**2023/12/16 00:00|Лекция. Очереди в потоках**

## Очереди в потоках

Продолжаем разбираться с потоками и процессами. На данном занятии изучим, что такое очереди, зачем они нужны, и как они помогают синхронизировать между собой потоки. Для этого рассмотрим первый пример.

У нас будет два потока «producer» и «consumer». В переводе с английского «producer» тот, кто производит; «consumer» тот, кто потребляет. То есть производитель и потребитель. Одна функция будет передавать данные, которые нам нужно будет получить из другой функции. Мы посмотрим, как с этим нам помогут очереди. Импортируем класс «Thread», очередь. И импортируем «sleep» (Рис.1). Он нам понадобится, чтобы имитировать работу какой-либо бизнес логики.

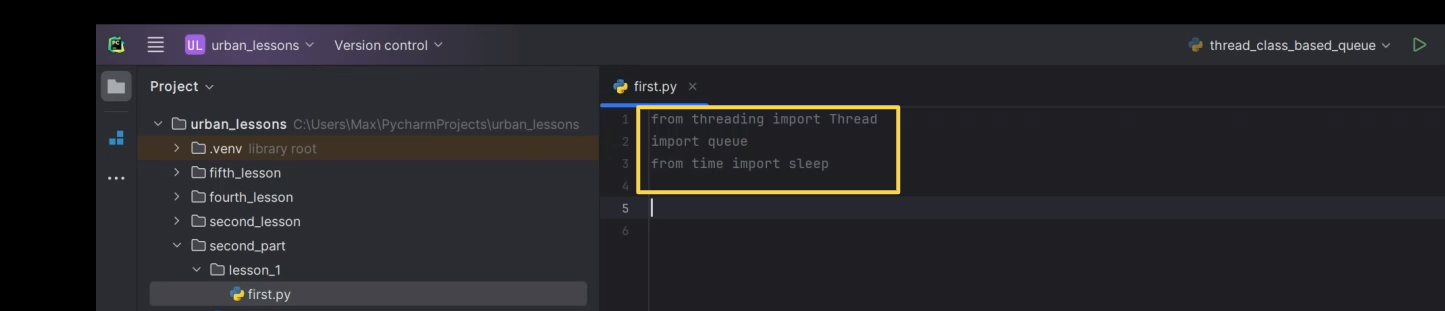


Рис.1

Пишем «producer» (Рис.2).

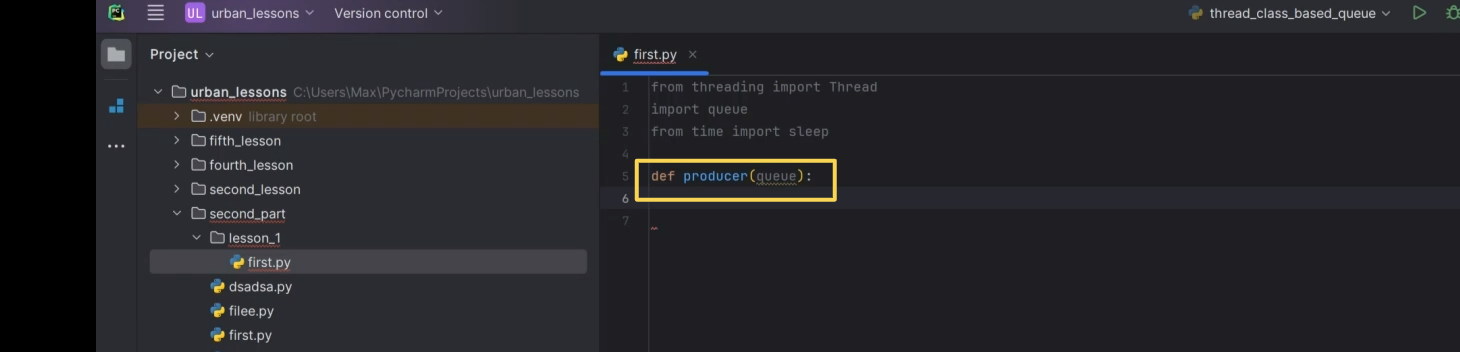


Рис.2

В нашем случае просто раз в секунду будет отправляться некоторое сообщение. Допустим, «ping». И его уже другой поток, называемый «consumer», будет принимать. «queue»-очередь. Это то, куда он это сообщение «ping» и будет засовывать. То есть он производит сообщение «ping» и кладёт его в «queue», ждёт секунду, кладёт снова в «queue». И вот так бесконечный идет цикл.

