при поддержке
{a}вижн

Лекция 19

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

Связные списки

Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.

Структура связного списка

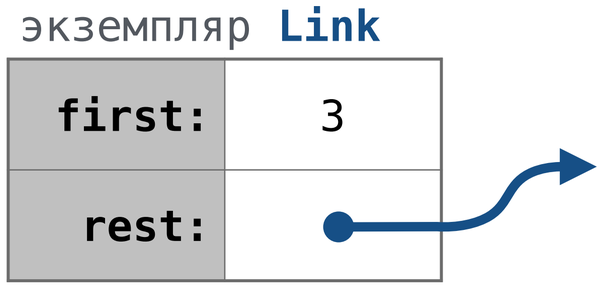
Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.

3 , 4 , 5

Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.

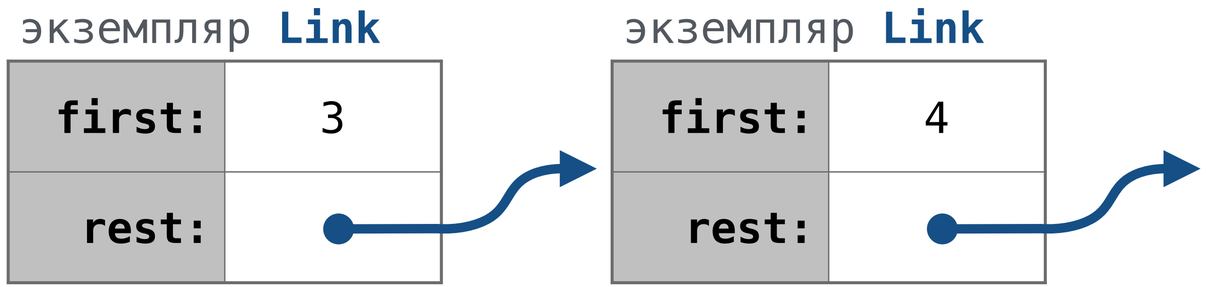
3 , 4 , 5



Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.

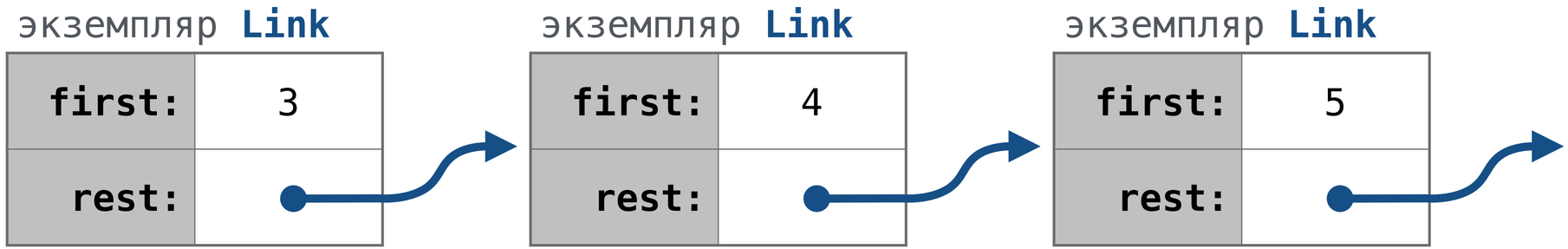
3 , 4 , 5



Структура связного списка

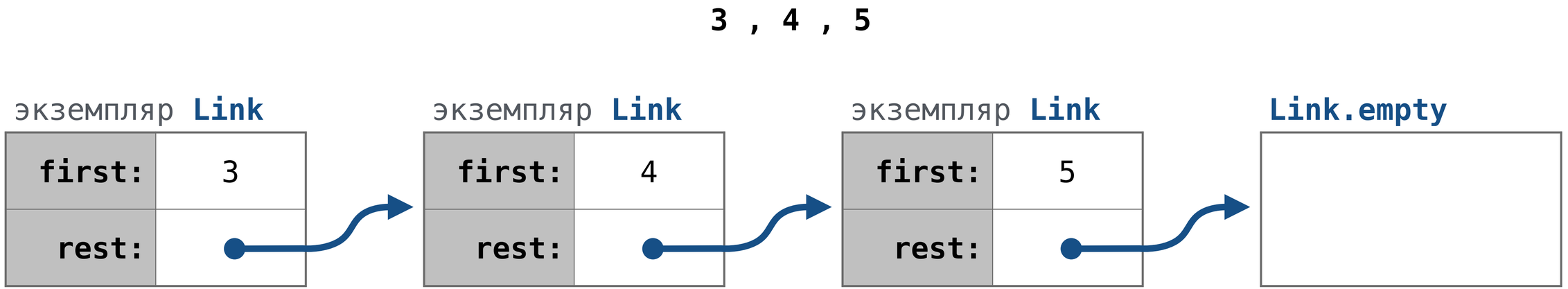
Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.

3 , 4 , 5



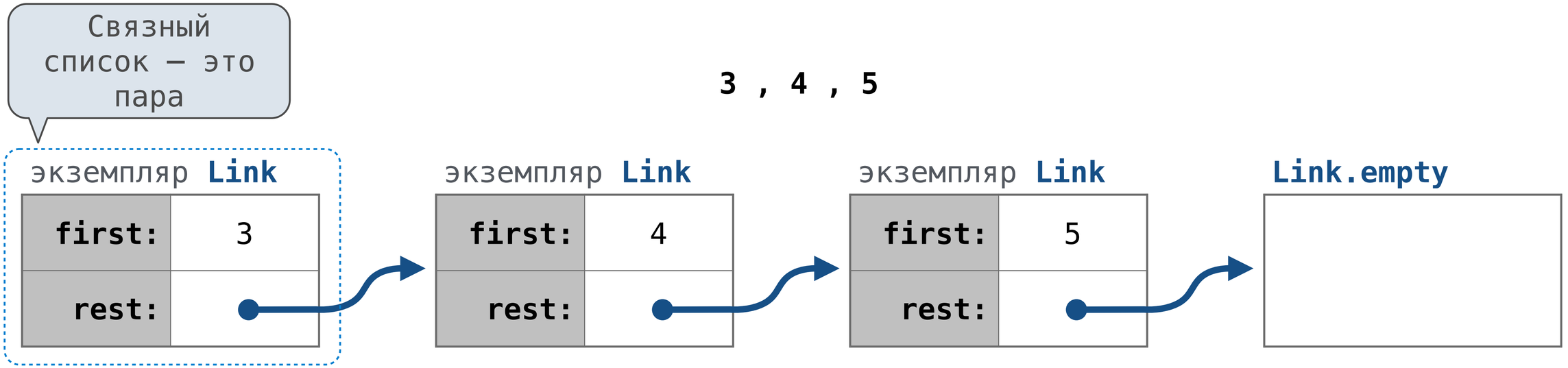
Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



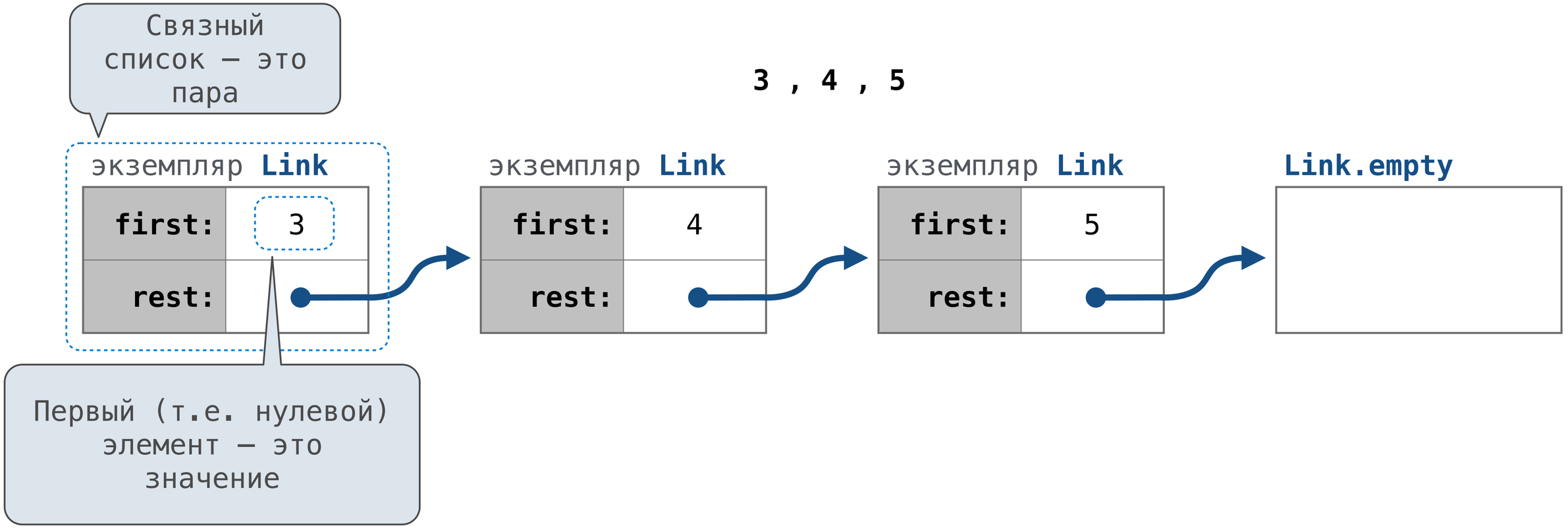
Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



