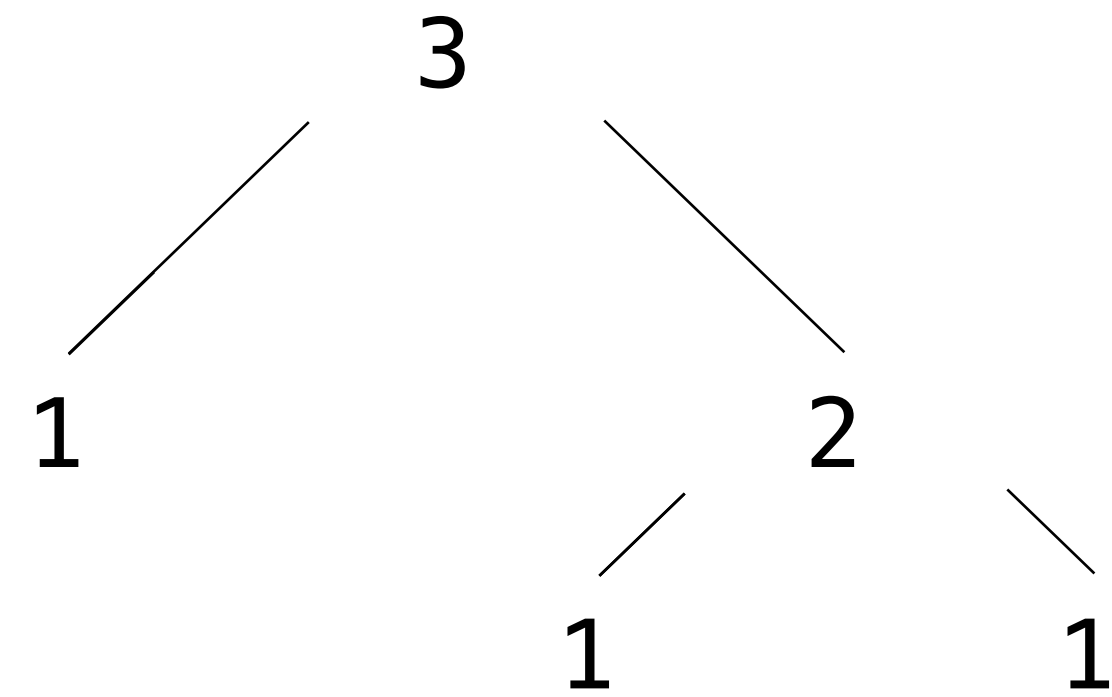


```
>>> tree(3, [tree(1),  
...      tree(2, [tree(1),  
...              tree(1)])])  
[3, [1], [2, [1], [1]]]
```

Создание абстракции корневого дерева

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

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

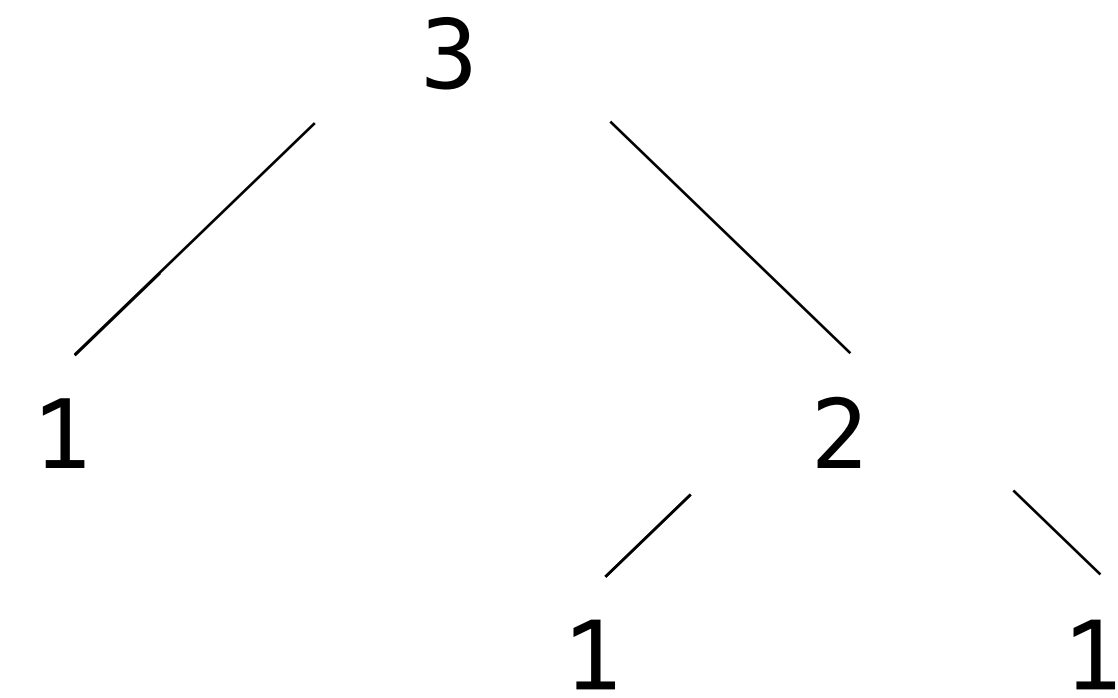


```
>>> tree(3, [tree(1),  
...         tree(2, [tree(1),  
...                   tree(1)])])  
[3, [1], [2, [1], [1]]]
```