Функция «consumer», которую мы напишем после этого из этой очереди, достаёт эти сообщения. Поэтому нас этот поток будет бесконечным. В целом вся программа будет из-за этого бесконечной.

Создаём переменную «message». Пусть она называется «ping». Кладём в очередь это сообщение. И пишем «sleep(1)» (Рис.3).

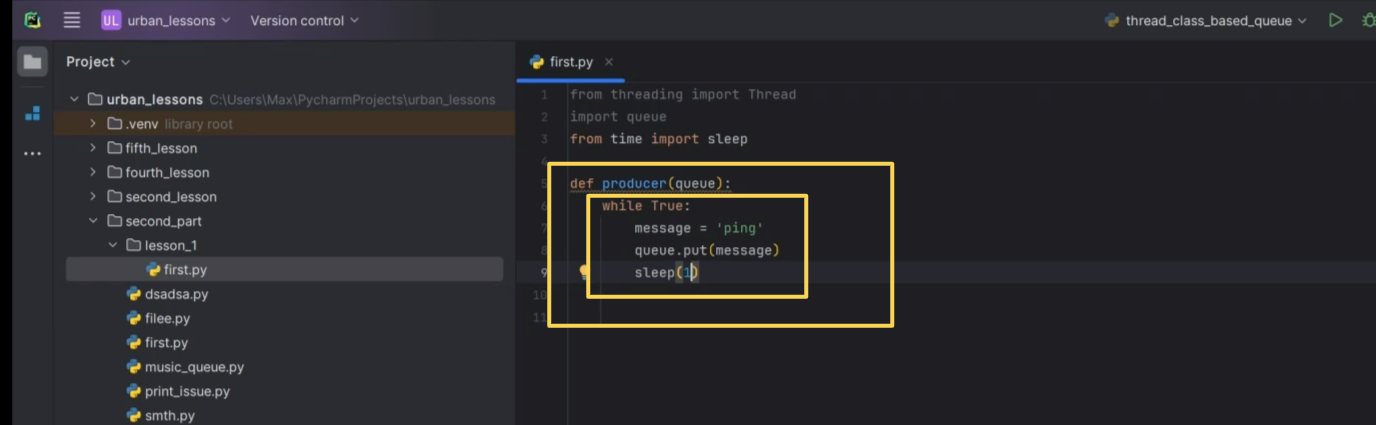


Рис.3

Очередь это то же самое, что и список. Но она обладает некоторыми методами. Допустим, метод «queue.put» кладёт сообщение в очередь.

Отличие очереди от списка заключается в том, что она имеет **определённое правило «first in, first out» или «первым вошёл-первым вышел».** Это правило обычной очереди. То сообщение, которое попадает первым в очередь, с помощью метода «queue.get» оно же и выйдет первым. (Этот метод мы будем использовать в другой функции). То есть здесь нет индексации, это не словарь, не список, и нет ключей, как у словаря. Кладутся определённые сообщения и просто в порядке очереди обрабатывается.

Логика здесь не сложная. Но, если речь заходит о более сложных процессах, это очень удобно, когда у нас несколько процессов или несколько потоков в порядке очереди занимаются обработкой каких-то данных.

Напишем функцию «worker», которая так же принимает «queue». Далее «message=queue.get». И выводим это сообщение (Рис.4).