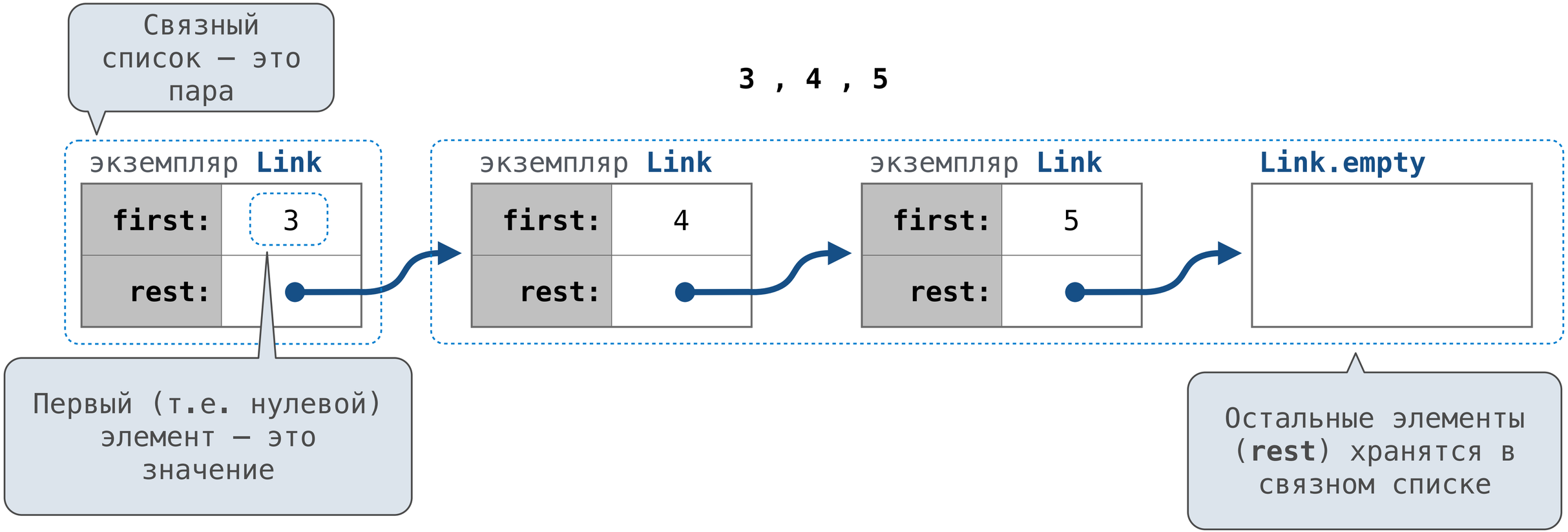
Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



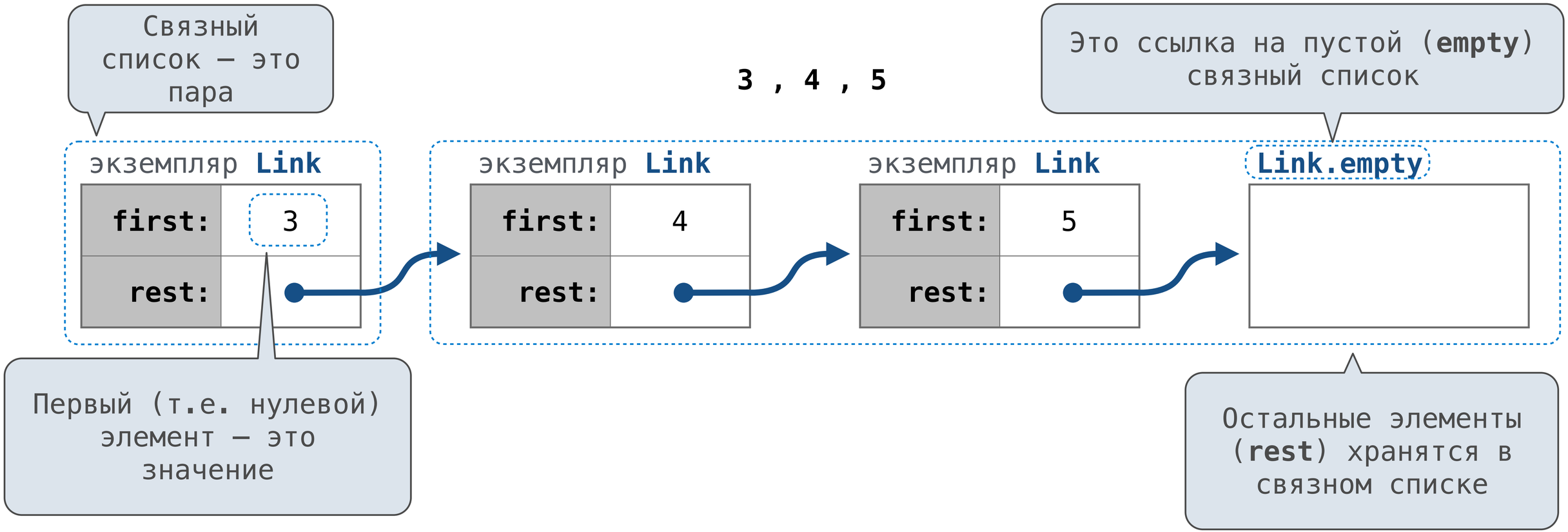
Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



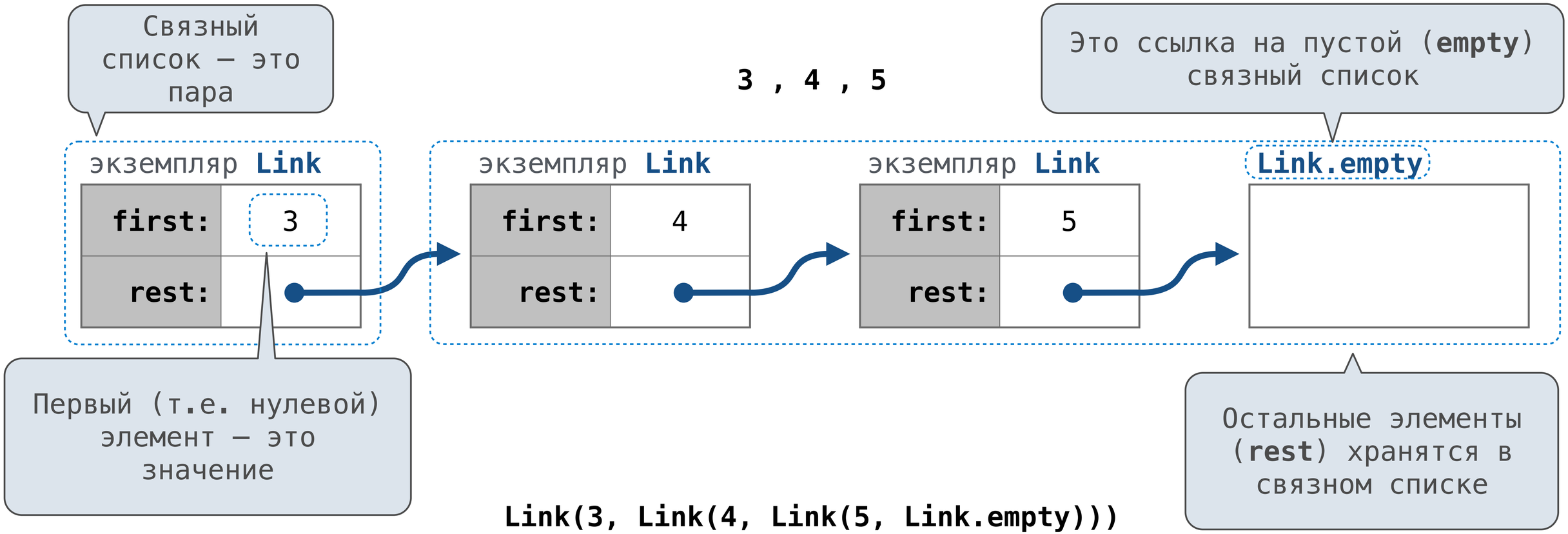
Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