Создание абстракции корневого дерева

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

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



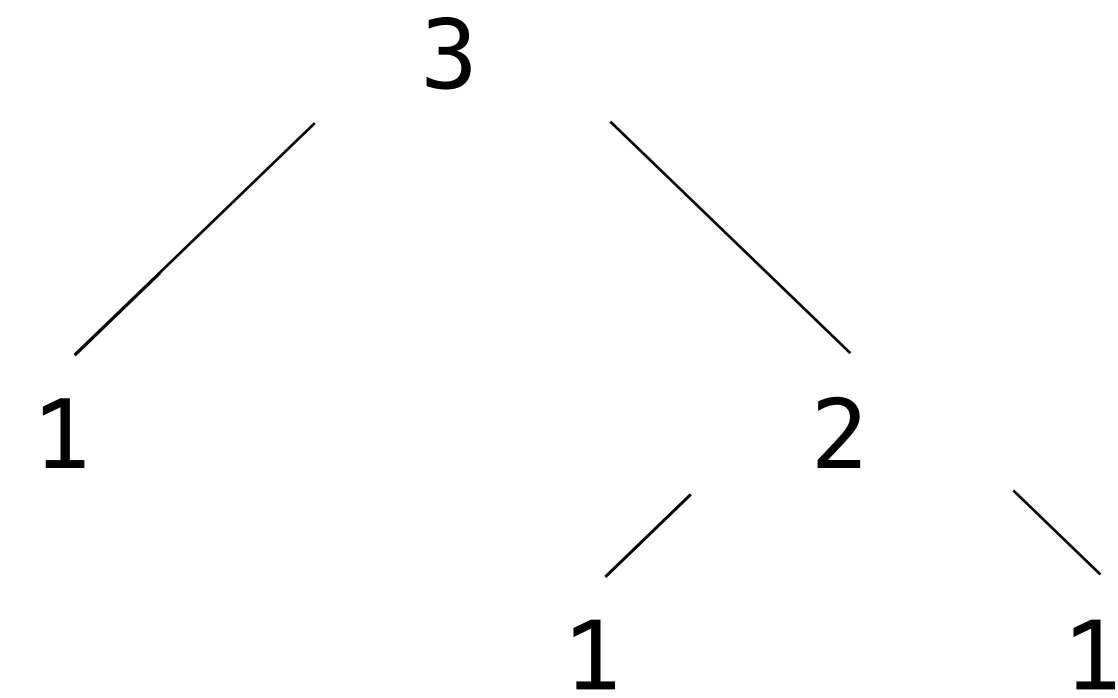
```
>>> tree(3, [tree(1),  
...         tree(2, [tree(1),  
...                 tree(1)])])  
[3, [1], [2, [1], [1]]]
```