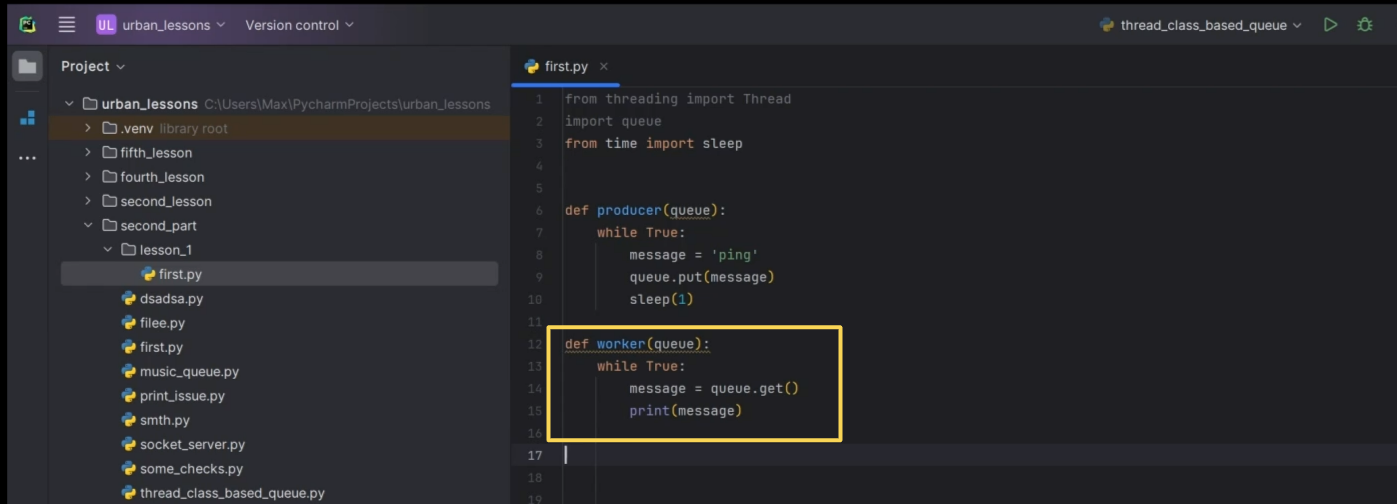


Рис.4

У нас созданы две target функции для потоков. Теперь создаём оба потока. Первый «th1=Thread (target=producer, args=(q, ))». В «args» у нас будет выступать объект из модуля «queue». И второй «th2=Thread (target=worker, args=(q, ))» (Рис.5). Оба потока работают с одним и тем же объектом: один в этот объект кладёт, другой из этого объекта достаёт.

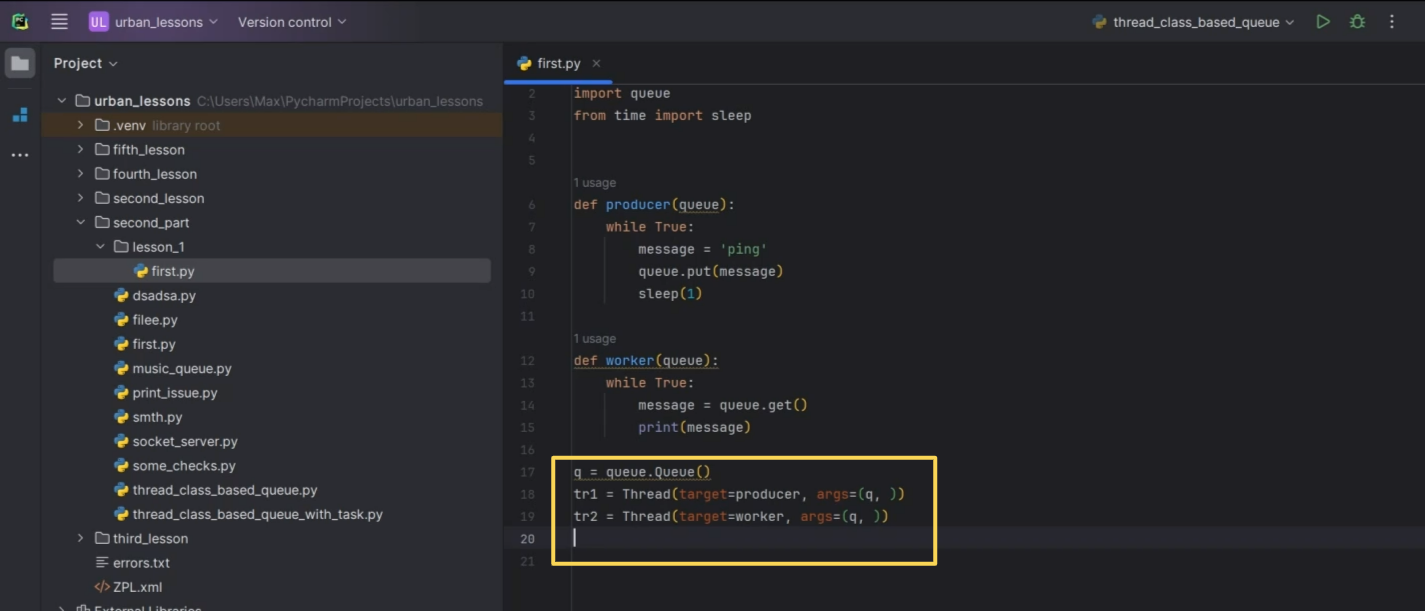


Рис.5

Напишем «th1.start()» и «th2.start()». Так же делаем «join», чтобы они моментально не закончились (Рис.6).