Структура связного списка

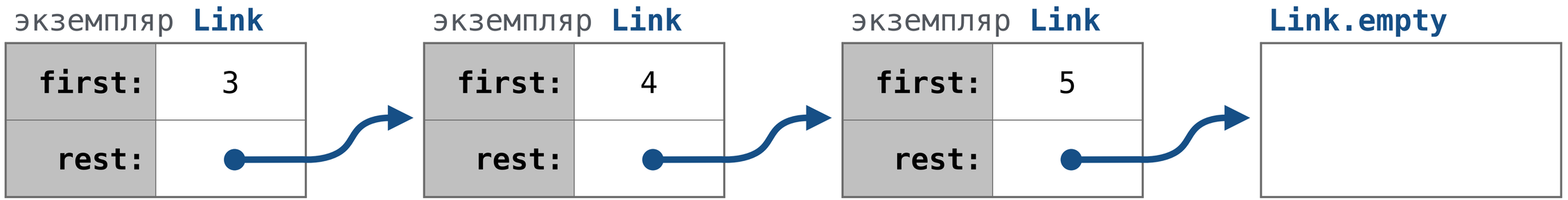
Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.

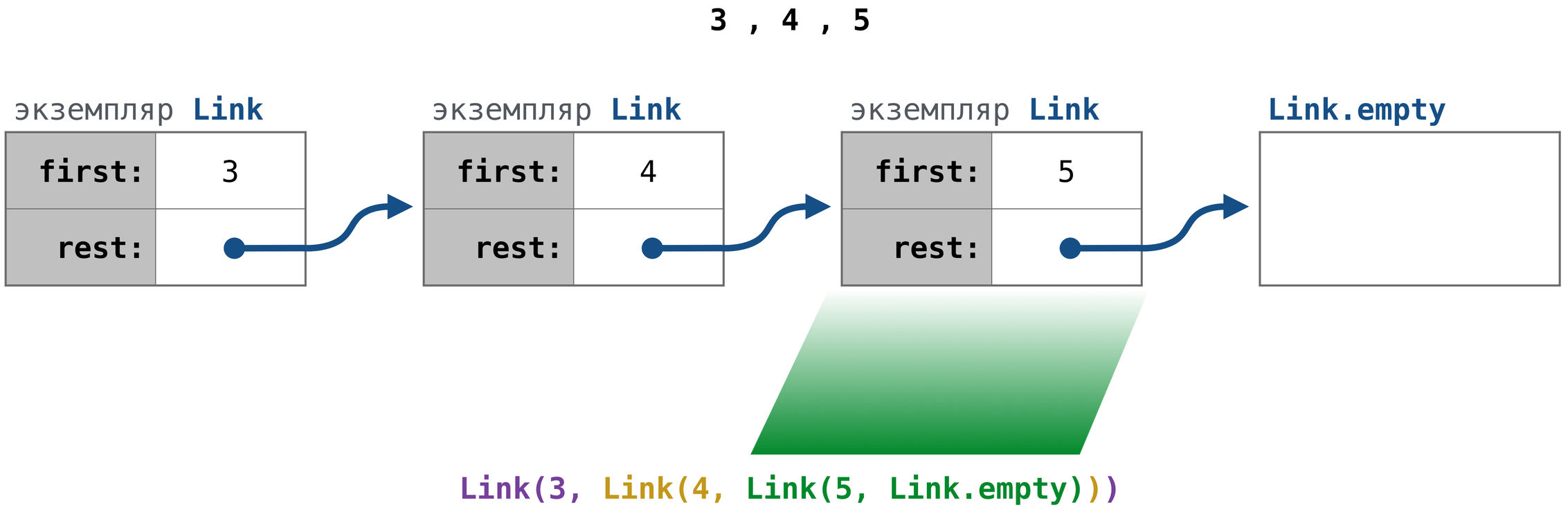
3 , 4 , 5




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

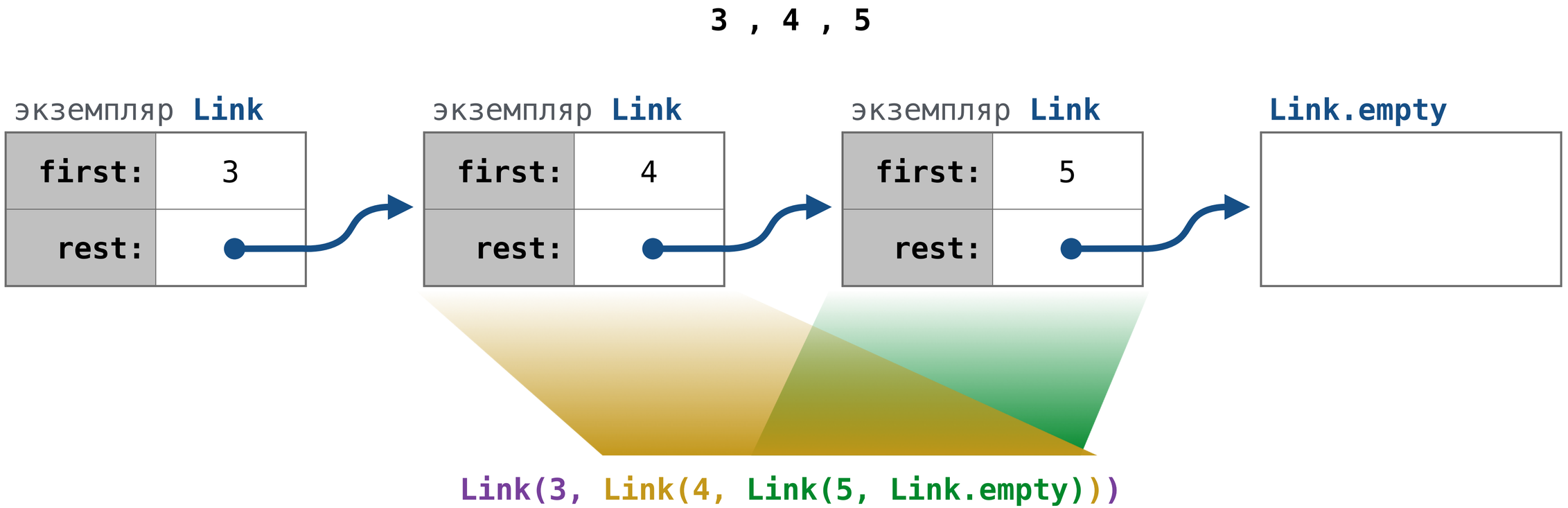
Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



