Asyncio (от англ. Asynchronous input/output ) – это встроенная в python библиотека. Она помогает реализовывать одновременное выполнение нескольких вычислительных процессов, которые могут взаимодействовать друг с другом.

СЛАЙД

Изначально asyncio была создана, чтобы предоставить стандартизованную базу для создания асинхронных фреймворков. pep3156 так же приписывает ей необходимость обеспечить предельно простую интеграцию в уже существовавшие асинхронные фреймворки (Twisted, Tornado, Gevent)

СЛАЙД

В основе библиотеки лежат 3 абстракции:

* Event Loop

Цикл событий. Он управляет и распределяет выполнение различных задач, регистрирует их и обрабатывает распределение потока управления между ними

* Future   
  Его можно охарактеризовать как placeholder для значения, которое вернет нам код, выполненный асинхронно.   
  Это Результат операции, который будет доступен в будущем. Он в чем-то схож с работой promise в JavaScript.
* Task

Специальный подкласс класса Future для запуска корутин на цикле событий. Запланированные корутины будут оборачиваются в Tasks, что является типом Future.

Давайте разберемся с каждым из них и начнем с event loop.

Для того, чтобы фрагменты кода могли выполняться асинхронно мы должны сделать их корутинами. Для этого мы используем ключевое слово async перед объявлением функции.

СЛАЙД

Как видите, func у нас теперь определяется как coroutine object, однако вызвать мы ее уже не можем, интерпретатор выдает ошибку. Корутины в asyncio могут вызываться только в EventLoop. Давайте добавим нашей функции какой то вывод, чтобы мы понимали что она выполнилась, а затем поместим ее в Event Loop. В asyncio Event Loop запускается при помощи asyncio.run

СЛАЙД

Вот, как видите, теперь код выполняется.

Теперь давайте разберем еще одно ключевое слово в asyncio “await”. Любую асинхронную функцию или корутину в asyncio необходимо вызывать при помощи ключевого слова “await”. Разберем следующий, немного измененный пример

СЛАЙД

Я изменил код. Добавил импорт библиотеки time, для того чтобы мы могли видеть за сколько выполнилась программа. В Event Loop я поместил функцию main, в которой я вызываю 2 асинхронные функции: func и asyncio.sleep (аналог time.sleep, только асинхронный)

Как можно заметить в терминале, обе функции выполнились, а также сработал sleep.

СЛАЙД

Теперь давайте посмотрим на таски, пока что на абстрактном примере.

Допустим, теперь корутине func, ПОСЛЕ вывода слова world нужно 3 секунды. Все эти 3 секунды она будет просто ждать выполнения asyncio.sleep. Однако, теперь, помимо hello world мы еще также хотим вывести hello everyone. Выполняя все эти корутины синхронно, мы теряем драгоценное время. При помощи тасков, мы можем, как бы, сказать event loop, что за эти 3 секунды ничего не делания можно приступить к другим корутинам.

СЛАЙД

Как вы видите, сказав asyncio что наша корутина func это task он приостановил ее выполнение и перешел к new\_func и вся программа выполнилась менее чем за секунду

СЛАЙД

Наш event loop рассматривает таски буквально как в настоящих туду листах, пока я гуляю с собакой я могу ответить мистеру Швинку, пока я жду ответа я могу начать делать домашку.   
Благодаря таскам куски кода можно распределять точно также как я сейчас распределил свои дела.

Теперь давайте перейдем к примеру посложнее, но зато на нем мы сможем узнать про третью абстракцию в asyncio – фьючеры, а также посмотреть как практически можно использовать модуль asyncio

VSCODE

Пусть у нас будет следующая задача. Нам надо написать программу для метеозонда. Каждые 10 секунд нам надо собирать информацию о погоде в нескольких городах с сервера и записывать ее в файл.

Не зная про асинхронность, мы бы как делали: собирали данные, записывали их в файл а затем ставили таймер на 10 секунд и ждали. Но вот в чем проблема: Ответ от сервера и запись в файл занимают время, каждый раз разное. Из-за этого получается, что данные будут идти не совсем верно, не раз в 10 секунд, а в 15 или 14 в зависимости от времени на запрос и на запись.

Как же нам поступить? При помощи asyncio мы можем, как минимум свести погрешность к минимуму.

Вот у нас есть 4 функции (описываю что каждая из них делает и показываю)

И есть main