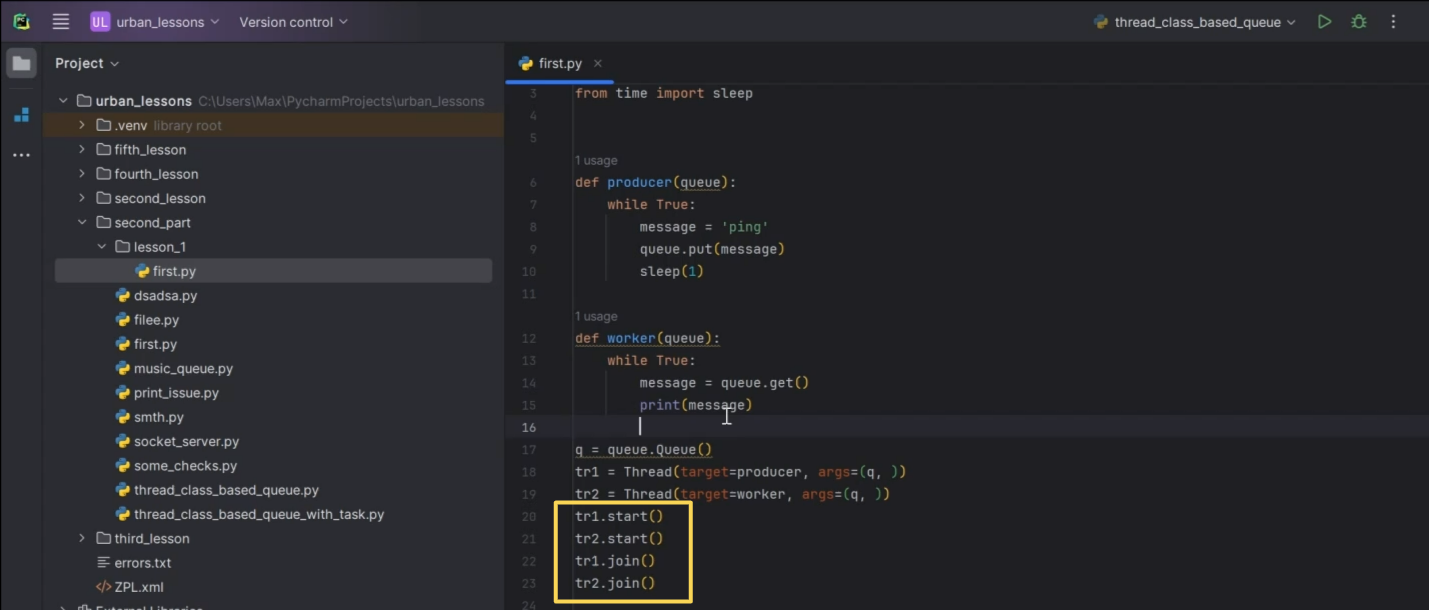


Рис.6

В данном случае у нас будет бесконечная работа, потому что у обоих есть «while True» (Рис.7).