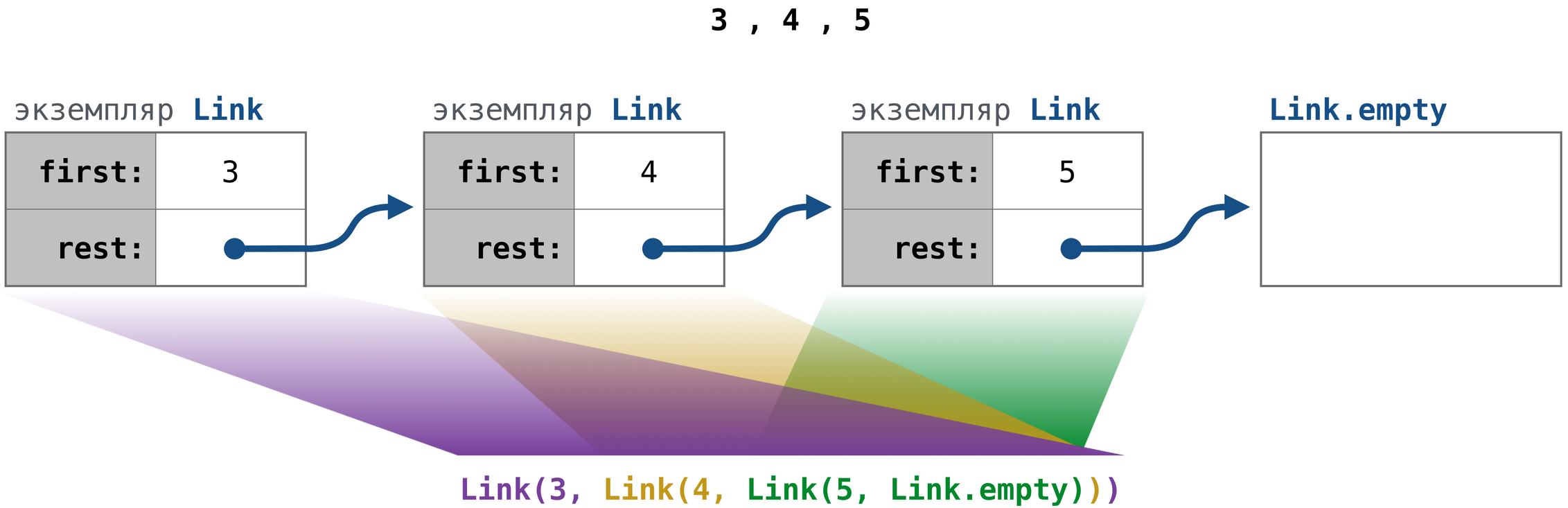
Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



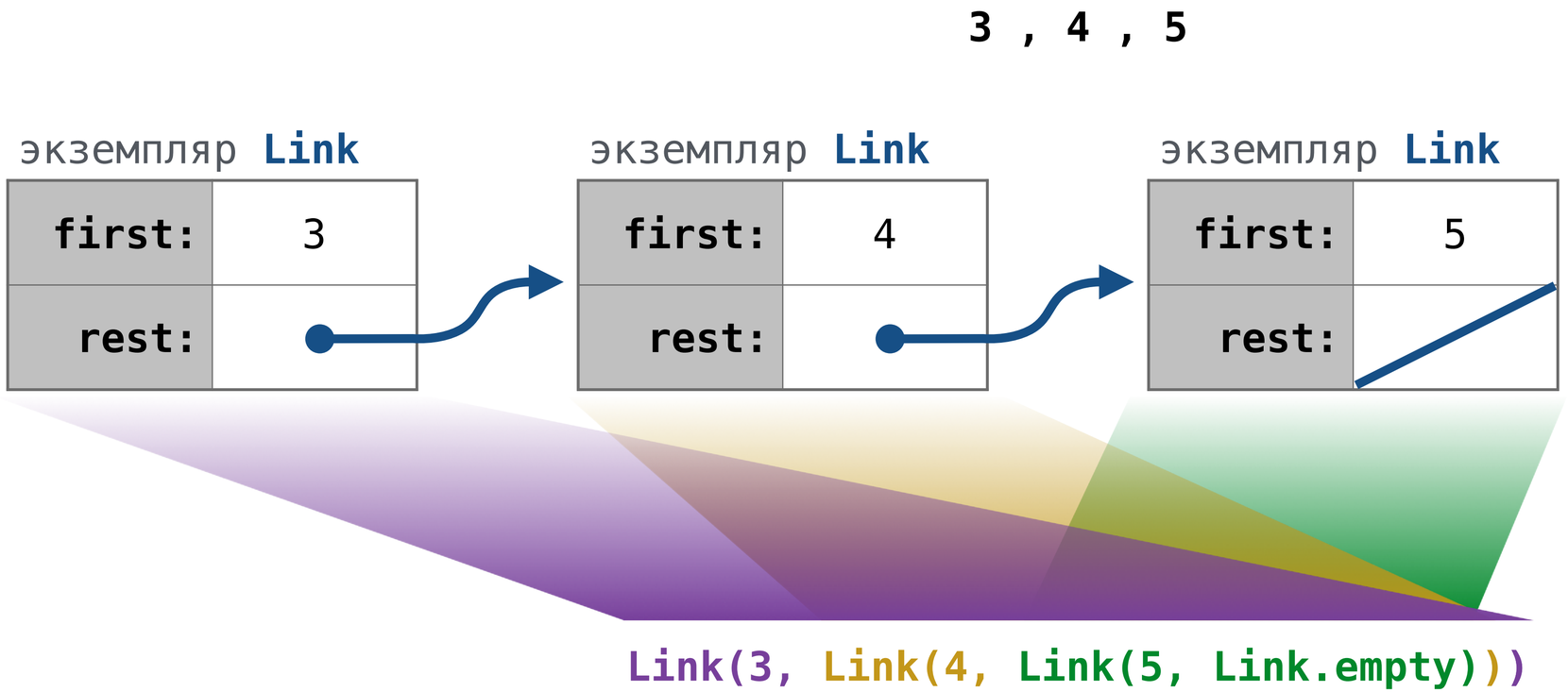
Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



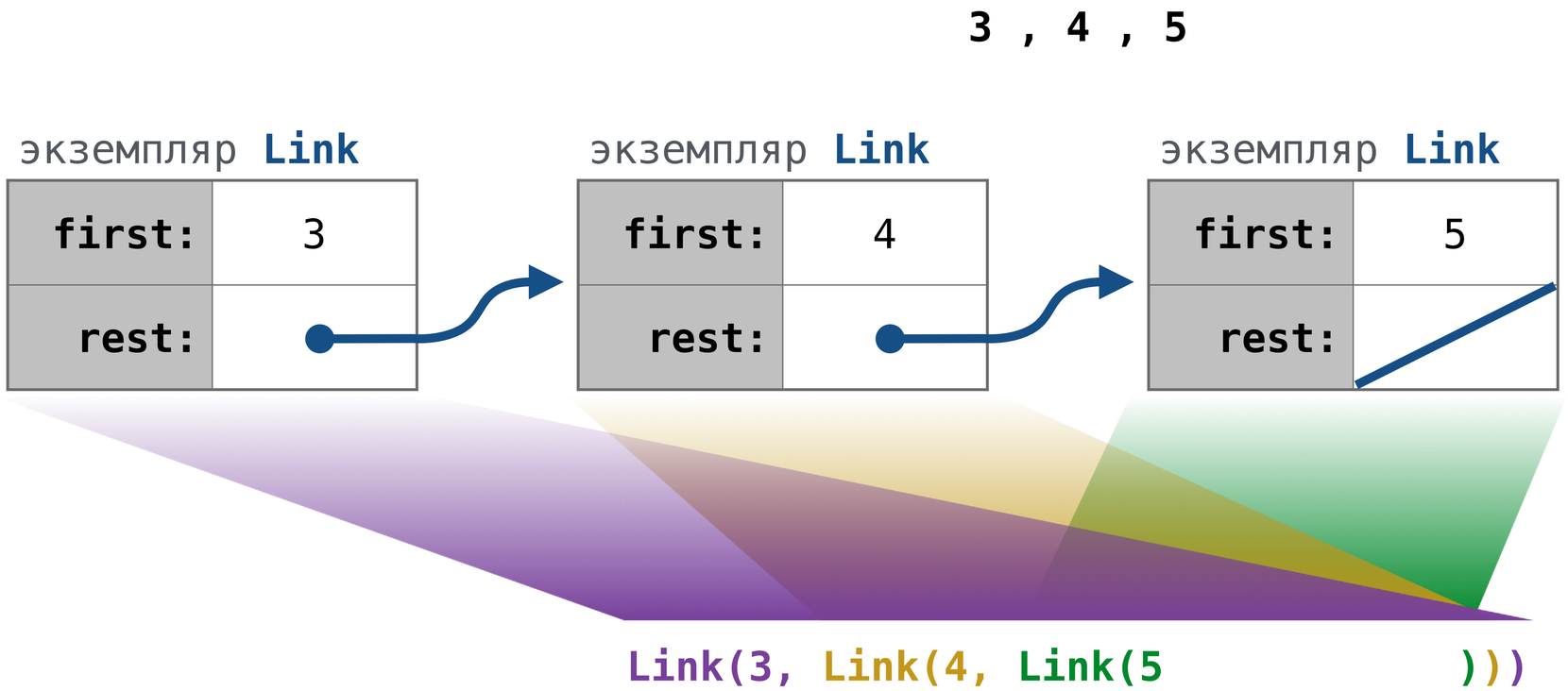
Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



Структура связного списка

Связный список — это либо пустой список, либо пара — элемент и ссылка на остаток списка.