Создание абстракции корневого дерева

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

```
def label(tree):
```

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



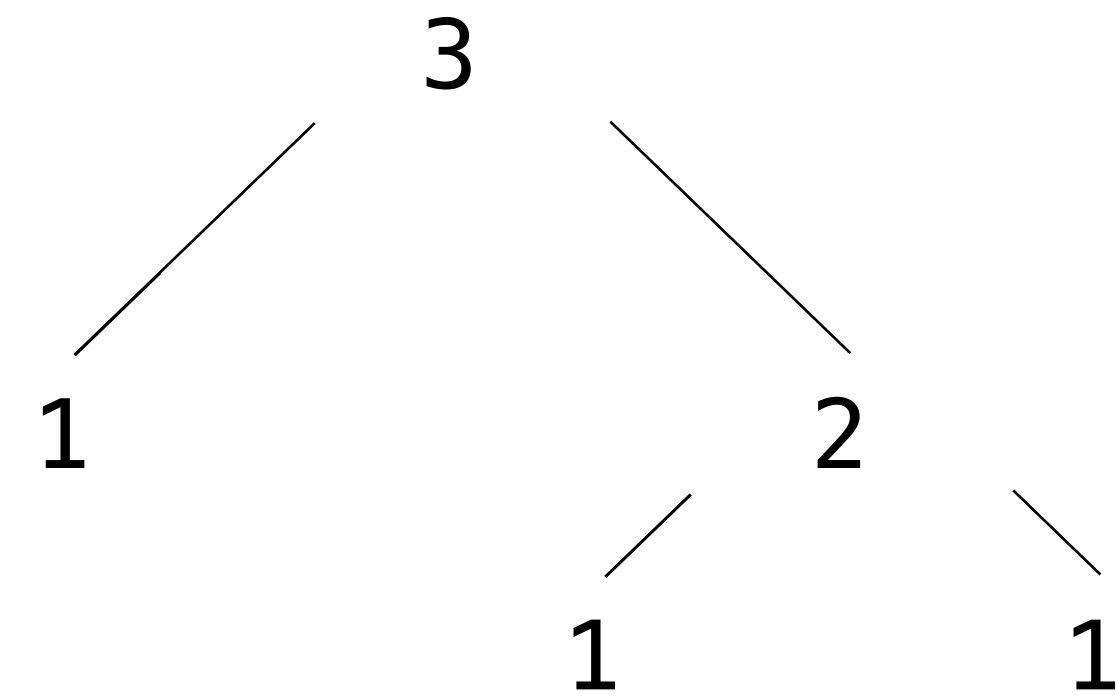
```
>>> tree(3, [tree(1),  
...         tree(2, [tree(1),  
...                   tree(1)])])  
[3, [1], [2, [1], [1]]]
```

Создание абстракции корневого дерева

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

```
def label(tree):  
    return tree[0]
```

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]]]
```

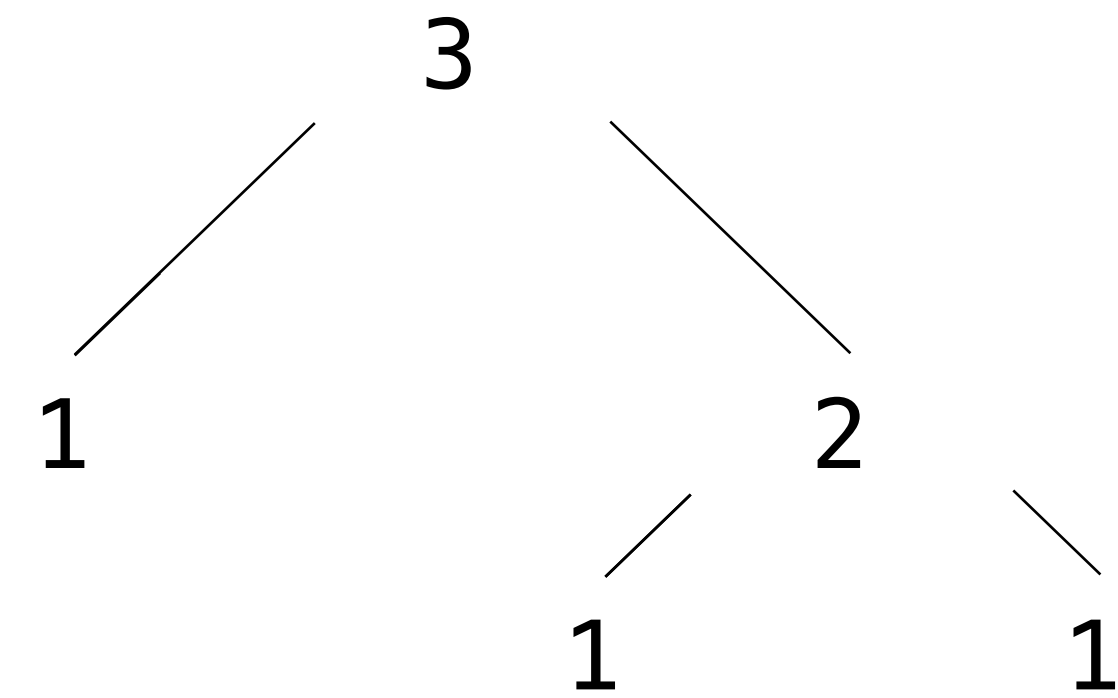
Создание абстракции корневого дерева

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

