Для организации многопоточности используются два модуля

threading:

Позволяет создавать потоки как объекты классов, настраивать между ними взаимосвязь (установка события, ожидание события, снятие события, другим потоком, тд), предоставляет стандартные элементы организации многопоточности

Но в Python существуют ограничения на использование многоядерной архитектуры процессора. Global Interpreter Lock. GIL заключается в том, что в каждый момент времени только один поток может исполняться процессором. Это сделано для того, чтобы между потоками не было борьбы за отдельные переменные. Исполняемый поток получает доступ по всему окружению. Такая особенность реализации потоков в Python значительно упрощает работу с потоками и дает определенную потокобезопасность, но из-за особенностей такой реализации это может привести к накладным затратам. Например, когда поток не нуждается в ресурсе процессора, то он освобождает GIL, а в этот момент его может попытаться получить и он сам, и другой поток, и главный поток. При этом операционная система, полагая, что ядер много, может усугубить все попыткой распределить потоки между ядрами и выйдет, что программа с использованием многопоточности будет работать медленнее, чем однопоточная.

Для того, чтобы избежать таких ограничений, в Python есть модуль subprocess. Мы можем написать программу, которую хотим исполнять в параллельном потоке (на самом деле уже процессе). И запускать ее в одном или нескольких потоках в другой программе. В этом смысле рассматривается другой модуль

multiprocessing:

Напоминает threading. Например, процессы можно создавать точно так же из обычных функций. Методы работы с процессами почти все те же самые, что и для потоков из модуля threading. Отличие в том, как устроена синхронизации процессов и обмена данными: здесь используются Queue и Pipe. В модуле есть классы Value для перемсенных и Array для массива, их можно делать общими для процессов. Предусмотрены классы-менеджеры. недостатком этого модуля считается его платформозависимость: в зависимости от платформы некоторые методы или прочие возможности модуля использовать не удастся.

Насколько я понимаю, анонимная функция – это функция без имени и то же самое, что и лямбда-выражение. Обращение к ней идет по адресу. Лямбда-выражение – это небольшая однострочная функция (но не инструкция, а выражение), ее можно использовать там, где функцию нельзя (например в вызове другой функции). Часто их по этим причинам используют со встроенными функциями вроде map, filter. Можно создавать списки или словари из лямбда-функций.

К ограничениям можно отнести следующие особенности:

* могут содержать только одну строку кода
* возвращаются автоматически
* им нельзя присвоить имя

Максимум, присвоить переменной..

def func(x, y):

return x\*\*2 + y\*\*2 функция возвращает х^2+y^2

Можно записать как лямбда-выражение:

func = lambda x, y: x\*\*2 + y\*\*2 до двоеточия указываем аргументы, после – что вернет лямбда-функция, можно не присваивать переменной, как видно, функция сразу обрабатывает данные и возвращает результат.

Пример списка из Лямбда функций:

plural\_rules = [

lambda n: 'all',

lambda n: 'singular' if n == 1 else 'plural',

lambda n: 'singular' if 0 <= n <= 1 else 'plural'

]