Класс Link

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

Класс Link

Класс Link: атрибуты передаются в __init__

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

Класс Link

Класс Link: атрибуты передаются в __init__

```
class Link:
```

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

Класс Link

Класс Link: атрибуты передаются в __init__

```
class Link:


    def __init__(self, first, rest=empty):
```

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

Класс Link

Класс Link: атрибуты передаются в __init__

```
class Link:


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

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

Класс Link

Класс Link: атрибуты передаются в __init__

```
class Link:



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

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

Класс Link

Класс Link: атрибуты передаются в __init__

```
class Link:

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

Является ли rest экземпляром Link

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

Класс Link

Класс Link: атрибуты передаются в __init__

```
class Link:



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

Является ли rest экземпляром Link

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

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

Класс Link

Класс Link: атрибуты передаются в __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
```

Является ли rest экземпляром Link

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

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

Класс Link

Класс Link: атрибуты передаются в __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
```

Последовательность нулевой длины

Является ли rest экземпляром Link

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

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

Класс Link

Класс Link: атрибуты передаются в __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
```

Последовательность нулевой длины

Является ли rest экземпляром Link

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

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

(Пример)

Методы свойств

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
```

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
```

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
```

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> f.second
6
```

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> f.second
6
```

Нет вызова метода!

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> f.second
6
>>> s
```

Нет вызова метода!

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> f.second
6
>>> s
Link(3, Link(6, Link(5)))
```

Нет вызова метода!

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> f.second
6
>>> s
Link(3, Link(6, Link(5)))
```

Нет вызова метода!

Декоратор @property перед описанием безаргументного метода указывает на то, что этот метод будет вызван при любом обращении к этому атрибуту.

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> f.second
6
>>> s
Link(3, Link(6, Link(5)))
```

Нет вызова метода!

Декоратор @property перед описанием безаргументного метода указывает на то, что этот метод будет вызван при любом обращении к этому атрибуту.

Декоратор @<атрибут>.setter применённый к методу, приведёт к тому, что этот метод будет вызван при присвоении этому атрибуту.

Методы свойств

Зачастую необходимо чтобы значения атрибутов экземпляра были непротиворечивы

```
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> f.second
6
>>> s
Link(3, Link(6, Link(5)))
```

Нет вызова метода!

Декоратор @property перед описанием безаргументного метода указывает на то, что этот метод будет вызван при любом обращении к этому атрибуту.

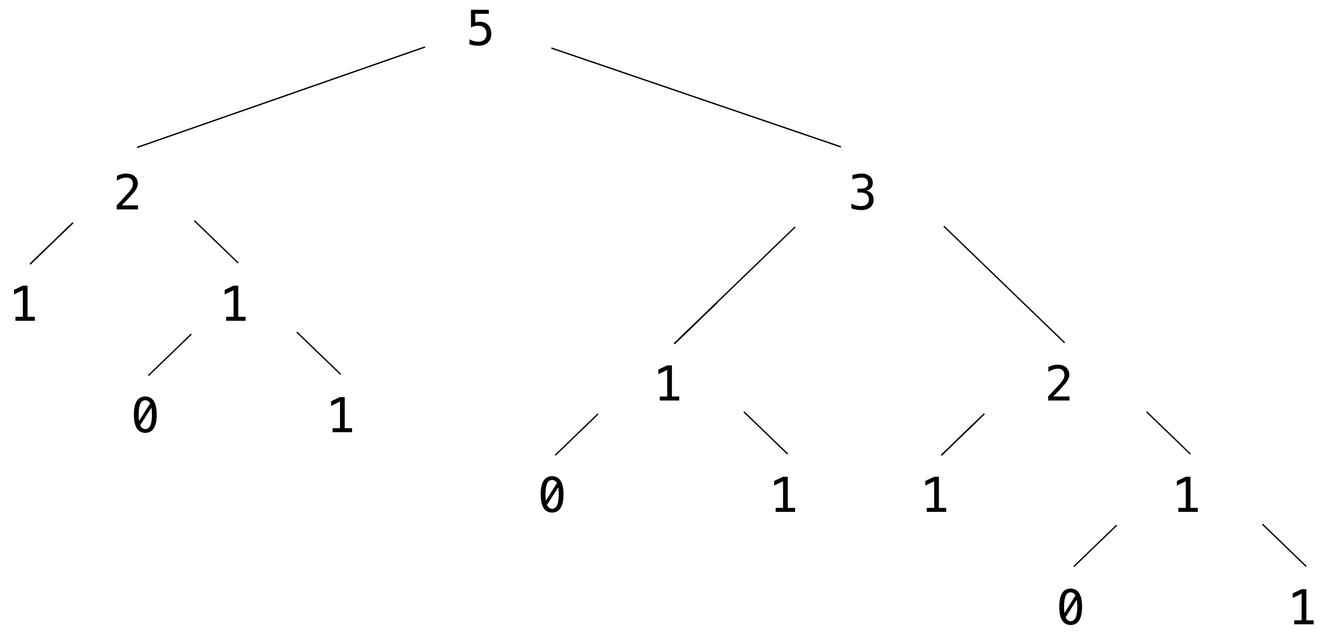
Декоратор @<атрибут>.setter применённый к методу, приведёт к тому, что этот метод будет вызван при присвоении этому атрибуту.

(Пример)