```
def label(tree):  
    return tree[0]
```

```
def branches(tree):
```

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]]]
```

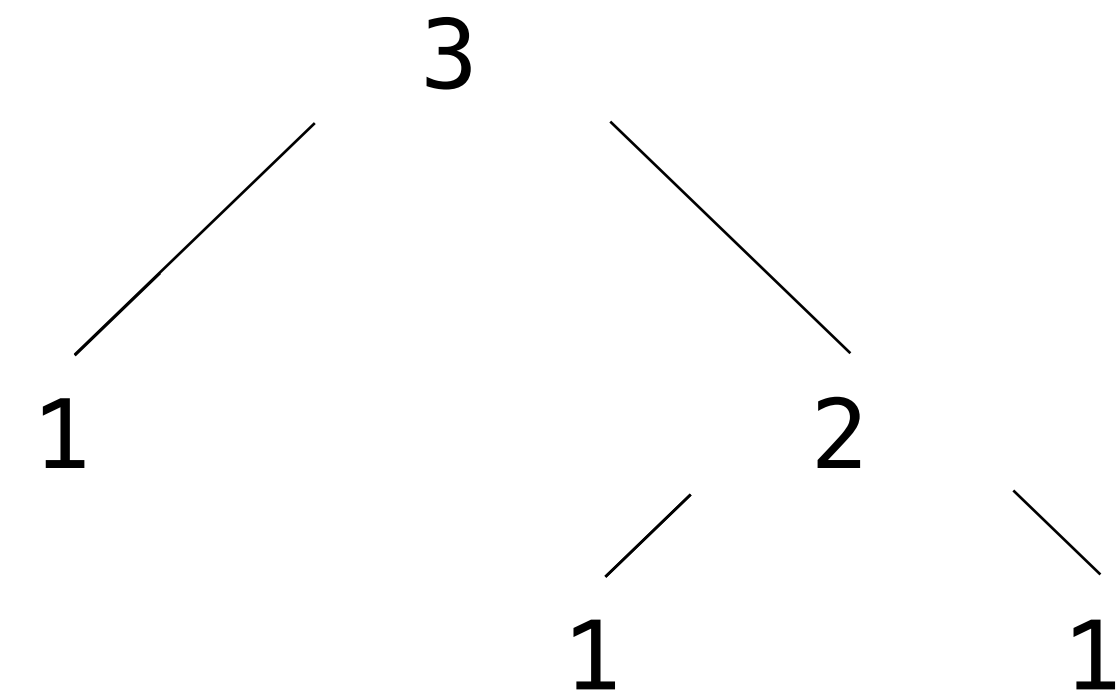
Создание абстракции корневого дерева

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

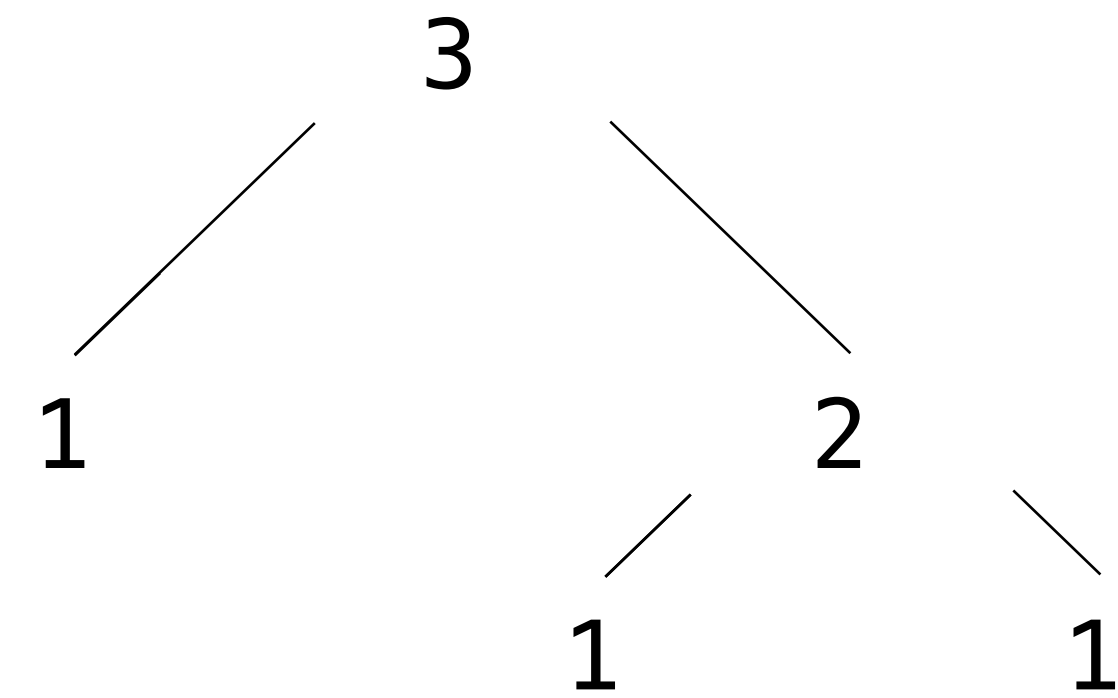
Создание абстракции корневого дерева

```
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:]
```