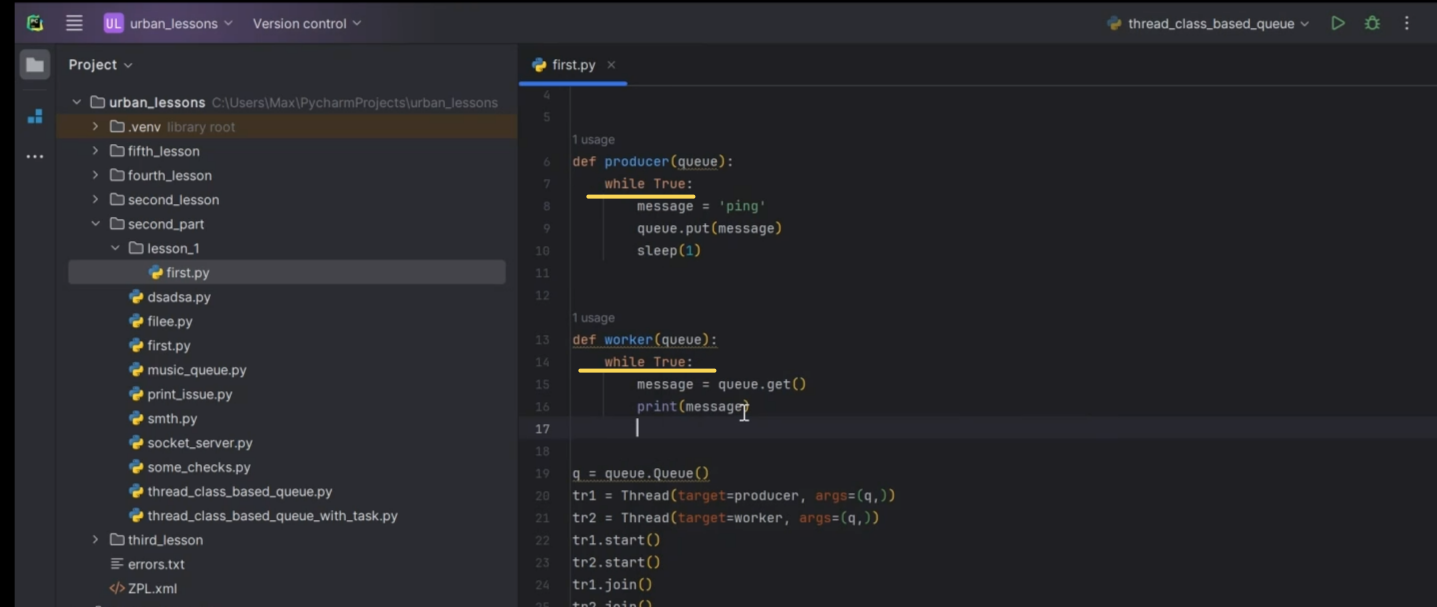


Рис.7

Ещё раз пройдёмся по тому, что происходит. У нас есть один «thread» в бесконечном цикле, который кладёт в конец очереди сообщение и потом засыпает на секунду. «Worker» это сообщение достаёт и печатает (Рис.8).

