Практика 4. Последовательности и деревья

Списковые включения

Списковые включения (*list comprehensions*) — это компактный способ создать список, элементы которого являются результатами применения заданного выражения к элементам некоторой последовательности.

```
[<отображающее выражение> for <имя> in <итерируемое выражение> if <фильтрующее выражение>]
```

Рассмотрим пример:

```
[x * x - 3 for x in [1, 2, 3, 4, 5] if x % 2 == 1]
```

Здесь описано создание нового списка после применения набора операций к заданной последовательности `[1, 2, 3, 4, 5]`. В новую последовательность попадут только элементы, удовлетворяющие *фильтрующему выражению* `x % 2 == 1`, то есть `1`, `3` и `5`. К каждому из этих элементов будет применено *отображающее выражение* `x * x - 3` прежде, чем из них будет создан новый список. Таким образом, результатом будет список `[-2, 6, 22]`.

Предложение `if` в генераторах списков является необязательным.

Вопрос 1

Что напечатает Python?

```
>>> [i + 1 for i in [1, 2, 3, 4, 5] if i % 2 == 0]
```

Ответ

```

```

```
>>> [i * i for i in [5, -1, 3, -1, 3] if i > 2]
```

Ответ

```

```

```
>>> [[y * 2 for y in [x, x + 1]] for x in [1, 2, 3, 4]]
```

Ответ

```
```

Вопрос 2

Определи однострочную функцию `foo`, принимающую список `lst` и возвращающую новый список, включающий только элементы из четных позиций списка `lst`, домноженные на собственные индексы в списке `lst`.

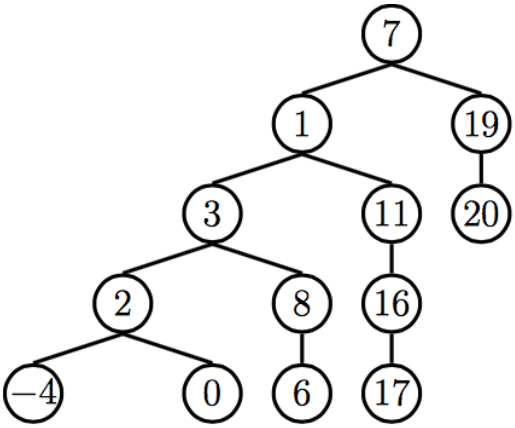
```
def foo(lst):
    """
    >>> x = [1, 2, 3, 4, 5, 6]
    >>> foo(x)
    [0, 6, 20]
    """
```

Ответ

```
```

Деревья

Деревья и рекурсивные структуры данных широко применяются для решения различных задач. Вот простое дерево:



Заметь, что такое дерево растет вверх ногами: **корень** дерева на самом верху, **листья** дерева — внизу.

Краткая терминология:

Родительский узел

узел, обладающий дочерними узлами.

Дочерний узел

узел, имеющий (единственный!) родительский узел.

Корень

самый верхний узел дерева (узел со значением `7` на рисунке).

Лист

узел без дочерних узлов (узлы со значениями `-4`, `0`, `6`, `17`, `20` на рисунке).

Поддерево

каждый дочерний узел является, в свою очередь, корнем меньшего дерева. Это причина того, что дерево является *рекурсивной* структурой данных: дерево состоит из поддеревьев, каждое из которых является деревом.

Глубина

насколько далеко рассматриваемый узел от корня дерева, или, другими словами, расстояние от узла до корня. Узел со значением `3` (смотри рисунок) имеет глубину `2`. Глубина корня равна нулю.

Высота

определяется глубиной самого нижнего узла. На рисунке узлы со значениями `-4`, `0`, `6`, `17` — самые нижние, их глубина равна `4`. Высота представленного дерева равна `4`.

Реализация

Дерево содержит и корневое значение, и набор ветвей. Пусть набор ветвей представлен списком поддеревьев.

- Аргументы конструктора `tree`: корневое значение и набор ветвей.
- Нужны селекторы: значения узла `label` и ветвей `branches`.

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

# Селекторы
def label(tree):
    return tree[0]

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

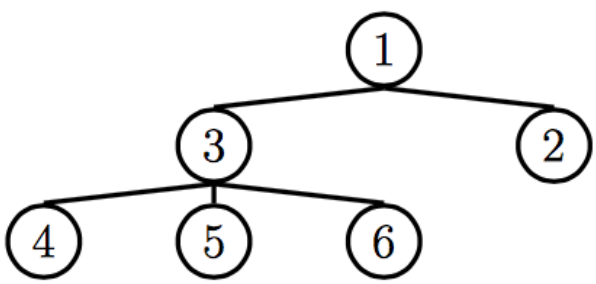
Для удобства понадобятся ещё две функции отвечающие на вопросы:

- `is_leaf` — является ли объект листом?
- `is_tree` — объект является деревом?