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

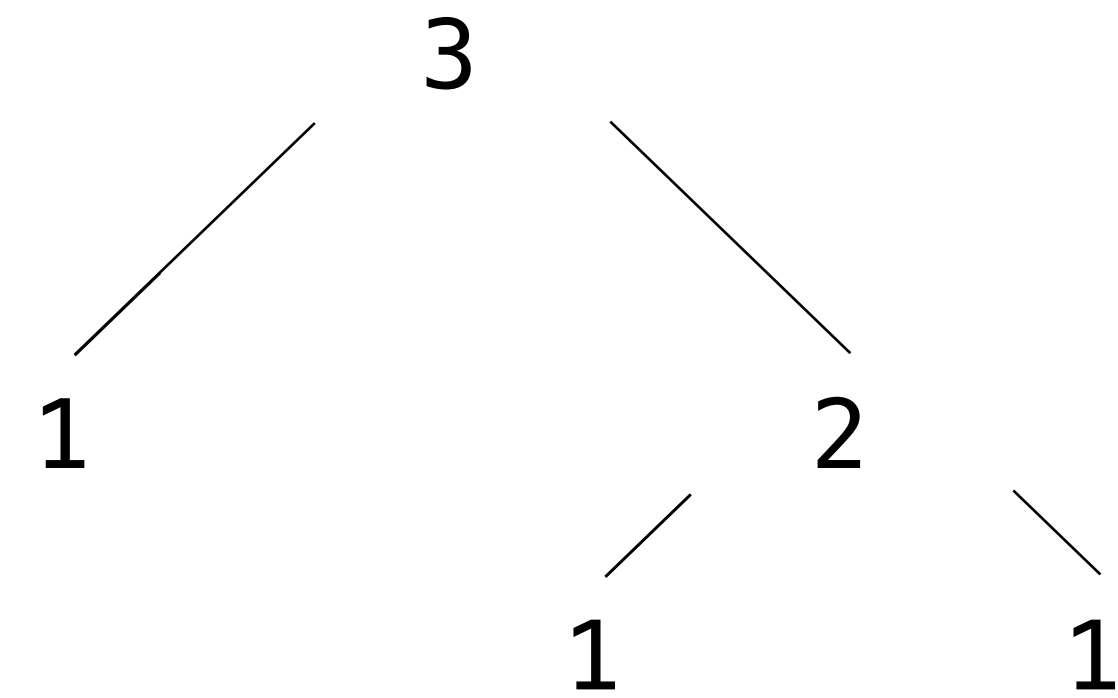
```
def tree(label, branches=[]):  
    for branch in branches:  
        assert is_tree(branch)  
    return [label] + list(branches)
```

```
def label(tree):  
    return tree[0]
```

