Лабораторная работа 1

Построение бинарного дерева

Разработайте программу на языке Python, которая будет строить **бинарное дерево** (*дерево, в каждом узле которого может быть только два потомка*). Отображение результата в виде словаря (как базовый вариант решения задания). Далее исследовать другие структуры, в том числе доступные в модуле collections в качестве контейнеров для хранения структуры бинарного дерева.

Алгоритм построения дерева должен учитывать параметры, переданные в качестве аргументов функции. Пример:

def gen_bin_tree(height=<number>, root=<number>):

    pass

Если параметры были переданы, то используются они. В противном случае используются параметры, указанные в варианте.

Дерево должно обладать следующими свойствами:

1. В корне дерева (root) находится число, которое задает пользователь (индивидуально для студента).
2. Высота дерева (height) задается пользователем (индивидуально для студента).
3. Левый (left leaf) и правый потомок (right leaf) вычисляется с использованием алгоритмов, **индивидуальных для каждого студента в группе** и приведен ниже.

1. Root = 1; height = 5, left_leaf = root*2, right_leaf = root+2
2. Root = 2; height = 6, left_leaf = root*3, right_leaf = root+4
3. Root = 3; height = 4, left_leaf = root+2, right_leaf = root*3
4. Root = 4; height = 4, left_leaf = root*4, right_leaf = root+1
5. Root = 5; height = 6, left_leaf = root^2, right_leaf = root-2
6. Root = 6; height = 5, left_leaf = (root*2)-2, right_leaf = root+4
7. Root = 7; height = 4, left_leaf = root*3, right_leaf = root-4
8. Root = 8; height = 4, left_leaf = root+root/2, right_leaf = root^2
9. Root = 9; height = 6, left_leaf = root*2+1, right_leaf = 2*root-1
10. Root = 10; height = 5, left_leaf = root*3+1, right_leaf = 3*root-1
11. Root = 11; height = 3, left_leaf = root^2, right_leaf = 2+root^2
12. Root = 12; height = 4, left_leaf = root^3, right_leaf = (root*2)-1
13. Root = 13; height = 3, left_leaf = root+1, right_leaf = root-1
14. Root = 14; height = 4, left_leaf = 2-(root-1), right_leaf = root*2
15. Root = 15; height = 6, left_leaf = 2*(root+1), right_leaf = 2*(root-1)
16. Root = 16; height = 3, left_leaf = root/2, right_leaf = root*2
17. Root = 17; height = 4, left_leaf = (root-4)^2, right_leaf = (root+3)*2
18. Root = 18; height = 5, left_leaf = (root-8)*3, right_leaf = (root+8)*2

Лабораторная работа 2

Реализовать нерекурсивный вариант функции, позволяющей строить бинарное дерево.

Реализацию задачи построения бинарного дерева рекомендуется разбирать на две части:

1. Вычисление элементов (попробовать понять по какой рекуррентной зависимости вычисляются все наши элементы) (см. реализацию в борде).
2. Поместить значения, которые мы вычислили в «правильное» место нашего дерева.

Дерево можно построить «сверху-вниз» или «снизу вверх». Для каждого из полученного в результате выполнения gen_bin_tree значения (см. борд) необходимо создать словарь, который поместить в промежуточную структуру (стек или очередь), из которой значения помещать уже в результирующее дерево.

Стартовый борд: https://replit.com/@zhukov/prog4-lr2#main.py

Протестировать реализацию построения с помощью 2-3 тестов (учесть граничные случаи).

С использованием борда https://replit.com/@zhukov/prog-4-lr2-1#main.py сравнить реализации (рекурсивной и нерекурсивной) построения бинарного дерева с точки зрения эффективности работы алгоритма (время выполнения).

Для этого следует переписать содержимое функции `setup_data` так, чтобы генерировались не списки чисел (в борде пример генерации данных для сравнения работы функции-факториала), а списки пар чисел (кортеж или словарь, представляющих `root` и `height`), также необходимо определить оптимальные значения параметров: количество «прогонов» тестов и длина списка с параметрами для построения деревьев.

Представить в качестве ответа:

- борда с реализацией нерекурсивного и рекурсивного вариантов;
- файлы с графиками, чтобы можно было понять какой алгоритм работает быстрее (внутри борда);
- `README.md` файл (см. борд), в котором описать краткий отчет о проделанной работе и дать комментарии по поводу сравнения эффективности нерекурсивного и рекурсивного реализаций.

ЛР3
Используя официальную документацию, расположенную на сайте docs.python.org и python.org/dev/peps, модуля collections (https://docs.python.org/3/library/collections.html), а также переводную документацию по новым типам данных (см. habr и другие профильные сообщества) найдите информацию по новым типам данных или изменениям, касающихся стандартных типов данных в Python.