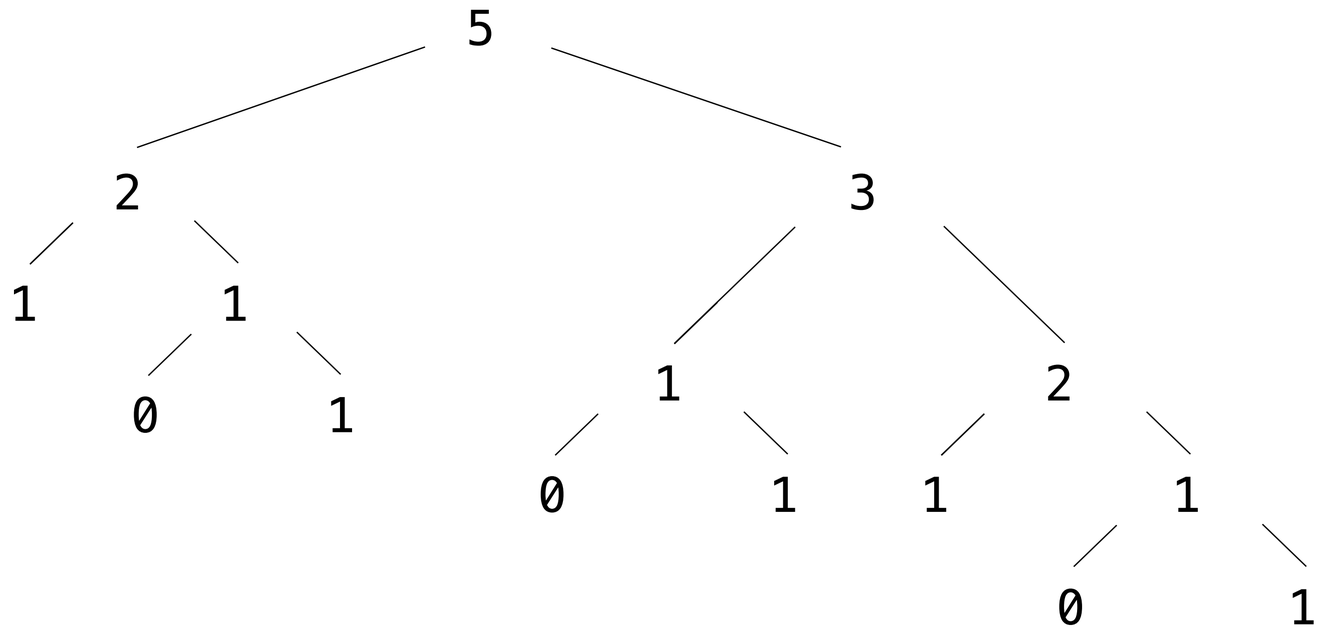
Класс Tree

Абстракция дерева

Абстракция дерева

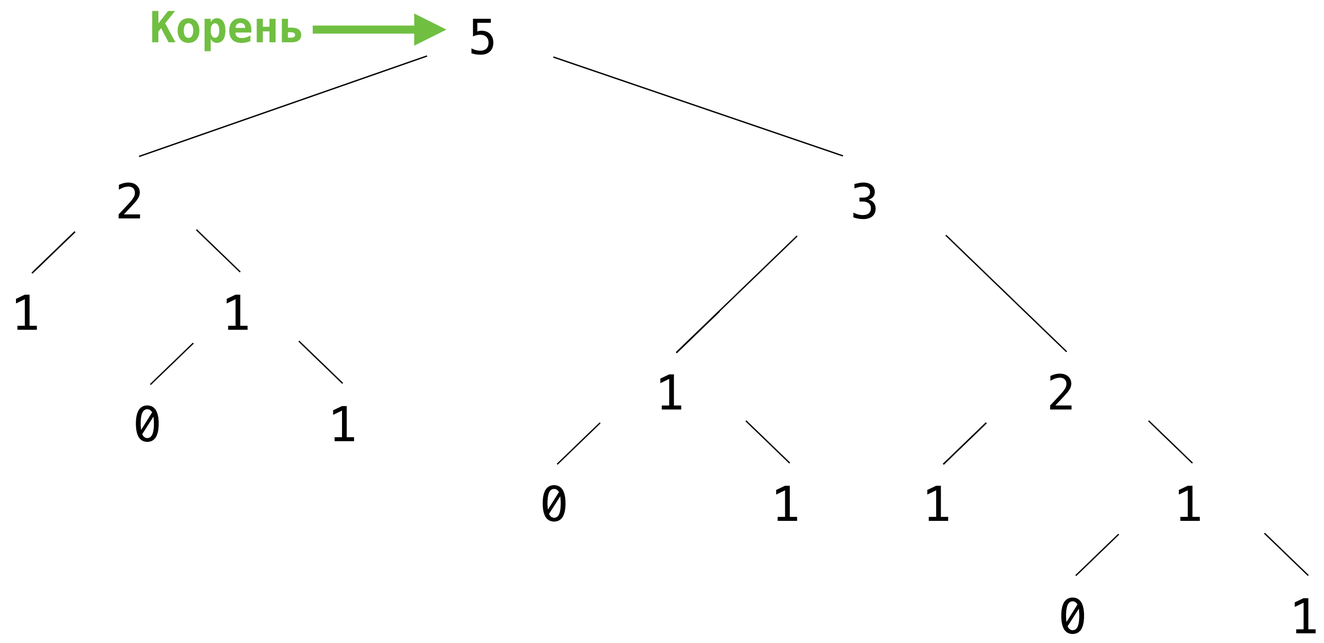


Абстракция дерева



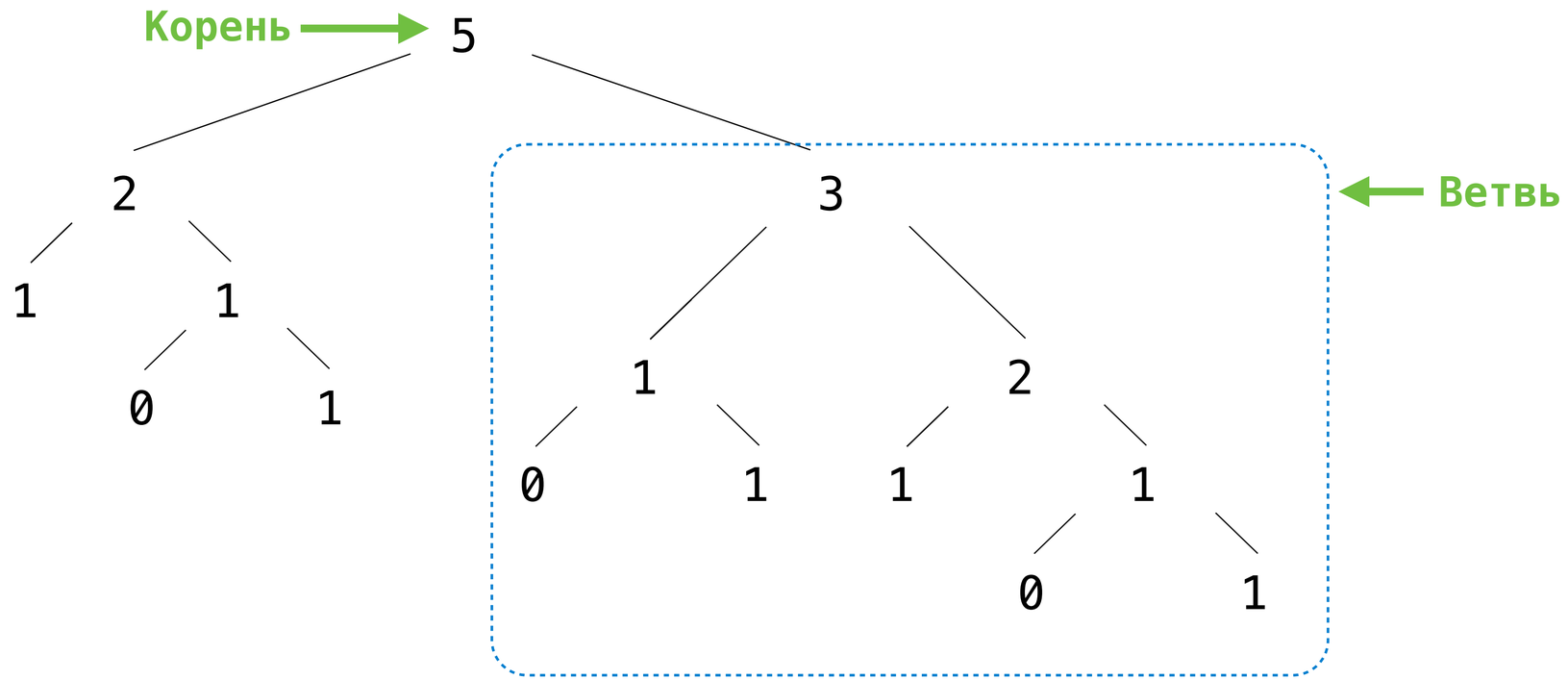
Дерево содержит корневое значение и набор ветвей; каждая ветвь — тоже дерево.