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

Список из
последовательности
ветвей

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)
```

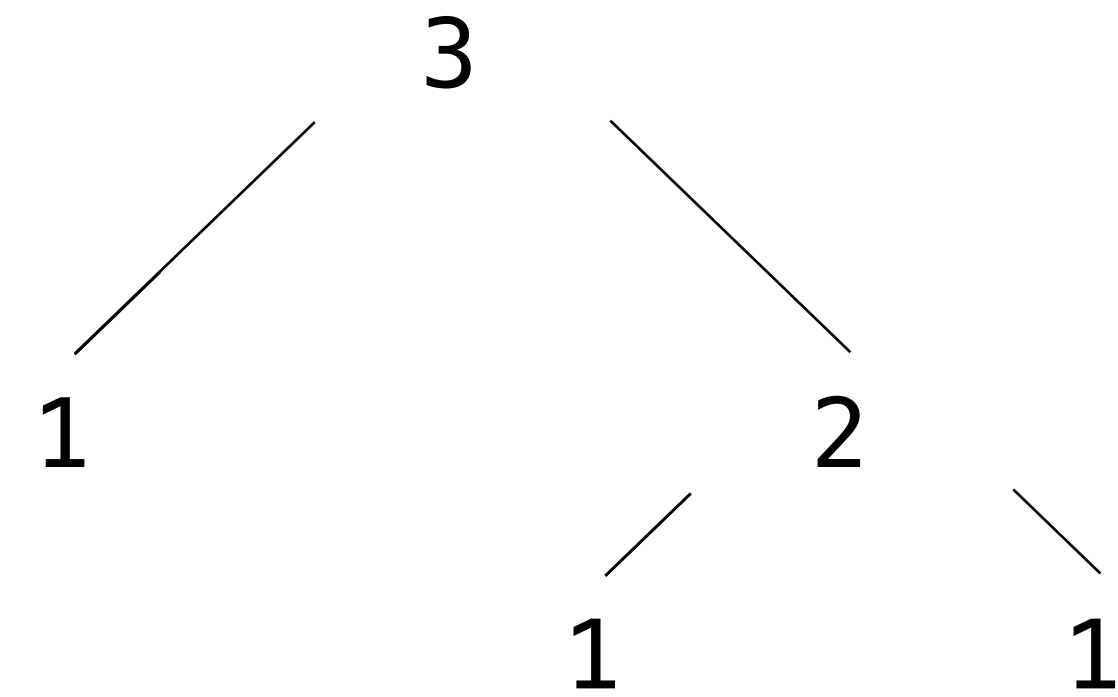
Ветви – это деревья!

```
def label(tree):  
    return tree[0]
```

Список из последовательности ветвей

```
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]]]
```

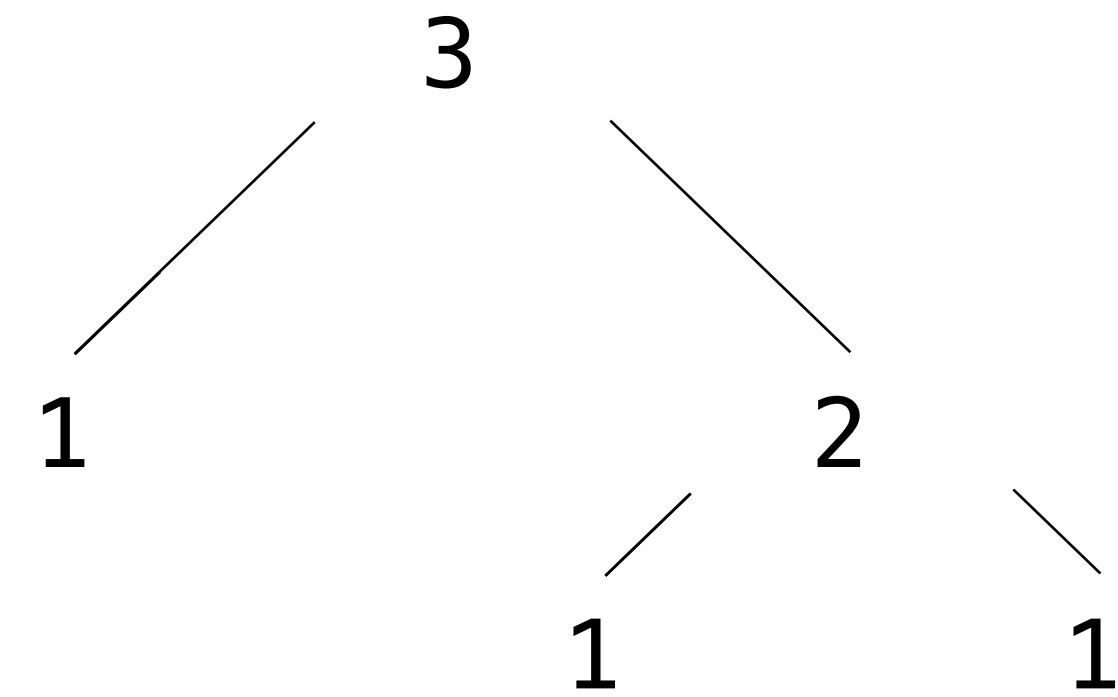
Создание абстракции корневого дерева

```
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

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)
```

Ветви – это деревья!

