**Python**

*Типы данных*. Изменяемые: list, dict, set. Неизменяемые: int, float, string, bool, nonetype, tuple, frozenset, bytes.

*Отличие == от is*. Оператор == сравнивает объекты по значению, а оператор is – по адресам в памяти.

*Передача аргументов в функцию* осуществляется либо по значению, либо по ссылкам. Неизменяемые типы передаются по значению, изменяемые – по ссылкам.

*Args, kwargs*. Args – аргумент функции, который принимает в себя неограниченное количество позиционных аргументов (представлен tuple). Kwargs – аргумент функции, который принимает в себя неограниченное количество аргументов, передаваемых с помощью ключевых слов (представлен dict).

*Аннотации типов*. Необязательная типизация. Не выполняется в runtime и служит в качестве вспомогательного средства.

*Lamda-функция*. Анонимная функция, не резервирующая место в пространстве имён и принимающая неограниченное количество аргументов. Использует неявный return и может содержать в своём «теле» выражение.

*Тернарный оператор*. Оператор ветвления записываемый в одну строку (if/else).

*Глубокое и поверхностное копирование*. Есть модуль copy, в котором наличествуют методы copy – для поверхностного копирования и deepcopy – для глубокого.

*Виртуальное окружение*. В контексте конкретного проекта, находящегося в отдельной «песочнице», это доступное данному проекту пространство имён. Используется встроенный модуль venv.

*Poetry*. Инструмент для управления зависимостями (аналог встроенного pip).

*Big O notation*. Оценка наихудшей асимптотической сложности алгоритма в зависимости от количества проделанных операций. О(х)…

*Hash-table* – структура данных для хранения пар ключ-значение, в которой положение элемента определяется его хэшем, вычисляемым хэш-функцией. В list и dict быстрее удаление/вставка, поиск везде O(n). Set и dict – хэш-таблицы. Хэширование – процесс преобразования входных данных определённого типа произвольной длины в выходную битовую строку фиксированной длины. Хэш-таблица – это структура данных, представляющая собой разреженный массив (ассоциативный), хранящий указатели на какие-либо данные. Данные хранятся в хэш-таблице по их хэшам. Совпадение хэшей по различным данным называется коллизией. Коллизия разрешается следующими методами: цепочкой – элементы, которым соответствует одно и то же хэш-значение, сцепляются в цепочку-список; метод открытой адресации – списки не создаются, а все данные хранятся в самой хэш-таблице, и каждая ячейка содержит либо элемент множества, либо null; квадратичное хэширование – возведение хэша в степень двойки; двойное хэширование – вычисление хэша при помощи двух хэш-функций.

*Сложность основных операций в коллекциях*. В списках: вставка/удаление элемента в конец О(1), вставка/удаление в начало/середину О(n), прохождение по списку О(n), удаление эдемента О(n).

*Self* – ссылка на созданный уникальный объект некоторого класса. Передаётся первым аргументом в методы объекта.

*Super* – необходим для доступа к родительскому классу. Представляет собой ссылку на объект родительской сущности.

*Методы класса и статические методы* можно вызывать без создания экземпляра класса. Создаются при помощи соответствующих декораторов. Метод класса первым аргументом принимает cls – ссылку на класс.

*Модификаторы доступа*. С \_ начинаются protected свойства и методы, а с \_\_ начинаются private свойства и методы.

*Декораторы*. Структурный паттерн проектирования, который динамически добавляет возможности некоторому объекту и является гибкой альтернативой наследованию. В python по канону декоратор реализуется через класс.

*Абстрактный класс* – это класс, из которого нельзя создать экземпляр. В абстрактном классе определены поля и методы, но не реализованы. Изначально необходимо отнаследоваться от абстрактного класса, затем переопределить его поля и методы, а только потом можно создавать экземпляры-объекты.

*Метаклассы* – шаблоны для создания классов, так же как классы шаблоны для создания объектов. Метаклассы используются для перехватывания создания классов и изменения их. У метаклассов 4 метода: new, init, prepare, call.

*Mixin* – небольшой класс, добавляющий в цепочку наследования некий функционал. Принято указывать в названии mixin в конце.

*MRO* – method resolution order – порядок разрешения методов, то есть последовательность поиска методов при diamond problem – при ромбовидном наследовании. MRO2 осуществлялся поиск в глубину, в MRO3 осуществляется поиск в ширину.

*Итератор и итерируемый объект*. Итератор – структура данных, которая возвращает свои элементы по одному за раз. Итерируемый объект – структура данных, по которой можно итерироваться поэлементно.

*Генератор* – функция, сохраняющая состояние для вычисления последующего элемента какой-либо коллекции и возвращающая данные по одному за раз с помощью оператора yield. Также это возможно итератор-генератор, возвращающий значения по одному. Есть мнение, что любой генератор является итератором, но не наоборот.