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

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
```

Создание дерева — это просто! Вот небольшое дерево и код для его создания:



```
t = tree(1,
    [tree(3,
        [tree(4),
        tree(5),
        tree(6)]),
    tree(2)])
```

Использование *необязательных* отступов помогает лучше считывать структуру дерева.

Вопрос 3

Напиши функцию `tree_max(t)`, возвращающую наибольший элемент дерева.

```
def tree_max(t):
    """Возвращает наибольший элемент дерева.
    >>> t = tree(4, [tree(2, [tree(1)]), tree(10)])
    >>> tree_max(t)
    10
    """
```

Ответ

Вопрос 4

Определи функцию `height(t)`, которая возвращает высоту дерева.

```
def height(t):
    """Возвращает высоту дерева.
    >>> t = tree(3, [tree(5, [tree(1)]), tree(2)])
    >>> height(t)
    2
    """
```

Ответ

Вопрос 5

Напиши функцию `square_tree(t)`, которая возводит в квадрат каждый элемент дерева `t`. Функция должна возвращать новое дерево. Можно считать, что любой элемент — число.

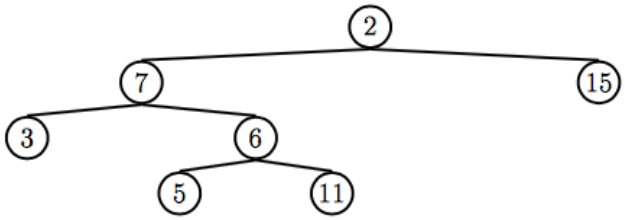
```
def square_tree(t):
```

Ответ

Вопрос 6

Определи функцию `find_path(tree, x)`, которая принимает корневое дерево `tree`, значение `x` и возвращает список узлов вдоль маршрута от корня `tree` до узла `x`. Если дерево не содержит `x`, возвращай `None`. Считай, что все элементы дерева уникальны.

Для следующего дерева функция `find_path(t, 5)` должна вернуть `[2, 7, 6, 5]`




```
def find_path(tree, x):
    """
    >>> t = tree(2, [tree(7, [tree(3), tree(6, [tree(5), tree(11)])]), tree(15)])
    >>> find_path(t, 5)
    [2, 7, 6, 5]
    >>> find_path(t, 6)
    [2, 7, 6]
    >>> find_path(t, 10)
    """
```

Ответ

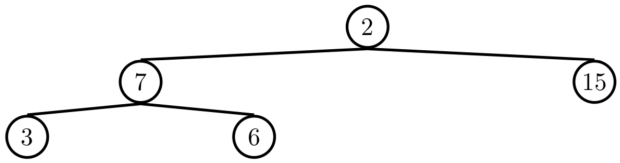
```
    if _________________________________:
        return _________________________________

    _________________________________:
        path = _________________________________
        if _________________________________:
            return _________________________________
```

Вопрос 7

Напиши функцию, которая принимает дерево и глубину `k` и возвращает новое дерево, содержащее только первые `k` уровней исходного.

Например, если `t` — дерево изображенное на рисунке в предыдущем вопросе, то вызов `prune(t, 2)` должен вернуть такое дерево.




```
def prune(t, k):
```

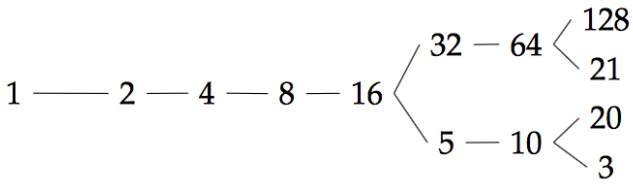
Ответ

Вопрос 8

Последовательность чисел-градин можно представить в виде дерева, показывающего путь различных чисел к единице.

Напиши функцию `hailstone_tree(n, h)`, которая создает дерево высоты h, содержащее числа-градины, которые достигают `n`.

Узел дерева чисел-градин всегда содержит не менее одной и не более двух ветвей (которые также являются деревьями чисел-градин). Подумай-ка при каких условиях появляется вторая ветвь?




```
def hailstone_tree(n, h):
    """
    >>> hailstone_tree(1, 0)
    [1]
    >>> hailstone_tree(1, 4)
    [1, [2, [4, [8, [16]]]]]
    >>> hailstone_tree(8, 3)
    [8, [16, [32, [64]], [5, [10]]]]
    """
```

Ответ