```
def label(tree):  
    return tree[0]
```

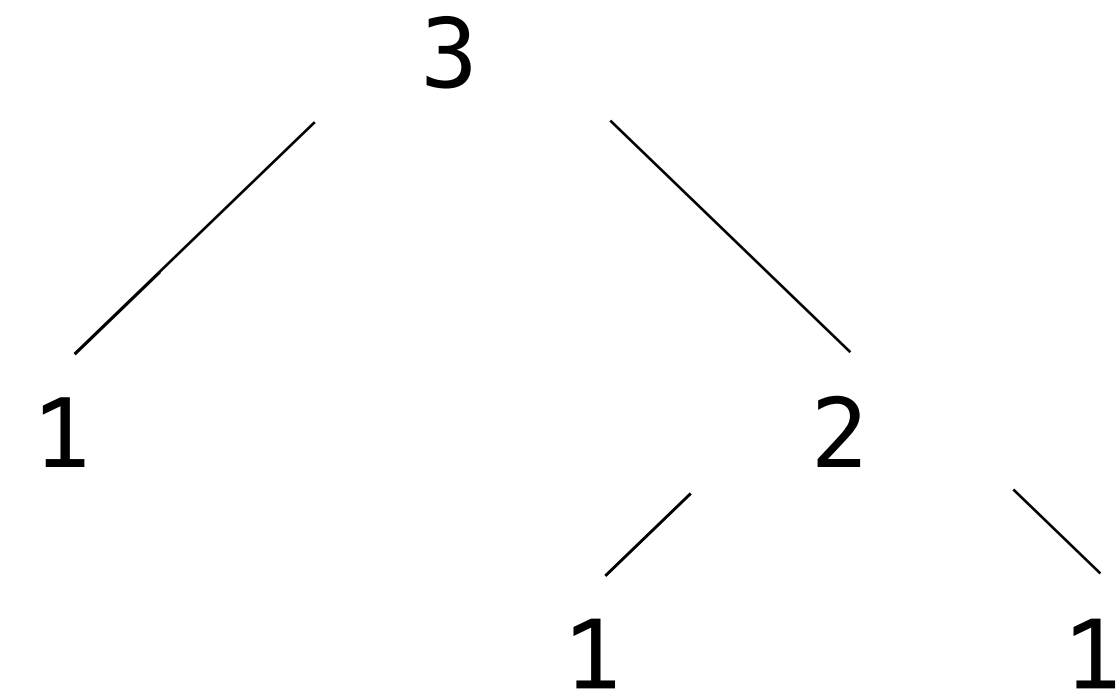
Список из последовательности ветвей

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

Проверяет, что дерево сделано на списках

```
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
```

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)
```