Абстракция дерева



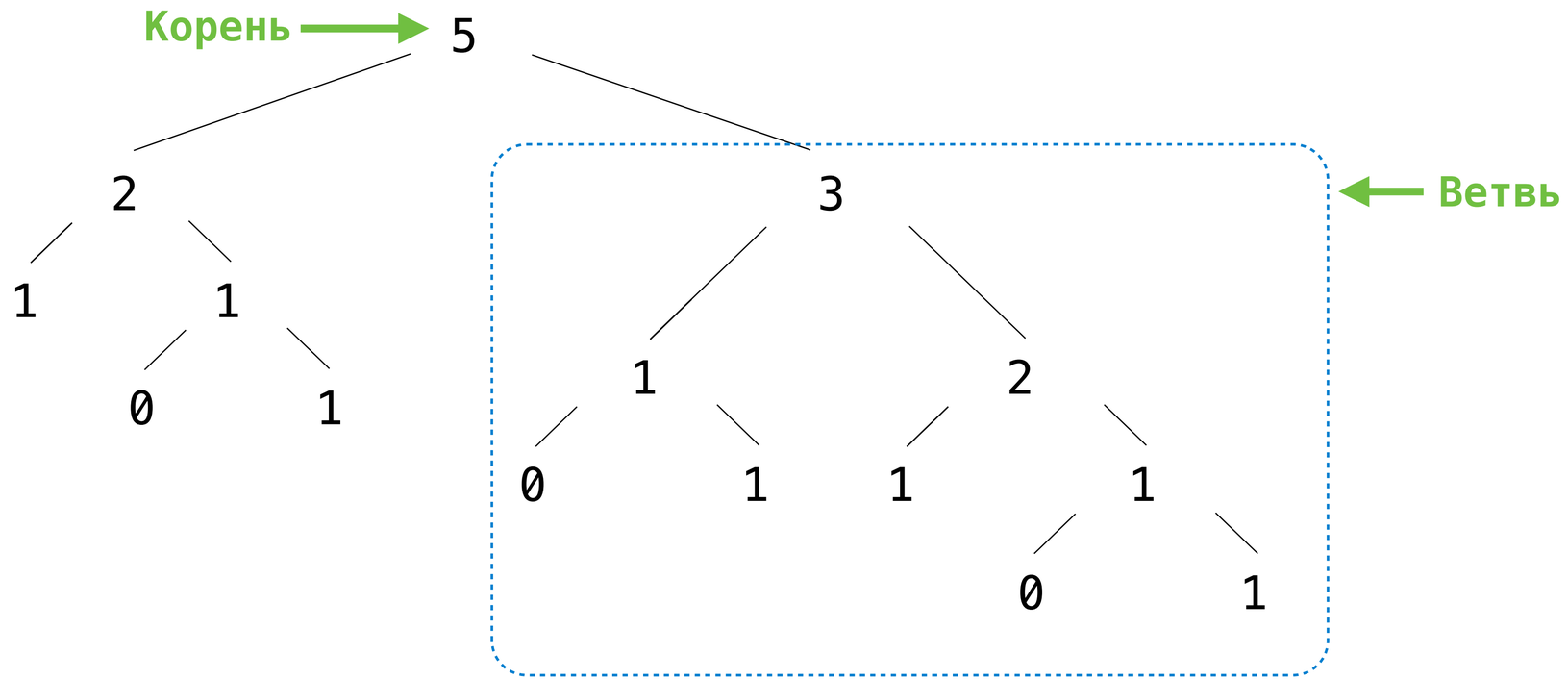
Дерево содержит корневое значение и набор ветвей; каждая ветвь — тоже дерево.

Абстракция дерева



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

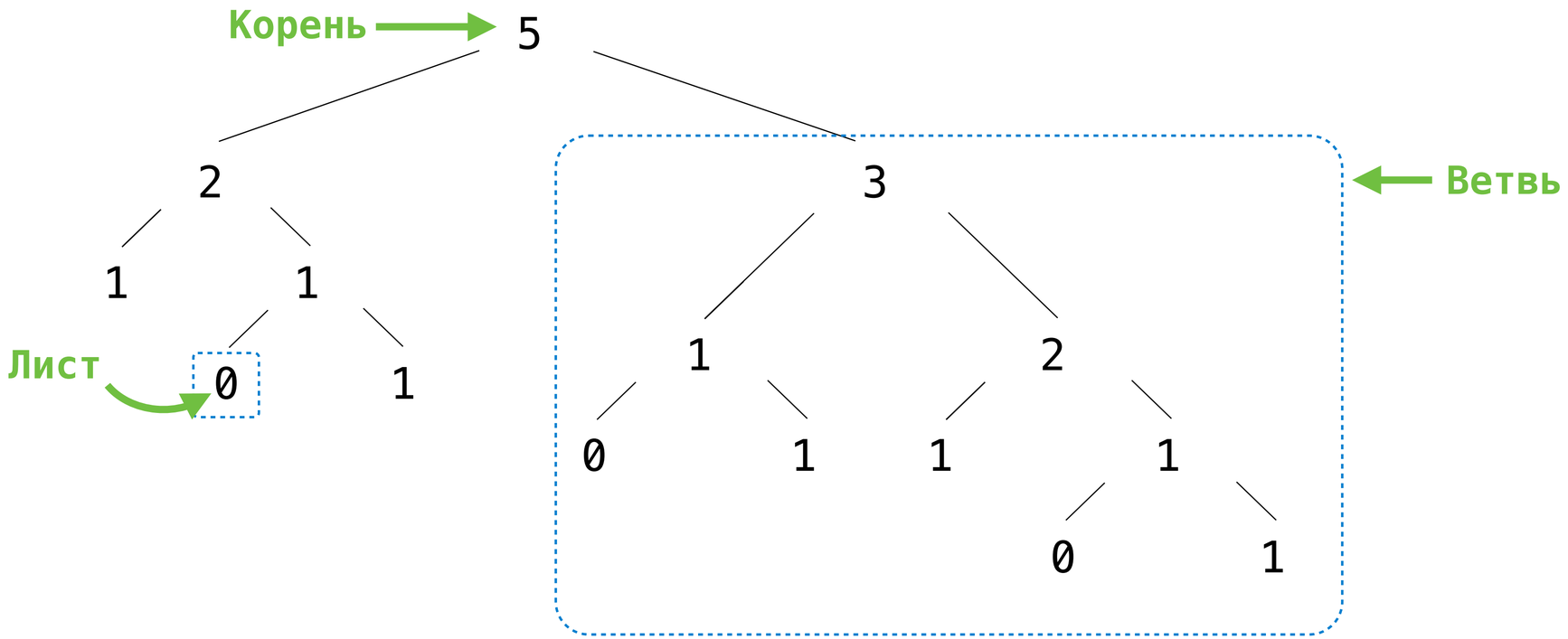
Абстракция дерева



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

Дерево без ветвей называют листом.

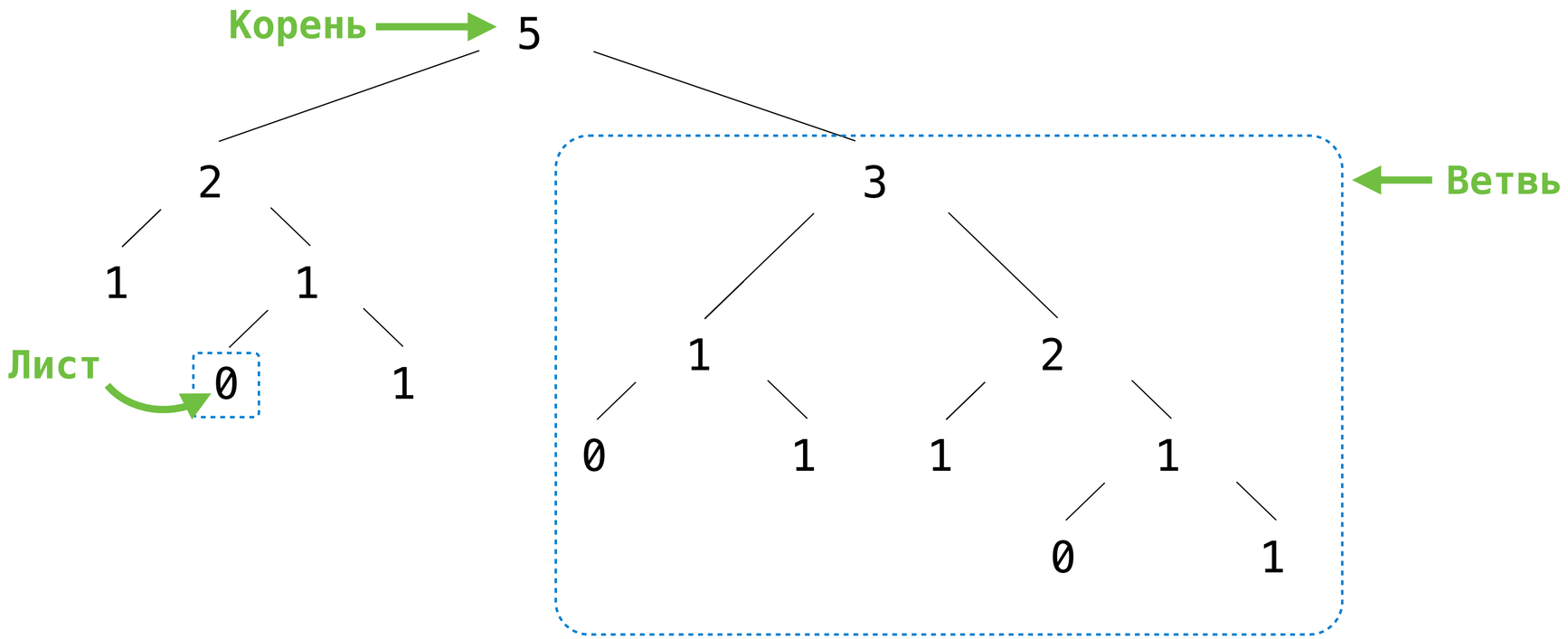
Абстракция дерева



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

Дерево без ветвей называют листом.