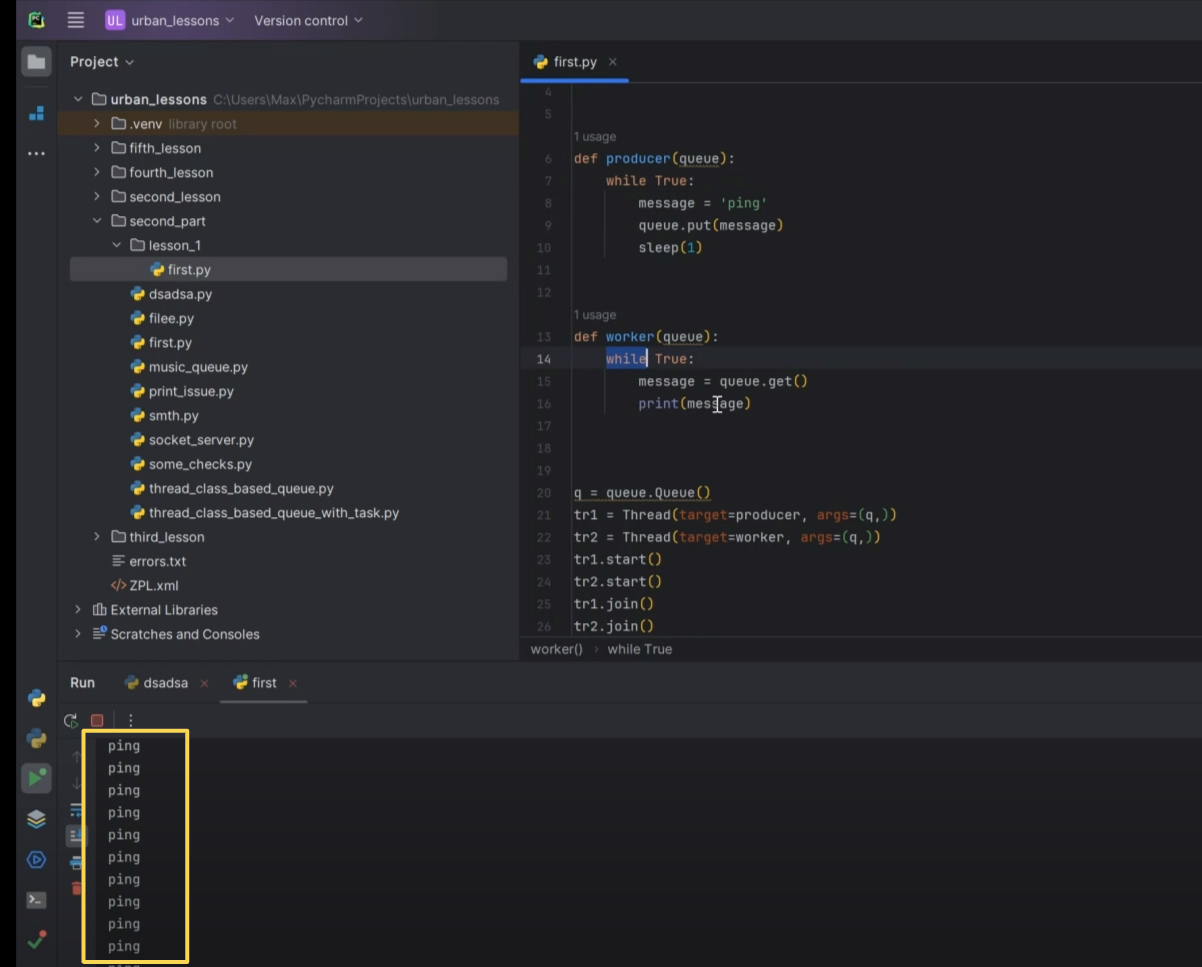


Рис.8

В чем суть очереди в этой задаче? У нас определено, как быстро работают «producer» и «worker». «Producer» гораздо дольше производит сообщения, чем «worker» их печатает (Рис.8). Потому что в «worker» никакой задержки нет.

Если мы удалим из функции «producer» «sleep(1)» и добавим в «worker», очередь позволяет «producer» производить сообщения в то время, как «worker» ещё не готов их принять (Рис.9).