*Корутина* – это функция, выполнение которой можно временно приостанавливать при помощи ключевого слова await.

*Библиотеки для асинхронного программирования* – sreding, multiprocessing, asincio.

*GIL* – Clobal Interpreter Lock – Глобальная Блокировка Интерпретатора. Блокировка, позволяющая только одному потоку управлять интерпретатором Python.

*Mock* – «пустышка», объект с ожидаемым поведением, который подменяет другой объект для превентивного купирования возможных непредсказуемых сайд-эффектов.

*Pathing* – патчинг – изменение функционала каких-либо объектов на mock-объекты.

*ACID* – Atomisity, Consistency, Isolation, Durability – Атомарность, Согласованность, Изоляция, Устойчивость. Набор требований к транзакционной системе. Сформированы Джимом Греем в конце 1970-х. Атомарность гарантирует, что никакая транзакция не будет выполнена частично. Согласованность – каждая успешная транзакция фиксирует только допустимые результаты. Изоляция – во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат. Устойчивость – независимо от проблем на нижних уровнях системы, изменения, сделанные успешной транзакцией должны быть сохранены.

*Уровни изолированности транзакции*. 4 уровня: чтение незафиксированных данных, чтение фиксированных данных, повторяющееся чтение, упорядочиваемость.

*Linked list* – связный список представляет собой структуру данных, которая хранит в качестве своих собственных элементов также ссылки на другие свои элементы. Каждый список содержит head, tail и остальные элементы. Бывают односвязные и двусвязные списки. В односвязных списках каждый элемент содержит value и next. В двусвязных – value, next и previous. С помощью списков удобно реализовывать stack и heap.

*Heap* – куча. Полное бинарное дерево, удовлетворяющее свойству кучи по принципам max-кучи – «если А дочерний элемент В, то А всегда больше или равно В» или min-кучи – то есть «если А дочерний элемент В, то А всегда меньше или равно В».

*Graph* *G* – пара множеств (V, E), где V – множество вершин, а E – неупорядоченное множество рёбер. Количество вершин обозначает порядок графа, а количество рёбер – размер графа. Граф без петель и кратных рёбер называется простым графом. Граф называется планарным или плоским, если он может быть изображён на плоскости без пересечения рёбер. Степень вершины графа – количество инцидентных (инцидентность – отношения между вершиной и рёбрами) ей рёбер. Рёбра называются смежными при наличии общей вершины. Висячая вершина – вершина с одним ребром (лист). Вершины называются связными, если есть цепь, их соединяющая. Вес ребра – это некоторое число, закреплённое за ребром. Взвешенный граф – граф, у которого все рёбра имеют вес.

Сумма степеней всех вершин всегда чётна и равна удвоенному количеству рёбер. Маршрут в графе – это конечная последовательность вершин и рёбер, в которой все вершины, кроме последней соединены с последующей вершиной ребром. Длина маршрута определяется количеством рёбер. Цепь – это маршрут без повторяющихся рёбер. Простая цепь – цепь без повторяющихся вершин. Путь – это ориентированный маршрут. Простой путь – путь без повторяющихся рёбер. Цикл – это цепь, в которой первая и последняя вершины совпадают. Элементарный путь – путь без повторяющихся рёбер и вершин. Графы X и Y считаются изоморфными, если существует биективное (взаимно однозначное) отображение ребра к ребру и вершины к вершине. Класс эквивалентности – множество всех вершин, связанных друг с другом. Компонента связности – подграф исходного графа, содержащий все вершины одного из классов эквивалентности по связности и все их рёбра. Связный граф – граф с одной компонентой связности. Мост – ребро, при удалении которого количество компонент связности увеличивается. Точка сочленения – вершина, при теоретическом удалении которой (в графе нельзя удалить вершину) количество компонент связности бы увеличилось.

При маршруте от вершины X к вершине Y степени начальной и конечной вершин уменьшаются на нечётное число, а у всех промежуточных – на чётное (кроме случаев с маршрутом от X к X – там чётное понижение степени X).

*Дерево* – это связный граф, в котором; между любыми двумя вершинами есть только одна цепь; нет простых циклов; количество вершин равно количеству рёбер + 1.

Корневое дерево – дерево, в котором одна из вершин считается корнем. Как только у дерева выбран корень оно становится иерархичным. Высота дерева – максимальное количество рёбер от листа до корня. Диаметр дерева меньше либо равен высоте дерева \* 2.

Остовное дерево – это подраф связного графа, включающий все его вершины и при этом являющийся деревом. Получается путём удаления некоторых рёбер, включённых в циклы.