Особое внимание следует уделить такого рода изменениям в версиях Python 3.7, 3.8, 3.9 и следующих версиях языка (см. предложения, описанные в PIP).

Примерный формат описания типа данных (коллекциям):
1. Название типа данных.
2. Является ли стандартной (встроенной) или необходимо подключать отдельно (откуда).
3. Для чего нужна: описание, примеры использования.
4. Какие методы, свойства есть.
5. Какими особенностями обладает (скорость доступа, выбрасываемые исключения).

Для выполнения лабораторной работы рекомендуется использовать сервис colab.research.google.com. Для работы с ним потребуется авторизоваться в Google. После этого вам станет доступно создание блокнота (Notebook), который представляет собой текстовый документ со вставками кода.

Лабораторная работа 4
Выполните лабораторную работу 4 и предоставьте ссылку на борд в repl.
В лабораторной работе 4 реализовать:

считывание из файла json (data.json) данных о пользователях (name, gender, email, phone, address, friends);
вывод этих данных на экран с помощью библиотеки pprint;
предложить дополнить считанные данные с клавиатуры;
запись этих данных в отдельный файл clients_data.json.
При этом необходимо обеспечить обработку исключительных ситуаций, которые могут возникнуть (обсуждалось на занятии 27.03):

отсутствие файла;
невозможность считать данные из файла;
отсутствие данных в файле (это не исключительная ситуация, но в случае с этой программой считаем её таковой) в целом или отдельных полей;
невозможность создать файл clients_data.json;
невозможность записать в этот файл данные;
Предусмотреть обработку этих и/или других исключительных ситуаций при выполнении кода программы.

Общий алгоритм действий, связанных с обработкой исключений, для большинства случаев может быть такой:

Анализируем ситуацию: может ли возникнуть исключение, если да, то какое конкретно, если несколько, то есть ли родительский класс исключений, который можно отловить и обработать конкретное. Помещаем потенциально опасную инструкцию в блок try: .. except.
Если ситуация не исключительная, но для другого места в коде - критичная (например, данных в файле нет, а мы их пытаемся вывести), не лучше ли самим выбросить исключение (raise)?!
Если выбросили исключение, можем ли мы исправить ситуацию и сделать что-то, чтобы вернуть ход выполнение программы в "нормальное" русло (нет файла -> пытаемся создать файл). Анализируем ситуацию (см. пункт 1).

Если ситуацию невозможно исправить, то какие действия нужно предпринять для того, чтобы завершить выполнение рассматриваемого кода корректно (блок else, finally в обработке исключений): логировать ошибку /вывести соответствующий текст на экран / отправить письмо разработчику на почту и т.д.

Лабораторная работа 5, 6
Выполните лабораторную работу 5 и предоставьте ссылку на борд в repl.

На основе кода, представленного в бордах:

https://replit.com/@zhukov/sem4-t1-lr5
https://replit.com/@zhukov/sem4-t1-lr5-1 (обратите внимание на файл templates.py)
В лабораторной работе 5 реализовать:

операции CRUD по работе с таблицей user;
вывод этих данных на экран;
При этом необходимо обеспечить обработку исключительных ситуаций, которые могут возникнуть:

отсутствие подключения к БД или самой БД;
отсутствие таблицы user;
отсутствие данных в таблице user для операции обновления данных.
Предусмотреть обработку этих и/или других исключительных ситуаций при выполнении кода программы. Выполнить рефакторинг кода, чтобы все действия (CRUD) с таблицей user выполнялись внутри функций.

Лабораторная работа 7
Выполните лабораторную работу 5 и предоставьте ссылку на борд в repl.

На основе кода, представленного в борде: https://replit.com/@zhukov/sem4-t1-lr7-1

В лабораторной работе 7 реализовать:

сеттеры, делитеры для двух полей класса User;
класс-синглтон для подключения к базе данных sqlite (на основе кода, конспекта лекции по ООП);
При этом необходимо обеспечить обработку исключительных ситуаций, которые могут возникнуть (например, отсутствие самой БД).

Предусмотреть обработку этих и/или других исключительных ситуаций при выполнении кода программы.

Лабораторная работа 8

На основе статьи на хабре: https://habr.com/ru/post/321510/ реализуйте сначала работу с БД с помощью выполнения SQL-запросов напрямую, а во второй части (ссылка) реализуйте использование ORM.

Вторая часть задания — разобраться с ORM Orator (https://orator-orm.com) и реализовать аналогичные запросы к БД. В коде repl.it приведены примеры основных действий, которые составляют CRUD:

https://replit.com/@zhukov/EcstaticCylindricalComputationalscience#main.py