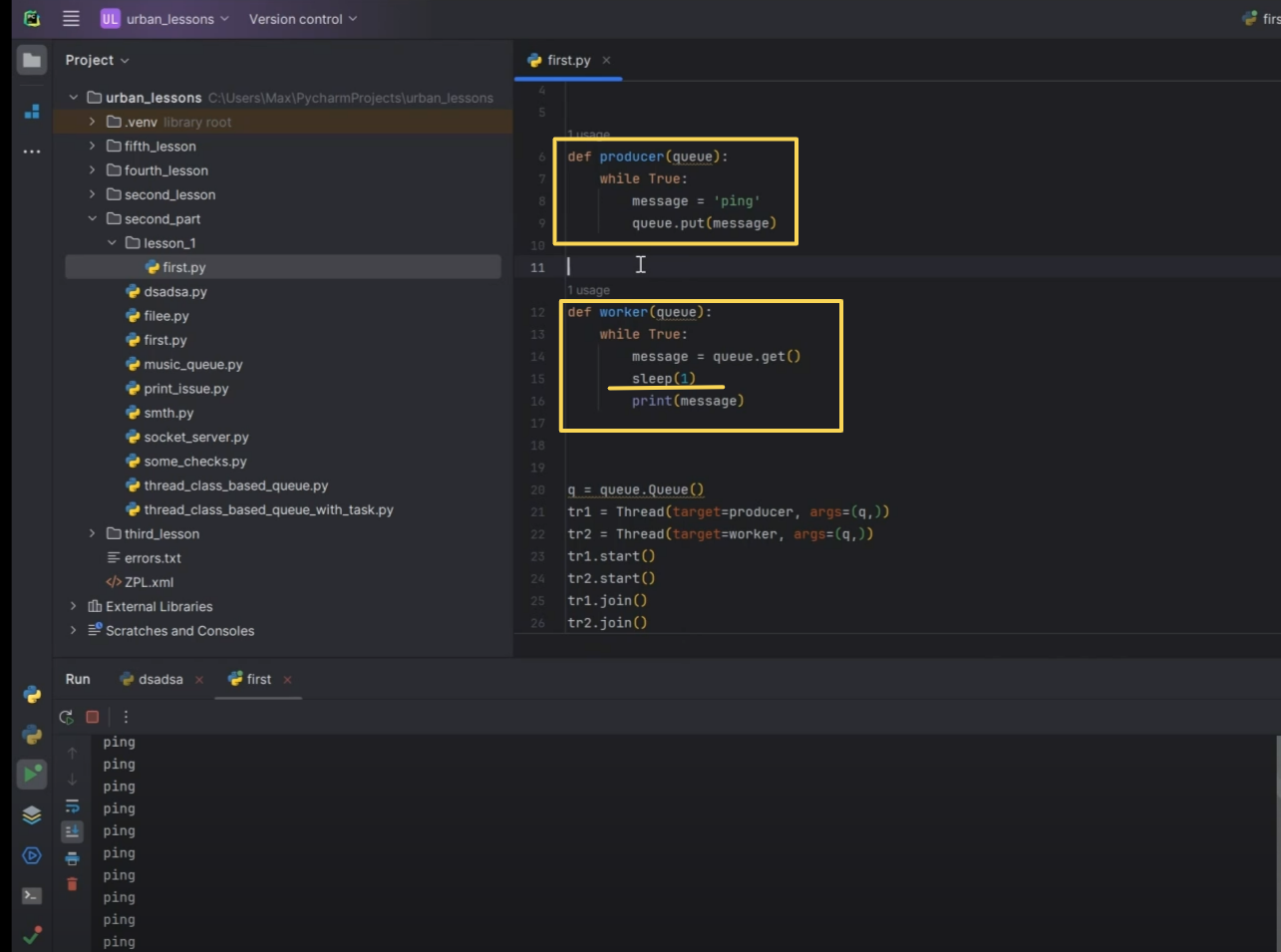


Рис.9

«Queue» это небольшая прослойка между двумя «thread», которая позволяет их синхронизировать. То есть иногда очередь будет пустая, потому что «producer» не успевает производить достаточное количество сообщений, которые «worker» будет обрабатывать. И тогда по итогу «producer» работает, а «worker» просто ждёт, пока придёт это сообщение.

Может происходить и наоборот: «worker» не успевает обрабатывать все сообщения, которые приходят от «producer». И чтобы не стоять на месте и не ждать пока «worker» обработает, «producer» продолжает дальше работать, производить новые сообщения и заполняет очередь, из которой достанут эти сообщения.

Перезапустим (Рис.10).

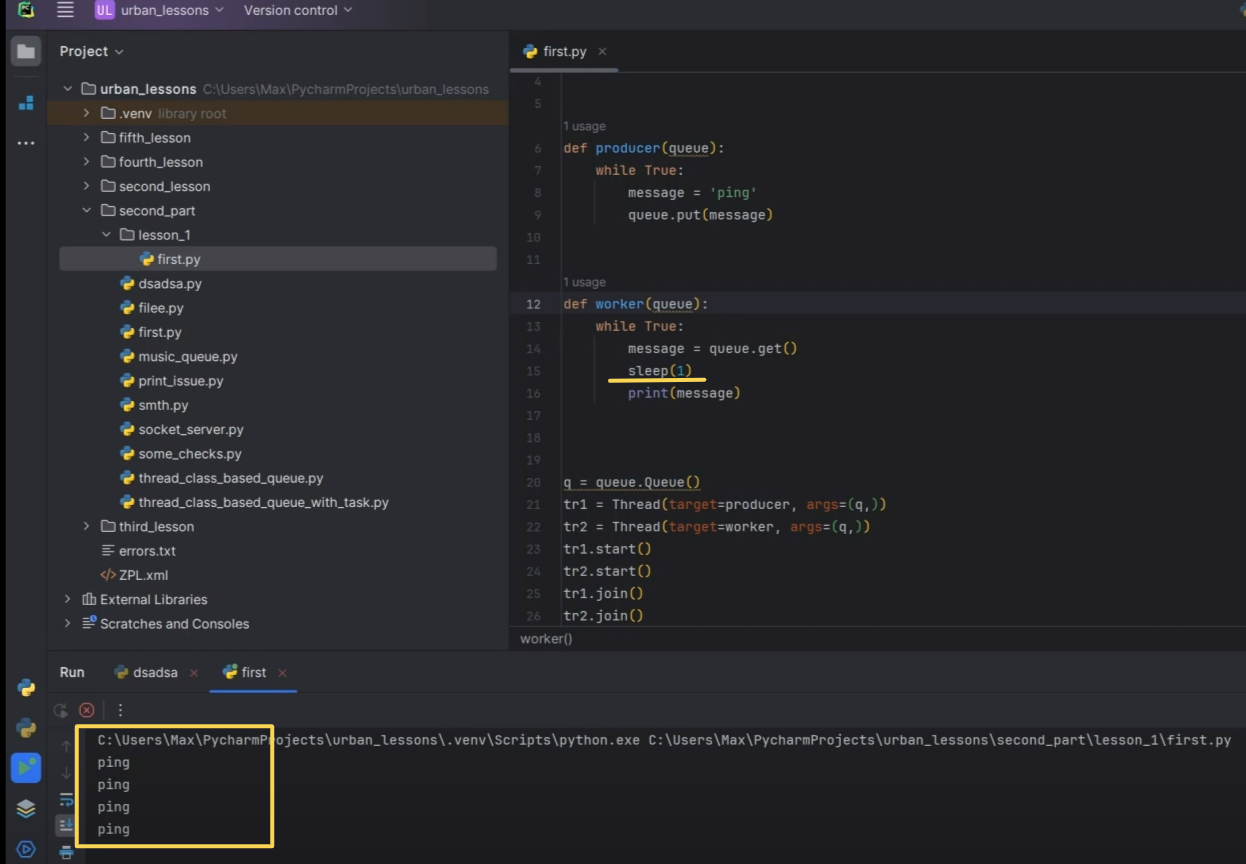


Рис.10

На этапе «print» ситуация никак не поменяется, просто «sleep» будет происходить в другом месте. Теперь у нас в очереди есть какое-то количество объектов, так как «producer» производит гораздо быстрее сообщения, чем «worker» их обрабатывает.