Ветви – это деревья!

```
def label(tree):  
    return tree[0]
```

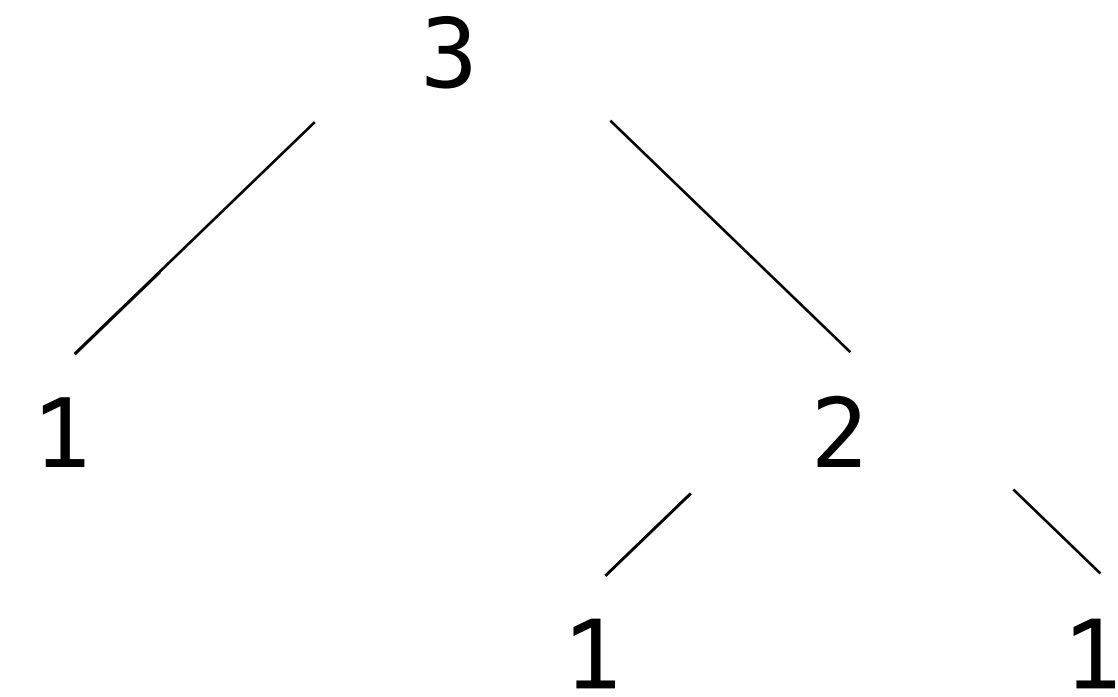
Список из последовательности ветвей

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

Проверяет, что дерево сделано на списках

```
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
```

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)
```

Создание абстракции корневого дерева

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

Ветви – это деревья!

```
def label(tree):  
    return tree[0]
```

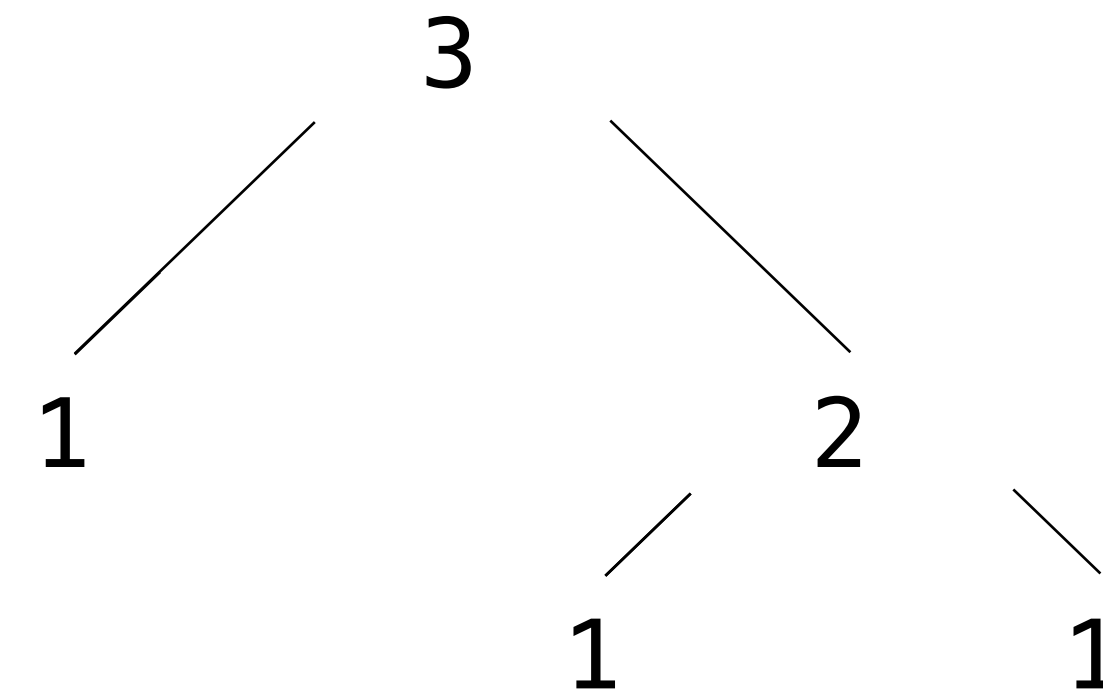
Список из последовательности ветвей

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

Проверяет, что дерево сделано на списках

```
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
```

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

Рекурсивная обработка деревьев

Обработка листа зачастую является базовым случаем функции обработки дерева.

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.

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)
```

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

Вопрос

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

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.

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]], [])
```

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]
```

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]], [])
```

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]
```

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], [])
```


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]
```

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]
    """
```

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 _____
```

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)], [])
```