Абстракция дерева

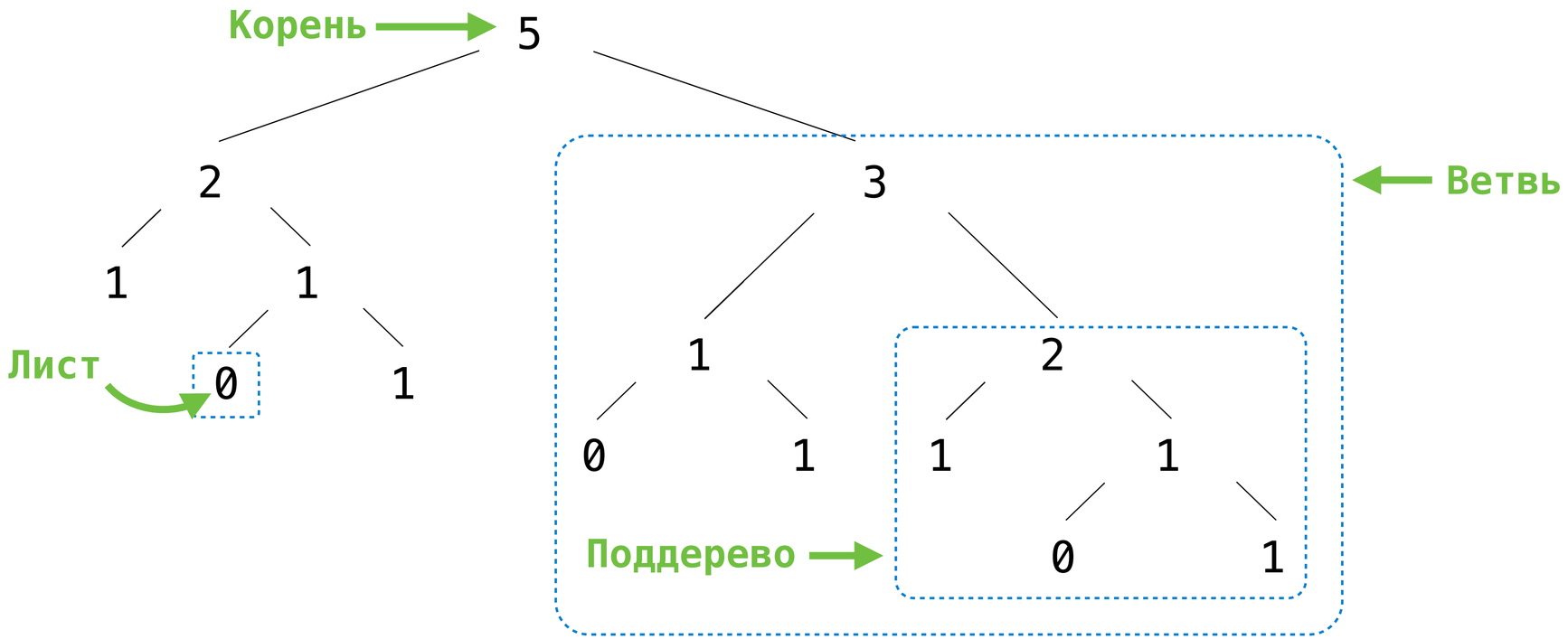


Дерево содержит корневое значение и набор ветвей; каждая ветвь — тоже дерево.

Дерево без ветвей называют листом.

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

Абстракция дерева

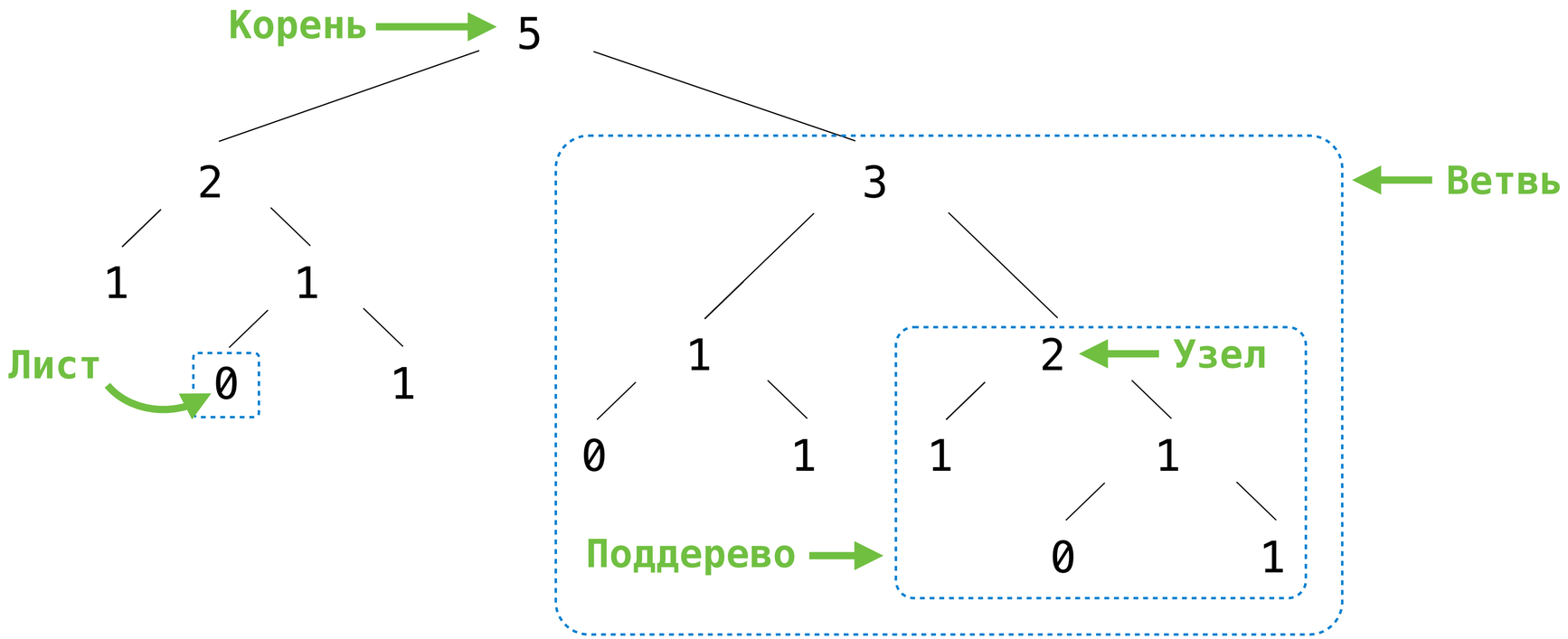


Дерево содержит корневое значение и набор ветвей; каждая ветвь — тоже дерево.

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.

Класс Tree

Класс Tree содержит метку label и список ветвей, каждая из которых — дерево.

Класс Tree

Класс Tree содержит метку label и список ветвей, каждая из которых — дерево.

```
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 fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = label(left) + label(right)
        return tree(fib_n, [left, right])
```

Класс Tree

Класс Tree содержит метку label и список ветвей, каждая из которых — дерево.

```
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)
```

```
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 fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = label(left) + label(right)
        return tree(fib_n, [left, right])
```

Класс Tree

Класс Tree содержит метку label и список ветвей, каждая из которых — дерево.

```
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)
```

```
def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.label + right.label
        return Tree(fib_n, [left, right])
```

```
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 fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = label(left) + label(right)
        return tree(fib_n, [left, right])
```

Класс Tree

Класс Tree содержит метку label и список ветвей, каждая из которых — дерево.

```
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)
```

```
def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.label + right.label
        return Tree(fib_n, [left, right])
```

(Пример)

```
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 fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = label(left) + label(right)
        return tree(fib_n, [left, right])
```