Добавим в «producer» переменную «c», чтобы мы увидели, что порядок у них соблюдается (Рис.11). Порядок очереди: первый попадает-первый обрабатывается в «worker». Потом идет следующий «ping 2». Важно понимать, что в тот момент пока «worker» обрабатывает возможно только первое сообщение, в очереди уже могут лежать сообщения 2, 3, 4, 5, 6 и так далее.

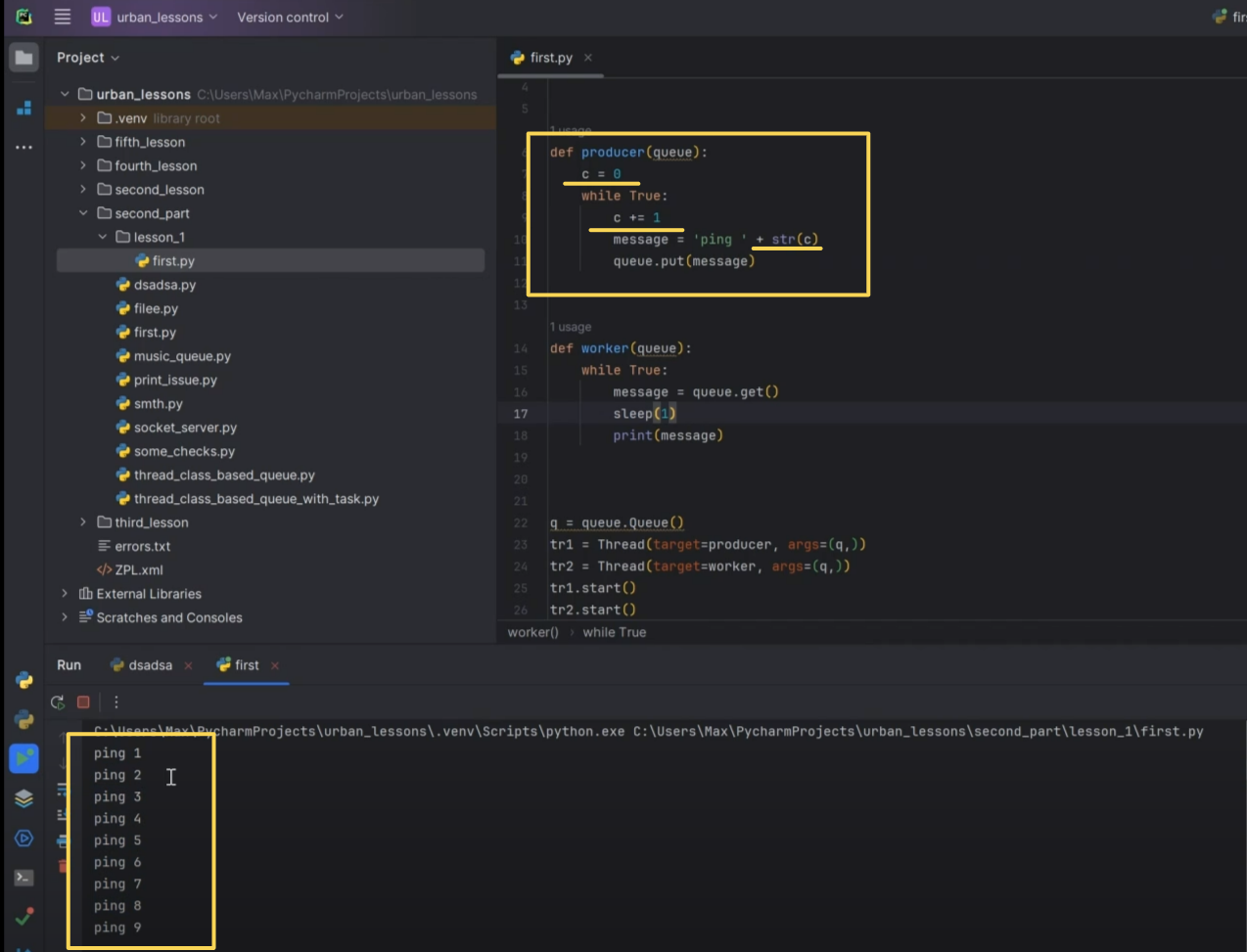


Рис.11